Лекция 1
Введение в параллельные вычисления

Понятие о вычислительной системе

1) «***Вычислительная система*** включает в себя технические средства и программное обеспечение, ориентированные на решение определенной совокупности задач» (А.М. Ларионов, С.А. Майоров, Г.И. Новиков Вычислительные комплексы, системы и сети. – Л: Энергоатомиздат, 1987. - 178 с.)
2) Вычислительная система – совокупность технических и программных средств, выходящая за пределы понятия ЭВМ (Б. М. Каган Электронные вычислительные машины и системы. – М: Энергоатомиздат, 1991. – 592 с.)
3) Под **вычислительной системой (ВС)** часто понимают совокупность взаимосвязанных и взаимодействующих процессоров или ЭВМ, периферийного оборудования и программного обеспечения, предназначенную для сбора, хранения, обработки и распределения информации. Отличительной особенностью ВС по отношению к ЭВМ является наличие в них нескольких вычислителей, реализующих параллельную обработку.

Параллельная вычислительная система

Параллельная вычислительная система - вычислительная система, у которой имеется по меньшей мере более одного устройства управления или более одного центрального обрабатывающего устройства, которые работают одновременно (Б.А. Головкин Параллельные вычислительные системы – М.: Наука, 1980. – 520 с.)

Параллелизм

Параллелизм - способность к частичному или полному совмещению (одновременному выполнению) операций.

Это – совместное свойство и системы, и задачи (алгоритма).

Система должна обеспечивать возможность совмещения во времени операций на своих разных частях.

Алгоритм должен обладать внутренней способностью к распараллеливанию своих операций.

Типы («истоки») параллелизма

- Параллелизм независимых задач;
- Параллелизм данных;
- Функциональный параллелизм;
- Геометрический параллелизм;
- Алгоритмический параллелизм;
- Конвейерный параллелизм;
- «Беспорядочный» параллелизм.

Параллелизм независимых задач

Самый простой тип параллелизма. N независимых программ или задач могут одновременно выполняться на N отдельных ЭВМ или на N процессорах с раздельной памятью.

Предельное ускорение – в N раз.

Параллелизм независимых данных

Параллелизм независимых данных имеет место тогда, когда по одной и той же (или почти по одной и той же) программе должна обрабатываться некоторая совокупность данных, поступающих в систему одновременно.

Пример – обработка информации от датчиков, измеряющих одновременно некоторые параметры и установленных на нескольких объектах.

Параллелизм независимых данных

Другой пример - операции над векторами и матрицами. Решение задачи при этом в значительной степени сводится к выполнению одинаковых операций над парами чисел двух аналогичных объектов. Так, например, сложение двух матриц размерностью заключается в сложении соответствующих элементов этих матриц. При этом операция сложения должна быть проведена над парами чисел. Очевидно, все эти операции могут выполняться параллельно и независимо друг от друга несколькими обрабатывающими устройствами.

Параллелизм независимых данных

Еще примеры – умножение вектора (матрицы) на скаляр. Нахождение суммы элементов вектора (матрицы).

Конвейерный параллелизм

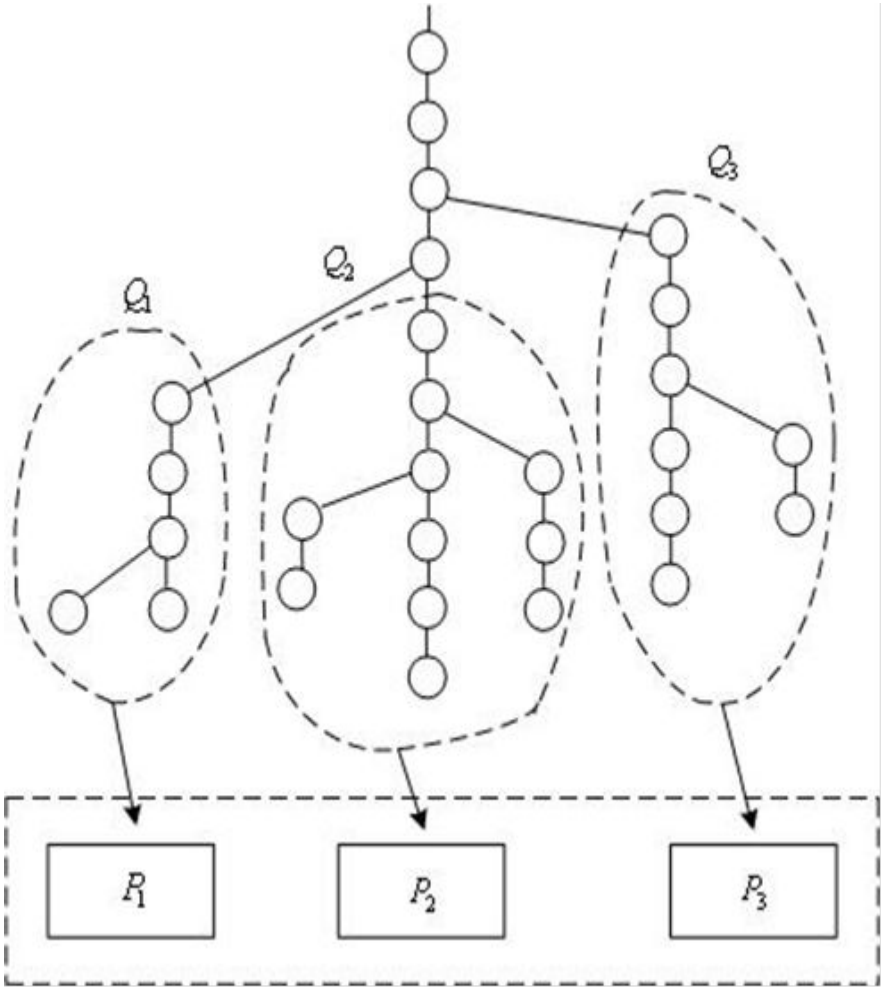
Конвейерный параллелизм связан с разбиением задачи на подзадачи таким образом, что результаты выполнения одной подзадачи являются входными данными для другой. Решение задачи может быть представлено в виде цепочки связанных подзадач.

При наличии конвейерной ВС и достаточно длинного потока входных данных отдельные подзадачи могут выполняться одновременно (для разных входных данных).

Внутренний параллелизм алгоритма

Алгоритмическим параллелизмом называется параллелизм, который выявляется путем выделения в данном алгоритме тех фрагментов, которые могут выполняться параллельно. Структура таких фрагментов может быть произвольной и зависит от свойств алгоритма (задачи).

Например, форма параллельных ветвей: Q_1, Q_2, Q_3 - ветви алгоритма. Кружки обозначают операторы алгоритма. P_1, P_2, P_3 – процессоры.



Уровень («зернистость», гранулированность) параллелизма

Определяется размером элементарного, неделимого объекта при исследовании параллелизма. Например – программа, часть программы, оператор, часть оператора.

Часто говорят о крупнозернистых и мелкозернистых параллельных алгоритмах.

Распараллеливание последовательных алгоритмов и программ

Распараллеливание последовательных программ

Первый подход - использовать имеющуюся последовательную программу для синтеза необходимого параллельного алгоритма. Последовательная программа разбивается на циклические и ациклические фрагменты, к которым применяются известные методики распараллеливания.

Распараллеливание ациклических участков программы (алгоритма)

Алгоритм распараллеливания ациклических участков программы обычно состоит из четырех этапов:

1)построение графа зависимостей между операторами программы;

2)построение той или иной формы (модели) параллельной программы, например, ярусно-параллельной формы (ЯПФ);

3)составление параллельной программы в той или иной форме;

4)выполнение полученной программы в используемой параллельной вычислительной системе.

Виды задач распараллеливания программ

- 1. Распараллеливание ациклических (последовательных) участков программ (алгоритмов).
- 2. Распараллеливание выражений.
- 3. Распараллеливание циклических участков программ (алгоритмов)

Граф зависимостей

Граф зависимостей между операторами программы строится на основе информационной (по данным) и логической (по управлению) зависимостей между операторами программы. Заметим, что о зависимости можно говорить на разных уровнях — от отдельных инструкций, до более крупных программных блоков. Представим программу в виде операторов двух видов – присваивания и ветвления.

Информационная зависимость между операторами программы

Оператор B зависит от оператора A информационно, если:

1)существует путь от входного оператора программы, проходящий через операторы A и B;

2)на этом пути оператор A является предшественником для оператора B;

3)выполняется одно из трех условий:

$$In(A) \cap Out(B) \neq \varnothing$$

$$In(B) \cap Out(A) \neq \varnothing$$

$$Out(A) \cap Out(B) \neq \varnothing$$

где *In(*)*— совокупность входных переменных оператора, *Out(*)* — совокупность выходных переменных оператора, Ø – пустое множество.

Логическая зависимость между операторами программы

Оператор B зависит от оператора A логически, если 1)существует путь от входного оператора программы до оператора A и 2)оператор A является "распознавателем", решающим, будет или нет выполняться оператор B.

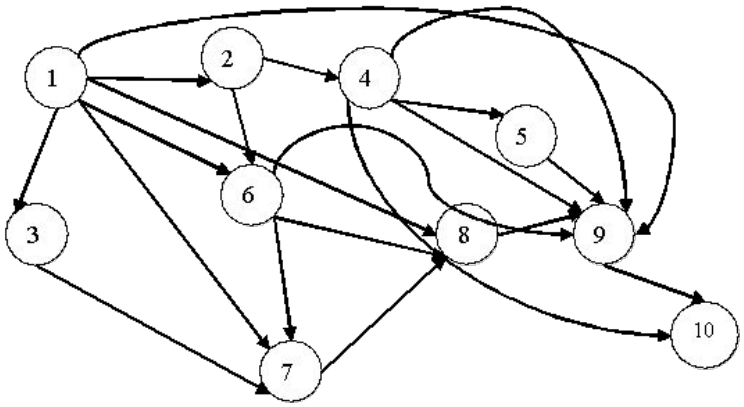
Логическая зависимость (зависимость по управлению) характерна для блоков ветвления (принятия решения) в алгоритме

Граф зависимости

Граф зависимости (ГЗ) имеет вершины, представляющие операторы программы (или блоки алгоритма).

Если оператор B зависит от оператора A логически или информационно, то в ГЗ вершина A соединяется с вершиной B дугой (направленной из A в B).

Например, участок программы (на абстрактном языке) и соответствующий ГЗ имеют вид:

1: ввод (x); 2: a:=sin(x); 3: b:=cos(x); 4: c:=2*a; 5: d:=c-3; 6: x:=a; 7: если x<b, то 8, иначе 9; 8: x:=3; 9: c:=x; 10: вывод c.	

Если пересекаются множества входных данных, то зависимости нет. Для операторов 10 и 7 участвующие переменные – входные. Для оператора 1 – выходные.

Распараллеливание ациклических программ

Пример 1. Рассмотрим следующую задачу. Пользователь вводит относительные адреса x, y начала и конца массива. Известно, что переменная, имеющее наибольшее значение соответствует концу массива, а переменная, имеющая наименьшее значение, – началу массива. Требуется распечатать массив, а также нижнюю и верхнюю его границы.

Рассмотрим также следующую программу, которая решает поставленную задачу (слева даны обозначения операторов программы; c – база массива):

```
A  ввод (x, y);
B  l := x;
C  h := y;
D  v := c+x;
E  z := c+y;
P  если x>y то F иначе G;
F  h := x; l := y;
G  печать от min (z, v) до max (z, v);
H  печать (l, h);
```

Граф непосредственных зависимостей для примера 1

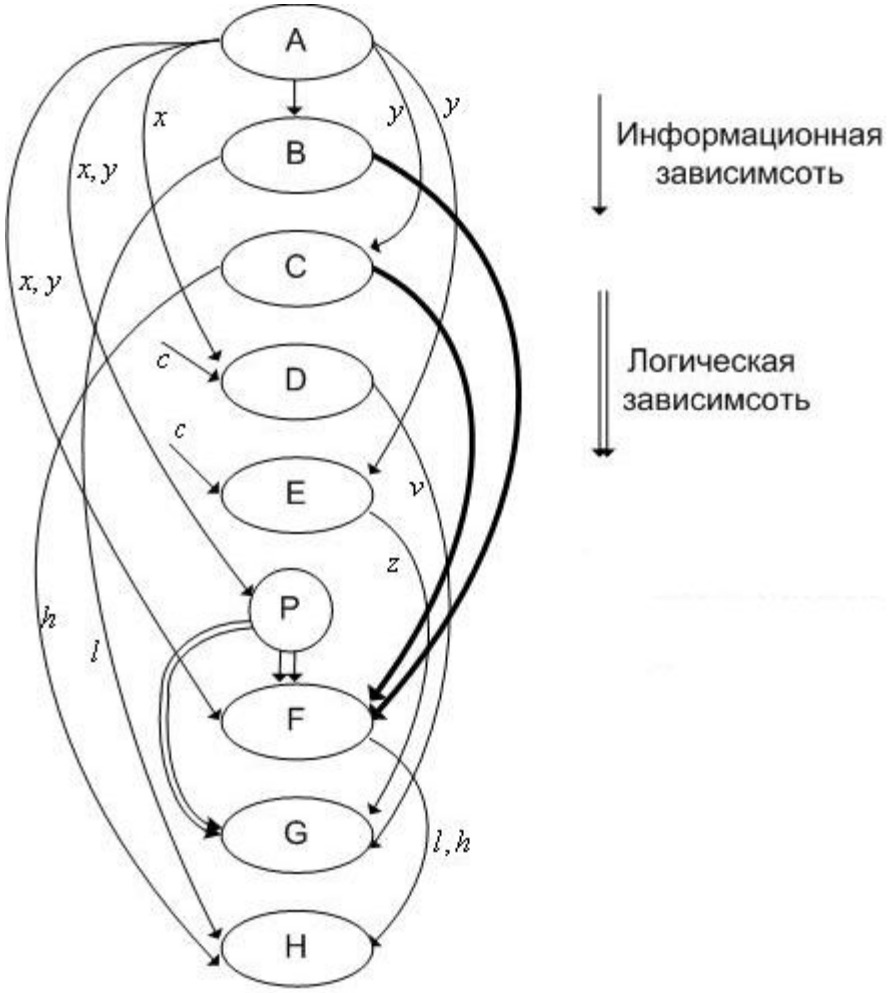


Рисунок 5.1 - Граф зависимостей программы для примера 1

Иногда на ГЗ показывают переменные связи.

Распараллеливание ациклических программ (продолжение)

Большой размер графа усложняет последующие этапы распараллеливания программы. Для уменьшения размера графа целесообразно произвести сжатие линейных участков программы в обобщенные операторы, т. е. выделить в исходном графе линейные подграфы и поставить в соответствие каждому из них одну обобщенную вершину графа.

Построение ярусно-параллельной формы программы

Алгоритм построения ярусно-параллельной формы программы:

1. На первый ярус ЯПФ заносятся все операторы, в которые не идут стрелки графа (непосредственных) зависимостей;
2. Если построено k ярусов, то в (k+1)-й ярус заносятся все те операторы, которые имеют входящие стрелки только от первых k ярусов.

ЯПФ для примера 1

Построим ЯПФ для программы, граф зависимостей которой приведен на рисунке 1.

Входящие стрелки отсутствуют только у оператора A. Поэтому на первый ярус ЯПФ помещаем только этот оператор.

Входящие стрелки только от операторов 1-го яруса имеют операторы B, C, D, E, F. Поэтому помещаем их на второй ярус.

Входящие стрелки только от операторов 1-го и 2-го ярусов имеют операторы F, G. Помещаем их на третий ярус.

Наконец, входящие стрелки только от операторов 1-го, 2-го и 3-го ярусов имеет только оператор H . Помещаем этот оператор на четвертый ярус (см. рисунок 2)

ЯПФ для примера 1

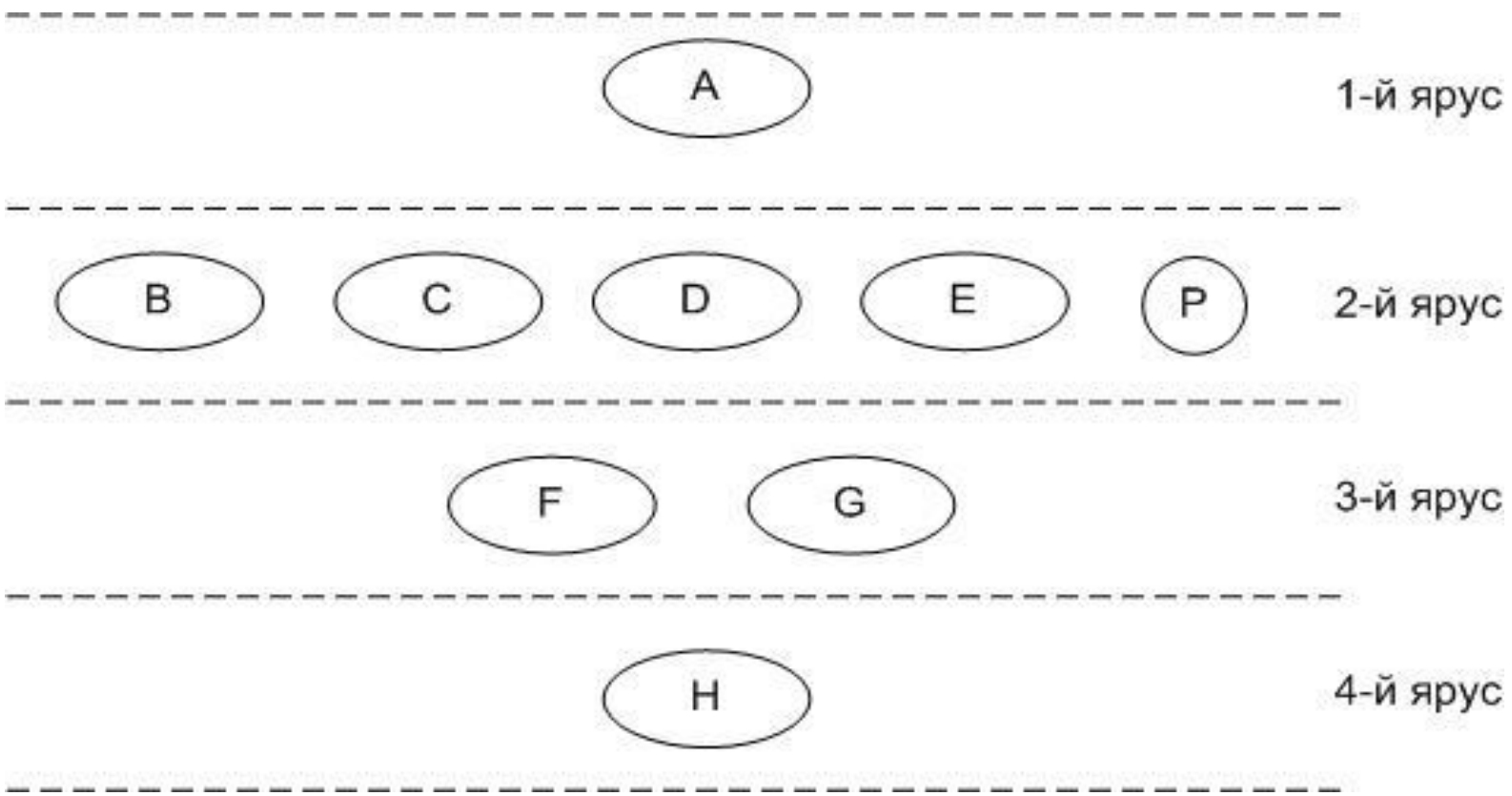


Рисунок 2 - Ярусно-параллельная форма для программы, граф зависимостей которой приведен на рисунке 1.

Параметры и характеристики ЯПФ

Из алгоритма построения ЯПФ следует, что все операторы, находящиеся на одном уровне этой формы могут выполняться параллельно. Ярус ЯПФ иногда называется уровнем готовности операторов.

Ярусы ЯПФ устанавливают между операторами отношение предшествования — к моменту начала вычислений на (k+1)-м ярусе должны быть закончены вычисления на k-м ярусе.

С ЯПФ связан ряд важных понятий.

Высотой ЯПФ называется количество ее ярусов.

Шириной яруса ЯПФ называется количество операторов на этом ярусе. Шириной ЯПФ называется максимальная из ширин ярусов данной ЯПФ. ЯПФ данного алгоритма, имеющая минимальную высоту, называется минимальной ЯПФ или максимально-параллельной ЯПФ.

Ширина минимальной ЯПФ называется шириной алгоритма, а ее высота – высотой алгоритма.

Ширина алгоритма и высота алгоритма представляют собой примеры мер параллелизма алгоритма.

Особенности распараллеливания выражений

Проблема распараллеливания выражений является наиболее изученной в параллельном программировании. Формально выражение – это ациклический процесс и к нему можно подходить так же, как рассмотрено выше. Однако здесь есть и принципиальное отличие – в выражениях не всегда указан порядок действий. Например, следующие выражения эквиваленты (с математической точки зрения): $a * b * c = (a * b) * c = a * (b * c)$.

Задача распараллеливания выражений ставится, например, следующим образом: *построить алгоритм, который каждому выражению ставит в соответствие эквивалентную ему с минимальной высотой ЯПФ.*

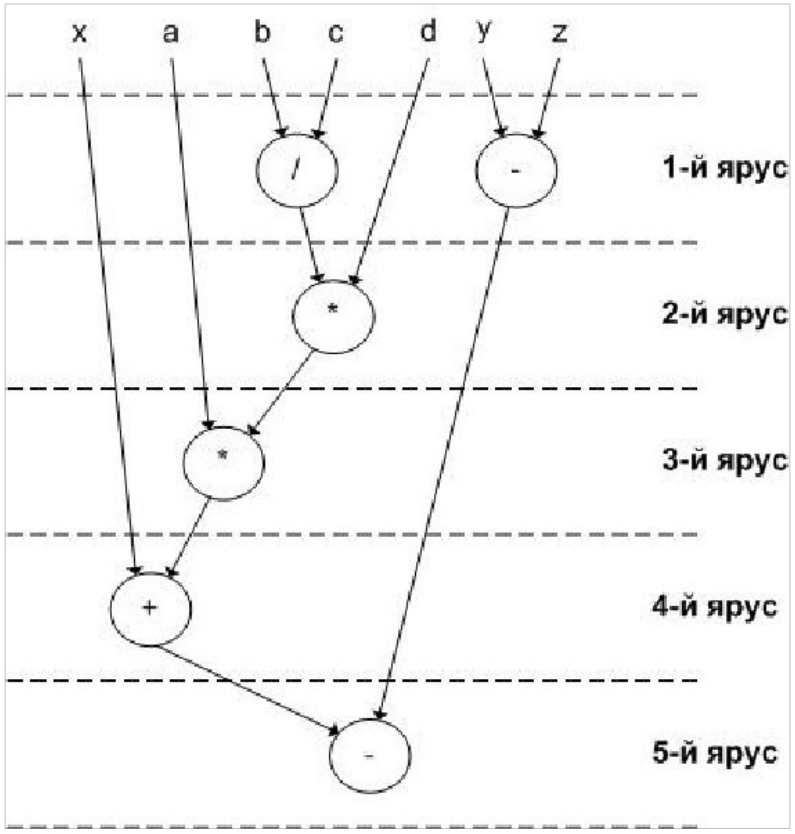
Здесь эквивалентность понимается в смысле применения обычных законов ассоциативности $(a * b) * c = a * (b * c)$, коммутативности $a * b = b * a$ и дистрибутивности $a * (b + c) = a * b + a * c$.

Пример 2

Рассмотрим выражение:

$$t = (x + (a * ((b / c) * d))) - (y - z) \qquad (11.1)$$

ЯПФ выражения (11.1):

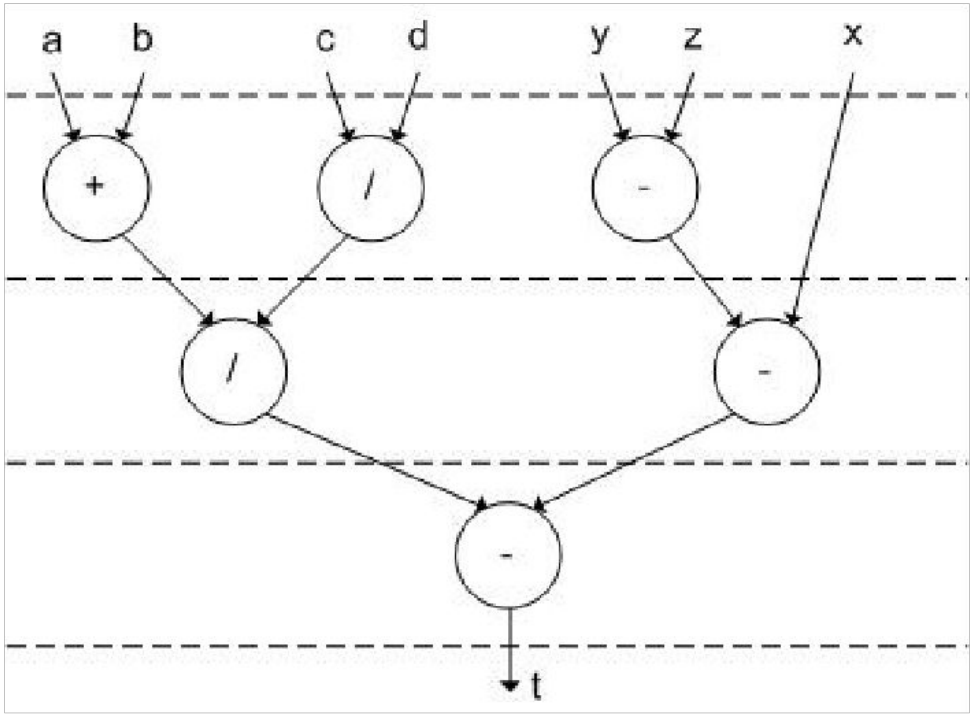


Пример 2 (продолжение)

Перепишем выражение (11.1) в эквивалентном виде

$$t = ((a * b) / (c / d)) - ((y - z) - x) \qquad (12.1)$$

ЯПФ, соответствующая этому выражению:



Распараллеливание циклических фрагментов программ

Основная работа, которую обычно производит ЭВМ, это обработка циклов и других повторяющихся участков программ. Потому теория распараллеливания циклических участков программ представляет собой одно из важнейших направлений параллельного программирования.

О распараллеливании циклов принято говорить в терминах *пространства итераций* (Лампорт Л., 70-е годы). Рассмотрим для примера следующий циклический фрагмент программы (T – тело цикла):

$$\begin{array}{l} \text{FOR } i_1 = 1, r_1 \text{ DO} \\ \text{FOR } i_2 = 1, r_2 \text{ DO} \\ \cdots\cdots \\ \text{FOR } i_n = 1, r_n \text{ DO } T(i_1, i_2, \ldots, i_n) \end{array} \qquad (1)$$

Пространством итераций гнезда циклов (1) называется множество целочисленных векторов

$$I = \{(i_1, i_2, \ldots, i_n) \mid 1 \le i_k \le r_k, k \in [1, n]\}.$$

В терминах пространства итераций задача распараллеливания гнезда циклов (1) ставится как *задача разбиения множества* I *на подмножества* $I_1, I_2, \ldots, I_s$ *такие, что вычисления тела цикла* $T(i_1, i_2, \ldots, i_n)$ *в каждом из них могут быть выполнены одновременно (с сохранением информационных связей исходного цикла, естественно).*

Распараллеливание циклов

То есть, внутри каждого подмножества $I_1, I_2, \ldots, I_s$ ерации независимы друг от друга.

Зависимость между итерациями цикла можно определить, например, с помощью развертки цикла в ациклический фрагмент:

T(1,1, … 1); T(1,1, … 2); … T(r_1, r_2,… r_k) и построения графа зависимостей между отдельными итерациями.

Методы и ограничения

В зависимости от типа областей $I_1, I_2, \ldots, I_s$ выделяют несколько методов распараллеливания циклов:

метод параллелепипедов;
метод гиперплоскостей;
метод пирамид.

Методы распараллеливания применимы только при выполнении ряда ограничений на операторы тела цикла. Перечислим основные из этих ограничений:

- тело цикла не содержит условных и безусловных переходов вне тела;
- внутри тела цикла передача управления осуществляется только вперед;
- тело цикла не содержит операторов ввода-вывода и обращений к подпрограммам;
- все индексные выражения линейны, т.е. отсутствуют индексы вида $X(I*J*K)$;
- отсутствует косвенная адресация вида $X(N(I))$;
- индексы не изменяются в теле цикла;

Метод параллелепипедов

Проиллюстрируем идею *метода параллелепипедов распараллеливания циклов* на примере.

Пример 17.1

Рассмотрим следующее гнездо циклов

```
FOR i = 2,5
   FOR j = 2,4
      X(i, j) = X(i - 1, j - 1)**2;
```

(17.1)

Соответствующие пространство итераций и информационные связи в теле гнезда циклов приведены на рисунке 17.1. Пространство итераций гнезда циклов есть совокупность точек, обозначенных светлыми кружками.

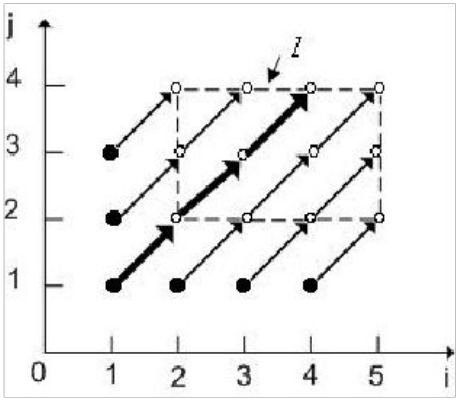


Рисунок 17.1 - Структура информационных связей в теле цикла для примера 17.1

Продолжение примера 17.1

Из рисунка 17.1 следует, что в цикле (17.1) информационно связаны, например, следующие итерации (на рисунке выделены жирным): *X(2,2)=X(1,1)**2; X(3,3)=X(2,2)**2; X(4,4)=X(3,3)**2.*

Из структуры информационных связей следует, что пространство итераций можно разбить на параллелепипеды так, как показано на рисунке 18.1.

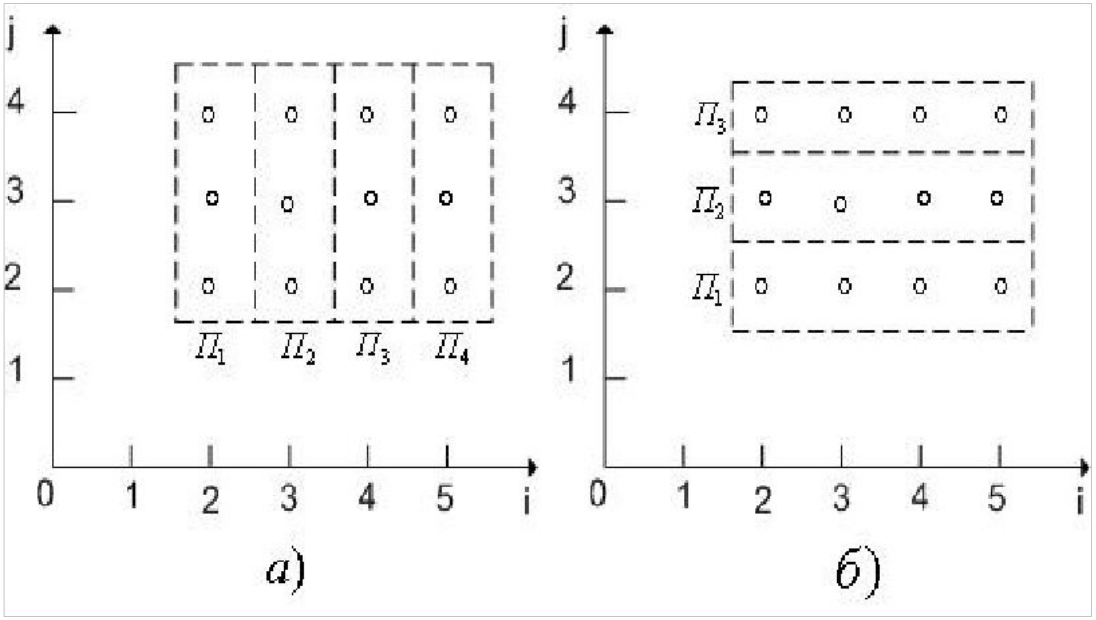
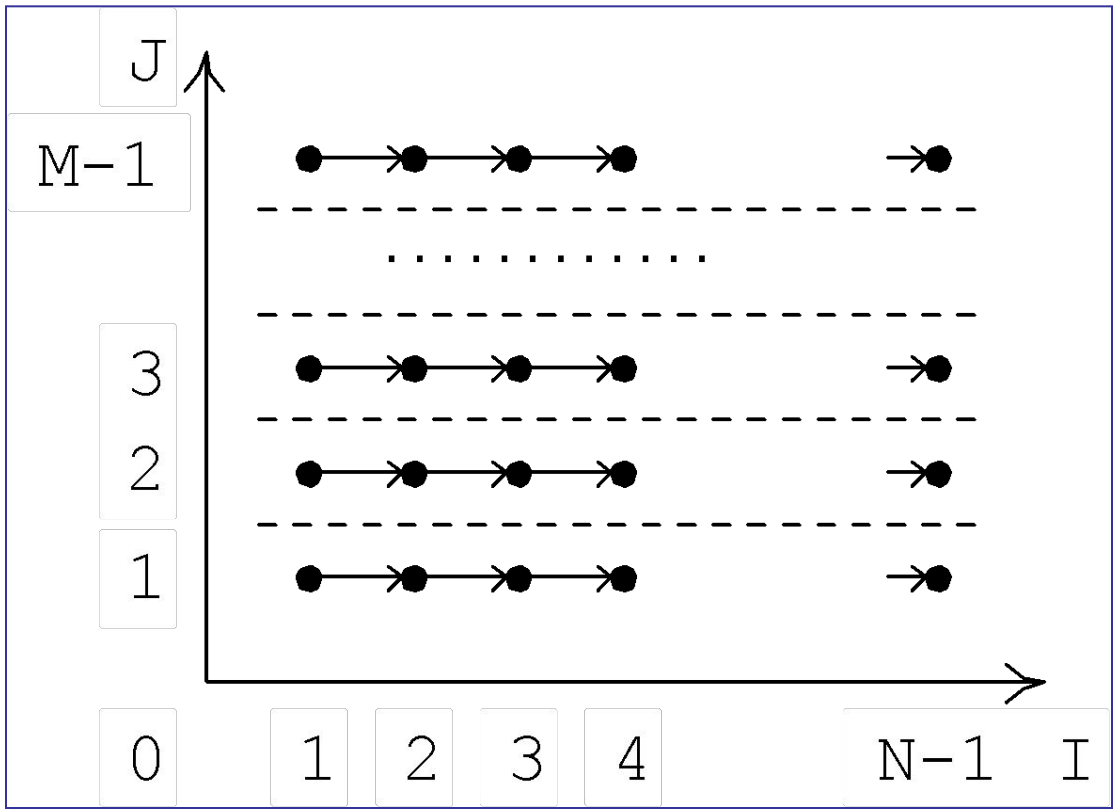


Рисунок 18.1 - Разбивка пространства итераций на параллелепипеды.

Пример

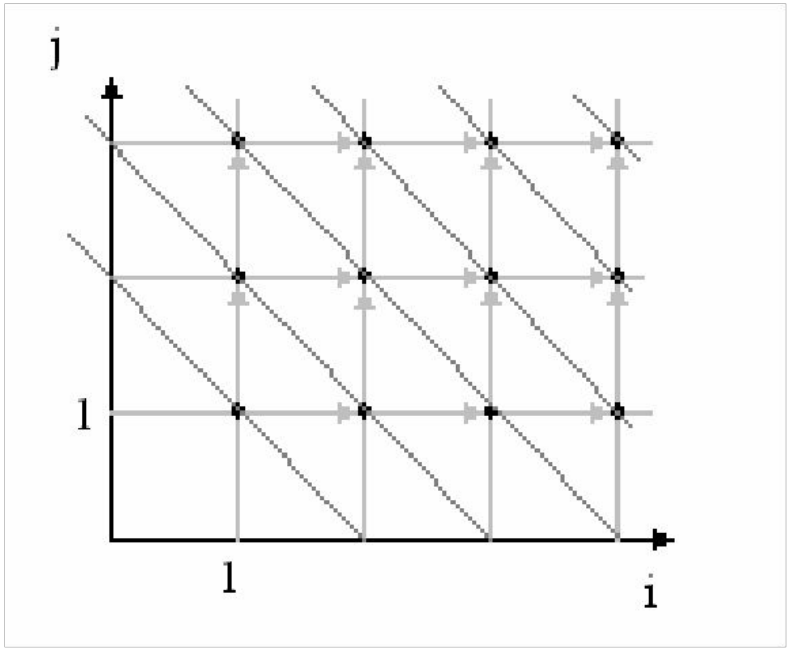
```
   do 1 i = 1, N-1
      do 1 j = 1, M-1
1     a(i,j) = a(i-1,j) + a(i,j)
```



Метод гиперплоскостей

Рассмотрим цикл:

```
for j:=1 to 3 do
for i:=1 to 4 do begin
x(i,j)=f(x(i-1,j)); y(i,j)=f1(y(i,j-1));
end;
```



Гиперплоскости для двумерного пространства итераций.

Оценка производительности параллельных ВС

Характеристики скорости выполнения операций вычислительных систем

Под производительностью (performance) понимают количество операций, выполняемых на данной вычислительной системе в единицу времени. Быстродействие (speed) – величина, обратная среднему времени выполнения одной операции.

Производительность измеряется в числе команд в секунду, например, в миллионах – MIPS (millions instructions per second) или в числе операций с плавающей запятой в секунду, например, в миллионах – MFLOPS (millions floating point operations per second).

Единица	Год	Значение
килофлопс	1949	10^3
мегафлопс	1964	10^6
гигафлопс	1987	10^9
терафлопс	1997	10^{12}
петафлопс	2008	10^{15}

Реальная производительность (производительность на тестах)

Существующие тестовые наборы можно разделить на три группы:

- тесты производителей (компаний-изготовителей компьютеров), предназначенные, как правило, для сравнения однотипных компьютеров, относящихся к одному семейству;
- стандартные тесты, разработанные независимыми аналитиками и предназначенные для сравнения широкого спектра компьютеров;
- пользовательские тесты, учитывающие специфику решаемых пользовательских задач.

В вычислительной практике чаще всего применяют стандартные тесты.

Реальная производительность (производительность на тестах)

Поскольку большую часть времени выполнения программ обычно занимают циклы, часто именно они применяются в качестве тестов.

В настоящее время наиболее известным тестом производительности является набор тестов Linpack, который представляет собой набор программ для решения СЛАУ методом исключения Гаусса. Основным параметром тестов Linpack является порядок СЛАУ N. Обычно используются тесты с N=100 и тесты N=1000.

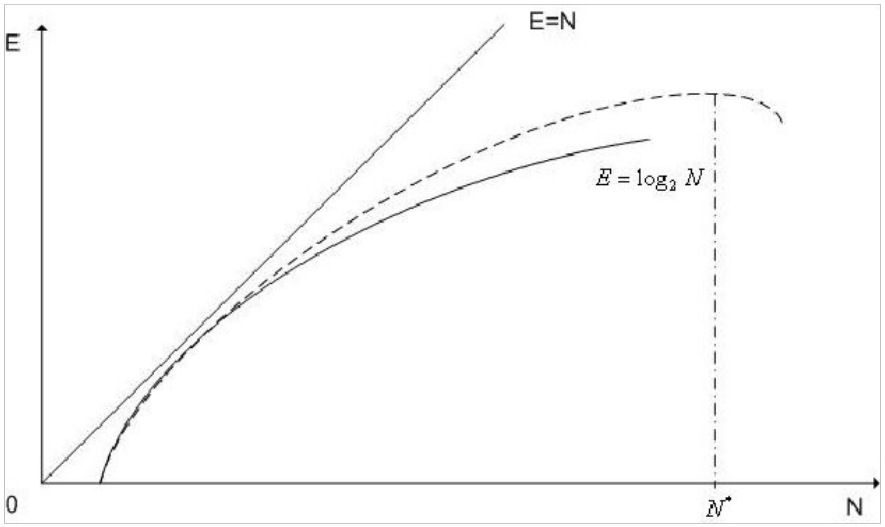
Известно количество операций (как функция размерности СЛАУ N), которые необходимо выполнить для решения систем линейных алгебраических уравнений (СЛАУ) методом исключения Гаусса.

Поэтому, зная время решения задачи, легко найти производительность системы в FLOPS.

Известный список TOP 500, включающий в себя 500 самых высокопроизводительных компьютеров мира, строится на основе тестирования с помощью тестов Linpack.

Гипотеза Минского

Гипотеза Минского. В N-процессорной системе, в которой производительность каждого процессора равна единице, общая производительность E растет как $log_2 N$



В первых параллельных вычислительных системах, когда количество процессоров было невелико, гипотеза Минского подтверждалась. В современных системах с большим количеством процессоров имеет место зависимость производительности от числа процессоров, показанная на рисунке 1 пунктиром.

Оценка эффективности параллельных алгоритмов

Оценка эффективности

Для оценки эффективности параллельных алгоритмов в основном используют следующие величины:

средняя степень параллелизма;
ускорение параллельного алгоритма;
эффективность параллельного алгоритма.

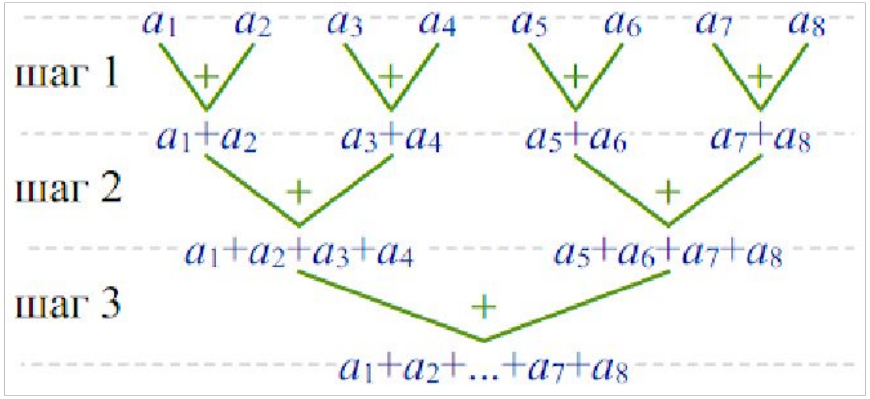
Средняя степень параллелизма

Средней степенью параллелизма называется отношение общего числа операций алгоритма к числу его этапов, например, к высоте ЯПФ алгоритма.

Например, для алгоритма сдваивания сложения n чисел средняя степень параллелизма равна

$$O\left(\frac{n}{\log_2 n}\right)$$

(поскольку общее число операций алгоритма равно $(n-1)$, а высота ЯПФ - $\lceil \log_2 n \rceil$.



Средняя степень параллелизма

Идеальный параллельный алгоритм имеет степень параллелизма  (например, алгоритм сложения двух векторов).

Средняя степень параллелизма характеризует только сам параллельный алгоритм. Из этой характеристики никак не следует эффективность этого алгоритма для конкретной параллельной вычислительной системы

Ускорение параллельного алгоритма

Ускорение параллельного алгоритма является его наиболее информативной характеристикой, которая показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение параллельного алгоритма определяется величиной

$$S_N = \frac{T_1}{T_N},$$

где T_1 - время выполнения алгоритма на одном процессоре, T_N - время выполнения того же алгоритма на N процессорах.

Ускорение параллельного алгоритма

Идеальным, очевидно, является ускорение $S_N=N$. В реальности это ускорение недостижимо. Перечислим причины невозможности достижения идеального ускорения:

- отсутствие максимального параллелизма в алгоритме;
- несбалансированность загрузки процессоров (если, например, необходимо сложить два вектора из 9 чисел каждый на восьмипроцессорной системе);
- временные затраты на обмен данными, конфликты в памяти и на синхронизацию.

Эффективность параллельного алгоритма

Эффективность параллельного алгоритма определяется величиной

$$E_N = \frac{S_N}{N}, \qquad (4)$$

где S_N- ускорение параллельного алгоритма, N- количество процессоров в системе.

Из того факта, что $S_N \le N$, следует ограничение сверху на величину эффективности: $E_N \le 1$.

Потери эффективности

Можно выделить следующие основные причины потери эффективности параллельных вычислений:

- время инициализации параллельной программы (startup);
- несбалансированность загрузки процессоров (load imbalance);
- затраты на коммуникации (communication costs);
- наличие в программе последовательных частей (serial part of the code).

Потери эффективности

Время инициализации параллельной программы – это время генерации начальных данных (часто на одном процессоре) и рассылки результатов по всем процессорам.

Несбалансированность загрузки процессоров приводит к тому, что часть процессоров вынуждена какое-то время простаивать.

Затраты на коммуникации определяются пропускной способностью коммуникационной сети, латентностью коммуникационной сети, диаметром коммуникационной сети (который в значительной мере определяется топологий коммуникационной сети. Коммуникационные затраты могут быть сокращены за счет использования встречных обменов данными и асинхронной передачи данных.

Закон Амдала (Amdahl)

Рассмотрим ускорение параллельного алгоритма: $S_N = \frac{T_1}{T_N}$,

Несколько детализируем величину T_N - положим

$$T_N = (\alpha_1 + \frac{\alpha_2}{v} + \frac{\alpha_3}{N}) T_1 \qquad (6)$$

Здесь

α_1 - доля операций, выполняемых на одном процессоре - минимальный параллелизм;

α_2 - доля операций, выполняемых на v процессорах ($v<N$) – частичный параллелизм;

α_3 - доля операций, выполняемых на N процессорах (максимальный параллелизм);

Закон Амдала

Выделим три следующих частных случая:

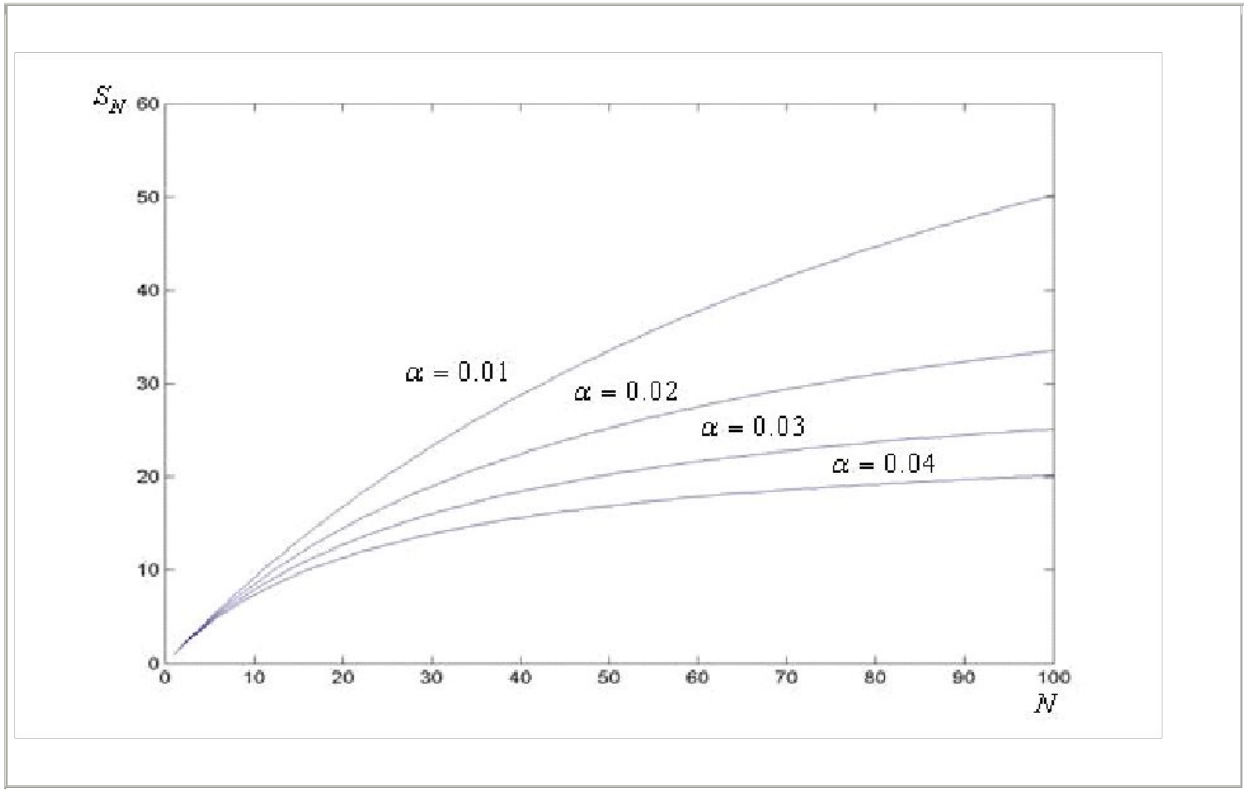
$\alpha_1 = \alpha_2 = 0, \alpha_3 = 1$ Легко видеть, что в этом случае ускорение максимально возможно и равно $S_N = N$;

$\alpha_1 = \alpha_3 = 0, \alpha_2 = 1$ Здесь ускорение равно средней степени параллелизма v, т.е. $S_N = v < N$;

$\alpha_1 = \alpha, \alpha_2 = 0, \alpha_3 = 1 - \alpha$ Ускорение в этом случае называется *законом Амдала*:

$$S_N = \frac{1}{\alpha + (1-\alpha)/N}. \qquad (7)$$

Закон Амдала



Если всего 1% (α=0.01) операций выполняется последовательно, то ускорение уменьшается вдвое по сравнению с максимально возможным.

Закон Амдала

Из закона Амдала следует, что если, к примеру, $\alpha=0.1$, т.е. всего 10% операций программы выполняется последовательно, то при любом, как угодно большом количестве используемых процессоров N, ускорение, превышающее 10, принципиально получить невозможно (см. табл.1).

Таблица 1. Предельное ускорение, как функция доли последовательных операций α в процентах и количества процессоров N.

Количество процессоров N	α=50%	α=25%	α=10%	α=5%	α=2%
2	1.33	1.60	1.82	1.90	1.96
8	1.78	2.91	4.71	5.93	7.02
32	1.94	3.66	7.80	12.55	19.75
51	1.99	3.97	9.83	19.28	45.63
2048	2.00	3.99	9.96	19.82	48.83

Области применения параллельных вычислительных систем

- предсказания погоды, климата и глобальных изменений в атмосфере;
- науки о материалах;
- построение полупроводниковых приборов;
- сверхпроводимость;
- разработка фармацевтических препаратов;
- генетика;
- астрономия;
- транспортные задачи;
- гидро- и газодинамика;
- управляемый термоядерный синтез;
- эффективность систем сгорания топлива;
- геоинформационные системы;

Области применения параллельных вычислительных систем

- разведка недр;
- наука о мировом океане;
- распознавание и синтез речи;
- распознавание изображений;
- военные цели.

- Ряд областей применения находится на стыках соответствующих наук.