Разработка параллельных алгоритмов

Основные этапы

1. Декомпозиция задачи
2. Выявление информационных зависимостей между подзадачами
3. Масштабирование подзадач
4. Балансировка нагрузки

PCAM = partitioning, communication, agglomeration, mapping

Декомпозиция

- Под декомпозицией понимается разбиение задачи на относительно независимые части (подзадачи).
- Виды декомпозиции: по заданиям, по данным, по информационным потокам.
- Выбор декомпозиции определяется: задачей и возможностями вычислительной системы

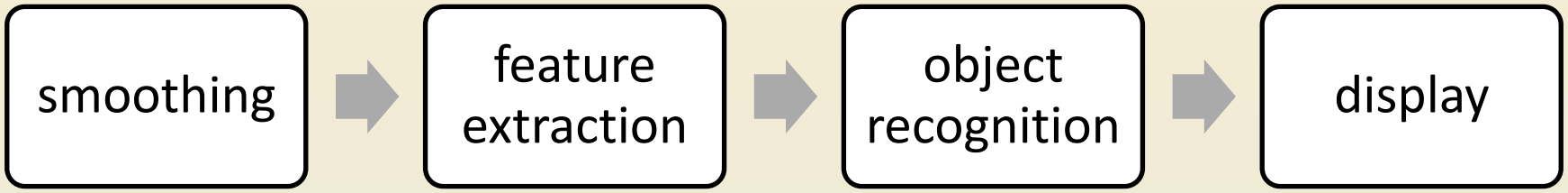
Декомпозиция по заданиям

- Функциональная декомпозиция (task parallelism) – разбиение задачи на подзадачи таким образом, чтобы каждая подзадача выполняла отдельную относительно независимую функцию

$$f(A) = f_1(A) * f_2(A) + f_3(A) * f_4(A)$$

Декомпозиция по заданиям: конвейер

- Один из вариантов функциональной декомпозиции: конвейерная обработка (**pipeline**)



Декомпозиция по заданиям: конвейер

Проблемы и ограничения конвейерной обработки:

- «холодный старт»: первая порция данных обрабатывается последовательно.
- балансировка нагрузки: разная нагрузка на каждом этапе обработки приводит к несбалансированности конвейера.
- ограниченная масштабируемость: определяется числом этапов конвейера

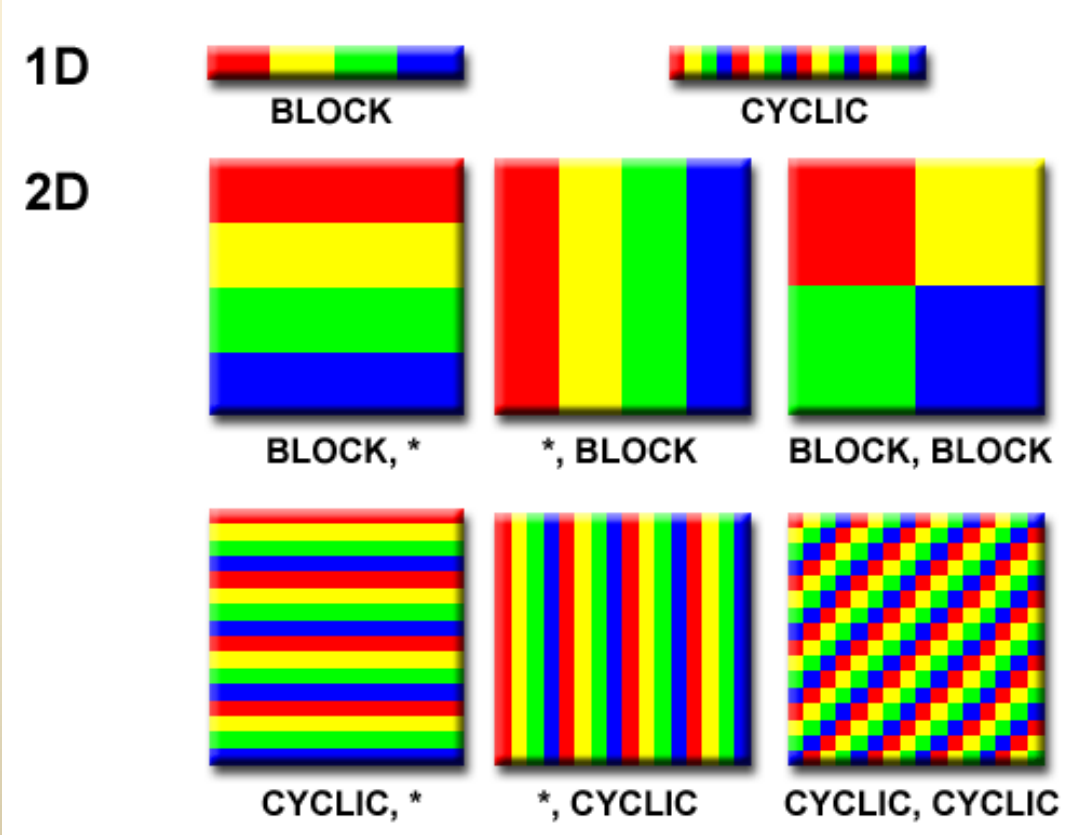
Декомпозиция по информационным потокам

- Декомпозиция по информационным потокам выделяет подзадачи, работающие с одним типом данных.

$$f(A,B,C) = f_1(A) * f_2(A) + f_3(B) * f_4(B) * f_5(C)$$

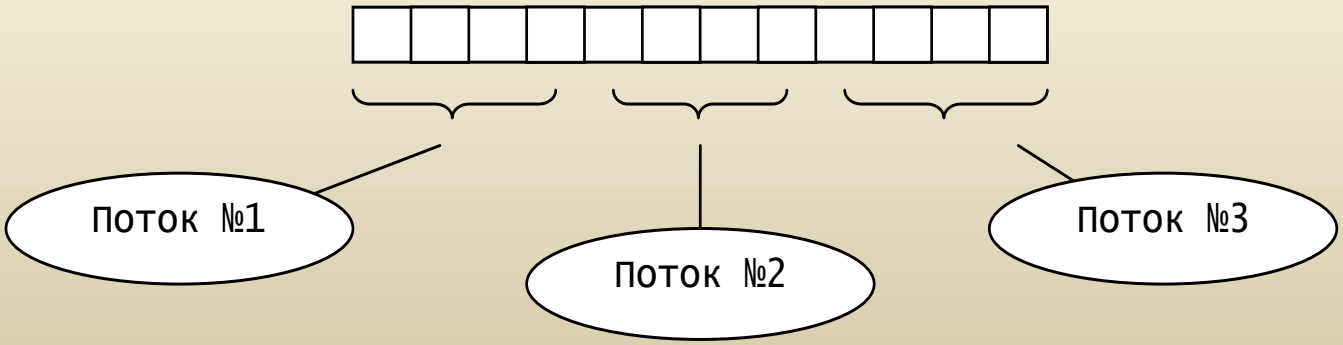
Декомпозиция по данным

- При декомпозиции по данным каждая подзадача выполняет одни и те же действия, но с разными данными (**data parallelism**)



Декомпозиция по данным

- Два основных принципа разделения данных между подзадачами – статический и динамический.
- При **статической декомпозиции** фрагменты данных назначаются потокам до начала обработки и, как правило, содержат одинаковое число элементов для каждого потока.
- Варианты статической декомпозиции: разделение по диапазону, круговое разделение

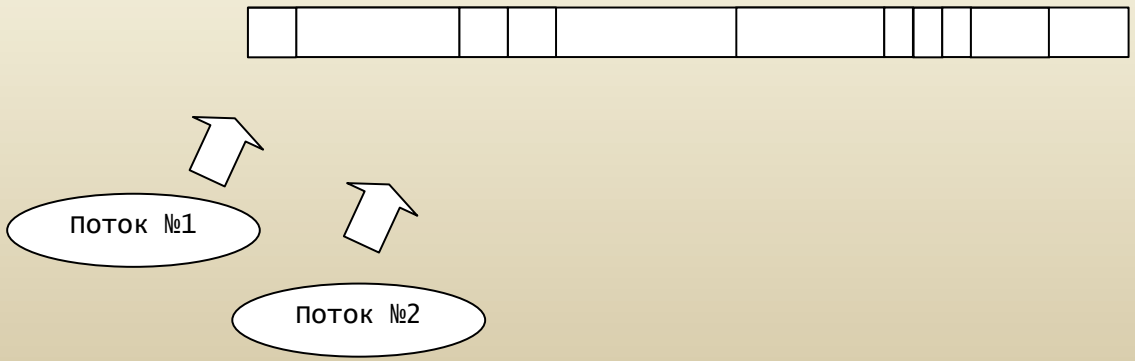


Декомпозиция по данным

- Основным достоинством статического разделения является независимость работы потоков (нет проблемы гонки данных) и, как следствие, нет необходимости в средствах синхронизации для доступа к элементам.
- Эффективность статической декомпозиции снижается при разной вычислительной сложности обработки элементов данных. Разделение по диапазону приводит к несбалансированности нагрузки разных потоков.
- В некоторых случаях декомпозиция может быть улучшена и при статическом разбиении, когда заранее известно, какие элементы обладают большей вычислительной трудоемкостью, а какие меньшей (например, круговая декомпозиция)

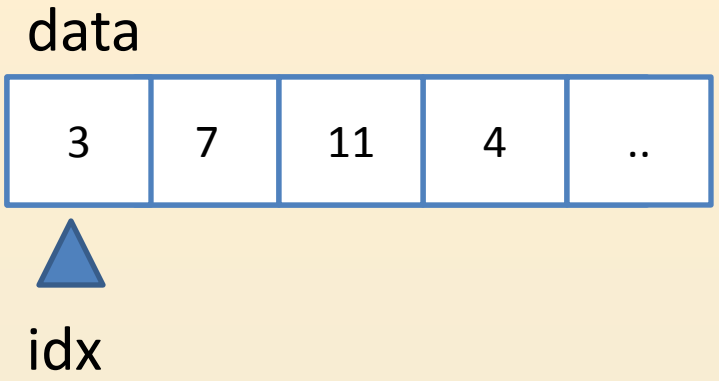
Динамическая декомпозиция

- В общем случае, когда вычислительная сложность обработки элементов заранее не известна, сбалансированность загрузки потоков обеспечивает динамическая декомпозиция.
- При динамической декомпозиции каждый поток, участвующий в обработке, обращается за блоком данных (порцией). После обработки блока данных поток обращается за следующей порцией.



```
// thread #1
..
myData = getData();
..
```

```
// thread #2
..
myData = getData();
..
```




```
int idx = 0;
int getData() {
    int v = data[idx];
    idx++;
    return v;
}
```

Динамическая декомпозиция

- Динамическая декомпозиция требует синхронизации доступа потоков к структуре данных.
- Синхронизация обладает накладными расходами
- Размер блока определяет частоту обращений потоков к структуре.
- Некоторые алгоритмы динамической декомпозиции увеличивают размер блока в процессе обработки. Если поток быстро обрабатывает элементы, то размер блока для него увеличивается.

Информационные зависимости

- Обнаружение и организация информационных зависимостей между задачами
- В лучшем случае между подзадачами нет информационных зависимостей (***embarrassingly parallel task)***
- В системах с общей памятью (многоядерные процессоры) информационные зависимости реализуются с помощью средств синхронизации
- В распределенных системах информационные зависимости реализуются с помощью средств передачи сообщений

Информационные зависимости

Виды коммуникаций между подзадачами:

- взаимодействие через общую память или с помощью передачи сообщений
- локальные/глобальные схемы передачи данных;
- синхронные/асинхронные способы взаимодействия;

Вопросы:

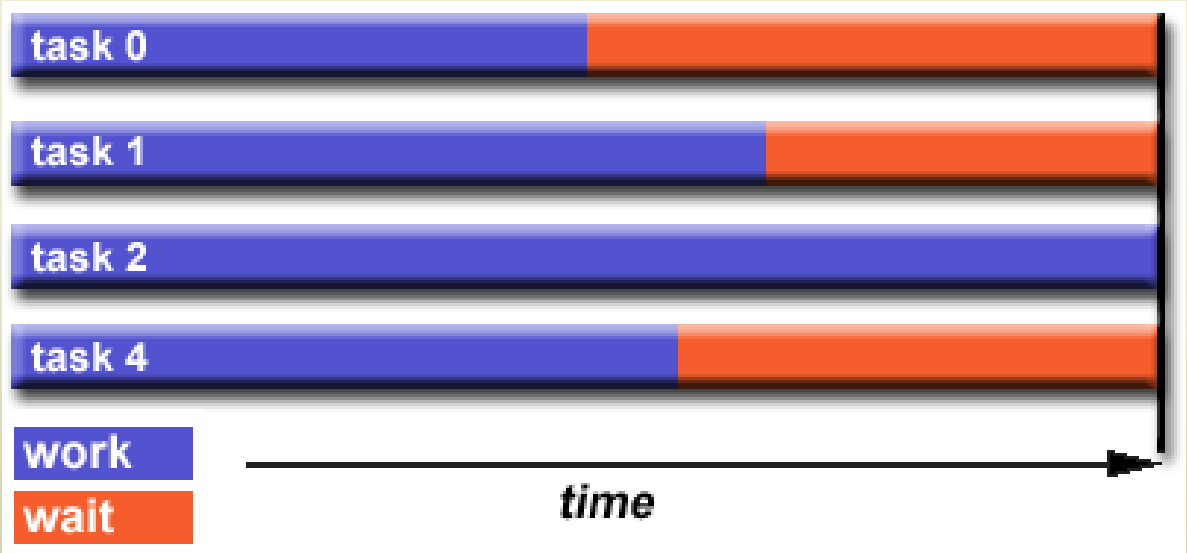
- соответствует ли вычислительная сложность подзадач интенсивности информационного взаимодействия?
- одинаковая ли интенсивность информационного взаимодействия для разных подзадач?

Масштабирование

- Свойство *масштабируемости* заключается в эффективном использовании всех имеющихся вычислительных ресурсов.
- Свойство масштабируемости приложения тесно связано с выбранным алгоритмом решения задачи.
- Обязательным условием масштабируемости приложения является возможность параметризации алгоритма в зависимости от числа процессоров в системе и в зависимости от текущей загруженности вычислительной системы.
- Современные платформы параллельного программирования предоставляют средства для автоматической балансировки нагрузки (пул потоков).

Балансировка нагрузки

- Максимальная эффективность параллельной обработки достигается при относительно равномерной загрузки вычислительных устройств (процессоров)
- *все* устройства должны работать *все* время (минимизация простоев)



Гранулярность

- Гранулярность определяется числом подзадач, на которые разбивается задача.
- Разбиение на большое число подзадач приводит к «мелко-зернистой» декомпозиции (fine-grained)
- Разбиение на мелкое число подзадач приводит к «крупно-зернистой» декомпозиции (coarse grained).

Катализаторы / ингибиторы параллелизма

Типовые программные структуры:

- Циклическая обработка ***for, foreach***

  data parallelism

- Вызовы функций/методов

  task parallelism

- Рекурсия

  task parallelism

Необходимое условие параллелизма:

независимость вычислений = результат не зависит от порядка вычислений