ГЛАВА 2
МОДЕЛИРОВАНИЕ И АНАЛИЗ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

При разработке параллельных алгоритмов решения сложных научно-технических задач принципиальным моментом является анализ эффективности использования параллелизма, состоящий обычно в оценке получаемого ускорения процесса вычислений (сокращения времени решения задачи). Формирование подобных оценок ускорения может осуществляться применительно к выбранному вычислительному алгоритму (оценка эффективности распараллеливания конкретного алгоритма). Другой важный подход может состоять в построении оценок максимально возможного ускорения процесса решения задачи конкретного типа (оценка эффективности параллельного способа решения задачи).

В данной главе описывается модель вычислений в виде графа «операции-операнды», которая может использоваться для описания существующих информационных зависимостей в выбираемых алгоритмах решения задач, приводятся оценки эффективности максимально возможного параллелизма, которые могут быть получены в результате анализа имеющихся моделей вычислений. Примеры использования излагаемого теоретического материала приводятся при рассмотрении параллельных алгоритмов в завершающих разделах настоящего учебного материала.

2.1. Модель вычислений в виде графа «операции–операнды»

Для описания существующих информационных зависимостей в выбираемых алгоритмах решения задач может быть использована модель в виде графа «операции–операнды» (см., например, [8,10,44]). Для уменьшения сложности излагаемого материала при построении модели будет предполагаться, что время выполнения любых вычислительных операций является одинаковым и равняется 1 (в тех или иных единицах измерения). Кроме того, принимается, что передача данных между вычислительными устройствами выполняется мгновенно без каких-либо затрат времени (что может быть справедливо, например, при наличии общей разделяемой памяти в параллельной вычислительной системе).

Представим множество операций, выполняемых в исследуемом алгоритме решения вычислительной задачи, и существующие между операциями информационные зависимости в виде ациклического ориентированного графа

$$G = (V, R),$$

где $V = \{1,...,|V|\}$ есть множество вершин графа, представляющих выполняемые операции алгоритма, а R есть множество дуг графа (при этом дуга $r = (i, j)$ принадлежит графу только в том случае, если операция j использует результат выполнения операции i). Для примера на рис. 2.1 показан граф алгоритма вычисления площади прямоугольника, заданного координатами двух противолежащих углов. Как можно заметить по приведенному примеру, для выполнения выбранного алгоритма решения задачи могут быть использованы разные схемы вычислений и построены соответственно разные вычислительные модели. Как будет показано далее, разные схемы вычислений обладают различными возможностями для распараллеливания и, тем самым, при построении модели вычислений может быть поставлена задача выбора наиболее подходящей для параллельного исполнения вычислительной схемы алгоритма.

В рассматриваемой вычислительной модели алгоритма вершины без входных дуг могут использоваться для задания операций ввода, а вершины без выходных дуг – для операций вывода. Обозначим через $\overline{V}$ множество вершин графа без вершин ввода, а через $d(G)$ диаметр (длину максимального пути) графа.

2.2. Описание схемы параллельного выполнения алгоритма

Операции алгоритма, между которыми нет пути в рамках выбранной схемы вычислений, могут быть выполнены параллельно (для вычислительной схемы на рис. 2.1, например, параллельно могут быть реализованы сначала все операции умножения, а затем первые две операции вычитания). Возможный способ описания параллельного выполнения алгоритма может состоять в следующем (см., например, [8,10,44]).

Пусть p есть количество процессоров, используемых для выполнения алгоритма. Тогда для параллельного выполнения вычислений необходимо задать множество (*расписание*)

$$H_p = \{(i, P_i, t_i) : i \in V\},$$

в котором для каждой операции $i \in V$ указывается номер используемого для выполнения операции процессора P_i и время начала выполнения опе-

рации t_i. Для того, чтобы расписание было реализуемым, необходимо выполнение следующих требований при задании множества H_p:

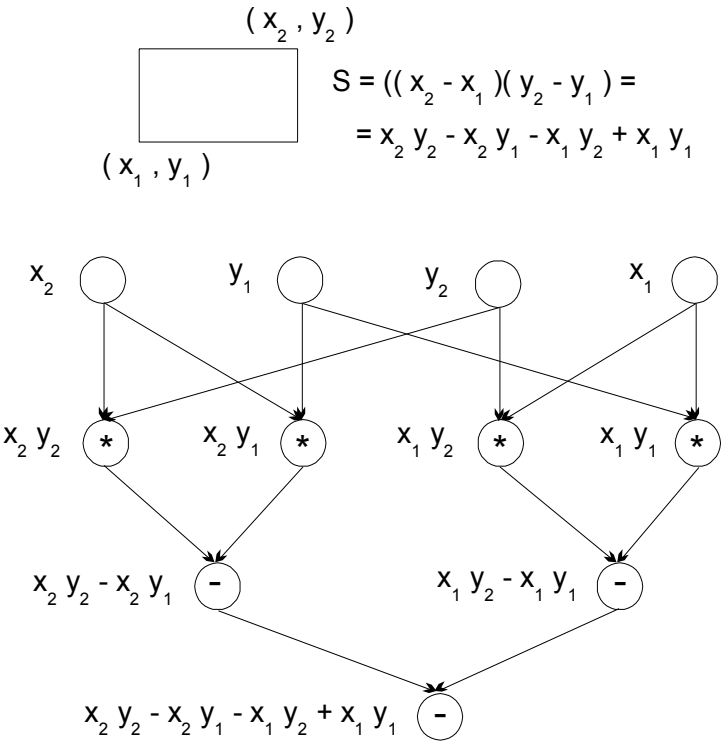

Рис. 2.1. Пример вычислительной модели алгоритма в виде графа «операции-операнды»

1) $\forall i, j \in V : t_i = t_j \Rightarrow P_i \neq P_j$, т. е. один и тот же процессор не должен назначаться разным операциям в один и тот же момент времени,

2) $\forall (i, j) \in R \Rightarrow t_j \geq t_i + 1$, т. е. к назначаемому моменту выполнения операции все необходимые данные уже должны быть вычислены.

2.3. Определение времени выполнения параллельного алгоритма

Вычислительная схема алгоритма G совместно с расписанием H_p может рассматриваться как модель параллельного алгоритма $A_p(G, H_p)$, исполняемого с использованием p процессоров. Время выполнения па-

раллельного алгоритма определяется максимальным значением времени, используемым в расписании

$$T_p(G, H_p) = \max_{i \in V}(t_i + 1).$$

Для выбранной схемы вычислений желательно использование расписания, обеспечивающего минимальное время исполнения алгоритма

$$T_p(G) = \min_{H_p} T_p(G, H_p).$$

Уменьшение времени выполнения может быть обеспечено и путем подбора наилучшей вычислительной схемы

$$T_p = \min_{G} T_p(G).$$

Оценки $T_p(G, H_p)$, $T_p(G)$ и T_p могут быть использованы в качестве показателей времени выполнения параллельного алгоритма. Кроме того, для анализа максимально возможного параллелизма можно определить оценку наиболее быстрого исполнения алгоритма

$$T_\infty = \min_{p \geq 1} T_p.$$

Оценку T_∞ можно рассматривать как минимально возможное время выполнения параллельного алгоритма при использовании неограниченного количества процессоров (концепция вычислительной системы с бесконечным количеством процессоров, обычно называемой *паракомпьютером*, широко используется при теоретическом анализе параллельных вычислений).

Оценка T_1 определяет время выполнения алгоритма при использовании одного процессора и представляет, тем самым, время выполнения последовательного варианта алгоритма решения задачи. Построение подобной оценки является важной задачей при анализе параллельных алгоритмов, поскольку она необходима для определения эффекта использования параллелизма (ускорения времени решения задачи). Очевидно, что

$$T_1(G) = |\bar{V}|,$$

где $|\bar{V}|$, напомним, есть количество вершин вычислительной схемы G без вершин ввода. Важно отметить, что если при определении оценки T_1 ограничиться рассмотрением только одного выбранного алгоритма решения задачи и использовать величину

$$T_1 = \min_{G} T_1(G),$$

то получаемые при использовании такой оценки показатели ускорения будут характеризовать эффективность распараллеливания выбранного алго-

ритма. Для оценки эффективности параллельного решения исследуемой задачи вычислительной математики время последовательного решения следует определять с учетом различных последовательных алгоритмов, т. е. использовать величину

$$T_1^* = \min T_1,$$

где операция минимума берется по множеству всех возможных последовательных алгоритмов решения данной задачи.

Приведем без доказательства теоретические положения, характеризующие свойства оценок времени выполнения параллельного алгоритма (см. [44]).

Теорема 1. Минимально возможное время выполнения параллельного алгоритма определяется длиной максимального пути вычислительной схемы алгоритма, т. е.

$$T_\infty(G) = d(G).$$

Теорема 2. Пусть для некоторой вершины вывода в вычислительной схеме алгоритма существует путь из каждой вершины ввода. Кроме того, пусть входная степень вершин схемы (количество входящих дуг) не превышает 2. Тогда минимально возможное время выполнения параллельного алгоритма ограничено снизу значением

$$T_\infty(G) = \log_2 n,$$

где n есть количество вершин ввода в схеме алгоритма.

Теорема 3. При уменьшении числа используемых процессоров время выполнения алгоритма увеличивается пропорционально величине уменьшения количества процессоров, т. е.

$$\forall q = cp, \quad 0 < c < 1 \Rightarrow T_p \le cT_q.$$

Теорема 4. Для любого количества используемых процессоров справедлива следующая верхняя оценка для времени выполнения параллельного алгоритма

$$\forall p \Rightarrow T_p < T_\infty + T_1 / p.$$

Теорема 5. Времени выполнения алгоритма, которое сопоставимо с минимально возможным временем T_∞, можно достичь при количестве процессоров порядка $p \sim T_1 / T_\infty$, а именно,

$$p \geq T_1 / T_\infty \Rightarrow T_p \leq 2T_\infty .$$

При меньшем количестве процессоров время выполнения алгоритма не может превышать более, чем в 2 раза, наилучшее время вычислений при имеющемся числе процессоров, т. е.

$$p < T_1 / T_\infty \Rightarrow \frac{T_1}{p} \leq T_p \leq 2\frac{T_1}{p}.$$

Приведенные утверждения позволяют дать следующие рекомендации по правилам формирования параллельных алгоритмов:

1) при выборе вычислительной схемы алгоритма должен использоваться граф с минимально возможным диаметром (см. теорему 1);

2) для параллельного выполнения целесообразное количество процессоров определяется величиной $p \sim T_1 / T_\infty$ (см. теорему 5);

3) время выполнения параллельного алгоритма ограничивается сверху величинами, приведенными в теоремах 4 и 5.

Для вывода рекомендаций по формированию расписания по параллельному выполнению алгоритма приведем доказательство теоремы 4.

Доказательство теоремы 4. Пусть H_∞ есть расписание для достижения минимально возможного времени выполнения T_∞. Для каждой итерации τ, $0 \leq \tau \leq T_\infty$, выполнения расписания H_∞ обозначим через n_τ количество операций, выполняемых в ходе итерации τ. Расписание выполнения алгоритма с использованием p процессоров может быть построено следующим образом. Выполнение алгоритма разделим на T_∞ шагов; на каждом шаге τ следует выполнить все n_τ операций, которые выполнялись на итерации τ расписания H_∞. Выполнение этих операций может быть выполнено не более, чем за $\lceil n_\tau / p \rceil$ итераций при использовании p процессоров. Как результат, время выполнения алгоритма T_p может быть оценено следующим образом

$$T_p = \sum_{\tau=1}^{T_\infty} \left\lceil \frac{n_\tau}{p} \right\rceil < \sum_{\tau=1}^{T_\infty} \left(\frac{n_\tau}{p} + 1 \right) = \frac{T_1}{p} + T_\infty .$$

Доказательство теоремы дает практический способ построения расписания параллельного алгоритма. Первоначально может быть построено расписание без учета ограниченности числа используемых процессоров (расписание для паракомпьютера). Затем, согласно схеме вывода теоремы,

может быть построено расписание для конкретного количества процессоров.

2.4. Показатели эффективности параллельного алгоритма

Ускорение (speedup), получаемое при использовании параллельного алгоритма для p процессоров, по сравнению с последовательным вариантом выполнения вычислений определяется величиной

$$S_p(n) = T_1(n) / T_p(n),$$

т. е. как отношение времени решения задач на скалярной ЭВМ к времени выполнения параллельного алгоритма (величина n используется для параметризации вычислительной сложности решаемой задачи и может пониматься, например, как количество входных данных задачи).

Эффективность (efficiency) использования параллельным алгоритмом процессоров при решении задачи определяется соотношением

$$E_p(n) = T_1(n)/(pT_p(n)) = S_p(n)/p$$

(величина эффективности определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально используются для решения задачи).

Из приведенных соотношений можно показать, что в наилучшем случае $S_p(n) = p$ и $E_p(n) = 1$. При практическом применении данных показателей для оценки эффективности параллельных вычислений следует учитывать следующие два важные момента:

- При определенных обстоятельствах ускорение может оказаться больше числа используемых процессоров $S_p(n) > p$ – в этом случае говорят о существовании *сверхлинейного* (*superlinear*) ускорения. Несмотря на парадоксальность таких ситуаций (ускорение превышает число процессоров), на практике сверхлинейное ускорение может иметь место. Одной из причин такого явления может быть неравноправность выполнения последовательной и параллельной программ. Например, при решении задачи на одном процессоре оказывается недостаточно оперативной памяти для хранения всех обрабатываемых данных и, как результат, необходимым становится использование более медленной внешней памяти (в случае же использования нескольких процессоров оперативной памяти может оказаться достаточно за счет разделения данных между процессорами). Еще одной причиной сверхлинейного ускорения может быть нелинейный характер зависимости сложности решения задачи в зависимости от объема обрабатываемых данных. Так, например, известный алгоритм пузырьковой сор-

тировки характеризуется квадратичной зависимостью количества необходимых операций от числа упорядочиваемых данных. Как результат, при распределении сортируемого массива между процессорами может быть получено ускорение, превышающее число процессоров (более подробно данный пример рассматривается в гл. 9). Источником сверхлинейного ускорения может быть и различие вычислительных схем последовательного и параллельного методов.

- При внимательном рассмотрении можно обратить внимание, что попытки повышения качества параллельных вычислений по одному из показателей (ускорению или эффективности) может привести к ухудшению ситуации по другому показателю, ибо показатели качества параллельных вычислений являются противоречивыми. Например, повышение ускорения обычно может быть обеспечено за счет увеличения числа процессоров, что приводит, как правило, к падению эффективности. И обратно, повышение эффективности достигается во многих случаях при уменьшении числа процессоров (в предельном случае идеальная эффективность $E_p(n)=1$ легко обеспечивается при использовании одного процессора). Как результат, разработка методов параллельных вычислений часто предполагает выбор некоторого компромиссного варианта с учетом желаемых показателей ускорения и эффективности.

При выборе надлежащего параллельного способа решения задачи может оказаться полезной оценка ***стоимости*** (***cost***) вычислений, определяемой как произведение времени параллельного решения задачи и числа используемых процессоров

$$C_p = pT_p .$$

В этой связи можно определить понятие ***стоимостно-оптимального*** (***cost-optimal***) параллельного алгоритма как метода, стоимость которого является пропорциональной времени выполнения наилучшего последовательного алгоритма.

Далее для иллюстрации введенных понятий в следующем пункте будет рассмотрен учебный пример решения задачи вычисления частных сумм для последовательности числовых значений. Кроме того, данные показатели будут использоваться для характеристики эффективности всех рассматриваемых далее параллельных алгоритмов при решении типовых задач вычислительной математики.

2.5. Вычисление частных сумм последовательности числовых значений

Рассмотрим для демонстрации ряда проблем, возникающих при разработке параллельных методов вычислений, сравнительно простую *задачу нахождения частных сумм* последовательности числовых значений

$$S_k = \sum_{i=1}^{k} x_i, \quad 1 \le k \le n,$$

где n есть количество суммируемых значений (данная задача известна также под названием *prefix sum problem*).

Изучение возможных параллельных методов решения данной задачи начнем с еще более простого варианта ее постановки – с *задачи вычисления общей суммы* имеющегося набора значений (в таком виде задача суммирования является частным случаем общей *задачи редукции*)

$$S = \sum_{i=1}^{n} x_i .$$

2.5.1. Последовательный алгоритм суммирования

Традиционный алгоритм для решения этой задачи состоит в последовательном суммировании элементов числового набора

$$S = 0,$$
$$S = S + x_1, \ldots$$

Вычислительная схема данного алгоритма может быть представлена следующим образом (см. рис. 2.2):

$$G_1 = (V_1, R_1),$$

где $V_1 = \{v_{01}, \ldots, v_{0n}, v_{11}, \ldots, v_{1n}\}$ есть множество операций (вершины $v_{01}, \ldots, v_{0n}$ обозначают операции ввода, каждая вершина v_{1i}, $1 \le i \le n$, соответствует прибавлению значения x_i к накапливаемой сумме S), а

$$R_1 = \{(v_{0i}, v_{1i}), (v_{1i}, v_{1i+1}), \quad 1 \le i \le n-1\}$$

есть множество дуг, определяющих информационные зависимости операций.

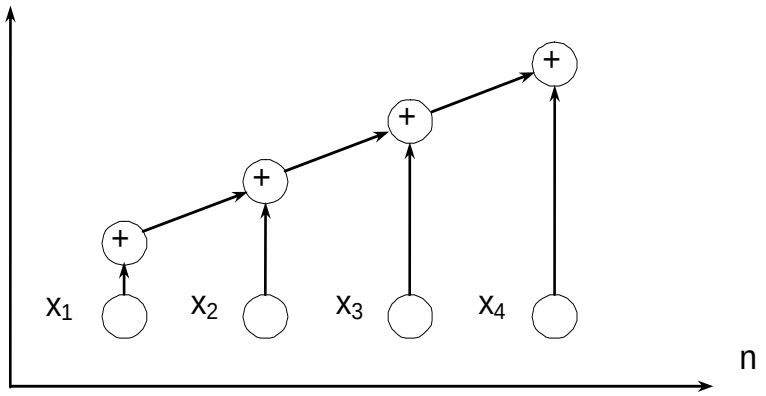

Рис. 2.2. Последовательная вычислительная схема алгоритма суммирования

Как можно заметить, данный «стандартный» алгоритм суммирования допускает только строго последовательное исполнение и не может быть распараллелен.

2.5.2. Каскадная схема суммирования

Параллелизм алгоритма суммирования становится возможным только при ином способе построения процесса вычислений, основанном на использовании ассоциативности операции сложения. Получаемый новый вариант суммирования (известный в литературе как *каскадная схема*) состоит в следующем (см. рис. 2.3):

– на первой итерации каскадной схемы все исходные данные разбиваются на пары, и для каждой пары вычисляется сумма их значений,

– далее все полученные суммы также разбиваются на пары, и снова выполняется суммирование значений пар и т. д.

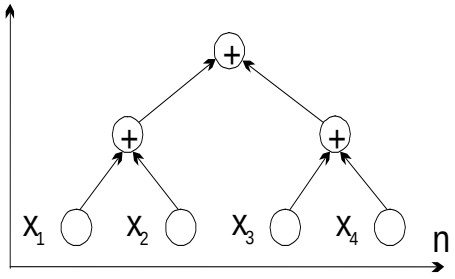

Рис. 2.3. Каскадная схема алгоритма суммирования

Эта вычислительная схема может быть определена как граф (пусть $n = 2^k$)

$$G_2 = (V_2, R_2),$$

где $V_2 = \{(v_{i1},...,v_{il_i}),\ \ 0 \le i \le k,\ \ 1 \le l_i \le 2^{-i} n\}$ есть вершины графа ($(v_{01},...,v_{0n})$ – операции ввода, $(v_{11},...,v_{1n/2})$ – операции первой итерации и т. д.), а множество дуг графа определяется соотношениями:

$$R_2 = \{(v_{i-1,2j-1} v_{ij}), (v_{i-1,2j} v_{ij}),\ \ 1 \le i \le k,\ \ 1 \le j \le 2^{-i} n\}.$$

Как нетрудно оценить, количество итераций каскадной схемы оказывается равным величине

$$k = \log_2 n,$$

а общее количество операций суммирования

$$\mathrm{K}_{посл} = n/2 + n/4 + ... + 1 = n - 1$$

совпадает с количеством операций последовательного варианта алгоритма суммирования. При параллельном исполнении отдельных итераций каскадной схемы общее количество параллельных операций суммирования является равным

$$\mathrm{K}_{пар} = \log_2 n.$$

Поскольку считается, что время выполнения любых вычислительных операций является одинаковым и единичным, то $T_1 = \mathrm{K}_{посл}$, $T_p = \mathrm{K}_{пар}$, поэтому показатели ускорения и эффективности каскадной схемы алгоритма суммирования можно оценить как

$$S_P = T_1 / T_P = (n-1) / \log_2 n,$$

$$E_p = T_1 / pT_p = (n-1) / (p \log_2 n) = (n-1) / ((n/2) \log_2 n),$$

где $p = n/2$ есть необходимое для выполнения каскадной схемы количество процессоров.

Анализируя полученные характеристики, можно отметить, что время параллельного выполнения каскадной схемы совпадает с оценкой для паракомпьютера в теореме 2. Однако при этом эффективность использования процессоров уменьшается при увеличении количества суммируемых значений

$$. \lim_{n \to \infty} E_p \to 0.$$

2.5.3. Модифицированная каскадная схема

Получение асимптотически ненулевой эффективности может быть обеспечено, например, при использовании модифицированной каскадной схемы (см. [44]). Для упрощения построения оценок можно предположить $n = 2^k, k = 2^s$. Тогда в новом варианте каскадной схемы все проводимые вычисления подразделяются на два последовательно выполняемых этапа суммирования (см. рис. 2.4):

– на первом этапе вычислений все суммируемые значения подразделяются на $(n/\log_2 n)$ групп, в каждой из которых содержится $\log_2 n$ элементов; далее для каждой группы вычисляется сумма значений при помощи последовательного алгоритма суммирования; вычисления в каждой группе могут выполняться независимо друг от друга (т. е. параллельно – для этого необходимо наличие не менее $(n/\log_2 n)$ процессоров);

– на втором этапе для полученных $(n/\log_2 n)$ сумм отдельных групп применяется обычная каскадная схема.

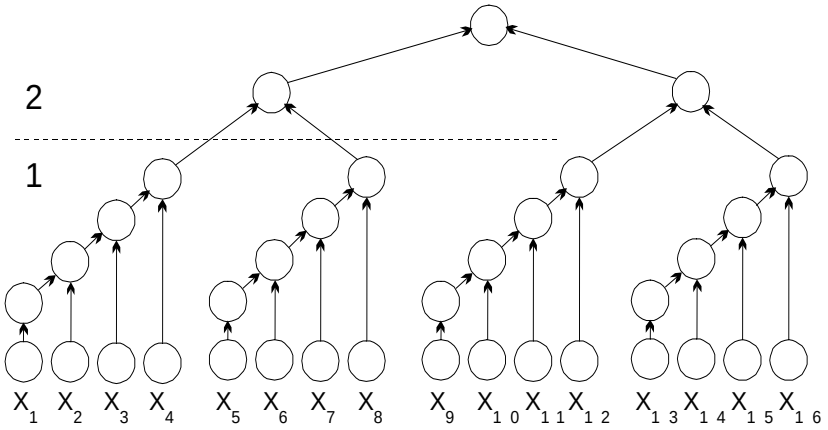

Рис. 2.4. Модифицированная каскадная схема суммирования

Тогда для выполнения первого этапа требуется выполнение $\log_2 n$ параллельных операций при использовании $p_1 = (n/\log_2 n)$ процессоров. Для выполнения второго этапа необходимо

$$\log_2 (n/\log_2 n) \le \log_2 n$$

параллельных операций для $p_2 = (n/\log_2 n)/2$ процессоров. Как результат, данный способ суммирования характеризуется следующими показателями:

$$T_P = 2\log_2 n\,,\ p = (n/\log_2 n)\,.$$

С учетом полученных оценок показатели ускорения и эффективности модифицированной каскадной схемы определяются соотношениями:

$$S_P = T_1 / T_P = (n-1)/2\log_2 n,$$

$$E_p = T_1 / pT_p = (n-1)/(2(n/\log_2 n)\log_2 n) = (n-1)/2n.$$

Сравнивая данные оценки с показателями обычной каскадной схемы, можно отметить, что ускорение для предложенного параллельного алгоритма уменьшилось в 2 раза, однако для эффективности нового метода суммирования можно получить асимптотически ненулевую оценку снизу

$$E_P = (n-1)/2n \geq 0.25,\ \lim_{n\to\infty} E_p \to 0.5.\ .$$

Можно отметить также, что данные значения показателей достигаются при количестве процессоров, определенном в теореме 5. Кроме того, необходимо подчеркнуть, что в отличие от обычной каскадной схемы, модифицированный каскадный алгоритм является стоимостно-оптимальным, поскольку стоимость вычислений в этом случае

$$C_p = pT_p = (n/\log_2 n)(2\log_2 n)$$

является пропорциональной времени выполнения последовательного алгоритма.

2.5.4. Вычисление всех частных сумм

Вернемся к исходной задаче вычисления всех частных сумм последовательности значений и проведем анализ возможных способов последовательной и параллельной организации вычислений. Вычисление всех частных сумм на скалярном компьютере может быть получено при помощи обычного последовательного алгоритма суммирования при том же количестве операций (!)

$$T_1 = n\,.$$

При параллельном исполнении применение каскадной схемы в явном виде не приводит к желаемым результатам; достижение эффективного распараллеливания требует привлечения новых подходов (может быть, даже не имеющих аналогов при последовательном программировании) для разработки новых параллельно-ориентированных алгоритмов решения задач. Так, для рассматриваемой задачи нахождения всех частных сумм, алго-

ритм, обеспечивающий получение результатов за $\log_2 n$ параллельных операций (как и в случае вычисления общей суммы), может состоять в следующем (см. рис. 2.5) [44]:

– перед началом вычислений создается копия S вектора суммируемых значений ($S = x$);

– далее на каждой итерации суммирования i, $1 \le i \le \log_2 n$, формируется вспомогательный вектор Q путем сдвига вправо вектора S на 2^{i-1} позиций (освобождающиеся при сдвиге позиции слева устанавливаются в нулевые значения); итерация алгоритма завершается параллельной операцией суммирования векторов S и Q.

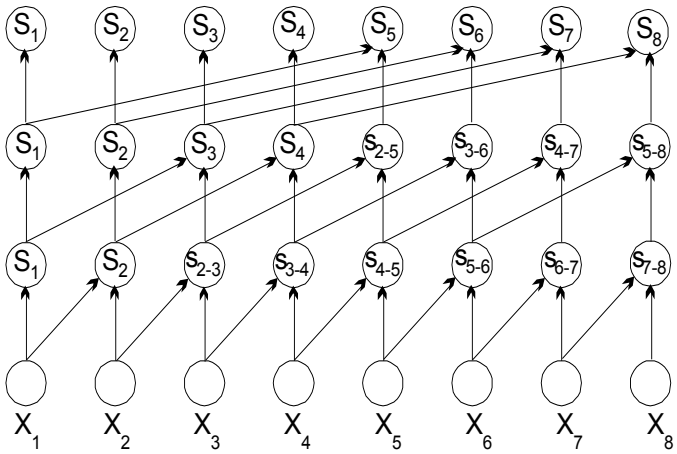

Рис. 2.5. Схема параллельного алгоритма вычисления всех частных сумм (величины S_{i-j} означают суммы значений от i до j элементов числовой последовательности)

Всего параллельный алгоритм выполняется за $\log_2 n$ параллельных операций сложения. На каждой итерации алгоритма параллельно выполняются n скалярных операций сложения и, таким образом, общее количество выполняемых скалярных операций определяется величиной

$$K_{пар} = n \log_2 n$$

(параллельный алгоритм содержит большее (!) количество операций по сравнению с последовательным способом суммирования). Необходимое количество процессоров определяется количеством суммируемых значений ($p = n$).

С учетом полученных соотношений, показатели ускорения и эффективности параллельного алгоритма вычисления всех частных сумм оцениваются следующим образом

$$S_P = T_1 / T_P = n / \log_2 n,$$

$$E_P = T_1 / pT_P = n/(p \log_2 n) = n/(n \log_2 n) = 1/\log_2 n .$$

Как следует из построенных оценок, эффективность алгоритма также уменьшается при увеличении числа суммируемых значений и при необходимости повышения величины этого показателя может оказаться полезной модификация алгоритма, как и в случае с обычной каскадной схемой.

2.6. Оценка максимально достижимого параллелизма

Оценка качества параллельных вычислений предполагает знание *наилучших* (*максимально достижимых*) значений показателей ускорения и эффективности, однако получение идеальных величин $S_p = p$ для ускорения и $E_p = 1$ для эффективности может быть обеспечено не для всех вычислительно трудоемких задач. Так, для рассматриваемого учебного примера в предыдущем пункте минимально достижимое время параллельного вычисления суммы числовых значений составляет $\log_2 n$. Определенное содействие в решении данной проблемы могут оказать теоретические утверждения, приведенные в начале данной главы. В дополнение к ним рассмотрим еще ряд закономерностей, которые могут быть чрезвычайно полезны при построении оценок максимально достижимого параллелизма.

1. Закон Амдаля. Достижению максимального ускорения может препятствовать существование в выполняемых вычислениях последовательных расчетов, которые не могут быть распараллелены. Пусть f есть *доля последовательных вычислений* в применяемом алгоритме обработки данных; тогда в соответствии с *законом Амдаля* (*Amdahl*) ускорение процесса вычислений при использовании p процессоров ограничивается величиной

$$S_p \le \frac{1}{f + (1-f)/p} \le S^* = \frac{1}{f}.$$

Так, например, при наличии всего 10% последовательных команд в выполняемых вычислениях, эффект использования параллелизма не может превышать 10-кратного ускорения обработки данных. Для рассмотренного учебного примера вычисления суммы значений для каскадной схемы доля последовательных расчетов составляет $f = \log_2 n / n$ и, как результат, величина возможного ускорения ограничена оценкой $S^* = n / \log_2 n$.

Закон Амдаля характеризует одну из самых серьезных проблем в области параллельного программирования (алгоритмов без определенной доли последовательных команд практически не существует). Однако часто доля последовательных действий характеризует не возможность параллельного решения задач, а последовательные свойства применяемых алгоритмов. Как результат, доля последовательных вычислений может быть существенно снижена при выборе более подходящих для распараллеливания методов.

Следует отметить также, что рассмотрение закона Амдаля происходит при предположении, что доля последовательных расчетов f является постоянной величиной и не зависит от параметра n, определяющего вычислительную сложность решаемой задачи. Однако для большого ряда задач доля $f = f(n)$ является убывающей функцией от n, и в этом случае ускорение для фиксированного числа процессоров может быть увеличено за счет увеличения вычислительной сложности решаемой задачи. Данное замечание может быть сформулировано как утверждение, что ускорение $S_p = S_p(n)$ является возрастающей функцией от параметра n (данное утверждение часто именуется как *эффект Амдаля*). Так, например, для учебного примера вычисления суммы значений при использовании фиксированного числа процессоров p суммируемый набор данных может быть разделен на блоки размера n/p, для которых сначала параллельно могут быть вычислены частные суммы, а далее эти суммы можно сложить при помощи каскадной схемы. Длительность последовательной части выполняемых операций (минимально-возможное время параллельного исполнения) в этом случае составляет

$$T_p = (n/p) + \log_2 p,$$

что приводит к оценке доли последовательных расчетов как величины

$$f = (1/p) + \log_2 p / n.$$

Как следует из полученного выражения, доля последовательных расчетов f убывает с ростом n и в предельном случае мы получаемом идеальную оценку максимально возможного ускорения $S^* = p$.

2. Закон Густавсона–Барсиса. Оценим максимально достижимое ускорение, исходя из имеющейся доли последовательных расчетов в выполняемых параллельных вычислениях:

$$g = \frac{\tau(n)}{\tau(n) + \pi(n)/p},$$

где $\tau(n)$ и $\pi(n)$ есть времена последовательной и параллельной частей выполняемых вычислений соответственно, т. е.

$$T_1 = \tau(n) + \pi(n),\ T_p = \tau(n) + \pi(n)/p.$$

С учетом введенной величины g можно получить:

$$\tau(n) = g \cdot (\tau(n) + \pi(n)/p),\ \ \pi(n) = (1-g)p \cdot (\tau(n) + \pi(n)/p),$$

что позволяет построить оценку для ускорения:

$$S_p = \frac{T_1}{T_p} = \frac{\tau(n) + \pi(n)}{\tau(n) + \pi(n)/p} = \frac{(\tau(n) + \pi(n)/p)(g + (1-g)p)}{\tau(n) + \pi(n)/p},$$

которая после упрощения приводится к виду *закона Густавсона–Барсиса* (*Gustafson–Barsis's law*): [85]

$$S_p = g + (1-g)p = p + (1-p)g.$$

Применительно к учебному примеру суммирования значений при использовании p процессоров время параллельного выполнения, как уже отмечалось выше, составляет:

$$T_p = (n/p) + \log_2 p,$$

что соответствует последовательной доле:

$$g = \frac{\log_2 p}{(n/p) + \log_2 p}.$$

За счет увеличения числа суммируемых значений величина g может быть пренебрежимо малой, обеспечивая получение идеального возможного ускорения $S_p = p$.

При рассмотрении закона Густавсона–Барсиса следует учитывать еще один важный момент. При увеличении числа используемых процессоров темп уменьшения времени параллельного решения задач может падать (после превышения определенного порога). Однако при этом за счет уменьшения времени вычислений сложность решаемых задач может быть увеличена (так, например, для учебной задачи суммирования может быть увеличен размер складываемого набора значений). Оценку получаемого при этом ускорения можно определить при помощи сформулированных закономерностей. Такая аналитическая оценка тем более полезна, поскольку решение таких более сложных вариантов задач на одном процессоре может оказаться достаточно трудоемким и даже невозможным, например, в силу нехватки оперативной памяти. С учетом указанных обстоятельств оценку ускорения, получаемую в соответствии с законом Густавсона–Барсиса, еще называют *ускорением масштабирования* (*scaled speedup*), по-

скольку данная характеристика может показать, насколько эффективно могут быть организованы параллельные вычисления при увеличении сложности решаемых задач.

2.7. Анализ масштабируемости параллельных вычислений

Целью применения параллельных вычислений во многих случаях является не только уменьшение времени выполнения расчетов, но и обеспечение возможности решения более сложных вариантов решаемых задач (таких постановок, решение которых не представляется возможным при использовании однопроцессорных вычислительных систем). Способность параллельного алгоритма эффективно использовать процессоры при повышении сложности вычислений является важной характеристикой выполняемых расчетов. В связи с этим параллельный алгоритм называют *масштабируемым* (*scalable*), если при росте числа процессоров он обеспечивает увеличение ускорения при сохранении постоянного уровня эффективности использования процессоров. Возможный способ характеристики свойств масштабируемости состоит в следующем.

Оценим *накладные расходы* (*total overhead*), которые имеют место при выполнении параллельного алгоритма

$$T_0 = pT_p - T_1.$$

Накладные расходы появляются за счет необходимости организации взаимодействия процессоров, выполнения некоторых дополнительных действий, синхронизации параллельных вычислений и т. п. Используя введенное обозначение, можно получить новые выражения для времени параллельного решения задачи и соответствующего ускорения:

$$T_p = \frac{T_1 + T_0}{p}, \quad S_p = \frac{T_1}{T_p} = \frac{pT_1}{T_1 + T_0}.$$

Используя полученные соотношения, эффективность использования процессоров можно выразить как:

$$E_p = \frac{S_p}{p} = \frac{T_1}{T_1 + T_0} = \frac{1}{1 + T_0 / T_1}.$$

Последнее выражение показывает, что если сложность решаемой задачи является фиксированной (T_1=*const*), то при росте числа процессоров эффективность, как правило, будет убывать за счет роста накладных расходов T_0. При фиксации числа процессоров эффективность использования процессоров

можно улучшить путем повышения сложности решаемой задачи T_1 (предполагается, что при росте параметра сложности n накладные расходы T_0 увеличиваются медленнее, чем объем вычислений T_1). Как результат, при увеличении числа процессоров в большинстве случаев можно обеспечить определенный уровень эффективности при помощи соответствующего повышения сложности решаемых задач. В этой связи важной характеристикой параллельных вычислений становится соотношение необходимых темпов роста сложности расчетов и числа используемых процессоров.

Пусть $E = const$ есть желаемый уровень эффективности выполняемых вычислений. Из выражения для эффективности можно получить:

$$\frac{T_0}{T_1} = \frac{1-E}{E} \quad \text{или} \quad T_1 = KT_0,\ K = E/(1-E).$$

Порождаемую последним соотношением зависимость $n = F(p)$ между сложностью решаемой задачи и числом процессоров обычно называют *функцией изоэффективности* (*isoefficiency function*) [72].

Покажем в качестве иллюстрации вывод функции изоэффективности для учебного примера суммирования числовых значений. В этом случае:

$$T_0 = pT_p - T_1 = p((n/p) + \log_2 p) - n = p\log_2 p$$

и функция изоэффективности принимает вид:

$$n = Kp\log_2 p.$$

Как результат, например, при числе процессоров $p = 16$ для обеспечения уровня эффективности $E = 0.5$ (т. е. $K = 1$) количество суммируемых значений должно быть не менее $n = 64$. Или же, при увеличении числа процессоров с p до q $(q > p)$ для обеспечения пропорционального роста ускорения $(S_q/S_p) = (q/p)$ необходимо увеличить число суммируемых значений n в $(qlog_2q)/(plog_2p)$ раз.

2.8. Краткий обзор главы

В данной главе описана модель вычислений в виде графа «операции–операнды», которая может использоваться для описания существующих информационных зависимостей в выбираемых алгоритмах решения задач. В основу данной модели положен ациклический ориентированный граф, в котором вершины представляют операции, а дуги соответствуют зависимостям операций по данным. При наличии такого графа для определения парал-

лельного алгоритма достаточно задать расписание, в соответствии с которым фиксируется распределение выполняемых операций по процессорам.

Представление вычислений при помощи моделей подобного вида позволяет получить аналитически ряд характеристик разрабатываемых параллельных алгоритмов, среди которых время выполнения, схема оптимального расписания, оценки максимально возможного быстродействия методов решения поставленных задач. Для возможности более простого построения теоретических оценок в главе рассматривается понятие *паракомпьютера* как параллельной системы с неограниченным количеством процессоров.

Для оценки оптимальности разрабатываемых методов параллельных вычислений приведены широко используемые в теории и практике параллельного программирования основные показатели качества – ***ускорение (speedup)***, показывающее, во сколько раз быстрее осуществляется решение задач при использовании нескольких процессоров, и ***эффективность (efficiency)***, которая характеризует долю времени реального использования процессоров вычислительной системы. Важной характеристикой разрабатываемых алгоритмов является ***стоимость*** (***cost***) вычислений, определяемая как произведение времени параллельного решения задачи и числа используемых процессоров.

Для демонстрации применимости рассмотренных моделей и методов анализа параллельных алгоритмов рассмотрена задача нахождения частных сумм последовательности числовых значений. На данном примере отмечается проблема сложности распараллеливания последовательных алгоритмов, которые изначально не были ориентированы на возможность организации параллельных вычислений. Для выделения «скрытого» параллелизма показана возможность преобразования исходной последовательной схемы вычислений и приведена получаемая в результате таких преобразований каскадная схема. На примере этой же задачи отмечается возможность введения избыточных вычислений для достижения большего параллелизма выполняемых расчетов.

В завершение главы рассмотрен вопрос построения оценок максимально достижимых значений показателей эффективности. Для получения таких оценок может быть использован *закон Амдаля* (*Amdahl*), позволяющий учесть существование последовательных (нераспараллеливаемых) вычислений в методах решения задач. *Закон Густавсона–Барсиса* (*Gustafson–Barsis's law*) обеспечивает построение оценок *ускорения масштабирования* (*scaled speedup*), используемое для характеристики того, насколько эффективно могут быть организованы параллельные вычисления при увеличении сложности решаемых задач. Для определения зависимости между сложностью решаемой задачи и числом процессоров, при соблюдении которой обеспечивается необходимый уровень эффективности параллельных вычислений, вводится понятие *функции изоэффективности* (*isoefficiency function*).

2.9. Обзор литературы

Дополнительная информация по моделированию и анализу параллельных вычислений может быть получена, например, в [8,10,44], полезная информация содержится также в [72,85]. Рассмотрение учебной задачи суммирования последовательности числовых значений было выполнено в [44].

Впервые закон Амдаля был изложен в работе [38]. Закон Густавсона–Барсиса был опубликован в работе [63]. Понятие функции изоэффективности было предложено в работе [61].

Систематическое изложение вопросов моделирования и анализа параллельных вычислений приводится в [102].

2.10. Контрольные вопросы

1. Как определяется модель «операция–операнды»?
2. Как определяется расписание для распределения вычислений между процессорами?
3. Как определяется время выполнения параллельного алгоритма?
4. Какое расписание является оптимальным?
5. Как определить минимально возможное время решения задачи?
6. Что понимается под паракомпьютером и для чего может оказаться полезным данное понятие?
7. Какие оценки следует использовать в качестве характеристики времени последовательного решения задачи?
8. Как определить минимально возможное время параллельного решения задачи по графу «операнды–операции»?
9. Какие зависимости могут быть получены для времени параллельного решения задачи при увеличении или уменьшения числа используемых процессоров?
10. При каком числе процессоров могут быть получены времена выполнения параллельного алгоритма, сопоставимые по порядку с оценками минимально возможного времени решения задачи?
11. Как определяются понятия ускорения и эффективности?
12. Возможно ли достижений сверхлинейного ускорения?
13. В чем состоит противоречивость показателей ускорения и эффективности?
14. Как определяется понятие стоимости вычислений?
15. В чем состоит понятие стоимостно-оптимального алгоритма?
16. В чем заключается проблема распараллеливания последовательного алгоритма суммирования числовых значений?

17. В чем состоит каскадная схема суммирования? С какой целью рассматривается модифицированный вариант данной схемы?

18. В чем различие показателей ускорения и эффективности для рассматриваемых вариантов каскадной схемы суммирования?

19. В чем состоит параллельный алгоритм вычисления всех частных сумм последовательности числовых значений?

20. Как формулируется закон Амдаля? Какой аспект параллельных вычислений позволяет учесть данный закон?

21. Какие предположения используются для обоснования закона Густавсона–Барсиса?

22. Как определяется функция изоэффективности?

23. Какой алгоритм является масштабируемым? Приведите примеры методов с разным уровнем масштабируемости.

2.11. Задачи и упражнения

1. Разработайте модель и выполните оценку показателей ускорения и эффективности параллельных вычислений:

- для задачи скалярного произведения двух векторов

$$y = \sum_{i=1}^{N} a_i b_i ,$$

- для задачи поиска максимального и минимального значений для заданного набора числовых данных

$$y_{\min} = \min_{i \le i \le N} a_i , y_{\max} = \max_{i \le i \le N} a_i ,$$

- для задачи нахождения среднего значения для заданного набора числовых данных

$$y = \frac{1}{N} \sum_{i=1}^{N} a_i .$$

2. Выполните в соответствии с законом Амдаля оценку максимально достижимого ускорения для задач п. 1.

3. Выполните оценку ускорения масштабирования для задач п.1.

4. Выполните построение функций изоэффективности для задач п. 1.

5. (*) Разработайте модель и выполните полный анализ эффективности параллельных вычислений (ускорение, эффективность, максимально достижимое ускорение, ускорение масштабирования, функция изоэффективности) для задачи умножения матрицы на вектор.