

БИБЛИОТЕЧКА  
ПРОГРАММИСТА

Н. И. КОЗЛОВ

# Организация вычислительных работ

**БИБЛИОТЕЧКА  
ПРОГРАММИСТА**

---

**Н. И. КОЗЛОВ**

**ОРГАНИЗАЦИЯ  
ВЫЧИСЛИТЕЛЬНЫХ  
РАБОТ**



**МОСКВА «НАУКА»  
ГЛАВНАЯ РЕДАКЦИЯ  
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ  
1981**



Scan AAW

22.19  
К 59  
УДК 519.6

**Организация вычислительных работ.** Н. И. К о з л о в. — М.: «Наука». Главная редакция физико-математической литературы, 1981.

Книга посвящена организации вычислительных работ для решения задач вычислительного эксперимента. Разбираются этапы вычислительного эксперимента — построение физической и математической моделей, построение дискретной модели, программирование, проведение расчетов, анализ полученных результатов.

В качестве иллюстрации приводятся примеры, в основном ориентированные на инженерно-физические задачи применительно к машине БЭСМ-6, операционной системе Диспак и монитрной системе Дубна.

*Николай Иванович Козлов*

## ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ РАБОТ

М., 1981 г., 240 стр. с илл.

Редакторы *В. П. Варакин, Н. Н. Васина*

Техн. редактор *Е. В. Морозова*

Корректор *А. Л. Ипатова*

ИБ № 11057

Сдано в набор 03.11.80. Подписано к печати 11.03.81. Т-05722. Бумага 84×108<sup>1</sup>/<sub>2</sub>, тип. № 1. Обыкновенная гарнитура. Высокая печать. Условн. печ. л. 12,6. Уч.-изд. л. 13,58. Тираж 24 000 экз. Зак. № 3804. Цена 85 коп.

Издательство «Наука» Главная редакция физико-математической литературы  
117071, Москва, В-71, Ленинский проспект, 15,

2-я типография издательства «Наука» 121099, Москва, Г-99, Шубинский пер., 10

К  $\frac{20204-042}{053(02)-81}$  66-80. 1702070000

© Издательство «Наука».  
Главная редакция  
физико-математической  
литературы, 1981

## ПРЕДИСЛОВИЕ

Развитие науки и техники, изучение физических процессов, конструирование сложных установок сегодня невозможно без подробного изучения управляющих ими закономерностей с помощью математических методов. Теоретические исследования всегда опирались на математические методы, дававшие количественные характеристики. Сложность математического аппарата определялась сложностью изучаемого объекта и степенью охвата всех его сторон. В основном применялись аналитические методы, арсенал которых был не велик: метод разделения переменных, линеаризация уравнений, сведение к задачам меньших размерностей, асимптотические методы, разложение по малому параметру и др. Во многих случаях использовать эти методы не удавалось и тогда прибегали к численным расчетам, в принципе позволявшим решать любые задачи.

В середине нашего века развитие таких важных областей, как ядерная физика, теория атомов и молекул, космические исследования, энергетика, конструирование сложных установок, привело к необходимости проводить сложные и большие расчеты, которые невозможно было провести с помощью существовавших в то время технических средств. Это явилось стимулом к созданию электронных вычислительных машин (ЭВМ), открывших новую эру в применении математических методов исследования. Стало возможным использовать сложные нелинейные математические модели, учитывающие все существенные черты объекта в широком диапазоне изменения параметров. Появилась возможность получить достаточно полное количественное описание изучаемого явления. Существенно расширился круг решаемых задач.

Применение ЭВМ изменило весь стиль и методику проведения исследований. По существу, появилось новое

направление в исследованиях, которое можно назвать вычислительным экспериментом или математическим экспериментом [4, 66]. Оно сводится к изучению свойств реальной конструкции или процесса на основе выбранной физической и математической модели. С помощью расчетов изучается поведение объекта при варьировании различных параметров (что может быть связано с некоторым изменением физической и математической модели). Это вполне соответствует физическому эксперименту. В ряде случаев вычислительный эксперимент может с успехом заменить реальный, особенно, когда проведение реального эксперимента требует больших материальных затрат или вообще невозможно. В [66] отмечается, что вычислительные возможности страны, состоящие из мощных ЭВМ и коллективов квалифицированных научных сотрудников, представляют на сегодня важный стратегический потенциал, который может быть использован для решения крупнейших национальных программ: проблемы ядерной физики, теория ядерных реакторов, управляемый термоядерный синтез, физика плазмы, МГД-преобразователи, физика лазеров, аэрогидродинамика, метеорология и другие.

Технологический цикл вычислительного эксперимента включает ряд этапов:

- выбор физической и математической модели,
- разработка дискретной модели (численный алгоритм),
- разработка программ,
- проведение расчетов,
- обработка и анализ результатов расчетов.

Под организацией вычислительных работ мы будем понимать реализацию всех пяти этапов \*). Важная их особенность — это взаимосвязанность. Например, даже на этапе построения физической и математической модели приходится учитывать структуру ЭВМ и характер применяемых программных средств, особенности вычислительного центра, оказывающие влияние на проведение расчетов.

В книге описываются вопросы технологии разработки вычислительного эксперимента, указывается перечень работ, выполняемых на каждом этапе, прослеживается взаимная связь этапов.

---

\*) Наравне мы будем употреблять термин проведение (разработка) вычислительного эксперимента.

Изложение на содержательном уровне (например, выкладки, доказательства и т. д.) не входит в задачу книги. Исключение составляют примеры, иллюстрирующие те или иные положения. Это связано с тем, что по некоторым этапам вычислительного эксперимента имеется специальная литература (например, по численным методам [4, 39 — 41, 70]). Поэтому целесообразно включить лишь те вопросы, которые еще не нашли отражения в литературе. Очень важно охватить весь комплекс вопросов в их взаимной связи, ограничиваясь, возможно, лишь постановкой акцентов.

Приводимые методические указания должны стимулировать читателя к самостоятельной деятельности в соответствующем направлении. В связи с этим, как правило, указывается цель того или иного приема без указания средства реализации (исключая примеры). Выбор такого средства в сильной степени зависит от опыта разработчика, «программной и аппаратной среды» и от других достаточно индивидуальных моментов.

В примерах взята ориентация на задачи инженерно-физического плана, а в качестве аппаратной и программной среды выбрана машина БЭСМ-6, операционная система Диспак и мониторная система Дубна.

Особое внимание уделяется вопросам, традиционно относившимся к области интуиции и искусства вычислителя и программиста. Для этого круга вопросов существовало не слишком подходящее наименование «кухня». В нем нашли отражение два важных обстоятельства: обилие «мелких» решений, которые надо принимать при кажущейся незначительности каждого из них, и отсутствие веских обоснований принятых решений. Между тем все эти «маленькие хитрости» оказывают большое влияние на эффективность разработки. Например, после выбора разностной схемы и метода решения полученной системы разностных уравнений до окончательного построения алгоритма нужно решить много вопросов, связанных с преобразованием формул счета, масштабированием, порядком вычисления величин и др. Аналогичные проблемы решаются и при создании программы. В книге сделана попытка классифицировать некоторые технологические приемы выполнения работ. Разумеется, классификация не претендует на завершенность и полноту.

В тексте встречается довольно много рекомендаций и высказываний общего характера. Самое простое из них

принадлежит Дейкстра [6] и сводится к совету писать простые программы. Подобные замечания, даже спорного характера, нам представляются полезными.

Сделаем замечание о стиле изложения. Каждое положение, как правило, иллюстрируется конкретным примером. Это нарушает связность повествования. Если приведенное высказывание для читателя не требует пояснений, пример без ущерба для понимания дальнейшего может быть пропущен. При изложении разделов документов также даются разъяснения содержания разделов в виде примеров; важно разъяснить, что кроется под наименованием той или иной работы. С другой стороны, такие отступления позволяют изложить дополнительно те или иные технологические приемы в виде некоторой системы.

Материал книги подобран с таким расчетом, чтобы чтение не требовало привлечения дополнительной литературы, а у читателя предполагается подготовка в объеме обычной вузовской программы. В целом материал выбран с учетом первого знакомства с организацией вычислительных работ. По этой же причине к наиболее сложным вопросам мы возвращаемся неоднократно, каждый раз отмечая новые моменты в порядке возрастания сложности. Такой подход одновременно способствует независимости разделов книги.

Материал по главам распределяется следующим образом.

В первой главе рассматриваются поколения машин, программного обеспечения и методов решения задач. Эта глава носит вводный характер и рассчитана на неподготовленного читателя. В настоящее время к вычислительным работам приобщаются специалисты самых разнообразных отраслей знаний и для некоторых из них материал этой главы может быть использован как подготовка к чтению более серьезной литературы [59]. Более осведомленный читатель может без ущерба пропустить эту главу. Заметим, что изложенный там материал достаточен для дальнейшего чтения книги.

Вторая и третья главы посвящены примерам задач вычислительного эксперимента. Они должны помочь читателю раскрыть смысл этого важного понятия.

Выбрав такие крупные задачи, мы, тем самым, лишили себя возможности в деталях продемонстрировать технологию организации вычислительных работ. С другой стороны, искусственно подобранные примеры не дают

возможности правильно воспринять проблемы технологии вычислительного эксперимента.

Четвертая глава посвящена вопросам технологического цикла вычислительного эксперимента. Обсуждаются особенности программирования, последовательность и содержание работ на каждом этапе. Приводятся примеры различных технологических приемов. Естественно, набор приемов не претендует на полноту и отражает опыт некоторой группы разработчиков. В этой главе уделено внимание документации. Даются рекомендации по составу документов вычислительного эксперимента. Уделено большое внимание вопросам стандартизации на методической основе стандартов Единой Системы Программной Документации (ЕСПД).

Пятая глава посвящена пакетам прикладных программ, т. е. технологии программирования при разработке вычислительного эксперимента. Излагаются принципы и приводятся примеры конкретных пакетов. Основная задача — помочь читателю ориентироваться в этой новой, быстро развивающейся области. Однако следует отдавать себе отчет в том, что предложенная трактовка проблемы пакетов — не единственная. Это лишь один из возможных вариантов.

Шестая глава посвящена некоторым вопросам организации работы вычислительных центров, оказывающим влияние на проведение вычислительного эксперимента. К числу таких вопросов могут относиться и чисто административные.

При написании книги особую сложность представляют вопросы терминологии. В основном терминология соответствует [59]. Вместе с тем термины, принятые в оригинальных работах и инструкциях, не заменялись на иные. Это соответствует месту системного программирования в книге. Оно обсуждается для разъяснения смысла описанных технологических приемов.

Список литературы не претендует на полноту и служит, главным образом, для указания источника, по которому можно более полно ознакомиться с рассматриваемым вопросом.

Нумерация рисунков и формул состоит из двух чисел — номера главы и порядкового номера в пределах глав.

В пределах каждого параграфа имеются рубрики, помогающие ориентироваться в материале параграфа. Эти рубрики в оглавление не выносятся.



Работа над материалом книги ведется с 1971 г. и во многом связана с курсом лекций, который автор читает для студентов факультета Вычислительной Математики и Кибернетики в МГУ им. М. В. Ломоносова.

Автор считает приятным долгом выразить благодарность профессору Л. Н. Королеву за постоянный интерес к тематике курса и полезные советы.

При подготовке книги автор знакомился с материалами «Всесоюзного семинара по комплексам программ математической физики», проводимого под руководством Н. Н. Яненко. Автор благодарен Н. Н. Яненко за предоставление возможности участия в заседаниях семинара.

На материал книги оказала большое влияние работа под руководством А. А. Самарского и участие автора в заседаниях рабочей группы по пакетам прикладных программ для задач математической физики при Комиссии Многостороннего Сотрудничества Академий наук стран СЭВ по Проблеме Вычислительная Техника.

Автор пользуется случаем выразить глубокую благодарность А. А. Самарскому, советами и неизменной поддержкой которого он постоянно пользовался.

В обсуждении отдельных разделов рукописи приняли участие Чуянов В. А., Аким Е. Л., Басс Л. П., Карпов В. Я., Корягин Д. А., Королев Л. Н., Мкртмян А. А. Помощь в подборе литературы оказали Мартынюк В. В. и Попов Ю. П. Всем этим товарищам автор благодарен за ценные советы и полезные критические замечания.

## ГЛАВА 1

### ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И ЗАДАЧИ

В этой главе мы рассмотрим программное обеспечение с точки зрения его функций для проведения вычислительного эксперимента. Попутно остановимся на описании аппаратуры в той мере, в какой это требуется для понимания средств программного обеспечения и возможных режимов работы с вычислительной системой. Изложение построим по поколениям вычислительных машин, чтобы представить средства программного обеспечения и организацию решения прикладных задач в развитии \*).

При этом постараемся выделить как в аппаратуре, так и в программном обеспечении те главные элементы структуры, которые претерпевают малые изменения и тем самым определяют черты организации вычислительных работ, общие для ряда поколений машин. Этот подход будем использовать и в дальнейшем: в аппаратуре и программном обеспечении выделяются комплексы структурных элементов с нарастающей степенью детализации. Наиболее общие, наиболее постоянные части структуры оказывают, как правило, наиболее заметное влияние на организацию вычислительных работ и требуют разработки специальных численных методов решения, учитывающих эти особенности структуры.

Заметим, что мы в основном рассматриваем задачи инженерно-физического характера. Однако в качестве примера приводятся и другие задачи. Они должны помочь читателю полнее представить те наборы<sup>1</sup> задач, в окружении которых будет решаться его собственная задача.

---

\*) Подробные сведения по аппаратуре и поколениям машин читатель найдет в [15, 16].

## § 1. О вычислительном эксперименте

**1. Вычислительный эксперимент как метод исследования.** В науке и технике существуют два метода исследования — экспериментальный и теоретический. Основным инструментом теоретических исследований служат математические методы. Еще до появления электронных машин среди математических методов заметное место занимали численные расчеты. К ним прибегали всякий раз, когда аналитические методы оказывались бессильными. С развитием науки и техники потребность в численных расчетах возрастала. Сами расчеты становились очень громоздкими. Для решения задач ядерной физики, гидродинамических расчетов, для конструирования сложных объектов уже нельзя было ограничиться прикидочными расчетами. Требовалось решать уравнения математической физики. Последние обычно выражают законы сохранения (массы, количества движения, энергии, заряда и т. д.) и представляют собой дифференциальные уравнения в частных производных, интегро-дифференциальные или интегральные уравнения. Это уравнения диффузии, газодинамики, электродинамики, кинетические уравнения переноса частиц и излучения и другие.

Решение таких уравнений требовало огромного числа арифметических действий при использовании самых совершенных численных методов. Кстати, разработка численных методов в этих условиях требовала большого искусства. И тем не менее даже большие расчетные бюро на расчет одного варианта тратили много времени (месяцы). Основная трудность состояла в том, что если инженер с помощью логарифмической линейки и номограмм мог «проиграть» десятки вариантов прежде чем выбрать какой-либо элемент простой конструкции, то в случае задачи большого объема такое «проигрывание» становилось невозможным и вычислительный эксперимент провести было нельзя.

Под напором потребностей практики в 50-х годах появились первые электронные машины. С их помощью пределы вычислительного эксперимента существенно расширились.

Когда электронные машины только появились, они, как правило, использовались для численных расчетов. С их помощью решались в основном инженерно-физические задачи. Постановка задач носила модельный характер, позволявший учитывать только принципиальные черты

явлений и процессов. Например, искусственно вводилась симметрия, чтобы можно было ограничиться одномерным расчетом. Эти расчеты позволяли понять принципы происходящих процессов, но использовать их для реального конструирования или для замены реального эксперимента можно было далеко не всегда. Переход к таким задачам оказался возможным благодаря серьезным успехам вычислительной техники и программного обеспечения.

На этом первоначальном этапе для решения каждой отдельной задачи создавались программы. Результаты одного расчета можно было использовать для другого расчета, просмотрев лишь отдельные контрольные числа, даже не проводя промежуточную выдачу. Изменилась и сама трактовка понятия задачи: задача в современном понимании — это комплекс относительно независимых задач.

Дальнейшее расширение круга задач, которые можно исследовать методами вычислительного эксперимента, связано с разбиением исходной задачи на «модули» — отдельные части. Модульная структура лежит в основе самих физических явлений, математических и вычислительных методов. Из таких модулей, как из отдельных «кирпичиков», можно «собрать» программу для решения задачи, если она относится к некоторому классу, для которого заранее построена «библиотека» модулей.

Такой подход мог бы решить основную трудность, возникшую при использовании методов вычислительного эксперимента и связанную с тем, что даже незначительные изменения постановки реального эксперимента иногда приводят к серьезной переделке алгоритма и программ. В таком случае работу над численным алгоритмом и программами приходится начинать заново. Модульные системы, позволяющие легко менять набор модулей, обслуживают некоторый класс задач и тем самым существенно расширяют возможности вычислительного эксперимента. (Модульные системы иногда называют пакетами прикладных программ, хотя это и не совсем точно.) Вычислительный эксперимент позволяет предсказать новые физические эффекты. Так был открыт эффект  $T$ -слоя: при некоторых условиях в плазме образуется самоподдерживающаяся область высокой температуры, в которой сосредоточен джоулев нагрев. Через 5—6 лет  $T$ -слой был обнаружен в физических экспериментах [150].

Следует иметь в виду, что выделение модулей на всех этапах вычислительного эксперимента применяется

очень давно. Новое здесь в обеспечении сборки на этапе программирования с затратами малых усилий, что достигается созданием специальных системных средств.

Применение методов вычислительного эксперимента характеризуется многими чертами развитого индустриального производства. Однако, в отличие от других видов научно-производственной деятельности, «вычислительная индустрия» еще не сформировала своих принципов с той степенью четкости, которая имеется, например, в машиностроении. И тем не менее уже настало время говорить о технологии решения задач на электронных машинах, о стандартах в практике вычислений, вполне сформировались отдельные службы вычислительных центров. Все чаще стали появляться такие термины, как аван-проект, техзадание, план-график, технология вычислительных работ, разделение труда, экономическая эффективность, появились первые стандарты и т. д.

Если проследить за развитием производительности труда в других отраслях, то можно выделить условно такие стадии: простая механизация, заменяющая простой труд, в нашем случае это — создание электронного арифмометра, т. е. электронных машин. Вторая стадия — автоматизация, здесь — это создание средств автоматизации программирования и, наконец, «индустриализация», позволяющая объединить усилия многих специалистов с тщательной обработкой трудоемких операций, многосерийным характером производства, созданием специализированного оборудования, широким разделением труда и т. д., в нашем случае — это создание вычислительных комплексов и сетей ЭВМ, специализация таких комплексов и программного обеспечения, разбиение процесса решения задачи на этапы, внедрение стандартизации, широкое развитие пакетов прикладных программ, банков данных и многое другое.

Из всего цикла вопросов мы выберем вопросы организации вычислительных работ по решению некоторого класса задач, причем под организацией мы будем понимать способы построения физических, математических, вычислительных моделей, методы построения соответствующих программных средств, проведения работ по программированию. И хотя многие высказывания могут относиться к широкому кругу задач, все же будем ориентироваться на задачи инженерно-физического характера, использующие предельные возможности существующих

машин. Остальными вопросами будем интересоваться лишь постольку, поскольку они относятся к нашей основной теме.

Появление новых мощных средств вычислительной техники и программного обеспечения качественно меняет весь стиль работы с машиной. К таким средствам относятся системы коллективного пользования, позволяющие совместно использовать как машину, так и банки данных, обмениваться информацией через машину [7], использовать модульные библиотеки, информационно-справочные системы и многое другое. В этих случаях приобретают большое значение устройства связи «человек — машина». Разумеется, это не полный перечень всего нового, что характеризует современный этап применения вычислительной техники. И все же базовой проблемой прикладного применения вычислительной техники является решение одной задачи, точнее задач некоторого класса.

Развитие вычислительной техники ставит все новые и новые проблемы, которых раньше не существовало. Например, решение задачи, полученное как функция от четырех переменных и представленное в виде таблиц и графиков, мало что дает исследователю; миллионы цифр нельзя ни истолковать, ни понять. Возникает сложный вопрос представления результатов расчета в удобном виде. Вообще исходная информация к задаче и результаты расчета в реальных постановках приобретают новый смысл. Правильная организация структуры этой информации, использование соответствующих аппаратных и программных средств создают своего рода «интерфейс» между исследователем и задачей. Здесь особенно существенны такие аппаратные средства, как устройства машинной графики, устройства непосредственного ввода информации в машину по линиям связи от устройства реального эксперимента. Не меньшую роль играют коллективные банки данных и др. В дальнейшем нас будут интересовать только вопросы структуры данных и форматы выданных.

**2. Технологический цикл проведения вычислительного эксперимента.** Перейдем к описанию работ для реализации вычислительного эксперимента [8, 9]. Решение задачи начинается с физической постановки или, говоря более современным языком, с формулировки физической модели. На этом этапе конкретизируются предположения, при которых будет решаться задача, рассматривают-

ся пределы изменения тех или иных величин, выделяющих некоторый класс задач, формулируются уравнения для описания процессов.

На этом же этапе формулируется математическая модель. Если уравнения уже были сформулированы, то могут добавляться некоторые дополнительные условия, выделяющие единственное решение, вводятся некоторые аналитические преобразования, выясняется характер зависимости решений от исходных данных и т. д.

Затем приступают к разработке дискретной модели. Для этого проводятся разработки численных методов решения задач. Численные методы сводят задачу к действиям, непосредственно выполнимым на машине. Это очень ответственный этап. Помимо выбора метода проводятся разнообразные преобразования, с помощью которых алгоритм решения приводится к счетному виду. Этот этап разработки дискретной модели в настоящее время только начинает исследоваться и пока целиком определяется искусством и интуицией разработчика.

Затем приступают к разработке программ. Современные программы редко пишут в виде жесткого комплекса для решения одной фиксированной задачи, точнее, разных ее вариантов. Чаще всего это некоторые пакеты, позволяющие обслуживать определенный класс задач, отличающихся по физической постановке, математическим или дискретным моделям. Здесь исходный класс задач представляется в виде набора модулей. Такой набор модулей представляет собой модель класса задач. Собирая из модулей те или иные конфигурации, получают конкретные задачи из данного класса. Для построения программного комплекса могут быть использованы специальные системные средства или средства серийного программного обеспечения. При создании таких систем, так же как и при разработке отдельного программного комплекса, выполняется некоторый цикл работ. Он обычно включает разработку вычислительной схемы счета или просто схемы счета. Схемы счета устанавливают последовательность выполнения модулей. В некоторых случаях с помощью специальных системных средств последовательность модулей может определяться автоматически [10]. Все же для больших задач автоматизация с трудом уживается с эффективностью [11] и схемы строятся разработчиком.

Решаются вопросы распределения памяти на всех уровнях. Это одна из наиболее сложных задач, она тесно свя-

зана с численными методами. Затем приступают к сегментации программы и данных. Для этого выделяют группы блоков, которые должны одновременно находиться в памяти, и строят схемы очередности загрузки сегментов программ и данных. Эта работа тесно увязана с распределением памяти. Далее строится схема обмена данными между блоками. Разрабатывается структура входных и выходных данных, форматы выдaч.

В процессе работы над вычислительным экспериментом, начиная с математической модели, разрабатывается схема отладки и тестирования. Особо большое значение уделяется схемам отладки при разработке программ. Установлено, что тщательная подготовка к отладочным работам на всех этапах всегда окупается. Иными словами, «машинный эксперимент», т. е. выход на ЭВМ должен быть тщательно спланирован. Само по себе число выходов на машину в расчете на определенное число отлаженных предложений программы определяет зрелость коллектива. По некоторым источникам [3] на отладку может уйти до 60% общего времени, потраченного на программирование.

Следующий этап — это проведение расчетов. Первые расчеты носят характер опытной эксплуатации, после них проводятся рабочие или производственные расчеты.

Полученные результаты анализируются и в результате обычно возникает необходимость вносить те или иные изменения в модели, так что работа носит «итерационный» характер [9]. Очень важен «календарный отрезок времени, который уходит на каждую «итерацию». Он в конечном счете и определяет полное время, потраченное на проведение вычислительного эксперимента.

Решение больших задач требует внимательного отношения к особенностям вычислительной техники и программной «среды», в которой будет решаться задача. Неправильный учет свойств операционной системы, или системы программирования, может привести к неоправданно большому времени пребывания задачи в машине для завершения расчетов, или к слишком большому времени занятости процессора. На эти же параметры может оказать влияние и порядок работы вычислительного центра, где будет решаться задача. Зачастую для большой задачи надо создавать специальные условия работы. Необходимость этих условий должна быть четко сформулирована.



Как видно, организация вычислительных работ — это комплексная задача, требующая для своего решения системного подхода.

При организации вычислительных работ большое значение имеет документация на разработку и стандартизация.

Документация должна разрабатываться параллельно с проведением разработки. Подготовка документации требует большого труда [3]. Этот труд в большой части носит рутинный характер. Для его облегчения целесообразно воспользоваться некоторыми специальными средствами программного характера, с целью возложить эту работу на машину. Не менее важно решить и административные вопросы организации работ над документацией, среди которых видное место занимают вопросы защиты авторских прав разработчиков.

В настоящее время уже появились первые стандарты, направленные на совершенствование организации вычислительных работ. Важно добиться понимания стандартов как школы передового опыта, как средства, сокращающего «интеллектуальные» затраты. Стандарты помогают избежать многообразия однотипных средств, на изучение и поддержание которых вычислительные центры и разработчики тратят много времени и труда.

Помимо государственных стандартов много пользы могут принести стандарты других категорий, например, стандарты предприятий, которые могут учесть специфику вычислительных центров и даже отдельных групп задач.

При анализе организации вычислительных работ обращает на себя внимание следующее обстоятельство. Несмотря на большие изменения, которые претерпевает вычислительная техника, в структуре ЭВМ можно выделить достаточно устойчивые элементы, оказывающие влияние на организацию вычислительных работ. В качестве примера можно привести многоуровневый характер памяти машин. Эффективный учет структуры памяти должен проводиться не только в процессе программирования, но и на стадиях математической и дискретной моделей. Разработка вычислительных методов, учитывающих структуру памяти машин, наряду с модульным анализом — важная задача [5].

В последние годы широкое развитие получили пакеты прикладных программ, являющиеся средствами реализации вычислительного эксперимента. Пакеты решают та-

кие важные вопросы, как накопление фонда программ в виде модульных библиотек, быструю сборку из модулей нужной задачи, позволяющей при малом изменении физической постановки задач получать результаты малыми затратами труда и времени, независимость разработки отдельных модулей, адаптация библиотек к другим машинам и т. д.

Для решения этих задач в пакетах предусмотрены специальные системные средства, или же могут быть использованы некоторые универсальные средства. Задача этих средств — установить между модулями по заданию пользователя, написанному на специальном языке пакета, управляющие и информационные связи. Если учесть требование — обеспечить высокую эффективность, — решение подобных задач требует весьма тонких системных средств и на сегодня не нашло еще полного решения.

Первые работы по вопросам, связанным с автоматическим построением вычислительных алгоритмов, относятся к 1965 г. [12, 13] и принадлежат Г. И. Марчуку. Однако до сих пор в этой области продолжается период становления. По многим принципиальным вопросам, включая, например, определение пакета, нет еще единого мнения. Это находит отражение при изложении принципов организации пакетов ([8, 9, 5, 14] и другие работы).

Естественно, при изложении различных сторон организации вычислительных работ мы не могли добиться исчерпывающей полноты и строгости. Причина в том, что, с одной стороны, некоторые вопросы еще не достигли нужного уровня завершенности, с другой стороны, — изложение с исчерпывающей полнотой даже неполного круга вопросов оказалось бы слишком громоздким и потому малополезным. В связи с этим мы ограничиваемся примерами и обсуждением тех вопросов, на которых надо заострить внимание. По нашему мнению, такой подход должен помочь читателю самому отыскивать узкие места в организации вычислительных работ.

Большое значение для успешного проведения вычислительного эксперимента имеет правильный учет режимов работы вычислительной техники, круг задач, решаемых в данном центре, статистические данные работы центра и многие чисто организационные вопросы. Влияние их на все этапы вычислительного эксперимента очень велико, а соответствующие ошибки для своего исправления требуют больших усилий. Большая часть таких ошибок

приводит к увеличению «календарного» времени решения задачи, к неопределенному в данных конкретных условиях расходу ресурсов. Подобные вопросы подробно обсуждаются в главе 6.

В заключение сделаем некоторые замечания по поводу терминологии. Многие термины, несмотря на длительное употребление, еще не устоялись и в разных источниках имеют разное значение. Мы везде употребляем термин «машина», имея в виду, по существу, вычислительную систему. Под «большой задачей» понимается задача, требующая больших объемов памяти и времени работы процессора и отличающаяся сложной логикой \*). Такие задачи обычно с трудом «помещаются» в машину. Термины «модуль» и «блок» мы практически употребляем наравне, хотя между ними в дальнейшем определяются некоторые различия. Вместо сочетания «программный комплекс» мы употребляем термин «программа». Обычно под «программным комплексом» понимается набор взаимоувязанных программ, каждая из которых решает относительно самостоятельную задачу. Однако употребление более короткого термина «программа» с учетом контекста не вносит трудностей. Слова «математическое обеспечение» мы стремимся заменять «программным обеспечением». Под этим термином понимается набор программных средств, постоянно имеющихся в распоряжении пользователя для любой задачи, если он пожелает ими воспользоваться. В таком же смысле употребляются термины «штатное» обеспечение или «серийное» обеспечение, хотя между ними и имеется некоторая разница.

## **§ 2. Поколения прикладных задач и программного обеспечения**

**1. Первое поколение.** На машинах первого поколения решались в основном вычислительные задачи. Это были одномерные задачи газовой динамики, теплопроводности, нейтронной физики, расчеты одномерной плазмы, задачи упругости и пластичности, задачи фильтрации, электромагнитных колебаний и ряд других. На этом этапе выяснялись принципиальные вопросы физики процессов.

---

\*) Термины большая задача и вычислительный эксперимент, где это не вызывает затруднений, мы употребляем наравне.

1.1. Каждый программист сам писал программу и сам отлаживал ее, как правило, находясь за пультом машины. Неясные места выполнялись в однократном режиме и это давало возможность найти ошибки в программе. Отладка занимала много машинного времени и требовала известного искусства: на пульт выдавались числа и команды в двоичном счислении.

По схеме выполнения команд это были машины последовательного действия, т. е. каждая команда выполнялась только после полного завершения предыдущей. Выборка из оперативной памяти также не совмещалась с выполнением команд. Работа с внешними устройствами не совмещалась с другими операциями. При сбое внешних устройств машина останавливалась и ждала действий оператора.

В архитектурном \*) плане машина состояла из таких узлов: арифметическое устройство, устройство управления (или центральное устройство), внешние запоминающие устройства, устройство ввода и вывода перфокарт, оперативное запоминающее устройство, операторский пульт.

Система команд содержала такие типы команд: специальные (работа с внешними запоминающими устройствами, устройствами ввода-вывода перфокарт, пультом и т. д.), арифметические, логические, команды для работы с регистрами-модификаторами, команды передачи управления. В реализации команд предусматривались команды, выполнявшие арифметические операции с плавающей запятой. В некоторых машинах аппаратно реализовывались операции с плавающей запятой. Известное распространение получили одноадресные машины.

1.2. Программное обеспечение состояло из таких компонент, как библиотеки стандартных программ, компиляторы, интерпретаторы, отладочные системы, автооператоры. Из библиотек стандартных программ отметим ИС-22 для машин типа М-20 [21]. Эта библиотека включала набор таких программ, как перевод чисел из десяти-

---

\*) Архитектура вычислительной системы (машины) — сравнительно новое понятие. Оно включает комплекс общих вопросов построения системы (машины), существенных в первую очередь для потребителя, для которого неважны детали технического исполнения. К архитектуре можно отнести вопросы общей структуры системы, вопросы организации хранения и использования информации и вопросы логической организации совместной работы устройств системы [16].

тичной системы счисления в двоичную и обратно, вычисление основных тригонометрических функций, численного интегрирования дифференциальных уравнений, вычисление определенных интегралов по квадратурным формулам и ряд других.

Недостаток принципа интерпретации состоял в том, что при каждом обращении должен был работать интерпретатор. Последнее вело к большим расходам машинного времени.

Поэтому наряду с этим принципом стал применяться принцип компиляции. В этом случае перед работой программу «пропускали через компилятор», который мог быть оформлен просто как некоторая «шапка» к программе. Просматривалась программа, и везде, где встречалось обращение к стандартной программе (СП), появлялся текст программы на рабочем поле компилирующей системы, настроенный по «месту».

Последнее могло оформляться двояко: либо текст вставлялся непосредственно в то место, где к нему обращались, либо (что проще) происходила отсылка с засылкой последующего возврата к тексту стандартной программы, помещаемой в конце основной программы. В последнем случае можно было обойтись одним экземпляром стандартной программы для обращений из разных мест основной, однако за это приходилось расплачиваться необходимостью проводить предварительную настройку при каждом обращении. Сама настройка могла также проводиться двояко: либо в стандартной программе представлялись в командах конкретные адреса, либо конкретные величины засылались в определенные ячейки памяти данной стандартной программы, а текст ее не менялся.

Компиляция давала экономию машинного времени, причем особенно экономна была жесткая настройка. Однако это вело к удлинению программы, а следовательно, к трудностям с оперативной памятью.

Сравнивая компилирующие и интерпретирующие системы, можно отметить, что как те, так и другие требуют рабочего поля для размещения программ. Однако для компилирующих систем поле должно позволять разместить все требующиеся стандартные программы и поэтому может оказаться большим, тогда как для интерпретирующих систем достаточно иметь поле, способное вместить самую большую СП. При компиляции все подпрограммы, требующиеся в основной программе, вызываются на

рабочее поле в оперативной памяти до выполнения основной программы. Следовательно, компиляция экономит время на служебные операции настройки подпрограмм. При интерпретации расходуется дополнительное время на настройку подпрограмм при обращении к СП. В чистом виде интерпретация и компиляция, как правило, не используются. Применяются системы, сочетающие в себе черты как компилирующих, так и интерпретирующих систем, как это имеет место в системе ИС-22.

Стали использоваться системы, позволяющие составлять из отдельных замкнутых частей программы (программных модулей) большую программу. Для этой цели перед модулем помещался «паспорт», в котором указывались данные, позволявшие автоматически с помощью специальной программы настраивать модуль на определенное место в оперативной памяти, т. е. модуль был перемещаемой программой. Такие программы назывались программами автоматического присвоения адресов (ПАПА) [22].

Большим достижением было появление автокодов. Это были провозвестники ныне многочисленных языков программирования\*). Автокоды ввели в практику программирования буквенные символы и позволили писать программы в привычных обозначениях, что резко сократило количество ошибок. Первые автокоды были автокодами «один в один», т. е. каждая строчка автокодной программы соответствовала одной команде. Для получения программы использовался транслятор, который одновременно проводил поиски простых ошибок, распределял память.

Появились первые системы для отладки программ. Они могли работать по принципу «прокрутки»: из программы извлекались на специальное поле команды, выполнялись, результат выдавался вместе с номером выполняемой команды. Это позволяло частично обойтись без отладки за пультом. Однако «прокрутить» всю программу было тяжело в силу огромного количества выдаваемой информации. Поэтому «прокрутке» подвергались лишь отдельные части программ, содержавшие ошибки. Для локализации таких частей использовались другие про-

---

\*) Языком будем называть определенный набор символов и правил, устанавливающих способы комбинации этих символов для записи осмысленных сообщений (текстов). Языки, предназначенные для записи программ, будем называть языками программирования.

граммы, позволявшие автоматически делать вставки в те или иные места с тем, чтобы при выполнении соответствующих команд можно было провести выдачи нужных величин, адресов выполняемых команд и других сведений, позволявших проследить ход выполнения программы, т. е. провести «трассировку». Отладочные системы позволяли выполнять такие функции: прерывать программу после выполнения команды с некоторым номером и выполнять указанные в отладочной системе приказы (например, выдачу содержимого ячеек на печать, запись в ячейки необходимой информации, передачу управления на заданную команду), выдавать значение номера предыдущей команды, прерывать программу при выполнении некоторого условия на информацию, содержащуюся в ячейках отлаживаемой программы (при записи в данную ячейку, считывании из заданной ячейки, при появлении в ней числа определенного типа и т. д.). Все эти операции были взяты из практики отладки за пультом и по сей день являются основой различных отладочных систем [23].

Большим недостатком была необходимость заранее спланировать всю отладку, поскольку в ходе отладки нельзя было что-то изменить.

Появились системы, позволявшие упростить работу оператора — автооператоры. Суть таких программ состояла в записи заданий на внешний носитель (магнитную ленту или магнитный барабан). После завершения очередного задания в работу вступала программа автооператор, которая либо запускала на счет следующее задание, либо проводила аварийную выдачу, если данное задание кончалось аварийным остановом. Помимо этого, автооператор мог указывать очередность выполнения заданий для машины, например, с целью завершить решение задачи к определенному часу, собрать те или иные сведения о решаемых задачах и оформить их в виде некоторой статистики работы машины.

**1.3.** Перейдем теперь к прикладным программам. На машинах первого поколения в основном выполнялись программы, написанные для решения одной отдельной задачи. При необходимости решать связанную последовательность задач поступали иногда следующим образом. Результаты расчета одной задачи выдавались на перфокарты, а затем вводились в машину как исходные данные следующей задачи. Вместо перфокарт часто использовали магнитные ленты.

В виде исключения уже на машинах первого поколения появились комплексы программ\*). В оперативную память вызывались последовательно блоки комплекса и необходимые для них данные. Характерная особенность таких систем состояла в том, что каждый блок помещался на фиксированные места оперативной памяти, так же как и данные для этого блока. При смене какого-либо блока на его место помещали другой вариант того же блока. При этом надо было отвести под блок память с запасом.

Работа по созданию такого комплекса начиналась с распределения памяти под данные и программы. Затем разрабатывалась схема смены блоков, в каждом блоке оформлялся свой подблок обмена, который управлял вызовом данных с внешней памяти и организацией поочередной загрузки сегментов данных в оперативную память. Несмотря на весьма современный вид такого комплекса, работа по замене блоков оказывалась трудоемкой. Это было связано с отсутствием программных средств, обеспечивающих автоматическое распределение памяти и привязку к конкретным адресам. Все это приходилось делать вручную.

**2. Второе поколение.** Перейдем теперь к характеристике задач второго поколения. Задачи численного счета стали решаться в постановке, более близкой к реальным условиям. Наметился переход к двумерным и трехмерным нестационарным задачам. Усложнилась физическая постановка задач. Существенно изменились численные методы. Определилась направленность численных методов как математической дисциплины, обслуживающей в основном вычислительные машины. Было разработано много методов для решения многомерных задач. Из них отметим важную группу аддитивных методов [24—26]. Существенным шагом явилось широкое распространение программных комплексов для решения группы связанных задач.

На машинах второго поколения стали решаться новые задачи, не решавшиеся на машинах первого поколения. Это — задачи обработки символьной и графической информации. Стали создаваться информационно-поисковые

---

\*) Под комплексом программ (или программным комплексом) мы будем в дальнейшем понимать набор относительно самостоятельных программ, точнее, программ для решения относительно самостоятельных задач.



системы, системы обработки результатов эксперимента, автоматизированные системы управления. То есть уже на этом этапе машины превратились в инструмент обработки любой информации, если эту обработку удавалось формализовать.

Задачи обработки эксперимента обычно приводят к необходимости автоматического ввода в машину данных непосредственно по специальным линиям связи. Здесь характерен огромный объем информации, обработка которой без машин невозможна.

Начали распространяться системы управления технологическими процессами. Такие системы открыли новые перспективы, поскольку позволили проводить обработку необходимой информации в «темпе» самого процесса, что без ЭВМ сделать невозможно.

Среди задач, решаемых в реальном времени, мы в дальнейшем остановимся на задачах баллистического управления космическим полетом.

Решение таких задач началось еще на машинах первого поколения. Уже тогда стало ясно, что они предъявляют к вычислительным системам новые требования: высокая надежность, сложная система приоритетов в обработке заказов, позволяющая нужный заказ обработать в течение заданного промежутка времени, причем повторение в случае сбойных ситуаций здесь недопустимо (как и потеря информации).

Широкое распространение получили задачи машинной графики. К таким задачам будем относить все задачи, связанные с изображением, выдаваемым на внешние устройства машины — дисплей, бумажную ленту (графопостроители), фотосистемы, ксерографические устройства и др.

**2.1.** Машины второго поколения существенно изменили стиль работы пользователей. В дополнение к традиционной индивидуальной работе появились разные формы коллективной работы, для которой использовались выносные пульта (терминалы), а в программном обеспечении — интерактивные системы общения с машиной и системы разделения времени.

Элементы этого были еще в машинах первого поколения. Пульт был один, но когда он передавался в полное распоряжение конкретного пользователя, который мог, используя пультовые команды, следить за выполнением программы и вносить соответствующие коррективы как в

текст программы, так и в содержимое информационных полей задачи, то налицо был интерактивный режим работы. Когда один пользователь сменял другого, который через некоторое время вновь должен был вернуться за пульт, то это было рождением систем разделения времени (СРВ). Причем, если пользователи сменялись через каждые полчаса, то можно было говорить о «квантовании». Разумеется, такие «ручные» СРВ и интерактивные режимы были крайне неэффективны, так как пока пользователь вводил информацию на пульте и размышлял над только что полученными результатами, машина просто бездействовала. Даже при малых мощностях машин первого поколения такая работа была экономически невыгодной.

В машинах второго поколения стало много выносных пультов, за которыми могли независимо друг от друга работать пользователи, не зная ничего друг о друге. Специальное программное обеспечение организовывало работу так, что каждый пользователь получал определенный квант машинного времени и все пользователи обслуживались «по кругу» \*). Если пользователь занят набором информации на выносном пульте или размышлением над полученной информацией, то причитающийся ему квант времени не выбирается и машина передается следующему пользователю. Квант времени выбирается в случае, если задача готова к работе или сам пользователь приготовил задачу для ввода. В силу того, что работа за пультом идет очень медленно по сравнению с работой машины, присутствие других пользователей не ощущается. Более того, остается еще довольно много времени для выполнения задач в «пакетном» режиме, т. е. задач, решаемых в отсутствие пользователя.

Системы коллективного пользования, как правило, допускали возможность отладки синтаксиса программы в интерактивном режиме (редактирование), дистанционного запуска на счет, а также возможность интерактивного общения с задачей в процессе ее решения.

**2.2. Задачи второго поколения с особой остротой поставили вопрос контакта человека с машиной.** Быстродействие и надежность столь возросли, что доля времени, затрачиваемого на подготовку информации и ввод ее в машину, наблюдения за ней в процессе обработки по сравнению с временем решения сильно возросла. Встала

---

\*) Здесь для краткости мы сильно упростили картину.

на повестку дня проблема использования машины как общего инструмента коллективами пользователей.

Это привело к необходимости пересмотреть внешние устройства и их связь с машинами. Существенно пополнился ассортимент внешних устройств: появилась целая серия графических устройств ввода-вывода (дисплеи алфавитно-цифровые, дисплеи графические с карандашами, управляющими шарами и рычагами, устройства для непосредственного ввода графической информации, ксерографические устройства для вывода информации, фотоустройства для вывода, в частности, для микрокопирования, оптические устройства для ввода символьной информации с распознаванием символов); появились новые запоминающие устройства (магнитные диски, фотоустройства); принял широкий размах ввод информации по телеграфным и телефонным каналам непосредственно в память машины; распространение получили выносные терминальные станции, включающие различные наборы оборудования: экранные пульта, ввод с перфокарт или перфолент, магнитных карт и многие другие.

В конструктивном отношении новым было совмещение различных процессов, идущих в машине. Выполнение команды включает в себя процессы выборки необходимой информации из памяти машины, выборки кода самой команды на определенный регистр и многие другие этапы. Для увеличения быстродействия и загрузки устройств машины можно организовать «конвейерную» обработку команд. Само слово «конвейер» связано с аналогией обработки изделий на конвейере: если бы на конвейер не запускалось последующее изделие до тех пор, пока предыдущее не сойдет полностью с конвейера, то возникали бы неизбежные простои всех рабочих постов конвейера, кроме одного, обрабатывающего в данный момент изделие. Загружая все посты, добиваются многократного увеличения производительности всей конвейерной линии. Правда, команды отличаются друг от друга и не все «посты» в одинаковой мере трудятся над каждой командой (как это получается с обычной конвейерной линией, где все изделия одинаковы). Кроме того, результаты обработки одной команды могут потребоваться для обработки другой, вызывая при этом неизбежное сбрасывание всего конвейера. Следовательно, не все последовательности команд могут оказаться при такой структуре равноценными. Переставляя последовательность команд, можно в таких

машинах существенно изменять время выполнения всей последовательности. Программисты и разработчики программного обеспечения обычно стремятся учесть эти особенности конвейерной системы [7].

В структуре оперативной памяти также появились изменения: память стали разбивать на блоки, при этом оказывалось, что «рядом» расположенные данные в программе пользователя располагались по разным блокам оперативной памяти. Выборка же из соседних блоков совмещалась. Начали конструировать специальную буферную сверхбыструю память (кэш-память) [15]. Выборка нужной информации проводилась в эту память заранее. Результаты также заносились в эту память. Выбирался специальный алгоритм, позволяющий заполнять эту память часто используемой информацией. Тогда задачи, связанные с большим количеством операций над узкой группой данных, работали практически в кэш-памяти.

Это позволяло для таких задач уменьшить влияние полного цикла на среднее быстродействие. В то же время векторные операции (например, сложение двух векторов) для каждой операции требовали выборки двух операндов, которые потом на протяжении вычисления всего векторасуммы не использовались. В таком случае выгоды от применения кэш-памяти не получалось.

Идея совмещения использовалась при организации работы внешних устройств. Поскольку внешние устройства работают медленно, нецелесообразно останавливать работу машины в ожидании завершения работы внешнего устройства. Однако для одной задачи трудно написать программу так, чтобы все операции с внешними устройствами были «закрыты» по времени работой основных устройств машины. Именно это привело к мультипрограммированию. Суть его состоит в следующем.

Оперативная память машины распределяется сразу под несколько задач. Пока идет обработка одной задачи центральным устройством машины (процессором \*)), другие либо ждут, либо заняты работой с внешними устройствами. Как только в обрабатываемой задаче возникает потребность в работе с каким-либо внешним устройством,

---

\*) Под процессором понимают арифметическое устройство, устройство управления, а иногда и сверхоперативное запоминающее устройство [16].

решение ее прерывается и машина переходит к решению другой задачи, уже загруженной в оперативную память. Таким образом, для каждой отдельной задачи работа идет так, как будто не существует совмещения: в период работы внешнего устройства, обслуживающего некоторую задачу, процессор занят другой задачей.

Организовать мультипрограммный режим обработки чисто аппаратными средствами оказалось трудным и дорогим. Помимо этого, каждая разработанная конструкция становилась весьма негибкой — ее трудно было приспособить для конкретных условий работы. Поэтому аппаратура стала проектироваться с учетом использования некоей специальной программы, органически дополняющей аппаратуру — операционной системы.

Аппаратура рассчитывалась на выполнение некоторых минимальных функций, причем помимо обычных функций по выполнению команд появились функции, обеспечивающие развитый режим работы с внешними устройствами и мультипрограммный режим. К их числу относятся аппарат приписки адресов, позволяющий физически располагать программу в любом свободном месте памяти, независимо от того, для каких адресов она написана пользователем, аппарат защиты, исключающий порчу информации одной задачи из другой. Для совмещенной работы используется развитый аппарат прерываний: при выдаче некоторого сигнала внешним устройством последовательность команд прерывается. Что делать дальше — решает блок операционной системы, занятый обработкой прерываний.

Аппарат прерываний оказался удобным не только для организации работы внешних устройств, но и для других весьма разнообразных целей (таковы, например, аварийные ситуации, экстракоды — специальные подпрограммы и др.) [27, 28].

В архитектурном плане машина второго поколения стала выглядеть так:

- процессор — устройство выборки и обработки команд;
- запоминающие устройства — тот же смысл, что и в первом поколении;

- внешние устройства;

- устройство управления внешними устройствами — блоки, занятые предварительной подготовкой информации и осуществляющие связь внешних устройств с центральными блоками машины.

Устройства управления внешними устройствами постепенно превращаются в специализированные машины-спутники для работы с каналами.

**2.3.** Дадим обзор систем программного обеспечения машин второго поколения. Операционные системы были довольно разнообразны — от простейших автооператоров до систем, управляющих всей работой машины. Такие системы включали организацию мультипрограммного режима работы, управление вводом-выводом, обменами, организацию пакетного режима работы, работ в интерактивном режиме, некоторые специальные возможности (аппарат передачи сообщений между задачами, аппарат процессов и событий, аппарат главной и подчиненной задач, статистика по задачам и устройствам, программные системы контроля аппаратуры машин и многое другое). Таким образом, операционные системы машин второго поколения достигли больших возможностей [28].

Системы программирования включали в себя трансляторы с различных языков программирования, а также средства работы с программами как протранслированными (загруженными), так и в виде первоначальных текстов (редакторы), библиотеки стандартных программ, средства отладки.

Появление различных языков программирования явилось развитием идей автокодов. Однако автокоды способны больше отражать специфику машины, чем задачи. Проблемные языки как раз стремились отразить специфику задач. Программа, написанная на таком языке, была удобочитаема, наглядна и коротка. Создание программы (транслятора) для перевода текста в машинные команды было трудным делом. Тем не менее на машинах второго поколения насчитывались десятки работающих трансляторов. Еще больше было проблемных языков: созданы были языки для инженерно-технических расчетов и решения научных задач (фортран, алгол), для экономических задач (кобол), для обработки символьных текстов (лисп), для задач моделирования (симула, санскрит), для системного программирования (блисс, астра, паскаль) и др.

На первых порах трансляторы работали автономно. После обработки текста на языке шла выдача текста в машинных командах на перфокарты или на магнитный носитель. Однако вскоре выяснилось, что программисту этого мало. Программу не всегда было целесообразно

писать всю на одном языке. Отдельные части могли, например, требовать написания на автокоде (или даже в машинных командах). Большие программы было целесообразно разбивать на отдельные блоки с трансляцией каждого такого блока отдельно. Последнее удобно, так как при изменении одного блока не надо перетранслировать другие. Возникла необходимость организовать личный архив из таких блоков. Нужны были средства для внесения изменений в тексты программ (редакторы текстов), средства компоновки готовых блоков с последующим запуском готовой программы на счет. Все эти задачи стали выполнять системы программирования. Они содержали трансляторы, переводившие блоки программы на язык машины в перемещаемом виде (модули загрузки), загрузчик, проводивший настройку модулей, транслятор с языка заданий, некоторые средства для ведения личного архива и работы с ним.

Библиотеки стандартных программ насчитывали сотни программ. Как правило, библиотека ориентируется на ту или иную систему программирования. Она включает тексты программ на принятом в библиотеке языке (иногда допускается писать тексты на нескольких языках), или модули загрузки, путеводитель по библиотеке, систему описаний программ (или алгоритмов).

Библиотеки подпрограмм могут содержать подпрограммы на исходном языке, т. е. на том языке, на котором они создаются программистом. Библиотеки модулей загрузки содержат транслированные подпрограммы, оформленные в виде модулей загрузки, допускающих настройку по месту при любом расположении в оперативной памяти. Модули загрузки должны допускать установление связей с другими подпрограммами. Для этого в модуле загрузки нужно вводить «внешние» адреса, значения которых определяются положением других подпрограмм, внутренние адреса, зависящие от положения модуля в оперативной памяти и абсолютные адреса, значения которых не зависят от положения самого модуля. Если модули загрузки оформляются по единому образцу, то с помощью специальных программ — «редактора внешних связей» и «загрузчика» — можно осуществлять связывание модулей и настройку их по месту. Существуют также библиотеки абсолютных модулей, т. е. готовых к исполнению программ. Такие библиотеки содержат системные программы или прикладные программы пользователей. Последние

могут использовать такие библиотеки как временные — на время решения задачи.

Системы разделения времени (помимо обеспечения нижнего уровня, являющегося частью ОС) включают различного рода редакторы, отладочные системы, средства дистанционного запуска задач, информационно-справочные системы и некоторые другие.

Начинают появляться архивы коллективного пользования. Основная задача архивов дать пользователю возможность обращаться к информации на внешних носителях по имени и предоставить ему возможность обмениваться информацией с другими пользователями непосредственно через машину. Все заботы по распределению памяти на внешних носителях, сохранности информации, по установке на машину нужных носителей берет на себя программное обеспечение архива. Помимо этого, архивы позволяют более экономно использовать внешние носители.

**2.4. Решение прикладных программ на машинах второго поколения,** как правило, сопровождается созданием программного комплекса. Программный комплекс строится из отдельных блоков. Однако, в отличие от первого поколения, здесь уже существуют программные средства, позволяющие существенно упростить сборку блоков в программный комплекс. В качестве примера таких средств можно назвать мониторинговую систему Дубна [29]. Однако в целом работа по объединению блоков в комплекс хотя и не требует кропотливой возни с машинными адресами, тем не менее все же очень трудоемка. Нужно тщательно распределять память в общих блоках, внимательно согласовывать формальные и фактические параметры. Эта работа обычно проводится с помощью текстового редактора, имеющегося в системе. При такой сборке возможно возникновение трудноулаживаемых ошибок. При большом количестве модулей сборка может занимать очень большое время.

**3. Третье поколение.** В настоящее время широкое распространение получают машины третьего поколения.

Задачи, приводящие к численным расчетам, теперь стремятся решать в постановке, близкой к реальным условиям.

С точки зрения численных методов здесь увеличивается размерность задач. Подобные расчеты вполне можно назвать численным экспериментом. Для численного экспе-



римента уже мало решать различные варианты задачи, соответствующие одной и той же физической постановке, пусть даже весьма полно отражающие реальную конструкцию. Нужно создать специальный проблемный комплекс программ, позволяющий без больших затрат времени переходить от одной физической постановки к другой. Только в этом случае действительно можно экспериментировать. К таким задачам относятся задачи энергетики, управляемый термоядерный синтез, расчеты конструкций реакторов и кампаний реакторов, моделирование погоды, задачи охраны окружающей среды, расчеты строительных конструкций, многомерные задачи гидродинамики, конструкционные расчеты различного рода изделий.

Системный характер начинают принимать и другие задачи обработки информации. Переход от упрощенных моделей для изучения эффектов к реальным моделям наблюдается и здесь. Для этих задач характерны высокие требования к быстродействию. Даже скорость в 100 миллионов операций в секунду уже не обеспечивает решения многих задач.

3.1. Широкий размах принимает использование систем коллективного доступа, различные диалоговые системы, вместе с тем пакетный режим обработки не теряет своего значения. Дальнейшее развитие получают средства общения человека с машиной. Основные функции устройств, правда, те же, но возможности их выше, а цена — ниже. Это содействует распространению таких устройств. Меняется сам метод подключения внешних устройств к машинам. Последние имеют специально разработанные интерфейсы (способы подключения) для различных внешних устройств, зачастую осуществляемые с помощью машин-спутников.

Таким образом, для третьего поколения характерны комплексы машин, которые мы будем именовать вычислительной системой.

В машинах третьего поколения развиваются методы микропрограммирования. Микропрограммирование дает двойное преимущество. Во-первых, можно при разработке системы команд машины ограничиться достаточно примитивным их набором, что значительно упростит аппаратуру. Во-вторых, имеется возможность с той же аппаратурой быстро переходить от одной системы команд к другой.

Дальнейшее развитие получили аппараты для реализации мультипрограммной работы (прерывания, защита памяти, автоматическое распределение ресурсов, виртуальная память и т. д.).

В обработке команд появляется магистральный принцип, существо которого состоит не только в совмещении выполнения команд, но и в совмещенном выполнении одной команды над группой операндов (типа умножения векторов).

3.2. К программному обеспечению, описанному ранее, следует добавить развитие универсальных языков (типа алгол-68, симула-60, PL/1), банков и баз данных, а также развитие программного обеспечения для работы в системах коллективного пользования.

Остановимся на понятии универсальных языков. До сих пор речь шла о машинно-ориентированных и проблемно-ориентированных языках. Машинно-ориентированные языки включают такие виды языков: машинный язык, автокод (или мнемокод), макроязык.

Машинный язык использует систему команд машины и цифровые адреса операндов. Мнемокод заменяет цифровые коды операций буквенными, а адреса операндов буквенно-цифровыми. Он существенно облегчает составление программ, автоматизируя работу по распределению памяти. Это позволяет легко менять положение программы в памяти машины и дает большие удобства по разделению большой программы на части с последующей сборкой ее в единую программу уже на этапе загрузки.

Макроязык наряду с символическими аналогами машинных команд, из которых состоит мнемокод, допускает также использование макрокоманд. При трансляции каждая макрокоманда заменяется на несколько машинных команд. Это сокращает размеры программ, делает их более обозримыми.

Вторую группу языков правильнее было бы назвать машинно-независимыми языками, оставив термин «проблемно-ориентированные» для языков, описывающих задачи. Тогда языки типа алгол-60, фортран, кобол, снобол, лисп и др. можно отнести к процедурно-ориентированным языкам, т. е. языкам, призванным для описания алгоритмов. Такие языки, естественно, должны ориентироваться на те или иные классы задач. Обилие языков всех групп вызывает ряд трудностей. Происходит уве-

личение числа типов машин, что увеличивает количество машинно-зависимых языков. Распирение сферы применения машин ведет к нарастанию количества процедурно-ориентированных языков. Даже на одной машине оказывается в ходу несколько языков. Замена машинного парка приводит к необходимости заново перерабатывать все системные и прикладные программы.

Выход из этих затруднений можно искать на пути создания серий машин, совместимых на программном уровне, и разработки универсальных языков.

Универсальные машинно-ориентированные языки должны объединить в себе черты некоторой группы машинно-ориентированных языков, обслуживающих соответствующую группу машин. Примером такого языка может служить язык алмо [21]. Универсальный язык программирования объединяет средства различных языков и учитывает возможности его реализации на существующих машинах. Можно развить два направления конструирования языков. Первое направление представляет подбор фундаментальных средств, на основе которых пользователь сам конструирует нужные ему варианты процедурно-ориентированных языков. Наиболее близок к этому типу язык алгол-68.

Другое направление предусматривает набор уже готовых средств, имеющихся в тех или иных языках, так что пользователь может сам выбирать нужные ему средства. По такому типу построен язык PL/1.

**4. Четвертое поколение.** В настоящее время намечается переход к машинам нового, четвертого поколения, если судить по степени интеграции элементной базы.

**4.1.** В отношении задач, которые предстоит решать на этих машинах, можно добавить только необходимость создания средств связи «человек — задача». Речь идет об обработке результатов расчета и приведения их в в легко обозримую форму.

Не менее важно пересмотреть технологию решения задач. Именно модульное программирование позволит легко переходить от одной постановки задачи к другой и тем самым проводить численный эксперимент.

**4.2.** Поскольку и второе поколение пока не исчерпало себя, трудно говорить о четвертом. Тем не менее можно согласиться, что в будущем получают развитие многопроцессорные вычислительные системы. Архитектура таких систем может быть различной. Например, общая память

и множество процессоров, работающих на нее, или одинаковые арифметические устройства, работающие под управлением одного центрального устройства управления (матричные системы) [17].

Широкое распространение примут сети машин.

Многопроцессорные системы ставят перед вычислительной математикой задачи разработки алгоритмов параллельных вычислений. Эти задачи крайне сложны. Здесь могут потребоваться новые средства программного обеспечения для автоматического распараллеливания алгоритмов. Без этих разработок использование параллельных и матричных систем вряд ли даст нужный эффект.

4.3. О программном обеспечении машин четвертого поколения мы не будем говорить. Соответствующие прогнозы можно найти в [151].

4.4. Еще во втором (и даже первом) поколении в технологии решения прикладных задач появились элементы пакетов прикладных программ. При этой технологии основное внимание уделяется простоте сборки больших программ из отдельных блоков или модулей. Второй аспект пакетов связан с возможностью накопления независимо созданных модулей, и следовательно, с объединением усилий больших коллективов программистов. Для оценки роли пакетов полезно привести классификацию, связанную с технологией использования ЭВМ для решения прикладных задач. Расчеты проводились и до появления машин, однако исследовательская, экспериментальная роль таких расчетов была невелика. Переход от одного варианта расчета к другому требовал практически такого же труда, как и расчет первого варианта. Таким образом, расчеты были инструментом вычисления конкретной величины.

С появлением ЭВМ после разработки алгоритма и и создания программы переход от одного варианта задачи к другому стал требовать малого дополнительного труда. Это сразу превратило расчеты в инструмент исследования эффектов численным моделированием. Вместе с тем при переходе к другой постановке задачи, хотя бы и очень близкой, требовалось создавать новый программный комплекс. Эта работа требовала столько же времени, сколько и работа по созданию первоначального комплекса. Поэтому при такой технологии вычислительный эксперимент не мог заменить реальный. Действитель-

льно, небольшие изменения физического эксперимента в численном эксперименте могут привести к необходимости, например, решать нелинейные уравнения вместо линейных. Разработанный ранее комплекс оказывается нерабатоспособным. Именно поэтому вычислительный эксперимент отставал от реального. Преодоление этих трудностей вычислительного эксперимента лежит на пути создания модульных систем. Такие системы позволяют накапливать программы, написанные разными коллективами разработчиков и автоматически из них конструировать необходимый комплекс. Эти два слагающие — коллективный труд над библиотекой модулей и автоматическое конструирование комплекса — позволяют с помощью модульной системы охватить некоторую предметную область.

Мы уже говорили о некоторой условности классификации задач по поколениям. Так, например, на машинах второго поколения сегодня успешно решаются задачи вычислительного эксперимента. Создаются пакеты, использующие машины второго поколения. Если говорить о постановке задач, то для наиболее мощных машин второго поколения характерна постановка, достаточно приближенная к реальной. Так, например, решаются нестационарные задачи в трехмерной геометрии.

Если иметь в виду задачи, то для них характерно отсутствие резких границ при смене поколений машин [18].

## ГЛАВА 2

### **ПРИМЕР ЗАДАЧИ ВЫЧИСЛИТЕЛЬНОГО ЭКСПЕРИМЕНТА ИЗ ОБЛАСТИ НЕЙТРОННОЙ ФИЗИКИ**

Задачи расчета реакторов представляют собой хороший пример вычислительного эксперимента. Для них характерно большое разнообразие: расчеты констант, нейтронно-физические расчеты, конструкционные, расчеты теплопереноса и др. Практически любой реакторный расчет служит для решения комплекса задач и поэтому соответствующая программа представляет собой программный комплекс.

За длительную историю развития расчеты реакторов достигли известного совершенства по организации вычислительных работ и все новое в этой области находит в них отражение, в первую очередь по сравнению с другими применениями вычислительной техники. Так, задачи расчета реакторов были одними из первых, где появились программные комплексы вместо решения последовательности задач, а затем и модульные системы [10, 11, 30—34].

Задачи эти отличаются большой сложностью организации вычислительных работ. Они, как правило, используют максимальные возможности существующих вычислительных средств [11]. Здесь эффективность играет решающую роль, особенно, если учесть массовость расчетов.

Для расчета реакторов вычислительный эксперимент представляет собой единственный путь исследования: реальные эксперименты очень дороги, а в ряде случаев, как, например, в кампании реакторов, такие эксперименты потребовали бы недопустимо большие времена для своего осуществления. Помимо этого, количество реаль-

ных экспериментов ограничено производственными мощностями, а сроки проведения работ достаточно жесткие, так как они связаны с энергетическими проблемами [35].

В главе приводится подробный разбор задачи расчета критических параметров реактора в упрощенной постановке. Описание физических основ дано в сжатом виде и рассчитано на читателя, не знакомого с предметом. Знание физических основ необходимо разработчикам вычислительного эксперимента для грамотной организации вычислительных работ, причем даже тем, которые непосредственно физической моделью не занимаются.

При изложении мы опустили многие важные вопросы расчета реакторов. В тексте главы не приводятся ссылки ввиду классического характера излагаемого материала. Читатель найдет соответствующие материалы в [36—38]. Численные методы изложены в работах Г. И. Марчука [39]. Вопросы общего характера, касающиеся ядерной энергетики, ее проблематики и перспектив, можно найти в книге А. П. Александрова [35]. При решении дифференциальных уравнений в практике реакторных задач широко применяются разностные методы. Теорию и практику разностных методов можно найти в книгах А. А. Самарского [40, 41, 4].

## § 1. Источники ядерной энергии

Как известно, ядра всех элементов состоят из частиц двух типов: нейтронов  $n$  и протонов  $p$ . Заряд ядра  $Z$  (а следовательно, порядковый номер его в таблице Менделеева) определяется количеством протонов, а массовое число ядра  $A$  — общим количеством нейтронов и протонов в ядре:

$$A = Z + N. \quad (2.1)$$

Протоны в силу кулоновского взаимодействия стремятся оттолкнуться друг от друга. Поэтому ядра не могли бы существовать в виде устойчивых образований, если бы на близких расстояниях не возникали ядерные силы связи. Эти силы начинают проявляться при сближении частиц на расстояния  $\approx 10^{-15}$  м (характерные размеры атома  $10^{-10}$  м). Если провести гипотетический опыт и удалить нуклоны в бесконечность, то придется затратить некоторую энергию, называемую энергией связи. Отнесем эту энергию к одному нуклону и построим кривую

зависимости этой энергии от массового числа ядра  $A$  (рис. 2.1).

Масса измеряется в атомных единицах массы (а.е.м.— это  $1/16$  массы атома кислорода  ${}_8\text{O}^{16}$  \*), что составляет  $1,667 \cdot 10^{-27}$  кг). Энергия измеряется в электрон-вольтах (эВ) или миллионах электрон-вольт (МэВ). Напомним,

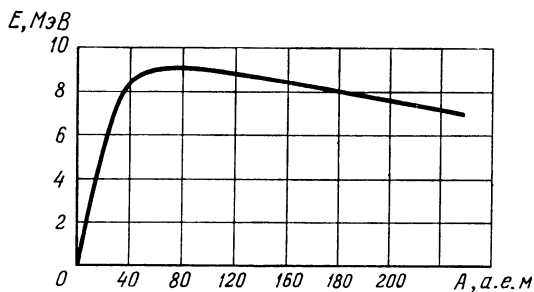


Рис. 2.1.

что масса нейтрона  $m_n = 1,008985$  а.е.м., масса протона  $m_p = 1,007593$  а.е.м., масса электрона  $m_e = 0,000548$  а. е. м.

Из этих чисел видно, что если подсчитать массу ядра как сумму масс нуклонов\*\*), мы не получим массы ядра, измеренной экспериментально (например, с помощью масс-спектрометра).

Последнее легко объяснить с помощью формулы теории относительности:

$$E = mc^2. \quad (2.2)$$

Масса «перешла» в энергию связи нуклонов в ядре.

Обратившись к рис. 2.1, можно указать два пути высвобождения ядерной энергии. Из легких ядер с массами, меньшими 60, образуются более тяжелые ядра. Этот путь называется синтезом. Второй путь — деление тяжелых ядер с образованием осколков. Масса таких ядер должна быть больше 60 а.е.м. Если возникают стабильные изотопы, то выделяющуюся энергию можно

\*) Справа вверху от химического символа элемента указывается массовое число, слева внизу заряд ядра. Элементы с одинаковым зарядом ядра  $Z$ , но разными массовыми числами  $A$ , называются изотопами.

\*\*) Влияние электронов мы не рассматриваем.



подсчитать по изменению энергии связи на нуклон в исходных и конечных продуктах реакции. Можно также воспользоваться изменением масс исходных и конечных ядер с помощью формулы (2.2).

Вывсвобождаемая энергия, если относить ее к килограмму исходных продуктов (удельное энерговыделение), для обоих случаев очень велика.

По сравнению с обычным топливом (например, уголь  $E_{\text{уд}} = 3 \cdot 10^6$  Дж/кг) здесь удельное энерговыделение в 10 млн. раз выше.

Для использования реакций синтеза серьезным препятствием явилось кулоновское отталкивание ядер. Чтобы заставить ядра сблизиться на расстояния  $10^{-15}$  м, нужны большие запасы кинетической энергии. Если использовать энергию теплового движения, то нужны огромные температуры — многие миллионы градусов. Кроме того, для заметного выделения энергии необходимо обеспечить известную плотность. Есть еще третье условие — в подобном состоянии вещество должно существовать достаточное время, чтобы реакции синтеза успели произойти в достаточном количестве. Возникает проблема удержания высокотемпературной плотной плазмы. В настоящее время работы в этом направлении ведутся очень интенсивно.

## § 2. Ядерные взаимодействия и сечения

Для описания взаимодействия между частицами и ядрами вводится понятие эффективного сечения взаимодействия, плотности и потока.

Пусть у нас каждая точка пространства задается радиусом-вектором  $\mathbf{r}$ . Тогда плотность частиц некоторой природы можно обозначить через  $n(\mathbf{r})$ , а количество частиц в бесконечно малом объеме  $d\mathbf{r}$  через  $n(\mathbf{r})d\mathbf{r}$ . Под  $d\mathbf{r}$  понимаем сокращенное обозначение элемента объема, содержащего внутри конец вектора  $\mathbf{r}$ . В декартовой системе координат  $d\mathbf{r} = dx dy dz$ .

Если вместо геометрического пространства рассматривать пространство скоростей, то так же можно определить плотность частиц в этом пространстве относительно скоростей  $\mathbf{v}$ . Ясно, что частицы, имеющие скорость, заключенную в объеме  $v, dv$ , могут пространственно быть разнесенными на большие расстояния,

Наряду с трехмерным пространством можно рассмотреть шестимерное пространство координат и скоростей  $r, v$  и плотность  $n$  в шестимерном пространстве  $r, v$ . Произведение  $n(r, v)drdv$  имеет смысл числа частиц в шестимерном объеме  $drdv$  в окрестности шестимерной точки  $r, v$ .

Для пространства скоростей применяют запись  $v = v\Omega$ , где  $\Omega$  есть единичный вектор, по направлению совпадающий с вектором скорости:

$$v = v\Omega. \quad (2.3)$$

Очень часто в нейтронных задачах система координат пространства скоростей задается в каждой точке  $r$ : тогда один и тот же вектор  $v$  будет иметь разные координаты в зависимости от того, для какой точки  $r$  он рассматривается.

Например, если для геометрического пространства ввести сферическую систему координат с  $r, \theta, \varphi$ , то в каждой точке определяются три взаимно перпендикулярных единичных вектора  $I_r, I_\theta, I_\varphi$ . Направление этих векторов показано на рис.2.2. На базе векторов  $I_r, I_\theta, I_\varphi$  можно построить сферическую систему координат, направив полярную ось вдоль  $I_r$ .

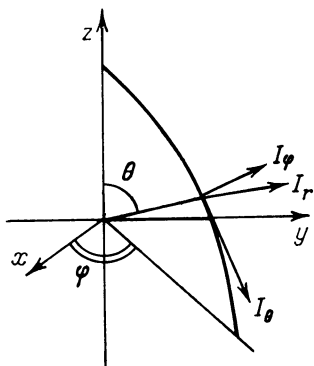


Рис. 2.2.

Если плотность  $n$  меняется в зависимости от времени, то говорят о нестационарном случае  $n(t, r, v)$ .

Поток частиц — это векторная величина  $n(t, r, v)v$ . Поток используется для подсчета числа частиц, проходящих через некоторую поверхность  $S$  в единицу времени

$$\int_S n(t, r, v)v ds. \quad (2.4)$$

Здесь  $ds$  — векторный элемент поверхности, т. е.  $d\sigma n$ , где  $d\sigma$  — элемент поверхности, а  $n$  — единичная нормаль к поверхности.

Для подсчета числа ядерных реакций часто используется понятие плотности потока. Так определяется величина

$$n(t, r, v)v, \quad v = |v|. \quad (2.5)$$

Пусть на единичную площадку падает одна частица (рис. 2.3). Если на каждую единичную площадку приходится  $N_{\text{я}}$  ядер, то вероятность ядерного взаимодействия

$$P = \sigma N_{\text{я}}$$

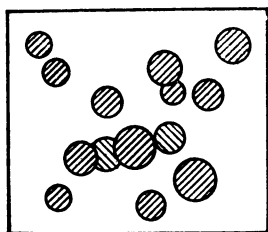


Рис. 2.3.

$$P = \sigma N_{\text{я}}. \quad (2.6)$$

Коэффициент пропорциональности  $\sigma$  и называется эффективным сечением взаимодействия некоторого типа. Он имеет размерность площади  $\sigma$  [м<sup>2</sup>]. Можно представить себе для упрощения, что каждое ядро окружено некоторой сферой влияния с площадью диаметального сечения  $\sigma \text{ м}^2$ , и если

частица попадает в пределы этой сферы, то взаимодействие происходит. Тогда формула (2.6) может быть истолкована так:  $\sigma N_{\text{я}}$  — это часть единичной площадки, при попадании на которую реакция происходит. Размерность произведения ядро/м. Оно дает долю площади, занятой на толщине в 1 м сферами влияния ядер. Подобные рассуждения законны, если «затенение» одной сферой влияния другой — редкое явление. Это при обычных плотностях действительно имеет место.

Эффективное взаимодействие для каждого рода взаимодействий при одних и тех же парах частиц разное. Характер взаимодействия обычно обозначается индексом:  $\sigma_a$  — сечение захвата,  $\sigma_s$  — сечение рассеяния,  $\sigma_f$  — сечение деления.

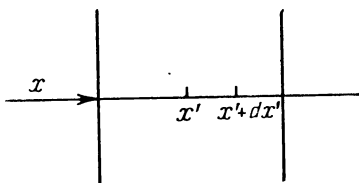


Рис. 2.4.

Пусть имеется некоторый слой вещества толщины  $x$  (рис. 2.4). Обозначим через  $P(x)$  вероятность прохождения частиц расстояния  $x$  без взаимодействия

$$P(x' + dx') = P(x')P(dx'). \quad (2.7)$$

$P(dx')$  можно выразить через эффективное сечение

$$P(dx') = 1 - \sigma N dx'. \quad (2.8)$$

Действительно,  $\sigma N dx'$  есть вероятность испытать на  $dx'$  столкновение; тогда  $1 - \sigma N dx'$  есть вероятность избежать столкновения. 0

Вместо функционального уравнения получаем обыкновенное дифференциальное уравнение

$$P'(x) dx = -\sigma N P(x) dx. \quad (2.9)$$

Имеем решение

$$P(x) = \exp\left(-\int_0^x N \sigma dx'\right). \quad (2.10)$$

Полученное выражение дает вероятность избежать взаимодействия при пробеге  $x$ . Определим вероятность взаимодействия на отрезке  $x, x + dx$ :

$$P(x + dx) - P(x) \approx -\sum \exp\left(-\int_0^x \Sigma dx'\right) dx = -p dx. \quad (2.11)$$

По смыслу  $p(x)$  есть плотность вероятности пробега без взаимодействия или свободного пробега  $x$ . Величина  $\Sigma = \sigma N$  называется макроскопическим сечением. Определим среднюю длину свободного пробега:

$$l = \int_0^{\infty} x p(x) dx = \frac{1}{\Sigma}. \quad (2.12)$$

Определим скорость взаимодействия. Эффективные сечения  $\sigma$ , как правило, не зависят от направления скорости  $v$  частицы относительно ядра. Поэтому в ряде случаев оказывается целесообразным рассматривать вместо  $n(t, r, v, \Omega)$  усредненную по направлениям плотность

$$n(t, r, v) = \frac{1}{4\pi} \int n(r, v, \Omega, t) d\Omega. \quad (2.13)$$

Рассмотрим частицу, имеющую скорость  $v$ . За время  $dt$  она пройдет путь  $v dt$ . Вероятность испытать взаимодействие на этом пути

$$v dt \Sigma = \frac{v}{l} dt \quad (2.14)$$

не зависит от направления  $\Omega$ . За единицу времени из  $n(t, r, v)$  частиц провзаимодействуют

$$n(r, v, t) v \Sigma. \quad (2.15)$$

Величину (2.15) называют плотностью поглощающих взаимодействий, а  $\Phi = n(r, v, t) v$  -плотностью потока нейтронов, в отличие от векторной величины потока нейтронов.

### § 3. Физические основы ядерных реакторов

**1. Реакция деления.** В реакторах используются в качестве делящихся веществ изотопы  ${}_{90}\text{Th}^{232}$ ,  ${}_{92}\text{U}^{233,235,238}$ ,  ${}_{94}\text{Pu}^{239}$  и некоторые другие. При поглощении ядром нейтрона получается составное ядро в возбужденном состоянии. Если энергия возбуждения достаточно велика, то происходит деление с образованием двух и более осколков. Величина энергии возбуждения определяется суммой энергии связи, вносимой нейтроном, и кинетической энергией нейтрона. Если даже кинетическая энергия равна нулю, то все равно выделяется энергия связи при образовании составного ядра.

В табл. 2.1 указаны энергии  $E$  в МэВ, необходимые для деления составного ядра, а также энергии связи нейтрона.

Т а б л и ц а 2.1

Ядро-мишень	Составное ядро	$E$ , в МэВ	Энергия связи	Примечание
${}_{90}\text{Th}^{232}$ ${}_{92}\text{U}^{238}$	$\text{Th}^{233}$ $\text{U}^{239}$	7,5 7	5,4 5,5	делится не при всех значениях энергии
${}_{92}\text{U}^{235}$ ${}_{92}\text{U}^{233}$ ${}_{94}\text{Pu}^{239}$	$\text{U}^{236}$ $\text{U}^{234}$ $\text{Pu}^{240}$	6,5 6,0 5,0	6,8 7,0 6,6	делится при всех значениях энергии

Из этой таблицы можно сделать заключение: изотопы с четным числом нуклонов после поглощения нейтрона и образования составного ядра могут разделиться, если энергия возбуждения будет больше 7,5 и 7 МэВ для первых двух строчек таблицы. Но нейтрон несет с собой только 5,4 и 5,6 МэВ соответственно. Это меньше минимума, нужного для деления. Следовательно, дефицит энергии можно ликвидировать за счет кинетической энергии нейтрона. Поэтому деление таких изотопов возможно нейтронами, обладающими энергией, большей некоторого порога.

Деление же изотопов с нечетным числом могут вызвать нейтроны любых энергий.

На рис. 2.5 дана графическая зависимость сечений деления различных изотопов от энергии нейтронов.

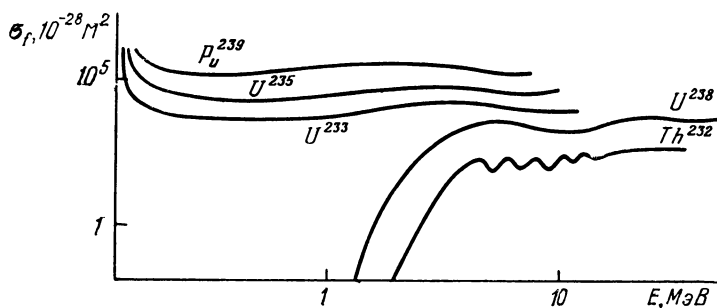


Рис. 2.5.

Обратим внимание на резкое возрастание  $\sigma_f$  для малых энергий у изотопов с нечетными номерами.

Энергия деления в основном сосредоточена в кинетической энергии осколков. Только 10% энергии выделяется в виде  $\gamma$ -квантов и нейтронов, а также другого проникающего излучения. В среднем выделяется 180 МэВ

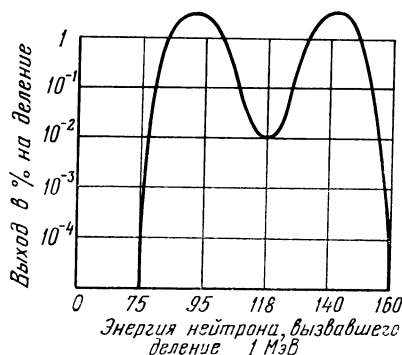


Рис. 2.6.

энергии на одно деление. Энергия осколков превращается в тепло, и, следовательно, выделяется внутри реактора, а энергия  $\gamma$ -квантов, нейтронов и других проникающих излучений может переноситься на большие расстояния. На рис. 2.6 приведено распределение осколков деления для  ${}_{92}U^{235}$ .

В результате деления ядра образуются вторичные нейтроны, которые в свою очередь могут вызвать деления, т. е. возможна цепная реакция. Изучим подробнее природу вторичных нейтронов. На рис. 2.7 воспроизведен спектр нейтронов.

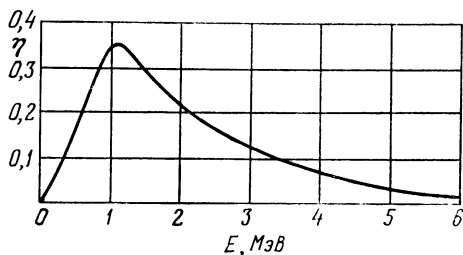


Рис. 2.7.

Величина  $\eta(E)$  имеет смысл плотности вероятности, т. е. доля нейтронов с энергиями в интервале от  $E_1$  до  $E_2$  может быть подсчитана в виде интеграла

$$P_{12} = \int_{E_1}^{E_2} \eta(E) dE. \quad (2.16)$$

Спектр позволяет найти  $E_{\text{ср}} = 2$  МэВ. Если разбить весь спектр на два участка:  $0 - 1,5$  МэВ и  $1,5 - 10$  МэВ, то оказывается, что доля нейтронов на обоих участках одинакова.

Большинство графиков приведено в этом разделе для  ${}_{92}\text{U}^{235}$ , для других делящихся веществ они такого же вида. Однако среднее число нейтронов на одно деление для разных делящихся веществ различно (табл. 2.2).

Таблица 2.2

Изотопы	$\text{U}^{233}$	$\text{U}^{235}$	$\text{U}^{238}$	$\text{Pu}^{239}$
Среднее число нейтронов на одно деление	2,507	2,442	2,881	2,800

При делении ядра не сразу рождаются все нейтроны. Помимо мгновенных существуют еще запаздывающие нейтроны. Доля их невелика — 0,76% от полного числа

вторичных нейтронов деления. Однако они оказывают существенное влияние на среднее время жизни поколения нейтронов. Понятие поколения введено для упрощения. Если рассмотреть на какой-то момент имевшие место акты деления ядер, то можно поставить вопрос о среднем времени жизни возникших в результате деления нейтронов. Это и будет среднее время жизни поколения. Обозначим через  $t_0$  среднее время жизни мгновенного нейтрона (обычно для нейтронов в реакторе  $t_0 \approx 5 \cdot 10^{-5}$  с). Тогда, если запаздывающие нейтроны появляются через  $t_{\text{ср}}^{\text{зап}}$  после мгновенных, а доля их  $\beta$ , полное время жизни поколения можно записать в виде

$$t_{\text{поколения}} = t_0 + \beta t_{\text{ср}}^{\text{зап}}, \quad t_0 \approx 5 \cdot 10^{-5} \text{ с.} \quad (2.17)$$

При этом собственным временем жизни запаздывающих нейтронов можно пренебречь, поскольку оно по порядку близко к  $t_0$ ,  $t_{\text{ср}}^{\text{зап}}$  составляет  $\approx 15$  с. Видно, что  $t_{\text{поколения}}$  определяется запаздывающими нейтронами, хотя их доля невелика. Энергия запаздывающих нейтронов меньше, чем мгновенных, и имеет порядок 0,3 МэВ.

**2. Реакции рассеяния и поглощения нейтронов.** В ядерной физике принято сокращенное написание реакций. Например, реакция поглощения нейтрона ядром с рождением  $\gamma$ -кванта запишется как

$$X(n, \gamma)Y,$$

где  $X$  — ядро-мишень,  $Y$  — вновь образовавшееся ядро.

Нейтроны могут испытывать на ядрах два типа рассеяния — упругое  $X(n, n)X$  и неупругое  $X(n, n')X'$  или  $X(n, n' + \gamma)X'$ . Первый тип реакций подобен абсолютно упругому столкновению бильiardных шаров, при этом энергия и количество движения сохраняются. Во втором случае возможно образование ядра в возбуж-

Т а б л и ц а 2.3

Ядро	Область энергий	$\sigma_s, 10^{-28} \text{ м}^2$	Ядро	Область энергий	$\sigma_s, 10^{-28} \text{ м}^2$
H <sup>1</sup>	1 эВ — 2 кэВ	20	U <sup>233</sup>	тепловые	40
H <sup>2</sup>	1 эВ — 3 кэВ	3,4	U <sup>235</sup>	—	17
Be <sup>9</sup>	1 эВ — 1 кэВ	6,0	Pu <sup>239</sup>	—	8
C <sup>12</sup>	2 эВ — 4 кэВ	4,6	Th <sup>232</sup>	—	13
O <sup>16</sup>	1 эВ — 200 кэВ	3,8	U <sup>238</sup>	—	13



денном состоянии, кинетическая энергия нейтрона соответственно изменяется. Возбужденное ядро может испустить  $\gamma$ -квант. Сечения рассеяния упругих столкновений для разных элементов даны в табл. 2.3.

Реакция вида  $X(n, \gamma)Y$  носит название «радиационный захват» (или резонансный захват). Реакции этого типа играют важную роль, так как ведут к бесполезной потере нейтронов, способных участвовать в цепной реакции. Вид эффективного сечения  $\sigma_c$  радиационного захвата от энергии нейтронов для  $U^{238}$  показан на рис. 2.8.

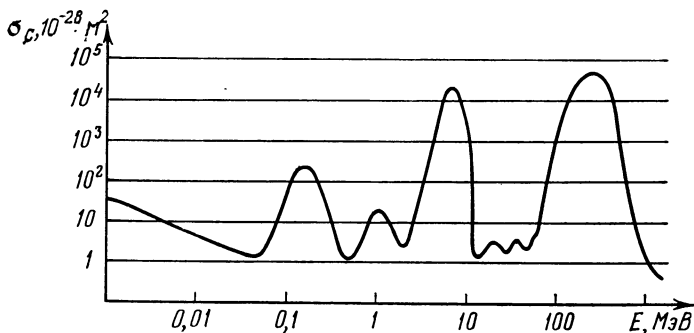


Рис. 2.8.

При определенных значениях энергии имеется множество резких возрастаний сечений — резонансы. При энергиях  $E < 0,1$  эВ наблюдается увеличение сечения поглощения по закону  $1/v$ , где  $v$  — скорость нейтрона. В целом сечение захвата возрастает при убывании энергии нейтрона.

**3. Цепная реакция.** Для практического использования реакций деления нужно добиться осуществления цепной реакции. Если в результате деления возникает больше одного нейтрона, то в принципе возможна цепная реакция. Однако в реакторе имеется много процессов, ведущих к уменьшению числа нейтронов. Это захват нейтронов без деления различными ядрами (как делящегося вещества, так и конструкционных материалов) и уход нейтронов за пределы реактора.

Вторая причина трудностей — это пороговое значение энергии деления наиболее распространенных изотопов  $U^{238}$  и  $Th^{232}$ . Действительно, средняя энергия нейтронов

деления 2 МэВ, а порог деления для  $U^{238}$  равен 1,4 МэВ. Между тем все нейтроны меньших энергий вообще не могут вызвать деления  $U^{238}$  и  $Th^{232}$ .

Есть еще одно осложнение: родившиеся в результате деления нейтроны не сразу поглощаются (с делением или без него), а могут испытывать многократное рассеяние.

Для организации цепной реакции необходимы другие изотопы:  $U^{235}$ ,  $U^{233}$  или  $Pu^{239}$ . В естественной руде встречаются только  $U^{235}$  в количестве 0,7% и  $U^{233}$  в совершенно ничтожных количествах.

Если учесть резкое увеличение эффективного сечения деления для этих изотопов при малых значениях энергии нейтронов, то становится ясно, что для реализации цепной реакции надо стремиться уменьшить энергию нейтронов. Правда, с уменьшением энергии нейтронов увеличивается сечение поглощения, однако эффект увеличения сечения деления перекрывает все другие эффекты. Поэтому при цепной реакции на естественном уране или слегка обогащенном  $U^{235}$  ( $Pu^{239}$  или  $U^{233}$ ) надо стремиться максимально «смягчить» спектр нейтронов, т. е. энергетическое распределение нейтронов должно в целом сдвигаться влево. Если энергии нейтронов будут близки к тепловым скоростям, то дальнейшее уменьшение энергии становится невозможным — распределение нейтронов по энергиям приобретает тепловой характер, они находятся (почти) в тепловом равновесии с окружающими ядрами.

Исторически удалось осуществить впервые цепную реакцию на тепловых нейтронах. Возможно осуществить цепную реакцию и на нейтронах более высоких энергий. Целесообразность этого связана со стремлением использовать  $U^{238}$  и получить новое делящееся вещество  $Pu^{239}$ . Однако в реакторах на быстрых нейтронах нужны более высокие концентрации изотопов типа  $U^{235}$ . К этим соображениям мы еще вернемся.

Можно приближенно определить интервалы энергии для тепловых, промежуточных и быстрых реакторов следующим образом: тепловые 0—0,2 эВ, промежуточные 0,2— $10^5$  эВ, быстрые  $10^5$ —10 МэВ.

Для быстрых и промежуточных реакторов сечение  $\sigma_{n\gamma}$  (радиационный захват) достаточно мало. Основную роль играют сечения  $\sigma_n$  (упругое рассеяние) и  $\sigma_n'$  (неупругое рассеяние).

Некоторые средние значения сечений для тепловых нейтронов даны в табл. 2.4.

Т а б л и ц а 2.4

Изотоп	$\sigma_a = \sigma_f + \sigma_{n\gamma},$ $10^{-28} \text{ м}^2$	$\sigma_f,$ $10^{-28} \text{ м}^2$	$\sigma_{n\gamma},$ $10^{-28} \text{ м}^2$	$\alpha = \sigma_{n\gamma}/\sigma_f$
U <sup>233</sup>	577	530	10	0,088
U <sup>235</sup>	678	580	17	0,17
Pu <sup>239</sup>	1012	741	8	0,36
Th <sup>232</sup>	?	0	13	—
U <sup>238</sup>	2,7	0	13	—
Уран естественный	7,68	4,1	13	—

Для тепловых реакторов правильно выбрать замедлитель — важная задача. Чем меньше времени будет находиться нейтрон в реакторе до замедления в тепловую область, тем вероятнее ему избежать резонансного захвата. Особенно существенно это для смеси естественного урана с одним из изотопов U<sup>235</sup> или Pu<sup>239</sup>, в силу того, что U<sup>238</sup> сильно поглощает нейтроны (радиационный захват) в резонансах.

Пусть до столкновения нейтрон имел скорость  $E'$ , после столкновения  $E$ , тогда  $\xi = \ln(E'/E)$  есть потеря энергии. Число столкновений, необходимое для замедления до тепловых энергий,

$$N = \frac{1}{\xi} \ln \frac{E_0}{E_T}, \quad (2.18)$$

где  $E_0$  — энергия, с которой нейтрон рождается (например, 2 МэВ — средняя энергия нейтронов деления). В табл. 2.5 дана характеристика замедлителей.

Т а б л и ц а 2.5

	$\xi$	$N$
H <sup>1</sup>	1	18
H <sup>2</sup>	0,725	25
He <sup>4</sup>	0,425	43
Be <sup>9</sup>	0,209	86
C <sup>12</sup>	0,158	114
U <sup>238</sup>	0,0084	2170

Т а б л и ц а 2.6

Замедлитель	$\xi \Sigma_s$	$\xi \Sigma_s / \Sigma_a$
Вода	135	61
Полиэтилен	161	61
Тяжелая вода	18,8	5700
Углерод	6,0	205
Гелий	0,009	45
Уран	0,33	0,009

Для замедлителей вводится также понятие замедляющей способности  $\xi \Sigma_s$  и коэффициента замедления (табл. 2.6).

Из приведенной таблицы 2.6 видно, что наилучшим замедлителем является тяжелая вода, но на практике чаще используется графит по экономическим соображениям.

**4. Критичность реактора.** Подсчитаем среднее число вторичных нейтронов, приходящихся на один захваченный нейтрон в «горючем» (коэффициент размножения). На  $\Phi \sigma_f N_a$  актов деления, дающих  $\nu \Phi \sigma_f N_a$  вторичных нейтронов, приходится  $\Phi \sigma_c N_a$  актов радиационного захвата. Число захваченных в горючем нейтронов  $(\Phi \sigma_f + \Phi \sigma_c) \cdot N_a$ . Число вторичных нейтронов в результате деления  $\nu \Phi \sigma_f N_a$ . Коэффициент размножения в горючем  $\eta$

$$\eta = \nu \sigma_f / (\sigma_f + \sigma_c) = \nu / (1 + \alpha),$$

где  $\alpha = \sigma_c / \sigma_f$ . Величина  $\nu$  слабо зависит от энергии нейтрона, вызвавшего деление, и от вида горючего. Для оценки качества горючего очень важна зависимость  $\eta$  (или  $\alpha$ ) от энергии (табл. 2.7).

Т а б л и ц а 2.7

	U <sup>235</sup>		Pu <sup>239</sup>		U <sup>238</sup>	
	тепловой	быстрый	тепловой	быстрый	тепловой	быстрый
$\nu$	2,47	2,51	2,91	2,97	2,51	2,55
$\sigma_f \cdot 10^{-28} \text{ м}^2$	582	1,59	746	1,83	527	2,37
$\sigma_c \cdot 10^{-28} \text{ м}^2$	112	0,32	280	0,32	54	0,20
$\alpha$	0,192	0,201	0,375	0,175	0,102	0,084
$\eta - 1$	1,07	1,09	1,72	1,53	1,28	1,35

Величина  $\eta - 1$  показывает прибыль нейтронов. Можно рассмотреть для простоты бесконечный реактор с нейтронами одинаковой энергии (моноэнергетический). Если считать, что реактор однородный и не содержит никаких материалов, помимо делящегося вещества, то для плотности потока можно получить уравнение

$$\frac{1}{v} \frac{d\Phi}{dt} = (\eta - 1) \Sigma_a \Phi. \quad (2.19)$$

Здесь  $v$  — скорость нейтронов,  $\Sigma_a = \Sigma_c + \Sigma_f$  — пол-

ное макроскопическое сечение поглощения нейтронов в горючем. Решение уравнения 2.19 дает:

$$\Phi(t) = \Phi(t_0) \exp [(\eta - 1)(t - t_0)/T_a]. \quad (2.20)$$

Здесь введено обозначение  $T_a = 1/(\nu \Sigma_a)$  для среднего времени жизни нейтрона по отношению к захвату в горючем.

Величину  $\eta$  можно обозначить через  $k_\infty$ . В случае бесконечной однородной среды в моноэнергетической модели  $\eta$  и  $k_\infty$  — просто обозначения одной и той же величины. В общем случае полиэнергетической модели под  $k_\infty$  понимается число вторичных нейтронов на захват одного первичного (безразлично какой энергии). Анализ формулы 2.20 показывает, что при  $k_\infty > 1$  развивается самоподдерживающаяся цепная реакция (надкритический реактор); при  $k_\infty < 1$  нейтронный поток убывает (подкритический реактор); при  $k_\infty = 1$  имеет место критическое состояние. Обычно при конструировании выбирают  $k$  близким к 1. Видно, что при  $T_a = 5 \cdot 10^{-5}$  с и  $k_\infty - 1 = 0,001$  за 50 мсек поток увеличивается в три раза, а за 1 с — в  $10^9$  раз. Естественно, что при таких параметрах нельзя было бы сделать работоспособный реактор (последний неизменно выходил бы из-под контроля). Положение спасают западывающие нейтроны, поскольку они эффективно увеличивают  $T_a$  до 0,1 с и период реактора (так называется время, за которое нейтронный поток увеличивается в  $e$  раз) становится в нашем примере равным 100 с, что уже вполне приемлемо для регулирования реактора.

**5. О конструкциях тепловых реакторов.** Существует большое разнообразие конструкций реакторов.

Тепловые реакторы в основном изготавливаются двух типов. Корпусные реакторы представляют собой стальные цилиндры диаметром до 4 м и длиной до 18 м с толщиной стенок до 60 мм. Внутри располагаются топливные элементы и поглощающие стержни. Под давлением до 20 МПа через корпус реактора прокачивается вода, которая отводит тепло и одновременно служит замедлителем. Эта вода отдает тепло воде вторичного контура, в парогенераторе которого производится пар, используемый в турбинах. В другом варианте кипящих реакторов пар образуется непосредственно в активной зоне. В этом случае используется давление около 8 МПа.

Более технологичны реакторы канального типа из отдельных топливных элементов, через которые проходят

трубки с циркулирующим теплоносителем, обычно водой. Канальные реакторы можно собирать из одинаковых элементов, получая мощности до 2 млн квт в каждом реакторе. (Вместе с тем по габаритам они больше корпусных реакторов).

Основой всех реакторов являются тепловыделяющие элементы (ТВЭЛы), образующие реакторную решетку.

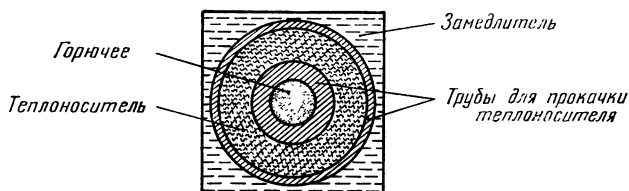


Рис. 2.9.

В качестве примера на рис. 2.9 приведен разрез квадратного ТВЭЛа. В реакторе количество ТВЭЛов может достигать нескольких сотен.

#### § 4. Математическая постановка задачи

Расчет реакторов представляет собой сложную комплексную задачу. Мы остановимся на вопросах физических расчетов реакторов. Теплофизические, конструкционные, расчеты защиты и прочие мы оставим в стороне.

Основными для физических расчетов ядерных реакторов являются расчеты критических размеров и пространственно-энергетического распределения нейтронного потока.

Математической основой расчетов служит кинетическое уравнение для плотности нейтронов  $n(\mathbf{r}, \Omega, v, t)$  в шестимерном пространстве  $\mathbf{r}$  и  $\mathbf{v} = \Omega v$ , где  $\Omega$  — единичный вектор вдоль скорости.

1. Вывод уравнения сводится к написанию баланса нейтронов в элементе  $dr d\Omega dE$  (здесь мы переходим в нормировке плотности от  $v$  к переменной  $E$  и попутно вводим новое обозначение для плотности  $N(\mathbf{r}, \Omega, E, t)$ ).

Рассмотрим поведение группы нейтронов в окрестности  $\mathbf{r}, \Omega, E$  с разбросом  $dr, d\Omega, dE$  в течение времени  $\Delta t$ .

Число нейтронов в данной группе в момент  $t$

$$N(\mathbf{r}, \Omega, E, t) dr d\Omega dE. \quad (2.21)$$

За время  $\Delta t$  в результате столкновений из группы исчезнет

$$\sigma(r, E) \nu N(r, \Omega, E, t) dr d\Omega dE \Delta t \quad (2.22)$$

нейтронов. В результате же столкновений в точке  $r$  в данной группе появится:

$$\iint \sigma(r, E') f(r; \Omega', E' \rightarrow \Omega, E) \nu' N(r, \Omega', E', t) \times \\ \times d\Omega' dE' dr d\Omega dE \Delta t. \quad (2.23)$$

В этом интеграле  $\sigma(r, E')$  — полное макроскопическое сечение для всех видов взаимодействий, а  $f$  — плотность вероятности перехода в результате взаимодействия любого вида от значений  $\Omega, E'$  к  $\Omega, E$ . Можно выразить  $\sigma f$  через элементарные процессы:

$$\sigma f = \sum_x \sigma_x f_x. \quad (2.24)$$

Если учесть еще внешние источники нейтронов, то можно написать баланс нейтронов в рассматриваемом элементе:

$$N(r + \Omega \nu \Delta t, \Omega, E, t + \Delta t) = \\ = N(r, \Omega, E, t) - N(r, \Omega, E, t) \sigma \nu \Delta t + \\ + \iint \sigma' f' \nu' N(r, \Omega', E', t) d\Omega' dE' \Delta t + Q(r, \Omega, E, t) \Delta t. \quad (2.25)$$

Здесь учтено, что в момент  $t + \Delta t$  координаты нейтронов будут  $r + \Omega \nu \Delta t$ . Левую часть выражения (2.25) можно преобразовать:

$$N(r + \Omega \nu \Delta t, \Omega, E, t + \Delta t) = \\ = N(r, \Omega, E, t) + \frac{\partial N}{\partial t} \Delta t + \nu \Omega \Delta t \frac{\partial N}{\partial r}. \quad (2.26)$$

После элементарных преобразований (2.25) получим

$$\frac{\partial N}{\partial t} + \nu \Omega \nabla N + \sigma \nu N = \iint \sigma' f' \nu' N' d\Omega' dE' + Q. \quad (2.27)$$

Это и есть кинетическое уравнение. Часто вместо плотности рассматривается поток нейтронов  $\Phi = \nu N$ . Для потока кинетическое уравнение запишется в следующем виде:

$$\frac{1}{\nu} \frac{\partial \Phi}{\partial t} + \Omega \nabla \Phi + \sigma \Phi = \iint \sigma' f' \Phi' d\Omega' dE' + Q. \quad (2.28)$$

Кинетическое уравнение носит интегро-дифференциальный характер и объединяет в себе трудности интегральных уравнений и уравнений в частных производных. В общем случае  $\Phi$  зависит от семи аргументов.

Для решения (2.28) нужно задать начальные и граничные условия. В качестве граничных условий возьмем условие свободной поверхности, т. е. потребуем, чтобы нейтроны не входили извне через граничную поверхность:

$$\Phi(r, \Omega, E, t)|_{r \in \Gamma} = 0, \quad n\Omega < 0, \quad (2.29)$$

где  $n$  — внешняя нормаль.

Характеристики среды  $\sigma$ ,  $f$ ,  $Q$  предполагаются кусочно-непрерывными функциями. Как правило, они даже кусочно-постоянны. Можно показать, что функция  $\Phi$  сохраняет непрерывность при переходе через границу разрыва как функция  $r$  и  $t$ . По остальным аргументам возможны разрывы при переходе через границы.

Введем понятие оператора переноса

$$LN = -v\Omega \nabla N - \sigma v N + \iint \sigma' f' v' N' d\Omega' dE';$$

тогда

$$\frac{\partial N}{\partial t} = LN + Q. \quad (2.30)$$

**2. Задача на собственные значения.** Большой практический интерес имеет задача на отыскание собственных значений оператора переноса

$$L\Phi_{\alpha_i} = \alpha_i \Phi_{\alpha_i}. \quad (2.31)$$

Результаты математического анализа этой задачи можно сформулировать во всех практически интересных случаях следующим образом: существует конечный набор собственных действительных значений  $\alpha_i$  и соответствующих им собственных положительных функций. Причем собственные значения удовлетворяют условию  $\alpha_i > -\sigma v$ . Наибольшему собственному значению  $\alpha_0$  соответствует всюду неотрицательная собственная функция. Если  $\alpha_0 > 0$  — решение возрастает (имеется в виду уравнение (2.30) с  $Q = 0$ ); если  $\alpha_0 < 0$  — решение убывает, если  $\alpha_0 = 0$  — решение не зависит от времени. Соответственно говорят о надкритической, подкритической и критической системах.

В ряде случаев представляет интерес стационарное решение уравнения (2.30) при условии, что  $L$  и  $Q$  от



времени не зависят. Такое решение существует при  $\alpha_0 < 0$ .

Теперь мы можем поставить задачу отыскания критических параметров системы и указать общий алгоритм решения этой задачи. Удобным оказалось введение вспомогательного параметра  $k$  — эффективного коэффициента размножения.

Выделим из интегрального члена оператора  $L$  слагаемое, соответствующее процессам деления,

$$\sigma'f = \sum_{x \neq f} \sigma_x f_x + \sigma_f \nu_f f_f (E' \rightarrow E). \quad (2.32)$$

В этом выражении опущены аргументы, под  $x$  в сумме понимается любой процесс, в котором возникают нейтроны (рассеяние упругое и неупругое, реакции  $n$ ,  $2n$  и т. д.), за исключением процессов деления. В слагаемом, описывающем процессы деления,  $\sigma_f$  — сечения деления,  $\nu_f$  — среднее число нейтронов на акт деления,  $f_f$  — спектр деления.

Вместо (2.31) запишем задачу нахождения собственных значений

$$\begin{aligned} & \nu \Omega \nabla N + \sigma \nu N = \\ & = \sum_{x \neq f} \iint \sigma'_x f'_x \nu' N d\Omega' dE' + \frac{1}{k} \iint \sigma_f \nu_f f_f \nu' N' d\Omega' dE'. \end{aligned} \quad (2.33)$$

Видно, что роль собственного значения здесь играет величина  $k$ . Введение множителя  $1/k$  соответствует замене  $\nu_f$  на  $\nu_f/k$ . Из физических соображений ясно, что, меняя  $\nu_f$ , т. е. среднее количество нейтронов на одно деление, мы всегда можем получить критическое состояние. С другой стороны, если второе слагаемое в (2.33) рассматривать условно как источник нейтронов, то (2.33) можно истолковать как уравнение для описания подкритической системы с источником. Как мы отмечали выше, такое уравнение всегда имеет единственное решение.

Теперь можно построить итерационный алгоритм для нахождения  $k$ , при котором (2.33) имеет ненулевое единственное решение (с точностью до постоянного множителя). Прежде чем начинать расчеты, выбирают некоторое начальное распределение нейтронов  $N^{(0)}$ . Затем рассчитывают интегральный член делений и находят первое приближение для  $N^{(1)}$ . Итерационная схема выглядит следующим

образом:

$$v\Omega\nabla N^{(n)} + \Sigma v N^{(n)} = \sum_{x' \neq f} \iint \sigma'_{xf} v' N'^{(n)} d\Omega' dE' + \\ + \frac{1}{k^{(n-1)}} \iint \sigma_f v_{ff} v' N'^{(n-1)} d\Omega' dE', \quad (2.34)$$

$$k^{(n)} = k^{(n-1)} \frac{\iint \sigma_f v_{ff} v' N'^{(n)} d\Omega' dE'}{\iint \sigma_f v_{ff} v' N'^{(n-1)} d\Omega' dE'}.$$

Условием окончания итераций обычно служат условия типа

$$\left| \frac{k^{(n)}}{k^{(n-1)}} - 1 \right| < \varepsilon, \quad (2.35)$$

либо

$$\max \left| \frac{N^{(n)}}{N^{(n-1)}} - 1 \right| < \varepsilon. \quad (2.36)$$

Такие итерации обычно называют внешними итерациями, или итерациями по источнику, в отличие от внутренних итераций, которые применяются для нахождения очередного приближения  $N^{(n)}$ .

При определении критических параметров решается не одна задача на отыскание собственного значения  $k$ , а несколько. Варьируя параметры, подбирают их так, чтобы критическое значение  $k = 1$ . Соответствующие значения параметров и будут критическими.

## § 5. Приближенные методы решения и организация вычислительного процесса

Решение кинетического уравнения в общем случае получить не удастся даже с помощью самых мощных вычислительных машин. Помимо семи независимых переменных, от которых зависит  $\Phi$ , известные трудности вносит сама конструкция реакторных систем. Если подробно описывать изменения ядерных характеристик внутри реактора, то по каждому из трех измерений придется задавать сотни и тысячи точек. Не меньшую трудность вносит и энергетическая зависимость ядерных сечений, имеющая весьма неплавный характер. В такой ситуации использование обычных методов дискретизации уравнений приводит к совершенно нереальной счетной работе.

Однако если бы такое решение и удалось получить численными методами, возникла бы не менее серьезная задача использования полученных результатов. Поэтому применяются упрощенные постановки задач, специальные численные методы и специальная организация вычислительного процесса.

В задачах на критические параметры положим  $\partial N / \partial t = 0$  и  $Q = 0$ .

**1. Кинетическое уравнение в различных геометриях.** Рассмотрим систему, обладающую симметрией. Это позволит уменьшить число геометрических и скоростных переменных. Для описания скоростной переменной  $\Omega$  употребляется сферическая система координат с углами  $\theta$  и  $\varphi$ . Разные случаи, которые будем разбирать далее, отличаются лишь ориентацией этой системы координат относительно системы координат для описания пространственных переменных. В плоском случае все функции зависят только от одной пространственной координаты, например  $x$ , и полярная ось системы для описания  $\Omega$  направлена по  $x$ . В силу симметрии зависимость от угла  $\varphi$  отсутствует. В этом случае можно записать оператор  $\Omega \nabla \Phi$

$$\mu \frac{\partial \Phi}{\partial x}, \quad (2.37)$$

и интеграл запишется так:

$$2\pi \int \sigma'(x, E') f(x; \mu', E' \rightarrow \mu, E) \Phi(x, \mu', E') d\mu' dE'. \quad (2.38)$$

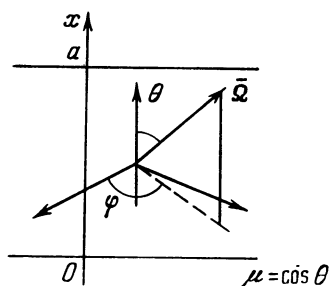


Рис. 2.10.

Существует еще одна геометрия — сферически-симметричная, в которой  $\Phi$  зависит только от одной пространственной переменной и одной скоростной  $\mu = \cos \theta$ . В этом случае полярная ось направлена вдоль радиуса вектора, проведенного в данную точку пространства. Поскольку зависимость от азимутального угла отсутствует, интегральный член полностью совпадает с (2.38).

Для оператора  $\Omega \nabla \Phi$  найдем

$$\mu \frac{\partial \Phi}{\partial r} + \frac{1 - \mu^2}{r} \frac{\partial \Phi}{\partial \mu}. \quad (2.39)$$

Появление производной  $\partial/\partial\mu$  связано с тем, что нейтрон с заданной скоростью  $\Omega$  в разных точках пространства имеет различные значения  $\mu$ .

Отметим еще общий случай прямоугольной системы координат для пространства и вложенной в нее сферической для  $\Omega$

$$\sqrt{1-\mu^2} \left( \cos \varphi \frac{\partial \Phi}{\partial x} + \sin \varphi \frac{\partial \Phi}{\partial y} \right) + \mu \frac{\partial \Phi}{\partial z}. \quad (2.40)$$

Важный класс задач представляют односкоростные задачи. В этих задачах предполагается неизменной абсолютная величина скорости нейтрона от его рождения до момента захвата:

$$|v| = \text{const}. \quad (2.41)$$

Односкоростное приближение играет большую роль в расчетах, поскольку к решению серии односкоростных задач сводится решение задачи, где энергетическая зависимость играет существенную роль. Односкоростное уравнение для  $\Phi(r, \Omega)$  можно записать в виде

$$\Omega \nabla \Phi + \sigma \Phi = \int \sigma(r) f(r; \Omega' \rightarrow \Omega) d\Omega' + Q(r, \Omega), \quad (2.42)$$

где граничные и начальные условия имеют прежний вид.

**2. Метод сферических гармоник.** Было предложено много методов численного решения уравнения (2.42). Все их можно разбить на две группы: 1) разложение функции  $\Phi(r, \Omega)$  по  $\Omega$  в ряд по некоторой системе функций; 2) замена непрерывной зависимости от  $\Omega$  на дискретную. При разложении в качестве полной системы чаще всего используются разложения по сферическим гармоникам:

$$Y_{lm}(\theta, \varphi) = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} P_l^{(m)}(\mu) \exp(im\varphi), \quad (2.43)$$

где  $\theta, \varphi$  — углы сферической системы координат, в которой рассматривается вектор  $\Omega$ ;  $P_l^m$  — присоединенные функции Лежандра.

Если зависимость от угла  $\varphi$  отсутствует, как это имеет место в плоском и сферически-симметричном случае, разложение можно вести по полиномам Лежандра. Для плоского случая можно записать

$$\mu \frac{\partial \Phi}{\partial x} + \sigma \Phi = \int \sigma(x) f(x, \mu_0) \Phi(x, \mu') d\Omega' + Q, \quad (2.44)$$

где  $\mu_0 = \Omega\Omega'$ . Таким образом, мы предполагаем, что  $f(x, \mu_0)$  зависит от угла между направлением скорости нейтрона до взаимодействия и после взаимодействия. Разложим  $f(x, \mu_0)$  также по полиномам Лежандра:

$$f(x, \mu_0) = \sum_{l=0}^{\infty} \frac{2l+1}{4\pi} f_l(x) P_l(\mu_0). \quad (2.45)$$

Для  $P_l(\mu_0)$  существует разложение

$$P_l(\mu_0) = P_l(\mu) P_l(\mu') + 2 \sum_{m=1}^l \frac{(l-m)!}{(l+m)!} P_l^m(\mu) P_l^m(\mu') \cos m(\varphi - \varphi'). \quad (2.46)$$

Если провести интегрирование по  $\varphi'$ , то в силу периодичности  $\cos m(\varphi - \varphi')$  член, отвечающий сумме в (2.46), даст нуль. Уравнение плоского случая тогда получим в виде

$$\mu \frac{\partial \Phi}{\partial x} + \sigma \Phi = 2\pi \sum_{l=0}^{\infty} \frac{2l+1}{4\pi} \sigma_{sl}(x) P_l(\mu) \int_{-1}^{+1} P_l(\mu') \Phi(x, \mu') d\mu'. \quad (2.47)$$

Функции  $\Phi$  и  $Q$  в виде разложений по полиномам Лежандра подставляем в (2.47). Используя ортогональность полиномов Лежандра, получим бесконечную цепочку уравнений для коэффициента разложения  $\Phi$ :

$$\begin{aligned} (n+1) \frac{d\Psi_{n+1}}{dx} + n \frac{d\Psi_{n-1}}{dx} + (2n+1) \sigma_n(x) \Psi_n(x) &= \\ &= (2n+1) Q_n, \quad (2.48) \\ \sigma_n(x) &\equiv \sigma(x) - \sigma_{sn}(x), \quad n=0, 1, 2, \dots \end{aligned}$$

Чтобы получить замкнутую систему уравнений, можно ограничиться первыми  $N+1$  уравнениями, положив  $d\Psi_{N+1}/dx = 0$ . Полученная система называется  $P_N$ -приближением.

Особое значение в реакторных расчетах имеет  $P_1$ -приближение:

$$\begin{aligned} \frac{d\Psi_1}{dx} + \sigma_0(x) \Psi_0(x) &= Q_0(x), \\ \frac{d\Psi_0}{dx} + 3\sigma_1(x) \Psi_1(x) &= 3Q_1(x). \end{aligned} \quad (2.49)$$

Заметим, что первые два коэффициента разложения  $\Psi_0$  и  $\Psi_1$  имеют физический смысл полного потока и векторного потока нейтронов:

$$\Psi_0(x) = 2\pi \int_{-1}^{+1} \Phi(x, \mu) d\mu, \quad (2.50)$$

$$\Psi_1(x) = 2\pi \int_{-1}^{+1} \Phi(x, \mu) \mu d\mu. \quad (2.51)$$

Если источник изотропный, то  $Q_1 = 0$  и (2.49) даст

$$\begin{aligned} \Psi_1(x) = & -\frac{1}{3\sigma_1(x)} \frac{d\Psi_0}{dx}, \quad D = \frac{1}{3\sigma_1(x)}, \\ & -\frac{d}{dx} \left[ D(x) \frac{d\Psi_0}{dx} \right] + \sigma_0(x) \Psi_0(x) = Q_0(x). \end{aligned} \quad (2.52)$$

Полученная система уравнений называется диффузионным приближением. Оно получается из  $P_1$ -приближения при  $Q_1 = 0$ .

В  $P_1$ -приближении граничное условие свободной поверхности не может быть удовлетворено точно, так как требование обращения в нуль входящего потока на границе, например, в плоском случае пластины, приводит к равенству

$$-\frac{\mu}{4\pi} [\Psi_0(0) + 3\mu\Psi_1(0)] = 0, \quad 0 \leq \mu \leq 1. \quad (2.53)$$

Отсюда следует  $\Psi_0 = 0$ ,  $\Psi_1(0) = 0$ . Аналогичное требование получаем на второй границе, при  $x = a$ . Таким образом, вместо двух условий, как это требуется для решения уравнения (2.52), получаем четыре.

Можно предложить несколько вариантов выхода из этого затруднения. Часто используют интегральные условия:

$$\begin{aligned} \int_0^1 \mu [\Psi_0(0) + 3\mu\Psi_1(0)] d\mu &= 0, \\ \int_{-1}^0 \mu [\Psi_0(a) + 3\mu\Psi_1(a)] d\mu &= 0. \end{aligned} \quad (2.54)$$

После интегрирования получаем:

$$\Psi_1(0) = -\frac{1}{2} \Psi_0(0), \quad \Psi_1(a) = \frac{1}{2} \Psi_0(a).$$

Соотношение между полным и векторным потоком можно записать в векторной форме

$$n\mathbf{J} = \frac{1}{2} \Psi_0. \quad (2.55)$$

Здесь  $\mathbf{J}$  — векторный поток нейтронов, но уже не для плоской задачи, а в общем случае. По аналогии с плоским случаем можно записать систему диффузионных уравнений в виде:

$$\begin{aligned} \mathbf{J} &= -D\nabla\Psi_0(r), \quad D = 1/(3\sigma_1), \\ -\nabla(D\nabla\Psi_0(r)) + \sigma_0(r)\Psi_0(r) &= Q_0(r). \end{aligned} \quad (2.56)$$

Точно такие же уравнения можно получить и из  $P_1$ -приближения, если проводить разложение по сферическим гармоникам, не предполагая какой-либо симметрии и положить  $Q_1 \equiv 0$ . Оставляя выкладки в стороне, приведем результат:

$$\nabla\mathbf{J} + \sigma_0(r)\Psi_0(r) = Q_0(r), \quad \nabla\Psi_0 + 3\sigma_1(r)\mathbf{J} = 3Q_1(r), \quad (2.56a)$$

где

$$\Psi_0(r) = \int \Phi(x, \Omega) d\Omega, \quad \mathbf{J}(r) = \int \Omega \Phi(x, \Omega) d\Omega.$$

$P_1$ -приближение и диффузионное приближение находят широкое применение в расчетах.

Однако для систем, малых по сравнению с длинами свободных пробегов, и вблизи поверхностей разрыва характеристик сред эти приближения оказываются непригодными.

**3. Метод дискретных ординат.** Рассмотрим методы второй группы — методы дискретных ординат — для плоского случая:

$$\begin{aligned} \mu_j \frac{\partial \Phi(x, \mu_j)}{\partial x} + \sigma(x) \Phi(x, \mu_j) = \\ = \sum_i \sigma(x) w_i(x) f_{ij} \Phi(x, \mu_i) + Q(x, \mu_j). \end{aligned} \quad (2.57)$$

Граничное условие свободной поверхности в этом случае может быть удовлетворено естественным образом:

$$\begin{aligned} \Phi(0, \mu_i) &= 0, \quad 0 \leq \mu_i \leq 1; \\ \Phi(a, \mu_i) &= 0, \quad -1 \leq \mu_i \leq 0 \end{aligned} \quad (2.58)$$

В методе дискретных ординат очень важно правильно выбрать точки  $\mu_i$  и квадратурную формулу для приближения интеграла по  $\mu$ .

Полученная система уравнений для  $\Psi_n$  (или  $\Phi_j = \Phi(x, \mu_j)$ ) должна быть проинтегрирована по пространственным переменным. В общем случае это система уравнений в частных производных с граничными условиями. Решение такой системы можно проводить разностными методами.

**4. Методы решения полученных уравнений.** Обратимся к общему диффузионному уравнению

$$-\nabla(D(r)\nabla\psi_0(r)) + \sigma_0(r)\psi_0(r) = Q_0(r). \quad (2.59)$$

Применение разностного метода для решения уравнения диффузии с граничными условиями сводится к разностной аппроксимации дифференциальных операторов из (2.59) и граничных условий и последующего решения полученной системы уравнений.

Для разностной аппроксимации надо выбрать разностную сетку. Выбор сетки должен учитывать геометрию задачи. В двумерном случае может применяться прямоугольная, гексагональная и другие сетки.

На рис. 2.11 изображена прямоугольная сетка, а на рис. 2.12 гексагональная; отмечены точки, в которых

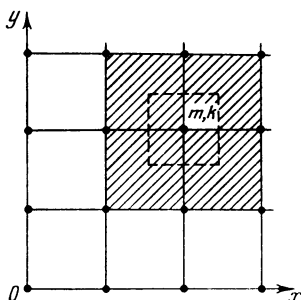


Рис. 2.11.

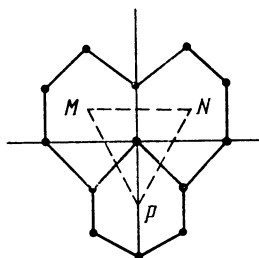


Рис. 2.12.

вычисляются искомые функции. Чтобы получить разностную аппроксимацию, проинтегрируем диффузионное уравнение по области, выделенной на рисунке пунктирной линией. Проведя обычные преобразования, можно получить систему разностных уравнений.

Для решения полученной разностной системы уравнений в двумерном и трехмерном случаях прямые методы решения оказываются малоприменимыми; для этой цели



используются итерационные методы. Изложение этих методов можно найти в [40], [41], [24], а применение их к реакторным задачам в [39] и [37].

Обратимся к методу дискретных ординат. Для плоского случая удобна итерационная схема, если применить следующую достаточно простую разностную аппроксимацию:

$$\mu_j \frac{\Phi^{(s)}(x_{k+1}, \mu_j) - \Phi^{(s)}(x_k, \mu_j)}{x_{k+1} - x_k} + \\ + \sigma(x_{k+1/2}) \frac{\Phi^{(s)}(x_{k+1}, \mu_j) + \Phi^{(s)}(x_k, \mu_j)}{2} = q^{(s-1)}(x_{k+1/2}, \mu_j). \quad (2.60)$$

В этом выражении через  $q^{(s-1)}$  обозначена вся правая часть (2.57), и вычисляется она по значениям  $\Phi^{(s-1)}$  с предыдущей итерации. Из (2.60) можно написать

$$\Phi_{k+1,j}^{(s)} = \frac{1 - \sigma_{k+1/2}(x_{k+1} - x_k)/2\mu_j}{1 + \sigma_{k+1/2}(x_{k+1} - x_k)/2\mu_j} \Phi_{k,j}^{(s)} + \\ + q^{(s-1)} \frac{x_{k+1} - x_k}{\mu_j (1 + \sigma(x_{k+1} - x_k)/2\mu_j)}. \quad (2.61)$$

Счет для  $\mu_j > 0$  ведем, начиная с  $k = 0$ , что соответствует нижней границе пластины. Условие свободной поверхности дает  $\Phi_{0,j} = 0$ .

Для  $\mu_j < 0$  надо в (2.61) выразить  $\Phi_{k,j}^{(s)}$  через  $\Phi_{k+1,j}^{(s)}$  и начинать счет с верхней границы пластины, полагая  $x_k = a$ . Условие свободной границы в этом случае дает  $\Phi_{k,j} = 0$ . Несколько сложнее выглядит схема итераций в случае криволинейных координат. Однако принцип остается тот же.

**5. Многогрупповое приближение.** До сих пор мы говорили только об односкоростных задачах. Однако большой интерес представляют задачи с энергетической зависимостью.

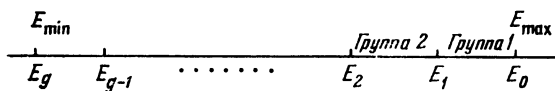


Рис. 2.13.

Решение таких задач проводится многогрупповыми методами. В этом методе вся энергетическая область делится на конечное число интервалов. Каждый энергетический интервал называется группой. Порядок нумерации групп поясняется на рис. 2.13.

Границы между энергетическими интервалами выбираются, как правило, из физических соображений. Например, при использовании двух групп выделяют тепловую и быструю группу. Если проводится точный расчет и число групп велико, то можно потребовать для граничных интервалов выполнения условия:  $E_g/H_{g-1} = \text{const}$ .

Вернемся к плоскому случаю. Кинетическое уравнение тогда примет вид

$$\begin{aligned} \mu \frac{\partial \Phi}{\partial x} + \sigma(x, E) \Phi(x, \mu, E) = \\ = \iint \sigma(x, E') f(x, \Omega', E' \rightarrow \Omega, E) \Phi(x, \mu', E') d\Omega' dE' + Q. \end{aligned} \quad (2.62)$$

Было бы естественно проинтегрировать (2.62) по интервалу группы, интеграл по  $E'$  заменить суммой интегралов по групповым интервалам. В качестве искомым функций можно было бы рассматривать усредненный по группе поток

$$\Phi_g(x, \mu) = \int_{E_g}^{E_{g-1}} \Phi(x, \mu, E) dE. \quad (2.63)$$

Однако такой путь оказывается непригодным из-за слишком большого количества групп, которые в этом случае потребовались бы. И связано это со сложным характером энергетической зависимости сечений и соответственно тонкой структурой нейтронного потока как функции от энергетической переменной. Между тем эта тонкая структура малоинтересна для глобального расчета реактора. Поэтому, задавая предварительно некоторый энергетический спектр, определяют усредненные сечения по группе по формулам типа:

$$\sigma_g(x, \mu) = \int_{E_g}^{E_{g-1}} \sigma(x, E) \Phi(x, \mu, E) dE \bigg/ \int_{E_g}^{E_{g-1}} \Phi(x, \mu, E) dE. \quad (2.64)$$

Приведенная формула неудобна, поскольку полное сечение, усредненное по группе, оказалось зависящим от угловой переменной. Поэтому целесообразно сначала перейти к уравнениям, например, метода сферических гармоник,

а затем проводить усреднение по группам:

$$(n+1) \frac{\partial \Psi_{n+1}}{\partial x} + n \frac{\partial \Psi_{n-1}}{\partial x} + (2n+1) \sigma(x, E) \Psi_n = \\ = (2n+1) \int \sigma_n(x, E' \rightarrow E) \Psi_n(x, E') dE' + (2n+1) Q_n(x, E), \quad (2.65)$$

$$\sigma_n(x, E' \rightarrow E) = 2\pi \int_{-1}^1 \sigma(x, E') f(x, E' \rightarrow E, \mu_0) P_n(\mu_0) d\mu_0.$$

Теперь уравнение для каждого значения  $n$  проинтегрируем по энергетическому интервалу группы:

$$(n+1) \frac{d\Psi_{n+1, g}}{dx} + n \frac{d\Psi_{n-1, g}}{dx} + (2n+1) \sigma_{n, g} \Psi_{n, g} = \\ = (2n+1) \sum_{g'=1}^G \sigma_{n, g' \rightarrow g} \Psi_{n, g'} + (2n+1) Q_{n, g}. \quad (2.66)$$

Здесь приняты обозначения:

$$\Psi_{n, g} = \int_{E_g}^{E_{g-1}} \Psi_n(x, E') dE', \quad \sigma_{n, g} = \\ = \int_{E_g}^{E_{g-1}} \sigma(x, E) \Psi_n(x, E) dE / \Psi_{n, g}, \quad (2.67)$$

$$\sigma_{n, g' \rightarrow g} = \int_{E_{g'}}^{E_{g'-1}} \Psi_n(x, E') \int_{E_g}^{E_{g-1}} \sigma_n(x, E' \rightarrow E) dE dE' / \Psi_{n, g'}.$$

Система уравнений (2.66) пока не содержит приближений, однако она содержит групповые константы, которые, вообще говоря, могут быть определены только после того, как будет получено решение задачи.

Для выхода из этого затруднения иногда используют решение для бесконечной среды. В другом случае считают, что для  $E > 1$  МэВ энергетическая зависимость соответствует спектру деления, а при меньших энергиях — пропорциональна  $1/E$ . Подобные приближения оправдывают себя при большом (больше 20) числе групп. Этот метод удобен тем, что значения констант могут быть вычислены заранее для каждого изотопа. Вообще, задаваясь тем или иным спектром нейтронов, можно

заранее заготовить системы констант, а по завершении многогруппового расчета проконтролировать совпадение полученного спектра с принятым при расчетах констант.

При малом числе групп целесообразна итерационная процедура корректировки констант.

Остановимся на организации расчетов в  $P_1$ -приближении в произвольной геометрии:

$$\begin{aligned} \nabla J_g^{(n)}(r) + \sigma_{0,g}(r) \Psi_g^{(n)}(r) &= \sum_{g'} \sigma_{s0, g' \rightarrow g} \Psi_{g'}^{(n)}(r) + \\ &+ \frac{1}{k^{(n-1)}} \sum_{g'} v \sigma_{f, g' \rightarrow g}(r) \Psi_{g1}^{(n-1)}(r), \quad (2.68) \\ \nabla \Psi_g^{(n)}(r) + 3\sigma_{1,g}(r) J_g^{(n)}(r) &= 3 \sum_{g'} \sigma_{s1, g' \rightarrow g}(r) J_{g'}^{(n)}(r). \end{aligned}$$

Здесь мы записали схему внешних итераций, выделив член источника, ввели эффективный коэффициент размножения и индекс итерации  $n$  и  $n-1$ , как это было сделано ранее.

Решение будем начинать с группы, отвечающей наибольшей энергии (индекс  $g=1$ ).

Тогда первая сумма справа содержит одно слагаемое  $g'=1$ , поскольку переходов из группы с меньшей энергией в группу с большей энергией за счет рассеяния мы не допускаем. Получаем односкоростную задачу для  $\Psi_1^{(n)}(r)$  с источником. При переходе к следующей группе в сумме у нас будет два слагаемых, одно из которых уже найдено. Таким образом, без итераций можно последовательно получить решения для всех групп.

**6. Гомогенизация.** Для организации вычислительного процесса в реакторных задачах существенна еще одна процедура. Как правило, реактор имеет сложную структуру. Тепловыделяющие элементы, число которых может составлять многие сотни и даже тысячи, сами по себе имеют сложное строение. Число типов таких элементов также велико. Для полного описания нейтронного потока в пределах одного твэла приходится задавать подробную сетку. Общее количество счетных точек по всему реактору получается грандиозным. Современные ЭВМ не могут справиться с такими расчетами. К счастью, надобность в них невелика — для всего реактора в целом детали поведения потока в пределах твэла несущественны. Вполне достаточно усредненной картины.

Усредненная картина может быть получена с помощью процедуры гомогенизации ячеек реактора. Сначала рас-

считывается подробно одна отдельная ячейка реактора. В качестве граничных условий выбираются условия периодичности: считается, что при переходе от одной ячейки к другой нейтронный поток ведет себя как периодическая функция. Это условие, конечно, не выполняется точно, поскольку реальный реактор имеет ограниченные размеры, и различные ячейки находятся в разных условиях.

Для расчета ячеек обычно применяют многогрупповые приближения с большим количеством групп (до 30 и более) и метод дискретных ординат или  $P_N$ -приближение. После завершения расчета ячеек проводят гомогенизацию, т. е. находят среднее по ячейке эффективное сечение

$$\sigma_x = \frac{\int_{\text{ячейка}} \sigma_x(r) \Phi(r) dV}{\int_{\text{ячейка}} \Phi(r) dV}. \quad (2.69)$$

В дальнейших расчетах нейтронного потока по всему реактору ячейки считают однородными по составу с характеристиками, определенными с помощью процедуры (2.69).

**7. Вычислительная схема.** Рассмотрим схему организации вычислительного процесса в задаче многогруппового расчета критических параметров реактора.

При организации вычислительного процесса больших задач важнейшим этапом является составление структурной иерархической схемы всего алгоритма в целом. Весь алгоритм разбивается на возможно меньшее число частей (блоков). При этом стремятся добиться минимальной связи между блоками. Тогда описание первого уровня иерархии состоит из описания функций каждого блока, их взаимодействия при реализации алгоритма и описания информационных связей блоков.

Рассмотрим наш пример. Приведем схему многогруппового расчета для первого уровня, считая, что количество блоков на каждом уровне не больше 10. Тогда блоки можно обозначить как Б 0.0.0, где первая цифра — это номер блока первого уровня, к которому относится данный блок, вторая — номер блока второго уровня, третья — номер блока третьего уровня и т. д.

Б 1.0.0. Обработка ядерных констант. Сбор данных по ядерным сечениям и оценка данных. Ядерные константы (микроскопические сечения) обычно в виде различных библиотек хранятся на внешних носителях ЭВМ. Такие библиотеки создаются и пополняются в результате физичес-

ких экспериментов и теоретических расчетов. Различные библиотеки имеют разные форматы данных и задача этого блока организовать извлечение нужных данных из разных источников, привести их в единый, принятый в расчете формат. Затем идет оценка данных, позволяющая выбрать достоверные данные и заполнить пробелы с помощью интерполяции.

ИД 1.0.0. Исходные данные: библиотеки констант на внешних носителях, например, магнитных лентах, задание на обработку.

ВД 1.0.0. Выходные данные: библиотека микроскопических сечений в заданном формате, удобная для проведения расчетов групповых сечений.

Б 2.0.0. Гомогенизация реактора. Этот блок дает исходные данные для расчета всего реактора в целом. Здесь проводятся расчеты ячеек в зависимости от их типа и положения в реакторной сборке. Расчеты ячеек проводятся в многогрупповом приближении. Для расчета углового и пространственного распределения используются обычно довольно точные методы, например,  $S_N$ - или  $P_N$ -методы. Число групп также выбирают достаточно большим. Для нахождения групповых констант по ячейке используют первоначально некоторое приближенное представление о спектре нейтронов. Это представление может корректироваться как после расчета ячейки, так и после полного расчета реактора.

Б 2.1.0. Расчет ячеек реактора.

Б 2.1.1. Подготовка к расчету ячейки некоторого типа.

Б 2.1.2. Определение групповых констант.

Б 2.1.3. Расчет нейтронных потоков по ячейке.

Б 2.1.4. Усреднение констант по ячейке.

Б 2.1.5. Проверка окончания счета ячеек. Если счет не окончен, то переход на Б 2.1.1.

Б 2.2.0. Определение групповых констант по реактору в целом, построение картограмм.

ИД 2.0.0. Исходные данные по составу и геометрии системы, наборы типов ячеек, данные для расчета ячеек.

ВД 2.0.0. Картограммы групповых констант.

Б 3.0.0. Расчет реактора на критические параметры.

Б 3.1.0.0. Начало цикла по внешним итерациям. Расчет источников деления.

Б 3.2.0.0. Проведение одной внешней итерации.

Б 3.2.1.0. Решение групповых уравнений, начиная с группы  $g = 1$ .

Б 3.2.1.1. Начало цикла по группам. Подготовка к расчету группы  $g$ .

Б 3.2.1.2. Выбор или расчет первого приближения для внутренних итераций. В нулевой итерации можно положить поток во всех точках, например, равным 1. В дальнейшем параметры потока можно брать из предыдущих итераций.

Б 3.2.1.3. Проведение внутренних итераций для группы  $g \neq 1$ .

Б 3.2.1.4. Расчет члена рассеяния. Проверка окончания расчета всех групп. Если цикл не закончен, то переход на Б 3.2.1.1.

Б 3.3.0.0. Проверка окончания внешних итераций. Если внешние итерации не закончены, то переход к Б 3.1.0.0.

Б 3.4.0.0. Проверка окончания расчета на критичность. Если  $k = 1$  с заданной точностью, то расчет окончен. Уход на блок Б 5.0.0.

Б 4.0.0.0. Выбор новых параметров для проведения критического расчета.

Б 4.1.0.0. Выбор новых данных по ячейкам, если при выборе параметров варьируются ячейки.

Б 4.2.0.0. Выбор новых данных по геометрии реактора в целом и по его составу.

Б 4.3.0.0. Корректировка спектра для расчета ячеек реактора с помощью спектра реактора в целом с учетом местоположения ячейки в сборке.

Б 4.4.0.0. Уход на Б 2.0.0.

Б 5.0.0. Расчеты некоторых величин по реактору с помощью подобранных критических параметров и полученного нейтронного потока.

В приведенной блок-схеме мы ограничились лишь двумя уровнями глубины и привели входные и выходные данные лишь для нескольких блоков в качестве примера. Мы также ограничились выписыванием основных блоков и не учли ряд расчетов, которые обычно производятся при нахождении критических параметров реактора. Отметим, что мы фактически в одной схеме объединили две: 1) схему счета, другими словами, схему последовательности выполнения блоков и 2) схему передачи информации от одного блока к другому. При графическом изображении каждая схема изображается отдельно с использованием графических символов, рекомендованных гостами ЕСПД [111, 112].

## ГЛАВА 3

### ПРИМЕР ЗАДАЧИ, РЕШАЕМОЙ В РЕАЛЬНОМ ВРЕМЕНИ

Характерной особенностью задач реального времени следует считать то, что они обслуживают какой-либо процесс в период его выполнения. Взаимодействие между вычислительной машиной и процессом целиком определяется процессом.

Естественно, что с точки зрения организации вычислительного процесса здесь должны быть свои особенности. Прежде всего они диктуются требованием, чтобы вычисления выполнялись по времени в «темпе» обслуживаемого процесса. Отсюда жесткое ограничение на время выполнения тех или иных расчетов. Большое значение приобретает система приоритетов на выполнение работ самого разнообразного характера. Помимо вычислительной схемы разрабатывается схема протекания процесса в реальном времени. Особую роль играет обмен информацией между задачей и обслуживаемым процессом. Наконец, очень важной особенностью таких задач является невозможность повторить расчет, поскольку для этого пришлось бы повторить и сам процесс.

Надо сказать, что круг задач реального времени достаточно широк. Это не только управление технологическими процессами, но и различные системы банковского типа, управление работой предприятий, информационно-справочные системы коллективного пользования.

При описании задач баллистического обеспечения, наземных средств управления полетом, организации работ мы не будем придерживаться конкретных вариантов. Иногда в интересах доступности изложения распространенные методы решения задач заменяются упрощенными схемами, дающими лишь представление о существе дела.



Ввиду классического характера излагаемых материалов в тексте не приводится ссылок. Читатель найдет интересные его материалы в [42—46]. Все численные примеры и константы взяты из [42] и [46].

## § 1. Баллистическое обеспечение космических полетов

**1. Задачи баллистического обеспечения.** Важной задачей, которую приходится решать при баллистическом обеспечении полетов космических аппаратов, является задача прогнозирования движения космического аппарата.

Ограничиваясь движением центра масс космического аппарата, задачу прогнозирования можно поставить следующим образом. Заданы начальное положение центра инерции аппарата и его скорость. Требуется по некоторой модели сил, действующих на космический аппарат, определить движение центра инерции аппарата. Если положение центра инерции космического аппарата задать вектором  $r$ , а скорость  $v = \dot{r}$ , то уравнение движения можно записать в виде

$$m\dot{v} = F_k + F_y,$$

где  $F_k$  — суммарный вектор внешних сил,  $F_y$  — вектор управляющих сил, сообщаемых космическому аппарату двигательными установками. В частном случае при пассивном полете  $F_y = 0$ . Далее мы будем рассматривать этот частный случай.

Приведенное нами уравнение можно записывать в различных координатах. Ниже мы приведем запись этого уравнения в декартовой системе координат, жестко связанной с Землей.

Трудности решения этого уравнения связаны со сложностью модели сил, действующих на космический аппарат. Только в идеальном случае, который на практике не осуществляется, она сводится к ньютоновым силам (3.1).

Другого рода трудность связана с тем, что используемая модель сил и начальные условия задаются с некоторой погрешностью. Последнее вынуждает ограничивать интервалы прогноза и проводить уточнение параметров движения космического аппарата по результатам навигационных измерений. Эта другая важная задача баллистического обеспечения будет рассмотрена нами ниже.

В случае упомянутых ньютоновых сил решение уравнения движения может быть проведено в замкнутом виде. Поскольку этот частный случай имеет большое познавательное значение, мы разберем его подробно.

**2. Движение под действием центральной силы  $F = -k/r^2$ .** Согласно закону всемирного тяготения два тела с массами  $M$  и  $m$  испытывают взаимное притяжение с силой

$$F = f Mm/r^2, \quad (3.1)$$

где  $f$  — гравитационная постоянная,  $r$  — расстояние между телами. В этой записи закона сами тела считаются точечными, т. е. малыми по сравнению с расстоянием между ними. В случае космических полетов в околопланетном гравитационном поле космические аппараты действительно можно считать точечными, что нельзя сказать о планетах. Однако если планету считать однородным шаром, то формула остается в силе — только  $r$  отсчитывается от центра планеты.

Для силы  $F$  можно ввести потенциал

$$U = -\frac{\alpha m}{r}, \quad (3.2)$$

где  $\alpha = fM$ . Для земли  $\alpha = 0,3986 \cdot 10^5$  м<sup>3</sup>/с. Знак минус в этой формуле показывает, что при перемещении тела  $m$  в бесконечно удаленную точку надо затратить энергию. При движении в поле с таким потенциалом сохраняется угловой момент

$$L = [r p]. \quad (3.3)$$

Здесь  $r$  — радиус-вектор, проведенный из центра сил в точку, где находится тело  $m$  (в дальнейшем мы будем считать массу тела единичной);  $p = m\dot{r}$  — импульс. Последнее соотношение показывает, что вектор  $r$  все время остается перпендикулярным одному и тому же вектору  $L$ , т. е. лежит все время в одной плоскости. Введем в этой плоскости полярные координаты с началом в центре сил. Тогда проекция момента на ось  $z$

$$L = r^2 \dot{\phi} = \text{const.} \quad (3.4)$$

Мы получили первый закон сохранения. Рассмотрим полную энергию системы

$$E = T_{\text{кин}} + U_{\text{пот}} = \text{const}, \quad (3.5)$$

$$E = \frac{1}{2} (\dot{r}^2 + r^2 \dot{\phi}^2) - \frac{\alpha}{r} = \text{const.}$$

Используя равенство (3.4), можно получить из (3.5) уравнение относительно  $r$ , которое решается методом разделения переменных:

$$E = \frac{1}{2} \left( \dot{r}^2 + \frac{L^2}{r^2} \right) - \frac{\alpha}{r}. \quad (3.6)$$

Если из (3.4) выразить  $dt = r^2 d\varphi / L$ , то получим производную  $\dot{r} = (L/r^2) dr/d\varphi$ , а вместо (3.6) получим

$$E = \frac{1}{2} \frac{1}{r^4} \left( \frac{dr}{d\varphi} \right)^2 + \frac{L^2}{2r^2} - \frac{\alpha}{r}. \quad (3.7)$$

Дифференциальное уравнение (3.7) проинтегрируем методом разделения переменных; получим

$$r = \frac{P}{1 + e \cos \varphi}. \quad (3.8)$$

В этом выражении  $P = L^2/\alpha$ ,  $e = \sqrt{1 + 2EL^2/\alpha^2}$  носят название параметра и эксцентриситета конического сечения, представленного уравнением (3.8). При  $e > 1$  мы имеем случай гиперболы, при  $e < 1$  — эллипса, при  $e = 0$  — окружности. Один из фокусов совпадает с центром начала координат.

Выражение (3.5) позволяет сделать некоторые заключения о характере движения. Для этого заметим, что всегда  $T_{\text{кин}} \geq 0$ , а в бесконечно удаленной точке  $U_{\text{пот}} = 0$ . Поэтому, если постоянная интегрирования  $E \geq 0$ , тело может «уйти» от центра на как угодно большое расстояние, в противном случае движение будет происходить в конечной области пространства. Записывая  $T_{\text{кин}}$  через  $v_2$ ,  $T_{\text{кин}} = \frac{1}{2} v_2^2$ , можем получить условие на  $v_2$ , при котором движение будет происходить в конечной области пространства:  $\frac{1}{2} v_2^2 - \alpha/r_2 \leq 0$  или  $v_2 \leq \sqrt{2\alpha/r_2}$ .

Обращаясь к (3.8), видим, что при  $E \geq 0$  эксцентриситет  $e \geq 1$ , т. е. движение происходит по гиперболе (или параболе).

Отсюда получаем первое важное следствие для космических полетов. Для того чтобы тело могло уйти за пределы земного притяжения, ему следует сообщить скорость равную или большую  $v_{\text{max}} = \sqrt{2\alpha/r_2}$ . Скорость  $v_{\text{max}}$  при  $r_2 = r_3$ , где  $r_3$  — средний радиус земли, называют второй космической скоростью.

Выясним, при каких наименьших энергетических затратах тело может превратиться в искусственный спутник Земли. Для этого обратимся к (3.8). Все множество замк-

нутых траекторий можно задавать с помощью двух параметров. Например,  $p$  и  $e$  (причем  $e < 1$ ), или  $L$  и  $E$  (причем  $E < 0$ ). Можно взять в качестве параметров минимальное расстояние от центра притяжения  $r_1$  и соответствующую ему скорость  $v_1$ . При этом для  $r = r_{\min}$   $\varphi = 0$  и скорость перпендикулярна радиусу, так что связь между  $L$  и  $E$  с  $r_1$  и  $v_1$ :  $L = v_1 r_1$ ,  $E = v_1^2/2 - \alpha/r_1$ . Требование минимума полной энергии дает нам одно соотношение между параметрами, так что произвольным остается один из параметров. Из (3.8) непосредственно усматривается соотношение между  $L$  и  $E$ :  $1 + 2EL^2/\alpha^2 = 0$ ,  $E = -\alpha^2/2L^2$ . В параметрах  $p$  и  $e$  оно выглядит как  $e = 0$ , т. е. минимум энергии реализуется для окружностей. Найдем такое соотношение для  $v_1$  и  $r_1$ :

$$\frac{v_1^2}{2} - \frac{\alpha}{r_1} = -\frac{\alpha^2}{2r_1^2 v_1^2}, \quad v_1 = \sqrt{\frac{\alpha}{r_1}}. \quad (3.9)$$

Выразим полную энергию через  $r_1$ , используя полученное соотношение:  $E = -\alpha/(2r_1)$ . Таким образом, из всех орбит с наименьшей полной энергией окажется круговая орбита с наименьшим возможным радиусом. Наименьший возможный радиус определяется радиусом Земли. Подставляя  $r_1 = r_3$ , получаем первую космическую скорость:  $v_1 = \sqrt{\alpha/r_3}$ . Следовательно, для околопланетных полетов скорость должна удовлетворять условию  $\sqrt{\alpha/r_3} < v < \sqrt{2\alpha/r_3}$ , разумеется, при  $\varphi = 0$ .

**3. Системы координат.** В исследовании космоса выделяют околопланетные полеты, межпланетные и полеты за пределы Солнечной системы.

В соответствии с такой классификацией вводятся системы координат для описания космических полетов (рис. 3.1). Для описания околопланетных полетов выбирается система координат с началом в центре

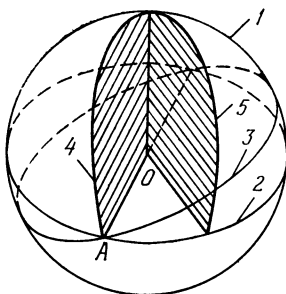


Рис. 3.1. 1 — небесная сфера, 2 — небесный экватор, 3 — эклиптика, 4 — часовой круг, проходящий через точку весеннего равноденствия, 5 — гринвичский меридиан, А — точка весеннего равноденствия, О — центр Земли.

масс планеты. Для Земли такие координаты называются геоцентрическими. Для выбора остальных элементов системы координат напомним некоторые понятия из астрономии. Плоскость, по которой движется Земля в своем годичном движении вокруг Солнца, называется плоскостью эклиптики. Линия пересечения плоскости эклиптики с экваториальной плоскостью проходит через восходящий узел эклиптики или точку весеннего равноденствия (см. рис. 3.1).

Плоскость  $XOY$  системы координат может совпадать с экваториальной плоскостью или с плоскостью эклиптики. В этих системах координат ось  $X$  направлена по линии пересечения плоскости экватора и плоскости эклиптики.

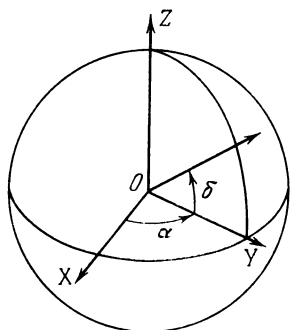


Рис. 3.2.

Ось  $Z$  выбирается перпендикулярно к выбранной плоскости, а ось  $Y$  дополняет систему координат до правой. Мы в дальнейшем будем пользоваться системой координат, в которой плоскость  $XOY$  совпадает с экваториальной плоскостью (рис. 3.2). Рассмотренные системы координат относятся к инерциальным координатам. Для межпланетных полетов используется система координат с началом в центре масс Солнца — гелиоцентрическая.

Для нас будут представлять интерес подвижные системы координат, связанные жестко с Землей. В этих системах за плоскость  $XOY$  также можно принять экваториальную плоскость. Однако плоскость  $ZOX$  совпадает с гринвичским меридианом. Такая система координат совершает вращение с угловой скоростью вращения земли  $\omega_3$ .

Особый интерес представляют орбитальные координаты. Плоскость  $\xi O \zeta$  в этих системах координат совпадает с плоскостью орбиты (рис. 3.3).

Ось  $\xi$  направлена на точку перигея \*) орбиты. Остальные две оси  $\eta, \zeta$  дополняют систему координат до правой \*\*). Положение орбитальной системы координат относительно

\*) Точка, наименее удаленная от центра притяжения.

\*\*) Ось  $\eta$  перпендикулярна к плоскости орбиты.

неподвижной геоцентрической задается углами  $\Omega$  — долготой восходящего узла,  $i$  — наклоном орбиты,  $\theta_k$  — истинная аномалия.

Из подвижных координат употребляются еще координаты, связанные с космическим аппаратом. Ось  $X$  может быть направлена к центру земли, другая ось — перпендикулярно к плоскости орбиты, а третья дополняет систему до правой. Возможно и другое направление осей, напри-

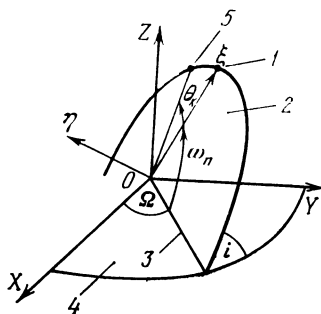


Рис. 3.3. 1 — точка перигея, 2 — плоскость орбиты, 3 — линия восходящего узла, 4 — плоскость экватора, 5 — космический аппарат,  $\xi$ ,  $\eta$  — оси орбитальных координат.

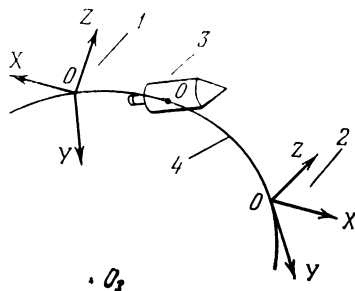


Рис. 3.4. 1 — система координат с началом в центре масс, 2 — другая система координат, 3 — космический аппарат, 4 — траектория движения, O — центр масс космического аппарата.

мер, одна из осей направлена по касательной к траектории полета, другая перпендикулярна к плоскости орбиты (рис. 3.4).

Выбор системы координат при описании движения космического аппарата определяется удобством решения соответствующей задачи. Поэтому употребляются самые разные системы координат, а переход от одной системы координат к другой задается соответствующими формулами преобразования, получить которые не составляет труда.

**4. Модель движения аппарата.** Перейдем к построению модели движения космического аппарата. Для этого нужно задать систему сил, действующих на аппарат, систему уравнений для описания движения и метод решения уравнений. При рассмотрении модели движения мы ограничимся движением центра масс. При полном расчете учитывается также движение аппарата относительно цент-

ра масс. Эти расчеты нужны для ориентации космического аппарата в заданном направлении.

Одна из основных сил, действующих на космический аппарат, — это гравитационная сила земного притяжения. Однако, в отличие от простейшего случая, разобранного нами, реальная Земля не является однородным шаром. Форма ее (геоид) достаточно сложна и не может быть описана даже трехосным эллипсоидом. Помимо этого Земля не обладает однородностью. Обычно потенциал Земли записывают в виде

$$U = -\alpha_1/r - U_v. \quad (3.10)$$

Для члена, описывающего «возмущение» ньютонова потенциала, строится разложение

$$U_v = \frac{\alpha_1}{r} \left\{ \sum_{n=2} \sum_{m=0}^n P_n^m(\theta) [c_{nm} \cos m\alpha + s_{nm} \sin m\alpha] \left(\frac{R}{r}\right)^n \right\}. \quad (3.11)$$

Здесь  $P_n^m(x) = \frac{(1-x^2)^{m/2}}{2^n n!} \frac{d^{n+m}}{dx^{n+m}} (x^2-1)^n$  — присоединенные полиномы Лежандра,  $\theta = \sin \delta$ , где  $\delta$  — географическая широта,  $\alpha$  — долгота (см. рис. 3.2),  $\alpha_1$  — гравитационный параметр,  $c_{nm}$  и  $s_{nm}$  — коэффициенты разложения, определяемые опытным путем,  $R$  — экваториальный радиус Земли.

В практических расчетах обычно ограничиваются несколькими членами разложения. Существуют другие формы представления возмущающего действия гравитационного поля Земли. Мы на них не останавливаемся, поскольку указанная форма чаще других применяется в расчетах.

На космический аппарат действуют еще силы сопротивления атмосферы. Хотя на высотах более 100 км плотность атмосферы мала, при больших скоростях учет атмосферы становится необходимым.

Для этих сил можно пользоваться значением ускорений, которые могут быть получены по формуле

$$a = c v^2. \quad (3.12)$$

В этом выражении  $c$  — баллистический коэффициент, зависящий от формы космического аппарата,  $\rho$  — плотность атмосферы,  $v$  — скорость движения аппарата.

Величины  $c$ ,  $p$  могут меняться в различных точках траектории в силу флуктуаций по плотности и составу атмосферы, а также в результате смены дня и ночи и других причин.

Помимо разобранных сил на движение космического аппарата оказывают влияние гравитационные поля других небесных тел, давление света, электродинамические силы от взаимодействия с электромагнитными полями земли и др. Рассматривая в основном околопланетные полеты, эти случаи мы оставим в стороне.

Из сказанного ясно, что силы, действующие на космический аппарат в пассивном полете, т. е. при выключенных двигателях, носят далеко не центральный характер. По этой причине движение уже не будет кеплеровым, тем более не будет замкнутым, так что характеризовать орбиты движения космического аппарата с помощью  $\Omega$ ,  $i$ ,  $\omega_n$ ,  $e$ ,  $p$  уже нельзя.

Используя связанную прямоугольную геоцентрическую систему координат, мы можем записать уравнения движения аппарата в следующем виде:

$$\begin{aligned} \dot{v}_x &= \left( \omega_3^2 - \frac{\alpha}{r^3} \right) x + \frac{\partial U_{\text{в}}}{\partial x} - e p v v_x + 2 \omega_3 v_y, \\ \dot{v}_y &= \left( \omega_3^2 - \frac{\alpha}{r^3} \right) y + \frac{\partial U_{\text{в}}}{\partial y} - e p v v_y - 2 \omega_3 v_x, \\ \dot{v}_z &= \left( \omega_3^2 - \frac{\alpha}{r^3} \right) z + \frac{\partial U_{\text{в}}}{\partial z} - e p v v_z. \end{aligned} \quad (3.13)$$

В уравнениях движения под  $\omega_3$  понимается угловая скорость вращения Земли, остальные обозначения соответствуют введенным ранее. Члены, содержащие  $\omega_3^2$ , связаны с центробежными ускорениями, а члены, содержащие  $2\omega_3$  — с силами Кориолиса. Эти члены появляются за счет использования вращающейся системы координат.

Уравнения (3.13) должны быть дополнены начальными условиями, которые называются векторами состояния. Вектор состояния — это значения  $x$ ,  $y$ ,  $z$ ,  $v_x$ ,  $v_y$ ,  $v_z$  в некоторый момент времени  $t$ .

Иногда в вектор состояния помимо этих семи чисел включают и другие параметры, связанные с моделями сил, например, значения баллистического коэффициента. Мы будем обозначать вектор состояния через

$$q(t) = \{x, y, z, \dot{x}, \dot{y}, \dot{z}, t\}. \quad (3.14)$$



Остановимся на понятии кеплеровых элементов орбиты. В произвольный момент времени  $t$  можно построить некоторую орбиту, соответствующую условиям невозмущенного движения. Такая орбита называется оскулирующей орбитой. В той точке, где в данный момент находится аппарат, оскулирующая орбита соприкасается с фактической орбитой, так что в некоторой окрестности они достаточно близки. Положение оскулирующей орбиты можно задавать тремя параметрами,  $\Omega$ ,  $i$ ,  $\omega_n$ , а саму орбиту — фокальными параметрами  $p$  и  $e$ . Тогда расположение аппарата на орбите можно задать с помощью параметра, характеризующего его положение относительно перигея, например, с помощью времени прохождения через перигей. Таким образом, мы получаем шесть элементов орбиты:  $\Omega$ ,  $i$ ,  $\omega_n$ ,  $p$ ,  $e$ ,  $\tau$ , которые эквивалентны заданию вектора состояния (3.14).

Используя кеплеровы элементы орбиты, можно записать уравнение движения.

В чем целесообразность описания движения в кеплеровых элементах орбиты. Если бы движение было невозмущенным, то очевидно, что элементы орбиты не менялись бы со временем. Изменение этих координат со временем отражает возмущение кеплерова движения. Если возмущение невелико, изменение элементов орбиты также невелико, что позволяет выбирать достаточно большой шаг интегрирования.

**5. Выполнение маневров.** Мы пока описывали физическую и математическую постановку задачи о движении центра масс в пассивном движении, т. е. при неработающих двигательных установках.

В режиме, когда работают двигательные установки, космический аппарат находится при выводе на орбиту, при выполнении орбитальных маневров, коррекции движения, выполнении маневров по стыковке на орбите, спуске аппарата на землю. Все эти управляемые движения осуществляются за счет работы двигательных установок. Их обычно делят на установки малой тяги и на установки большой тяги.

Мы не будем подробно формулировать все возникающие здесь задачи. Заметим лишь, что для математического моделирования движения космического аппарата на участке работы двигательных установок нужно в уравнении (3.13) включить ускорения, развиваемые двигательными установками,

Опишем алгоритмы управления маневрами, ограничиваясь простейшими предположениями. При выполнении маневра выбирается некоторый интервал времени маневрирования  $[t_{\text{вкл.}}^{(i)}, t_{\text{вык.}}^{(i)}]$ , где  $i$  — номер интервала включения и выключения двигательных установок. На каждом таком интервале космический аппарат получает изменение скорости движения. Рассмотрим пример маневра. Пусть нам требуется перевести космический аппарат с одной круговой орбиты малого радиуса на круговую орбиту большего радиуса. Весь маневр можно провести в два этапа. Сперва дается разгонный импульс, переводящий аппарат с круговой орбиты на эллиптическую, касающуюся в точке апогея большой круговой орбиты. Второй разгонный импульс переводит корабль с эллиптической орбиты на круговую орбиту большего радиуса.

Время сообщения разгонного импульса для случая большой тяги обычно невелико. Поэтому в приближенных расчетах иногда полагают, что разгонный импульс сообщается аппарату мгновенно. Для выполнения одного и того же маневра можно использовать несколько схем. Например, для перевода с одной орбиты на другую, не компланарную первой, может потребоваться несколько разгонных импульсов. Если орбиты пересекаются, то можно предложить варианты одного, двух и трех разгонных импульсов. Выбор схемы маневра обычно связан со стремлением провести его с минимальными затратами энергетических ресурсов.

Рассмотрим постановку задачи на оптимизацию управления маневрами. При расчетах управляемых движений обычно используется система координат, связанная с космическим аппаратом, «замороженная» на момент включения двигательных установок. Пусть ось  $Y$  направлена по местной вертикали, ось  $X$  — по касательной к траектории движения корабля. Сферическую систему координат вводим углами  $\psi$  и  $\theta$ . Угол  $\psi$  образован проекцией вектора тяги с осью  $X$  (отсчет идет против часовой стрелки). Угол  $\theta$  образован вектором тяги и плоскостью  $ZX$  (угол тангажа). Для дальнейшего введем определение вектора управления  $s^i = \{t_{\text{вкл.}}^{(i)}, \theta^i, \psi^i, |\Delta v^{(i)}|\}$ . Таким образом, вектор управления — это четыре числа.

Задача расчета маневра может быть поставлена следующим образом. Пусть задано два момента времени и соответствующие этим моментам векторы состояний  $q(t_n)$

и  $q(t_k)$ . Требуется выбрать совокупность векторов управления  $s^{(i)}$  так, чтобы в результате решения уравнений движения полученный вектор состояний  $\tilde{q}(t_k)$  совпал с заданным:  $\tilde{q} = q(t_k)$ . При этом должны быть удовлетворены естественные требования, состоящие в том, что управление проводится в пределах указанного временного интервала  $[t_n, t_k]$ . На векторы  $\Delta v^i$  могут быть наложены некоторые дополнительные требования. Например, чтобы они лежали в заданных плоскостях, т. е. чтобы выполнялись условия типа  $b_i \Delta v^i = 0$  ( $b_i$  — нормаль к заданной плоскости). Из предварительно разработанной схемы маневра могут быть известны временные интервалы, в которых должны располагаться моменты включения двигательных установок. Обычно всех этих условий недостаточно для определения векторов маневрирования. Оставшаяся свобода выбора используется для минимизации некоторых функционалов. Обычно минимизируют энергию, т. е. ищут минимум функционала

$$\sum_{i=1}^n |\Delta v^i|^2 c^{(i)} = \Phi.$$

Эта задача относится к разряду задач на поиски оптимальных управлений. Несложно видоизменить постановку задачи таким образом, чтобы вместо дискретных управлений использовать непрерывные управления. Однако даже в такой упрощенной постановке задача требует проведения больших численных расчетов. Для уменьшения объема расчетов прибегают к упрощениям модели движения. Иногда сперва проводят оптимизацию только по  $|\Delta v^i|$ , а выбрав  $|\Delta v^i|$ , проводят оптимизацию по  $t_{\text{вкл}}^{(i)}$ . Получив некоторые предварительные результаты, проводят расчеты с точной моделью движения, а оптимизацию проводят и по  $t_{\text{вкл}}^{(i)}$  и по  $|\Delta v^i|$  сразу.

В процессе полета в результате погрешности расчетов, ошибок измерений, неадекватной физической картины движения космического аппарата траектория полета может существенно отличаться от расчетной. В связи с этим в заранее запланированные промежутки времени проводится коррекция движения. По существу, коррекция представляет собой маневр, однако с заметно меньшими изменениями траектории движения, чем при маневрах.

Рассмотрим, например, предпосадочную коррекцию. Посадка аппарата должна произойти в заданном районе,

где сосредоточены определенные средства доставки и, кроме того, при посадке надо позаботиться о том, чтобы приземление спускаемого аппарата произошло в светлое время суток. Если в процессе полета возникли отклонения от номинального режима, то вполне возможен вариант, когда их нельзя компенсировать за счет маневра спуска аппарата. Кроме того, маневр спуска относится к весьма ответственным маневрам и естественно стремление проводить его в соответствии с заранее разработанным планом. Отсюда возникает необходимость предпосадочной коррекции.

Пусть, например, контролируется время прохождения восходящего узла орбиты (точка пересечения орбиты с экваториальной плоскостью). Именно этот параметр оказывает существенное влияние на условия посадки в заданном районе. Номинальное время обозначим через  $t_n$ , прогнозируемое по результатам траекторных измерений — через  $t_n$ . Если  $t_n - t_n \neq 0$ , то необходимо проводить коррекцию. Момент включения двигательных установок для проведения коррекции обычно выбирается заранее,  $t_{вкл.} < t_n$ . Для коррекции также должен быть рассмотрен вектор управления, как и для маневра. Используя вектор управления, модель движения космического аппарата и модель работы двигательных установок, путем решения соответствующих дифференциальных уравнений движения можно получить из вектора состояний  $q(t_{вкл.})$  вектор состояний  $q(t_{вык.})$ . Проводя интегрирование далее (при выключенных двигательных установках) уравнений движения от момента  $t_{вык.}$  до некоторого момента  $t'_n$ , близкого к  $t_n$ , с использованием в качестве начальных условий вектора состояний  $q(t_{вык.})$ , можно получить новое значение  $q(t'_n)$ . Выразим теперь восходящий узел орбиты через координаты:  $z_y = 0$ . Остается провести интерполяцию, используя значения координаты  $z$  в момент времени  $t_{вык.}$  и  $t'_n$ :

$$\frac{z(t'_n) - z(t_{вык.})}{t'_n - t_{вык.}} = \frac{-z(t_{вык.})}{t_y^{кор} - t_{вык.}}.$$

Если  $t_y^{кор}$  достаточно близко к  $t_n$ , то вектор управления найден. В противном случае берется другое приближение для вектора коррекций. Дальнейший подбор вектора коррекций можно проводить методом итераций, проводя интерполяцию для получения очередного приближен-

ного значения вектора коррекций. Время проведения расчета коррекции также подбирается заранее и, естественно, должно быть таким, чтобы время окончания расчета коррекции удовлетворяло требованию  $t_k^{\text{рас.}} < t_{\text{вкл.}}$ .

Аналогично проводится и расчет маневра спуска. Заранее задается время включения  $t_{\text{вкл.}}$  установок для разгонного импульса спуска. Задается вектор состояний после самых последних траекторных измерений на некоторый момент времени  $t_0$  (время последних измерений). Используя модель движения космического аппарата, можно рассчитать  $q(t_{\text{вкл.}})$ . Используя затем модель работы двигательных установок и модель движения, можно найти  $q(t_{\text{вык.}})$ . Далее находится вектор состояний для момента входа в плотные слои атмосферы (высота около 100 км). Движение в плотных слоях атмосферы происходит уже по другим законам. Здесь для торможения используются аэродинамические свойства спускаемого аппарата и непрерывное управление. Используя соответствующие законы, можно провести расчет вектора состояний на момент раскрытия парашюта. С помощью вектора состояний можно определить координаты приземления аппарата. Если они не соответствуют заданным, то использованный вектор управлений необходимо откорректировать и провести весь расчет спуска заново. Можно фиксировать разгонный импульс  $|\Delta v|$ , а подбирать  $t_{\text{вкл.}}$ . Если позволяют энергетические ресурсы, можно варьировать и величину разгонного импульса для улучшения других параметров спуска (нагрев, возникающие ускорения и т. д.).

**6. Обработка траекторных измерений.** Перейдем к методам определения параметров движения по траекторным измерениям. При движении космического аппарата ограничиваться заранее рассчитанными параметрами движения нельзя. Существует много причин, по которым расчетные номинальные траектории движения будут отличаться от фактических. Об этих причинах мы уже говорили раньше. Поэтому для определения фактических значений параметров движения производят специальные измерения, по которым и рассчитываются параметры движения.

Непосредственно измерять параметры движения не представляется возможным. В качестве величин, поддающихся непосредственному измерению, выступают дальности  $r$ , углы  $\theta$ , разности дальностей  $\Delta r$ , производные по времени от этих величин и т. д. Назовем эти параметры навигационными, или параметрами измерения. Введем

для них обозначение  $R$ . В качестве искоемых параметров движения можно рассмотреть вектор состояний, элементы орбиты:  $q = \{x, y, z, \dot{x}, \dot{y}, \dot{z}\}$ ,  $\Omega, i, \omega_n, p, l, \tau$ . Вводя номер измерения  $i$ , можем получить функциональные связи типа  $R_i(q_1, q_2, q_3, q_4, q_5, q_6, t_i)$ , или, сокращенно,  $R_i(q, t)$ , где под  $q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$  понимается некоторый набор параметров движения. Для измерений выбираются ориентиры, называемые навигационными точками, от которых собственно и производятся соответствующие отсчеты. Измерения могут проводиться с борта космического аппарата, с самой навигационной точки или с какой-либо другой точки. В качестве таких точек могут быть использованы земные ориентиры, искусственные спутники Земли, специальные станции слежения командно-измерительного комплекса. Могут использоваться также естественные небесные тела. Навигационные точки могут иметь фиксированные координаты, но могут перемещаться в пространстве, меняя свое положение по известным законам. К последним относятся искусственные спутники Земли и небесные тела. Обозначая параметры движения таких точек через  $Q = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6\}$ , можем включить их в функциональную зависимость для параметров измерения  $R_i(q, Q, t_i)$ .

Измерения данного навигационного параметра относительно различных навигационных точек или одной подвижной, в различные моменты времени, позволяет фиксировать положение космического аппарата при надлежащем выборе измеряемых параметров. После проведения соответствующих измерений, используя функциональные зависимости, нужно определить параметры движения. Здесь возможны два подхода. Можно так подобрать измеряемые параметры и количество проведенных измерений, что задача определения параметров движения будет иметь решение, и при том единственное. Этот подход редко используется, так как дает большие погрешности. Второй подход состоит в получении избыточного количества данных измерений. В таком случае, применяя статистические методы обработки результатов измерений, можно существенно повысить точность получаемых параметров движения и получить оценки погрешностей.

Последняя задача и носит название задачи нахождения параметров движения по траекторным измерениям. Ее можно разбить на два этапа. Сначала проводится статистический анализ выборки измерений, в результате

которого отбрасываются измерения, имеющие аномальные значения, возникшие за счет неисправностей приборов, грубых искажений при передаче и т. д. Поясним на примере процедуру отбраковки измерений. Пусть имеется приближенное значение вектора состояний  $q_0$ , которое в результате измерений должно уточняться. Это значение может быть известно по расчетам на прогнозирование. Тогда можно рассчитать величину  $\tilde{R}_i(q_0, t_i)$ , используя упомянутые выше функциональные связи. Найдя разность между измеренным значением и рассчитанным  $\Delta R_i = R_i - \tilde{R}_i$ , можно подсчитать средние значения  $\Delta R$  и дисперсию

$$\Delta R = \frac{1}{n} \sum_{i=1}^n \Delta R_i, \quad \sigma_R = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (\Delta R_i - \Delta R)^2}. \quad (3.15)$$

Здесь предполагается, что выборка содержит  $n$  измерений одного и того же типа. Тогда для отбраковки аномальных значений можно использовать неравенство:

$$|\Delta R_i - \Delta R| \leq v \sigma_R. \quad (3.16)$$

Здесь  $v$  — некоторая постоянная, определяемая опытным путем. Те значения, которые не удовлетворяют этому условию, отбрасываются. На этом отбраковка не кончается, так как таким образом удастся отбросить только слишком явные аномальные результаты. Дальнейшая отбраковка использует более тонкие методы (анализ дисперсий и средних по некоторому набору подвыборок и т. д.). При отбраковке руководствуются правилом: лучше потерять годное измерение, чем оставить негодное.

Оставшиеся измерения используют для решения уравнений

$$R_i = \tilde{R}_i(q, t), \quad \delta_i = R_i - \tilde{R}_i, \quad i = 1, 2, \dots, n. \quad (3.17)$$

Здесь, как и раньше,  $R_i$  — числовое значение измерения,  $\tilde{R}_i$  — вычисленное значение по уточненному значению вектора состояний. Поскольку число уравнений значительно больше шести  $n > 6$ , то система уравнений оказывается несовместной — число уравнений больше числа неизвестных. Для решения таких систем применяют известный прием: неизвестные находят из требования минимума выражения  $\sum_{i=1}^n \alpha_i^2 \delta_i^2 = \Phi$ . В этом выражении весовые

множители  $\alpha_i$  отражают свойства аппаратуры для измерения соответствующего  $R_i$ . Требование минимума приводит к нелинейной системе уравнений:

$$\frac{d\Phi}{dq_k} = 0, \quad k = 1, 2, \dots, 6. \quad (3.18)$$

Для решения этой системы применяется метод итераций в сочетании с разложением  $\tilde{R}_i$  в ряд по  $q$  в окрестности  $q_0$ :

$$\Delta q = q - q_0, \quad \tilde{R}_i \approx \tilde{R}_{i0} + \sum_{k=1}^6 \left( \frac{\partial \tilde{R}_i}{\partial q_k} \right)_0 \Delta q_k. \quad (3.19)$$

Для  $\delta_i$  найдем выражения:

$$\delta_i = R_i - \tilde{R}_{i0} - \sum_{k=1}^6 \left( \frac{\partial R_i}{\partial q_k} \right)_0 \Delta q_k, \quad i = 1, \dots, n. \quad (3.20)$$

Здесь индекс 0 показывает, что вычисление соответствующих функций проводится при  $q = q_0$ . Используя такие выражения для  $\delta_i$ , из (3.18) получим линейную систему уравнений для нахождения  $\Delta q$ . После решения этой системы находится первое приближение  $q_1 = q_0 + \Delta q_1$ . Используя  $q_1$  точно так же, как и  $q_0$ , можно получить вторую поправку  $\Delta q_2$ . Процесс окончится, когда поправки будут достаточно малыми. При этом каждое новое приближение используется также для отбраковки аномальных значений измерений. Приведем крупноблочную схему обработки измерений.

Б 1.00. Предварительная обработка сеансов измерений для получения выборок измерений навигационных параметров.

Б 2.00. Отбраковка результатов измерений.

Б 2.10. Отбраковка по критерию средних по выборке.

Б 2.20. Сравнение результатов измерений в различных сеансах измерений.

Б 2.30. Отбраковка с помощью анализа дисперсий.

Б 2.40. Отбраковка по сравнению средних в различных подвыборках.

Б 2.50. Фильтрация выборки с помощью чебышевских аппроксимаций.

Б 3.00. Нахождение вектора  $q_{j+1}$  по вектору  $q_j$ .

Б 3.10. Вычисление  $\tilde{R}_{ij}$  и  $\partial \tilde{R}_{ij} / \partial q_k$  при  $q = q_j$ .



Б 3.20. Вычисление коэффициентов уравнения  $d\Phi/dq_k = 0$ .

Б 3.30. Решение уравнения  $d\Phi/dq_k = 0$ , нахождение  $q_{j+1}$ .

Б 3.40. Проверка окончания счета итераций. Если  $|\Delta q_{kj}| < \epsilon_k$ , то переход на Б 4.00, в противном случае переход на Б 2.00.

Б 4.00. Передача полученных значений  $q$  в другие блоки.

Рассмотренные нами задачи входят в состав баллистического или навигационного обеспечения космических полетов. Задача навигации космических аппаратов — не только провести их по заданной траектории, но и обеспечить проведение различных экспериментов. Например, приходится заботиться об определенной ориентации космического аппарата относительно Солнца или других небесных тел, при выполнении исследований, связанных с этими телами. В отличие от навигации земных средств передвижения (самолеты, корабли, подводные лодки и т.д.), космическая навигация отличается рядом характерных особенностей. Это большие скорости, значительное время полета, большие расстояния, ограниченность энергетических ресурсов, невозможность прекратить полет в случае необходимости. Особое отличие составляет сочетание активной и пассивной фаз движения \*). И хотя время движения в активной фазе невелико, ошибки, допущенные в этой фазе, требуют для своего исправления расхода энергетических ресурсов корабля, а именно, энергетический ресурс определяет в конечном счете возможности проведения экспериментов в космосе и безопасность выполнения полета.

Отсюда возникают высокие требования к точности проведения всех навигационных расчетов. Естественно, что решение сложных задач баллистического обеспечения полета в процессе полета должно проводиться с помощью вычислительных средств комплексов управления полетом. Правда, космические аппараты располагают бортовыми вычислительными машинами, однако их возможности ограничены. Даже если бы мощности бортовой машины хватило для проведения необходимых расчетов, анализ результатов и принятие необходимых решений оказались

---

\*) Будем под активной фазой движения понимать движение космического аппарата при работающих двигательных установках (в отличие от пассивной).

бы не под силу малочисленному экипажу пилотируемого космического аппарата. Естественно, с развитием вычислительной техники и программного обеспечения на бортовые системы можно будет возлагать гораздо больше функций, чем это удастся сделать на сегодня. Последнее особенно важно для дальних межпланетных перелетов, где большие расстояния являются серьезным препятствием обслуживания полетов с Земли.

## **§ 2. Организация обслуживания полетов с помощью наземных средств**

Перед выполнением космического полета проводится серия работ по баллистическим исследованиям, на основании которых в дальнейшем разрабатывается программа полета, в которой с точностью до секунд рассчитываются навигационные события и параметры движения космических аппаратов. На этом этапе предварительной подготовки происходит выбор номинальных орбит, выбирается момент старта, решаются вопросы, связанные со схемами маневрирования, разрабатываются схемы проведения измерений и коррекции орбит.

Здесь появляются подробные описания всех событий полета, начиная от старта и кончая спуском. Все события обязательно рассчитываются, в результате чего появляются номинальные данные. Разрабатываются также ситуации, связанные с отклонениями от номинальных режимов. Для проведения расчетов выбирается модель движения аппарата, модель работы двигательной установки, модель процессов при спуске. Выбираются определенные алгоритмы расчета баллистических данных. Готовится соответствующее программное обеспечение для проведения расчетов.

Следующий этап — это алгоритмическая и программная подготовка к проведению фактического обеспечения управления полетом. Здесь готовятся алгоритмы и программы для проведения баллистического обеспечения. Вся эта работа базируется на программе конкретного космического полета. В результате завершения этого этапа программный комплекс управления полетом оказывается записанным на соответствующие носители информации электронных машин. Информационно-вычислительный комплекс центра управления полетом готов к проведению баллистического обеспечения.

Этап проведения баллистического обеспечения поясняется рисунками 3.5 и 3.6. Дальнейшие пояснения относятся к рис. 3.5. Центр управления полетом (3) оборудован вычислительными средствами, образующими информационно-вычислительный комплекс (8). Устройства управления (9) через вычислительный комплекс позволяют управлять как вычислительным процессом, так и космическим полетом путем выдачи команд и запросов в командный измерительный комплекс и в бортовые системы корабля. Средства отображения (10) служат для обеспечения информацией персонала центра управления. Информация о полете выдается на экранные пульта операторов полета, на большой экран, на различные графопостроители и через громкоговорители. Большой экран управляется непосредственно от вычислительной машины.

Для передачи информации в каналы связи служит аппаратура передачи данных (7). Она занимается преобразованием данных к нужному для передачи формату, а также чисто связной работой. Данные передаются на командный измерительный комплекс (2), состоящий из станций слежения и коммутационного центра (на рисунке он не изображен). Они располагаются так, чтобы по возможности охватить зонами радиовидимости траекторию полета космического аппарата. Для связи станций слежения с центром управления может использоваться искусственный спутник Земли (5). Помимо измерений, станции слежения передают на борт аппарата команды. Кроме главного центра управления обычно для целей повышения надежности могут использоваться дублирующие центры (4).

Используя рис. 3.6, опишем порядок проведения баллистического обеспечения и соответствующие информационные связи.

Перед запуском с полигона по линиям связи поступают данные о метеоусловиях и телеметрические данные о состоянии систем корабля. После обработки этих данных в центре управления принимается решение о проведении полета, и управление передается программе управления полетом. В период от старта до вывода корабля на рабочую орбиту с полигонного комплекса информация продолжает поступать в центр управления полетом. В дальнейшем согласно заранее разработанной программе на станциях слежения проводятся измерения и передаются вме-

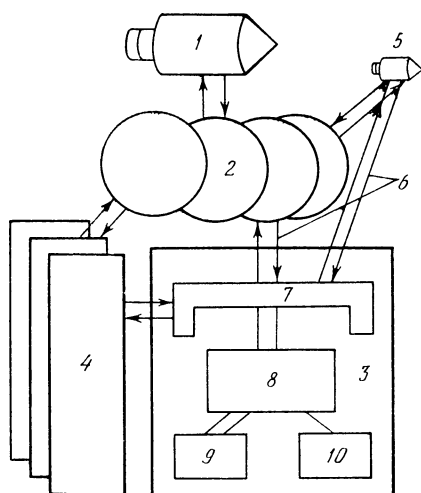


Рис. 3.5.

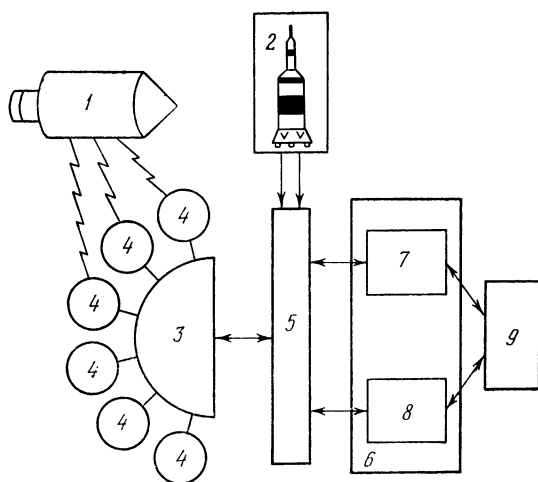


Рис. 3.6. 1 — космический аппарат, 2 — полигонный комплекс, 3 — коммутационный центр командно-измерительного комплекта, 4 — станции слежения, 5 — приемно-передающие системы, 6 — центр управления, 7 — вычислительная система центра управления, 8 — вычислительная система горячего резерва, 9 — зал управления полетом.

сте с принятыми телеметрическими измерениями в центр управления.

Центр управления, проводя расчеты по прогнозированию параметров движения, выдает на станции слежения информацию для наведения измерительных средств, а также командную и иную информацию на борт корабля. Согласно программе полета, также заложенной в памяти вычислительной системы, в надлежащие моменты проводятся расчеты коррекции по данным траекторных измерений и расчеты маневров.

На борту космического аппарата имеется вычислительная система. В ее задачи входит прием и выдача информации в радиолинии, управление автоматикой корабля, снабжение экипажа информацией в достаточно удобном виде. Она осуществляет выдачу команд устройствам ориентации и стабилизации корабля, а также устройствам управления двигательными установками. Бортовая машина осуществляет контроль систем корабля.

### **§ 3. Некоторые особенности организации программного обеспечения задачи**

Информационно-вычислительный комплекс центра управления может иметь в своем составе несколько идентичных вычислительных машин. Вот возможная схема. Две машины заняты непосредственно управлением полетом. Одна проводит все расчеты и передает информацию в линии. Другая, находящаяся в резерве, также проводит все расчеты по управлению, однако она только принимает информации, сама же ничего не выдает. При возникновении аварийной ситуации переход на резервную машину занимает совсем немного времени. Одна машина не занята полетом и обслуживает новые разработки. Две другие машины используются для целей моделирования полета. Моделирующая программная система осуществляет моделирование всех сигналов, поступающих в центр управления от станций слежения, полигонного комплекса, включающая тренажер космонавтов [46].

Характерной особенностью оборудования следует считать развитую систему каналов для передачи данных, что связано с большим объемом передаваемой информации. Комплекс снабжен большим количеством средств отображения информации.

На рис. 3.7 изображена связь оборудования и средств математического обеспечения.

Для задач баллистического обеспечения универсальная операционная система нуждается в доработке. В операционную систему реального времени целесообразно включить режимы ускоренной работы с различными видами памяти \*). Память имеет многоуровневый характер и должна обязательно включать такие внешние виды памяти, как диски, барабаны. Оперативная память служит для записи резидентных блоков операционной системы, прикладных программ, работающих в данный момент, и необходимой информации. На барабанах располагаются модули загрузки прикладных программ, информация к ним и нерезидентные блоки операционной системы. Операционная система и аппаратура должны обеспечивать быстрый сброс информации из оперативной памяти и столь же быструю последующую загрузку. Такое положение возникает при необходимости прервать выполнение некоторой прикладной программы в связи с появлением более приоритетной.

На магнитных лентах хранятся аварийные копии состояния центрального процессора и оперативной памяти на некоторый момент времени, а также информация,

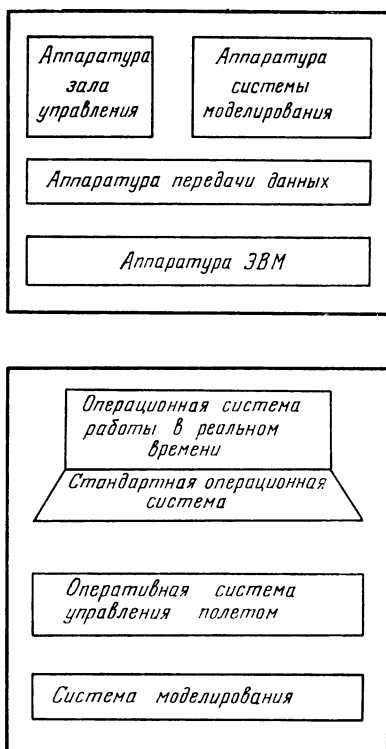


Рис. 3.7.

\*) Обсуждение особенностей развитых операционных систем реального времени не входит в наши планы, поэтому мы ограничиваемся лишь отдельными замечаниями, следуя в основном [46].

не используемая в данный момент в счете. При необходимости работать с этой информацией осуществляется перепись ее на магнитные диски или барабаны.

Для обслуживания многочисленных средств отображения трудно использовать центральный процессор, поскольку это может затормозить выполнение срочных

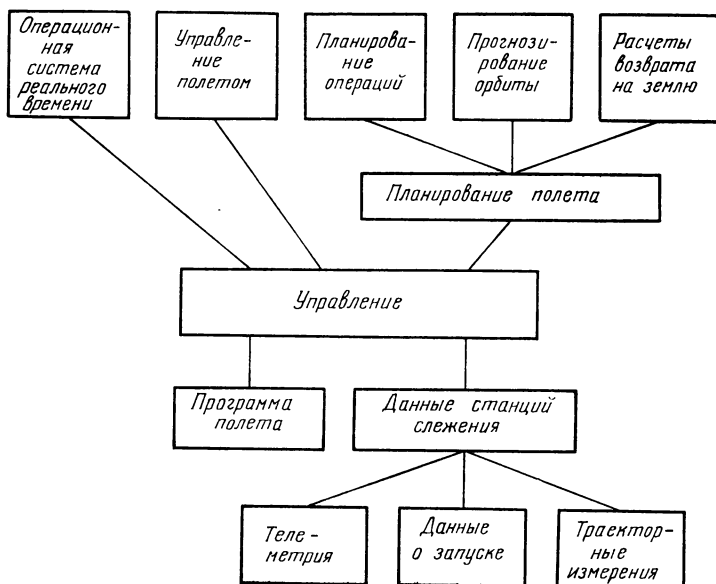


Рис. 3.8.

задач прикладного порядка. Для этой цели могут быть использованы специальные периферийные машины.

Из других особенностей программного обеспечения системы реального времени отметим необходимость предусмотреть динамическое распределение ресурсов для обеспечения пропуска срочных задач в нужном темпе. Особую роль играет приоритетная система на выполнение всех видов работ. Эта система должна быть доступна прикладным программам. Большую роль играет контроль правильности выполнения вычислений, а также способность локализовать последствия машинных сбоев. На случай выхода из строя тех или иных средств должна быть предусмотрена возможность в автоматическом режиме

провести перестройку. Вообще должна предоставляться возможность комплектовать из блоков наиболее рациональные варианты.

Очень важен блочный характер, позволяющий не только набирать из блоков нужный вариант программного обеспечения, но и добавлять в результате доработок новые блоки.

Операционная система должна быть удобной для работы с ней. Не только специальные системные программисты должны иметь доступ к системе, но и программисты прикладных задач. Последнее требование, естественно, ведет к многоуровневой системе. Наконец, для системы такого рода эффективность является важнейшим качеством.

На рис. 3.8 приведена укрупненная блок-схема программы управления полетом. Программа собирается из блоков после отработки всех частей программы полета. Центральное место в ней занимает управляющий блок.

Характерная особенность программного комплекса по управлению полетом — блочная структура. Такие системы насчитывают до миллиона команд и постоянно развиваются. Поэтому здесь особую роль приобретает документация, соглашения «о связях» между разработчиками, строгое соблюдение дисциплины разработок. Весь комплекс является как бы продолжением операционной системы реального времени, и его можно рассматривать как верхний уровень операционной системы.



## ГЛАВА 4

### ТЕХНОЛОГИЧЕСКИЕ АСПЕКТЫ ВЫЧИСЛИТЕЛЬНОГО ЭКСПЕРИМЕНТА

На протяжении длительного времени программирование в вычислительных работах считалось одним из наиболее трудоемких процессов. В связи с этим анализу технологии программирования придавалось большое значение [47, 48, 2, 49]. Однако уровень производительности труда при программировании все же продолжал оставаться недостаточным.

Это явилось причиной пересмотра всего процесса проведения вычислительных работ. Было введено понятие «вычислительного эксперимента» [4], распадающегося на ряд этапов, среди которых программирование занимало один этап. Разработке всех звеньев технологических процессов вычислительного эксперимента стало уделяться большое внимание. Как результат этой деятельности, появилось более глубокое понимание путей сокращения трудоемкости, что привело к созданию пакетов прикладных программ, основанных на модульном анализе всех компонент вычислительного эксперимента.

Ниже мы остановимся на разборе некоторых вопросов технологии отдельных этапов вычислительного эксперимента.

Разумеется, содержательное изложение этапов с постановкой и разрешением возникающих проблем на сегодня вряд ли возможно. Поэтому мы ограничимся постановкой акцентов на различных технологических проблемах, лишь в отдельных случаях предлагая возможные пути решения.

В этой главе начнем с рассмотрения процесса программирования. К некоторым вопросам будем возвращаться несколько раз, при каждой «итерации» рассматривая их в ином аспекте.

## § 1. Особенности программирования

1. Использование многоуровневой памяти. Программный комплекс и его эксплуатация — предпоследний этап работ, выполняемых при реализации вычислительного эксперимента [4]. По результатам работы с программой приходится снова возвращаться на другие этапы. Рассмотрения, проведенные нами в предыдущих главах, позволяют сделать некоторые выводы, касающиеся программирования. Во-первых, это жесткие сроки выполнения работ. Вычислительный эксперимент должен идти даже несколько впереди физического. Во-вторых, это жесткие требования экономии ресурсов. В самом деле, приведенные нами примеры показывают, что для решения задач такого рода приходится использовать предельные возможности вычислительной машины. Проведение же эксперимента требует расчета большого количества вариантов задач, вообще говоря, отличающихся по физической постановке \*).

В этих условиях приобретает большое значение технология разработок больших программных комплексов, под которой «мы понимаем весь комплекс методов и автоматических средств, предназначенных для проектирования, разработки, документирования, управления работой программистов и сопровождения программ» [52].

Перейдем к изложению некоторых отдельных вопросов программирования.

Построенная нами вычислительная блок-схема и схема данных задачи в приведенных примерах пока огирается только на алгоритм и не связана с машиной. В каждом вычислительном блоке описываются данные. Обычно фигурируют группы разнородных данных, которые мы будем называть блоками данных. Например, для счета внутренних итераций нужны следующие данные: коэффициенты разностной схемы, начальные приближения для нейтронного потока, источники, параметры для управления процессом итераций. Блоки данных можно разбить на три категории: исходные данные, выходные данные и внутреннее или локальные данные. Первые две категории имеют смысл вне блока и потому могут быть названы глобальными данными.

---

\*) Пренебрежение вопросами эффективности во многих случаях ставит под угрозу сам факт проведения вычислительного эксперимента.

Локальные данные бывает иногда полезно разбить на две группы: данные, которые нужно сохранять после выхода из блока и данные не сохраняемые, например, разностные коэффициенты при счете внутренних итераций можно отнести к сохраняемым. Это означает, они будут нам нужны каждый раз при работе блока и что они не меняются при многократном использовании блока. Эта ситуация несколько искусственна. Действительно, расчет таких данных можно вести вне этого блока и рассматривать их как исходные для него. Однако часто возникает такое положение, когда во внешних блоках приходится рассчитывать большое количество величин, нужных где-то во внутренних блоках. Может оказаться, что «расстояние по алгоритму» между вычисленной величиной и ее использованием будет очень большим. Нарушается один из принципов наглядной программы: «расстояние по алгоритму между точкой получения величины и ее использованием должно быть по возможности минимальным». Разумеется, этот принцип нельзя толковать как абсолютное и безусловное правило. Вместе с тем иногда целесообразно повторять вычисление некоторых величин и даже внутри цикла проводить вычисления, которые можно вынести за пределы цикла. Повторное вычисление одних и тех же величин часто практикуется и по соображениям экономии памяти, но это уже связано с конкретной вычислительной установкой.

На организацию вычислительного процесса аппаратура, программное обеспечение и весь стиль организации работ на машинах оказывают самое непосредственное влияние. Особую роль занимает организация использования памяти машины. Память может иметь весьма сложную структуру. Для упрощения выделим оперативную память, которую будем считать однородной по своим характеристикам, и внешнюю память. Последняя подразделяется на память с произвольным и последовательным доступом. Каждый вид памяти можно также подразделять на ряд уровней [15]. Память с произвольным доступом для нас будет представлена устройствами двух типов — барабанами и дисками, а память с последовательным доступом — только магнитными лентами. Барабаны, как правило, имеют меньшие времена доступа к данным, чем диски. Однако объем хранящейся на них информации значительно меньше, чем на дисках. Диски имеют еще одно преимущество — их можно использовать для постоянного

хранения информации. Однако время доступа на дисках большого объема может все-таки зависеть от порядка выборки, что иногда приходится учитывать.

Будем считать, что у нас имеется полистовая структура памяти, так что обмены могут идти только целыми порциями по 1024 \*) слова. Задача может получить в свое распоряжение некоторое число трактов ( $K$  слов) на барабанах. Отметим, что барабаны и диски позволяют проводить выборку данных в произвольном порядке, чего нельзя сказать о магнитной ленте: перематка ленты может потребовать значительного времени (несколько минут). Кроме того, расходы процессорного времени на организацию обмена с барабаном меньше, чем для ленты.

После этих предварительных замечаний перейдем к анализу некоторых вопросов использования различных уровней памяти. Размер оперативной памяти весьма невелик и не позволяет разместить все данные, необходимые для расчета. Пусть в приведенном нами расчете используется диффузионное приближение [37]. Для расчета выбирается 27 групп. Тогда в двумерном расчете нам потребуется с учетом разностных коэффициентов под данные одной группы отвести около 20 тысяч слов, а под все группы 540 тысяч. Таким образом, в оперативной памяти мы можем разместить данные только для проведения внутренних итераций. Закончив расчет некоторой группы, мы должны результаты внутренних итераций записать на внешний носитель для последующего использования, а данные для расчета следующей группы перевести в оперативную память. После завершения одной внешней итерации нам может потребоваться провести следующую итерацию. Для этого, как следует из приведенной схемы, нужно будет рассчитать источники с только что полученным нейтронным потоком. Однако если используется магнитная лента, придется прибегнуть к перематке, что займет несколько минут. Чтобы избежать удвоения перемоток ленты, целесообразно результаты расчета нейтронного потока для одной внешней итерации хранить на магнитном барабане вместе с соответствующими коэффициентами, нужными для расчета новых источников. Источники рассчитываются для каждой группы перед началом проведения внутренних итераций. Это займет около 140 трактов. Всего для рабо-

---

\*) О куске в 1024 слова говорят, что он занимает  $K$  слов.

ты потребуется две магнитные ленты, 140 трактов на барабане и 20 листов оперативной памяти. Напомним, лист, тракт и зона позволяют разместить 1024 слова соответственно в оперативной памяти, на барабане и ленте. На одной ленте записываются данные задачи, на другой — программы, данные для проведения расчетов на случай сбоя и некоторые данные, не связанные с проведением внешних итераций [37].

В случае двумерной задачи вместо лент удобнее пользоваться магнитным диском. Для трехмерной задачи использование дисков становится необходимым, так как требуемые объемы памяти возрастают в 15—20 раз [37]. Подобное положение характерно для многих больших задач. Основная трудность оказывается связанной с нехваткой памяти нужного уровня [11, 53].

В связи с этим важное значение приобретает правильная организация работы с данными. Ниже мы остановимся не отдельных вопросах этой работы и на вопросах использования различных уровней памяти.

Пусть имеется некоторая величина, заданная в виде массива. В процессе обработки массива мы можем последовательно использовать элементы массива, «продвигаясь по массиву» в определенном направлении. Некоторая «локальная непоследовательность» даже в пределах одного листа \*) несущественна при условии, что в целом имеется «продвижение» в определенном направлении. Такой случай мы будем называть последовательной выборкой. Если же выбираемые элементы массива расположены произвольно, то выборку будем называть произвольной.

В процессе обработки мы можем обращаться к массиву однократно или многократно. Например, при умножении матрицы на вектор сам вектор будет многократно используемым массивом, а матрица — массивом однократного использования.

Приведенные определения не отличаются строгостью и не претендуют на полноту охвата всех случаев работы с данными. Мы приводим их лишь для того, чтобы показать, как различные формы работы с данными влияют на формы организации работы с памятью.

Рассмотрим случай, когда данные не могут быть расположены все целиком в оперативной памяти. В этом

---

\*) Здесь мы имеем в виду машину с листовой структурой памяти, как например, БЭСМ-6. Размер листа — 1024 слов.

случае для них отводятся «резидентные области», куда и происходит считывание данных, хранящихся на различных уровнях внешней памяти. Для данных с последовательной выборкой можно применять магнитные ленты. Однако стратегия использования других уровней внешней памяти для многократно используемой величины и однократно используемой будет различной.

В первом случае следует разместить данные на барабанах. (До начала работы они могут находиться на магнитных лентах, а затем считываются на барабаны.) В процессе работы они последовательно считываются в оперативную память на резидентные места. Каким должен быть максимальный размер «резидента»? Чтобы ответить на этот вопрос, разберем схематичный пример.

Имеется достаточно большой одномерный массив данных. Обработку массива можно рассматривать как последовательность «актов» обработки, в каждом из которых завязаны компоненты  $a_{k_1}, a_{k_2}, a_{k_3}, \dots, a_{k_m}$ , где  $k_1, k_2, \dots, k_m$  — возрастающая последовательность индексов. Следующий акт обработки проводится с индексами  $k'_1, k'_2, \dots, k'_m$ , для которых справедливо соотношение  $k'_j = k_j + p$ ,  $j = 1, 2, \dots, m$ . В простейшем случае может быть  $p = 1$ , и обработка состоит в расчете формулы  $f(a_{k_1}, \dots, a_{k_m})$ . Тогда  $k_m - k_1 = k'_m - k'_1 = h$  — «шаг» по данным. В общем случае структура данных и характер обработки могут быть весьма разнообразной природы. Вместо одномерного массива может быть массив записей, каждая из которых занимает несколько машинных слов, например, таблица ядерных данных, а обработка может сводиться к поиску по таблицам, символьной обработке и т. д. Шаг может оказаться переменным, а набор данных в каждом акте обработки также может менять свою структуру, например, в зависимости от получаемых результатов. В таком случае можно ввести понятие максимального шага по данным, для которого мы сохраним обозначение  $h$ .

Пусть  $p = 1$  и  $k_{j+1} = k_j + 1$ ,  $j = 1, \dots, m$ , т. е. в актах обработки используются подряд идущие элементы. В таком случае каждый элемент будет использоваться в нескольких актах обработки. Если при этом окажется, что шаг по данным  $h$  больше размеров резидентной области, то при переходе от одного акта обработки к другому мы будем несколько раз считывать одни и те

же данные с барабана. Приведенные выше условия на  $p = 1$ ,  $k_{j+1} = k_j + 1$  не играют роли и выбраны лишь для простоты рассуждений. Сформулируем вывод: всякий раз, когда  $p < h$ , мы должны требовать, чтобы  $h$  было меньше размера резидентной области. При  $p > h$  на размеры области нет ограничений сверху, так как в этом случае каждый элемент данных один раз используется в обработке.

Таким образом, если каждый элемент данных используется многократно в нескольких актах обработки, или многократно в одном акте, на размеры «области» накладывается ограничение  $n_{\text{рез}} > h$  \*) в противном случае такого ограничения нет. Итак, «резидентная» область не может быть меньше листа, а верхняя граница определяется с учетом  $h$  и кратности использования элемента данных. Заметим, что кратность использования всего массива данных и кратность использования каждого элемента данных имеют разный смысл. Первая связана с количеством «проходов» вдоль всего массива данных, вторая связана с тем, насколько долго мы задерживаемся на каждом элементе. Первая определяет необходимость использования барабана как промежуточной памяти, вторая определяет размер резидентной области.

Чем выше кратность использования элементов данных и чем меньше кратность использования всего массива данных, тем меньше относительные потери на обмен. Это надо учитывать при организации структуры данных и вычислительного процесса.

Следующий пример показывает влияние структуры данных и организации вычислительного процесса на размеры резидентной области. Пусть у нас имеется три сеточные функции  $u_i$ ,  $v_i$ ,  $w_i$ , заданные в точках сетки  $x_i = it$ ,  $i = 1, 2, \dots, N$ . Определим структуру этих данных в виде последовательно расположенных трех массивов  $u$ ,  $v$ ,  $w$ , каждый из которых состоит из  $N$  компонент. Обработка этих данных сводится к расчетам по формуле  $f(u_i, v_i, w_i)$ . Ясно, что при выбранной структуре рам придется держать в оперативной памяти в качестве незидентной области три листа. В другом случае, если

---

\*) Неравенство  $n_{\text{рез}} > h$  не дает конкретного наименьшего значения  $n_{\text{рез}}$ . Оно зависит от программной реализации работы с массивом и других обстоятельств. В качестве одного из вариантов можно предложить  $n_{\text{рез}} = E(h/k) + h + 1$ , где  $E$  — целая часть числа,  $k$  — размер листа.

структура данных будет представлять собой массив записей, каждая из которых занимает три машинных слова под компоненты  $z_i = \{u_i, v_i, w_i\}$ , то достаточно всего одного листа. Однако если придется рассчитывать  $f(u_i)$ , то количество обменов возрастет в три раза по сравнению с предыдущей структурой.

Следующий пример показывает влияние организации вычислительного процесса. Пусть нам надо провести такую обработку одномерного массива:

$$a_i, \quad i = 1, 2, \dots, N,$$

$$b_j = \sum_{k=0}^4 a_{j+k}, \quad j = 1, 2, \dots, N-4,$$

$$r = \sum_{j=1}^{N-4} b_j^2.$$

Разумеется, можно вести последовательный счет  $b_j$  для  $j$ , возрастающих от 1 до  $N-4$ . Тогда шаг по массиву составит  $h = 5$ . Однако можно ввести пять рабочих величин  $c_1, c_2, c_3, c_4, c_5$ , которые в дальнейшем будут использоваться как буфер для расчета  $b_j$ . Начальный шаг состоит в засылке первых пяти компонент массива  $a_1, a_2, a_3, a_4, a_5$  соответственно в  $c_1, c_2, c_3, c_4, c_5$ . Далее рассчитывается  $b_j = c_1 + c_2 + c_3 + c_4 + c_5$ , затем идут пересылки  $c_1 := c_2, \dots, c_5 := a_j$  и в  $c_5$  засылается очередная компонента основного массива. В этом случае шаг по массиву получается равным единице. Подобный прием удобен при безындексной записи расчетных формул. Он также позволяет отделить организацию обмена от расчетной части. При такой организации обменом ведает специальная подпрограмма, которая при обращении проверяет, нужно ли проводить обмен, и если нужно, то производит обмен и записывает в рабочие ячейки соответствующие данные.

При другой организации работы, когда рабочие ячейки не используются, возникает вопрос стыковки двух последовательных сегментов данных, считываемых в резидентную область. Для этого приходится заводить дополнительный буфер на  $h-1$  машинное слово.

Для однократно используемой величины предыдущие рассуждения сохраняют силу в отношении резидентной области. Однако в этом случае предварительная перепись на магнитный барабан не нужна, поскольку считывание



в оперативную память происходит лишь один раз, а перепись на барабан все равно идет через оперативную память.

До сих пор мы говорили о данных с последовательной выборкой. Для данных с произвольной выборкой дать рекомендации практически невозможно без предварительного изучения характера выборки. Может оказаться, что, например, есть некоторые вероятностные оценки, описывающие характер работы с данными. В таком случае данные, к которым наиболее вероятно обращение, могут храниться во время работы в оперативной памяти или на барабане.

Использование в таких случаях магнитной ленты исключается. Однако трудности работы с такими величинами очень велики. Поэтому следует избегать алгоритмов, приводящих к данным с произвольной выборкой.

Организация внешней памяти для больших задач — наиболее сложная часть работы. Имеющиеся теоретические исследования вопросов оптимального использования памяти [55, 54] в практических случаях мало помогают, так как соответствующие оптимизационные задачи даже в простейших модельных вариантах решаются с трудом. Поэтому часто пользуются эвристическими приемами, которые, хотя и не дают оптимальных решений, в практическом плане вполне приемлемы. С другой стороны, в распоряжении разработчика остаются структура дан-

ных и организация вычислительного процесса, изменяя которые, можно решить некоторые вопросы работы с памятью.

Рассмотрим еще один пример. Пусть задача решается разностными методами. Используемая сетка по двум переменным  $y$  и  $x$  имеет вид, изображенный на рис. 4.1.

На самом деле в задаче имеются еще две переменные, так что каждой счетной точке на рис. 4.1 соответствует многотысячный массив данных. В точках на  $OY$  и  $OB$  заданы значения всех величин. Для нахождения неизвестных величин в точках, например в точке  $P$ , используются известные из предыдущих расчетов величины в точках

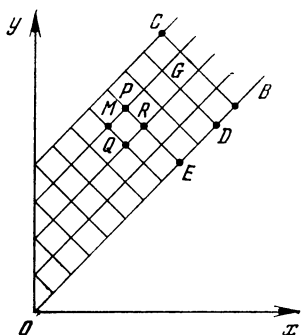


Рис. 4.1.

чения всех величин. Для нахождения неизвестных величин в точках, например в точке  $P$ , используются известные из предыдущих расчетов величины в точках

$M, Q, R$ . Учитывая это замечание, можно в области, определенной точкой  $C$ , найти решение, если вести счет вдоль линий, параллельных  $OB$ , или вдоль линий, параллельных  $CB$ . Однако при этом нам придется в памяти машины держать значения величин на двух соседних линиях, параллельных  $OB$ . Разумеется, мы отказались от намерения держать в памяти машины все найденные величины во всех счетных точках и поэтому при вычислении величин на очередной линии полученные новые значения заносим на места старых. Две линии держим на случай машинного сбоя, когда придется повторять расчет. При этом линии эти не обязаны быть соседними, а для большей надежности следует иметь данные на трех линиях, поскольку в момент переписи могут быть сбои, которые испортят оба массива данных. Однако ценой некоторого усложнения можно все-таки ограничиться двумя линиями. Об этом скажем несколько позже.

При такой организации вычислительного процесса мы встречаемся с рядом принципиальных трудностей. Во-первых, под данные на две «линии» может потребоваться до 12 магнитных лент и более. Счет задачи с таким количеством магнитофонов может сам по себе вызвать трудности. Во-вторых, наши данные имеют характер многократно используемых, и поэтому для работы с ними нужно пользоваться памятью на барабанах. Однако при такой организации вычислительного процесса, как мы описали, это невозможно. Выход из положения состоит в разбиении всей области на прямоугольники типа  $RGDE$ . В каждом таком прямоугольнике счет идет по линиям, параллельным  $OB$ . При этом все данные многократного использования можно целиком разместить на барабанах. Для этого надо только выбрать определенную «длину» сторон прямоугольника. Поскольку задача считается частями, то и количество занятых накопителей при каждом сеансе счета может быть небольшим. Такой несложный прием организации вычислительного процесса позволяет решить практически все трудности с памятью в этой задаче. В других случаях могут потребоваться значительно большие усилия.

Таким образом, работа по организации памяти в задаче начинается с изучения использования различных ее уровней. При этом главное внимание уделяется величинам, для размещения которых требуются большие объемы памяти. Затем приступают к составлению «таблицы рас-

пределения памяти», в которой перечисляются все величины, вычислительные блоки, объем отводимой памяти, как оперативной, так и внешней.

**2. Перемещение данных в процессе решения задачи.** С распределением памяти тесно связан другой этап работы над памятью. Назовем его условно использованием памяти в динамике. Здесь нужно тщательно отработать схему перемещения данных в процессе решения задачи, включив в вычислительную схему задачи блоки обмена.

Перемещение данных может проходить из памяти одного уровня в память другого уровня или в пределах одного уровня.

Первое связано с экономией памяти нужного уровня (в основном оперативной). Для этого данные разбиваются на группы, называемые сегментами. Под разные сегменты используются одни и те же участки оперативной памяти. Это возможно, так как на каждый момент не требуется, чтобы в оперативной памяти находился весь набор сегментов. За счет этого и достигается экономия памяти: на место сегментов, которые стали не нужными, в разные моменты времени загружаются другие сегменты.

Заметим, что сегменты могут включать данные разной природы или одной природы (см. приведенный выше пример сегментации массива).

Сегментация данных и разработка схемы загрузки сегментов тесно связаны с сегментацией программы и, по существу, представляют собой единый рабочий процесс.

Если сегментация данных производится без использования специальных средств программного обеспечения, то она сводится к организации обменов.

В некоторых случаях имеются системы (административные), организующие загрузку сегмента при появлении запроса на него. Если места в памяти не хватает, то исключается сегмент, к которому долго не было обращения.

Для облегчения сегментации данных используется математическая или виртуальная память. Это, по существу, фиктивная память с диапазоном адресов, превышающим размеры физической оперативной памяти. В этом случае пользователь может вообще не заниматься сегментацией данных, так как транслятор готовит программу в математических адресах. Вся математическая память разделена на сегменты. Когда возникает потребность в том или ином сегменте, он переписывается на свободное место памяти. Одновременно происходит настрой-

ка сегмента по месту. Если места в памяти не хватает, то неиспользуемые сегменты переписываются во внешнюю память. Если эти сегменты не подвергались изменениям за время своего пребывания в оперативной памяти, то их можно не переписывать во внешнюю память [59, 27].

Следует, однако, иметь в виду, что применение подобных средств может привести к большим накладным расходам, поэтому требуется тщательный анализ принципов работы этих средств на эффективность применительно к данной задаче.

Перемещение данных в процессе решения задачи может быть связано с необходимостью образовать новую структуру данных. Схематичный пример такого рода мы приводили, когда рассматривали три функции  $u$ ,  $v$ ,  $w$ . Переход к новой структуре там давал выигрыш в эффективности программы. Аналогичное положение может быть тогда, когда все данные помещаются в оперативной памяти. В этом случае новая структура может оказаться более целесообразной в смысле организации вычислительного процесса. Однако чаще всего переход к другой структуре данных связан с вопросами памяти, как это имело место в приведенном примере.

Следующая причина перемещения связана с интерфейсами между блоками. Как мы уже упоминали, каждый блок характеризуется исходными, выходными или результатными и внутренними данными. Организация интерфейса и состоит в передаче в распоряжение блока нужных данных. Это может потребовать выполнения загрузки одних данных и удаления из памяти других.

Естественно, что перемещение требует ресурсов машины — расходуется время центрального процессора, заняты каналы, по которым происходит пересылка данных. Поэтому перемещение должно быть оправдано.

Сразу оговоримся, что наличие определенных средств программного обеспечения может существенно видоизменить техническую сторону работы по использованию памяти в больших задачах. Например, вместо физических адресов на устройствах внешней памяти программист работает с именами наборов данных. Вопросы интерфейса подпрограмм, экономного использования памяти без перемещений с помощью COMMON-блоков, сегментации данных могут успешно решаться с помощью таких средств, как мониторная система Дубна, автокодная система Арап [29, 56, 57] и т. д.

Тем не менее основные вопросы работы с памятью должны быть решены программистом, правда само решение будет проводиться в других терминах. Когда достигнута ясность в использовании памяти, приступают к реализации, в которой роль средств программного обеспечения может оказаться весьма существенной. Однако, применяя универсальное программное обеспечение, не нужно забывать об эффективности.

**3. Сегментация программы.** Следующий этап — это схема загрузки в оперативную память вычислительных блоков. При размещении программ обычно требуется десятки тысяч машинных слов. Поэтому разумно держать в оперативной памяти только те блоки, которые в данный момент «работают».

Для такой организации работы нужно всю программу разбить на сегменты. Одновременно в оперативной памяти может располагаться несколько сегментов, занятых совместной обработкой данных. В процессе работы те или иные сегменты могут удаляться из оперативной памяти с одновременной загрузкой на их место других сегментов.

Таким образом, при разбиении программы на сегменты обычно учитывают имеющиеся ресурсы памяти и функциональные связи сегментов в те или иные моменты работы программы.

Сегменты могут состоять из одного или нескольких блоков. Возможны варианты, когда один блок делится на несколько сегментов. Последний случай мы, однако, не будем рассматривать.

Уделим внимание терминологии. До сих пор части, на которые разбивается программа, мы называли блоками. Еще для части программы у нас было наименование сегмент. Введем еще понятие модуля. Модуль и блок — в нашем понимании — близки, но все же несколько отличаются. Блок — это часть некоторой определенной программы. Тот же блок, но рассматриваемый самостоятельно, вне связи с программой, будем именовать модулем. В дальнейшем мы в основном будем пользоваться термином модуль для определения части программы, даже когда целесообразнее было бы употребить термин блок.

Рассмотрим в качестве примера сегментацию программ, написанных на автокоде [57, 58].

Модули, написанные на автокоде, после трансляции превращаются в модули загрузки. Эти модули на языке загрузки машины обычно оформляются в перемещаемом

виде, т. е. в некоторых относительных адресах. В каждом модуле могут быть объекты, например, переменные, которые должны иметь те же адреса, что и некоторые другие переменные в других модулях. Поскольку трансляция модулей происходит независимо, то в процессе трансляции им нельзя придать одинаковые адреса, возможно, относительные. Эти связи модулей, именуемые внешними связями, могут быть разрешены путем совместной обработки группы модулей. Для этой цели служит системная программа, именуемая редактором внешних связей [27, 58, 59]. После того как внешние связи обработаны, производится загрузка объединенного модуля на истинные адреса, в которых программа должна работать. Эти две операции — объединение и загрузка — выполняются одновременно [58].

При загрузке и объединении группы модулей указывается для первого модуля адрес загрузки, остальные «грузятся вплотную» друг к другу. Однако это делается в том случае, если не указан для остальных начальный адрес загрузки. Этим можно воспользоваться для образования сегментов, состоящих из одного или нескольких модулей, и загружаемых на одни и те же места памяти.

Эта работа должна начинаться с разработки схемы вариантов загрузки сегментов. При этом надо помнить, что смена одного сегмента в памяти на другой может привести к образованию пустых участков памяти, или к перекрытию с соседним сегментом, что приведет к ошибке. Кроме того, сегмент должен иметь внешние связи только с тем сегментом, с которым он будет находиться в памяти одновременно.

Мы описали примерный вариант технических средств для сегментации программ. Могут быть и другие средства, несколько отличающиеся по возможностям и реализации [60, 29], но принципиально они достаточно близки. Описанные средства реализуют статическую загрузку, т. е. загрузка осуществляется до начала счета задачи. Существует вариант динамической загрузки сегментов, когда загрузка происходит в процессе работы программ. Последняя имеет преимущество, поскольку не требует предварительного знания, будет ли и в каком порядке осуществляться загрузка сегментов. Однако при частой загрузке расходуется много времени на работу самого загрузчика [29].

Некоторого пояснения требует само понятие «работающий в данный момент сегмент». Ведь в каждый момент времени в работе принимает участие всего несколько машинных команд. Поэтому в принципе размер сегмента можно свести до нескольких команд. Это очень сэкономило бы оперативную память. Критерием выбора размеров сегмента можно считать минимум потерь на перезагрузку сегментов по сравнению с временем работы сегмента. Второй критерий — это наличная оперативная память.

Поясним на примере трудности, которые здесь могут возникнуть. Пусть некоторый цикл состоит из двух блоков, так что после окончания первого переходят ко второму, затем к первому и т. д. Если нельзя разместить оба блока в оперативной памяти, то остается организовать поочередную загрузку блоков. Обозначив затраты процессорного времени на работу блоков через  $t_6$ , а на перезагрузку блоков через  $t_n$ , при выполнении условия  $t_6 \gg t_n$  можно считать, что потери находятся в допустимых пределах. Если же  $t_6$  и  $t_n$  сравнимы, а еще хуже,  $t_6 < t_n$ , потери становятся большими. В большинстве случаев такое положение возникает в результате упущений при составлении структурной схемы алгоритма и схемы перезагрузки и может быть исправлено пересмотром этих схем. Разумеется, могут быть «злокачественные» случаи, корнящиеся в самой природе алгоритма, например, когда блок имеет большие объемы, а при работе выполняется лишь малая часть его операторов, причем заранее неизвестно, каких. В этих случаях приходится принимать меры к расширению оперативной памяти, занятой под программы, или пользоваться в качестве промежуточной памяти барабанами.

Итак, мы получаем дополнительное требование к структурной схеме алгоритма и к схеме загрузки блоков: с одной стороны, вычислительные блоки должны описываться программами, которые можно разместить в оперативной памяти, с другой стороны, при перезагрузке блоков должно выполняться требование  $t_6 \gg t_n$  для блоков, часто загружаемых в память.

Обычная схема организации работы такова: постоянно в оперативной памяти хранится управляющая программа, которая и организует поочередный вызов сегментов, хранящихся на ленте, барабане или дисках. Отдельные сегменты могут перед началом работы переписываться на барабан. Возможен другой вариант, когда каждый сегмент

по окончании работы сам вызывает следующий. Этот способ менее употребителен, и он хуже поддается автоматизации с помощью программного обеспечения.

#### **4. Пути сокращения «астрономического времени счета».**

Для дальнейшего нам важно обратить внимание на одну деталь мультипрограммного режима работы. Если некоторая задача, находящаяся в решении, запросила обмен с внешними устройствами, решение задачи прерывается на время обмена, а процессор передается в распоряжение другой задачи. После завершения обмена задача готова к счету. Однако это не означает, что она сразу же получит процессор. Порядок обслуживания задач, находящихся в решении, задается достаточно сложным алгоритмом операционной системы, который, несмотря на большое разнообразие вариантов, строится таким образом, что переключение процессора на другую задачу из-за обмена происходит лишь в том случае, когда обмен касается листов, используемых в данный момент для счета [27].

Такой принцип с точки зрения операционной системы уменьшает непроизводительные затраты процессорного времени, так как операция переключения на другую задачу дороже, чем прерывание на организацию обмена.

Для пользователя, решающего большую задачу, уменьшение числа переключений дает двойную выгоду: сокращаются непроизводительные расходы, которые нередко относятся за счет задачи, и уменьшается астрономическое время решения задачи, т. е. время от загрузки задачи до получения результата. Последнее обстоятельство крайне важно для задач с большим процессорным временем. Проведение расчетов с большим астрономическим временем трудоемко, а процессорное время может возрасти за счет сбоев и необходимых пересчетов.

При некоторых условиях можно так организовать обмены в задаче, что число переключений «по вине» задачи сильно уменьшится.

Сформулируем эти условия: среднее время на обработку «единицы» данных должно быть больше среднего времени на обмен.

Для некоторых задач такое условие не выполняется. Поэтому часто для упрощения организации работы с памятью в задачах из главы 2 разностные коэффициенты пересчитывают при проведении итераций, а для трехмерной задачи — это, видимо, единственный вариант при объеме оперативной памяти в 32 К слов.



Важность правильной организации вычислительного процесса можно проиллюстрировать такими цифрами. Пусть процессорное время расчета составит 60 часов. Если астрономическое время будет в шесть раз больше, т. е. 360 часов, то расчет не удастся завершить и за месяц. Здесь на первый план выступают особенности организации работы на машинах и характерные трудности работы с нашей задачей. Опишем ее.

Задача занимает почти всю доступную оперативную память, использует не менее половины барабанной памяти, имеет в своем распоряжении «личный диск», ленты. По условиям машины типа БЭСМ-6 такая задача трудна в эксплуатации: она долго не «входит в счет» из-за большой оперативной и внешней памяти на барабанах, для нее надо освобождать дисковод, устанавливать две-три ленты. Эту задачу нельзя отнести к числу «фоновых» \*), так как она имеет много обменов, и для того чтобы потери не были слишком велики, вместе с ней должны решаться другие задачи.

Таким образом, для нашей задачи надо подбирать специальный режим работы. Она не может решаться в дневное время, когда загрузка машины значительна за счет мелких задач и работ в режиме коллективного пользования. В ночное время она может решаться только при наличии «фоновых» задач. Учитывая все эти трудности, вряд ли можно рассчитывать на использование шести часов астрономического времени в сутки. Так что для решения такой задачи потребовалось бы два месяца. Если же учесть сбой и другие непредвиденные, но неизбежные потери, то и три месяца — не очень реальный срок.

Этих рассуждений, видимо, достаточно, чтобы убедить читателя в важности «хорошей» организации вычислительного процесса. Та же задача, но с астрономическим временем 80—90 часов и с меньшим объемом внешней памяти на магнитных барабанах, может быть просчитана за 10—12 дней.

В приведенном нами расчете явно нецелесообразно экономить на вычислении разностных коэффициентов. Повторное вычисление коэффициентов сократит количество обменов, а увеличение процессорного времени

---

\*) Так называют задачи с небольшим объемом обменов с внешними устройствами. При таком определении понятие фоновой задачи достаточно условно.

с лихвой окупится сокращением различного рода потерь и упрощением эксплуатации программы \*).

**5. Организация записей на случай сбоя или продолжения счета.** Решение большой задачи — длительный процесс, поэтому важно обеспечить сохранность данных на случай повторения расчета при машинном сбое или продолжения расчета, если он по каким-либо причинам был прерван. Для этого через определенные промежутки времени проводятся записи всех данных, нужных для продолжения расчета. При организации соответствующего обмена должна учитываться возможность сбоя в процессе записи. Например, если на ленту была записана очередная внешняя итерация, то по мере получения данных следующей итерации нельзя помещать результаты в те же зоны, так как случайный сбой может прервать работу, а ранее записанные данные будут испорчены частичной записью новых. Наиболее важные данные дублируются в разных участках ленты. Поскольку память на лентах приходится также экономить, применяют иногда достаточно сложные алгоритмы записи.

Приведем пример такого алгоритма. Пусть по массивам одинаковой размерности  $A, B, C$  пересчитываются массивы той же природы  $A', B', C'$ . Если для счета  $A'$  нужны все три массива  $A, B, C$ , то результаты расчета будем писать в резервный массив  $P$ . Если для счета  $B'$  нужны  $B$  и  $C$ , то  $B'$  пишем на место  $A$ , если для  $C'$  нужно только  $C$ , то  $C'$  пишем на место  $B$ . Теперь массив  $C$  превратился в резервный.

В трехмерном варианте задачи из главы 2 организация аварийной записи может быть достаточно сложной, а расположение данных может отличаться от рабочего варианта. В двумерном случае с пересчетом разностных коэффициентов аварийная запись занимает не более 30 зон, так что здесь может быть организован поочередный «сброс» данных в две группы по 30 зон [37].

**6. Вопросы отладки.** Особое внимание при организации вычислительного процесса в больших задачах следует уделять отладке. Обычно различают три вида отладочных работ: синтаксическую отладку, арифметическую отладку и отладку алгоритма. Эти названия достаточно условны, но применяются они часто.

---

\*) Кроме того, на обмены расходуется довольно много процессорного времени, хотя, может быть, и меньше, чем на пересчет.

При использовании программных средств (язык кодирования и другие средства автоматизации) возможны ошибки, связанные с неправильным использованием конструкций языка или других средств. Здесь самым подходящим методом можно считать составление перечня наиболее частных ошибок, специфичных для каждого программиста, и проверки программ согласно этому перечню. Использование развитых языков программирования с хорошей диагностикой, диалоговые редакторы и дистанционный запуск задач в пакет упростили этот вид отладки.

Арифметическая отладка должна убедить программиста в полном соответствии вычислительного алгоритма и составленной программы. На этом пути могут встретиться очень большие трудности, возникающие при отладке древовидного алгоритма, так как число возможных путей выполнения программы может оказаться грандиозным, и проверить все пути окажется невозможным. Конечно, можно пытаться доказать правильность программы, но это удастся только в простейших случаях. Поэтому проверяют отдельные практически наиболее вероятные пути и делают отсюда вывод, что верны все возможные пути. Основной метод отладки — это встроенные отладочные выдачи, позволяющие проследить путь работы программы, а также проверить соответствие тестовому варианту. Сам тестовый вариант можно подготовить с помощью упрощенной отладочной программы. После отладки операторы, порождающие выдачи, могут быть устранены из текста программы.

Разработаны специальные отладочные программы, позволяющие работать как в пакетном, так и диалоговом режиме [61—63, 23]. Существо таких программ сводится к прерыванию выполнения основной программы при удовлетворении заданного критерия (например, выполнения команды с определенным кодом, выполнения команды с определенным номером, выборки или записи числа в некоторую ячейку, перехода на другую команду и т. д.).

Мы сформулировали эти критерии в терминах машинных команд, однако их можно сформулировать и в терминах некоторого языка отладчика. После прерывания выполнения команд основной программы программист задает специальные команды, которые могут обеспечивать выдачу нужной информации, осуществлять «прокрутку» отдельных частей программы, передавать управление в нужное место и т. д. Следует отметить, что как отладочные выдачи, так и отладочные программы требуют тщательной отла-

ботки схемы отладки на основе глубокого знания алгоритма. Основной принцип арифметической отладки при любой методике — за один проход проверить всю программу, лучше сказать, отлаживаемый блок. Для этого надо предусмотреть действия на случай «застревания» программы — заикливания, авоста по запрещенным операциям, передачи управления в запрещенную область и т. д. Проще всего этого добиться, разбив блок на мелкие части, каждая из которых отлаживается независимо от других. Тогда входные данные для каждой части могут готовиться на основании тестового варианта и задаваться специальными отладочными вставками.

Наиболее трудоемка отладка алгоритма. Здесь не удастся использовать тестовую задачу с малым числом счетных узлов. Поэтому методические приемы отладки здесь разнообразнее. Помимо отладочных вставок и выдач, помимо программ-отладчиков предусматривается постановка заранее запланированных экспериментов и проверка заранее подобранных утверждений [64].

В качестве отладочных могут служить выдачи на терминал или печать. В случае использования терминала счет задачи прекращается в ожидании указаний автора задачи.

Запланированный эксперимент сводится к включению в счетную схему дополнительных вариантов отдельных частей. Переход на другой вариант может осуществляться после просмотра соответствующей отладочной выдачи. Здесь важно, что обычно вся промежуточная информация сохраняется, даже имеется возможность вернуться назад «по алгоритму» на несколько шагов и провести расчет по другой ветви.

Утверждения подбираются разработчиком алгоритма и представляют собой априорную информацию. Например, может быть известно, что некоторая величина положительна, или меньше другой, убывает от шага к шагу и т. д. В качестве таких утверждений могут выступать соотношения баланса, поведение разностей высокого порядка и многое другое.

Отладка планируется обычно одновременно с работой над алгоритмом, схемами организации вычислительного процесса и кодированием программы. Отладка выполняется сначала по отдельным блокам, а затем комплексно [2].

При отладке уделяется специальное внимание контролю вводимых и промежуточных величин. Для контроля

должна предусматриваться выдача введенных и промежуточных данных [2].

При изложении вопросов отладки мы ограничились только изложением принципов отладки не потому, что это менее важный этап работы, чем, например, структурная схема алгоритма. Во-первых, детальное рассмотрение схемы отладки было бы слишком обременительно для читателя, а отдельные моменты все равно соответствовали бы тем же принципам. Во-вторых, на схему отладки оказывают весьма сильное влияние индивидуальные особенности коллектива разработчиков, поэтому общее здесь и единственно ценное — это принципы организации отладки.

**7. Схемы счета.** Следует сказать, что почти все отмеченные нами стороны организации вычислительного процесса в больших задачах находили свое отражение в самых первых расчетах сложных физических задач. Однако современные расчеты приобретают характер численного эксперимента, являющегося моделью физического эксперимента. И если физический эксперимент отражает всю многоликость изучаемого явления, то численный эксперимент, как его модель, должен обладать таким же свойством [4].

Таким образом, численный эксперимент — это работа с целым классом прикладных задач. В процессе эксперимента приходится менять физическое и математическое описание задачи. Изменению должны подвергаться численные алгоритмы и, соответственно, комплекс программ.

Вместе с тем на примере многогруппового расчета видно, что общая функциональная схема расчета остается, вообще говоря, неизменной. Меняются лишь блоки, их функции остаются неизменными. Например, расчеты ячеек реактора могут проводиться в различных приближениях, с разными краевыми условиями. Еще разнообразнее численные алгоритмы.

Следовательно, для численного эксперимента необходимо иметь целую библиотеку модулей, из которых можно образовать по некоторой схеме счета [8] комплекс программ для решения конкретной задачи [9]. Число блоков, или модулей, может достигать в некоторой предметной области нескольких сотен и даже тысяч. Они объединяют в себе труд многих людей. Для успеха численного эксперимента важно, насколько трудоемка сборка программы для решения конкретной задачи. Возникающие здесь проблемы решаются методами модульного программирования.

Модульные системы берут на себя рутинную работу по сборке программы из библиотечных модулей, сопряжения их по оперативной и внешней памяти. Однако при этом не должно создаваться иллюзии, что рассмотренные нами вопросы организации вычислительного процесса исчезли — просто они стали решаться на стадии разработки модульной системы, схем счета и библиотеки модулей.

## **§ 2. Технологические этапы вычислительного эксперимента**

**1. Пять этапов вычислительного эксперимента.** В исследовании сложных явлений методом вычислительного эксперимента можно выделить ряд этапов.

Работа начинается с физической и математической формулировки задачи. На этом этапе определяется круг физических процессов, подлежащих исследованию, и строится физическая модель. Само слово модель показывает, что строится приближенное физическое описание интересующей нас системы. Искусство построения физической модели сводится к правильному выбору существенных явлений и к их адекватному описанию с помощью физических соотношений. Например, при рассмотрении реакторных задач мы ограничились для описания поведения нейтронов уравнением переноса, не учитывающим ряд существенных в данной задаче процессов, таких, скажем, как взаимодействие нейтронов друг с другом. При изучении околоземных космических полетов часто не учитывают влияние на полет космического аппарата таких космических тел, как Луна и планеты. Реальные физические процессы происходят в пространстве трех измерений и поэтому должны описываться тремя пространственными координатами. Если же удастся усмотреть в разбираемом явлении некоторую симметрию, то при надлежащем выборе системы координат число переменных для пространственного описания удастся уменьшить.

Одновременно с построением физической модели строится математическая модель. Это описание физических законов с помощью тех или иных математических средств. В качестве таких средств могут быть дифференциальные уравнения с граничными и начальными условиями, интегральные уравнения и др. Математическая модель может быть представлена в виде некоторой оптимизационной задачи.

Характерная особенность математической модели — неединственность. Например, можно дифференциальные уравнения заменить интегральными. В нейтронных задачах вместо интегро-дифференциальных уравнений часто рассматривают интегральные [36]. В электродинамике вводят потенциалы для электромагнитных полей и задачу нахождения полей сводят к задаче нахождения потенциалов [65]. Можно привести и другие примеры [66]. Это обстоятельство используется для выбора наиболее удобной с точки зрения расчетов формы модели.

Второй этап — разработка дискретной модели или вычислительной схемы. Под дискретной моделью понимается совокупность приемов, позволяющих по некоторым исходным данным получить численное значение интересующей нас величины. Это могут быть разностные методы, разложение исходных функций в ряды, методы интерполяции. Арсенал методов вычислений велик и дискретная модель представляет собой обычно совокупность ряда таких методов. Таким образом, последняя сводит решение задачи к последовательному выполнению арифметических операций, выполняемых с помощью вычислительной машины.

Третий этап состоит в разработке программного комплекса. Он включает разработку различного рода схем, написание программ, проведение отладки, тестовые просчеты.

Четвертый этап — собственно расчеты по готовым программам, т. е. эксплуатация готового комплекса. Эксплуатация включает в себя не только проведение серий расчетов с различными входными данными, но и определенную перестройку комплекса для приспособления его к решению некоторого класса прикладных задач [8].

Пятый этап состоит в анализе результатов расчета. Анализ результатов проводится с точки зрения всех этапов. В результате анализа могут быть приняты решения об изменении любой из моделей, а также программного комплекса и характера проводимых расчетов. Таким образом, после пятого этапа, как правило, происходит возврат к первому этапу. Наблюдается итерационный процесс [8, 50].

**2. Построение физической и математической моделей.** Перейдем к более подробному рассмотрению каждого этапа. На этапе построения физической модели или, как иногда говорят, физической постановки задачи, описывается совокупность физических процессов и различных констант,

указываются необходимые допущения. Как правило, такое описание завершается некоторой системой уравнений. Однако последнее не обязательно. Например, задача о движении тела в центральном поле сил может быть сформулирована так: рассчитать траекторию движения тела в поле центральных сил по законам классической механики. Задается закон центральных сил и начальные координаты и скорость тела. Разумеется, это простейший пример. В таких случаях вывод системы уравнений переносится на стадию формулировки математической модели.

При изучении сложных процессов в самой постановке задачи можно раскрыть некоторую структуру, т. е. разбить весь сложный процесс на более простые процессы и указать между ними связь. Например, в реакторных задачах выделяются процессы переноса нейтронов, выгорания, теплопередачи и т. д. Каждый из этих процессов в свою очередь можно разбить на более мелкие. Скажем, выгорание разбивается на выгорание тех или иных групп изотопов. В некоторых случаях связь между процессами может быть взаимной. Такова, например, самосогласованная система уравнений Власова для кулоновской плазмы [67]. В этом случае можно выделить процессы переноса ионов в плазме под действием собственных кулоновских полей и полей внешних источников и процессы создания электрических полей, где в качестве источников служат упомянутые выше ионы:

$$\frac{\partial N_a}{\partial t} + v \frac{\partial N_a}{\partial r} + e_a \left( E_m + E_0 + \frac{1}{c} [v B_0] \right) \frac{\partial N_a}{\partial p} = 0, \\ \operatorname{rot} E_m = 0, \quad (4.1)$$

$$\operatorname{div} E_m = 4\pi \sum_a e_a \int N_a(r, p, t) dp,$$

где  $N_a$  — плотность ионов,  $v$  — скорость ионов,  $p$  — импульс,  $e_a$  — заряд ионов,  $E_m$  — собственное электрическое поле ионов,  $E_0$  и  $B_0$  — внешние электрические и магнитные поля. Такая задача встречается при изучении динамики сильнооточных ионных пучков в ускоряюще-фокусирующих устройствах.

Изучение структуры сложного процесса позволяет свести его к ряду простых, возможно, уже исследованных процессов. Подобный анализ мы будем называть модульным анализом физической модели задачи, поскольку он



позволяет свести задачу к более простым «физическим модулям».

Разработка математической модели, помимо уже упомянутых уравнений, включает математическую постановку задачи: формулируются дополнительные условия, необходимые для однозначной разрешимости уравнений, вводятся с учетом физической модели дополнительные ограничения на искомые величины, как, например, условия на бесконечности, требования к непрерывности решений. Узкие пики могут быть заменены  $\delta$ -функциями. Проводится анализ особых точек, выбирается система координат и т. д.

Структура, полученная на этапе формулировки задачи, естественно, находит отражение в математической модели. Математическая модель может привести к дополнительной структуре. Часто модульный анализ проводится при построении математической модели. Например, математическая модель для описания газодинамического движения плазмы в электромагнитном поле очень сложна. Поэтому для описания такого движения строится модель магнито-гидродинамическая, сокращенно — МГД модель. Она включает: уравнения движения и неразрывности среды при заданном электромагнитном поле и температуре; уравнение для внутренней энергии при заданных газодинамических характеристиках движения и полях, уравнения Максвелла при заданных температуре и газодинамических характеристиках [66].

Отметим два важных аспекта, связанных с математической моделью. Одна и та же математическая модель может служить для описания различных физических процессов. Например, диффузия, теплопроводность, процессы намагничивания описываются одними и теми же уравнениями. Это обстоятельство широко используется в модульном программировании, существенно сокращая количество программ для решения различных прикладных задач [8].

С другой стороны, одной и той же физической задаче может соответствовать несколько математических моделей. Мы уже говорили об этом.

**3. Дискретная модель.** Особую роль в организации вычислительного эксперимента играют численные методы. Разработанную методику численного решения задачи мы назвали дискретной моделью.

Так же, как и в случае математической модели, в отношении дискретной модели мы имеем выбор, поскольку

одной и той же математической модели могут быть поставлены в соответствие различные дискретные модели. Обычно к численным методам предъявляются довольно жесткие требования по «экономичности» и объему используемой оперативной памяти. Экономичность определяется необходимым числом операций для нахождения решения. Следовательно, она определяет затраты времени на решение задачи. Оперативная память в конечном счете также определяет общее время решения задачи, поскольку использование внешней памяти при нехватке оперативной приводит к слишком частым обменам, ведущим к большим расходам машинного времени.

Приведенные характеристики численных методов неединственные. Важную роль играют точность методов и универсальность. Точность определяет, насколько близко к точному решению полученное приближенное. Обычно повышение точности в рамках одного и того же метода приводит к увеличению времени счета. Под универсальностью мы понимаем пределы изменения параметров задачи и самих численных методов, при которых данный метод позволяет получить некоторое приближенное решение.

Из других характеристик можно упомянуть сложность алгоритма. Естественно стремление к простому алгоритму в целях сокращения работы по программированию.

Дискретная модель должна отражать структуру предыдущих этапов. Помимо этого, она сама вносит дополнительные возможности для структурирования. Поясним это примером.

Если математическая модель содержит уравнения в частных производных, то для численного решения обычно применяется метод конечных разностей. Он позволяет свести решение задачи к решению систем алгебраических уравнений. Характерная особенность таких систем в случае многомерных задач — очень высокий порядок, не позволяющий разместить в оперативной памяти все необходимые данные. Чтобы обойти эту трудность, применяют аддитивные методы. При этом решение разностного уравнения заменяется на решение разностных уравнений значительно меньшего порядка. Правда, таких уравнений приходится решать большое количество. Для решения используются экономичные прямые методы. Выигрыш от применения подобных методов достигается за счет разбиения всего массива данных на блоки, легко размещаемые в оперативной памяти. При этом основная работа с данн

ми идет в пределах каждого блока. Это позволяет ограничить обмены между оперативной памятью и внешней памятью машины.

Аддитивные методы позволили решать многомерные задачи на машинах с ограниченной оперативной памятью. Кроме того, удалось свести решение сложной разностной задачи к решению ряда более простых задач. В этом состоит модульный анализ дискретной модели: последняя рассматривается как некоторая иерархия вычислительных модулей. Здесь под вычислительным модулем мы понимаем некоторую часть численного алгоритма, выделенную по тем или иным соображениям. При таком определении вычислительный модуль не обязательно должен быть некоторой законченной в функциональном отношении единицей [8], совокупность же модулей также образует модуль.

Подобным образом можно было бы определить «физический и математический» модули. Однако подобная терминология малоупотребительна, тогда как термин «вычислительный модуль» имеет некоторое распространение [69].

При разработке дискретных моделей учитываются некоторые дополнительные соображения помимо тех, о которых мы уже говорили. Известную роль играет сам характер задачи: много или мало вариантов будет рассчитываться по готовому программному комплексу. Учитывается: «программное окружение» и средства автоматизации программирования, которыми предполагается воспользоваться; порядок работы вычислительного центра, где будет решаться задача; возможные ресурсы и особые режимы работы, которыми можно обеспечить задачу. Естественно, при этом учитываются особенности конкретной вычислительной машины, ее конфигурация и даже, казалось бы, такие мелочи, как аппаратная реализация команд машины, разрядность, методы округления и т. д. Все эти моменты могут, с одной стороны, существенно повлиять на эффективность метода [70], а с другой, — отрицательно сказываться на адаптации готового программного комплекса к другим вычислительным машинам, или к условиям другого вычислительного центра.

**4. Методические вопросы разработки программ.** Третий этап — разработка программного комплекса. Она начинается с разработки схем счета, реализующих вычислительную модель, схем информационных связей между блоками, схем использования памяти разных уровней при

размещении данных задачи, схем обмена данными в процессе решения задачи, схем динамической перезагрузки блоков программ в процессе решения, схем отладки, тестовых просчетов и опытной эксплуатации программного комплекса. Следующие работы — программирование, отладка, подготовка тестов и опытная эксплуатация. После этого программный комплекс передается в эксплуатацию.

Обычно структура программного комплекса может быть описана следующим образом. В оперативной памяти постоянно присутствует управляющая программа — резидент комплекса. Она организует работу всего комплекса: управляет загрузкой блоков, выполняет передачу данных между блоками, организует запись в архив и т. д. Помимо управляющей программы и блоков, выполняющих основную содержательную работу по расчетам, используются наборы сервисных программ, управляющих обменов, запуском тех или иных блоков и т. д. В программных комплексах присутствуют блоки обработки начальных данных, программы для обработки результатов, программы сбора и обработки статистики по работе комплекса, блоки разбора непредвиденных ситуаций, контроля хода вычислений, блоки, ведущие дневник эксплуатации комплекса и др.

Такая структура позволяет вынести управляющие и информационные связи между блоками в сравнительно небольшую управляющую программу. В этом случае замена блоков, изменения в распределении оперативной и внешней памяти затронут лишь те разделы комплекса, к которым они непосредственно относятся.

При построении программного комплекса полезным оказывается понятие схемы счета. Схема счета естественно возникает при реализации решения задач вычислительного эксперимента как некоторая система взаимосвязанных блоков, выполняющих определенные функции. При переходе от одной задачи к другой в процессе проведения вычислительного эксперимента заменяется не сам набор функциональных блоков, а конкретная реализация каждого блока. В приведенной нами схеме расчета критических параметров реактора такими функциональными блоками были: расчет ячеек реактора, расчет многогрупповых констант и др. В каждом расчете присутствуют блоки, реализующие эти функции, хотя содержание их в различных расчетах может быть разным.

Таким образом, переход от одной задачи вычислительного эксперимента к другой может сопровождаться лишь

заменой отдельных блоков при сохранении функций, которые они выполняют.

С появлением средств автоматизации программирования, позволяющих автоматизировать сборку программ из частей, использование схемы счета получило дальнейшее развитие. Прежде всего схема счета «материализовалась» в виде некоторой «программы-скелета», в которой блоки, призванные выполнять содержательную работу, на предварительном этапе заменялись пустыми блоками и только на стадии сборки комплекса заменялись конкретными программами. Блоки, составлявшие «скелет», выполняли управляющие и вспомогательные функции и, следовательно, образовывали «организующую ткань» комплекса, неизменную для данного класса задач. Кроме того, проводилась четкая классификация программных компонент, вводились стандартные обозначения, что содействовало разработке весьма эффективных системных средств для сборки комплекса и его модификации [72, 73].

Дальнейшая работа связана с непосредственным программированием блоков (кодированием). Эта работа начинается с разработки микросхем на каждый блок макросхемы, детализации структуры данных, плана отладки [2].

**5. Выбор языка программирования.** Очень ответственным моментом следует считать выбор языка программирования. Этот выбор делается между проблемно-ориентированными языками высокого уровня и языками низкого уровня, типа автокода. Первые учитывают специфику задачи и процесса программирования. Они удобны для разработчика, сокращают сроки создания программ, уменьшают количество ошибок, делают программу наглядной и имеют еще много других преимуществ. Автокоды имеют одно основное преимущество — высокое качество объектной программы, т. е. программы, полученной после трансляции. Это свойство автокодов связано с тем, что они отражают специфику вычислительной машины и в принципе позволяют наиболее полно использовать ее аппаратные возможности.

Надо сказать, что для разработки больших программных комплексов вопросы качества объектной программы могут оказаться решающими. Качество готовой программы можно оценить по объему используемых ресурсов (машинному времени и памяти) и удобству эксплуатации в условиях конкретного вычислительного центра. Ранее

мы говорили о том, что программа, требующая установки большого количества магнитных лент и дисков, или использующая большой объем памяти на магнитных барабанах, в случае плохой приспособленности к условиям операционной системы или режиму работы в данном вычислительном центре, может оказаться непригодной для использования даже при условии высокой эффективности по машинному времени. Последнее зависит от рода задачи и от качества выполнения всех предыдущих этапов, и, как мы видели, не является самоцелью. Действительно, что дает экономия 5—10% машинного времени, если программу попросту нельзя эксплуатировать. Поэтому полезно рассмотреть некоторый обобщенный ресурс, куда помимо ресурсов по времени и памяти будут включаться и другие ресурсы вычислительного центра, связанные с разработкой и эксплуатацией программы. Естественно, что этот обобщенный ресурс не может превышать некоторый предельно допустимый, определяемый важностью задачи и возможностями вычислительного центра. С другой стороны, ясно, что улучшение показателей программы по ресурсам требует увеличения трудозатрат и, в частности, при одном и том же коллективе разработчиков ведет к увеличению времени разработки. Последнее, однако, тоже имеет верхний предел. Иногда при программировании больших задач программный комплекс успевает морально устареть, еще не увидев свет [71].

Таким образом, при разработке программного комплекса выбор языка и других средств автоматизации нуждается в системном подходе.

Имеется группа задач, которые при очень больших размерах программного комплекса (десятки тысяч команд) оказываются весьма терпимыми к ресурсам машинного времени, но зато очень требовательны к удобству эксплуатации и критичны к срокам разработки.

В других случаях [71] программы могут также насчитывать десятки тысяч команд при сравнительно небольшом времени счета — не более десятков часов. Однако число вариантов расчета по готовой программе может достигать тысяч. Третий случай дают программы, время счета которых очень велико — сотни часов, зато число вариантов может быть малым.

Нас, в основном, интересуют два последних случая. Для них характерно сочетание языков высокого уровня и автокодов [2, 71, 48].

Интересен в методическом отношении подход к использованию средств автоматизации программирования на базе мониторной системы Дубна, изложенный в [74]. В виду важности вопроса приведем его здесь с некоторыми сокращениями.

Для решения газодинамических задач в одномерной области был разработан программный комплекс, насчитывающий 200 блоков. Общий объем программ — 45 тысяч машинных слов БЭСМ-6. Большая часть блоков была написана на фортране: по числу машинных слов — 93 %. На автокоде были написаны блоки, потребляющие существенную часть машинного времени. Выбор таких блоков определялся частотой обращения к ним в каждой точке разностной сетки. Суммарное время, затрачиваемое на работу таких блоков, составило 20 %. К автокоду прибегали также в тех случаях, когда было целесообразно работать с разрядами в машинном слове. Отказаться от использования разрядов можно было ценой увеличения информационных массивов. С помощью автокода оказалось целесообразным реализовать некоторые сервисные возможности, например, задание времени счета некоторого участка программы, чтобы обойти закикливание.

При написании блоков на фортране учитывались особенности транслятора с фортрана (транслятор не проводит оптимизации арифметических выражений, поэтому такая работа проводилась при программировании).

Если переменная с индексом многократно встречается в цикле, ее значение лучше присвоить рабочей переменной.

В арифметических выражениях не следует без особой необходимости применять целые и вещественные переменные. Так, например, запись  $S = 2 * 8.31 * a$  экономнее записи  $S = 2 * 8.31 * a$ , если  $a$  — вещественная переменная.

Оператор IF арифметический транслируется короче, чем IF логический. Вообще сравнения удобно проводить с помощью оператора GO TO вычисляемый, который транслируется одной передачей управления.

Нельзя пользоваться двумерными и трехмерными массивами, так как работа с индексами сделана неэкономно. Такие массивы следует заменять одномерными.

Связь по информации между подпрограммами — блоками — осуществлялась через COMMON-блоки, что гораздо экономнее как по времени, так и по размеру объект-

ной программы. COMMON-блоки использовались в программе для организации буфера при обменах с внешней памятью, для передачи промежуточной информации из блока в блок через оперативную память, для организации рабочих полей, общих для ряда блоков (что приводит к экономии оперативной памяти, так как в противном случае для каждого блока заводится свое рабочее поле).

Вопросы обмена с внешней памятью могут быть решены различными путями: с помощью операторов ввода-вывода фортрана, с помощью стандартной программы обмена с барабаном IWR, с помощью комплекса стандартных программ обмена с лентами, дисками и барабанами, и, наконец, с помощью экстракода обмена при программировании на уровне автокода.

Первый путь малопригоден из-за потерь машинного времени при поиске необходимой информации, поскольку операторы обеспечивают только последовательный доступ к внешним устройствам. Кроме того, возникает много лишних обращений к внешней памяти, поскольку в качестве буфера используется буферный лист мониторной системы. При работе с магнитным барабаном может быть использован объем только одного барабана, что для задачи недостаточно. Стандартная программа IWR допускает режим прямого доступа к семи барабанам и экономит обращения за счет собственного буфера. Комплекс программ ОПД [29] допускает прямой доступ, в качестве буфера использует буферный лист мониторной системы, вследствие чего количество обращений к устройствам возрастает. Последний путь наиболее экономный, хотя и неудобный [29]. Учитывая приведенные соображения, все обмены с барабанами проводились с помощью ОПД, обмены с лентами с помощью ОПД, а информационные связи с программами на автокоде, созданными вне мониторной системы, осуществлялись с помощью экстракода обмена.

Поскольку все блоки, участвующие в решении некоторого варианта задачи, не удавалось разместить в оперативной памяти, была разработана экономная система вызова блоков с барабана в оперативную память. При этом анализировались три возможности работы с блоками: блок загружается один раз в оперативную память и там находится в течение всего времени решения задачи; загрузка осуществляется динамически, т. е. при каждом обращении к блоку происходит его загрузка, связанная с дополнительной работой по настройке загружаемого блока; вызов



блока происходит из библиотеки предварительно загруженных программ.

Наиболее экономный по затратам времени первый способ, однако он требует много оперативной памяти. Динамическая загрузка наименее экономна по машинному времени. Учитывая эти соображения, а также частоту использования блоков, удалось создать вариант загрузок, сокративший расходы машинного времени по сравнению с первоначальным вариантом почти в два раза.

Экономия оперативной памяти и машинного времени была достигнута также за счет отказа от использования оператора **FORMAT**, так как программы обработки этого оператора занимают 3000 машинных слов и расходуют довольно большое время. Выдачи накапливались на барабане и в дальнейшем распечатывались с помощью специальной программы.

В результате проигрыш машинного времени оказался практически неощутимым по сравнению с программами, написанными на автокоде.

Подобный подход целесообразно применять при оценке всех средств автоматизации программирования, а не только языков. Как мы видели на примере, для этого требуется проведение различных экспериментов. После этого приходится возвращаться к доработке предыдущих стадий и менять уже разработанные схемы.

В некоторых случаях, когда непосредственная проверка затруднена, прибегают к моделированию тех или иных сторон работы комплекса. Особенно полезным моделирование может оказаться при решении вопросов, связанных с построением схем, ответственных за динамическое использование памяти во время работы программы. Дело в том, что логическая структура алгоритма мало помогает в оценке частоты обращений к тем или иным блокам или данным в процессе решения.

**6. Методические указания общего характера.** Помимо изложенного подхода, заключающегося в сочетании в программном комплексе средств автоматизации различного уровня, иногда применяют двухстадийное программирование — сперва программа пишется на языке высокого уровня, а затем, после получения работающего варианта, переписывается на автокоде [2]. Такой прием может быть полезен как своего рода методическая доктрина при разработке всех компонент, необходимых для реализации вычислительного эксперимента. Действительно,

нет никакого смысла «отшлифовывать» программу, если придется в корне менять дискретную модель или одну из схем программного комплекса.

Этот подход применяется не только на стадии программирования, он характерен и для остальных стадий: для получения первого результата применяют «быстрые», пусть неэффективные, методы. После выяснения принципиальных вопросов можно перейти к разработке более эффективных приемов.

Надо научиться пользоваться простыми, стабильными и быстро ведущими к результату приемами и методами. Заметим [49], что выбор стабильных методов в арсенале средств автоматизации программирования — не такая простая задача. Нужно отказываться от сложных решений, если это не вызывается крайней необходимостью [6].

Разработка должна строиться по итерационному принципу, когда на первых итерациях отбрасываются все детали, которые можно отбросить. Дополнение деталями и сервисными возможностями разработки должно проходить постепенно. Что касается постепенной оптимизации, то для нее могут потребоваться некоторые дополнительные разработки, которые также не всегда надо начинать на первой итерации.

**6.1. О б о з н а ч е н и я п е р е м е н н ы х.** При программировании нужно принять меры к тому, чтобы программа была удобочитаема. О многих мероприятиях, направленных на это, мы уже говорили — разработка структурной схемы программы, исключение конструкций, не являющихся наглядными вроде GO TO [2, 6], сокращение количества переменных, передаваемых из блока в блок, сближение участков получения величин и их использования, а также о многих других, которые удастся усмотреть в конкретных случаях.

Сейчас мы обратим внимание на обозначение переменных и рабочий листинг программы. Переменные можно классифицировать по их роли: переменные, описывающие физические величины, переменные, описывающие параметры вычислительной модели, например, шаг сетки, переменные ввода-вывода и т. д. Естественно, нужно стремиться к тому, чтобы идентификаторы таких переменных отражали общепринятые обозначения. Соглашения на этот счет обычно формулируются до начала разработки комплекса. Помимо этого целесообразно ценой одного-двух символов в наименовании переменной зафиксировать

ее роль в программе, как это, например, предлагается в [76] для программы на фортране. С таких букв начинаются соответствующие переменные.

Роль в тексте	Тип переменной		
	вещественная, комплексная	целая, текстовая	логическая
Формальный параметр	P	K	KL
Глобальная переменная	A—H, Q—Y	L, M, N	LL, ML, NL
Локальная переменная	Z	I	IL
Счетчик цикла		J	

**6.2. Ввод исходных данных.** В некоторых случаях приходится вводить много исходных данных в программу. В этих случаях применяют запись данных на бланках в определенных позициях [2]. Иногда приходится вводить таблицы, геометрические объекты, данные, организованные в некоторые структуры и пр. Общий принцип, которого следует придерживаться — сократить по возможности количество вводимых величин, а если они получаются в результате некоторого другого расчета, то лучше их вводить с помощью перфоленты, или с использованием магнитного носителя информации.

В других случаях целесообразно воспользоваться интерполяционной формулой. Вместо задания вершин правильного шестиугольника парой чисел можно задать координаты центра и одной вершины. Наконец, имеется возможность непосредственно передавать в машину информацию, содержащую результаты физических экспериментов. Правда, для этого может потребоваться дополнительная аппаратура. Все же имеются случаи, когда сократить количество вводимых данных не удастся, как, например, при вводе статистических данных о землетрясениях, результатов давно сделанных экспериментов. В таком случае могут помочь упомянутые бланки и тщательный контроль. Помимо обычного визуального контроля, можно с помощью введения избыточной информации проводить машинный контроль. Например, можно подсчитывать сумму цифр, использовать некоторые соотношения или ограничения, которым должны удовлетворять данные.

Если вводятся данные определенной структуры, например, изотопная таблица, то можно организовать контроль на число данных, вводимых в каждой записи, что избавит от возможных пропусков и перестановок. Если вводимые числа, скажем, атомный номер, содержат всегда три цифры, то можно ввести еще четвертую, которая будет нулем для четной суммы цифр и единицей для нечетной.

Входные данные полезно объединять в группы по смыслу. Если данные вводятся с перфокарт, то целесообразно для каждой группы выбрать перфокарты определенного цвета. Если проводится некоторая серия расчетов, то полезно для группы предусмотреть ввод «по умолчанию», т. е. вместо данных этой группы вводится некоторый признак, который говорит, что данные этой группы должны быть такими же, как и в некотором предыдущем расчете. Желательно организовать ввод так, чтобы порядок ввода групп не имел значения. Тогда можно вообще не вводить данную группу, если она совпадает с предыдущим расчетом или соответствует некоторому стандартному варианту [2].

Появление новых устройств ввода, наподобие дисплеев со световым пером, оптических вводящих устройств, позволяющих вводить текст и графические данные, может существенно видоизменить процесс ввода данных.

Неизменными останутся: меры по контролю вводимых данных, структурирование входных данных, непременная контрольная выдача. Последнее — абсолютно обязательная стадия. Выдача может проводиться лишь частично, однако, предусмотрена должна быть в полном объеме. Предметом особой заботы является формат выдачи, поскольку обычно расчет каждого варианта сопровождается полной выдачей входных данных и служит своеобразным паспортом выдачи результатов. Поэтому формат должен содержать исчерпывающую информацию, с подробными записями о входных величинах.

**6.3. Организация выдачи.** Несколько слов об информации выдач. Перечислим виды информации в выдачах: входные данные; сведения о данном расчете хронологического и административного порядка; сведения о физической модели; сведения о математической модели; сведения о дискретной модели; данные о программном комплексе; дневник проведения расчета; методы контроля расчетных величин; результатная выдача.

Поясним смысл этих видов информации. О входных данных мы уже говорили. Сведения второго типа включают дату завершения и начала расчета, могут включать фамилию лица, проводящего расчет, номер какого-либо директивного документа, согласно которому проводится расчет, имя заказчика и др.

Сведения о моделях связаны с тем, что в большинстве случаев программный комплекс создается с помощью или в рамках какой-либо системы пакетного типа. Поэтому при сохранении общей схемы упомянутые модели в нем могут поменяться. Если при этом характер рассчитываемых величин не меняется, то редко меняется и формат выдач результатов. В таком случае через некоторое время уже невозможно установить, к каким моделям относится данный расчет. Например, при счете газодинамических задач может меняться уравнение состояния вещества и обычно в выдачах указывается, к каким состояниям вещества относится данный расчет. Здесь очень важно подчеркнуть, что соответствующие комментарии к примененным моделям ни в коем случае нельзя задавать перед расчетом с перфокарт или каким-либо другим способом. Обязательно будет ошибка. Комментарии должны выводиться в автоматическом режиме, путем непосредственного анализа перед выдачей текста программного комплекса. Это относится к большинству комментариев (так, время запуска следует получать с часов, встроенных в машину).

Сведения о программном комплексе могут содержать данные об использованных программных блоках, данные по схемам обменов, распределения памяти, об используемых ресурсах и т. д. Естественно, что такие сведения могут представлять интерес, если они меняются при работе с программным комплексом.

Дневник расчета может включать даты \*), когда проводился расчет, данные о смене шага, о контроле тех или иных величин, на основании которого могли меняться рабочие параметры.

Методы контроля включают данные о способах контроля получаемых в расчете величин. Например, иногда следят за поведением разностей высокого порядка или за отклонением какой-либо балансовой величины. В каждом

---

\*) Здесь имеются в виду не только даты начала и конца расчета, но и даты проведения промежуточных сеансов расчета, если задача считалась в несколько приемов.

варианте методы контроля могут быть разными. Здесь же могут быть указаны пути, по которым шел расчет. Например, в случае «ветвистой» программы может быть указано, что внутренняя энергия рассчитывалась до некоторого шага по одной формуле, а после — по другой.

Результаты могут выдаваться в различном виде: таблицы, графики, изображение на экране дисплея, некоторая графическая конструкция на бумажной ленте, или передаваться по линиям связи на управляемый объект.

Поскольку имеет место очень большое разнообразие методов обработки результатов, начиная от выдачи простой таблицы до распознавания закономерностей по результатам расчетов, отметим только некоторые методические моменты.

Программы обработки результатов целесообразно выделить в отдельный блок.

Если решаемая задача не относится к задачам реального времени, то выдачу целесообразно сосредоточить в конце расчета [2], собирая необходимые данные на внешнем машинном носителе. Исключение могут составлять многочасовые расчеты, когда полезно выдавать отдельные величины для контроля хода расчета. Выдача должна быть управляемой и позволять выводить не все полученные данные, а только необходимые, причем управляемой должна быть и обработка выдач. По мере возможности надо стремиться к сохранению максимально возможного количества результирующей информации, поскольку заранее обычно не ясно, могут ли понадобиться те или иные данные расчета.

**6.4. Хранение результатов расчета.** Для решения вопроса хранения результирующей информации полезно проработать методы ее уплотнения [75].

Эти методы обычно связаны с различными приемами упаковки результатов. Если не требуется сохранять в числовой информации много разрядов, то можно в одно машинное слово поместить несколько чисел. Иногда может оказаться, что часть разрядов в массиве не меняется на протяжении ряда чисел, не говоря уже о порядке или знаке числа. Это также открывает некоторые возможности упаковки. Но особенно эффективны различные методы интерполяции [70]. Они могут дать выигрыш в несколько порядков.

**6.5. Листинг программы.** Листинг появляется после завершения программирования и в дальнейшем

используется как рабочий инструмент для отладки программы. Для удобства пользования следует позаботиться о наглядности листинга. В [76] рекомендуется листинг по каждому блоку, образующему некоторую программную единицу [56], разбивать на более мелкие единицы, называемые секциями. Секции нумеруются в каждой программной единице последовательно и отделяются друг от друга знаком тире. Всего имеется 72 позиции, в первой позиции располагается символ C. Следующая строка содержит комментарий с номером и названием секции. Она начинается символами CL. Номер секции записывается с 17-й позиции и может принимать значения от 1 до 9. Каждая секция может делиться на подсекции. Число подсекций также может быть не больше девяти. Запись номера подсекции начинается с 21-й позиции, а запись названия с 30-й позиции. В начале строки помещается два символа CL. Одна подсекция отделяется от другой пропуском строки. Номер подсекции изображается с помощью записи 4.2, где 4 означает номер секции, а 2 — номер подсекции. Комментарии располагаются, начиная с 7-й позиции, чтобы не мешать слежению за метками. Для выделения комментариев можно оставить пустую строку. Запись операторов следует начинать с 10-й позиции, а метки располагать так, чтобы их последняя цифра стояла в 5-й позиции.

В [77] речь идет о синтаксически ориентированном листинге. В этом случае запись операторов выбирается такой, чтобы можно было выделить синтаксические конструкции. Так, например, вложенные циклы можно расположить «лесенкой». Если написать вспомогательную программу, то можно уже готовый листинг преобразовать к нужному виду.

Обратим внимание также на необходимость достаточно подробных комментариев к листингу [56]. Вместе с тем комментарии не могут заменить описания программы, поскольку комментарии достаточно большого объема даже затрудняют работу с листингом. В связи с этим полезно позаботиться об описании программы, которое также целесообразно хранить на машинных носителях вместе с текстом программы. К этому вопросу мы вернемся в следующем параграфе.

7. Методика отладки. Отладка программы представляется крайне важным и сложным этапом работы [2,3]. Ранее мы останавливались на вопросах отладки. Ввиду

важности отладки вернемся к соображениям, высказанным ранее, дополнив их.

Итак, ранее мы разделили отладку на синтаксическую, арифметическую и методическую [2]. По поводу синтаксической отладки можно только добавить, о чем мы уже говорили, что не следует использовать непроверенные, нестабильные или просто незнакомые конструкции. Следует ограничиться необходимым их минимумом. Тогда дело сведется к внимательному визуальному контролю. Чтобы сократить выходы на машину, этот контроль лучше проводить по некоторому перечню возможных ошибок [64], который, например, может, начинаться так: наличие описаний типов переменных; положение декларативных операторов; правильность обозначений вещественных переменных; они не должны начинаться с I, J, K, L, M, N; — употребление неописанных переменных и т. д. \*).

Под арифметической отладкой мы понимаем просто соответствие программы алгоритму. Термин арифметическая отладка не совсем удачен, так как речь идет о проверке логики программы и о прочих операциях не арифметического характера. Однако этот термин употребляется и за неимением другого мы будем им пользоваться. Для проведения арифметической отладки обычно готовится тестовая задача. Блоки программы разбиваются на микроблоки [78] и результаты работы каждого микроблока сравниваются с результатами расчетов тестовой задачи.

Размер микроблока выбирается с таким расчетом, чтобы в случае возникновения расхождения между результатами тестовой задачи и расчетов по отлаживаемой программе можно было легко локализовать источник возникших расхождений. При подготовке тестовой задачи следует обращать особое внимание на вырожденные случаи и граничные значения параметров [47]. Если при составлении тестовой задачи приходится делать много расчетов, то их можно выполнить с помощью некоторой вспомогательной программы.

При составлении арифметического теста не обязательно следовать естественному ходу выполнения отлаживаемой программы. Последнее могло бы в отдельных случаях разветвленной задачи привести к большому объему тее-

---

\*) Речь идет о программах, написанных на фортране.



товых расчетов. В этом случае вместо связанного теста употребляют несвязанный тест, в котором естественный ход вычислений по программе прерывается, вносятся искусственные значения переменных, результаты расчета с которыми проверяются по тесту. Такой подход менее надежен, но зато и менее трудоемок, а иногда и просто единственный.

Реализация отладки осуществляется с помощью вставок, которые можно после отладки убрать, перетранслировав программу. Вставки делаются в некоторых контрольных точках и в конечном счете передают управление на некоторую программу отладки. В программе отладки может проводиться сравнение результатов работы микроблока с результатами тестовой задачи, сопровождаемое выводом на печать соответствующего сообщения, или просто выдачей результатов работы микроблока. Там же может проводиться загрузка некоторых искусственных значений переменных с целью проверки работы следующего блока. Если нужно проверить работу блока по нескольким вариантам значений переменных, то организуется цикл. Вставка может служить просто для трассировки программы, и тогда полезной информацией будет сообщение о том, что данная контрольная точка пройдена. При построении отладочной программы следует позаботиться, чтобы за один проход получить максимально возможную информацию [79].

В связи с этим в программе отладки должны быть запланированы действия на случай возникновения аварийных ситуаций. Наилучший вариант состоит в том, чтобы выдать нужные величины для разбора аварийной ситуации и заслать нужные данные для работы следующего микроблока, взятые из расчетов тестовой задачи. В связи с этим целесообразно данные тестовой задачи иметь «под рукой» при отладке, чтобы можно было воспользоваться ими в автоматическом режиме.

Для проведения арифметической отладки можно воспользоваться различными средствами автоматизации программирования, такими, например, как отладочные программы, работающие в пакетном режиме [61], а также диалоговые отладочные системы, работающие с машинными командами или в символах языка [62—64]. (Существует мнение, что диалоговая отладка не дает больших преимуществ [79].) Иногда программа отладки создается разработчиками на базе существующих системных средств. Вари-

антов здесь может быть много, однако есть два принципа отладки, которые не оспариваются никем: отладка должна тщательно планироваться заранее, каждый выход на машину должен обеспечивать получение максимально возможной информации [2, 47, 48, 80, 79].

Методическая отладка в корне отличается от предыдущих. Она проверяет работу метода, т. е. отлаживает дискретную модель. Однако по используемым средствам она походит на арифметическую отладку, и это служит основанием для их совместного рассмотрения. Весь арсенал средств арифметической отладки может использоваться и для методической. Помимо этого, широко применяется метод проверки утверждений [64], запланированные заранее эксперименты и тесты нового типа.

Остановимся на тестах. Тесты для методической отладки отличаются большим разнообразием, чем тесты для арифметической отладки. Это могут быть модельные задачи, допускающие аналитическое решение, задачи, имеющие меньшую размерность, для которых уже есть работающие программы. При решении уравнений иногда задают готовое решение, подставляют его в уравнение, получают правую часть, а затем с этой правой частью находят численно известное решение. Некоторые тесты основаны на проверке соотношений баланса. Есть тесты, основанные на другой математической модели. Есть чисто вычислительные тесты, например, удвоение шага, подсчет разностей и др. Можно проверять одну дискретную модель с помощью другой. Выбор методического теста, несмотря на обилие возможностей, трудная задача. Она решается на этапах формулировки соответствующей модели. При выборе теста естественно стремиться к тому, чтобы его использование не превратилось в самостоятельную задачу, равносильную тестируемой.

В [47] рассматривается некоторая система тестов: вырожденный тест; проверка граничных значений параметров; систематическая проверка частных функций; проверка аварийных реакций; комплексный тест блока; тест стыковки с ближайшими соседями.

Здесь фигурирует тест, не упоминавшийся нами, — тест проверки аварийных реакций. Пусть мы предусмотрели некоторые действия в случае расходимости итераций. Тест, проверяющий реакцию на эту аварийную ситуацию, должен убедить нас в правильности обслуживания программой аварийной ситуации.

После завершения автономной отладки блоков приступают к отладке групп блоков [2], а затем к комплексной отладке. Комплексная отладка может идти не только путем объединения блоков «снизу». В некоторых случаях бывает удобно отлаживать программу «сверху». В этом случае начинают отладку с головной управляющей программы, а еще не отлаженные блоки заменяют некоторыми макетами [2]. Для такой отладки «сверху» разработаны системные возможности в пакете прикладных программ Сафра [72]. Они используют понятие схемы счета. Отладка начинается с верификации схемы счета, которая постепенно наполняется блоками, проходящими отладку. Удобство отладки «сверху» состоит в том, что комплексная отладка практически исключается.

В качестве некоторого итога сделаем три замечания. После завершения отладки следует разгрузить программу от отладочных вставок и выдач. При этом, естественно, могут возникнуть ошибки (при любом исправлении могут возникнуть ошибки). Следовательно, для отладки важно, чтобы процесс поиска ошибок был «сходящимся». Отсюда — при внесении исправлений нужна особая тщательность. Внесение ни в коем случае не должно носить характера «заплат», поскольку при этом полностью исчезает обозримость внесенного. Наконец, любые индивидуальные средства отладки также приходится отлаживать. Чтобы процесс не получился «расходящимся», следует проявлять умеренность в выборе объема и индивидуальности средств отладки. Возможно, удастся обойтись типовыми средствами, лишь с небольшими изменениями.

В нашем изложении отладки мы не касались функционирования и принципов построения штатных средств отладки, знакомство с которыми крайне полезно. Обзор этих вопросов можно найти в [23] и в [64].

В ряде случаев может оказаться полезным использовать средства отладки для целей выявления «узких» мест в программе. По мнению некоторых специалистов [8], основное время затрачивается при счете программы на выполнение команд, составляющих всего 3% от общего числа команд. Приведенный выше пример использования фортрана и автокода в одной программе также показывает пользу проведения исследований для выяснения вопроса, куда же на самом деле при счете программы расходуется машинное время.

### § 3. Вопросы организации и информационного обеспечения разработок

1. Значение документации. В предыдущем параграфе мы изложили технологию проведения вычислительного эксперимента, имея в виду его содержательную сторону. Здесь мы рассмотрим организационную сторону технологии, которая находит свое отражение в информационных документах, сопровождающих разработку. Эти документы выполняют следующие функции:

- отражают проделанную работу на содержательном уровне, т. е. в них приводятся полученные формулы дискретной модели, разработанные схемы программного комплекса, текст написанной программы и т. д.;

- определяют план работы, начиная с указаний типа «разработать язык для обработки результатов расчета» и кончая перечнем формул, которые надо запрограммировать;

- являются отчетом о проделанной работе;

- служат обоснованием для проведения тех или иных работ в рамках данной организации, так как содержат утверждающие подписи;

- служат для сопровождения и эксплуатации уже готового программного комплекса.

Под сопровождением понимается выявление ошибок, разбор непредвиденных ситуаций, внесение изменений. Эксплуатационная документация содержит сведения, необходимые для нормального использования всех возможностей, заложенных в программном комплексе. Качественная документация способствует «непотопляемости» разработки при смене состава разработчиков, их болезни, включения в производственный процесс новых людей.

Для того чтобы документация не запаздывала по срокам, нужно соблюдать важный принцип разработки документации: ее подготовку нельзя откладывать «на потом» [2]. Она должна идти параллельно с продвижением разработки. Соображения о том, что многое придется переделывать в результате доработок — неубедительны, переделывать всегда легче, чем писать все заново, тем более, если эта работа будет выполняться постепенно.

У документации есть еще одна функция, связанная с внедрением разработки, разумеется, если такое предполагается. Качественная документация, разработанная вовремя, служит не только сокращению трудозатрат на

внедрение, но и некоторой рекламой разработки. В результате этого зачастую получают распространение не лучшие разработки, а хорошо и своевременно документированные.

**2. Варианты проведения разработки.** Возможны варианты, когда подготовкой физической модели занят «заказчик». Остальные работы проводит группа разработчиков, объединенная в тесный коллектив и работающая в рамках другой организации. В этом случае эксплуатация, сопровождение и доводка проводятся силами разработчиков. Следовательно, отпадает самостоятельная поставка программного комплекса. По существу, внедрение отсутствует. По такой схеме выполняются многие работы вычислительного эксперимента.

В другом варианте «заказчик» получает готовый программный комплекс и дальнейшие работы с ним проводит самостоятельно. В этом случае имеет место внедрение. Оба эти случая характеризуются узостью использования разработки, что вполне согласуется с узкой специализацией научных исследований.

Однако иногда разработка может представлять широкий интерес. К таким разработкам могут относиться разработки по реакторным расчетам, газодинамическим задачам и многим другим. В этом случае приобретает большое значение внедрение, а требования к качеству документации повышаются.

**3. Общие вопросы стандартизации.** Описанные нами этапы вычислительного эксперимента, а также разработка документации к нему представляют собой сложный вид научной и производственной деятельности, требующей взаимосвязанной работы больших коллективов людей. Даже в тех случаях, когда в разработке непосредственно принимает участие всего один человек, в целом труд носит коллективный характер, поскольку используются вычислительные средства, обслуживаемые коллективами инженеров, операторов и др., в процессе разработки затрагиваются производственные интересы многих специалистов, работающих в том же вычислительном центре [2]. В этих условиях большое значение приобретает стандартизация. В нашей стране действует Государственная система стандартизации. Можно так определить ее задачи: установление требований к качеству продукции, установление единой системы показателей качества, единой системы мер, единой терминологии, норм и требова-

ний в области проектирования и производства, внедрение унификации, установление стандартов на технологию и организацию работ и некоторые другие.

Особое внимание в системе Государственной стандартизации уделяется вопросам документации. По документации создаются целые комплексы стандартов. Существуют комплексы Единая система конструкторской документации ЕСКД, Единая система программной документации ЕСПД и некоторые другие.

Для стандартов введены категории: ГОСТ — государственные стандарты, ОСТ — отраслевые стандарты, РСТ — республиканские стандарты, СТП — стандарты предприятий. Категория стандарта определяется теми объектами, на которые он распространяется.

Для всех стандартов имеется область распространения. Государственные стандарты обязательны для применения во всех предприятиях и организациях страны: отраслевые — для предприятий и организаций данной отрасли; стандарты предприятий — распространяются на данное предприятие [86].

Порядок разработки и утверждения стандартов соответствует их категории и области распространения. Так, например, стандарты предприятий разрабатываются и утверждаются на предприятиях.

Остановимся на комплексе стандартов ЕСПД. Этот комплекс состоит из 29 стандартов. Объектом их является программная документация и вопросы организации разработки программ [87—115]. ЕСПД регламентирует набор программных документов, содержание, момент разработки документа — по отношению к стадиям разработки самой программы, оформление документов, порядок внесения изменений, а также размножения документов, применяемые обозначения и термины, графические обозначения для построения блок-схем, устанавливает последовательность и содержание работ при проведении разработок программ. Обозначение каждого стандарта состоит из пяти цифр: ГОСТ 19.202-78. Через черточку дается год утверждения стандарта. Первые две цифры 19 означают класс стандарта, цифра после точки — порядковый номер группы, две следующие 02 означают порядковый номер в группе.

Область распространения ЕСПД включает все без исключения разработки программ и программной документации.

Поэтому вполне естественно, что стандарты определяют лишь достаточно общие требования на документацию и технологию разработки программ и, по существу, не ограничивают разработчика в выборе материала или стиля изложения и технологических приемов, связанных со спецификой задачи. Дальнейшие уточнения, учитывающие эту специфику, могут быть проведены в рамках стандартов предприятия.

При изложении вопросов документации на программный комплекс мы будем придерживаться требования ЕСПД. Поскольку на документацию по другим этапам — физическая постановка, математическая, вычислительная — стандартов нет, так же как на технологию выполнения работ, то здесь нам придется опираться на некоторый опыт, изложенный в ряде работ [2, 4, 5]. Этот опыт вполне может рассматриваться как некоторые стандарты предприятий.

**4. Стандарты вычислительного центра на документацию и организацию разработок.** Прежде чем переходить к изложению организационных сторон описанной нами технологической цепочки, сделаем три общих замечания.

Первое замечание. В дальнейшем для удобства изложения мы будем ориентироваться на большие задачи математической физики, составляющие суть многих научных и инженерных расчетов, лежащих в основе вычислительного эксперимента. Вместе с тем многое можно применить и для других задач.

Второе замечание касается стандартизации. Обычно в коллективе разработчиков сама собой появляется система передового опыта проведения вычислительных работ. Сюда относятся и вопросы технологии, как, например, рекомендуемые обозначения, рекомендуемые конструкции языка, регламентирование использования ресурсов машины, различные соглашения для организации совместной работы, дополнительные рекомендации по документации [117—119, 76], формальный порядок проведения разработок и многое другое. Квалифицированные специалисты придают большое значение фиксации этого опыта [78].

**5. Общие соображения о стиле и содержании документов.** Третье замечание касается общих проблем документации разработок вычислительного эксперимента. Основное правило разработки документации — разрабатывать документацию параллельно с самой разработкой — мы уже

упоминали. Теперь о содержании документа. Мы не будем останавливаться на обычных чертах любой хорошей рукописи: четкости, логической стройности, убедительности, краткости и точности, конкретности и т. д. Это все свойства хорошего стиля изложения [116]. Несколько необычны другие требования. Изложение должно быть доступным. По мере возможности, следует избегать ссылок на источники, без знакомства с которыми дальнейший материал не может быть понят. Особенно, если эти источники не входят в серию изучаемых документов. Ссылки в основном должны служить для понимания приоритетных вопросов или для углубленного изучения предмета. Особенно следует избегать терминов из специальных дисциплин, которые легко заменяются более ясными большинству синонимами. Лучше потратить несколько строчек на разъяснения, которые могут показаться специалисту тривиальными, чем ставить кого-либо из участников или пользователей в тупик. Надо заранее помнить, что знакомиться с документами придется людям разных узких специальностей и разной квалификации: руководители, физики, математики, вычислители, программисты. Уже этого списка достаточно, чтобы понять, что документ, написанный в стиле научной статьи, совершенно не выполнит своей задачи.

Существует другая точка зрения, согласно которой в документах следует добиваться краткости изложения, в частности, за счет ссылок. Мы же считаем, что при выборе между краткостью и доступностью лучше пожертвовать первой. Увеличение документа на 20% вряд ли прибавит лишние трудозатраты в той степени, в которой их прибавляет поиск нужных документов по катастрофически разрастающемуся дереву ссылок.

**6. Автоматизация документирования.** Весьма перспективным следует считать использование вычислительной машины для хранения и обработки документации. Тексты документов могут храниться на внешних носителях вместе с текстами программы. Приведем возможные виды обработки.

Это — редактирование текста и автоматическая сборка из заготовок нужного документа. Выдача текста в стандартном виде с учетом того или иного шаблона, даже если при подготовке он оформлялся произвольным образом. Исключаются переписывание текста, машинописные работы.



Это — широкое использование шаблонов, например, в виде заготовок титульных листов, отдельных стереотипных частей документации. Возможно накопление в специальном архиве различных вариантов таких стереотипов, которые можно использовать при подготовке соответствующей документации методом сборки по некоторой «схеме счета», например, с помощью системы Сафра [72]. Возникает возможность использования при построении документации модульного принципа.

Записи в текстовом архиве можно использовать как средство взаимной информации участников разработки. Например, можно выяснить, какие обозначения глобальных переменных на данный момент уже использованы другими разработчиками, и сообщить им о занимаемых обозначениях. Функций такой почтовой связи можно и расширить, например, сообщать о дате выхода на машину, передавать данные по отладке при организации комплексной отладки и т. д. Может поменяться весь стиль проведения разработки, если функция администратора разработки перейдет к машине, машина же будет выполнять функции лаборанта и секретаря.

Для программной документации возможна автоматизация самого процесса создания документа, например, за счет поиска по тексту программы, скажем, при составлении перечня идентификаторов программы. Делались попытки автоматического построения графических блок-схем с использованием текста программы. О надежности такого метода мы уже говорили — ничто не будет забыто или перепутано.

Подобный метод имеет дополнительные преимущества несколько другого рода. Он обеспечивает большие удобства при передаче программной разработки на внедрение. Заманчива перспектива легкости и надежности охраны авторских прав на использование программного комплекса, а также выборочной защиты информации о разработке.

Известно, что на пути между пользователем и автором документов стоит отдел размножения и библиотечный отдел. Предлагаемая система подготовки документов в развитом варианте может либо совсем ликвидировать необходимость в подобных посредниках, либо свести ее к допустимому минимуму и тем самым существенно сократить время получения нужной информации о разработке.

Мы подробно остановились на таком методе подготовки документации в силу ряда причин. Во-первых, tradi-

ционный метод в принципе не может решить вопрос подготовки документации на должном уровне: слишком много промежуточных рутинных, черновых операций, требующих большого штата низкоквалифицированных работников. Во-вторых, он в принципе не решает многих вопросов, вроде охраны авторских прав, почты, администрации разработок и многие другие. В-третьих, уже на сегодня отдельные элементы изложенного подхода широко используются, правда, в более ограниченном варианте, чего иногда вполне достаточно.

В настоящее время имеется ряд готовых систем, реализующих отдельные стороны подготовки документации с помощью машины. В [120] рассматривается система для машины БЭСМ-6, обеспечивающая хранение текстовой информации в специальном архиве на магнитной ленте, редактирование этой информации, некоторую сервисную обработку с выводом на АЦПУ-128, терминал, перфоленту с последующей распечаткой на пишущей машинке с перфолентой Оптима-528. В результате последней операции появляется обычный машинописный материал. В [121] рассматривается система информационно-поискового характера. При невозможности воспользоваться готовыми системами иногда целесообразно разработать собственными силами некоторые указанные выше средства.

Очень важное замечание касается шаблонов или моделей документов [122]. Модели документов могут экономить много сил и времени на всех этапах работы с документами. Существо такого метода состоит в том, что все рубрики, заголовки и типовые места какого-либо документа заготавливаются заранее путем проведения анализа наилучших образцов. Помимо этого шаблон снабжается замечаниями о содержании разделов, о стиле изложения. Стил ь изложения может иллюстрироваться примерами. В дальнейшем словосочетания устойчивого характера, а также расположение материала могут использоваться при составлении документа. Таким образом, шаблон — это некоторый стандарт на данный документ. Эффективность шаблонов складывается из экономии труда разработчика и высокого качества документа. Многие вопросы уже продуманы. Качество документа обеспечивается, поскольку шаблон разрабатывается особо квалифицированными людьми. Наконец работу по шаблонам можно автоматизировать. За шаблонами громадное будущее,

поскольку в написании документов на самом деле совсем не так много творческой работы, как принято думать. Именно поэтому решение вопроса путем создания специальности «технический писатель» вряд ли у нас будет популярным.

7. Материалы, появляющиеся на этапах разработки физической, математической и дискретной моделей. Перейдем к описанию документации на разработки вычислительного эксперимента. Документы, которые связаны с работами, выполняемыми до разработки программ, мы выделим в группу документов по тематическим разработкам. Это материалы, возникающие при разработке физической, математической и вычислительной моделей. Они носят характер научных отчетов и поэтому должны оформляться в соответствии с [116]. Разобьем эти материалы на ряд групп.

1. Перечень материалов.

1. 1. Перечень административных материалов.

1. 2. Перечень материалов по разработке.

2. Административные материалы.

2. 1. Докладная записка по разработке в целом.

3. Основные материалы, возникающие по ходу разработки.

3. 1. Материалы по физической модели.

3. 2. Материалы по математической модели.

3. 3. Материалы по дискретной модели.

4. Группа описательных материалов.

4. 1. Описание физической модели.

4. 2. Описание математической модели.

4. 3. Описание дискретной модели.

5. Материалы, описывающие использование построенных моделей.

5. 1. Материалы по работе с физической моделью.

5. 2. Материалы по работе с математической моделью.

5. 3. Материалы по работе с дискретной моделью.

Нумерация введена нами для удобства изложения и переносить ее в документы не следует.

На всю разработку в целом могут вестись и другие документы, например, «дневник разработки». В дневнике отражается ход работ, приводятся планы, распределяются ресурсы и т. д.

Дальнейшее наше изложение вычислительного эксперимента ориентируется на научно-исследовательские работы.

**7.1. Документы, возникающие при разработке физической и математической моделей.** Проведение разработки, как правило, начинается со встреч заказчика и разработчиков. К этому времени группы, которые будут заниматься задачей, обычно уже сформированы. Эти встречи могут носить характер семинаров, на которых сначала выступают заказчики с физической постановкой задачи, а затем разработчики с изложением своих возможностей и встречных предложений по постановке задачи. В результате заказчики готовят документ, который мы называли заданием на расчет.

Оно содержит:

1. Титульный лист с утверждающей подписью заказчика и согласующей подписью от разработчиков.

2. Общую пояснительную часть, в которой может даваться общее описание задачи, значимость задачи, а также ссылки на директивные документы по ней.

3. Физическую постановку задачи, которая может содержать несколько подпунктов и включать уравнения, а также формулировку сделанных приближений.

4. Систему обозначений, физические константы.

5. Описание величин, которые должны быть рассчитаны.

6. Описание разброса значений параметров, для которых придется проводить расчет, а также возможные вариации постановки задачи.

7. Сроки проведения работ. В этом документе должна достаточно четко определяться структура задачи на физическом уровне, так, чтобы задачу можно было представить как совокупность «физических модулей», из которых впоследствии путем набора будет составляться их конкретный вариант.

После подробного изучения полученного задания группа разработчиков готовит докладную записку в адрес ответственного лица \*).

Примерное содержание записки может быть таким:

1. Общие соображения по заданию на расчет: необходимость внести изменения, сократив или расширив постановку задачи, соображения о перспективности задачи, преемственность с предыдущей деятельностью группы и прочее.

---

\*) Во многих случаях докладная записка не готовится.

2. Пути реализации с точки зрения численных методов, а также существующего программного обеспечения и ресурсов машины.

3. Общий объем работы — необходимые методические разработки, трудозатраты на получение вычислительной модели, составление и отладку программы.

4. Соображения о последующем сопровождении и эксплуатации программы.

5. Необходимые административные мероприятия — временное привлечение к работе «узких» специалистов, финансовые вопросы, особые режимы работы на машинах и др.

Содержание записки должно быть тщательно продумано, отличаться конкретностью и аргументированностью, так как на основании записки принимаются решения, зачастую требующие больших затрат.

Одновременно с изучением задания на расчет и переходом к разработке следующих моделей начинается работа над двумя документами: 4.1 «Описание физической модели» и 5.1 «Материалы по работе с физической моделью». Иногда эту работу выполняет заказчик, но чаще она выполняется совместно.

Документ 4.1 — это отчет по физике задачи и поэтому он выполняется в соответствии с [116]. В отличие от 3.1, он содержит не только окончательные результаты, но и обоснования, доказательства, выкладки, описания сделанных приближений, возможные пути уточнения, оценки точности модели, результаты эксперимента и другие сведения. В 5.1 содержатся способы выделения из всего класса задач тех, которые представляют наибольший интерес. Например, указывают области значений температур и плотностей, при которых еще справедливы те или иные выражения для уравнений состояния или физические условия, при которых от уравнений одного типа приходится переходить к уравнениям другого типа для описания некоторого явления. Для небольших задач эту часть иногда включают в 4.1 или в 3.1. В других случаях бывает полезно явно указать способ выделения из целого класса конкретных задач путем соответствующего набора «физических моделей». Документ 5.1 может формулироваться в таких выражениях:

1) для высоких плотностей ... уравнения полей выбираем в виде...

2) для плотностей в интервале .... в уравнении ... полагаем ...

3) для задач с цилиндрической симметрией в уравнении ... член ... заменяем на ...

Здесь многоточие — это конкретные ссылки на формулы из 3.1 или конкретные условия. Таким образом, здесь дается набор «физических схем», или система правил, по которой этот набор может быть построен. Набор и определяет класс задач. Сведения о физике задачи оказались распределенными по трем документам: сведения, необходимые для построения расчетов, — в 3.1, сведения для расширения класса задач — в 4.1, сведения, нужные для практического выбора нужной физической схемы, — в 5.1. Естественно, все варианты «физических модулей» должны быть приведены в 3.1, поскольку они должны быть в конце концов отражены в расчете. Однако там не требуются сведения, каким задачам какие варианты соответствуют.

Далее идет математическая модель 3.2. Она включает обычно шесть разделов.

1. Перечень величин в задачах и их классификация. В этом разделе приводится перечень величин и поясняется смысл каждой. При математической формулировке задачи могут появиться новые величины, которых не было в 3.1, в то же время некоторые величины могут «исчезнуть». Например, в 3.1 могут приводиться уравнения Максвелла с электрическими и магнитными полями в качестве неизвестных, в 3.2 они заменяются потенциалами; может появиться плотность электромагнитной энергии, поток электромагнитной энергии. Обычно количество величин при переходе к 3.2 существенно возрастает.

Поскольку выбор обозначений на этом этапе играет определяющую роль для выбора обозначений на всех последующих этапах, то надо подойти к этой проблеме с особым вниманием.

Обратим внимание также на то, что уже здесь можно подробно разобрать, какие величины в будущем расчете будут входными, а какие — выходными.

Далее указываются величины, не меняющиеся на протяжении расчета, т. е. константы. Обычно для них приводится следующая классификация: константы, постоянные для всего класса задач, и константы, связанные с переходом от одной задачи к другой, а также константы, определяющие вариант расчета данной задачи. Выделяют константы математические и физические. Есть тип констант, связанных с граничными условиями.

С обозначением связано еще и понятие глобальных величин. Так называются величины, имеющие одинаковый смысл всюду в «математической модели» задачи. Таким образом, обозначение для такой величины должно быть одинаковым во всех «математических модулях» задачи \*). Другое дело — «локальные величины», имеющие смысл только в пределах данного «математического модуля». Их обозначения в разных модулях могут совпадать. Для этого, разумеется, нужно привести перечень величин по «математическим модулям» с указанием их смысла. Однако зачастую складывается такое положение, что в разных «математических модулях» «глобальные величины» неудобно обозначать одной переменной. В таком случае для них можно ввести разные обозначения, причем следует оговорить, что под ними понимается одна и та же величина.

Отметим также, что в этом разделе в виде подраздела помещаются сведения о структуре входных и выходных величин. Подробно об этих вопросах мы говорили ранее.

2. Подсчет коэффициентов уравнений по формулам. Этот раздел может возникнуть в связи с переходом к другой системе единиц, с «обезразмериванием» уравнений. Такие преобразования помогают сократить число параметров, от которых зависит вариант задачи. А это очень важно, так как, по существу, позволяет экономить машинное время. Например, задача с нулевыми начальными условиями

$$a \frac{d^2 y}{dx^2} + bxy = c \quad (4.2)$$

заменой  $u = (a/c)y$ ,  $k = b/a$ ,  $u'' + kxu = 1$  приводится к уравнению с одним параметром. Ясно, что численное изучение (4.2), приведенного к уравнению с одним постоянным коэффициентом, потребует меньших усилий.

3. Математическая формулировка задачи. Здесь дается, например, окончательный вид уравнений, удобный для применения численных методов.

На этом этапе готовятся все необходимые аналитические выкладки (преобразования уравнений, введение новых переменных, разложение искомых функций в ряды с последующим выводом уравнений для коэффициентов,

---

\*) Как мы уже говорили, «математический модуль» — это часть математической модели, выделенная по некоторым соображениям. Например, группа уравнений.

преобразования, связанные с особыми точками и т. д.). Результаты всех этих преобразований и приводятся в этом разделе.

4. Результаты исследования свойств и характеристик решений. Математическая постановка включает в себя изучение таких вопросов, как существование и единственность решения, дифференциальные или иные свойства решения, класс функций, в котором ищется решение. Результатом такой работы является вся постановка задачи, включая формулировку уравнений с дополнительными условиями. Здесь же мы имеем в виду разного рода зависимости в особых точках, в точках разрывов, асимптотические зависимости и др.

5. Тестовый раздел. Здесь приводятся тесты, обеспечивающие проверку методики задачи. Этими тестами могут быть аналитические решения задачи, полученные при упрощающих предположениях (например, автомодельные решения), могут быть тесты, полученные в результате сведения задачи к задаче с меньшей размерностью, для которой могут существовать уже отработанные численные методы решения. Иногда используются искусственные тесты с правой частью, полученной в результате подстановки в уравнение заданных функций, а также тесты, полученные решением исходных уравнений с отброшенными членами. Последняя операция нередко позволяет получить аналитическое решение.

В этом же разделе приводятся интегральные соотношения, некоторые проверочные соотношения, учитывающие особые свойства решений: периодичность решения, перемену знака при переходе через некоторую точку. Полезными могут быть также априорные оценки величин, а иногда и качественные суждения о поведении решения. Вариантов тестов может быть очень много. В этом разделе приводятся конкретные тесты, наиболее удобные для данной задачи.

6. Обработка результатов расчета. Очень часто те величины, которые могут быть получены в результате решения уравнений, преобразовываются для выдачи на печать. Видов такой обработки может быть достаточно много. Иногда полученные величины пересчитывают по конечным формулам

$$u_i = f(x_1, \dots, x_m), \quad (4.3)$$

где  $x_k$  — набор посчитанных величин,  $u_i$  — величины,



идущие на выдачу. К числу выполняемых при такой обработке преобразований относятся масштабные преобразования, переход к другим координатам, образование новых величин (например, энергии поля по значениям напряженности) и т. п. В некоторых случаях подсчитываются интегралы или вычисляются производные:

$$u(r) = \frac{\partial u}{\partial r}, \quad v(x) = \int_{x_1}^x u(x') f(x') dx'. \quad (4.4)$$

В ряде случаев отыскиваются максимальные или минимальные значения, подсчитываются некоторые функционалы, проводятся интегральные преобразования, например, Лапласа, находятся линии уровня, а в точках линий уровня организуется вывод на печать каких-либо других величин. Очень часто те или иные функции заменяют выражениями, которые их приближают, т. е. различного рода аппроксимациями.

Особый раздел составляют методы обработки результатов расчета, основанные на распознавании той или иной закономерности. Один из способов — взяв некоторое выражение с неизвестными коэффициентами, по методу наименьших квадратов найти эти коэффициенты. Подобные методы могут использоваться для построения различных эмпирических формул, полезных при конструировании, нахождении асимптотических формул. Эти методы похожи на те, которые применяют при обработке результатов экспериментов.

Отметим еще группу преобразований, связанных с графическим представлением результатов расчетов. К ним относятся построения различных проекций, объемных изображений, разрезов, используемых не только при конструировании, но и при графическом изображении функциональных зависимостей. Иногда для наглядности строят изображения процесса, имитируя плотность количеством точек, выданных на квадратный сантиметр, или другими способами.

Мы подробно остановились на этом разделе, так как с появлением задач, решаемых в реальной геометрии и описывающих реальные, а не модельные объекты, роль его в задачах неизмеримо возросла. Уже не представляется возможным просто просмотреть выдачу, содержащую десятки миллионов чисел, да и увидеть что-нибудь в ней почти невозможно. Значительную часть этой работы можно

выполнить уже на стадии математической постановки задачи.

Раздел 4.2 по стилю изложения и содержащемуся в нем материалу аналогичен разделу 3.2. Изложение должно соответствовать [116]. Содержание 5.2 также аналогично содержанию 5.1. При этом все варианты 5.1, естественно, должны найти отражение и в 5.2. Однако в 5.2, как мы уже говорили, могут появиться новые варианты (упоминавшаяся нами двойственность описания с помощью интегральных и дифференциальных уравнений и т. д.), т. е. число «математических модулей» в рамках одной «физической схемы», иначе говоря, задачи, для описания одной функции может оказаться больше единицы. На чем остановиться здесь выбор,— таков смысл этого раздела.

Относительно существования разделов 4.2 и 5.2 замечание аналогично замечанию о 4.1 и 5.1.

**7.2. Д и с к р е т н а я м о д е л ь.** Следующий раздел относится к вычислительным методам и является ответственнойнейшим этапом. От него зависит успех проведения вычислительного эксперимента в большей степени, чем от других этапов.

Вот примерные работы, выполняемые на этом этапе: разработка разностной схемы, в результате которой появляется система разностных уравнений; разработка метода решения полученной системы разностных уравнений. Независимо от того, является ли задача линейной или нелинейной, в конечном счете мы приходим к алгебраической системе уравнений высокого порядка. Здесь выполняются работы, связанные с корректностью постановки разностной задачи, а также проводятся все необходимые преобразования разностных уравнений. Решаются вопросы построения сеток, разработки тестов, последовательности выполнения расчетов и др. Особо отметим вопросы, связанные с ориентировкой на будущее программирование, в частности, вопросы необходимой памяти, вопросы обозначений, построения структуры данных и др. Однако окончательно этот круг вопросов решается на стадии разработки программного комплекса.

Результаты этой работы в окончательном виде приводятся в 3.3. Этот документ ввиду большой важности часто проходит этап согласования и утверждения, особенно в тех случаях, когда в программировании принимают участие дополнительные силы, а не только те, которые до сих пор принимали участие в разработке.

Приведем возможные разделы в 3.3.

1. Перечень величин в задаче. Здесь может играть важную роль подраздел, связанный с обозначениями. От удачно выбранных обозначений существенно зависит наглядность счетных формул раздела и в конечном счете будущего программного комплекса. Особые трудности обычно связаны с индексами, которых может быть много — до 5 и более.

Другой подраздел связан со структурой входных и выходных данных задачи, упомянутой ранее классификацией величин. Обратим внимание на появление дополнительных величин, связанных с вычислительным алгоритмом (например, сеточные данные, величины, управляющие точностью расчета).

2. Возможные значения числовых констант разного происхождения. Особо рекомендуется выделить разброс значений величин, связанных с численными методами, например, сеточных данных, так как от них зависит точность и другие свойства численной модели, с одной стороны, а с другой, — необходимые ресурсы при программировании. Выделяются величины, управляющие счетом с точки зрения численных методов: к ним относятся точки выдачи контрольных значений, записи промежуточных данных, смены алгоритмов, данные, управляющие числом итераций и многие другие.

3. Описание графа управления или информационного графа, а иногда того и другого вместе. Информационный граф — это схема передачи информации от одного блока к другому, или информационная схема связей блоков. Граф управления — это вычислительная схема, т. е. последовательность выполнения блоков, вообще говоря, зависящая от хода выполнения расчета. Этот случай соответствует ветвящемуся алгоритму.

В этом разделе указываются пути выбора последовательности работы блоков в конкретном программном комплексе, реализующем конкретную задачу класса.

4. Раздел расчета сеток. В этом разделе при использовании сеточного метода помещается расчет сеток, а также другие, например, расчеты весов квадратурных формул и коэффициентов интерполяции.

5. Содержательная часть алгоритма. Формулы, приведенные к окончательному счетному виду для каждого модуля с указанием перечня величин, фигурирующих в нем.

6. Тестовая задача. В этом разделе могут быть приведены данные арифметического теста или какие-либо другие проверки, о которых уже шла речь.

7. Обработка результатов расчета для выдачи. Здесь, помимо упомянутых ранее расчетов, могут содержаться расчеты, связанные с численным методом решения уравнений (например, расчеты расхождения балансов).

О содержании 5.3 мы уже говорили — это соображения о выборе, например, параметров, если с их помощью подбирается схема счета. В 3.3 указано как это делать, а в 5.3 даны случаи применения. В 4.3 излагается методика получения дискретной модели. Кроме того, в 4.3 должна найти отражение разработка численных методов, которые обязательно сопровождаются методическими расчетами.

Изложенная система документации не отражена в стандартах и носит рекомендательный характер. Естественно, что в конкретных случаях можно дать более содержательные рекомендации по стилю изложения, набору материалов в рубриках, составу самих документов. Такого рода шаблоны, на наш взгляд, целесообразно создавать для узких классов задач (например, для газодинамических расчетов, нейтронно-физических расчетов, магнитогидродинамических и т. д.).

**7.3. Программирование.** Дальнейшая часть работы связана с разработкой программного комплекса, который мы для краткости далее будем именовать программой \*).

Это важный и трудоемкий этап. Однако было бы неправильным видеть основные сложности реализации вычислительного эксперимента только в программировании. Следует еще раз подчеркнуть, что выделение в технологическом цикле пяти этапов носит в значительной мере условный характер. На самом деле все этапы тесно связаны между собой и служат основной цели — получению с необходимой точностью с приемлемыми затратами ресурсов количественного описания изучаемого физического явления, процесса или конструкции.

Это единство мы постарались отразить и в документации, появляющейся на различных этапах вычислительного эксперимента, которая, по нашему мнению, должна разрабатываться на единой методической основе. В качестве

---

\*) Программный комплекс есть совокупность относительно независимых программ.

такой основы целесообразно воспользоваться комплексом взаимосвязанных государственных стандартов под общим названием «Единая система программной документации» — ЕСПД. Об этом комплексе мы уже говорили попутно при обсуждении общих вопросов стандартизации. Сейчас мы займемся ЕСПД специально.

При описании этапов физической и математической постановки и разработки дискретной модели все документы были разделены на пять групп \*). Напомним их. К первой группе был отнесен перечень, содержащий список всех документов. Вторая группа — «административные» документы. Этим термином мы определили такие документы, как докладные и служебные записки, приказы и распоряжения, связанные с разработкой.

Третья — это рабочие материалы. Они содержат сведения, необходимые для разработки и изготовления соответствующих компонент вычислительного эксперимента.

Четвертая группа — это документы, необходимые для сопровождения. Под сопровождением понимается внесение изменений, доводка, выяснение узких мест и причин аварийных ситуаций.

Пятая группа — это документы, необходимые для проведения эксплуатации. В них сообщаются сведения, необходимые для организации правильной работы и использования всех возможностей компонент \*\*).

Мы сейчас покажем, что подобный подход к документации вполне согласуется с методической основой комплекса стандартов ЕСПД. Для этого обратимся к описанию стандартов.

**7.3.1. Краткое описание содержания стандартов ЕСПД.** Первая очередь стандартов ЕСПД из двадцати девяти ГОСТов устанавливает единые в стране правила документирования программ для ЭВМ и систем обработки данных независимо от их назначения и области применения. В частности, стандарты ЕСПД устанавливают виды программ и программных документов, их обозначения и стадии разработки, правила учета и хранения програм-

---

\*) Эти группы не имеют отношения к группам стандартов ЕСПД, о которых речь будет идти ниже.

\*\*) Например, выбор физического приближения и математическая формулировка задачи — это этап технологического цикла вычислительного эксперимента. С другой стороны, построенные физическая и математическая модели — это некоторая составная часть или компонента,

мных документов, правила оформления документов, выполненных печатным способом, и требования к содержанию каждого вида программного документа.

Стандарты ЕСПД [87] подразделяются на группы.

**Общие положения** (нулевая группа). Сюда относится ГОСТ 19.001-77 «ЕСПД. Общие положения». Он устанавливает цели, назначение, область распространения, классификацию стандартов, входящих в ЕСПД. ГОСТ 19.002-80 «ЕСПД. Схемы алгоритмов и программ. Обозначения условные графические» определяет графические символы для построения схем. ГОСТ 19.003-80 «ЕСПД. Схемы алгоритмов и программ. Правила выполнения» содержит описание правил построения схем. ГОСТ 19.004-80 «ЕСПД. Термины и определения» содержит определение ряда терминов, применяемых в программировании.

**Основополагающие стандарты** (первая группа). ГОСТ 19.101-77 «ЕСПД. Виды программ и программных документов» устанавливает виды программ (компонента и комплекс) и виды программных документов. ГОСТ 19.102-77 «ЕСПД. Стадии разработки» устанавливает стадии разработки программ и программной документации, этапы работы и их содержание. ГОСТ 19.103-77 «ЕСПД. Обозначение программ и программных документов» устанавливает структуру обозначений программ и программных документов. ГОСТ 19.104-78 «ЕСПД. Основные надписи» устанавливает форму, размеры, расположение и порядок заполнения основных надписей листа утверждений и титульного листа. ГОСТ 19.105-78 «ЕСПД. Общие требования к программным документам» определяет общие правила к оформлению документов на любом носителе данных, а также составные части документа. ГОСТ 19.106-78 «ЕСПД. Правила выполнения программных документов, выполненных печатным способом» содержит описание формата машинописных страниц, расположение материала на страницах, описание рубрик, оформление иллюстраций, ссылок и тому подобное.

**Правила выполнения документации разработки** (вторая группа). ГОСТ 19.201-78 «ЕСПД. Техническое задание» описывает формат и содержание программного документа, именуемого «техническое задание» (см. ниже). ГОСТ 19.202-78 «ЕСПД. Спецификация». То же, что и предыдущий стандарт, но

только по отношению к программному документу «спецификация».

**Правила выполнения документации изготовления (третья группа).** ГОСТ 19.301-79 «ЕСПД. Порядок и методика испытаний» определяет формат и содержание документа по методике и порядку проведения испытаний программ.

**Правила выполнения документации сопровождения (четвертая группа).** ГОСТ 19.401-78 «ЕСПД. Текст программы» описывает программный документ, содержащий запись программы с комментариями. ГОСТ 19.402-78 «ЕСПД. Описание программы», ГОСТ 19.403-79 «ЕСПД. Ведомость держателей подлинников», ГОСТ 19.404-79 «ЕСПД. Пояснительная записка» содержат описания соответствующих программных документов, на которых мы остановимся несколько позже.

**Правила выполнения эксплуатационных документов (пятая группа).** Эта группа включает стандарты, содержащие описания эксплуатационных программных документов: ГОСТ 19.501-78 «ЕСПД. Формуляр», ГОСТ 19.502-78 «ЕСПД. Общее описание», ГОСТ 19.503-79 «ЕСПД. Руководство системного программиста», ГОСТ 19.504-79 «ЕСПД. Руководство программиста», ГОСТ 19.505-79 «ЕСПД. Руководство оператора», ГОСТ 19.506-79 «ЕСПД. Описание языка», ГОСТ 19.507-79 «ЕСПД. Ведомость эксплуатационных документов».

**Правила обращения программной документации (шестая группа).** Здесь содержатся: ГОСТ 19.601-78 «ЕСПД. Общие правила дублирования, учета и хранения», ГОСТ 19.602-78 «ЕСПД. Правила дублирования, учета и хранения документов, выполненных печатным способом», ГОСТ 19.603-78 «ЕСПД. Общие правила внесения изменений», ГОСТ 19.604-78 «ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом». Содержание этих стандартов не требует пояснений.

**7.3.2. Виды программных документов по ЕСПД и их содержание.** Перечислим виды программных документов и приведем их содержание [88].

**Спецификация.** Включает состав программы и программной документации. Требования к содержанию и оформлению спецификации устанавливаются в [95].

**Ведомость держателей подлинников.** Указывает перечень предприятий, где хранятся подлинники программных документов. Требования к содержанию и оформлению этого документа устанавливаются в [108].

**Текст программы.** Запись программы с необходимыми комментариями.

**Описание программы.** Описываются функции программы, ее логическая структура, составные части и связи между ними, сведения о языке программирования, входных и выходных данных. При описании логики программы необходима привязка к тексту программы. Требования к содержанию и оформлению этого программного документа содержатся в [97].

**Порядок и методика испытаний.** Содержит требования, подлежащие проверке при испытаниях программы, а также порядок и методику их контроля. Требования к содержанию и оформлению этого документа описаны в [115].

**Техническое задание.** Описывается назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки. Об этом документе см. [94].

**Пояснительная записка.** Составляется к эскизному и техническому проектам программы. Она включает: схему алгоритма, общее описание алгоритма и (или) функционирования программы, а также обоснование принятых технических и технико-экономических решений. Требования к содержанию и оформлению приведены в [113].

**Ведомость эксплуатационных документов.** Содержит перечень эксплуатационных документов на программу. Ниже приведены наименования и содержание видов эксплуатационных документов. Требования к оформлению и содержанию этого документа устанавливаются в [109].

**Общее описание.** В отличие от описания программы, в котором содержатся сведения, относящиеся к логической структуре и функционированию программы, здесь помещаются сведения о классе решаемых задач, ограничениях для применения, об используемых методах, назначении программы, области применения, минимальной конфигурации технических средств и о составе програм-



много обеспечения. Требования к содержанию и оформлению общего описания можно найти в [99].

**Руководство системного программиста** содержит сведения, необходимые для обеспечения функционирования и настройки программы на условия конкретного применения. Об этом документе см. [100].

**Руководство программиста.** В этом документе приводятся сведения для эксплуатации программы: запуск, подготовка входных данных, меры по контролю за ходом расчета [101].

**Руководство оператора.** Содержит сведения, необходимые для общения оператора с вычислительной системой в процессе выполнения программы [102].

**Описание языка.** Описание синтаксиса и семантики языка [103].

**Руководство по техническому обслуживанию.** Разрабатывается для программ, проверяющих работоспособность технических средств [114].

Помимо требований к содержанию и оформлению конкретных видов программных документов, ЕСПД устанавливает также общие требования к программным документам [92] и требования к программным документам, выполненным печатным способом [93].

Теперь мы приведем состав документов, соответствующих программированию так, как это было сделано для двух предыдущих этапов вычислительного эксперимента.

1. Перечень документов.

1.1. Административные документы.

1.2. Документы по разработке программы.

2. Административные документы.

2.1. Докладная записка.

2.2. Прочие документы.

3. Рабочие документы по программе.

3.1. Техническое задание.

3.2. Эскизный проект. Пояснительная записка.

3.3. Технический проект. Пояснительная записка.

3.4. Рабочий проект.

3.5. Порядок и методика испытаний.

4. Программная документация сопровождения.

4.1. Текст программы.

4.2. Описание программы.

4.3. Ведомость держателей подлинников.

5. Эксплуатационная документация.

- 5.1. Общее описание.
- 5.2. Описание языка.
- 5.3. Руководство программиста.
- 5.4. Руководство системного программиста.
- 5.5. Руководство оператора.
- 5.6. Формуляр \*).

Обратим внимание на важное обстоятельство. Приведенные документы, помимо программных документов, описанных в ЕСПД, содержат документы и материалы, либо совсем не упоминающиеся, либо только названные, но подробно не описанные. К первым относятся документы первой и второй групп. Ко вторым — такие документы, как эскизный проект, технический проект и рабочий проект. Последние документы рассматриваются совместно в [89] в краткой форме.

**7.3.3. О согласовании и утверждении документов.** Перейдем к описанию стадий разработки. Разработка программы часто связана с привлечением дополнительных специалистов по программированию. После некоторого изучения уже разработанной нами группы документов разработчики составляют докладную записку, в которой в произвольной краткой форме излагается сложившаяся обстановка, дается оценка предстоящей работы, а также перечисляются необходимые ресурсы.

В оценку предстоящей работы включается краткий анализ возможных путей решения задачи, предложения по использованию существующих средств вычислительной техники и программного обеспечения. Оценка ресурсов включает в себя оценку человеческих и машинных ресурсов, сроков разработки, а также требования на дополнительные разработки системного или аппаратного характера.

Докладная записка тщательно готовится, так как задача ее авторов — получить разрешение на дальнейшую работу, связанную с расходом довольно больших и дефицитных ресурсов.

Подготовка документов включает три этапа [2]: 1) оформление, 2) согласование, 3) утверждение. Первый этап сводится к подготовке машинописного документа в соответствии с принятыми стандартами. Под документом собираются подписи. Согласование означает, что возра-

---

\*) Описание этого документа мы не приводили. Его можно найти в [98].

жения со стороны тех, кого этот документ может как-то затрагивать, отсутствуют. Утверждение означает согласие с документом ответственных лиц. Выполнение двух последних стадий может потребовать переделок, так что здесь, как и во всей разработке, имеется цикличность.

Отметим, что такая подготовка документов связана подчас с большой затратой труда. Однако попытка отнести ее «на потом» может привести к возникновению конфликтных ситуаций в коллективе и к неоднозначности решений.

**7.3.4. Разработка технического задания.** Дальнейшие стадии разработки соответствуют ЕСПД \*). Первая стадия — «Техническое задание». Она требует согласования и утверждения и оформляется в соответствии со стандартом [91, 92].

Для разработки технического задания осуществляется дополнительное изучение постановки задачи. При этом основное внимание обращается на вопросы, связанные с формулировкой класса задач, для решения которых создается программа. Если такого рода программы существовали раньше, то нужно уяснить, чем вызвана необходимость разработки новой программы. Это часто связано именно с постановкой задачи. Например, прошлый вариант касался одномерной задачи, а нужна двумерная. Было ограничение на шаги, не позволявшее считать в пространственных областях нужного размера. Наконец, программа была построена не по принципу модульности.

Метод решения задачи просматривается под углом создания программы. Естественно, что это делалось и раньше. Однако разработчики дискретной модели вынуждены в некоторой мере «абстрагироваться» от программистских вопросов, поскольку в противном случае «задание на программирование» вряд ли удалось бы создать в обозримые сроки. Кроме того, они не обязаны знать все тонкости процесса программирования.

Здесь же формулируются принципы реализации программы, а также требования на ее характеристики (быстродействие, использование ресурсов и т. д.).

На стадии разработки технического задания анализируются программные средства, обеспечивающие реализацию требований вычислительного эксперимента (напри-

---

\*) В ЕСПД рассматриваются стадии разработки, а в пределах стадии выделяются этапы работ. Для вычислительного эксперимента в целом [66] вводится понятие этапов технологического цикла.

мер, средства пакетного типа, позволяющие осуществлять сборку и сопряжение модулей комплекса).

В этом документе рассматривается структура входных и выходных данных, форматы выдачи. Определяются возможные методы тестирования, отладки, контроля хода расчетов. Принимаются предварительные решения об использовании средств программного обеспечения и, в частности, языков программирования.

В техническом задании «в первом приближении» должна быть дана оценка сроков разработки. Для этого требуется знать ее стадии и состав документов, которые надо готовить.

Приведем разделы технического задания согласно [94] с некоторыми комментариями.

1. **Наименование и область применения.** Указывается наименование и описывается область применения программы.

2. **Основания для разработки.** Указывается документ, на основании которого проводится разработка, организация, утвердившая документ, и тема, в рамках которой ведется разработка.

3. **Назначение разработки.** Указывается класс задач и ограничения для программы. К таким ограничениям могут относиться пределы изменения входных данных, которые допускает программа, ограничения на точность получаемых результатов и т. д.

4. **Технические требования к программе.** Функциональные характеристики: набор функций, временные характеристики и т. д. Указывается набор возможных режимов функционирования комплекса, например, рабочий режим, тестовый режим, режим с контролем, режим с промежуточной записью, режим с выдачей выходных данных на ленту или на печать АЦПУ. Здесь могут приводиться требования, предъявляемые к функциональным возможностям комплекса (например, обеспечение выполнения функций сборки, проведение расчетов, работы с архивом и прочие). Приводятся требования на контроль входных и выходных данных; требования к составу аппаратного и программного обеспечения, требования на доработку существующих средств и т. д.

5. **Технико-экономические показатели.** Для разработок научного характера здесь показатели, как правило, качественные.

6. Стадии и этапы разработки. Приводятся стадии и этапы разработки, а также состав программной документации.

7. Порядок контроля и приемки. Не требует пояснения.

8. Приложение. Сюда могут быть включены обоснования, которые в текст технического задания не включаются, но могут быть полезны при разработке, например, методические работы, схемы алгоритма и другие материалы.

Все документы ЕСПД описываются по единому шаблону: приводится некоторая общая шапка об объекте стандартизации и области распространения. Дается указание по оформлению, например, ссылкой на другой стандарт. Затем описываются требования к оформлению листа утверждения и титульного листа [91], информационной части [92], состоящей из аннотации и содержания, листа регистрации изменений [107]. Такая последовательность связана с тем, что каждый документ включает: титульную часть, информационную часть, основную часть, регистрацию изменений.

Для технического задания, как и для всех документов, титульная часть обязательна, информационная и регистрационная части могут отсутствовать.

Кроме этого, делается замечание о возможности дополнять, объединять разделы, уточнять их содержание.

7.3.5. Разработка эскизного проекта. Следующая стадия носит название эскизный проект. Если на стадии технического задания мы определили требования к программе, то на стадии эскизного проекта намечаются пути достижения поставленных целей.

На этой стадии осуществляется окончательное уточнение методов решения. Если в счетных формулах были параметры, управляющие выбором методов счета, то им приписывают определенные значения, т. е. здесь проводится окончательный выбор методов счета. Затем приступают к разработке алгоритма решения задачи, причем на этой стадии проработка деталей не проводится.

Степень ее детализации зависит от задачи и от «стиля» работы группы. Может создаться впечатление, что «Задание на программирование» предполагает полное формирование алгоритма. На самом же деле оно не дает всей последовательности выполнения операций в явном виде. Эта последовательность может быть получена посредст-

вом изучения задания и, что весьма важно, в виде нескольких вариантов. Приведем пример. Пусть вычисляются величины  $x = f(u, v, t)$ ,  $y = \varphi(u, v, t)$ ,  $z = \psi(u, v, t)$ . Ясно, что порядок вычислений с точки зрения окончательного результата здесь безразличен. В случае, если в формулы  $f$  и  $\varphi$  будут входить  $x$  и  $y$ :  $x = f(u, v)$ ,  $y = \varphi(x, t)$ , то здесь порядок не безразличен. А вот другой пример, из которого видно значение порядка вычислений. Пусть надо решить систему разностных уравнений:

$$\begin{aligned}x_n &= f(x_{n-1}), & y_n &= \varphi(x_{n-1}, y_{n-1}), \\n &= 1, 2, \dots, & x_0 &= a, \quad y_0 = b.\end{aligned}\tag{4.5}$$

Простейший вариант: вычисляется  $x_n$  по первой формуле по  $x_{n-1}$ . Затем  $y_n$  по второй формуле по  $x_{n-1}$ ,  $y_{n-1}$ . Используя найденные  $x_n$  и  $y_n$ , переходим к  $x_{n+1}$ ,  $y_{n+1}$ . Таким образом, под промежуточные данные потребуется всего две ячейки памяти. Выдача, например, может проводиться через каждые пять шагов. Вторым вариантом может сводиться к следующему: сначала по формуле для  $x_n$  рассчитывается пять последовательных величин  $x_k$ , а затем по второй формуле пять величин  $y_k$ . Ясно, что в этом случае, как минимум, придется под промежуточные величины отвести шесть ячеек памяти. Разумеется, приведенные нами примеры носят тривиальный характер, но они поясняют, что имеется в виду.

Здесь же решается вопрос общей структуры системы.

При подготовке эскизного проекта требуется провести модульный анализ алгоритма: выделить состав модулей, определив принцип выделения частей программы в модуль и указав необходимую дополнительную информацию по каждому модулю, точнее, по каждому типу модулей.

На этой же стадии решаются вопросы работы с памятью. Здесь могут быть приняты решения о «жестком» и «настраиваемом» распределении памяти. Разновидностью «жесткого» распределения памяти (т. е. когда данные привязываются к конкретным адресам оперативной памяти) может быть выделение этой привязки в специальный модуль «памяти». При разработке специальных «схем настройки» памяти должны быть изложены соответствующие алгоритмы.

В эскизном проекте приводится перечень модулей с указанием информационных связей и величин, которые ранее мы называли входными и выходными. Приводить

тексты модулей здесь не требуется, но надо определить их структуру (сообразно с требованием сборки и т. д.). Если для построения пакета применяются штатные средства автоматизации программирования, следует привести стратегию их использования. Если предполагается организовать сегментацию программы для экономии оперативной памяти, то описывается принцип построения сегментов и организация их загрузки. Описывается система тестовых задач, а также общая методика проведения различных видов отладки. Приводятся соображения по структуре входных и выходных данных и применяемые методы их контроля. Дается технико-экономическое обоснование выбранных путей реализации.

Эскизный проект не содержит детальной проработки всех компонент, а лишь намечает пути реализации. Однако от того, насколько удачно они выбраны, зависит успех всей дальнейшей разработки. Поэтому оформление эскизного проекта столь же серьезно, как и подготовка технического задания. Перед согласованием и утверждением проекта разрабатывается документ «пояснительная записка». В него включаются необходимые схемы, общее описание функций программы и методов ее реализации, технико-экономические обоснования принятых решений. Записка должна быть краткой, но исчерпывающей. Она как бы воедино связывает различные документы, соответствующие проработке эскизного проекта.

**7.3.6. Разработка технического проекта.** Следующая стадия носит название технический проект. На этой стадии принятые пути реализации превращаются в окончательные схемы с полной проработкой деталей. Завершается разработка алгоритма программы. В смысле их алгоритмического содержания окончательно определяются модули. Разрабатывается структура программы. Сюда входят различные схемы: схема счета, указывающая последовательность выполнения блоков, информационные схемы, схема распределения памяти, схема обмена с внешними устройствами, схема сегментации программы и управления работой сегментов, схема отладки.

При разработке технического проекта должны решаться некоторые вопросы технологического характера. В отдельных случаях надо предусмотреть возможность внесения изменений. Может быть указано правило построения обозначений, приводятся соглашения о рекомендуемых языковых конструкциях, об использовании регист-

ров, рабочих областей памяти, приводятся соображения по стилю написания программ, которые должны обеспечить наглядность программ и т. д.

При разработке технического проекта особое внимание следует обратить на выбор структуры данных и форматов их представления. В рассматриваемых нами задачах форматы входных и выходных данных играют заметную роль. Немаловажны они и для отладочных выдaч.

Дадим некоторые пояснения, относящиеся к содержанию этой части технического проекта. Как уже говорилось ранее, данные разбиваются на две большие категории, константы и переменные. Константы делятся на физические и математические. Физические константы могут быть постоянными для всего класса задач (например, скорость света, скорость звука и т. д.), постоянными для задачи или для варианта задачи. Заметим, что такое деление не абсолютно и определяется предметной областью (например скорость света в некоторых задачах может меняться даже в пределах одного варианта, т. е. вообще не быть постоянной). Математические константы также могут подразделяться на константы, связанные с математической моделью задачи и на константы, связанные с вычислительной моделью (например, сеточные константы и т. д.). Не менее разнообразны и переменные. Напомним, что все данные могут делиться на входные и выходные.

Проводить полную классификацию всех групп данных в общем случае нецелесообразно. Лучше проделать это для определенного класса задач, скажем, нейтронных, газодинамических, электродинамических и т. д. В рамках более узких предметных областей возможна и более конкретная классификация.

Заметим, что данные объединяются в известные смысловые группы или группы, определяемые по признаку использования. В связи с этим можно ввести понятие модулей данных, имеющих иерархическую, древовидную структуру организации.

Таким образом, данные образуют не безликий набор чисел, а некоторое дерево [11]. Иногда модули данных в работающем комплексе мы будем называть блоками данных.

В техническом проекте требуется детальное описание структуры данных и метода доступа к ним.

Форма представления выходных данных должна соответствовать структуре их организации. Если данные выдаются на устройстве типа АЦПУ, в техническом проекте



следует дать описание формата «итогового документа». В случае графической выдачи наряду с описанием формата выводимой информации указывается программное обеспечение, используемое для организации (например, комплекс ГРАФОР).

В техническом проекте формулируются требования на оборудование и программное обеспечение (на операционную систему, систему программирования, специальные средства автоматизации программирования, библиотеку стандартных программ и т. д.), наличие которых обуславливает успешную реализацию проекта.

Заканчивается работа над техническим проектом (как и над эскизным проектом) составлением краткой пояснительной записки, содержание которой аналогично записке для эскизного проекта. Она отражает результаты, полученные при выполнении технического проекта, давая ссылки на различные разработанные документы. В записке также указывается перечень компонентов технического проекта.

Технический проект проходит процедуру согласования и утверждения \*).

**7.3.7. Рабочий проект и внедрение.** Следующая стадия связана с программированием модулей и отладкой программы. Изготовление документации по отлаженной программе также включается в эту стадию.

Рабочий проект заканчивается проведением испытаний. Для этого разрабатывается, согласовывается и утверждается методика их проведения. После испытаний по их результатам осуществляется корректировка программы и документации.

Последняя стадия связана с внедрением программы. На этой стадии осуществляется подготовка программ и программной документации для передачи на сопровождение и изготовление. Оформляется и утверждается акт о передаче.

Изложенная нами система документации по разработке вычислительного эксперимента мало чем отличается от распространенной, если не считать некоторой перегруппировки материала, проведенной в интересах создания единой методической основы для всех документов.

---

\*) В технический проект включается план мероприятий по внедрению программ.

Мы не остановились на вопросах оформления документации, поскольку они достаточно детально изложены в соответствующих ГОСТах ЕСПД. В отдельных случаях полный набор документов может оказаться излишним. Вместе с тем надо учесть, что подготовка документации в качестве важного побочного продукта дает более глубокое понимание сложных вопросов самой разработки. Она способствует и более четкой организации работ при создании программного комплекса и его эксплуатации даже тогда, когда этим занимается один и тот же человек.

## ГЛАВА 5

### ПАКЕТЫ ПРИКЛАДНЫХ ПРОГРАММ

Пакеты прикладных программ — интенсивно развивающееся направление автоматизации программирования. На него возлагают большие надежды, оно накладывает отпечаток на развитие общего программного обеспечения и аппаратуры [5].

Из всех определений пакета наиболее целесообразным является, видимо, то, которое указывает цель пакета. При таком определении все, что входит в средства достижения цели, и будет пакетом. Разумеется, средства должны носить специфический характер [119]. В противном случае операционную систему также можно отнести к пакету. В [8] приведено следующее определение: «Под пакетом прикладных программ мы понимаем комплекс взаимосвязанных прикладных и системных программ, обеспечивающих адекватное покрытие некоторой прикладной деятельности».

Идея разбиения программ на отдельные части (блоки), как известно, не нова. Одна из целей, которые преследуются при таком разбиении — сохранить возможность легкой замены отдельных блоков. В ранних разработках это удавалось в случаях, если блоки заменяемый и заменяющий вычисляли одни и те же величины и располагались в тех же ячейках памяти, что и старые. Задачи, которые при этом ставились, носили вполне современный характер: обеспечить накопление сменных блоков, позволяющих обслуживать задачи, несколько отличающиеся по физической постановке. На современном этапе постановка той же проблемы приобрела новую качественную окраску.

1. Существенно расширился и усложнился класс задач. Число накапливаемых блоков сильно возросло. Появилась проблема «независимости» разработки блоков.

2. Задачи вычислительного эксперимента требуют чтобы расчеты шли с некоторым опережением физического эксперимента и отражали эволюцию физического эксперимента [5].

3. Пакет должен обладать свойством «непрерывной зависимости» от модели: малые изменения физической модели должны приводить к малым трудозатратам на построение соответствующего программного комплекса.

4. Принцип накопления дополняется принципом эффективности пакета: при заданном числе модулей должно быть обеспечено как можно большее число задач, которые можно построить из них.

5. Должна достигаться высокая эффективность использования ресурсов вычислительной машины.

Более развитая система требований имеется в [71].

Пакет можно рассматривать состоящим из трех компонент: функциональное наполнение пакета, отражающее конкретный класс задач, и системное наполнение пакета, отражающее уровень автоматизации работы с пакетом. Третья компонента — язык для работы с пакетом.

Разработка пакета связана с решением ряда сложных проблем. Вот некоторые из них: выбор модулей, определение структуры модулей, технология сборки программы из модулей, интерфейсы модулей, язык пакета.

## **§ 1. Принципы организации пакетов прикладных программ**

1. Построение наборов модулей. Начнем с уточнения понятия модуля. Во всем предыдущем изложении мы понимали под модулем некоторую часть физической модели, математической вычислительной модели, алгоритма, программы и т. д. Выделение этой части, ее структура должны, по нашему мнению, определяться в каждом конкретном случае теми задачами, ради решения которых это выделение осуществляется [8].

Остановимся на принципах выделения модулей. Известно, что алгоритм, который реализуется в программном комплексе, имеет иерархическую структуру. Весь алгоритм можно рассматривать как модуль некоторого уровня. Он состоит из модулей следующего более низкого уровня, а каждый из модулей этого уровня также может состоять из более мелких модулей и т. д. Эту зависимость можно изобразить в виде некоторого дерева. Построение

такого дерева и называется модульным анализом. Именно на этом этапе определяются «размеры» модулей и уровень их расположения в иерархии. Примеры такого анализа можно найти в [126]. Существо этих примеров состоит в том, что выделение модулей основывается на функциях, которые они выполняют. Таким образом, как правило, модуль — функционально законченная единица.

Однако это только один источник модульной структуры. Как мы уже говорили, надо стремиться как можно меньшим числом модулей «покрыть» как можно большее число задач. В связи с этим в качестве модуля может быть выделена часть алгоритма, не имеющая законченного смысла. В качестве примера можно привести общие блоки в текстах программ на фортране [9]. При таком подходе может оказаться, что задачи, совершенно не похожие друг на друга по физической постановке, могут состоять из одинаковых наборов модулей.

Следующий источник — машина. Зачастую модуль выбирается по соображениям объема используемой памяти при сегментации программы. В [5] вводится понятие простого, автономного и не автономного модулей. Простой модуль можно разместить в оперативной памяти машины вместе с требуемой для него информацией. Если это не удастся, то можно разбить модуль на части, каждая из которых не имеет законченного смысла. Такие части называются неавтономными модулями. Правда, для этого приходится видоизменять алгоритм и вводить дополнительные модули «стыковки».

Часто модуль выбирают, исходя из соображений удобства. Например, в [11] выбирается модуль величиной не более листа оперативной памяти, т. е. 1024 ячеек.

При выборе модулей стремятся к максимально возможному сокращению объема входных и выходных данных для упрощения связей между модулями. Этой же цели служит стремление выбрать модули с таким расчетом, чтобы по возможности сократить число переключений с одного модуля на другой. В предельном случае обращение к каждому модулю происходит только один раз за время выполнения программы. Учет этих соображений особенно важен при недостаточно эффективно организованном интерфейсе между модулями.

Относительно размеров модулей добавим следующее соображение. Слишком маленькие модули неудобны, они сводят на нет преимущества пакета. Действительно, наи-

меньший размер модулей можно получить, если спуститься до уровня основных операторов языка, в результате чего пакет исчезнет — выродится в набор операторов.

При выделении модулей учитываются также соображения удобства деления разработки на части для организации одновременного выполнения ее группой специалистов. Этот момент весьма важен, так как независимая разработка модулей пакета позволяет объединить в пакете труд многих специалистов. Коллективный характер разработки решает также важный вопрос накопления фонда алгоритмов.

Таким образом, модульная структура пакета определяется модульной структурой физических моделей, математических и дискретных моделей, программной реализацией вычислительного эксперимента на машине и организацией вычислительных работ [2, 8, 50].

**2. Типы модулей и их структура.** Что же представляют собой модули? Вначале предполагалось, что модули обязательно должны представлять собой функционально законченную единицу. В настоящее время эта точка зрения претерпела изменения, и в качестве модуля могут рассматриваться некоторые части программного комплекса, выделение которых целесообразно по тем или иным причинам.

В качестве модулей могут рассматриваться отдельные задачи в рамках некоторой операционной системы [127]. Это могут быть некоторые программные единицы [56], например, подпрограмма, подпрограмма-функция и головная программа в фортране, процедура в алголе [128]. В качестве модулей могут быть просто фрагменты текста, которые затем с помощью средств типа редакторов или генераторов компоуются в программный комплекс [8]. Редакторы собирают пакет из частей программы, не внося в них изменений. В случае макрогенерации допускается некоторая настройка по дополнительной информации этих частей перед компоновкой, что может существенно уменьшить общее число модулей, необходимых для сборки программного комплекса.

Модули могут задаваться в текстовом виде, на некотором промежуточном языке загрузки, в машинных командах. Соответственно различают модули исходные, объектные и абсолютные. Исходные модули — это модули на исходном языке пакета. Объектные модули — это модули, полученные в результате трансляции, однако не обрабо-

танные еще редактором внешних связей и загрузчиком [27]. Иногда объектные модули называют модулями загрузки. Абсолютные модули — это модули в истинных адресах оперативной памяти, записанные в машинных командах [57].

Если из некоторого набора модулей можно построить программу для любой задачи данного класса, то говорят о полном наборе модулей. В случае, когда различные модули выполняют одну и ту же функцию, то говорят о функциональной избыточности [123].

Модули самого нижнего уровня иногда называют базисными модулями, или базисными элементами. В последнем случае обычно подразумевается неперекрывающийся набор модулей.

Отметим, что текст модулей может быть оформлен на некотором специальном проблемном языке или в виде набора заданий для системной части пакета. Очень часто вводится различие по функциям: вычислительные, управляющие, системные, описывающие схему счета и т. д. Вычислительные, как показывает само название, осуществляют расчеты, управляющие выполняют функции управления вычислительным процессом в готовом программном комплексе, в других случаях управляющие модули осуществляют выбор направлений расчета, т. е. выбор модулей, когда для выполнения одной функции допускается несколько модулей. Аналогично можно охарактеризовать и другие группы. Часто вводится стандартизация в наименования модулей для того, чтобы можно было по самому наименованию установить выполняемую функцию [9].

Современная тенденция может быть охарактеризована как стремление включать в пакет модули, разнообразные как по функциям, так и по форме [129].

Подводя итог, можно сказать, что задача модульного анализа некоторого класса задач сводится к выбору множества модулей с такими пожеланиями:

- максимально большой размер модулей при минимально возможном числе модулей;
- максимально большое число программ для задач данного класса, которые из них можно собрать;
- максимальное удобство сборки комплекса.

Ценность каждого модуля определяется тем, насколько часто он встречается в задачах данного класса. Для этой цели можно ввести числовую характеристику [5].

Совокупность всех модулей пакета входит в тело пакета. Помимо модулей тело пакета может включать статистическую информацию об использовании различных компонент пакета, каталоги, справочные данные и др. Пакет может использовать в виде некоторой подсистемы архив для хранения тела пакета и другой информации. Различные компоненты пакета могут храниться на исходном языке, языке загрузки, в машинных командах, на языке пакета и на языке управления заданиями.

Перейдем теперь к изложению структуры модуля. Модуль может состоять из тела модуля и паспорта модуля. Тело модуля — это его содержательная часть. В паспорте могут помещаться сведения, нужные для последующей сборки. К их числу относятся входные и выходные величины, точки входа в модуль, точки появления резуль-татных величин, размеры модуля и др. [130]. В некоторых случаях паспорт модуля не составляется, а необходимая информация извлекается непосредственно из самого модуля, как это делается в модулях на фортране, описанных выше.

**3. Вопросы сборки модулей.** Дальнейшая задача — это собрать с помощью системных средств пакета нужную программу и запустить ее на счет. Сборка может проводиться в автоматическом режиме или по указанию пользователя. В последнем случае используется язык управления пакетом. При автоматической сборке схему счета задачи строит «планировщик» пакета, к которому мы вернемся несколько позже.

При сборке надо выполнить стыковку модулей по управлению и информации. Обычная структура готового комплекса выглядит так: в оперативной памяти постоянно присутствует управляющая программа, которая организует поочередный вызов и загрузку сегментов программы в память. Сегменты сами организуют работу с внешней памятью. Возможны и другие варианты.

При стыковке по управлению надо обеспечить управляющую программу сведениями о порядке включения в работу модулей собираемой программы. Кроме того, каждый модуль должен иметь возможность получать данные от другого модуля, считывая их с определенных областей оперативной памяти и помещая в соответствующие области свои результаты для другого блока. В этом и состоит установление информационных связей. Рассмотрим различные варианты сборки, следуя [14].



**3.1. Библиотечный способ сборки.** Простейший способ сборки основан на использовании библиотек и систем программирования. Например, можно использовать мониторинговую систему Дубна и язык фортран.

В библиотеке мониторинговой системы хранятся модули загрузки. Они транслируются независимо друг от друга и никак не связаны (ни по управлению, ни по информации). Пользователь сам в тексте своей программы организует эти связи указанием обращений к подпрограммам (связи по управлению) и за счет указания фактических параметров в обращениях и переменных в общих блоках (связи по информации). В этом случае пользователь целиком берет на себя рутинную работу по сборке: он должен везде указать фактические параметры, внимательно распределить память в общих блоках.

Чтобы понять неудобства использования библиотек и систем программирования, перечислим требования, которые нам нужно предъявлять к модулям и средствам сборки [145].

1. **Независимость разработки**, т. е. возможность вести разработку модулей без взаимного общения разработчиков.

2. **Комбинируемость**, т. е. возможность получения нового модуля при комбинировании имеющихся модулей средствами сборки. Это — важное свойство, так как возможность построения модулей из совокупности других модулей значительно сокращает сборку и упрощает получение новых модулей.

3. **Контекстная независимость**, т. е. возможность проводить сборку программы, не зная, как работает каждый модуль. В большой системе всегда происходят изменения, и если при каждой сборке надо выяснять, какие были внесены изменения, то ясно, что эффективность такой системы невелика.

4. **Информационная независимость.** Если происходят изменения в некоторой группе данных, которые в модуле не используются, то, естественно, это не должно приводить к изменению такого модуля [128].

Для фортрана эти требования оказываются невыполненными. Комбинируемость не выполняется, поскольку программа, полученная комбинированием головной программы и подпрограммы, не будет подпрограммой. Информационная связь модулей по формальным параметрам трудоемка при большом их количестве, так как их при-

ходится задавать при каждом обращении, хотя, по существу, они являются общими для всей программы и их указание связано только с независимостью трансляции подпрограмм. Следующая трудность состоит в сложности согласования параметров фактических и формальных по типу, количеству и т. д. Например, при изменениях в одной подпрограмме придется просматривать все остальные модули, где эти изменения могли сказаться.

Можно привести много примеров, когда внесение изменения в один модуль может привести к ошибке. Общие блоки также вызывают трудности, во-первых, из-за невозможности употреблять одинаковые имена общих блоков, во-вторых, из-за изменений в информации, не касающейся данной подпрограммы. Пусть мы, например, увеличили размер массива, который данной подпрограммой не используется, а только передается в другую подпрограмму. Если не учесть это обстоятельство, то возникнет ошибка.

Надо сказать, что аналогичного типа ситуации возникают и при использовании алгола.

Таким образом, организация информационных связей при сборке с использованием разобранных средств — достаточно трудоемкая процедура.

**3.2. Сборка с изначальным заданием связей.** Другая система сборки основана на жестком задании связей как по памяти, так и по управлению, причем предприняты специальные меры для облегчения замены модулей. Такие системы эффективны в случае, когда схема счета задач остается практически неизменной. Меняется только модуль, а функции его остаются неизменными. С таким обстоятельством мы сталкиваемся, когда заменяем модуль на другой, в котором счет идет по иной методике. Могут быть модули одинаковые по входным и выходным данным, но отличающиеся по физической постановке реализуемой ими задачи (например, можно взять другое уравнение состояния вещества).

Применение редактора внешних связей и загрузчика значительно расширяет возможности такой системы сборки. В этом случае, например, при написании программ на автокоде [57] все переменные, связанные с получением или передачей данных, оформляются как внешние переменные. Теперь остается включить в пакет один специализированный модуль «память», который в процессе счета не работает; единственная его функция — в момент

обработки программ с помощью редактора связей осуществлять передачу сведений в модули через «внешние метки». Таким образом, все информационные связи модулей при сборке концентрируются в одном месте — модуле памяти. При желании изменить распределение памяти нет необходимости менять модули, достаточно лишь поменять модуль памяти. Саму «загрузку» также можно проводить с помощью некоторой вспомогательной программы, в которой указывается последовательность загрузки блоков и место их загрузки, возможно, с учетом сегментации программы. Выбор модулей загрузки может проводиться с помощью выбора параметров, причем вовсе нет необходимости собирать все имеющиеся программы и строить ветвящуюся программу, конкретный путь по которой выбирается согласно заданным параметрам [11].

Неудобство таких методов сборки состоит в том, что заранее должны быть запрограммированы все связи, предусмотрены все возможные варианты программ, настройка на которые проводится с помощью параметров. Все же эта система сборки отличается простотой и большой эффективностью готовых программ, особенно если сборка запрограммирована на автокоде, и позволяет использовать все возможности машины.

**3.3. Сборка с изначальным заданием только информационных связей.** Следующий вариант сборки предусматривает только задание информационных связей между модулями. Задавая цепочку входных и выходных данных, в конце которой получаются нужные нам величины, мы с помощью «планировщика» сможем построить последовательность модулей, реализующих эту цепочку, т. е. получить схему счета задачи. Здесь возможны различные варианты постановки задачи сборки. Можно, например, задавать те величины, которые должны быть найдены. В таком случае решение может получиться не единственным. Пусть нам надо найти энергию, потребляемую активной электрической цепью с некоторым сопротивлением. Эту задачу можно решить с помощью двух модулей: на вход одного! подается напряжение и сопротивление, на вход другого подается ток и сопротивление. Соответствующая схема представляется в виде направленного информационного графа. Здесь решение не будет единственным (рис. 5.1).

Если же задавать и входные величины, то вычисляющий модуль определяется однозначно. В общем случае

строится полный информационный граф для всех модулей, и по нему отыскивается нужный нам подграф, реализующий задачу. Фактическое задание информационного графа сводится к заданию списка, элементами которого служат записи, включающие наименование модуля, перечень входных и выходных величин. Связь по информации устанавливается путем разработки системы стандартных величин.

В таком случае, например, величина  $X$  является для одного модуля входной, а для другого — выходной величиной, что и задает необходимую связь. Если под стандартные ве-

личины распределить память заранее, то при известной последовательности выполнения блоков заботиться о согласовании по информации не нужно.

В других случаях можно не распределять память под входные и выходные модули и не собирать программу до счета, а выполнять сборку и счет одновременно, память же распределять под стандартные величины динамически. Для этого потребуются специальные системные средства.

Подобные системы сборки применяются тогда, когда количество модулей велико. Они находят применение при проектировании, в конструкторских расчетах, а также в ряде задач математической физики и инженерных расчетах [10, 124].

**3.4. Сборка путем установления связей с помощью языка.** Еще один вид сборки похож на вариант библиотек. Здесь также изначально модули никак не связаны, но могут быть связаны с помощью некоторого специального языка. Имеется два варианта. Можно на этом языке задавать информационные связи, а связи по управлению будут строиться автоматически. Можно задавать связи по управлению, а информационные связи будут строиться автоматически [14].

**3.5. Обсуждение вариантов сборки.** Прокомментируем приведенную классификацию систем сборки программ. Мы рассмотрели случай, когда задаются информационные связи между модулями и ищутся управляющие связи, т. е., по существу, строится последова-

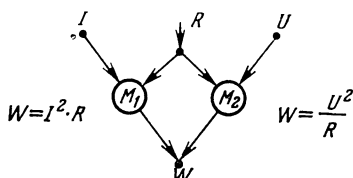


Рис. 5.1.

тельность модулей, которые должны выполняться при счете программы. Надо отметить, что с точки зрения пользователя это наиболее естественная постановка: задаются величины, которые нужно найти. Дальнейшая задача сводится к нахождению последовательности операций для определения этих величин [131]. Отметим, что автоматический способ построения цепочки модулей, очень заманчивый по своей сути, наталкивается в задачах математической физики на большие трудности. Это связано с тем, что процесс решения задач трудно формализовать. В [133] приводятся следующие соображения на этот счет:

— выбор математической модели зависит не только от задачи, но и от величины, которую надо рассчитать, и от требуемой точности;

— численные методы опираются на особенности решения, на модель физических процессов, которые имеют свои области применения, не всегда четко определенные;

— плохо известны критерии правильности решения.

Итак, во втором варианте сборки мы изначально задаем информационные связи, а связи по управлению устанавливаем. Нельзя ли поступить наоборот? Покажем, что с точки зрения пользователя это неестественный вариант.

Разберем нарочито примитивный пример. Пусть у нас имеется неограниченная оперативная память, в которой хранятся модули и наборы данных, являющихся входными и выходными для модулей. Будем для простоты считать, что модули написаны в машинных командах. Наборы входных и выходных данных обозначим через  $X_i$ . Представим теперь информационный граф в виде колонок. Это и будут информационные связи. В первой вертикальной колонке мы записываем наборы входных данных для модуля, во второй — номера модулей, в третьей — номера выходных наборов данных:

$X_1X_2$	1	$X_3X_4$
$X_3$	2	$X_5$
$X_3X_4$	3	$X_6X_7$
$X_1X_5$	4	$X_8$
$X_3X_4$	5	$X_9$
$X_6X_2$	6	$X_{10}$
$X_7X_1$	7	$X_{11}$
$X_4X_7$	8	$X_6$
	$\vdots$	

Пусть теперь нам надо установить управляющие связи для задачи, которая вычисляет набор данных  $X_{10}$ . Находя в правой колонке этот набор, устанавливаем номер модуля, который должен проработать последним. В левой колонке находим входной набор. Отыскивая очередные модули, которые могут вычислить этот набор данных, мы получим следующее звено цепочки и т. д. Единственная трудность, которая здесь может встретиться, — это наличие нескольких модулей, вычисляющих данный набор. В таком случае помимо информационного графа для выбора модуля нужна дополнительная информация. Обычно такая информация содержится в исходных данных на сборку программы. В приведенном ранее примере такой информацией, например, могло быть требование в качестве входных данных для всей задачи иметь набор  $I, R$ . В других случаях это может быть требование на точность или указание, что решение должно проводиться в  $R, Z$ -геометрии и др.

Теперь определим, что значит изначально установить информационные связи между модулями в пакете? Для простоты будем понимать это как жесткое закрепление за наборами данных определенных областей памяти. Адреса этих областей заданы в текстах модулей и, следовательно, если модулю передано управление, то он готов к счету: берет данные из одних заданных областей памяти и помещает результаты в другие, также заранее определенные области. Это жесткий интерфейс, причем даже форма его излишне жесткая. В дальнейшем мы еще вернемся к этому вопросу. Заметим, что если наборы данных стандартизированы для всего пакета, то, имея перед глазами таблицу распределения памяти для этих стандартных величин, можно независимо создавать модули, обрабатывающие стандартные величины, входящие в эти наборы данных. При появлении очередной стандартной величины, под нее отводится соответствующая запись в этой таблице. Тогда для задания управляющих связей можно отвести всего одну ячейку в каждом модуле, где будет размещаться команда передачи управления на другой модуль. Построение цепочки будет сводиться к занесению в эту ячейку нужной команды управления. Если функционирование модуля заканчивается на этой ячейке, то нужная цепочка оказывается сформированной. Еще раз отметим, что предложенная схема в таком виде никогда не реализуется из-за ее неэкономности. Далее мы рассмотрим

реальные схемы. Здесь же нас интересует только выяснение существа дела.

Обратим внимание на то обстоятельство, что при построении цепочки рассмотренным способом не приходится ничего делать с информационными связями. Они уже сформированы. Мы заняты только выбором последовательности модулей. Иное дело, если пытаться построить граф управления в виде таблицы. Такая таблица состоит из двух столбцов. В левом указывается модуль, а в правом — модули, которые могут выполняться после каждого из них.

M1	M8,2,10	M3	M4		
M2	M7,11	M4	M12		
M3	M2	M5	M6	M7	
M4	M5,6	M8	M3	M10	M9
M5	M3	M6	M9	M4	

Уже из этой таблицы видна разница между предыдущим и рассматриваемым вариантами установления связей. В предыдущем варианте каждому набору данных соответствовал, как правило, один вычисляющий модуль. Иные случаи составляли исключение. Однако главное отличие состоит в том, что при таком варианте установления связей в процессе построения цепочки придется заботиться не только об информационных связях, но, практически, вновь устанавливая управляющие связи. Возвращаясь к нашему примеру, можно сказать, что нам придется в каждом модуле указать конкретные адреса наборов данных, устанавливая информационную связь. Однако помимо этого мы должны еще сформировать и ту ячейку, где должна помещаться команда передачи управления на другой модуль, так как, например, M1 может передавать управление трем модулям.

Возможна и другая система организации перехода от одного модуля к другому, однако во всех случаях мы должны заботиться о реализации передачи управления на конкретный модуль. Обратим внимание на то обстоятельство, что мы допустили ветвление, например, запись M3, 7, 4 означает, что в зависимости от результатов счета переход может происходить на 3-й, 4-й или 7-й модули. Очевидно, что выбор этого пути никак не связан с построением цепочки.

Правда, есть один случай, когда изначальное задание управляющих связей не потребует при построении це-

почки заниматься дополнительной обработкой по управляющим связям. Этот случай будет иметь место, если каждому модулю «разрешен» переход только на один модуль, но в таком случае цепочка уже построена.

Вместе с тем задавать управляющие связи оказывается целесообразным при использовании схем счета [9, 73, 134, 135]. Случай схем счета как пакетного средства сборки программы имеет свои особенности. Поясним их кратко. Вводится понятие функциональных модулей. Суть этого понятия в том, что под функциональным модулем подразумевается целая группа модулей. Схема счета строится в виде цепочки для функциональных модулей. Для того чтобы построить конкретную программу, нужно установить соответствие между именами функциональных модулей и именами конкретных модулей данной функциональной группы. Каждый конкретный модуль может в значительной мере отличаться от модуля той же функциональной группы, в частности, по информационным связям. Для стыковки модулей в этом плане существует специальный системный аппарат, о котором мы вскоре расскажем.

Таким образом, удобство такой сборки программы состоит в том, что не надо организовывать управляющие связи по всей задаче в целом, они уже организованы: требуется только установить соответствие функционального имени конкретному.

Из разбора последнего способа сборки программы видно, что приведенные варианты не исчерпывают всех возможностей организации сборки, существует много промежуточных вариантов.

Могло создаться впечатление, что задание соответствия имен не менее трудоемко, чем задание управляющих связей, и тогда выбор предварительных управляющих связей для функциональных модулей теряет смысл. На самом деле это не так. Ведь задание управляющих связей требует определить местоположение каждого модуля в программном комплексе. При этом нужно учесть, что каждый модуль может встречаться несколько раз. Соответствие имен модулей не требует знания местоположения модуля в программном комплексе, а также местоположения и даже функций остальных модулей, из которых составляется комплекс. При длительной работе часть модулей может «осесть» в схеме счета постоянно, так что соответствие имен будет касаться только тех



компонент программного комплекса, которые изменяются.

Обычная схема организации программного комплекса может быть охарактеризована следующим образом: имеется управляющая программа, организующая весь счет, и вычислительные программы. В некоторых случаях добавляется группа системных блоков, организующая обмены, интерфейсы и т. д. В таком случае схему счета можно рассматривать как управляющий блок, настройка которого на конкретную задачу происходит в процессе сборки программы с помощью специальных системных средств.

Может возникнуть вопрос, насколько широка в смысле класса задач та или иная схема счета? На самом деле, с помощью схем счета можно «покрыть» довольно широкую предметную область, и при создании специализированных пакетов, для научно-исследовательских целей и инженерных разработок сложных систем, схемы счета могут оказаться весьма эффективным инструментом.

Вернемся к нашей примитивной схеме. Пусть теперь не заданы изначально ни информационные, ни управляющие связи, т. е. вместо реальных адресов в модулях имеются условные адреса входных и выходных данных, а ячейка, где должна располагаться команда передачи управления, пустует. Здесь возможно два варианта. С помощью некоторых средств, или просто вручную, мы задаем управляющие связи. Когда известна последовательность модулей, можно установить информационные связи. Однако мы уже показали на примере фортрана и мониторной системы Дубна, что это сложная процедура, приводящая к большому количеству ошибок. Последнее обстоятельство требует создания некоторых системных средств для осуществления информационных связей в автоматическом режиме с использованием дополнительной информации, которая может в виде специальных записей храниться при модуле. Эта информация должна содержать сведения о входных и выходных данных, а также другие сведения, которые составляют «паспорт» модуля [119, 130]. Они появляются при написании модуля. Этот способ сборки программы требует создания специальных языковых средств, и мы определили его как четвертый вариант сборки.

Если с помощью специального языка задавать информационные связи, а затем в автоматическом режиме определять управляющие связи, то можно получить еще одну

разновидность сборки программ. Такого рода системы находят применение при описании технологических процессов и при моделировании [14].

Перспективы расширения круга задач с двумя последними вариантами сборки программы связаны с тем, что для сборки фактически достаточно задать либо управляющие связи, либо информационные. Одни, по существу, определяют другие. Совместное задание связей обоих типов связано с недостаточностью системного обеспечения. А ведь именно при одновременном задании двух типов связей возникает наибольшее число ошибок!

Мы изложили принципы сборки программы. На практике при разработке модульных систем не придерживаются какой-либо одной схемы. Действительно, в случае задания информационных связей изначально, т. е. до сборки, могут существовать модули, которые нецелесообразно связывать по информации, например модули, выполняющие универсальные функции. К таким функциям может относиться интегрирование, суммирование, обработка полученной информации для выдачи и т. д. В других примерах даются схемы счета, где связи управляющего характера частично определены, или же, наоборот, пользователь указывает лишь группы модулей, а выбор связей внутри группы происходит с помощью системы [30].

**4. Реализация информационных связей.** Следующий важнейший вопрос построения пакетов — информационные связи модулей или интерфейсы по данным. Разбирая этот вопрос, мы оставим в стороне способ организации интерфейсов («ручной» или автоматический). Под термином «ручной» мы, как и везде, понимаем способ организации интерфейсов непосредственно пользователем, а не программными средствами.

**4.1. Общие вопросы.** При обсуждении сборки пакетов мы говорили о том, что информационные связи могут устанавливаться изначально, т. е. до сборки программы. Может создаться впечатление, что в этих случаях вопрос организации интерфейсов не играет большой роли. Однако одно дело — указать связи, другое дело — их реализовать. И хотя эта работа может выполняться при изготовлении пакета, до сборки, тем не менее она сильно влияет на эффективность пакета в процессе эксплуатации. Во-первых, при введении каждого очередного модуля в систему (система должна быть открытого типа) приходится заботиться о реализации интерфейсов.

Во-вторых, пакет — это не мертвая конструкция, она непрерывно развивается, и при неудачно организованных интерфейсах эксплуатация такого пакета может стать очень трудоемкой.

Приведенный нами ранее пример организации интерфейсов с помощью системы программирования — фортран в мониторной системе Дубна — показал, что для организации интерфейсов нужны более совершенные системные средства, позволяющие удовлетворить таким важным требованиям, как независимость разработки, контекстная независимость и независимость информационная. Все эти требования сводятся к одному — обеспечить возможность проведения сборки без изучения текста модуля. При внесении изменений в модули или данные должна быть уверенность, что эти изменения коснутся только тех модулей или данных, к которым они непосредственно относятся, не затрагивая остальные компоненты пакета (их менять не следует). Так, например, если мы, в какой-либо набор данных добавили несколько новых элементов, это не должно отражаться на организации работы с другими данными, т. е. не должно возникать необходимости что-то исправлять в других модулях или наборах данных.

Шагом в этом направлении является исключение интерфейсов из тела исходного модуля. Мы приводили пример использования системы [57], позволяющей сосредоточить все вопросы распределения памяти в одном специальном «модуле памяти» [11]. Таким же свойством обладает аппарат формальных параметров в фортране. Второй шаг — включение данных, содержащих сведения об интерфейсах, в паспорт модуля [130], который может располагаться либо в самом модуле, либо отдельно. Паспорт обычно содержит сведения о входных и выходных данных, т. е. список наборов данных с указанием имени (номера) списка, длин массивов, мест входа в модуль и выхода из него и т. д.

В некоторых случаях для облегчения составления паспорта прибегают к помощи специальной системной программы [32], которая из обычного модуля может удалить интерфейсы и организовать записи, эквивалентные паспорту. Такой подход особенно целесообразен при использовании старых программ, которые тем не менее нужно включить в пакет.

**4.2. Нижний уровень реализации интерфейсов.** Рассмотрим реализацию интерфей-

сов на нижнем уровне. Пусть память уже распределена вручную или автоматически, т. е. для каждого элемента данных определено абсолютное положение в оперативной или внешней памяти. Тогда возможны три способа осуществления информационных связей.

Первый способ состоит в засылке конкретных адресов в тело модуля вместо некоторых условных. Именно так организованы связи по формальным параметрам в фортране, т. е. под эти величины «внутри» модуля не отведено памяти.

В других случаях для модуля отводится специальная область памяти, жестко закрепленная за ним. Информационные связи осуществляются пересылкой результатов работы одного модуля в соответствующие участки памяти другого модуля.

Третий способ реализации интерфейсов сводится к использованию общих блоков памяти, он не требует выполнения каких-либо действий в момент установления связей, как это имеет место в двух предыдущих. Таким образом, в первом случае память «не принадлежит» модулю, который использует данные, расположенные в этой области памяти, во втором — принадлежит модулю, которому передаются данные, в третьем — память общая для этих модулей.

Эти три случая представляют собой жесткие интерфейсы, поскольку под каждый элемент данных отводится определенное место в памяти, известное заранее до счета задачи. В случае гибкого интерфейса строится каталог и по нему определяется положение элемента данных в памяти. Гибкие интерфейсы позволяют устанавливать истинные адреса элементов данных в процессе решения задачи, иначе говоря, динамически.

Наиболее экономичными в смысле использования машинного времени являются общие блоки памяти, а также связи по типу формальных параметров. Менее экономичны пересылки и динамические интерфейсы.

Организация интерфейсов непосредственно связана с типами модулей. Модули могут быть самостоятельными программными единицами, которые (как это характерно для фортрана) делят ресурсы памяти между собой. В других случаях — это программы, монопольно использующие ресурсы (например, оперативной памяти [10]), или просто отдельные программы операционной системы [127]. В последних двух случаях модуль обычно выступает как

функционально законченная единица. Стыковка может происходить на уровне исходных текстов или модулей загрузки, или даже на уровне готовых программ. (Очень часто стоит задача использования программ, созданных ранее, которые целесообразно включить в пакет.)

**4.3. Верхний уровень организации интерфейсов.** Для работы с данными пакета обычно строятся банки данных. Помимо собственно данных они включают систему управления данными. Сами же данные образуют иерархическую структуру. Для них также вводится понятие наборов данных, блоков данных и даже модулей данных. Наборы данных могут включать в себя, например, геометрические параметры реактора в случае реакторных задач, групповые потоки нейтронов, макросечения и т. д.

Для работы с данными вводятся специальные средства, например, языки, позволяющие строить различные структуры данных из существующих, заводить новые наборы данных и прочее. Язык работы с данными часто имеет характер управляющего языка и может содержать операторы типа: данные набора с именем ..., ввести на рабочие области памяти номер... Иногда для работы с данными строится некоторая административная система, которая может быть вызвана с помощью некоторых специальных подпрограмм, например, из фортрана [132]. В этом случае вместо компилятора с языка разрабатывается набор стандартных подпрограмм для работы с данными.

Рассмотрим достаточно распространенный вариант записи аргументов в память модуля. В этом случае при вызове модуля происходит загрузка данных на заданные участки памяти. Этой работой ведает системный модуль, который в разных случаях называется по-разному. Он отыскивает данные в банке данных, иногда может провести некоторую обработку наборов данных, а затем переписывает их на соответствующие участки памяти модуля. Например, преобразование может состоять в том, что из заданного набора данных выбираются четные по порядку элементы, и из них составляется новый набор.

Изложим конкретный вариант такого установления информационных связей, следуя в основном [127]. Пусть модуль представляет собой программу, которая целиком «распоряжается» выделенными ей ресурсами. Для каждого модуля составляется паспорт, содержащий описание входов и выходов, аргументов и результатов работы модуля,

полных имен этих аргументов и результатов (включающих имя модуля и внутреннее имя набора данных в модуле), а также размеры памяти, необходимой для размещения набора данных. При обработке модуля специальной программой, называемой преобразователем, в те точки, в которых вырабатывается нужный набор данных, вставляется передача управления на системный модуль, осуществляющий передачу данных. Когда модуль вызван и находится в работе после выработки соответствующего набора данных, происходит передача управления на модуль передачи данных. Одновременно передается информация о наборе данных. Модуль передачи данных записывает набор в банк данных.

Перед сборкой программы составляется управляющая программа с использованием фортрана и специальных псевдооператоров. В этой программе пользователь указывает все управляющие связи, а также описывает все интерфейсы. Управляющая программа обрабатывается системной программой, называемой анализатором. На втором, рабочем этапе, управляющая программа работает совместно с системными модулями передачи управления и передачи данных и организует всю работу комплекса. При вызове очередного модуля определяются наборы данных, необходимые для его работы, и с помощью модуля передачи данных происходит загрузка их в память модуля. Таким образом, в системе пользователь сам организует процессы передачи данных в области памяти модуля из банка данных и из этой области в банк данных, используя соответствующие программные средства. Системные средства предоставляют пользователю в одних случаях возможность передавать данные через оперативную память, в других случаях передача идет с использованием внешней рабочей памяти — пула данных.

Обсуждение вопроса организации интерфейсов закончим беглым обзором разных вариантов. Подробно с этими вариантами можно ознакомиться по цитированной литературе и [145].

Как мы уже говорили, для работы с данными могут использоваться специальные языки. Они позволяют менять структуру данных, настраивать модуль на данную структуру, выполнять некоторые управляющие операции. Возможна, например, такая последовательность: преобразовать некоторым образом набор данных (следует список имен наборов данных); загрузить полученный

набор в участок памяти (следует соответствие номеров участков памяти и номеров наборов данных); настроить модуль на заданную конфигурацию данных (следует имя модуля) [132].

Если работа пользователя с пакетом идет в рамках некоторого языка, например, фортрана, то работа с данными организуется с помощью специальных подпрограмм.

Работа с наборами данных дает большие преимущества при стандартизации обозначений. Это позволяет обеспечить, с одной стороны, независимость разработки модулей, с другой стороны, — удобство сопряжения модулей по данным, так как с набором данных можно работать как с отдельной единицей информации.

Сами интерфейсы отличаются разнообразием и представляют собой набор штатных средств программного обеспечения и специальных программных средств по связи модулей, настройке их на конкретные наборы данных.

В системах, где модули совместно используют ресурсы оперативной памяти, применяются общие блоки памяти в сочетании с выделением отдельных областей памяти под модули (области снабжаются номерами). При организации интерфейсов настройка модулей выполняется так, как это делается для формальных параметров в фортране. В целом организация интерфейсов в этом случае оказывается несколько сложнее [31].

Большая часть пакетов представляет собой библиотечные системы, в которых для сборки приходится полностью задавать управляющие и информационные связи [31—34]. Организация интерфейсов в большинстве случаев учитывает специфику предметной области, как в отношении стандартных обозначений, так и в отношении некоторых системных средств типа модулей «связки по данным». В таких случаях при переходе к другой предметной области приходится создавать системные средства заново [30—34].

Обратим внимание на организацию интерфейсов с помощью системы [9], когда применяется комбинирование текстов на базе библиотеки модулей, подкрепленное методом схем счета. В этом случае общие блоки фортрана в текстах модулей можно рассматривать как функциональные модули. Если для некоторого класса задач разработать систему набора стандартных общих блоков, то, вводя соответствие функциональных имен конкретным

именам, можно получить информационную связь на текстовом уровне. Аналогично можно поступить и с формальными параметрами, если обращения к подпрограммам в модулях заменить функциональными модулями. В результате установления соответствия получается обычная фортрановская программа. В основе подобного текстового комбинирования лежит техника макрогенерации в сочетании со стандартизацией в рамках соответствующей предметной области.

Мы не будем останавливаться на динамической организации распределения памяти с установлением гибких интерфейсов и отложим обсуждение этого вопроса до следующего параграфа.

**5. Виды пакетов и системные компоненты пакетов.** Рассмотрим вопрос структуры системной части пакета и общей организации работ. Здесь имеется большое разнообразие. Начнем с описания режимов работы.

Первая группа пакетов — это «пакеты-изготовители». С их помощью формируются программные комплексы, которые потом могут существовать независимо от пакета. В таком случае работа разбивается на два этапа: сперва формируются программный комплекс по информации, задаваемой пользователем пакета, а затем начинается эксплуатация [9]. Это не мешает готовому программному комплексу пользоваться услугами пакета, например архивами. К помощи пакета прибегают при модернизации комплекса и экспериментальных работах с ним. Так что в целом использование программного комплекса целесообразно проводить совместно с пакетом, а не отдельно. Однако в принципе программный комплекс — независимая программа.

При использовании других пакетов также отмечаются две стадии. Первая стадия может носить подготовительный характер, типа перетрансляции с некоторого специального языка пакета. В результате получаются программы, например, на фортране. Однако готовый программный комплекс существенно использует системные средства пакета — специальную управляющую программу, набор программ для работы с данными и для реализации управляющих связей [11, 127].

Третий вариант сводится к режиму совместной работы системных средств пакета и программного комплекса. Здесь могут быть различные варианты. Например, программный комплекс может оформляться как некоторый



«полуфабрикат», который возможно использовать только при участии системных средств пакета. В других случаях сам процесс изготовления может как таковой отсутствовать [10].

Системные средства пакета в последнем случае называют монитором пакета. Монитор пакета также состоит из ряда модулей, образующих определенные функциональные подсистемы [123]. Постоянно в оперативной памяти находится ядро пакета или резидент пакета, который организует всю работу. Анализируя управляющие операторы, содержащиеся в задании пользователя, резидент вызывает нужные подсистемы. Перечислим эти подсистемы.

**Процессор входного языка.** Переводит задание с языка пакета на некоторый внутренний язык.

**Планировщик.** Составляет последовательность модулей. Если последовательность модулей задается пользователем явно, то планировщик вырождается в программу обработки заданий пользователя на сборку.

**Исполнитель.** Реализует построенную планировщиком цепочку. На этапе работы исполнителя между модулями организуются интерфейсы. Исполнитель может работать как компилятор и тогда он создает рабочую программу и передает ее на счет. Он может работать как генератор программ или как интерпретатор. В последнем случае спланированная цепочка выполняется по мере «встраивания» очередного модуля. Если планировать не требуется — цепочка задана явно — исполнитель вызывает модуль, реализуя управляющие и информационные связи, и тут же передает его на исполнение, т. е. интерпретирует задание пользователя.

Остановимся на языке пакета. Обычно он является синтезом языка управления и языка описания задачи.

В качестве управляющих операторов могут служить операторы типа «записать в архив», «провести сборку», «передать на счет» и т. д. Возможные атрибуты таких операторов: «заголовок», «тело оператора». В заголовке описываются действия, которые надо провести, тело может содержать, например, текст модуля или описывать последовательность модулей и их информационные связи.

Управляющие операторы выполняют различные функции: они могут управлять процессами подготовки к работе, могут осуществлять управление работой пакета — запуск пакета, включение в работу различных системных

программ, могут осуществлять управление работой готовой программы [11], выдавать сообщение оператору о ходе работы пакета, осуществлять ввод-вывод данных и т. д.

Управляющий язык может включать в себя некоторые конструкции типа условных операторов или циклов, позволяющих более гибко организовывать работу пакета. Например, результаты расчета могут инициировать новое задание на сборку комплекса или циклическое выполнение сборок.

Язык для описания задачи — это проблемный макро-язык высокого уровня. Он должен включать содержательные понятия соответствующего класса задач.

Языки пакетов, как правило, представляют собой некоторое расширение универсальных языков за счет включения псевдооператоров [127]. Это позволяет провести при описании задачи некоторую дополнительную обработку, для которой нет модуля или его нецелесообразно вводить. К языкам, описывающим задачу, добавляются языки для работы с данными. В целом язык пакета должен отличаться доступностью для пользователя за счет использования терминологии предметной области, к которой пользователи привыкли.

## § 2. Примеры пакетов прикладных программ

В этом разделе мы приведем примеры реальных пакетов \*). В настоящее время существует большое количество пакетов. Можно привести такую классификацию пакетов по предметным областям [5].

А. Аналитические пакеты:

- а) алгоритмы и программы линейной алгебры,
- б) специальные функции,
- в) пакеты решения краевых задач для эллиптических уравнений,
- г) решение системы линейных алгебраических уравнений,
- д) пакет построения оптимальных сеток,
- е) пакет решения систем дифференциальных уравнений.

Б. Пакеты физических и инженерных программ.

---

\*) При изложении в случае необходимости мы будем отступать от прототипа. Это делается в целях упрощения изложения, которое и без того обременено деталями. Следовательно, приведенные примеры надо понимать как примеры некоторых типов пакетов, а не как примеры конкретных пакетов.

В. Пакеты программ по расчету явлений природы — метеорологические, океана и т. д.

Г. Пакеты задач управления.

Д. Пакеты управления технологическими процессами.

Е. Пакеты обработки экспериментальных данных в режиме реального времени.

Ж. Экономические задачи.

З. Пакеты программ для обработки массивов информации.

И. Пакеты для решения задач в реальном времени.

К. Пакеты, допускающие работу ЭВМ в режиме разделения времени.

**1. Система «Радуга».** В качестве первого примера рассмотрим систему из пакетов типа Б [11]. Она служит для расчета переноса излучения в некоторой среде без размножения. Математической моделью явления служит стационарное кинетическое уравнение. Рассматриваются цилиндрически симметричные задачи. В этом случае искомая функция является функцией пяти независимых переменных:  $r, z$  — пространственные переменные,  $\theta, \varphi$  — угловые переменные,  $E$  — переменная энергии. Результат расчета подвергается довольно сложным видам обработки: строятся системы графиков, таблиц, вычисляются интегралы и т. д.

Задача решается в многогрупповом приближении с использованием конечно-разностных методов. Переход от одного метода к другому осуществляется за счет параметров в записи аппроксимирующих уравнений. Мы не будем подробно останавливаться на организации вычислений в этой задаче, хотя она представляет интерес, как пример задачи, работающей на пределе возможностей машины. Читатель может найти подробное изложение в [11].

Для решения этой задачи была разработана модульная система, позволяющая рассматривать задачи, отличающиеся по физической, математической и дискретной модели, с ее помощью может рассматриваться задача с различными источниками — точечным на оси  $Z$ , параллельным оси  $Z$ , распределенным по точкам с различными граничными условиями. В этой системе предусмотрены разные варианты метода счета, рассматриваются варианты с ускорением сходимости итераций, без ускорения и т. д. Уже этих перечислений достаточно, чтобы понять целесообразность создания модульной системы.

Согласно приведенной классификации эта система относится к разряду систем с изначально заданными информационными и управляющими связями. Эти связи программируются заранее в тех или иных программных компонентах пакета. Конкретная цепочка модулей, определяющая задачу, выделяется заданием набора параметров.

Работа проводится в два этапа. Первый этап связан с выполнением подготовительных операций к сборке. Он заканчивается сборкой программного комплекса.

Подготовительный этап состоит в формировании трех компонент: модуля памяти, управляющего модуля и информации для сборки. Модуль памяти включает в себя все интерфейсы вычислительных модулей, так что в исходном виде последние таких интерфейсов не содержат. Сам по себе модуль памяти носит служебный характер, в нем содержится информация о распределении памяти всех уровней. Через внешние переменные при загрузке с помощью этого распределения реализуются интерфейсы модулей.

Получение модуля памяти может быть автоматизировано. Для этого может быть разработана специальная программа, которая по исходной информации будет готовить модуль памяти. В качестве такой информации могут быть параметры памяти. Они определяются количеством счетных точек, числом различных веществ, методом решения уравнения и другими характеристиками физической, математической и дискретной моделей. Кроме параметров памяти, надо задать набор массивов и коэффициентов уравнения переноса. Эти данные определяются также конкретной задачей. Помимо этих данных, связанных с задачей, нужно задать наличные ресурсы памяти, а также сообщить некоторые указания о месте расположения тех или иных массивов. Обработывая эти исходные данные, описываемая программа должна построить модуль памяти, который использует распределение памяти в относительных адресах, т. е. длины массивов.

Управляющая программа организует сегментацию программ и данных на второй стадии. При ее создании используется исходная информация на сборку. По ней строятся сегменты. Управляющая программа должна обеспечить вызов этих сегментов во время работы.

На подготовительной стадии готовится информация для сборки программ. Основой для этой информации слу-

жит набор модулей и указания по сегментации программ. Эта информация должна готовиться в соответствии с требованиями редактора внешних связей и загрузчика [57]. Помимо вычислительных модулей, загружаются модуль памяти и управляющий модуль. Модуль памяти в расчетах не используется, а модуль управления порождает сегмент управления, который при расчетах постоянно присутствует в оперативной памяти.

Эту информацию можно готовить вручную, а можно использовать некоторую автоматизацию: разработать программу, которая по заданным параметрам физической, математической и вычислительных моделей сформирует все необходимое. Модули хранятся в двух видах: в виде исходных модулей на автокоде и в виде модулей загрузки. Для хранения модулей используется специальный архив системы [57]. Исходные модули на автокоде хранятся в виде заготовок, используя которые с помощью операторов редактирования можно получить нужные варианты модулей. Такая организация хранения исходных модулей значительно сокращает количество хранимых модулей.

Загрузка организована таким образом, что транслируются только те модули, которые отсутствуют в архиве модулей загрузки, чем достигается экономия машинного времени.

Данные разбиваются на две группы — общие данные и данные для каждой из энергетических групп. Основная часть данных сегментов хранится во внешней памяти и циклически по энергетическим группам считывается в оперативную память для всех сегментов, осуществляющих расчет данной группы.

Под размещение сегментов используется 3К \*) оперативной памяти. Средний объем программного комплекса составляет 15 К. Библиотека модулей загрузки занимает 60 К памяти. Исходные модули занимают 170 К. Общее число модулей загрузки — около 45. Текст модуля составляет от 100 до 1000 предложений языка. Всякая конкретная программа состоит не менее чем из 10 сегментов.

**2. Система Фихар.** В качестве примера системы с изначально заданными информационными связями, рассмотрим систему Фихар [10, 132]. Эта система предназначена

---

\*) Система разработана для машины БЭСМ-6. К — это объем памяти для размещения 1024 слов.

для реакторных расчетов. Изложим принципы реализации системы с некоторыми упрощениями. С точки зрения задания связей здесь имеется случай, когда связи изначально заданы по данным. Второй особенностью системы является наличие автоматического планировщика, который позволяет строить цепочки модулей, для вычисления интересующих нас величин без участия пользователя. Третья черта — это возможность обращения к системе из универсального языка Алгол-60, точнее,  $\alpha$ -языка.

Работа в Фихар организована следующим образом. Вводится система стандартных обозначений для набора стандартных величин. Для них заводятся таблицы, в которых указывается место хранения величин в системе. Конкретный адрес стандартной величины заранее не известен и система сама, в процессе работы, под появляющиеся значения стандартных величин отводит память, т. е. имеет место динамическое распределение памяти. Величина, однажды вычисленная, хранится в системе до конца расчета. Кроме того, система имеет специальный архив для долгосрочного хранения вычисленных величин. Запись в архив идет по наборам. Каждому набору соответствует определенный номер задачи, при решении которой она получена. В дальнейшем имеется возможность сослаться на номер задачи и получить нужную величину, обращаясь к ней по имени.

Для каждой стандартной величины обязательно имеется модуль, ее вычисляющий. Исключение составляют стандартные величины, именуемые как DATA — это исходные данные задачи, задаваемые пользователем.

Для расчета пользователь составляет программное задание. Текст задания пишется на некотором подмножестве языка Алгол-60 [146], а обращения к системе осуществляются с помощью неописанной процедуры ФИХАР. Процедура имеет единственный параметр — строку в смысле алгола, а содержание строки определяют специальные операторы системы, содержащие стандартные величины. Заметим, что введенные обозначения имеют смысл стандартных величин только внутри оператора ФИХАР. Вне этого оператора те же обозначения имеют другой смысл. Содержимое строки на самом деле является описанием задания пользователя для системы на некотором специальном языке. Конструкции этого языка сходны с соответствующими конструкциями алгола, что значительно упрощает его усвоение.

Принцип работы системы можно описать следующим образом. При выполнении программного задания пользователя система, встретив «внутри» оператора ФИХАР обозначение стандартной величины, выясняет характер оператора системы, в котором она появилась. Если этот оператор требует присвоить стандартной величине некоторое вычисленное значение, то происходит присвоение этого значения. Если же оператор использует значение стандартной величины, то система сперва пытается найти его в банке данных. Если это не удастся, то строится схема вычислительного процесса для нахождения значения стандартной величины, по которой она вычисляется.

Выбор схемы вычислений, т. е. последовательности модулей, может происходить автоматически, а может использовать и указания пользователей. В первом случае система обращается к специальной таблице, которая содержит строки, задающие соответствие аргументов модуля, номера самого модуля и результатов работы модуля. По этой таблице определяется модуль, вычисляющий данную стандартную величину. Если пользователь явно задал этот модуль, то в схему вычислений он включается без обращения к таблице. В некотором случае для одной стандартной величины имеется несколько вычисляющих ее модулей. С помощью специального управляющего модуля происходит выбор нужного пути вычислений. Управляющий модуль в качестве дополнительной информации может использовать указания пользователей, данные по геометрии системы и другие. Выбрав модуль, система анализирует его аргументы. Если они имеются в банке данных, то планирование на этом заканчивается и модуль включается в схему вычислительного процесса. Если среди аргументов модуля есть стандартные величины, обнаружить которые в банке данных не удалось, происходит выбор следующего модуля и так до тех пор, пока не встретится модуль, аргументы которого известны.

Система стандартных величин отражает предметную область. В качестве стандартных обозначений употребляются идентификаторы в смысле алгола. Они должны соответствовать привычным обозначениям предметной области. Стандартные обозначения могут содержать параметры и по внешней форме они напоминают переменную с индексами. Однако параметры могут иметь условный характер и не обязательно принимают численные

значения. С их помощью может определяться номер группы по энергии, номер зоны, счетная точка, в которой вычисляется величина. Это может быть условное наименование вещества, граничное условие или численная методика. Смысл параметра определяется его местоположением.

Существует группа параметров, определяющих действия над стандартными величинами. Это параметры типа INT, ALL. Если вместо некоторого параметра стоит INT, то это означает, что надо проинтегрировать величину по всем значениям этого параметра. Запись  $SG(ALL):=300$  означает, что для всех значений параметра, т. е. всем компонентам, следует присвоить одно и то же значение 300.

Допускаются пустые позиции параметров. Например, запись  $TEF( ):=B[ ]$  может означать: компонентам стандартной величины присваиваются соответствующие компоненты некоторого вектора В. Аналогичный смысл имеет запись  $FR( , , , )$ , означающая, что некоторой операции подвергаются все компоненты пятимерной величины.

Следует обратить внимание на стандартное обозначение CODE («стандартное обозначение»). Если мы хотим, чтобы некоторая величина, фигурирующая в качестве параметра CODE, вычислялась с помощью некоторого модуля 'PIDIR', то нужно записать оператор в виде:

$CODE(EFK):='PIDIR'$

Если для некоторой величины мы хотим ввести начальные данные или начальные приближения при итерациях, то вместо  $QR(RG, NG)$  пишется  $QRO(RG, NG)$ .

Обратимся теперь к рассмотрению специальных системных операторов.

Начнем с оператора извлечения значения стандартной величины. В результате выполнения этого оператора происходит поиск стандартной величины или ее вычисление. Оператор извлечения может задаваться в двух видах. Первый соответствует случаю, когда в правой части стоит некоторая стандартная величина:

$B[, ]:=FR( , , INT, INT, INT)$ .

Такая запись означает, что компонентам некоторого двумерного вектора В присваиваются компоненты стандартной величины, «оставшиеся» после интегрирования по значениям остальных параметров.



Второй вид соответствует ссылке на результаты расчета некоторой другой задачи, хранящиеся в архиве:

$R := REF [5, DX (1)]$

Смысл этой записи таков: величине R присваивается значение стандартной величины DX (1), полученной в результате расчета задачи под номером 5. Оператор REF задает обращение к архиву.

Оператор задания значений стандартной величины позволяет присвоить значение некоторой стандартной величине. Теперь в левой части оператора присваивания стоит стандартная величина, которой надо присвоить значение. Левая часть оператора присваивания может строиться в виде алгольной конструкции списка:

$AD (235, 2) := AD (235, 1) := 0,00037$

Эта запись истолковывается как присвоение плотности  $U^{235}$  в разных зонах реактора одной и той же величины. Возможна также запись типа ссылки на вариант:

$DATA := REF [3, DATA]$

Последнее означает, что исходные данные для расчета должны быть взяты такими же, как и в задаче 3.

В записях оператора задания в качестве значений стандартных величин допускаются строки. Пусть имеется некоторая SE стандартная величина, значение которой определяет систему координат, в которой рассчитывается задача. Если величина равна 1, то это означает, что используются прямоугольные координаты, 2 — цилиндрические координаты, 3 — сферические. Однако вместо чисел в операторах задания можно писать строки 'ПРЯМ', 'ЦИЛИНД', 'СФЕРА' соответственно. Например, запись

$SE := 'СФЕРА'$

эквивалентна записи

$SE := 3$

Оператор вызова модуля записывается следующим образом:

$M 201 (EFK); M 026 (RC (238, 1, , ,))$

Подобная запись означает, что величину EFK нужно рассчитывать с помощью модуля M201, а RC(238, 1, , ,) — с помощью модуля M026.

Для задания типа памяти используется системный оператор, например:

MEM (FR):='МБ'

MEM (XSC):='МЛ'

Эти записи означают, что величины FR и XSC должны соответственно располагаться на магнитном барабане и магнитной ленте. Оператор задания типа памяти служит для задания системе указаний о расположении стандартных величин. Такие указания могут быть направлены на повышение эффективности использования памяти. При отсутствии указаний память под величины распределяется автоматически.

Оператор вывода записывается в виде

ВЫВОД (TEF (4), FR (, , , , ))

В приведенном примере он задает вывод одной компоненты стандартной величины TEF и всех компонент величины FR (, , , , ). Заметим, что если величины, записанные в операторе вывода, не вычислялись ранее, то оператор задает и их вычисление.

Аналогичный смысл имеет и оператор вывода результатов на магнитные носители архива:

АРХИВ (TEF (4), FR (, , , , ))

Истолкование его такое же, как и для оператора вывода. Если мы хотим что-то записать в архив, то предварительно нужно задать оператор:

NTASK:=L

Именно по номеру L в дальнейшем мы будем выбирать эти данные. Все указанные операторы фигурируют внутри оператора обращения к системе. Приведем пример такого обращения:

ФИХАР ('ВЫВОД (TEF ( ))')

Эта запись означает, что на печать будут выданы все компоненты величины TEF.

Обращение к системе может быть произведено из любого места алгольной программы. Следовательно, некоторые дополнительные вычисления можно проводить помимо системы.

Мы не будем подробно приводить полную систему стандартных обозначений. Она в значительной мере определяется областью приложений. Тем не менее можно отметить общие принципы. Это стремление сократить число обозначений за счет разумного введения параметров, сделать обозначения наглядными, обеспечить удобство их использования в операторах. Для сокращения записи вводят специальные параметры типа ALL. В число стандартных величин включают величины, в которых отражается методика счета, системы применяемых констант, так что значениями стандартных величин могут быть не только числа.

Оценивая в целом роль введения стандартных величин, следует отметить, что они являются мощным средством сокращения объема программного задания, делают его наглядным, не требуют описывать в каждом задании набор величин, решают вопросы интерфейсов модулей по данным с точки зрения реализации.

В то же время разработка системы стандартных обозначений весьма трудоемкая работа, требующая привлечения специалистов высокой квалификации. Кроме того, количество стандартных величин очень велико. Сами обозначения неизбежно громоздки, так как иначе они перекрывались бы. Эти соображения наводят на мысль, что метод стандартных величин требует дополнительных средств для создания удобств пользователей. Такими средствами могут быть системы перекрывающихся стандартных обозначений, другими словами, подсистемы. Можно говорить о создании словарей стандартных обозначений.

Заканчивая обсуждение языка заданий, отметим два важных момента, о которых мы упоминали вскользь. Первый связан с алголоподобной формой всех специальных операторов и обеспечивает быстрое освоение языка заданий пользователями. Второй момент обеспечивает наглядность программного задания, так как при записи задания на расчет фактическое обращение к модулям для вычисления стандартных величин не обязательно. Более того, такое обращение является исключением. Достаточно, если в правых частях операторов присваивания, или в операторах вывода, операторах работы с архивом, встретится стандартная величина, чтобы задать ее вычисление или поиск. Такой подход создает естественные формы записи задания с точки зрения пользователя.

Следующая компонента пакета — это тело пакета, т. е. набор модулей. В данном случае он представляет собой программы для решения реакторных задач. Все модули написаны так, что за некоторым исключением могут быть использованы в трехмерной геометрии. Одномерный и двумерный случаи рассматриваются как частные случаи трехмерного. Исключения, о которых мы говорили, относятся к расчетам нейтронного потока, которые могут проводиться только в одномерном и двумерном варианте для прямоугольной или цилиндрической геометрии.

Системная часть пакета состоит из ряда подсистем. Сама системная часть называется монитором пакета. Перечислим подсистемы.

**Б л о к в в о д а.** Осуществляет ввод программного задания и его трансляцию. Трансляция идет в два этапа. Сначала проводится трансляция специальных системных операторов на язык алгол, так называемая ф-трансляция. Она не затрагивает алгольного текста. На втором этапе происходит трансляция с алгола на язык машины. Подобная двухстадийная трансляция с языка заданий характерна для многих пакетов, в которых этот язык представляет собой сочетание универсального языка и некоторых специальных операторов.

**Б л о к в ы в о д а.** Осуществляет вывод стандартных величин, их обозначений, номера модулей, схем вычислительного процесса, диагностики по ошибкам пользователей.

**Б л о к п а м я т и.** Организует хранение программных заданий, модулей системы, стандартных и промежуточных величин. Для организации работы с банком стандартных величин разработаны специальные программы доступа.

Каждый модуль независимо от других помещает свои результаты в банк данных и берет оттуда нужные для него аргументы. Для этого имеется специальная таблица, содержащая указатели стандартных величин, число которых равно числу стандартных величин. В указателях содержится набор сведений о местоположении стандартных величин и методах доступа к ним. При завершении расчета стандартной величины происходит обращение к системе динамического распределения памяти, которая определяет место в памяти для стандартной величины и заносит соответствующие сведения в указатель этой стандартной величины. В случае выборки величины так-

же происходит обращение к блоку памяти, который с помощью указателей находит нужную величину и передает ее затребовавшему ее модулю. Банк данных системы реализован на оперативной, барабанной и ленточной памяти. Для каждой величины предопределен свой вид памяти. Пользователь может вмешиваться в этот процесс и сам определять вид памяти, которую должна использовать величина.

**Б л о к   у п р а в л е н и я.** Является резидентным, т. е. постоянно присутствующим в памяти. Он интерпретирует все обращения к системе и осуществляет извлечение и задание стандартных величин, проводит запись стандартных величин в архив и извлекает их из архива, осуществляет вывод стандартных величин, вызывает и выполняет модули. Для выполнения этих функций блок управления осуществляет обращение к соответствующим программам доступа.

**П р о ц е с с о р.** Планирует и реализует вычислительный процесс.

**П л а н и р о в щ и к.** Планирует цепочку модулей, используя граф информационных связей. Планирование, которое осуществляется до выполнения вычислений, называется статическим. Планирование, выполняемое в процессе вычислений, называется динамическим.

**И с п о л н и т е л ь.** Реализует выполнение вычислительного процесса. В системе Фихар программный комплекс не создается. Используется интерпретация, т. е. модуль вызывается и выполняется. После каждого выполнения модуля происходит обращение к динамической части планировщика.

**3. Система Сафра.** В заключение рассмотрим систему Сафра [9]. Она представляет возможность разработки пакетов на базе достаточно универсальных системных средств. «Увязывание» по информации и управлению модулей происходит на текстовом уровне. Пользователь сам указывает связи. В системе Сафра используются схемы счета с функциональными именами и вводится модуляризация, о чем будем говорить ниже. В предыдущих разделах шла речь о схемах счета. В системе Сафра этому понятию придается глубокий смысл.

Программа каждой большой задачи разбивается на ряд модулей, выполняющих отдельные функции. В нашем примере расчета параметров реактора к числу таких модулей относятся модули, рассчитывающие многогруппо-

вые константы ячейки реактора и другие. При этом каждую функцию может выполнять несколько модулей. Они могут отличаться по физической, математической и дискретной моделям, а также программной реализацией. Для нас важно, что они выполняют в некотором смысле одну и ту же функцию и какой-то один из них обязательно должен присутствовать в расчетах задачи из некоторого класса. Поэтому в дальнейшем мы будем для удобства употреблять термин функциональный модуль, имея в виду под ним некоторое собирательное понятие. Функциональному модулю можно поставить в соответствие некоторое имя. Будем называть его функциональным именем.

Теперь последовательность функциональных модулей можно рассматривать как представление задач некоторого класса. Построенная последовательность называется схемой счета некоторого класса задач. Она не задает управляющие связи полностью, так как для конкретной задачи требуется устанавливать соответствие между функциональными именами и архивными именами конкретных модулей. Одна из задач пакета заключается в том, чтобы с помощью набора системных средств предоставить пользователю возможность выбирать конкретные модули и устанавливать между всеми модулями необходимые информационные связи.

Обратим внимание на то, что подобная конкретизация осуществляется по указанию пользователя, так что автоматический выбор цепочки здесь не применяется. В ряде случаев он может оказаться нецелесообразным [4, 8, 11].

В Сафре предложен весьма остроумный и простой способ реализации управляющих связей с помощью задания двойного имени модуля: одно имя — функциональное, другое — архивное. Одному функциональному имени может соответствовать несколько архивных. Имея схему счета, с помощью системных средств пакета можно проблему сборки конкретной задачи свести к построению таблицы соответствий функциональных и архивных имен. Меняя отдельные строки этого соответствия, можно получать программы для различных задач данного класса. Класс задач определяется схемой счета.

Вторая задача пакета — реализация информационных связей, имеет непосредственное отношение к модуляризации. Внутри модуля можно выделить части текста, одинаковые для разных модулей и имеющие относительно самостоятельное значение. В качестве примера таких

частей можно указать операторы COMMON, DIMENSION. Такие части могут рассматриваться как модули низшего уровня. Они также могут иметь функциональные и архивные имена. Таким образом, Сафра относится к пакетам, где в качестве модулей выбираются не только самостоятельные программные единицы, допускающие независимую трансляцию, а в принципе произвольные части текста. Для включения модулей такого типа в цепочки используется макровывоз, т. е. в тексте, где в результате сборки должен появиться такой модуль, стоит вызов, оформленный в виде специального предложения языка системы.

Следовательно, для установления межмодульных связей в схеме счета имеется два пути с точки зрения технологии.

Первый путь предполагает использование конструкций языка программирования для вызова подпрограмм (например, CALL для языка фортран). В этом случае модуль оформляется как самостоятельная программная единица. Второй путь — это макровывоз модулей низшего уровня посредством предложения + INSERT.

Информационные связи можно организовать через общие блоки, которые оформляются в виде модулей низшего уровня. Поскольку к одному и тому же такому модулю возможно обращение из разных модулей системы, то путем установления соответствий между функциональными и архивными именами можно провести вставки модулей низшего уровня в многие модули.

Опишем дисциплину работы с пакетом. В системе вводится несколько типов модулей.

1. Программная единица PRUNIT.
2. Вставка MACRO.
3. Схема счета SCHEME.
4. Версия VERSION.
5. Вариант VARIANT.

Для хранения модулей имеется специальный архив [136, 137]. Средства системы позволяют помещать модуль в архив, исключать его из архива, а также проводить операции сборки для модулей, хранящихся в архиве.

Дадим пояснения, касающиеся типов модулей.

Модуль «программная единица» понимается как единица языка программирования [56]. Для фортрана такими модулями могут быть головные программы, подпрограммы, подпрограммы-функции, в языке алгол — это процедуры. Модуль может содержать некоторые систем-

ные операторы, которые при сборке конкретного варианта заменяются на соответствующие языковые конструкции.

«Вставка» — это модуль низшего уровня, включающий несколько предложений исходного языка или некоторые системные операторы. Смысл этих системных операторов состоит в том, что при сборке они заменяются на предложения исходного языка, т. е. эти операторы имеют характер макровыводов. Модуль «вставка» может содержать в себе ссылки на другие «вставки».

Модуль «схема счета» задает последовательность функциональных модулей. При сборке конкретной задачи нужно установить соответствие функциональных имен архивным именам. Это делается в модулях «версия» и «вариант».

Модуль «версия» служит для описания конкретной версии задачи. Здесь уже установлены соответствия для большинства функциональных имен.

Модуль «вариант» окончательно устанавливает все соответствия функциональных и архивных имен. На этом этапе обычно уточняются наборы исходных данных, ресурсы машины и т. д.

Мы видим, что по мере продвижения от схемы счета к варианту детализация нарастает. В связи с этим при разработке модулей возможно провести разделение труда, поручив составление схемы счета наиболее квалифицированным специалистам, а составление варианта специалистам меньшей квалификации.

Обратим внимание на общую направленность системы. Если каждый раз, например, задавать все соответствия и не использовать модули «версия» и «вариант», то работа обычно идет так, что от одной сборки к другой изменения невелики, поэтому для экономии труда целесообразно основную массу соответствий представить как модуль, а остальные «переменные» соответствия — как некоторые «вставки». При этом надо иметь в виду, что одному и тому же функциональному имени может ставиться в соответствие несколько архивных имен. Но реализовано будет только последнее соответствие. Отсюда возникает такая технология работы. В виде модуля «версия» составляются все соответствия имен. Если необходимо поменять одно из соответствий, то к модулю «версия» добавляется вставка, содержащая новое соответствие.

Таким образом, общая направленность системы состоит в учете общей технологии работ по вычислительно-



му эксперименту, когда каждый раз вносятся небольшие изменения. Им должны соответствовать небольшие трудозатраты в программировании, что и достигается за счет введения указанной иерархии модулей.

Обратим внимание на то, что приведенные типы модулей можно разбить на две группы. Первая группа — это «программные единицы», внутри которых содержатся ссылки на «вставки». Вторая группа — это «вставки», которые также внутри себя могут содержать ссылки на другие «вставки». При этом, правда, «вставка» не может ссылаться сама на себя. Все типы модулей, начиная со второго и кончая пятым, могут оформляться и использоваться как некоторые «вставки». Системные средства пакета позволяют по ссылке на некоторую «вставку» поместить ее при сборке в нужный модуль, причем ссылки даются на функциональное имя. При сборке же надо указать соответствие функционального и архивного имен. Модуль с таким архивным именем окажется в нужном месте текста.

Заметим, что вместо ссылок на функциональные имена можно организовать ссылки на архивные имена, употребляя их в ссылках так же, как и функциональные. Если система при поиске соответствий не найдет в модуле, задающем эти соответствия, данное имя, то она воспримет его как архивное имя. Таким образом, последовательность модулей в цепочке можно задавать явно, заменив функциональные имена на архивные и опустив соответствие имен, и можно задавать неявно с помощью соответствий имен.

При изложении принципов работы системы мы отмечали важную особенность системы, предоставляющую пользователю возможность экономить трудозатраты при внесении изменений. Вторая особенность состоит в том, что система позволяет вести отладку «сверху вниз». Этот путь, наиболее естественный с точки зрения вычислительного эксперимента, встречает серьезные затруднения при использовании штатного программного обеспечения. Приходится сначала проводить автономную отладку модулей, а затем переходить к комплексной отладке. В системе Сафра отладку можно вести сверху, начиная со схемы счета, в которой функциональные модули заменяют «пустышками». Можно также проводить несвязанную отладку, если работу неотлаженных модулей имитировать с помощью программ-макетов, выдающих в качестве результатов заготовленные заранее наборы данных. При

такой системе отладки надобность в комплексной отладке практически не возникает.

Остановимся на языке заданий системы. Задание для работы системы состоит из ряда пунктов. Пункт задания состоит из заголовка пункта и тела пункта. Формат заголовка имеет вид:

⟨признак⟩ ⟨операция⟩ ⟨тип модуля⟩: ⟨архивное имя модуля⟩

В качестве признака предложения языка заданий нужно выбрать некоторый символ, позволяющий отличить предложения языка заданий от других предложений, например, знак +. Операция определяет смысл пункта. Тип модуля указывается только в тех случаях, когда это существенно, например, для операций записи в архив. Тело пункта — это либо предложения задания, либо некоторый текст, например, программа на фортране. Задание оканчивается комбинацией

§ + : E

Пусть нам надо записать некоторую подпрограмму в архив. Тогда соответствующий пункт задания будет выглядеть так:

```
+ STORE PRUNIT : OLMU 04
  SUBROUTIN R VAR (NT, PV)
  COMMON / COMBAS /
  1 ALTIME, OPTIME, NSTEP
  2 NONLIN, NOUT, NRES
  1 NLEND, NL RES
```

```
. . . . .
  END
```

+ : E

Аналогично оформляется и исключение модуля из архива. В последнем случае тип модуля можно не указывать.

Работа с модулями типа «вставка» проводится с помощью предложения INSERT. Пусть COMMON-блок с фортранным именем COMBAS нужен в нескольких модулях. Тогда целесообразно выделить его в виде некоторой вставки

```
+ STORE MACRO : OLMC 11
  COMMON / COMBAS /
  1 ALTIME, OPTIME, NSTEP
  2 NONLIN, NOUT, NRES
  1 NLEND, NLRES
```

Теперь предыдущий модуль с фортранным именем RVAR можно записать в архив в виде

```
+ STORE PRUNIT : OLMU 04
  SUBROUTINE RVAR (NT, PV)
+ INSERT MACRO : OLMC 11
. . . . .
  END
+ : E
```

При сборке предложение INSERT заменится на приведенное выше выражение.

Пример работы со вставкой как раз дает нам случай явного указания модуля «вставки» — прямо называется архивное имя. Для неявного задания используются предложения MATCH, включаемые в модули типа «вариант» или «версия». В приведенном примере вместо строки + INSERT MACRO: OLMC 11 запишем строку + INSERT MACRO: COMBAS. Тогда для того, чтобы при сборке получить тот же результат, нужно оформить модуль типа «вариант»

```
+ STORE VARIANT: OLMVR 1
  MATCH COMBAS = OLMC 11
```

Выгода от подобной технологии становится заметной, если имеется много модулей, содержащих ссылку на функциональное имя COMBAS. В таком случае при необходимости изменить этот модуль без системы мы должны были бы проделать эти изменения в большом числе модулей. При этом могли остаться модули, где такие изменения по ошибке не были проведены. Описанная технология не только исключает подобные ошибки, но и существенно сокращает рутинную работу по замене модуля COMBAS на другой.

Опишем еще один пункт задания EXCUTE. EXCUTE служит для сборки задачи и запуска ее на счет. Для этой цели нам придется заготовить два модуля типа «вставка». Один из них имеет функциональное имя ДИСПАК и связан с необходимостью сформировать задачу в рамках операционной системы Диспак [125]. Второй модуль имеет функциональное имя ДУБНА и связан с базовой системой программирования, в качестве которой используется мониторинговая система Дубна. В модуле ДУБНА описывается сборка, т. е. сама цепочка модулей. Делает-

ся это либо с помощью предложений INSERT, либо в виде ссылки на некоторую схему счета.

Теперь окончательно соберем колоду перфокарт для сборки и запуска задачи на счет. Воспользуемся примером из инструкции по работе с системой Сафра.

А. Управляющие карты операционной системы Диспак.

ШИФР 031119 3С2-

ЛЕНТ 61 (2С)-

ЛЕНТ 60 (824 — 100 — 3П)-

ВРЕМЯ 100-

ЕЕВ1А3

Б. Управляющие карты мониторной системы Дубна.

\*NAME ПРИМЕР

\*TAPES : 60100

\*CALL EXECUTE

В. Заготовка управляющих карт для запуска задачи.

+ STORE MACRO: ДИСП 1

ШИФР 0311163 СК-

ЛИСТЫ 037-

ВРЕМЯ 200-

ТРАКТЫ 100-

Г. Заготовка карт по задаче для мониторной системы Дубна.

+ STORE MACRO: ДУБНА 1

+ S: \*NAME CRONUS

+ INSERT OLMSCH

+ S: \*PERSO: 60060, CONT

+ S: \*CALL TEMPO

+ S: \*EXECUTE

Д. Пункт, осуществляющий сборку и передачу на счет.

+ EXECUTE

+ INSERT OLMVR 1

MATCH ДИСПАК = ДИСП 1

MATCH ДУБНА = ДУБНА 1

Е. Группа карт конца.

+ :Е

\*END FILE

Д конец

Е конец

В пункте А и Б приведены управляющие карты операционной системы и мониторной системы Дубна, используемые для запуска системной части пакета. Пункт В — это уже работа пакета: в архив записывается модуль, представляющий собой управляющие карты операционной системы, нужные для запуска нашей задачи на счет. В пункте Г в архив записываются управляющие карты мониторной системы Дубна для нашей задачи. Поскольку управляющие карты мониторной системы начинаются с символа \*, карты этого пункта при работе Сафры могли бы восприняться как управляющие карты, а не как информации. Чтобы этого не произошло, вводится сочетание символов + S:. Здесь мы указали последовательность функциональных модулей с помощью ссылки на модуль типа схема OLMSCH. В пункте Д нужно задать соответствие имен. Для модуля OLMSCH такое соответствие мы задаем с помощью модуля типа вариант OLMVR 1, записанного в архив ранее; эту запись ради экономии места не приводим. Два других соответствия задаются предложениями MATCH.

Заметим, что если ДИСП 1 уже был записан в архив ранее, то не потребовалось бы приводить его в картах задания.

При работе системы будет сформирован пакет для мониторной системы по нашей задаче. Затем будет сформирована задача по запуску этого пакета на счет и он будет считаться уже с помощью мониторной системы. Обратим внимание на то, что пользователю не приходится иметь дело с модулями загрузки мониторной системы. Вся работа идет для него на уровне текстов. Однако при трансляции те модули, которые были уже протранслированы ранее, повторно не транслируются, а сразу загружаются во временную библиотеку мониторной системы. Это сделано для экономии машинного времени.

При изложении мы не коснулись многих возможностей системы. С ними можно ознакомиться по цитированной литературе.

## ГЛАВА 6

### ВОПРОСЫ ОРГАНИЗАЦИИ РАБОТЫ ВЫЧИСЛИТЕЛЬНЫХ ЦЕНТРОВ И ИХ СВЯЗЬ С РЕАЛИЗАЦИЕЙ ВЫЧИСЛИТЕЛЬНОГО ЭКСПЕРИМЕНТА

В предшествующих разделах мы видели, что организация вычислительных работ тесно связана со структурой самой задачи, методами ее решения, с программным окружением и конфигурацией машины. Попутно мы упоминали о роли дисциплины работы вычислительного центра, где предполагается проводить разработку и расчеты.

Говоря об организации работ здесь, как и на всем протяжении книги, мы имеем в виду не административные мероприятия по составлению графиков работ, подбора кадров и т. п. Эти вопросы нашли известное отражение в литературе [80 и др.]. В центре нашего внимания методический подход к разработке моделей задач, структуры программы, технологические аспекты программирования и проведения расчетов. С этой точки зрения для нас представляет интерес и организация работы вычислительного центра. Состав и назначение подразделений, режим и порядок выделения ресурсов, порядок проведения расчетных работ на машинах, различные положения и установки, действующие в рамках данного вычислительного центра — все эти факторы находят весьма заметное отражение на организации вычислительных работ в рамках решения той или иной задачи.

Если, например, имеется подразделение, которое может проводить сложные системные разработки, или сильная группа эксплуатации программного обеспечения, то принимаемые решения по разработке вычислительного эксперимента могут существенно измениться по сравнению со случаем, когда такие подразделения отсутствуют.

Большую роль играет возможность организации для задачи некоторого специального режима. Не менее существенную роль играет распорядок работы на машинах, возможность вести отладку в нужном режиме.

Ниже мы постараемся разобрать отдельно вопросы такого рода. Главная наша задача не столько дать исчерпывающее изложение, сколько заострить внимание на роли моментов при проведении вычислительного эксперимента.

Приводимые рисунки, графики и числа имеют иллюстративный характер и не относятся к какому-либо конкретному центру.

## § 1. Подразделения вычислительных центров и их функции

1. **Вычислительные центры.** В организационном плане вычислительные центры представляют собой весьма пеструю картину. В одних случаях вычислительные центры входят в состав предприятий или научных учреждений. В других — представляют собой самостоятельные организации, занятые решением сложных прикладных задач. Они включают в свой состав различные структурные подразделения, занятые постановкой задач, разработкой вычислительных методов, прикладным программированием, технической эксплуатацией оборудования и т. д. Существуют и специализированные центры, которые не ведут самостоятельных разработок, а принимают заказы на расчеты по готовым программам (своего рода коммерческие вычислительные центры). В основу классификации вычислительных центров могут быть положены и другие принципы, однако мы не будем на них останавливаться.

Нас будут интересовать центры только второй группы. Они в основном заняты поисковыми работами в области вычислительной и прикладной математики. Как правило, подразделения такого центра поддерживают тесные контакты с «предприятиями-заказчиками», т. е. предприятиями, где возникают прикладные задачи.

Число вычислительных центров очень велико [1, 2, 52]. Однако при этом трудно найти два центра, похожие друг на друга. Тем не менее мы постараемся составить «коллективный портрет» вычислительного центра второй группы.

**2. Подразделения вычислительных центров.** На рис. 6.1 мы привели структурную схему такого вычислительного центра, ориентируясь на круг наших задач. Наиболее крупные подразделения, будем их условно называть секторами, возглавляют известные специалисты в данной области. Секторы можно разбить на две группы — это прикладные секторы, занятые решением прикладных задач, и секторы, связанные непосредственно с машинами — это секторы системного программирования, эксплуатации программного обеспечения и аппаратуры. Надо сказать, что секторы прикладного программирования в центрах второй группы встречаются довольно редко. Однако это не означает, что прикладное программирование «исчезло» — им занимаются специалисты прикладных секторов. Набор прикладных секторов, приведенный нами на схеме, естественно, взят для примера. Он в значительной части определяется историей центра, наличием специалистов нужного профиля и организациями-заказчиками по данной тематике.

Несколько более стабильна вторая группа секторов. Правда, имеется некоторый разноречивый в организационных началах: например, системные подразделения и подразделения, занятые аппаратурой, могут быть объединены в одно подразделение, а могут существовать раздельно. То же самое можно сказать и о эксплуатационных подразделениях. Таким образом, эти группы представляют в том или ином виде следующие темы: системное программирование — новые разработки, развитие существующих средств; аппаратура — небольшие целевые разработки, например, объединение машин в комплексы, подключение новых внешних устройств и т. д.; эксплуатация программного обеспечения — доводка существующих средств программного обеспечения и их сопровождение, организация работ на машинах; эксплуатация аппаратуры — работы по поддержанию аппаратуры на заданном уровне.

В течение некоторого времени решение прикладных задач организовывалось силами прикладных секторов. Секторы системного программирования и пр. занимались лишь вопросами штатного программного обеспечения. Однако в связи с развитием пакетов, как технологии решения прикладных задач вычислительного эксперимента, появилась необходимость создавать комплексные группы разработчиков из прикладных секторов и системного сектора, а иногда и других «машинных» секторов [8].



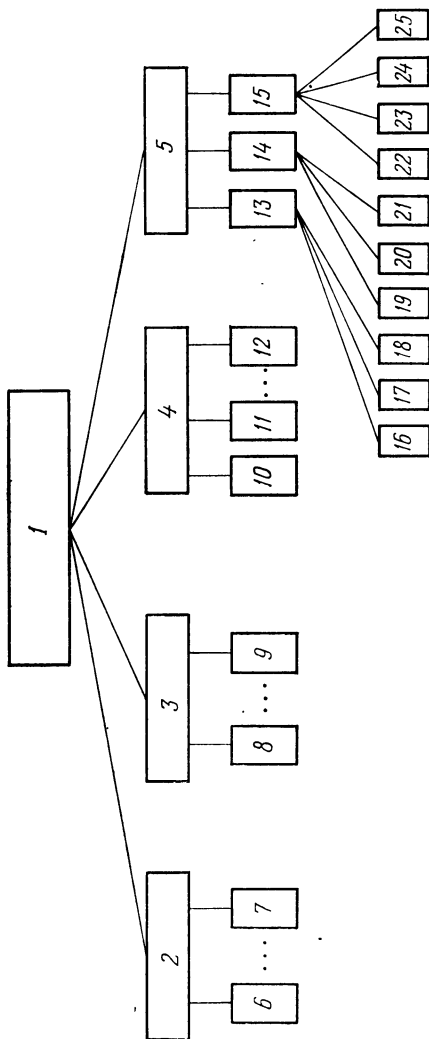


Рис. 6.1. Примерная схема вычислительного центра: 1 — директор ВЦ, 2 — заместитель директора и одновременно начальник сектора по группе прикладных задач, 3 — по другой группе, 4 — заместитель директора и начальник сектора по группе проблем вычислительной техники и программирования, 5 — заместитель директора и начальник сектора по машинной обработке. Примерные профили лабораторий: 6 — «физика атмосферы», 7 — «задачи плазмы», 8 — «механика сплошной среды», 9 — «задачи переноса», 10 — «системное программирование», 11 — «вычислительные системы», 12 — «системные вопросы модульного программирования», 13 — «эксплуатация вычислительной техники», 14 — «эксплуатация программного обеспечения», 15 — «организация работ на ЭВМ». Возможные группы в составе лабораторий: 16 — эксплуатация внешних устройств, 17 — эксплуатация электронного оборудования по типам, 18 — ремонтные мастерские, 19 — группа развития ПО (программного обеспечения), 20 — группа работы с пользователем, 21 — группа информатора (документации по ПО), 22 — группа пропуска задач, 23 — операторская группа, 24 — группа перфорации, 25 — группа статистического учета. Точки означают возможное увеличение числа подразделений.

3. Подразделения вычислительного центра и организация вычислительных работ. Возможности системных секторов оказывают существенное влияние на выбор технических решений при разработках вычислительного эксперимента. В некоторых случаях целесообразно удовлетвориться более простыми системными средствами, чем рисковать успехом всей разработки в целом. Иногда удастся значительно упростить решение ряда проблем, если прибегнуть к некоторой доработке штатных средств программного обеспечения и, в частности, операционной системы. Такие доработки могут касаться создания экстракодов, новых алгоритмов обслуживания очередей и др. разделов. Для систем программирования могут быть разработаны специальные стандартные программы типа программ обмена с магнитными носителями, программ обработки результатов расчета, специальные программы оформления выдач [151], а также программы работы в интерактивном режиме. Отказываться от использования таких системных разработок нецелесообразно, так как средствами прикладного программирования многие трудности преодолеть не удастся, тогда как средствами системного программирования преодолеть их не составляет труда.

Большую роль играют подразделения эксплуатации. От них зависит выбор штатных средств программного обеспечения, например, языков программирования. Подразделения эксплуатации обычно заняты подготовкой дополнительной документации по штатным средствам, проведением консультаций. В их обязанности входит также накопление опыта работы с различными средствами программного обеспечения. У них же концентрируются сведения по внутренней структуре организации таких средств.

Нужно иметь в виду, что при недостаточно высокой квалификации сотрудников подразделения эксплуатации лучше пользоваться проверенными средствами, так как переход на новые потребует знакомства с документацией, сопровождающей штатные средства. Последняя, как правило, в лучшем случае содержит описание возможностей, однако абсолютно не касается цели, ради которой они создавались. Зачастую документация расходуется с реально существующими на машине программами. Много трудностей вносит нестабильность программных средств. Исправляются ошибки, появляются но-

вые версии [49]. Разобраться в этой путанице без посторонней помощи не представляется возможным. В связи с этим целесообразно вносить в документацию, сопровождающую разработку, пункты о проведении консультаций, семинаров со стороны группы эксплуатации. Если участие групп эксплуатации затруднено — лучше обойтись более простыми средствами, чем рисковать всей разработкой.

На нашей структурной схеме отсутствуют два подразделения, которые, по мнению ряда специалистов, играют важную роль в работе всего центра — это служба стандартизации и патентная служба. Служба стандартизации может превратиться в источник передового опыта. Она может следить за выполнением порядка внедрения средств программного обеспечения и режимом их сопровождения, разрабатывать соответствующие инструктивные материалы. Патентная служба может взять на себя, помимо обычных ее обязанностей, защиту авторских прав на программную продукцию. Возникающие здесь проблемы можно пояснить следующим образом [138].

Программный комплекс, и тем более пакет, требует для своего создания большого труда. Если подсчитать стоимость разработки пакета и стоимость проведения расчетов по готовому пакету, т. е. стоимость реализации вычислительного эксперимента, то последняя составит лишь некоторую часть первой.

В то же время заказчик, получив программный комплекс в готовом виде и проведя по нему расчеты, будет восприниматься как основной производитель работ и единственный автор полученных результатов. Возникающие здесь коллизии могут привести вообще к стремлению не создавать полноценную документацию и не расставаться с программным комплексом, что явно наносит ущерб делу.

Правда, системой разумных административных мероприятий часто удается в значительной мере преодолеть возникающие здесь трудности.

Наконец отметим подразделение, занятое организацией работ на машинах. Зачастую для решения сложных задач требуется создавать специальные условия пропуска задач, подбирая для них специальную мультипрограммную смесь. Требуется включение некоторых специальных режимов работы машины или программного обеспечения, например, ограничение потока мелких задач, выбор другой дисциплины обслуживания очередей. Надо

ранее понять, насколько удастся воспользоваться помощью этого подразделения. Его участие лучше оговорить в административных документах, сопровождающих разработку.

Нужно хорошо знать возможности различных подразделений и понимать, что их влияние на разработку очень велико — вплоть до изменения расчетных формул.

## **§ 2. Характеристики вычислительных центров и их влияние на проведение вычислительного эксперимента**

**1. Некоторые замечания о вычислительных средствах.** Основная характеристика каждого вычислительного центра — это состав вычислительных средств. В большинстве случаев в вычислительном центре имеется не одна машина, а некоторый парк машин, объединенный в систему. В настоящее время существует много вариантов построения многомашинных комплексов. Большое распространение имеет вариант, в котором машины относительно независимы, но могут обмениваться информацией через каналы внешней памяти [140], [148]. В таком случае можно считать, что для решения одной задачи используется одна машина. На самом деле, возможно распараллеливание решения больших задач, но оно реализуется за счет использования средств внешней памяти, причем типичные задачи распараллеливания алгоритмов практически не возникают.

В настоящее время широкое развитие [141] получают многопроцессорные универсальные вычислительные системы, обладающие соответствующими аппаратными средствами для распараллеливания алгоритмов. Вместе с тем метод последовательного счета, которым мы занимаемся в этой книге, по всей вероятности, имеет перспективы длительного применения.

**2. Семейство задач вычислительного центра.** Следующей важной характеристикой вычислительных центров является набор решаемых задач. Задачи могут классифицироваться по предметным областям: инженерно-физические, экономические, задачи управления, проектирования, задачи реального времени и др.

Приведем другую классификацию задач: методические и производственные. Первые характеризуются

малым числом расчетов по готовой программе, для вторых это число достаточно велико [71]. По виду работ задачи могут делиться на счетные и отладочные. По режиму работы различаются задачи пакетные (может быть, запускаемые в пакет с терминала) и интерактивные задачи, т. е. задачи, с которыми поддерживается двусторонняя связь в процессе решения.

Для наших целей очень важны характеристики задач, связанные с использованием ресурсов машины. Раньше мы упоминали понятие — фоновая задача. Под такими задачами понимаются задачи с относительно большим временем счета по сравнению со временем обмена (задачи с большим временем счета и малым числом обращений к внешним устройствам).

Представляют интерес и динамические структуры задачи. Исследование таких структур можно проводить с помощью отладочных систем [23]. Поясним термин динамические структуры. Программа может выполняться последовательно — команда за командой. Это линейные участки программы. В программе могут быть циклы, ветвления. Отдельные команды программы могут выполняться многократно, другие — однократно. Есть изменяемые в процессе выполнения программы [6]. Эти и другие характеристики программы, проявляющиеся на стадии выполнения, образуют ее динамическую структуру.

Важная характеристика задач — используемое машинное время. Существует много «видов» машинного времени. Наиболее широко используемый вид — время центрального процессора (ЦП), затрачиваемое процессором на решение задачи. Второй вид — коммерческое время — это время полного обслуживания задачи, включая работу каналов внешних устройств. В него не входит время ожидания, когда операционная система не производит никакой полезной работы для задачи. Третий вид — время занятия счетного канала. Время от момента включения задачи в мультипрограммную смесь до окончания ее счета. Иногда говорят о заказанном процессорном времени и фактическом процессорном времени. Наконец важно понятие — астрономическое время. Это время от введения задачи в машину, до полного завершения решения задачи. Из других характеристик задачи важное значение имеют число обращений к внешним устройствам и длина листинга, выдаваемого на АЦПУ при решении задачи.

При изучении коллективного «портрета» задачи поступают следующим образом. Разбивают все задачи по процессорному времени на классы, например, задачи с процессорным временем от 0 до 1 минуты — первый класс,

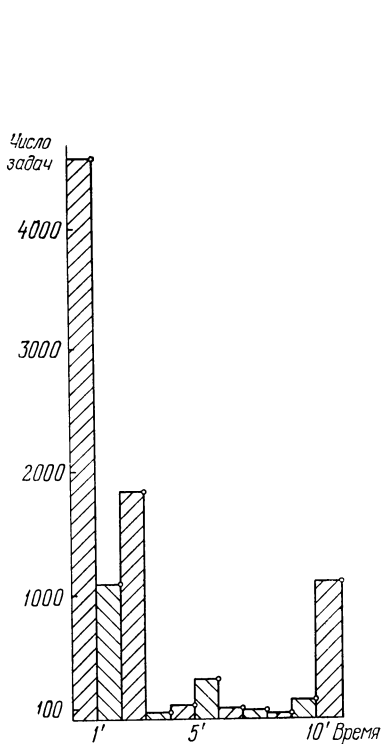


Рис. 6.2.

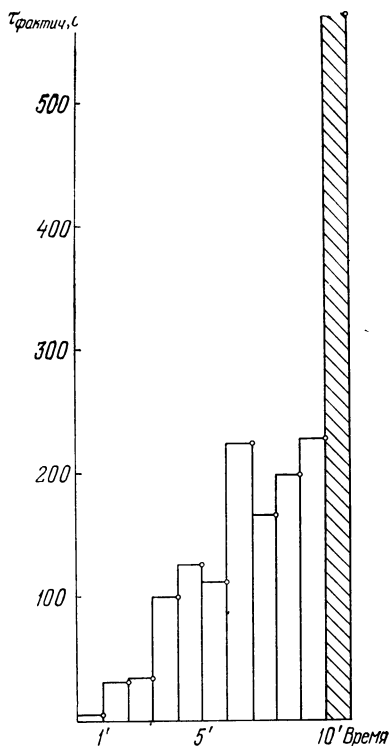


Рис. 6.3.

от 1 до 2 минут — второй класс и т. д. А все указанные характеристики определяют в виде полигона частот в зависимости от классов по процессорному времени. На рис. 6.2 изображено количество задач в каждом классе, причем взято заказанное процессорное время. Этот график носит название спектра задач по процессорному времени \*). На рис. 6.3 — среднее фактическое процессорное

\*) По оси ординат отложено число задач. Спектр составлялся по выборке из 10 000 задач. По оси абсцисс отложено время в минутах.

время \*). Последний рисунок показывает, что при малых временах заказанное время значительно превышает израсходованное. Аналогичное положение имеет место и для таких ресурсов, как оперативная память, память на барабанах.

Помимо процессорного времени для построения различных зависимостей могут использоваться другие независимые переменные: оперативная память, память на барабанах и др.

При анализе изменения состава задач в течение суток или дней недели можно составлять подобные полигоны частот для соответствующих интервалов времени. Данные по задачам часто подвергают статистической обработке.

Для вычислительного эксперимента изучение характеристик задач может оказать неоценимую услугу в двух направлениях. Во-первых, выбор структуры ресурсов задачи целесообразно проводить с учетом дефицита на те или иные ресурсы. Это в значительной степени облегчит работы с задачами. Здесь можно использовать возможности увеличения одних ресурсов за счет уменьшения других.

Как мы видели, уменьшение объема оперативной памяти может быть достигнуто за счет дополнительных пересчетов. Однако существуют и более сложные случаи учета сведений о задачах. При большом количестве мелких задач надо стремиться сократить число выходов на машину. Это может потребовать упрощения синтаксических конструкций и алгоритмов задачи, однако, возможно, ценой увеличения процессорного времени счета или оперативной памяти. В целом такой подход может дать выигрыш. В случае большого количества задач счетного плана целесообразно экономить процессорное время, однако количество выходов на машину может возрасти. Если приходится экономить бумажную ленту, то следует писать предварительно выдачи на магнитный носитель. Подобные дилеммы могут быть разрешены только за счет изучения потока задач вычислительного центра. Многообразие представленных здесь вариантов очень велико. В каждом конкретном случае необходимо принимать соответствующее решение, а чтобы оно было правильным, организуя вычислительные работы, всегда нужно иметь

---

\* ) По оси ординат отложено в секундах фактически израсходованное процессорное время.

перед глазами «коллективный портрет» задач вычислительного центра.

Во-вторых, часто возникает необходимость в создании специальных условий для больших задач. К таким условиям могут относиться некоторые доработки штатных программных средств, использование других средств автоматизации программирования помимо тех, что используются обычно. Задание на доработку должны составлять разработчики тех задач, для которых это необходимо, а последнее также требует хорошего знания «задачного» окружения.

**3. Программное обеспечение.** Следующей важной характеристикой для наших целей является состав программного обеспечения вычислительного центра. О роли систем программирования, архивов, сервисных программ мы уже говорили, когда обсуждали вопросы выбора языка и модульное программирование. В большинстве случаев разработчики достаточно осведомлены об этих компонентах программного обеспечения. Несколько сложнее обстоит дело с операционной системой. Остановимся на вопросе обслуживания очередей задач. Почему этот вопрос важен? Если бы под каждую большую задачу можно было отвести отдельную вычислительную машину — этого вопроса могло не быть. На самом же деле, на средней машине с быстродействием около одного миллиона операций количество запусков задач в течение суток составляет от 400 и более.

Вся эта работа организуется операционной системой с помощью аппарата обслуживания очередей [27]. В основном, можно выделить три очереди, возникающие при «прохождении» задач. Первую очередь (назовем ее «диспетчерской») образуют задачи, заявки на счет которых переданы диспетчеру. Порядок счета подобных задач определяет диспетчер. Таким образом, обслуживание этой очереди идет вручную. Могут быть различные пути автоматизации этого процесса. Например, можно организовать вспомогательный диспетчерский буфер, на котором будут собираться эти задачи, и откуда они по команде диспетчера будут передаваться в буфер ввода системы. Почему в буфер ввода операционной системы нельзя вводить все задачи сразу? Это связано, с одной стороны, с ограниченным размером буфера, а с другой стороны с недостаточно развитыми средствами управления разгрузкой очереди на буфере ввода, предоставленными



диспетчеру (эту задачу система берет целиком на себя, так что вмешательство в работу системы — скорее исключение, чем правило).

Вторая очередь образуется на буфере ввода системы, куда попадают не только пакетные задачи, введенные с перфокарт, но и задачи, запущенные в пакет с удаленных терминалов. Эти задачи по специальному алгоритму операционной системы вводятся в мультипрограммную смесь. Такую работу в операционной системе выполняет планировщик, который учитывает приоритеты задач, время ожидания на буфере ввода, заказанные ресурсы. Для выбора задач, которые должны быть введены в мультипрограммную смесь, строится порой довольно сложный алгоритм.

Третья очередь — это порядок обслуживания задач, находящихся в решении. Этим также заведует специальный планировщик системы. Мы будем называть его внутренним планировщиком. Один из вариантов планирования заключается в разбиении всех задач на две группы. Первая группа задач обслуживается процессором по системе «круговорота». При этом каждой задаче выделяется определенный квант процессорного времени, который она может использовать. Если задача использовала свой квант или же не использовала, но дальше она выполняться не может (требуется обмен с внешним устройством), процессор передается следующей по кругу задаче. Может быть такой случай, что при обходе всего круга не оказалось ни одной готовой к счету задачи. Тогда процессор передается задачам второй группы. Эти задачи могут обслуживаться в порядке приоритетов занимаемых ими каналов. Сначала анализируется состояние задач в счетных каналах. Если задача в первом канале не готова к выполнению, то анализируется состояние задачи во втором канале и т. д. Как только выясняется, что задача с большим приоритетом готова к счету, процессор передается этой задаче.

При подобной дисциплине обслуживания целесообразно задачи, связанные с терминалами, обслуживать в режиме «круговорота» [27].

Приведенная нами схема достаточно примитивна, обычные системы обслуживания очередей достаточно сложны. В частности, алгоритмы могут учитывать оптимальную загрузку устройств машины с применением прогноза на некоторый интервал времени.

Несмотря на такой сложный и развитый аппарат обслуживания очередей, все же на сегодня для пользователя имеются две трудности. Первая трудность связана с тем, что это самый нестабильный раздел операционной системы, подвергающийся наибольшим изменениям. Последнее связано с тем, что любой жесткий алгоритм в каких-то условиях оказывается неудовлетворительным. Отсутствие стабильности не позволяет выбрать разработчику удобную для него стратегию использования машины. Вторая трудность носит принципиальный характер и связана с тем, что обслуживание очередей — в нашем случае, трудно формализуемая задача. Заметим, что в особенности это относится к первой очереди, не связанной с операционной системой, которая должна учитывать административные требования срочности задачи, процент использования бюджета, сложившееся состояние очереди на данный момент и ряд других. Трудности возникают также и при попытке выбрать критерий оптимального обслуживания.

Существует большое количество алгоритмов обслуживания очередей. Большая их часть опирается на введение для каждой задачи одного параметра, типа «цены», которую можно заплатить за задачу с данным приоритетом срочности. С другой стороны, для той или иной категории срочности обслуживания задачи вводится некоторая стоимость. В соответствии с возможностью уплатить некоторую цену и производится обслуживание задач. В других случаях цену устанавливают в соответствии со спросом на ресурсы, нужные для той или иной задачи. Таким образом, ресурсы в разных категориях срочности не только могут иметь разную цену, но и цена эта еще меняется в зависимости от конъюнктуры «рынка», причем заранее неизвестным образом. В иных случаях предлагается своего рода «аукцион» на категорию срочности. В ряде работ рассматривается система цен, корректируемая с учетом сбалансирования загрузки оборудования [139]. Общий недостаток этих систем в том, что они мало связаны со второй и третьей очередью, их использование не исключает возникновения стихийных и неуправляемых ситуаций. Практика показывает, что требуется с известной гибкостью управлять не только первой очередью, но и внутренними очередями, т. е. второй и третьей.

В реализации алгоритмов обработки очередей также наблюдается большое разнообразие. При делении на

три очереди в алгоритме обслуживания первой, диспетчерской очереди, можно учесть требования административных приоритетов, принадлежность ее к той или иной теме, степень использования бюджета, количество запусков, время последнего пуска и ряд других. Для второй очереди учитывается среднее время ожидания и типы задач, используемые ими ресурсы и сбалансированная загрузка устройств машины и др. Третья очередь может учитывать оптимальное использование ресурсов машины, сбалансированную загрузку, режимы счета специальных задач и прочие моменты.

По-видимому, наиболее рациональный подход можно описать следующим образом. Подверженные частым изменениям компоненты алгоритмов обслуживания очереди следует вынести на другой уровень операционной системы для возможности гибкого управления обслуживанием очередей. «Жесткие» компоненты следует оставить на базовом уровне операционной системы. Это связано с тем, что требования к дисциплинам обслуживания очередей меняются в зависимости от многих обстоятельств, которые невозможно предусмотреть заранее. Проще и эффективнее поручить рутинную работу машине, оставив за диспетчером лишь функции определения общей политики обслуживания очередей. Диспетчер должен располагать набором средств разгрузки очередей, позволяющих управлять этим процессом. Приведем пример такого типа управляющих указаний: пропустить в первую очередь задачи по некоторой теме, обеспечить приоритет задачам с данным временем счета и т. д.

Алгоритмы обслуживания очередей могут составлять открытую библиотеку, допускающую любое расширение.

Для принятия решений диспетчер должен иметь возможность получить оперативную информацию о состоянии очереди в обозримом виде. Соответствующие системные средства должны носить диалоговый характер, с тем, чтобы диспетчер мог пользоваться ими как справочником, задавая вопросы и получая ответы. Соответствующие алгоритмы также могут входить в открытую библиотеку.

Таким образом, речь идет о создании модульной системы, позволяющей конструировать из набора модулей на каждый данный момент нужную дисциплину обслуживания, а также информационно-справочную систему опроса состояния очереди.

Мы подробно остановились на дисциплине обслуживания «прохождения» задач, организуемой программным путем, так как она может иметь прямое отношение к разработке вычислительного эксперимента. Прежде всего задачи вычислительного эксперимента сами обслуживаются этой же системой. Кроме того разработка средств решения соответствующих задач в значительной мере зависит от дисциплины обслуживания прохождения задач. Возьмем хотя бы такой момент, как отладка задачи с большим количеством выходов на машину, трудности пропуска задач с большими ресурсами при малом процессорном времени. Дисциплину обслуживания очередей, как и поток задач, можно исследовать статистическими методами. Большой интерес представляют числовые характеристики. Например, среднее время ожидания в той или иной очереди для задач разных классов или максимальное время ожидания. Знание статистических характеристик дисциплин обслуживания очередей очень важно для организации вычислительных работ. Число таких характеристик может быть достаточно большим. Они могут показывать зависимость времени того или иного обслуживания от заказанных ресурсов и тем самым помогать выбрать оптимальную стратегию организации вычислительных работ.

Обсуждение этих интересных и важных вопросов заняло бы слишком много места. Мы ограничимся для иллюстрации зависимостью среднего времени (по заданному классу задач) ожидания на буфере ввода операционной системы (рис. 6.4) и среднего времени занятия счетного канала (рис. 6.5)\*).

В ряде случаев для завершения разработки могут потребоваться изменения дисциплины разгрузки очередей. В таком случае приходится предпринимать меры по перестройке соответствующей дисциплины. Именно поэтому разработчики больших задач проявляют большой интерес к вопросам пропуска задач.

Приведенный нами пример далеко не исчерпывает всех компонент операционной системы, интересных с точки зрения разработки вычислительного эксперимента.

Подводя итог сказанному, можно сделать заключение, что проведение вычислительного эксперимента требует

---

\*) На рис. 6.4 и 6.5 по оси ординат отложено время в секундах, а также указаны точки, соответствующие времени в 1 час, 2 часа, 20 минут (20'), 40 минут (40').

от разработчиков глубокого знакомства с разнообразными разделами программного обеспечения, даже такими, которые раньше традиционно интересовали лишь системных программистов, занятых разработкой штатного программного обеспечения. Мы имеем в виду, например, операционную систему.

4. Системы административных правил и инструкций. Следующая характеристика вычислительного центра — это система правил и распоряжений, регламентирующих работу по проведению расчетов. Они носят характер нормативно-технической документации и их можно рассматривать как стандарты предприятия. Ряд специалистов придает

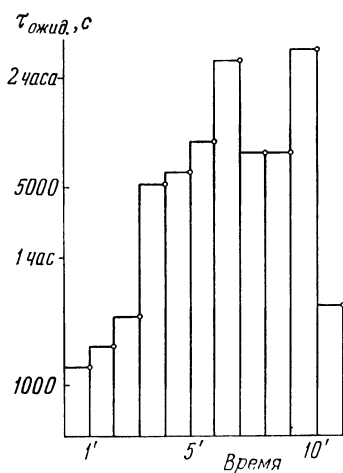


Рис. 6.4.

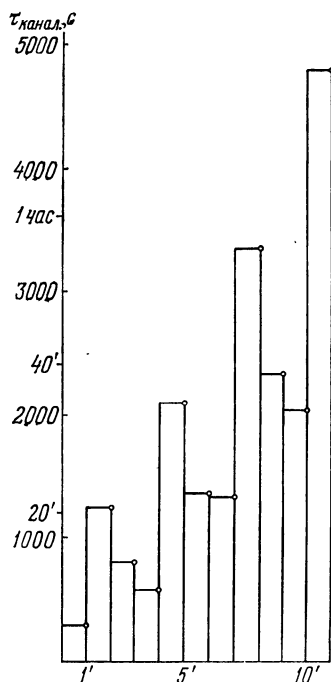


Рис. 6.5.

им большое значение [2, 78]. Приведем некоторые примеры таких материалов.

«Положение об эксплуатации вычислительных машин». В нем определяется порядок работы вычислительных машин, указываются режимы работы (функционирование той или иной операционной системы, используемая конфигурация аппаратуры, график работы машин, опреде-

ляется режим профилактических работ). Может быть оговорен специальный режим работы без операционной системы для профилактики средств программного обеспечения.

«Положение о порядке сдачи материалов на счет». Здесь может быть описано оформление инструкции для оператора, время сдачи всех материалов по задаче, необходимых для счета на машине, порядок их приема.

«Положение об использовании и распределении ресурсов». Здесь указывается перечень распределяемых ресурсов, принципы распределения, а также порядок распределения и участия в нем заинтересованных лиц. Указывается периодичность распределения, а также система документов, сопровождающих процесс распределения. Распределяться могут процессорное время, память в архивном пространстве, бумажная лента, магнитные носители, сеансы работы за терминалами. Указывается порядок предъявления претензий по распределению ресурсов.

«Режим работы математиков». Описываются режимы работы. Указывается оборудование, к которому разрешен доступ, регламентируется проход в различные помещения, связанные с машиной.

«Положение о системе отчетности по использованию ресурсов машины». Приводится перечень и периодичность материалов отчетности по использованию ресурсов машины. Указываются лица, ответственные за подготовку этих материалов.

Помимо положений могут быть подготовлены инструкции по работе в перфораторной, за терминалом, за пультом, по установке носителей на устройства и многие другие.

Наконец важная группа документов — положения о правах и обязанностях отдельных подразделений и лиц, например, групп эксплуатации, диспетчеров, операторов и т. д.

Разработкой таких положений обычно занимаются подразделения, ответственные за оборудование, эксплуатацию программного обеспечения, дирекция. Однако после подготовки соответствующих документов рекомендуется [2] провести опрос всех заинтересованных лиц. Во всяком случае, разработчики сложных задач вычислительного эксперимента должны принимать участие в таких обсуждениях, отстаивая свои интересы.

## ЛИТЕРАТУРА

1. Липаев В. В. Состояние и проблемы производства программного обеспечения для систем управления и обработки информации.— УС и М, 1980, 1.
2. Шараяев Д. Я. Организация больших программ.— Свердловск: Уральский гос. университет, 1977.
3. Давыдова И. М., Давыдов Ю. М. Организация больших программ.— М.: МФТИ, 1963.
- 3а. Давыдова И. М., Давыдов Ю. М. Элементы организации больших программ.— М.: МФТИ, 1977.
4. Самарский А. А. Теория разностных схем.— М.: Наука, 1977.
5. Яненко Н. Н., Карначук В. И., Коновалов А. Н. Проблемы математической технологии.— В кн.: Численные методы механики сплошной среды.— Новосибирск: ВЦ СО АН СССР, 8, 3, 1977.
6. Дал Д., Дейкстра Э., Хоор К. Структурное программирование.— М.: Мир, 1975.
7. Зельдинова С. А., Паремский М. В., Тюрин В. Ф. Некоторые базовые возможности ОС ДИСПАК.— М.: Препринт № 21 ИПМ АН СССР, 1976.
8. Карпов В. Я., Корягин Д. А., Самарский А. А. Принципы разработки пакетов прикладных программ для задач математической физики.— ЖВМ и МФ, 1978, 18, 2.
9. Горбунов-Посадов М. М. и др. Пакет прикладных программ САФРА. Системное наполнение.— М.: Препринт ИПМ АН СССР, № 85, 1977.
10. Башмачников А. Н. и др. ФИХАР — модульная система программ для реакторных расчетов.— В кн.: Труды III семинара по комплексам программ математической физики/Под ред. Яненко Н. Н.— Новосибирск: ВЦ СО АН СССР, 1973.
11. Басс Л. П., Гермогенова Т. А., Хмылев А. Н. Модульная структура программ в осесимметричных задачах теории переноса. Система «Радуга».— М.: Препринт ИПМ АН СССР, № 97, 1973.
12. Марчук Г. И. Об автоматическом построении вычислительных алгоритмов.— *Aplocace matematicy*, 1965, 10, 3.
13. Marchuk G. I., Yershov A. P. Man-machine interaction in solving a certain class of differential equations.— *Infor. proc.*, 1965. *Proc. of 6 IFIP. congress*, v. 2, Longon, 1965.  
О системе программирования некоторого класса дифферен-

- циальных уравнений, базирующейся на взаимодействии человека и машины.— В кн.: Труды конференции ИФИП, 2, 1965.
14. Столяров Л. Н. Краткий обзор принципов организации пакетов прикладных программ.— В кн.: Труды IV Всесоюзного семинара по комплексам программ математической физики / Под ред. Яненко Н. Н.— Новосибирск, ВЦ СО АН СССР, 1976.
  15. Королев Л. Н. Структуры ЭВМ и их математическое обеспечение.— М.: Наука, 1978.
  16. Коган Б. М., Каневский М. М., Цифровые вычислительные машины и системы.— М.: Энергия, 1974.
  17. Мямлин А. Н., Соснин А. А., Супер ЭВМ: архитектура и области применения.— М.: Препринт ИПМ АН СССР, № 15, 1977.
  18. Четвертое поколение / Перевод с англ. под ред. Смирнова В. К.— М.: ИПМ АН СССР, 1973.
  19. Вычислительная техника за рубежом в 1976 г. / Под ред. Зейденберга В. К.— М.: ИТМ и ВТ АН СССР, 1977.
  20. Карцев М. А. Вычислительная машина М10.— ДАН СССР, 1979, 245, 2.
  21. Шура-Бура М. Р. Библиотека стандартных программ.— М., 1961.
  - 21а. Камынин С. С., Любимский Э. З. Алгоритмический машинно-ориентированный язык — АЛМО.— Алгоритмы и алгоритмические языки, вып. I, 1967.
  22. Мартынюк В. В. О методе символических адресов.— В кн. Проблемы кибернетики, вып. 6.— М.: Физматгиз, 1961.
  23. Топалов Н. Н. Системы и методы отладки программ.— М.: Препринт ИПМ АН СССР, № 81, 1976.
  24. Самарский А. А. Введение в теорию разностных схем.— М.: Наука, 1971.
  25. Самарский А. А. Аддитивные схемы.— В кн.: Тезисы докладов на Международном съезде математиков в Москве, 1966.
  26. Марчук Г. И., Яненко Н. Н. Применение метода расщепления (дробных шагов) для решения задач математической физики.— В кн.: Некоторые вопросы вычислительной и прикладной математики.— Новосибирск: Наука, 1966.
  27. Цикритзис Д., Бернстейн Ф. Операционные системы.— М.: Мир, 1977.
  28. Шулепов Н. И. Диалого-пакетная система для многомашинного комплекса ЭВМ БЭСМ-6 (ОС ДИАПАК).— Ж. Вопросы атомной науки и техники. Серия: методики и программы численного решения задач математической физики, вып. I.— М.: ЦНИИАтоминформ, 1979.
  29. Мазный Г. Л. Программирование на БЭСМ-6 в системе «Дубна».— М.: Наука, 1978.
  30. Bussari G., Fattori G., Mongini — Tamagnini C. CARONTE — the Euratom System for Automatic Control of Linked Calculations / Proceedings of Conference on the effective use of Computers in Nuclear Industry.— Knoxville, Tennessee, 1969.
  - 30а. Mongini — Tamagnini C. CARONTE — the Euratom Modular Calculational System.— Newsletter of the ENEA Computer Programm Library, 1971, v. 11.



31. Bayard J. P., Boudet R. The CODNUC System,— a Modular System of Subroutines for Reactor Calculations / Proceedings of Conference on the effective use of Computers in Nuclear Industry.— Knoxville, Tennessee, 1969. Bayard J. P. CODNUC — a Modular System for Reactor Calculations. Newsletter of the ENEA Computer Programme Library, v. 11, 1971.
32. Watanaba T., Arai K., Noda T. Hitachi Nuclear Codes Control System, NCCS / Proceedings of Conference on the effective use of Computers in Nuclear Industry.— Knoxville, Tennessee, 1969.
33. Honeck H. C., Suich J. E., Jensen J. C. e. a. JOSHUA — a Reactor Physics Computational System / Там же.
34. The Argonne Reactor Computation (ARC) System.— ANL-7332, 1967. The System Aspects and Interface Data Sets of the Argonne Reactor Computation (ARC) System.— ANL-7741, 1972.
35. Александров А. П. Атомная энергетика и научно-технический прогресс.— М.: Наука, 1978.
36. Глестон С., Бал Д. Теория ядерных реакторов.— М.: Атомиздат, 1974.
37. Вычислительные методы в физике реакторов/Сборник статей.— М.: Атомиздат, 1972.
38. Смелов В. В. Лекции по теории переноса нейтронов, изд. 2.— М.: Атомиздат, 1978.
39. Марчук Г. И. Численные методы расчета ядерных реакторов.— М.: Атомиздат, 1958.
- 39а. Марчук Г. И. Численные методы в теории переноса нейтронов.— М.: Атомиздат, 1971.
40. Самарский А. А., Андреев В. Б. Разностные методы для эллиптических уравнений.— М.: Наука, 1976.
41. Самарский А. А., Николаев Е. С. Методы решения сеточных уравнений.— М.: Наука, 1978.
42. Бажинов И. К., Ястребов В. Д. Навигация в совместном полете космических кораблей «Союз» и «Аполлон».— М.: Наука, 1978.
43. Бебенин Г. Г., Скребушевский Б. С., Соколов Г. А. Системы управления полетом космических аппаратов.— М.: Машиностроение, 1978.
44. Брандин В. Н., Разоренов Г. Н. Определение траектории космических аппаратов.— М.: Машиностроение, 1978.
45. Шебшаевич В. С. Введение в теорию космической навигации.— М.: Советское радио, 1971.
46. Джеральд У. Эбкер. Роль вычислительного комплекса, работающего в реальном масштабе времени, при осуществлении проекта «Аполлон» / Доклад на Международном симпозиуме по космической науке и технике, Токио, 1968 / Перевод под ред. Платонова А. К. — М.: ИПМ АН СССР, 1968.
47. Ершов А. П. Технология разработки систем программирования.— В кн.: Системное и теоретическое программирование.— М.: ВЦ СО АН СССР, 1972.
48. Отладка систем управляющих алгоритмов. ЦВМ реального времени / Под ред. Липаева В. В.— М.: Советское радио, 1974.

49. Ершов А. П. Программирование за рубежом.— В кн.: Труды II Всесоюзной конференции по программированию, вып. 2.— Новосибирск: ВЦ СО АН СССР, 1970.
50. Самарский А. А. Общие вопросы модульного программирования для задач математической физики.— В кн.: Тезисы докладов Международной конференции «Структура и организация пакетов».— Тбилиси, «Мецниераба», 1976.
51. Яненко Н. Н. Проблемы математической технологии.— В кн.: Тезисы докладов Международной конференции «Структура и организация пакетов».— Тбилиси: «Мецниераба», 1976.
- 51а. Яненко Н. Н., Коновалов А. Н. Технологические аспекты численных методов математической физики.— Acta Universitatis Carolinae, Mathematica et Physica, 1974, 1—2.
52. Мясников В. А. Совершенствование технологии программирования — важнейшая народнохозяйственная задача.— УС и М, 1980, 1.
53. Софронов И. Д. Оценка параметров вычислительной машины, предназначенной для решения задач механики сплошной среды.— В кн.: Численные методы механики сплошной среды, 6, 3.— Новосибирск: ВЦ СО АН СССР, 1975.
54. Сергиенко И. В. Методы организации вычислительного процесса на вычислительных машинах.— Киев: ИК АН УССР, 1971.
55. Ершов А. П. Введение в теоретическое программирование (беседы о методе).— М.: Наука, 1977.
56. Карпов В. Я. Алгоритмический язык фортран.— М.: Наука, 1976.
57. Бочкова З. Ф. и др. Руководство по работе с программами на автокоде БЭМШ.— М.: ИПМ АН СССР, 1974.
58. Морозова Л. Б. Математическое обеспечение БЭСМ-6. Редактор Внешних Связей — РВС. Инструкция.— М.: ИПМ АН СССР, 1970.
59. Донован Дж. Системное программирование.— М.: Мир, 1975.
- 59а. Лебедев В. Н. Введение в системы программирования.— М.: Статистика, 1975.
60. Единая система ЭВМ. Операционная система. Редактор связей. Руководство программиста. — М., 1976.
61. Бокова И. Д. и др. Операционная система ДИСПАК для БЭСМ-6 (пользователю). Вып. I.— М.: ИПМ АН СССР, 1973.
62. Корякин В. К., Соколов В. П. Символьный отладчик в ОС ДИАПАК.— Ж. Вопросы атомной науки и техники. Серия: «Методики и программы численного решения задач математической физики», 1979, 1.
63. Карначук В. И., Позникова Г. Е., Шустов Г. В. Терминальная отладка программ в мониторинной системе «Дубна».— Новосибирск: ВЦ СО АН СССР, 1976.
64. Средства отладки больших систем / Под ред. Растина Р., перевод.— М.: Статистика, 1977.
65. Власов А. А. Макроскопическая электродинамика.— М.: ГИТТЛ, 1955.

66. Самарский А. А. Математическое моделирование и вычислительный эксперимент.— Вестник АН СССР, 1979, 5.
67. Остроумов П. Н. Моделирование трехмерной динамики частиц в установках формирования и ускорения сильнооточных ионных пучков.— М.: ИЯИ АН СССР, 1980.
68. Самарский А. А., Попов Ю. П. Разностные схемы газовой динамики.— М.: Наука, 1975.
- 68а. Головинин В. М., Самарский А. А., Фаворский А. П. Об искусственной вязкости и устойчивости разностных уравнений газовой динамики.— М.: Препринт № 70 ИПМ АН СССР, 1976.
69. Коновалов А. Н., Яненко Н. Н., Модульный принцип построения программ как основа создания пакета прикладных программ решения задач механики сплошной среды.— В кн.: Комплексы программ математической физики. Новосибирск: ВЦ СО АН СССР, 1972.
70. Марчук Г. И. Методы вычислительной математики.— М.: Наука, 1977.
71. Легоньков В. И., Петров А. А. Некоторые общие вопросы разработки и эксплуатации больших программ для счета задач математической физики.— В кн.: Комплексы программ математической физики.— Новосибирск: ВЦ СО АН СССР, 1972.
72. Горбунов-Посадов М. М., Корягин Д. А., Красотченко В. В. Реализация системного наполнения пакета прикладных программ САФРА (версия 2.0).— М.: Препринт ИПМ АН СССР № 186, 1979.
73. Roberts K. V. An Introduction to the OLYMPUS system.— Computer phis. commun. 7, 1974.
74. Андрющенко Н. В., Банных В. И., Легонькова Г. С. Вопросы экономизации больших программ, создаваемых на базе мониторной системы «Дубна».— В кн.: Численные методы механики сплошной среды, 8, 3, 1977.— Новосибирск: ВЦ СО АН СССР, 1977.
75. Старостина А. А. Выдача результатов в виде функций от времени.— В кн.: Тезисы докладов участников семинара по проблемам решения больших задач на ЭЦВМ.— Новосибирск: ВЦ СО АН СССР, 1971.
- 75а. Карпечина З. С., Легоньков В. И., Старостина А. А. Выдача результатов при счете задач математической физики.— В кн.: Численные методы механики сплошной среды, 8, 3, 1977.— Новосибирск: ВЦ СО АН СССР, 1977.
76. Карпов В. Я. Оформление программ, написанных на фортране.— Инструкция.— М.: ИПМ АН СССР, 1976.
77. Бобровников Б. А., Афанасьев В. Н. Синтаксически ориентированный листинг.— УС и М, 1980, 1.
78. Брандон Д. Х. Организация работ на вычислительном центре.— М.: Статистика, 1970.
79. Сакман Г. Решение задач в системе человек — ЭВМ.— М.: Мир, 1973.
80. Брукс Ф. П., мл. Как проектируются и создаются программные комплексы / Пер. с англ. Под ред. Ершова А. П.— М.: Наука, 1979.

81. Кнут Д. Искусство программирования для ЭВМ.— М.: Мир, 1971.
82. Walstow C. E., Felix C. P. A method of programming measurement and estimation.— IBM System J., 1977, 16, 1.
83. Байцёр Б. Архитектура вычислительных комплексов.— М.: Мир, 1974.
84. Zempolich B. A. Effective software management requires consideration of many factors.— Defence management J., 1975, 11, 4.
85. Гаганов П. Г., Липаев В. В. Характеристики ошибок в процессе разработок комплексов программ.— Программирование, 1976, 2.
86. Харитонов Р. П. Фонды нормативно-технической документации на службе стандартизации и научно-технической информации.— М.: Комитет станд., мер и изм. при Совмине СССР, 1971.
87. ГОСТ 19.001-77. ЕСПД. Общие положения.
88. ГОСТ 19.101-77. ЕСПД. Виды программ и программных документов.
89. ГОСТ 19.102-77. ЕСПД. Стадии разработки.
90. ГОСТ 19.103-77. ЕСПД. Обозначения программ и программной документации.
91. ГОСТ 19.104-78. ЕСПД. Основные надписи.
92. ГОСТ 19.105-78. ЕСПД. Общие требования к программным документам.
93. ГОСТ 19.106-78. ЕСПД. Правила выполнения программных документов, выполненных печатным способом.
94. ГОСТ 19.201-78. ЕСПД. Техническое задание.
95. ГОСТ 19.202-78. ЕСПД. Спецификация.
96. ГОСТ 19.401-78. ЕСПД. Текст программы.
97. ГОСТ 19.402-78. ЕСПД. Описание программы.
98. ГОСТ 19.501-78. ЕСПД. Формуляр.
99. ГОСТ 19.502-78. ЕСПД. Общие описания.
100. ГОСТ 19.503-78. ЕСПД. Руководство системного программиста.
101. ГОСТ 19.504-79. ЕСПД. Руководство программиста.
102. ГОСТ 19.505-79. ЕСПД. Руководство оператора.
103. ГОСТ 19.506-79. ЕСПД. Описание языка.
104. ГОСТ 19.601-78. ЕСПД. Общие правила дублирования, учета и хранения.
105. ГОСТ 19.602-78. ЕСПД. Правила дублирования, учета и хранения программных документов, выполненных печатным способом.
106. ГОСТ 19.603-78. ЕСПД. Общие правила внесения изменений.
107. ГОСТ 19.604-78. ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом.
108. ГОСТ 19.403-79. ЕСПД. Ведомость держателей подлинников.
109. ГОСТ 19.507-79. ЕСПД. Ведомость эксплуатационных документов.
110. ГОСТ 19.004-80. ЕСПД. Термины и определения (ПГ 604-677-78).
111. ГОСТ 19.002-80. ЕСПД. Схемы алгоритмов и программ. Обозначения условных графические (ПГ 604-32-79).
112. ГОСТ 19.003-80. ЕСПД. Схемы алгоритмов и программ. Правила выполнения (ПГ 604-33-79).
113. ГОСТ 19.404-79. ЕСПД. Пояснительная записка (ПГ 604-268-78).

114. ГОСТ 19.508-79. ЕСПД. Руководство по техническому обслуживанию (ПГ 604-267-78).
115. ГОСТ 19.301-79. ЕСПД. Порядок и методика испытаний (ПГ 604-266-78).
116. ГОСТ 19.600-74. Отчет о научно-исследовательской работе.
117. Михайлов А. П., Шараев Д. Я. Стандартная документация, сопровождающая программы задач математической физики.— Новосибирск: ВЦ СО АН СССР, 1975.
118. Воронков А. В., Карпов В. Я., Шараев Д. Я. Оформление документации программ для решения задач математической физики.— М.: ИПМ АН СССР, 1977.
119. Воронков А. В., Фридрих Ф. Вопросы подготовки информационных материалов для программ, библиотек и пакетов программ задач математической физики.— М.: ВЦ АН СССР, 1979.
120. Болтышев А. М., Козлов Н. И. Некоторые аспекты редактирования текстовой информации.— М.: ВИНТИ АН СССР, деп. № 515-78, 1978.
121. Авраменко В. С. Назначение и функциональные возможности математического обеспечения диалоговой информационной системы (МОДИС).— Ж. Вопросы атомной науки и техники. Серия: «Методики и программы численного решения задач математической физики». Вып. 1 (3), М.: ЦНИИатоминформ, 1979.
122. Уолш Д. Руководство по созданию документации для математического обеспечения.— М.: Наука, 1975.
123. Стукало А. С. Некоторые требования к пакетам прикладных программ (обзор и анализ).— В кн.: Труды IV Всесоюзного семинара по комплексам программ математической физики.— Новосибирск: ВЦ СО АН СССР, 1976.
124. Тыугу Э. Х. Система модульного программирования для ЭВМ «Минск-22». Общее описание.— Таллин, 1970.
125. Бокова И. Д. и др. Операционная система ДИСПАК для БЭСМ-6.— М.: ИПМ АН СССР, 1973.
126. Труды IV Всесоюзного семинара по комплексам программ математической физики / Под ред. Яненко Н. Н.— Новосибирск: ВЦ СО АН СССР, 1976.
- 126а. Комплексы программ математической физики (Материалы V Всесоюзного семинара по комплексам программ математической физики) / Под ред. Яненко Н. Н.— Новосибирск: Институт теоретической и прикладной механики СО АН СССР, 1978.
127. Воронков А. В. и др. Система обеспечения комплексов программ математической физики.— М.: Препринт № 49 ИПМ АН СССР, 1979.
128. Козловский С. Э. Применение системы программирования в модульном программировании. В кн.: Комплексы программ математической физики (материалы V Всесоюзного семинара по комплексам программ математической физики).— Новосибирск: Ин-т теор. и прикл. мех. СО АН СССР, 1978.
129. Ильин В. П. Об одном варианте системы модульного программирования.— Новосибирск: Препринт № 9 ВЦ СО АН СССР, 1976.

130. Воронков А. В. и др. Проект представления модули прикладной программы.— Новосибирск: ВЦ СО АН СССР, 1975.
131. Глушков В. М. и др. Человек и вычислительная техника.— Киев, Наукова Думка, 1971.
132. Зизин М. Н. и др. Автоматизация реакторных расчетов.— М.: Атомиздат, 1974.
133. Басс Л. П., Гермогенова Т. А., Хмылев А. Н. Модульная структура системы программ для решения осесимметричных задач теории переноса.— В кн.: Труды III семинара по Комплексам программ математической физики.— Новосибирск: ВЦ СО АН СССР, 1973.
134. Гайфулин С. А. Карпов В. Я., Мищенко Т. В. САФРА. Функциональное наполнение. Система OLYMPUS.— М.: Препринт № 27 ИПМ АН СССР, 1980.
135. Карпов В. Я., Красотченко В. В. Задачи информационного обеспечения вычислительного эксперимента. М.: Препринт № 127 ИПМ АН СССР, 1979.
136. Горбунов-Посадов М. М., Хиздер Л. А. Система ОХРА. Основные понятия. Базовые операции.— М.: Препринт № 67 ИПМ АН СССР, 1979.
137. Горбунов-Посадов М. М., Хиздер Л. А. Работа с регионами и файлами в системе ОХРА.— М.: Препринт № 81 ИПМ АН СССР, 1979.
138. Павлов Б. М., Пасконов В. М. О проекте системы обслуживания алгоритмами Аэрогидромеханики.— В кн.: Комплексы программ математической физики.— Новосибирск: ВЦ СО АН СССР, 1972.
139. G h a n e m S. B. Computing center optimization by a pricing-priority policy.— IBM System, J., 1975, 4, 3.
140. Шалфеев А. Д. Выбор функциональной структуры и исследование средств, обеспечивающих режим общих ресурсов в рамках многомашиного вычислительного комплекса.— Ж. Вопросы атомной науки и техники. Серия: «Методики и программы численного решения задач математической физики», вып. 1979, 3 (5).
- 140а. Опарин А. А., Бартенев Ю. Г., Кривов Е. С. Программные средства, предоставляемые пользователям для организации параллельных вычислений на системе машин БЭСМ-6.— Там же.
141. Энслоу Ф. Г. Мультипроцессорные системы и параллельные вычисления.— М.: Мир, 1976.
- 141а. Мультипроцессорные вычислительные системы / Под ред. Хетагурова Я. А.— М.: Энергия, 1971.
142. Петренко А. К., Усов С. А. Диалоговый монитор — ДИМОН. Запуск задач в пакет.— М.: Препринт № 69 ИПМ АН СССР, 1975.
143. Ершов А. П. О человеческом и эстетическом факторах в программировании.— Кибернетика, 1972, 5.
144. Стреттон Дж. А. Теория электромагнетизма.— М.: Гостехиздат, 1948.
145. Dennis J., B. Modularity.— Lecture Notes in Computer Science, 1975.
146. Ершов А. П. и др. Руководство по использованию системы АЛЬФА.— Новосибирск: Наука, 1968.

147. П о в е щ е н к о Ю. А., П о п о в Ю. П. Текон-пакет программ для решения тепловых задач.— М.: Препринт № 65 ИПМ АН СССР, 1978.
148. Г р е б е н н и к о в Е. А. и др. Топология вычислительного комплекса, организуемого на базе электронно-вычислительных машин БЭСМ-6.— М.: ИТЭФ-134, 1977.
149. Т и х о н о в А. Н. и др. О многоцелевой проблемно-ориентированной системе обработки результатов экспериментов.— М.: Препринт № 142 ИПМ АН СССР, 1976.
150. Т и х о н о в А. Н., С а м а р с к и й А. А. и др. Нелинейный эффект образования самоподдерживающегося высокотемпературного электропроводного слоя газа в нестационарных процессах магнитной гидродинамики.— ДАН СССР, 1967, 173, 4.
151. Е р ш о в А. П. Математическое обеспечение четвертого поколения.— Новосибирск: Препринт ВЦ СО АН СССР, 1971.

## ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

- Адреса абсолютные 30  
— внешние 30  
— внутренние 30  
Архитектура вычислительной системы 19
- Библиотека модулей загрузки 30  
Блок 18
- Внедрение 168  
Время решения календарное 18  
— коммерческое 220
- Данные исходные 130
- Задание техническое 160, 163  
Задача большая 18  
—, решаемая в реальном времени 24  
— тестовая 136  
Записка докладная 160, 161  
— пояснительная 160, 168
- Квантование 25  
Комплекс программный 18, 23, 31  
Конвейерная обработка 26
- Листинг 133
- Модель дискретная 153  
— математическая 146, 149  
— физическая 146, 149  
Модуль сборки 175  
Модуль 17, 18  
Мультипрограммирование 27
- Обеспечение серийное 18  
— штатное 18  
Обработка конвейерная 26  
Отладка 113, 136  
— алгоритма 115, 136  
— арифметическая 114, 136  
— диалоговая 136
- Пакетный режим 25  
Пакеты прикладных программ 35, 170
- План отладки 108, 124  
Пояснительная записка 160, 168  
Программа 18  
— объектная 124  
Программирование 155  
Программный комплекс 18, 23, 31  
Проект рабочий 160, 168  
— технический 166  
— эскизный 160, 166
- Режим интерактивный 25  
— пакетный 25
- САФРА 138  
Сборка модулей 175  
Связи информационные 122, 155, 179  
— логические 123  
— управляющие 123, 179  
Сегментация 108  
Система вычислительная 32  
Сопровождение 139  
Стандарты ЕСПД 156  
Схема отладки 116  
— счета 116
- Тест несвязанный 136  
— связанный 136  
Тестовая задача 136  
Трассировка 22, 136
- Формат выдач 131
- Хранение результатов 133
- Штатное обеспечение 18
- Эскизный проект 160, 166  
Этапы вычислительного эксперимента 117
- Язык машинно-зависимый 33  
— проблемно-ориентированный 33  
— программирования 21  
— процедурно-ориентированный 33



## ОГЛАВЛЕНИЕ

Предисловие . . . . .	3
Г л а в а 1. Программное обеспечение вычислительных систем и задачи . . . . .	9
§ 1. О вычислительном эксперименте . . . . .	10
§ 2. Поколения прикладных задач и программного обеспечения . . . . .	18
Г л а в а 2. Пример задачи вычислительного эксперимента из области нейтронной физики . . . . .	37
§ 1. Источники ядерной энергии . . . . .	38
§ 2. Ядерные взаимодействия и сечения . . . . .	40
§ 3. Физические основы ядерных реакторов . . . . .	44
§ 4. Математическая постановка задачи . . . . .	53
§ 5. Приближенные методы решения и организация вычислительного процесса . . . . .	57
Г л а в а 3. Пример задачи, решаемой в реальном времени . . . . .	71
§ 1. Баллистическое обеспечение космических полетов . . . . .	72
§ 2. Организация обслуживания полетов с помощью наземных средств . . . . .	89
§ 3. Некоторые особенности организации программного обеспечения задачи . . . . .	92
Г л а в а 4. Технологические аспекты вычислительного эксперимента . . . . .	96
§ 1. Особенности программирования . . . . .	97
§ 2. Технологические этапы вычислительного эксперимента . . . . .	117
§ 3. Вопросы организации и информационного обеспечения разработок . . . . .	139
Г л а в а 5. Пакеты прикладных программ . . . . .	170
§ 1. Принципы организации пакетов прикладных программ . . . . .	171
§ 2. Примеры пакетов прикладных программ . . . . .	193
Г л а в а 6. Вопросы организации работы вычислительных центров и их связь с реализацией вычислительного эксперимента . . . . .	213
§ 1. Подразделения вычислительных центров и их функции . . . . .	214
§ 2. Характеристики вычислительных центров и их влияние на проведение вычислительного эксперимента . . . . .	219
Литература . . . . .	230
Предметный указатель . . . . .	239

85 к.

