1. Методы и алгоритмы параллельного обхода дерева

1.1. Методы обхода дерева

Деревом с корнем (далее деревом) называется связный ациклический неориентированный граф с одной выделенной вершиной, называемой *корнем*. Пусть v – произвольная вершина некоторого дерева. Существует единственный элементарный путь (далее путь), соединяющий корень дерева с вершиной v. Все вершины, находящиеся на этом пути и отличные от v, являются *предками* вершины v. Если вершина q является предком вершины v, то v называется *потомком* q. Если, кроме того, q и v смежные, то v называется *непосредственным потомком* q. Число непосредственных потомков вершины v называется ее *степенью ветвления*. Вершина с нулевой степенью ветвления называется *листом дерева*, или *концевой вершиной*. Все вершины, соединенные с корнем путями длины t, образуют t-й ярус дерева. Для каждой вершины v можно рассмотреть дерево, состоящее из v и всех потомков v, в котором v считается корнем. Оно называется *поддеревом* с корнем v.

Во многих методах решения комбинаторных задач дерево используется в качестве модели пространства возможных решений, а его обход – как модель алгоритма поиска решения. Дерево, возникающее в комбинаторных алгоритмах, может иметь настолько большие размеры, что его невозможно (или неразумно) представить в компьютере целиком. По этой причине дерево обрабатывают поэлементно, моделируя процесс его построения (воссоздания) по определенным правилам, свойственным каждой конкретной задаче. Этот процесс и называется *обходом*. Выполнить обход дерева – значит, в некотором порядке воспроизвести все его узлы (вершины). Воспроизвести узел может означать на практике, например, следующее: обработать некоторую информацию, сопоставленную этому узлу, выполнить в нем некоторую процедуру, перечислить все непосредственные потомки узла, и т.п.

На однопроцессорном компьютере обход дерева выполняется последовательно – узлы дерева воспроизводятся один за другим по одному. В этом случае в зависимости от порядка выбора узлов различают следующие виды обхода – в глубину, в ширину, снизу вверх и симметричный обход для бинарных деревьев [12, 10]. Первые два находят свое применение во многих методах решения комбинаторных задач. Обход в глубину используется в методах полного перебора, поиска с возвращением, сокращенного обхода дерева; обход в ширину – в методе ветвей и границ.

При обходе в глубину в каждом поддереве дерева сначала проходится корень, а затем его поддеревья. При обходе в ширину вначале проходится корень дерева – вершина нулевого яруса, затем все вершины первого яруса, после чего все вершины второго яруса и т.д., так, что никакая вершина некоторого яруса не может быть пройдена, пока не будут пройдены все вершины предыдущего яруса.

В алгоритме обхода в глубину для упорядочивания обхода используется стек *S*, и узлы дерева воспроизводятся по мере их извлечения из стека.

Будем полагать, что непосредственные потомки каждой вершины упорядочены некоторым образом, например, при графическом изображении – слева направо. Тогда в алгоритме 1.1 – алгоритме *левого* обхода дерева в глубину, чтобы поддеревья проходить слева направо, их корневые вершины помещают в стек в обратном порядке, т.е. справа налево.

Алгоритм 1.1

1. Занести в стек *S* корень дерева.
2. Если стек *S* пуст, то п. 3; в противном случае:
    1) извлечь из стека *S* узел *p*;
    2) воспроизвести узел *p*;
    3) поместить в стек *S* непосредственные потомки узла *p* в порядке справа налево и п. 2.
3. Обход завершен.

Для организации обхода в ширину подобным же образом используется очередь. В следующем алгоритме 1.2 обхода дерева в ширину непосредственные потомки каждой вершины поступают в очередь в порядке слева направо.

Алгоритм 1.2

1. Занести в очередь Q корень дерева.
2. Если очередь пуста, то п. 3; в противном случае:

    1) извлечь из очереди Q узел p;

    2) воспроизвести узел p;

    3) поместить в очередь Q непосредственные потомки узла p в порядке слева направо и п. 2.
3. Обход завершен.

Число узлов в очереди достигает ширины дерева, что существенно ограничивает применение обхода в ширину для больших деревьев. Однако данный алгоритм может быть использован для обхода части дерева (нескольких верхних ярусов) и построения списка, содержащего заданное число вершин. В результате все вершины, прошедшие через очередь, оказываются пройденными, и вершины, оставшиеся в очереди, принадлежат последнему пройденному и следующему за ним ярусам (в соответствии с порядком обхода в ширину). Допускается, что число вершин в очереди немного превышает заданное число вершин m – с целью исключения ситуации, когда в очередь попадают не все непосредственные потомки некоторой вершины. Описанные действия выполняет

Алгоритм 1.3

1. Занести в очередь Q корень дерева, $j := 1$ (текущее число вершин в очереди).
2. Если очередь пуста или $j \geq m$, то п. 3, в противном случае:

    1) извлечь из очереди Q узел p, $j := j - 1$;

2) воспроизвести узел p;

3) поместить в очередь Q непосредственные потомки узла p в порядке слева направо;

4) $j := j +$ «число непосредственных потомков узла p»;

5) перейти в пункт 2.

3. Обход завершен.

Для дерева, представленного на рис. 1.1, алгоритм 1.1 воспроизводит узлы в порядке: 1, 2, 4, 7, 5, 6, 8, 9, 3; алгоритм 1.2 – в порядке 1, 2, 3, 4, 5, 6, 7, 8, 9; алгоритм 1.3 при $m = 3$ воспроизведет узлы 1, 2 и в очереди (построенном списке) останутся узлы 3, 4, 5, 6.

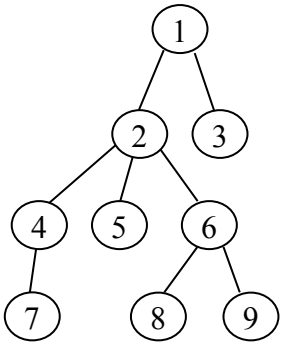

Рис.1.1. Дерево из 9 узлов

1.2. Общие методы параллельного обхода

Применение многопроцессорной вычислительной системы позволяет обход дерева *распараллелить* – воспроизводить одновременно несколько узлов в дереве. Основная проблема, возникающая на этом пути, состоит в таком динамическом распределении неявно заданного дерева между процессорами, при котором достигает минимума время обхода, или максимизируется коэффициент эффективности использования системы. Для ее решения необходимо минимизировать число и объем обменов данными между процессорами системы и обеспечить равномерную загрузку процессоров.

Предлагаются два метода параллельного обхода дерева в глубину – метод назначаемых поддеревьев и метод выделяемых поддеревьев. В первом методе

упор делается на сокращение взаимодействия между процессорами, во втором – на равномерную загрузку процессоров системы.

1.2.1. Обзор методов параллельного обхода в глубину

Классификация существующих методов распараллеливания (построения параллельных алгоритмов) обхода дерева поиска представлена схематически на рис. 1.2. Она составлена на основе обзоров [25, 26] и других публикаций, охватывающих более 20 работ, в основном, зарубежных авторов за последние 25 лет исследований в данной области. В нее укладываются и результаты диссертанта, на схеме их место обозначено курсивом. Прокомментируем данную классификацию, не поясняя терминов, общепринятых в теории параллельных вычислений [4, 7, 14]. В этом комментарии будут ссылки и на некоторые работы [21, 23, 33, 35, 38], информация о которых получена из обзоров [25, 26]. Для удобства изложения далее мы будем говорить о системе параллельных процессов, где каждый процесс есть программа, выполняемая на отдельном процессоре и выполняющая действия, предписанные алгоритмом для данного процесса.

Как видно из схемы, все методы параллельного обхода дерева поиска в глубину классифицируются по способам разбиения дерева на поддеревья (задачи на подзадачи) и распределения последних между процессами. В зависимости от решаемой задачи и архитектуры вычислительной системы эффективнее оказываются те или иные способы.

Статический подход к разбиению дерева может использоваться только для деревьев со строго определенной структурой, что не свойственно большинству комбинаторных задач. В работе [18] этот подход используется для обхода в глубину дерева разбиений. Поддеревья распределяются по процессорам в соответствии с их номерами. Особенностью большинства комбинаторных задач являются как раз нерегулярность и непредсказуемость структуры дерева, что для обеспечения высокой эффективности требует динамического разбиения дерева.

Методы динамической балансировки загрузки процессоров отличаются друг от друга тем, кто разбивает дерево на поддеревья и кто распределяет поддеревья. Если какой-либо из процессов завершает обход назначенной ему части дерева раньше других, то он становится *свободным* и должен получить, став *приемником*, часть работы какого-либо занятого процесса, называемого в данном случае *источником*.

В литературе представлены методы, в которых: 1) разбиение и передача инициируются источником [33, 38, 39]; 2) разбиение инициируется источником, а передача выполняется по запросу свободного процесса [20, 23]; 3) разбиение и передача выполняются при запросе свободного процесса [24, 31, 36, 37].

В тех методах, где разбиение и передача инициируются источником, процессы-источники генерируют и рассылают подзадачи независимо от того, в каком состоянии находятся другие процессы. При обходе дерева порождение потомков подразумевает выполнение ряда полезных для решения задачи действий. Если их число в каждой вершине велико, то, наверное, оправданным будет подход, описанный в работе [38]. Там осуществляется назначение генерируемых потомков случайно выбранным процессам, что в некоторой степени обеспечивает равномерную загрузку процессов. Однако такая схема сталкивается с рядом технических трудностей при реализации. Коммуникации возникают при каждом ветвлении, что перегружает сеть. Отметим, что число вершин на каждом процессоре может расти неограниченно, что приводит к ограничениям по памяти.

В работе [33] представлен вариант описанной выше схемы для сетей с архитектурой butterfly или гиперкуба. Описана динамическая схема, по которой порождаемые вершины бинарного дерева распределяются по соседним процессорам. Показано, что с высокой степенью вероятности время, требуемое для параллельного обхода дерева с N вершинами на s процессорах, равно $O(N/s + \log s)$. Эта схема локализует все пересылки и поэтому оказывается более эффективной по сравнению с представленной в [38]. Введение ограничения

типа «не передавать вершины, расположенные выше заданного граничного яруса» позволяет повысить эффективность [31]. Все вершины, расположенные выше граничного яруса, проходятся породившим их процессом, а это снижает объем коммуникаций.

Рис. 1.2. Классификация подходов и методов параллельного обхода дерева в глубину

К третьей (самой распространенной) группе относятся методы, в которых и разбиение и передача подзадач выполняется по запросам от свободных процессов. Эти методы представлены в работах [20, 21, 22, 25, 28, 37]. Опишем общую схему. Каждый из процессов выполняет обход в глубину назначенного ему поддерева, при этом периодически проверяет наличие запросов от других процессов. Когда процесс завершает обход назначенного ему поддерева, он становится свободным и выбирает для себя процесс, которому посылает запрос на работу. Получив запрос, процесс-отправитель либо посылает отказ, либо выполняет разбиение оставшейся части работы на две и посылает часть свободному процессу.

Методы последней группы различаются по способам выбора процесса, подзадачи, начального распределения подзадач, условий посылки отказа.

Описано несколько стратегий выбора процесса, к которому направляется запрос. В стратегии случайного голосования (random polling) [20, 22, 27, 36, 37] выбор осуществляется случайным образом. Если в ответ от выбранного процесса приходит отказ, то случайный выбор повторяется. Минусом этой стратегии является большое число отказов. В стратегии глобальной кольцевой очереди [29] имеется глобальный указатель на процесс, к которому надо направлять запрос. После запроса указатель – переменная, содержащая номер процесса, переходит к следующему (увеличивается по модулю числа процессов). В [29] показано, что использование глобального указателя позволяет значительно уменьшить число запросов для большого числа видов деревьев, однако при большом числе процессов возникает конкуренция за доступ к глобальному указателю.

В ответ на запрос освободившегося процесса отправитель должен выделить часть своей невыполненной работы (не пройденную часть дерева), чтобы передать свободному процессу. Встречаются две стратегии выделения работы: выделяется одно поддерево, путем исключения одной вершины из стека (node splitting) [21, 22]; выделяется несколько поддеревьев, путем

расщепления стека на два (stack splitting) [28, 29, 31]. Во втором случае свободный процесс получает стек вершин. Стек расщепляется так, чтобы как в стеке свободного процесса, так и в стеке источника оказалось примерно одинаковое число вершин и в каждом стеке были бы вершины разных ярусов.

Начальное распределение подзадач между процессами можно организовать по-разному. В некоторых методах изначально все дерево назначается одному процессу, а остальные являются свободными. Затем дерево постепенно динамически распределяется между процессами [31, 36, 37]. Этот подход характеризуется большим числом коммуникаций уже на начальном этапе. Каждый из процессов может генерировать для себя первую задачу по определенным правилам в зависимости от его номера [18].

Условия посылки отказа определяются скорее не методом параллельного исполнения поиска, но задачей и алгоритмом, к которому он применяется. Одно из самых простых условий связано с числом уже полностью пройденных ярусов в дереве источника, а именно: если предположительное число оставшихся ярусов очень мало (насколько мало, зависит от размерностей задачи и характеристик используемой системы), то высылается отказ.

С помощью описанных выше методов разными авторами построены алгоритмы решения таких задач как: задача коммивояжера, тур коня, пятнашки [6], проверка тавтологии, автоматическое генерирование тестов, задача о рюкзаке в оптимизационной постановке, задача о выполнимости КНФ [40] и некоторые др. Алгоритмы исследовались на многопроцессорных системах различных типов, и почти для всех задач тем или иным методом достигалось линейное ускорение.

В заключение обзора остановимся на некоторых особенностях поисковых комбинаторных задач, которые приходится учитывать при разработке методов параллельного обхода дерева поиска их решений.

Обычно требуется найти хотя бы одно решение задачи поиска. В этом случае во всех методах параллельный поиск прекращается, как только какой-

нибудь из процессов находит решение. В задачах такого типа наиболее заметен эффект несовпадения просматриваемых областей поиска для параллельного и последовательного алгоритмов и, как следствие, выполнение работы разного объема. В параллельном случае он может оказаться как меньше, так и больше, чем в последовательном. *Фактор избыточности поиска R* определяется как отношение объема работы (числа просмотренных вершин) последовательного алгоритма к объему работы параллельного. Верхняя граница ускорения для *s* процессов тогда есть *sR*. В некоторых случаях фактор избыточности может оказаться меньше единицы, что приводит к суперлинейному ускорению. Если фактор избыточности поиска в среднем меньше единицы, это означает, что решения в дереве поиска распределены неравномерно или что выбранный последовательный алгоритм не является лучшим для этой задачи.

В работе [32] проведен анализ этой проблемы. Предлагаются абстрактные модели пространства поиска, просматриваемого алгоритмами параллельного поиска в глубину двух типов: простого и с упорядочиванием потомков. Анализируется и сравнивается среднее число узлов просматриваемых последовательным и параллельным алгоритмами. Для простого поиска доказывается, что если распределение решений по областям поиска разных процессоров одинаково, то области поиска последовательного и параллельного алгоритмов примерно одинаковы, и если распределения решений по областям поиска разных процессоров различаются, то область поиска последовательного алгоритма больше области поиска параллельного. Для упорядоченного поиска доказывается, что число узлов, просматриваемых параллельным алгоритмом, не больше числа узлов, просматриваемых последовательным алгоритмом.

1.2.2. Метод назначаемых поддеревьев

Идея данного метода заключается в разбиении дерева на большое число «маленьких» поддеревьев и назначении их для обхода в порядке освобождения процессов. С этой целью в системе из $s+1$ параллельных процессов выделяется

управляющий процесс, функции которого отличаются от функций остальных участвующих в алгоритме *рабочих* процессов.

На первом этапе управляющий процесс выполняет построение и обход части дерева в ширину, начиная от его корня по алгоритму 1.3. Обход дерева ведется до тех пор, пока число вершин в очереди не достигнет заранее заданного числа m, которое *много больше* s. Вершины, попавшие в очередь, образуют множество корневых вершин поддеревьев, причем их родительские вершины уже пройдены. Число m определяется в зависимости от размерности задачи и числа процессов.

На втором этапе вершины из очереди управляющим процессом последовательно посылаются рабочим процессам по мере выполнения теми обходов ранее назначенных им поддеревьев, которые могут быть разного размера. Каждый процесс выполняет последовательный обход в глубину (алгоритм 1.1) назначенного ему поддерева.

Отметим, что в то время, когда один из процессов выполняет обход одного поддерева, другой процесс может выполнить обход нескольких поддеревьев. Если очередь пуста, то процесс, завершивший обход очередного поддерева, останавливается, в то время как другие могут продолжать работу. В данном алгоритме управление работой процессов не требует значительных затрат. Рабочие процессы обмениваются данными с управляющим процессом лишь при назначении очередного поддерева и завершении его обхода. Таким образом, не происходит никаких пересылок сообщений в процессе обхода назначенного поддерева, за исключением случаев решения задач оптимального поиска. Однако существует опасность, что некоторые процессы получат деревья, во много раз превышающие другие по размеру, и в то время, когда они будут продолжать их обход, остальные деревья будут пройдены, и часть процессов будет простаивать. Другими словами, уменьшается загруженность процессоров. Эффективность распараллеливания с помощью данного метода зависит от конкретных исходных данных, определяющих вид дерева. Для

повышения эффективности можно увеличить число m назначаемых поддеревьев, тогда большие по размеру деревья, возможно, разделятся на две или более частей. С другой стороны, это может привести к слишком частым обращениям к управляющему процессу, вследствие чего рабочие процессы будут простаивать в очереди, ожидая, пока управляющий процесс обслужит предыдущие запросы.

Ниже поведение каждого из процессов в алгоритме описывается в терминах состояний, переходов между состояниями и операций процесса в каждом состоянии, выполняемых в зависимости от управляющих сигналов, которые могут поступать извне алгоритма или со стороны самих процессов, сигнализируя о наступлении того или иного события по ходу выполнения алгоритма. Управляющие сигналы, воспринимаемые или вырабатываемые процессом, называются его соответственно *входными* и *выходными* сигналами. Управляющий сигнал, являющийся одновременно входным и выходным, называется *внутренним* сигналом данного процесса.

Поведение управляющего процесса описывается диаграммой переходов (рис. 1.3).

H_0 – начальное состояние.

H_1 – переход в это состояние выполняется при построении очереди из m корневых вершин поддеревьев, то есть при завершении процедуры обхода в ширину.

H_2 – состояние ожидания; первый раз переход в это состояние выполняется после передачи первых корневых вершин рабочим процессам. В этом состоянии управляющий процесс ожидает сообщений о завершении обхода от рабочих процессов.

H_3 – переход в это состояние выполняется при приеме сообщения о завершении обхода одним из процессов.

H_4 – в это состояние управляющий процесс переходит при завершении обхода всего дерева.

Входные сигналы: M – получение числа m (числа требуемых поддеревьев), E_k – приход сообщения о завершении обхода процессом с номером k.

Выходные сигналы: S – передача первых s корневых вершин рабочим процессам, R_k – передача очередной корневой вершины процессу с номером k, F_k – передача сообщения процессу с номером k о завершении работы.

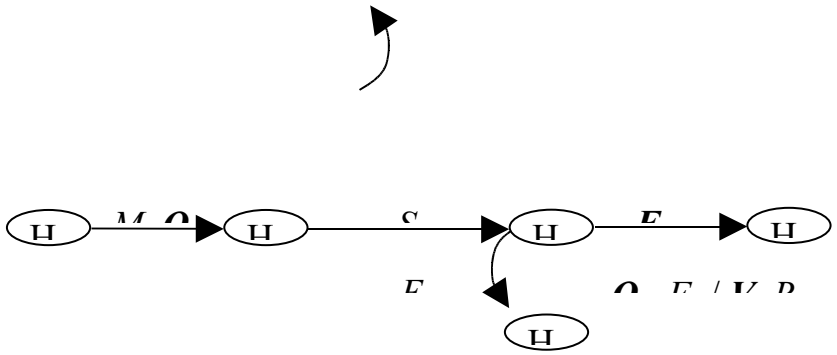

Рис. 1.3. Диаграмма переходов управляющего процесса в методе назначаемых поддеревьев

Внутренние сигналы: Q_m – создание очереди из m корневых вершин, F – завершение всех процессов, V – извлечение очередной вершины из очереди, Q_0 – пустая очередь.

Поведение всех рабочих процессов описывается одной диаграммой переходов, представленной на рис. 1.4.

S_0 – начальное состояние.

S_1 – переход в это состояние первый раз выполняется при получении первой корневой вершины по сигналу S и затем всякий раз при получении очередной корневой вершины по сигналу R_k от управляющего процесса. В этом состоянии выполняется обход в глубину выделенного поддерева.

S_2 – состояние ожидания; переход в это состояние выполняется при завершении обхода очередного поддерева. Процесс ожидает сообщения от управляющего процесса.

S_3 – конечное состояние; переход в него выполняется при приеме сообщения о завершении работы.

Входные сигналы: S, R_k – приход сообщения с корневой вершиной, F_k – приход сообщения о завершении работы.

Внутренние сигналы: E – завершение обхода поддерева.

Выходные сигналы: E_k – посылка сообщения о завершении обхода.

События, сопоставленные выходным сигналам управляющего процесса, инициируют события, сопоставленные входным сигналам рабочих процессов, и наоборот. Все процессы выполняются асинхронно. Из диаграмм видно, что взаимодействие осуществляется лишь между управляющим и рабочими процессами. Данный метод легко применим для конкретных задач и рекомендуется для использования в тех случаях, когда можно получить поддеревья, близкие по размеру.

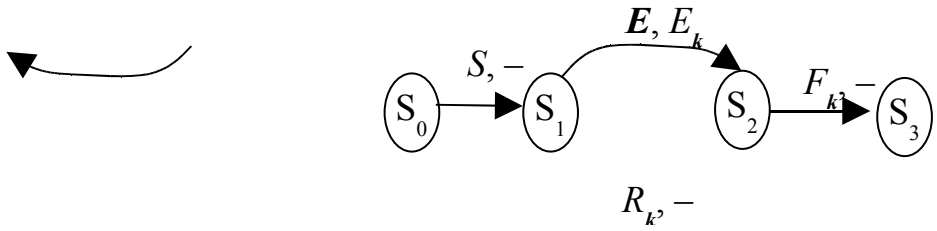

Рис. 1.4. Диаграмма переходов рабочего процесса с номером k в методе назначаемых поддеревьев

При программной реализации алгоритмов в технологии MPI действия всех процессов описываются в одной программе, копии которой рассылаются на узлы кластерной системы. Процессы идентифицируются по их номерам, и выполняют лишь те части программы, которые им предназначены. Время работы программы, реализующей какой-либо параллельный алгоритм, определяется как разность между моментом, когда какой-либо из процессов начинает выполнять предписанные ему алгоритмом действия, и моментом, когда все процессы завершат свои действия. Программа, реализующая метод назначаемых поддеревьев, содержит два последовательных этапа. Время 1-го этапа складывается из времени, затрачиваемого управляющим процессом на рассылку исходных данных и генерирование корневых вершин назначаемых

поддеревьев. На втором этапе часть времени уходит на выполнение полезной работы – обход поддеревьев, часть – на обмен данными между управляющим и рабочими процессами. Оценим минимальное время работы такой программы. Введем обозначения: T_Q – время выполнения первого этапа, оно зависит от числа m назначаемых поддеревьев и сложности построения одной корневой вершины; T_0 – время обхода всего дерева последовательным алгоритмом; T_m – время, требуемое на запрос, передачу и прием m корневых вершин. На втором этапе управляющий процесс завершается, отдав последнее поддерево, либо получив результаты работы последнего из рабочих процессов. В любом случае будем считать, что он завершается не позже рабочих процессов. Предположим, что дерево разбито на поддеревья так, что все рабочие процессы выполняют одинаковый объем полезной работы, т.е. посещают одинаковое число узлов. В этом идеальном случае время, затрачиваемое одновременно выполняющимися s рабочими процессами на полезную работу, равно T_0/s. Предположим также, что всем рабочим процессам в итоге назначается одинаковое число поддеревьев. Это означает, что время, затрачиваемое каждым рабочим процессом на получение назначенных ему корневых вершин, равно T_m/s. Тогда время работы программы с s рабочими процессами равно $T_Q + T_m/s + T_0/s$. Следовательно, ускорение не превышает величины $\frac{T_0}{T_Q+T_m/s+T_0/s}$, и эффективность рабочих процессов E_s (здесь это, по сути, доля времени, затрачиваемого на полезную работу) не превышает величины $\frac{T_0}{sT_Q+T_m+T_0} < 1$.

Например, если $T_Q = T_m = T_0 \cdot 10^{-3}$, то $E_s \le 1 - \frac{s}{s+1000}$, откуда $E_1 \le 0{,}9980$, $E_{10} \le 0{,}989$, $E_{20} \le 0{,}979$, $E_{100} \le 0{,}908$, $E_{1000} \le 0{,}499$. Очевидно, что чем больше T_Q, тем больше и T_m, поэтому слишком большое число назначаемых поддеревьев может привести к снижению эффективности с ростом числа процессов. Однако на практике величины T_Q и T_m достаточно малы.

Время работы управляющего процесса есть $T_Q + T_m$, большую часть времени работы программы он простаивает, ожидая запросы. Если считать эффективность U_s, учитывая работу управляющего процесса и s рабочих процессов, то она не превышает величины $\frac{T_0}{\lceil s+1 \rceil (T_Q + T_m/s + T_0/s)}$. Для тех же значений $T_Q = T_m = T_0 \cdot 10^{-3}$ получаем, что $U_1 \le 0{,}4990$, $U_{10} \le 0{,}899$, $U_{20} \le 0{,}932$, $U_{100} \le 0{,}89$, $U_{1000} \le 0{,}49$. В начале с ростом s оценка величины U_s возрастает, так как уменьшается вклад низкой эффективности управляющего процесса (максимальная оценка достигается при $s = 31$), затем она начинает убывать.

1.2.3. Метод выделяемых поддеревьев

В основе этого метода лежит принцип динамического выделения поддеревьев, т.е. разбиение на подзадачи происходит не сразу, а по мере выполнения обхода. На первом этапе управляющий процесс выполняет обход дерева в ширину, начиная от его корня, до тех пор, пока число вершин в очереди не достигнет числа параллельных процессов s. Допускается, что число вершин в очереди может превышать число процессов – с целью исключения ситуации, когда в очередь попадают не все непосредственные потомки некоторой вершины.

На втором этапе корневые вершины распределяются по процессам, и каждый процесс выполняет обход в глубину соответствующего поддерева. По окончании обхода своего поддерева выполнявший его процесс освобождается. Для загрузки освободившегося процесса выбирается поддерево, обход которого в этот момент ведется одним из процессов, и в нем выделяется поддерево, которое передается для обхода освободившемуся процессу. При выборе поддерева проверяется некоторое *условие разбиения* дерева, показывающее целесообразность выделения в нем поддерева. Например, максимальный номер полностью пройденного яруса должен быть много меньше числа всех ярусов в дереве. Будем говорить, что дерево, выделенное какому-либо процессу, стало

неделимым, если для него при обходе перестало выполняться условие разбиения.

Выбор процесса-источника для выделения поддерева выполняется управляющим процессом по предлагаемой здесь стратегии *оптимального выбора*, нацеленной на то, чтобы направлять запрос к наиболее загруженному процессу.

Каждый процесс периодически посылает управляющему информацию о пройденной части поддерева, например, при увеличении максимального номера полностью пройденного яруса. Как только дерево некоторого процесса становится неделимым, он в последний раз посылает сигнал управляющему процессу, который исключает его из числа процессов – претендентов на дробление дерева. Эти посылки выполняются на фоне вычислений.

Получив сообщение об окончании обхода, управляющий на основании собранной им информации выбирает из рабочих процессов, для которых еще не

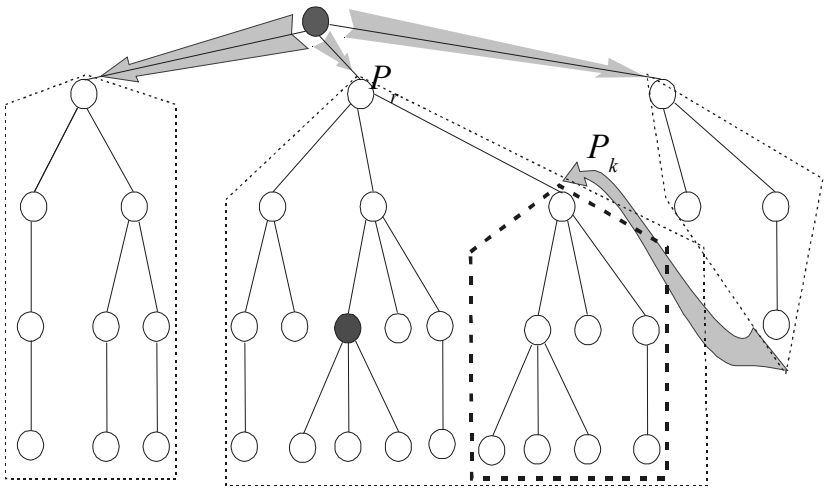

Рис. 1.5. Выделение и передача поддерева

нарушено условие разбиения, тот процесс, который имеет наибольшую непройденную часть поддерева, после чего пересылает ему номер освободившегося. Получив указанный номер, процесс-источник выделяет в своем дереве поддерево для свободного процесса, что требует от него некоторой задержки при обходе. Для выделения поддерева по возможности большего размера ищется первая непройденная вершина наиболее высокого

яруса. При обходе в глубину такая вершина находится на дне стека, и она является корнем непройденного поддерева. Эта вершина удаляется из стека, чем гарантируется, что данное поддерево не будет проходиться дважды, и передается свободному процессу в качестве корневой. Если у этой вершины есть непройденные соседние вершины, то первая из них заносится на ее место в стек. Перечисленные операции, а также проверка условия разбиения требуют дополнительного времени, что увеличивает время работы программы.

Если остались лишь неделимые деревья, то освободившийся процесс полностью завершается. Обход дерева заканчивается при завершении обхода всех поддеревьев. На рис. 1.5 схематично представлен механизм выделения поддерева для процесса P_k в дереве, обход которого выполняется процессом P_r при условии, что в данный момент процессом P_r проходится выделенная вершина.

Управляющий процесс в методе выделяемых поддеревьев выполняет действия, иллюстрируемые диаграммой на рис. 1.6.

H_0 – начальное состояние; в него процесс устанавливается при запуске программы; определив в этом состоянии число рабочих процессов, он строит множество корневых вершин, после чего вырабатывается внутренний сигнал R для перехода в следующее состояние.

H_1 – переход в это состояние выполняется после построения множества корневых вершин поддеревьев, т. е. сразу после завершения процедуры обхода в ширину.

H_2 – состояние ожидания; первый раз переход в это состояние выполняется после передачи корневых вершин рабочим процессам. В этом состоянии управляющий процесс ожидает от рабочих процессов сообщения двух типов: об изменении пройденной части поддерева и об окончании обхода поддерева.

H_3 – переход в это состояние выполняется при приеме сообщения о завершении обхода одним из рабочих процессов; условие разбиение для последнего считается нарушенным.

H_4 – переход в это состояние выполняется при завершении обхода всего дерева.

Входные сигналы: W – определение числа рабочих процессов; B_k – приход сообщения от процесса с номером k об увеличении пройденной части поддерева; E_k – приход сообщения о завершении обхода процессом с номером k, при этом процесс исключается из очереди претендентов на дробление.

Внутренние сигналы: R – построение множества корневых вершин поддеревьев, L – сигнал, показывающий, что выбран процесс-источник, L_0 – сигнал, показывающий, что поддеревья всех процессов стали неделимыми, F – завершение обхода всеми процессами.

Выходные сигналы: S – передача корневых вершин поддеревьев рабочим процессам, H_{kl} – передача сообщения процессу с номером l с запросом на выделение поддерева для освободившегося процесса с номером k, F_k – передача сообщения для всех процессов с требованием о завершении работы.

Поведение рабочего процесса с номером k представлено на диаграмме (рис. 1.7).

S_0 – начальное состояние; процесс устанавливается в начальное состояние при запуске.

S_1 – переход в это состояние выполняется при получении корневой вершины; в этом состоянии выполняется обход в глубину выделенного поддерева.

S_2 – выделение поддерева другому процессу.

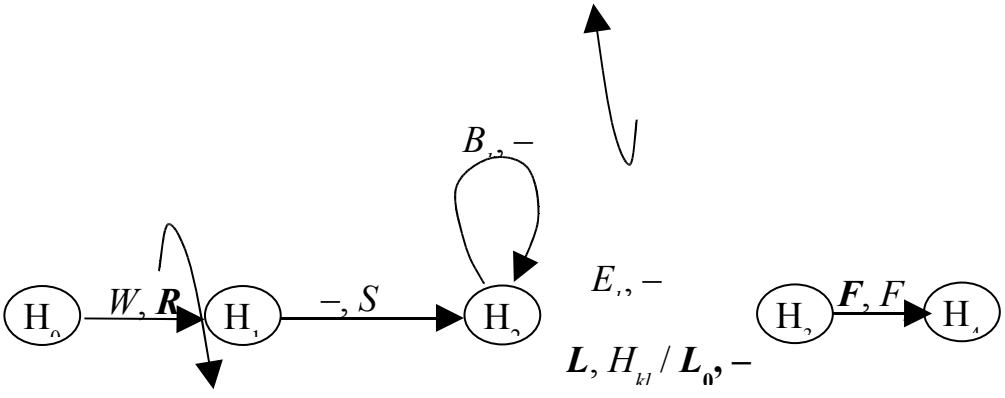

Рис. 1.6. Диаграмма переходов управляющего процесса в методе выделяемых поддеревьев

S_3 – переход в это состояние выполняется при нарушении условия разбиения.

S_4 – состояние ожидания; переход в это состояние выполняется при завершении обхода выделенного поддерева. Процесс ожидает сообщения от других рабочих процессов или от управляющего.

S_5 – переход в это состояние выполняется при получении запроса на выделение поддерева после нарушения условия разбиения, и соответственно окончания обхода. При выходе из этого состояния вырабатывается сигнал отказа (пустое поддерево) для процесса, пославшего запрос.

S_6 – конечное состояние; переход в это состояние выполняется при завершении всех процессов.

Пусть k – номер процесса.

Входные сигналы: S – получение корневой вершины первого поддерева, H_{kl} – прием сообщения с запросом на выделение поддерева для процесса с номером l, T_{lk} – прием сообщения с корнем выделенного поддерева от процесса с номером l, E_{lk} – прием сообщения с корнем пустого дерева от процесса с номером l, F_k – прием требования о завершении работы.

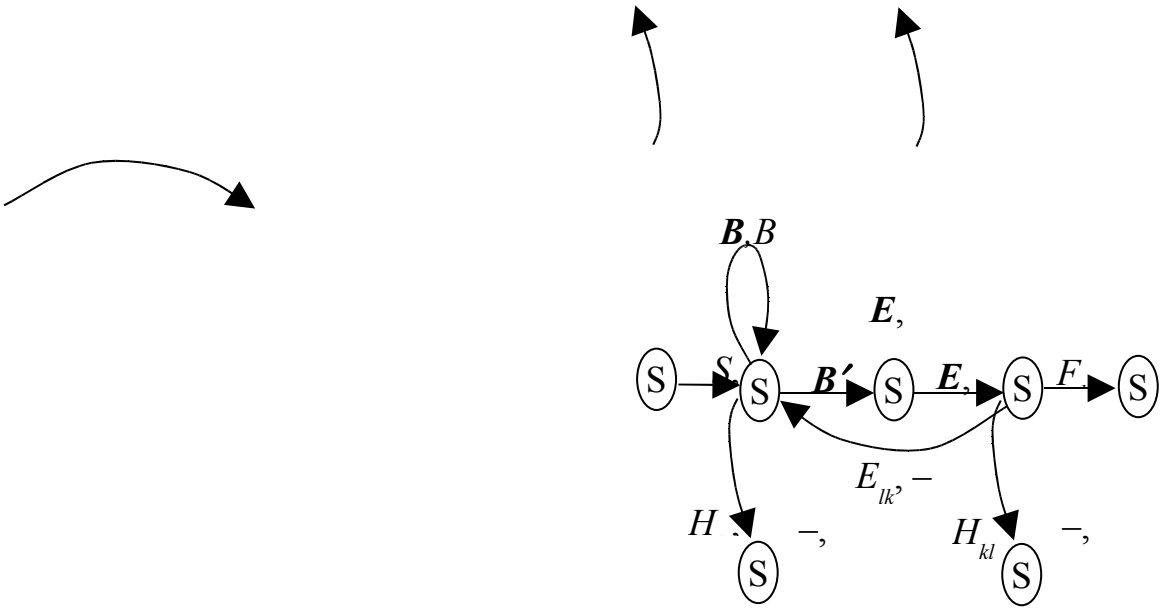

Рис. 1.7. Диаграмма переходов рабочего

Внутренние сигналы процесса: B – существенное увеличение пройденной части поддерева, B' – нарушение условия разбиения, E – завершение обхода выделенного поддерева.

Выходные сигналы k-го процесса: B_k – передача сообщений о начале обхода, об увеличении пройденной части дерева, о нарушении условия разбиения; E_k – передача сообщения о завершении обхода очередного поддерева, T_{kl} – передача сообщения с корнем выделенного поддерева для процесса с номером l, E_{kl} – передача сообщения с отказом (с корнем пустого дерева) для процесса с номером l, запрос от которого пришел после выхода процесса из состояния S_1.

1.2.4. Экспериментальное исследование методов параллельного обхода

Важным этапом при создании параллельных программ является построение экспериментальных оценок их ускорения и эффективности. На практике ускорение C_s определяется отношением времени T_0 решения задачи на одном процессоре (или на минимальном допустимом числе процессоров) ко времени решения той же задачи на системе из p таких же процессоров. Эффективность использования параллельным алгоритмом процессоров однородной вычислительной системы определяется отношением ускорения к числу процессоров. Эффективность показывает среднюю долю времени выполнения алгоритма, в течение которого процессоры реально используются для решения задачи. Параллельный алгоритм и его программная реализация являются масштабируемыми, если ускорение зависит линейно от количества используемых процессоров, т.е. $C_s = O(s)$. На практике программы, для которых $C_s = O(s/(\ln s))$, также считаются масштабируемыми.

Для программной реализации предлагаемых в работе параллельных алгоритмов используется язык C++ и функции библиотеки MPI (Message Passing Interface) [4, 11], представляющей собой стандартизованный набор средств для обмена сообщениями между процессорами. В основном

компьютерное моделирование алгоритмов проводилось на многопроцессорном вычислительном кластере Томского государственного университета, состоящем из 9 двухпроцессорных узлов.

Для экспериментального исследования эффективности предложенных методов на большом количестве примеров разработана программа генерации *полных k-ичных деревьев* (далее *k-ичных деревьев*) – деревьев, все листья которых находятся на одинаковой глубине, а все внутренние вершины имеют степень ветвления k [10]. Программа порождает их не целиком, но по мере и в порядке обхода рассматриваемым параллельным алгоритмом.

Графики на рис. 1.8 показывают зависимость ускорения для 10 процессов, включая управляющий, от размера обходимых двоичных и троичных деревьев. Для деревьев с небольшим числом узлов (меньше 10^6) ускорение как для метода выделяемых поддеревьев (МВП), так и для метода назначаемых поддеревьев (МНП) невелико, что объясняется большой долей затрат на организацию

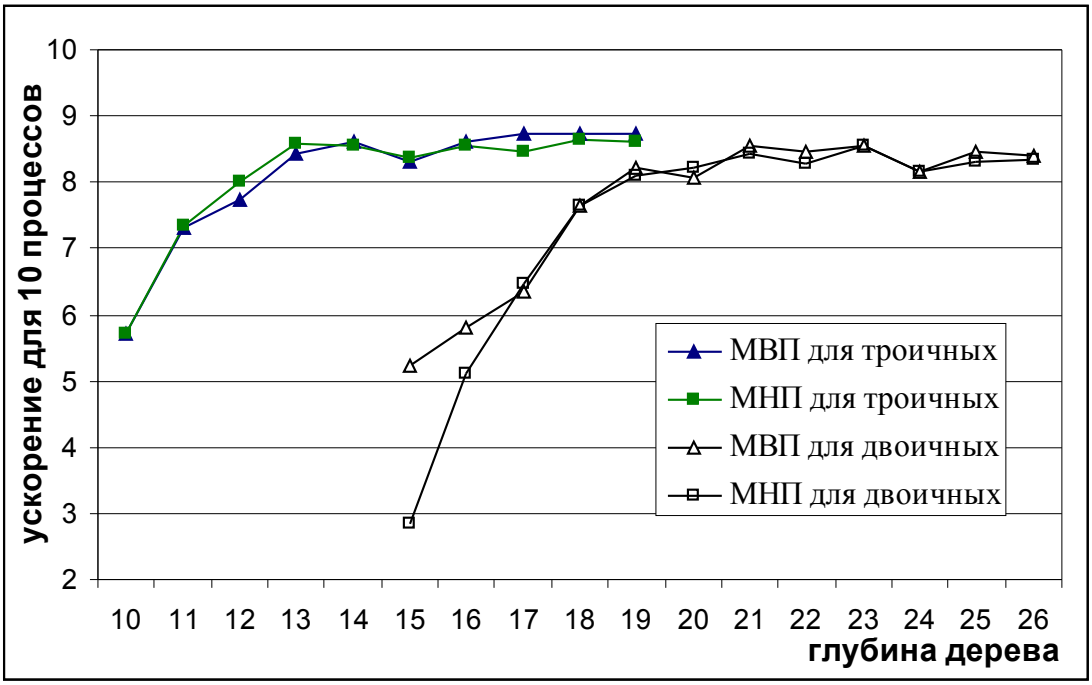

Рис. 1.8. Диаграмма зависимости ускорения от размера дерева при обходе двоичных и троичных деревьев

параллельных вычислений. С ростом размерности деревьев эта доля сокращается, и ускорение стабилизируется в районе 8,5.

На рис. 1.9 представлены графики, построенные по усредненным значениям ускорения для полных двоичных деревьев глубины от 25 до 28 и троичных деревьев глубины от 15 до 18. Графики показывают линейное ускорение, что позволяет надеяться на хорошую масштабируемость.

Аналогичные исследования проводились для ряда значений k в диапазоне от 4 до 25. В каждом примере рассматривались k-ичные деревья глубины h, h –

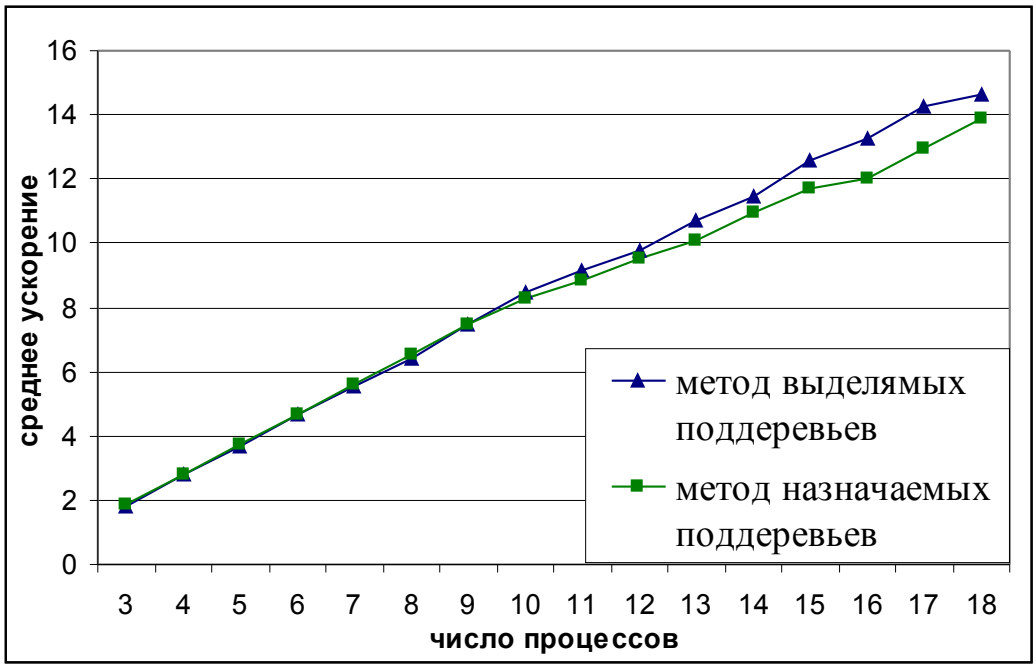

Рис. 1.9. Среднее ускорение при обходе двоичных и троичных деревьев

1, h – 2, где h максимальная глубина, при которой время последовательного обхода не превышает 10 часов. По результатам экспериментов среднее ускорение для 10 процессов в методе выделяемых поддеревьев равно 8,9, в методе назначаемых поддеревьев – 8,8. Эффективность U_s вычисляется из соотношения $U_s = C_s/s$, и равна соответственно 0,89 и 0,88. Можно заметить, что для k-ичных деревьев незначительное преимущество имеет метод выделяемых поддеревьев.

Наряду с оценками эффективности параллельных алгоритмов на k-ичных деревьях интересны также оценки их эффективности на деревьях, возникающих при решении конкретных комбинаторных задач. Этим исследованиям посвящён следующий раздел.

1.3. Параллельные алгоритмы решения комбинаторных задач обходом дерева

Изложенные в разделах 1.2.2 и 1.2.3 методы параллельного обхода дерева можно применить для разработки параллельных алгоритмов типовых комбинаторных задач – перечислительных и оптимизационных. С этой точки зрения определенный интерес представляют такие хорошо известные задачи, как перечисление сочетаний, перечисление разбиений, задача перечисления подмножеств с заданной суммой элементов (о рюкзаке) и оптимизационная задача о назначении.

1.3.1. Перечисление сочетаний

Здесь под *сочетанием* понимается всякое упорядоченное по возрастанию k-элементное подмножество заданного n-элементного множества $N = \{1, 2, \ldots, n\}$, т.е. любой такой вектор $(b_1, b_2, \ldots, b_k)$ длины $k \leq n$ с компонентами в N, называемый также сочетанием из n по k, в котором $b_i < b_{i+1}$ для $i = 1, \ldots, k-1$, а под перечислением сочетаний – перечисление всех таких векторов. Такое перечисление возникает при решении многих дискретных задач и часто является наиболее трудоемкой составляющей алгоритмов их решения. Распараллеливание процедуры перечисления позволяет использовать для выполнения этих алгоритмов многопроцессорные системы.

Для распараллеливания перечисления сочетаний определяется дерево

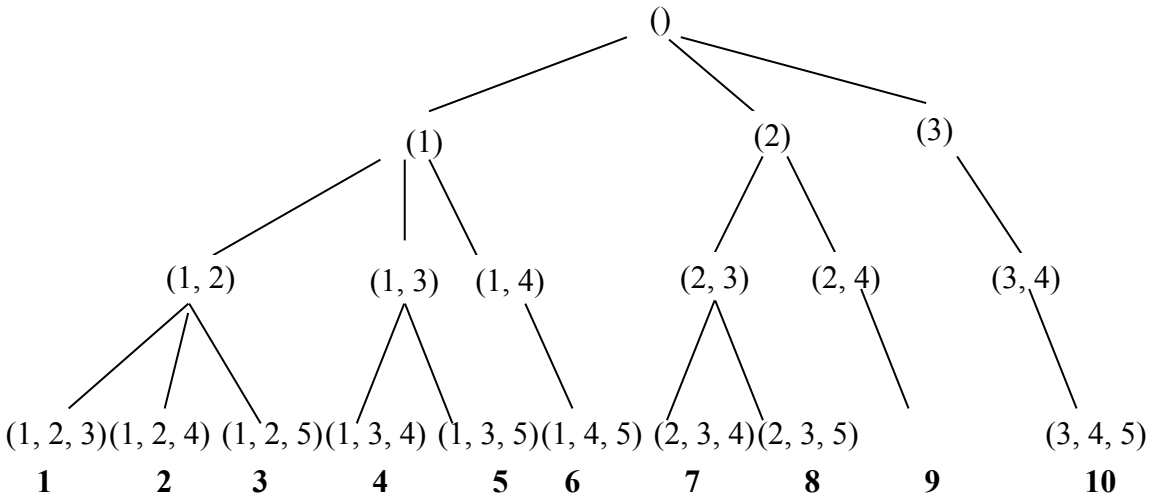

Рис. 1.10. Дерево сочетаний с параметрами $n = 5$, $k = 3$

сочетаний – ДС. Корень дерева сопоставляется пустому вектору, вершины i-го яруса для $i \geq 1$ – векторам длиной i. Вершины 1-го яруса сопоставляются однокомпонентным векторам с компонентами 1, 2, …, $n - k + 1$. Пусть некоторая вершина v i-го яруса ($0 < i \leq k$) сопоставлена вектору (b_1, b_2, …, b_i). Тогда если $i = k$, то v является концевой вершиной; в противном случае непосредственные потомки v соответствуют векторам (b_1, b_2, …, b_i, x) для всех $x \in \{b_i+1, b_i+2, \ldots, n - k + i\}$. Концевые вершины такого дерева представляют все сочетания из n по k. Рис. 1.10 демонстрирует дерево сочетаний из 5 по 3.

При обходе ДС методом левого обхода в глубину с достижением каждой концевой вершины получается очередное сочетание, и все сочетания перечисляются в лексикографическом порядке, т.е. так, что сочетание (b_1, b_2, … , b_k) предшествует сочетанию (a_1, a_2, …, a_k), если для некоторого i, $1 \leq i < k$, имеет место $b_1 = a_1$, $b_2 = a_2$, …, $b_i = a_i$ и $b_{i+1} < a_{i+1}$. Например, все сочетания из 5 по 3 по дереву, представленному на рис. 1.10, перечисляются в следующем порядке: (1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 3, 4), (1, 3, 5), (1, 4, 5), (2, 3, 4), (2, 3, 5), (2, 4, 5), (3, 4, 5). Дерево сочетаний интересно своей структурой, в том смысле, что в нем встречаются как поддеревья с очень большим числом вершин, так и

много поддеревьев, небольших по числу вершин, более того, имеющих линейную структуру.

Параллельные алгоритмы перечисления сочетаний основаны на параллельном обходе дерева сочетаний по методу назначаемых и выделяемых поддеревьев. В обоих случаях параллельный алгоритм состоит из алгоритма управляющего процесса и алгоритма рабочих процессов. В основе обоих алгоритмов лежит один из самых быстрых последовательных алгоритмов перечисления сочетаний, который можно найти в [13]. В этих и последующих алгоритмах перечислений комбинаторных объектов обозначение $\diamond$ используется, чтобы показать, что комбинаторный объект, после которого оно указывается, построен.

Алгоритм 1.4. (параллельный алгоритм перечисления сочетаний из n по k методом назначаемых поддеревьев)

Блок управляющего процесса

Входные данные: n; k; s – число рабочих процессов; m – минимальное число корневых вершин; $Q = ()$ – пустая очередь.

I-й этап (обход ДС в ширину с построением очереди, содержащей корневые вершины)

1. $Q := ((1), (2), \ldots, (n-k))$; $(n-k+1, n-k+2, \ldots, n)$ $\diamond$.
2. Пока $0 < |Q| < m$

   1) извлечь из Q очередной вектор, пусть это будет вектор $(b_1, b_2, \ldots, b_i)$;

   2) для каждого $x \in \{b_i+1, b_i+2, \ldots, n-k+i\}$ вектор $(b_1, b_2, \ldots, b_i, x)$ занести в Q;

   3) $(b_1, b_2, \ldots, b_i, n-k+i+1, n-k+i+2, \ldots, n)$ $\diamond$.

II-й этап (передача корневых вершин рабочим процессам)

1. Послать каждому рабочему процессу очередной вектор из Q.
2. Ожидать запрос от рабочего процесса.
3. Получить запрос.
4. Если $|Q| = 0$, то $s := s - 1$ и послать «отказ»;

иначе послать запросившему рабочему процессу очередной вектор из Q.

5. Если $s > 0$ (т.е. есть работающие процессы), то перейти в п. 2;
   иначе завершить работу.

Блок рабочего процесса

1. Получить от управляющего процесса префикс $(b_1, b_2, \ldots, b_i)$.
2. (Перечисление всех сочетаний с префиксом $(b_1, b_2, \ldots, b_i)$)
   1) $(b_1, b_2, \ldots, b_k) := (b_1, b_2, \ldots, b_i, b_i + 1, b_i + 2, \ldots, b_i + k - i), j := k$.
   2) Если $b_k < n$, то $b_k := b_k + 1$, $(b_1, b_2, \ldots, b_k)$ $\diamondsuit$, $j := k$ и п. 2.2.
   3) $j := j - 1$. Если $j = i$, то перейти в п. 3.
   4) $b_j := b_j + 1$, $b_{j+1} := b_j + 1$, …, $b_k := b_{k-1} + 1$.
   5) $(b_1, b_2, \ldots, b_k)$ $\diamondsuit$ и п. 2.2.
3. Послать запрос управляющему процессу.
4. Ожидать сообщение от управляющего процесса.
5. Если получен «отказ», то завершить работу;
   иначе перейти в п. 1.

В п.1 первого этапа управляющий процесс сразу помещает в очередь все вершины первого яруса, и строит последнее в лексикографическом порядке сочетание. На втором шаге управляющим процессом выполняется обход ДС в ширину до тех пор, пока в очереди не накопится m вершин. Существует ровно по одному сочетанию с префиксами вида $(b_1, \ldots, b_i, n - k + i + 1)$, поэтому такие сочетания целиком строятся и обрабатываются управляющим процессом в п. 2.3. Ожидание запросов от рабочих процессов (п.2 второго этапа) реализуется в программе с помощью функции MPI_Probe() из библиотеки MPI, п.3 – с помощью MPI_Recv(). Управляющий процесс завершает работу после того, как отправит всем рабочим процессам сообщение «отказ».

В п. 2.1 в блоке рабочего процесса строится первое в лексикографическом порядке сочетание с полученным префиксом (b_1, b_2, …, b_i). Переменная j содержит наименьший из номеров тех элементов, которые изменяются при переходе к следующему сочетанию. Поэтому условие п. 2.4 выполняется только, если построено последнее сочетание с префиксом (b_1, b_2, …, b_i). Рабочий процесс завершает работу, получив «отказ» от управляющего.

Экспериментальное исследование алгоритма 1.4 проводилось на одном из кластеров Научно исследовательского вычислительного центра МГУ (НИВЦ МГУ). Для примера на рис. 1.11 представлены графики, показывающие изменение эффективности с ростом числа процессов при перечислении сочетаний из 40 по 20. Здесь при вычислении эффективности в качестве T_0 использовалось время исполнения параллельной программы с одним

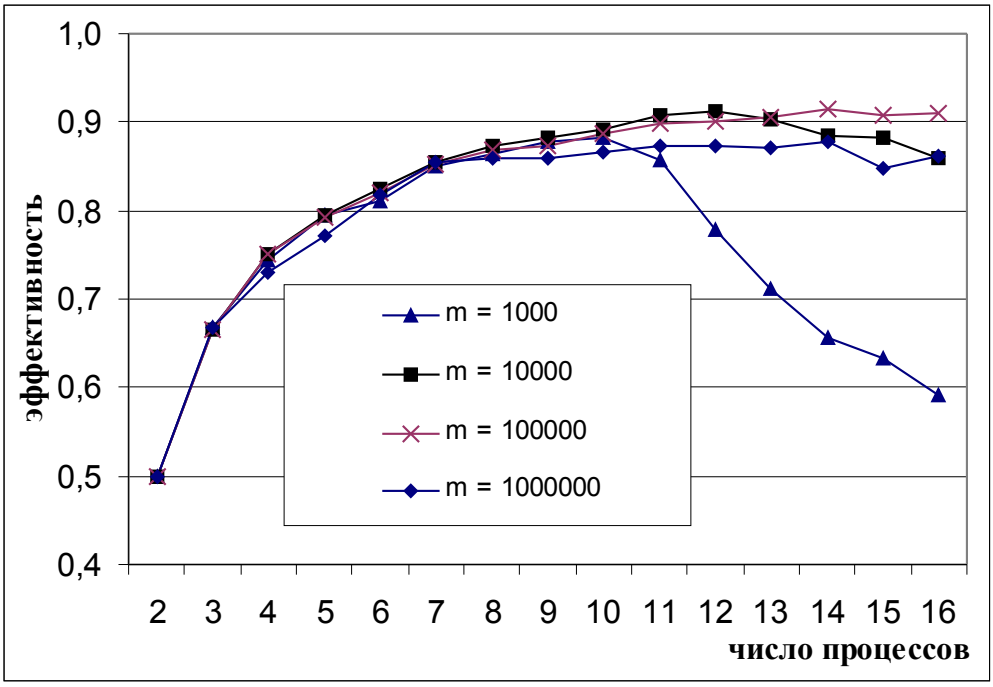

Рис. 1.11. Эффективность параллельного перечисления сочетаний из $n = 40$ по $k = 20$ методом назначаемых поддеревьев

управляющим и одним рабочим процессом. Графики построены для различных значений числа m корневых вершин. Как и ожидалось, при малом их числе (m = 1000) с увеличением числа процессов эффективность работы всей системы падает. Причину этого легче понять, если обратиться к временному графику

(рис. 1.12). Начиная с 11-ти процессов время работы параллельного алгоритма (максимальное время работы всех процессов) остается почти неизменным (m = 1000) и совпадает со временем работы одного из рабочих процессов, которому для обхода назначается такое большое поддерево, что обход всех остальных

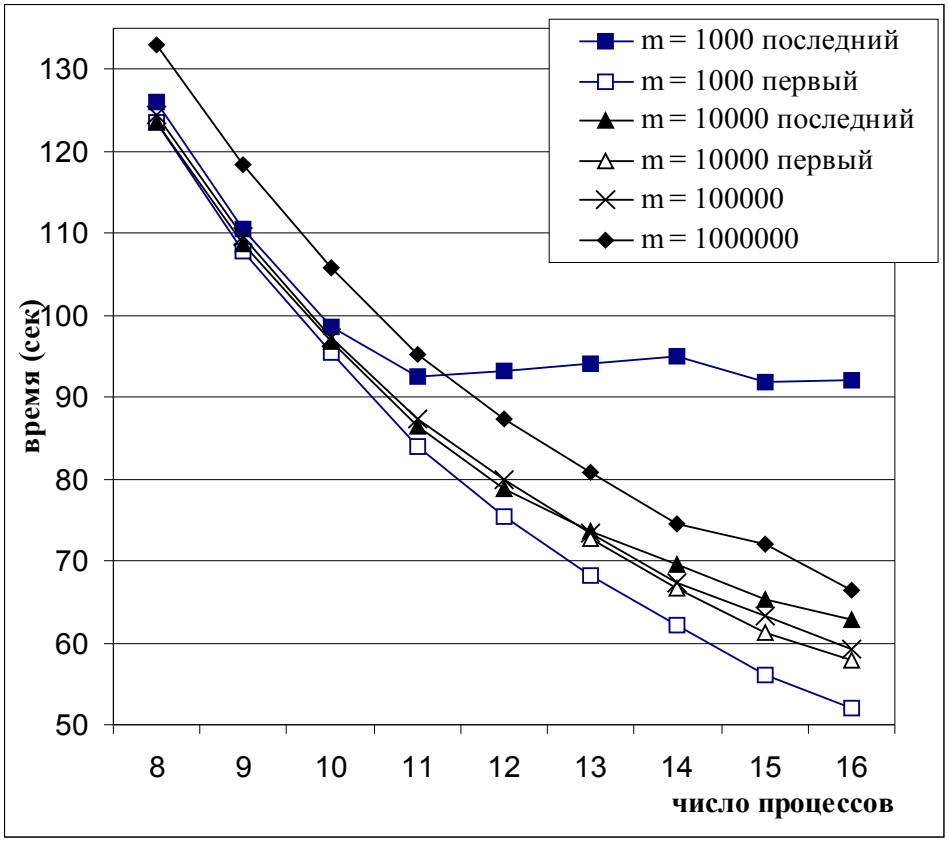

Рис. 1.12. Время параллельного перечисления сочетаний из 40 по 20 методом назначаемых поддеревьев

поддеревьев завершается раньше. На рис. 1.12 для сравнения представлены графики, показывающие время работы первого и последнего из завершившихся рабочих процессов. Для m = 1000 и m = 10000 заметна существенная разница.

С увеличением m *назначаемые* поддеревья получаются более близкими по размеру, что обеспечивает равномерную нагрузку процессоров и высокую эффективность. Вместе с тем с дальнейшим ростом m, во-первых, возрастает время, затрачиваемое на построение корневых вершин, во-вторых, увеличивается число обменов между управляющим и рабочими процессами, что снижает общую эффективность алгоритма, как показывает график для m =

1000000. Для $n = 40$, $k = 20$ (ДС состоит примерно из $2{,}7 \cdot 10^{11}$ вершин) наиболее подходящим оказалось значение $m = 100000$.

В целом экспериментальное исследование показало, что правильный выбор числа m назначаемых поддеревьев почти во всех случаях позволяет добиться достаточно равномерной загрузки рабочих процессов и высокой эффективности (для 16 процессов $\approx$ 0,9). Исключение составляют «глубокие» деревья например такие, как при $n = 100$, $k = 93$. Причина в том, что при значении m порядка 10^5 одно из назначаемых поддеревьев для этого примера содержит примерно половину всех сочетаний, а при m порядка 10^6 – одну пятую.

Алгоритм 1.5 (параллельный алгоритм перечисления сочетаний из n по k методом выделяемых поддеревьев)

Блок управляющего процесса

Входные данные: n; k; s – число рабочих процессов; $Q = ()$ – пустая очередь; $(v_1, v_2, \ldots, v_s) = (1, 1, \ldots, 1)$ – вспомогательный массив для выбора процесса-источника, в нем v_r содержит минимальный номер яруса, с которого могут выделяться поддеревья в дереве r-го процесса; M – максимальная глубина для выделения поддерева (максимальная длина передаваемого префикса).

I-й этап работы управляющего процесса такой же, как в методе назначаемых поддеревьев для $m = s$.)

II-й этап (управление выделением поддеревьев)

1. Послать каждому рабочему процессу очередной вектор из Q; $w := s$.
2. Ожидать сообщение от любого рабочего процесса.
3. Если от процесса с номером r получено значение t (минимальный номер яруса, с которого могут выделяться поддеревья), то $v_r := t$ и перейти в п.2.
4. Если от процесса с номером r получен запрос, то $v_r := M$.
5. Если $|Q| = 0$, то перейти в п. 6;

иначе послать запросившему рабочему процессу очередной вектор из Q и п.2.

6. Выбрать $l \in \{1, \ldots, s\}$ такое, что v_l – минимальное.
7. Если $v_l < M$, то послать процессу с номером l запрос на выделение поддерева для процесса с номером r;
   иначе $w := w - 1$.
8. Если $s > 0$ (т.е. есть работающие процессы), то перейти в п. 2;
   иначе послать всем процессам «отказ» и завершить работу.

Блок рабочего процесса

Входные данные: n; k; T – период (число построенных сочетаний, после которого проверяется наличие сообщений); M – максимальная глубина для выделения поддерева (максимальная длина передаваемого префикса).

1. Получить от управляющего процесса префикс $(b_1, b_2, \ldots, b_i)$.
2. $t := i + 1$, послать управляющему процессу t.
3. $(b_1, b_2, \ldots, b_k) := (b_1, b_2, \ldots, b_i, b_i + 1, b_i + 2, \ldots, b_i + k - i)$, $j := k$, $p := T$, $d := 0$.
4. Если $b_k < n$, то $b_k := b_k + 1$, $(b_1, b_2, \ldots, b_k) \diamondsuit$, $p := p - 1$, $j := k$ и п. 4.
5. $j := j - 1$.
6. Если $j = i$, то обход поддерева завершен, перейти в п. 18.
7. Если $j = t$, то $b_j := b_j + d$, $d := 0$.
8. $b_j := b_j + 1$, $b_{j+1} := b_j + 1$, …, $b_k := b_{k-1} + 1$.
9. $(b_1, b_2, \ldots, b_k) \diamondsuit$, $p := p - 1$.
10. Если $p > 0$, то перейти в п. 4; иначе $p := T$.
11. Если $t > M$ или $b_t + 1 \geq n - k + t$, то перейти в п. 4.
12. Если запросов от управляющего процесса нет, то перейти в п. 4.
13. Послать запросившему процессу префикс $(b_1, \ldots, b_{t-1}, b_t + d + 1)$.
14. Если $b_t + d + 2 < n - k + t$, то $d := d + 1$ и п. 4.
15. $(b_1, \ldots, b_{t-1}, b_t + d + 2, \ldots, n - 1, n) \diamondsuit$, $p := p - 1$.
16. $d := 0$, $i := t$, $t := t + 1$; послать управляющему процессу значение t и перейти в п. 4.

17. Послать запрос управляющему процессу.

18. Ожидать сообщений от любого процесса.

18.1. Если от управляющего процесса получен запрос на выделение префикса для процесса A, то процессу A послать «отказ» и п. 18.

18.2. Если от одного из рабочих процессов получен «отказ», то п. 17.

18.3. Если от управляющего процесса получен префикс (b_1, b_2, …, b_i), то п. 2.

18.4. Если от одного из рабочих процессов получен префикс (b_1, b_2, …, b_i), то п. 2.

18.5. Если от управляющего процесса получен «отказ», то завершить работу.

В этом алгоритме, кроме взаимодействия процессов, особое внимание следует уделить механизму выделения поддеревьев. Алгоритм позволяет выделять любое число поддеревьев и обеспечивает их корректное исключение из дерева. Выделение поддерева требует достаточно небольшого числа операций.

В блоке рабочего процесса использованы обозначения: t – номер яруса, вершины которого могут быть выделены в качестве корневых; d – число выделенных поддеревьев, с корневыми вершинами на t-м ярусе. В п. 3 строится первое в лексикографическом порядке сочетание с префиксом (b_1, b_2, …, b_i). В п. 4 перечисляются сочетания, отличающиеся только последним элементом. Пункты 5 – 9 отвечают за преобразование текущего сочетания в следующее за ним.

Наличие запроса от управляющего процесса проверяется с помощью функции MPI_Iprobe() из библиотеки MPI. Слишком частое выполнение этой функции приводит к заметному замедлению программы. Поэтому эффективней производить эту проверку с некоторой периодичностью, например, после построения T сочетаний (п. 10). Значение T может отличаться для задач

различной размерности и используемых вычислительных систем и поэтому выбирается экспериментально.

В п. 11 проверяются два условия неделимости поддерева. Первое из них – ($t > M$) ограничивает высоту передаваемых поддеревьев. Второе условие ($b_t + 1 \geq n - k + t$) выполняется, когда на t-м ярусе нет непройденных вершин степени больше 1. В ДС это означает, что и ни на каком из следующих ярусов таких вершин тоже нет. Можно добавить третье условие разбиения, накладываемое на число вершин в выделяемом поддереве. Корень выделяемого поддерева соответствует префиксу (b_1, …, b_{t-1}, $b_t + d + 1$), такое поддерево содержит

$$n - \left\lfloor \frac{t}{k - t} \right\rfloor + d + 1$$

righ

сочетаний.

Если поступил запрос на выделение поддерева и выполняются условия разбиения дерева, то алгоритм выделяет самое левое из непройденных поддеревьев максимальной высоты. В переменной t хранится минимальный номер яруса, с которого могут выделяться поддеревья. Вначале переменной t присваивается значение ($i + 1$), затем t увеличивается лишь тогда, когда на t-м ярусе остается только вершина степени 1, при этом сразу же строится последнее сочетание с префиксом (b_1, …, b_{t-1}), соответствующее линейному поддереву (п.п. 14, 15). Это исключает передачу поддеревьев, имеющих всего

одну концевую вершину. При изменении t изменяется и значение i (п. 16), по которому проверяется условие завершения обхода в п. 6.

Если все условия разбиения выполнены, то посылается префикс (b_1, …, b_{t-1}, $b_t + d + 1$), соответствующий корневой вершине выделенного поддерева. Если эта вершина не последняя на своем ярусе со степенью ветвления больше единицы, то переменная d увеличивается на единицу (п. 14). Это делается для исключения выделенного поддерева из обхода (п. 7) и на случай выделения следующего поддерева на этом же ярусе (п. 13).

Эксперименты проводились на ряде примеров, в которых использовались различные деревья сочетаний, в том числе: содержащие максимальное для заданного n число концевых вершин (например, $n = 34$, $k = 17$; $n = 40$, $k = 20$); «глубокие» ($n = 100$, $k = 93$; $n = 50$, $k = 42$; $n = 50$, $k = 35$) и «широкие» ($n = 50$, $k = 9$; $n = 50$, $k = 15$; $n = 100$, $k = 8$). Для достижения максимального ускорения подбирались значения параметров T и M. Исследования показали, что подходящими, речь идет о примерах с числом сочетаний больше 10^9, являются значения T порядка 10^4, 10^5. Оптимальные значения M существенно различаются для разных типов деревьев: для «глубоких» – около (k – 10), для «широких» – $k/2$. Средняя достигнутая эффективность равна 0,86.

1.3.2. Перечисление разбиений

Разбиением множества A на k блоков называется семейство $R = \{R_1, R_2, \ldots, R_k\}$ непустых и попарно непересекающихся подмножеств множества A таких, что $k \le n$ и $R_j = A$ ¿ [17]. Подмножества R_1, R_2, …, R_k называются *блоками* разбиения.

Число разбиений n-элементного множества на k блоков выражается числом Стирлинга второго рода $S(n, k)$, определяемого рекуррентным соотношением $S(n, k) = S(n - 1, k - 1) + k \cdot S(n - 1, k)$ при $S(n, n) = 1$, $S(n, 1) = 1$. Число всех разбиений n-элементного множества равно $B_n = \sum_{k=1}^{n} S(n, k)$.

Пусть в множестве $A = \{a_1, a_2, \ldots, a_n\}$ элементы упорядочены некоторым фиксированным образом. Будем также фиксировать порядок блоков в любом разбиении R множества A так, что для каждого $i \in \{1, \ldots, n\}$ элемент a_i содержится в блоке с номером $p_i \le i$.

Определим *дерево разбиений* n-элементного множества $A = \{a_1, \ldots, a_n\}$. Разбиение может состоять из 1, 2, …, n блоков. Обозначим $A_m \subseteq A$ такое, что A_m

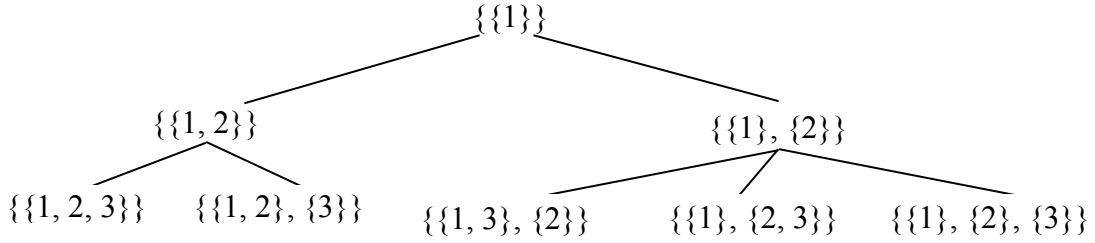

Рис. 1.13. Дерево разбиений множества $A = \{1, 2, 3\}$

$= \{a_1, \ldots, a_m\}$, $1 \le m \le n$. Корень дерева (здесь вершина 1-го яруса) сопоставляется единственному разбиению одноэлементного множества A_1. Вершины i-го яруса сопоставляются всевозможным разбиениям множества A_i, их число равно B_i. Вершины n-го яруса являются концевыми. Пусть вершине i-го яруса соответствует разбиение $R = \{R_1, R_2, \ldots, R_k\}$. Тогда она имеет $(k + 1)$ непосредственных потомков, которые сопоставляются следующим разбиениям множества A_{i+1}: $\{R_1 \cup \{a_{i+1}\}, R_2, \ldots, R_k\}$, $\{R_1, R_2 \cup \{a_{i+1}\}, \ldots, R_k\}$, …, $\{R_1, R_2, \ldots, R_k \cup \{a_{i+1}\}\}$, $\{R_1, R_2, \ldots, R_k, \{a_{i+1}\}\}$. На рис. 1.13 представлено дерево разбиений множества $A = \{1, 2, 3\}$.

Разбиение R множества A будем представлять вектором $p = (p_1, p_2, \ldots, p_n)$, в котором p_i это номер блока, содержащего элемент a_i. Таким образом, $p_1 = 1$ и для $1 \le i \le n - 1$ имеем: $p_{i+1} = j$, если a_{i+1} содержится в блоке с номером j и $j \le k_i$, где k_i – количество блоков в R, пересекающихся с $\{a_1, \ldots, a_i\}$, или $p_{i+1} = k_i + 1$ в противном случае. При таком представлении любое разбиение множества однозначно определяется своим *представляющим вектором*, и перечисление разбиений можно осуществить как перечисление их векторов.

Для перечисления всех разбиений множества известны последовательные как рекурсивный, так и не рекурсивный алгоритмы [13]. В рекурсивном алгоритме (см. алгоритм 1.6) на очередном шаге рекурсии к текущему разбиению множества $\{a_1, \ldots, a_m\}$ добавляется новый элемент a_{m+1}, который последовательно проходит через все блоки и затем образует отдельный блок. Рекурсивные вызовы завершаются после добавления последнего элемента разбиваемого множества. По сути, выполняется рекурсивный алгоритм обхода в глубину дерева разбиений. Этот алгоритм используется рабочими процессами в обоих методах.

Алгоритм 1.6 (последовательный рекурсивный алгоритм перечисления всех разбиений множества $A = \{a_1, a_2, \ldots, a_n\}$)

Входные параметры: r – число не пустых блоков в разбиении; m – число уже размещенных по блокам элементов, а именно элементов $a_1, a_2, \ldots, a_m$.

1. Если $m = n$, то $(p_1, p_2, \ldots, p_n)$ $\diamondsuit$ и п. 5.
2. Для каждого $i \in \{1, 2, \ldots, r\}$:

1) $p_{m+1} := i$ (поместить элемент a_{m+1} в существующий i-й блок);

2) выполнить алгоритм 1.6 с входными параметрами r и $(m + 1)$.

3. $p_{m+1} := r + 1$ (поместить $(m + 1)$-й элемент в блок с номером $(r + 1)$).

4. Выполнить алгоритм 1.6 с входными параметрами $(r + 1)$ и $(m + 1)$.

5. Конец.

Следующий алгоритм 1.7 предназначен для параллельного перечисления разбиений n-элементного множества методом назначаемых поддеревьев. На первом шаге первого этапа управляющий процесс сразу помещает в очередь обе вершины второго яруса. На втором шаге он выполняет обход дерева разбиений в ширину до тех пор, пока в очереди не накопится m вершин. Величина $\max\{p_1, p_2, \ldots, p_t\}$ показывает число блоков в разбиении, представляемом вектором $(p_1, p_2, \ldots, p_t)$.

Алгоритм 1.7 (параллельный алгоритм перечисления разбиений n-элементного множества методом назначаемых поддеревьев)

Блок управляющего процесса

Входные данные: n; s – число рабочих процессов; m – минимальное число корневых вершин; $Q = ()$ – пустая очередь.

I-й этап (обход дерева разбиений в ширину с построением очереди, содержащей корневые вершины)

1. $Q := ((1, 1), (1, 2))$.
2. Пока $0 < |Q| < m$

   1) извлечь из очереди вектор $(p_1, p_2, \ldots, p_t)$;

   2) для каждого $x \in \{1, 2, \ldots, \max\{p_1, p_2, \ldots, p_t\} + 1\}$ вектор $(p_1, p_2, \ldots, p_t, x)$ занести в Q.

II-й этап (этап передачи корневых вершин рабочим процессам) такой же, как в алгоритме 1.4.

Блок рабочего процесса

1. Получить от управляющего процесса префикс $(p_1, p_2, \ldots, p_t)$.
2. Выполнить алгоритм 1.6 с входными параметрами $r = \max\{p_1, p_2, \ldots, p_t\}$ и $m = t$ (т.е. перечислить все разбиения с префиксом $(p_1, p_2, \ldots, p_t)$).
3. Послать запрос управляющему процессу.
4. Ожидать сообщение от управляющего процесса.
5. Если получен «отказ», то завершить работу;
   иначе перейти в п. 1.

Алгоритм 1.8 (параллельный алгоритм перечисления разбиений n-элементного множества методом выделяемых поддеревьев)

Блок управляющего процесса

На I-м этапе управляющий процесс выполняет обход дерева разбиений в ширину с построением очереди, содержащей корневые вершины, так же, как в алгоритме 1.7, только для $m = s$, где s – число рабочих процессов. II-й этап работы управляющего процесса такой же, как в алгоритме 1.5 перечисления сочетаний.

Блок рабочего процесса

Входные данные: n; T – период (число построенных разбиений, после которых проверяется наличие сообщений); M – минимальная высота выделяемого поддерева.

1. Получить от управляющего процесса префикс $(p_1, p_2, \ldots, p_i)$.
2. $t := i + 1$, послать t управляющему процессу.
3. $q := T$; $(g_1, g_2, \ldots, g_i) := (0, 0, \ldots, 0)$.
4. Выполнить алгоритм 1.9 с входными параметрами $r = \max\{p_1, p_2, \ldots, p_i\}$ и $m = i$.
5. Послать запрос управляющему процессу.
6. Ожидать сообщений от любого процесса.
   - 6.1. Если от управляющего процесса получен запрос на выделение префикса для процесса A, то процессу A послать «отказ» и п. 6.
   - 6.2. Если от одного из рабочих процессов получен «отказ», то п. 5.
   - 6.3. Если от управляющего процесса получен префикс $(p_1, p_2, \ldots, p_i)$, то п. 2.
   - 6.4. Если от одного из рабочих процессов получен префикс $(p_1, p_2, \ldots, p_i)$, то п. 2.
   - 6.5. Если от управляющего процесса получен «отказ», то завершить работу.

В блоке рабочего процесса в п. 4. вызывается приводимый ниже рекурсивный алгоритм 1.9 перечисления разбиений с выделением поддеревьев. Для алгоритма 1.9 все параметры, определенные в алгоритме 1.8, кроме m и r, являются глобальными, т.е. общими для всех рекурсивных вызовов.

Алгоритм 1.9 (рекурсивный алгоритм перечисления разбиений с выделением поддеревьев)

Входные параметры: r – число не пустых блоков в разбиении; m – число уже размещенных по блокам элементов.

1. Если $m < n$, перейти в п. 6.

2. $(p_1, p_2, \ldots, p_n) \diamondsuit$, $q := q - 1$.
3. Если $q > 0$, то перейти в п. 9; иначе $q := T$.
4. Если нет запроса на выделение поддерева, то перейти в п. 9.
5. Если существует t такое, что $i < t < n - M$ и $p_t < g_t$, то
   1) найти наименьшее t такое, что $i < t < n - M$ и $p_t < g_t$;
   2) послать запросившему процессу префикс $(p_1, \ldots, p_{t-1}, g_t)$; $g_t := g_t - 1$;
   3) если $g_t = 0$, то послать управляющему процессу значение $t + 1$ (минимальный номер яруса, с которого могут выделяться поддеревья).
   4) перейти в п. 9.
6. $m := m + 1$, $g_m := r$.
7. $p_m := r + 1$ (поместить m-й элемент в блок с номером $r + 1$). Выполнить алгоритм 1.9 с входными параметрами $(r + 1)$ и m.
8. $j := 1$.
   8.1. Если $j > g_m$, то п. 9.
   8.2. $p_m := j$ (поместить элемент a_m в существующий j-й блок).
   8.3. Выполнить алгоритм 1.6 с входными параметрами r и m.
   8.4. $j := j + 1$, перейти в п. 8.1.
9. Конец.

В алгоритме 1.9, в отличие от алгоритма 1.6, добавляемый на очередном шаге рекурсии элемент вначале образует отдельный блок (п. 7) и только затем последовательно проходит через все блоки (п. 8). Это изменение сделано для упрощения процедуры выделения поддеревьев. Никогда не выделяется поддерево с корневой вершиной, в которой последний элемент образует отдельный блок. Это поддерево проходится самим процессом в первую очередь. Для выделения поддерева ищется минимальный ярус t, на котором есть непройденные вершины (п. 5.1). В элементе g_t вспомогательного массива g хранится максимальное допустимое значение для элемента p_t при условии, что t-й элемент не образует отдельного блока. Всякий раз выделяется поддерево с

корнем (p_1, …, p_{t-1}, g_t), после чего g_t уменьшается на 1. Если бы не было запросов на выделение поддеревьев, то обход этого поддерева самим процессом выполнялся бы в последнюю очередь. Условие разбиения в этом алгоритме заключается в существовании непройденных поддеревьев высоты M.

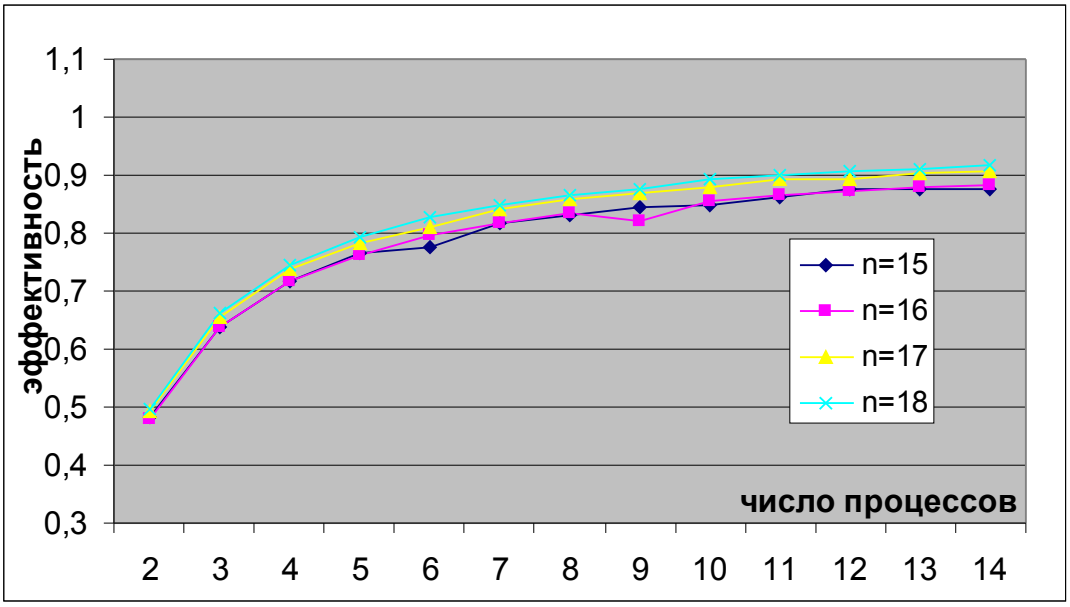

Рис. 1.14. Графики эффективности параллельного перечисления разбиений множеств мощности n методом назначаемых поддеревьев

В п. 8.3 вызывается рекурсивный последовательный алгоритм 1.6, не содержащий дополнительных операций (п.п. 3 – 5 алгоритма 1.9), предназначенных для выделения поддеревьев. Такой подход приводит к значительному ускорению работы алгоритма по сравнению с вариантом, в котором каждый раз вызывался бы алгоритм 1.9.

Как последовательный, так и параллельный алгоритмы можно ускорить примерно в полтора раза, если прекратить рекурсивные вызовы на один шаг раньше. В этом случае п.п. 1, 2 алгоритма 1.9 изменятся следующим образом.

1. Если $m < n - 1$, то перейти в п. 6.

2. Для каждого $i \in \{1, 2, \ldots, r, r + 1\}$:

$p_n := i$, $(p_1, p_2, \ldots, p_n)$ $\diamond$, $q := q - 1$.

На рис. 1.14. представлены графики, показывающие эффективность работы

параллельного алгоритма перечисления разбиений методом назначаемых поддеревьев. При этом число корневых вершин $m = 10000$ для $n = 15, 16, 17$ и $m = 100000$ для $n = 18$. Хорошая эффективность метода назначаемых поддеревьев в данном случае объясняется тем, что назначаемые поддеревья в дереве разбиений получаются почти одинакового размера.

В эксперименте, проведенном для исследования эффективности перечисления разбиений методом выделяемых поддеревьев, варьировались два параметра метода: T и M. Маленькие значения T (см. рис. 1.15, $T = 1000$) приводят к многократным вызовам функции *MPI_Iprobe*() для проверки

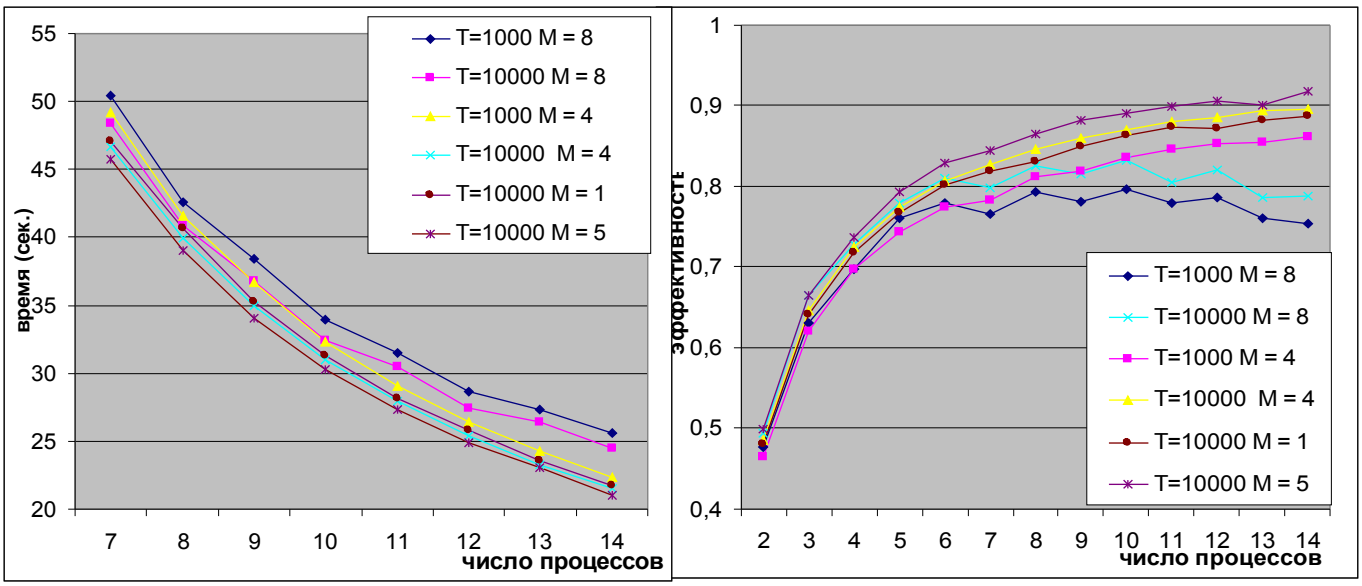

Рис. 1.15. Графики времени и эффективности для $n = 16$ и различных значений параметров T и M в методе выделяемых поддеревьев

наличия запроса от управляющего процесса (п. 4 алгоритма 9), что существенно замедляет выполнение программы. Однако слишком большие значения T ($T > 100000$) приводят к простою свободных процессов и, как следствие, к падению эффективности. Второй параметр M накладывает ограничение на минимальную высоту выделяемых поддеревьев, чтобы не допустить ситуации, в которой на передачу поддерева потребуется больше времени, чем на его обход процессом-источником. На рис. 1.15. представлены графики времени и эффективности для $n = 16$ и различных значений параметров T и M, наибольшее ускорение достигается при $T = 10000$ и $M = 5$.

Для задач, рассматриваемых в последующих разделах, мы ограничимся описанием соответствующего дерева поиска и особенностей, которые возникают при разработке и исследовании алгоритмов параллельного обхода этого дерева.

1.3.3. Задача о рюкзаке

В качестве примера из класса задач поиска комбинаторных объектов с заданными свойствами может быть рассмотрена задача о рюкзаке. Алгоритмы решения таких задач методом поиска с возвращением отличаются от алгоритмов перечисления тем, что, во-первых, дерево поиска не имеет определенной структуры в том смысле, что число вершин той части дерева, которая реально просматривается во время обхода, заранее определить достаточно трудно, и, во-вторых, при обходе дерева в каждой вершине выполняется набор дополнительных действий, иногда требующих значительных временных затрат.

Задача о рюкзаке рассматривается в следующей постановке: заданы натуральное число ω и набор неотрицательных чисел $B = (b_1, b_2, \ldots, b_n)$. Требуется найти, если существуют, все такие подмножества $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$, для которых $\sum_{r=1}^{k} b_{i_r} = \omega$. Предполагается, что элементы в наборе B упорядочены по невозрастанию**:** $b_1 \geq b_2 \geq \ldots \geq b_n$. Подмножества $I \subseteq \{1,\ldots, n\}$ будем перебирать, представляя их векторами $(i_1, \ldots, i_k)$, где $\{i_1, \ldots, i_k\} = I$ и $b_{i_1} \geq \ldots \geq b_{i_k}$.

В подобной постановке задача встречается в криптографии, в связи с криптоанализом рюкзачной (ранцевой) криптосистемы [16] с тем отличием, что там достаточно найти не все, но хотя бы одно решение. Мы же, в целях исследования параллельного обхода полного дерева поиска, не ограничиваемся первым найденным решением.

Алгоритм решения задачи о рюкзаке строится по методу поиска с возвращением. В его основе лежит выполнение шагов двух типов для некоторого текущего набора I из $\{1,\ldots, n\}$: первый тип – расширение, если оно возможно, построенного на данный момент набора I за счет добавления очередного элемента, и, если расширить набор нельзя, то второй тип – возврат, состоящий в удалении последнего добавленного элемента.

С целью выяснения возможности расширения построенного на данный момент набора $I = (i_1, \ldots, i_{k-1})$ для $k \in \{1, \ldots, n\}$ строится множество S_k, которое состоит из всех $j \in \{1,\ldots, n\}$, удовлетворяющих условиям:

1) $j > i_{k-1}$;

2) $b_j \le \omega - \sum_{r=1}^{k-1} b_{i_r} \le \sum_{r=j}^{m} b_r$,

где $i_0 = 0$. Тогда расширение $I = (i_1, \ldots, i_{k-1})$ возможно, если и только если $S_k \neq \varnothing$, и алгоритм поиска с возвращением для решения задачи о рюкзаке имеет следующий вид.

Алгоритм 1.10.

0. $k := 1$.

1. Вычислить S_k.

2. Если $S_k = \varnothing$, то перейти в п. 6.

3. Очередной (наименьший) элемент $j \in S_k$ исключить из S_k и $i_k := j$.

4. Если $\sum_{r=1}^{k} b_{i_r} = \omega$, то $\{i_1, \ldots, i_k\}$ $\diamondsuit$.

5. $k := k + 1$; перейти в п. 1.

6. $k := k - 1$. Если $k = 0$, то конец алгоритма; иначе перейти в п. 2.

Данный алгоритм обеспечивает обход в глубину следующего дерева поиска. Корень дерева сопоставляется пустому вектору, вершины i-го яруса для $i \ge 1$ – векторам длиной i. Для каждого элемента из S_1 вершина 1-го яруса сопоставляется 1-компонентному вектору с этим элементом. Пусть некоторая

вершина v k-го яруса ($0 < k \le n$) сопоставлена вектору ($i_1, \ldots, i_k$). Тогда если $S_{k+1} = \varnothing$, то v является концевой вершиной; в противном случае непосредственные потомки v соответствуют векторам $(i_1, \ldots, i_k, x)$ для всех $x \in S_{k+1}$. Концевая вершина с сопоставленным ей вектором ($i_1, \ldots, i_k$) задает решение задачи, если $\sum_{r=1}^{k} b_{i_r} = \omega$.

Для распараллеливания обхода описанного дерева использовались оба метода. В алгоритме, разработанном по методу назначаемых поддеревьев, управляющий процесс выполняет обход дерева в ширину до тех пор, пока не получит требуемого числа m корневых вершин. При этом учитывается возможность построения некоторых решений уже во время обхода дерева в ширину. Каждая корневая вершина представляется соответствующим ей вектором, который и передается рабочему процессу при поступлении запроса. Получив вектор ($i_1, \ldots, i_{r-1}$), рабочий процесс выполняет алгоритм 1.10 с модификациями в п. 1 ($k := r$) и в п. 6 (вместо $k = 0$ используется $k < r$). В экспериментальном исследовании алгоритма лучшие результаты получены для значений m порядка 10^4 и при числе узлов в деревьях порядка 10^9, 10^{10}. В среднем на построение корневых вершин и обмен данными затрачивается 1 – 2% времени работы последовательного алгоритма. Средняя эффективность распараллеливания для 12 процессов равна 0,86.

Для параллельного алгоритма, разработанного по методу выделяемых поддеревьев, исследовались две версии, отличающиеся друг от друга условием разбиения. В первой – в качестве условия разбиения поддерева указывается максимальный номер яруса M, на котором может быть расположена его корневая вершина. Во второй – указывается минимальное число узлов N в разбиваемом поддереве. Для оценки числа узлов в поддереве используется метод Монте-Карло [15]. Суть метода состоит в том, что проводится ряд испытаний, и в каждом испытании случайным образом строится путь в дереве

поиска от корня до концевой вершины. Для каждого узла x, входящего в путь, вычисляется величина $P(x) = q(x)P(f(x))$, где $q(x)$ число непосредственных потомков узла x и $f(x)$ – предыдущий узел в пути. Сумма величин $P(x)$ для всех узлов x, входящих в путь, показывает примерное число узлов в дереве поиска. За окончательный результат принимается среднее значение по всем проведенным испытаниям.

Экспериментальный анализ показал, что вторая версия алгоритма не дает предполагаемого выигрыша по сравнению с первой. При оптимальном подборе значений M и N в первой и второй версиях в среднем получаются одинаковые результаты по времени. Таким образом, затраты на оценивание размера поддеревьев во второй версии не оправдывают себя. И хотя алгоритм в обоих случаях обладает хорошей эффективностью и масштабируемостью (с ростом числа процессов эффективность не снижается, в среднем остается равной 0,81), по времени он показывает худшие результаты, чем алгоритм метода назначаемых поддеревьев.

1.3.4. Задача о назначении

Задача о назначении относится к классу задач поиска оптимального решения, использующих некоторую целевую функцию – характеристику решения. В задачах данного типа в каждой вершине дерева поиска проводится анализ, связанный с полученным на данный момент лучшим решением, по результатам которого продолжается ветвление дерева из рассматриваемой вершины, либо выполняется возврат на предыдущий уровень.

Задача о назначении состоит в том, что при заданных положительных числах a_{ij} для всех i и j в $N = \{1, 2,\ldots, n\}$ требуется найти перестановку (k_1, k_2,…, k_n) чисел 1, 2,…, n с минимальным значением целевой функции $W = \sum_{i=1}^{n} a_{i,k_i}$ [1].

Дерево поиска решения данной задачи определяется по следующим правилам. Его вершинам сопоставляются подмножества множества N, а ребрам

– элементы в N. Если некоторой вершине i-го яруса сопоставлено подмножество Y, то ребрам i-го яруса, исходящим из нее, сопоставляются элементы множества Y, а их концам – вершинам (i+1)-го яруса – множества, полученные исключением из Y элементов, сопоставленных этим ребрам соответственно. Корню дерева сопоставляется множество N. Вершины, которым сопоставлены пустые множества, являются концевыми. Максимальная глубина такого дерева равна n. Последовательности элементов, сопоставленных ребрам пути от корня к концевым вершинам и перечисленных в порядке их вхождения в путь, образуют всевозможные перестановки множества N.

На множестве последовательностей элементов из N длины l ($l = 1, \ldots, n$) определим функцию $F(k_1, k_2, \ldots, k_l) = \sum_{i=1}^{l} a_{i,k_i}$. Для сокращения обхода дерева в каждой достигнутой вершине с сопоставленным ей подмножеством $Y = N - \{k_1, k_2, \ldots, k_r\}$ вычисляются значение функции $F(k_1, k_2, \ldots, k_r)$ и число $F_0(Y) = \max(A, B)$, где $A = \sum_{i=r+1}^{n} a_{i,j}$¿ , $B = \sum_{j \in Y} a_{i,j}$¿ . Значение $F(k_1, k_2, \ldots, k_r)$ есть *вклад* в значение целевой функции уже построенной перестановки $(k_1, k_2, \ldots, k_r)$, а $F_0(Y)$ – нижняя оценка вклада в него перестановки элементов в Y. В случае $F(k_1, k_2, \ldots, k_r) + F_0(Y) \geq W$, где W – достигнутое на данный момент значение целевой функции, эта вершина объявляется бесперспективной, и ветвления из нее не делается, так как ни один путь, проходящий через эту вершину, не представляет перестановки, значение целевой функции для которой меньше известного. С достижением концевой вершины и соответствующей ей перестановки всех чисел 1, 2, …, n значение целевой функции для последней становится новым значением W.

При построении очередной перестановки изменяется достигнутое значение целевой функции, и дальнейшее сокращение обхода дерева зависит от этого значения, пока не будет найдено лучшее. Предположим, что обход дерева выполняется параллельно несколькими процессами в соответствии с одним из

методов, изложенных выше. При нахождении некоторым из процессов нового лучшего решения значение целевой функции W следует обновить на всех процессорах системы, т. е. оно должно быть передано всем остальным процессам. В многопроцессорных системах, не имеющих общей памяти, необходима явная передача найденного значения, что требует дополнительных временных затрат. Заметим, что если текущее значение целевой функции у процесса хуже глобального лучшего, то это лишь снижает эффективность его поиска, но не влияет на корректность найденного в итоге решения.

Обновление значения целевой функции в методе назначаемых поддеревьев выполняется следующим образом. При назначении очередного поддерева управляющий процесс передает рабочему минимальное из известных ему значений целевой функции W. В свою очередь, если рабочий процесс находит в своем поддереве назначение со стоимостью $W' < W$, то заменяет значение W на W' и пересылает W' управляющему процессу.

В методе выделяемых поддеревьев всякий раз, когда управляющий получает от одного из рабочих процессов новое значение W', он сравнивает его с известным ему значением W и, если $W' < W$, то заменяет значение W на W' и рассылает W' всем рабочим процессам. В свою очередь рабочие процессы периодически проверяют, не передано ли им от управляющего процесса новое значение W, и в случае передачи изменяют у себя значение целевой функции на присланное.

В задачах оптимизации время поиска решения, достигаемое при использовании многопроцессорной системы, зависит от исходных данных, определяющих тип дерева поиска. В тех случаях, когда решение быстро строится используемым последовательным алгоритмом, применение параллельного алгоритма может не дать выигрыша по времени независимо от числа процессов. Так, допустим, что решение задачи о назначении есть перестановка, представленная самым левым путем в дереве поиска, т. е. перестановка (1, 2, …, n). При решении задачи с помощью последовательного

алгоритма данная перестановка находится сразу, и, благодаря найденному значению целевой функции, остальные ветви дерева отсекаются уже на первых ярусах, после чего обход быстро завершается. При использовании параллельного алгоритма решение будет получено в лучшем случае за то же время. Возможна и обратная ситуация, в которой решение представлено одним из последних просматриваемых при последовательном алгоритме путей в дереве поиска, но этот путь оказывается первым в поддереве одного из параллельных процессов. Тогда время работы параллельного алгоритма будет складываться из времени, необходимого для построения этого пути, времени, необходимого для передачи найденного значения целевой функции остальным процессам, и времени на вычисление нижних оценок для вершин верхних ярусов. В этом случае время поиска решения может быть во много раз меньше, чем время, затрачиваемое последовательным алгоритмом, в результате чего возникает эффект суперлинейного ускорения [26]. На рис. 1.16 представлены результаты работы алгоритмов для примера такого типа. Время работы последовательного алгоритма для этого примера (n = 100) равно 2378 секунд, на 16 процессорах методом назначаемых поддеревьев решение было найдено за 27 секунд, т.е. почти в 80 раз быстрее.

Результаты экспериментального анализа показали также, что при решении данной задачи преимущество имеет метод назначаемых поддеревьев.

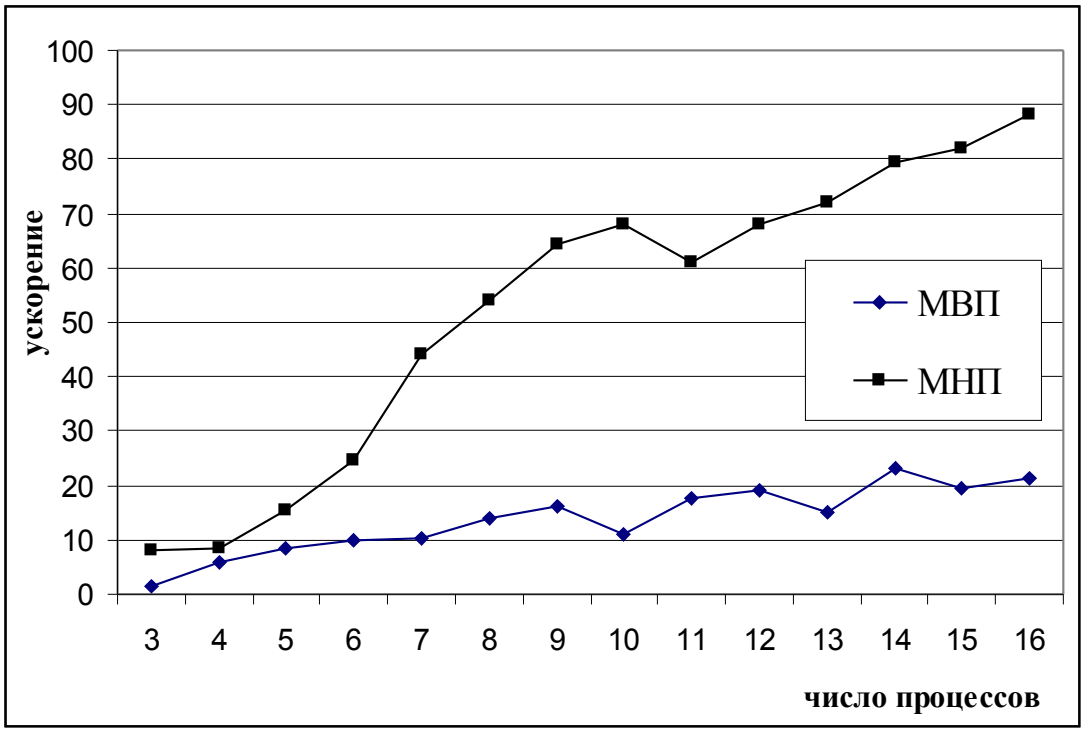

Рис. 1.16. Суперлинейное ускорение для метода выделяемых (МВП) и назначаемых поддеревьев (МНП)