С. В. Борзунов, С. Д. Кургалин, А. В. Флегель

# Практикум по параллельному программированию

- *Теоретическая часть: методы параллельного программирования и средства распараллеливания*
- *Разработка программ с помощью технологий OpenMP и MPI*
- *Большое число разобранных задач, примеров и упражнений*
- *Контрольные вопросы, указания к решениям, ответы*
- *Дополнительная справочная информация*

bhv®

УДК 519.6+004.42(075.8)
ББК 32.973я73
Б82

**Борзунов, С. В.**

Б82 Практикум по параллельному программированию: учеб. пособие / С. В. Борзунов, С. Д. Кургалин, А. В. Флегель. — СПб.: БХВ, 2017. — 236 с.: ил. — (Учебная литература для вузов)

ISBN 978-5-9909805-0-1

В учебное пособие включены основные теоретические сведения о методах программирования для многопроцессорных вычислительных систем, указания по разработке параллельных программ с помощью технологий OpenMP и MPI, а также контрольные вопросы и задачи широкого спектра сложности как для проведения занятий в компьютерных классах и аудиториях, так и для самостоятельного решения. Многие задачи снабжены ответами или решениями, в том числе с образцами кода.

*Для студентов, аспирантов*
*и преподавателей профильных вузов*

УДК 519.6+004.42(075.8)
ББК 32.973я73

РЕЦЕНЗЕНТЫ:

*В. П. Гергель*, д-р тех. наук, проф., зав. кафедрой программной инженерии Нижегородского государственного университета, директор Института информационных технологий, математики и механики;

Кафедра высшей математики и физико-математического моделирования Воронежского государственного технического университета (зав. кафедрой д-р физ.-мат. наук, проф. *И. Л. Батаронов*)

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-5-9909805-0-1

# Оглавление

# Глава 1

# Классификация архитектур вычислительных систем

Разработка программ для многопроцессорных вычислительных систем или, как говорят, параллельное программирование, в настоящее время привлекает все большее внимание как исследователей, так и прикладных программистов. К этому приводят следующие причины.

Во-первых, в нашей стране (и в мире в целом) возникают все новые суперкомпьютерные центры, оснащаемые многопроцессорными вычислительными системами, в которых используются методы параллельного программирования. Суперкомпьютерные центры привлекаются для решения наиболее сложных задач, в вузах — и для подготовки современных специалистов в области информационных технологий самого высокого уровня. Во-вторых, большинство широко распростаненных компьютерных систем снабжается многоядерными процессорами, и, кроме того, в них присутствуют несколько (два, четыре, восемь и более) процессоров. В-третьих, задачи, с которыми приходится иметь дело программистам, имеют тенденцию к резкому повышению требований по отношению к ресурсам вычислительной системы как по сложности вычислений, так и по объему памяти.

В силу названных причин актуальным является освоение современных суперкомпьютерных средств с максимальным использованием в них ресурсов параллелизма.

Трудности, встречающиеся при разработке обычных (последовательных) версий программ, безусловно, остаются и при создании параллельных версий таких программ. Кроме того, к ним добавляются проблемы, отражающие особенности решения задач на многопроцессорных вычислительных системах. Конечно, желательно иметь возможность пользоваться стандартными средствами распараллеливания, однако они, как

будет показано ниже, обладают целым рядом недостатков, главным из которых является не слоишком высокая эффективность таких программ. В противном случае возникают проблемы, связанные с сопровождением и переносимостью программных комплексов. К счастью, такие средства реализованы для многих параллельных архитектур. Для дальнейшего рассмотрения нам потребуется классификация архитектур параллельных вычислительных систем.

## 1.1. Классификация Флинна

Большое разнообразие существующих архитектур вычислительных систем приводит к необходимости их классификации по различным параметрам. Исторически один из первых способов разделения архитектур по критерию множественности потоков команд и потоков данных был предложен Флинном[1].

**Поток** определяется как последовательность команд или данных, выполняемых или обрабатываемых процессором [47, 50]. С этой точки зрения программа предоставляет процессору поток инструкций для исполнения; данные для обработки поступают также в виде потока. В соответствии с классификацией Флинна потоки команд и потоки данных предполагаются независимыми. В связи с этим вычислительные системы разделяют на следующие классы.

1. Один поток команд, один поток данных (Single Instruction stream, Single Data stream; SISD). Такими характеристиками обладают стандартные компьютеры с одноядерным процессором, которые в каждый момент времени могут выполнять только одно действие.

2. Один поток команд, несколько потоков данных (Single Instruction stream, Multiple Data stream; SIMD). В таких системах одна и та же операция выполняется одновременно над различными данными. К данному классу относят, например, векторные вычислительные системы, в которых одна команда может выполняться над множеством элементов данных.

3. Несколько потоков команд, один поток данных (Multiple Instruction stream, Single Data stream; MISD). Несмотря на недостаточную практическую значимость данного подхода, машины MISD могут быть полезны в некоторых узкоспециальных задачах.

[1] Флинн (Michael J. Flynn) (род. 1934) — американский исследователь, специалист в области архитектур вычислительных систем.

4. Несколько потоков команд, несколько потоков данных (Multiple Instruction stream, Multiple Data stream; MIMD). Системы MIMD составляют наиболее обширную и разнообразную группу в классификации Флинна. Большинство современных многопроцессорных вычислительных систем относится именно к этому классу.

Свойства рассмотренных классов вычислительных систем схематично представлены в табл. 1.1. В ячейках записаны примеры арифметических операций, доступных для одновременного выполнения соответствующими системами.

Таблица 1.1

*Классификация Флинна*

Вычислительные системы

| | | Поток данных | |
|---|---|---|---|
| | | один | несколько |
| Поток команд | один | SISD<br>$a_1 + b_1$ | SIMD<br>$a_1 + b_1$<br>$a_2 + b_2$<br>$a_3 + b_3$ |
| | несколько | MISD<br>$a_1 + b_1$<br>$a_1 - b_1$<br>$a_1 * b_1$ | MIMD<br>$a_1 + b_1$<br>$a_2 - b_2$<br>$a_3 * b_3$ |

Широко используется уточнение классификации Флинна, согласно которому происходит разделение категории MIMD по способу организации памяти вычислительной системы. Среди MIMD-систем выделяют **мультипроцессоры** — машины с общей памятью (Uniform Memory Access, UMA), и **мультикомпьютеры** — машины, не обладающие удаленным доступом к памяти (NO Remote Memory Access, NORMA). Взаимодействие между процессорами в мультикомпьютерах осуществляется с помощью механизма передачи сообщений [3, 90].

Для целей параллельного программирования принципиальным моментом является принадлежность архитектуры используемой вычислительной системы к одному из трех вариантов: системы с общей памятью; системы с распределенной памятью; гибридные системы, совмещающие элементные базы двух вышеперечисленных.

Классификация многопроцессорных вычислительных систем класса MIMD по принципу организации памяти приведена на рис. 1.1.

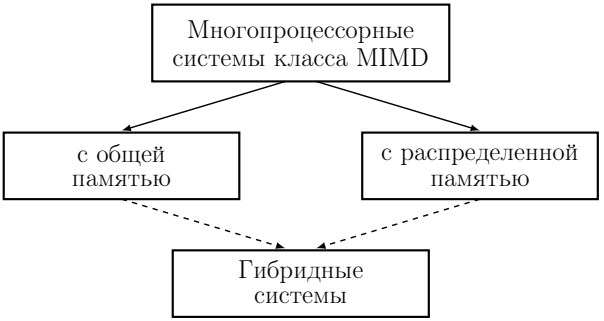


*Рис. 1.1.* Классификация многопроцессорных вычислительных систем класса MIMD по принципу организации памяти

*Примечание.* В последнее время все большее внимание привлекают вычислительные системы, использующие графические процессоры (GPU). Программирование таких систем имеет свои особенности, поскольку современные GPU, в отличие от центральных процессоров, представляют из себя массивно-параллельные вычислительные устройства с большим количеством вычислительных ядер и иерархически организованной собственной памятью [41]. В силу этого работа с графическими процессорами изучается обычно после получения базовых знаний в области программирования мультипроцессоров и/или мультикомпьютеров.

В системах с общей памятью несколько физических процессоров или процессорных ядер (как говорят, **вычислительных узлов**) имеют возможность совместно использовать единую область памяти. Данной ситуации отвечает **модель с общей памятью**. Вопросы одновременного доступа нескольких вычислительных узлов к участкам памяти занимают центральное место в модели и решаются с помощью механизмов **синхронизации**, к которым относят барьеры, замки́, семафоры и др. (см. далее и в [4, 14, 51]).

В системах с распределенной памятью каждый вычислительный узел имеет доступ только к принадлежащему ему участку памяти — локальной памяти. В этом случае для межпроцессорного обмена данными преду-

сматривается возможность отправки и приема сообщений по коммуникационной сети, объединяющей вычислительную систему. Соответствующая модель программирования носит название **модели передачи сообщений**.

Необходимо отметить, что программирование гибридных систем основано на использовании вышеперечисленных моделей или их комбинаций.

## 1.2. Средства параллельного программирования

Сформулированы несколько подходов к использованию ресурсов параллелизма в разрабатываемой программе [15]:

1) автоматическое распараллеливание последовательной версии программы средствами компилятора;

2) использование специализированных языков для параллельного программирования;

3) использование библиотек, предоставляющих возможности параллельного исполнения кода;

4) программирование с использованием особых расширений языка — средств распараллеливания.

Каждый из этих подходов обладает как преимуществами, так и недостатками.

Например, автоматическое распараллеливание программ (первый пункт) очень удобно для прикладных программистов, но часто его результаты, особенно для программ, основанных на сложных алгоритмах, оказываются неудовлетворительными.

Второй и третий подходы обладают положительной стороной, связанной с неизменностью кода по сравнению с последовательной версией программы.

Среди недостатков перечисленных трех подходов следует выделить отсутствие некоторых возможностей для оптимизации компилятором исполняемого кода, достаточно высокие накладные расходы при обращении к библиотечным процедурам и функциям, а также снижение переносимости исходного кода программы по сравнению с ее последовательной версией.

Наибольшие возможности для использования ресурсов параллелизма предоставляет четвертый подход — использование особых расширений языка (средств распараллеливания). Этот подход и будет подробно рассмотрен в данном учебном пособии.

## Контрольные вопросы к главе «Классификация архитектур вычислительных систем»

1. Опишите классификацию Флинна архитектур вычислительных систем.
2. На основании какого критерия производится деление MIMD-систем на мультипроцессоры и мультикомпьютеры?
3. Приведите классификацию многопроцессорных вычислительных систем по принципу организации памяти.
4. Расскажите о преимуществах и недостатках различных подходов к использованию ресурсов параллелизма.

# Глава 2

# Основы параллельных вычислений

## 2.1. Модели RAM и PRAM

Для анализа производительности программ широко используется модель вычислительной системы, называемая **машиной с произвольным доступом к памяти** (Random Access Machine, RAM) [58, 82].

Перечислим основные свойства RAM.

Система, работающая в рамках модели RAM, состоит из процессора, устройства доступа к памяти (системной шины) и памяти, состоящей из конечного числа ячеек (рис. 2.1).

Процессор последовательно выполняет команды, заложенные в программе П; ему доступны основные арифметические и логические операции и чтение/запись данных в памяти. При этом постулируется, что каждая команда выполняется за фиксированное время.

Произвольное действие процессора состоит из трех этапов:

1) чтение данных из памяти в один из своих регистров $r_i$, где $1 \leqslant i \leqslant N$;
2) выполнение арифметической или логической операции над содержимым своих регистров;
3) запись данных из регистра $r_j$, где $1 \leqslant j \leqslant N$, в некоторую ячейку памяти.

Считается, что исполнен ие трех перечисленных шагов требует времени $\Theta(1)$. (Функция $\Theta(f(n))$ используется для оценки скорости работы алгоритма в зависимости от размера $n$ его входных данных [1, 65]. Так, например, $\Theta(n^2)$ указывает на соответствующую квадратичную зависимость, $\Theta(\log_2 n)$ — на логарифмическую, а $\Theta(1)$ означает отсутствие зависимости от размера входных данных. Более подробные сведения

об оценках скорости работы алгоритмов см. в Приложении А «Методы оценки эффективности алгоритмов».)

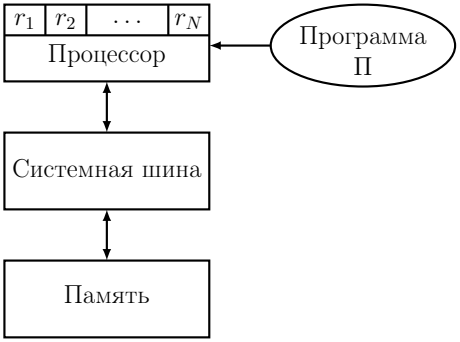


*Рис. 2.1.* Модель RAM

Одна из самых распространенных моделей параллельных компьютерных систем — параллельная машина с произвольным доступом к памяти (Parallel Random Access Machine, PRAM) [45, 82, 90]. В PRAM объединены $p$ процессоров, общая память и устройство управления, которое передает команды программы П процессорам (рис. 2.2).

Важной особенностью PRAM является ограниченное время доступа любого из процессоров системы к произвольной ячейке памяти. Как и в случае RAM, шаг алгоритма здесь также соответствует трем действиям процессора:

1) чтение данных процессором $P_i$ из $j$-й ячейки памяти;
2) выполнение арифметической или логической операции процессором $P_i$ над содержимым своих регистров;
3) запись данных в $k$-ю ячейку памяти.

Как было отмечено выше, любой шаг алгоритма выполняется за время $\Theta(1)$.

Одновременное обращение двух и более процессоров к одной и той же ячейке памяти приводит к **конфликтам доступа**. Они подразделяются на **конфликты чтения** и **конфликты записи**.

Если несколько процессоров пытаются прочесть данные из одной ячейки, то возможны два варианта дальнейших действий.

1. Исключающее чтение (Exclusive Read, ER). В данный момент времени чтение разрешено только одному процессору, в противном случае происходит ошибка выполнения программы.

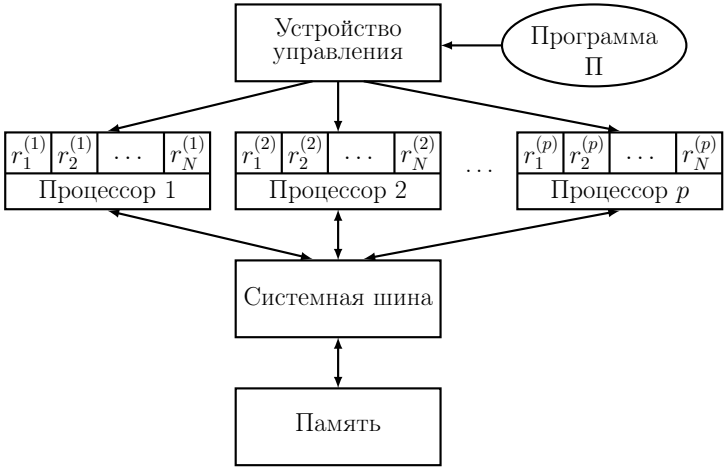


*Рис. 2.2.* Модель PRAM

2. Одновременное чтение (Concurrent Read, CR). Количество процессоров, получающих доступ к одной ячейке памяти, не ограничивается.

Если более одного процессора пытаются записать данные по одному адресу, разделяют два способа действия.

1. Исключающая запись (Exclusive Write, EW). Только одному процессору разрешено осуществлять запись в данную ячейку в конкретный момент времени.

2. Одновременная запись (Concurrent Write, CW). Несколько процессоров сразу получают доступ для записи к одной ячейке памяти.

Возникающие в последнем случае варианты того, по какому правилу будет определяться процессор (или процессоры), который осуществит запись, представлены ниже [72, 82, 90].

*Запись общего значения.* Предполагается, что все процессоры, готовые производить запись в одну и ту же ячейку памяти, обязаны записывать только общее для всех них значение, иначе инструкция записи в отведенную область памяти считается ошибочной.

*Произвольный выбор.* Процессор, осуществляющий запись, выбирается случайным образом.

*Запись с учетом приоритетов.* Каждому из конкурирующих процессоров присваивается некоторый приоритет, например, его порядковый номер, при этом сохраняется только то значение, которое поступило от процессора с заранее определенным приоритетом (например, наименьшим).

*Комбинированный выбор.* Все процессы выдают значения для записи, из них по определенному правилу формируется результат (например, сумма значений, максимальное значение и др.), который и записывается.

Классификация PRAM по методам разрешения конфликтов представлена на рис. 2.3.

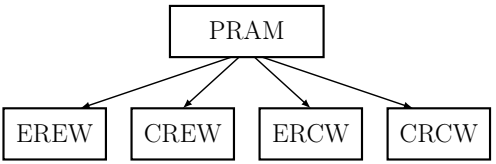


*Рис. 2.3.* Методы разрешения конфликтов в PRAM

Таким образом, системы EREW обладают существенными ограничениями, налагаемыми на работу с ячейками памяти. С другой стороны, системы CREW, ERCW, CRCW с большим количеством процессоров трудно построить по техническим причинам, поскольку количество вычислительных ядер, одновременно получающих доступ к некоторому участку памяти, ограничено. Однако существует важный и несколько неожиданный результат, позволяющий моделировать работу CRCW-машины на системе, построенной в соответствии с принципом EREW. Он представлен в теореме об эмуляции [94].

**Теорема об эмуляции.** *Пусть алгоритм для CRCW-машины решает некоторую задачу с параметром размера $N$ за время $T(N)$, используя $p$ процессоров. Тогда существует алгоритм для той же задачи на EREW-системе с $p$ процессорами, который может быть исполнен за время $O(T(N)\log_2 N)$. (Объем памяти PRAM должен быть увеличен в $O(p)$ раз.)*

В отличие от модели RAM, основной мерой сложности алгоритмов для многопроцессорных вычислительных систем является время исполнения алгоритма. Введем обозначения: $T_1(N)$ — время, затрачиваемое последовательным алгоритмом на решение задачи, сложность которой оценивается параметром $N$; $T_p(N)$ — время, затрачиваемое параллельным алгоритмом на машине с $p$ процессорами, причем $p > 1$. Посколь-

ку, как следует из определения RAM, каждая операция требует вполне определенного времени, величина $T_1(N)$ пропорциональна числу вычислительных операций в используемом алгоритме.

Заметим, что минимальное время исполнения алгоритма наблюдается в случае $p \to \infty$. Гипотетическую вычислительную систему с бесконечно большим количеством доступных процессоров называют **паракомпьютером**. Асимптотическую сложность алгоритма для паракомпьютера обозначают через $T_\infty(N)$.

*Примечание.* Краткие сведения о способах математической оценки свойств алгоритмов приведены в Приложении А «Методы оценки эффективности алгоритмов».

Для анализа параллельных алгоритмов широко применяются понятия **ускорения**, **эффективности** и **стоимости**. Прежде всего следует обратить внимание на то, насколько быстрее будет решена задача по сравнению с решением на однопроцессорной машине.

**Ускорение** $S_p(N)$, получаемое при использовании параллельного алгоритма на машине с $p$ процессорами, равно

$$S_p(N) = \frac{T_1(N)}{T_p(N)}.$$

Это мера прироста производительности по сравнению с *наилучшим* последовательным алгоритмом. Чем больше ускорение, тем больше отличается время решения задачи на многопроцессорной системе от продолжительности работы алгоритма на системе с одним процессором.

**Эффективность** $E_p(N)$ использования процессоров конкретным параллельным алгоритмом равна

$$E_p(N) = \frac{T_1(N)}{pT_p(N)} = \frac{S_p(N)}{p}.$$

**Стоимость вычислений** $C_p(N)$ определяется как произведение времени параллельного решения задачи и числа используемых процессоров: $C_p(N) = pT_p(N)$. **Стоимостно-оптимальный алгоритм** характеризуется стоимостью, пропорциональной сложности наилучшего последовательного алгоритма [14], и в этом случае

$$\frac{C_p(N)}{T_1(N)} = \Theta(1).$$

*Пример 2.1.* Время работы последовательной версии некоторого алгоритма $\mathcal{A}$ равно $T_1(N) = 2N\log_2(N)\,\tau$, где $N$ — размер входных данных, $\tau$ — время выполнения одной вычислительной операции. В предположении, что алгоритм допускает максимальное распараллеливание, т. е. время работы на вычислительной системе с $p$ процессорами равно $T_p(N) = \dfrac{T_1(N)}{p}$, вычислите время работы алгоритма $\mathcal{A}$ в случае $N = 64$, $p = 8$.

*Решение.*

Задача решается непосредственной подстановкой в соотношение $T_p(N) = \dfrac{T_1(N)}{p}$ данных из условия:

$$T_p(N) = \frac{2N\log_2(N)\,\tau}{p}.$$

Используя численные значения, получим

$$T_p(N) = \frac{2 \cdot 64\log_2(64)\,\tau}{8} = 96\tau.$$

□

Предельные значения величин ускорения и эффективности, как непосредственно следует из их определений, равны $S_p = p$, $E_p = 1$. Максимально возможное значение $S_p$ достигается, когда удается равномерно распределить вычисления по всем процессорам и не требуется дополнительных действий для обеспечения взаимодействия между процессорами на этапе работы программы и для объединения результатов. Попытка увеличить ускорение, увеличив число процессоров, приведет, как правило, к уменьшению величины $E_p$, и наоборот. Максимальная эффективность достигается при использовании единственного процессора ($p = 1$).

Здесь не обсуждается эффект сверхлинейного ускорения [99], который заключается в том, что $S_p > p$, и может возникнуть в моделях, отличных от PRAM, по нескольким причинам:

— последовательный алгоритм, используемый для сравнения, неоптимален;

— архитектура многопроцессорной системы имеет специфические особенности, ускоряющие некоторые вычислительные операции;

— алгоритм недетерминирован, что приводит к различным временам работы программы даже на одних и тех же входных данных;

— имеются существенные различия в объемах доступной оперативной памяти, когда последовательному алгоритму требуется обращаться к относительно «медленной» периферической памяти, а параллельный алгоритм использует только «быструю» память.

Многие из параллельных алгоритмов, описанных ниже, требуют наличия достаточно большого количества процессоров. К счастью, это не ограничивает их практическое применение, поскольку для любого алгоритма в модели PRAM существует возможность его модификации для системы с меньшим числом процессоров. Последнее утверждение носит название леммы Брента[1] [1, 57, 79].

**Лемма Брента.** *Пусть параллельный алгоритм $\mathcal{A}$ для решения некоторой задачи исполняется на RAM за время $T_1$, а на паракомпьютере — за время $T_\infty$. Тогда существует алгоритм $\mathcal{A}'$ для решения данной задачи, такой, что на PRAM с $p$ процессорами он исполняется за время $T(p)$, причем* $T(p) \leqslant T_\infty + \dfrac{T_1 - T_\infty}{p}$.

Доказательство леммы Брента приведено в решении упражнения **2.7**.

В любой параллельной программе есть последовательная часть, которая образована операциями ввода/вывода, синхронизации и т. д. Предположим, что по сравнению с последовательным способом решения:

1) при разделении задачи на независимые подзадачи временные затраты на межпроцессорное взаимодействие и объединение результатов пренебрежимо малы;

2) время работы параллельной части программы уменьшается пропорционально числу вычислительных узлов.

При сделанных предположениях известна оценка величины $S_p$.

**Закон Амдала**[2]. *Пусть $f$ — доля последовательных вычислений в алгоритме $\mathcal{A}$. Тогда ускорение $S_p$ при использовании $\mathcal{A}$ на системе из $p$ процессоров удовлетворяет неравенству*

$$S_p \leqslant \frac{1}{f + (1-f)/p}.$$

[1] Брент (Richard Peirce Brent) (род. 1946) — австралийский математик, специалист в области компьютерных наук.

[2] Амдал (Gene Myron Amdahl) (1922–2015) — американский исследователь, специалист в области вычислительной техники.

Для доказательства подсчитаем время исполнения алгоритма при работе на мультипроцессоре. Оно складывается из последовательных операций $fT_1$ и тех операций, которые могут быть распараллелены, и равно $T_p = fT_1 + \dfrac{1-f}{p}T_1$. Следовательно, верхний предел для ускорения может быть представлен в виде:

$$(S_p)_{\max} = \frac{T_1}{fT_1 + (1-f)T_1/p} = \frac{1}{f + (1-f)/p},$$

что и доказывает закон Амдала.

Неравенство $S_p \leqslant (S_p)_{\max}$ показывает, что существование последовательных вычислений, которые не могут быть распараллелены, накладывает ограничение на $S_p$. Даже при использовании паракомпьютера ускорение не превысит величины $S_\infty = \dfrac{1}{f}$.

На рис. 2.4 изображены зависимости ускорения $S_p$ от числа процессоров $p$ для типичных значений параметра $f$ в вычислительных задачах.

Эмпирически установлено [14], что для широкого класса вычислительных задач доля последовательных вычислений убывает с ростом размера входных данных задачи. Поэтому на практике ускорение может быть увеличено за счет увеличения вычислительной сложности решаемой задачи.

Несмотря на широкое распространение модели PRAM, не следует забывать и о ее недостатках. В частности, игнорируется различие в скоростях передачи данных для разных процессоров в конкретных архитектурах вычислительных систем.

Расмотрим пример, в котором иллюстрируется закон Амдала.

*Пример 2.2.* Предположим, исследователю требуется решить ресурсоемкую вычислительную задачу $\mathcal{Z}$. Последовательная версия программы, решающей задачу $\mathcal{Z}$, выполняется за время $T_1$. Ее параллельная версия содержит долю $f$ последовательных вычислений, $0 < f < 1$. Сторонняя организация предоставляет доступ к вычислительной системе, состоящей из $p$ процессоров ($1 < p < 512$), стоимость доступа к системе составляет $w_p(t) = \alpha \ln p + \beta t^\gamma$, где $\alpha, \beta, \gamma$ — const, $t$ — время работы над задачей исследователя. Сколько процессоров следует использовать, чтобы минимизировать функцию стоимости работы над задачей $\mathcal{Z}$? Расчет проведем для следующих значений параметров:

$f = 0{,}1$; $T_1 = 10{,}0$; $\alpha = 1{,}21$; $\beta = 5{,}37$; $\gamma = 1{,}5$.

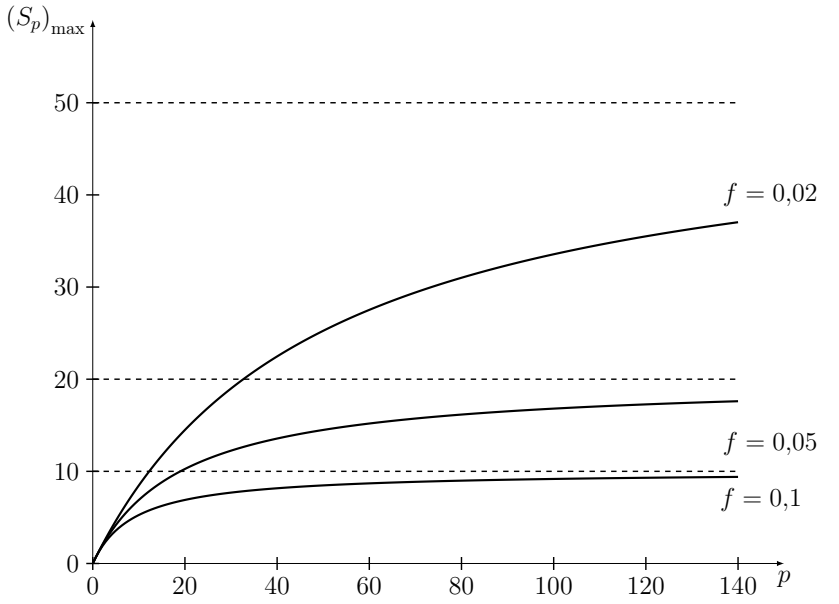


*Рис. 2.4.* Иллюстрация закона Амдала. Зависимости значений максимального ускорения $(S_p)_{\max}$ от числа процессоров $p$ при разных $f$ — долях последовательных вычислений

*Решение.*

Согласно закону Амдала, ускорение при использовании вычислительной системы, состоящей из $p$ процессоров, равно

$$S_p = \frac{T_1}{T_p} = \frac{1}{f + (1-f)/p},$$

следовательно, время работы над параллельной версией задачи $\mathcal{Z}$ будет составлять величину

$$T_p = T_1\Big(f + \frac{1-f}{p}\Big).$$

По условию функция стоимости $w_p(t)$ зависит от числа отдельных вычислительных узлов и от времени работы над задачей: $w_p(t) = \alpha \ln p + {} + \beta t^\gamma$. Подставив сюда полученное значение $T_p$, определим функцию стоимости в зависимости от параметра $p$:

$$w_p = \alpha \ln p + \beta T_1^\gamma \Big(f + \frac{1-f}{p}\Big)^\gamma.$$