РГАСНТИ 27.47.21 ISSN 0236—3127

ИТОГИ НАУКИ И ТЕХНИКИ

ВЫЧИСЛИТЕЛЬНЫЕ НАУКИ

Том 8

Москва 1991

ГОСУДАРСТВЕННЫЙ КОМИТЕТ СССР
ПО НАУКЕ И ТЕХНОЛОГИЯМ

АКАДЕМИЯ НАУК СССР

ВСЕСОЮЗНЫЙ ИНСТИТУТ НАУЧНОЙ И ТЕХНИЧЕСКОЙ ИНФОРМАЦИИ
(ВИНИТИ)

ИТОГИ НАУКИ И ТЕХНИКИ

СЕРИЯ

ВЫЧИСЛИТЕЛЬНЫЕ НАУКИ

Том 8

Научный редактор
академик В. А. Мельников

Серия издается с 1989 г.

МОСКВА 1991

УДК 519.68

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ: ТЕОРИЯ И АЛГОРИТМЫ

Н.Н. Кузюрин, М.А. Фрумкин

Введение

Развитие современной микроэлектронной техиологии и создание высокопроизводительных многопроцессорных ЭВМ стимулировало разработку параллельных алгоритмов решения различиых задач и развитие теории параллельных вычислений. Теория параллельных вычислений является базой, которая позволяет оценить предельные возможностн вычнслнтельных систем, предлагает прнципы построения параллельных алгоритмов и способы нх реалнзацни на параллельных ЭВМ, дает классификацню задач, допускающнх глубокое распараллелнванне.

Интенснвная разработка параллельных алгорнтмов началась в середнне 70-х годов. Тогда же возннкла теорня параллельных вычнслений как попытка опнсать общие методы построенння параллельных алгоритмов н объяснить некоторые трудности, прнсущие этой областн. В последние годы нитерес к параллельным вычислениям стремительно нарастал как со стороны разработки параллельных алгоритмов решення конкретных задач, так н с точки зрения осмыслення общнх закономерностей. На ежегодных крупнейшнх международных конференцнях по Computer Science - Foundations of Computer Science (FOCS) н Symposium on the Theory of Computing (STOC) существенная часть докладов обычно бывает посвящена интересным результатам, касающимся параллельных вычислений. Кроме того, ежегодню проводятся спецнальные международные конференции по параллельным вычнсленням, за рубежом издается несколько журналов только по этой тематике, а в большинстве журналов по алгоритмам, сложности вычислений, днскретной математике, графам и комбинаторнке публнкуются статьи по параллельным алгоритмам. Несмотря на то, что огромное чнсло работ в области параллельных вычислений носят прнкладной характер и орнентнрованы на практическую реализацню для существующих параллельных архнтектур, имеется и большое число теоретических работ с серьезнымн математическими результатами. Ряд трудных задач в этой области был решен буквально

в последние 2 – 3 года. Поэтому имеет смысл подвести некоторые предварительные итоги и попытаться охарактеризовать, хотя бы в краткой форме, основные достижения, описать связи между различными моделями параллельных вычислений, выделить нерешенные задачи.

Целью настоящего обзора является изложение основных понятий, методов и результатов теории параллельных вычислений. Работа состоит из трех частей. В первой части приводятся основные определения моделей параллельных вычислений и соотношения между ними, определения класса задач, допускающих эффективное распараллеливание (класса NC) и класса задач наиболее трудных для распараллеливания (P–полных задач), понятие вероятностных (рандомизированных) параллельных алгоритмов и класса задач, эффективно решаемых этими алгоритмами (класс RNC). В терминах теории графов формулируется задача моделирования одной параллельной архитектуры на другой и задача оптимальной маршрутизации, возникающие при работе с параллельными архитектурами. Рассматривается задача о моделировании наиболее распространенного представления параллельной машины – PRAM на более реалистичной модели параллельных вычислений – сети процессоров и связанная с ней задача моделирования параллельной памяти.

Во второй части приводятся некоторые общие методы распараллеливания последовательных алгоритмов и построения параллельных алгоритмов. Параллельные алгоритмы, как и многие другие эффективные алгоритмы, строятся искусным сочетанием ряда приемов. В ядре алгоритма обычно лежат некоторые математические тождества и соотношения. На базе этих тождеств с помощью таких алгоритмических и программных конструкций, как "разделяй и властвуй", рекурсия, префиксная техника, матричные методы, специальные структуры данных строятся быстрые параллельные алгоритмы. В в сжатом виде приводится ряд известных результатов о параллельной сложности решения задач в следующих областях: алгебра, целочисленная арифметика, ряды и многочлены, комбинаторика, теория графов, вычислительная геометрия, сортировка и поиск, теория расписаний.

В третьей заключительной части обсуждается проблема отображения параллельных PRAM–алгоритмов на реальные параллельные архитектуры, подводятся некоторые общие итоги и формулируется ряд нерешенных проблем в теории параллельных вычислений.

Обзор можно использовать как справочник по конкретным па-

раллельным алгоритмам в различных предметных областях. Авторы надеются также на то, что обзор будет содействовать более глубокому ознакомлению советских математиков с понятиями теории параллельных вычислений и привлечет внимание к нерешенным проблемам, что может содействовать развитию этой области науки как у нас в стране, так и за рубежом.

Часть 1. Основные понятия теории параллельных вычислений

1.1. Модели параллельных вычислений

Интуитивное представление человека о параллельных вычислениях имеет много различных формализаций, которые реализуются в различных моделях параллельных вычислений. Приведем далеко не полный список таких моделей:

1. *Параллельная машина с произвольным доступом к памяти* – PRAM;
2. *Сети процессоров с локальной памятью*;
3. *Мультикаскадные сети типа процессоры–память*;
4. *Булевы схемы*;
5. *Булевы схемы с неограниченной степенью входа элементов*;
6. *Сверхбольшие интегральные схемы – СБИС* (VLSI);
7. *Систолические вычислители*;
8. *Векторные машины*;
9. *Машина Тьюринга*;
10. *Альтернирующая машина Тьюринга*;
11. *Машина с управлением потоком данных* (data flow).

Читатель вынужден смириться со столь вольным использованием теоретиками зарезервированных в технике названий и не должен прибегать к буквальной подстановке известных ему технических понятий.

В литературе при описании параллельных алгоритмов чаще всего используется параллельная машина с произвольным доступом к памяти – PRAM (Parallel Random Access Machine) [120],[116], являющаяся удобной и простой моделью для записи параллельных алгоритмов. Благодаря абстрагированию от многих специфических осо-

бенностей параллельных архитектур, PRAM позволяет достаточно просто реализовывать параллельные алгоритмы, не отвлекаясь на технические детали, возникающие при работе с реальными параллельными архитектурами, и обладает в силу этого свойствами универсальной параллельной ЭВМ. PRAM можно считать эталонной моделью абстрактной параллельной ЭВМ. Она состоит из p идентичных RAM (называемых процессорами), имеющих общую память. Определение RAM и все сведения о ее работе можно найти в [2]. Процессоры работают синхронно и каждый из них имеет возможность переписать в свой сумматор содержимое любой ячейки памяти. Команды одноадресные, локальной памяти данных нет за исключением одного сумматора в каждом процессоре. Как и в [2], будем считать, что каждый процессор работает по своей программе, хранящейся в локальной памяти, и может выполнять следующие операции:

Load x – загрузить операнд из ячейки x памяти в сумматор,
Store x – отослать содержимое сумматора в ячейку x памяти,
Add x (Sub x) – сложить(вычесть) содержимое ячейки x и сумматора,
Mult x (Div x) – умножить (разделить) содержимое сумматора и ячейки x,
Jump M – переход (безусловный) на команду с меткой M,
JGTZ M – переход на команду с меткой M по условию "содержимое сумматора больше 0",
JZERO M – переход на команду с меткой M по условию "содержимое сумматора равно 0",
Read x – считывание в ячейку x очередного символа из входной ленты,
Write x – запись содержимого x на выходную ленту,
Halt – останов,
Wait – ожидание (процессор никаких действий не производит).

Операнды могут быть одного из следующих типов:
#x – означает целое число x,
i – означает содержимое i-й ячейки памяти,
*i – означает содержимое ячейки памяти, номер которой записан в ячейке i (косвенная адресация).

В PRAM имеется одна общая для всех процессоров входная лента. Каждый процессор может считывать содержимое любой ячейки этой ленты независимо от других процессоров с помощью команды Read. У каждого процессора имеется также одна выходная лента, на

которую он может записывать содержимое сумматора с помощью команды Write. Память является общей для всех процессоров и состоит из потенциально бесконечного числа ячеек, в каждой из которых может быть записано произвольное целое число. Программа для PRAM представляет собой объединение программ для каждого отдельного процессора. Программа каждого процессора представляет собой последовательность команд из перечисленного выше списка.

Вычисление осуществляется синхронно всеми процессорами, начиная с первой команды. Если в вычислении встретится команда деления на 0, то считается, что результат вычисления не определен. Вычисление заканчивается после того, как каждый процессор выполнит команду Halt. Результатом вычисления считается набор из p последовательностей чисел на выходных лентах.

Время вычисления на PRAM равно максимуму из времен вычислений на каждом процессоре. Напомним, что в качестве времени вычисления на одном RAM-процессоре обычно используется одна из двух мер сложности: *равномерная* и *логарифмическая*. В первом случае время выполнения каждой команды полагается равным единице и время работы программы равно просто общему числу выполненных команд. Во втором случае время выполнения команды полагается по порядку равным логарифму величины максимального операнда, а общее время работы равно сумме времен выполнения всех выполненных команд. Мы будем в дальнейшем использовать равномерную меру, полагая при этом, что величины всех операндов не превосходят некоторого полинома от длины входа. Это ограничение необходимо для сохранения адекватности (полиномиальной эквивалентности) RAM-вычислений и вычислений на других канонических моделях (машине Тьюринга и т.п.).

PRAM удобна тем, что наиболее полно отражает черты реальных параллельных ЭВМ, абстрагируясь от технических ограничений. PRAM, конечно, не может рассматриваться как физически реализуемая модель, поскольку при большом числе процессоров и ячеек памяти становится невозможно за константное время осуществить выборку на любой процессор числа из произвольной ячейки памяти. Это ограничение учитывается в рассматриваемой далее в этом разделе модели *сети процессоров*, состоящих из процессоров, соединенных коммуникационной сетью.

Различают три типа PRAM в зависимости от того, что происходит, когда несколько процессоров одновременно обращаются к одной

ячейке памяти. В модели EREW PRAM (exclusive read – exclusive write) одновременное обращение (запись или считывание) к одной и той же ячейке несколькими процессорами запрещено. Это – самая слабая модель PRAM. В модели CREW PRAM (concurrent read – exclusive write) нескольким процессорам разрешается одновременное чтение из одной ячейки памяти, но не разрешается одновременная запись в одну и ту же ячейку памяти. В наиболее сильной модели CRCW PRAM (concurrent read – concurrent write) разрешается как параллельное считывание из одной ячейки несколькими процессорами, так и параллельная запись из нескольких процессоров в одну ячейку.

В зависимости от способа разрешения конфликта по записи в ячейку памяти несколькими процессорами выделяются следующие типы CRCW PRAM, перечисленные в порядке не убывания вычислительных возможностей:

WEAK CRCW – в этой модели разрешается нескольким процессорам записывать в одну и ту же ячейку только число 0.

COMMON CRCW – в этой модели разрешается нескольким процессорам записывать в одну и ту же ячейку памяти одинаковое число.

ARBITRARY CRCW – в этой модели процессоры могут записывать в одну и ту же ячейку разные числа, а содержимое ячейки становится одним из этих чисел, но не известно каким. При этом при различном исполнении результат может быть различным.

PRIORITY CRCW – результатом записи в одну и ту же ячейку является значение, поступившее от процессора с большим номером.

STRONG CRCW – результатом записи в одну и ту же ячейку является максимальное из записываемых чисел.

Отметим, что каждая следующая модель PRAM может моделировать любой шаг предыдущей за константное число шагов. В то же время можно показать, что каждый шаг самой сильной из этих моделей STRONG CRCW PRAM с p процессорами моделируется log p шагами самой слабой модели EREW PRAM с p процессорами [116].

Приведем несколько результатов о моделировании одних типов PRAM другими. Первый из них относится к моделированию PRAM с большим числом процессоров посредством PRAM с меньшим числом процессоров [73]:

Теорема 1.1. *Если параллельное вычисление может быть выполнено за время* t *и использует* N *операций на* PRAM *данного типа (с произвольным числом процессоров), то оно может быть выполнено за*

время $t + (N - t)/p$ *на* PRAM *того же типа с* p *процессорами.*

Справедливость утверждения вытекает из того факта, что каждый параллельный шаг i, содержащий N_i операций, может быть промоделирован за время N_i/p путем объединения независимых операций в блоки размера не более p и выполнения всех операций одного блока за один шаг.

Число процессоров исходной PRAM может быть любым, но обычно эта теорема используется, когда это число превосходит число процессоров моделирующей PRAM.

Следующий результат связывает время вычисления на самой сильной модели PRAM с временем вычисления на самой слабой модели [247], [325]

Теорема 1.2. *Параллельное вычисление, которое может быть выполнено за время* t *на* p*-процессорной* STRONG CRCW PRAM, *может быть выполнено за время* $O(t \log p)$ *на* p*-процессорной* EREW PRAM.

Приведем идею доказательства. Каждый шаг STRONG CRCW PRAM моделируется $O(\log p)$ шагами EREW PRAM. При чтении из памяти каждый процессор EREW записывает в специальный массив пару чисел номер процессора и номер ячейки памяти, из которой осуществляется считывание. При запись в память каждый процессор EREW записывает в специальный массив тройку чисел: номер процессора, адреса в памяти и значение записываемого числа.

Затем они сортируются лексикографически по их содержимому. Как мы покажем в дальнейшем параллельная сортировка p чисел на p процессорной EREW PRAM может быть выполнена за время $O(\log p)$ (см. раздел 2.2.7.). При записи в память каждый процессор читает сначала тройку чисел, которой он приписан, и следующую тройку. Если адрес в следующей тройке отличается от адреса, приписанного процессору, процессор записывает в память значение из его тройки по адресу из его тройки. Упорядоченность списка гарантирует, что только процессор с наибольшим значением для данного адреса преуспеет при запись. Сходным образом осуществляется и моделирование считывания из памяти.

Таким образом каждый шаг исходной STRONG CRCW PRAM моделируется на EREW PRAM за время $O(\log p)$.

Приведем теперь результат о моделировании одних моделей PRAM другими с тем же временем, но с большим числом процессоров [206]

Теорема 1.3 *Параллельное вычисление, которое можно выпол-*

нить за время t *на* p*–процессорной* STRONG CRCW PRAM *может быть выполнено за такое же по порядку время на* WEAK CRCW PRAM *с* $O(p^2)$ *процессорами.*

Доказательство Для каждой пары исходных процессоров p_i и p_j добавляем процессор q_{ij} и записываем 1 в f_i – i–ю ячейку дополнительной памяти ($i = 1,\ldots,p$). Если процессор p_i хочет записать значение v_i по адресу a_i, он записывает v_i и a_i в ячейки дополнительной памяти, откуда все процессоры q_{ij} считывают их. Каждая пара процессоров q_{ij} сравнивает затем их адреса и содержимое, то есть пары (v_i, a_i) и (v_j, a_j). Если $a_i \neq a_j$, ничего не делается. В противном случае сравниваются значения v_i и v_j, и если $v_i \leq v_j$ при $i < j$, то процессор q_{ij} записывает 0 в f_i. Затем все процессоры p_i, у которых значение f_i остается равным единице, осуществляют запись в память.

Нижние оценки для PRAM. Известны результаты о разделении вычислительной силы различных типов PRAM, которые опираются на нижние оценки времени вычисления на PRAM. В [99] показано, что дизъюнкцию n булевых переменных можно вычислить на COMMON CRCW PRAM с n процессорами за константное время , но для вычисления ее на n процессорной CREW PRAM требуется время $\Omega(\log n)$. Известно также, что [295] поиск по ключу в отсортированной таблице размера n может быть выполнен на CREW PRAM с p процессорами за время $O(\log n/\log p)$, но требует времени $\Omega(\log n - \log p)$ на EREW PRAM с p процессорами. В [249] дана точная характеризация времени, требуемого идеальной CREW PRAM для вычисления булевой функции f от n переменных. Для данного булева вектора w длины n и подмножества S индексов из $\{1,2,\ldots,n\}$ скажем, что f является чувствительной к S на w, если значение f изменяется при изменении компонент w с номерами из S. Скажем, что f k–чувствительна на w, если f чувствительна на w к каждому из k различных индексных множеств. Чувствительность bs(f) определяется как наибольшее k, для которого на некотором w f является k–чувствительной. В [249] показано, что время, требуемое для вычисления булевой функции f на идеальной CREW PRAM, ограничено сверху и снизу величинами вида $c \log bs(f) + d$, где c и d – некоторые константы.

Отметим, что при доказательстве нижних оценок обычно используется модель так называемой идеальной PRAM, в которой отсутствуют ограничения на вычислительную силу отдельных процессоров и предполагается, что каждый процессор за единичное время

может вычислить любую функцию от аргументов, хранящихся в локальной памяти этого процессора. При этом предполагается, что исходные данные хранятся вначале в глобальной памяти. Нижние оценки, полученные для такой модели, имеют очень большую общность, поскольку не зависят от конкретного набора команд процессоров PRAM. Приведенные выше нижние оценки получены как раз для идеальных моделей PRAM.

Известны результаты и о разделении вычислительной силы различных типов CRCW PRAM с заданным числом процессоров n и при ограничении на размер используемой памяти [118], [119]. Основной вывод, вытекающий из этих результатов, заключается в том, что при ограничениях на доступную память все рассмотренные модификации CRCW PRAM различаются по своим вычислительным возможностям. В произвольном случае, при возможности использования неограниченной памяти , все типы CRCW PRAM могут моделировать друг друга с константным замедлением, если в моделирующей машине разрешается использовать n^2 процессоров. Трудности получения нижних оценок для CRCW PRAM с неограниченным размером используемой памяти и числом процессоров показывает и тот факт, что любую булеву функцию от n переменных можно вычислить за константное время на CRCW PRAM с $n2^n$ процессорами. В этом можно убедиться, рассматривая представление булевой функции в виде дизъюнктивной нормальной формы и проводя вычисления всех ее членов параллельно.

Поэтому естественно при получении нижних оценок число процессоров ограничивать полиномом . При таком ограничении для идеальной PRIORITY CRCW PRAM получены нижние оценки вида $\Omega(\log n/\log\log n)$ для задачи суммирования n битов по модулю два [56] и для других задач, сводящихся к ней (вычисление префиксных сумм, ранжирование списка, задача о стягивании дерева), для PRIORITY CRCW PRAM – оценки вида $\Omega(\log n)$ для сложения n целых чисел с полиномиальным числом бит [214]. При получении нижних оценок используются комбинаторные методы (теорема Рамсея) и понятие колмогоровской сложности.

Эффективность алгоритма для PRAM характеризуется двумя величинами – числом процессоров и временем исполнения. Нетрудно видеть, что параллельный алгоритм для PRAM с p процессорами, выполняемый за время T_p, может быть промоделирован на одном процессоре RAM за время pT_p, то есть справедливо неравенство $T_p \geq T_1/p$. Обобщением этого неравенства является соотношение

$T_p \geq qT_q/p$, справедливое при $p \geq q$ и известное как теорема Брента [73].

Для оценки того, во сколько раз быстрее задача может быть решена на p процессорах по сравнению с одним процессором, вводится величина $S_p = T_1/T_p$, называемая *ускорением*. Загруженность процессоров характеризуется отношением S_p/p, которое называется *эффективностью распараллеливания*.

Распараллеливание называется оптимальным, если для ускорения выполнено соотношение $S_p = O(p)$. Естественно при этом (по определению) эффективность распараллеливания равна O(1). В этом же смысле часто употребляют термин *оптимальный параллельный алгоритм* как алгоритм, параллельное время работы которого умноженное на число процессоров, по порядку равно времени работы последовательного алгоритма.

Идеализированность PRAM как модели параллельных вычислений заключается в наличии параллельной памяти, к которой могут одновременно обращаться несколько процессоров, и в том, что не учитывается время на обмен данными между процессорами. Иерархия PRAM по степени разрешения конфликтов в памяти – EREW, CREW, CRCW является удобным инструментом при разработке и анализе параллельных алгоритмов благодаря своей универсальности и независимости от конкретных параллельных архитектур, но не учитывает два упомянутых выше важных факта. Более реалистичной по сравнению с PRAM моделью, в которой учитывается время на обмен между процессорами и доступ к памяти является последовательным, представляются *сети процессоров с локальной памятью*.

Сеть процессоров можно представлять в виде графа, в вершинах которого находятся RAM, а ребра соответствуют каналам связи между процессорами RAM Память каждого процессора в сети отличается от памяти PRAM тем, что к ней возможен только последовательный доступ в каждый момент времени из локальной памяти данного процессора можно считать или записать только одно число. При этом возможно параллельное считывание (запись) из памяти различных процессоров В систему команд RAM добавляются команды пересылки содержимого произвольной ячейки памяти данного процессора в любую ячейку памяти другого процессора, соединенного с данным процессором каналом связи (ребром в графе) На каждом шаге каждый процессор может передавать или принимать данные только от непосредственно связанных с ним ребрами процессоров Для

обмена данными между несмежными процессорами требуются несколько шагов обмена. При этом возникает задача маршрутизации, состоящая в поиске в некотором смысле оптимальных путей передачи данных между процессорами с целью сокращения времени обмена. В общей модели считается, что через данное ребро в единицу времени может проходить только одно сообщение и при наличии конкурирующих сообщений проходит одно из них, а остальные ожидают в очереди (сеть с очередями). Иногда считается, что одновременно может осуществляться обмен данными между парами процессоров по ребрам, образующим паросочетание, т.е. каждый процессор может передавать или принимать сообщения только от одного процессора и все обменивающиеся пары должны состоять из различных вершин (обмены "один в один"). При этом очереди не образуются (так называемый бесконфликтный обмен), но задача выбора маршрутов становится сложнее, чем при маршрутизации с очередями.

Если n-вершинный граф является полным, т.е. в нем присутствуют ребра между любой парой вершин, то такая сеть процессоров обычно называется MPC (module parallel computer). Однако на практике реализовать полный граф связей из-за технологических ограничений удается только при небольших значениях n. Поэтому в качестве графов коммуникации рассматривают обычно графы с числом ребер, не превосходящим n log n. Одним из самых распространенных графов коммуникации является n-мерный булев куб, часто называемый гиперкубом. Его вершинами являются все двоичные векторы длины n, а ребрами соединяются вершины, которые отличаются только в одной координате.

Поскольку более удобной для записи параллельных алгоритмов все же является PRAM, возникает вопрос, какую цену приходится платить за переход к более реалистичной модели сети процессоров? Для сравнения силы этих моделей рассматривают задачу моделирования PRAM на сети процессоров и оценивают, с каким замедлением возможно это моделирование. Эта задача разбивается на два этапа:

1. моделирование параллельной памяти;

2. решение задачи оптимальной маршрутизации (при моделировании обмена между процессорами в полном графе на графе ограниченной степени).

Первая задача заключается в распределении данных по локальной памяти процессоров таким образом, чтобы требуемые данные всегда можно было переслать в нужный процессор за сравнительно

небольшое время (порядка логарифма числа процессоров). При этом используются несколько копий данных, которые размещаются "максимально распределенно" по процессорам. Это делается для того, чтобы избежать предельной ситуации, в которой все копии необходимых некоторому процессору данных находятся в локальной памяти другого процессора, поскольку в этом случае пересылка данных займет время, пропорциональное числу пересылаемых данных (напомним, что доступ к локальной памяти является последовательным). Формально на первом этапе каждый шаг PRAM с n процессорами и m используемыми ячейками памяти моделируется на MPC с n процессорами. В [310] показано, что такое моделирование возможно за время O(log n) с большой вероятностью, а в [41] – за время O(log m) детерминированно. При этом вероятностное моделирование является равномерным, а детерминированное – неравномерным по n (для каждого n доказывается существование соответствующего моделирования). При доказательстве возможности соответствующего моделирования [41] существенно используется конструкция, позволяющая за константное время на n процессорах с не более, чем εn ошибками осуществить разбиение n чисел на две половины, в первой из которых все числа меньше всех чисел из второй половины (конструкция ε-halver из [37]), основанная на существовании *расширителей* с заданными параметрами – специальных графов, часто используемых как для доказательства нижних оценок, так и для построения конкретных алгоритмов (см. раздел 2.2.7). Представляет интерес построение равномерного по n моделирования с аналогичным [41] замедлением.

Вторая задача – задача о моделировании MPC на сети процессоров ограниченной степени (включающая в себя решение задачи оптимальной маршрутизации при межпроцессорных обменах) решается путем использования сетей сортировки или сортирующих алгоритмов на сетях ограниченной степени [37], [210], [211], гарантирующих моделирование n-процессорной MPC на n-процессорной сети ограниченной степени с замедлением O(log n). Таким образом, замедление при общем моделировании (n,m)-PRAM на n процессорной сети ограниченной степени есть величина O(log m log n). Имеется [181] и прямое моделирование n-процессорной PRAM с m ячейками памяти на n-процессорной сети ограниченной степени: T шагов PRAM моделируется на сети за время O(T log m) с вероятностью, стремящейся к единице при n или T, стремящимися к единице. В [274] описано

достаточно простое вероятностное моделирование одного шага CRCW PRAM на сети ограниченной степени (бабочке Фурье) за время O(log n) с константным буфером в каждой вершине. Детерминированных алгоритмов моделирования с такими параметрами в настоящее время неизвестно, и построение их представляется весьма интересным и важным как с теоретической, так и с практической точек зрения. Самостоятельный интерес представляет для различных графов и задача оптимальной маршрутизации, которая подробнее будет рассмотрена в разделе 1.5.

Необходимо отметить также деление реальных многопроцессорных архитектур на два типа – SIMD (single instruction – multiple data) и MIMD (multiple instruction – multiple data) в зависимости от того, имеется ли одна глобальная для всех процессоров программа (SIMD) или каждый процессор работает по своей собственной программе (MIMD). При этом определении PRAM является MIMD–машиной, но имеется много примеров многопроцессорных систем, являющихся SIMD–машинами.

Другой моделью параллельных вычислений, получившей широкое распространение в теории параллельных вычислений, являются *равномерные семейства схем.*

Сформулируем основные определения, касающиеся булевых и алгебраических схем. Различные варианты, модификации и взаимосвязи можно найти в [67],[283],[332].

Определение. *Схемой над множеством* A *будем называть ориентированный ациклический граф, вершинам которого приписаны операции из множества* Ω, *определенные на* A. *Каждой входной дуге некоторой вершины соответствует аргумент операции, выполняемой в этой вершине. Входным вершинам соответствуют входные переменные, принимающие значения из множества* A, *выходным вершинам – выходные переменные.*

Предполагается, что арности операций из множества Ω и, следовательно, степени вершин графа ограничены.

Наибольшее практическое значение имеют *булевы схемы*, входными переменными которых являются булевы переменные

$$\Omega = \{\&, \vee, \neg\}, \qquad A = \{0,1\},$$

и *алгебраические схемы*

$$\Omega = \{\pm, *, /\}, \qquad A = k - \text{поле}.$$

Отметим, что понятие алгебраической схемы эквивалентно понятию неветвящейся программы, часто используемой для записи ал-

гебранческих алгоритмов.

Размером схемы c называется число s(c) ее вершин, а *глубиной* – d(c) – максимальная длина ориентированного путн в ее графе. Схема с n входными вершинами и m выходными вершинами вычисляет функцию f: $A^n \longrightarrow A^m$, которая определяется естественным образом по индукции.

Каждая отдельная схема вычисляет функцию с конкретным числом переменных. Для вычнсления последовательностн функций, зависящих от растущего числа аргументов, рассматрнвают семейства схем. Для задания схемы обычно используется стандартное кодирование, которое каждому элементу ставит в соответствие его номер, номера его прямых предшественников н тнп операции, выполняемой этим элементом.

Определение. *Будем называть семейство схем* $C = \{c_n\}$ *равномерным, если по номеру схемы ее стандартный код может быть построен на машине Тьюринга с использованием памяти* $O(d(c_n))$.

Часто в теорин параллельных вычислений в качестве модели вычислений нспользуются *булевы схемы с неограниченной степенью входа элементов*, в которых элементы реализуют булевы функции от неограниченного числа переменных (напрнмер, конъюнкцию н дизъюнкцию). Как будет показано в дальнейшем, вычнсления с помощью равномерных семейств булевых схем с неограниченной степенью входа тесно связаны с вычислеиннями на модели CRCW PRAM.

Ориентированной переключательной схемой называется ацикличесний орграф (в котором допускаются кратные ребра) с двумя выделенными вершинами – входом н выходом, некоторым ребрам которого приписаны снмволы нз множества $\{x_1,\ldots,x_n,\neg x_1,\ldots,\neg x_n\}$. Значение переключательной схемы на булевом векторе $(a_1,\ldots,a_n)$ равно 1, если существует орнентнрованный путь от входа к выходу в подграфе, полученном нз схемы удаленнем ребер с метками x_i, для $a_i = 0$ и $\neg x_i$ для $a_i = 1$, $i = 1,\ldots,n$. *Размером переключательной схемы* DS называется число ее ребер. Переключательная схема, у которой ребра не орнентнрованы и все помечены символами переменных лнбо нх отрнцаннй, называется *контактной схемой*.

Ветвящейся программой называется ацнклнческий орграф, (в котором допускаются кратные ребра) такой, что имеется один вход и два выхода, соответствующне значенням 0 и 1, и нз каждой вершины (кроме выходных) выходнт два ребра, помеченные противоположными символами x_i н $\neg x_i$ $i = 1,\ldots,n$. Входной булев вектор

$a = (a_1,\ldots,a_n)$ определяет естественным образом единственный путь из входа в один из выходов (по ребрам со значениями равными 1), значение которого и равно значению ветвящейся программы на булевом векторе $a = (a_1,\ldots,a_n)$. Размер ветвящейся программы ВР равен числу ее вершин минус 2, а ширина – равна ширине соответствующего частично упорядоченного множества.

Введенные сложностные характеристики – глубина и размер – связаны с традиционными памятью и временем, затрачиваемыми стандартными моделями вычислительных устройств, например *машинами Тьюринга*. Глубина схем, вычисляющей функцию f, отражает объем памяти, необходимой для вычисления f на машине Тьюринга, а размер схем – время вычисления f. Именно имеет место следующая теорема, принадлежащая К. Шнорру [16],[332],[286].

Теорема 1.4. *Если функция вычислима на машине Тьюринга за время* T(n) *с памятью* S(n), *то она вычислима равномерным семейством схем размера* O(T(n) log S(n)).

Справедлива также следующая теорема [67].

Теорема 1.5. *Если функция* f *вычислима на машине Тьюринга с использованием памяти* $S(n) \geq \log n$ *и времени* T(n), *то существует равномерное семейство схем* (c_n) *для вычисления* f, *имеющее глубину* $d(c_n) = O(S(n) \log T(n))$. *Если же* f *вычислима равномерным семейством схем глубины* $d(c)_n \geq \log n$, *то* f *вычислима на машине Тьюринга с использованием памяти, не большей* O(d(n)).

В [258],[16], а затем в [255] показано, что имеют место следующие утверждения.

Теорема 1.6.. k-*ленточная машина Тьюринга, использующая память* S(n) *и выполняющая* R(n) *поворотов головок, может быть смоделирована равномерным семейством схем глубины* $O(R(n) \log^2 S(n) \operatorname{loglog} S(n))$ *и ширины* $O(S(n)^k)$.

Теорема 1.7. k-*ленточная машина Тьюринга, затрачивающая время* T(n) *и выполняющая* R(n) *поворотов головок, может быть смоделирована равномерным семейством схем глубины* $O(R(n) \log^2 T(n) \operatorname{loglog} T(n))$ *и размера* $O(R(n)\, T(n)^k \log T(n) \operatorname{loglog} T(n))$.

Интересный открытый вопрос относится к возможности моделирования машин Тьюринга семействами схем с полиномиальной связью пар параметров времени – памяти и размера – глубины. Более точно, верно ли, что для любой булевой *функции* f *вычислимой на машине Тьюринга с использованием памяти* $S(n) \geq \log n$ *и времени* T(n), *существует равномерное семейство схем* (c_n) *для вычисления*

f, имеющее глубину $d(c_n) = Poly(S(n))$ и размер $s(c_n) = Poly(T(n))$? (poly – обозначает полином).

Имеется следующая связь памяти машины Тьюринга с размером ветвящейся программы, моделирующей ее работу на словах длины n [269], [286]:

Теорема 1.8. *Машина Тьюринга с памятью* $S(n) \geq \log n$ *может быть смоделирована равномерным семейством ориентированных переключательных схем и ветвящихся программ размера* $2^{O(S(n))}$.

Идея моделирования заключается в рассмотрении всего множества возможных конфигураций и сопоставлении их вершинам ориентированной переключательной сети. Две вершины соединяются ребром, если соответствующий переход от одной конфигурации к другой возможен на машине Тьюринга. Переменная, приписанная ребру, определяется по позиции головки машины Тьюринга и символу на ленте в конфигурации, соответствующей первой вершине. Полученную ориентированную переключательную сеть нетрудно преобразовать в ветвящуюся программу требуемого размера.

Таким образом, размер ветвящейся программы соответствует логарифмической памяти машин Тьюринга.

Приведенные факты позволяют использовать и классическую модель вычислений – машину Тьюринга для описания параллельных вычислений. Действительно, исследование большого числа моделей параллельных вычислений показало, что если не налагать никаких ограничений на число процессоров, то класс проблем, решаемых параллельными машинами за время, полиномиальное от T(n), совпадает с классом проблем, вычислимых детерминированной машиной Тьюринга с размером памяти (зоной), полиномиальным от T(n). Это было доказано в [261] для векторных машин , в [120] для PRAM без ограничений на длину записываемого в ячейку слова, в [77] для альтернирующих машин Тьюринга и в [67] для равномерных семейств булевых схем.

Все эти попытки охарактеризовать возможности параллельных вычислений через сложностные характеристики машины Тьюринга привели к формулировке *тезиса о параллельных вычислениях*:

Параллельное время полиномиально эквивалентно размеру памяти (зоны) машины Тьюринга.

Подобно известному тезису Черча тезис о параллельных вычислениях не может быть формально доказан, поскольку он связывает строго определенное понятие (память машины Тьюринга) с интуитив-

ным понятием (параллельное время). Для многих параллельных моделей этот тезис оказывается справедливым как строгое математическое утверждение. Блюм [64] заметил, что тезис о параллельных вычислениях утрачивает справедливость, если можно использовать гораздо больше, чем $2^{O(T(n))}$ процессоров, где $T(n)$ – параллельное время.

В литературе встречается также *расширенный тезис о параллельных вычислениях: параллельное время и оборудование (в различных моделях параллельных вычислений) одновременно полиномиально связаны с числом поворотов головки и памятью машины Тьюринга.*

Расширенный тезис о параллельных вычислениях также строго доказан для ряда конкретных моделей параллельных вычислений (см. [258] и теорему 1.7).

В частности, классами задач, разрешимых с логарифмической (полилогарифмической) памятью на машине Тьюринга являются классы **LOGSPACE** (**POLYLOGSPACE**). Класс **LOGSPACE** содержится в классе **P**, состоящем из задач, разрешимых за полиномиальное последовательное время. Для **POLYLOGSPACE** такое включение неизвестно, то есть в **POLYLOGSPACE** имеются задачи, для которых неизвестны последовательные полиномиальные алгоритмы решения. Поэтому рассматривается класс **SC** = **POLYLOGSPACE** ∩ **P**, состоящий из задач, разрешимых на машине Тьюринга за полиномиальное время и с полилогарифмической памятью. Хотя класс **SC** отражает в некоторой степени свойства задач, которые могут быть эффективно решены параллельно, в следующем разделе будет определен класс **NC**, прямо отражающий свойства эффективной распараллеливаемости. Необходимо отметить также гипотезу [95],[98] о несовпадении классов **SC** и **NC**. Кроме того, более удобными, по сравнению с машинами Тьюринга, для теоретического описания параллельных вычислений (наряду с PRAM) оказываются равномерные семейства булевых схем и альтернирующие машины Тьюринга [98],[77].

Альтернирующая машина Тьюринга (*ATM*) является обобщением недетерминированной машины Тьюринга. ATM имеет одну входную ленту, несколько рабочих лент и конечное число состояний, разбитых на три класса: нормальные, экзистенциальные и универсальные состояния. Из экзистенциальных и универсальных состояний ATM может переходить в два состояния. Нормальные состояния могут быть при-

нимающими, отвергающими и незаключительными. В каждый момент времени конфигурация АТМ является строкой, в которой записано: состояние АТМ, положение головки на входной ленте и положения головок на рабочих лентах и содержимое входной и рабочих лент. Вычисление на АТМ может быть представлено следующим образом:

– если состояние нормальное и незаключительное, то осуществляется переход в единственное следующее состояние в соответствии с программой,

– принимающие и отвергающие состояния являются заключительными

– находясь в экзистенциальном или универсальном состоянии, АТМ порождает несколько своих копий (в соответствии с программой), каждая из которых продолжает работу с той же конфигурации, но в других состояниях (в тех, в которые переходит АТМ в соответствии с программой). При этом исходная АТМ уничтожается.

АТМ завершает работу, когда все копии доходят до заключительного состояния (принимающего или отвергающего).

Вычисление на АТМ естественно представлять в виде дерева, вершины которого соответствуют конфигурациям АТМ, а дуги – переходам из одной конфигурации в другую в соответствии с программой АТМ. Каждый лист дерева соответствует конфигурации с принимающим или отвергающим состоянием. Временем работы АТМ называется глубина дерева вычислений, памятью – максимальный размер конфигурации в дереве, числом альтернирований – максимальное число переходов из экзистенциального состояния в универсальное и наоборот.

Результат вычисления для заданного входа определяется индуктивно следующим образом. Внутренняя вершина, в которой АТМ находилась в нормальном состоянии, называется принимающей, если ее единственный потомок в дереве является принимающим. Внутренняя вершина, в которой АТМ находилась в экзистенциальном состоянии, называется принимающей, если найдется непосредственный потомок в дереве являющийся принимающим. Внутренняя вершина, в которой АТМ находилась в универсальном состоянии, называется принимающей, если все ее непосредственные потомки в дереве являются принимающими. АТМ принимает заданный вход (значение на входе равно 1) тогда и только тогда, когда корень соответствующего дерева вычислений является принимающим.

Вычисление на альтернирующей машине Тьюринга является более

общим по сравнению с недетерминированной машиной Тьюринга (в которой имеются только экзистенциальные состояния), поскольку в недетерминированной машине Тьюринга дерево вычислений является принимающим, если найдется хотя бы один принимающий выход (дизъюнкция выходов), а в АМТ дерево вычислений является принимающим, если значение соответствующей этому дереву формулы истинно. При этом экзистенциальное состояние трактуется как дизъюнкция, а универсальное – как конъюнкция, принимающее состояние – как 1, отвергающее – как 0.

Альтернирующая машина Тьюринга является удобным инструментом для описания важных в теории параллельных вычислений сложностных классов AC^k, определяемых изначально с помощью равномерных семейств булевых схем с неограниченной степенью входа (см. следующий параграф).

В качестве модели параллельных вычислений используется также *векторная машина* [261], состоящая из набора битовых процессоров и набора регистров, которые могут содержать любое целое число. Все процессоры работают по одной программе, которая может состоять из побитовых булевых операций над содержимым регистров, условный переход по нулю, сдвиг влево или вправо содержимого одного регистра на величину, записанную в другом регистре, взятие маски. Некоторые регистры идентифицируются как входные или выходные. В каждый момент времени i-й процессор считывает i-е биты операндов, выполняет требуемую логическую операцию и записывает результат в i-ю позицию регистра-результата. В случае выполнения операции сдвига процессор передает данный бит в нужную позицию регистра-результата. Параметрами векторной машины являются число процессоров и время. В следующем параграфе будет приведена характеризация с помощью векторной машины класса эффективно распараллеливаемых задач.

В последнее время одной из наиболее изучаемых моделей параллельных вычислений оказались *систолические вычислители*, поскольку они идеально подходят для реализации с помощью технологии СБИС и очень удобны с точки зрения реализации стандартных алгоритмов (систолические массивы известны для многих десятков алгоритмов решения разнообразных задач – от линейной алгебры до САПР СБИС)

Под *систолическим процессором, или вычислителем*, понимается

сеть модулей, соединенных регулярным способом каналами обмена. Число соседей одного модуля и число типов графов соединений ограничено. Все вычисления и передачи данных по массиву модулей происходят синхронно. Каждый модуль прокачивает данные, выполняя в каждый момент времени короткие вычисления с тем, чтобы в сети сохранялся регулярный поток данных [4], [25]. Свое название систолические вычислители получили благодаря аналогии их работы с работой сердечной мышцы.

Систолические процессоры (СП) являются мощным аппаратным средством реализации стандартных алгоритмов, разработаны обширные библиотеки систолических алгоритмов. В работе [303] сравниваются возможности СП и ЭВМ типа SIMD. Здесь доказан важный факт, свидетельствующий об излишности глобальной связи по командам в ЭВМ типа SIMD. Глобальная связь по выдаче команд может быть заменена локальной пересылкой команд почти без увеличения времени решения задачи.

Теорема 1.9. *Для любой ЭВМ типа* SIMD, *имеющей* n *связанных в цепь процессоров, работающих на общую память, и выполняющей вычисления за время* T(n), *существует систолический процессор с* n *клетками, связанными в цепь, моделирующий работу* SIMD-машины *за время* $2T(n) + 3n + O(1)$.

Этот результат можно также интерпретировать как результат о конвейеризации – все команды, выполняемые в ЭВМ типа SIMD всеми процессорами одновременно, могут быть последовательно прокачены через все клетки СП, породив при этом тот же самый результат. Время $3n$ расходуется на ввод исходных данных и вывод результатов.

Подводя итоги рассмотрению некоторых наиболее часто используемых моделей параллельных вычислений, можно резюмировать следующее. С одной стороны, такие классические вычислительные модели как машина Тьюринга, альтернирующая машина Тьюринга, булевы схемы (с ограниченной и неограниченной степенью входа элементов), не использующие в своем определении специфику параллельности, позволяют тем не менее достаточно адекватно формулировать принципиальные проблемы теории параллельных вычислений. Эти модели используются, в основном, как инструмент для качественной характеризации различных классов параллельной сложности и не имеют каких-либо практических воплощений.

С другой стороны, такие модели как сети процессоров с локальной памятью, мультикаскадные сети типа процессоры-память, СБИС, систолические вычислители в значительной мере отражают существующие реальности в практике параллельных вычислений. PRAM занимает промежуточное положение между этими полюсами. Являясь достаточно абстрактной моделью, она в то же время близка по архитектуре к существующим многопроцессорным машинам и чрезвычайно удобна для записи параллельных алгоритмов. Это послужило причиной ее широкого распространения и сделало основным инструментом для записи параллельных алгоритмов.

1.2. Класс NC

Интуитивное представление о классе задач, для решения которых существуют эффективные параллельные алгоритмы, формализуется в определении класса **NC**. *Этот класс состоит из всех задач, которые могут быть решены на* PRAM *с полиномиальным от длины входа числом процессоров за время, ограниченное некоторым полиномом от логарифма длины входа* (*полилогарифмическое время*). При этом, как и в случае RAM, рассматривается равномерное время и вычисления с числами, длина которых ограничена полиномом от длины входа. Более формально, для каждой задачи из **NC** существуют константы c и k и алгоритм для PRAM с $O(n^c)$ процессорами и временем работы $O(\log^k n)$ на входах длины n. Будем для удобства сравнения с классом **P** рассматривать только задачи распознавания, ответ в которых получается в виде "да" или "нет". В определении **NC** Кука [95],[98] рассматривались произвольные вычислительные задачи, ответ в которых является целым числом. Иногда, чтобы отметить это различие, для класса вычислительных задач, аналогичных **NC**, используют обозначение FNC.

Поскольку каждый шаг наиболее сильной модели (STRONG CRCW PRAM) можно смоделировать на самой слабой (EREW PRAM) за время $O(\log p)$, где p – полиномиально ограниченное число процессоров, определение класса **NC** не зависит от выбранной разновидности PRAM [116].

При использовании равномерной меры в качестве меры параллельной сложности необходимость ограничения на максимальную длину чисел, с которыми работает PRAM, видна из следующего примера.

Рассмотрим задачу "Сумма подмножеств", которая заключается в следующем: существует ли для данных натуральных чисел $a_1, \ldots, a_n$ и натурального k булев вектор $(x_1, \ldots, x_n)$ такой, что $\sum a_i x_i = k$. Известно, что эта задача является NP-полной [8]. В тоже время, используя операции со сколь угодно большими числами, можно построить NC-алгоритм решения этой задачи, если в PRAM допустить использование команд сложения и умножения чисел в формате с плавающей запятой. Действительно, рассмотрим полином

$$f(t) = (1 + t^{a_1}) \ldots (1 + t^{a_n}) = \sum b_i t^i$$

Нетрудно заметить, что b_k равно числу решений уравнения $\sum a_i x_i = k$. Для натурального $M = 2^{n+1}$ вычислим $Y_k = f(M) \pmod{M^{k+1}}$. Заметим, что $Y_k \geq M^k$, тогда и только тогда, когда исходное уравнение имеет решение. В самом деле, если $b_k \geq 1$, то $Y_k \geq M^k$, а если $b_k = 0$, то $Y_k < 2^n M^{k-1} < M^k$. Значение f(M) легко вычислить за время O(log n) на PRAM с O(n) процессорами при возможности арифметических операций со сколь угодно большими числами, заданными в формате с плавающей запятой. При этом для записи числа отводятся две ячейки памяти – для мантиссы и для порядка. Тогда степени 2 вычисляются занесением порядка в соответствующую ячейку памяти, а произведение скобок можно вычислить методом сдваивания, вычисляя на каждом шаге попарные произведения четных и нечетных сомножителей до тех пор, пока не останется один сомножитель, который и является результатом (см. раздел 2.1.1.).

Этот пример показывает необходимость ограничения длины слова в ячейках памяти некоторым полиномом от длины входа. Необходимость ограничений на процесс вычислений PRAM подтверждается и другими результатами. Так, при наличии естественных арифметических и логических команд на PRAM за полиномиальное время моделируется PSPACE [120] – класс функций, вычислимых на машине Тьюринга с полиномиальной памятью. Более того, любая булева функция от n переменных может быть вычислена на идеальной CRCW PRAM с p процессорами за время $\log n - \log\log p/n + O(1)$, где идеальная PRAM отличается от обычной тем, что каждый процессор за единичное время может вычислять любую булеву функцию от аргументов в локальной памяти этого процессора [57]. Идеальная PRAM, будучи моделью не адекватной реальным вычислениям, используется лишь для получения нижних оценок, которые, с учетом силы модели, оказываются верны и для реальных разновидностей PRAM.

Можно говорить, что задачи из класса **NC** эффективно распараллеливаются в том же смысле, что и задачи из класса **P** эффективно разрешимы. Как и класс **P**, класс **NC** не зависит от выбора разумной формальной модели в данном случае параллельных вычислений.

Подобный вывод подтверждает и другой взгляд на задачи класса **NC** при его определении с помощью схем [95],[98],[332].

Определение. *Определим класс всех функций, вычислимых с помощью равномерных семейств булевых схем* (c_n) *таких, что*

$s(c_n) = n^{O(1)}$ *и* $d(c_n) = O(\log^k n)$ *и обозначим его как класс* $\mathbf{NC}^k$. *Положим также* $\mathbf{NC} = \cup_k \mathbf{NC}^k$.

Это определение эквивалентно определению класса **NC** через PRAM. Доказательство основано на моделировании PRAM схемами и наоборот с соответствующими оценками времени (размера) и памяти (глубины). В [301] установлена возможность моделирования схемами с неограниченным числом входов у элементов PRAM с ограниченным набором команд. Ограничение заключается в том, что используются только операции, увеличивающие длину слова не более, чем на один бит. Таким образом, исключаются умножения и деления и после полиномиального числа шагов длина записи в ячейках памяти не превышает полинома от длины входа. При таком моделировании для PRAM с p процессорами и с временем работы t строится схема с неограниченной степенью входа элементов глубины O(t) и размера O(p), вычисляющая ту же функцию на словах длины n.

При рассмотрении алгебраических схем аналогичным образом определяется алгебраический класс $\mathbf{NC}^k$. Он естественным образом возникает и используется при исследовании параллельной сложности алгебраических задач.

Прежде, чем описать идею моделирования PRAM равномерными семействами булевых схем, приведем определение классов $\mathbf{AC}^k$, играющих важную роль в теории параллельных вычислений.

Определение. *Определим* $\mathbf{AC}^k$ *как класс всех функций, вычислимых с помощью равномерных семейств булевых схем* (c_n) *с неограниченной степенью входа таких, что* $s(c_n) = n^{O(1)}$ *и* $d(c_n) = O(\log^k n)$ *Положим также* $\mathbf{AC} = \cup_k \mathbf{AC}^k$.

Имеют место следующие включения:

$$\mathbf{NC}^k \subseteq \mathbf{AC}^k \subseteq \mathbf{NC}^{k+1}$$

Первое включение очевидно, а второе следует из возможности замены каждого элемента конъюнкции или дизъюнкции с неограниченной степенью входа на двоичное дерево, составленное из соответствующих элементов с двумя входами. При полиномиальном числе элементов в схеме, глубина дерева по порядку равна логарифму n, откуда и вытекает второе включение. Очевидно также, что $\mathbf{NC} = \mathbf{AC}$. Вопрос о различии классов $\mathbf{AC}^k$ и $\mathbf{NC}^{k+1}$ остается открытым. Известно лишь, что $\mathbf{AC}^0$ не совпадает с $\mathbf{NC}^1$ ([123],[124] и в более общем классе схем [19]).

Моделирование схем посредством CREW PRAM можно выполнить естественным образом.

Если имеется схема вычислений размером S *и глубиной* d, *то ее можно вычислить на* CREW PRAM *с* $\lceil S/d \rceil$ *процессорами за время* 2d.

Соответствующее вычисление можно осуществить поярусно. Время вычисления при этом не превышает величины $T = \sum_i(n_i/\lceil S/d\rceil + 1) \le \sum_i n_i/\lceil S/d\rceil + d \le S/\lceil S/d\rceil + d \le 2d$, где n_i – размер i-го яруса.

Подобное моделирование схем возможно и на EREW PRAM [335], [167]. В частности, если все выходные степени элементов в схеме ограничены константой, можно каждый шаг CREW PRAM заменить на константное число шагов EREW PRAM, используя повторные считывания из памяти.

Опишем теперь идею моделирования одного шага CRCW PRAM с полиномиальным числом процессоров булевыми схемами полиномиального размера и логарифмической глубиной.

Поскольку суммарное число операций, выполняемых всеми процессорами полиномиально, числу ячеек памяти, используемых в процессе работы всеми процессорами, также полиномиально. Моделирование одного шага работы PRAM будем осуществлять путем эквивалентного изменения конфигурации PRAM. Под конфигурацией понимается булев вектор, кодирующий состояние всех ячеек памяти (полиномиального размера), текущих команд, выполняемых каждым процессором, всех сумматоров. Будем считать, что в конфигурации PRAM сначала кодируются программы всех процессоров, затем текущие команды, выполняемые каждым процессором, затем состояния сумматоров каждого процессора и, наконец, номера и содержимое всех ячеек памяти, используемой в процессе работы PRAM.

Каждый шаг PRAM моделируется схемой, состоящей из трех кас-

каков: первый моделнрует считывание нз памяти в соответствующие процессоры, второй – выполнение арифметическнх операций, третнй – запись полученных результатов в память . Параллельно аналогичным образом моделнруется выбор очередной команды, нсполняемой каждым процессором . Каждому процессору будут соответствовать несколько блоков, выполняющих операцни с числами в соответствнн с командамн PRAM (отметнм, что сложенне, умноженне, деленне могут быть реалнзованы схемамн логарифмнческой глубнны н полнномиального размера).

На первом каскаде задача заключается в том, чтобы на вход каждого блока подать нужную часть конфнгурацнн PRAM, соответствующую чнслу, с которым выполняет операцню данный процессор. Для этого вначале схема определяет адрес, нз которого данный процессор счнтывает нз памятн. . Адрес, в завнсимостн от способа адресацнн в команде, равен одному нз трех чнсел: чнслу x, запнсанному в команде в виде операнда, чнслу y, являющемуся содержимым ячейки с номером x, н чнслу z, являющемуся содержнмым ячейкн с номером y. Прн этом осуществляется параллельное счнтываннe нз соответствующей частн конфнгурацнн содержнмого ячеек x, y и z, а затем в завнсимостн от тнпа адреса в команде (число, содержимое адреса, содержимое содержимого адреса) выбирается одно нз счнтанных чнсел

Второй каскад состоит нз блоков, осуществляющих арифметические и логическне операцнн над операндамн в соответствнн с набором команд PRAM Известно, что нх можно выполнить схемамн логарнфмнческой глубнны н полнномиального размера (см. раздел 2.2.2).

Третнй каскад аналогнчен первому, только преобразованне конфнгурацнн нитерпретируется как запись в память. Еслн над содержнмым ячейкн памятн не пронзводнтся никакой операцнн, соответствующая часть конфигурацнн не нзменяется. При нспользовании косвенной адресацнн номера нспользуемых ячеек памятн могут быть большимн (с полиномнальной длнной записи) н, чтобы нзбежать работы с большимн чнсламн, адреса всех ячеек памятн в конфнгурацнн сортнруются н формнруется таблнца соответствнй адресов и нх упорядоченых номеров Аналогнчно сортнруются адреса операндов н формируется таблнца соответствия. Эти сортнровкн можно выполнить схемой глубнны O(log n) и полнномиального размера. Схема имеет

дело не с адресами (которые могут быть большими из-за наличия косвенной адресации), а с их номерами, которые не превосходят полинома от длины входа и определяются по таблице соответствия за логарифмическое время.

Существуют различные характеризации классов NC и NC^k, например, с помощью альтернирующих машин Тьюринга [283], детерминированных машин Тьюринга [258], выразимости в логике первого порядка [171], ветвящихся программ (ВП) ограниченной ширины [53],[332], векторных машин [187].

Имеет место следующая характеризация класса NC, основанная на детерминированных машинах Тьюринга [258] (см. теорему 1.7):

Теорема 2.1. *NC это класс задач, разрешимых детерминированными машинами Тьюринга за полиномиальное время с не более чем полилогарифмическим числом поворотов головки машины Тьюринга.*

Имеют место следующие характеризации класса NC, основанные на альтернирующих машинах Тьюринга [283].

Теорема 2.2. *Функция принадлежит классу* NC, *если и только если она вычислима альтернирующей машиной Тьюринга за время* $\log^{O(1)}n$ *с использованием памяти* $O(\log n)$.

Имеет место аналогичная характеризация класса NC^k, основанная на альтернирующих машинах Тьюринга [283].

Теорема 2.3. *Функция принадлежит классу* NC^k, *если и только если она вычислима альтернирующей машиной Тьюринга за время* $O(\log^k n)$ *с использованием памяти* $O(\log n)$.

Имеет место следующая характеризация класса NC, основанная на векторных машинах [187].

Теорема 2.4. *Функция принадлежит классу* NC, *если и только если она вычислима равномерным семейством векторных машин с полиномиальным числом процессоров за полилогарифмическое время.*

Альтернирующие машины Тьюринга (АТМ) оказываются удобными для характеризации различных сложностных классов, связанных с параллельными вычислениями. Так например, класс AC^k характеризуется как класс функций вычислимых АТМ с логарифмической памятью и числом альтернирований $O(\log^k n)$.

Приведем теперь характеризацию класса NC^1 в терминах ветвящихся программ ограниченной ширины. При этом рассматриваются только синхронные ветвящиеся программы, в которых длины всех путей между любыми двумя данными вершинами одинаковы.

Теорема 2.5 [53]. *Функция принадлежит классу* NC^1, *если и только если она вычислима ветвящейся программой полиномиального размера и константной ширины.*

Доказательство. Пусть g_n – ВП полиномиального размера $p(n)$ и константной ширины k, вычисляющая функцию f_n. Покажем, что f_n может быть вычислена схемой глубины $O(\log n)$. Обозначим через v_{ij} i-ю вершину j-го яруса ВП g_n. Пусть $g_{i,j,l,m}(x) = 1$ если при входе x существует путь от вершины v_{ij} в вершину v_{lm}. Отметим, что $f_n = g_{1,0,1,p(n)}(x)$. Пусть $j < s < m$. Поскольку каждый путь из яруса j в ярус m проходит через ярус s, то

$$g_{i,j,l,m}(x) = \bigvee_{1 \le t \le k} g_{i,j,t,s}(x) \cdot g_{t,s,l,m}(x). \qquad (1.1)$$

Для вычисления по этой формуле можно воспользоваться методом "разделяй и властвуй" (см. раздел 2.1.3). Пусть для вычисления $g_{i,j,t,s}(x)$ и $g_{t,s,l,m}(x)$ схемы глубины d уже построены, тогда, подставляя эти схемы в формулу (1.1), получим схему глубины $d + O(\log k)$. Отсюда при соответствующем выборе s ($s = \lfloor (j + m)/2 \rfloor$) получаем соотношение:

$$d(p(n)) \le d(p(n)/2) + O(\log k),$$

где $d(p)$ – минимальная глубина схемы, эквивалентной ветвящейся программе размера p. Это соотношение приводит к оценке $d(p(n)) = O(\log k \log p(n)) = O(\log p(n)) = O(\log n)$.

Для доказательства обратного утверждения о возможности вычисления функции класса NC^1 ветвящейся программой константной ширины и полиномиального размера введем новый тип ветвящихся программ, называемых перестановочными ВП.

Определение. Перестановочной ВП (ПВП) шириной k и глубиной l называется последовательность команд $(j(i), g_i, h_i)$ $0 \le i < l$, $1 \le j(i) \le n$ и $g_i, h_i \in S_k$ (S_k – группа подстановок k элементов). Каждый ярус состоит из k вершин и на i-м ярусе реализуется подстановка $\sigma_i(x) = g_i$, если $x_{j(i)} = 0$, и $\sigma_i(x) = h_i$, если $x_{j(i)} = 1$. ПВП вычисляет $\sigma(x) = \sigma_{l-1}(x) \cdot \ldots \cdot \sigma_0(x)$ на входе x. ПВП вычисляет с помощью τ функцию f_n, зависящую от n переменных, если для всех x выполнено соотношение:

$$\sigma(x) = \begin{cases} id, & \text{если } f(x) = 0, \\ \tau \ne id, & \text{если } f(x) = 1, \end{cases}$$

где id – тождественная перестановка.

Лемма. *ПВП шириной* k *и глубиной* l *может быть промоделирована ВП шириной* k *и глубиной* kl.

Доказательство. ПВП имеет k входных вершин. Каждая входная вершина порождает ВП шириной k и глубиной l. Вершинам v_{mi} сопоставляется переменая $x_{\chi(i)}$. Построим r-ю ВП следующим образом. Ее входом является r-я вершина первого яруса ПВП, единичным выходом – τ(r)-я вершина последнего яруса ПВП, нулевыми вершинами – остальные вершины последнего яруса ПВП. Эта ВП выдает 1 тогда и только тогда, когда σ(x)(r) = τ(r). Следовательно, f(x) = 1 тогда и только тогда, когда все k таких ВП выдают 1. Используя известные методы, построим ВП, вычисляющую конъюнкцию всех этих ВП.

Как будет показано далее, для моделирования NC^1 достаточно ограничиться рассмотрением ВП ширины 5.

Лемма. *Если имеется ПВП, вычисляющая* f *с помощью циклической перестановки* τ *длины 5 (5-цикла) и* ρ *– другой 5-цикл, то существует ПВП с такой же глубиной, вычисляющая* f *с помощью* ρ.

Доказательство. Пусть θ – такая подстановка, что $\rho = \theta\tau\theta^{-1}$. Для получения эквивалентной ПВП заменим перестановки g_i и h_i на $\theta g_i\theta^{-1}$ и $\theta h_i\theta^{-1}$.

Лемма. *Если имеется ПВП, вычисляющая* f *с помощью 5-цикла* τ, *то существует ПВП такой же глубины, вычисляющая* ¬f *с помощью 5-цикла.*

Доказательство. Заметим, что τ^{-1} – 5-цикл. Заменим последнюю команду g_{l-1} на $\tau^{-1}g_{l-1}$ и h_{l-1} на $h_{l-1}\tau^{-1}$. Нетрудно заметить, что существует 5-цикл τ_1 и τ_2 такие, что $\tau_1\tau_2\tau_1^{-1}\tau_2^{-1}$ является 5-циклом, поскольку

$$(12345)(13542)(54321)(24531) = (13254)$$

Теорема. *Пусть* f *вычислима схемой глубины* d *в базисе* (&,∨,¬). *Тогда существует ПВП ширины 5 и глубины* 4^d, *вычисляющая* f.

Доказательство. Для d = 0 утверждение очевидно. Не ограничивая общности предположим, что $f = f_1 \& f_2$. Предположим, что f_1 и f_2 вычислимы ПВП G_1 и G_2 глубиной 4^{d-1} с помощью 5-циклов ρ_1 и ρ_2 соответственно. Согласно приведенным выше леммам, ρ_1 и ρ_2 можно заменить на τ_1 и τ_2 так, что $\tau_1\tau_2\tau_1^{-1}\tau_2^{-1}$ является 5-

циклом. Более того, существуют ПВП G_3 и G_4 глубиной 4^{d-i}, вычисляющие f_1 и f_2 с помощью 5-циклов τ_1^{-1} и τ_2^{-1} соответственно. Последовательное соединение G_1, G_2, G_3 и G_4 дает ПВП G глубиной 4^d.

Обозначим через $\sigma(x) = \sigma_1(x)\sigma_2(x)\sigma_3(x)\sigma_4(x)$ перестановку, вычисляемую ПВП G. Рассмотрением 4-х случаев (в зависимости от значений f_1 и f_2) можно убедиться, что G вычисляет $f = f_1 \& f_2$.

Используя приведенную выше лемму о моделировании ветвящейся программы шириной k перестановочной ветвящейся программой шириной k, получаем доказательство основного результата.

Приведенная характеризация касается неравномерного NC^1 (без требования равномерности семейства схем). Для равномерного случая характеризация также остается справедливой [53].

В классе NC^1 содержится большое число практически важных функций. В частности, NC^1 принадлежат сложение n n-битовых чисел, умножение n-битовых чисел, деление (неравномерному NC^1), сортировка n n-битовых чисел, умножение двух целочисленных матриц размером nxn, составленных из n-битовых чисел.

В теории параллельных вычислений считается, что класс эффективно распараллеливаемых задач совпадает с классом NC. Однако, часто на практике не требуется столь глубокое распараллеливание, а достаточно получить лишь значительное ускорение при решении задач. Например, для задачи линейного программирования и задачи о максимальном паросочетании, принадлежность которых классу NC весьма проблематична, известны достаточно хорошие параллельные алгоритмы [312], [144]. Для более детальной классификации таких задач рассматривают иногда классы OC и PC. В классе OC лежат задачи, разрешимые на PRAM с полиномиальным числом процессоров за время $o(n^e)$ для всякого $e > 0$, где n - длина входа задачи. В классе PC лежат задачи, которые можно решить на PRAM с полиномиальным числом процессоров так, что соотношение между параллельным и последовательным временем решения стремится к нулю при стремлении длины входа к бесконечности (то есть достигается ускорение, не ограниченное константой). Выполняется естественное включение $NC \subseteq OC \subseteq PC \subseteq P$. Таким образом, класс PC представляет задачи, которые можно распараллелить "частично", класс OC - хорошо распараллеливаемые задачи и класс NC - предельно распараллеливаемые задачи.

1.3. Сводимость и P-полнота

Из определения класса **NC** следует, что имеет место включение **NC** ⊆ **P**, где **P** – класс задач, разрешимых за полиномиальное время. Хотя для многих задач класса **P** доказана их принадлежность классу **NC**, неизвестно, совпадают ли эти классы. Их совпадение означало бы, что любая задача, для которой существует эффективный последовательный алгоритм, допускает эффективное распараллеливание. Имеются примеры задач, для которых многочисленные попытки построить параллельные алгоритмы не привели к успеху. Для описания "самых трудных" для распараллеливания задач из класса **P** вводится понятие P-полной задачи, принадлежность которых классу **NC** означала бы совпадение классов **P** и **NC**.

Удобным инструментом для изучения параллельной сложности задач является понятие NC^1-сводимости. Применение NC^1-сводимости аналогично использованию полиномиальной сводимости при классификации задач класса **NP**.

Определение. *Будем говорить, что функция* f NC^1*-сводится к функции* g (*обозначение* $f \le g$), *если найдется такое равномерное семейство схем* (c_n), *вычисляющее* f, , *что глубина* $d(c_n) = O(\log n)$, *причем в* (c_n) *допускается использование вершин, вычисляющих* g, *то есть вершин с входами* $y_1,\dots,y_r$ *и выходами* $z_1,\dots,z_m$, *для которых выполняется соотношение* $(z_1,\dots,z_m) = g(y_1,\dots,y_r)$. *Глубина такой вершины считается равной* $\lceil \log r \rceil$, *а ее размер* – $r + m$, *при этом длина пути определяется как сумма глубин вершин.*

Нетрудно установить, что NC^1-сводимость является рефлексивным и транзитивным отношением.

Определение. *Функция* $f \in P$ *называется* P*-полной, если любая функция из* **P** *к ней* NC^1*-сводится.*

Понятию NC^1-сводимости близко понятие LOGSPACE-сводимости, или сводимости с логарифмической памятью. *Сводимость с логарифмической памятью функции* f *к функции* g (*обозначение* $f \le_{\log} g$) *означает, что существует функция* ϕ, *вычислимая на машине Тьюринга с логарифмической памятью, такая, что* $\phi(g) = f$.

Поскольку классы NC^1 и **LOGSPACE** содержатся в классе **NC**, то из того, что $g \in NC$ и $f \le g$ или $f \le_{\log} g$ следует, что $f \in NC$.
В частности, если некоторая P-полная задача принадлежит классу **NC**, то **P** = **NC**. Поскольку $NC^1 \subseteq$ **LOGSPACE** то NC^1-сводимость "жестче", чем LOGSPACE-сводимость и иногда построить ее оказывается

труднее. Мы будем говорить о P-полноте задач применительно как к NC^1-сводимости, так и к LOGSPACE-сводимости, не оговаривая в дальнейшем это различие. Как отмечено в [95], почти все известные в литературе P-полные задачи являются полными относительно NC^1-сводимости. Кроме того, NC^1-сводимость удобна для описания полных задач в классах NC^k при $k \geq 2$.

В настоящее время установлена P-полнота нескольких известных задач класса P. Ниже рассматриваются некоторые из известных P-полных задач.

Первая P-полная задача была сформулирована С. Куком, которую он назвал *«Достижимость системой путей»* [95],[97].

Вход. Система путей (X,S,T,R), $(S \subseteq X, T \subset X, R \subseteq X \times X \times X)$.

Вопрос. Существует ли достижимый узел в S (узел называется достижимым, если $x \in T$ или существуют такие достижимые узлы $y,z \in X$ что $(x,y,z) \in R$).

Одной из наиболее известных P-полных задач, которая часто используется для доказательства P-полноты ряда других задач, является задача *"Значение схемы"* [209].

Вход. Булева схема и значения входных переменных для нее.

Вопрос. Определить значение, вычисляемое схемой при заданном входе.

Доказательство P-полноты задачи "Значение схемы" аналогично доказательству NP-полноты задачи 3-выполнимость [8]. При этом достаточно показать, что длялюбой машины Тьюринга, работающей полиномиальное время, можно построить равномерное семейство схем, вычисляющих ту же функцию, причем само построение можно осуществить на машине Тьюринга с логарифмической памятью. Это семейство схем строится по протоколу вычисления машины Тьюринга достаточно просто (см. [16]).

Пусть машина Тьюринга M с односторонней лентой на входах длины n работает не более T тактов. Схема, моделирующая работу этой машины, состоит из одинаковых блоков, соединяемых однотипным образом. Каждый ярус схемы состоит из $T + 1$ блока, а всего в схеме имеется T ярусов. Очередной t-й такт работы машины Тьюринга моделируется t-м ярусом схемы.

Два соседних яруса схемы образуют двудольный граф, причем i-я вершина одной доли соединена ребрами с $(i - 1)$-й, i-й и $(i + 1)$-й вершинами второй доли (кроме первой и последней вершин, для которой отсутствуют ребра, ведущие в вершины с номерами -1 и $T +$

1). Блоки, из которых состоит схема, устроены следующим образом. Они имеют r + s + 3 входа и столько же выходов, где r – число букв алфавита машины M, s – число состояний. Все r + s входов и выходов соответствуют ребру, соединяющему i-ю вершину каждого яруса с i-й вершиной следующего яруса. Оставшиеся три входа x_1, x_2, x_3 и выхода y_1, y_2, y_3 связывают i-ю вершину яруса соответственно с (i-1)-й, i-й и (i+1)-й вершинами соседних ярусов. Эти переменные x_1, x_2, x_3 и y_1, y_2, y_3 моделируют движение головки машины Тьюринга: $x_1 = 1$ соответствует движению головки вправо, $x_3 = 1$ соответствует движению головки влево и $x_2 = 1$ соответствует отсутствию движения головки. Блок реализует следующую функцию: 1) если $x_1 = x_2 = x_3 = 0$, то осуществляется тождественное преобразование (значения входов равны значениям выходов);
2) если один из трех входов x равен 1, то в соответствии с программой машины Тьюринга M первые r выходов кодируют новый символ на ленте, который заменяет старый символ, следующие s выходов кодируют новое состояние, в которое переходит машина M, и полагается $y_1 = 1$ в случае движения головки влево, $y_2 = 1$ в случае, если головка не меняет своего положения, и $y_3 = 1$ в случае движения головки вправо. Число булевых элементов в блоке равно константе, поскольку константами являются параметры r и s.

Легко проверяется, что данная булева схема моделирует работу машины M, поскольку на каждом ярусе происходит модификация всей записанной на ленте машины конфигурации в соответствии с программой машины M. Для доказательства P-полноты задачи "Значение схемы" остается только убедиться, что построение схемы можно выполнить в NC^1.

Задача остается P-полной, если ограничиться только монотонными (не содержащими отрицаний) или только плоскими (граф которых планарен) схемами [145]. В первом случае достаточно вынести отрицания (по правилам де Моргана) во входы схемы, а во втором – заменить каждое пересечение дуг в произвольной укладке схемы на плоскости на планарную схему, состоящую из трех сложений по модулю два и реализующую функцию f(y, z) = (z, y), реализующую на выходах перестановку двух входных значений. Оба эти преобразования схем могут быть выполнены в NC^1. Задача о значении монотонной схемы остается P-полной также в случае схем со степенью выхода каждого элемента не более двух, поскольку произвольная схема преобразуется в такую схему заменой каждого ветвления степени

больше двух бинарным деревом, и это преобразование можно также выполнить в NC^1.

Интересно отметить, что задача о значении монотонной планарной схемы лежит в классе NC (см. [146],[228] и раздел 2.1.4).

Задача о максимальном потоке [147], [97].

Вход. Ориентированный взвешенный граф G, в котором выделены исток и сток (пропускные способности дуг предполагаются неотрицательными целыми, заданными в двоичной системе). Целое число M.

Вопрос. Верно ли, что M-й разряд величины максимального потока равен единице?

К этой задаче сводится, например, задача о значении монотонной булевой схемы со степенью ветвления выходов элементов не превосходящих двух. Сведение осуществляется следующим образом. Пусть $(\alpha_n, \ldots, \alpha_0)$ - монотонная булева схема, где α_i - либо входная переменная, либо элемент вида AND(j,k) либо элемент вида OR(j,k), где $j,k < i$. Множество вершин графа G состоит из чисел $(n, \ldots, 0)$, истока s и стока t. Дуги графа определяются в зависимости от типа элемента α_i:

- если α_i - вход, то (s,i) - дуга графа G с пропускной способностью 2^i, если $\alpha_i = 1$, и 0, если $\alpha_i = 0$. Дуга (i,s) имеет пропускную способность 2^i

- если α_i = AND(j,k) то (j,i) и (k,i) - дуги графа G с пропускной 2^j и 2^k соответственно и (i,t) - дуга с пропускной способностью $2^j + 2^k - d2^i$, где d - число выходов элемента α_i $(d \le 2)$.

- если α_i = OR(j,k) то (j,i) и (k,i) - дуги графа G с пропускной 2^j и 2^k соответственно и (i,s) - дуга с пропускной способностью $2^j + 2^k - d2^i$, где d - число выходов элемента α_i $(d \le 2)$.

Наконец, (0,t) имеет пропускную способность 1.

Утверждение. *Величина максимального потока в G нечетна в том и только в том случае, если значение построенной выше схемы равно* 1.

Это утверждение доказывается прямым построением потока, моделирующего вычисление на булевой схеме и являющегося максимальным, и использованием свойства четности всех пропускных способностей дуг за исключением дуги, идущей из выхода схемы в сток. В силу этого свойства значение максимального потока нечетно тогда и только тогда, когда дуге, ведущей из выхода схемы в

сток, приписано значение 1, т.е. тогда и только тогда, когда значение схемы есть 1.

Задача о максимальном потоке принадлежит NC для планарных графов [173] и разрешима рандомизированными NC-алгоритмами, в случае, если пропускные способности дуг заданы в унарной системе (см. раздел 2.1.6).

Транспортная задача.

Вход. Неотрицательные числа a_i, b_j, c_{ij}, $i = 1,\ldots,n$, $j = 1,\ldots,m$.

Выход. Матрица $x_{ij} \geq 0$, минимизирующая $\sum_{i,j} c_{ij}x_{ij}$ при ограничениях:

$$\sum_j x_{ij} \leq a_i, \quad \sum_i x_{ij} \geq b_j.$$

P-полнота вытекает из известной сводимости задачи о максимальном потоке в сети к транспортной задаче [23].

Задача линейного программирования [97],[191].

Вход. Целочисленные векторы c и b, целочисленная матрица A, целое число C.

Вопрос. Существует ли x, такое, что $cx > C$, $Ax < b$?

Как известно, эта задача принадлежит классу P [26]. Она является P-полной, так как ее частный случай – задача о максимальном потоке P-полна. Задача остается P-полной даже в случае, если числа заданы в унарной системе [319],[191]. Действительно, имеется следующее прямое сведение задачи о значении схемы к задаче линейного программирования. Каждому элементу схемы сопоставляется булева переменная, которая будет равна 1, если значение элемента равно 1 и равна 0, если значение элемента равно 0. Более точно, для каждого единичного входа полагаем соответствующую переменную $x_j = 1$ и для нулевых входов полагаем соответствующую переменную $x_j = 0$. Для элемента конъюнкции с входами от h-го и k-го элементов полагаем

$$x_j \leq x_h, \; x_j \leq x_k, \; x_j \geq 0, \; x_j \geq x_h + x_k - 1.$$

Для элемента отрицания с входом от i-го элемента полагаем

$$x_j = 1 - x_i.$$

Для выходного элемента с входом от k-го элемента полагаем

$$x_j = x_k,\ x_j = 1.$$

Элементы дизъюнкции исключаются из схемы. Нетрудно проверить, что всякое решение этой линейной программы (всей совокупности описанных неравенств) является 0–1 вектором и допустимое решение существует тогда и только тогда, когда значение схемы равно единице. Кроме того, можно проверить, что описанное преобразование булевой схемы в линейную программу является LOGSPACE-сводимостью. Таким образом, задача линейного программирования остается P-полной даже если все коэффициенты принимают значения из множества (–1, 0, 1) и в каждом столбце и каждой строке матрицы ограничений имеется не более трех ненулевых значений.

Для двух переменных известен алгоритм с временем работы $O(\log n)$ на $O(n/\log n)$ процессорах CRCW PRAM [108]. Если в каждом неравенстве встречается не более двух переменных, то задача остается P-полной и может быть решена за время $O((\log m + \log^2 n)\log^2 n)$ на CREW PRAM с $mn^{O(\log n)}$ процессорами [222]. При фиксированном числе переменных задача линейного программирования почти наверное может быть решена параллельно за константное время [38] на CRCW PRAM с $O(n)$ процессорами. В [312] предложен параллельный алгоритм решения задачи линейного программирования с n переменными и m неравенствами-ограничениями за время $O((mn)^{1/4}\log^3 m\ L)$ на $O(M(n)m/n + n^3)$ процессорах, где $M(n)$ – число операций, необходимое для умножения двух матриц размером $n \times n$.

Приближенная задача линейного программирования также является P-полной [290]. При этом доказательство P-полноты ε-приближенной задачи аналогично приведенному выше и основано на сведении ее к задаче нахождения ε-приближения числа элементов булевой схемы, принимающих при данном значении входных переменных единичные значения. Задача нахождения ε-приближения числа единичных элементов булевой схемы легко сводится к задаче о значении булевой схемы путем многократного дублирования выходного элемента схемы и взятия конъюнкции этих элементов. При этом значение всех дубликатов на данном входе всегда одинаково и достаточно найти приближенное значение числа единичных элементов схемы, чтобы определить каково значение выхода исходной схемы.

Возможно также другое простое сведение точной задачи к приближенной, если добавить одну дополнительную переменную z с

достаточно большим коэффициентом C_0 при ней в целевой функции и таким столбцом в матрице ограничений, чтобы все неравенства были выполнены при $z = 1$, $x_1 = \ldots = x_n = 0$. Тогда если целевая функция больше или равна C_0, то решения у исходной задачи нет, а в противном случае – она разрешима. Открытым остается вопрос о сложности нахождения ε-приближенного решения задачи линейного программирования с неотрицательными исходными данными.

Лексикографически первое дерево поиска в глубину для графа [280].

Вход. Конечный граф с выделенным ребром.

Вопрос. Содержится ли выделенное ребро в лексикографически первом дереве поиска в глубину этого графа ?

Для планарных графов, а также для ациклических орграфов эта задача принадлежит NC [158]. Для произвольных графов и орграфов задача принадлежит RNC [33],[32].

Наличие k-связного подграфа [194].

Вход. Граф, натуральное k ($k \geq 3$).

Вопрос. Имеется ли в графе k-связный подграф?

К этой задаче сводится задача о значении монотонной схемы.

Лексикографически первое максимальное по включению независимое множество [97].

Вход. Конечный граф с выделенной вершиной и линейным порядком на множестве вершин.

Вопрос. Содержится ли выделенная вершина в лексикографически первом максимальном по включению независимом множестве?

К этой задаче сводится задача о значении монотонной схемы. П усть элементы некоторой схемы А топологически отсортированы (разбиты на ярусы) и записаны в виде упорядоченного массива $[g_0, g_1, g_2, \ldots, g_n]$. Построим по схеме граф с множеством вершин $V = \langle v_0, v_1, v_2, \ldots, v_n \rangle \cup \langle w_0, w_1, \ldots, w_n \rangle$ и линейный порядок на множестве его вершин.

Линейный порядок таков, что при $i < j$, v_i и w_i предшествуют v_j и w_j. Вершина w_0 предшествует вершине v_0 и v_1 предшествует w_1.

1) $g_i = g_j \,\&\, g_k$. Тогда v_i предшествует w_i и в граф включаются ребра вида $\langle v_i, w_j \rangle$ и $\langle w_k, v_i \rangle$.

2) $g_i = g_j \vee g_k$. Тогда w_i предшествует v_i и в граф включаются ребра вида $\langle v_j, w_i \rangle$ и $\langle v_k, w_i \rangle$. Кроме этого, всегда добавляются ребра вида $\langle v_i, w_i \rangle$.

В построенном таким образом графе G каждому элементу схемы g_i соответствуют две вершины графа v_i и w_i таким образом, что лексикографически первая клика в G включает v_i тогда и только тогда, когда значение схемы в элементе g_i равно 1 и эта клика включает w_i тогда и только тогда, когда значение схемы в g_i равно 0. Это свойство можно доказать индукцией по n. Построение графа по схеме описанным способом можно выполнить в LOGSPASE, то есть описанное сведение задачи о значении булевой схемы к задаче построения лексикографически первого независимого множества в графе является LOGSPACE-сводимостью.

В последовательном варианте эта задача тривиально решается жадным алгоритмом, выбирающим на каждом шаге вершину с минимальным номером, не смежную с ранее построенными вершинами.

Задача остается P-полной для класса двудольных графов с максимальной степенью вершин 3 и для класса планарных графов с максимальной степенью вершин 3 [236]. В классе деревьев и графов с максимальной степенью вершин 2 задача принадлежит NC^2 [237].

Лексикографически первый максимальный по включению подграф с заданным наследственным свойством π.

Вход. Конечный граф с выделенной вершиной.

Вопрос. Содержится ли выделенная вершина в лексикографически первом максимальном по включению подграфе, обладающем свойством π?

Свойство π называется наследственным, если из того что граф обладает свойством π вытекает, что им обладает и всякий вершинный подграф. Примерами наследственного свойства являются отсутствие в графе гамильтонова цикла, отсутствие клики заданного размера. Свойство π называется нетривиальным для класса графов D, если бесконечное число графов из D удовлетворяют π и найдется граф из D, не удовлетворяющий π. В [236] показано, что для любого наследственного нетривиального полиномиально проверяемого свойства π задача нахождения лексикографически первого максимального по включению подграфа, обладающего свойством π, является P-полной. В последовательном варианте эта задача тривиально решается жадным алгоритмом.

Лексикографически первый максимальный по включению подграф с максимальной степенью вершин 1.

Вход. Конечный граф с выделенной вершиной.

Вопрос. Содержится ли выделенная вершина в лексикографически первом максимальном по включению подграфе с максимальной степенью вершин 1?

В [236] доказана P-полнота этой задачи для класса планарных двудольных графов с максимальной степенью вершин, равной 3.

Лексикографически первый максимальный путь в графе.

Вход. Граф с занумерованными вершинами, выделенная вершина v.

Выход. Лексикографически первый максимальный по включению путь, начинающийся в вершине v.

В [42] доказана P-полнота задачи даже для случая планарных графов, степени вершин которых не превышают 3. Найти некоторый максимальный путь для графов с максимальной степенью вершин d можно за время $O(d \log^3 n)$ на PRAM с $O(n^2)$ процессорами, а для планарных графов за время $O(\log^3 n)$ на n^2 процессорах.

Списочное расписание.

Вход. Упорядоченый список из n задач, время t_i исполнения i-й задачи, два процессора.

Вопрос. Существует ли такое списочное расписание выполнения задач без прерываний, что разность между суммарным временем занятости первого и второго процессоров равна 4?

Списочное расписание обладает следующими свойствами:

а) в каждый момент времени на одном процессоре выполняется не более одной задачи;

б) начала исполнения задач упорядочены в соответствии со списком.

Последовательный алгоритм решения задачи на каждом шаге выбирает очередную задачу из списка и назначает на процессор с меньшим суммарным временем исполняемых на нем задач. В [164] доказана P-полнота этой задачи путем сведения к ней задачи о значении булевой схемы. Рассматриваются схемы только из элементов "отрицание дизъюнкции". Поскольку этот элемент образует базис, любая схема может быть легко преобразована в схему, состоящую только из таких элементов. Элементы нумеруются от входов к выходам поярусно. Двум ребрам, ведущим в i-й элемент приписываются числа 4^{2i} и 4^{2i+1}. Выходному ребру приписывается число 4. Список строится следующим образом. Первая задача имеет время выполнения, равное сумме всех ребер идущих из единичных входов. В убывающем порядке по i поставим в соответствие i-му элементу

схемы 17 задач в списке – одну с временем 2 4^{2i+1}, 14 с временем $4^{2i}/2$ и две – с временем $(4^{2i}+ V_i)/2$, где V_i – сумма чисел на ребрах, выходящих из i-го элемента. Для двух процессоров списочное расписание для такого списка обладает свойством, что после выполнения всех задач разность между суммарным временем занятости процессоров равна 4 тогда и только тогда, когда значение схемы равно единице. Поскольку описанное сведение может быть выполнено в NC^1, задача является P-полной.

В [164] показано, что если времена выполнения ограничены сверху величиной $2^{L(n)}$, то списочное расписание может быть построено за время O(L(n) log L(n)) на EREW PRAM с n^2процессорами. Таким образом, задача принадлежит NC при L(n) = $O(\log^c n)$ для любой константы c.

Упаковка в контейнеры по упорядоченному списку.

Вход. Упорядоченный в невозрастающем порядке список объемов предметов (числа между 0 и 1) и числа i и b.

Вопрос. Упакует ли алгоритм упаковки по списку в соответствии с FFD эвристикой (first fit decreasing) i-й предмет в b-й контейнер?

P-полнота этой задачи доказана [43] путем сведения к ней задачи о значении монотонной булевой схемы со степенью ветвления не более двух. Задача остается P-полной, даже если все объемы заданы в унарной записи.

Унификация термов.

Вход. Два символьных выражения.

Вопрос. Существуют ли подстановки термов в переменные, переводящие выражения в синтаксически равные?

К этой задаче сводится задача о значении булевой схемы.

Частный случай этой задачи, когда одно из выражений не содержит свободных переменных, принадлежит классу NC^2 [111]. Число требуемых процессоров при этом $O(n^5)$. Для этого частного случая известен также RNC^2-алгоритм, использующий M(n) процессоров, где M(n) – число умножений, необходимых для умножения n×n матриц [112], а также детерминированный алгоритм для CREW PRAM с временем работы O(log n) и O(n) процессорами [202].

В силу известного включения $NC^k \subseteq AC^k \subseteq NC^{k+1}$ и замкнутости NC^k относительно NC^1-сводимости представляет интерес выделение полных в NC^k задач. Примером задачи из класса NC^k, полной относительно NC^1-сводимости в AC^k (и, следовательно, в NC^{k+1}), явля-

ется задача BRE(k) вычисления значения *булева разностного выражения* k-го *порядка:*

Вход. (M, B, F, j), где M – булева матрица размера m×n;
B – булева матрица размера n×n;
F – булев вектор размера n×1;
j – целое, $0 \le j \le \log^k n$.

Выход. Вычислить $M\ Y_j$, где $Y_r = F$, при $r = 0$,
$= B(Y_{r-1})^C$ при $r > 0$,
и Y^C обозначает дополнительный к Y булев вектор.

Интерес представляет также исследование с помощью NC^1-сводимости сложности проблем, лежащих между классами NC^1 и NC^2. Многие важные задачи из NC лежат в этих двух классах и их классификацию можно суммировать следуя [95] в следующей цепочке включений:

$$NC^1 \subseteq FL \subseteq NL^* \left(\begin{array}{l} \subseteq CFL^* \subseteq AC^1 \\ \subseteq DET \end{array} \right) \subseteq NC^2 \text{, где}$$

FL – класс функций, вычислимых детерминированной машиной Тьюринга с памятью O(log n);
NL^* и **CFL^*** – классы функций, NC^1-сводимых соответственно к множествам, принимаемым недетерминированной машиной Тьюринга с памятью O(log n), и контекстно-свободным языкам;
DET – класс функций, NC^1-сводимых к вычислению детерминанта матрицы размера n×n, составленной из n-битовых целых чисел.

Все перечисленные классы замкнуты относительно NC^1-сводимости. Кроме того, в классах **FL**, **NL^***, **CFL^***, **DET** известны полные относительно NC^1-сводимости проблемы. Так полной в **FL** является задача определения наличия цикла в данном неориентированном графе. Полными в **NL^*** являются задача построения транзитивного замыкания булевой матрицы и задача нахождения кратчайших путей в графе с положительными длинами ребер (заданными в унарной записи).

Полной в классе **DET** является задача вычисления степеней $A^2, A^3, \ldots, A^n$ данной матрицы размера n×n, составленной из n-битовых целых чисел, задача обращения A и задача вычисления произведения n таких матриц. В классе **DET** содержится большое число интересных задач из алгебры матриц и кольца полиномов над рациональными числами [95].

1.4. Вероятностные параллельные алгоритмы и класс RNC

Для решения ряда таких задач, как распознавание простоты чисел, вычисление перманента, нахождение максимального паросочетания в графе, построение дерева поиска в глубину в графе, нахождение компонент связности, обработка различных структур данных и сортировка, а также для многих других задач можно эффективно использовать датчики случайных чисел для построения эффективных вероятностных алгоритмов. Класс задач, для решения которых существуют эффективные параллельные вероятностные алгоритмы, получил название RNC (R – первая буква английского слова random – случайный).

Вероятностный алгоритм кроме обычных входных данных получает на вход некоторую случайную последовательность нулей и единиц, длина которой ограничена полиномом от длины обычного входа. Такой алгоритм вычисляет булеву функцию f, если для всякого входа x вероятность получения правильного ответа на выходе не меньше 3/4.

Обычно различают два типа вероятностных (рандомизированных) алгоритмов: вероятностные алгоритмы с односторонними ошибками и вероятностные алгоритмы с двусторонними ошибками. Если на некотором входном значении вероятностный алгоритм с односторонними ошибками выдает 1, то это всегда правильный ответ, а если выдает 0, то это верный ответ с некоторой вероятностью. Вероятностные алгоритмы с двусторонними ошибками вычисляют правильный ответ с некоторой вероятностью (вероятность ошибки меньше 1/4).

Пусть задан некоторый класс алгоритмов Ω и через $P(B)$ – обозначается вероятность осуществления события B.

Определение. *Задача вычисления семейства булевых функций* (f_n) *разрешима вероятностным* Ω*-алгоритмом с односторонней ошибкой, если найдется многочлен* q *и алгоритм* $A \in \Omega$, *на вход которого кроме обычных исходных данных* x *подается не более* q *независимых случайных битов* y *и выполнены условия*:

1) при $f(x) = 0$ $A(x,y) = 0$ для любого y;

2) при $f(x) = 1$ $P(A(x,y) = 1) \geq 1/2$.

Определение. *Задача вычисления семейства булевых функций* (f_n) *разрешима вероятностным* Ω*-алгоритмом с двусторонней*

ошибкой, если найдется многочлен q *и алгоритм* $A \in \Omega$, *на вход которого кроме обычных исходных данных* x *подается не более* q *независимых случайных битов* y *и выполнено условие*:

$$P(\ A(x,y) = f(x)\ ) \geq 3/4.$$

Класс **RNC** *состоит из функций, вычислимых вероятностными* NC*-алгоритмами с односторонней ошибкой.* Выполняется естественное включение $NC \subseteq RNC$.

В теории сложности последовательных вычислений для класса $\Omega = P$, задач разрешимых за полиномиальное время, множество задач, разрешимых вероятностными P-алгоритмами с односторонней ошибкой обычно обозначают через **RP** (random P) или просто **R**, а разрешимых Монте-Карло P-алгоритмами с двусторонними ошибками – через **BPP** (bounded probabilistic error P) [31],[141]. Однако, применительно к сложности параллельных вычислений рассматривается [95] и более широкое определение **RNC** (по аналогии с **BPP**) как класса функций, каждый бит значения которых вычислим вероятностными NC-алгоритмами с двусторонними ошибками (которые для краткости часто называются RNC-алгоритмами).

В [31] предложена конструкция, которая позволяет по вероятностному алгоритму строить неравномерное семейство детерминированных схем, имеющих близкие сложностные характеристики, то есть для заданной длины входа гарантируется существование соответствующей булевой схемы, всегда дающей правильный результат.

Пусть функция f(x) вычислима вероятностным NC-алгоритмом с односторонними ошибками. Покажем, как построить для нее схему полиномиального размера и полилогарифмической глубины, вычисляющую f(x) для всех входов x длины n. Воспользуемся эквивалентным определением NC через равномерное семейство схем. Тогда по определению вероятностного NC-алгоритма с односторонними ошибками имеется булева схема G(x,y) полиномиального размера и полилогарифмической глубины, такая, что число дополнительных входов y не превосходит полинома от n и

1) при $f(x) = 0$ $G(x, y) = 0$ для любого y;

2) при $f(x) = 1$ $P(\ G(x, y) = 1) \geq 3/4$.

Рассмотрим матрицу A размером $2^N \times 2^M$, где $N = |x|$, $M = |y|$, и $a_{ij} = 1$ тогда и только тогда, когда $G(x,y) = 1$ и $a_{ij} = 0$ в остальных случаях. В силу свойств 1 и 2 каждый столбец матрицы состоит либо из одних нулей, либо содержит не менее 3/4

M единиц. Представим детерминированную схему G(x) для вычисления f(x) в следующем виде:

$$G(x) = \bigvee_{y \in T} G(x, y),$$

где T – подмножество столбцов, покрывающих все ненулевые строки матрицы A, то есть такое, что для каждой ненулевой строки найдется столбец из T, содержащий единицу в этой строке. Нетрудно убедиться, что схема G(x) вычисляет функцию f(x).

Хорошо известно, что из условия наличия в каждом ненулевом столбце не менее константной доли единиц вытекает существование покрытия T ненулевых строк матрицы A с $|T| = O(\log M)$. Поскольку $\log M$ не превосходит полинома от длины входа n, размер схемы G(x) полиномиален по n. Производя вычисление дизъюнкции по методу сдваивания, получим, что глубина схемы G(x) не превосходит некоторого полинома от $\log n$.

Покажем теперь, что для любой функции конечного числа переменных из класса **RNC** (то есть вычислимой вероятностными NC-алгоритмом с двусторонними ошибками), как и для алгоритмов с односторонними ошибками, существует детерминированная булева схема полиномиального размера и полилогарифмической глубины.

Доказательство этого факта основано на вычислении функции в нескольких случайных точках (соответствующих случайным битам) и применении функции голосования к полученным результатам. При этом существенно используется неравенство для вероятности больших уклонений сумм бернуллиевых случайных величин.

Более точно, по определению **RNC** существует многочлен q и булева схема G(x, y) полиномиального размера и полилогарифмической глубины такая, что число дополнительных входов y не превосходит $q(|x|)$ и $P(f(x) = G(x, y)) > 3/4$. Построим новую схему $H(x) = MAJ(G(x, y_1), \ldots, G(x, y_t))$, где функция MAJ – это функция голосования, равная 1 тогда и только тогда, когда не менее половины ее аргументов равны 1. В силу известного неравенства для сумм независимых случайных величин для всякого заданного x

$$P(H(x) \neq f(x)) \leq (1/2)(3/4)^{t/2} < 2^{-q(n)}$$

при $t = O(q(n))$.

Выберем $q(n) = n$ и оценим вероятность P_S ошибочной реализа-

ции, равную вероятности того, что схема H(x) реализует функцию, не равную f(x) хотя бы на одном наборе x. В силу вышеприведенного неравенства

$$P_S \le \sum_x P(\, H(x) \neq f(x)\,) < 2^n\, 2^{-n} = 1,$$

и поэтому найдется набор входов y, при которых H(x) = f(x) на любом наборе x.

Схема H(x) имеет полилогарифмическую глубину и полиномиальный размер, поскольку функция MAJ реализуется схемой полиномиального размера и логарифмической глубины [332].

Очевидно, что таким образом строится неравномерное семейство схем, поскольку нет регулярной процедуры выбора нужных значений $y_1,\ldots y_t$, существование которых гарантируется.

В классе RNC лежат задачи отыскания максимального паросочетания, максимального потока в сети с пропускными способностями дуг, заданными в унарной записи (в которой натуральное число n представляется n единицами), дерева поиска в глубину в произвольном графе и некоторые другие задачи, для которых не известна принадлежность классу NC.

Кроме того, для ряда задач из класса NC известны параллельные вероятностные алгоритмы с лучшими характеристиками (время, число процессоров), чем детерминированные . Так, например, время работы наилучшего детерминированного параллельного алгоритма построения диаграммы Вороного на плоскости есть $O(\log^2 n)$ [34], в то время как вероятностного алгоритма – $O(\log n)$ [278].

Одной из наиболее известных задачи класса P, для которой не доказана принадлежность NC, но доказана принадлежность RNC, является задача о нахождении максимального паросочетания в графе. Напомним, что паросочетанием называется множество попарно не смежных ребер графа. Максимальным паросочетанием называется паросочетание с максимальным числом ребер.

В работах [218],[188],[242] показано, как используя датчики случайных чисел, можно за полиномиальное время найти максимальное паросочетание. Принадлежность классу NC задачи о существовании совершенного паросочетания (паросочетания, покрывающего все вершины) в двудольном графе – открытая проблема. Известна лишь принадлежность NC задачи о максимальном паросочетании для дву-

дольных регулярных графов, планарных графов и графов, не содержащих подграфов, гомеоморфных $K_{3,3}$ (три дома – три колодца), графов с не более чем полиномиальным числом паросочетаний [227], [153].

Многие задачи из класса RNC будут рассмотрены в разделе 2.1.6.

1.5. Моделирование параллельных архитектур и вложение графов

Выше основное внимание было уделено идеализированным моделям параллельных вычислений – PRAM и равномерным семействам булевых схем. Более близкой к реальным параллельным вычислительным системам моделью является *сеть процессоров с локальной памятью*, представляющих собой набор RAM, некоторые из которых соединены каналами связи. Такую сеть процессоров удобно описывать графом, в вершинах которого находятся RAM, а ребра соответствуют каналам связи между процессорами RAM. К памяти каждого процессора сети возможен только последовательный доступ: в каждый момент времени из локальной памяти данного процессора можно считать или записать только одно число. При этом возможно параллельное считывание (запись) из памяти различных процессоров. В систему команд RAM добавляются команды пересылки содержимого произвольной ячейки памяти данного процессора в любую ячейку памяти другого процессора, соединенного с данным процессором каналом связи (ребром в графе). Процессор может передавать или принимать информацию только от связанных с ним ребрами других процессоров. Для обмена данными с несмежными процессорами требуется несколько шагов обмена. Таким образом, в данной модели учитываются затраты на обмен данными между процессорами.

В такой модели параллельных вычислений естественным образом возникают две задачи:

1) задача обмена данными между различными процессорами, формализуемая в виде задачи оптимальной маршрутизации на графе;

2) задача моделирования одной архитектуры на другой, формализуемая как задача вложения одного графа в другой.

Кроме того, при изучении сравнительной вычислительной силы этой модели естественно возникает задача о сложности моделирова-

ния PRAM на сети процессоров и, в частности, упоминавшаяся в разделе 1.1 задача о моделировании параллельной памяти.

Опишем сначала основные типы графов, используемых в сетях процессоров для связи между ними, а затем рассмотрим основные результаты, относящиеся к решению перечисленных задач. Наиболее часто встречаются следующие типы графов, используемые в сетевых архитектурах:

- *цепь*
- *цикл*
- *двумерная сетка*
- *многомерная сетка*
- *тор*
- *гиперкуб* (n-*мерный булев куб*)
- *кубическая сеть* (Cube Connected Cycle - CCC)
- *полное бинарное дерево*
- *сеть совершенной тасовки*
- *сеть Бэкмана-Офмана*
- Ω-*сеть* (*бабочка*)
- *универсальная сеть*

Цепь и цикл определяются очевидным образом.

Двумерная сетка. Вершины графа задаются целыми точками прямоу- гольника (i,j), $0 \le i < n$, $0 \le j < m$. Вершина (i,j) соединена с вершинами $(i \pm 1, j)$ и $(i, j \pm 1)$, если последние принадлежат указанному прямоугольнику. Если к двумерной сетке добавить диагональные ребра $((i,j),(i \pm 1, j \mp 1))$, то возникает *гексагональная сетка.*

Тор. Если в графе сетки размера $n \times m$ добавить ребра вида $((0,j),(n-1,j))$ и $((i,0),(i,m-1))$, то получится тор. Аналогичную конструкцию можно осуществить с гексагональной сеткой. В результате получится *скрученный тор* - граф, в котором вершина

$$(i,j), \quad 0 \le i < n, \quad 0 \le j < m$$

соединена с вершинами

$(i \pm 1 \pmod n, j)$, $(i, j \pm 1 \pmod m)$,

$(i \pm 1 \pmod n, j \mp 1 \pmod m)$

Булев куб (*гиперкуб*) размерности d, $B = (0,1)^d$. Вершины этого графа соответствуют всем булевым векторам длины d. Две вершины соединены ребром, если соответствующие им булевы векторы отличаются ровно одной компонентой. *Кубическая сеть* CCC (Cube

Connected Cycle) получается при замене каждой вершины гиперкуба циклом длины d (степень каждой вершины при этом равна 3). Эта сеть используется для реализации большого числа алгоритмов.

Полное бинарное дерево - это корневое дерево, каждая внутренняя вершина которого имеет двух потомков, а все пути от корня до листьев имеют одинаковую длину.

Сеть совершенной тасовки PS_d. Вершины графа совершенной тасовки нумеруются d-разрядными двоичными числами. Вершина с номером i соединена с вершинами, номера которых получаются из i применением одной из двух операций тасовка S и обмен E: S(i) циклически сдвигает двоичную запись числа i на один разряд влево, E(i) меняет последний разряд числа i.

Сеть Бакмана-Офмана W_d. Эта сеть имеет d + 1 ярусов по 2^d вершин в каждом ярусе. Вершины последовательных ярусов разбиты на пары, которые соединены ребрами. Сеть W_d строится индуктивно: если построена сеть с d ярусами W_d, то сеть W_{d+1} получается из двух экземпляров сети W_d присоединением d + 1 яруса, состоящего из 2^{d+1} вершин, так что i-я вершина соединяется с выходами i и i + 2^d (mod 2^{d+1}). При этом выходы двух экземпляров схемы W_d нумеруются подряд числами от 0 до $2^{d+1} - 1$.

Ω-сеть (бабочка) - это конкретная реализация на элементах-коммутаторах сети W_d. С точки зрения теории графов изоморфна последней.

Универсальная сеть U_d получается последовательным соединением двух экземпляров сетей W_d, вторая из которых берется с обратной последовательностью ярусов (перевернутая сеть W_d).

Как уже отмечалось, PRAM является удобной и универсальной моделью для записи параллельных алгоритмов. Ее простота и абстрагирование от конкретных способов обмена данными при параллельных вычислениях позволяют в компактной форме представлять параллельные алгоритмы, выявлять принципиальные возможности распараллеливания. Возникает вопрос, какова же цена за переход к более реалистичной модели сети процессоров? Иными словами, с каким замедлением возможно моделирование произвольного PRAM-вычисления на сети процессоров? Еще раз отметим, что задача решается в два этапа:

1. моделированием параллельной памяти (PRAM на MPC);

2 решением задачи оптимальной маршрутизации (моделированием обменов между процессорами в полном графе на

графе ограниченной степени).

Формально на первом этапе PRAM с n процессорами и m используемыми ячейками памяти моделируется на MPC (сети процессоров с полным графом связей) с n процессорами. Как показано в [310] и в [41], соответственно, такое моделирование возможно за O(log n) с большой вероятностью, а - за O(log m) детерминированно. При этом вероятностное моделирование является равномерным, а детерминированное - неравномерным по n (для каждого n доказывается существование соответствующего моделирования). Основной прием, используемый здесь, заключается в размножении промежуточных результатов и поддержании максимально равномерного распределения их по процессорам (в локальной памяти каждого процессора).

В качестве нерешенной задачи представляет интерес построение равномерного по n детерминированного моделирования с аналогичным [41] замедлением.

Вторая задача - задача о моделировании MPC на сети процессоров ограниченной степени (включающая в себя решение задачи оптимальной маршрутизации при межпроцессорных обменах) решается путем использования сетей сортировки или сортирующих алгоритмов на сетях ограниченной степени [37], [211], гарантирующих моделирование n-процессорной MPC на n процессорной сети ограниченной степени с замедлением O(log n). Эта задача более подробно будет рассмотрена в конце настоящего раздела. Таким образом, замедление при общем моделировании (n,m)-PRAM на n-процессорной сети ограниченной степени есть величина O(log m log n). Имеется [181] и прямое моделирование n процессорной PRAM с m ячейками памяти на n-процессорной сети ограниченной степени: T шагов PRAM моделируется на сети за время O(T log m) с вероятностью, стремящейся к единице при n или T, стремящихся к единице. При этом в каждой вершине предполагается наличие буфера размера O(log n) для сохранения данных в очереди при разрешении конфликтов. В [274] предложен достаточно простой алгоритм моделирования одного шага CRCW PRAM на сети W_n за O(log n) шагов с использованием буферов лишь константного размера. Детерминированных алгоритмов моделирования с такими параметрами в настоящее время неизвестно.

Одной из важнейших задач, возникающих как при моделировании одних сетевых архитектур на других (или, как рассматривалось выше, PRAM на сети процессоров), так и при работе с реальными параллельными архитектурами, является задача быстрого обмена

данными между процессорами. Формально ее можно поставить как задачу реализации заданной перестановки на множестве вершин графа путем транспозиций по непересекающимся ребрам (бесконфликтная маршрутизация) или как задачу маршрутизации с очередями, в которой требуется выбрать маршруты движения сообщений таким образом, чтобы минимизировать время обмена, причем при возникновении конфликта (несколько сообщений одновременно требуют проход по одному и тому же ребру) проходит одно из сообщений, а остальные образуют очередь. В случае бесконфликтной маршрутизации пути во времени не пересекаются (в данный момент времени через любое ребро требует проход не более одного сообщения), но построить такие пути, вообще говоря, сложнее, чем при маршрутизации с очередями.

Рассмотрим на примере универсальной сети U_d ряд известных результатов о задаче бесконфликтной маршрутизации.

Вход. Перестановка π на множестве входов.

Выход. Множество непересекающихся путей из i в $\pi(i)$, $i = 1, \ldots, 2^d$.

Известно, что для любой перестановки существует искомое множество путей [18]. Это означает, что при построенной системе путей время обмена сообщениями между входами и выходами схемы равно $2d$. Пути на i-м и $(2d - i)$-м ярусах выбираются на основе разложения на циклы перестановки $a_j \pi^{-1} a_j \pi$, где a_j переставляет элементы j и $j + 2^{d-i}$. Сначала (при $i = 1$) для одного цикла такой перестановки входы, соответствующие π отображаются в левую половину сети, а входы, соответствующие π^{-1} – в правую. Конфликтов при этом не возникает по определению цикла перестановки и процесс продолжается по индукции. Получаем две аналогичные задачи на двух независимых сетях меньшего размера. Продолжая этот процесс, после d шагов будет построена непересекающаяся система путей.

Описанное построение использует разложение перестановки на циклы на каждом шаге, что является нетривиальной задачей и требует параллельного времени, сравнимого с глубиной сети [212].

Аналогичный результат для Ω-сети (или для сети W_d) получен в [256]. Показано, что любую перестановку можно бесконфликтно реализовать за три итерации. Здесь под одной итерацией понимается прохождение через сеть W_d, а следующая итерация начинается с полученной перестановки (выходы замкнуты на соответствующие вхо-

ды). Здесь в отличие от [18] для универсальности не требуется наличия в сети перевернутого дубликата W_d, однако при этом число дубликатов W_d увеличивается с двух до трех. Основной недостаток этих результатов для универсальных сетей состоит в том, что при реализации перестановок не учитывается время на построение непересекающихся путей. Поэтому часто используют сети, имеющие большую глубину, но с более простыми алгоритмами поиска непересекающихся путей для данной перестановки.

В [18],[54] (см. также [9]) предложен простой алгоритм построения бесконфликтной реализации перестановки за время $d(d-1)/2$. Он состоит из d этапов, каждый из которых переводит сообщения не обязательно в нужные вершины, но во все более близкие к ним так, что последний этап переводит сообщения в нужные вершины. При этом после i-го этапа сообщения оказываются в вершинах, номера которых совпадают с номерами адресатов в первых i битах. Выбор маршрута на каждом шаге при этом оказывается достаточно простым.

Формализацией изложенного выше являются различные понятия универсальных графов.

Определение [315]. *Граф* U *называется универсальным для класса* Γ, *если для любого* $G \in \Gamma$ *можно построить вершинный гомеоморфизм* ϕ: $G \Rightarrow U$ (*то есть гомеоморфизм, отображающий вершины в вершины, а дуги – в непересекающиеся по дугам пути*), *при котором входные и выходные вершины графа* G *переходят в аналогичные вершины графа* U.

Конструкция универсального графа приведена в [315].

Теорема 5.1. *Пусть* $\Gamma_k(N)$ – *множество ациклических графов на* N *вершинах с полустепенью вершин, равной* k. *Существует такой универсальный для* $\Gamma_k(N)$ *граф* $U_k(N)$, *что*:

1. *Число вершин* $U_k(N)$ *не превосходит* $5/2\ kN \log N$.

2. *в* $U_k(N)$ *имеется* N *вершин полустепени не более* k, *остальные вершины имеют полустепень не более 2.*

Если потребовать от гомеоморфизма более сильное свойство – чтобы при заданном отображении входных вершин во входные, а выходных – в выходные дуги отображались в пути, не пересекающиеся по вершинам (*сильный гомеоморфизм*), то граф $U_1(N)$ не будет универсальным

Теорема 5.2 [18] *Граф* U_n *универсален для класса* $\Gamma_1(2^n)$ *относительно сильного гомеоморфизма.*

Для того чтобы сеть обмена могла реализовывать n! различных наборов путей, необходимо, чтобы число различных ее состояний было бы не меньше n!. Если в N узлах сети расположены компараторы, которые могут принимать одно из двух возможных значений, то число различных состояний сети равно 2^N и, следовательно $N \geq \log n! = \Omega(n \log n)$. Это показывает оптимальность по порядку указанных конструкций универсальных графов.

Несколько иначе ставится проблема построения универсальной схемы, или, в терминологии работы [18], универсального автомата. В универсальной схеме настройка на моделируемую схему должна осуществляться схемным образом исходя из описания моделируемой схемы, подаваемого на управляющие входы универсальной схемы. Пусть $C(n,m,N)$ – класс булевых схем размером N с n входами, m выходами и полустепенью вершин не больше 2.

Определение [18]. *Схема* $U \in C(n + R,m,N)$ *называется универсальной для класса* $C(n,m,N)$, *если для любой схемы* $c \in C(n,m,N)$ *и любого момента времени* $t > 0$ *имеет место равенство* $c(x,t) = U(x,K,c_1 t + c_0)$, *где* K – *код схемы* c, c_1 *и* c_0 – *константы, называемые временем настройки и временем задержки соответственно.*

Теорема 5.3 [18]. *Для класса* $C(n,n,O(n))$ *существуют универсальные схемы* U^0 *и* U^1, *такие, что*

$U^0 \in C(\theta(n \log n),n,\theta(n \log^2 n))$, $c_1^0 = \theta(\log^3 n)$, $c_0^0 = \theta(\log^2 n)$, $U^1 \in C(\theta(n \log n),n,\theta(n \log n))$, $c_1^1 = \theta(\log^3 n)$, $c_0^1 = \theta(\log^3 n)$.

Справедливость следующей теоремы вытекает из стандартных мощностных соображений.

Теорема 5.4 [18]. *Для класса* $C(n,n,O(n))$ *не существует универсальной схемы* U, *для которой или размер равен* $o(n \log n)$ *или время настройки равно* $o(\log n)$.

В [96] рассмотрены схемы, универсальные в классе схем с фиксированной глубиной.

Теорема 5.5. *Если* $N \geq n$ *и* $N \geq d \geq \log N$, *то для схем класса* $C(n,n,N)$, *имеющих глубину не больше* d, *существует универсальная схема размером* $O(N^3 d/\log N)$ *и глубиной* $O(d)$.

При более реалистичной постановке задачи маршрутизации, допускается пересечение путей во времени за счет организации очередей в вершинах. В такой модели Бребнер и Валиант [72] предложили рандомизированный алгоритм маршрутизации на гиперкубе,

который заканчивает работу за время 8d с вероятностью, не меньшей $1 - (3/4)^d$. Недавние результаты (см. [316]) показывают, что с вероятностью, стремящейся к единице, алгоритм Бребнера-Вэлианта заканчивает работу за время, асимптотически равное 2d. Многочисленные исследования посвящены также вопросу о пропускной способности сети W_d (или Ω-сети) при передаче через нее случайных запросов (перестановок).

Принципиально вопрос о существовании универсальных схем с простым управлением решен в [37] путем построения сети сортировки с $O(n \log n)$ компараторами, имеющей глубину $O(\log n)$. Действительно, сеть сортировки можно рассматривать как перестановочную сеть, управление в которой осуществляется очевидным образом – локально и просто: два пришедших в компаратор сообщения сравниваются и подаются на выход в порядке возрастания (убывания). На выходе сети сообщения оказываются на нужных местах по определению сортирующей сети. Поскольку в [210] показано, что алгоритм из [37] можно реализовать на графах с ограниченной степенью, то для таких графов можно считать принципиально решенной задачу оптимальной маршрутизации: любая перестановка может быть осуществлена бесконфликтно и с учетом времени построения непересекающихся путей за время $O(\log n)$. Другая сеть ограниченной степени с константными буферами в вершинах для решения задачи маршрутизации за время $O(\log n)$ предложена Пиппенджером [259]. Однако, остается интересная для практических приложений задача построения достаточно простых алгоритмов маршрутизации на ранее упомянутых графах – гиперкубе, бабочке, решетке, поскольку мультипликативные константы в конструкции [37] очень большие. Кроме того, неизвестно хорошее моделирование сети Лейтона на гиперкубе и, как показано в [102], сеть Лейтона не может быть смоделирована на сети совершенной тасовки и кубической сети за время, по порядку меньшее $\log^2 n$, что совпадает с временем сортировки по Бэтчеру, реализуемой на гиперкубе как раз за такое время. Наилучшая детерминированная сортировка (а следовательно и маршрутизация) на гиперкубе за время $O(\log n (\log \log n)^2)$ предложена в [103].

Большое количество типов сетей процессоров (деревья, решетки, гиперкубы, тасующие схемы, бабочки) делает актуальной задачу переноса алгоритмов с одной архитектуры на другую и задачу моделирования одной архитектуры на другой. Задача моделирования од-

ной сетевой архитектуры на другой в терминах графов формализуется следующим образом [239].

Определение. *Отображением* f *графа* G = (V,E) *в граф* H = (W,F) *называется пара отображений* f_v: V ---> W *и* f_e: E ---> P(H), *где* P(H) – *множество путей в* H *и для несмежных ребер* e_1 *и* e_2 *графа* G *их образы* $f_e(e_1)$ *и* $f_e(e_2)$ *являются непересекающимися по вершинам путями.*

Определение. *Растяжением* d *отображения* называется максимум из длин путей, в которые переходят ребра, а отношение r = |W|/|V| называется *расширением*. Отображение называется *вложением*, если f_v – однозначное отображение и называется *изоморфным вложением*, если растяжение при этом равно единице.

Если имеется вложение одного графа в другой с растяжением d, то это можно интерпретировать как возможность моделирования архитектуры, соответствующей первому графу, на второй архитектуре с замедлением d, поскольку обмен данными по одному ребру заменяется не более чем d обменами по ребрам соответствующего пути. Величина расширения при этом характеризует возрастание числа процессоров при переходе от первой архитектуры ко второй. Поэтому при сравнении возможностей различных параллельных архитектур рассматривается задача вложения одного класса графов в другой с минимально возможными растяжением и расширением.

Вообще говоря, определение, существует ли для двух заданных графов вложение одного в другой, является трудной задачей. Например определить, существует ли вложение данного дерева в гиперкуб, является NP-полной задачей [330].

Приведем примеры известных результатов о сложности вложения некоторых архитектур в гиперкуб:

Цепь (цикл) —> гиперкуб.

Имеется простое (см. лемму 5.1) вложение с параметрами d = 1, r = 1 (при длине цепи, являющейся степенью двойки).

Двумерная сетка (тор) —> гиперкуб.

Прямое произведение вложений цепей дает вложение с параметрами d = 1, r = 1 (если размеры сетки – степени двойки).

Многомерная сетка —> многомерная сетка.

Заданы две многомерные сетки:

$$G = g_1 \times \ldots \times g_d \qquad \text{и} \qquad H = h_1 \times \ldots \times h_d,$$

причем, $g_1 \ldots g_d \le h_1 \ldots h_d$. Можно отобразить G в H со средним

растяжением $\alpha = \max_{1 \le i \le d} (g_{i+1} \cdots g_d / h_{i+1} \cdots h_d)^{1/i}$.

Возможно вложение *полного бинарного дерева глубины* n *в гиперкуб размерности* n + 1 с параметрами d = 2 и r = 1 + 1/(2^{n+1} − 1).

Возможно вложение Ω-*сети глубины* n *в гиперкуб* размерности n + ⌈log n⌉ с параметрами d = 1 и r = $2^{\lceil \log n \rceil - \log n}$ [239].

Приведенные результаты касаются отображений более "бедных" архитектур в более "богатые", поэтому в этой ситуации можно обойтись вложениями. При рассмотрении обратных ситуаций (например, при моделировании Ω-сети на плоской сетке или при моделировании большей сети на меньшей) естественным образом возникает необходимость использования отображений общего вида, не являющихся вложениями (при которых в одну вершину может переходить несколько вершин исходного графа).

Лемма 5.1. *Плоская сетка размером* $2^m \times 2^n$ *может быть вложена в* B^{m+n} *– гиперкуб размерности* n + m

Доказательство. Покажем вначале, что цепь длиной 2^m может быть вложена в куб размерности m. Это очевидно при m = 1. Пусть имеется вложение цепи длиной 2^{m-1} в куб размерности m − 1. Вложим два экземпляра куба размерности m − 1 с фиксированными вложениями цепей в параллельные грани m-мерного куба и соединим свободные концы цепей, расположенных в противоположных гранях, ребром. Очевидно, эта конструкция определяет вложение цепи длины 2^m в m-мерный куб.

Пусть теперь заданы вложения цепей длиной 2^m и 2^n в кубы размерностью m и n соответственно. Так как сетка размером $2^m \times 2^n$ вкладывается в произведение цепей, то произведение вложений цепей дает вложение сетки в произведение кубов размерностей m и n, что, очевидно, является кубом размерности m + n.

Эта конструкция легко может быть распространена на сетки произвольной размерности, а также многомерные торы, являющиеся произведениями циклов.

Лемма 5.2. *Сеть Бакмана–Офмана* W_n *может быть отображена в* B^n *с растяжением* 2.

Доказательство. Нетрудно видеть, что приводимая ниже конструкция обладает нужными свойствами. Отображаем W_1 в пару вершин, соединенных ребром очевидным образом. Пусть построено отображение W_n в B^n. Возьмем две копии образа W_n и разместим их в

параллельных гранях куба B^{n+1}. Вершины (n + 1)-го яруса сети W_{n+1} отобразим в соответствующие вершины этих двух копий W_n так, чтобы выполнялось свойство 1. Отобразим элементарные бабочки, соединяющие вершины последнего и предпоследнего ярусов схемы W_{n+1} в ребра, соединяющие те параллельные грани, в которые были вложены две копии W_n.

Заметим, что в построенных вложениях многомерных сеток и сетей Ваксмана-Офмана в многомерный куб ребра последнего используются для обменов неэффективно в отличие от сетей совершенной тасовки PS и кубических сетей CCC.

В [329] показано, что бинарное дерево на N вершинах может быть вложено в булев куб размерности O(log N loglog N).

Лемма 5.3 [25]. *Сеть совершенной тасовки PS_d не может быть вложена в куб размерности d так, чтобы ребра переходили в ребра, даже если отбросить две петли, имеющиеся в PS_d.*

Конструкция, использованная для вложения пути длиной 2^d в d-мерный куб, имеет важное значение для построения отображения и других сетей в d-мерный куб. Остановимся на ней подробнее. Пусть $s_d(0, \ldots, 2^d - 1) \to B^d$ есть построенное выше отображение пути в d-мерный куб. Тогда отображение s_{d+1} строится следующим образом:

$$s_{d+1}(i) = \begin{cases} 0s_d(i), & \text{если } i < 2^d, \\ 1s_d(2^{d+1}-i), & \text{если } i \geq 2^d. \end{cases}$$

Отображение s_d задает так называемый код Грэя.

Определение. *Кодом Грея размерности d называется перестановка чисел от 0 до $2^d - 1$, при которой соседние числа переходят в числа, отличающиеся точно в одном разряде. (Числа называются соседними, если они либо последовательные, либо составляют пару $(0, 2^{d-1})$.)*

Примером использования кодов Грея для реализации конкретных алгоритмов на сетевых архитектурах является следующее

Утверждение [25]. *Если элементы данных для БПФ (быстрого преобразования Фурье) распределить по вершинам куба согласно коду Грея, то при выполнении БПФ на первом такте данные будут проходить расстояние, равное 1, а на остальных тактах – расстояния, равные 2.*

Напомним, что *дерево – это связный граф без циклов. Корневое дерево – это дерево с выделенной вершиной, которая*

называется корнем. Листом дерева, или висячей вершиной, называется вершина со степенью один. В ориентированном дереве все дуги ориентированы от корня к листьям или от листьев к корню. Высотой дерева называется максимум длин ориентированных путей. k-ичным деревом называется корневое дерево, у которого корень имеет степень k, *а все нелистья - степень* k + 1. *Полным k-ичным деревом называется дерево, у которого расстояния от корня до всех листьев равны между собой.* Полное k-ичное дерево высотой n имеет $(k^n - 1)/(k - 1)$ вершин, из которых k^n составляют листья.

В таких приложениях, где используются бинарные деревья, как базы данных, словари, запись поступает через корень и, проходя обработку по ключевому слову, опускается до нужного листа, где сравнивается с имеющейся в нем записью и после обработки остается на хранение. Выборка по ключевому слову происходит в обратном порядке. При такой работе на каждом такте оказываются загруженными log N процессоров (вершин дерева). Для повышения загруженности можно использовать "толстое дерево" [25], то есть дерево, в котором число каналов, ведущих к вершине k-го яруса, равно 2^{n-k}.

Определение. *Снеп-деревом* ST_n *называется полное бинарное дерево высотой* n, *дополненное* 2^n *дугами (снеп-дугами), ведущими от листьев. Из каждого листа выходит пара снеп-дуг, в корень заходят две снеп-дуги, а в остальные вершины - по одной снеп-дуге. Снеп-дерево называется циклическим, если в нем имеются два непересекающихся по дугам гамильтоновых цикла.*

Теорема 5.6 [213]. *Существует отображение полного бинарного дерева высотой* m *в такое циклическое снеп-дерево , что соседние вершины отображаются в соседние и разность числа прообразов вершин снеп-дерева не превосходит* 1.

В настоящее время известно большое число реализаций конкретных параллельных алгоритмов на различных сетевых архитектурах, однако их рассмотрение выходит за рамки настоящего обзора.

Часть 2. Параллельные алгоритмы

Параллельные алгоритмы, как и вообще многие быстрые алгоритмы, обычно строятся с помощью искусного сочетания различных приемов. Сначала предлагаются некоторые новые математические тождества и соотношения, а затем на их базе с использованием некоторых стандартных приемов строятся быстрые параллельные алгоритмы. В этой части будут рассмотрены некоторые приемы построения параллельных алгоритмов, а также методы распараллеливания последовательных алгоритмов. В следующих разделах будут приведены соотношения, используемые в эффективных параллельных алгоритмах алгебры, целочисленной арифметики, рядах и многочленах, комбинаторике и дискретной оптимизации, теории графов, вычислительной геометрии, сортировке и поиске, теории расписаний.

При этом для каждой рассматриваемой задачи обычно приводятся наилучшие из известных параллельных алгоритмов. Необходимо отметить, что понятие оптимального параллельного алгоритма означает лишь оптимальное распараллеливание последовательного алгоритма (в смысле равенства по порядку произведения параллельного времени на число процессоров последовательному времени работы алгоритма). Поэтому для каждой задачи целесообразно распараллеливать последовательные алгоритмы с минимальным временем работы. Кроме того, в ряде предметных областей, в первую очередь дискретной оптимизации, теории расписаний, теории графов, для многих задач неизвестны полиномиальные алгоритмы. Поэтому в этих областях часто распараллеливаются приближенные полиномиальные алгоритмы с гарантированными оценками точности получаемого решения или наиболее известные алгоритмы для полиномиально разрешимых частных случаев задачи.

Все приводимые далее алгоритмы будем рассматривать в предельно распараллеленной форме. Если число процессоров ограничено и задано в виде параметра p, и известно время T_q работы алгоритма на q процессорах, то, используя упомянутую ранее теорему Брента, можно получить время вычисления на p процессорах в следующем виде (при $p \le q$)

$$T_p \le \lceil q/p \; T_q \rceil.$$

2.1. Общие методы распараллеливания

2.1.1. Метод сдваивания и префиксная техника

Алгоритм сдваивания для выполнения суммирования (или любой другой ассоциативной операции) $N = 2^n$ элементов $a = \sum_{i=1}^{N} a_i$ реализуется за время $O(\log N)$ на EREW PRAM с $N/\log N$ процессорами следующим образом. Слагаемые разбиваются на $N/\log N$ блоков по $\log N$ элементов. Каждому блоку назначается один процессор, который и производит последовательное суммирование элементов блока за время $O(\log N)$. Затем число частичных сумм, равное $N/\log N$, уменьшается вдвое при их параллельном попарном сложении. Выполняя $O(\log N)$ таких шагов, получим одно число, которое и является искомой суммой.

Этот метод можно применить, например, к вычислению произведения m матриц порядка n и получить алгоритм с временем работы $O(\log n \log m)$ на EREW PRAM с $O((m/\log m)(n^3/\log n))$ процессорами. Отметим, что применение быстрых алгоритмов умножения матриц с последовательным временем $O(n^\omega)$ приводит к параллельным алгоритмам с временем $O(\log n \log m)$ и числом процессоров $O((m/\log m)(n^\omega/\log n))$ (см. следующий раздел).

Сдваивание путей.

Метод сдваивания является наиболее простым и общим приемом распараллеливания не только численных вычислений, но и комбинаторных алгоритмов, связанных с обработкой структур данных и допускающих теоретико-графовое описание. В этом случае он обычно называется методом сдваивания путей.

Рассмотрим его применение на примере задачи поиска для любой вершины ориентированного леса корня дерева, которому принадлежит данная вершина. Будем считать, что ребра деревьев ориентированы от листьев к корню, вершины хранятся в памяти и в каждой вершине имеется ссылка на последователя в дереве. Припишем каждой вершине свой процессор. На первом шаге в каждой вершине соответствующий процессор вычисляет вершину-последователь на расстоянии 2, переписывая для этого ссылку из своего непосредственного последователя и заменяя ею существующую ссылку на непосредственного последо-

вателя. На втором шаге каждый процессор снова переписывает ссылку из непосредственного последователя и заменяет ею старую ссылку. Таким образом, каждая вершина знает своих последователей на расстоянии 4 после второго шага, на расстоянии 8 после третьего шага и т.д. Принципиально важно, что имеется точка отсчета в виде корня дерева, и если некоторая вершина получает ссылку на нее, то далее эта ссылка не изменяется, так как у корня нет последователей. После log n шагов, где n – число вершин в лесе, каждая вершина будет содержать информацию о корне того дерева, которому она принадлежит. Таким образом, за время O(log n) на n-процессорной CREW PRAM можно решить рассмотренную задачу описанным методом сдваивания путей.

Параллельное вычисление префиксов.

Задача о префиксе заключается в вычислении для заданных значений $a_1,\ldots,a_n$ и любой ассоциативной операции $\bullet$ всех префиксов вида: $p_1 = a_1$, $p_2 = a_1 \bullet a_2,\ldots,p_n = a_1 \bullet \ldots \bullet a_n$. Известен простой параллельный алгоритм [208] решения задачи о префиксе на EREW PRAM с O(n/log n) процессорами за время O(log n). Он заключается в следующем.

1. Для каждого элемента с четным номером i вычислить величину $x_{i/2} = a_{i-1} \bullet a_i$.

2. Рекурсивно вычислить префиксы в массиве x_i и занести их в массив y_i.

3. Для каждого четного i положить $p_i = y_{i/2}$, при нечетном $i \neq 1$ положить $p_i = y_{(i-1)/2} \bullet a_i$.

Шаги 1 и 3 на n процессорах выполняются за константное время. Рекурсивный шаг 2 выполняется для задачи вдвое меньшего размера. Общее число выполнений рекурсивного шага 2 равно O(log n), поэтому время работы алгоритма на n процессорах есть величина O(logn). В действительности же достаточно O(n/log n) процессоров, поскольку на каждом очередном рекурсивном шаге число элементов уменьшается вдвое и, следовательно, общее число операций есть O(n), а по теореме Брента такие вычисления можно проделать на p процессорах за время O(log n) + O(n)/p. Подставляя в это выражение значение p = O(n/ log n), получаем, что время вычисления есть O(log n).

Имеется модификация этого алгоритма для модели CRCW PRAM с временем работы O(log n/ log log n), где исходный массив состоит из целых чисел с длиной записи O(log n) и $\bullet$ является операцией

обычного сложения [276], [158].

Отметим, что префиксные максимумы можно вычислить за время O(loglog n) на WEAK CRCW PRAM с O(n) процессорами [116]. Заметим также, что необходимым условием применимости описанного алгоритма является задание данных в виде линейно упорядоченного массива, поскольку для каждого элемента нужно сразу определить, четный или нечетный у него номер в данном массиве. Алгоритмы обработки данных, заданных в виде связного списка, будут рассмотрены в последующих разделах.

2.1.2. Матричная техника

Решение многих задач может быть сведено к операциям над матрицами. Кроме многочисленных задач линейной алгебры таковыми являются задачи о кратчайших путях во взвешенном графе, транзитивном замыкании орграфа, построении дерева поиска в ширину для графа и другие. В зависимости от операций, выполняемых с элементами матрицы, выделяют два типа матричного умножения - арифметический и булев. В первом случае используются операции сложения и умножения над числами, а во втором - операции конъюнкции и дизъюнкции булевых элементов.

Для заданных матриц A размером m×n и B размером n×p произведение C = AB, определяемое по формуле

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk} \qquad (2.1)$$

может быть вычислено за время O(log n) на CREW PRAM с O(nmp/log n) процессорами и за время O(log max(n,p)) на EREW PRAM с O(nmp/log n) процессорами. В первом случае это делается непосредственным применением изложенного выше метода сдваивания к формуле (2.1), а во втором применение того же метода сдваивания предваряется размножением матрицы A для получения p копий.

Для булевого произведения матриц справедливы те же оценки для моделей EREW и CREW PRAM с тем же числом процессоров, а на модели WEAK CRCW PRAM с O(nmp) процессорами задача решается за константное время, поскольку дизъюнкция n переменных за константное время может быть вычислена на n-процессорной WEAK CRCW PRAM следующим образом: в ячейку-результат заносится единица, каждой переменной назначается один процессор, который записывает в ячейку-результат 0 тогда и только тогда, когда соответст-

вующая переменная равна 1. Затем из единицы вычитается полученный результат, что и выдается в качестве ответа. Как было отмечено ранее, дизъюнкция n переменных не может быть вычислена на CREW PRAM с произвольным числом процессоров по порядку быстрее, чем за log n.

Это отличие в сложности вычисления дизъюнкций и конъюнкций на моделях CRCW с одной стороны и CREW и EREW PRAM с другой объясняет, почему многие графовые алгоритмы на модели CRCW часто вычисляются в O(log n) раз быстрее, чем на CREW и EREW PRAM.

Применение быстрых алгоритмов умножения двух n×n матриц с последовательным временем $O(n^{\omega})$ приводит к параллельным алгоритмам с временем O(log n) и числом процессоров $O(n^{\omega}/\log n)$. Это вытекает из следующего утверждения [151], [20]:

Если имеется последовательный алгоритм умножения матриц с числом операций $O(n^{\omega})$, *то на* p *процессорах* PRAM *задача может быть решена за время* $O(n^{\omega}/p + \log n)$.

Доказательство утверждения основывается на некоторых свойствах билинейных программ. Билинейной программой, вычисляющей набор билинейных форм $b_i(x_1,\ldots,x_n;y_1,\ldots,y_m)$, $i = 1,\ldots,q$, над кольцом, называется произвольная неветвящаяся программа (алгебранческая схема), состоящая из трех частей:

1. Вычисление линейных комбинаций

$$l_i(x_1,\ldots,x_n),\ k_i(y_1,\ldots,y_m),\ i = 1,\ldots,r.$$

2. Вычисление r произведений

$$a_i = l_i(x_1,\ldots,x_n)\cdot k_i(y_1,\ldots,y_m),\ i = 1,\ldots,r.$$

3. Вычисление q линейных комбинаций

$$b_j = \sum c_{ji}a_i,\quad j = 1,\ldots,q.$$

Известно, что любая неветвящаяся программа умножения матриц с мультипликативной сложностью r может быть представлена в виде билинейной программы с r умножениями [20], [302].

Рассмотрим M×M-матрицы, где $M = n^k$. Построим индуктивно алгебранческую схему умножения таких матриц. Заметим, что по билинейной программе умножения матриц над кольцом R можно построить билинейную программу умножения тех же матриц над кольцом матриц размером m×m над R причем с той же мультипликативной сложностью. Для этого достаточно каждый из коэффициентов линейных комбинаций в исходной билинейной программе заменить на диа-

гональную матрицу размером $m \times m$ с этим коэффициентом на диагонали.

Теперь применим эту процедуру к матрицам порядка n над кольцом матриц размером $n^{k-1} \times n^{k-1}$ и заметим, что необходимые умножения матриц размером $n^{k-1} \times n^{k-1}$ можно выполнить параллельно. Поэтому имеем соотношения:

$$t(k) \le t(k-1) + c(n),$$
$$p(k) \le \max(n^{\omega} p(k-1), c_1(n) n^{2k-2}),$$

где $t(k)$ – глубина схемы умножения матриц размером $n^k \times n^k$, а $p(k)$ – ее размер, $c(n)$ и $c_1(n)$ – некоторые константы.

Учитывая соотношение $n^{\omega} > n^2$, из этих рекуррентных неравенств следует, что

$$t(k) \le c(n)k = O(\log M), \qquad p(k) = O(n^{k\omega}) = O(M^{\omega})$$

Поскольку все линейные комбинации на первом и третьем этапах при достаточном числе процессоров можно вычислять параллельно, приходим к оценкам $c(n) = O(\log n)$, $c_1(n) = O(n^{\omega+2} + n^{2\omega+1})$.

Переход к модели PRAM легко осуществляется на основе известной связи между PRAM-вычислениями и схемными вычислениями (см. раздел 1.2).

Для EREW PRAM можно вычислять $O(n^{\omega})$ линейных комбинаций длины n^2 на первом этапе и n^2 линейных комбинаций длины $O(n^{\omega})$ на третьем этапе билинейной программы последовательно. При этом параллельно выполняются только операции сложения пар матриц за константное время на $O(n^{2k-2})$ процессорах.

Конечно, приведенные оценки констант можно улучшить, но нам важен здесь лишь принципиальный вывод о возможности эффективного распараллеливания быстрых алгоритмов умножения матриц.

Умножение теплицевой матрицы на вектор.

Теплицева матрица – это матрица, в которой элементы, стоящие на каждой диагонали, параллельной главной, одинаковы. Для умножения теплицевой матрицы на вектор известен параллельный алгоритм с временем работы $O(\log n)$ на $O(n)$ процессорах. Это можно сделать с помощью параллельного вычисления быстрого преобразования Фурье (БПФ) (см. раздел 2.2.2), если использовать тот факт, что любая теплицева матрица может быть представлена как сумма циркулянта и антициркулянта. (Матрица C называется циркулянтом, если $c_{ij} = c_{pq}$ при $i - j \equiv p - q \bmod n$, и называется антициркулянтом, если она становится циркулянтом после изменения знака

всех поддиагональных элементов). Умножение на циркулянт есть вычисление свертки, а умножение на антициркулянт - вычисление отрицательно обернутой свертки [2]

2.1.3. Метод "разделяй и властвуй".

Метод заключается в разбиении задачи на подзадачи меньшего размера, параллельном решении этих подзадач и конструировании из решений подзадач решения основной задачи. Метод дает NC-алгоритм, если размеры подзадач не превышают cn (n - размер исходной задачи и константа c < 1) и время слияния (конструирования общего решения из решений подзадач) не превосходит полинома от логарифма размера задачи

Область применимости этого метода для построения параллельных алгоритмов чрезвычайно широка она включает в себя многие задачи вычислительной геометрии, теории графов, алгебры, комбинаторной оптимизации и других областей

Примерами применения этого метода являются конструкции параллельных алгоритмов нахождения максимума и минимума n чисел, построения выпуклой оболочки на плоскости с временем работы $O(\log n)$ [34] и построения диаграммы Вороного на плоскости с временем работы $O(\log^2 n)$ [34], вычисления всех префиксных сумм [208], асимптотически оптимального решения задачи о максимальном независимом множестве в планарном графе на основе теоремы о сепараторах [215]

Рассмотрим использование методики "разделяй и властвуй" при нахождении максимума n чисел. В основе параллельного алгоритма нахождения максимума лежит процедура Reduce(r), которая сводит задачу для n чисел к задаче для n/r чисел с использованием rn операций. Числа разбиваются на n/r блоков размером r и для каждого блока с использованием r^2 процессоров на блок находится максимум за константное время в каждом блоке. Максимум всех n чисел равен максимуму среди максимумов в n/r блоках.

Положим $m = \log \log_k n$ (k - некоторый произвольный параметр) и произведем $\log \log \log_k n$ обращений к процедуре Reduce(2), сводя общее число значений к m. Поскольку каждая следующая итерация использует в два раза меньше операций, общее число операций есть $O(n)$. Далее осуществим $\log \log_k m$ итераций, каждая из которых выполняется за константное время. На i-й итерации применяется процедура $\text{Reduce}(k^{2^{i-1}})$. Число процессоров, необходимых на i-й

итерации, есть $O(k^{2^{i-1}} m / k^{2^{i-1}-1}) = O(km)$ и общее число процессоров есть снова $O(km)$.

Таким образом, получен алгоритм нахождения максимума n чисел на WEAK CRCW PRAM за время $t = O(\log \log_k n)$ на $O(nk/t)$ процессорах [293]. Интересно отметить, что на CREW PRAM нахождение максимума n чисел требует времени $\Omega(\log n)$ даже если число процессоров не ограничено и существует простой алгоритм с таким временем работы на EREW PRAM [187].

Рассмотрим обобщенную задачу о префиксах, заключающуюся в вычислении для данного множества элементов $a_1,\ldots,a_n$ всех сумм по заданной системе подмножеств $S = (S_i,\ i \in I)$: $Z_i = \sum_{j \in S_i} a_j$.
Стандартная задача о префиксах заключается в вычислении таких сумм (или других ассоциативных операций) по системе подмножеств специального вида: $S_i = (1,2,\ldots,i)$, $i \in I = (1,2,\ldots,m)$.

Тривиальным алгоритмом решения этой задачи является независимое вычисление каждой из m требуемых сумм по методу сдваивания за время $O(\log n)$. Для этого достаточно $O(m\ n/\ \log n)$ процессоров. Вопрос заключается в том, можно ли вычислить требуемые суммы за такое же по порядку время на меньшем числе процессоров (на $O(\max(n,m))$ процессорах)?

Пусть семейство обладает следующим наследственным свойством: существует такое разбиение исходного множества на два (бисекция), что величины каждого из этих множеств отличаются не более чем на единицу и число различных подмножеств в каждом из двух семейств, на которые распадается система $S = (S_i,\ i \in I)$ не превосходит $|I|/2$ и пусть $m = |I|$, $N = \max(n\ ,\ m)$. Тогда, если указанное разбиение можно построить параллельно за константное время, получим рекуррентное соотношение для параллельного времени $t(n,m)$ решения обобщенной задачи о префиксах:

$$t(n,m) \le t(n/2, m/2) + O(1),$$

что приводит к оценке вида $t(n) = O(\log n)$ на N процессорах CREW PRAM (поскольку при при бисекции суммарное число подмножеств не возрастает).

Специальным случаем обобщенной задачи о префиксах является случай инвариантных относительно бисекций систем подмножеств, когда в результате бисекции получаются системы подмножеств, аналогичные исходной и число m подмножеств в исходном семействе не меньше n. Примерами таких систем подмножеств являются все под-

множества с числом элементов, не превышающим k при заданном $1 \le k \le n$, система префиксов или суффиксов данного множества и другие. Для всех них справедливы приведенные выше оценки времени и числа процессоров.

В [300] дается решение обобщенной задачи о префиксах, в случае, когда система подмножеств задается последовательностью элементов $y(i)$, $1 \le i \le n$ так, что k-е подмножество состоит из тех $j < k$, для которых выполнено $y(j) < y(k)$. Для этого случая известно решение на $O(n)$ процессорной CREW PRAM за время $O(\log n)$.

Другим примером применения методики "разделяй и властвуй" является решение задач на графах с использованием сепараторов. Известно, что в любом планарном n-вершинном графе имеется вершинный сепаратор размером $O(\sqrt{n})$ [215], то есть множество вершин, удаление которых разбивает граф на компоненты размером не более 2/3 n. Более общее утверждение заключается в существовании сепаратора размером $O(\sqrt{n/\varepsilon})$, разбивающего граф на компоненты размером не более εn. Эта теорема имеет многочисленные приложения. В частности, в [216] показано, как можно использовать сепараторы для нахождения максимального независимого множества вершин в планарном графе. Изложим сейчас параллельную версию этого алгоритма.

Находим ε-сепаратор размером $O(n/\sqrt{k(n)})$ с $\varepsilon = k(n)/n$. В каждой связной компоненте полным перебором находим максимальное независимое множество вершин. Выберем в качестве приближенного решения объединение этих множеств. Ясно, что построенное решение отличается от оптимума не более, чем на размер ε-сепаратора. Таким образом, относительное отклонение полученного решения от оптимума не превышает $O(1/\sqrt{k(n)})$, поскольку оптимум не меньше n/4 (граф раскрашиваем в 4 цвета). Выбирая $k(n) = \log n$, получим параллельный алгоритм нахождения $O(1/\sqrt{\log n})$-оптимального решения о максимальном независимом множестве с временем работы $O(\log n)$ на $O(n)$ процессорах при построенном ε-сепараторе.

Возможность применения этой теоремы в параллельных вычислениях опирается на существование параллельного NC-алгоритма нахождения таких сепараторов за время $O(\log^2 n)$ [133]. Основная трудность заключается в построении обычного сепаратора, так как

ε-сепараторы строятся из них рекурсивно. Действительно, после k-го шага величина максимальной компоненты не превышает $(2/3)^k n$. Эта величина не превосходит εn при $k = \lceil \log_{3/2}(1/\varepsilon) \rceil = O(\log n)$. Таким образом, число итераций - логарифмическое по порядку. Обозначим размеры построенных сепараторов через $K_1, \ldots, K_t$. Пусть $n_1, \ldots, n_t$ - размеры компонент, на которые распадается граф при удалении сепараторов. Учитывая, что $\sum_{j=1}^{t} n_j \le n$, имеем

$$\sum_{j=1}^{t} K_j \le c \sum_{j=1}^{t} \sqrt{n_j} \le c \sum_{j=1}^{t} \sqrt{n/t} \le c \sqrt{nt}.$$

Поскольку $t \le 2^k$ и $(2/3)^{k-1} n \ge \varepsilon n$, то $k \le 1 + \log_{3/2}(1/\varepsilon)$ и $t = O(1/\varepsilon)$. Отсюда сразу вытекает, что суммарный размер ε-сепаратора есть величина $O(\sqrt{n/\varepsilon})$.

Описанная параллельная процедура в сочетании с параллельным алгоритмом построения сепаратора позволяет строить NC-алгоритмы асимптотически точного решения многих NP-полных задач в планарных графах. Выше мы проиллюстрировали это на примере задачи о максимальном независимом множестве. На самом деле, требуется лишь, чтобы оптимум был по порядку не меньше n и объединение допустимых решений из непересекающихся подграфов было бы допустимым решением. Этими свойствами обладают, например, задачи о нахождении максимальных подграфов, не содержащих заданного подграфа.

2.1.4. Распараллеливание схем ограниченной степени

Пусть G = (V,A) - схема над коммутативным полукольцом, вычисляющая значение функции f. Степень вершины схемы определяется индуктивно: листья имеют степень 1, степень вершины сложения равна максимуму степеней вершин-предшественников, степень вершины умножения равна сумме степеней вершин-предшественников. Степень схемы есть максимум из степеней вершин.

Теорема 2.1 [233]. *Пусть имеется схема размером* N *и степени* d, *вычисляющая значение функции* f. *Тогда для вычисления значения* f *существует эквивалентная схема глубиной* $O(\log(N\ d)\ \log\ N)$ *и размером* $M(N)$, *где* $M(n)$ - *число умножений, достаточное для умножения двух матриц порядка* n. *При этом имеется простой алгоритм преобразования исходной схемы в параллельную.*

Этот общий метод позволяет распараллелить многие алгебра-

ические вычисления, например, вычисление определителя, решение систем линейных уравнений, поскольку определитель есть многочлен n-й степени от n^2 переменных, который может быть вычислен последовательным алгоритмом за время $O(n^{\omega})$. Кроме того, поскольку частным случаем схем над полукольцом являются булевы схемы, задача о значении булевой схемы степени d и размером n может быть решена за время $O(\log n(\log nd))$ на PRAM с M(n) процессорами.

Алгоритм вычисления значения схемы состоит из циклического обращения к процедуре ФАЗА, состоящей, в свою очередь, из трех процедур: MM(U), ADD(U), MULT(U), где $U = (u_{ij})$ – матрица, представляющая схему. Элемент u_{ij} равен весу дуги, ведущей от i-го элемента к j-му. Процедура MM вычисляет матрицу

$$U = U(X,+) \cdot U(+,+) + U(X,X),$$

где матрица U(+,+) совпадает с U в позициях, которые соответствуют дугам, соединяющим вершины сложения, и содержит нули в остальных позициях; матрица U(X,+) совпадает с U в позициях, которые соответствуют дугам, ведущим в вершины сложения и содержит нули в остальных позициях; матрица U(X,X) совпадает с U в позициях, которые соответствуют дугам, соединяющим вершины, хотя бы одна из которых не является сложением, и содержит нули в остальных позициях.

Процедура ADD(U) для всех вершин v_j, предшественники которых являются листьями, вычисляет значения в этих вершинах по формуле

$$\text{value}(v_j) = \sum_{i=1}^{n} \text{value}(v_i) \cdot U_{ij},$$

полагает равными нулю элементы U_{ij} и отбрасывает всех предшественников v_j.

Процедура MULT(U) для всех вершин умножения, оба предшественника v_i и v_k которых являются листьями, вычисляет значение

$$\text{value}(v_j) = \text{value}(v_k) \cdot \text{value}(v_i),$$

полагает равными нулю элементы U_{kj} и U_{ij} и отбрасывает v_i и v_k. Затем MULT(U) пересчитывает значения, соответствующие дугам:

$$F_{ji} = \text{value}(v_k) \cdot U_{ji},$$

где U_{μ} – дуга, ведущая из вершины умножения v_j причем v_k – лист, а v_i – не лист, и для всех пар (i, i) вычисляет

$$W_{ii} = \sum_j F_{ij\mu}, \quad U_{ii} = U_{ii} + W_{ii}, \quad \text{где} \quad U_{\mu\mu} = 0,$$

Проверкой по индукции можно убедиться, что описанная процедура ФАЗА осуществляет эквивалентное преобразование исходной схемы.

Для доказательства теоремы 2.1 вводится понятие высоты вершины, определяемое индуктивно: листья имеют высоту 1, вершины умножения имеют высоту, равную сумме высот предшественников, высота вершины сложения равна max(a + 1/2, m), где a равно максимуму высот предшественников, являющихся сложениями, а m – максимуму высот предшественников, являющихся либо листьями либо умножениями. Высота схемы равна максимуму высот вершин.

Справедлива следующая оценка высоты схемы U степени d с e дугами, соединяющими вершины сложения:

$$\text{высота } U \leq 1/2\ e \cdot d + d \qquad (2\ 2)$$

Высота схемы является ключевым параметром, позволяющим оценить число применений процедуры ФАЗА для вычисления значения схемы высоты h. Если U – схема высотой h, то вычисление значений во всех вершинах требует не более, чем log h + 1 применений процедуры ФАЗА.

Для доказательства теоремы 2.1 достаточно заметить, что согласно (2.2) $h = O(ed)$ и, очевидно, $e = O(N^2)$.

Сходным методом распараллеливания является метод, предложенный при параллельном решении *задачи динамического вычисления дерева* [228]. Для того, чтобы сформулировать эту задачу рассмотрим систему (P, I, Z), состоящую из:

1. множества P булевых переменных $p_1, \ldots, p_N$;
2. множества I правил вывода вида $p_i p_k \longrightarrow p_j$ или $p_k \longrightarrow p_j$;
3. подмножества переменных $Z \subseteq P$, называемых аксиомами.

Минимальной моделью для системы (P, I, Z) называется минимальное подмножество $M \subseteq P$, удовлетворяющее следующим свойствам:

1. $Z \subseteq M$,
2. если в левой части правил вывода стоят переменные из M, то переменная в правой части также принадлежит M.

Выводу в описанной системе естественным образом

соответствует корневое дерево. Задача динамического вычисления дерева заключается в нахождении минимальной модели для данной системы (P, I, Z)

Опишем параллельный алгоритм решения этой задачи на PRAM, называемый в дальнейшем процедурой DTEP.

BEGIN *массив* DI[1..N,1..N]; *обнуление массивов* DI *и* P.
P[1]:=1 *для всех* $p_i \in Z$
DI[j,i]:=1 *для* i = j *и всех* $p_j \to p_i \in I$.
выполнить t *раз*
begin
для i *таких, что* P[i] = 1 *параллельно выполнить*
если ((P[j]=P[k]=1) *и* ($p_j p_k \to p_i \in I$)) *или* ((P[j]=1) *и* (DI[j,i]=1)) *то* P[i]:= 1;
если (P[k]=1) *и* ($p_j p_k \to p_i \in I$) *то* DI[j,i]:=1;
DI:=DI•DI;
end
END

Отметим, что содержимое (i,j) позиции массива DI равно 1, если $p_j \to p_i$ или если $p_j p_k \to p_i$ и p_k = 1. Заметим также, что при возведении в квадрат матрицы DI (i,j)-й элемент становится равным 1, если есть цепочка выводов из p_i в p_j.

Лемма. *Если переменная* p_i *принадлежит минимальной модели* M *и для нее существует дерево вывода размером* m, *то* P[i] = 1 *после не более чем* t = 2.41 log m *итераций внутреннего цикла процедуры* DTEP. *Для* $p_i \in M$ P[i] *всегда равно 0.*

Доказательство основано на справедливости следующего факта: после каждой итерации внутреннего цикла существует дерево вывода для p_i размером не более 3/4 от размера дерева, полученного на предыдущей итерации, при использовании правил вывода из I и дополнительных правил, соответствующих единичным элементам DI, и аксиом p_j, соответствующих единичным значениям P[j].

Начальный шаг индукции очевиден. Пусть после t-й итерации получено дерево вывода T, и |T| = m. Рассмотрим два случая:

1. T имеет не менее m/4 листьев. В этом случае после выполнения цикла все листья добавляются к аксиомам и тривиально размер дерева сокращается не менее, чем на m/4.

2. Пусть r_k – число максимальных цепей с k внутренними вершинами и b – число листьев в дереве вывода T. Имеем:

$$m = 2b - 1 + \sum_{k \geq 1} k r_k,$$

Тогда для числа n вершин нового дерева, полученного после очередной (t + 1)-й итерации, справедливо неравенство.

$n \le b - 1 + \sum_{k \ge 1} \lceil k/2 \rceil r_k \le b - 1 + 1/2(\sum_{k \ge 1} k r_k + \sum_{k \ge 1} r_k) \le b - 1 + 1/2(m + 1 - 2b) + (1/2)2b < 3/4\ m.$

Первое неравенство следует из того, что при возведении в квадрат матрицы DI длины всех цепей уменьшаются вдвое. Третье неравенство вытекает из того факта, что в дереве с b листьями число максимальных цепей не превосходит 2b - 1.

Таким образом, после не более, чем $\log_{3/4} m \le 2.41 \log m$ итераций получим тривиальное дерево вывода. Отсюда следует, что справедлива следующая

Теорема 2.2. *Пусть* (P, I, Z) - *определенная выше система, для которой любой элемент* p *в минимальной модели* M *имеет дерево вывода размером не более* $2^{\log^c N}$. *Тогда существует* NC-*алгоритм вычисления* M.

Доказательство вытекает из леммы. Отметим, что время работы получаемого NC-алгоритма есть $O(\log^{c+1} N)$ на PRAM с N^3 процессорами. Это число процессоров обусловлено использованием в качестве основной операции алгоритма умножения матриц и может быть уменьшено при использовании быстрых алгоритмов умножения матриц.

Как показано в [228] рассмотренный метод применим для построения NC-алгоритмов решения следующих задач - транзитивного замыкания графов и орграфов, нахождения наибольшей общей подстроки для двух строк, вычисления значения монотонной планарной схемы, , вычисления значения алгебраических схем ограниченной степени, а также задач, распознаваемых альтернирующими машинами Тьюринга с логарифмической памятью за полилогарифмическое время

Рассмотрим сначала применение этого метода для получения упомянутого в разделе 1.2 результата о том, что всякая функция, вычислимая на альтернирующей машине Тьюринга (ATM) за полилогарифмическое время с памятью O(log n), принадлежит классу NC. Поскольку вычисления на ATM проводятся с ограничением на размер памяти в O(log n), число различных конфигураций ограничено некоторым полиномом от n. Пронумеруем все конфигурации и будем считать переменными модели $p_1, \ldots, p_m$, полагая что $p_i = 1$ тогда и только тогда, когда i-я конфигурация является принимающей. Аксиомами являются конфигурации, состояния которых являются принимаю-

шнмн н правнла вывода задаются так: 1) $p_j p_k \longrightarrow p_i$ тогда и только тогда, когда p_i содержит универсальное состояние и две конфигурации j-я и k-я являются непосредственными последователями i-й конфигурации; 2) $p_j \longrightarrow p_i$ во всех других случаях, когда j-я конфигурация является непосредственным последователем i-й конфигурации.

Можно проверить по индукции, что имеется точное соответствие между деревьями вывода для p_1, соответствующими начальной конфигурации АТМ и некоторыми поддеревьями дерева вычислений АТМ. Эти поддеревья получаются удалением для всех вершин, конфигурации которых содержат универсальное состояние, одного произвольного потомка этой вершины вместе с соответствующим ему поддеревом. Поскольку дерево вычислений АТМ - полиномиального размера, любое дерево вывода получается описанным выше построением. Применяя теперь к нему теорему 2 2, получим, что всякая функция, вычислимая на альтернирующей машине Тьюринга за полилогарифмическое время с памятью O(log n), принадлежит классу NC.

Обратное моделирование NC на АТМ осуществляется просто: по равномерной последовательности булевых схем полиномиального размера и полилогарифмической глубины строится программа АТМ, причем вычисление идет от выхода схемы ко входам. При этом в вершине-конъюнкции происходит переход в универсальное состояние (в вершине-дизъюнкции - в экзистенциальное), и выбирается вершина, непосредственно предшествующая данной в схеме до тех пор, пока не будут получены входные вершины. В них осуществляется переход в отвергающее состояние, если значение входа равно нулю, и в принимающее, если значение входа равно единице.

Рассмотрим кратко применение этого метода для получения изложенного выше результата о распараллеливании схемных вычислений ограниченной степени. Сначала схема (неветвящаяся программа) приводится к последовательности вычисления билинейных форм. Коэффициенты билинейных форм можно вычислить путем вычисления степеней матрицы A, построенной по исходной программе следующим образом. Каждая вершина, соответствующая операции умножения в исходном орграфе, заменяется на три вершины: первая сопоставляется левому операнду, вторая - правому, а третья - результату. Матрица A является матрицей инцидентности этого нового орграфа (с петлей в каждой вершине). Коэффициенты билинейных форм можно вычислить за время $O(\log^2 n)$ на PRAM с $M(n)$ процессорами, где $M(n)$ - число процессоров, необходимое для умножения двух $n \times n$ матриц за время

O(log n)

Определим для орграфа B, соответствующего последовательности вычислений линнейных программ, операции, аналогичные операциям DTEP. Для i-й вершины переменные заменяются на пару (c_i, v_i), где c_i – булева переменная, показывающая, выполнена ли операция в i-й вершине, и v_i – равно значению i-й вершины, если $c_i = 1$. Вершина называется 1-вершиной, если ровно один из ее двух операндов уже вычислен. Остальные невычисленные вершины называются 2-вершинами.

Определим некоторые матрицы, необходимые для описания параллельного алгоритма.

$C^{(l)} = (c_{ij})$, где c_{ij} равен коэффициенту на левом входном ребре из i-й вершины в j-ю, причем j-я вершина является 2-вершиной в противном случае $c_{ij} = 0$.

$C^{(r)}$ – аналогичная матрица, но для правых операндов.

$D = (d_{ij})$ – для j-й 1-вершины d_{ij} равен коэффициенту на ребре, ведущему из i-й вершины в j-ю.

C – ненулевыми являются только диагональные элементы, соответствующие 2-вершинам.

Одна итерация модифицированного DTEP алгоритма состоит из следующих шагов.

Для всех i с $c_i = 0$ и двумя вычисленными операндами параллельно вычисляются значения v_i и $c_i = 1$.

Для всех i, ставших 1-вершинами, параллельно умножить коэффициенты всех входных невычисленных ребер на соответствующие вычисленные множители, обновить все матрицы в соответствии с изменениями множеств 1-вершин и 2-вершин:

$C^{(l)} := (C + D)\,C^{(l)}$, $C^{(r)} := (C + D)\,C^{(r)}$, $D := (C + D)\,D$,

обновить орграф B.

Можно проверить, что линнейная программа, вычисляемая модифицированным после одной итерации DTEP орграфом, вычисляет те же значения, что и исходная линнейная программа. Одна итерация DTEP может быть вычислена на PRAM с M(n) процессорами за время O(logn). Для оценки числа необходимых итераций до полного распараллеливания в [228] в отличие от [233] не вводятся понятие высоты орграфа, а прямо оценивается число итераций через формальную степень исходного орграфа вычислений. В результате оказывается, что O(log d) итераций процедуры DTEP достаточно для вычисления исходного орграфа. Тем самым, несколько другим способом устанавливается справедливость теоремы 2.1, сформулированной в нача-

ле данного раздела.

Рассмотрим теперь применение этого метода для доказательства того, что значение монотонной планарной схемы можно вычислить в NC. При этом будем предполагать, что схема разбита на ярусы, ярусы состоят из элементов одного типа (конъюнкций или дизъюнкций), ярусы разных типов чередуются, схема является синхронной, то есть все пути от входов к некоторому элементу имеют одинаковую длину.

Эти ограничения не являются существенными, поскольку произвольная монотонная планарная схема легко может быть преобразована в NC к такому виду. Интервалом (l,i,j) называется множество вершин в l-м ярусе, которые зависят от i-й вершины и от которых в свою очередь зависит вершина j (вершина i зависит от k, если имеется ориентированный путь из k в i). Рассмотрим булевы переменные $p_{l,i,j}$, равные единице, если значение каждого элемента из интервала (l,i,j) равно единице. Аксиомы задаются всеми единичными входными интервалами. Правила вывода определяются следующим образом:

1. $p_{l-1,i,r} \longrightarrow p_{l,i,j}$ для всякого непустого интервала элементов-дизъюнкций, i обозначает первый вход первого элемента, r – последний вход последнего элемента интервала,

2. $p_{l,i,k}\, p_{l,k+1,j} \longrightarrow p_{l,i,j}$ для всех $i \le k < j$,

3. $p_{l,i,r} \longrightarrow p_{l,i,j}$ для всех ярусов l из элементов-дизъюнкций и всех пар (i,r), где $i \le r$ и i-й и r-й элементы является входами l-го элемента.

Нетрудно проверить, что число переменных и число аксиом полиномиально по n. По индукции доказывается, что значение i-го элемента в l-м ярусе равно единице тогда и только тогда, когда $p_{l,i,i}$ выполнено в минимальной модели. Пусть $p_{l,i,j}$ принадлежит минимальной модели. Убедимся, что для него имеется дерево вывода линейного размера. Рассмотрим подсхему, состоящую из всех элементов, от которых зависят элементы $p_{l,i,j}$. При использовании правил вывода 1 и 3 вершина в дереве вывода, соответствующая левой части правила вывода, сопоставляется интервалу (l,i,j). В случае, когда интервал в l-м ярусе состоит из элементов-дизъюнкций и в $(l-1)$-м ярусе имеется более одного максимального по включению интервала, предшествующего данному, необходимо использовать правило вывода 2. Интервал (l,i,j) расщепляется на подинтервалы (l,i,k) и $(l,k+1,j)$ так, что один из входов элемента k является нулевым. Поставим в соответствие левой части правила вывода 2 интервал

элементов (l - 1)-го яруса с нулевыми значениями, содержащими этот вход. В силу планарности схемы, правила, использующие для вывода $p_{l,l,k}$ и $p_{l,k+1,j}$, соответствуют различным множествам интервалов. Следовательно, число максимальных интервалов с одинаковым выходным значением является верхней оценкой размера минимального дерева вывода для любого элемента минимальной модели. Поэтому из теоремы 2.2 вытекает существование NC-алгоритма вычисления значения монотонной планарной схемы.

2.1.5. Распараллеливание с использованием сепараторов

Одним из общих приемов получения параллельных алгоритмов из последовательных алгоритмов является использование сепараторов для разбиения схемы (графа алгоритма) на две части и применения методики "разделяй и властвуй". Напомним, что вершинным c-сепаратором в графе называется такое множество вершин графа, при удалении которого граф распадается на компоненты связности с числом вершин в каждой не более cn, где n - число вершин графа, c < 1. Обычно 2/3-сепаратор называется просто сепаратором. Рассматриваются также и реберные сепараторы.

Определение. *Ориентированным реберным сепаратором ациклического орграфа называется множество ребер* R, *при удалении которых орграф распадается на два изолированных подграфа с числом вершин не более* cn, *где* c < 1 *-константа, причем все ребра в* R *ориентированы из одного подграфа в другой.*

Имеется много путей использования сепараторов для построения параллельных алгоритмов [216]. В предыдущем разделе был приведен пример их применения для построения параллельных алгоритмов на планарных графах, существенно использующий теорему Липтона-Тарьяна о существовании сепараторов размера $O(\sqrt{n})$ в произвольном планарном графе. Ниже рассматривается использование сепараторов при распараллеливании схемных вычислений.

Пусть задана булева схема в полном базисе.

Утверждение. *Если любая схема сложности* L *из данного класса имеет сепаратор размера не более* S(L), *то найдется эквивалентная схема глубины, не превосходящей* O(S(L) log L). *При этом под сепаратором схемы понимается ориентированный реберный сепаратор*

соответствующего ей ациклического орграфа.

Доказательство основано на выделении сепаратора и рассмотрении двух образующихся частей схемы G_1 и G_2. При этом при заданном входе параллельно и независимо вычисляются значения схемы G_2 на всех возможных 2^S промежуточных входах, образующихся за счет S ребер из G_1 в G_2. Получаем булеву функцию $f(y_1,\ldots,y_S)$, зависящую не более, чем от S переменных. Путем интерполяции можем восстановить эту булеву функцию по ее значениям в 2^S точках. Это приводит к рекуррентному соотношению для глубины d(G) схемы G:

$$d(G) \le \max\ (\ d(G_1),\ d(G_2))\ +\ O(S),$$

Учитывая, что S это c–сепаратор, получаем, что

$d(\ L\ ) \le d(\ cL\ ) + O(\ S\ )$, где L – размер схемы. Отсюда

нетрудно получить,

что $d(\ L\ ) = O(\ S \log L\ )$.

Возможность построения интерполирующей схемы глубины O(S) вытекает из того, что базис является полным и в нем в виде суперпозиций с отождествлениями переменных выразима любая булева функция, в частности, конъюнкция, дизъюнкция и отрицание, и по 2^S значениям функции $f(y_1,\ldots,y_S)$ можно построить вычисляющую ее схему глубины O(S) в соответствии с дизъюнктивной нормальной формой

для $f(y_1,\ldots,y_S) = \bigvee_{(a_1,\ldots,a_S)} y_1^{a_1}\ldots y_S^{a_S}$, где $y^a = \neg y$ при $a = 0$ и

$y^a = y$ при $a = 1$.

Подобные методы распараллеливания с тонкими оценками констант для класса формул изучались в работах [52], [28], [265]. Отметим, что для класса формул соответствующая схема является деревом и для него существует c–сепаратор размера 1. В этом нетрудно убедиться, если двигаться из корня дерева по ветви, содержащей большее число вершин до момента, пока это число не станет меньше n/2, где n – число вершин дерева. Эта вершина является, как нетрудно проверить, 3/4 сепаратором. Отсюда сразу следует соотношение для глубины вида: $d(\ L\ ) \le c \log L + c_1$. Оценка наименьшей константы c для класса формул является очень трудной задачей. В [52] получена оценка $c \le 2$, в [265] – $c \le 1.82$, в

[28] - $c \leq 1.73$.

Другим примером применения сепараторов для распараллеливания схемных вычислений являются так называемые синхронные схемы. Напомним, что синхронной схемой называется схема, в которой длины всех ориентированных путей одинаковы. В терминах частично упорядоченных множеств это означает, что частично упорядоченное множество, соответствующее схеме, является ранжируемым. Покажем, что для всякой синхронной схемы размера L существует эквивалентная ей схема глубиной $O(L^{1/2}\log L)$.

Убедимся сначала, что синхронная схема с максимальным размером яруса k (ширины k) может быть преобразована в эквивалентную схему глубины $O(k \log L)$. Это является прямым следствием доказанного выше утверждения, поскольку в качестве сепаратора всегда можно выбрать средний ярус.

Выделим теперь в произвольной синхронной схеме размера L средний ярус и рассмотрим все ярусы на расстоянии не более $\sqrt{L}$ от него. Либо среди них найдется ярус размера $O(\sqrt{L})$, являющийся c-сепаратором при некотором $c > 0$, и тогда все доказано, либо глубина схемы есть $O(\sqrt{L})$ и также все доказано, либо на расстоянии не более $\sqrt{L}$ от среднего яруса найдутся два яруса размера $O(\sqrt{L})$ - выше и ниже среднего яруса, являющиеся такими разрезами, что в двух частях схемы, кроме средней части, содержится $\alpha(L)$ элементов схемы. Тогда, индуктивно применяя доказанное выше утверждение для трех частей схемы, получим эквивалентную схему глубины $O(L^{1/2}\log L)$.

Из описанного метода распараллеливания синхронных схем заданной ширины нетрудно получить приведенную в разделе 1.1 теорему о связи времени T и памяти S машины Тьюринга с глубиной моделирующей ее на входах заданной длины схемы. Действительно, рассматривая протокол вычисления машины Тьюринга, отметим, что все конфигурации имеют длину O(S) и их число в протоколе не превосходит T. Аналогично доказательству P-полноты задачи о значении булевой схемы строится синхронная булева схема, каждый ярус которой моделирует один шаг машины Тьюринга. Ширина этой схемы не превосходит S, а глубина - T. Используя приведенное выше утверждение о распараллеливании с помощью сепараторов, можно построить эквивалентную

схему глубины S log T.

Аналогичный метод распараллеливания с использованием сепараторов можно использовать и в алгебраических схемах, соответствующих, например, алгоритмам линейной алгебры. Рассмотрим его на примере распараллеливания вычисления многочленов по схеме Горнера. Нетрудно видеть, что алгебраическая схема вычисления по схеме Горнера имеет 1/2-сепаратор размера 1. Поэтому вторую часть схемы можно рассматривать как многочлен от одной переменной ax + b. Вычислив первую часть схемы на входном значении и параллельно вторую часть схемы в некоторых двух точках, можно восстановить многочлен ax + b по его значениям в двух точках путем интерполяции за константное время. Это приводит к реккурентному соотношению для времени параллельного вычисления схемы Горнера t(n):

$$t(n) \le t(n/2) + O(1)$$

и, следовательно, $t(n) = O(\log n)$.

Другой способ распараллеливания схемы Горнера, основанный на представлении ее в виде линейной реккурентной последовательности, описан в разделе 2.2.1.

Таким же способом можно распараллелить алгоритм прогонки для решения систем линейных уравнений с трехдиагональной матрицей. Имеется также большое число работ, в которых распараллеливаются алгебраические формульные вычисления [73],[75],[264],[15]. Соответствующая схема вычислений для них является деревом, поэтому для них также имеется c-сепаратор размера 1 и наибольшая трудность заключается в оценке константы c. В [73] доказано, что $c \le 4$, в [15] - $c \le 2.88$. Для арифметических выражений без операции деления имеются следующие оценки константы c : [75] доказано, что $c \le 2.465$, в [264] - $c \le 2.1507$, в [15] - $c \le 2.08$.

Этот метод дает хорошие результаты при наличии сепаратора малого размера, что дает возможность одновременно получать хорошие верхние оценки и для числа необходимых процессоров. В [202] указан способ разбиения деревьев, который позволяет так перестроить выражение, что его глубина будет O(log n) и размер $O(n \log^c n)$

Сходные соображения , связанные с использованием реберных сепараторов, использовались также в [257] при доказательстве известной теоремы о существовании для любой булевой схемы сложности L эквивалентной схемы глубины O(L/logL) (см. также [332]).

2.1.6. Вероятностные параллельные алгоритмы и дерандомизация

Имеется ряд задач, допускающих хорошее распараллеливание с использованием датчиков случайных чисел. К ним относятся нахождение максимального паросочетания в графе, построение дерева поиска в глубину для графа, построение диаграммы Вороного на плоскости и ряд других задач. Класс задач, для решения которых существуют эффективные параллельные вероятностные алгоритмы, получил название **RNC** (см. раздел 1.4). Опишем алгоритмы из этого класса для некоторых известных задач.

Одной из первых задач, для которых были предложены эффективные вероятностные алгоритмы решения, была *проверка тождественного равенства нулю многочлена.*

В этой задаче требуется проверить, равен ли тождественно нулю данный многочлен. Прямой подход к ее решению основан на представлении многочлена в виде суммы элементарных произведений (мономов). Однако, этот алгоритм далеко не всегда завершает работу за полиномиальное время. Вероятностный алгоритм решения задачи, предложенный в [288], основан на следующем факте: *если многочлен* $F(x_1,\ldots,x_m)$ *не равен тождественно нулю и* $N \geq c \deg F$, *где* deg F *степень многочлена* F, *то* F *имеет не более* $c^{-1}N^m$ *целых корней на множестве* $[1,N]^m$.

Алгоритм заключается в независимом случайном выборе m целых чисел из интервала [1,N], где N = 2 deg F, и вычислении для этого набора значений переменных значения многочлена F. Если выдается значение, не равное нулю, то многочлен не равен нулю. Если выдается нуль, то вероятность того, что многочлен не равен тождественно нулю, в силу приведенного выше факта не превышает 1/2. Таким образом, мы получаем вероятностный алгоритм с односторонними ошибками для решения задачи. Его сложностные характеристики зависят от способа задания многочлена и сложности его вычисления. Если многочлен можно вычислить в NC, то имеем NC-алгоритм проверки равенства многочленов (тождественной равности многочлена нулю).

Задача о совершенном паросочетании в n*-вершинном графе.*

Требуется проверить, существует ли совершенное паросочетание (т.е. множество из $n/2$ попарно несмежных ребер) в заданном графе. Известно, что число совершенных паросочетаний в двудольном графе равно перманенту его матрицы смежности. Напомним, что пер-

манент отличается от определителя лишь тем, что все члены суммируются со знаком плюс. Однако, в отличие от определителя, который легко вычислить NC-алгоритмом, задача вычисления перманента является #P-полной [313] (напомним, что #P обозначает вычислительный аналог класса NP).

Первый RNC-алгоритм для задачи о совершенном паросочетании вдвудольном графе предложил Л. Ловас [218]. Он основан на вычислении вместо перманента определителя матрицы смежности и использует быстрый рандомизированный алгоритм проверки равенства многочленов [288].

В работе [242] прямо строится максимальное паросочетание в RNC^2 с использованием $O(n^{3.5}m)$ процессоров. Алгоритм существенно использует следующую общую лемму.

Лемма. *Дано множество из* n *элементов и система его подмножеств. Каждому элементу множества случайно и независимо с одинаковой вероятностью* 1/2n *приписывается вес - целое число из отрезка* [1, 2n]. *Тогда вероятность того, что в системе имеется единственное подмножество минимального веса будет не меньше* 1/2.

Доказательство основано на использовании понятия существенного элемента: элемент называется существенным, если существует подмножество минимального веса из системы, не содержащее этого элемента, и существует подмножество минимального веса из системы, содержащее этот элемент. Для произвольного i зафиксируем веса всех элементов кроме i-го. Тогда имеется критический вес α_i такой, что если вес i-го элемента не превосходит α_i, то i-й элемент содержится в некотором подмножестве минимального веса, а если вес i-го элемента больше α_i, то i-й элемент не содержится ни в одном подмножестве минимального веса. Ключевое рассуждение основано на том, что элемент может являться существенным, если только его вес равен α_i.

Отсюда, в силу независимости весов, вытекает, что вероятность для данного элемента оказаться существенным не превосходит 1/2n. Суммируя n раз эти вероятности, получим, что вероятность существования хотя бы одного существенного элемента в системе не превосходит 1/2, то есть с вероятностью не меньшей 1/2 в системе нет ни одного существенного элемента: каждый элемент входит либо во все подмножества минимального веса, либо ни в одно из них. А это и означает, что подмножество минимального веса единственно.

В качестве множества рассматриваются далее ребра графа, а в

качестве системы подмножеств – его максимальные паросочетания. Тогда совершенное паросочетание минимального веса в графе будет единственным с вероятностью не меньше 1/2. Рассмотрим элементы матрицы смежности графа $d_{ij} = d_{ji} = 1$ и заменим их символами переменных x_{ij} и $-x_{ij}$ соответственно. Для полученной таким образом матрицы A справедлив критерий Татта: определитель матрицы A тождественно не равен нулю тогда и только тогда, когда в графе есть совершенное паросочетание.

Заменим теперь x_{ij} на $2^{w_{ij}}$, полученную матрицу обозначим через B. Тогда если в графе есть единственное совершенное паросочетание минимального веса w, то определитель B отличен от нуля и 2^{2w} является его делителем.

Лемма. *Пусть* M – *единственное совершенное паросочетание минимального веса* w. *Тогда* $(v_i, v_j) \in M$ *тогда и только тогда, когда* $|B_{ij}|\ 2^{w_{ij}} / 2^{2w}$ – *нечетно, где* B_{ij} – *ij-й минор* B.

Таким образом, суммируя все сказанное, получаем следующую простую рандомизированную процедуру нахождения совершенного паросочетания.

1. Приписать всем ребрам случайные целые веса из отрезка [1,2m].
2. Вычислить |B| и найти w.
3. Для всех пар i, j вычислить миноры B_{ij}.
4. Для всех ребер параллельно вычислить величину $|B_{ij}|\ 2^{w_{ij}}/2^{2w}$ и, если она нечетна, включить ребро (v_i, v_j) в паросочетание.

Приведем пример еще одной задачи, связанной с паросочетаниями в графе, для которой не известна принадлежность ее классу P, но имеется RNC^2-алгоритм.

Точное паросочетание.

Вход: Граф G = (V,E), подмножество $F \subseteq E$ красных ребер, натуральное число k.

Вопрос: Существует ли в графе G совершенное паросочетание, включающее в себя ровно k красных ребер.

Снова используем приведенную выше лемму о вероятности существования единственного подмножества минимального веса. В качестве основного множества рассматриваются ребра графа, а в качестве

системы подмножеств – все совершенные паросочетания графа, включающие в себя ровно k красных ребер. Если веса элементов полиномиально ограничены и имеется ровно одно паросочетание минимального веса, то следующая процедура, предложенная Л. Ловасом находит вес этого паросочетания. В матрице Татта для G заменяем x_e на 2^{w_e}, если $e \in E \setminus F$, и на $2^{w_e}y$, если $e \in F$. Пусть B – полученная матрица. Тогда $|B| = (pf(B))^2$, где pf(B) – пфаффиан B. Вычисляем pf(B), возводя в квадрат определитель, который, в свою очередь, вычисляется параллельно с использованием интерполяции [68]. Степень двойки в коэффициенте при y^k равна в точности минимальному весу совершенного паросочетания, включающего в себя ровно k красных ребер. Поскольку единственность совершенного паросочетания минимального веса следует из леммы (точнее, что соответствующая вероятность не меньше 1/2), получаем RNC^2-алгоритм решения задачи о точном паросочетании.

Еще одной важной задачей, для которой не известна принадлежность классу NC, но существуют RNC-алгоритмы является задача о построении дерева поиска в глубину для данного графа.

Дерево поиска в глубину в неориентированном графе G = (V,E). Напомним, что корневое остовное дерево называется деревом поиска в глубину, если нет ребер графа, соединяющих две различные ветви дерева. Известно, что задача определения, содержится ли данное ребро графа в лексикографически первом дереве поиска в глубину является P-полной. RNC-алгоритм построения такого дерева был предложен в [33]. Он основан на на стратегии "разделяй и властвуй" и решении задачи о максимальном паросочетании в специальном графе.

Сначала строится начальный сегмент дерева T_1, обладающий свойством: каждая компонента в $V-T_1$ содержит не более n/2 вершин. Начальный сегмент можно достроить до дерева поиска в глубину следующим образом. Пусть C – компонента связности в $V - T_1$. Имеется единственная вершина $x \in T_1$, наиболее удаленная от корня дерева T_1 и смежная некоторой вершине $y \in C$. Построим дерево поиска в глубину для C с корнем в y и присоединим его к дереву T_1 с помощью ребра (x,y). Это построение может быть выполнено независимо для каждой компоненты $V - T_1$.

Поскольку после такого этапа размер задачи уменьшается вдвое, глубина рекурсии не превосходит log n. Основную сложность на каждом этапе представляет построение начального сегмента дере-

ва с описаннымн выше свойствамн. Алгорнтм построенння начального сегмента состонт нз двух частей. Множество Q вершнннo различных путей называется сепаратором, еслн нанбольшая компонента связиостн V – Q нмеет размер не более n/2. В первой части алгорнтма стронтся сепаратор Q, состояший нз фнксированного чнсла путей. Во второй частн нз сепаратора Q стронтся начальный сегмент дерева пнпска в глубнну. Нанболее трудным является построенне сепаратора. Это построение осуществляется обращеннем в цнкле к процедуре Reduce(Q), которая нз множества путей Q, являюшнхся сепаратором, стронт такое же множество, но с чнслом путей в 11/12 раз меньше, чем в нсходном множестве. В начале работы Q состонт нз n вершнн, каждая нз которых является путем длнны 0. Поскольку после каждого прнменення процедуры Reduce(Q) чнсло путей уменьшается в константу раз, то после O(log n) прнменений этой процедуры получнм требуемое множество, состоящее нз фнксированного чнсла путей.

В процедуре Reduce(Q) множество вершнно непересекающнхся путей Q разбнвается на две частн L н S н нщется множество вершнно непересекающнхся путей, каждый нз которых нмеет одну концевую вершнну на путях нз L и другую на путях нз S, а внутренние вершны – нз V – Q. Задача построенння максимального множества таких путей сводntся к построению совершенного паросочетания максимального веса в некотором двудольном взвешенном графе. Прн этом чнсло вершнн в этом графе не превышает 2n, а вес каждого ребра не превосходнт n. Для решения этой задачи нзвестен RNC-алгорнтм [242].

Общее время работы этого RNC-алгорнтма есть величина $O(\log^5 n)$ на PRAM с n $M(n)$ процессорами.

Другнм нзвестным вероятностным алгорнтмом является алгорнтм Рейфа-Сена построенння днаграммы Вороного на плоскостн [278]. Время его работы O(log n) на CREW PRAM с O(n) процессорами, где n – чнсло точек. Алгорнтм нспользует $O(\log^2 n)$ случайных бнтов н завершает работу за указанное время с вероятностью $1 - n^{-\alpha}$ для некоторого $\alpha > 0$. Отметнм, что наилучшнй нзвестный детерминнрованный алгорнтм для этой задачн нмеет время работы $O(\log^2 n)$ также на n процессорах.

Поскольку нспользованне вероятностных методов часто прнводнт к эффектнвным параллельным вероятностным алгорнтмам, возннкает вопрос, нельзя лн нспользовать вероятностные соображения также н

для построения детерминированных параллельных алгоритмов на основе вероятностных алгоритмов. Изложим сначала общую идею дерандомизации в последовательном варианте.

Предположим, что мы имеем рандомизированный алгоритм, который в вероятностном пространстве Ω осуществляет поиск "хороших" точек (удовлетворяющих нужным условиям), причем существование таких точек гарантируется из вероятностных соображений. Для нахождения хорошей точки достаточно осуществить бисекцию вероятностного пространства и оценить число точек в каждой части пространства, а затем выбрать ту половину пространства, в которой содержится больше хороших точек. Эта процедура повторяется до тех пор, пока не останется только одна хорошая точка. Нетрудно видеть, что эта процедура требует $\log|\Omega|$ итераций. Если $|\Omega|$ экспоненциально велико, то прямое применение метода не дает хорошего параллельного алгоритма. К тому же основная трудность заключается в подсчете или оценке числа хороших точек на каждой итерации.

В самой общей форме идея последовательной дерандомизации изложена в [10],[270],[299] и показана ее эффективность в задаче о покрытии и α-глубине (0,1)-матриц и в задаче о балансировке множеств. Для функции f, отображающей пространство K в R, и функционала F рассматривалась задача нахождения точки $x \in K$: $f(x) \le F(f,K)$. Выделено свойство, позволяющее последовательно путем отсечений находить нужную точку:

$$F(f, K) \ge \min (F(f, K_1), F(f, K_2)), \qquad (2.3)$$

где $K = K_1 \cup K_2$ – произвольное разбиение пространства K. Было отмечено, что свойством (2.3) обладают такие функционалы, как среднее значение функции на множестве, математическое ожидание, вероятность и ряд других. В случае, когда множество K является декартовым произведением n конечных множеств, искомую точку x можно строить покомпонентно. Для этого достаточно на i-м шаге, при условии, что уже найдены значения первых i-1 координат $x_1 = a_1, \ldots, x_{i-1} = a_{i-1}$, вычислить значения функционала $F(a_1, \ldots, a_{i-1}, x_i)$ и ри всех возможных x_i и выбрать значение x_i, на котором достигается минимум из этих значений. При этом свойство (2.3) гарантирует, что для так построенной точки x выполнено: $f(x) \le F(f,K)$.

Основная трудность метода, частным случаем которого являются процедуры покомпонентной дерандомизации Спенсера-Рагхавана [299], [270], заключается в вычислении значений функционала $F(a_1, \ldots, a_i)$ при заданных значениях $x_1 = a_1, \ldots, x_i = a_i$. Для ряда

задач эти трудности удается преодолеть, что приводит к полиномиальным асимптотически точным алгоритмам [10], [270], [299], [11]. Однако, даже в лучшем случае число итераций равно n и алгоритмы, полученные таким способом, являются существенно последовательными.

Для построения параллельных дерандомизированных NC-алгоритмов можно применять идею свертки (сжатия) вероятностного пространства. Она заключается в следующем. Предположим, что можно ввести небольшое по объему (полилогарифмическое) вероятностное пространство, в котором на основе вероятностных соображений удается доказать существование хороших (обладающих искомыми свойствами) точек. Тогда, если число точек в пространстве не превосходит некоторого полинома от n, можно найти решение просто полным перебором. Если же пространство имеет больший объем (например, n в степени полилогарифм от n) можно применить изложенную выше идею покомпонентной дерандомизации с помощью вычисления условных вероятностей (математических ожиданий).

В ряде работ идея свертки большего вероятностного пространства в меньшее с сохранением нужных теорем существования разработана с использованием k-*независимых случайных величин*, т.е. величин, любые k из которых независимы. Пусть требуется построить n случайных булевых векторов, любые k из которых линейно независимы. Для k=2 в [221] рассмотрена следующая конструкция вероятностного пространства. Пусть $l=\lceil \log n \rceil + 1$ и $\omega = (\omega_1,\dots,\omega_l)$ – равномерно распределенный булев вектор на Z_2^l. Для каждого i пусть $(i_1,\dots,i_l)$ обозначает двоичную запись числа i. Новые случайные величины определяются следующим образом:

$$X_i = \Big(\sum_{j=1}^{l-1} i_j \omega_j + \omega_l\Big) \bmod 2.$$

Таким образом, из $\lceil \log n \rceil + 1$ независимых случайных величин строятся n попарно независимых случайных величин. Отметим, что полный перебор в таком вероятностном пространстве можно осуществить всего за O(n) шагов.

В более общем случае (при $k \geq 2$) эта конструкция обобщается следующим образом. Пусть l не превышает некоторого полинома от логарифма n и ω – снова равномерно распределенный булев вектор длины l. Рассмотрим булевы векторы $a_1,\dots,a_n$ такие, что $a_i \in Z_2^l$,

и положим $X_i = a_i\omega$ (скалярное произведение векторов). Справедливо следующее простое

Утверждение: *$X_{i_1},\dots,X_{i_k}$ независимы тогда и только тогда, когда $a_{i_1},\dots,a_{i_k}$ линейно независимы над Z_2.*

Таким образом, для построения k-независимых случайных величин достаточно построить систему булевых векторов, любые k из которых линейно независимы над Z_2. Такие системы хорошо известны в теории кодирования и используются для построения кодов, исправляющих заданное число ошибок. В частности, в качестве $a_1,\dots,a_n$ можно взять столбцы проверочной матрицы кода БЧХ длины $h=\lceil\log(n+1)\rceil$ k/2 [13].

Подробный анализ такого способа свертки вероятностных пространств и его использование для построения параллельных алгоритмов будет рассмотрен в следующих разделах.

Для построения эффективных параллельных алгоритмов желательно уметь строить вероятностные пространства как можно меньшего размера, но так, чтобы соответствующие случайные величины обладали достаточной независимостью (k-независимость соответствующих случайных величин). Выше приведен пример построения такого вероятностного пространства с помощью известных линейных кодов с заданным расстоянием. В общем виде задача построения вероятностного пространства минимального размера, порождающего k-независимые случайные величины, которые принимают значения 0 и 1 с вероятностью 1/2, формализуется в виде задачи о существовании ортогональных таблиц.

Определение. *Ортогональной таблицей с n ограничениями, двумя уровнями, силой k и индексом $M/2^k$ называется матрица размера M x n, любые k столбцов которой содержат каждый двоичный вектор длины k ровно $M/2^k$ раз и k является максимальным числом, удовлетворяющим этому свойству.*

Нетрудно убедиться, что если в качестве вероятностного пространства рассматривать n столбцов такой ортогональной таблицы и каждой строке сопоставить свою случайную величину, то эти случайные величины принимают значения 0 и 1 с вероятностью 1/2 и являются k-независимыми. Таким образом, задача минимизации размера вероятностного пространства, порождающего k-независимые случайные величины, эквивалентна задаче минимизации n (при заданных M и k), при которых существует ортогональная таблица с n ограничениями,

двумя уровнями, силой k и индексом $M/2^k$. Для построения таких таблиц также можно использовать хорошие коды, поскольку любой двоичный код, состоящий из M векторов длины n, является ортогональной таблицей с n ограничениями, двумя уровнями, силой D – 1 и индексом $M/2^{D-1}$, где D – дуальное расстояние кода [13].

Поскольку в практических приложениях можно ограничиваться не точной k-независимостью, а приближенной, рассматриваются различные обобщения понятия k-независимости случайных величин, в частности, понятие (ε,k)-*независимости случайных величин* [244], [40]. Для любой k-выборки таких случайных величин вероятность пересечения событий отличается от произведения вероятностей не более, чем на ε. Таким образом, (0,k)-независимые случайные величины являются k-независимыми.

Случайная величина X, принимающая значения 0 и 1, называется ε-*равномерной*, если $|P(X = 0) - P(X = 1)| \le \varepsilon$. Пусть $S_n \subseteq (0,1)^n$ и $X = x_1 \ldots x_n$ – равномерная выборка из S_n. Пространство S_n называется ε-*равномерным по отношению к линейным тестам*, если для всякого $a \in (0,1)^n \setminus (0)^n$ случайная величина (a,X) является ε-равномерной (через (a,X) обозначается скалярное произведение a и X в поле GF(2)).

Известно, что из ε-равномерности по отношению к линейным тестам вытекает (ε,k)-независимость для любого k [323]. Нетрудно проверить, что если рассмотреть пространство S_n в виде (0,1)-матрицы, строками которой являются векторы из S_n, то оно будет ε-равномерным по отношению к линейным тестам тогда и только тогда, когда столбцы матрицы являются базисом линейного кода с минимальным расстоянием $n/2(1 - \varepsilon)$ и максимальным расстоянием $n/2(1 + \varepsilon)$. Эти два утверждения подчеркивают интересную связь теории кодирования с параллельными вычислениями. В отличие от стандартной задачи теории кодирования здесь требуется, чтобы все попарные расстояния между кодовыми векторами не только были ограничены снизу некоторой величиной d_{min}, но и ограничены сверху некоторой величиной d_{max}. При

$$d_{min} = n/2(1 - \varepsilon), \quad d_{max} = n/2(1 + \varepsilon)$$

эта задача напоминает задачу построения эквидистантных кодов (при $\varepsilon = 0$).

Для построения таких ε-эквидистантных кодов можно использо-

вать коды, двойственные к линейным кодам с небольшим кодовым расстоянием (в частности, двойственные к БЧХ-кодам). Известно, что в кодах, двойственных к БЧХ-кодам с расстоянием $d + 1$, все веса кодовых векторов заключены в интервале от $n/2 - O(d\sqrt{n})$ до $n/2 + O(d\sqrt{n})$ ([13], разд.9.9, теорема 18). Отсюда легко получаются верхние оценки для минимального размера $L(n,\varepsilon) = |S_n|$ пространства S_n, являющегося *ε-равномерным по отношению к линейным тестам*, вида $L(n,\varepsilon) = O(n^2/\varepsilon^2\log^2(n/\varepsilon))$. Логарифм мощности такого пространства есть, очевидно, $O(\log n + \log(1/\varepsilon))$.

Для построения (ε,k)-независимых случайных величин в [244] предложен способ совмещения конструкции k-независимых случайных величин и конструкции пространства, являющегося ε-равномерным по отношению к линейным тестам. Его суть заключается в том, что в конструкции k-независимых случайных величин в виде $x_i = a_i\omega$, $i = 1,\ldots,n$, двоичные векторы ω длины $h = O(k \log n)$ выбираются из пространства, являющегося ε-равномерным по отношению к линейным тестам. Логарифм мощности такого пространства есть величина

$$O(\log k + \log\log n + \log(1/\varepsilon)).$$

Несколько конструкций для порождения ε-равномерного по отношению к линейным тестам вероятностного пространства со сходными оценками размера построены в [40], [244]. Для сравнения отметим, что из теоретико-кодовых соображений вытекает нижняя оценка вида $L(n,\varepsilon) = \Omega(n/(\varepsilon^2\log(1/\varepsilon)))$ (при условии, что эта величина не превосходит 2^n). Как видно, верхняя оценка примерно в n раз больше нижней. Простые вероятностные рассуждения позволяют показать существование пространств S_n с $L(n,\varepsilon) = O(n/\varepsilon^2)$.

С использованием рандомизации при построении параллельных алгоритмов тесно связана также задача построения *псевдослучайных датчиков* для различных сложностных классов, связанных с параллельными вычислениями. Эти псевдослучайные датчики для некоторого класса алгоритмов S, вычисляющих булевы функции, выглядят как случайные для всякого алгоритма из S.

Определение. *Генератор* $G: (0,1)^m \longrightarrow (0,1)^n$ *называется псевдослучайным для класса* S *с параметром* ε, *если для любого алгоритма* A *из* S *при достаточно больших* n *на входах длины* n *выполнено*:

$$| Pr_y(A(y) = 1) - Pr_x(A(G(x)) = 1) | \le \varepsilon$$

где y *выбирается случайно и равновероятно из* $(0,1)^n$, *а* x – *из* $(0,1)^m$.

В [36] построены псевдослучайные генераторы для класса схем полиномиального размера и константной глубины (с неограниченной степенью входа), то есть для класса AC^0. В [251] эта конструкция улучшена, обобщена и показано, как с помощью "нижних оценок" сложности класса S можно построить псевдослучайный генератор для S. В [250] построен псевдослучайный генератор для класса алгоритмов с заданной памятью S – SPACE(S), переводящий O(S log R) случайных битов в R битов. В частности, всякий рандомизированный алгоритм с полилогарифмическим временем работы на полиномиальном числе процессоров (класс NC) можно перевести в NC-алгоритм лишь с полилогарифмическим числом случайных битов. Отсюда следует, в частности, что всякий алгоритм из RNC можно промоделировать детерминированным алгоритмом с временем работы $2^{\log^c n}$, где c – некоторая константа. Вопрос о том, возможно ли полиномиальное детерминированное моделирование алгоритмов из RNC остается открытым. Примеры различных задач из RNC, для которых возможна дерандомизация в P и даже в NC будут рассмотрены в следующих разделах.

2.2. Параллельные алгоритмы в различных предметных областях

2.2.1. Алгебра

Вычисление линейной рекуррентной последовательности первого порядка $x_i = a_i x_{i-1} + b_i$, $i = 1,\dots,n$.

Запишем рекуррентное соотношение в матричной форме

$$\begin{vmatrix} x_i \\ 1 \end{vmatrix} = \begin{vmatrix} a_i & b_i \\ 0 & 1 \end{vmatrix} \begin{vmatrix} x_{i-1} \\ 1 \end{vmatrix},$$

$$\begin{vmatrix} x_n \\ 1 \end{vmatrix} = \prod_{i=n}^{1} \begin{vmatrix} a_i & b_i \\ 0 & 1 \end{vmatrix} \begin{vmatrix} x_0 \\ 1 \end{vmatrix}.$$

Параллельный алгоритм вычисления линейной рекуррентной последовательности первого порядка предложен Стоуном [22] и основан на

применении метода сдваивания к вычислению произведения матриц

$$\prod_{i=n}^{1} \begin{vmatrix} a_i & b_i \\ 0 & 1 \end{vmatrix}.$$

Время работы приведенного алгоритма вычисления линейной рекуррентной последовательности на O(n/log n) процессорах EREW PRAM равно O(log n). Этот метод можно использовать и для параллельного вычисления рекуррентных соотношений k-го порядка за время O(log k log n).

Вычисление линейной рекуррентной последовательности k-го порядка. Линейная рекуррентная последовательность k-го порядка задается уравнениями: $x_i = a_{i,1}x_{i-1} + a_{i,2}x_{i-2} + a_{i,3}x_{i-3} + \ldots + a_{i,k}x_{i-k} + a_{i,k+1}$, $i = 1,\ldots,n$. Задача вычисления первых n членов такой линейной рекуррентной последовательности может быть решена на $O(k^{\omega-1}/\log k\ n/\log n)$ процессорах EREW PRAM за время O(log k log n) [151], где ω – показатель сложности умножения матриц.

Это можно сделать следующим образом. Задача переписывается в матричной форме: $u_i = u_{i-1} A_i$, где

$$u_i = (x_i, x_{i-1}, \ldots, x_{i-k+1}, 1),$$
$$A_i = (a_i, e_1, \ldots, e_{k-1}, e_{k+1}),$$
$$a_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,k}, a_{i,k+1})^T,$$

e_i – единичный вектор столбец, все компоненты которогоравны нулю за исключением i-й. Положим

$$v_i = u_{ik}, \quad B_{i+1} = A_{ik+1} * A_{ik+2} * \ldots * A_{(i+1)k}.$$

Тогда при $i \geq 0$ выполнено: $v_i = v_{i-1}B_i$ и для решения исходной задачи необходимо вычислить $v_1, v_2, \ldots, v_{n/k}$.

Алгоритм состоит из трех этапов. Для всех i, $0 < i \leq n/k$ вычислить:

1. B_i.
2. $C_i = B_1 * B_2 * \ldots * B_i$.
3. $v_i = v_0 C_i$.

На первом этапе произведения матриц вычисляются по методу сдваивания, причем на (h + 1)-м шаге сдваивания вычисляются $k/2^{h+1}$ произведений матриц вида

$$\begin{vmatrix} Z & I & 0 & \cdot & 0 & 0 \\ Z & 0 & I & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & I & \cdot \\ Z & 0 & 0 & \cdot & 0 & 0 \\ y & 0 & 0 & \cdot & 0 & 1 \end{vmatrix},$$

где Z – матрица размером $2^h \times 2^h$, y – матрица размером 1×2^h, I– единичная матрица размера $2^h \times 2^h$.

На втором этапе вычисляются префиксы относительно операции произведения матриц (по методу разд. 2.1.1). На третьем этапе вычисляются n/k прямых матричных произведений.

Сложность алгебраической схемы для вычисления произведения двух матриц из первого этапа составляет $O((k/2^h)2^{h\omega})$, а глубина – $O(h)$. Сложность вычисления B_i оценивается сверху величиной

$$\sum_{h=0}^{\log k} (k/2^{h+1})(k/2^h)2^{h\omega}) = k^2 \sum_{h=0}^{\log k} 2^{h(\omega-2)-1} = O(k^\omega),$$

а глубина соответствующей схемы есть $O(\sum_{h=0}^{\log k} h) = O(\log^2 k)$. Наконец, сложность параллельного вычисления всех B_i есть величина $O(n\ k^{\omega-1})$.

Сложность вычисления второго этапа схемой [208] составляет $O(k^{\omega-1}\ n)$, а глубина этой схемы – $O(\log n \log k)$.

Сложность вычисления третьего этапа схемой составляет $O(k^{\omega-1}n)$, а глубина этой схемы – $O(\log k)$.

Применяя утверждение из раздела 1.2 о связи схемных вычислений и вычислений на PRAM, получим алгоритм решения задачи на EREW PRAM с $O((n/\log n)\ k^{\omega-1}/\log k)$ процессорами за время $O(\log n \log k)$.

Решение треугольной системы.

Алгоритм с временем работы $O(\log^2 n)$ на EREW PRAM с $O(n^\omega/\log n)$ процессорами для *решения треугольной системы линейных уравнений и обращения треугольной матрицы* может быть получен посредством рекурсивного применения тождества

$$\begin{vmatrix} A & C \\ 0 & B \end{vmatrix}^{-1} = \begin{vmatrix} A^{-1} & -A^{-1}CB^{-1} \\ 0 & B^{-1} \end{vmatrix},$$

и использования параллельного алгоритма умножения матриц. Алгоритм с временем работы $O(\log^2 n)$ на EREW PRAM с $O(n^{\omega}/\log^2 n)$ процессорами получается из описанного выше алгоритма вычисления линейных рекуррентных последовательностей, поскольку решение треугольной системы можно представить как решение линейных рекуррентных соотношений n-го порядка.

Для параллельного вычисления *коэффициентов характеристического многочлена матрицы* A

$$f(t) = \det(tI - A) = t^n + \sum_{i=1}^{n} a_i t^{n-i}$$

воспользуемся *формулами Ньютона*

$$\begin{vmatrix} 1 & & & \\ s_1 & 2 & & \\ \cdots & \cdots & \cdots & \\ s_{n-1} & \cdots & s_1 & n \end{vmatrix} \begin{vmatrix} a_1 \\ a_2 \\ \cdots \\ a_n \end{vmatrix} = - \begin{vmatrix} s_1 \\ s_2 \\ \cdots \\ s_n \end{vmatrix}$$

и параллельным алгоритмом решения треугольных систем. Здесь

$$s_k = t_1^k + \ldots + t_n^k = tr(A^k),$$

– суммы Ньютона, $t_1, \ldots, t_n$ – собственные значения матрицы A. Вычисление следов степеней матрицы есть частный случай задачи о префиксах. Время работы алгоритма равно $O(\log^2 n)$ на EREW PRAM с $O(n^4/\log^2 n)$ процессорами. Использование быстрых алгоритмов умножения матриц приводит к аналогичной оценке времени на $O(n^{\omega+1}/\log^2 n)$ процессорах.

Параллельный алгоритм решения невырожденных систем линейных уравнений и обращения матриц может быть построен с использованием метода Леверье [22]. По теореме Гамильтона–Кэли, $f(A) = 0$. Следовательно,

$$A^{-1} = -1/a_n(A^{n-1} + a_1 A^{n-2} + \ldots + a_{n-1} I).$$

Время работы алгоритма равно $O(\log^2 n)$ на EREW PRAM с $O(n^4/\log^2 n)$ процессорами. Использование быстрых алгоритмов умножения матриц приводит к аналогичной оценке времени на

$O(n^{\omega+1}/\log^2 n)$ процессорах. Метод исключения Гаусса также может быть распараллелен и приводит к алгоритму с временем работы $O(\log^2 n)$ на $O(n^{\omega+1})$ процессорах. Эти результаты справедливы для любого поля. Для поля характеристики 0 в [266] показано, что для построения параллельного алгоритма с таким же временем работы достаточно $O(n^{\omega+1/2})$ процессоров. Обращение матрицы Вандермонда размером $n \times n$ над любым полем F, для которого возможно N-точечное быстрое преобразование Фурье ($n \le N \le cn$ для некоторой константы c), можно выполнить за время $O(\log n)$ на $O(n^2)$ процес-сорах [263].

Для вычисления ранга матрицы размером $n \times n$ над произвольным полем F известен алгоритм с временем работы $O(\log^2 n)$ и полиномиальным числом процессоров [83], [241]. Пусть A – квадратная симметрическая матрица (если это не выполнено, то заменим A на квадратную матрицу вида $\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$, которая удовлетворяет этим свойствам, и ее ранг равен удвоенному рангу A). Воспользуемся тем, что ранг сохраняется при расширении поля и расширим поле F добавлением одного x: $G = F(x)$. Построим матрицу C над G следующим образом $C = XA$, где $X = \mathrm{diag}(x_{ii} = x^{i-1})$. Поскольку X невырождена $\mathrm{rk}(C) = \mathrm{rk}(A)$. Для матрицы C справедливо следующее утверждение: $\mathrm{rk}(C) = \mathrm{rk}(CC)$. С использованием этого утверждения несложно проверить, что для любого натурального $k \ge 1$ $\mathrm{Ker}(C^k) = \mathrm{Ker}(C)$ и $K(C) = \bigcup_{k \ge 1} \mathrm{Ker}(C^k) = \mathrm{Ker}(C)$. Поскольку $\mathrm{rk}(A) = n - \dim(K(A))$, $\mathrm{rk}(C) = \mathrm{rk}(A)$ и $\dim(K(C)) = \dim(\mathrm{Ker}(C))$, то для нахождения ранга достаточно найти значение $\dim(\mathrm{Ker}(C))$, равное максимальной степени m, для которой t^m делит характеристический многочлен для C.

Параллельный алгоритм вычисления ранга матрицы A заключается в вычислении характеристического многочлена $Q(t) = \det(tI - XA)$ и нахождении максимального значения m, для которого t^m делит $Q(t)$. Тогда для ранга матрицы A справедливо равенство: $\mathrm{rk}(A) = n - m$. Характеристический многочлен можно вычислить за время $O(\log^2 n)$ на $O(n^{4.5})$ процессорах [68], что и является мерой сложности параллельного вычисления ранга матрицы.

Для вычисления *подобия двух матриц* известен алгоритм с временем работы $O(\log^2 n)$ и полиномиальным числом процессоров [334], который основан на параллельном алгоритме отыскания ранга и теореме Диксона, утверждающей, что две матрицы подобны в том и только в том случае, если

$$\operatorname{rank}(A \otimes I - I \otimes B)^2 = \operatorname{rank}(A \otimes I - I \otimes A)\ \operatorname{rank}(B \otimes I - I \otimes B).$$

Параллельный алгоритм отыскания LU*–разложения* симметрической положительно определенной *матрицы* A можно получить, используя следующее тождество:

$$A_i = \begin{vmatrix} I & 0 \\ Y_i W_i^{-1} & I \end{vmatrix} \cdot \begin{vmatrix} W_i & 0 \\ 0 & A_{i+1} \end{vmatrix} \cdot \begin{vmatrix} I & W_i^{-1} X_i \\ 0 & I \end{vmatrix},$$

где

$$A_{i+1} = Z_i - Y_i\ W_i^{-1}\ X_i, \qquad A_i = \begin{vmatrix} W_i & X_i \\ Y_i & Z_i \end{vmatrix}, \quad A_0 = A.$$

Время работы алгоритма равно $O(\log^3 n)$ на CREW PRAM с $O(n^4/\log^2 n)$ процессорами.

Параллельный алгоритм отыскания QR*–разложения* можно построить, используя параллельный алгоритм отыскания LU–разложения симметрической положительно определенной матрицы. В самом деле, если

$$A'A = R'R, \quad \text{и} \quad Q = AR^{-1}, \quad \text{то} \quad Q'Q = I,$$

и A = QR – разложение матрицы A в произведение ортогональной и верхнетреугольной матриц. Сложность алгоритма по порядку такая же, как у алгоритма LU–разложения.

Параллельный алгоритм вычисления нормальной формы Хессенберга основан на использовании *матрицы Крылова*

$$K(A, v, m) = [v, Av, \ldots, A^{m-1}v].$$

(Подпространство, натянутое на столбцы матрицы Крылова, называется *подпространством Крылова*). Тождество

$$K(A, v, 2m) = [K(A, v, m), A^m K(A, v, m)]$$

показывает, что K(A, v, m) можно вычислить, выполнив O(log m) умножений матриц размером n×n и n×k, где k < m [66, с.128].

Задача приведения матрицы к нормальной форме Хессенберга

сводится к отысканию QR-разложения матрицы Крылова с помощью следующего утверждения (см. [25]):

Теорема 3 . *Если матрица* $K = K(A, v, n-1)$ *невырождена и* $K = QR$ – *ее QR-разложение, то* $Q'AQ$ *имеет верхнюю форму Хессенберга.*

Доказательство. Пусть

$$Q = [q_1, \ldots, q_n], \quad \text{тогда} \quad Q'AQ = \|q_i' A\, q_j\|.$$

Так как Q – ортогональная матрица, то q_i ортогонален первым $i-1$ ее столбцам и, следовательно, первым $i-1$ столбцам матрицы $Q = KR^{-1}$. Поэтому q_i ортогонален первым $i-2$ столбцам матрицы $AKR^{-1} = AQ$ и, следовательно, $q_i' A q_j = 0$ при $i > j+1$, то есть $Q'AQ$ имеет верхнюю форму Хессенберга.

Для *отыскания собственных значений якобиевой матрицы с точностью* $\varepsilon > 0$ известен алгоритм с временем работы $O(\log^4 n + \log^3 n\,(\log\log 1/\varepsilon)^2)$ на $O(n(\log^4 n + \log^3 n\,(\log\log 1/\varepsilon)^2))$ процессорах [121].

Быстрый параллельный алгоритм вычисления собственных значений якобиевой матрицы основан на рекурсивном соотношении для характеристических многочленов якобиевых матриц, методе локализации собственных значений якобиевой матрицы собственными значениями ее подматриц и быстром параллельном алгоритме вычисления значений характеристического многочлена якобиевой матрицы в нескольких точках.

Пусть $A = J - tI$, где J – якобиева матрица порядка n

$$J = \begin{pmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{pmatrix}, \quad \gamma_i = b_i \cdot c_{i+1} > 0,$$

B_i и C_j – верхний левый и нижний правый миноры матрицы A порядка i и j соответственно. Тогда выполнено реккурентое соотношение

i и j соответственно. Тогда выполнено реккуррентое соотношение

$$\det A = \det \begin{vmatrix} B_k & \gamma_k B_{k-1} \\ C_{n-k-1} & C_{n-k} \end{vmatrix}$$

Из этого соотношения вытекает, что собственные значения J (то есть корни det A) локализованы в последовательных интервалах, на которые вещественная прямая разбивается корнями многочленов B_k, B_{k-1}, C_{n-k-1}, C_{n-k}. Более точные приближения собственных значений могут быть получены с помощью следующего алгоритма.

Алгоритм REF($\alpha_0, \alpha_\infty, B_k, B_{k-1}, C_{n-k-1}, C_{n-k}, A, e$).

Вход. Элементы A, собственные значения B_k, B_{k-1}, C_{n-k-1}, C_{n-k}.

Выход. Приближения с точностью e собственных значений A.

Шаг 1. Отсортировать B_k, B_{k-1}, C_{n-k-1}, C_{n-k} в последовательность $\alpha_0 < \alpha_1 \le \ldots < \alpha_{2(n+k-1)} \le \alpha_{2(n+k)-1} < \alpha_\infty$.

Шаг 2. параллельно от i=0 до n–1 выполнить шаги 3–6.

Шаг 3. до $\alpha_{2i+1} - \alpha_{2i} \ge e$ выполнить шаги 4–6.

Шаг 4. положить $\beta_i = (\alpha_{2i} + \alpha_{2i+1})/2$.

Шаг 5. положить $z_i = A(\beta_i)$.

Шаг 6. если i mod 2 = 0 то если $z_i > 0$ то $\alpha_{2i} = \beta_i$ иначе $\alpha_{2i+1} = \beta_i$).

если i mod 2 = 1 то если $z_i > 0$ то $\alpha_{2i+1} = \beta_i$ иначе $\alpha_{2i} = \beta_i$).

Время работы алгоритма REF есть величина $O(n^2 \log M/e)$, поскольку n $\log M/e$ выполнений шага 5 требуют времени O(n) и глубина REF есть $O(n \log M/e)$. Эти значения могут быть уменьшены до $O(n \log^3 n \log M/e)$ и $O(\log^3 n \log M/e)$ соответственно, если использовать следующий быстрый параллельный алгоритм вычисления характеристического многочлена трехдиагональной матрицы в нескольких точках.

Пусть

$$d_k(t) = (a_k - t)\cdot d_{k-1}(t) - \gamma_k \cdot d_{k-2}(t), \quad k=2,\ldots,n,$$

$$d_1 = a_1 - t, \quad d_0 = 1,$$

– линейные рекуррентные соотношения для характеристического многочлена J. Рекуррентные соотношения можно переписать в матричной форме

$$\begin{vmatrix} d_n(t) \\ d_{n-1}(t) \end{vmatrix} = \begin{vmatrix} a_n - t & -\gamma_n \\ 1 & 0 \end{vmatrix} \begin{vmatrix} d_{n-1}(t) \\ d_{n-2}(t) \end{vmatrix} = \ldots = \prod_{i=n}^{1} Q_i(t) \begin{vmatrix} a_1 - t \\ 1 \end{vmatrix} .$$

Рассмотрим несколько более общую задачу быстрого вычисления матричных произведений

$$\prod_{i=n}^{1} Q_i(t_j), \quad j=1,\ldots,m$$

где элементы 2•2 матриц $Q_i(t)$ являются линейными функциями.

Элементы произведений s таких матриц являются многочленами степени меньшей или равной s и определяются единственным образом по значениям в s различных точках, например, в корнях из единицы $w_i = \exp(2\pi \sqrt{-1}\, i/s)$.

Пусть a – вектор значений многочлена f(t) в точках t_j, $j=1,\ldots,m$, y – вектор значений многочлена в корнях из единицы, $V(t_1,\ldots,t_m)$ – матрица Вандермонда, построенная на $t_1,\ldots,t_m$, $F_s = V(w_0,\ldots,w_{s-1})$ – матрица s-точечного преобразования Фурье. Тогда

$$a = V(t_1,\ldots,t_m) F_s y \qquad (2.3)$$

и, используя быстрое преобразование Фурье и быстрый алгоритм умножения матрицы Вандермонда на вектор, можно вычислить a по y за время $O(m \log^2 n)$ $(m \le n)$.

Пусть

$$Q_j^i(t) = Q_{(j-1)\cdot 2^i}(t) \cdot \ldots \cdot Q_{j\cdot 2^i - 1}(t), \quad i=1,\ldots,l; \quad j=1,\ldots,2^{l-i}.$$

Элементы матриц $Q_j^i(t)$ являются многочленами степени 2^i или меньше и они могут быть вычислены в корнях из единицы степени 2^i и затем, используя соотношение (2.3), – в других корнях степени 2^{i+1}.

Используя последние значения, можно прямо вычислить элементы матрицы $Q_j^{l+1}(t) = Q_{2j}^{l}(t) * Q_{2j+1}^{l}(t)$ в корнях.

Алгоритм заключается в следующем.

для l=1 **до** l **выполнить**

параллельно j=1 **до** 2^{l-1} **выполнить**

begin **параллельно** k=1 **до** 2^{l-l} **выполнить**

параллельно p,q=0 **до** 1

$$Q_j^l[p,q](w_{k,l}) = Q_{2j-1}^{l-1}[p,0](w_{k,l}) * Q_{2j-1}^{l-1}[0,q](w_{k,l}) +$$

$$Q_{2j-1}^{l-1}[p,1](w_{k,l}) * Q_{2j-1}^{l-1}[1,q](w_{k,l});$$

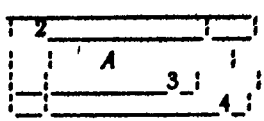
$$(Q_j^l[p,q](w_{1+2^l,l+1}),\ldots,Q_j^l[p,q](w_{2^{l+1}-1,l+1}))^t =$$

$$= F_{2^l} * F_{2^l}^{-1}\,(Q_j^l[p,q](w_{0,l}),\ldots,Q_j^l[p,q](w_{2^l-1,l}))^t;$$

end

где $w_{k,l} = \exp(2\pi \sqrt{-1}\, k/2^l)$, $F_s = V(w_{s,2s},\ldots,w_{2s-1,2s})$.

Пусть A[i], i=1,...,4 – миноры A, определенные ниже:



A[1] – (1,3), A[3] – (2,3),
A[2] – (1,4), A[4] – (2,4)

Пусть B[i] и C[i] (аналогично A[i]) – миноры левой верхней и нижней правой подматриц B и C матрицы A порядка k и n–k соответственно. Нетрудно видеть, что

$$A[1] = \det \begin{vmatrix} B[2] & \gamma_k & B[1] \\ C[3] & & C[1] \end{vmatrix} \qquad A[2] = \det \begin{vmatrix} B[2] & \gamma_k & B[1] \\ C[4] & & C[2] \end{vmatrix}$$

$$A[3] = \det \begin{vmatrix} B[4] & \gamma_k & B[3] \\ C[3] & & C[1] \end{vmatrix} \qquad A[4] = \det \begin{vmatrix} B[4] & \gamma_k & B[3] \\ C[4] & & C[2] \end{vmatrix}$$

Пусть A_j^i – подматрица матрицы A порядка 2^i, расположенная в строках и столбцах с номерами $j*2^i+1,\ldots,(j+1)*2^i$. Тогда следующий алгоритм вычисляет собственные значения матрицы J порядка $n=2^l$ (α_0 и α_∞ – априорные оценки для спектра).

Алгоритм EIGVAL(A,e).

Шаг 1. **для** i=1 **до** l **выполнить шаги** 2–3.

Шаг 2. **параллельно** j=0 **до** $2^{(l-i)}-1$ **выполнить шаг** 3.

Шаг 3. **положить** $k=j*2^i$;

$$REF(\alpha_0,\alpha_\infty,A^{i-1}_{2*j}[2],A^{i-1}_{2*j+1}[1],A^{i-1}_{2*j}[1],A^{i-1}_{2*j+1}[3],A^{i}_{j}[1],e);$$

$$REF(\alpha_0,\alpha_\infty,A^{i-1}_{2*j}[2],A^{i-1}_{2*j+1}[4],A^{i-1}_{2*j}[1],A^{i-1}_{2*j+1}[2],A^{i}_{j}[2],e);$$

$$REF(\alpha_0,\alpha_\infty,A^{i-1}_{2*j}[4],A^{i-1}_{2*j+1}[3],A^{i-1}_{2*j}[3],A^{i-1}_{2*j+1}[1],A^{i}_{j}[3],e);$$

Ряд алгебраических задач, для которых предложены параллельные алгоритмы, относится к абелевым группам подстановок.

Принадлежность абелевой группе.

Вход. Подстановки, порождающие абелеву группу G; подстановка h.

Вопрос. Является ли h элементом группы G ?

В [229] показано, что эта задача принадлежит NC^3.

Изоморфизм абелевых групп.

Вход. Подстановки, порождающие абелеву группу G; подстановки, порождающие абелеву группу H.

Вопрос. Изоморфны ли группы G и H как абстрактные группы ?

В [229] показано, что эта задача принадлежит NC^3.

Порядок абелевой группы.

Вход. Подстановки, порождающие абелеву группу G.

Выход. Разложение |G| на простые множители.

В [229] показано, что эта задача принадлежит NC^3.

Факторизация абелевой группы.

Вход. Подстановки, порождающие абелеву группу G.

Выход. Нетривиальные подстановки $h_1,\ldots,h_s \in G$ такие, что $|h_{i+1}|$ делит $|h_i|$, $i = 1,\ldots,s$ и $G = \langle h_1\rangle \times\ldots\times \langle h_s\rangle$.

В [229] показано, что эта задача принадлежит NC^4.

Эти и многие другие результаты о параллельной сложности решения задач в абелевых группах получены в [229] путем установления NC^1-сводимостей между этими задачами, рядом задач из теории графов (нахождения путн между заданнымн вершинами орграфа, существования путн заданной длины между двумя вершинами орграфа) и некоторыми задачами линейной алгебры. Ряд эффективных NC алгоритмов для групп перестановок предложен в [51], [223], [224].

2.2.2 Целочисленная арифметика

Естественной моделью параллельных вычислений с целыми числами представляются определенные выше булевы схемы. *Для сложения целых чисел* Офман [17] предложил булеву схему глубиной O(log n) и размером O(n). Брент [74] предложил схему глубиной $\log n + O(\log^{1/2} n)$ и размером O(n log n). Храпченко [27] построил схему размером 9n и глубиной $\log n + O(\log^{1/2} n)$. С помощью техники [208] несложно построить схему глубиной 2 log n и размером 8n.

Трудность построения схемы логарифмической глубины для сложения целых чисел связана с тем, что признак переноса может распространяться на произвольное число разрядов. Пусть требуется сложить два целых числа $a = \sum a_i 2^i$ и $b = \sum b_i 2^i$, представленных в двоичной записи. Для разрядов x результата $x = a + b$ и признаков переноса t_i можно написать рекуррентные соотношения:

$$x_i = a_i \oplus b_i \oplus t_{i-1}, \quad t_{-1} = 0,$$

$$t_i = \begin{cases} 0, & \text{если } a_i + b_i + t_{i-1} < 2, \\ 1, & \text{если } a_i + b_i + t_i \geq 2. \end{cases}$$

Следуя [17], покажем, как можно реализовать эти рекуррентные соотношения схемой глубины O(log n). Для этой цели обозначим $u_i = a_i + b_i$ и заметим, что

$$t_i = \begin{cases} 0, & \text{если } u_i = 0, \\ 1, & \text{если } u_i > 1, \\ u_i + t_{i-1}, & \text{если } u_i = 1. \end{cases}$$

Отсюда получаем, что для вычисления t_i нужно найти первую отлич-

ную от нуля компоненту справа от u_i, то есть $= \max(j < i \mid u_j \neq 1)$ и положить

$$t_i = \begin{cases} 0, & \text{если } u_k = 0, \\ 1, & \text{если } u_k > 1. \end{cases}$$

Для параллельного вычисления первой справа отличной от нуля компоненты можно воспользоваться схемой сдваивания. При этом, однако, получается схема размером $O(n^2)$ и глубиной $O(\log n)$. Для построения схемы размером $O(n)$ и глубиной $O(\log n)$ воспользуемся схемой глубины $O(\log n)$ для решения задачи о префиксе [208]. Для этого положим

$$t_i = F(a_i, b_i)(t_{i-1}), \text{ где } F(a,b)(t) = ab \,\&\, (a \vee b)t$$

и заметим, что

$$t_i = F(a_i, b_i)(t_{i-1}) = F(a_i, b_i) \circ F(a_{i-1}, b_{i-1}) \circ \ldots \circ F(a_0, b_0)(0),$$

где $\circ$ – ассоциативная операция композиции функций. Кроме того, нетрудно видеть, что $F(a,b) \circ F(x,y) = F(u,v)$ для некоторых u и v. Используя приведенные выше соотношения, можно показать, что задача сложения целых чисел лежит в AC^0. В [110] показано, что сложение нельзя реализовать схемой с неограниченной степенью входа, имеющей константную глубину и линейный размер.

Умножение целых чисел очевидным образом сводится к сложению n чисел. Для сложения n чисел Офман [17] предложил следующий метод, дающий для умножения схему глубины $O(\log n)$ и размера $O(n^2)$. Строим схему глубиной $O(1)$ для представления суммы трех слагаемых в виде суммы двух слагаемых. Используя дерево высоты $O(\log_{3/2} n)$, в вершинах которого стоят такие схемы, сводим задачу сложения n чисел к сложению двух чисел.

Позднее для умножения целых чисел Шенхаге и Штрассен [29] построили схему той же глубины, но размера $O(n \log n \log\log n)$. Их конструкция является равномерной и основана на схемах быстрого вычисления дискретного преобразования Фурье над конечными кольцами. В [123] установлено, что умножение не принадлежит классу AC^0. Это вытекает из основного утверждения этой работы о том, что функция четности n аргументов не принадлежит классу AC^0.

Алгоритм для деления целых чисел с временем работы $O(\log^2 n)$ на $O(n \log n \log\log n)$ процессорах приведен в [2]. Он основан на алгоритме быстрого умножения чисел и использовании ньютоновой итерации для вычисления $y = 1/x$. Величина y_1 вычисляется таблично с использованием нескольких первых разрядов x. Итерация Ньютона

$$y_{i+1} = 2\, y_i - y_i^2\, x$$

удваивает число верных знаков. Алгоритм с временем работы $O(\log n \log\log n)$ на $O(M(n))$ процессорах, где $M(n)$ – время умножения матриц, приведен в [279].

Неравномерное семейство схем глубиной $O(\log n)$ для деления целых чисел предложено в [55]. Это семейство строится последовательным сведением к вычислению аппроксимации обратной величины, вычислению степеней, вычислению произведения нескольких чисел и использует китайскую теорему об остатках.

Задача деления заключается в нахождении для данных n-разрядных чисел x и y n-разрядного представления числа $\lfloor x/y \rfloor$. В [55] показано, что задача деления целых чисел NC^1-сводится к задаче вычисления степеней целого числа, заключающейся в следующем: найти n^2-разрядное представление степеней x^i для $i = 0, 1, \ldots, n$. Верна также и обратная NC^1-сводимость. Для построения параллельного алгоритма деления используются следующие вспомогательные утверждения:

1) для n-разрядного x и $m \le n$ задачи вычисления $x \bmod m$, $\lfloor x/m \rfloor$ и $x^{-1} \bmod m$ (если обратный элемент существует) принадлежат NC^1.

2) для данных целых $x_1, \ldots, x_n$ и данной степени простого числа $p^l \le n$, где $x_1, \ldots, x_n \le p^l$, произведение $x_1 \cdot \ldots \cdot x_n \bmod p^l$ может быть вычислено в NC^1.

3) для n-разрядных целых a и b величину $a^b \bmod m$, где $m \le n$, можно вычислить в NC^1.

4) задача восстановления числа x по его остаткам от деления на попарно взаимно простые $c_1, \ldots, c_n$ (то есть нахождения $x \bmod \prod_{i=1}^{n} c_i$) при условии $1 < c_1 < \ldots < c_n \le n^2$ NC^1-сводится к задаче вычисления $c = \prod_{i=1}^{n} c_i$

Важную роль играет понятие хорошей модулярной последователь-

ности: $M_1, M_2 \ldots$ называется хорошей модулярной последовательностью если и только если существуют многочлены $q(n)$ и $r(n)$ такие, что для всех n:

1) $2^n \le M_n \le 2^{q(n)}$.

2) для всякого простого p, из $p^l \mid M_n$ вытекает, что $p^l \le r(n)$.

Основу конструкции составляет следующее утверждение:

вычисление произведения n n-разрядных чисел $x_1 \cdot \ldots \cdot x_n$ NC^1-сводится к задаче вычисления хорошей модулярной последовательности (M_n).

Это делается следующим образом: из определения вытекает, что $M_{n^2} \ge 2^{n^2} > x_1 \cdot \ldots \cdot x_n$. Находим M_{n^2} и разлагаем его на простые множители $c_i = p_i^{l_i}$, $i = 1, \ldots, s$. Вычисляем параллельно $b_{ij} = x_i \bmod c_j$ для $i = 1, \ldots, n$, $j = 1, \ldots, s$. Находим $b_j = \prod_{i=1}^{n} b_{ij} \bmod c_j$. Вычисляем $\prod_{i=1}^{n} x_i \bmod M_{n^2}$, используя китайскую теорему об остатках.

Заключительное утверждение, из которого вытекает основной результат, выглядит следующим образом: произведение n-разрядных чисел $x_1 \cdot \ldots \cdot x_n$ можно вычислить P-равномерным семейством булевых схем глубины $O(\log n)$. Для доказательства нужно построить хорошую модулярную последовательность. За полиномиальное время найдем n первых простых чисел и вычислим их произведение. Поскольку n-е простое не превосходит $O(n \log n)$, то их произведение есть $2^{O(n \log n)}$. Также очевидно, что это произведение $\ge 2^n$. Значит произведение первых n простых чисел образует хорошую модулярную последовательность.

Для обращения в конечном поле $GF(2^n)$ известен NC-алгоритм с временем работы $O(\log n)$ [217].

Быстрое преобразование Фурье. Дискретным преобразованием Фурье (ДПФ) вектора a называется вектор $b = F_n a$, где F_n – матрица Вандермонда $V(1, w, \ldots, w^{n-1})$ и w – корень степени n из единицы в поле комплексных чисел. Прямая реализация преобразования Фурье дает алгоритм с временем $O(\log n)$ на $O(n^2)$ процессорах. Используя специфические свойства матрицы преобразования Фурье, Кули и Тьюки

предложили алгоритм быстрого преобразования Фурье (БПФ), выполняющий ДПФ за время O(log n) на O(n) процессорах. Алгоритм БПФ основан на следующем тождестве:

$$F_{rs} = (F_r \otimes I_s)D_r^s(I_r \otimes F_s)P_r^s$$

где D_r^s – диагональная матрица, I_r и I_s – единичные матрицы, P_r^s – перестановочная матрица [282].

Теоретико-числовое преобразование (ТЧП). Пусть $p = 2^n + 1$ – простое число (число Ферма), $F_{n+1} = V(1,w,\ldots,w^n)$, где w – корень степени $n + 1$ из единицы в $\mathbb{Z}/p\mathbb{Z}$ (например, $w = 2$). Теоретико-числовым преобразованием вектора вычетов $a = (a_0,\ldots,a_n)$ называется произведение $b = F_{n+1}{\cdot}a$, а $n + 1$ – называется порядком преобразования.

ТЧП порядка n над $\mathbb{Z}/p\mathbb{Z}$ может быть вычислено схемой глубины $O(\log(n \log p))$ и размера $O(n \log p \log(p \log n))$. Использование ТЧП вместо БПФ и аналога теоремы о свертке позволяют перенести многие результаты о степенных рядах на целые числа.

Если N – степень 2, то схема умножения m N-разрядных чисел по модулю $2^n + 1$ имеет глубину $O(\log m \log\log N + \log N)$ и размер $(mN)^{O(1)}$.

Вычисление значения многочлена степени n *в точке с точностью* $o(2^{-n})$ может быть выполнено схемой глубиной $O(\log n (\log\log n))$ и размером $n^{O(1)}$.

2.2.3 Ряды и многочлены

Наилучшие известные булевы схемы *для вычисления элементарных функций* были предложены в [74] и в [207]. Булевы схемы для вычисления элементарных функций, как и схема вычисления обратной величины, используют $\Omega(\log n)$ итераций Ньютона второго порядка, каждая из которых требует умножения целых чисел и, следовательно, имеют глубину $\Omega(\log^2 n)$.

Основной аппарат, используемый для построения схем логарифмической глубины для алгебраических функций, – многомерные свертки и их вычисление с помощью БПФ. Ниже приводятся равномерные схемы глубиной $O(\log n \log\log n)$ для вычисления элементарных функций.

Умножение многочленов. Теорема о свертке заключается в том, что для умножения двух многочленов можно вычислить значения сомножителей в нескольких точках, перемножить полученные значения и, используя интерполяционные формулы, вычислить коэффициенты искомого произведения. При этом число точек, в которых производится вычисление значений, должно быть больше степени произведения. Если в качестве таких точек выбрать корни из единицы, то как вычисление значений многочленов, так и интерполяция сводятся к прямому и обратному БПФ, и теорема о свертке для умножения многочленов степени n может быть записана в матричной форме:

$$c = F^{-1}(Fa \cdot Fb)$$

где $a = (a_0,\ldots,a_n,0,\ldots,0)^t$ и $b = (b_0,\ldots,b_n,0,\ldots,0)^t$ – векторы размерности 2n, а $\cdot$ обозначает покомпонентное произведение векторов. Отсюда следует, что произведение многочленов можно вычислить схемой глубиной O(log n) и размером O(n log n).

Используя теорему о свертке для произведения нескольких многочленов и метод сдваивания, нетрудно показать, что произведение $A_1(x)\cdot\ldots\cdot A_m(x)$, где $A_i(x)$ – многочлены степени n, могут быть вычислены схемой глубиной O(log (mn)) и размером O(mn log (mn)). То же самое верно и для вычисления произведения многочленов и степени многочлена по модулю $x^n + 1$.

Вычисление значения многочлена и всех его производных в точке (полная схема Горнера). Согласно определению,

$$f^{(k)}(x) = \sum_{i \le n-k} (k+i)!/i!\; a_{i+k}\, x^i$$

и вычисление многочлена и всех его производных сводится к умножению вектора на диагональную и ганкелеву матрицы (матрицу, у которой элементы, стоящие на каждой диагонали, параллельной побочной, одинаковы). Для умножения ганкелевой матрицы на вектор известен параллельный алгоритм с временем работы O(log n) на O(n) процессорах. Это можно сделать с помощью параллельного вычисления БПФ, если вспомнить, что любая ганкелева матрица может быть представлена как сумма циркулянта и антициркулянта (см. раздел 1.2.2).

Аналогичная техника применима и в случае вычисления *частных*

производных многочлена от нескольких переменных заданного алгебраической схемой. Пусть для многочлена $F(x_1,\ldots,x_m)$ задана алгебраическая схема его вычисления, принадлежащая NC (примером такого многочлена может служить определитель) и требуется вычислить все его частные производные до k-й включительно по переменной x_i. Используя известные соотношения:

$$\frac{\partial^k(f+g)}{\partial x_i^k} = \frac{\partial^k f}{\partial x_i^k} + \frac{\partial^k g}{\partial x_i^k} \quad \text{и}$$

$$\frac{\partial^k(f\cdot g)}{\partial x_i^k} = \sum_{j\le k} \binom{k}{j} \frac{\partial^j f}{\partial x_i^j} \cdot \frac{\partial^{k-j} g}{\partial x_i^{k-j}}$$

исходную алгебраическую схему нетрудно преобразовать в схему, вычисляющую все нужные частные производные [50]. Для этого достаточно для каждого элемента сложения $f + g$ добавить k элементов сложения их частных производных, вычисленных на предыдущих шагах, а для каждого элемента умножения $f\cdot g$ добавить схему, параллельно вычисляющую частные производные по указанному выше правилу. При этом сходным с описанным выше способом для многочлена от одной переменной можно представить вычисления в виде нескольких умножений теплицевой и диагональной матриц на вектор, что гарантирует наличие для каждой вершины дополнительной схемы, вычисляющей все частные производные по некоторой переменной (до n-го включительно), размером $O(n \log n)$ и глубиной $O(\log n)$. Таким образом, если исходная алгебраическая схема, вычисляющая многочлен от n переменных, имела размер L и глубину d, размером $O(L\, n \log n)$ и глубиной $O(d \log n)$ то ее можно преобразовать в схему, вычисляющую и все частные производные по любой переменной (до n-го включительно).

Вычисление НОД многочленов и всех неполных частных может быть осуществлено схемой глубины $O(\log^2 n)$, [135] на основе следующего факта из алгебры. Пусть f и g – многочлены степеней n и m ($n \ge m$):

$$f(t) = \sum_{i=0}^{n} f_i t^i, \qquad g(t) = \sum_{i=0}^{m} g_i t^i \qquad \text{и}$$

$$R = \begin{vmatrix} f_0 & & & g_0 & & \\ f_1 & f_0 & & g_1 & g_0 & \\ \vdots & & \ddots & \vdots & & \ddots \\ f_n & f_{n-1} & & g_m & g_{m-1} & \\ & & \ddots & & & \vdots \\ & & & f_n & & g_m \end{vmatrix}$$

– *матрица Сильвестра* многочленов f и g, R_i – матрица порядка n+m–2i, расположенная в левом верхнем углу матрицы R. Степень наибольшего общего делителя многочленов f и g равна наибольшему i, такому, что $\det R_{i-1} = 0$. Вычисляя $\det R_i$ параллельно, можно за $O(\log^2 n)$ параллельных шагов найти степень НОД. В частности, степень НОД равна нулю, если *результант* f и g – det R отличен от нуля.

Последнее утверждение понять совсем легко. В самом деле, deg НОД(f,g) > 0 тогда и только тогда, когда существуют многочлены x,y, deg x < deg g, deg y < deg f, такие что f x + g y = 0. Последнее соотношение можно записать, используя матрицу Сильвестра, как систему линейных уравнений относительно коэффициентов многочленов x и y:

$$R\,(x_0,\ldots,x_{m-1},y_0,\ldots,y_{n-1})^t = 0,$$

которая имеет ненулевое решение в том и только в том случае, когда det R = 0. Интересно отметить, что наилучший известный алгоритм вычисления НОД n–разрядных чисел имеет глубину $O(n \log\log n/\log n)$, и принадлежность этой задачи классу NC не известна [179].

Быстрое параллельное вычисление степеней в факторкольце многочленов над конечным полем. В [6] показано, что вычисление $x^m \bmod f$, где f – многочлен степени n над GF(q), можно выполнить со сложностью $O((\log (qn \log_q m)) \log_2)$ на

$$O\left[\frac{\max(q,\sqrt{\log_q m}\ n^{\alpha-2},\ \log_q m)}{\log (qn \log_q m)} \frac{n^2}{\log n}\right]$$

процессорах. Этот алгоритм основан на следующем соотношении:

$$\langle a^{q^s} \rangle = \langle a \rangle B^s,$$

где $\langle a \rangle$ – коэффициенты остатка от деления a на f, B – матрица Берлекэмпа многочлена f:

$$B = \begin{vmatrix} 1 \\ A^q \\ \vdots \\ A^{q(n-1)} \end{vmatrix},$$

а A – сопровождающая матрица многочлена f. Кроме того, в доказательстве используется следующее утверждение:

Лемма. Пусть R – произвольное коммутативное кольцо с единицей, $\alpha > \omega(R)$, $n, r \in \mathbb{N}$, M – $n \times n$ матрица, z_i, $i = 1,\ldots,r$ – векторы размерности r. Набор векторов $(z_i M^i)$, $i = 1,\ldots,r$ можно вычислить со сложностью $O((\log r)\log n)$ на

$$O\left[\frac{\max(r, \sqrt{r\ n})}{\log r} \frac{n^{\alpha-1}}{\log n}\right] \quad \text{процессорах.}$$

Набор векторов (zM^i), $i = 1,\ldots,r$ можно вычислить со сложностью $O((\log \max(r,n))\log n)$ на

$$O\left[\frac{\max(r,n)}{\log \max(r,n)} \frac{n^{\alpha-1}}{\log n}\right] \quad \text{процессорах.}$$

Элементарные функции и операции над степенными рядами. Первые n членов композиции двух степенных рядов, элементарные функции и элементарные симметрические функции с вещественными коэффициентами можно вычислить схемами глубины $O(\log n)$. Это следует из схемы логарифмической глубины для вычисления произведения многочленов [275].

Деление, интерполяция и обращение степенных рядов также могут быть вычислены схемами глубины $O(\log n)$. Пусть

$$A(x) = 1 + \sum_{i=1}^{n} a_i x^i \in R[[x]],$$

есть степенной ряд с вещественными коэффициентами. Тогда $A^{-1}(x) = \sum (1 - A(x))^i$ может быть вычислен схемой глубины

$O(\log n)$.

Интерполяция многочлена по заданным вычетам и реверсирование степенного ряда (относительно операции композиции степенных рядов) могут быть выполнены схемой глубины $O(\log n)$ [275].

Вычисление значения многочлена степени n *в точке с точностью* $o(2^{-m})$ может быть выполнено схемой глубины $O(\log n\ (\log\log n))$ и размера $n^{O(1)}$.

Вычисление значения элементарной функции на интервале, где сходимость ряда Тейлора имеет порядок $o(2^{-m})$, можно выполнить схемой глубины $O(\log n \log\log n)$ и размера $n^{O(1)}$.

Если для аппроксимации обратной величины использовать формулу

$$r = \prod_{i=0}^{\log(n+1)-1} (1 + (1 - 2^{-m}a)^{2^i}),$$

то получим, что $|r - a^{-1}| = o(2^{-m})$. Используя построенные выше схемы малой глубины для операций над степенными рядами, можно построить булеву схему для аппроксимации обратной величины, имеющую глубину $O(\log n \log\log n)$ и размер $n^{O(1)}$ [279].

Аппроксимация Паде степенного ряда $f(t) = \sum_{i=0}^{\infty} f_i t^i$. Пара многочленов $(p(t),q(t))$ называется (m,n)-аппроксимацией Паде для $f(t)$, если $\deg p(t) = n$, $\deg q(t) = m$ и

$$f(t)q(t) - p(t) = O(t^{n+m+1}).$$

Отыскание аппроксимаций Паде сводится к решению систем линейных уравнений. Пусть

$$p(t) = \sum_{i=0}^{n} p_i t^i, \quad p_n = 1, \qquad q(t) = \sum_{i=0}^{m} q_i t^i,$$

тогда из условия аппроксимации получаем системы линейных уравнений:

$$\begin{vmatrix} f_n & & f_{n-m} \\ \vdots & \ddots & \vdots \\ f_{n+m} & & f_n \end{vmatrix} \begin{vmatrix} q_0 \\ \vdots \\ q_m \end{vmatrix} = \begin{vmatrix} p_n \\ 0 \\ \vdots \\ 0 \end{vmatrix},$$

$$\begin{vmatrix} p_0 \\ \vdots \\ p_{n-1} \end{vmatrix} = \begin{vmatrix} f_0 & & 0 \\ \vdots & \ddots & \vdots \\ f_m & & f_0 \\ \vdots & & \vdots \\ f_{n-1} & & f_{n-m-1} \end{vmatrix} \begin{vmatrix} q_0 \\ \vdots \\ q_m \end{vmatrix},$$

Применяя параллельные алгоритмы решения систем линейных уравнений и умножения матрицы на вектор, получаем NC-алгоритм для вычисления аппроксимации Паде с временем работы $O(\log^2(n+m))$.

Интерполяция многочленов нескольких переменных над конечными полями. Пусть задан многочлен f над полем GF(q) в виде схемы, способной вычислять значение f в расширениях поля GF(q). Известно, что число ненулевых коэффициентов многочлена равно t. Требуется, вычисляя значения многочлена в некотором расширении $GF(q^s)$ поля GF(q) (выбирая s как можно меньше), проинтерполировать многочлен, то есть найти все его ненулевые коэффициенты в GF(q).

В [152] предложен детерминированный параллельный алгоритм решения этой задачи, используя вычисления f в поле $GF(q^{\lceil 2\log_q(nt)+3\rceil})$ за время $O(\log^3(ntq))$ на PRAM с $O(n^2t^6\log^2(ntq) + q^{2.5}\log^2 q)$ процессорами. Для фиксированных полей время работы алгоритма есть $O(\log^3(nt))$ на $O(n^2t^6\log^2(ntq))$ процессорах.

Отметим, что при подсчете времени работы алгоритма время вычисления значения полинома в точке полагается равным 1 (так называемая оракульная сложность алгоритма). В качестве основной процедуры, к которой алгоритм интерполяции обращается рекурсивно, используется специальный параллельный алгоритм проверки тождественности нулю полинома.

Этот алгоритм состоит из нескольких этапов.

1. Выбрать $s = \lceil 2\log_q(nt) + 3 \rceil$ и построить поле $GF(q^s)$, рассматривая все многочлены степени s с коэффициентами из GF(q) и проверяя неприводимость многочленов с помощью алгоритма Берле-

кэмпа. После нахождения неприводимого многочлена φ получим, что $GF(q^S)$ изоморфно $GF(q)[z]/\varphi$. Найдем образующий ω циклической группы $GF(q^S)$ следующим образом. Пусть $q^S - 1 = \prod p_i^{n_i}$ – разложение на простые множители. Для любого

$$a \in GF(q^S)$$

вычислим $a^{\frac{q^s-1}{p_i}}$ для всех i, используя двоичное представление показателя.

2. Положить $N = \left\lceil \frac{q^s - 1}{4\ nq} \right\rceil$ и с помощью решета Эратосфена найти простое число p такое, что $2N < p \le 4N$. Построить NxN матрицу Коши $C = \| 1/(i + j) \|$ над полем GF(p). Известно, что все ее миноры отличны от нуля.

3. Вычислить $c_{ij} = 1/(i + j) \bmod p$.

4. Пусть $B = \|b_{ij}\|$ – произвольная подматрица матрицы C размера Nxn. Для каждой строки матрицы B и для всех $0 \le l < t$ вычислить значение многочлена f в точке $(x_1,\ldots,x_n)$, где $x_j = \omega^{lb_{ij}}$.

Ключевым является следующее утверждение: если $f(x) \not\equiv 0$, то в некоторой точке $f(\omega^{lb_{ij}}) \neq 0$, где вместо x_j подставляется значение $\omega^{lb_{ij}}$. Оно доказывается следующим образом: сначала показывается, что все мономы различны при значениях переменных $\omega^{b_{ij}}$, а затем, подставляя последовательно степени элемента $\omega^{b_{ij}}$, вычисляются в них значения функции. Если найдется единичное значение, то многочлен тождественно не равен нулю, а если $f(\omega^{lb_{ij}}) = 0$ при $l = 1,2,\ldots,t$, то делается вывод, что $f(x) \equiv 0$, поскольку соответствующая txt матрица является матрицей Вандермонда. Описанная процедура применяется затем рекурсивно для восстановления всех ненулевых коэффициентов многочлена.

Нормальные формы Эрмита и Смита для матриц над кольцом многочленов.

В то время как для целочисленных матриц построению парал-

лельных алгоритмов нахождения нормальных форм Эрмита и Смита препятствует отсутствие параллельного алгоритма вычисления наибольшего общего делителя, нахождение нормальной формы Эрмита матрицы с полиномиальными элементами NC^1-сводится к решению систем линейных уравнений, а задача отыскания нормальной формы Смита такой матрицы принадлежит классу RNC^2 [177].

Нормальной формой Смита называется диагональная матрица S, диагональные элементы d_i которой имеют старшие коффициенты, равные 1, и многочлен d_i делится на многочлен d_{i+1}. Матрица S называется нормальной формой Смита матрицы A, если существуют унимодулярные матрицы P и Q такие, что A = PSQ. Известно, что d_i есть наибольший общий делитель миноров порядка i матрицы A. Для нахождения d_i можно поступить следующим образом: выбрать полиномиальные nxn матрицы R,S,U,V и вычислить B = RAS, C = UAV. Вычислим δ_i – наибольший общий делитель главных ixi миноров матриц B и C. Оказывается, что элементы матриц R,S,U,V для которых $d_i \neq \delta_i$ удовлетворяют уравнению $P_i = 0$, где P_i – многочлен степени, не превосходящей $4i^2d$. Поэтому, выбирая элементы матриц R, S, U, V случайно из некоторого множества S, получим, что

$$Prob(d_i = \delta_i) \geq 1 - 4n^3d/|S|.$$

Отсюда нетрудно вывести, что задача отыскания нормальной формы Смита принадлежит RNC^2.

Для приближенного вычисления корней многочленов с заданной точностью $\varepsilon > 0$ при условии, что все корни вещественны, предложен NC-алгоритм с временем работы

$T = O(\log^3 n(\log n + \log \log^2(L/\varepsilon))$ и числом процессоров $O(nT)$ [58], [121], где L – длина записи многочлена. NC-алгоритм решения этой задачи в произвольном случае предложен в [25].

База максимального веса в матроиде.

Вход: Матроид с неотрицательными весами элементов.

Выход: База матроида с максимальной суммой элементов.

Известен так называемый жадный параллельный алгоритм нахождения базы максимального веса матроида [95]. Пусть задан матроид $M = (V,A)$ и элементам множества V приписаны веса. Напомним, что матроидом называется пара (V,A), где A – семейство подмножеств множества V, удовлетворяющее условиям:

1) если некоторое множество принадлежит A, то все его подмножества также принадлежат A.

2) для любого $V' \subseteq V$ все максимальные по включению подмножества (называемые базами) из сужения A на V' имеют одинаковую мощность.

Пусть $rk(V')$ – ранговая функция матроида, равная мощности максимального по включению подмножества из сужения A на V'. Параллельный жадный алгоритм поиска базы матроида максимального веса основан на следующем утверждении:

Пусть $V = (v_1,\ldots,v_n)$ – перечисление элементов множества V в порядке неубывания их весов. Тогда

$$B = \{v_i \mid rk(v_1,\ldots,v_i) > rk(v_1,\ldots,v_{i-1})\}$$

есть база матроида максимального веса. Алгоритм параллельно применяется к отсортированной в порядке неубывания весов последовательности элементов $V = (v_1,\ldots,v_n)$ и отмечает те ребра, для которых выполняется соотношение:

$$rk(v_1,\ldots,v_i) > rk(v_1,\ldots,v_{i-1}).$$

Отмеченные элементы будут образовывать базу матроида максимального веса. Таким образом, задача свелась к параллельному вычислению ранговой функции в матроиде. Отметим, что параллельный жадный алгоритм параллельно выполняет n вычислений функций rk, что приводит к эффективным параллельным алгоритмам нахождения базы матроида максимального веса, при условии, что для вычисления функции ранга имеется эффективный параллельный алгоритм (как например в задаче об остовных деревьях минимального веса в графе,

где вычисление ранга сводится к нахождению числа компонент связности подграфа с заданным множеством ребер, и задаче о нахождении базиса в данном множестве векторов, где ранг в матроидном определении совпадает с алгебраическим рангом, для вычисления которого известен NC-алгоритм над любым конечным полем [241], [83]).

Сумма подмножеств.

Вход: Массив натуральных чисел $a_1,\ldots,\ a_n$ и число K.

Вопрос: Существует ли булев вектор $x_1,\ldots,x_n$, являющийся решением уравнения

$$\sum_{i=1}^{n} a_i x_i = K \ ?$$

Если числа заданы в двоичной записи, то задача является NP-полной [8]. Покажем, что если числа заданы в унарной системе, то задача принадлежит NC.

Рассмотрим производящую функцию

$$Q_n(y) = \prod_{i=1}^{n} (1 + y^{a_i}).$$

Заметим, что коэффициент при члене y^a равен числу решений уравнения:

$$\sum_{i=1}^{n} a_i x_i = a$$

Поэтому достаточно вычислить многочлен $Q_n(y)$. Это можно сделать, перемножая выражения в скобках по методу сдваивания. При этом число членов в каждом получающемся промежуточном полиноме не превосходит $A = \sum_{i=1}^{n} a_i$, что является полиномиальной по длине входа величиной, если числа заданы в унарной системе. Перемножить два полинома степени A можно за время O(log A) на EREW PRAM с $O(A^2)$ процессорами, получая A копий первого полинома за время O(log A), перемножая переменные за один шаг и приводя подобные члены по методу сдваивания за время O(log A). Общее время работы алгоритма O(log n log A) на EREW PRAM с $O(nA^2/(\log n \log A))$ процессорами. Если для умножения полиномов степени n пользоваться быстрыми параллельными алгоритмами, основанными на преобразовании Фурье и

имеющими время работы O(log n) на O(n) процессорах, то для достижения времени O(logn log A) достаточно O(An/log n) процессоров.

ε-оптимальный рюкзак.

Вход: Массив весов $c_1,\ldots, c_n$ и объемов $a_1,\ldots, a_n$, объем рюкзака W.

Выход: Булев вектор $x_1,\ldots,x_n$, являющийся ε-оптимальным решением задачи о рюкзаке:

$$\sum_{i=1}^{n} c_i x_i \longrightarrow \max \quad , \qquad x_i \in \{0,1\} \ , \quad i=1,\ldots,n$$

$$\sum_{i=1}^{n} a_i x_i \le b \qquad\qquad a_i,\ c_i \ge 0$$

Прямое распараллеливание ε-оптимального полиномиального алгоритма методом сдваивания дает параллельный алгоритм решения этой задачи за время O(log n log (n/ε)) на PRAM с $O(n^3/\varepsilon^2)$ процессорами. При этом последовательный алгоритм с временем работы $O(n^3/\varepsilon^2)$ и памятью O(n/ε) преобразуется в алгоритм с параллельным временем работы O(log nlog (n/ε)) на PRAM с $O(n^3/\varepsilon^2)$ процессорами.

Это делается следующим образом. Осуществим сначала параллельную сортировку последовательности a_i/c_i в невозрастающем порядке и положим максимально возможное число первых в этом порядке x_i равными единице так, чтобы еще было выполнено неравенство-ограничение. Полученное допустимое решение x_G отличается от оптимального решения x_{opt} по значению целевой функции $f = \sum c_i x_i$ не более, чем в два раза, т.е. $f(x_{opt})/ f(x_G) \le 2$. Отбросим далее младшие t битов в каждом c_i, где t определяется как максимальное значение, удовлетворяющее условию:

$$n\ 2^t \le f(x_G)\ \varepsilon/2 \quad , \qquad t \ge 0.$$

При этом, если такого t не существует, то полагаем t=0.

Рассмотрим теперь полином

$$Q_n(x,y) = \prod_{i=1}^{n} (1 + x^{c_i} y^{a_i}).$$

Заметим, что коэффициент при члене $x^c y^a$ равен числу решений системы:

$$\sum_{i=1}^{n} c_i x_i = c, \qquad \sum_{i=1}^{n} a_i y_i = a$$

Кроме того, оптимум в редуцированной системе (с отброшенными битами в c_i) не превосходит величины

$$\frac{f(x_{opt})}{2^t} \le \frac{4n\, f(x_{opt})}{\varepsilon\, f(x_G)} \le \frac{4n}{\varepsilon}$$

Теперь мы можем вычислять $Q_n(x)$, отбрасывая все члены вида $x^c y^a$, где $c > 4n/\varepsilon$, а также члены вида $x^d y^r$, если существует член $x^d y^p$ с $p < r$. Это раскрытие скобок и отбрасывание членов можно осуществить по методу сдваивания. При этом длина выражения в каждой скобке не превосходит $4n/\varepsilon$, поэтому два выражения в скобках можно перемножить за время $O(\log(n/\varepsilon))$ на PRAM с $O(n^2/\varepsilon^2)$ процессорами также по методу сдваивания. Проделав $O(\log n)$ таких итераций, вычислим $Q_n(x)$. Общее число процессоров для такой процедуры $O(n^3/\varepsilon^2)$ и общее время работы $O(\log n \log(n/\varepsilon))$.

Возможны и другие варианты ε-оптимального параллельного алгоритма в задаче о рюкзаке, использующие меньшее число процессоров, но имеющие большую сложность.

Задача о покрытии и α-глубине (0,1)-матриц

Задача о покрытии имеет многочисленные приложения в различных областях, одной из которых является вычислительная геометрия [114], [78]. Приведем точную формулировку задачи.

Вход: Матрица A с элементами $a_{ij} \in (0,1)$, натуральное число $\alpha \ge 1$.

Выход: Наименьшее число строк матрицы A таких, что в образованной ими подматрице в каждом столбце имеется не менее α единиц (α-покрытие).

Такая задача называется также задачей об α-глубине (0,1)-матрицы. В комбинаторике и с точки зрения приложений (в частности в вычислительной геометрии [78],[60]) задача часто рассматривается при дополнительном ограничении, что число единиц в каждом столбце не меньше k. Такой класс матриц с n строками и m

столбцами будем обозначать через A(n,m,k), а величину минимального α-покрытия (α-глубину) матрицы A через $\varepsilon_\alpha(A)$.

В [60] предложен алгоритм для задачи об α-покрытии, который на O(n) процессорах работает время $O(\log m \log^3 n)$ и строит оптимальное по порядку покрытие кратности $O(\alpha)$ и мощности $O(n\alpha/k)$ при $\alpha = O(\log m)$. Идея этого алгоритма заключается в использовании 2-независимых случайных величин для доказательства существования нужного решения и описанного выше метода свертки и дерандомизации для построения детерминированного параллельного алгоритма. Строки матрицы A выбираются в покрытие попарно независимо и случайно с вероятностью δ/k. Затем как в модифицированном жадном алгоритме [78] вес каждого покрытого элемента уменьшается вдвое и осуществляется следующая итерация. Показывается, что при некоторых условиях выбор попарно независимых случайных величин позволяет уменьшить вес в $(1/2)(\delta - 2\delta^2)$ раз (вес равен сумме весов элементов, а начальные веса элементов равны 1). Таким образом, после $O((\log m)/\delta)$ итераций можно уменьшить вес от m до $1/m$, что достаточно для $O(\log m)$-покрытия. Мощность получаемого покрытия при этом по порядку не превосходит величины

$$O(((\log m)/\delta)(\delta n/k)) = O((n \log m)/k).$$

Процедура дерандомизации осуществляется по методу [220] путем подбора специальной целевой функции $f(x)$, для которой достаточно найти x^*: $f(x^*) \geq M$ [$f(x)$].

Задача о покрытии с гарантированной оценкой отклонения от оптимума не более чем в $(1 + \varepsilon)\log d$ *раз.*

Вход: Множество V и система его подмножеств E, число $\varepsilon > 0$.

Выход: Покрывающая V подсистема подмножеств из E с числом подмножеств не более чем в $(1 + \varepsilon)\log d$ раз большим числа подмножеств в минимальном покрытии (d - максимальное число подмножеств из E, содержащих какой либо элемент из V).

Здесь приведена эквивалентная формулировка задачи о покрытии в терминах систем подмножеств. Последовательный алгоритм построения покрытия с числом подмножеств, превышающим минимальное не более, чем в $1 + \log d$ раз , является жадным алгоритмом, выбирающим на каждом шаге подмножество, покрывающее максимальное число непокрытых элементов. Известен параллельный алгоритм решения этой задачи за время $O(\log^2 n \log m)$ на PRAM с O(n) процессо-

рамн, где n =|V| +|E| [19]. Алгоритм основан на том факте, что в сбалансированной системе подмножеств (где каждый элемент содержится в одинаковом числе подмножеств) случайный выбор с вероятностью (log d)/d каждого подмножества оставляет непокрытой 1/d долю всех элементов, которую можно покрыть произвольно, и получается покрытие с теми же характеристиками, что и покрытие, получаемое жадным алгоритмом.

Можно показать, что в случае сбалансированной системы подмножеств для построения покрытия, отличающегося по мощности от минимального не более, чем в $(1+\varepsilon)(1 + \log d)$ раз, попарной независимости случайных величин оказывается достаточно, если выбирать каждое подмножество с вероятностью δ/d (δ – некоторая малая константа) и рассматривать один такой выбор в качестве итерации. После каждой итерации удаляются уже покрытые элементы и выполняется очередная итерация. Таким образом, можно использовать вероятностные пространства полиномиального объема для дерандомизации описанной итерации и получения детерминированного параллельного алгоритма [220],[39],[60].

В случае произвольной системы подмножеств все подмножества разбиваются на классы так, что j-му классу принадлежат подмножества с числом элементов от $(1 + \varepsilon)^{j-1}$ до $(1 + \varepsilon)^{j}$. Затем в каждом полученном классе исходное множество элементов разбивается на подмножества так, что i-му подклассу принадлежат те элементы, которые содержатся не менее, чем в $(1 + \varepsilon)^{i-1}$, и не более, чем в $(1 + \varepsilon)^{i}$ подмножествах j-го класса. Это делается с целью приближения задачи к рассмотренному случаю сбалансированной системы подмножеств. Алгоритм состоит из нескольких фаз, разбитых на подфазы, соответствующие описанным выше подклассам. Более точно, подфаза (i,j) заключается в выборе подмножеств до тех пор, пока не будут покрыты элементы i-го подкласса. При этом, если имеются подмножества, покрывающие не менее $\delta^3/(1 + \varepsilon)$ доли от всех элементов подкласса, то выбирается одно из таких подмножеств. В противном случае случайным образом выбирается система подмножеств P, покрывающая не менее $|P| (1 + \varepsilon)^{i-1} (1 - 6\delta - 2\varepsilon\delta)$ элементов, включая $\delta/2$ долю элементов i-го подкласса. Для того, чтобы случайная система обладала этим свойством, оказывается достаточно

попарной независимости случайных величин и можно применить описанный ранее способ дерандомизации для построения детерминированного параллельного NC-алгоритма.

Нахождение ε-хорошей раскраски в задаче о балансировке множеств.

Вход: Множество $V = (v_1,\ldots,v_m)$ и система его подмножеств $S = (S_1,\ldots,S_n)$, числю $\varepsilon > 0$.

Выход: Вектор цветов $X = (x_1,\ldots,x_m)$ с компонентами из множества $\{+1, -1\}$такой, что для любого i

$$\Delta(S,x) = \max_i \left| \sum_{j:\, v_j \in S_i} x_j \right| \le d^{1/2} + \varepsilon \sqrt{\log n},$$

где d – максимальный размер множества в семействе. Известны алгоритмы решения этой задачи на PRAM с полиномиальным числом процессоров за время $O(\log^3 n)$ [59],[240]). Мы опишем сейчас этот алгоритм и его простую модификацию с временем работы $O(\log^2 n)$, использующие описанные в разделе 2.1.6 конструкции k-независимых случайных величин и соответствующий метод дерандомизации. Приведем также результат из [244] о существовании NC-алгоритма построения ε-хорошей раскраски с временем работы O(log n), опирающийся на конструкцию (ε,k)-независимых случайных величин, описанную в разделе 2.1.6. Столь подробное изложение обусловлено тем, что задача о балансировке подмножеств является частным случаем известной задачи аппроксимации в решетке, тесно связанной с задачами целочисленного линейного программирования. Помимо важности для практических приложений, подробное изложение позволяет также продемонстрировать широко применяемую в настоящее время технику использования вероятностных методов и процедур дерандомизации при построении параллельных алгоритмов.

Сначала опишем параллельный NC-алгоритм построения ε-хорошей раскраски с временем работы $O(\log^3 n)$ и его модификацию с временем работы $O(\log^2 n)$. Алгоритм основан на общей идее сжатия вероятностного пространства, изложенной в предыдущих разделах, и использовании покомпонентного способа нахождения решения [39],[221],[10]. Для сжатия вероятностного пространства отказываются от рассмотрения независимых случайных величин и переходят к k-независимым, т.е. величинам, любые k из которых независимы.

Покажем, что k-независимости достаточно для построения ε-хорошего булева вектора. Пусть $x_1,\ldots,x_n$ – k-независимые слу-

чайные величины, принимающие значения 0 и 1 с вероятностью 1/2. Возьмем в качестве искомого вектора $Y = (y_1,\dots,y_n)$, где $y_i = (-1)^{x_i}$, и рассмотрим случайную величину

$$Y_i = \sum_{j \in S_i} y_j .$$

Подсчитаем k-е моменты величин Y_i (полагая k четным):

$$E(Y_i^k) = E((\sum_{j \in S_i} y_j)^k)$$

Для сокращения записи не будем указывать при суммировании принадлежность j к S_i и опустим индекс i у Y_i.
В силу k-независимости y_i имеем:

$$E(Y^k) = E((\sum_j y_j)^k) = \sum_{j_1} \sum_{j_2} \dots \sum_{j_k} E(y_{j_1} y_{j_2} \dots y_{j_k}) =$$

$$= \sum_{j_1, j_2, \dots, j_k} E(y_{i_1}^{s_1}) \dots E(y_{i_t}^{s_t})$$

где $i_1,\dots,i_t$ – попарно различные компоненты вектора $j_1,\dots,j_k$, s_i – кратность вхождения элемента i_1 в набор $j_1, j_2, \dots, j_k$.

Заметим, что если хотя бы одно $s_i = 1$, то соответствующее $E(y_i^{s_i}) = 0$. Поэтому имеем

$$E(Y^k) \le \sum_{t \le k/2} \sum_{i_1, i_2, \dots, i_t} \sum_{\substack{s_1, s_2, \dots, s_t: \\ \sum s_i = k,\ s_i \ge 2}} \frac{k!}{s_1! \dots s_t!} \le$$

$$\le \sum_{t \le k/2} p(k,t)\ k! \sum_{i_1, \dots, i_t} 1 \le \sum_{t \le k/2} p(k,t)\ k!\ d^t \le$$

$$\le k!\ d^{k/2} \sum_{t \le k/2} p(k,t),$$

где $p(k,t)$ – число решений уравнения: $\sum s_i = k$, $s \geq 2$ – натуральные. Очевидно, что $p(k,t) \leq C_{k-1}^{t-1}$, $\sum_{t \leq k/2} C_{k-1}^{t-1} \leq 2^k$, поэтому $E(Y^h) \leq k!\, d^{k/2}\, 2^k \leq k^k\, d^{k/2}$ и можно подобрать $k = O(\frac{\log n}{\varepsilon \log d})$, чтобы выполнялось неравенство

$$(E(Y^h))^{1/k} \leq d^{1/2} + \varepsilon (\log n)^{1/2},$$

то есть существовала ε-хорошая раскраска.

Чтобы построить параллельный детерминированный алгоритм нахождения такого вектора X, применим изложенную выше идею компактного задания вероятностного пространства и покомпонентный метод нахождения случайного вектора. Предположим, что k четно и положим $h = \lceil \log(m+1) \rceil\, k/2$. Возьмем в качестве a_i i-й столбец проверочной матрицы кода БЧХ размера $h \times m$ с кодовым расстоянием $k+2$. Любые k векторов из $a_1, \ldots, a_m$ линейно независимы над полем GF(2), а соответствующие им случайные векторы $x_i = a_i \omega$ являются k-независимыми.

Для построения параллельного NC-алгоритма используем одну из разновидностей метода условных математических ожиданий. Напомним, что общий метод ([10], [299], [270]) заключается в покоординатном поиске "хорошего" вектора, удовлетворяющего условию

$$f(x) \leq F(f,K), \qquad (2.5),$$

где $f: K \longrightarrow R$, F– функционал, обладающий свойством квазивыпуклости: для всякого разбиения $K = K_1 \cup K_2$ выполнено:

$$F(f,K) \geq \min \{ F(f, K_1), F(f, K_2) \} \qquad (2.6)$$

Метод заключается в покомпонентном определении "хорошего" вектора $X = (x_1, \ldots, x_m)$, удовлетворяющего (2.6). Пусть уже определены первые i компонент $x_1 = a_1, \ldots, x_i = a_i$. Вычисляем значение функционала F при фиксированных i компонентах $x_1 = a_1, \ldots, x_i = a_i$ и при всех возможных значениях (i+1)-й компоненты и выбираем из этих значений минимальное, полагая $x_i = a_i$, на котором достигается максимум. В силу (2.6) для полученного таким способом вектора будет выполнено (2.5).

В [59], [240] для задачи о балансировке множеств и поиска ε-хорошего булева вектора в качестве функционала F рассматривалось математическое ожидание при равномерно распределенном x,

а в качестве множества K_n – единичный n-мерный куб. Таким способом в [59], [240] строится параллельный алгоритм нахождения ε-хорошего вектора с временем работы $O(\log^3 n)$ на PRAM с полиномнальным числом процессоров.

Покажем теперь, как вычислять условные средние $E(Y|\omega_1 = s_1,\ldots,\omega_t = s_t)$, необходимые для построения детерминированного параллельного NC-алгоритма нахождения ε-хорошего булева вектора X. Имеем:

$$E(Y|\omega_1=s_1,\ldots,\omega_t=s_t) = \sum_{j_1}\sum_{j_2}\ldots\sum_{j_k} E(y_{j_1} y_{j_2}\ldots y_{j_k})|$$

$$\omega_1=s_1,\ldots,\omega_t=s_t) \quad , \text{ где } Y = \sum_{i=1}^{n} Y_i^k$$

Учитывая, что $y_i = (-1)^{x_i}$, внутреннюю сумму можно преобразовать так: $S_{j_1,j_2,\ldots,j_k} =$

$$= E(y_{j_1} y_{j_2}\ldots y_{j_k}|\omega_1 = s_1,\ldots,\omega_t = s_t) =$$

$$E((-1)^{\sum_{i=1}^{k} x_{j_i}} \quad \omega_1=s_1,\ldots,w_t=s_t) =$$

$$= E\,((-1)^{\sum_{i=1}^{k} a_{j_i}\omega} |\omega_1=s_1,\ldots,\omega_t=s_t) = E((-1)^{a\omega} \mid \omega_1=s_1,\ldots,\omega_t=s_t),$$

где $a = \sum_{i=1}^{k} a_{j_i}$. Пусть r – номер последнего единичного бита в векторе a. Если $t < r$, то $S_{j_1,\ldots,j_k}(s) = 0$. В противном случае $a\omega$ одно и то же для всех ω, совпадающих с s в первых t позициях и $S_{j_1,\ldots,j_k}(s) = E((-1)^{a\omega})$. В предположении, что a и r вычислены заранее, $S_{j_1,\ldots,j_k}(s)$ можно вычислить за конс-

тантное время, расширяя частичную сумму $\sum_{j=1}^{t} a_j s_j$ на каждой итерации и полагая $S_{j_1,\ldots,j_k}(s) = 0$, если $t < r$ и

$$S_{j_1,\ldots,j_k}(s) = (-1)^{\sum_{j=1}^{t} a_j s_j}$$

при $t \geq r$. Для параллельного вычисления условных математических ожиданий $E(Y \mid \omega_1 = s_1,\ldots,\omega_t = s_t)$ по изложенной схеме необходимо md^k процессоров. Чтобы число процессоров было полиномиально, должно выполняться соотношение $k = O(\log n/\log d)$. При минимально возможном $k = \lceil \log n/\varepsilon \log d \rceil$ получаем, что достаточно $n^{1/\varepsilon+3}$ процессоров.

При этом можно параллельно вычислить за константное время величины $S_{j_1,\ldots,j_k}(s)$ и затем просуммировать их по методу сдва-ивания за время $O(\log n)$. Поскольку требуется определить l битов, последовательно применяя эту процедуру l раз, найдем требуемый ε-хороший вектор C за время $O(l \log n) = O(\log^3 n)$. Почти аналогичная схема доказательства этого результата была одновременно анонсирована в [59] и [240].

Покажем, что время работы параллельного алгоритма построения ε-хорошего булева вектора C с такими же гарантированными оценками величины $\Delta(S,C)$ можно уменьшить до $O(\log^2 n)$ за счет увеличения числа процессоров всего лишь в n раз. Для этого перейдем от побитового вычисления вектора ω к вычислению блоков координат ω длины $r = \lceil \log n \rceil$. При этом проходит тот же анализ вычисления величин $S_{j_1,\ldots,j_k}(s)$, что был изложен выше для побитового вычисления. Но при этом вместо двух возможных продолжений ω придется рассмотреть $2^{\lceil \log n \rceil}$ продолжений, для каждого такого продолжения вычислить величины $E(Y \mid \omega_1 = s_1,\ldots,\omega_{tr} = s_{tr})$ и затем, выбрав из них наименьшее, зафиксировать очередной блок координат ω. Эти вычисления проводятся аналогично изложенным выше для побитовых вычислений и выполняются параллельно, так как не зависят друг от друга. Такое распараллеливание потребует примерно в n раз больше процессоров, уменьшит время вычисления (число последовательных итераций) в $O(\log n)$ раз и приведет к алгоритму с полиномиальным числом процессоров, который строит ε-хороший булев вектор в зада-

че о балансировке множеств за время $O(\log^2 n)$.

Пусть уже определены значения первых tr компонент $\omega_1 = s_1, \ldots, \omega_{tr} = s_{tr}$.

Подсчитаем значения 2^r условных математических ожиданий f при условиях: $\omega_1 = s_1, \ldots, \omega_{tr} = s_{tr}$, $\omega_{tr+1} = \alpha_1, \ldots, \omega_{(t+1)r} = \alpha_r$, где $\alpha_1, \ldots, \alpha_r$ принимают всевозможные 2^r значений, $\alpha_i \in (0,1)$. Затем выбирается вектор $(\alpha_1, \ldots, \alpha_r)$, максимизирующий величину

$$M\,(\,f(x) \mid \omega_1 = s_1, \ldots, \omega_{tr} = s_{tr},\ \omega_{tr+1} = \alpha_1, \ldots, \omega_{(t+1)r} = \alpha_r)$$

и положим $\omega_{tr+1} = \alpha_1, \ldots, \omega_{(t+1)r} = \alpha_r$.

Лемма. После t итераций описанной процедуры

$$M\,[f(x) \mid \omega_1 = s_1, \ldots, \omega_{(t+1)r} = \alpha_r] \geq M\,[f(x)].$$

В справедливости леммы нетрудно убедиться, используя индукцию.

Таким образом, на выходе нашей процедуры получается вектор ω^*, для которого $f(\omega^*) \geq M\,[f(\omega)]$ (поскольку математическое ожидание для неслучайной величины равно этой величине).

Параллельный алгоритм вычисляет все $S_{j_1, \ldots, j_k}(s)$ параллельно за константное время, и для всех 2^r значений $\omega_{tr+1}, \ldots, \omega_{(t+1)r}$ просуммируем соответствующие члены за время $O(\log n)$. Среди вычисленных сумм найдем максимум по всем 2^r значениям $\omega_{tr+1}, \ldots, \omega_{(t+1)r}$ за время $O(t)$. Общее параллельное время на d итерациях есть $O(d\,(t + \log n))$. Полагая $r = \lceil \log n \rceil$, число итераций оценивается величиной $d = \lceil h/r \rceil = O(\log^2 n / \log n) = O(\log n)$. При таких параметрах общее время работы параллельного алгоритма есть $O(\log^2 n)$ и число процессоров полиномиально.

Таким образом, уменьшение в $O(\log n)$ раз сложности параллельного алгоритма оказалось возможным благодаря переходу от покоординатного двоичного поиска к недвоичному поиску, в котором на каждой итерации определяется сразу целый блок двоичных координат (длины r) по методу условных математических ожиданий.

Опишем теперь идею NC-алгоритма построения ε-хорошей раскраски с временем работы $O(\log n)$ [244]. Ключевым моментом является использование вместо k-независимых случайных величин (δ, k)-независимых величин при подходящем выборе параметра δ. При

этом важную роль играет следующий аналог неравенства больших отклонений для сумм (δ,k)-независимых случайных величин с нулевым математическим ожиданием:

$$M[Y^k] \le B_k + \delta\, m^k, \text{ где } B_k = M[Z^k], \quad (2.7)$$

$Z = \sum_{j=1}^{m} Z_j$ Z_j – k-независимые случайные величины, принимающие значения 0 и 1 с вероятностью 1/2,

$Y = \sum_{j=1}^{m} Y_j$ Y_j – (δ,k)-независимые случайные величины, принимающие значения 0 и 1 с вероятностью 1/2.

Для задачи о балансировке множеств для случайной (δ,k)-независимой раскраски имеем с учетом (2.7):

$$M[Y^k] \le k^k\, d^{k/2} + \delta\, d^k.$$

В разделе 2.1.6 и в [244], [40] отмечалось, что конструкция, основанная на использовании кодов, двойственных к кодам БЧХ, позволяет построить n (δ,k)-независимых случайных величин из вероятностного пространства, логарифм размера которого есть $O(\log k + \log\log n + \log 1/\delta)$. Полагая $k = O(\log n/\log d)$, $\delta = d^{-k/2}$ Тогда число случайных битов есть

$$O(\log k + \log\log n + (k/2)\log d) = O(\log n) \text{ и}$$

$$M[Y^k] \le (k^k + 1)\, d^{k/2},$$

откуда вытекает существование ε-хорошей раскраски при выборе (δ,k)-независимой раскраски при указанных параметрах δ и k. А построить искомый булев вектор можно полным перебором по вероятностному пространству, поскольку его размер полиномиален по n.

Задача аппроксимации на решетке

Задача аппроксимации на решетке является обобщением рассмотренной выше задачи о балансировке множеств и неформально заключается в следующем. Имеется некоторая точка p в n-мерном пространстве, требуется найти точку q с целочисленными координатами, "хорошо аппроксимирующую" p в смысле величины скалярного произведения вектора p–q и столбцов некоторой матрицы A. Более точная формулировка заключается в следующем:

Вход: Матрица A размера nxm с элементами $a_{ij} \in [0,1]$; вектор $p \in [0,1]^m$ и $c \in R^n$ такой, что $Ap = c$.

Выход: m-битовый вектор $q \in [0,1]^m$ минимизирующий величину $||\Delta||_\infty$, где $\Delta = |A(p-q)| = |c-Aq|$ и $||x||_\infty$ равна максимальной компоненте вектора x.

Задача о балансировке множеств является частным случаем этой задачи при $p_i = 1/2$, $p = (p_1,\ldots,p_m)$.

Рагхаван [270] предложил последовательный детерминированный алгоритм, основанный на описанном в предыдущем разделе методе условных вероятностей, который строит вектор q такой, что i-я компонента вектора Δ есть $O(\sqrt{c_i \log n})$.

В работе [59] описан детерминированный параллельный NC-алгоритм построения вектора q, для которого i-я компонента вектора Δ есть $O(||a_i||_1^{1/2+\varepsilon}\sqrt{\log n})$, где $a_i = \max_j a_{ij}$.

В [240] представлен следующий результат: существует NC-алгоритм построения вектора q, для которого i-я компонента вектора Δ есть $O(c_i^{1/2+\varepsilon} + \log^{2/\varepsilon} n)$ при всех $i = 1, 2,\ldots,n$ и $0 < \varepsilon < 1/2$.

Общая идея получения этих результатов заключается в рассмотрении L-битового представления чисел p_i, где $L = O(\log n)$, отбрасывании остальных битов в силу их малого влияния на конечный результат, а затем итерационном подсчете вектора q. Число таких итераций равно L и на каждой итерации решается задача типа уже рассмотренной нами задачи о балансировке множеств. При этом производятся манипуляции только со случайными величинами, принимающими значения 0 и 1 с вероятностью 1/2. Такой подход мотивирован трудностями прямого манипулирования со случайными величинами, принимающими значения 1 и 0 с вероятностями p_i и $1-p_i$.

Уточним вышесказанное. Пусть каждое $p_j \in [0, 1]$ и представлено битами $p_j^{(0)}, p_j^{(1)},\ldots,p_j^{(L)}$ своего двоичного представления. На каждой итерации специальным образом округляется последний бит каждого p_j так что после L-й итерации от каждого p_j

останется только один бит q_j, которые в совокупности и являются искомым ответом.

Процесс округления происходит следующим образом. Если последний бит равен нулю, то он просто отбрасывается. Если же последний бит рассматриваемого p_j равен единице, то с вероятностью 1/2 он отбрасывается (округление вниз), и также с вероятностью 1/2 он отбрасывается с прибавлением к оставшемуся k-битовому числу числа 2^{-k} (округление вверх).

Каждая такая итерация может рассматриваться как случайное решение взвешенной задачи о балансировке множеств. Дерандомизация, то есть построение детерминированного NC-алгоритма методом подсчета условных математических ожиданий осуществляется аналогично задаче о балансировке множеств. Оценка точности такого итерационного решения осуществляется с использованием вероятностной техники, также сходной с той, которая использовалась в задаче о балансировке.

Приближенные параллельные алгоритмы для задачи целочисленного линейного программирования.

Вход. Матрица $A = (a_{ij})$, вектор $c = (c_1,\ldots,c_n)$ и вектор $b = (b_1,\ldots,b_m)$ с неотрицательными целыми компонентами.

Выход. Булев вектор $x = (x_1,\ldots,x_n)$, минимизирующий значение целевой функции $\sum_{i=1}^{n} c_i x_i$ при ограничениях

$$\sum_{i=1}^{n} a_{ij} x_i \geq b_j , \quad j = 1,\ldots,m$$

В такой постановке задача является NP-полной. Однако имеется ряд результатов о возможности эффективного нахождения ее ε-приближенных решений, которые интересно было бы распараллелить. В [11] показано, что по решению соответствующей задачи линейного программирования (без ограничения целочисленности переменных) для широкого класса задач можно эффективно построить ε-оптимальное целочисленное решение. При этом процесс построения целочисленного решения по рациональному можно распараллелить аналогично параллельному решению задачи приближения в решетке (см. выше и [59], [240]) и , кроме того, для этого оказывается достаточно знания δ-приближенного решения соответствующей задачи линейного программирования. Поэтому основная трудность построения

параллельного приближенного алгоритма длязадачи ЦЛП заключается в построении параллельного алгоритмарешения задачи линейного программирования с заданной точностью $\delta > 0$. В общем случае, эта задача является P-полной [290], однако в интересующем нас классе задач все исходные данные неотрицательны и выполнено ограничение снизу на величину правых частей вида $b_j / a_j \log m \longrightarrow \infty$.

Вопрос о параллельной сложности такого класса задач линейного программирования остается открытым. При этом с точки зрения ЦЛП достаточно построить параллельный алгоритм нахождения δ-приближенного решения задачи ЛП с неотрицательными коэффициентами, величина которых ограничена полиномом от размерности задачи.

Другим примером эффективных приближенных алгоритмов решения задачи ЦЛП являются жадные алгоритмы, гарантирующие нахождение допустимого целочисленного решения, отличающегося от оптимума (по величине целевой функции) не более, чем в $1 + \ln B$ раз, где $B = \sum_{i=1}^{n} b_i$ [12], и от решения соответствующей задачи ЛП – не более, чем в $a(1 + \ln B)$ раз, где $a = \max_{i,j} a_{ij}$. Распараллеливание этих алгоритмов представляет значительный интерес. В [59] предложен NC-алгоритм решения этой задачи для случая $b_j = 1$, $a_{ij} \in (0,1)$, $c_i = O(2^{\log^c n})$.

Упаковка в контейнеры.

Вход: Предметы объема $a_1,\ldots,a_n$, $0 < a_i < 1$, $i = 1,\ldots,n$.

Выход: Минимальное число контейнеров единичного объема, в которые можно упаковать эти предметы.

В такой постановке задача является NP-полной. Однако для любого фиксированного $\varepsilon > 0$ известен линейный по n алгоритм нахождения ε-оптимального решения [117]. Этот алгоритм можно путем распараллеливания перевести в NC алгоритм.

Напомним, что ε-оптимальный алгоритм упаковки в контейнеры заключается в огрублении объемов предметов путем дискретизации так, чтобы получилось фиксированное число возможных объемов и минимальный объем предмета был ограничен снизу некоторой константой. Для таких огрубленных данных решается задача оптимальной упаковки и показывается, что полученное решение является хорошей аппроксимацией в исходной задаче. Уточним сказанное.

Выделим в отдельный массив K_0 все числа из массива

$a_1,\ldots,a_n$, меньшие $\varepsilon/2$, а оставшуюся часть массива параллельно отсортируем и выберем $m = O(1/\varepsilon)$. Пусть $n = m\,h + r$. Разбиваем полученный отсортированный массив на m равных частей (кроме последней части) в порядке возрастания чисел. Получим упорядоченный массив вида: $L = K_0\, y_1\, K_1\, y_2 \ldots K_{m-1}\, y_m\, K_m\, R$, где $|K_i| = h - 1$ $(i > 0)$ – упорядоченные блоки длины $h - 1$, K_0 состоит из элементов массива, меньших $\varepsilon/2$, R – остаток массива, состоящий из r чисел.

Задача упаковки в контейнеры предметов из списка L аппроксимируется сверху и снизу аналогичной задачей для списков L_1 и L_2 следующего вида: $L_1 = K_0\, y_1^h\, y_2^h \ldots y_m^h\, R$ и $L_2 = K_0\, y_2^h \ldots y_m^h\, 1^h\, R$, где запись a^h означает h-кратное повторение символа a. Имеем: $V(L_1) \le V(L) \le V(L_2)$, где $V(L)$ обозначает минимальное число контейнеров, достаточное для упаковки предметов из списка L. Нетрудно показать, что

$$V(L_1) \le (1 + \varepsilon)\, V(L_2) + \text{const}.$$

Таким образом, задача свелась к решению аналогичной задачи на мультимножестве с фиксированным числом $m = O(1/\varepsilon)$ предметов различного размера, причем размер каждого предмета ограничен снизу константой. Пусть $M = (x_1, n_1, \ldots, x_m, n_m)$ – такое мультимножество, где x_i – размер i-го предмета, а n_i – его кратность, причем все $x_i \ge \varepsilon$. Число способов заполнения контейнера предметами m типов не превосходит некоторой константы (зависящей от ε экспоненциально). Пусть a_{ij} – число предметов i-го типа в j-м заполнении. Сопоставим j-му заполнению переменную x_j и присвоим ей значение, равное числу выборов j-го заполнения при данной упаковке предметов в контейнеры. Тогда получим следующую задачу линейного целочисленного программирования:

$$\sum_{j=1}^{c} x_j \longrightarrow \min$$

$$\sum_{j=1}^{c} a_{ij} x_j \ge b_i\,, \qquad i = 1,\ldots,m$$

Поскольку все a_{ij} ограничены сверху некоторой константой (зависящей от ε), эта задача решается последовательно за константное время. Для ее решения с константной аддитивной точностью можно пользоваться решением соответствующей задачи линейного программирования (ЛП), полученной из исходной отбрасы-

ванием требования целочисленности переменных, поскольку найдется решение задачи ЛП, в котором не более m отличных от нуля компонент. Округляя эти компоненты вверх до ближайшего целого, получим допустимый целочисленный вектор, отличающийся от оптимума (по величине целевой функции) не более чем на m (при этом нижняя оценка целочисленного оптимума при фиксированном m имеет вид $\Omega(n)$). Поэтому достаточно решить соответствующую задачу ЛП одним из известных методов [26],[287]. Таким образом, первый этап алгоритма распараллеливается применением известных параллельных алгоритмов сортировки, а второй этап, даже при последовательном выполнении, добавляет лишь аддитивную константу (зависящую ε). Таким образом, при любом фиксированном ε задача решается за время $O(\log n)$ на $O(n)$ процессорах EREW PRAM. Число процессоров можно уменьшить, используя вместо сортировки параллельные алгоритмы выбора k-го по порядку элемента массива. Для получения полиномиальных по $1/\varepsilon$ оценок можно использовать алгоритм, предложенный в [185].

Проверка вхождения подстроки в строку.

Вход. Строка длины n и строка длины m.

Выход. Определить все вхождения второй строки в первую в качестве подстроки.

В [76] предложен оптимальный параллельный алгоритм решения этой задачи за время $O(\log \log m)$ на CRCW PRAM с $O(n/\log \log m)$ процессорами.

Построение оптимального префиксного кода.

Вход. Вероятности появления букв $p_1,\ldots,p_n$.

Выход. Оптимальный префиксный код, минимизирующий математическое ожидание длины кодового слова.

Напомним, что рассматриваются только побуквенное кодирование и код называется префиксным, если никакое кодовое слово не является началом другого кодового слова. Последовательный алгоритм решения этой задачи известен как метод Хаффмэна. В [306] предложен параллельный алгоритм построения оптимального префиксного кода за время $O(\log^2 n)$ на PRAM с полиномиальным числом процессоров. Алгоритм основан на равномерном сведении исходной задачи к вычислению значения схемы линейной степени над коммутативным полукольцом и применении результата о распараллеливании таких схем [233].

2.2.5. Теория графов

Параллельные алгоритмы решения задач теории графов имеют важное значение, поскольку часто используются для работы со специальными структурами данных [230]. В качестве методов построения параллельных алгоритмов на графах используются описанные в предыдущих разделах метод сдваивания, матричные методы, поиск в глубину и поиск в ширину. Приведем сначала примеры применения более простых матричных методов.

Транзитивное замыкание.

Вход: Ориентированный граф, заданный матрицей смежности.

Выход: Граф, являющийся транзитивным замыканием исходного графа.

Напомним, что транзитивным замыканием графа G называется граф T(G), в котором ребро между вершинами i и j имеется тогда и только тогда, когда в G имеется путь между вершинами i и j.

Опишем простой параллельный алгоритм, основанный на матричных методах. Алгоритм построения матрицы смежности A(T) транзитивного замыкания графа основан на следующем представлении:

$$A(T) = A \vee A^2 \vee A^3 \quad \vee A^n,$$

которое можно преобразовать в эквивалентную форму:

$$A(T) = (A \vee A^2)^t, \qquad (2.8)$$

где A – матрица смежности исходного графа, n – число его вершин, $t = 2^{\lceil \log n \rceil}$, дизъюнкция берется поэлементно, а в качестве умножения рассматривается булево произведение матриц, отличающееся от обычного лишь тем, что сложение заменяется на дизъюнкцию, а умножение – на конъюнкцию. Справедливость формулы следует из того, что между вершинами графа i и k имеется путь длины не более 2l тогда и только тогда, когда найдутся вершина j и пути длины не более l между вершинами i и j, j и k соответственно.

Аналогично обычному произведению матриц параллельное булево умножение двух матриц размером nxn можно осуществить за время O(log n) на EREW PRAM с $O(n^{\omega}/\log n)$ процессорами. Вычисления по формуле (2.8) производятся методом сдваивания. Общее время работы

алгоритма есть величина $O(\log^2 n)$ на EREW PRAM с $O(n^\omega/\log^2 n)$ процессорами. Для модели COMMON CRCW PRAM с $O(n^\omega/\log n)$ процессорами время работы практически этого же алгоритма есть $O(\log n)$, поскольку булево произведение матриц реализуется за константное время .

Кратчайшие пути.

Вход: Граф, ребрам которого приписаны веса.

Выход: Кратчайшие расстояния между всеми парами вершин.

Будем предполагать, что в графе нет циклов отрицательного веса. Простой параллельный алгоритм нахождения матрицы $A(K)$ кратчайших расстояний между всеми парами вершин графа основан на использовании матричных методов и следующем представлении:

$$A(K) = \min_{1 \le i \le \log n} (\min (A, A^2))^{2^i},$$

где A – матрица смежности исходного графа, минимум берется поэлементно, а в качестве умножения рассматривается произведение матриц, отличающееся от обычного лишь тем, что сложение заменяется на минимум, а умножение – на сложение. Аналогично обычному произведению матриц такое параллельное умножение двух матриц размера $n \times n$ можно осуществить за время $O(\log n)$ на EREW PRAM с $O(n^3/\log n)$ процессорами. Общее время работы алгоритма есть величина $O(\log^2 n)$ на EREW PRAM с $O(n^3/\log n)$ процессорами. Для модели STRONG CRCW PRAM с $O(n^3)$ процессорами время работы есть $O(\log n)$, поскольку минимум n чисел может быть найден за константное время на $O(n)$ процессорах. На $O(n^{3+\varepsilon})$ процессорах ($\varepsilon > 0$ – произвольная константа) WEAK CRCW PRAM время работы алгоритма также есть величина $O(\log n)$, поскольку минимум n чисел может быть найден за константное время на $O(n^{1+\varepsilon})$ процессорах WEAK CRCW PRAM (см. раздел 2.1.4.).

В [281] показано, как быстрые алгоритмы умножения матриц над произвольным кольцом можно применять для нахождения кратчайших путей в графах с целочисленными весами ребер за время $O(n^\omega)$. Распараллеливание этих быстрых алгоритмов может приводить к уменьшению числа процессоров с $O(n^3)$ до $O(n^\omega)$.

Ранжирование списка.

Вход: Массив из n элементов, образующих список.

Выход: Определить для каждого элемента массива расстояние до конца списка.

Отметим, что список задан, если для каждого его элемента указан номер его потомка. Эта задача часто возникает в виде процедуры при решении многих задач. Построение параллельного алгоритма на n процессорах с временем работы O(log n) не представляет трудностей. Алгоритм строится на основе изложенного в разделе 2.1.1 метода сдваивания цепей и запоминания всех расстояний. Известны оптимальные алгоритмы с временем работы O(log n) на EREW PRAM с O(n/log n) процессорами [44],[92]. Однако, они используют ряд нетривиальных соображений и являются весьма сложными и громоздкими. Приведем поэтому описание идеи более простого оптимального алгоритма для CRCW PRAM.

Алгоритм основан на использовании рекурсии. Строится подмножество S элементов списка, состоящее из не более чем cn вершин ($c < 1$) такое, что расстояние от каждого элемента списка до множества S "небольшое". Строится стянутый список, состоящий из элементов S с весами, равными их номерам плюс число элементов до последующего элемента. Рекурсивно решается задача для стянутого списка и суффиксные суммы весов элементов стянутого списка и являются суффиксными суммами весов исходного списка. Затем решение распространяется на все элементы исходного списка (с использованием свойства "близости") за время, пропорциональное максимальному расстоянию в исходном списке между соседними элементами S. Техника, которая применяется для построения такого множества S, получила название "детерминированное бросание монеты" [91]. Она основана на понятии r-правильного множества.

Подмножество S *списка называется* r-*правильным, если в* S *не содержатся соседние элементы списка и каждый элемент списка находится на расстоянии не более* cr *от некоторого элемента из* S.

Следующая процедура позволяет найти r-правильное множество n-элементного списка за время $O(k)$ с использованием n процессоров [91], где $r = \log^{(k)} n$ – k-я итерация логарифма. Будем полагать, что в списке имеются ссылки как на предшественников p(i), так и на последователей s(i).

Для всех i **выполнить** c(i):= 1;
Выполнить k **итераций**
для всех i **выполнить параллельно**
найти самый правый разряд q, **в котором** c(i) **отличается от** c(s(i)); **в** c(i) **заслать булев вектор, первый разряд которого совпадает с** q-**м**

разрядом c(i), а остальные - с двоичной записью числа q;

для всех i **выполнить параллельно**

если c(p(i)) ≤ c(i) **и** c(s(i)) ≤ c(i) **занести** i **в** r-**правильное множество.**

Можно проверить, что на каждой итерации c(i) ≠ c(s(i)) и максимальное значение каждого c(i) после j-й итерации есть $O(\log^{(j)} n)$. Обычно в различных приложениях используют описанную процедуру при конкретном значении k. Эта процедура также используется и в более сложных построениях оптимального алгоритма ранжирования списка на EREW PRAM с O(n/ log n) процессорами, имеющего время работы O(log n) [44], [92].

Эйлеров путь в дереве.

Вход: Корневое дерево, заданное в виде списка потомков каждой вершины.

Выход: Эйлеров путь, то есть такой обход всех ребер, что каждое ребро оказывается пройденным в прямом и обратном направлениях по одному разу.

Известен параллельный алгоритм с временем O(log n) на EREW PRAM с O(n/log n) процессорами [227], [116].

Нумерации вершин дерева.

Вход: Корневое дерево.

Выход: Нумерация вершин двух типов - сверху вниз (pre-order), снизу вверх (post-order).

Нумерации определяются следующим образом:
дерево с корнем R, состоящее из двух поддеревьев: левого - A и правого - B нумеруется в такой последовательности:

при нумерации сверху вниз - R A B;

при нумерации снизу вверх - A B R;

При заданном эйлеровом обходе дерева нумерации сверху вниз и снизу вверх могут быть найдены за время O(log n) на EREW PRAM с O(n/log n) процессорами [227].

Рассматриваются также другие типы нумераций, в частности, нумерации по уровням:

нумерация в ширину, определяемая следующим образом: вершины дерева нумеруются от корня по ярусам слева направо;

нумерация в ширину-глубину, определяемая рекурсивно: для данного дерева T с корнем r, соединенного ребрами с k поддеревьями $T_1,\ldots,T_k$, вершины нумеруются от корня в первом ярусе дерева

T, затем вершины T_k пронумерованные в ширину-глубину (за исключением корня), затем вершины T_{k-1}, пронумерованные в ширину-глубину (за исключением корня), и так далее до T_1.

В [82] предложены параллельные алгоритмы нумерации произвольных n-вершинных корневых деревьев в ширину-глубину и в ширину за время O(log n) на EREW PRAM с O(n/log n) процессорами.

Эйлеров цикл в эйлеровом графе.

Вход: Произвольный граф, удовлетворяющий условию Эйлера.

Выход: Эйлеров цикл в графе (цикл, прохододящий через каждое ребро графа один раз).

Известен алгоритм с временем O(log n) для CRCW PRAM с O((n + m)/log n) процессорами, где n – число вершин, а m – число ребер графа, если граф задан списком ребер в каждой вершине, или O(n + m) процессорами, если граф задан только списком ребер [47]. Эйлеров тур в орграфе можно построить за время O(logn) на CRCW PRAM с O((nlog n + m)/ log n) процессорами [93].

Максимальное число вершинно различных циклов.

Вход: Произвольный граф.

Выход: Максимальное число циклов в графе, непересекающихся по ребрам.

Известен алгоритм с временем O(log m) для CRCW PRAM с O((n + m)) процессорами, где n – число вершин, а m – число ребер графа [200].

Восстановление графа по его реберному графу.

Вход: Произвольный реберный граф G с n вершинами и m ребрами.

Выход: Граф H, реберным графом которого является G.

В [246] предложен параллельный алгоритм восстановления графа по его реберному графу за время O(log n) на CRCW PRAM с O(m) процессорами. Алгоритм основан на методе "разделяй и властвуй" и эффективной процедуре слияния построенных на рекурсивном шаге подграфов.

Стягивание дерева.

Элементарной операцией алгоритма сжатия дерева является операция шунтирования. Применение операции шунтирования к висячей вершине l дерева T приводит к дереву T′, в котором дуги, входившие в l и в предшественник p вершины l, сжаты в точку, а второй потомок вершины p становится потомком предшественника p.

Вход: Бинарное дерево, каждая вершина которого, отличная от

внсячнх, имеет ровно двух потомков.

Выход: Последовательность стягиваний ребер, превращающая граф в точку.

Стягиванне дерева является одним нз способов параллельного вычнсления выражений, заданных в внде дерева. Известно несколько оптнмальных алгоритмов с временем работы O(log n) на EREW PRAM с n/log n процессорамн [138], [201], [92], [187].
Опишем алгорнтм предложенный в [201].

Алгоритм состонт нз трех этапов:

1. С нспользованнем теннки построенння эйлеровых путей в дереве занумеровать лнстья дерева слева направо чнслами от 1 до n.

2. Выполнить ⌈log n⌉ нтерацнй, состоящнх нз следующнх шагов:

a) применить операцию шунтирования параллельно ко всем нечетным листьям, являющнмся левымн потомкамн своих предшественнников.

b) то же самое сделать для правых потомков.

c) разделнть метку каждого остающегося лнста на два без остатка.

Все операцни, используемые в алгорнтме, прямо реализуются на EREW PRAM. После каждой нтерацин половина лнстьев удаляется нз текущего дерева и после ⌈log n⌉ нтерацнй дерево стянется в одну вершнну.

Первый этап требуют временн O(log n). Последовательное время выполнення второго этапа есть велнчнна

$$O\left(\sum_{i=1}^{\lceil \log n \rceil} n/2^i\right) = O(n)$$

и общее параллельное время равно O(log n) на O(n/log n) процессорах.

Компоненты связности.

Вход: Граф.

Выход: Компоненты связностн графа.

Вычнсленне компонент связностн часто встречается в разлнчных задачах и является одной нз базовых задач теорни графов. Последовательное время ее решенння есть O(n + m). В [166] предложен следующнй алгорнтм, позволяющнй найтн связные компоненты параллельно за время O(log n) на CRCW PRAM.

Алгоритм состоит из O(log n) этапов и использует специальное представление данных в виде объединения деревьев. На каждом этапе вершины графа представлены в виде леса ориентированных деревьев, с ребрами ориентированными от каждой вершины к ее предшественнику. Все вершины одного дерева принадлежат одной компоненте связности. На первом этапе лес состоит из отдельных не связанных между собой вершин графа, а на последнем – из деревьев глубины 1, вершины которых и являются компонентами связности исходного графа. При переходе от этапа i к этапу i + 1 деревья, содержащие смежные в графе вершины, соединяются в одно дерево и производится его сжатие. При этом каждая вершина, не являющаяся корнем нового дерева или его потомком, получает в качестве нового предшественника вершину, являющуюся предшественником своего предшественника (сдваивание путей).

Известны различные реализации изложенного выше подхода [49], [187]. Они позволяют получить алгоритмы с временем работы O(log n) на ARBITRARY CRCW PRAM с O(m + n) процессорами. Этот алгоритм также легко адаптировать для нахождения остовного дерева минимального веса во взвешенном графе с сохранением оценок на время и число процессоров. Более тонкая техника, основанная на оптимальном параллельном алгоритме ранжирования списка, позволяет построить параллельный алгоритм с временем работы O(log n) на CRCW PRAM с $O((m + n)\alpha(m,n)/\log n)$ процессорами [91],[93], где α – функция, обратная к функции Аккермана (чрезвычайно медленно растущая с ростом m и n функция). В частности, при $m \geq n \log^* n$ отсюда вытекает оптимальный параллельный алгоритм нахождения компонент связности. Напомним, что $\log^* n$ равен минимальному k, для которого $\log^{(k)} n \leq 2$, где $\log^{(k)} n = \log \log^{(k-1)} n$. Имеется также оптимальный вероятностный алгоритм решения задачи за время O(log n) на ARBITRARY CRCW PRAM с O((n + m)/log n) процессорами [131].

Для планарных графов известен и оптимальный детерминированный параллельный алгоритм нахождения компонент связности [156]. В [175] предложен алгоритм нахождения компонент связности графа за время $O(\log^{3/2} n)$ на CREW PRAM с O(n + m) процессорами.

Отыскание остовного леса минимального веса.

Вход: Граф, ребрам которого приписаны неотрицательные веса.

Выход: Остовный лес с минимальной суммой весов ребер.

Известен NC-алгоритм с временем работы $O(\log^2 n)$ на CREW PRAM

NC-алгоритм с временем работы O(log n) на CRCW PRAM [204]. Алгоритмы основаны на общем параллельном жадном алгоритме поиска базы минимального веса в матроиде (см. раздел 2.2.4.). Напомним, что он основан на параллельном вычислении ранговой функции матроида. В данной задаче множество остовных лесов образует матроид на множестве ребер графа. Нетрудно проверить, что ранг данного множества ребер E (равный числу ребер остовного леса в подграфе G, порожденном ребрами из E) выражается через число вершин n, смежных хотя бы с одним ребром из E, следующим образом: rk(E) = n - c, где c - число компонент связности G. Отсюда и из оценок для параллельных алгоритмов нахождения компонент связности графа и вытекают приведенные выше оценки на время параллельного построения остовного леса минимального веса в графе.

Несложная адаптация к данной задаче описанного выше алгоритма отыскания компонент связности приводит к алгоритму с временем работы O(log n) на ARBITRARY CRCW PRAM с O(m + n) процессорами.

Проверка двудольности.

Вход. Граф.

Вопрос. Является ли граф двудольным?

Воспользуемся следующим критерием двудольности графа: граф является двудольным тогда и только тогда, когда он не содержит циклов нечетной длины. Пусть A - матрица смежности графа. Будем вычислять все степени матрицы A и проверять наличие единичных диагональных элементов в нечетных степенях матрицы A. Все степени матрицы можно вычислить по методу сдваивания за время $O(\log^2 n)$ на EREW PRAM и за время O(log n) на CRCW PRAM с соответствующим числом процессоров, необходимых для параллельного умножения матриц. Отметим, что как показано в [237], задача нахождения лексикографически первого максимального по включению двудольного подграфа является P-полной.

k-*связность.*

Вход: Граф.

Вопрос: Является ли граф вершинно k-связным ?

Для фиксированного k известен алгоритм решения задачи с временем работы $O(k^2 \log n)$ на CRCW PRAM с $(n + k^2)$ k C(n,m) процессорами, где C(n,m) - число процессоров, необходимых для нахождения компонент связности графов с n вершинами и m ребрами за логарифмическое время [196], [197]. Для случая k = 3 в [234].

[272] построен параллельный алгоритм с временем O(log n) на CREW PRAM с O((n + m)log^{2}n) процессорами. Для k = 2 известен алгоритм с временем работы O(log n) на CRCW PRAM с O((nlog n + m)/ log n)процессорами [93].

По определению, граф является вершинно k-связным, если удаление любых k-1 вершин оставляет его связным, но найдутся k вершин, удаление которых сделает его несвязным. Известно, что граф является вершинно k-связным, если между любой парой вершин найдется k непересекающихся по вершинам путей. Известно, что достаточно проверить не все пары вершин, а только kn: любые k вершин с остальными. Параллельный алгоритм решения задачи с временем работы O(k^2 log n) на CRCW PRAM с (n + k^2) k C(n,m) процессорами проверяет наличие k непересекающихся путей между kn парами вершин паралельно. Таким образом, основным шагом является проверка наличия k непересекающихся путей между данной парой вершин.

Пусть G = (V,E) – неориентированный граф с двумя выделенными вершинами s и t, не соединенными ребром. Алгоритм проверки наличия k непересекающихся путей между данной парой вершин s и t состоит из пяти этапов:

Этап 1. Если k = 1, найти путь из s в t. Для k > 1 рекурсивно найти k – 1 непересекающихся путей из s в t, если они существуют, или разделяющее множество вершин размера k – 2.

Пусть k – 1 путей $P_1,\ldots,P_{k-1}$ построены. Для подграфа H графа G и ребер f и e в G \ H определим отношение эквивалентности между ребрами: f $=_H$ e тогда и только тогда, когда существует путь в G, проходящий через f и e, но не имеющий общих вершин с вершинами из H. Подграфы, индуцированные ребрами классов эквивалентности E(G) – E(H) относительно отношения эквивалентности $=_H$, называются мостами G относительно H. Соприкасающимися вершинами (с–вершинами) моста B относительно H называются вершины в V(B) ∩ V(H).

Этап 2. Разбить граф на мосты относительно подграфа $P_1 \cup \ldots \cup P_{k-1}$. Определим линейный порядок на с–вершинах, лежащих на пути P_i следующим образом. Для двух таких вершин a и b на P_i скажем, что a < b, если a левее b на пути P_i. При этом будем полагать, что s – левый конец, а t – правый конец пути P_i.

Для моста B_i и пути P_j пусть l_{ij} – самая левая с–вершина из B_i на P_j, а r_{ij} – самая правая.

Этап 3. Строим орграф $G_B = (V_B, E_B)$, называемый графом мос-

тов, следующим образом:

$V_B = (B_i | B_i$ – мост$) \cup (s,t)$.

E_B есть объединение следующих множеств:

1. множество ребер, выходящих из s; множество ребер, входящих в t;

2. ребра между вершинами, соответствующими мостам. Эти ребра разбиты на множества $D_1,\ldots,D_{k-1}$. Ребро (b_i,b_x) добавляется к D_j если r_{jx} (самая правая вершина на P_j) находится справа от r_{ji} среди с-вершин мостов, чьи самые левые с-вершины на некотором P_y находятся левее самой правой с-вершины B_i на P_y.

Отметим, что степень выхода для каждой вершины графа G_B не превосходит k – 1.

Этап 4. Находим ориентированный путь из s в t в графе G_B, если он существует.

Этап 5. Если такой путь построен, то строим k вершинно непересекающихся путей между s и t в исходном графе. Предположим, что не существует ориентированного пути из s в t в графе G_B. Рассмотрим все вершины, достижимые из s в G_B. Каждая такая вершина соответствует некоторому мосту B_i в G. Пусть w_j – с-вершина, наиболее удаленная от s на P_j. Если ни один из этих мостов не имеет с-вершины на P_j, то w_j по определению является соседом s на P_j и множество $(w_1,\ldots,w_{k-1})$ отделяет s от t.

Корректность алгоритма вытекает из следующей теоремы:

В графе G *между вершинами* s *и* t *имеется* k *вершинно непересекающихся путей тогда и только тогда, когда в графе* G_B *имеется ориентированный путь из* s *в* t.

Оценим теперь трудоемкость каждого этапа. Первый этап является рекурсивным. Простой путь может быть найден за время O(log n) на CRCW PRAM с C(n,m) процессорами путем построения остовного дерева. На втором этапе мосты получаются путем построения остовного леса в подграфе, индуцированном вершинами $V(G) - V(\cup P_i)$. Третий этап можно выполнить за время O(log n) на n/log n процессорах с использованием параллельного алгоритма вычисления префиксов с операцией максимум. Четвертый этап наиболее сложный. В произвольных орграфах неизвестны оптимальные алгоритмы построения ориентированного пути с временем O(log n) на C(n,m) процессорах CRCW PRAM. Однако, в [197] утверждается, что, используя специальную структуру графа мостов G_B, это удается сделать.

Пятый этап выполняется с использованием параллельных алгоритмов

вычисления префиксов и ранжирования списков.

Аналогичный результат справедлив и для реберной k-связности.

Максимальное (по включению) независимое множество

Вход: Граф.

Выход: Максимальное (по включению) множество попарно несмежных вершин.

Первый NC-алгоритм для этой задачи был предложен Карпом и Вигдерсоном [186]. Известны алгоритм решения этой задачи с временем работы $O(\log^2 n)$ на CREW PRAM с $O(n^2 m)$ процессорами [221], NC-алгоритм с линейным числом процессоров [220] и алгоритм с временем работы $O(\log^4 n)$ на EREW PRAM с $O(n)$ процессорами [142], где n – число вершин, а m – число ребер графа. Первые три алгоритма основаны на процедуре дерандомизации путем использования описанных в разделе 2.1.6. метода свертки вероятностного пространства и метода условных вероятностей. Рандомизированный алгоритм, к которому применяется этот метод, использует лишь попарно независимые случайные величины для получения нужного решения. Он заключается в следующем.

На каждом следующем этапе, пока это возможно, к уже построенному независимому множеству добавляется некоторое независимое множество в подграфе, порожденном множеством вершин, не соединенных ребром хотя бы с одной вершиной независимого множества. Таким образом, когда не останется таких вершин получим максимальное по включению независимое множество. Основная трудность заключается в том, чтобы выбрать на каждом этапе достаточно большое независимое множество и обеспечить завершение работы алгоритма за полилогарифмическое время. Построение такого независимого множества на каждом этапе можно осуществить следующим вероятностным алгоритмом:

Параллельно для всех вершин $v \in V$ *выполнить*
включить v *в множество* V *с вероятностью* $1/(2d(v))$
параллельно для всех ребер графа (u,v) *из* X *выполнить*
если $d(u) < d(v)$, *то удалить* u *из* X;
если $d(v) < d(u)$, *то удалить* v *из* X;
если $d(v) = d(u)$, *то удалить каждую из вершин из* X *с вероятностью* $1/2$.
Выдать X *в качестве независимого множества.*

Основу анализа алгоритма составляет следующая

Лемма. *Математическое ожидание числа ребер графа* G,

инцидентных вершинам построенного описанным выше вероятностным алгоритмом независимого множества либо инцидентных смежным им вершинам, не меньше 1/8 m, *где* m – *число ребер графа.*

Из этой леммы вытекает, что среднее число обращений к описанной выше вероятностной процедуре нахождения независимого множества до построения максимального по включению независимого множества есть O(log n), где n – число вершин исходного графа.

Для графов, не содержащих подграфов гомеоморфных $K_{3,3}$, известен алгоритм решения за время $O(\log^2 n)$ на CRCW PRAM с O(n) процессорами [195].

Раскраска вершин в Δ + 1 *и* Δ *цветов.*

Вход: Граф с максимальной степенью вершин Δ.

Выход: Правильная раскраска вершин графа в Δ + 1 цветов.

Последовательный алгоритм решения тривиален: выделяем произвольную вершину и рекурсивно раскрашиваем остальные вершины. Поскольку степень выделенной вершины не превосходит Δ, найдется свободный цвет, в который и окрашивается эта вершина. Известен алгоритм с временем работы $O(\log^3 n \log\log n)$ на CREW PRAM с O(n +m) процессорами [220]. Известен также NC–алгоритм нахождения правильной раскраски вершин в Δ цветов на CRCW PRAM [180]. В[161] независимо предложен алгоритм решения этой задачи с временем работы $O(\log^4 n)$на CRCW PRAM с $O(n^4)$ процессорами.

Реберная раскраска планарного графа.

Вход: Планарный граф с n вершинами и максимальной степенью вершин Δ.

Выход: Правильная раскраска ребер в max(Δ, 19) цветов.

Известен алгоритм с временем работы $O(\log^2 n)$ на EREW PRAM с O(n) процессорами [84]. Известен оптимальный параллельный алгоритм раскраски вершин планарного графа в 5 цветов с временем работы $O(\log n \log^* n)$ на EREW PRAM с $O(n/ \log n \log^* n)$ процессорами [160].

Ушное разложение.

Ушным разложением *графа* G = (V,E) *называется последовательность простых не пересекающихся по ребрам путей* $P_0, P_1, \ldots, P_r$ *путей, где* P_0 – *цикл и только концы* P_i, i > 0 *принадлежат путям с меньшими номерами. В открытом ушном разложении концы всех* P_i, i > 1 *должны различаться.* Ушное разложение у графа существует тогда и только тогда, когда он является 2–реберно связным, а открытое ушное разложение – тогда и только тогда, когда он является

2-вершинно связным.

Вход: 2-реберно связный граф.

Выход: Ушное разложение.

Известен NC-алгоритм с той же сложностью, что и алгоритм построения остовного дерева [227],[116],[187]. Алгоритм состоит из следующих этапов:

1. Найти в G остовное дерево.

2. Выделить в нем корень и занумеровать вершины в порядке "сверху вниз".

3. Пометить каждое ребро вне остовного дерева номером наименьшего общего предшественника его концевых вершин.

4. Приписать последовательные числа ребрам вне остовного дерева в порядке неубывания их меток.

5. Пронумеровать каждое ребро остовного дерева минимальным номером ребра вне дерева, в фундаментальном цикле которого оно содержится.

Основная сложность алгоритма заключается в первом этапе. Остальные этапы выполняются с помощью рассмотренной выше техники для других задач.

Ушное разложение имеет многочисленные приложения. В частности, оно позволяет построить st-нумерацию двусвязного графа. В этой нумерации вершин предполагается наличие ребра между вершинами s и t, которые занумерованы соответственно числами 1 и n, и все остальные вершины занумерованы различными числами так, что каждая вершина имеет соседей как с большим, так и с меньшим номером. Такие нумерации используются в процедурах проверки планарности графов. Кроме того, ушное разложение играет важную роль при построении трисвязных компонент и в ряде других задач.

Известно также, что при найденном остовном дереве в графе без мостов ушное разложение может быть найдено за время O(log n)на CREW PRAM с O((n + m)/log n) процессорами [219], [225], [116]. В работе [93] с использованием ранее предложенной техники показано, чтоимеется алгоритм построения ушного разложения за время O(log n)на CRCW PRAM с O((n + m)/log n) процессорами.

Проверка планарности.

Вход: Граф G.

Вопрос: Является ли граф планарным ?

Известен алгоритм с временем работы O(log n) на CRCW PRAM с O(C(n,m)) процессорами, где C(n,m) - число процессоров, необходимых для нахождения компонент связности графов с n вершинами и m ребрами за логарифмическое время [273].

Алгоритм строит открытое ушное разложение графа G и получает на основе этого так называемый граф локального размещения G_1 вместе с орграфом его st-нумерации и остовным деревом. Для каждой части ушного разложения выделяется фундаментальный цикл, образованный относительно остовного дерева единственным ребром этой части ушного разложения, не содержащемся в остовном дереве. Для каждой части разложения алгоритм находит некоторое приближение мостов этого фундаментального цикла, касающихся некоторой части ушного разложения и называемых n- мостами (nonanchor bridges) и a-мостами (anchor bridgelets). Первые - это мосты фундаментального цикла, касающиеся только внутренних вершин части ушного разложения, а вторые - подграфы мостов, касающиеся как внутренних вершин части ушного разложения, так и остающейся части цикла. По этим n-мостам и a-мостам строится граф G^*, обладающий следующим свойством: если G планарен, то каждая правильная 2-раскраска вершин G^* в {0,1} дает плоскую укладку G_1 с ребром (s,t) на внешней грани.

Перечислим основные этапы параллельного алгоритма проверки планарности:

1. Зафиксировать ребро (s,t) в графе. Построить открытое ушное разложение, начиная с ребра (s,t), и получить орграф st-нумерации G_1, его остовное дерево T_1 и ассоциированное открытое ушное разложение $D_1 = [P_0,\ldots,P_{r-1}]$.

2. Построить граф ограничений G^*, формируя parity-граф для каждой части ушного разложения и добавляя для них связи некоторые ребра из a-мостов различных частей ушного разложения.

3. Найти 2-раскраску G^*.

4. Каждой части ушного разложения P_i приписать все касающиеся ребра, чьи соответствующие вершины в G^* окрашены цветом 0 внутри P_i, и остающиеся касающиеся ребра вне P_i.

5. Найти вложенную структуру мостов, приписанных каждой стороне данной части ушного разложения и, следовательно, порядок

ребер вокруг каждой вершины.

6. Вычислить число граней в этом комбинаторном вложении и применить формулу Эйлера для проверки планарности графа.

7. Если G – планарный, то стянуть все вершины в LT_v в вершину v для каждого v. Результирующая упорядоченность ребер вокруг каждой вершины дает плоское вложение G.

Тщательный анализ каждого этапа алгоритма с использованием предыдущих результатов в этой области позволяет получить следующую теорему:

планарность можно распознать на CRCW PRAM *за время* $O(\log n)$ *с числом процессоров, достаточным для распознавания связности за логарифмическое время* [273].

Сепаратор в планарном графе.

Вход: Планарный граф с n вершинами.

Выход: Множество вершин, удаление которых разделяет граф на компоненты связности мощности не более 2/3 n.

Известен NC-алгоритм с временем работы $O(\log^2 n)$ [133].

Разрезы и потоки в планарных графах.

Вход: Планарный n-вершинный граф с заданными весами ребер и выделенными истоком и стоком.

Выход: Минимальный разрез (максимальный поток).

Известен алгоритм с временем работы $O(\log^2 n)$ на PRAM с $O(n^3)$ процессорами. Для ориентированного графа известен алгоритм с тем же временем и числом процессоров $O(n^6)$ и алгоритм с временем работы $O(\log^3 n)$ на $O(n^4)$ процессорах [173].

Изоморфизм планарных графов.

Вход. Два планарных графа G и H.

Вопрос. Изоморфны ли графы G и H ?

В [132] предложен параллельный алгоритм решения этой задачи с временем работы $O(\log^3 n)$ на CRCW PRAM с $O(n^{3/2}/\log^2 n)$ процессорами. Алгоритм включает в себя последовательное нахождение $O(\log n)$ сепараторов, выделение и анализ компонент трисвязности на изоморфизм, сведение задачи к проблеме изоморфизма деревьев.

Максимальное паросочетание.

Вход: Граф.

Выход: Максимальное паросочетание (то есть максимальное число попарно несмежных ребер).

Известны NC-алгоритмы для следующих классов графов: регулярных двудольных, сильно хордовых, плотных графов (степень каждой вершины не меньше n/2), графов без подграфов, гомеоморфных $K_{3,3}$ [321], интервальных графов [80], двудольных графов с не более, чем полиномиальным числом совершенных паросочетаний [153]. Для интервальных графов параллельный алгоритм работает $O(\log^2 n)$ на CREW PRAM с $O(n^5/\log n)$ процессорами [80].

Для графов, без подграфов, гомеоморфных $K_{3,3}$, известен алгоритм с временем работы $O(\log^2 n)$ на PRAM с $O(n^{3.5})$ процессорами [321]. Алгоритм основан на параллельном построении *ориентации Пфаффа*. Напомним, что ориентация ребер графа называется ориентацией Пфаффа, если каждый хороший цикл четно ориентирован. Здесь хорошим называется цикл четной длины, удаление которого приводит к графу, имеющему совершенное паросочетание. Цикл четной длины называется четно ориентированным, если четное число его ребер ориентированы в одну сторону.

Важное свойство ориентации Пфаффа заключается в следующем. Пусть A - матрица смежности графа. Построим новую матрицу B, которая получается из A приписыванием элементу a_{ij} знака + , если в ориентации Пфаффа имеется ребро (i,j) и знака минус, если имеется ребро (j,i). Тогда определитель матрицы B равен квадрату числа совершенных паросочетаний графа. Поскольку определитель можно вычислять NC-алгоритмом, задача сводится к построению параллельного алгоритма нахождения ориентации Пфаффа.

Важную роль в построении параллельного алгоритма играет хорошее описание трисвязных компонент графов, не содержащих подграфов, гомеоморфных $K_{3,3}$: каждая трисвязная компонента таких графов либо является планарным графом, либо в точности K_5. На основе этой леммы Холла (1943) и прямого распараллеливания полиномиального алгоритма Кастелейна для подсчета числа совершенных паросочетаний в планарном графе и строится параллельный NC алгоритм подсчета числа совершенных паросочетаний в графе без подграфов, гомеоморфных $K_{3,3}$. Основные этапы алгоритма заключаются в следующем:

1. Найти трисвязные компоненты G, построить дерево декомпозиции.

2. Выбрать корень в произвольной вершине дерева (обозначим его H_r).

3. Вычислить в каждой вершине (компоненте) ориентацию Пфаффа.

4. Согласовать все построенные ориентации.

Не очевидным является этап 3, который опирается на лемму Литтла (1974) о построении ориентации Пфаффа двусвязного графа по ориентации его специальных подграфов.

В качестве следствия показывается также, что число совершенных паросочетаний можно подсчитать NC-алгоритмом для графов без гомеоморфных $K_{3,3}$ подграфов и для задачи о *точном паросочетании.*

Для произвольного графа известен RNC-алгоритм нахождения максимального паросочетания.[242].

Дерево поиска в ширину для графа.

Остовное дерево называется деревом поиска в ширину, если все ребра графа, не содержащиеся в дереве, соединяют вершины, находящиеся на разных ветвях дерева (дерево является корневым).

Вход: Неориентированный граф.

Выход: Дерево поиска в ширину.

Задачу можно решить поиском кратчайших путей в графе от заданного корня до всех остальных вершин с запоминанием кратчайших путей. Это можно выполнить параллельно с помощью матричной техники за время $O(\log^2 n)$. Объединение таких путей и дает дерево поиска в ширину. Отметим, что требуемое число процессоров $O(n^3)$ можно понижать до $O(M(n))$, используя технику быстрого умножения матриц, где $M(n)$ – число процессоров, достаточное для умножения $n \times n$ матриц за время $O(\log n)$ [281],[134],[100].

Дерево поиска в глубину для графа.

Вход: Неориентированный граф и некоторая вершина.

Выход: Дерево поиска в глубину с корнем в этой вершине.

Остовное дерево называется деревом поиска в глубину, если все ребра графа, не содержащиеся в дереве, соединяют вершины, одна из которых предшествует другой в дереве (дерево является корневым).

Деревья поиска в глубину являются одним из основных инструментов при построении эффективных последовательных алгоритмов на графах, поэтому вопрос о существовании параллельных алгоритмов построения таких деревьев чрезвычайно важен как с теоретической, так и с практической точек зрения

В общем случае задача построения лексикографически первого

дерева поиска в глубину является P-полной [280]. Известен RNC-алгоритм решения общей задачи [33]. Для планарных графов эта задача решается за время $O(\log^2 n)$ на $O(n)$ процессорах CRCW PRAM или за время $O(\log^2 n \log^* n)$ на $O(n)$ процессорах CREW PRAM [143] или за время $O(\log n)$ на $O(n^3)$ процессорах PRIORITY CRCW PRAM [158]. Для графов, не содержащих подграфов, гомеоморфных $K_{3,3}$, известен алгоритм решения с временем работы $O(\log^3 n)$ на CRCW PRAM с $O(n)$ процессорами [195].

Опишем кратко алгоритм построения дерева поиска в глубину в неориентированном планарном графе. Время работы алгоритма - $O(\log^2 n)$ на CRCW PRAM с $O(n)$ процессорами [163].

1. Найти двусвязные компоненты графа G (обозначим их G_i, $i=1,\ldots,q$). Пусть x_j - смежные с r вершины в G_j. Далее будут построены деревья поиска в глубину T_j для G_j с корнем x_j. Объединение остовных деревьев T_j является деревом поиска в глубину для G с корнем r.

2. Найдем циклический сепаратор C в G, который существует для любого двусвязного планарного графа [232].

3. Найдем путь P_1 из корня r в некоторую вершину x на C такую, что в P_1 нет других вершин C. Положим $P_r = P_1 \cup C \setminus (e)$, где e - ребро, инцидентное вершине x.

4. Найти связные компоненты G_i графа $G \setminus P_r$.

5. В каждой компоненте найти существенное граничное ребро e_i. Пусть x_i - конец ребра e_i в G_i.

6. Строим комбинаторное представление подграфов G_i по комбинаторному представлению G.

7. Рекурсивно применить алгоритм построения дерева поиска в глубину к каждому G_i параллельно, получаем деревья T_i в G_i с корнями x_i.

8. Полагаем $T = \cup_i (T_i \cup e_i) \cup P_r$ и выдаем его в качестве ответа.

Пусть P_r - простой путь в G, r - один из его концов. Пусть G_i - компоненты связности графа $G \setminus P_r$. Отметим что граничным ребром для данной компоненты G_i называется ребро, один конец которого принадлежит G_i, а другой нет. Существенным граничным ребром называется граничное ребро, пересекающее P_r в наиболее удаленной от r вершине (среди всех граничных ребер).

В случае ориентированных графов известен RNC-алгоритм решения для произвольного орграфа [32].

Та же задача для ориентированных ациклических графов решается NC алгоритмом [79], [190]. Время работы алгоритма $O(\log d \log n)$ на CREW PRAM с $O(n^2\lceil n/\log n\rceil)$ процессорами, где d – диаметр графа, n – число его вершин.

Определим дерево $T(x,k)$ с корнем x как все вершины, достижимые из вершины x ориентированным путем длины не более 2^k. Обозначим через $F(y|\ T(x,k))$ непосредственный предшественник вершины y в дереве $T(x,k)$. PRE и POST обозначают соответственно обходы дерева в порядке "сверху вниз" и "снизу вверх".

Основные этапы алгоритма заключаются в следующем:

1. Повторить в цикле $\lceil\log d\rceil$ раз:
   для каждой пары вершин x, $y = 1, 2, \ldots, n$ вычислить $POST(y|\ T(x,y))$.
2. Для каждой тройки вершин x,y,z выполнить:
   если $(z = x)\&(F(y|\ T(x,k) \neq 0)$ то
   $M_x^k(y,z) = POST(y|\ T(x,y))$, иначе если
   $(z \in T(x,k)\&(F(y|\ T(z,k)) \neq 0)$ то
   $M_x^k(y,z) = POST(z|\ T(x,k))$ и в противном случае
   $M_x^k(y,z) = \infty$.
3. В каждой матрице найти минимальный столбец:
   $M_x^k(y,z_1) = \min\limits_z(M_x^k(y,z)|\ z = 1,2,\ldots,n)$.
4. Вычисление новых деревьев.
   Положить $k := k + 1$ и для каждой пары вершин x, $y = 1, 2, \ldots, n$ $F(y|\ T(x,k)) := F(y|\ T(z_1,k-1))$.
5. Вычислить упорядочение в ширину.
   Для вершин x, $y = 1, 2, \ldots, n$ вычислить $PRE(y|\ T(x,\lceil\log d\rceil))$.

Восстановление турниров.

Вход: Список из n натуральных чисел.

Вопрос: Существует ли турнир с такими степенями выхода вершин?

Напомним, что турниром называется полный ориентированный ациклический граф. Для решения этой задачи известен простой последовательный алгоритм жадного типа. Параллельный алгоритм с временем работы $O(\log n)$ на CREW PRAM с $O(n^2/\log n)$ процессорами предложен в [298]. Поскольку размер выхода есть $O(n^2)$, то алгоритм является оптимальным.

Гамильтоновы циклы в плотных графах.

Вход: Граф с n вершинами, степень каждой вершины которого не меньше n/2.

Выход: Гамильтонов цикл (по теореме Дирака такой цикл всегда существует).

Известен алгоритм с временем работы $O(\log^4 n)$ для CREW PRAM с O(n) процессорами [105].

Распознавание хордовых графов.

Вход: Граф.

Вопрос: Является ли граф хордовым, т.е. всякий ли цикл длины, не меньшей четырех, имеет хорду?

Задача распознавания хордовых графов, а также многие задачи для них лежат в классе NC [115].

Распознавание кографов и parity-*графов*

Вход: Граф с n вершинами и m ребрами.

Вопрос: Является ли граф кографом (parity-графом) ?

Напомним что класс кографов порождается из одновершинных графов применением операций объединения графов и взятия дополнения к графу, а в parity-графе для любой пары вершин длины всех минимальных путей, их соединяющих, имеют одинаковую четность. Известны NC-алгоритмы решения задачи распознавания parity-графов [30] за время $O(\log^2 n)$ на ARBITRARY CRCW PRAM с O(mn) процессорами и за то же время на CREW PRAM с $O(n^4/\log^2 n)$ процессорами [268]. Для кографов известны параллельные алгоритмы решения за время $O(\log^2 n)$ на ARBITRARY CRCW PRAM следующих задач: построение транзитивной ориентации, нахождение доминирующего множества, проверка изоморфизма, нахождение клики максимального веса и вершинной раскраски минимального веса. Число процессоров для первых трех задач есть O(mn) и для последних двух - $O(n^3)$.

Ориентация ребер для получения сильно связного орграфа.

Вход. Связный неориентированный граф.

Выход. Ориентация ребер графа, дающая сильно связный орграф.

Алгоритм, решающий эту задачу за логарифмическое время, предложен в [326]. В [93] описан алгоритм для этой задачи с временем работы $O(\log n)$ на CRCW PRAM с $O((n\log n + m)/\log n)$ процессорами.

Выделение минимального сильно связного остовного подграфа.

Вход. Сильно связный орграф.

Выход. Минимальный сильно связный остовный подграф.

Известен параллельный алгоритм решения этой задачи с временем работы $O(\log^4 n)$ на CREW PRAM с $O(n^3)$ процессорами [139].

2.2.6. Вычислительная геометрия

Основным методом, используемым для построения эффективных параллельных алгоритмов решения задач вычислительной геометрии, является комбинация метода "разделяй и властвуй" с методом случайного выбора. Такой подход к задача вычислительной геометрии оказался очень плодотворным и позволил построить ряд оптимальных или близких к ним параллельных алгоритмов, как вероятностных, так и детерминированных.

Локализация точки на плоскости.

Вход: Разбиение плоскости на многоугольники.

Выход: Построить структуру данных, позволяющую для любой заданной точки указать многоугольник, ее содержащий.

В [46] предложен алгоритм решения задачи локализации точки за время $O(\log n)$ на CREW PRAM с n процессорами. В случае, если разбиение задается триангулированным планарным n-вершинным графом, задача решается за время $O(\log n)$ на CREW PRAM с $O(n/\log n)$ процессорами [89].

Трапецоидальное разбиение. Для заданного множества отрезков на плоскости для каждой конечной точки каждого отрезка определить первые отрезки, с которыми пересекаются вертикальные лучи, проходящие через эту точку.

Алгоритм сложности $O(\log n)$, использующий $O(n)$ процессоров, предложен в [46].

Определение пересечения сегментов.

Вход: На плоскости дано n сегментов.

Выход: Определить, пересекаются ли хотя бы два из них.

Известен алгоритм решения задачи за время $O(\log n)$ на CREW PRAM с n процессорами [46].

Определение всех соседних вершин в выпуклом многоугольнике.

Вход: n точек на плоскости, являющихся вершинами выпуклого многоугольника.

Выход: для каждой вершины найти ближайшие к ней вершины в евклидовой метрике.

В [285] предложен параллельный алгоритм решения задачи с временем $O(\log \log n)$ на CRCW PRAM с $O(n/\log \log n)$ процессорами. Более того, показано, что время решения этой задачи на любой CRCW PRAM с $O(n\log^c n)$ процессорами по порядку не меньше, чем $\log \log n$.

Нахождение выпуклой оболочки на плоскости и в трехмерном пространстве.

Вход: n точек на плоскости или в трехмерном пространстве.

Выход: Выпуклая оболочка этих точек.

Для плоскости известен параллельный алгоритм с временем работы $O(\log n)$ на $O(n)$-процессорной PRAM [34]. В нем используется параллельная сортировка n точек по первой координате за время $O(\log n)$ на n процессорах [37], [87], методика "разделяй и властвуй", по которой исходное множество точек разбивается на $\sqrt{n}$ подмножеств по $\sqrt{n}$ элементов в каждом, затем параллельно задача решается в каждом подмножестве и осуществляется параллельное слияние для построения искомой выпуклой оболочки. Такой подход приводит к следующему рекуррентному соотношению для $T(n)$ – времени решения задачи на n процессорах:

$$T(n) \le T(\sqrt{n}) + O(\log n), \quad T(1) = 1,$$

решением которого является функция $T(n) = O(\log n)$.

В трехмерном пространстве известен алгоритм с временем работы $O(\log^2 n \log^* n)$ на PRAM с полиномиальным числом процессоров [34].

Диаграмма Вороного на плоскости.

Вход: n точек на плоскости.

Выход: Диаграмма Вороного этого множества точек.

Известен детерминированный алгоритм с временем работы $O(\log^2 n)$ на $O(n)$-процессорной PRAM [34]. Вероятностный алгоритм Рейфа– Сена [278] решает эту задачу на CREW PRAM с $O(n)$ процессорами за время $O(\log n)$ с вероятностью, не меньшей $1 - n^{-c}$ для некоторой константы $c > 0$.

Детерминированный параллельный алгоритм основан на разбиении исходного множества точек S на два непересекающихся равномощных подмножества (проведением соответствующей вертикальной прямой),

построении для них диаграмм Вороного ("разделяй и властвуй") и процедуре параллельного слияния построенных диаграмм Вороного для подмножеств разбиения. Рассмотрим множество точек x, лежащих справа от вертикальной прямой L и таких, что окружность с центром в x, касающаяся прямой L, проходит через некоторую точку из Q, но внутри окружности не содержится других точек из Q. Множество таких точек называется береговой линией Q и обладает тем свойством, что является связной линией, делящей полуплоскость справа от L на две области точек: ближе к L, чем к Q (beachhead) и ближе к Q, чем к L. Пересечение ячеек диаграммы Вороного с beachhead может состоять из нескольких связных компонент, называемых B–ячейками. Справедлива следующая

Лемма. *За время* O(log n) *на* O(n) *процессорах можно построить береговую линию и соответствующую структуру данных, позволяющую для каждой точки* v *определять за время* O(log n), *принадлежит ли* v beachhead B *и если да, какой* B–*ячейке она принадлежит.*

Алгоритм состоит из следующих этапов:

1. Отсортировать множество S по координате x. Далее вертикальной линией множество S разбивается на два одинаковых по мощности непересекающихся подмножества P и Q и рекурсивно строится для них диаграмма Вороного.

2. Построим береговую линию для Q и соответствующую структуру данных для локализации точки в beachhead. Аналогично построим структуру данных для локализации точки в каждой части выпуклой оболочки Q. То же самое делается для подмножества P. Эти построения можно выполнить за время O(log n) на n процессорах.

3. Для каждой точки, ограниченной ребрами диаграммы Вороного для P, определим, лежит ли она ближе к Q, чем к P.

4. Вычислим линейную последовательность триангулированных подъячеек ячеек диграммы Вороного для P, объединение которых покрывает контур так, что он пересекает каждую триангулированную подъячейку в точности один раз, причем в заданном линейном порядке. Назовем эту структуру P–трубкой. Искомый контур будет найден путем слияния P–трубки с Q–трубкой (рассматриваемых как связные списки).

5. Осуществляется слияние P–трубки с Q–трубкой.

6. Построение диаграммы Вороного для S по диаграммам

Вороного для P и для Q.

Детальное распнсывание н анализ каждого этапа алгоритма приводят к упомянутой выше оценке времени работы – $O(\log^2 n)$ на $O(n)$ процессорах.

Симплициальная упаковка в d-мерном пространстве.

Вход. n гиперплоскостей в d-мерном пространстве и достаточно большое число r.

Выход. Симплнциальная упаковка (то есть разбиение всего пространства на снмплексы) с числом симплексов $O(r^d)$, покрывающая все пространство и такая, что все ее симплексы пересекают $O((n \log r)/ r)$ гиперплоскостей.

Как установлено в [85], при случайном выборе подмножества R, состоящего из r гиперплоскостей, с большой вероятностью каждый снмплекс всякой трнангуляцнн R пересекает $O((n \log r)/ r)$ гиперплоскостей. В [78] предложен метод дерандомнзацин такого вероятностного алгорнтма, основанный на детермнннрованном алгорнтме построенння кратного покрытия гнперграфа. В [60] предложен метод распараллелнвання этого алгоритма, основанный на NC-алгоритме построения кратного покрытня гиперграфа (см. раздел 2.2.4), дающнй NC-алгоритм решения задачи снмплицнальной упаковкн.

Дробное каскадирование.

Вход. Задан ориентнрованный граф $G = (V,E)$ в каждый вершнне v которого задан отсортнрованный спнсок $C(v)$.

Выход. Построить структуру данных такую, что по любому заданному маршруту $(v_1,\ldots,v_m)$ в G для произвольного элемента x одни процессор мог бы быстро локалнзовать все вхождення x во все $C(v_i)$.

Длнной входа естественно счнтать $n = |V| + |E| + \sum_{v \in V} |C(v)|$.

Известен алгорнтм с временем работы $O(\log n)$ на $O(n/\log n)$ процессорах [46].

Трехмерный максимум (множество Парето-оптимальных точек).

Вход. Множество S точек в трехмерном пространстве.

Выход. Множество точек нз S, для которых нет других точек нз S, нмеющнх все три не меньшие координаты .

Для решения этой задачн нзвестен параллельный алгорнтм с временем работы $O(\log n)$ на $O(n)$-процессорной PRAM [46].

Подсчет числа доминант.

Вход. Два множества точек A и B на плоскости.

Выход. Для каждой точки b из B определить все точки из A, имеющие обе координаты меньше, чем соответствующие координаты b.

Длиной входа естественно считать n = |A| + |B|. Для решения этой задачи известен параллельный алгоритм с временем работы O(log n) на O(n)-процессорной PRAM [46].

Видимость из точки.

Вход. Множество непересекающихся отрезков и точка на плоскости.

Выход. Указать часть плоскости, видимую из заданной точки (все отрезки считаются непрозрачными).

Для решения этой задачи известен параллельный алгоритм с временем работы O(log n) на O(n)-процессорной PRAM [46].

Триангуляция выпуклого многоугольника.

Вход. Выпуклый многоугольник на плоскости.

Выход. Разбиение этого многоугольника на треугольники.

Для решения этой задачи известен параллельный алгоритм с временем работы O(log n) на O(n)-процессорной CREW PRAM [150], для монотонного многоугольника эта задача решается за время O(log n) на O(n/log n)-процессорной CREW PRAM [150]. Для решения этой задачи Чазелли недавно предложил последовательный алгоритм с линейной сложностью.

2.2.7. Сортировка и поиск

Основными задачами в этой области являются следующие: сортировка, слияние, поиск k-го по порядку элемента в массиве. Наряду с PRAM при решении задач сортировки и поиска применяется специальная вычислительная модель PCM (parallel comparison model) – параллельная модель сравнений Вэлианта [318], в которой в качестве меры сложности учитывается только число произведенных сравнений. Если исходный массив состоит из n чисел и имеется p процессоров, то на каждом шаге могут быть выполнены любые p из $\binom{n}{2}$ возможных сравнений, причем не обязательно различных. В качестве параллельной меры сложности решения задачи берется параллельное число таких шагов, совершаемых до получения информации, по которой однозначно восстанавливается ответ.

По сравнению с PCM даже PRAM представляется более реалистичной моделью, поскольку в PCM подсчитывается только число сравнений и не учитываются затраты на другую обработку информации. Это различие особенно наглядно можно проиллюстрировать на примере задачи сортировки n чисел на n^2 процессорах. На PCM эта задача решается за константное время путем выполнения всех попарных сравнений, а для CRCW PRAM с полиномиальным числом процессоров имеется нижняя оценка времени решения вида $\Omega(\log n/ \log \log n)$ [56]. Таким образом, на обработку полной информации о соотношении всех пар чисел уходит большая часть времени алгоритма сортировки на PRAM.

Более слабой по сравнению с PCM и PRAM моделью является сеть сортировки [9]. Она имеет n входов и n выходов и состоит из компараторов – элементов с двумя входами и выходами, которые сравнивают входные элементы и подают на левый выход минимум, а на правый – максимум. *Схема с* n *входами и* n *выходами, построенная из таких компараторов, называется сетью сортировки, если она правильно сортирует любой массив из* n *чисел.* Основное ее отличие от PCM и от PRAM заключается в том, что все производимые сравнения определены заранее и не зависят от результатов предыдущих сравнений. Таким образом, в сети сортировки жестко фиксировано, результаты каких сравнений сравниваются в данном компараторе.

Нахождение максимума (минимума).

Вход: Массив целых чисел $a_1,\ldots,a_n$.

Выход: Максимальное (минимальное) по величине число в массиве $a_1,\ldots,a_n$.

Задача решается на WEAK CRCW PRAM за время $t = O(\log \log_k n)$ на $O(nk/t)$ процессорах, за константное время на $O(n^{1+\varepsilon})$ процессорах и за константное время на $O(n)$ процессорах при условии, что каждое число представляется $O(\log n)$ битами. [293].

Для получения приведенных выше результатов используется стратегия "разделяй и властвуй". Первый алгоритм изложен в разделе 2.1.3, второй – является частным случаем первого при $k = n^{\varepsilon}$.

В третьем случае, когда все числа записываются $O(\log n)$ битами, разбиваем каждое битовое слово на константное число частей длины $(\log n)/2$. Блоки обрабатываются последовательно, начиная со старших битов. Внутри блока, используя результат о моделировании t шагов p-процессорной STRONG CRCW PRAM асимптотически за то же

время на p^2-процессорной WEAK CRCW PRAM, можно найти максимум из $\sqrt{n}$ чисел за константное число шагов на n-процессорной WEAK CRCW PRAM. Затем переходим к следующему блоку битов и повторяем процедуру. При достижении и обработке последнего блока будет найден искомый максимум, битовая запись которого является конкатенацией битовых записей в блоках.

Вэлиант [318] показал, что первая и вторая оценки являются неулучшаемыми по порядку в классе алгоритмов, использующих только сравнения в качестве элементарных операций. Третья оценка использует битовые операции и поэтому в ряде случаев может оказаться лучше нижней оценки Вэлианта.

Поиск k-го по порядку элемента в массиве.

Вход: Массив целых чисел $a_1,\ldots,a_n$.

Выход: k-е по величине число в массиве $a_1,\ldots,a_n$.

Известен алгоритм для EREW PRAM с O(n/(log n loglog n)) процессорами, решающий эту задачу за время O(log n loglog n) [327]. Пусть p = n / log n log log n - число процессоров. Алгоритм состоит из 5 этапов. Предварительно полагаем m:=n; l:=k; пересылаем m и l всем процессоры; Y:=$(a_1,\ldots,a_n)$. Перед каждой итерацией входом является массив Y и требуется найти l-й наименьший элемент.

1. на i-м процессоре найти с помощью последовательного алгоритма за время O(m/p) медиану массива Y[(i - 1)m/p + 1,...,im/p].

2. Отсортировать p медиан, найденных на первом шаге, за время O(log p) на p/2 процессорах [37]. Обозначить медиану этих медиан через a. Переслать a во все процессоры.

3. Записать в ячейки s_1, s_2 и s_3 соответственно число элементов меньших, равных и больших a. Переслать s_1, s_2 и s_3 всем процессорам.

4. Если $s_1 < l \le s_1 + s_2$, в качестве ответа выдать a. В противном случае если $l \le s_1$ переслать в Y все элементы, меньшие a, m:=s_1; иначе переслать в Y все элементы, большие a; m:=s_3; l:=l - (s_1 + s_2).

5. Отсортировать не более p элементов массива Y за время O(log p). Выдать l-й элемент в качестве ответа.

Путем суммирования затрат на каждом этапе алгоритма, в [327]

показано, что общее время работы составляет $O(n/p)$ при использовании $p \le n/(\log n \log\log n)$ процессоров.

В [88] предложен оптимальный параллельный алгоритм решения задачи нахождения k-го по величине элемента в массиве n чисел за время $O(\log n \log^* n)$ на EREW PRAM с $O(n/\log n \log^* n)$ процессорами. Для CRCW PRAM соответствующие оценки параметров оптимального параллельного алгоритма таковы: время – $O(\log n \log^* n/\log\log n)$, число процессоров – $O(n \log\log n/(\log n \log^* n))$.

Имеется алгоритм [90] нахождения медианы ($k = n/2$) на n процессорах PCM за время $O((\log\log n)^2)$.

Эта оценка улучшена в [35], где построен алгоритм на n процессорах PCM, решающий эту задачу за время $O(\log\log n)$. На каждом шаге этого алгоритма осуществляется n сравнений, которые образуют новые n ребер строящегося по алгоритму графа G. В результате сравнений ребра оказываются ориентированными от большего элемента к меньшему. В качестве аппроксимации номера в упорядоченной последовательности (ранга) элемента v рассматриваются величины:

$$\rho_v^- = |\{ j: v_j \le v\}| \text{ – аппроксимация снизу,}$$

$$\rho_v^+ = (m+1) - |\{ j: v \le v_j\}| \text{ – аппроксимация сверху.}$$

Здесь m обозначает число вершин в графе, а отношение $\le$ порождается транзитивным замыканием полученного из графа частичного порядка (орграфа). В качестве графа G, по ребрам которого производятся сравнения элементов на каждом шаге, будем рассматривать некоторый граф на m вершинах с n ребрами, являющийся *расширителем*. Пусть даны числа $A > 1$, α, причем $\alpha A < 1$. Говорят, что $G = (V,E)$ является (A,α)-расширителем, если для всех $T \subseteq V$ с $|T| \le \alpha m$ выполнено $|N(T)| \ge A|T|$. Здесь $N(T)$ обозначает границу множества T в G, то есть множество всех вершин G, смежных хотя бы с одной вершиной из T. Граф G называется *слабым* (A,α)-расширителем, если для всех $T \subseteq V$ с $|T| \ge \alpha m$ выполнено $|N(T)| \ge A\alpha m$. Поскольку $N(T)$ монотонно по T, расширитель является слабым расширителем.

Пусть $T, U \subseteq V$. Граф называется (A,α)-расширителем из T в U (обозначение $T \longrightarrow U$), если для всех $X \subseteq T$ с $|X| \le \alpha|U|$ выполнено $|N(X) \cap U| \ge A|X|$.

Лемма о расширителях. Пусть $A > B > 1$ и α,β таковы, что $\alpha A < 1$ $\beta B < 1$ и пусть $c = (1 - \alpha(A - B))/(1 - \beta B)$. Пусть G – (A,α)–расширитель. Тогда для данного $U \subseteq V$, $|U| \geq cm$ существует $T \subseteq V$, $|T| < \alpha m$ такое , что G является (B,β)–расширителем из $V \setminus T$ в U.

Алгоритм состоит из двух этапов. На первом этапе осуществляется сведение задачи к аналогичной на множестве V с числом элементов $O(n/\log^3 n)$. На втором – задача решается с помощью процедуры SELECT. Опишем сначала второй этап.

Процедура SELECT(j,V) в заданном множестве V находит элемент $v_{(j)}$ – j–й элемент массива. На каждом шаге она использует сравнения в соответствии с ребрами графа G, являющегося расширителем. Из леммы, которая будет приведена ниже, вытекает, что за константное число шагов можно найти такие элементы $l,r \in V$, что

$$l \leq v_{(j)} \leq r,$$

$$\rho_r - \rho_l \leq 40(m\log n)^2/n. \qquad (2.9)$$

Теперь параллельно можно определить $V_1 = \{u \in V: l \leq u \leq r\}$ и ρ_l. Поскольку $v_{(j)}$ является элементом ранга $j + 1 - \rho_l$, то алгоритм далее работает рекурсивно с SELECT($j + 1 - \rho_l, V_1$). Из (2.9) следует, что $|V_1| \leq 40|V|(\log n)^2/(n/|V|)$. Следовательно, если $|V| \leq n/(\log n)^{1+\delta}$, то V_1 не превосходит $40|V|/(\log n)^{2\delta}$. Такой скорости сходимости достаточно, чтобы за $O(\log\log n)$ обращений к процедуре SELECT получить множество размером не более $\sqrt{n}$. Это множество можно отсортировать за константное время на PCM с n процессорами тривиально (заметим, что этот шаг не проходит на модели PRAM). Таким образом, если $|V| \leq n/\log^3 n$, то достаточно $O(\log\log n)$ обращений к процедуре SELECT(j, V) для нахождения j–го по величине числа в множестве V. Учитывая, что процедура SELECT выполняется за константное время, получим требуемый результат.

Пусть снова имеется массив целых чисел $a_1,\ldots,a_n$ и натуральное k. Сведем задачу нахождения k–го элемента в массиве A к аналогичной задаче на массиве размером $O(n/\log^3 n)$. Преобразуем A, проводя $O(\log\log n)$ этапов сравнений в соответствии с начальными ярусами сравнений в AKS–сети сортировки [37]. Пусть $m = n/\log^4 n$. Из доказательства в [37] следует, что после выполнения $O(\log\log n)$ ярусов AKS–сети получим подмножество $T \subseteq A$ размера не более m такое, что

$$|\{t \in T: |\rho_t - (k - m)| < m\}| \geq 9\,|T|/10.$$

За O(log log n) шагов SELECT(m/2, T) находит элемент $l = t_{(m/2)}$ такой, что $k - 2m \leq \rho_l \leq k$. Дополнительные O(log log n) шагов AKS дают другое множество U размера m, 9/10 элементов которого имеют ранг от m до $k + m$. Используя дополнительно O(log log n) шагов, процедура SELECT(m/2, U) находит элемент $r = u_{(m/2)}$ такой что $r > a_{(k)}$ и $\rho_r - k \leq 2m$. Таким образом, найдены элементы l и r в A, которые аппроксимируют k-й элемент в A сверху и снизу, а их ранги отличаются не более чем на $4m \leq n/\log^3 n$.

Для завершения сведения два дополнительных шага дают ρ_l и $V = \{x \in A: l \leq x \leq r\}$ – множество размером не более $n/\log^3 n$, в котором элемент ранга $k + 1 - \rho_l$ является k-м элементом в исходном массиве A. Применяя теперь процедуру SELECT($k + 1 - \rho_l$), за O(log log n) шагов получаем искомый результат (см. выше). Для завершения доказательства приведем использованную выше лемму:

Лемма. Пусть G – слабый (A, α)–расширитель, $\alpha = 1/(A + 1)$, с вершинами $V = (v_1, \ldots, v_m)$ и n ребрами. Для данного $j \leq m$ существуют вершины l и r такие, что

$$j - D \leq \rho_l^- \leq \rho_l^+ \leq j \quad \text{и}$$

$$j \leq \rho_r^- \leq \rho_r^+ \leq j + D,$$

где $D = (20\ m \log m)/A$. Элементы l и r могут быть найдены параллельно за константное число шагов.

Отметим, что на CRCW PRAM с полиномиальным числом процессоров, осуществляющих только сравнения, для поиска медианы необходимо время $\Omega(\log n/\log\log n)$ [56].

Слияние.

Вход: Упорядоченный массив A целых чисел $a_1 < \ldots < a_n$ и упорядоченный массив B целых чисел $b_1 < \ldots < b_m$.

Выход: Упорядоченный массив целых чисел, состоящий из $a_1, \ldots, a_n$, $b_1, \ldots, b_m$.

Известен параллельный алгоритм решения этой задачи на EREW PRAM с $p = O(N/\log N)$ процессорами за время $O(N/p + \log N)$, где $N = n + m$ [155], [159].

Известен алгоритм решения задачи на PCM с $\lfloor (mn)^{1/2} \rfloor$ процессорами за время O(loglog n), где $n \leq m$ [318]]. Он заключается в

следующем:

1) отметить в массиве A элементы с номерами $i\lceil\sqrt{n}\rceil$ и в массиве B – с номерами $i\lceil\sqrt{m}\rceil$, $i = 1, 2, \ldots$. Таких элементов в каждом массиве будет не более $\sqrt{n}$ и $\sqrt{m}$ соответственно.

2) сравнить попарно все отмеченные элементы из A с отмеченными элементами из B.

3) отмеченные элементы в B указывают интервал, в котором должен находиться любой отмеченный элемент из A. Сравним каждый отмеченный элемент из A с каждым элементом интервала. Пусть $(A_1,B_1),\ldots,(A_s, B_s)$ – интервалы. Имеем $\sum_{j=1}^{s}|A_j| \le n$, $\sum_{j=1}^{s}|B_j| \le m$. Из неравенства Коши следует, что

$$\sum_{i=1}^{s}\sqrt{|A_i||B_i|} \le \sqrt{\sum_{i}|A_i||B_i|} \le \lfloor\sqrt{nm}\rfloor.$$

Из этого неравенства вытекает, что для параллельного слияния интервалов (A_i,B_i) имеющихся процессоров достаточно . По индукции, длина компоненты каждого интервала ограничена корнем квадратным из длины компоненты на предыдущем шаге. На шаге 2i каждая пара интервалов производит компоненты размером не более $t_i \le \lfloor\sqrt{t_{i-1}}\rfloor$, где $t_0 = n$. Отсюда получается рекуррентное соотношение:

$$t_i \le n^{1/2^i},$$

из которого следует требуемый результат. В [70] показано, что в качестве модели для реализации этого алгоритма можно взять CREW PRAM. Это означает, что задача слияния может быть решена за время $O(\log\log n)$ на CREW PRAM с $O(n + m)$ процессорами.

Отметим, что для слияния известна нижняя оценка [70] вида $\Omega(\log\log n)$ для $n\log^c n$-процессорной РСМ (для любой константы c).

Интересно отметить, что если все целые числа сливаемых массивов принадлежат интервалу $[1,\ldots,s]$, то имеется параллельный алгоритм слияния с временем работы $O(\log\log\log s)$ на CREW PRAM с $O(N / \log\log\log s)$ процессорами, где $N = n + m$ [61]. Более того, если $s = n$, то имеется параллельный алгоритм слияния с вре-

менем работы $O(\alpha(n))$ на CREW PRAM с $O(n/\alpha(n))$ процессорами, где α – функция, обратная к функции Аккермана.

Опишем кратко, на каких идеях основаны эти алгоритмы. Отметим сначала, что задача слияния эквивалентна задаче вычисления перекрестного ранга массивов A и B, заключающейся в нахождении массивов r(A) и r(B), i-е элементы которых равны соответственно числу элементов в B, меньших i-го элемента из A, и числу элементов в A, меньших или равных i-му элементу из B. В качестве простой базовой процедуры используется следующая задача: для данного массива битов $D = [d_1,\ldots,d_n]$, из которых только один равен единице, найти номер единичного бита и переслать его всем другим битам. Алгоритм решения этой задачи на CREW PRAM с n процессорами за константное время заключается в следующем: припишем каждому d_i свой процессор и для $1 \le i \le n$ и некоторой рабочей ячейки с номером k выполним:

1) i-й процессор записывает в ячейку с номером k значение i, если $d_i = 1$, и

2) считывает значение, хранящееся в k-й ячеке.

Далее опишем базовый алгоритм слияния с константным временем работы, использующий n log s процессоров. Предварительно задача вычисления *перекрестного ранга* (эквивалентная, как отмечалось выше, задаче слияния) сводится к *задаче нахождения ближайших единиц*, заключающейся в следующем: для данного вектора $D = (d_1,\ldots,d_s)$ с компонентами, принимающими три значения 1, 0, –1, требуется для каждого $d_i = 1$ найти номер ближайшей слева l(i) и ближайшей справа r(i) единицы и передать эту информацию элементам $d_j = 0$. Формально, для каждого i такого, что $d_i = 0$, требуется найти $r(i) > i$ такое, что $d_{r(i)} = 1$ и для всех $i < j < r(i)$, $d_j \le 0$. Предполагается, что число неотрицательных компонент d_i не превосходит n.

Сведение осуществляется за константное время на n log s процессорах. Входом задачи вычисления перекрестного ранга являются два упорядоченных массива целых чисел A и B длины n и m соответственно, причем все числа принадлежат отрезку [1,s]. Пусть каждому элементу этих массивов приписано по log s процессоров. Сопоставим массивам A и B массив $D = (d_1,\ldots,d_s)$ следующим образом. Рассмотрим индекс j и три возможных случая:

1) $b_k = j$ для некоторого k. Положим $d_j = 1$ и $min(j) = l$

минимальный индекс такой, что $b_i = j$. Определим ближайшие слева и справа единицы: $l(j) = b_{\min(j)-1}$; $r(b_{\min(j)-1}) = j$.

2) не выполнены условия первого случая и $a_r = j$ для некоторого r. Положим $d_j = 0$.

3) не выполнены условия первого и второго случаев. Положим $d_j = -1$.

Предположим, что задача о поиске ближайшей единицы решена. Покажем, как найти ранг элемента a_i в массиве B (обозначим его x_i). Он равен максимальному индексу x_i такого, что $b_{x_i} < a_i$. Имеем два случая:

$$1)\ d_{a_i} = 1, \text{ тогда } x_i = \min(a_i) - 1.$$

$$2)\ d_{a_i} = 0, \text{ тогда } x_i = \min(r(a_i)).$$

Теперь опишем кратко алгоритм решения задачи нахождения ближайших единиц за константное время на n log s процессорах.

1. Построим полное бинарное дерево, листьями которого являются элементы из D. Для каждого неотрицательного листа d_i, $1 \le i \le s$, припишем один процессор d_i каждому из log s предшественников d_i в дереве.

2. Для каждой внутренней вершины v вычислим r(v) – самую правую единицу среди потомков v и l(v) – самую левую единицу среди потомков v. Вычисление основано на следующем наблюдении: r(v) является в точности единичным листом-потомком v, ближайшая справа единица которого не является потомком v. Для этого требуется константное время и 2s ячеек памяти.

Для каждого листа $d_j = 0$ пусть u_j самый первый предшественник в дереве, имеющий единичный лист.

3. Найдем u_j. Рассмотрим некоторый предшественник v листа d_j. Пусть u является непосредственным предшественником v. Если v не имеет единичного листа, но его имеет u, положим $u_j = u$. Поскольку один среди log s предшественников d_j удовлетворяет этому условию, шаг 3 можно выполнить за константное время (см. базовую процедуру, описанную выше).

4. Найдем ближайший справа элемент к d_j. Обозначим потомок u_j, являющийся также предшественником d_j, через v_j. Если v_j является левым потомком u_j, то ближайший справа к d_j есть $l(u_j)$,

в противном случае ближайший справа к d_j есть ближайший справа к $r(u_j)$.

Искомый алгоритм слияния с временем O(log log log s) на CREW PRAM с O(N / log log log s) процессорами состоит из трех этапов:

1. Разобъем входные массивы A и B на подмассивы, состоящие из log s элементов каждый, и выберем в качестве представителей первые элементы этих массивов. Получим новые массивы A_1 и B_1 длиной n/log s и m/ log s, соответственно.

2. Найдем ранг каждого элемента из A_1 в B и аналогично каждого элемента из B_1 в A с помощью описанного выше алгоритма. Получим N/ log s задач слияния размером log s каждая.

3. Выполним каждое такое слияние параллельным алгоритмом слияния с двойным логарифмическим временем работы за время O(loglog log s).

Алгоритм слияния массивов n целых чисел из интервала [1,n] производится по сходной схеме, но несколько отличаются вспомогательные задачи, используемые в виде процедур в основном алгоритме. Используется также специальная рекурсивная структура данных.

Сортировка.
Вход: Массив целых чисел $a_1,\ldots,a_n$.
Выход: Упорядоченный по возрастанию массив

$$a_{p(1)} < \ldots < a_{p(n)},$$

где p – перестановка на множестве $\{1,\ldots,n\}$.

Параллельная реализация *алгоритма Бэтчера* имеет время работы $O(\log^2 n)$ на EREW PRAM с O(n) процессорами. Более тщательная реализация приводит к алгоритму с временем работы $O(\log^2 n)$ на EREW PRAM с O(n/log n) процессорами [63].

Вэлиант [318] предложил параллельный алгоритм сортировки, основанный на его быстрой процедуре слияния двух отсортированных массивов за время O(log log n) на n процессорах PCM. Время работы его алгоритма сортировки на n процессорах составляло величину O(log n log log n). Препарата [262], используя алгоритм быстрого слияния Вэлианта, предложил параллельный алгоритм сортировки с

временем работы O(log n) на CREW PRAM с O(n log n) процессорами. Модификация алгоритма Препараты, предложенная в [203], имеет время работы O(log n log log n/ log log log n) на n процессорах.

В [37] была построена сеть сортировки, состоящая из O(n logn) компараторов и имеющая глубину O(log n). Поскольку сеть сортировки легко моделируется на PRAM , то это дает алгоритм сортировки на PRAM с n процессорами и временем работы O(logn). Его недостатком является использование расширителей, которые строятся очень сложно и приводят к большим мультипликативным константам [14], [125].

Параллельный алгоритм сортировки на EREW PRAM с n процессорами за время O(log n) был предложен в [87]. Этот алгоритм основан на процедуре слияния, причем не на быстрой процедуре Вэлианта с временем O(log log n), а на обычной процедуре. Оптимальное время получается за счет использования конвейеризации при слиянии: новые последовательности начинают сливаться, не дожидаясь окончания предыдущей операции слияния. При таком подходе удается показать, что каждое новое слияние требует всего лишь константного времени.

Опишем кратко этот алгоритм. Входные данные размещаются на листьях дерева. Пусть u – внутренняя вершина дерева. В ней вычисляется отсортированный подмассив данных L(u), соответствующий листьям поддерева с корнем u. В качестве приближения к отсортированному массиву вычисляется массив UP(u), являющийся подмножеством L(u). В процессе вычислений UP(u) увеличивается и все более точно аппроксимирует L(u). Поясним процесс вычислений в каждой внутренней вершине дерева.

На входе в вершину поступает массив UP(u) и на выходе получается массив NEWUP(u). Старый массив UP(u) переобозначается как OLDUP(u). В вершине порождается также упорядоченный по возрастанию массив SUP(u), состоящий из элементов UP(u), номера которых кратны четырем. Пусть v и w – предшественники u в дереве. Вычисляем NEWUP(u) = SUP(v) ∪ SUP(w), где ∪ – обозначает слияние.

Лемма 1. Если $0 < |UP(u)| < |L(u)|$, то $|NEWUP(u)| = 2|UP(u)|$.

Лемма 2. Алгоритм состоит из 3 log n ярусов.

Основная трудность заключается в доказательстве того, что слияние на каждом ярусе можно осуществить за константное время на O(n) процессорах. Введем некоторые новые понятия. Для двух упоря-

доченных последовательностей L и J и элемента f ∈ J рангом f в L называется номер наибольшего e ∈ L такого, что f > e.

L *называется* c*-покрытием* J, *если каждый интервал между последовательными элементами массива* L *содержит не более* c *членов массива* J. *Будем говорить, что* L *упорядочен в* J, *если для любого элемента из* L *известен его ранг в* J *(обозначение* L ⟶ J).

Следующий факт используется для осуществления слияний за константное время:

OLDSUP(v) является 3-покрытием SUP(v) для любого v; поскольку UP(u) = OLDSUP(v) ∪ OLDSUP(w) получаем, что UP(u) является 3-покрытием SUP(v), аналогично UP(u) является 3-покрытием SUP(w).

Опишем как выполняется слияние. Допустим мы знаем ранги UP(u) ⟶ SUP(u) и UP(u)⟶ SUP(w). Используя эти ранги на шаге 1 вычислим NEWUP(u) и на шаге 2 вычислим NEWUP(u)⟶NEWSUP(u) и NEWUP(u)⟶ NEWSUP(w).

Шаг 1. Пусть e – некоторый элемент SUP(v), ранг e в NEWUP(u) = SUP(v) ∪ SUP(w) равен сумме его рангов в SUP(v) и в SUP(w). Для выполнения слияния вычисляем перекрестные ранги SUP(v) и SUP(w) по методу описанному ниже. Для каждого элемента e в SUP(v) кроме его ранга в NEWUP(u) известны два элемента d и f в SUP(w), аппроксимирующие e сверху и снизу, и ранги d и f в NEWUP(u).
Для каждого элемента в NEWUP(u) запоминаем, принадлежит ли он SUP(v) или SUP(w), и запоминаем ранги аппроксимирующих элементов из другого множества.

Пусть e – элемент из SUP(v). Покажем, как вычислить его ранг в SUP(w).

1. Для каждого элемента в SUP(v) вычислим его ранг в UP(u) с помощью процессоров, приписанных элементам UP(u). Пусть y ∈ UP(u). Рассмотрим интервал I(y) в UP(u), порожденный y, и рассмотрим элементы в SUP(v), принадлежащие I(y). Отметим, что имеется не более трех таких элементов по свойству 3-покрытия. Для каждого из этих элементов определяется его ранг в UP(u) с помощью процессора, приписанного y. Этот этап занимает константное время, если каждому элементу UP приписан один процессор.

2. Для каждого элемента e в SUP(v) вычислим его ранг в – SUP(w). Найдем два элемента d и f в UP(u), аппроксимирующие e сверху и снизу. Пусть d и f имеют ранги r и t в SUP(w), соответственно. Тогда все элементы ранга не более r меньше, чем элемент e

(мы предположили, что все элементы различны), а все элементы ранга больше t больше элемента e. Таким образом, остается неясен порядок элементов с рангами s, $r < s \leq t$. Но имеется не более трех таких элементов по свойству 3-покрытия. Делая не более двух сравнений, определим относительный порядок e и этих трех элементов. Этот этап также занимает константное время при приписывании одного процессора каждому элементу массива SUP.

Шаг 2. Для каждого элемента e в NEWUP(u) необходимо определить его ранг в NEWSUP(v) (и, аналогично, в NEWSUP(w)). Зная ранги элемента из UP(u) в SUP(v) и в SUP(w), можно легко вычислить ранг этого элемента в NEWUP(u) = SUP(v) ∪ SUP(w) (он равен сумме рангов). Аналогично вычисляются ранги элементов UP(v) в NEWUP(v). Это дает ранги SUP(v) в NEWSUP(v). Значит, для каждого элемента e в NEWUP(u), полученного из SUP(v), известен его ранг в NEWSUP(v). Остается вычислить ранг элементов NEWUP(u), полученных из SUP(w).

Напомним, что для каждого элемента e из SUP(w) найдены аппроксимирующие его сверху и снизу элементы из SUP(v) и их ранги r и t в NEWSUP(v). Каждый элемент ранга не более r в NEWSUP(v) меньше e, а каждый элемент ранга больше t больше e. Значит, остается неопределенным относительно e только положение элементов с рангами s, $r < s \leq t$. Но имеется не более трех таких элементов (по свойству 3-покрытия). Как и раньше можно определить их расположение относительно e не более, чем за два сравнения. Шаг 2 также занимает константное время при приписывании одного процессора каждому элементу массива NEWUP.

Алгоритм описан и остается лишь доказать свойство 3-покрытия и оценить сложность алгоритма.

Лемма. *Для* k > 1 *на каждом этапе описанного алгоритма любые* k *смежных интервалов в* SUP(u) *содержат не более* 2k + 1 *элементов из* NEWSUP(u).

При k = 1 получаем требуемое свойство 3-покрытия.

Оценим сложность алгоритма. Подсчитаем общее число элементов в массивах UP. Если $|UP(u)| \neq 0$ и v – внутренняя вершина, то $2|UP(u)| = |NEWUP(u)| = |SUP(v)| + |SUP(w)| = (1/4)(|UP(v)| + |UP(w)|) = (1/2)|UP(v)|$.

Таким образом, суммарная величина массивов UP на ярусе u равна 1/8 от размера массивов UP на ярусе v, где v – внутренняя вершина. Это не верно для внешних вершин v. На первом ярусе, в

котором v является внешней вершиной, суммарный размер массивов UP ограничен сверху величиной n + n/8 + n/64 + ... = n + n/7; на втором ярусе – величиной n + n/4 + n/32 + ...= n + 2n/7; на третьем – величиной n + n/2 + n/16 + ...= n + 4n/7. Аналогично, на первом ярусе суммарный размер массивов SUP (и массивов NEWUP во внутренних вершинах) ограничен сверху величиной n/4 + n/32 + n/256 + ...= 2n/7; на втором ярусе – величиной n/2 + n/16 + n/128 + ...=4n/7; на третьем ярусе – величиной n + n/8 + n/64 + ...= 8n/7. Из этого заключаем , что алгоритм слияния выполняется за константное время на O(n) процессорах. При этом каждому элементу массивов UP, SUP и NEWUP приписывается по одному процессору. Если оценить более аккуратно, то можно получить верхнюю оценку для числа процессоров вида 5/2 n log n. Все вышеизложенное можно суммировать в виде следующей теоремы:

Существует алгоритм сортировки n *чисел за время* O(log n) *на* CREW PRAM *с* n *процессорами, производящий не более* 5/2 n log n *сравнений.*

Более сложные построения показывают, что существует также алгоритм сортировки n чисел за время O(log n) на EREW PRAM с n процессорами [87].

В [157] описан алгоритм сортировки n целых чисел из множества $\{1,\ldots,n^c\}$ за время O(log n) на O(n log log n/ log n) процессорах PRIORITY CRCW PRAM. Оптимальный вероятностный алгоритм решения этой задачи при c = 1 предложен в [276]. Время его работы – O(log n) на ARBITRARY CRCW PRAM.

Принципиально важным с теоретической точки зрения является полученный в [37] результат о существовании сетей сортировки с O(n log n) компараторами и глубины O(log n). Напомним, что основное отличие сети сортировки от алгоритма заключается в том, что сравниваемые пары элементов задаются заранее и не зависят от результатов проводимых сравнений.

Общая идея построения заключается в следующем. Показывается, что существует схема с O(n) компараторами, имеющая константную глубину и позволяющая делить n элементное множество на две равные части так, что в первой части все элементы меньше, чем во второй и при этом доля ошибок не превосходит ε (то есть доля числа элементов, которые должны принадлежать этому интервалу, но не включаются в него). Более точно, для любого начального отрезка длины k , $1 \le k \le n/2$, число неправильных элементов не пре-

восходит εk. Эта сеть называется ε-halver. При этом ε полагается равным достаточно малой константе ($\varepsilon = 10^{-9}$).

Затем с помощью сети ε-halver (при $\varepsilon = 10^{-15}$) строится сеть ε-nearsort (при $\varepsilon = 10^{-6}$), которая состоит из O(n) компараторов и сортирует массив из n чисел за константное время не более, чем с εk ошибками (для любого начального или конечного отрезка длины k, $1 \le k \le n/2$). Построение производится применением процедуры ε-halver ко всему массиву, затем к обоим его половинам, затем ко всем четвертям и так далее, пока размеры частей не станут меньше $n\ 10^{-9}$. Таким образом, процедура ε-nearsort получается конечным числом применений процедуры ε-halver и имеет ,следовательно, константную глубину и линейную сложность.

Затем сортируемый массив располагается на n регистрах, которые занумерованы числами от 1 до n. Для каждого i, $0 \le i \le 3 \log n$, задается разбиение множества регистров на подмножества и к каждому подмножеству разбиения независимо применяется процедура ε-nearsort. Утверждается, что исходный массив после этого будет правильно отсортирован. Число итераций, таким образом, равно $3 \log n$, глубина такой схемы есть O(log n) и число компараторов O(n log n). Поскольку сеть ε-halver достаточно просто строится с помощью *расширителей* (см. выше), а ε-nearsort – совсем просто – из ε-halver, то основная трудность заключается в описании разбиения на системы подмножеств, к которым применяется процедура ε-nearsort.

Это разбиение строится по бинарному дереву. Для удобства описания вводится дискретный параметр t, обозначающий время. В начальный момент все регистры приписаны корню дерева. Каждой вершине будет соответствовать естественный интервал регистров (как при естественной дихотомии). На ярусе k длина этих интервалов равна $n/2^k$. В момент времени t в вершинах ярусов с номерами, большими t, нет регистров. В момент времени t в вершине с номером j яруса i, $0 \le i \le t-1$, $1 \le j \le 2^i$, находится 2 интервала J_1 и J_2:

$$J_1 = (j-1)n\ 2^{-i} + [Y_t(i) + 1,\ Y_t(i+1)],$$

$$J_2 = (j-1)n\ 2^{-i} + [n\ 2^{-i} - Y_t(i+1) + 1,\ n\ 2^{-i}],$$

если j нечетно, и два интервала J_1 и J_2:

$$J_1 = (j-1)n\ 2^{-i} + [\ 1,\ Y_t(i+1)],$$

$$J_2 = (j-1)n\ 2^{-i} + [n\ 2^{-i} - Y_t(i+1) + 1,\ n\ 2^{-i} - Y_t(i)],$$

если j четно.

Регистры, приписанные j-й вершине яруса t, $1 \le j \le 2^t$, образуют интервал J:

$J = (j - 1)n\ 2^{-t} + [Y_t(t) + 1,\ n\ 2^{-t}]$, при нечетном j,

$J = (j - 1)n\ 2^{-t} + [\ 1,\ n\ 2^{-t} - Y_t(t)]$, при четном j.

Разбиения, к которым применяется процедура ε-nearsort, строятся по помеченым выше бинарным деревьям следующим образом. Разбиение Zig задается треугольниками в дереве с корневыми вершинами в четных ярусах (в каждой вершине треугольника находится некоторое множество регистров и треугольник дает подмножество разбиения, равное объединению этих регистров), а разбиение Zag - такими же треугольниками, но с вершинами в нечетных ярусах. Пусть t = 0, 1, ..., log n. Для каждого такого t рассматривается три разбиения - Zig, Zag и снова Zig. Всего получаем 3 log n разбиений, к которым и применяем последовательно процедуру ε-nearsort, в результате чего получается правильно отсортированная последовательность чисел. При этом порядок применения процедуры ε-nearsort задается числом t (от меньшего к большим), а для данного t - в таком порядке: Zig, потом Zag, потом снова Zig.

Приведем нижнюю оценку для времени сортировки на CRCW PRAM с полиномиальным числом процессоров. Эта оценка вида $\Omega(\log n/\ \log \log n)$ получена в [56]. Для сравнения отметим, что в [87] предложен алгоритм сортировки на CRCW PRAM с $n \le p \le n^2$ процессорами за время $O(\frac{\log n}{\log \log(2p/n)})$.

Топологическая сортировка.

Вход. Массив X элементов $(x_1,\ldots,x_n)$ с заданным частичным порядком на нем.

Выход. Нумерация элементов в соответствии с частичной упорядоченностью.

В задаче требуется так занумеровать элементы частично упорядоченного множества, чтобы из условия i > j (в смысле частичного порядка) вытекало бы соответствующее неравенство для их номеров: N(i) > N(j). Фактически для частичного порядка это означает разбиение на ярусы.

Рассмотрим матрицу смежности A ациклического орграфа, порожденного заданным частичным порядком. По определению k-й ярус содержит вершины, самый длинный путь до которых от входных вершин

(не имеющих предшественников) имеет длину k. Поскольку орграф ациклический, это расстояние можно вычислить последовательным вычислением степеней матрицы A от 1 до n.

2.2.8. Теория расписаний

Напомним, что под расписанием понимается распределение (нумерация) вершин орграфа (работ) по процессорам. Расписание называется корректным, если все ребра ориентированы от меньшего номера к большему. Для работ могут быть заданы времена их исполнения. Различают расписания с прерываниями и расписания без прерываний. В первом случае выполнение одной работы может прерываться и продолжаться далее на другом процессоре, а во втором случае начатая работа выполняется на одном процессоре без прерываний.

При единичных временах исполнения работ и заданном орграфе предшествования временем выполнения на p процессорах некоторого расписания называется наименьшее число подмножеств, на которые можно разбить вершины орграфа в соответствии с нумерацией вершин так, что в каждом подмножестве содержится не более p независимых (не связанных ребрами) вершин.

Списочным расписанием называется расписание, в котором вершины (работы) распределяются по процессорам в соответствии с заданной упорядоченностью (списком).

Расписания с прерываниями для идентичных машин.

Вход: n независимых работ с заданными временами исполнения $p_1,\ldots,p_n$, m идентичных машин.

Выход: Расписание с прерываниями с минимальным временем исполнения.

Отметим, что расписание с прерываниями приписывает каждой работе набор троек вида (i, s, t), где s и t обозначают начало и конец исполнения этой работы на i-й машине. Расписание называется корректным, если временные интервалы выполнения различных работ на каждой машине не пересекаются, а временные интервалы исполнения j-й работы (j = 1,...,n) также не пересекаются и в сумме дают p_j. Расписание называется оптимальным, если оно является корректным и время его исполнения минимально среди всех корректных расписаний.

Последовательный алгоритм построения оптимального расписания имеет сложность O(n) и заключается в вычислении значения T, являющегося очевидной нижней оценкой времени исполнения для любого расписания, а затем построении расписания с временем исполнения равным T. Формально алгоритм записывается следующим образом.

```
T:= max (max (p_j | 1 ≤ j ≤ n), sum(p_j | 1 ≤ j ≤ n)/ m);
s:=0; i:=1;
для j:=1 to n выполнить
    если s + p_j ≤ T то приписать (i, s, s + p_j) j-й работе;
        s:=s + p_j;
            иначе приписать (i, s, T) и (i + 1, 0, p_j +
        s - T) j-й работе;
        s:=p_j + s - T;   i:=i + 1;
```

В [107] описан параллельный алгоритм решения этой задачи за время O(log n) на EREW PRAM с O(n/log n) процессорами, основанный на прямом распараллеливании описанного последовательного алгоритма. В нем использованы параллельные процедуры вычисления префиксных сумм и максимумов [208]. Параллельный алгоритм можно записать следующим образом:

```
T:= max (max (p_j | 1 ≤ j ≤ n), sum(p_j | 1 ≤ j ≤ n)/ m);
par [1 ≤ j ≤ n]  q_j:= sum (p_k | 1≤ k ≤ j - 1);
par [1 ≤ j ≤ n]
        s_j:= q_j mod T;   i_j:= ⌊q_j/T⌋ + 1;
        если s_j + p_j ≤ T то приписать j-й работе
                        (i, s_j, s_j + p_j)
        иначе приписать j-й работе (i, s_j, T)
                и (i_j + 1, 0, p_j + s_j - T);
```

Расписания с прерываниями для однородных машин.

Вход: n независимых работ с заданными временами исполнения $p_1,\ldots,p_n$ на машине с единичной скоростью, m однородных машин с заданными скоростями $s_1,\ldots,s_m$.

Выход: Расписание с минимальным временем исполнения.

Задача является обобщением предыдущей (при $s_i = 1$, $i = 1,\ldots,m$). Отметим, что время выполнения j-й работы на i-й машине равно p_j/s_i. Для этой задачи известен последовательный алгоритм решения с временем работы $O(n + m \log m)$ (см. [191]). Как и в

предыдушей задаче, сначала находится нижняя оценка T длины оптимального расписания, а затем строится расписание с временем исполнения T. Пусть машины упорядочены по невозрастанию скоростей и m – 1 наибольшие работы упорядочены по невозрастанию времен исполнения p_j и предшествуют остальным n – m + 1 работам. Последовательный алгоритм можно представить следующим образом:

$$T := \max\left((p_1/s_1), (p_1 + p_2)/(s_1 + s_2), \ldots, (p_1 + \ldots + p_{m-1})/(s_1 + \ldots + s_{m-1}), (p_1 + \ldots + p_n)/(s_1 + \ldots + s_m)\right);$$

Построим машину, которая имеет быстродействие s_i в интервале $[(i-1)T, iT]$ $(i = 1,\ldots,m)$ и быстродействие 0 в интервале $[mT, \infty)$ (композиционную машину);

для j:=1 to n **выполнить**

найти наиболее поздний интервал (s, s + T), **в котором композиционная машина может выполнить** j–**ю работу и приписать интервал** (s, s + T) **этой работе, заменить скорость композиционной машины в момент времени** s + t **исходной скоростью машины в момент времени** s + t + T **для всех** t > 0.

После составления расписания для наибольших m – 1 работ композиционная машина имеет в каждом интервале длины T с положительной скоростью вычислительные возможности большие, чем требуется для любой из остающихся работ.

В [226] предложен параллельный алгоритм решения этой задачи за время $O(\log n + \log^3 m)$ на n–процессорной PRAM, который вначале составляет расписание для m – 1 наибольших работ, а потом для остающихся работ составляет расписание аналогично параллельному алгоритму для случая идентичных машин.

p–процессорное расписание при ограничениях предшествования в виде дерева.

Вход: Ориентированное дерево с n вершинами, соответствующими выполняемым работам, натуральное число p.

Выход: Оптимальное p–процессорное расписание.

Для этой задачи известен последовательный полиномиальный алгоритм Ху [169], выбирающий для исполнения на очередном шаге вершину с наибольшим расстоянием от корня дерева. Известен параллельный алгоритм решения за время $O(\log n)$ на EREW PRAM с n^3 процессорами [164]. Отметим, что параллельно вычислить расстояния от каждой висячей вершины до корня дерева можно методом сдваивания

путей (поскольку от каждой вершины до корня ведет ровно один путь) за время O(log n) на PRAM с O(n/log n) процессорами.

Двухпроцессорное расписание

Вход: Орграф G с n вершинами, соответствующими выполняемым работам.

Выход: Оптимальное 2-процессорное расписание.

Это частный случай NP-полной задачи "p-процессорное расписание". Для этого частного случая задачи (p=2) известен последовательный полиномиальный алгоритм [86]. Известно простое сведение этой задачи к задаче нахождения максимального паросочетания в графе. Действительно, для этого достаточно взять транзитивное замыкание орграфа ограничений предшествования и в качестве графа рассмотреть его дополнение. Нетрудно убедиться в справедливости следующего утверждения:

в построенном графе имеется совершенное паросочетание тогда и только тогда, когда в исходной задаче имеется плотное 2-процессорное расписание.

Однако, для последней задачи известны лишь RNC-алгоритмы решения. В работе [165] построен параллельный NC алгоритм решения задачи составления 2-процессорного расписания за время $O(\log^2 n)$ на EREW PRAM с полиномиальным числом процессоров .

Параллельный NC-алгоритм построения расписания является довольно сложным, поэтому мы опишем только параллельный NC-алгоритм вычисления длины оптимального 2-процессорного расписания (времени выполнения), которая будет обозначаться OPT(G).

Расстоянием D(v,u)) между вершинами v и u орграфа называется число шагов, необходимых для выполнения всех вершин t: $v < t < u$. Если вершины v и u несравнимы в орграфе, полагаем расстояние равным нулю. Добавим в отношение предшествования две вершины: начальную, от которой зависят все вершины первого яруса, и заключительную, которая зависит от всех вершин последнего яруса.

Лемма. Пусть для вершин u *и* v *существуют натуральные числа* j,k *и непустое подмножество вершин* U *такие, что для всех* $d \in U$:

$v < d < u$,

$D(v,d) \geq j$, и $D(d,u) \geq k$.

Тогда $D(v,u) \geq j + k + \lceil |U|/2 \rceil$.

Алгоритм вычисления расстояний между всеми парами вершин основан на этой простой нижней оценке. Сначала расстояния между

всеми парами вершин полагаются равными нулю. На каждой итерации проверяются все пары зависимых вершин и заменяется расстояние по лемме. По завершении работы алгоритма расстояние между начальной и заключительными вершинами будет равно длине оптимального расписания. Прямая реализация приводит к алгоритму с временем $O(\log^2 n)$ и числом процессоров PRAM $O(n^5)$.

Приведем формальную запись алгоритма:

для r:=1 **до** $\lceil \log n \rceil$ **выполнить**
  для всех t,z **с** t < z **параллельно выполнить**
  для всех $0 \le j,k < n-1$ **параллельно выполнить**

$U_{t,z,j,k} := \{ x: t < x < z,\ d_{r-1}(t,x) \ge j,\ d_{r-1}(x,z) \ge k \};$

$d_r(t,z) := \max_{U_{t,z,j,k} \neq \varnothing} (d_{r-1}(t,z),\ j + k + \lceil |U_{t,z,j,k}| \rceil)$

Сходимость алгоритма после $O(\log n)$ итераций гарантируют следующие леммы:

Лемма 1. *Алгоритм выдает расстояние между двумя вершинами не большее, чем истинное растояние.*

Это с очевидностью вытекает из того, что в алгоритме в качестве расстояния берется нижняя оценка.

Лемма 2. *После выполнения* $O(\log n)$ *итераций основного цикла алгоритма вычисленное расстояние между начальной и заключительной вершинами равно истинному расстоянию.*

Доказательство. Из [86] известно, что существует разбиение множества вершин на $X_1,\dots,X_k$ такое, что все X_i являются предшественниками X_{i+1} и $OPT(G) = \sum_i \lceil |X_i|/2 \rceil$. Добавим начальную вершину X_0 и заключительную вершину X_{k+1}. Пусть $d(X_i,X_j)$ обозначает расстояние между множествами вершин X_i и X_j, равное минимуму d(t,z) по всем $t \in X_i$, $z \in X_j$. Покажем индукцией по r, что

$$d_r(X_i, X_{i+2^r}) \ge \sum_{i < l < i+2^r} \lceil |X_l|/2 \rceil .$$

Это справедливо для r = 1, поскольку все вершины из X_{i+1} заключены между X_i и X_{i+2}. Пусть неравенство справедливо также для r. Рассмотрим, как изменяются расстояния между вершиной $t \in X_i$ и $z \in X_{i+2^{r+1}}$ на итерации r + 1. Когда $j = \sum_{i < l < i+2^r} \lceil |X_l|/2 \rceil$ и

$k = \sum_{1+2^r < i < l+2^{r+1}} \lceil |X_i|/2 \rceil$, множество $U_{t,z,j,k}$ содержит все множество X_{l+2^r}. Следовательно,

$$d_{r+1}(X_l, X_{l+2^{r+1}}) \geq |X_{l+2^r}|/2\rceil + \sum_{l < i < l+2^r} \lceil |X_i|/2\rceil +$$

$$\sum_{l+2^r < i < l+2^{r+1}} \lceil |X_i|/2\rceil \geq \sum_{l < i < l+2^{r+1}} \lceil |X_i|/2\rceil.$$

Поскольку каждое подмножество X_i содержит хотя бы одну вершину, в G имеется не более n подмножеств X_i. Отсюда и вытекает, что $d_{\lceil \log n \rceil}(X_0, X_{k+1}) \geq OPT(G)$. Таким образом, вычисляя нижние оценки, приближаемся к точному решению и находим его на итерации с номером $\lceil \log n \rceil$.

Решение задачи о 2-процессорном расписании дает NC-алгоритмы для нахождения максимального паросочетания в интервальных графах и перестановочных графах. Это вытекает из упомянутой выше связи задачи составления двухпроцессорного расписания с задачей о максимальном паросочетании в дополнении графа зависимостей работ. Вообще говоря, из описанного результата вытекает NC-алгоритм нахождения максимального паросочетания в неориентированных графах, чьи дополнения допускают транзитивную ориентацию.

Многопроцессорное расписание с ограничением на входную степень вершин в орграфе предшествования.

Вход. Число процессоров p, орграф G с n вершинами, соответствующими выполняемым работам, входной степенью вершин, не превосходящей некоторого числа k, и величиной каждого яруса не менее kp.

Выход. Оптимальное p-процессорное расписание.

При перечисленных ограничениях имеется очень простой алгоритм построения оптимального расписания. В последнем ярусе выделяется произвольное множество из p вершин и рассматривается множество его предшественников в предыдущем ярусе. Оно состоит не более, чем из kp вершин. Выбираем из этих вершин произвольные p и повторяем построение: рассматриваем их предшественников в предыдущем ярусе, выбираем из них произвольные p и так до тех пор, пока не выберем некоторые p вершин в первом ярусе. Оптимальное расписание теперь строится по такой системе подмножеств следующим

образом: оно является поярусным (до исчерпания всех вершин очередного яруса переход к вершинам следующего яруса не производится) и сначала в ярусе всегда исполняются выделенные вершины (их не более kp), а остальные – в произвольном порядке. Очевидно, что при переходе от яруса к ярусу выделенные p вершин готовы к исполнению в силу того, что все их предшественники уже выполнены на предыдущих шагах и суммарное время исполнения равно $\lceil n/p \rceil$. При заданных константах k и p и заданном разбиении орграфа на ярусы сложность такого алгоритма линейна по числу вершин орграфа.

Однако, приведенное построение выделенной системы подмножеств в ярусах является последовательным и неясно, можно ли его эффективно распараллелить. Конечно, имея $O(n^p)$ процессоров, можно построить описанную систему выделенных подмножеств за время $O(\log n)$, но интересно получить параллельный алгоритм с полиномиальным числом процессоров, не зависящим от параметра p.

Списочное расписание

Вход: Список из n независимых работ с целочисленными временами исполнения.

Выход: Оптимальное двухпроцессорное расписание в заданном списком порядке.

В рассматриваемой задаче суммарное время исполнения работ в соответствии с упорядоченностью по списку зависит лишь от распределения по процессорам. В [164] показано, что эта задача в общем случае является P-полной, но в случае ограниченности длин двоичной записи длительностей исполнения работ величиной $O(\log^c n)$ существует NC-алгоритм ее решения на PRAM с n^2 процессорами.

Часть 3. Некоторые итоги. Открытые проблемы.

В настоящем обзоре в основном рассматривались параллельные алгоритмы для абстрактных PRAM моделей, в которых почти не учитываются вопросы обмена информацией между процессорами в процессе вычислений (обмены производятся через общую память и различаются лишь дисциплинами EREW, CREW, CRCW) и другие накладные расходы, присущие вычислениям на реальных параллельных архитектурах. Если мы хотим воспользоваться каким-нибудь эффективным параллельным алгоритмом для PRAM с целью использования в реальной параллельной архитектуре, то возникает задача *отображения параллельного алгоритма на архитектуру* [3], [4], [5].

Этих вопросов с принципиальной точки зрения мы касались лишь слегка в разделах 1.1 и 1.5, но реализация конкретных параллельных алгоритмов на сетях процессоров с различной конфигурацией там не рассматривалась. Это было сделано как в силу ограниченности объема работы, так и с целью единообразного изложения основных параллельных алгоритмов, поскольку существует множество различных конфигураций процессоров и для каждой из них требуется решать задачу оптимальной реализации параллельных алгоритмов с учетом времени межпроцессорных обменов. Основные конфигурации сетей процессоров были перечислены в разделе 1.5, где рассматривались в основном вопросы существования универсальных конфигураций и вопросы оптимального вложения одних конфигураций в другие. Одной из самых популярных параллельных архитектур является *гиперкуб* или различные его модификации. Реально существующими машинами с такой архитектурой являются Hypercube, Connection machine и ряд других.

Конечно, при отображении конкретного параллельного алгоритма на данную сеть процессоров можно пользоваться общими результатами о сложности моделирования PRAM на сети процессоров с данной архитектурой, но, во-первых, наилучшее моделирование с логарифмическим замедлением является вероятностным, а , во-вторых, для конкретных алгоритмов часто удается построить соответствующее отображение на конкретную архитектуру без существенного замедления работы алгоритма. При отображении параллельных алгоритмов на конкретную архитектуру очень удобным оказывается представление алгоритмов в виде графов (схем), позволяющее нагляднее формулировать и решать возникающие задачи распределения частей графа

алгоритма по процессорам с минимизацией межпроцессорных обменов (разбиение вершин на p равных частей, минимизирующее число ребер, связывающих вершины из разных частей) и составления расписаний (определение порядка выполнения операций и распределение ресурсов).

При разработке параллельных алгоритмов для реальных машин качественно выделяются два типа параллельных систем:

1) параллельные системы с небольшим параллелизмом, когда число процессоров порядка десятка и процессоры обладают внутренним параллелизмом (Cray Y-MP, Cray-2, Cyber-205);

2) параллельные системы с большим параллелизмом, когда число процессоров измеряется сотнями, тысячами, а в ближайшее время и миллионами, но процессоры имеют простую структуру (Hypercube, Connection Machine).

Если в первом случае основу распараллеливания составляет хорошее отображение на один процессор (векторизация, конвейеризация, распределение регистров), а разрезание алгоритма на процессоры во многом аналогично модели PRAM, то во втором случае основная сложность заключается в разрезании задачи на процессоры, минимизирующем межпроцессорные обмены, и синхронизации выполнения частей задачи. В каждом из этих случаев необходимым начальным условием создания эффективной параллельной реализации алгоритма является принципиальная возможность его распараллеливания, то есть наличие эффективного PRAM-алгоритма. Именно в силу этой причины основное внимание в работе было уделено PRAM-алгоритмам. При этом, круг вопросов, связанных с отображением конкретных параллельных алгоритмов на различные параллельные архитектуры, практически не рассматривался нами и многие интересные результаты в этой области не нашли отражения в этой работе.

В [254] предпринята интересная попытка формализовать понятие эффективности параллельных алгоритмов с учетом времени на обмен данными при вычислениях, но абстрагируясь от конкретных параллельных архитектур. При таком подходе фундаментальную роль играет так называемое время задержки конкретной параллельной архитектуры τ (коммуникационная задержка), равное минимальному времени, достаточному для обмена данными между произвольными двумя процессорами. Для различных параллельных архитектур величина τ по разному зависит от числа процессоров n: от $\log n$ – для гиперкуба, до $\sqrt{n}$ – для плоской сетки и n – для кольца (цикла длины n).

Для алгоритма, заданного в виде ациклического орграфа, формулируется задача составления расписания (распределение вершин орграфа по процессорам и порядок выполнения) минимизирующее время его выполнения на параллельной архитектуре с коммуникационной задержкой τ:

если i-я вершина выполняется на p-м процессоре в момент времени t и j-я вершина является ее непосредственным предшественником в орграфе, то

1) либо j-я вершина выполнена на том же процессоре к времени t-1,

2) либо j-я вершина выполнена на другом процессоре к времени $t-1-\tau$.

Минимальное время выполнения данного орграфа (алгоритма) на параллельной архитектуре с коммуникационной задержкой τ получается как решение указанной задачи составления оптимального расписания. Несмотря на то, что в [254] показано, что эта задача является NP-полной, там же предложен простой приближенный алгоритм ее решения, дающий расписание, отличающееся по времени от оптимального не более чем в два раза. В [254] получены также точные оценки времени выполнения известных стандартных алгоритмов (сложение по методу сдваивания, быстрое преобразование Фурье и др.) на такой модели. Близкие по постановкам задачи рассмотрены также в [176].

Открытые проблемы.

В теории параллельных вычислений имеется целый ряд нерешенных задач. Основной вопрос, конечно, касается возможности эффективного распараллеливания алгоритмов. Он формализуется в виде вопроса о соотношении классов **NC** и **P**. Этот вопрос для теории параллельных вычислений имеет примерно то же значение, какое в теории сложности вычислений имеет вопрос о соотношении классов **P** и **NP** и так же далек от своего решения, поскольку предполагает получение сверхполилогарифмических оценок параллельной сложности решения некоторых задач из класса **P**. Одним из возможных путей его решения является получение нижних оценок глубины схем (без требования равномерности) для булевой функции, соответствующей некоторой задаче из класса **P**. Подобные оценки известны сейчас лишь при различных ограничениях на класс схем (монотонные схемы,

схемы константной глубины).

Важную роль играет вопрос о соотношении силы вероятностных и детерминированных параллельных алгоритмов. Ключевым здесь является вопрос о соотношении классов NC и RNC. Интерес представляет также и вопрос о соотношении классов RNC и P.

В более слабой форме вопрос о возможности эффективного распараллеливания алгоритмов можно сформулировать следующим образом: можно ли каждый полиномиальный последовательный алгоритм преобразовать в параллельный с полиномиальным числом процессоров так, чтобы отношение времени работы параллельного алгоритма к времени работы последовательного алгоритма стремилось к нулю с ростом размерности задачи (возможность неглубокого распараллеливание). Этот вопрос формализуется в виде вопроса о соотношении классов P и PC (см. раздел 1.2).

Представляет также интерес вопрос о возможности моделирования машин Тьюринга схемами с одновременным соответствием размера схемы времени, а глубины схемы – памяти. Он формализуется в виде вопроса о соотношении классов NC и SC (см. раздел 1.1).

Среди конкретных задач, для которых неизвестна ни принадлежность их к NC, ни P–полнота выделим следующие:

1. *максимальное паросочетание в графе*;

Известна принадлежность задачи классу RNC, а также параллельный алгоритм для случая двудольных графов с временем работы $O(n^{2/3}\text{polylog}(n))$ на $M(n)$ процессорах [144].

2. *наибольший общий делитель двух n–разрядных чисел*;

Известен параллельный алгоритм с временем работы $O(n\log\log n / \log n)$ [179].

3. *дерево поиска в глубину в произвольных неориентированных и ориентированных графах*;

Задача принадлежит классу RNC. Для двудольных графов известен алгоритм с временем работы $O(n^{1/2}\text{polylog}(n))$ и $n + m$ процессорами [144].

4. *линейное программирование с точностью ε при неотрицательных исходных данных (та же задача в более слабой постановке с исходными данными в унарной системе)*;

найти минимум с мультипликативной точностью ε в задаче:

$$\sum_{i=1}^{n} c_i x_i \longrightarrow \min ,$$

$$\sum_{i=1}^{n} a_{ij} x_i \geq b_j , \quad j = 1,\ldots,m$$

при условии неотрицательности исходных данных c_i, a_{ij}, b_j

5. *аппроксимация задачи целочисленного линейного программирования с неотрицательными исходными данными с гарантированной оценкой отклонения от оптимума не более, чем в* $(1 + \varepsilon)(1 + \ln B)$ *раз, где* B – *сумма правых частей неравенств-ограничений.*

Найти минимум с мультипликативной точностью (ε = const) $(1 + \varepsilon)(1 + \ln B)$, где $B = \sum_j b_j$ в задаче:

$$\sum_{i=1}^{n} c_i x_i \longrightarrow \min , \qquad x_i \in (0,1)$$

$$\sum_{i=1}^{n} a_{ij} x_i \geq b_j , \quad j = 1,\ldots,m$$

при условии неотрицательности исходных данных c_i, a_{ij}, b_j. В [60] предложен NC-алгоритм для частного случая этой задачи ($a_{ij} \in (0, 1)$, $b_j = 1$, $c_i = O(\exp(\log^c n))$. Последовательно задача легко решается алгоритмом типа жадного.

6. *максимальное независимое множество в гиперграфе.*

Для заданной системы подмножеств конечного множества (гиперграфа) найти максимальное по включению независимое множество элементов исходного множества, то есть максимальное по включению множество элементов, никакие два из которых не содержатся ни в одном из подмножеств системы (ребре гиперграфа). Для случая обычных графов (каждое подмножество состоит из двух элементов) известен NC-алгоритм [186],[220].

7. *возведение целого в степень: для данных* n-*битовых чисел* a, b *и* m *вычислить* $a^b \bmod m$.

Для ряда задач из класса NC интерес представляет нахождение либо оптимальных параллельных алгоритмов, либо параллельных алгоритмов с наилучшими известными оценками сложности (времени, числа процессоров).

1. *компоненты связности в графе*;

Существуют ли алгоритмы нахождения компонент связности неориентированного n-вершинного графа с временем работы $O(\log^{1+\varepsilon} n)$ на CREW PRAM и с временем работы $O(\log n)$ на CRCW PRAM с полиномиальным числом процессоров ? Известны нижние оценки вида $\Omega(\log n)$ и $\Omega(\log n/ \log\log n)$ соответственно. В [175] предложен алгоритм, который на CREW PRAM с $O(n + m)$ процессорами находит компоненты связности за время $O(\log^{3/2} n)$.

2. *деление целых чисел*;

принадлежит ли деление классу NC^1 ? Известны P-равномерные семейства схем для деления с логарифмической глубиной [55],[279].

2. *целочисленная сортировка* n *целых чисел из отрезка* $[1...n^k]$, *где* k – *фиксировано.*

Существует ли детерминированный оптимальный алгоритм решения этой задачи за время $O(\log n)$ на EREW PRAM с $O(n/ \log n)$ процессорами ?. Известен детерминированный алгоритм решения этой задачи за время $O(\log n)$ на PRIORITY CRCW PRAM с $O(n \log\log n/ \log n)$ процессорами и памятью $\Omega(n^{1+\varepsilon})$ [157] и рандомизированный оптимальный алгоритм ее решения при $k = 1$ на ARBITRARYCRCW PRAM [276].

3. *Сортировка на гиперкубе.*

Возможно ли отсортировать массив из 2^n чисел на гиперкубе размерности n за время $O(n)$? Тот же вопрос для массива целых n-битовых чисел. Наилучший результат по первому вопросу получен в [103], где предложен детерминированный алгоритм с временем работы $O(n \log n)$. Вероятностный алгоритм с временем $O(n)$ предложен в [277].

4. *Моделирование одного шага* PRAM *на сети процессоров с локальной памятью.*

С какой сложностью возможно детерминированное моделирование одного шага PRAM на сети процессоров с локальной памятью ? В частности, если в качестве сети процессоров используется гиперкуб ? Вероятностное моделирование за логарифмическое время одного шага PRAM на бабочке или гиперкубе известно [274].

Заключение

Результаты, приведенные в статье, свидетельствуют о быстром развитии теории параллельных вычислений. В этой области имеется ряд существенных достижений при разработке параллельных алгоритмов решения конкретных задач. Ежегодно проводится несколько крупных международных конференций, на которых широко представлены текущие результаты в этой области. Несмотря на очевидный прогресс имеется и ряд трудных нерешенных задач, часть из которых была перечислена выше. Хотелось бы отметить ряд интересных фактов, специфичных именно для теории параллельных вычислений. Зачастую построение оптимального распараллеливания оказывается очень сложной проблемой даже для задач, имеющих тривиальные последовательные алгоритмы решения. Примеры такой ситуации дают задачи ранжирования списка, нахождения компонент связности графа, построения максимального по включению независимого множества в гиперграфе, построения двухпроцессорного списочного расписания и многие другие задачи.

С другой стороны при сохранении всех трудностей, присущих сложностной теории последовательных вычислений, теория параллельных вычислений дала все же обнадеживающую ситуацию, касающуюся получения нижних оценок времени вычисления на PRAM. Для самой сильной модели CRCW PRAM получены нижние оценки времени сортировки n чисел, сложения n битов по модулю два и решения других естественных задач вида $\Omega(\log n / \log \log n)$ при единственном условии, что число процессоров ограничено сверху некоторым полиномом от n. Более того, эти оценки или по порядку совпадают с известными верхними оценками или близки к ним. Таким образом, принципиальные трудности возникают лишь при попытках получения полилогарифмических и более высоких нижних оценок с целью разделения классов NC и P. По-видимому, наибольший интерес в настоящее время представляют разработка общих методов распараллеливания алгоритмов и новых методов получения нижних оценок сложности параллельных алгоритмов.

Литература

1. *Амербаев В.М., Пак И.Т.* Параллельные вычисления в комплексной плоскостн. – Алма-Ата: Наука. – 1984. – 184 С.

2. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ алгоритмов для ЭВМ. М.: Мнр. – 1979. – 536 с.

3. *Вальковский В.А., Константинов В.И., Миренков Н.Н.* Теорня и практнка параллельной обработки // Зарубежная радноэлектроннка. – 1976. – N 5. – С. 72 – 89.

4. *Воеводин В.В.* Математнческне моделн и методы в параллельных процессах. – М.: Наука, 1986. – 294 С.

5. *Глушков В.М., Капитонова Ю.В., Летичевский А.А.* Эффективность параллельных вычнслений прн ограниченных ресурсах. –Докл. АН СССР. – 1980. – *254*, N 3. – С. 527 – 530.

6. *Голованов П.Н., Солодовников В.И.* Быстрое параллельное вычнсленне степеней в факторкольце многочленов над конечным полем // Мат. Заметкн. – 1987. – *42*, N 6. – С. 886 – 894.

7. *Головкин Б.А.* Параллельные вычнслительные системы. – М.: Наука. – 1980. – 520 С.

8. *Гэри М., Джонсон Д.* Вычнслнтельные машины и труднорешаемые задачн. – М.: Мнр. – 1982. – 416 с.

9. *Кнут Д.* Искусство программнрования. – т. 3. – Сортнровка и понск. – М.: Мнр. – 1978. – 844 С.

10. *Кузюрин Н.Н.* Значення функционала, мажорнруемые средним значеннем, и задача об α-глубине (0,1)-матрнц// Респуб. семннар по дискретной оптнмнзацнн. Тезнсы докладов. – Киев. – 1985 – С. 73 – 74.

11. *Кузюрин Н.Н.* Аснмптотнчески точные полиномнальные алгоритмы в целочисленном линейном программнрованнн// Дискрет. матем. – 1990. – *1*, N 2. – С. 78 – 85.

12. *Кузюрин Н.Н.* О связн оптимумов в задачах линейного и целочнсленного лннейного программирования// Дискрет. матем. – 1991. – *3*, N 1. – С. 98 – 104.

13. *Мак Вильямс Ф. Дж., Слоэн Н. Дж. А.* Теорня кодов, нсправляющнх ошнбкн. – М.: – Связь. – 1979. – 744 с.

14. *Маргулис Г.* Явные конструкцнн концентраторов // Проб. пер. информацнн. – 1973. – *9*, N 4. – С. 71 – 80.

15. *Мюллер Д.Е., Препарата Ф.П.* Перестроение арнфметнческих выражений для параллельного вычнслення. – Киб. сборнпк. – вып.

16. – М.: Мир. – 1979. – С. 5 – 22.

16. *Нигматуллин Р.Г.* Сложность булевых функций. Казань: Изд-во Казан. ун-та. – 1983. – 208 с.

17. *Офман Ю.П.* Алгоритмическая сложность дискретных функций// Докл. АН СССР. – 1962. – *145*, N 1. – С. 48 – 51.

18. *Офман Ю.П.* Универсальный автомат// Труды Московского матем. общества. – М. – 1965. – *14*, С. 186 – 199.

19. *Разборов А.А.* Нижние оценки размера схем ограниченной глубины в полном базисе, содержащем функцию логического сложения // Матем. заметки. – 1987. – *41*, N 4. С. 598 – 607.

20. *Солодовников В. И.* Параллельные прямые методы решения задач линейной алгебры// Кибернетика и выч. техника. – 1988. – вып. 4, М.: Наука. – С. 28 – 55.

21. *Стокмейер Л.* Классификация вычислительной сложности проблем // Киб. сбор. – М.: Мир. – 1989. – вып. 26. – С. 20 – 83.

22. *Фаддеев Д.К., Фаддеева В.Н.* Вычислительные методы линейной алгебры. М.: Физматгиз, 1963. – 734 с.

23. *Форд Л.Р., Фалкерсон Р.Д.* Потоки в сетях. – М.: Мир – 1966. – 276 С.

24. *Фрумкин М.А.* Параллельный алгоритм отыскания корней многочлена // Препринт ЦЭМИ АН СССР. – нюль, 1990.

25. *Фрумкин М.А.* Систолические вычисления – М.: Наука. – 1990.– 191 С.

26. *Хачиян Л.Г.* Полиномиальный алгоритм в линейном программировании // Докл. АН СССР. – 1979.– *20*, С. 191 – 194.

27. *Храпченко В.М.* Асимптотическая оценка времени сложения параллельного сумматора // Мат. заметки. – 1967. – *9*. С. 35–40.

28. *Храпченко В. М.* О соотношении между сложностью и глубиной формул // Методы дискретного анализа. – 1978. – вып. 32. – Новосибирск. – С. 76 – 94.

29. *Шенхаге А., Штрассен Ф.* Быстрое умножение больших чисел // Киб. сборник. – М.: Мир. – 1973. – вып. 10. – С. 87 – 98.

30. *Adhar Gur Saran, Peng Shietung* Parallel algorithms for cographs and parity graphs with applications// J. of Algorithms. – 1990.– *11*, N 1. – С. 252 – 284.

31. *Adleman L.* Two theorems on random polynomial time // Proc. 19th Annu. Symp. on Found. of Comput. Sci. – 1978. – С. 75 – 83.

32. *Aggarwal A., Anderson R.J., Kao Ming-Yang* Parallel

depth-first search in general directed graphs// SIAM J. Comput. – 1990. – *19*, N 2. – C. 397 – 409.

33. *Aggarval A., Anderson R.J.* A random NC algorithm for depth first search// Combinatorica. – 1988. – *8*, N 1. – C. 1 – 12.

34. *A. Aggarwal, B. Chazelle, L. Guibas, C.O'Dunlaing, C.Yap.* Parallel Computational Geometry // Algorithmica. – 1988. – *3*, N 3. – C. 293 – 327.

35. *Ajtai M., Komlos J., Szemeredi E., Steiger W.L.* Optimal parallel selection has complexity O(loglog n) // J. Comput. System Sci. – 1989. – *38*, N 1. – C. 125 – 133.

36. *Ajtai M., Wigderson A.* Deterministic simulation of probabilistic constant depth computation // Proc. 26th Annu. Symp. Found. of Comput. Sci. – 1985. – C. 11 – 19.

37. *Ajtai M., Komlos J., Szemeredi E.* Sorting in c log n parallel steps // Combinatorica. – 1983. – *3*, C. 1 – 19.

38. *Alon N., Megiddo N.* Parallel linear programming in fixed dimension almost surely in constant time // Proc. 30th Annu. Symp. Found. of Comput. Sci. – 1989. – *2*. – C. 574 – 582.

39. *Alon N., Babai L., Itai A.* A fast and simple randomized algorithm for the maximal independent set problem // J. Algorithms. – 1986. – 7, C. 367 – 383.

40. *Alon N., Goldreich O., Hastad J., Peralta R.* Simple constructions of almost k-wise independent random variables // Proc. 31th Annu. Symp. Found. of Comput. Sci. – 1990. – *2*. – C. 544 – 553.

41. *Alt H., Hagerup T., Mehlhorn K., Preparata F.P.* Deterministic simulation of idealized parallel computers on more realistic ones // SIAM J. Comput. – 1987. – *16*, N 5. – C. 808 – 835.

42. *Anderson R.J., Mayr E.W.* Parallelism and the maximal path problem // Inf. Process. Lett. – 1987. – *24*, N 2. – C. 121 – 126.

43. *Anderson R.J., Mayr E.W., Warmuth M.K.* Parallel approximation algorithms for bin packing/ Inf. and Comput. – 1989. – *82*. – C. 262 – 277.

44. *Anderson R.J., Miller G.L.* Deterministic parallel list ranking // Proc. 3rd Aegean Workshop on Computing: VLSI Algorithms and Architectures, AWOC-88. Corfu, Greece, 1988. – C.

81 - 90.

45. *Angluin D.* A note on a construction of Margulis // Inf. Process. Lett. - 1979. - *8*, N 1. - C. 17 - 19.

46. *Atallah M.J., Cole R., Goodrich M.T.* Cascading divide-and-concquer: a technique for designing parallel algortithms // SIAM J. Computing. - 1989. - *18*, N. 3. - C. 499 - 532.

47. *Atallah M.J., Vishkin U.* Finding Euler tours in parallel // J. Comput. Syst. Sci. - 1984. - *29*, N 3. - C. 330 - 337.

48. *Avenhaus J., Madlener K.* The Nielsen reduction and P-complete problems in free groups// Theor. Comput. Sci. - 1984. - *32*, C. 61 - 74.

49. *Awerbuch B., Shiloach S.* New connectivity and MFS algorithms for shuffle-exchange network and PRAM // IEEE Trans. Comput. - 1987.- *C-36*, N 10. - C. 1158 - 1163.

50. *Baur W., Strassen V.* The complexity of partial derivatives // Theor. Comput. Sci. - 1983. - *22*, N 3. - C. 317 - 330.

51. *Babai L., Luks E.M., Seress A.* Permutation groups in NC // Proc. 19th Annu. Symp. on the Theory of Comput. - 1987. - C. 409 - 420.

52. *Barak A., Shamir E.* On the parallel evaluation of boolean expressions // SIAM J. Comput. - 1976. - *5*, N 4. - C. 678 - 681.

53. *Barrington D.* Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1 // J. of Comput. and Syst. Sci. -1989. - *38*, N 1. - C. 150 - 164.

54. *Batcher K.* Sorting networks and their applications // AFIPS Spring Joint Computer Conference, 1968. - *32*, C. 307 - 314.

55. *Beame P.W., Cook S.A., Hoover J.* Log depth circuits for division and related problems // SIAM J. Comput. - 1986. - *15*, N 4. - C. 994 - 1003.

56. *Beame P., Hastad J.* Optimal bounds for decision problems on the CRCW PRAM // J. Assoc. Comput. Mach. - 1989. - *36*, N 3. - C. 643 - 670.

57. *Beame P.W.* Limits on the power of concurrent-write parallel machines // Inform. and Comput. - 1988.- *76*, C. 13 - 28.

58. *Ben-Or M., Feig E., Kozen D., Tiwari P.* A fastparallel algorithm for determing all roots of a polynomialwith real roots

// SIAM J. Computing. - 1988. - *17*, N.6. - C. 1081 - 1092.

59. *Berger R., Rompel J.* Simulating ($\log^c n$)-wise independence in NC// Proc. 30th Annu. Symp. Found. of Comput. Sci. - 1989. - C. 2 - 7.

60. *Berger B., Rompel J., Schor P.W.* Efficient NC algorithms for set cover with applications to learning and geometry// Proc. 30th Annu. Symp. Found. of Comput. Sci. - 1989. - C. 54 - 59.

61. *Berkman O., Ja Ja J., Krishnamurthy S., Thurimella R., Vishkin U.* Some triply-logarithmic parallel algorithms // Proc. 31th Annu. Symp. on Found. of Comput. Sci. - 1990. - *2*. - C. 871 - 881.

62. *Bercowitz S.J.* On computing the determinant in small parallel time using a small number of processors // Inf. Process. Lett. - 1984. - *18*. - C. 147 - 150.

63. *Bilardi G., Nicolau A.* Bitonic sorting with O(n log n) comparisons // Proc. 20th Annu. Conf. on Int. Sci. and Systems. - 1986. - C. 336 - 341.

64. *Blum M.* A note on the parallel computation thesis // Inf. Process. Lett. - 1983. - *17*. - C. 203 - 205.

65. *Boppana R.* Optimal separations between concurrent-write parallel machines // Proc. 21th ACM STOC, 1989.- C. 320 - 326.

66. *Borodin A., Munro I.* The computational complexity of algebraic and numeric problems. N.-Y. Elsevier. - 1975. - 173 C.

67. *Borodin A.* On relating time and space to size and depth // SIAM J. Comput. - 1977. - *6*. C. 733 - 744.

68. *Borodin A., Cook S.A., Pippenger N.* Parallel computation for well-endowed rings and space bounded probabilistic machines // Inf. and Control. - 1983. - *58*. - C. 113 - 136.

69. *Borodin A., Hopcroft J.E.* Routing, merging and sorting on parallel models of computation // Proc. 14th Annu. Symp. on Theory of Comput. - 1982. - C. 338 - 344.

70. *Borodin A., Hopcroft J.E.* Routing, merging and sorting on parallel models of computation // J. Comput. and Syst. Sci.- 1985. - *30*, C. 130 - 145.

71. *Boyar J.F., Karloff H.J.* Coloring planar graphs in parallel // J. Algorithms. - 1987. - *8*, N 4. - C. 470 - 479.

72. *Brebner G.J., Valiant L.G.* Universal schemes for parallel communication // Proc. 13th Annu. Symp. on Theory of

Comput.- 1981. - C. 263 - 277.

73. *Brent R.P.* The Parallel Evaluation of General Arithmetic Expressions //J. Assoc. Comput. Mach. - 1974. - *21*, N 2. - C. 201 - 206.

74. *Brent R.P.* Fast multiple-precision evaluation of elementary functions//J. Assoc. Comput. Mach. - 1976.- *23*. - C. 242 - 251.

75. *Brent R.P., Kuck D.J., Maruyama K.* The parallel evaluation of arithmetic expressions without divisions // IEEE Trans. Comput. - 1973. - *C-22*, N 5. - C. 532 - 534.

76. *Breslauer D., Galil Z.* An optimal O(log log n) time parallel string matching algorithm // SIAM J. Comput. - 1990. - *19*, N 6. - C. 1051 - 1058.

77. *Chandra A.K., Kozen D.C., Stockmeyer L.J.* Alternation // J. Assoc. Comput. Mach. - 1981. - *28*, N 1. - C. 114 - 133.

78. *Chazelle B., Friedman J.* A deterministic view of random sam-pling and its use in geometry // Proc. Annu. Symp. on Found. of Comput. Sci. - 1988. - C. 539 - 548.

79. *Chaudhuri P.* Finding and updating depth-first spanning trees of acyclic digraphs in parallel // Computer J. - 1990. - *33*, N 3. - C. 247 - 251.

80. *Chen L.* Efficient parallel algorithms for several intersection graphs // Proc. IEEE Int. Symp. Circuits and Syst. - Portland. - New York. - 1989. - *2*. - C. 973 - 976.

81. *Chen S.C., Kuck D.J.* Time and parallel processor bounds for linear recurrence systems // IEEE Trans. Comput. - 1975. - *C-24*, C. 701 - 717.

82. *Chen C. C.-Y., Das S.K.* Parallel algorithms for level-ordered traversals of general trees // J. Combinatorics, Inf. and Syst. Sci. - 1989. - *14*, N 2-3. - C. 135 - 162.

83. *Chistov A.L.* Fast parallel calculation of the rank of matricies over a field of arbitrary characteristic // Lect. Notes Comp. Sci. - 1985. - *199*. - C. 63 - 69.

84. *Chrobak M., Yung M.* Fast algorithms for edge-coloring planar graphs // J. Algorithms. - 1989. - *10*, C. 35 - 51.

85. *Clarkson K.L.* A randomized algorithm for closest point queries // SIAM J. Comput. - 1988. - *17*, C. 830 - 847.

86. *Coffman E., Jr., Graham R.* Optimal scheduling for two processor systems// Acta Informatica. - 1972. - *1*, C. 200 - 213.

87. *Cole R.* Parallel merge sort// SIAM J. Comput. – 1988. – *17*, C. 770 – 785.

88. *Cole R.* An optimally efficient selection algorithm // Inf. Process. Lett. – 1988. – *26*, N 6. – C. 295 – 299.

89. *Cole R., Zajicek O.* An optimal parallel algorithm for building a data structure for planar point location // J. Parallel Distr. Comput. – 1990. – *8*. – C. 280 – 285.

90. *Cole R., Yap C.K.* A parallel median algorithm // Inf. Process. Lett. – 1985. – *20*, C. 137 – 139.

91. *Cole R., Vishkin U.* Deterministic coin tossing with applications to optimal parallel list ranking // Inf. Control. – 1986. – *70*, N 1. – C. 32 – 53.

92. *Cole R., Vishkin U.* Approximate parallel scheduling. Part 1: The basic technique with applications to optimal parallel list ranking in logarithmic time// SIAM J. Comput. – 1988. – *17*, N 1. – C. 128 – 142.

93. *Cole R., Vishkin U.* Approximate parallel scheduling. 11.: Applications to logarithmic-time optimal parallel graph algorithms // Inf. and Comput. – – 1991. – *92*, N 1. – C. 1 – 47.

94. *Cole R., Vishkin U.* Faster optimal parallel prefix sums and list ranking // Inf. Comput. – 1989. – *81*, N 3. – C. 334 – 352.

95. *Cook S.A.* A taxonomy of problems with fast parallel algoritms // Inform. and Contr. – 1985. – *64*. C. 2 – 22.

96. *Cook S.A., Hoover H.J.* A depth-universal circuit// SIAM J. Comput. – 1985. – *14*, N 4. – C. 833 – 839.

97. *Cook S.A. McKenzie P.* Problems complete for deterministic logarithmic space//J. Algorithms. – 1987.– *8*, N 3. – C. 372 – 384.

98. *Cook S.A.* Towards a complexity theory of synchronous parallel computation // Enseign. Math. – 1981. – *27*, C. 99 – 124.

99. *Cook S.A., Dwork C., Reischuk R.* Upper and lower time bounds for parallel random access machines without simultaneous writes // SIAM Computing. – 1986. – *15*, N 1. – C. 87 – 97.

100. *Coppersmith D., Winograd D.* Matrix multiplication via arithmetic progressions // Proc. 19thAnnu. Symp. on Theory of Comput. – New York. – 1987. – C. 1 – 6.

101. *Csanky L.* Fast parallel matrix inversion algorithms //

SIAM J. Comput. - 1976. - *5*, N 4. - C. 618 - 623.

102. *Cypher R.E.* Theoretical aspects of VLSI pin limitations// Techn. Report 89-02-01. - Univ. of Washington. - Dep. of Comput. Sci. - February 1989.

103. *Cypher R.E., Plaxton C.G.* Deterministic sorting in nearly logarithmic time on the hypercube and related computers // Proc. of the 22nd Annu. ACM Symp. on Theory of Computing. - 1990. - C. 193 - 203.

104. *Dadoun N., Kirpatrick D.* Parallel construction of subdivision hierarchies // J. Comp. Syst. Sci. - 1989. - *39*, C. 153 - 165.

105.. *Dahlhaus E., Hajnal P., Karpinski M.* Optimal parallel algorithm for the Hamiltonian cycle problem on dense graphs // Proc. Annu. Symp. on Found. of Comput. Sci. - 1988. - C. 186 - 193.

106. *Dekel E., Nassimi D., Sahni S.* Parallel matrix and graph algorithms// SIAM J. Comput. - 1981. - *10*, C. 657 - 673.

107. *Dekel E., Sahni S.* Parallel scheduling algorithms// Oper. Res. - 1983. - *31*, N 1. - C. 24 - 49.

108. *Deng X.* An optimal parallel algorithm for linear programming in the plane // Inf. Proc. Letters. - 1990. - *35*, C. 213 - 217.

109. *Dobkin D., Lipton R.J., Reiss S.P.* Linear programming is logspace hard for P//Inform. Process. Lett. - 1979. - *8*. C. 96 - 97.

110. *Dolev D., Dwork C., Pippenger N., Wigderson A.* Superconcentrators, generalizers and generalized connectors with limited depth // Proc. 15th Annu. Symp. on Theory of Comput. - 1983.- C. 42 - 51.

111. *Dwork C., Kanellakis P.C., Mitchell J.C.* On the sequential nature of unification // J. Logic Programming. - 1984. - *1*. - C. 35 - 50.

112. *Dwork C., Kanellakis P.C., Stockmeyer L.* Parallel algorithms for term matching // SIAM J. Comput. - 1988. - *17*, N 4. - C. 711 - 731.

113. *Eberly W.* Very fast polynomial arithmetic.// SIAM Computing. - 1989. - *18*, N. 5. - C. 955 - 976.

114. *Edelsbrunner H.* Algorithms in Combinatorial Geometry. - Springer-Verlag. - 1987.

115. *Edenbrandt A.* Chordal graph recognition is in NC // Inf. Process. Lett. – 1987. – *24*, C. 239 – 241.

116. *Eppstein D., Galil Z.* Parallel algorithms techniques for combinatorial optimization // Ann. Rev. Comput. Sci. – 1988. – *3*, C. 233 – 283.

117. *Fernandez de la Vega, W, Lueker G.* Bin packing can be solved within 1 + ε in linear time// Combinatirica. – 1981.– *1*, N 4. – C. 349 – 355.

118. *Fich F.E., Li M., Ragde P., Yesha Y.* On the power of concurrent-write PRAMs with read-only memory // Inf. and Comput. – 1989. – *83*. – C. 234 – 244.

119. *Fich F.E., Ragde P., Wigderson A.* Relations between concurrent-write models of parallel computation // SIAM J. Comput. – 1988. – *17*, N 3. – C. 606 – 627.

120. *Fortune S. Wyllie J.* Parallelizm in random access machines. // Proc. 10th Annu. Symp. on Theory of Computing. – San-Diego. – USA. – 1978. – C. 114 – 118.

121. *Frumkin. M.A.* A Fast Parallel Algorithm for Eigenvalue Problem of Jacobi Matrices // MFCS . – Lect. Notes in Comput. Sci. – 1988. – *324*. – C. 291 – 299.

122. *Fussel D., Ramachandran V., Thurimella R.* Finding triconnected components by local replacements // Proc. 16th Int. Colloq. on Automata, Languages and Programming // Italy. – 1989. – Lect. Notes in Comput. Sci. – 1989. – C. 379 – 393.

123. *Furst M., Saxe J.B., Sipser M.* Parity, circuits and the polynomial time hierarchy // Math. Systems Theory. – 1984. – *17*, C. 13 – 28.

124. *Furst M., Saxe J., Sipser M.* Parity, circuits and the polynomial-time hierarchy // Proc. 22th Annu. Symp. on Found. of Comput. Sci. – 1981. – C. 260 – 270.

125. *Gabber O., Galil Z.* Explicit constructions of linear size superconcentrators // Proc. 20th Annu. Symp. on Found. of Comput. Sci. – 1979. – C. 364 – 370.

126. *Gabow H.N., Tarjan R.E.* Almost-optimum speed-ups of algorithms bipartite matching and related problems // Proc. 20st Annu. Symp. on Theory of Comput. – 1988.– C. 514 – 527.

127. *Galil Z.* Sequential and parallel algorithms for finding maximum matchings in graphs // Ann. Rev. Comput. Sci.– 1986. – *1*, C. 197 – 224.

128. *Gafni E., Naor J., Ragde P.* On separating the EREW and CREW PRAM models // Theor. Comput. Sci. – 1989. – *68*. – C. 343 – 346.

129. *Galil Z.* Optimal parallel algorithms for string matching // Inf. Control. – 1986. – *67*. – C. 144 – 157.

130. *Galil Z., Pan V.* Improved processor bounds for algebraic and combinatorial problems in RNC // Proc. 26th Annu. Symp. on Found. of Comput. Sci. – New York. – 1985. – C. 490 – 495.

131. *Gazit H.* An optimal randomized parallel algorithm for finding connected components in a graph // Proc. 27th Annu. Symp. on Found. of Comput. Sci. – New York. – 1986. – C. 492 – 501.

132. *Gazit H.* A deterministic parallel algorithm for planar graps isomorphism // Proc. 32th Annu. Symp. on Found. of Comput. Sci. – 1991. – C. 723 – 732.

133. *Gazit H., Miller G.L.* A parallel algorithm for finding a separator for planar graphs // Proc. 28th Annu. Symp. on Found. of Comput. Sci. – 1987. – C. 238 – 248.

134. *Gazit H., Miller G.L.* An improved parallel algorithm that computes BFS numbering of a directed graph // Inf. Process. Lett. – 1988. – *28*, N 2. – C. 67 – 70.

135. *von zur Gathen J.* Parallel algorithms for algebraic problems//Proc. 15th Annu. Symp. on Theory of Comput. – 1983. – C. 17 – 23.

136. *von zur Gathen J.* Computing powers in parallel // SIAM J. Comput. – 1987. – *16*, N 5. – C. 930 – 945.

137. *von zur Gathen J.* Parallel arithmetic computations // Proc. 12th Int. Symp. on Math. Found. of Comput. Sci. – Lecture Notes in Computer Science. – 1986. – *233*. – C. 93 – 113.

138. *Gibbons A.M., Rytter W.* An optimal parallel algorithm for dynamic expression evaluation and its applications // Proc. Symp. on Foundations of Software Technology and Theor. Comput. Sci. – 1986. – C. 453 – 469.

139. *Gibbons A.M., Karp R., Ramachandran V., Soroker D., Tarjan R.* Transitive compaction via branchings // J. Algorithms. – *12*, N 1. – C. 110 – 126.

140. *Gibbons A.M., Rytter W.* Efficient parallel algorithms, Cambridge Univ. Press, UK, 1988.

141. *Gill J.* Computational complexity of probabilistic,

Turing machines/ SIAM J. Comput. – 1977. – *6*, C. 675 – 695.

142. *Goldberg M., Spenser T.* A new parallel algorithm for the maximal independent set problem // SIAM J. Comput.– 1989. – *18*, N 2. – C. 419 – 427.

143. *Goldberg A.V., Plotkin S.A. , Shannon G.E.* Parallel symmetry-breaking in sparse graphs // SIAM J. Comput. – 1988. – *1*. – C. 434 – 446.

144. *Goldberg A.V., Plotkin S.A., Vaidya P.M.* Sublinear-time parallel algorithms for mathing and related problems // Proc. 29th Annu. Symp. on Found. of Comput. Sci. – 1988. – C. 174 – 185.

145. *Goldsclager L.* The monotone and planar circuit value problem are log-space complete for P// SIGACTNews. – 1977. – *9*, N 2. – C. 25 – 29.

146. *Goldsclager L.* A space efficient algorithm for the monotone planar circuit value problem // Inf. Process. Lett. – 1980. – *10*, N 1. – C. 25 – 27.

147. *Goldschlager L.M., Shaw R.A., Staples J.* The maximum flow problem is log space complete for P // Theor. Comput. Sci.– 1982.– *21*, C. 105 – 111.

148. *Goldschlager L.M.* Synchronous parallel computation// J. Assoc. Comput. Mach. – 1982. – *29*, N 4. – C. 1073 – 1086.

149. *Goodrich M.T., O'Dunlaning C., Yap C.K.* Constructing the Voronoi diagram of a set of line segments in parallel// Lect. Notes in Comput. Sci. – 1989. – *382*, C. 12 – 23.

150. *Goodrich M.T.* Triangulating a polygon in parallel // J. of Algorithms. – 1989. – *10*. – C. 327 – 351.

151. *Greenberg A.C., Ladner R.E., Paterson M.S., Galil Z.* Efficient parallel algorithms for linear recurrence relations // Inf. Process. Lett. – 1982. – *15*. – C. 31 – 35.

152. *Grigoriev D. Y., Karpinski M., Singer M.F.* Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields// Report No. 8523-CS, Inst. fur Informatic der Univ. Bonn, May, 1988.

153. *Grigoriev D. Y., Karpinski M.* The matching problem for bipartite graphs with polynomially bounded permanents is in NC// Proc. 28th Annu. Symp. on Found. of Comput. Sci. – 1987. – C. 166 – 172.

154. *Grolmusz V., Ragde P.* Incomparability in parallel

computation // Discrete Appl. Math. - 1990. - *29*. - С. 63 - 78.

155.. *Guan Xiaojun, Langston M. A.* Time-space optimal parallel merging and sorting// Proc. Int. Conf. Parall. Process. - University Park (Pa). - London. - 1989. - *3*. - С. 1 - 8.

156. *Hagerup T.* Optimal parallel algorithms on planar graphs // Proc. 3rd Aegean Workshop on Comput. - Lect. Notes in Comput. Sci. - Springer-Verlag. - 1988. - *319*. - С. 24 - 32.

157. *Hagerup T.* Towards optimal parallel bucket sorting // Inf. and Comput. - 1987. - *75*. - С. 39 - 51.

158. *Hagerup T.* Planar depth-first search in O(log n) parallel time // SIAM J. Comput. - 1990. - *19*, N 4. - С. 678 -704.

159. *Hagerup T., Rub C.* Optimal merging and sorting on the EREW PRAM // Inf. Process. Lett. - 1989. - *33*. - С. 181 - 185.

160. *Hagerup T., Chrobak M., Diks K.* Optimal parallel 5-colouring of planar graphs // SIAM J. Comput. - 1989. - *18*, N2. - С. 288 - 300.

161. *Hajnal P., Szemeredi E.* Brooks colorings in parallel// SIAM J. Discrete Appl. Math. - 1990. - *3*, N 1. - С. 74 - 80.

162. *He X., Yesha Y.* Parallel recognition and decomposition of terminal series parallel graphs // Inf. Comput. - 1987. - *75*, N 1. - С. 15 - 38.

163. *He X., Yesha Y.* A nearly optimal parallel algorithm for constructing depth first spanning trees in planar graphs // SIAM J. Comput. - 1988. - *17*, N 3. - С. 486 - 491.

164. *Helmbold D., Mayr E.* Fast scheduling algorithms on parallel computers/ Adv. in Comp. Res. - 1987. - *4*, С. 39 - 68.

165. *Helmbold D., Mayr E.* Two processor scheduling is in NC// SIAM J. Comput.- 1987.- *16*, N 4. - С. 747 - 759.

166. *Hirschberg D.S., Chandra A.K., Sarwate D.V.* Computing connected components on parallel computers // Comm. ACM - 1979.- *22*. - С. 461 - 464.

167. *Hoover H.J., Klawe M.M., Pippenger H.J.* Bounding fan-out in logical networks // J Assoc. Comput. Mach. - 1984.- *31*, С. 13 - 18.

168. *Hornick S.W., Preparata F.P.* Deterministic PRAM simulation with constant redundency // Inf. and Comput. - 1991. - *92*, N 1. - С. 81 - 96.

169. *Hu T.C.* Parallel sequencing and assembly line problems //Oper. Res. - 1961. - *9.*- C. 841 - 848.

170. *Hyafil L., Kung H.T.* The complexity of parallel evaluation of linear recurrences // J. Assoc. Comput. Mach. - 1977. - *24*, C. 513 - 521.

171. *Immerman N.* Expressibility and parallel complexity // SIAM J. Comput. - 1989.- *18*, N 3. - C. 625 - 638.

172. *Israeli A., Shiloach Y.* An improved parallel algorithm for maximal matching // Inf. Process. Lett. - 1986. - *22*, N 2. - C. 57 - 60.

173. *Johnson D.B.* Parallel algorithms for minimum cuts and maximum flows in planar networks // J. Assoc. Comput. Mach. - 1987. - *34*, N 4. - C. 950 - 967.

174. *Jones N.D., Laaser W.T.* Complete problems for deterministic polynomial time// Theor. Comput. Sci. - 1977. - *3*, C. 105 - 117.

175. *Johnson D.B., Metaxas P.* Connected components in $O(\log^{3/2}|V|)$ parallel time for the CREW PRAM // Proc. 32th Annu. Symp. on Found. of Comput. Sci. - 1991. - C. 688 -697.

176. *Jurgen H., Kirousis L., Spirakis P.* Lower bounds and efficient algorithms for multiprocessor scheduling of dags with communication delays // Proc. 1989 ACM Symp. on Parallel Algorithms and Architectures. - Santa Fe, NM. - 1989. - C. 254 - 264.

177. *Kaltofen E., Krishnamoorthy M.S., Saunders B.D.* Fast parallel computation of Hermite and Smith forms of polynomial matrices // SIAM J. Alg. Discrete Methods. - 1987.- *8*. - C. 683 - 690.

178. *Kanevsky A., Ramachandran V.* Improved algorithms for graph four-connectivity // Proc. 28th Annu. Symp. on Found. of Comput. Sci. - 1987. - C. 252 - 259.

179. *Kannan R., Miller G., Rudolph L.* Sublinear parallel algorithm for computing the greatest common divisor of two integers // SIAM J. Comput. - 1987. - *16*, N 1. - C. 7 - 16.

180.. *Karchmer M., Naor J.* A fast parallel algorithm to color a graph with Δ colors // J. Algorithms. - 1988. - *9*. - C. 83 - 91.

181. *Karlin A.R., Upfal E.* Parallel hashing - an efficient

implementation of shared memory (preliminary version)// Proc. 18th Annu. Symp. on Theory of Comput. - Berkeley. - USA. - 1986. - C. 160 - 168.

182. *Karloff H.J., Ruzzo W.L.* The iterated mod problem // Inform. and Comput. - 1989.- *80*, C. 193 - 204.

183. *Karloff H.J.* A Las-Vegas RNC algorithm for maximum matching // Combinatorica. - 1986.- *6*, C. 387 - 392.

184. *Karloff H.J., Shmoys D.B.* Efficient parallel algorithms for edge coloring problems // J. Algorithms. - 1987. - *8*, N 1. - C. 39 - 52.

185. *Karmarkar N., Karp R.M.* An efficient approximation scheme for the one-dimensional bin-packing problem// Proc. 23rd Annu. Symp. on Found. of Comput. Sci. - 1982. - C. 312 - 320.

186. *Karp R.M., Wigderson A.* A fast parallel algorithm for the maximal independent set problem // J. Assoc. Comput. Mach. - 1985. - *32*, N 4. - C. 762 - 773.

187. *Karp R.M., Ramachandran V.* Parallel algorithms for shared memory machines // in Handbook of Theoretical Computer Science. - J. van Leeuwen, ed. - Amsterdam: - North Holland. - 1988. - C. 869 - 942.

188. *Karp R.M., Upfal E., Wigderson A.* Constructing perfect matching is in random NC// Combinatorica. - 1986. - *6*, N 1.- C. 35 - 48.

189. *Karp R.M., Upfal E., Wigderson A.* The complexity of parallel computation on matroids// Proc. 26th Annu. Symp. on Found. of Comput. Sci. - 1985. - C. 229 - 234.

190. *Kim T., Chwa K.* Parallel algorithms for a depth first search and a breadth first search // Int. J. Comput. Math. - 1986. - *19*. - C. 39 - 54.

191. *Kindervater G.* Exercises in parallel combinatorial computing. - Amsterdam. - 1989.

192. *Kindervater G., Lenstra J.K.* An introduction to parallelism in combinatorial optimization // Discrete Appl. Math. - 1986. - *14*. - C. 135 - 156.

193. *Kirkpatrick D.G., Przytycka T.* Parallel recognition of complement reducible graphs and cotree construction// Discrete Appl. Math. - 1990. - *29*, N 1. - C. 79 - 96.

194. *Kirousis L., Serna M., Spirakis P.* The parallel complexity of the subgraph connectivity problem // Proc. 30th

Annu. Symp. on Found. of Comput. Sci. – 1989. – C. 294 – 299.

195. *Khuller S.* Extending planar graph algorithms to $K_{3,3}$-free graphs // Inf. and Comput.– 1990. – *84*. – C. 13 – 25.

196. *Khuller S., Schieber B.* Efficient parallel algorithms for testing connectivity and finding disjoint s–t paths in graphs // Proc. 30th Annu. Symp. Found. Comput. Sci. –1989. – C. 288 – 293.

197. *Khuller S., Schieber B.* Efficient parallel algorithms for testing connectivity and finding disjoint s–t paths in graphs // SIAM J. Comput. – 1991. – *20*, N 2. – C. 352 – 375.

198. *Klein P.N.* Efficient parallel algorithms for chordal graphs // Proc. 29th Annu. Symp. on Found. of Comput. Sci. – 1988. – C. 150 – 161.

199. *Klein P.N., Reif J.H.* Parallel time O(log n) acceptence of deterministic CFLs on an exclusive-write P–RAM // SIAM J. Comput. – 1988.– *17*. – C. 463 – 485.

200. *Klein P., Stein C.* A parallel algorithm for eliminating cycles in undirected graphs// Inform. Process. Lett. –1990.– *34*.– C. 307–312.

201. *Kosaraju S.R., Delcher A.L.* Optimal parallel evaluation of tree–structured computations by raking // Proc. 3rd Aegean Workshop on Computing: VLSI, Algorithms and Architectures. – AWOC'88, Corfu, Greece. – 1988. – Lect. Notes in Comput. Sci. – 1988. – *319*.– C. 101 – 110.

202. *Kosaraju S.R., Delcher A.L.* A tree–partitioning technique with applications to expression evaluation and term matching // Proc. Annu. Symp. on Found. of Comput. Sci. – 1990. – C. 163 – 172.

203. *Kruskal C.* Searching, merging and sorting in parallel computations // IEEE Trans. Comput. – 1983. – *32*, C. 942 – 946.

204. *Kruskal C.P., Rudolph L., Snir M.* Efficient parallel algorithms for graph problems // Algorithmica. – 1990. – *5*. – C. 43 – 64.

205. *Kruskal C.P., Rudolph L., Snir M.* A complexity theory of efficient parallel algorithms // Theor. Comput. Sci. – 1990. – *71*, N 1. – C. 95 – 132.

206. *Kucera L.* Parallel computation and conflicts in memory access // Inf. Process. Lett. – 1982. – *14*, C. 93 – 96.

207. *Kung H.T.* New algorithms and lower bounds for parallel

evaluation of certain rational expressions and recurrences // J. Assoc. Comput. Mach. – 1976. – *23*. C. 252 – 261.

208. *Ladner R.E., Fischer M.J.* Parallel prefix computation// J. Assoc. Comput. Mach. – 1980. – *27*, C. 831 – 838.

209. *Ladner R.* The circuit value problem is log-space complete for P// SIGACT News. – 1975. – 7, N 1. – C. 583–590.

210. *Leighton T.* Tight bounds on the complexity of parallel sorting // Proc. 16th Annu. Symp. on Theory of Comput. – 1984. – C. 71 – 80.

211. *Leighton T.* Tight bounds on the complexity of parallel sorting //IEEE Trans. on Computers. – 1985. – *C–34*. – C. 344 – 354.

212. *Lev G., Pippenger N., Valiant L.G.* A fast parallel algorithm for routing in permutation networks // IEEE Trans. Comput.– 1981. – *C–30*, N 2. – C. 93 – 100.

213. *Li P.P., Martin A.J.* The sneptree a versatile interconnection network // Proc. Symp. Appl. Math. – 1985. – C. 20 – 27.

214. *Li Ming, Yesha Y.* New lower bounds for parallel computation // J. Assoc. Comput. Mach.– 1989. – *36*, N 3. – C. 671 – 680.

215. *Lipton R.J., Tarjan R.E.* A separator theorem for planar graphs // SIAM J. Appl. Math. – 1979. – *36*, N 2. – C. 177 – 189.

216. *Lipton R.J., Tarjan R.E.* Applications of a planar separator theorem // SIAM J. Computing. – 1980. – *9*, N 3. – C. 615 – 627.

217. *Litow B.E., Davida G.I.* O(log n) parallel time finite field inversion // in VLSI Algorithms and Architectures, Proc. 3rd Aegean Workshop on Computing, Lecture Notes in Computer Science.– 1988. – *319*. – Berlin. – C. 74 – 80.

218. *Lovasz L.* Determinants, matchings and random algorithms// Fundamentals of Computing Theory. – L. Budach (ed.) FCT–79, Berlin:– Akad. Ferlag. – 1979. – C. 565 – 574.

219. *Lovasz L.* Computing ears and branchings in parallel // Proc. 26th Annu. Symp. on Found. of Comput. Sci. – New York. – 1985. – C. 464 – 467.

220. *Luby M.* Removing randomness in parallel computation without a processor penalty // Proc. 29th Ann. Symp. Found. Comput. Sci.– 1988. – C. 162 – 173.

221. *Luby M.* A simple parallel algorithm for the maximal independent set problem// SIAM J. Comput. – 1986. – *5*, N 4. – C. 1036 – 1042.

222. *Lueker G.S., Megiddo N., Ramachandran V.* Linear programming with two variables per inequality in poly-log time // SIAM J. Comput. – 1990. – *19*, N 6. – C. 1000 – 1010.

223. *Luks E.M.* Parallel algorithms for permutation groups and graph isomorphism // Proc. 27th Annu. Symp. on Found. of Comput. Sci. – 1986.– C. 292 – 302.

224. *Luks E.M., McKenzie P.* Fast parallel computation with permutation groups // Proc. 26th Annu. Symp. on Found. of Comput. Sci. – 1985.– C. 505 – 514.

225. *Maon J., Schieber B., Vishin U.* Parallel ear decomposition (EDS) and st-numbering in graphs// Theor. Comput. Sci. – 1986. – *47*, N 3. – C. 277 – 298.

226. *Martel C.U.* A parallel algorithm for preemptive scheduling of uniform machines// J. of Parall. and Distrib. Comput. – 1988. – *5*, N 6. – C. 700 – 715.

227. *Mayr E.W.* Basic parallel algorithms in graph theory// Comput. Suppl. – 1990. – 7. – C. 69 – 91.

228. *Mayr E.W.* The design of parallel algorithms. Principles and problems // Parallel systems and Computation. – Amsterdam. – 1988. – C. 117 – 133.

229. *Mckenzie P., Cook S.A.* The parallel complexity of abelian permutation group problems// SIAM J. Comput. – 1987. – *16*, N 5. – C. 880 – 909.

230. *Melhorn K.* Data structures and algorithms. – v. *1*: Sorting and searching // Berlin: Springer. – 1984.

231. *Melhorn K., Vishkin U.* Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories // Acta Inform. – 1984. – *21*, C. 339 – 374.

232. *Miller G.L.* Finding small simple cycle separator for 2–connected planar graphs // J. Comput. Syst. Sci. – 1986. – *32*, C. 265 – 279.

233. *Miller G.L., Ramachandran V., Kaltofen E.* Efficient parallel evaluation of straight-line code and arithmetic circuits // SIAM Computing. – 1988. – *17*, N 4. – C. 687 – 695.

234. *Miller G. L., Ramachandran V.* A new graph

triconnectivity algorithm and its parallelization// Proc. 19th Annu. ACM Symp. on the Theory of Comput. – 1987. – C. 335 – 344.

235. *Miller G.L., Reif J.* Parallel tree contruction and its application // Proc. 26th Annu. Symp. on Found. of Comput. Sci. – 1985. – C. 478 – 489.

236. *Miyano S.* The lexicographically first maximal subgraph problems: P-completeness and NC algorithms// Math. Syst. Theory. – 1989. – *22*, N 1. – C. 47 – 73.

237. *Miyano S.* A parallelizable lexicographically first maximal edge-induced subgraph problem// Inf. Process. Lett. – 1988. – *27*, C. 75 – 78.

238. *Moitra A., Johnson R.C.* A parallel algorithm for maximum matching on interval graphs // Int. Conf. on Paral. Processing. – 1989. – C. 111 – 120.

239. *Monien B., Sudborough H.* Embedding one interconnection network in another // Computing Suppl. – 1990. – *7*. – C. 257–282.

240. *Motwani R., Naor J., Naor M.* The probabilistic method yields deterministic parallel algorithms// Proc. 30th Annu. Symp. on Found. of Comput. Sci. – 1989. – C. 8 – 13.

241. *Mulmuley K.A.* Fast parallel algorithm to compute the rank of a matrix over an arbitrary field // Combinatorica. – 1987. – *7*, N 1. – C. 101 – 104.

242. *Mulmuley K.A., Vazirani U.V., Vazirani V.V.* Matching is as easy as matrix invertion // Combinatorica. – 1987. – *7*, N 1. – C. 105 – 120.

243. *Naor J.* A fast parallel coloring of planar graphs with five colors // Inf. Process. Lett. – 1987. – *25*, N 1. – C. 51 – 53.

244. *Naor J., Naor M.* Small-bias probability spaces: efficient constructions and applications// Proc. 22th Annu. Symp. on Theory of Comput. – 1990. – C. 213 – 223.

245. *Naor J., Naor M., Schaffer A.A.* Fast parallel algorithms for chordal graps // SIAM J. Comput. – 1989. – *18*, N 2. – C. 327 – 349.

246. *Naor J., Novick M.B.* An efficient reconstruction of a graph from its line graph in parallel// J. of Algorithms. – 1990. – *11*, N 1. – C. 132 – 143.

247. *Nassimi D., Sahni S.* Data broadcasting in SIMD computers// IEEE Trans. Comput. – 1981. – *C-30*. – C. 101 – 107.

248. *Nisan N., Soroker D.* Parallel algorithms for zero-one supply-demand problems // SIAM J. Discrete Math. - 1989.- 2, C. 108 - 125.

249. *Nisan N.* CREW PRAMs and desicion trees // Proc. 21st Annu. Symp. on Theory of Comput. - 1989.- C. 327 - 335.

250. *Nisan N.* Psuedorandom generators for space-bounded computation // Proc. 22th Annu. Symp. on Theory of Comput. - 1990. - C. 204 - 212.

251. *Nisan N., Wigderson A.* Hardness versus randomness // Proc. 29th Annu. Symp. on Found. of Comput. Sci. - New York. - 1988. - C. 2 - 12.

252. *Pan V.* Complexity of parallel matrix computations// Theor. Comput. Sci. - 1987. -*54*. - C. 65 - 85.

253. *Pan V., Reif J.* Efficient parallel solution of linear systems // Proc. 17th Annu. Symp. on Theory of Comput. - 1985. - C. 143 - 152.

254. *Papadimitriou C.H., Yannakakis M.* Towards an architecture-independent analysis of parallel algorithms // SIAM J. Comput. - 1990. - *19*, N 2. - C. 322 - 328.

255. *Parberry I.* Some practical simulation of impractical parallel computers // in VLSI Algorithms and Architectures. ed. Berolazzi and Luccio F. - Elsevier. - 1985. - C. 27 - 38.

256. *Parker D.S.* Notes on shuffle/exchange type networks // IEEE Trans. Comput. Sci. - 1980. - *C-29*. - C. 213 - 222.

257. *Paterson M.S., Valiant L.G.* Circuit size is nonlinear in depth // Theor. Comp. Sci. - 1976. - *2*. - C. 397 - 400.

258. *Pippenger N.* On simultaneous resource bounds // Proc. 20th Annu. Symp. on Found. of Comput. Sci. - 1979. - C. 307 - 311.

259. *Pippenger N.* Parallel communication with bounded buffers // Proc. 25th Annu. Symp. on Found. of Comput. Sci. - 1984. - C. 127 - 136.

260. *Pippenger N.* Communication networks // in Handbook of Theoretical Computer Science. - J. van Leeuwen, ed. -v. A. - Amsterdam: - Noth-Holland. - 1990. - C. 805 - 833.

261. *Pratt V.R., Stockmeyer L.J.* A characterization of the power of vector machines // J. of Comput. and Syst. Sci. - 1976. - *12*, C. 198 - 221.

262. *Preparata F. P.* New parallel sorting schemes // IEEE

Trans. Comput. – 1978. – *C-27*, C. 669 – 673.

263. *Preparata F.P.* Inverting a Vandermonde matrix in minimum parallel time // Inf. Process. Lett. – 1991. – *38*, N 6. – C. 291 – 295.

264. *Preparata F.P., Muller D.E.* The time required to evaluate division-free arithmetic expressions // Inf. Process. Lett. – 1975. – *3*, N 5. – C. 144 – 146.

265. *Preparata F.P., Muller D.E.* Efficient parallel evaluation of boolean expressions // IEEE Trans. Comput. – 1976. – *C-25*, N 5. – C. 548 – 549.

266. *Preparata F. P., Sarwate D.V.* An improved parallel processor bound in fast matrix invertion // Inf. Process. Lett. – 1978. – *7*, N 3. – C. 148 – 150.

267. *Preparata F. P., Vuillemin J.* The cube-connected-cycles: a versatile network for parallel computation // Proc. 20th Annu. Symp. on Found. of Comput. Sci. – 1979. – C. 140 – 147.

268. *Przytycka T., Corneil D.G.* Parallel algorithms for parity graphs // J. Algorithms. – 1991. – *12*, N 1. – C. 96 – 110.

269. *Pudlak P.* The hierarchy of boolean circuits// Ceskoslovenska Akademie VED, Matematicky Ustav. – 1986. – C. 1 – 30.

270. *Raghavan P.* Probabilistic construction of deterministic algorithms: approximating packing integer programs// J. Comput. and Syst. Sci. – 1988. – *37*, N 4. – C. – 130 – 143.

271. *Ramachandran V.* Fast parallel algorithms for reducible flow graphs // Concurrent computatiopns: Algorithms, Architecture and Technology. – 1988. – Plenum Press, C. 117 – 138.

272. *Ramachandran V., Vishkin U.* Efficient parallel triconnectivity in logarithmic time // Proc. 3rd Aegean Workshop on Computing: VLSI, Algorithms and Architectures. – AWOC'88, Corfu, Greece. – 1988. – Lect. Notes in Comput. Sci.– 1988. – *319*.– C. 33 – 42.

273. *Ramachandran V., Reif J.* An optimal parallel algorithm for graph planarity // 30th Annu. Symp. on Found. of Comput. Sci. – 1989. – C. 282 – 287.

274. *Ranade A.G.* How to emulate shared memory // J. Comput. and Syst. Sci. – 1991. – *42*, N 3. – C. 307 – 326.

275. *Reif J.H.* Logarithmic depth circuits for algebraic functions// SIAM J. Comput. - 1986. - *15*, N 1. - C. 231 - 242.

276. *Reif J.H.* An optimal parallel algorithm for integer sorting // Proc. 26th Annu. Symp. on Found. of Comput. Sci. - New York. - 1985. - C. 496 - 504.

277. *Reif J.H., Valiant L.G.* A logarithmic time sort for linear size networks // J. Assoc. Comput. Mach. - 1987.- *34*, C. 60 - 76.

278. *Reif J.H., Sen S.* Polling: a new randomized sampling technique for computational geometry // Proc. Annu. Symp. on Theory of Comput. - 1989. - C. 394-404.

279. *Reif J.H., Tate S.R.* Optimal size integer division circuits // SIAM J. on Computing. - 1990. - *19*. - C. - 912 - 924.

280. *Reif J. H.* Depth-first search is inherently sequential// Inf. Process. Lett. - 1985, - *20*, N 5. - C. 229 - 234.

281. *Romani F.* Shortest path problem is not harder than matrix multiplication // Inf. Process. Lett. - 1980. - *11*. - C. 134 - 136.

282. *Rose D.J.* Matrix identities of the FFT// Linear Algebra and its Appl. - 1980. - *29*, N 1. - C. 423 - 443.

283. *Ruzzo W.L.* On uniform circuit complexity// J. Comput. and Syst. Sci. - 1981. - *22*, N 3. - C. 365 - 383.

284. *Savage C., Ja'Ja' J.* Fast efficient parallel algorithms for some graph problems // SIAM J. Comput. - 1981.- *10*, N 4, C. 682 - 691.

285. *Schieber B., Vishkin U.* Finding all nearest neighbors for convex polygons in parallel: a new lower bound technique and a matching algorithm // Discret. Appl. Math. - 1990. - *29*. - C. 97 - 111.

286. *Schnorr C.P.* The network complexity and the Turing machine complexity of finite functions // Acta Inform. - 1976. - 7, C. 95 - 107.

287. *Schrijver A.* Theory of linear and integer programming.- John Wiley and sons. - Great Britain. - 1986.

288. *Schwartz J.T.* Fast probabilistic algorithms for verification of polynomial identities // J. Assoc. Comput. Mach. - 1980. - *27*, C. 701 - 717.

289. *Schwartzkopf O.* Parallel computation of discrete Voronoi diagrams // Lect. Notes in Comput. Sci. – 1989. – *349*, C. 193 – 204.

290. *Serna M.* Approximating linear programming is log-space complete for P // Inf. Process. Lett.– 1991. – *37*, C. 233 – 236.

291. *Siegel H.J.* A model of SIMD machines and a comparison of various interconnection networks // IEEE Trans. Comput.– 1979. – *C-28*, C. 907 – 917.

292. *Shankar N., Ramachandran V.* Efficient parallel circuits and algorithms for division // Inf. Process. Lett. – 1988.– *29*, N 6. – C. 307 – 313.

293. *Shiloach Y., Vishkin U.* Finding the maximum, merging and sorting in a parallel computation model // J. Algorithms. – 1981. – *2*. – C. 88 – 102.

294. *Smith J.R.* Parallel algorithms for depth-first searches. 1. Planar graphs// SIAM J. Comput. – 1986. – *15*, N 3. – C. 814 – 830.

295. *Snir M.* On parallel searching // SIAM J. Comput. – 1985. – *14*, N 2. – C. 688 – 709.

296. *Snir M., Barak A.* A direct approach to the parallel evaluation of rational expressions with a small number of processors // IEEE Trans. Comput. – 1977. – *C-26*. – C. 933 937.

297. *Soroker D.* Fast parallel algorithms for finding hamiltonian paths and cycles in tournaments // J. Algorithms. – 1988.– *9*, C. 276 – 286.

298. *Soroker D.* Optimal parallel construction of prescribed tournaments// Discrete Appl. Math. – 1990.– *29*, N 1, C. 113 – 126.

299. *Spencer J.* Ten lectures on the probabilistic method. – SIAM. – Philadelphia. – 1987.

300. *Springsteel F., Stojmenovic I.* Parallel general prefix computations with geometric, algebraic, and other applications// Int. J. of Parallel Program. – 1989. – *18*, N 6. – C. 485 – 503.

301. *Stockmeyer L., Vishkin U.* Simulation of parallel random access machines by circuits// SIAM J. Comput. – 1984. – *13*, N 2, C. 409 – 422.

302. *Strassen V.* Vermeidung von divisionen // J. Reine Angew Math. – 1973. – *264*. – C. 184 – 202.

303. *Tarjan R. E., Vishkin U.* Finding biconnected components and computing tree functions in logarithmic time // Proc.25th Ann. Symp. on Found. of Comput. Sci. - 1984. - C. 12 - 20.

304. *Tarjan R. E., Vishkin U.* An efficient parallel biconnectivity algorithm // SIAM J. Comput. - 1985.- *14*, N 4. - C. 862 - 874.

305. *Teng C.-Y., Das S.K.* Parallel algorithms for level-order traversals of general trees// J. of Combinatorics, Inform. and Syst. Sci. - 1989. - *14, N 2 - 3.* - C. *135 - 162.*

306. *Teng Shang-Hua The* construction of Huffman-equivalent prefix code in NC// SIGACT News. - 1987. - *18*, N 4. - C. 54 - 61.

307. *Tsin Y.H., Chin F.J.* Efficient parallel algorithms for a class of graph theoretic problems // SIAM J. Comput. - 1984.- *13*, N 3. - C. 580 - 599.

308. *Umeo H.* A class of SIMD machines simulated by systolic VLSI arrays // VLSI algorithms and architectures. - ed. P. Bertolazzi, F. Luccio. - Elsevier. - 1985. - C. 39 - 48.

309. *Upfal E.* An O(log n) deterministic packet routing scheme// Proc. 21st Annu. Symp. on Theory of Computing - 1989. - C. 241 - 250.

310. *Upfal E.* A probabilistic relation between desirable and feasible models of parallel computation // Proc. 16th Annu. Symp. on Theory of Comput. - 1984. - C. 258 - 265.

311. *Upfal E., Wigderson A.* How to share memory in a distributed system // J. Assoc. Comput. Mach. - 1987. - *34.* - C. 116 - 127.

312. *Vaidya P.M.* Reducing the parallel complexity of certain linear programming problems (extended abstract) // Proc. 31th Annu. Symp. on Found. of Comput. Sci. - 1990. - C. 583 - 589.

313. *Valiant L.G.* The complexity of computing the permanent // Theor. Comput. Sci. - 1979. - *8*, C. 189 - 201.

[Va80] *Valiant L.G.* Computing multivariate polynomials in parallel // Inf. Process. Lett. - 1980. - *11*, N 1. - C. 44 -45.

314. *Valiant L.G.* Computing multivariate polynomials in parallel // Inf. Process. Lett. - 1980. - *11*, N 1. - C. 44 - 45.

315. *Valiant L.G.* Universality considerations in VLSI circuits // IEEE Trans. Comput. - 1980. - *30*, N 2. -

C. 135 – 140.

316. *Valiant L.G.* General purpose architectures // In Handbook of Theoretical Computer Science. – J. van Leeuwen, ed. – v. A. – Amsterdam: – Noth–Holland. – 1990. – C. 943 – 972.

317. *Valiant L.G.* A scheme for fast parallel communication/ SIAM J. Comput. – 1982. – *11*, C. 350 – 361.

318. *Valiant L.G.* Parallelism in comparison problems // SIAM J. Comput. – 1975. – *4*, N 3. – C. 348 – 355.

319. *Valiant L.G.* Reducibility by algebraic projections// Enseign. Math. – 1982. – *28*, N 2. – C. 253 – 268.

320. *Valiant L.G., Skyum S., Berkowitz S., Rackoff C.* Fast parallel computation of polynomials using few processors // SIAM J. Comput. – 1983. – *12*. C. 641 – 644.

321. *Vazirani V.V.* NC algorithms for computing the number of perfect matchings in $K_{3,3}$–free graphs and related problems // Inf. and Comput. – 1989. – *80*. – C. 152 – 164.

322. *Vazirani U.V., Vazirani V.V.* Two processor scheduling problem is in RNC // Proc. 17th Annu. Symp. on Theory of Comput. – 1985.– C. 11 – 21.

323. *Vazirani U.V., Vazirani V.V.* Efficient and secure pseudo–random number generation// Proc. 25th Annu. Symp. on Found. on Comput. Sci. – 1984. – C. 458 – 463.

324. *Vishkin U.* Optimal parallel pattern matching in strings // Inform. and Control. – 1985.– *67*, C. 91 – 113.

325. *Vishkin U.* Implementation of simulations memory access in models that forbid it // J. Algorithms. – 1983. – *4*, N 1, C. 45 – 50.

326. *Vishkin U.* On efficient parallel strong orientation // Inf. Process. Lett. – 1985.– *20*, C. 235 – 240.

327. *Vishkin U.* An optimal parallel algorithm for selection // Advances in Comput. Res. – 1987. – *4*, C. 79 – 86.

328. *Vitter J.S., Simons R.A.* New classes for parallel complexity: a study of unification and other complete problems for P // IEEE Trans. on Computers, 1986. – *35*, C. 403 – 418

329. *Wagner A.* Embedding arbitrary binary trees in a hypercube // J. Par. and Distr. Comput. – 1989. – *7*. – C. 503 – 520.

330. *Wagner A., Corneil D.G.* Embedding trees in hypercube is NP–complete // SIAM Computing. – 1990. – N. 3, C. 570 – 590.

331. *Waksman A.* A permutation network // J. Assoc. Comput. Mach. – 1968. – *15*, C. 637 – 647.

332. *Wegener I.* The complexity of Boolean functions // Wiley. – Wiley-Teubner Ser. in Comput. Sci. – Stutgart. – 1987.– 457 C.

333. *Wegener I.* Efficient simulation of circuits by EREW PRAMs // Inf. Process. Lett. – 1990. – *35*, N 2. – C. 99 – 102.

334. *Zalkstein Y., Garzon M.* An NC^2 algorithm for testing simularity of matrices // Inf. Process. Lett. – 1989. – *30*, C. 253 – 254.

Содержание

УДК 519.68
Н.Н.Кузюрин, М.А.Фрумкин Параллельные вычисления: теория и алгоритмы. "Вычислительные науки, т. 8 (Итоги науки и техн. ВИНИТИ АН СССР)". М., 1991, С. 1 - 211.

Рассмотрены модели параллельных вычислений и соотношения между ними. Описан ряд общих методов построения параллельных алгоритмов. Приводятся параллельные алгоритмы из различных предметных областей: алгебры, арифметики, теории рядов и многочленов, комбинаторики и дискретной оптимизации, теории графов, вычислительной геометрии, сортировки и поиска, теории расписаний. Библ. 334.

Технический редактор Л.А. Киселева

Сдано в набор 17.12.91		В печать 13.12.91
Формат 60x90 1/16		Печать офсетная
Усл. печ. л. 13,25	Усл. кр.-отт. 13,44	Уч.-изд. л. 12,35
Тир. 420 экз.	Зак. 9137	Цена 4р.40к.

Адрес редакции: 125219, Москва, ул. Усиевича, д. 20а
Тел. 155-43-38
Производственно-издательский комбинат ВИНИТИ
140010, Люберцы 10, Московской обл.,
Октябрьский проспект, 403

4 р. 40 к. Индекс 56875

ISSN 0236—3127. ИНТ. Вычислительные науки. Т. 8. 1991. 1—212