

Тема практического занятия от 10_12_21:

Сложные задачи, их характеристики и пути эффективного решения

Основные вопросы:

1. Основные этапы решения сложной задачи на параллельных ВС.....1
2. Сложная задача. Характеристики сложности (размерность, трудоемкость, объем требуемой памяти и др.) Основные классы сложных задач..... 4
3. Характеристики параллельности (ускорение, эффективность, стоимость и ценность параллельного решения). 10
4. Закон Аамдаля и его следствия..... 122

1. Основные этапы решения сложной задачи на параллельных ВС

Весь процесс решения некоторой задачи на параллельной ВС можно разбить на следующие этапы:

- формулировка задачи;
- составление модели исследуемого в задаче объекта;
- определение метода решения задачи для получения необходимой результирующей информации на основании используемой модели;
- разработка алгоритма решения задачи на основе используемого метода и модели исследуемого в задаче объекта;
- выбор технологии программирования;
- разработка программы для соответствующей параллельной ВС на основе имеющегося алгоритма и получение результирующей информации после выполнения программы.

Данный процесс представлен на рис. 1. Эти этапы важны и для обычных ЭВМ, но при использовании параллельных ВС они приобретают особую значимость. Любая параллельная ВС – это тщательно сбалансированная система, которая может дать фантастический результат или показать свою неэффективность при решении той или иной задачи. Такие системы специально проектируются для того, чтобы работать с огромной производительностью. Но параллельные ВС не могут работать одинаково производительны на любых программах. Если структура программы, реализующей некоторую задачу, не соответствует особенностям архитектуры параллельной ВС (особенностям аппаратно-программной среды ВС), то эффективность решения задачи на используемой ВС неизбежно падает. Указанное несоответствие может возникнуть на любом этапе решения задачи. Чем аккуратнее выполняется каждый этап, чем больше структура программы соответствует особенностям архитектуры параллельной ВС, тем выше эффективность решения соответствующей

задачи.

Если на каком-либо одном шаге не были учтены особенности целевой ВС, то, скорее всего, большой производительности достичь не удастся. В самом деле, ориентация на параллельную векторную систему или вычислительный кластер с РП во многом определяет особенности метода решения задачи. В одном случае в программе необходимо векторизовать внутренние циклы, а в другом надо думать о распараллеливании значительных фрагментов кода. И то, и другое свойство программ определяется свойствами заложенных в них методов. Не обладает выбранный метод такими свойствами, не будет их и в программе, а, значит, и не будет высокой эффективности выполнения задачи. Всё то же самое касается и алгоритмов. При этом нужно всё же иметь в виду, что иногда структура программы и соответственно алгоритма не позволяет в принципе получить эффективную реализацию задачи, но такая ситуация встречается не так часто.

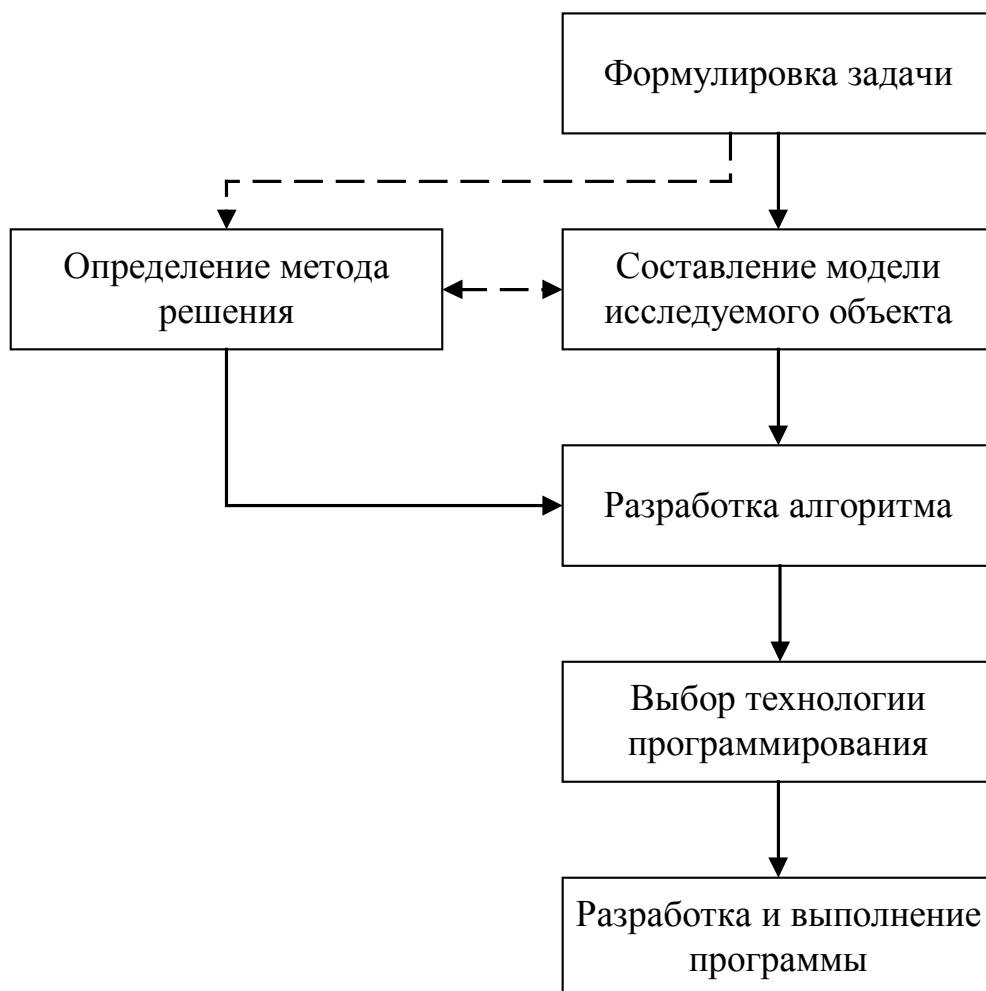


Рис. 1. Процесс решения задачи на параллельной ВС

Что подлежит распараллеливанию в процессе решения задачи на современных архитектурах?

- 1.Задача (декомпозиция на подзадачи меньшей размерности)
- 2.Метод решения
- 3.Алгоритм
- 4.Программа
- 5.Циклы
- 6.Выражения

2. Сложная задача. Характеристики сложности (размерность, трудоемкость, объем требуемой памяти и др.). Основные классы сложных задач

Неформальное определение: сложная задача – это задача, которая не может быть эффективно решена на существующих массовых вычислительных средствах с помощью традиционных методов и алгоритмов решения за допустимое время.

Примеры сложных задач:

- Электро- и гидродинамика,
- сейсморазведка,
- расчет прогноза погоды,
- моделирование химических соединений,
- исследование виртуальной реальности и т.д.

Характеристики сложности:

1. **T** – трудоемкость решения задачи (часто определяется числом мультипликативных операций).
2. **O** – объем обрабатываемой информации.
3. **t_{доп.}** – допустимое время решения задачи.
4. **P** – требуемая производительность, $P = T/t_{\text{доп.}}$

Декомпозиция процессов решения сложных задач – основа построения эффективных параллельных алгоритмов и разработки параллельных программ

Основным подходом к решению любой сложной задачи является декомпозиция (разбиение задачи на совокупность подзадач меньшей размерности).

$$Z(n) \xrightarrow{\text{декомпозиция}} \{Z_1, Z_2, \dots, Z_i, \dots, Z_p, Z_{\text{св}}\},$$
где Z_i – i-ая подзадача.
 $Z_{\text{св}}$ – задача связи.

Обозначим:

T₁(n) – трудоемкость решения некоторой задачи **Z(n)** на одном вычислителе с помощью **наилучшего последовательного алгоритма**, где **n** – это размерность задачи.

T_p(n) – трудоемкость решения задачи **Z(n)** на **p** вычислителях с использованием некоторого **параллельного алгоритма**.

T(Z_i) – трудоемкость решения подзадачи **Z_i**

T(Z_{св}) – трудоемкость решения задачи связи **Z_{св}**

В зависимости от соотношения трудоемкостей задачи связи **Z_{св}** и подзадач **Z_i** зависит к какому классу относится исходная сложная задача **Z**.

Принято выделять четыре основных класса сложных задач:

1. **Несвязная сложная** задача.
2. **Слабосвязная сложная** задача.
3. **Среднесвязная сложная** задача
4. **Сильносвязная сложная** задача.

Для классификации сложных задач также можно воспользоваться показателем W_{ij} – трудоемкость обменных взаимодействий между $Z_{i-ой}$ и $Z_{j-ой}$ подзадачами в процессе решения задачи $Z(n)$.

Если:

1. $\sum_j \sum_i W_{ij} \rightarrow 0$, или $T(Z_{св}) \rightarrow 0$, то $Z(n)$ – **несвязная сложная задача**
2. $\sum_j \sum_i W_{ij} \ll \max_i T(Z_i)$, или $T(Z_{св}) \ll \max_i T(Z_i)$, то $Z(n)$ – **слабосвязная задача**
3. $\sum_j \sum_i W_{ij} \cong \max_i T(Z_i)$, или $T(Z_{св}) \approx \max_i T(Z_i)$, то $Z(n)$ – **среднесвязная задача**
4. $\sum_j \sum_i W_{ij} \gg \max_i T(Z_i)$, или $T(Z_{св}) \gg \max_i T(Z_i)$, то $Z(n)$ – **сильносвязная задача**.

Рассмотрим примеры классов сложных задач:

1. Сложная задача – решение СЛАУ (система линейных алгебраических уравнений) **большой размерности**

Пусть имеем: СЛАУ, которая представлена в матричной форме:

$$A \times X = B \quad X - ?$$

Будем определять вектор неизвестных через обратную матрицу коэффициентов A^{-1} : $X = A^{-1} \times B$

Считаем, что задача относится к классу сложных ввиду большой размерности. Декомпозируем СЛАУ на p подзадач, каждая из которых также является СЛАУ, но уже меньшей размерности:

$$Z \rightarrow \{Z_1, Z_2, \dots Z_i, \dots Z_p, Z_{св}\};$$

$$Z_i \rightarrow A_i X_i = B_i,$$

где $\dim A = n \times n$,

$\dim A_i \cong (n/p)^2 = n_i \times n_i$, где $n_i = n/p$, где p – число подсистем

В зависимости от портрета матрицы коэффициентов A эта задача может быть отнесена к разным классам сложных задач: **несвязной, слабосвязной, среднесвязной** или **сильносвязной**.

Если вне диагональных блоков все элементы равны 0, то имеем **несвязную сложную задачу**.

Представим матрицу A в виде блочно-диагональной матрицы, как на рис. ниже:

$$\begin{matrix} & A & X & B \\ \left[\begin{array}{ccc} \square & & \\ & \square & \\ & & \square \end{array} \right] & \left[\begin{array}{c} \\ \\ \end{array} \right] & = & \left[\begin{array}{c} \\ \\ \end{array} \right] \end{matrix}$$

Если вне диагональных блоков имеются неравные 0 элементы, то задачу решения СЛАУ можно, в зависимости от числа этих элементов вне диагональных блоков, отнести к остальным классам: **слабосвязной, среднесвязной или сильносвязной задачи.**

Кратко рассмотрим один из подходов к решению СЛАУ вида $A \cdot X = B$. Для эффективного решения такого рода СЛАУ целесообразно преобразовать матричную форму к виду блочно-диагональной с окаймлениями (БДМО). Для этого можно воспользоваться операцией разверстки матрицы. На рис. ниже x – это ненулевые элементы вне выделенных диагональных блоков. В результате операции разверстки эти ненулевые элементы удаляются из основной структуры матрицы коэффициентов в так называемое окаймление матрицы. В результате размерность модифицированной матрицы A' увеличивается, однако появляется возможность декомпозировать исходную СЛАУ на подсистемы меньшей размерности.

$$\begin{matrix} & A & X & B \\ \left[\begin{array}{ccc} \square & & x \ x \\ x \ x & \square & x \\ x & & \dots \square \end{array} \right] & \left[\begin{array}{c} \\ \\ \end{array} \right] & = & \left[\begin{array}{c} \\ \\ \end{array} \right] \\ \dim(A) = n \times n & & & \\ A' & X & B & \Rightarrow \text{операция разверстки матрицы } A \end{matrix}$$

$$\begin{matrix} \left[\begin{array}{ccc} \square & & \\ & \square & \\ & & \dots \square \end{array} \right] & \left[\begin{array}{c} \\ \\ \end{array} \right] & = & \left[\begin{array}{c} \\ \\ \end{array} \right] \\ \left. \begin{array}{c} \square \\ \square \\ \dots \square \end{array} \right\} n_c & \leftarrow Q & W_c & X_{CB} & B_{CB} \end{matrix}$$

$\dim A' = (n + N_c) \times (n + N_c)$ размерность увеличена на N_c за счет переноса ненулевых элементов вне диагональных блоков в окаймление.

В результате получим систему вида:

$$\underline{A'} \underline{X'} = \underline{B'}$$

$$\left[\begin{array}{cccc|c} A_1 & & & & C_1 \\ & A_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & A_p & C_p \\ \hline Q_1 & Q_2 & \dots & Q_p & W_c \end{array} \right] \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \\ X_c \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_p \\ B_c \end{bmatrix} \quad (*)$$

Размерность «новой системы» равна:

$\dim \underline{A'} = (n+N_c) \times (n+N_c)$, где N_c – число внешних связей между подзадачами (подсистемами), n – размерность исходной системы $AX=B$;

p – число подсистем вида $A_i X_i = B_i$

$$\dim A_i = n_i$$

$$\sum n_i = n$$

Q_i, C_i – матрицы связи подсистем:

$$\dim Q_i = N_c \times n_i$$

$$\dim C_i = n_i \times N_c$$

N_c – размерность уравнений связи

$$\dim X_c = N_c; \dim B_c = N_c; \dim W_c = N_c \times N_c$$

Используя блочный (клеточный) подход к решению (*), запишем (*) в виде:

$$\left[\begin{array}{c|c} \overline{A} & C \\ \hline Q & W_c \end{array} \right] \begin{bmatrix} X \\ \hline X_c \end{bmatrix} = \begin{bmatrix} B \\ \hline B_c \end{bmatrix}$$

Тогда получим систему матричных уравнений, из которой определим все неизвестные, а именно X_c и X :

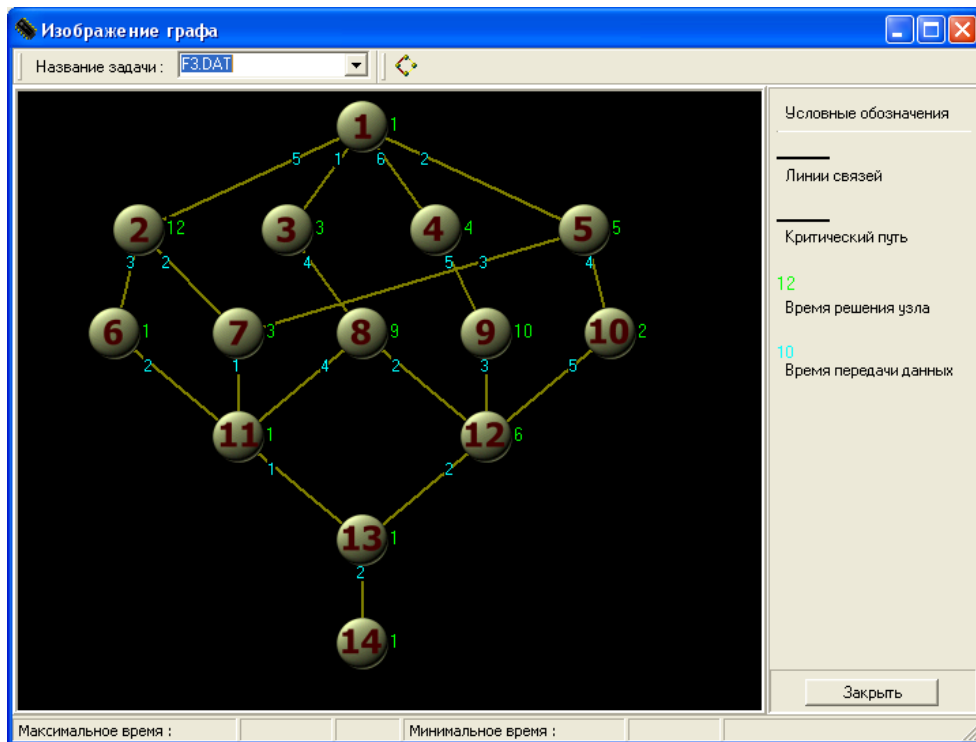
$$\begin{cases} \overline{A}X + CX_c = B & \Rightarrow X_c - ? \\ QX + W_c X_c = B_c & \Rightarrow X - ? \end{cases}$$

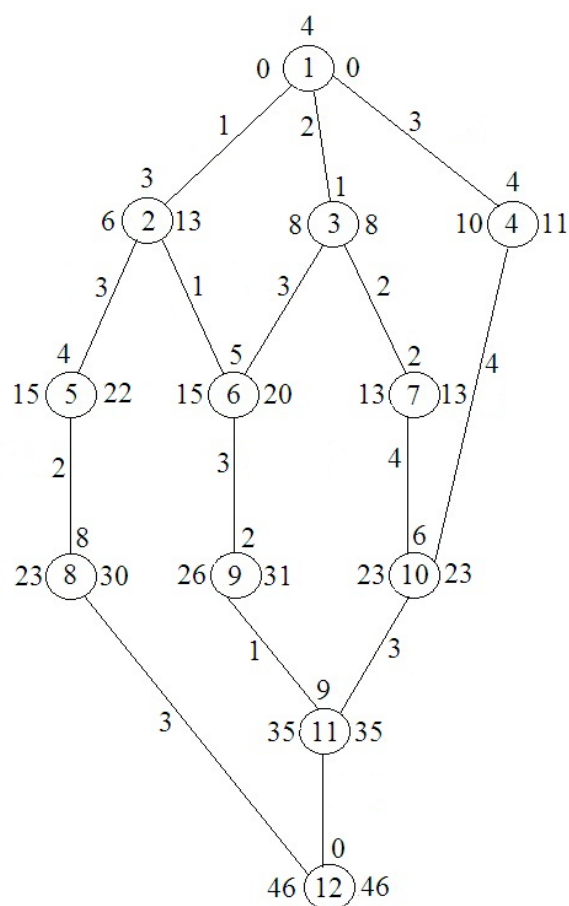
Если $N_c < n_i$, то данная задача относится к **классу слабосвязных**.

Если $N_c \approx n_i$, то данная задача относится к **классу среднесвязанных**

Если система линейных уравнений $AX = B$ имеет матрицу коэффициентов плотно заполненную элементами, то данная СЛАУ относится к классу **сильносвязных задач**.

2. Другой пример сложной задачи – сложная задача, которая моделируется ориентированным взвешенным по вершинам и ребрам графом. В зависимости от соотношения **средних значений весов вершин и ребер**, граф сложной задачи может моделировать (отображать) *слабосвязную, среднесвязную или сильносвязную задачу*.





В качестве модели задачи может выступать направленный граф, узлы которого отображают подзадачи. Каждой подзадаче соответствует часть (ветвь) программы, начав выполнение которой процессор заканчивает ее без прерываний. Дуги графа моделируют связи по данным между соответствующими подзадачами (ветвями программы).

Узлы графа взвешиваются целыми числами, соответствующими временам их выполнения в условных единицах (например, тактах) на процессорах. Дуги графа взвешиваются тоже целыми числами, соответствующими временам занятия канала связи (шины) при передаче данных от одного узла графа к другому.

Рассмотрим графы задач *без обратных связей*, с различным соотношением **средних значений времен выполнения узлов графа** ($t_{\text{вып}}$) и *передачи данных* ($t_{\text{пер}}$).

В соответствие с этим соотношением различают:

- **слабосвязные задачи** ($t_{\text{вып}} \gg t_{\text{пер}}$),
- **среднесвязные** ($t_{\text{вып}} \approx t_{\text{пер}}$),
- **сильносвязные** ($t_{\text{вып}} \ll t_{\text{пер}}$).

3. Характеристики параллельности (ускорение, эффективность, стоимость и ценность параллельного решения)

Степень параллелизма задачи (алгоритма)

Максимально возможное число независимых элементарных операций (действий), выполняемых при реализации задачи (алгоритма) на неограниченных аппаратных ресурсах (в режиме паракомпьютинга)

Ускорение параллельного вычисления

$$\xi = T_1(n) / T_p(n)$$

Чем ближе значение ускорения к p ($\xi \rightarrow p$), тем лучше разработанный параллельный алгоритм и соответствующая параллельная программа!

Эффективность параллельного вычисления

$$\xi^* = \xi / p = T_1(n) / (T_p(n) * p)$$

Чем ближе значение ускорения к p , тем более эффективен ($\xi^* \rightarrow 1$) параллельный алгоритм и соответствующая параллельная программа!

Цена параллельного решения

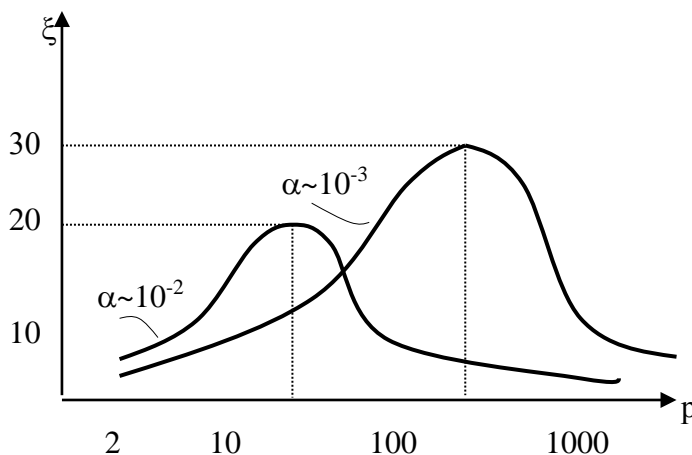
$$C_p = p * T_p(n)$$

Ценность параллельного решения

$$F_p = \xi / C_p = T_1(n) / (p * T_p^2(n)).$$

Иллюстрация приведенных характеристик на примере решения СЛАУ большой размерности путем декомпозиции на подзадачи меньших размеров. На графике ниже приведена зависимость ускорения при решении СЛАУ размерности $n=10^4$ на архитектурах с небольшой производительностью $\sim 10^6$ оп/сек. Однако характер кривых сохраняется для любых других соотношений размерности, связности и производительности.

Пример:



СЛАУ $n = 10^4$

α - коэффициент связности

$\alpha \rightarrow 10^{-4}$

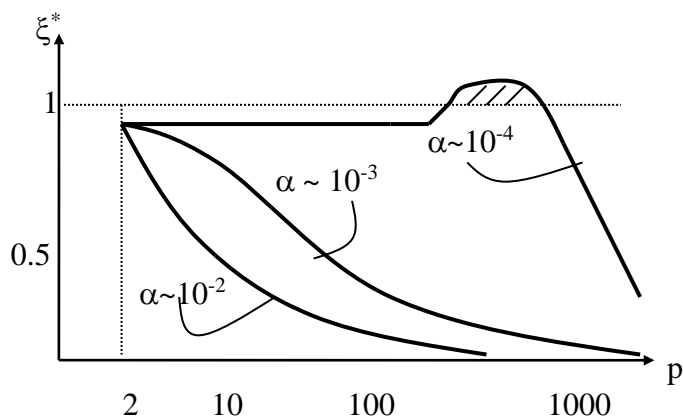
p - число вычислителей и число подзадач

Вывод: чем менее связная задача, тем больше число подсистем, на которые целесообразно разбивать исходную сложную задачу. Чем более связная задача, тем меньше число подсистем p на которые следует разбивать исходную сложную задачу.

На рис. ниже приведены графики эффективности решения в зависимости от связности подсистем уравнений.

Эффективность параллельных вычислений $\xi^* = \xi/p \rightarrow 1$, т.к. $\xi \rightarrow p$.

Если получаем реально эффективность в диапазоне 0.5-0.7, то решение считается достаточно успешным!

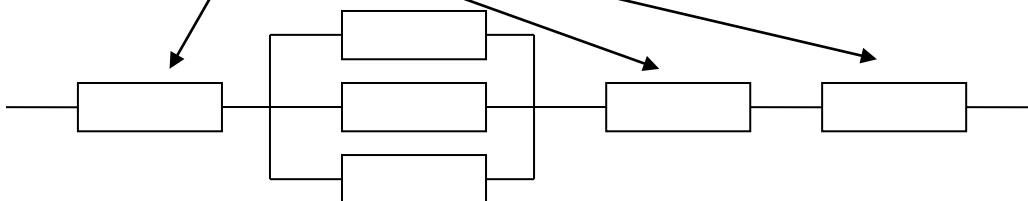


$$\xi^* = T_1(n) / (T_p(n) * p)$$

4. Закон Амдаля и его следствия

Используется для оценки параллельных алгоритмов, структура которых предполагает выполнение **всех операций либо с максимальной, либо с минимальной степенью** параллелизма, время на подготовку передачи данных отсутствует.

Например, для структуры алгоритма, изображенной ниже, где минимальная степень $=1$, а максимальная $=3$, можно использовать оценку ускорения по закону Амдаля.



Но прежде чем использовать сам закон для априорной оценки ускорения, выведем формулу закона исходя из приведенных выше определений.

Будем определять ускорение параллельного алгоритма исходя из анализа частей (долей) алгоритма, выполняемых последовательно и параллельно.

Предположим, что в алгоритме (программе) **доля** операций, которые нужно выполнять последовательно, равна β , где $0 \leq \beta \leq 1$ (при этом доля понимается не по статическому числу строк кода, а по числу операций в процессе выполнения).

Крайние случаи в значениях β соответствуют полностью **параллельным** ($\beta = 0$) и полностью **последовательным** ($\beta = 1$) алгоритмам и программам. Очевидно, что $(1 - \beta)$ – доля операций, выполняемых параллельно.

Пусть имеем некоторый компьютер, состоящий из p процессоров.

Определим ускорение выполнения алгоритма (программы) как отношение времени последовательного решения к времени параллельного решения:

$$S = \frac{T_1(n)}{(\beta + \frac{1-\beta}{p})T_1(n) + t_{\text{дон}}},$$

где $t_{\text{дон}}$ – время на накладные расходы (обмены, простои и т.п.);

β – доля операций, выполнение которых невозможно одновременно с другими операциями.

Проанализируем полученное соотношение и выведем следствия, одно из которых и называется законом Амдаля:

1. Если $\beta = 0, t_{don} = 0$, тогда $S = p$ (т.е. алгоритм полностью параллелен, отсутствует последовательная часть и ускорение максимально $S = p$)
2. Если $\beta < 0, t_{don} = 0$, тогда $S = \frac{1}{(\beta + \frac{1-\beta}{p})}$ - это выражение и есть **Закон**

Амдаля

3. Если β – любое, $t_{don} > 0$, тогда $S = \frac{T_1(n)}{t_{don}} < 1$,

Таким образом, если имеем большие затраты на обмен данными, простой и синхронизацию ($t_{don} > 0$), то все это может привести к неэффективности параллельного алгоритма ($S < 1$).

Используем закон Амдаля для определения ускорения алгоритма со структурой, приведенной в начале раздела 4. Будем использовать 3 процессора, что вполне соответствует структуре алгоритма.

Тогда: $\beta = 3/6 = 1/2$ и $(1 - \beta) = 1/2$. Ускорение $S = 1/(1/2 + 1/6) = 1,5$. Т.е. используя 3 процессора, получили ускорение только 1,5!

Рассмотрим такой пример: пусть **9/10** программы выполняется параллельно, а **1/10** по-прежнему последовательно, то ускорения более, чем в **10 раз** получить в принципе невозможно вне зависимости от качества реализации параллельной части кода и числа используемых процессоров (ясно, что ускорение=10 получается только в том случае, когда время исполнения последовательной части равно 0).

Посмотрим на проблему с другой стороны: а какую же часть кода надо ускорить (а значит и предварительно исследовать с точки зрения распараллеливания), чтобы получить заданное ускорение?

Ответ можно найти в следствии из закона Амдала: для того чтобы ускорить выполнение программы в **q** раз необходимо ускорить не менее, чем в **(1-1/q)**-ю часть программы. Следовательно, если есть желание ускорить программу в **100 раз** по сравнению с ее последовательным вариантом, то необходимо получить ускорение не менее, чем на **99.99%** кода, что почти всегда составляет значительную часть программы!

Отсюда **первый вывод** - прежде, чем основательно переделывать код для перехода на параллельный компьютер или суперкомпьютер надо основательно подумать. Если, оценив заложенный в программе алгоритм, ясно, что доля последовательных операций велика, **то на значительное ускорение рассчитывать явно не приходится и нужно думать о замене отдельных компонент алгоритма.**

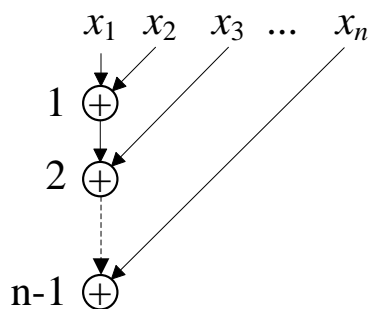
В ряде случаев последовательный характер алгоритма изменить не так сложно. Допустим, что в программе есть следующий фрагмент для вычисления суммы n чисел:

```

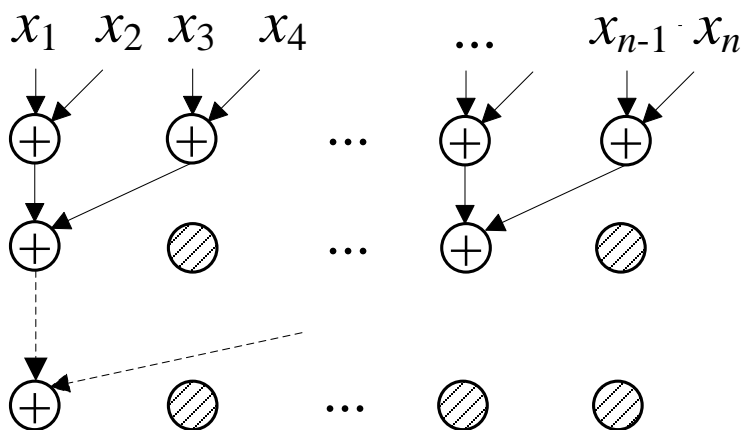
s = 0
Do i = 1, n
    s = s + x(i)
EndDo

```

По своей природе он строго последователен, так как на i -й итерации цикла требуется результат с $(i-1)$ -й и все итерации выполняются одна за одной. Имеем 100% последовательных операций, а значит и никакого эффекта от использования параллельных компьютеров. Ниже приведено дерево последовательного вычисления такой суммы n чисел:



Вместе с тем, в большинстве реальных программ нет существенной разницы, в каком порядке складывать числа, поэтому можно выбрать иную схему сложения. Например, сначала найдем сумму пар соседних элементов: $x(1)+x(2)$, $x(3)+x(4)$, $x(5)+x(6)$ и т.д. Заметим, что при такой схеме все пары можно складывать одновременно! На следующих шагах опять суммируем пары абсолютно аналогично, в результате получим вариант параллельного алгоритма (*схема сдвигания, попарное суммирование*), схема которого приведена ниже:



Казалось бы, в данном случае все проблемы удалось разрешить. Но представим, что доступные процессоры разнородны по своей производительности. Значит наступит такой момент, когда какой-либо

процессор все еще выполняет свои вычисления, а другой уже все вычислил и бесполезно простаивает в ожидании. Отсюда важный вывод: **Если разброс в производительности компьютеров большой, то и эффективность всей системы при равномерной загрузки процессоров будет крайне низкой.**

Возникает еще одна проблема. Предположим, что все процессоры одинаковы, и они выполнили свою работу, но результат-то надо передать другому для продолжения процесса суммирования. На **передачу уходит время и в это время процессоры опять простаивают.**

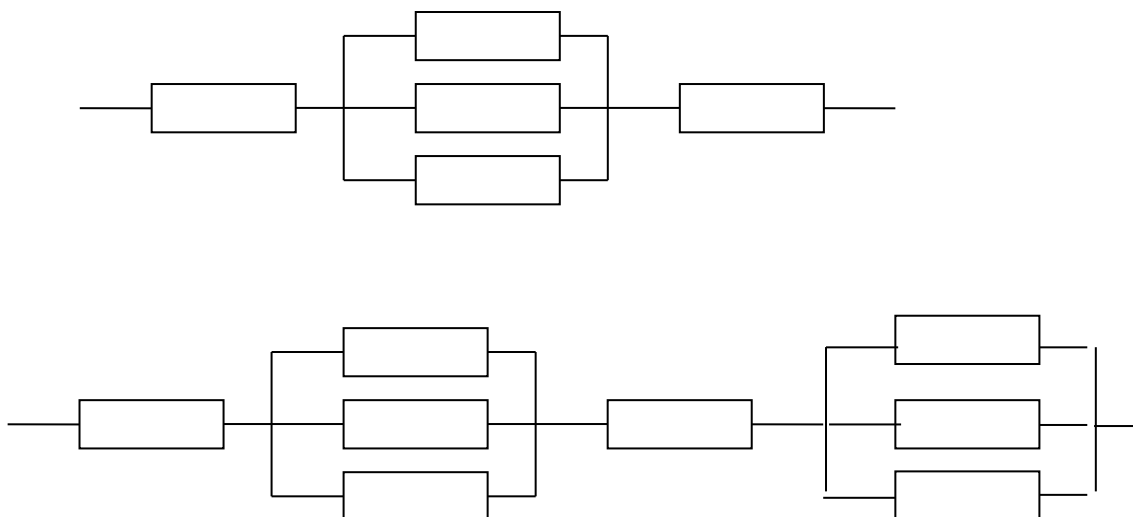
Напрашивается общий вывод: чтобы параллельная вычислительную систему или супер-ЭВМ работали с максимальной эффективностью на конкретной программе, **необходимо тщательное согласование структуры программ и алгоритмов с особенностями архитектуры параллельных вычислительных систем.**

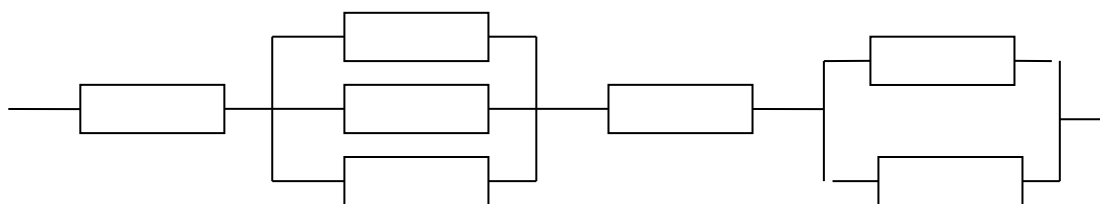
Замечание: Верно ли утверждение: чем мощнее компьютер, тем быстрее на нем можно решить данную задачу?

Нет, это не верно. Можно пояснить простым бытовым примером. Если один землекоп выкопает яму $1\text{м} \times 1\text{м} \times 1\text{м}$ за 1 час, то два таких же землекопа это сделают за 30 мин - в это можно поверить. А за сколько времени эту работу сделают 60 землекопов? За 1 минуту? Конечно же нет! Начиная с некоторого момента, они будут просто мешаться друг другу, не ускоряя, а замедляя процесс. Так же и в компьютерах: если задача слишком мала, то мы будем дольше заниматься распределением работы, синхронизацией процессов, сборкой результатов и т.п., чем непосредственно полезной работой.

Упражнение для самостоятельного выполнения:

1. Оценить возможное **ускорение и эффективность** при параллельной реализации алгоритмов со следующими структурами:





2. Определить **ускорение и эффективность** для схемы сдваивания для случаев: $n=8, 11, 15$.