2. Моделирование и анализ параллельных вычислений.

Коммуникационная трудоемкость параллельных алгоритмов для кластеров ($l = 1$).

Основной способ выполнения коммуникационных операций – **пакетный метод**

Оценка трудоемкости операции передачи данных между 2 узлами кластера

- **Подход 1**:
  - $t_н$ не зависит от объема данных,
  - $t_с$ не зависит от числа пакетов

$$t_{пд} = t_н + mt_к + t_с$$

Ограничения не соответствуют действительности

Оценка времени (трудоемкости) **неточна**

Оценка трудоемкости операции передачи данных между 2 узлами кластера

- **Подход 2:**
  - **Учитывается**
    - n **- число пакетов,** $n = m/(V_{max} - V_c)$
    - V_c **- объем служебных данных в каждом пакете,**
    - V_{max} **- максимально возможный для сети размер пакета,**
    - $tнач_0$ **– аппаратная (сетевая) задержка (латентность),**
    - $tнач_1$ **– время подготовки к передаче в сети 1 байта.**

Оценка трудоемкости операции передачи данных между 2 узлами кластера

- **Предполагается**
  - **Подготовка** данных для 2,3, … пакетов **совмещена** с **пересылкой** предшествующих пакетов.
  - **Учитывается увеличение** объема передаваемой информации за счет добавления **служебных данных** (заголовков пакетов)

Оценка трудоемкости операции передачи данных между 2 узлами кластера

- Подход 2 – **итоговое соотношение**

$$t_{пд} = \begin{cases} t_{нач_0} + m \cdot t_{нач_1} + (m + V_c) \cdot t_к, & n = 1 \\ t_{нач_0} + (V_{\max} - V_c) \cdot t_{нач_1} + (m + V_c \cdot n) \cdot t_к, & n > 1 \end{cases}$$

Оценка трудоемкости операции передачи данных между 2 узлами кластера

- **Подход 3 (упрощение подхода 1)** – ***модель Хокни*** (R.W. Hocney, 1994) – используется для грубых оценок трудоемкости

$$t_{пд} = t_н + mt_к = t_н + m/R$$

- Оценки через вычислительные эксперименты на кластере:
  - $t_н$ - время передачи сообщения длины 0 для подходов 1 и 3,
  - $tнач_0$, $tнач_1$ для подхода 2 - можно оценить через аппроксимацию
    $t_{пд}$ - времени передачи сообщений размером от 0 до V_{max}
  - $R = max\ (t_{пд} / m)$ при варьировании m

Этапы разработки параллельных алгоритмов (распараллеливания)

1. **Анализ** общей схемы вычислений - для **разделения** на независимые (относительно) **подзадачи**.
2. **Определение** информационных **взаимодействий** между **подзадачами**.
3. **Масштабирование** алгоритма с учетом числа CPU (**укрупнение** или **детализация** подзадач).
4. **Распределение подзадач** между CPU системы

Примечание. Это общий подход, независимо от типа вычислительной системы, исходной задачи и метода решения.

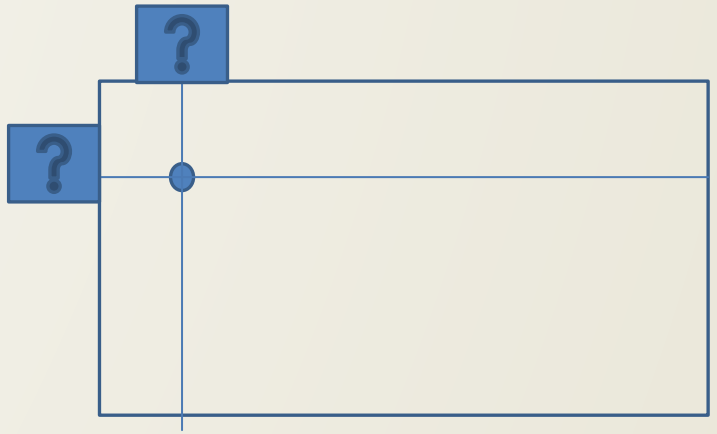
Дополнительные предположения

- **Равномерность** загрузки всех CPU (балансировка).
- **Минимизация** коммуникационных взаимодействий между подзадачами.
- **Возможность пересмотра** шагов после анализа показателей производительности.
- Информационные взаимодействия:
  - **Передача сообщений**
    для МВС с распределенной памятью.
  - **Операции доступа** к **общим переменным**
    для МВС с общей памятью

Этап 1 разработки параллельных алгоритмов

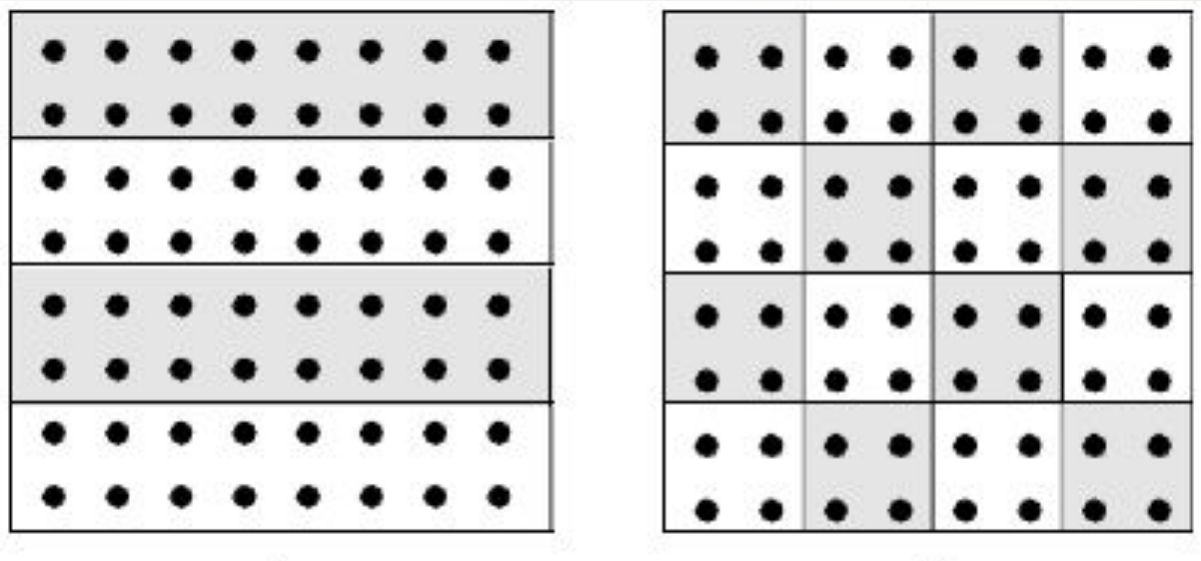
1. Разделение вычислений на независимые подзадачи

- **Требования** к подзадачам:
  - **Равные объемы** вычислений
  - **Минимум** информационных зависимостей
    - **Меньше** передач данных
    - **Больше** объем сообщений



Два основных типа вычислительных схем, основанных на разделении данных:

Ленточная схема **Блочная схема**



Сфера применимости – **однотипная** обработка большого набора данных:

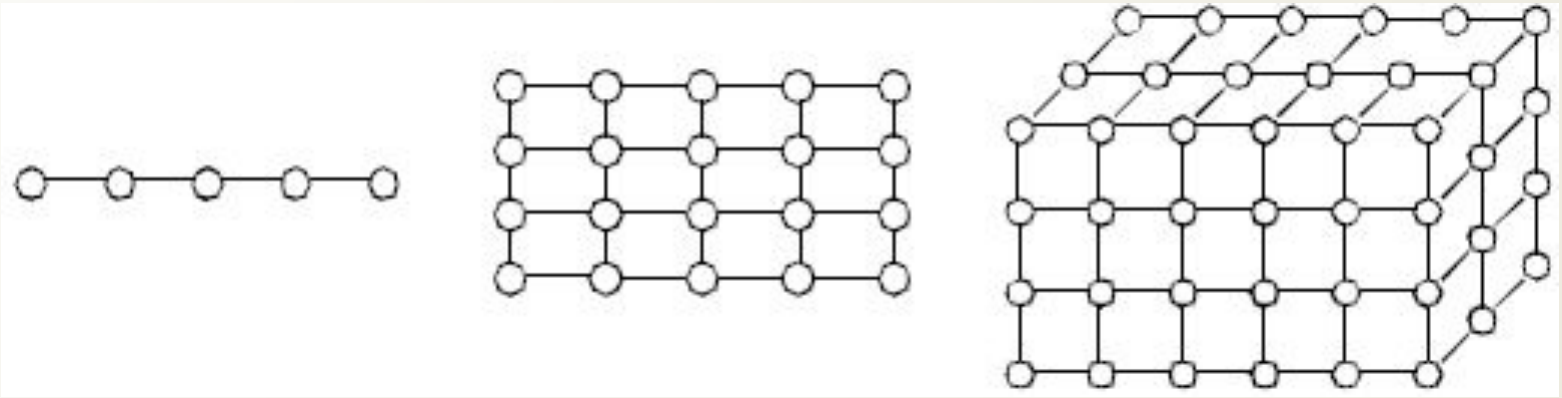
- Матричные вычисления
- Численные методы решения уравнений в частных производных.

Имеет место **параллелизм по данным**

Разделение на подзадачи = разделение данных.

Возможны 1,2,3D наборы подзадач с информационными связями между **ближайшими соседями** – **сетки**, или **решетки**.

Сетки, или решетки



Выполнение **разных** операций над одним набором данных – **функциональный параллелизм**

- Обработка **разных** запросов к БД
- **Одновременное** выполнение разных алгоритмов для одних и тех же данных

Функциональная декомпозиция м.б. использована для **конвейерной обработки данных:**

- **Ввод**
- **Обработка**
- **Сохранение**

Этап 2 разработки параллельных алгоритмов

2. Выделение информационных зависимостей

- **Взаимосвязан** с этапом 1:
  - Выделение подзадач должно учитывать возможные информационные связи
  - Анализ объема информационных обменов может потребовать изменения декомпозиции

Формы информационного взаимодействия

- **Схемы передачи данных**:
  - **Локальные** – **обмен для части подзадач** (как правило, на соседних CPU).
  - **Глобальные - обмен между всеми подзадачами.**
  - **Структурированные – стандартные регулярные схемы** (кольцо, решетка и т.д.).
  - **Произвольные .**
  - **Статические – фиксируются при проектировании программы вычислений**
  - **Динамические – определяются во время выполнения программы** (run-time)

Формы информационного взаимодействия

Схемы передачи данных:

- **Синхронные** – операции передачи данных
  - начинают выполняться только при готовности всех участников,
  - завершаются только по окончанию всех обменов
- *Просты для применения*
- **Асинхронные** – участники могут не ждать других для начала и завершения обменов
- *Снижают временные задержки*

Этап 3 разработки параллельных алгоритмов

3. Масштабирование

- Необходимо, если
  число подзадач ≠ количеству CPU

- **Типы масштабирования:**
  - **Агрегация**
  - **Декомпозиция**

Агрегация

Укрупнение вычислений для **уменьшения** числа подзадач.

- В **результате** должно соблюдаться *(см. этап 1)* :
  - **Одинаковая** вычислительная сложность подзадач.
  - **Минимально** возможный **уровень** объема и интенсивности **информационных взаимодействий** между подзадачами

- *В первую очередь объединяют коммуникационно трудоемкие подзадачи*

Декомпозиция

Детализация вычислений (увеличение числа подзадач) **для загрузки** всех доступных CPU.

Декомпозиция выполняется до базовых задач – с известными параллельными алгоритмами решения

Этап 4 разработки параллельных алгоритмов

4. Распределение подзадач между CPU.

- **Необходимо** для МВС с распределенной памятью.
- **Не требуется**, если

1) число подзадач = количеству CPU

2) все CPU связаны напрямую (полный граф)

3) **система с общей памятью** – распределение выполняется автоматически ОС.

Практические рекомендации

- **Анализируем** задачу для выделения подзадач, которые могут выполняться одновременно
- **Изменяем** структуру задачи для эффективного выполнения подзадач:
  - найти зависимости между подзадачами,
  - организовать исходный код для эффективного управления.
- **Реализуем** параллельный алгоритм в исходном коде с помощью технологий параллельного программирования

Технологии параллельного программирования

В основе может быть:

- **Язык параллельного программирования.**
- **Прикладной программный интерфейс** (API), реализованный с помощью библиотечного интерфейса.
- **Расширение** языка последовательного программирования

Примеры технологий ПП

- **OpenMP**: директивы компилятора для простого параллельного программирования.
- **MPI**: библиотечные подпрограммы для реализации высокоэффективной переносимости.
- **Java**: параллельность заложена в языке программирования на основе встроенных типов данных.

3. Основы технологии OpenMP.

Модель «fork-join».

Классификация переменных.

Основные директивы и их опции.

Распараллеливание по данным и по операциям.

Решение проблемы синхронизации.

Технология разработки параллельных программ для МВС с общей памятью (OpenMP)

- **OpenMP** (Open Multi-Processing) —
  - открытый развивающийся стандарт для распараллеливания программ на языках C, C++, Fortran,
  - включает описание
    - директив компилятора,
    - библиотечных процедур,
    - переменных ОС,

  для программирования многопоточных приложений для МВС с общей памятью (имеется версия и для кластеров).
- Наиболее популярная задача OpenMP — написание программ, ориентированных на циклы

Модель программирования OpenMP

Разветвление-объединение (*fork-join*)

- Работа программы начинается с **одного** (корневого) потока, или **нити** (**thread** – нить, **тред**).
- Для добавления в программу параллелизма выполняется разветвление на несколько потоков, чтобы создать **группу потоков**.
- Потоки группы выполняются параллельно в рамках фрагмента кода, т.наз. **параллельного участка**.
- В конце параллельного участка все потоки заканчивают свою работу и снова **объединяются** вместе.
- После этого корневой поток продолжает выполняться до тех пор, пока не начнется **следующий** параллельный участок (или не наступит **конец** программы).

Модель программирования OpenMP