Тема 4.1: Сложные задачи, их характеристики и пути эффективного решения

Основные вопросы:

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.

Если структура программы, реализующей некоторую задачу, не соответствует особенностям архитектуры параллельной ВС (особенностям аппаратно-программной среды ВС), то эффективность решения задачи 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 шаге не были учтены особенности ВС, то, скорее всего, большой производительности достичь не удастся. В самом деле, ориентация на параллельную векторную систему предполагает векторизацию внутренних циклов, а ориентация на вычислительный кластер требует распараллеливания значительных фрагментов кода. И то, и другое свойство программ определяется свойствами заложенных в них методов. То же самое касается и алгоритмов. При этом нужно 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. $\mathbf{t_{доп.}}$ – допустимое время решения задачи.
4. **P** – требуемая производительность, $\mathbf{P = T/t_{доп.}}$

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).

$$\mathbf{З} \xrightarrow{\text{декомпозиция}} \mathbf{\{З_1, З_2, \ldots З_i, \ldots З_p, З_{св}\}},$$

где $З_i$ – i-ая подзадача.

$З_{св}$ – задача связи.

Обозначим:

$\mathbf{T_1(n)}$ - трудоемкость решения некоторой задачи З(n) на одном вычислителе с помощью **наилучшего последовательного алгоритма**, где n – это размерность задачи**.**

$\mathbf{T_p(n)}$ – трудоемкость решения задачи **З(n)** на **p** вычислителях с использованием некоторого **параллельного алгоритма**.

$\mathbf{T(З_i)}$ – трудоемкость решения подзадачи $\mathbf{З_i}$

$\mathbf{T(З_{св})}$ – трудоемкость решения задачи связи $\mathbf{З_{св}}$

В зависимости от соотношения трудоемкостей задачи связи $\mathbf{З_{св}}$ и подзадач $\mathbf{З_i}$ зависит к какому классу относится исходная сложная задача **З**.

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 $\mathbf{W_{ij}}$ – трудоемкость обменных взаимодействий между $З_{i\text{-ой}}$ и $З_{j\text{-ой}}$ подзадачами в процессе решения задачи З.

Если:

1. $\mathbf{\sum_j \sum_i W_{ij} \to 0}$, или $\mathbf{T(З_{св}) \to 0}$, то **З(n) – несвязная сложная задача.**
2. $\mathbf{\sum_j \sum_i W_{ij} << \max T(З_i)}$, или $\mathbf{T(З_{св}) << \max T(З_i)}$, то **З(n) – слабосвязная задача.**
3. $\mathbf{\sum_j \sum_i W_{ij} \cong \max T(З_i)}$, или $\mathbf{T(З_{св}) \approx \max T(З_i)}$, то **З(n) – среднесвязная задача.**
4. $\mathbf{\sum_j \sum_i W_{ij} >> \max T(З_i)}$, или $\mathbf{T(З_{св}) >> \max T(З_i)}$, то **З(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:

$$\mathbf{A \times X = B} \qquad \mathbf{X - ?}$$

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

Рассмотрим задачу как сложную, декомпозированную на **p** подзадач меньшей размерности:

$$\mathbf{З \to \{З_1, З_2, \ldots З_i, \ldots З_p, З_{св}\};}$$

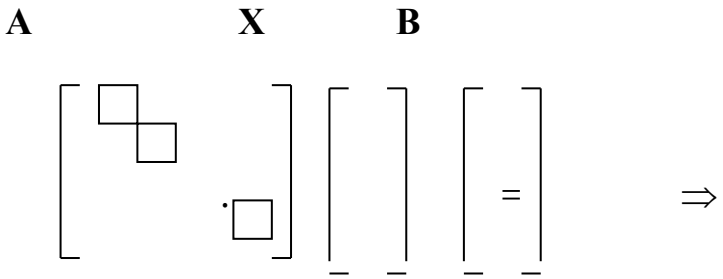
$$\mathbf{З_i \to A_i X_i = B_i,}$$

где $\mathbf{dim A = n \times n,}$

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

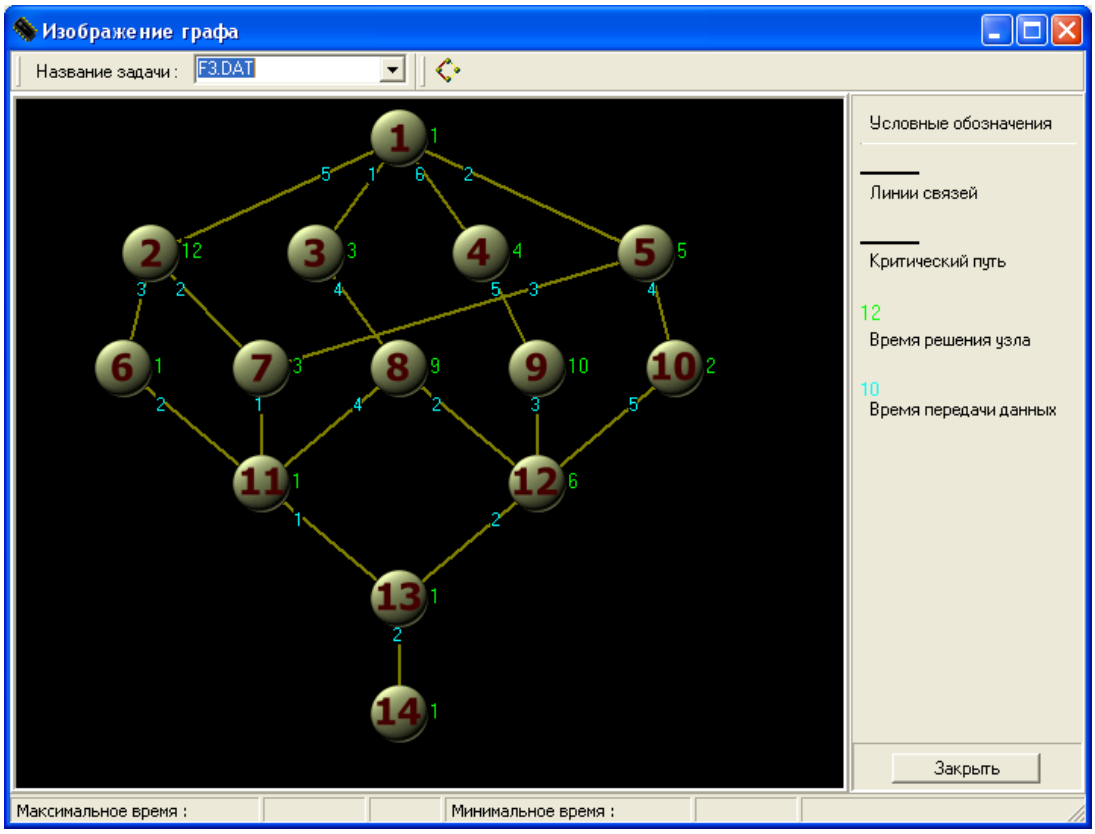
В ***зависимости от портрета матрицы коэффициентов*** **A** эта задача может быть отнесена к ***несвязной, слабосвязной, среднесвязной*** или ***сильносвязной***.

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



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.***

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 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* (**$t_{вып}$**) и *передачи данных* (**$t_{пер}$**).

В соответствие с этим соотношением различают:

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

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 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

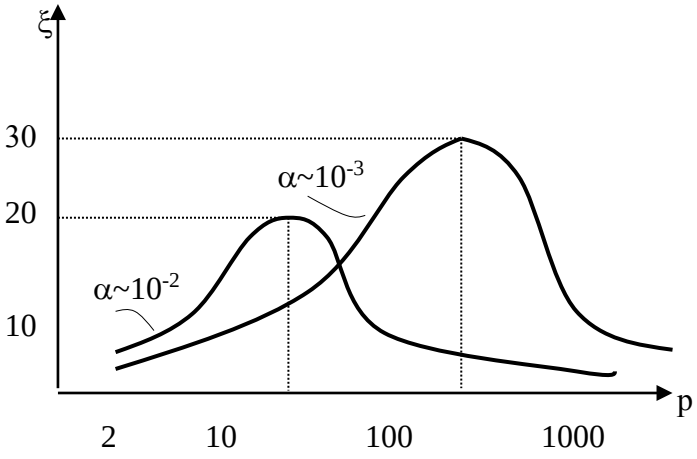
$$C_p = p*T_p(n)$$

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

Пример:



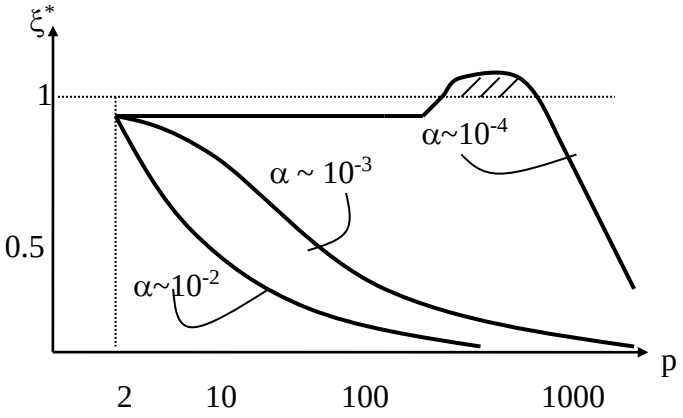
СЛАУ n = 10^4
α - коэффициент связности
$\alpha \rightarrow 10^{-4}$
p – число вычислителей и число подзадач

Чем менее связная задача, тем больше число подсистем, на которые целесообразно разбивать исходную сложную задачу. Чем более связная

задача, тем меньше число подсистем p на которые следует разбивать исходную сложную задачу.

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

(Реально 0.5-0.7)

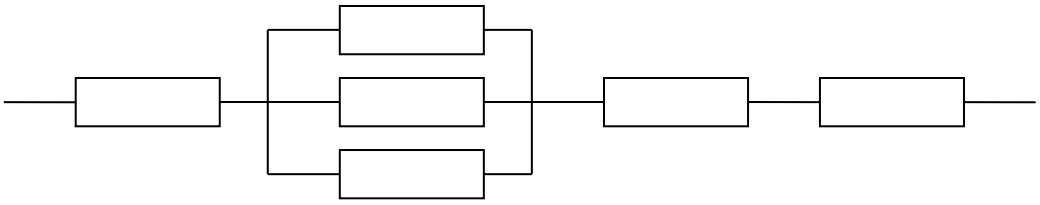


$$\xi^* = T_1(n) / (T_p(n) * p)$$

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, 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 в алгоритме (программе) ***доля*** операций, которые нужно выполнять последовательно, равна β, где **0<= β <=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 (β =0)** и полностью **последовательным (β =1)** программам.

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

Определим ускорение как отношение времени последовательного решения к времени параллельного решения:

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

где $t_{доп}$ – 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 β **=0**, $t_{доп}$ **=0**, тогда $S=p$ (т.е. алгоритм полностью параллелен, отсутствует последовательная часть и ускорение максимально $S=p$)
2. Если β **<>0**, $t_{доп}$ **=0**, тогда $S = \frac{1}{(\beta + \frac{1-\beta}{p})}$ **- Закон Амдаля**

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

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

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 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.***

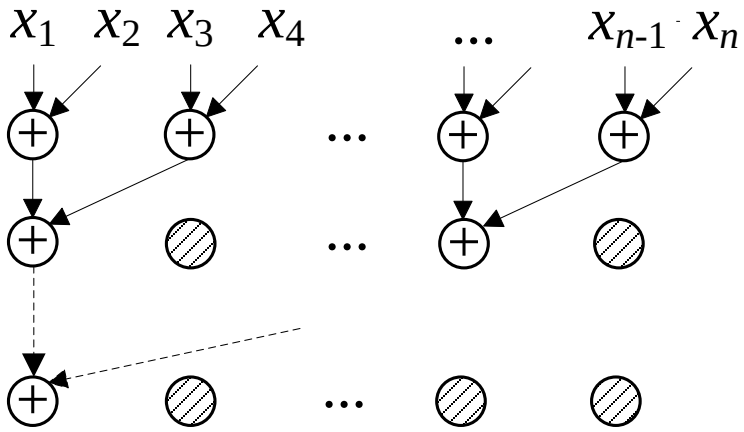
В ряде случаев последовательный характер алгоритма изменить не так сложно. Допустим, что в программе есть следующий фрагмент для вычисления суммы n чисел:

```
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.

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: ***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 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 загрузки 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 эффективность 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 1м*1м*1м 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.*