

Тема. Сложные задачи, их характеристики и пути эффективного решения

Основные вопросы:

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

Основные этапы решения сложной задачи на параллельных вычислительных системах

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. 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.

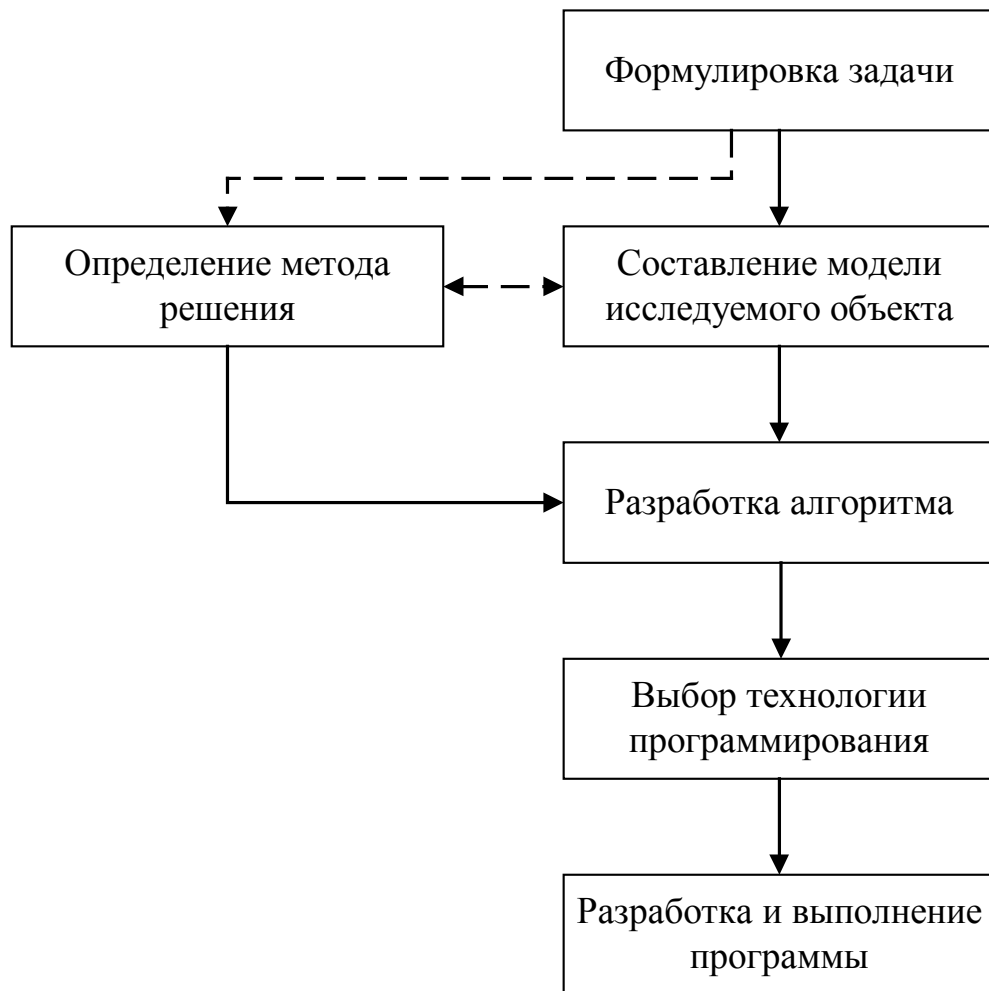


Рис. 1. Процесс решения задачи на параллельной ВС



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

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

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 программ

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

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

$T_1(n)$ – трудоемкость решения некоторой задачи $Z(n)$, где n – это размерность задачи, на одном вычислителе с помощью **наилучшего последовательного алгоритма**.

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

$T(Z_i)$ – трудоемкость решения подзадачи Z_i

$T(Z_{\text{св}})$ – трудоемкость решения задачи связи $Z_{\text{св}}$

В зависимости от соотношения трудоемкостей задачи связи и подзадач 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.

Пусть W_{ij} – трудоемкость обменных взаимодействий между $Z_{i\text{-ой}}$ и $Z_{j\text{-ой}}$ подзадачами в процессе решения задачи Z .

Если:

1. $\sum_j \sum_i W_{ij} \rightarrow 0$, или $T(Z_{\text{св}}) \rightarrow 0$, то Z относится к **классу несвязных сложных**.
2. $\sum_j \sum_i W_{ij} < \max T(Z_i)$, или $T(Z_{\text{св}}) < \max T(Z_i)$, то Z – **слабосвязная задача**.
3. $\sum_j \sum_i W_{ij} \cong \max T(Z_i)$, или $T(Z_{\text{св}}) \approx \max T(Z_i)$, то Z – **среднесвязная задача**.
4. $\sum_j \sum_i W_{ij} > \max T(Z_i)$, или $T(Z_{\text{св}}) > \max T(Z_i)$, то Z – **сильносвязная задача**.

Иллюстрация классов сложных задач на примере решения СЛАУ (система линейных алгебраических уравнений) большой размерности

Пусть имеем: СЛАУ, которая представлена в матричной форме:

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

$$X = A^{-1} \times B$$

Рассмотрим задачу как сложную, декомпозированную на p подзадач 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 эта задача может быть отнесена к **несвязной, слабосвязной, среднесвязной и сильносвязной**.

$$\begin{array}{ccc} A & X & B \\ \left[\begin{array}{ccc} \boxed{} & & \\ & \boxed{} & \\ & & \boxed{} \end{array} \right] & \left[\begin{array}{c} \\ \\ \end{array} \right] & \left[\begin{array}{c} \\ \\ \end{array} \right] \\ & n \times n & \end{array} \Rightarrow$$

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

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

Для эффективного решения такого рода СЛАУ целесообразно преобразовать матричную форму к виду блочно-диагональной с окаймлениями (БДМО)

— — — — —

$$AX=B$$

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

Размерность «новой системы» равна $(n \times N_c) \times (n + N_c)$, 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:

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

Тогда имеем систему:

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

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

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

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

Характеристики параллельности

Ускорение параллельного вычисления

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

Чем менее связная задача, тем больше число подсистем на которые целесообразно разбивать исходные подзадачи. Чем более связная задача, тем меньше число подсистем p на которые следует разбивать исходные подзадачи. Чем ближе значение ускорения к p ($\xi \rightarrow p$), тем лучше 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

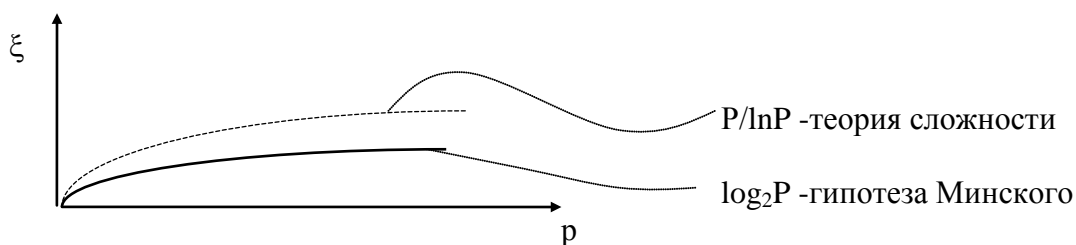
Цена параллельного решения

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

Ценность параллельного решения

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

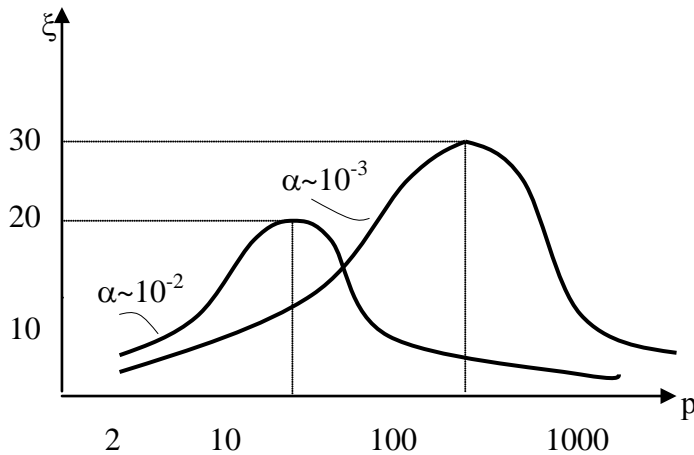
Иллюстрация приведенных характеристик на примере решения СЛАУ большой размерности путем декомпозиции на подзадачи меньших размеров



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

(Реально 0.5-0.7)

Пример:



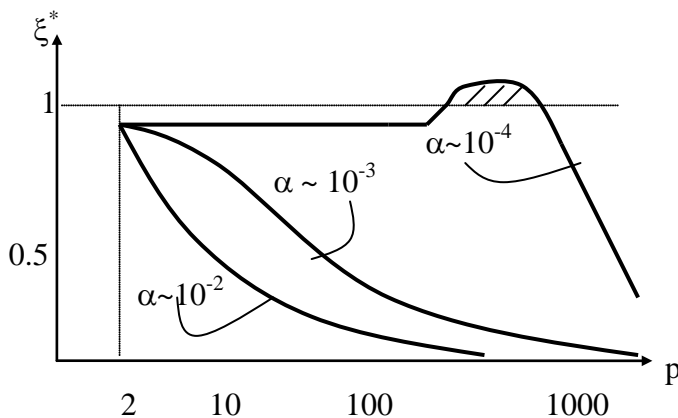
СЛАУ $n = 10^4$

α - коэффициент связности

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

p – число вычислителей и число подзадач

Чем менее связная задача, тем больше число подсистем, на которые целесообразно разбивать исходные подзадачи. Чем более связная задача, тем меньше число подсистем p на которые следует разбивать исходные подзадачи.

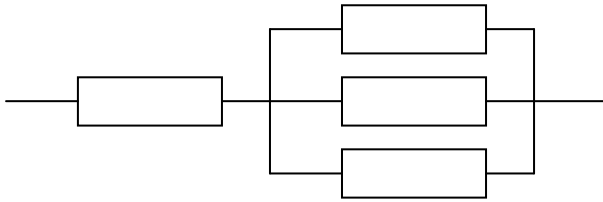


$$\xi^* = T_1(n) / (T_p(n) * p)$$

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 изображенной ниже, где минимальная степень = 1, максимальная = 3, можно использовать оценку ускорения по закону Амдаля.



Определим ускорение параллельного алгоритма исходя из анализа частей алгоритма, выполняемых последовательно и параллельно:

Предположим, что 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 ($\beta = 1$) программам.

Чтобы оценить, какое ускорение S может быть получено на компьютере из p процессоров при данном значении β , можно воспользоваться законом Амдала:

$$S = \frac{T_1(n)}{(\beta + \frac{1-\beta}{p})T_1(n) + t_{\text{доп}}}, \text{ где } t_{\text{доп}} - \text{ время на накладные расходы}$$

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

Проанализируем полученное соотношение:

1. Если $\beta = 0$, $t_{\text{доп}} = 0$, тогда $S = p$ (т.е. алгоритм полностью параллелен, отсутствует последовательная часть)
2. Если $\beta > 0$, $t_{\text{доп}} = 0$, тогда $S = \frac{1}{(\beta + \frac{1-\beta}{p})}$ - Закон Амдала
3. Если β – любое, $t_{\text{доп}} \gg 0$, тогда $S = \frac{T_1(n)}{t_{\text{доп}}} < 1$,

Имеем большие затраты на обмен данными и синхронизацию, что может привести к неэффективности параллельного алгоритма.

Если 9/10 программы исполняется 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 10 получается только в том случае, когда время исполнения параллельной части равно 0).

Посмотрим на проблему с другой стороны: а какую же часть кода надо ускорить (а значит и предварительно исследовать), 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 + a(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.

Вместе с тем, выход очевиден. Поскольку в большинстве реальных программ нет существенной разницы, в каком порядке складывать числа, выберем иную схему сложения. Сначала найдем сумму пар соседних элементов: $a(1)+a(2)$, $a(3)+a(4)$, $a(5)+a(6)$ и т.д. Заметим, что при такой схеме все пары можно складывать одновременно! На следующих шагах будем действовать абсолютно аналогично, получив вариант параллельного алгоритма.

Казалось бы в данном случае все проблемы удалось разрешить. Но представьте, что доступные вам процессоры разнородны по своей производительности. Значит будет такой момент, когда кто-то из них еще трудится, а кто-то уже все сделал и бесполезно простаивает в ожидании. Если разброс в производительности компьютеров большой, то и эффективность всей системы при равномерной загрузке процессоров будет крайне низкой.

Но пойдем дальше и предположим, что все процессоры одинаковы. Проблемы кончились? Опять нет! Процессоры выполнили свою работу, но результат-то надо передать другому для продолжения процесса суммирования, а на передачу уходит время и в это время процессоры опять простаивают.

Словом, заставить параллельную вычислительную систему или супер-ЭВМ работать с максимальной эффективностью на конкретной программе это,

задача не из простых, поскольку **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.