Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Высокопроизводительные вычисления

Эдуард Храмченков

High performance computing

- HPC – аппаратные (железо) и программные (софт) средства которые сокращают время решения вычислительной задачи
- «Время = деньги» – использование HPC это необходимость, вызванная научно-техническим прогрессом
- Потребность в HPC будет расти до смены технологической парадигмы (и дальше?)

HPC в науке

- Современные научные задачи требуют сложных математических моделей
- Расчет таких моделей занимает много времени
- HPC уменьшает «время до открытия»
- Моделирование климата, биология, молекулярная и квантовая химия, ядерная физика, космические исследования, механика и гидродинамика, etc

HPC в IT

- Поисковики и социальные сети – HPC в области Big Data
- Задачи распознавания образов в realtime
- Нейронные сети с использование HPC – AlphaGo (1202 CPU + 176 GPU)
- Компьютерная графика – визуализация, сложных систем, реалистичные 3d движки
- Компьютерная безопасность

HPC в промышленности

- HPC в области финансов – минимизация времени обработки данных для принятия решения
- Аэрокосмическая и автомобильная промышленность – специализированные пакеты моделирования используют HPC
- Нефтегазовая сфера – снижение себестоимости разведки и добычи с помощью HPC

Как измерить HPC

- Производительность – количество операций с float(double) в секунду FLOPs
- №1 в Top-500 СК Tianhe-2 ≈ 34 PFLOPs (10^{15})

$$k_{\text{ускорение}} = \frac{\text{Время выполнения последовательного кода}}{\text{Время выполнения параллельного кода}}$$

- Для хорошо параллелизуемой задачи ускорение (почти) линейно зависит от количества потоков

Как измерить HPC

$$k_{\text{эффективность}} = \frac{k_{\text{ускорение}}}{\text{Количество ядер}} \cdot 100\%$$

- Необходима оценка максимально возможного ускорения конкретного кода
- α – доля чисто последовательных вычислений, p – количество процессоров
- Закон Амдала

$$k_{\text{ускорение}} \leq \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

Как измерить HPC

- Закон Амдала не учитывает затраты на обмен, синхронизацию и управление потоками
- При увеличении числа потоков не учитывается увеличение количества данных
- Закон Густафсона

$$k_{\text{ускорение}} \leq p + (1 - p)\alpha$$

- В данном случае α должна определяться для каждого p

Как измерить HPC

- Для вычисления ускорения следует использовать самый оптимальный последовательный код
- Ускорение следует указывать в виде множителя, а не процентов
- Суперлинейное ускорение – коэффициент ускорения больше числа ядер, за счет попадания данных в локальный кэш

Железо

- Многоядерные CPU
- Многопроцессорные системы
- Массивно-параллельные системы (GPU, MIC)
- Суперкомпьютеры (кластеры) – связанные быстрой сетью вычислительные узлы
- Гетерогенные кластеры – узлы содержат в себе CPU+GPU

Софт

- Алгоритмы
- Языки программирования
- Фреймворки/API для работы с HPC-железом
- Библиотеки – специализированные наборы решений для стандартных задач HPC (сортировка, редукция, операции линейной алгебры, etc)

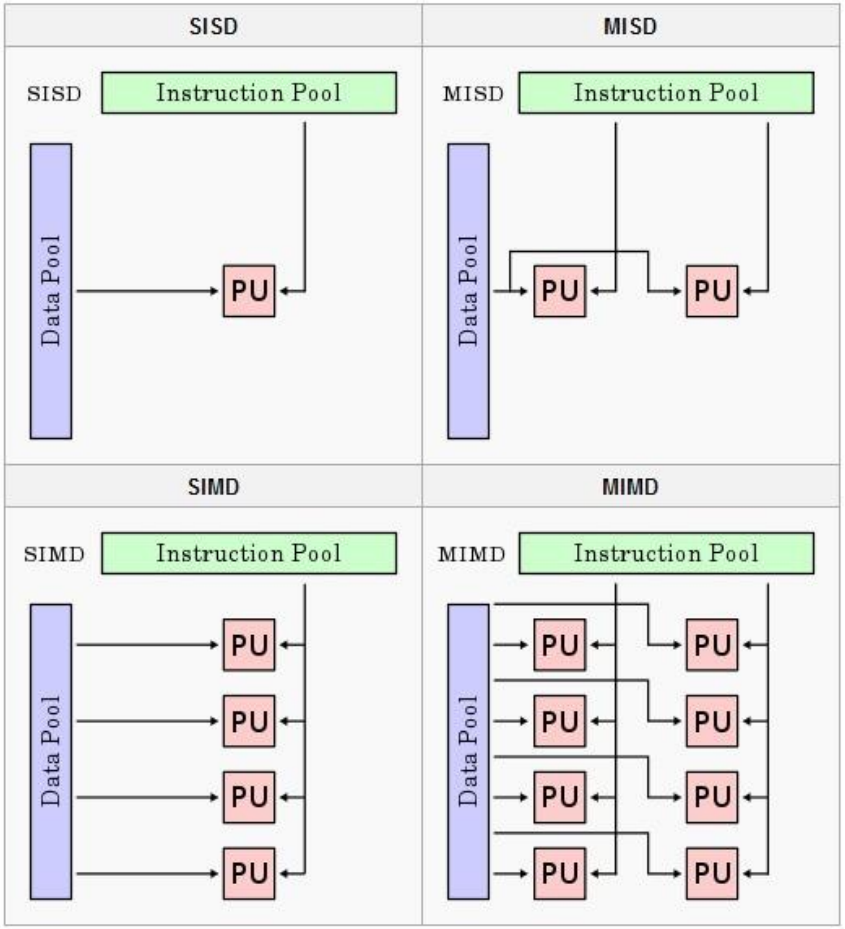
Параллельное программирование

- Метод – реализация параллельных алгоритмов на конкретных ЯП в рамках фреймворков под выбранную аппаратную архитектуру с использованием необходимых библиотек
- Цель – максимально эффективно задействовать доступные вычислительные ресурсы для минимизации времени выполнения программы

Таксономия Флинна

- Предложена в 1966 М. Флинном для классификации архитектур
- SISD – Single Instruction Single Data
- MISD – Multiple Instructions Single Data
- SIMD – Single Instruction Multiple Data
- MIMD – Multiple Instruction Multiple Data
- Все параллельные системы – либо SIMD и MIMD классов по Флинну

Таксономия Флинна



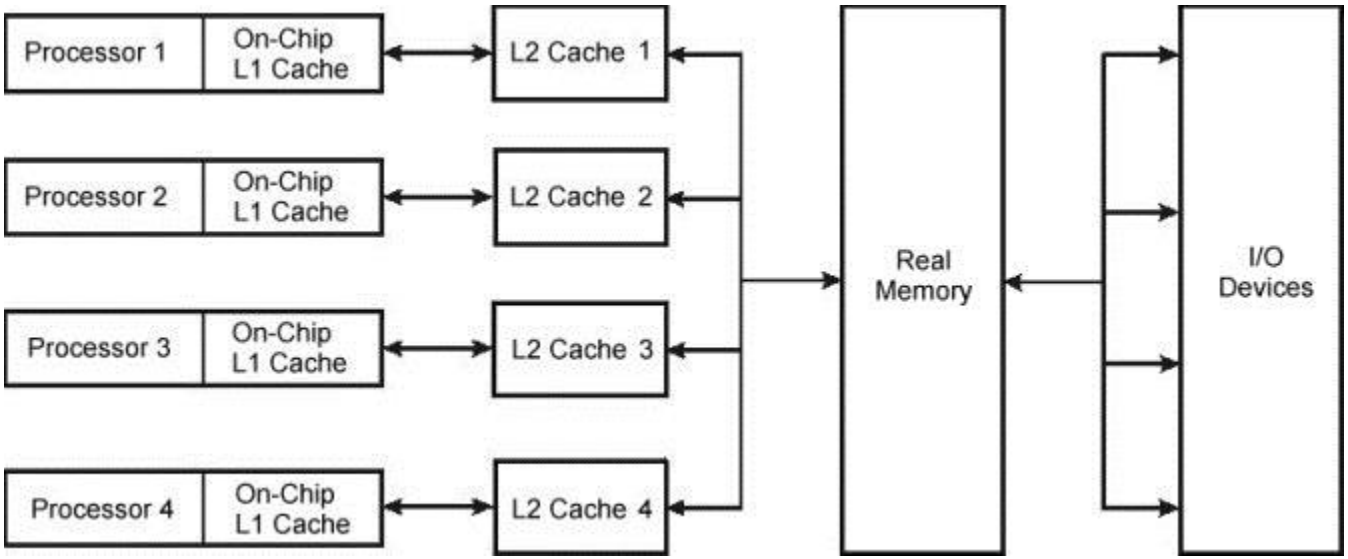
Параллельные архитектуры

- На основе разделяемой памяти (Symmetrical Multiprocessor, SMP) – многоядерные и многопроцессорные системы (SIMD)
- На основе передачи сообщений – распределенные многопроцессорные и кластерные системы (MIMD)
- Гибридная архитектура (MIMD)

Архитектура SMP

- На каждом ядре/процессоре системы выполняется один поток
- Общение потоков происходит через разделяемую память по общей шине
- Производительность и масштабируемость ограничены шиной
- Легкость и удобство в программировании, доступ к памяти одинаков для всех потоков

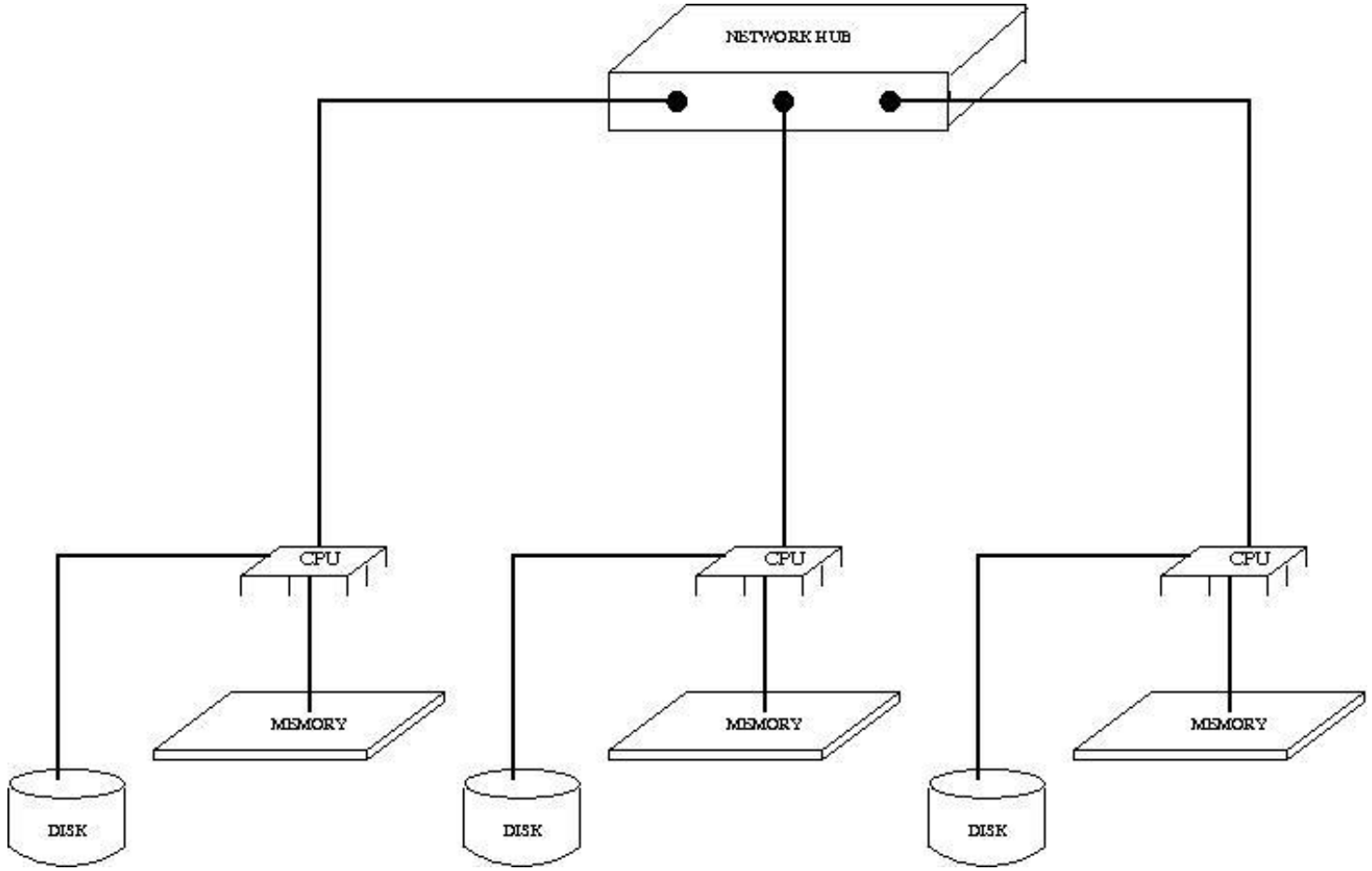
Архитектура SMP



Распределенная архитектура

- Каждый поток получает в свое пользование отдельный узел
- У каждого потока свой «экземпляр» памяти программы
- Узлы обмениваются данными через какую-либо соединяющую сеть при помощи специальных сообщений
- Хорошая масштабируемость системы

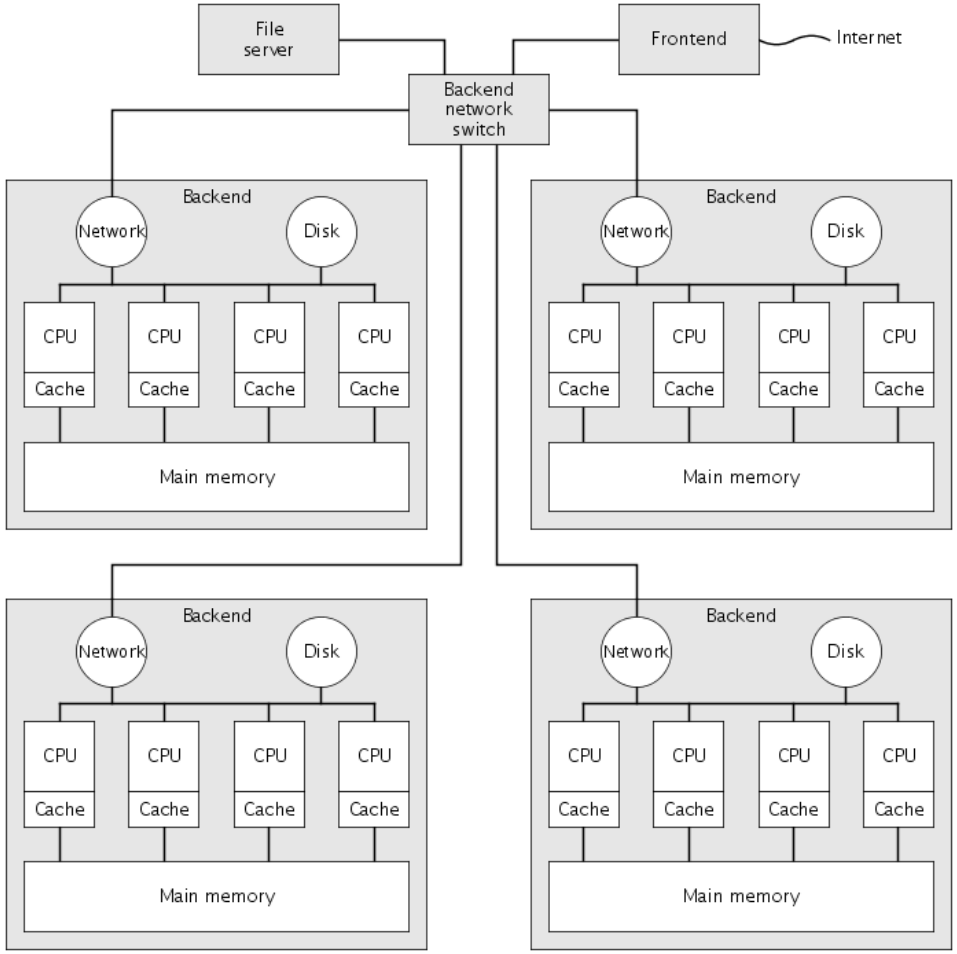
Распределенная архитектура



Гибридная архитектура

- Сочетание SMP и распределенной архитектуры
- Распределенная система, узлы которой используют SMP архитектуру
- GPGPU – вычисления на видеокартах
- GPU – отдельный многоядерный сопроцессор со своей памятью
- Гетерогенная архитектура – в узлах кластера многоядерные CPU+GPU

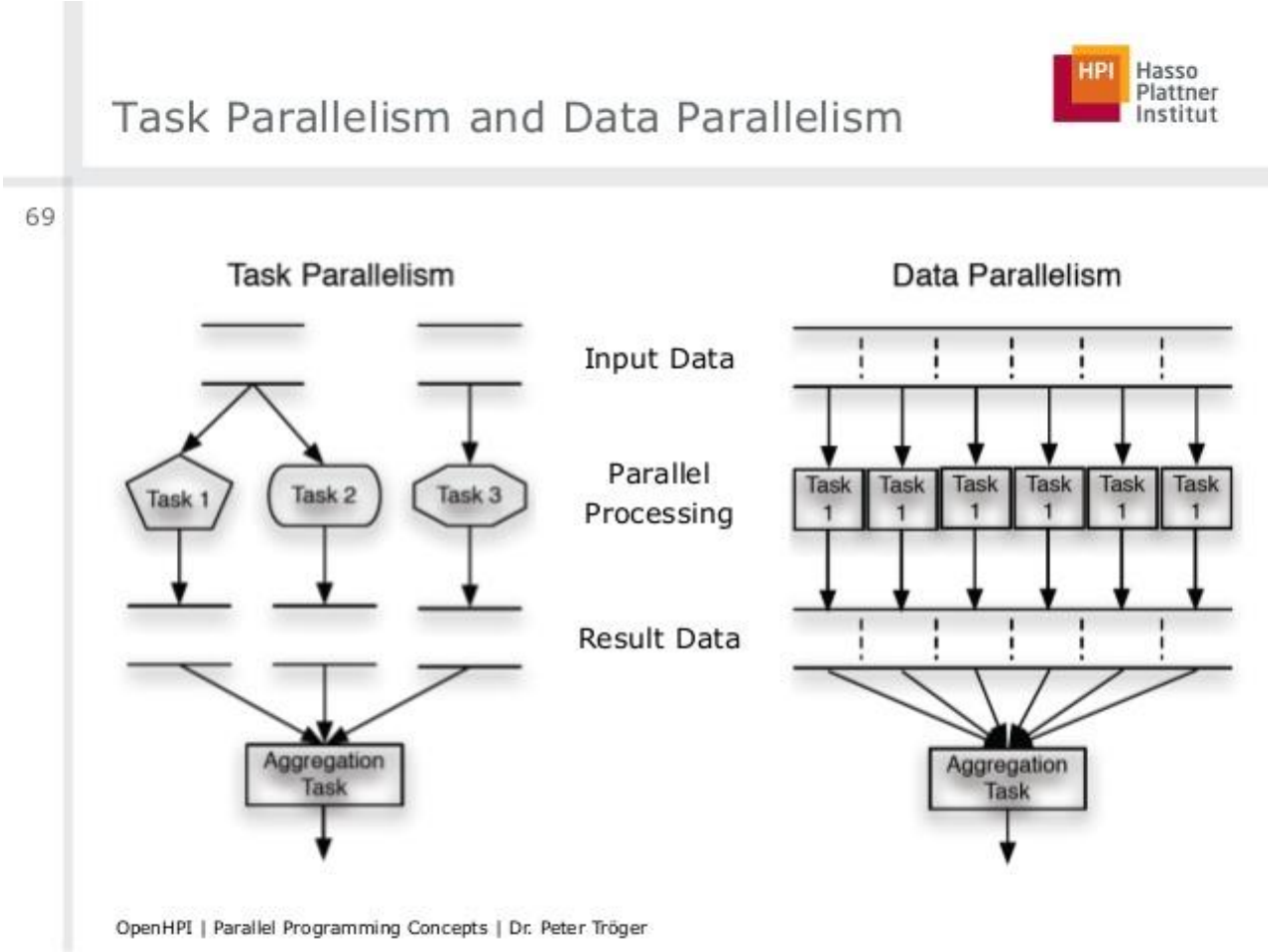
Гибридная архитектура



Параллельность задач

- Разные задачи по разному параллелятся на разных архитектурах
- Параллелизм по данным – множество данных над которыми потоки выполняют одну и ту же инструкцию/функцию
- Функциональный параллелизм – разные потоки выполняют разные функции с разными данными

Параллельность задач



Казанский федеральный университет

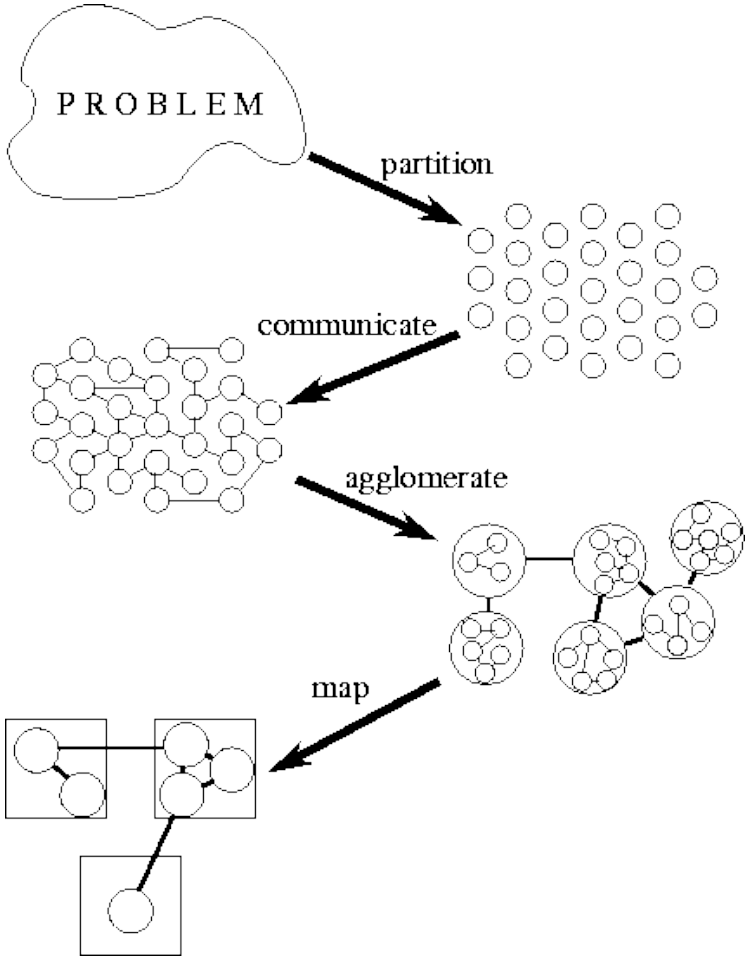
Параллельные алгоритмы

- Тривиальный последовательный алгоритм может иметь сложную параллельную версию
- Основные проблемы: коммуникация между потоками, обращение к общим участкам памяти и «race condition»
- Состояние гонки – работа системы зависит от порядка выполнения операций

Метод Фостера

- Универсальный машиннонезависимый метод разработки параллельных алгоритмов
- 1-й этап: Разделение (Partitioning)
- 2-й этап: Взаимодействие (Communication)
- 3-й этап: Агрегация (Agglomeration)
- 4-й этап: Отображение (Mapping)

Метод Фостера



Partitioning

- Разделение вычислений и данных на отдельные составляющие
- Декомпозиция области – разделение данных на блоки, потом определение как вычисления соотносятся с блоками
- Функциональная декомпозиция – разбиение вычислительных операций на группы, удобные для конвейерной обработки (pipelinening)

Partitioning

- Каждый из элементов разбиения – элементарная задача (primitive task, PT)
- Количество PT должно быть как минимум на порядок больше количества процессоров
- Излишние вычисления и обращения к памяти минимизированы
- PT примерно одного размера
- Количество PT – функция размера задачи

Communication

- Локальная коммуникация – когда РТ для работы нужны данные от небольшого числа других РТ
- Глобальная коммуникация – когда значительному количеству РТ необходимо внести вклад в вычисление
- Коммуникации между РТ – издержки (overhead) параллельного алгоритма, необходима их минимизация

Communication

- Коммуникации должны равномерно распределяться по РТ
- Каждое задание должно общаться с небольшим количеством соседей
- РТ должны иметь возможность общаться одновременно
- РТ должны иметь возможность проводить вычисления одновременно

Agglomeration

- Цель – снизить издержки алгоритма
- Группирование РТ в более крупные блоки, количество блоков зависит от задачи и архитектуры
- Агрегация позволяет увеличить локальность – уменьшение коммуникаций
- Корректная агрегация позволяет построить масштабируемый алгоритм

Agglomeration

- Агрегация должна повышать локальность
- Агрегированные коммуникации должны занимать меньше времени чем исходные
- Алгоритм должен быть масштабируемым
- Агрегированные блоки должны иметь схожий размер и коммуникации
- Количество блоков должно быть возрастающей функцией от размера задачи

Agglomeration

- Количество блоков должно быть с одной стороны минимально возможным, с другой должно быть по меньшей мере равным количеству процессоров на целевой архитектуре
- Затраты на рефакторинг кода в соответствии с выбранной стратегией агрегации должны быть разумными

Mapping

- Процесс назначения процессорам блоков заданий
- Отображение должно максимизировать загрузку процессоров
- Отображение должно минимизировать количество коммуникаций
- Проблема поиска оптимального отображения относится к NP-сложным

Mapping


Количество заданий
Статическое
Динамическое
Структурированные коммуникации
Неструктурированные коммуникации
Частые коммуникации между заданиями
Много небольших заданий, мало коммуникаций
Одинаковое вычислительное время на задание
Разные вычислительные времена на задание
Статический механизм распределения нагрузки
Динамический алгоритм балансировки загрузки
Алгоритм распределения заданий на этапе выполнения
Одно задание на процессор, минимизация коммуникаций
Задания циклически раскидываются по процессорам для балансировки загрузки

Mapping

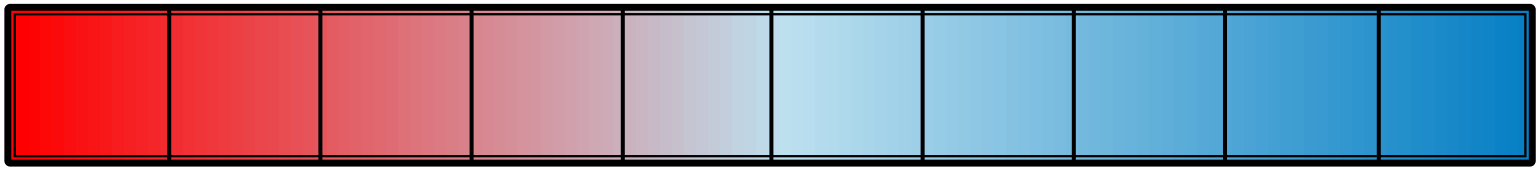
- Следует изучить возможность как статического так и динамического распределения заданий по процессорам
- В случае использования динамического распределения менеджер заданий не должен быть узким местом
- В случае статического распределения соотношение заданий к числу процессоров не менее 10:1

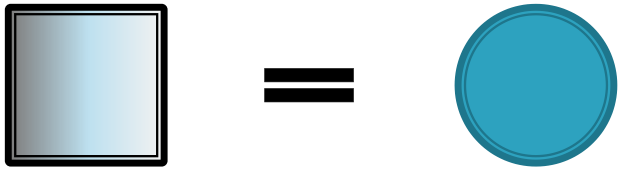
Пример

- Одномерная задача о нагреве стержня
- Стержень разбит на *n* элементов
- Температура вычисляется на протяжении *m* шагов по времени
- Формула для расчета температуры *i*-ого элемента на *j+1* временном шаге

$$u_{i,j+1} = ru_{i-1,j} + (1-2r)u_{i,j} + ru_{i+1,j}$$

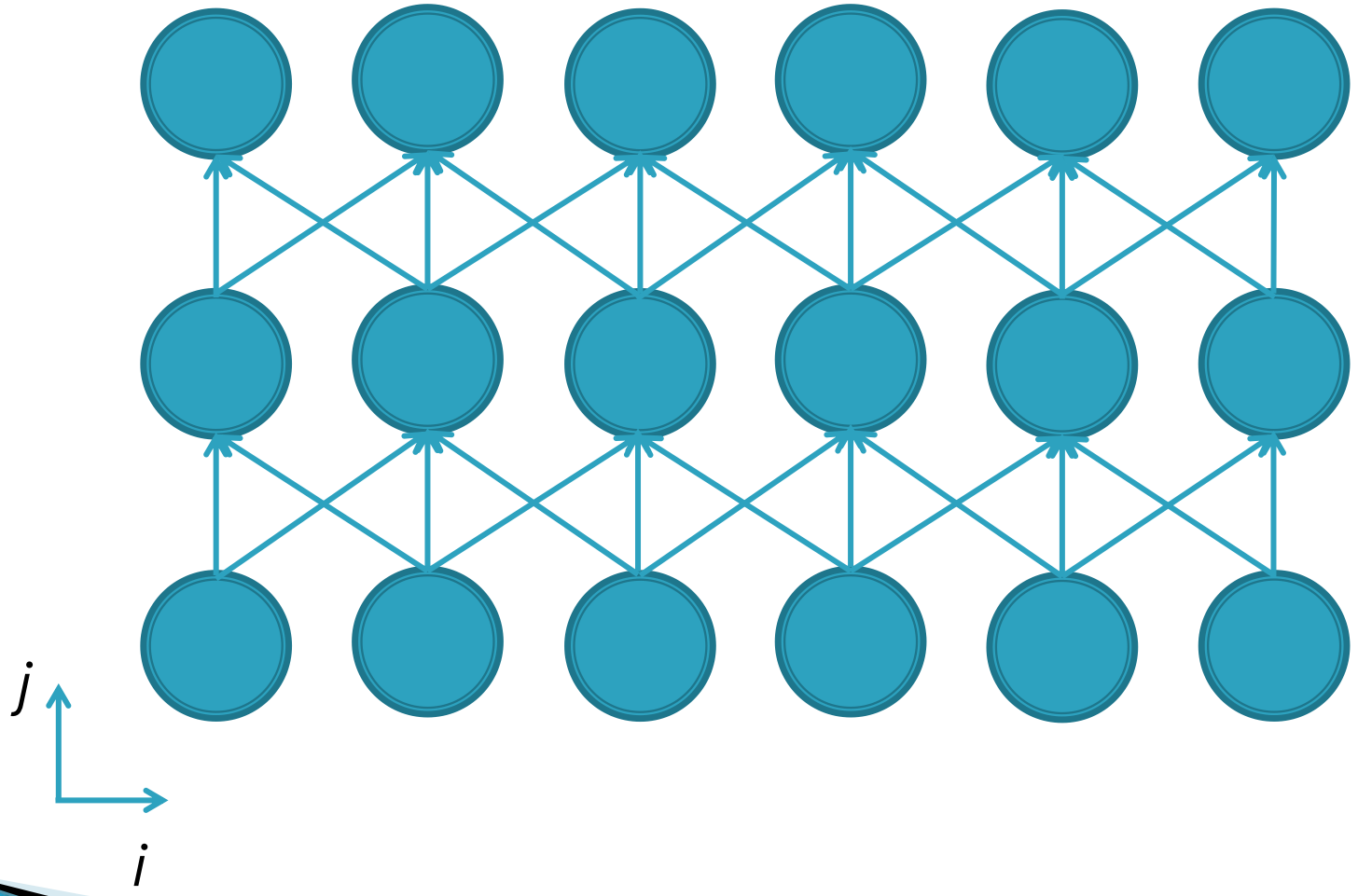
Пример



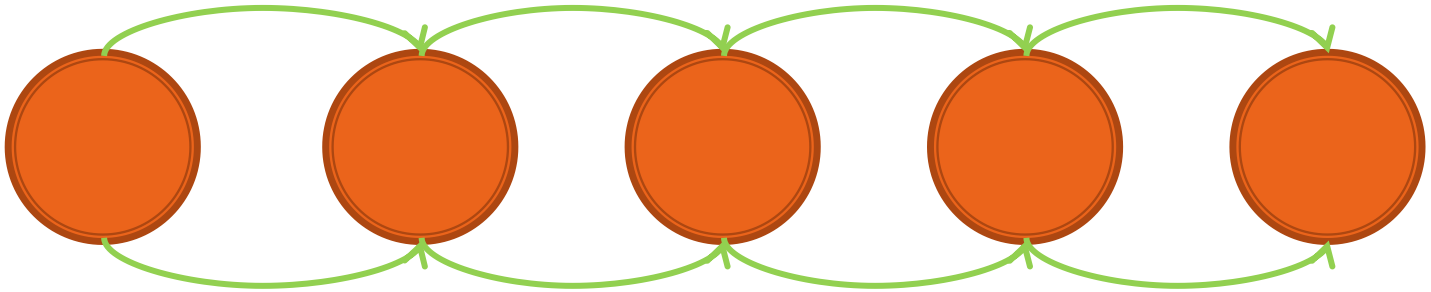


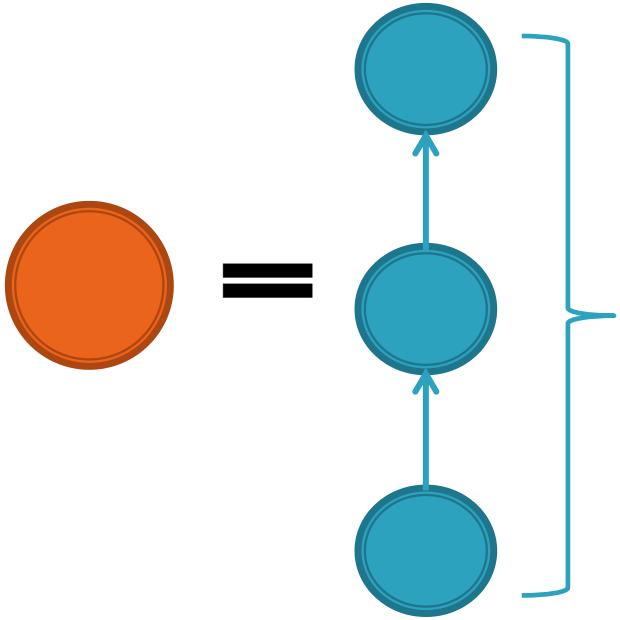
Разбиение по данным
Элемент стержня = РТ

Пример

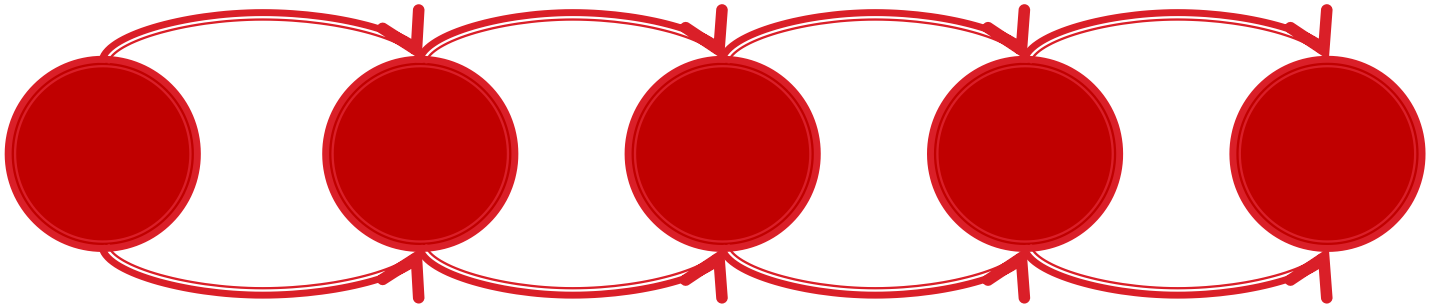


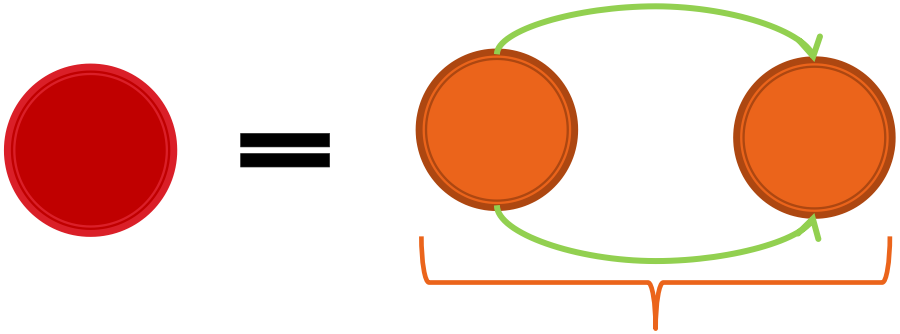
Пример





Пример





?

Зависит от числа процессоров

Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Вопросы

ekhramch@kpfu.ru