Лекция 5
Основы параллельного программирования (Parallel programming introduction)

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

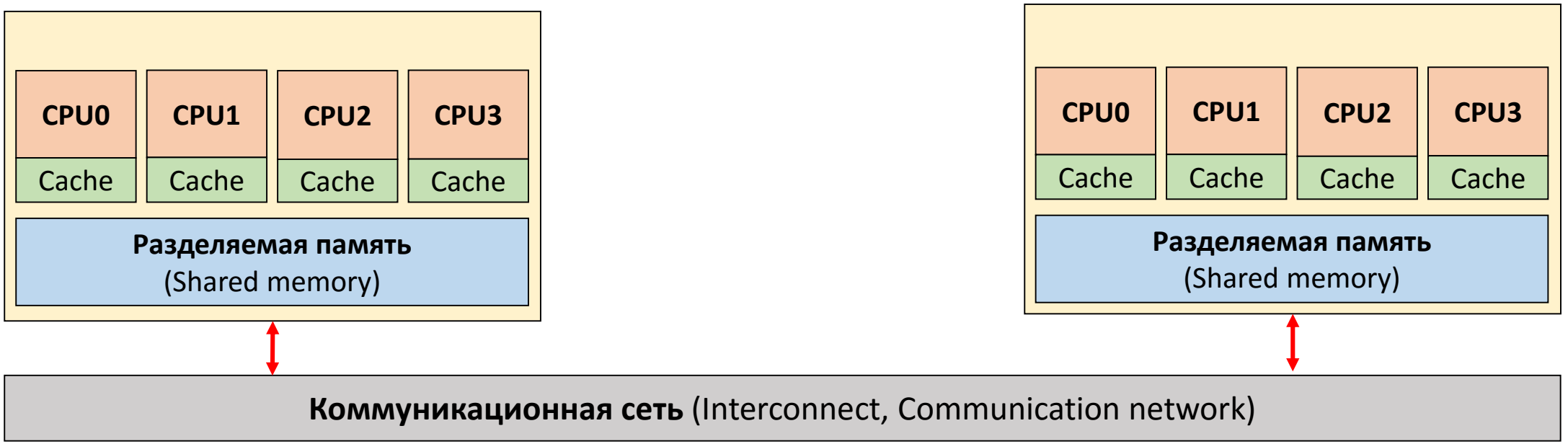
Курс «Высокопроизводительные вычислительные системы»
Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)
Осенний семестр, 2015

Параллельные вычисления (Parallel Computing)

- **Цели использования многопроцессорных вычислительных систем (ВС):**
  - решение задачи за меньшее время на нескольких процессорах
  - решение задачи с бо́льшим объёмом входных данных – использование распределенной памяти нескольких вычислительных узлов
  - решение задачи с большей вероятностью получения корректного решения (дублирование вычислений – параллельный пересчет)

Архитектура вычислительных систем с распределенной памятью

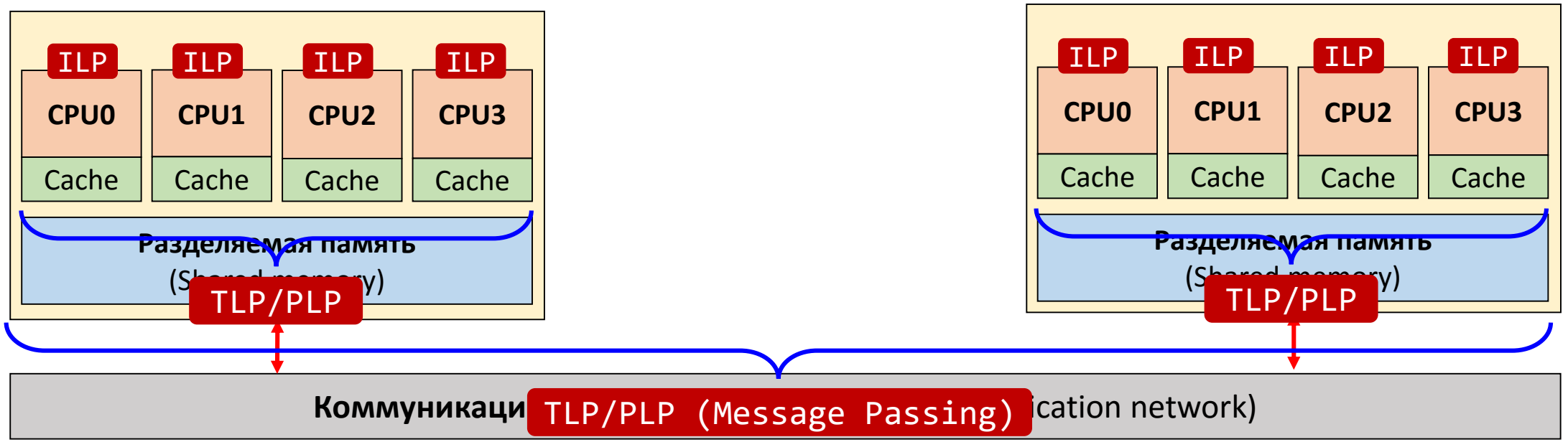
Вычислительные системы с распределенной памятью



- **Вычислительная система с распределенной памятью** (Distributed memory computer system) – совокупность вычислительных узлов, взаимодействие между которыми осуществляется через коммуникационную сеть (InfiniBand, Gigabit Ethernet, Cray Gemeni, Fujitsu Tofu, …)
- Каждый узел имеет множество процессоров/ядер, взаимодействующих через разделяемую память (Shared memory)
- Процессоры могут быть многоядерными,
  ядра могут поддерживать одновременную многопоточность (SMT)

- http://www.top500.org
- http://www.green500.org
- http://www.graph500.org
- http://top50.supercomputers.ru

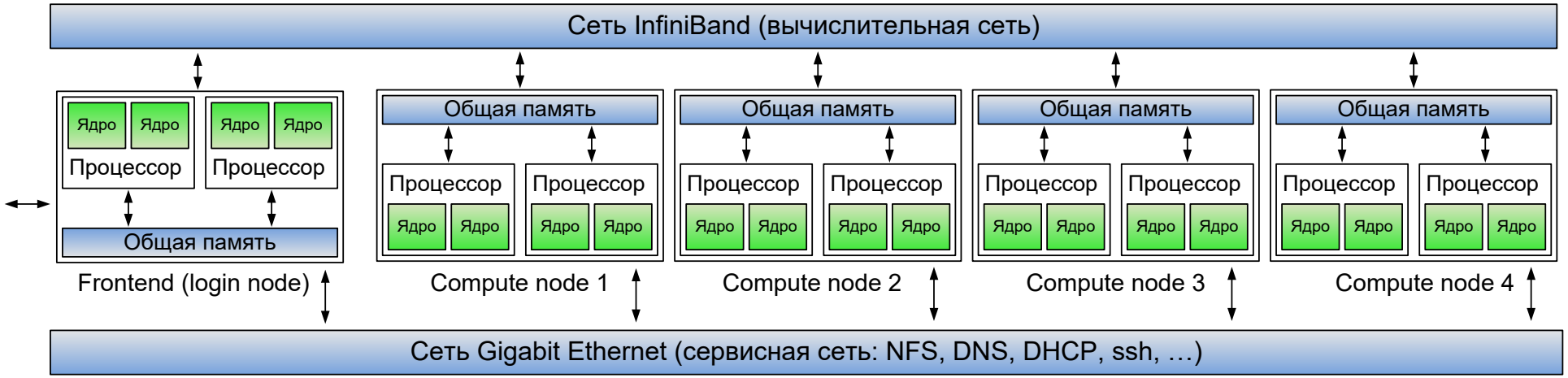
Вычислительные системы с распределенной памятью



- **Вычислительная система с распределенной памятью** (Distributed memory computer system) – совокупность вычислительных узлов, взаимодействие между которыми осуществляется через коммуникационную сеть (InfiniBand, Gigabit Ethernet, Cray Gemeni, Fujitsu Tofu, …)
- Каждый узел имеет множество процессоров/ядер, взаимодействующих через разделяемую память (Shared memory)
- Процессоры могут быть многоядерными,
  ядра могут поддерживать одновременную многопоточность (SMT)

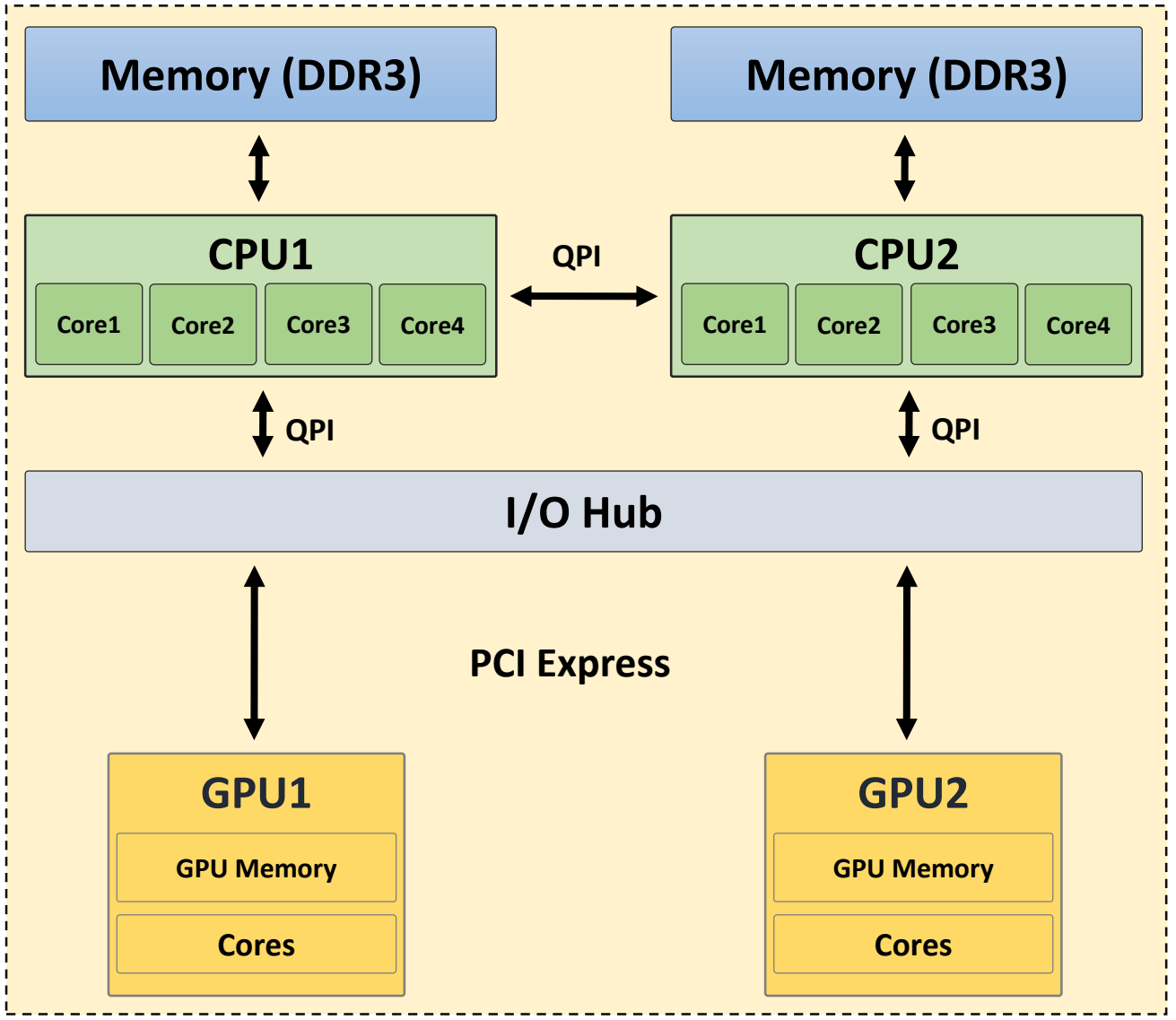
- http://www.top500.org
- http://www.green500.org
- http://www.graph500.org
- http://top50.supercomputers.ru

Вычислительные кластеры (Computer cluster)



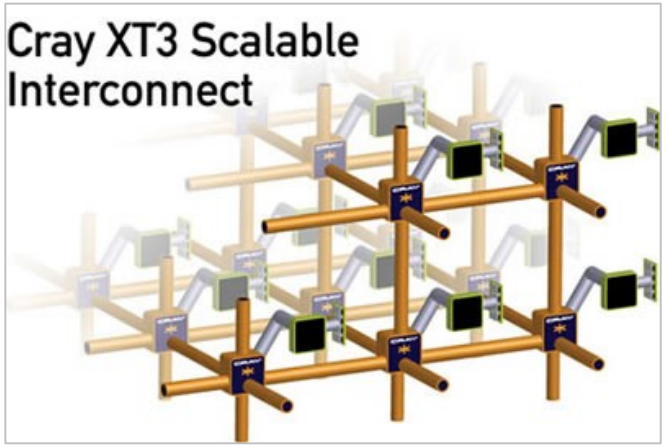
- **Вычислительные кластеры строятся на базе свободно доступных компонентов**
- Вычислительные узлы: 2/4-процессорные узлы, 1 – 8 GiB оперативной памяти на ядро (поток)
- Коммуникационная сеть (сервисная и для обмена сообщениями)
- Подсистема хранения данных (дисковый массивы, параллельные и сетевые файловые системы)
- Система бесперебойного электропитания
- Система охлаждения
- Программное обеспечение: GNU/Linux (NFS, NIS, DNS, …), MPI (MPICH2, Open MPI), TORQUE/SLURM

Гибридные вычислительные узлы



Коммуникационные сети (Interconnect)

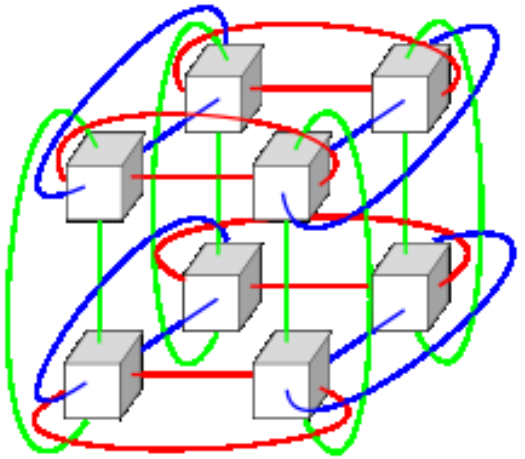
- **С фиксированной структурой (статической топологией) – графом связей вычислительных узлов**
- Каждый вычислительный узел имеет сетевой интерфейс (маршрутизатор) с несколькими портами, через который он напрямую соединён с другими узлами



- **С динамической структурой – на базе коммутаторов**
- Каждый вычислительный узел имеет сетевой интерфейс с несколькими портами
- Порты интерфейсов подключены к коммутаторам, через которые происходит взаимодействие узлов

Сети с фиксированной структурой

- **Структура сети (топология)** – это граф
  - узлы – машины (вычислительные узлы, computer nodes)
  - ребра – линии связи (links)
- **Показатели эффективности структур:**
  - диаметр (максимальное из кратчайших расстояний)
  - средний диаметр сети
  - бисекционная пропускная способность (bisectional bandwidth)
- **Примеры структур**
  - ***kD*-тор** (3D, 4D, 5D): Cray Titan (#2) 3D torus, IBM Sequoia 5D torus (#3), Fujitsu 6D torus (#4)
  - **Гиперкуб**
  - **Решетка, кольцо, графы Кауца, циркулянтные структуры, …**

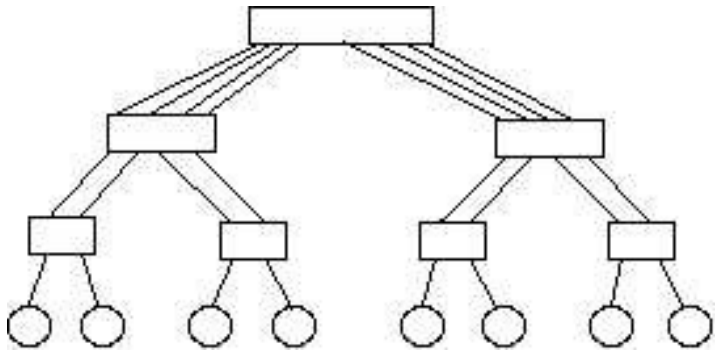


3D-тор

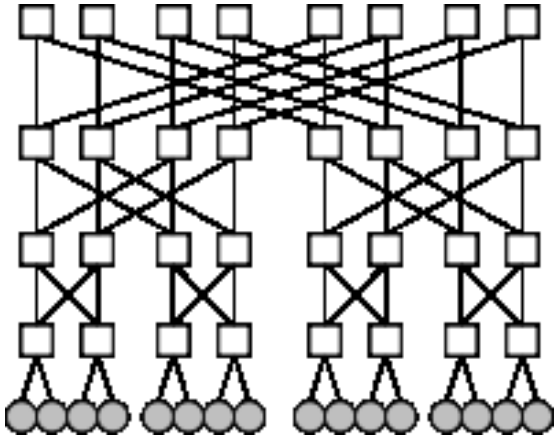
Сети с динамической структурой

- **Вычислительные узлы связаны через активные устройства – сетевые коммутаторы (network switches)**
- Коммутатор имеет фиксированное число портов (24, 48, 96, 324)
- Как объединить в сеть тысячи узлов?
- Топология “толстого дерева” (fat tree)
  Charles E. Leiserson. **Fat-trees: universal networks for hardware-efficient supercomputing** //
  IEEE Transactions on Computers, Vol. 34, No. 10, 1985

- **Сетевые технологии**
  - ❑ Gigabit Ethernet
  - ❑ 10 Gigabit Ethernet
  - ❑ InfiniBand DDR/QDR/FDR



Fat tree



Показатели эффективности коммуникационных сетей

- **Пропускная способность (Bandwidth)** – бит/сек. (10 – 54 Gbps – Giga bit per second)
- **Латентность (Latency)** – сек. (10 us – 100 ns)
- **Бисекционная пропускная способность (Bisectional bandwidth)**

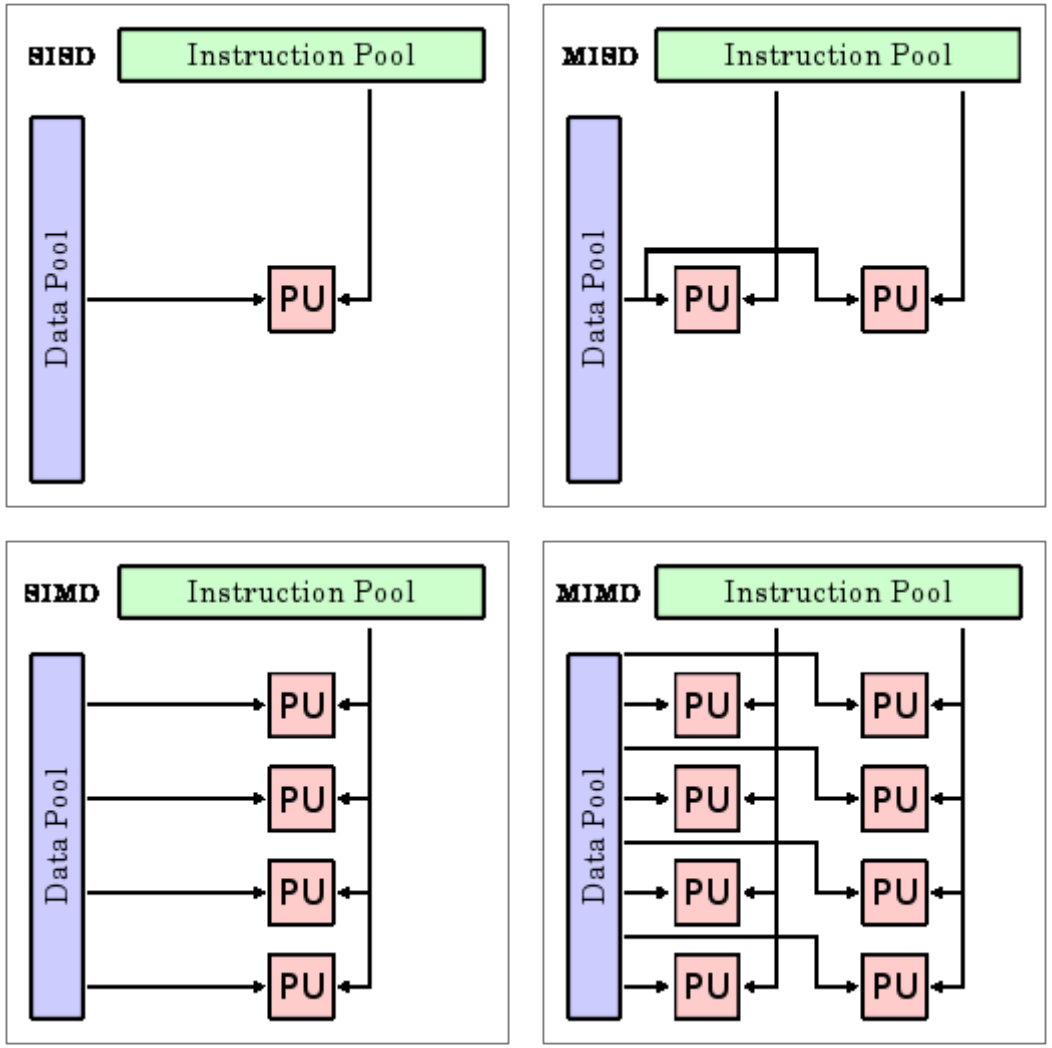
Классификация архитектур ВС

- **Классификация Флинна (M. J. Flynn, 1966)**

Flynn's taxonomy		
	Single instruction	**Multiple instruction**
Single data	SISD	MISD
Multiple data	SIMD	MIMD

- **По организации памяти:** с общей памятью (SMP/NUMA), с распределенной памятью
- **По типу поддерживаемых операндов:** скалярные и векторные системы
- **По доступности компонентов**, из которых строится система: проприетарные (заказные), на основе общедоступного аппаратного обеспечения (commodity computing, проект Beowulf)
- …

Классификация Флинна



- **Современные ВС нельзя однозначно классифицировать по Флинну** – современные системы мультиархитектурны (SIMD + MIMD)
- **Процессор** – скалярные + векторные инструкции (SIMD**)**
- **Вычислительный узел** – SMP/NUMA-система на базе многоядерных процессоров (MIMD) + графические процессоры (SIMD)
- **Совокупность нескольких вычислительных узлов** – MIMD

Рейтинги мощнейших ВС

1. www.top500.org – решение системы линейных алгебраических уравнений методом LU-факторизации (High-Performance Linpack, FLOPS – Floating-point Operations Per Seconds)

2. www.graph500.org – алгоритмы на графах (построение графа, обход в ширину, TEPS – Traversed Edges Per Second)

3. www.green500.org – главный критерий – энергоэффективност (объем потребляемой эленктроэнергии, kW)

4. http://top50.supercomputers.ru – рейтинг мощнейших вычислительных систем СНГ (тест High-Performance Linpack)

5. **Как создать свой тест производительности?**

www.top500.org (Июнь 2015)

	NAME	SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	Tianhe-2 (Milkyway-2)	Intel Ivy Bridge (12C 2.2 GHz) & Xeon Phi (57C 1.1 GHz), Custom interconnect	NUDT	China	3,120,000	33.9	17.8
2	Titan	Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect	DOE/SC/ORNL	USA	560,640	17.6	8.2
3	Sequoia	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	17.2	7.9
4	K computer	Fujitsu SPARC64 VIIIfx (8C 2.0 GHz), Custom interconnect	RIKEN AICS	Japan	705,024	10.5	12.7
5	Mira	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/SC/ANL	USA	786,432	8.59	3.95

- Среднее количество вычислительных ядер в системе: **50 464**
- Среднее количество ядер на сокет (процессор): **10** (4, 6, **8, 10, 12, 14, 16**)
- Среднее энергопотребление: **866** kW
- Коммуникационная сеть: Infiniband (52%), 10G (16%), Custom (14%), Gigabit Ethernet (13%), Cray (3%), …
- Процессоры: Intel (~ 86%), IBM Power, AMD Opteron, SPARC64, ShenWei, NEC
- Ускорители (~ 17% систем): NVIDIA Kepler (6.6%), Intel Xeon Phi (6.2%), NVIDIA Fermi (3%), ATI Radeon (0.8%), PEZY-SC (0.6%)
- Операционная система: GNU/Linux & Unix (~97%), IBM AIX (9 шт.), Cray Microsoft (1 шт.)

www.top500.org (Июнь 2015)

	NAME	SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	Tianhe-2 (Milkyway-2)	Intel Ivy Bridge (12C 2.2 GHz) & Xeon Phi (57C 1.1 GHz), Custom interconnect	NUDT	China	3,120,000	33.9	17.8
2	Titan	Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect	DOE/SC/ORNL	USA	560,640	17.6	8.2
3	Sequoia	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864		7.9
4	K computer	Fujitsu SPARC64 VIIIfx (8C 2.0 GHz), Custom interconnect	RIKEN AICS	Japan	705,024	.5	12.7
5	Mira	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/SC/ANL	USA		8.59	3.95

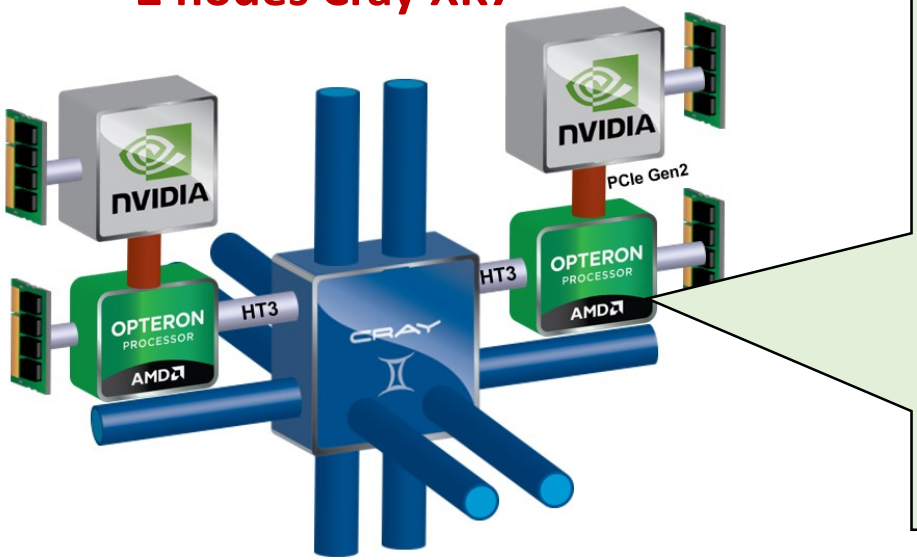
- Среднее количество вычислительных ядер в системе: **50 464**
- Среднее количество ядер на сокет (процессор): **10** (4, 6, **8, 10,**
- Среднее энергопотребление: **866** kW
- Коммуникационная сеть: Infiniband (52%), 10G (16%), Custom (14%), Gigabit Ethernet (13%), Cray (3%), …
- Процессоры: Intel (~ 86%), IBM Power, AMD Opteron, SPARC64, ShenWei, NEC
- Ускорители (~ 17% систем): NVIDIA Kepler (6.6%), Intel Xeon Phi (6.2%), NVIDIA Fermi (3%), ATI Radeon (0.8%), PEZY-SC (0.6%)
- Операционная система: GNU/Linux & Unix (~97%), IBM AIX (9 шт.), Cray Microsoft (1 шт.)

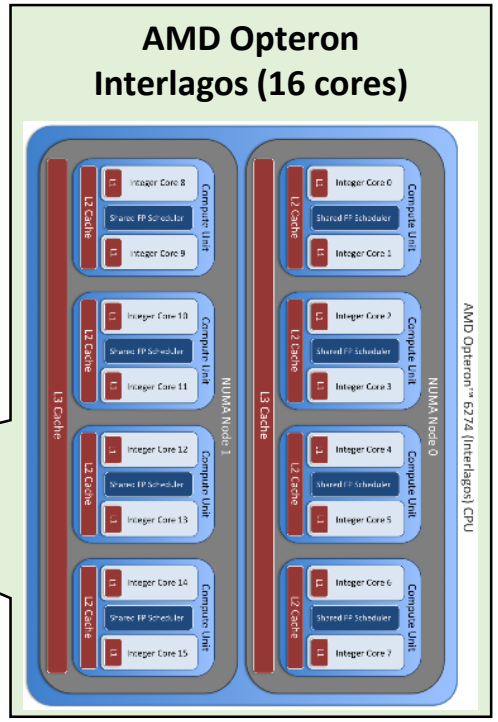
- **Новосибирская ГЭС – 460 МВт**
- **Новосибирская ТЭЦ-5 – 1200 МВт**

Мультиархитектура современных ВС

- **Разработка эффективных параллельных программ для мультиархитектурных ВС требует владения стеком технологий**
  - **Internode communications:** MPI, Cray Chapel, IBM X10, Shmem, Unified Parallel C, Coarray Fortran, Global Arrays
  - **Multithreading:** OpenMP, Intel TBB/Cilk Plus, C11/C++11 Threads
  - **GPU:** NVIDA CUDA, OpenCL, OpenACC, OpenMP 4.0
  - **Vectorization (SIMD):** SSE/AVX, AltiVec

2 nodes Cray XK7

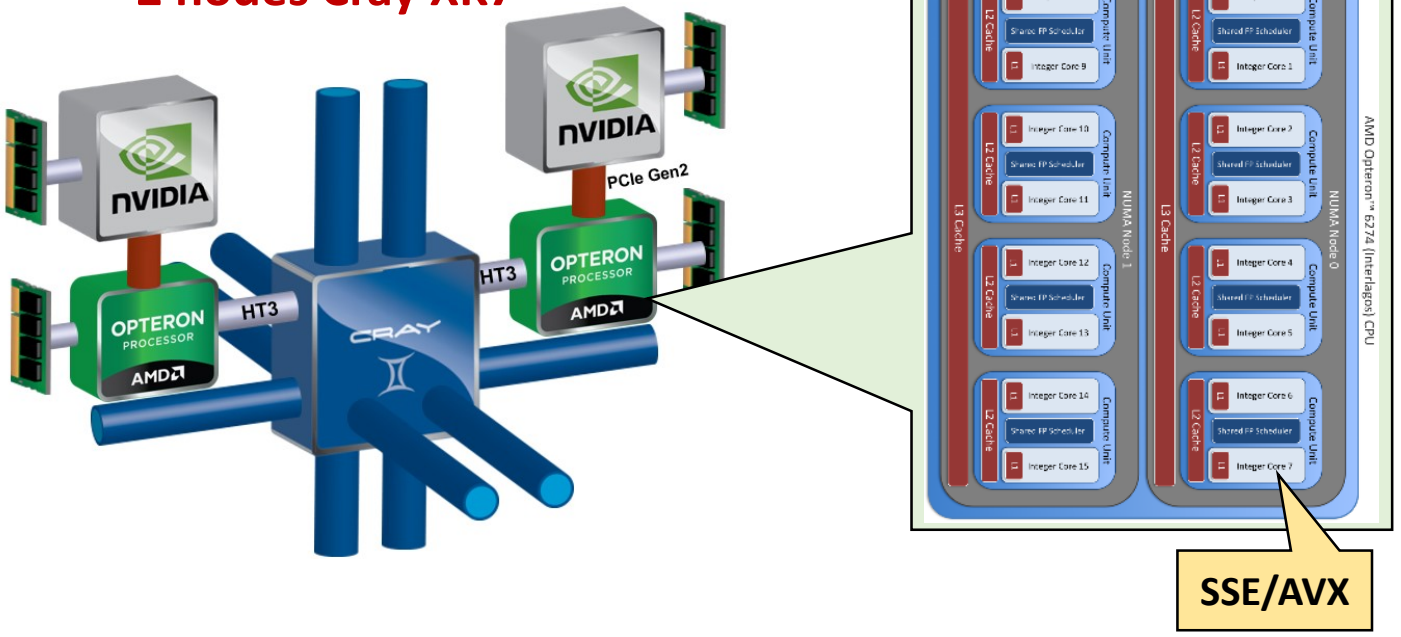




Мультиархитектура современных ВС

- **Разработка эффективных параллельных программ для мультиархитектурных ВС требует владения стеком технологий**

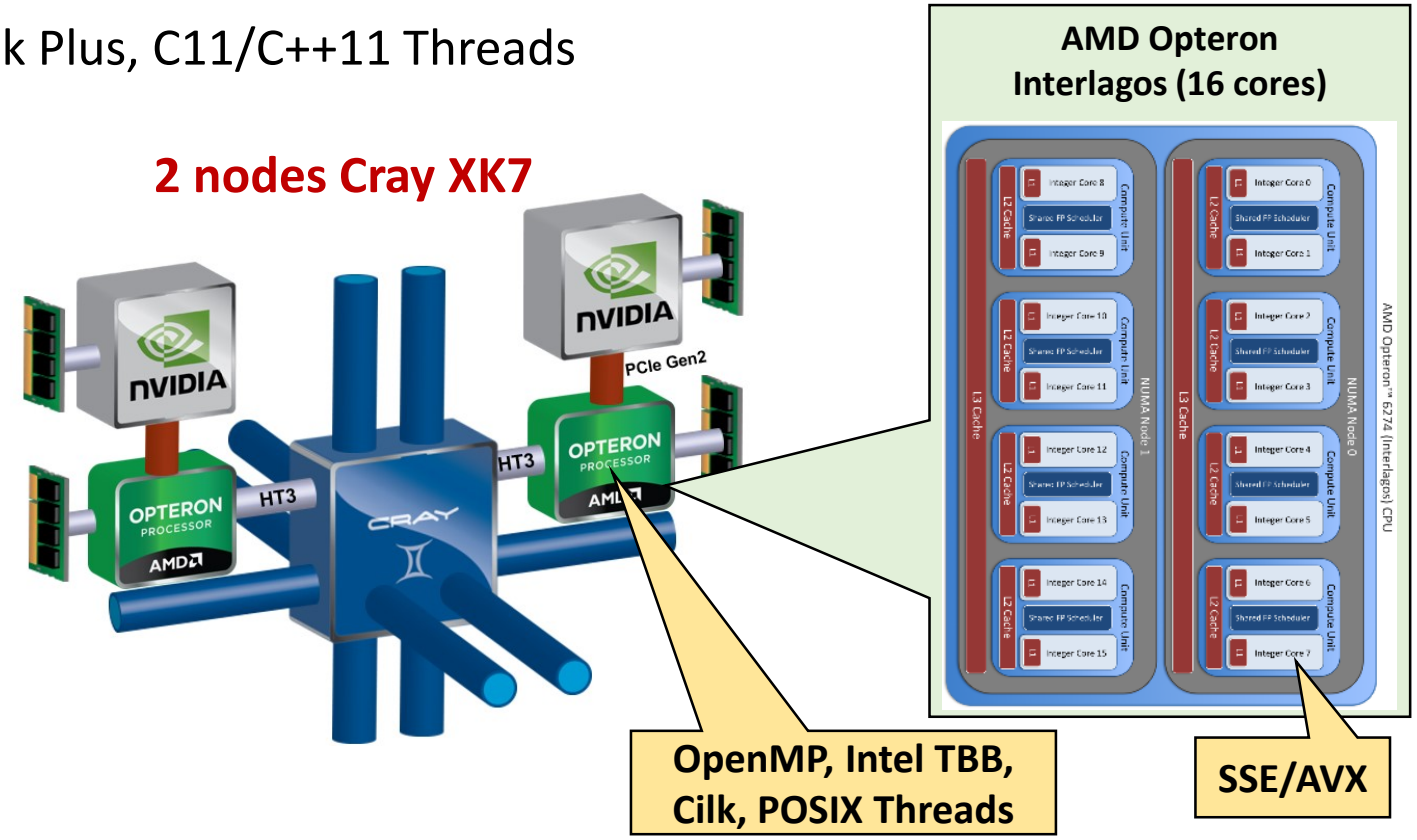
  - **Internode communications:** MPI, Cray Chapel, IBM X10, Shmem, Unified Parallel C, Coarray Fortran, Global Arrays
  - **Multithreading:** OpenMP, Intel TBB/Cilk Plus, C11/C++11 Threads
  - **GPU:** NVIDA CUDA, OpenCL, OpenACC, OpenMP 4.0
  - **Vectorization (SIMD):** SSE/AVX, AltiVec



Мультиархитектура современных ВС

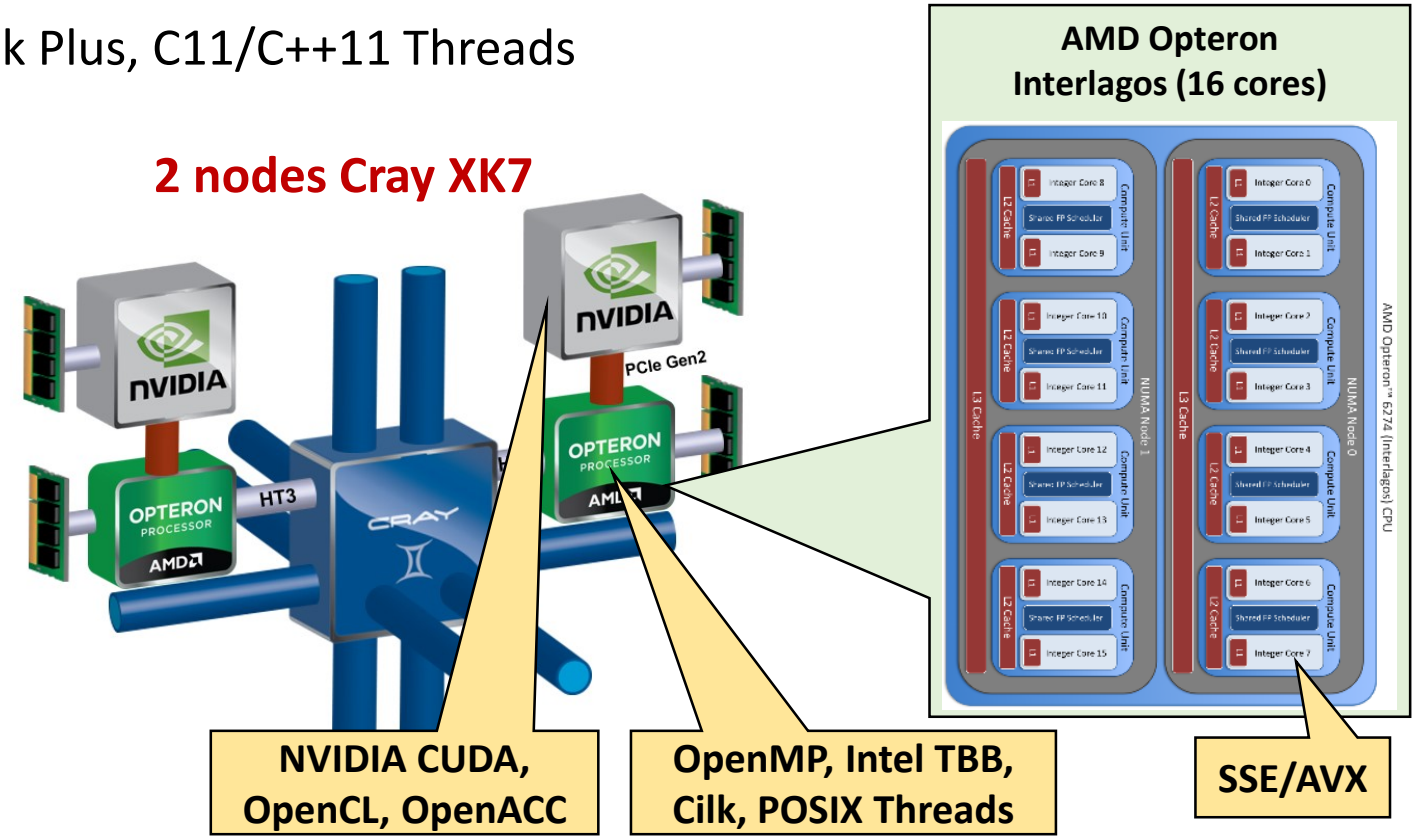
- **Разработка эффективных параллельных программ для мультиархитектурных ВС требует владения стеком технологий**

  - **Internode communications:** MPI, Cray Chapel, IBM X10, Shmem, Unified Parallel C, Coarray Fortran, Global Arrays
  - **Multithreading:** OpenMP, Intel TBB/Cilk Plus, C11/C++11 Threads
  - **GPU:** NVIDA CUDA, OpenCL, OpenACC, OpenMP 4.0
  - **Vectorization (SIMD):** SSE/AVX, AltiVec



Мультиархитектура современных ВС

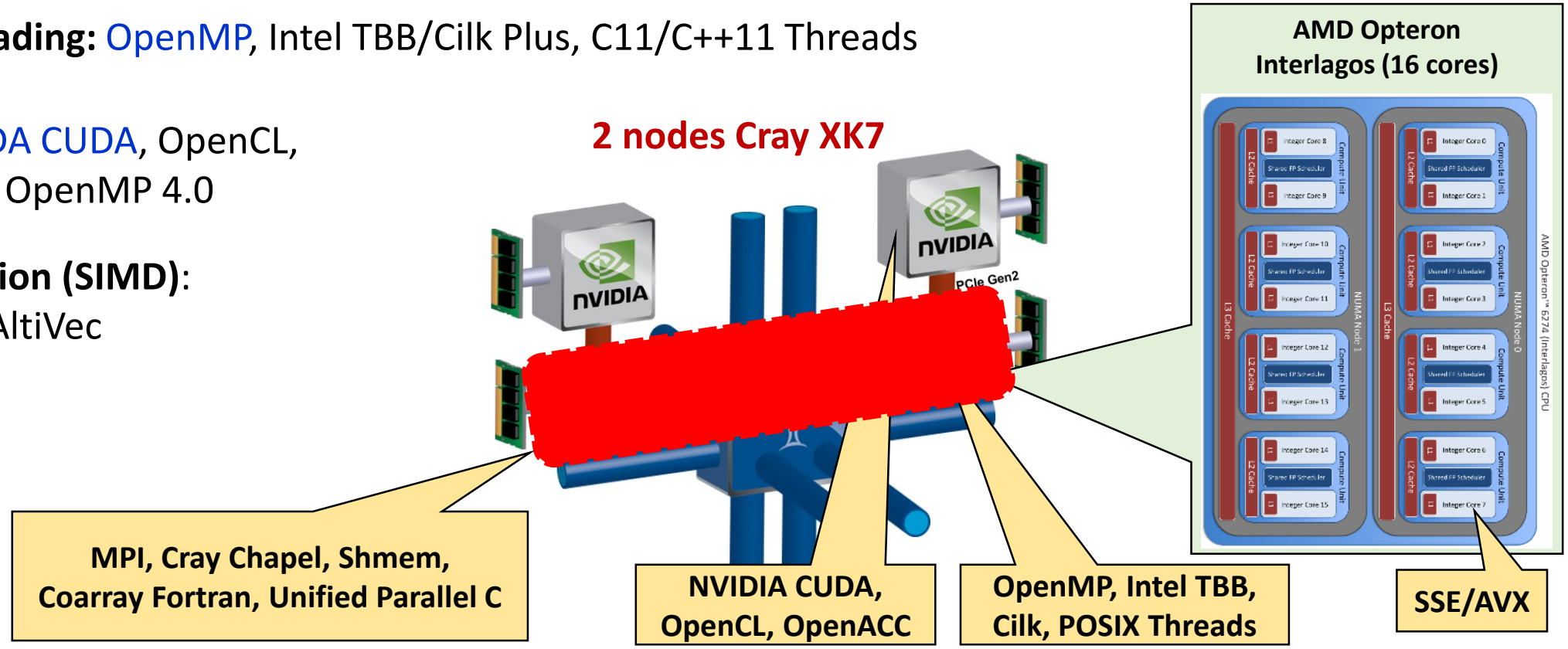
- **Разработка эффективных параллельных программ для мультиархитектурных ВС требует владения стеком технологий**
  - **Internode communications:** MPI, Cray Chapel, IBM X10, Shmem, Unified Parallel C, Coarray Fortran, Global Arrays
  - **Multithreading:** OpenMP, Intel TBB/Cilk Plus, C11/C++11 Threads
  - **GPU:** NVIDA CUDA, OpenCL, OpenACC, OpenMP 4.0
  - **Vectorization (SIMD):** SSE/AVX, AltiVec



Мультиархитектура современных ВС

- **Разработка эффективных параллельных программ для мультиархитектурных ВС требует владения стеком технологий**

  - **Internode communications:** MPI, Cray Chapel, IBM X10, Shmem, Unified Parallel C, Coarray Fortran, Global Arrays
  - **Multithreading:** OpenMP, Intel TBB/Cilk Plus, C11/C++11 Threads
  - **GPU:** NVIDA CUDA, OpenCL, OpenACC, OpenMP 4.0
  - **Vectorization (SIMD):** SSE/AVX, AltiVec



Многопроцессорные ВС с общей памятью (SMP, NUMA)

- **Плюсы**
  - Привычная модель программирования (потоки, процессы)
  - Высокая скорость обмена данными между потоками/процессами
- **Минусы**
  - Синхронизация при доступе к общим данным (критические секции)
  - Когерентность кэшей, ложное разделение данных
  - Относительно низкая масштабируемость
    (как правило, 4 – 16 процессоров на узле)
  - Трудоемкая организация эффективного использование памяти в NUMA-системах

Многопроцессорные ВС с распределенной памятью

- **Плюсы**
  - Высокая масштабируемость (сотни, тысячи и миллионы процессорных ядер)
  - Меньше проблем с синхронизацией (у каждого узла/сервера своя память)
  - Декомпозиция на крупные подзадачи
- **Минусы**
  - Необходимость использования передачи сообщений (message passing)
  - Высокие временные задержки и низкая пропускная способность (все взаимодействия по сети – Gigabit Ethernet, Infiniband)
  - Неоднородность, отказы узлов

Теоретические основы параллельных вычислений

Параллельные вычисления (Parallel Computing)

- **Разработка параллельного алгоритма**
  - ❑ Поиск параллелизма в известном последовательном алгоритме, его модификация или создание нового алгоритма
  - ❑ Декомпозиция задачи на подзадачи, которые могут выполняться параллельно
  - ❑ Анализ зависимостей между подзадачами
- Параллельная версия самого эффективного последовательного алгоритма решения задачи необязательно будет самой эффективной параллельной реализацией

Параллельные вычисления (Parallel Computing)

- **Реализация параллельного алгоритма в виде параллельной программы**
  - Распределение подзадач между процессорами (task mapping, load balancing)
  - Организация взаимодействия подзадач (message passing, shared data structures)
  - Учет архитектуры целевой вычислительной системы
  - Запуск, измерение и анализ показателей эффективности параллельной программы
  - Оптимизация программы

Показатели эффективности параллельных алгоритмов

- Коэффициент ускорения (Speedup)
- Коэффициент эффективности (Efficiency)
- Коэффициент накладных расходов
- Показатель равномерности загруженности параллельных ветвей (процессов, потоков)

Коэффициент ускорения (Speedup)

- Введем обозначения:
  - $\boldsymbol{T(n)}$ – время выполнения последовательной программы (sequential program)
  - $\boldsymbol{T_p(n)}$ – время выполнения параллельной программы (parallel program) на *p* процессорах
- **Коэффициент $S_p(n)$ ускорения** параллельной программ (Speedup):

$$\boldsymbol{S_p(n) = \frac{T(n)}{T_p(n)}}$$

- Коэффициент ускорения $S_p(n)$ показывает во сколько раз параллельная программа выполняется на *p* процессорах быстрее последовательной программы при обработке одних и тех же входных данных размера *n*
- Как правило

$$S_p(n) \leq p$$

Коэффициент ускорения (Speedup)

- Введем обозначения:
  - $T(n)$ – время выполнения последовательной программы (sequential program)
  - $T_p(n)$ – время выполнения параллельной программы (parallel program) на p процессорах
- **Коэффициент $S_p(n)$ ускорения** параллельной программ (Speedup):

$$S_p(n) = \frac{T(n)}{T_p(n)}$$

- **Цель распараллеливания – достичь линейного ускорения на максимально большом числе процессоров**

$$S_p(n) \approx p \quad \text{или} \quad S_p(n) = \Omega(p) \quad \text{при} \quad p \to \infty$$

Коэффициент ускорения (Speedup)

- **Какое время брать за время выполнения последовательной программы?**
  - Время лучшего известного алгоритма (в смысле вычислительной сложности)?
  - Время лучшего теоретически возможного алгоритма?

- **Что считать временем выполнения $T_p(n)$ параллельной программы?**
  - Среднее время выполнения потоков программы?
  - Время выполнения потока, завершившего работу первым?
  - Время выполнения потока, завершившего работу последним?

Коэффициент ускорения (Speedup)

- **Какое время брать за время выполнения последовательной программы?**
  - **Время лучшего известного алгоритма или время алгоритма, который подвергается распараллеливанию**

- **Что считать временем выполнения $T_p(n)$ параллельной программы?**
  - **Время выполнения потока, завершившего работу последним**

Коэффициент ускорения (Speedup)

- **Коэффициент относительного ускорения** (Relative speedup) – отношения времени выполнения параллельной программы на *k* процессорах к времени её выполнения на *p* процессорах (*k* < *p*)

$$S_{Relative}(k,p,n) = \frac{T_k(n)}{T_p(n)}$$

- **Коэффициент эффективности (Efficiency)** параллельной программы

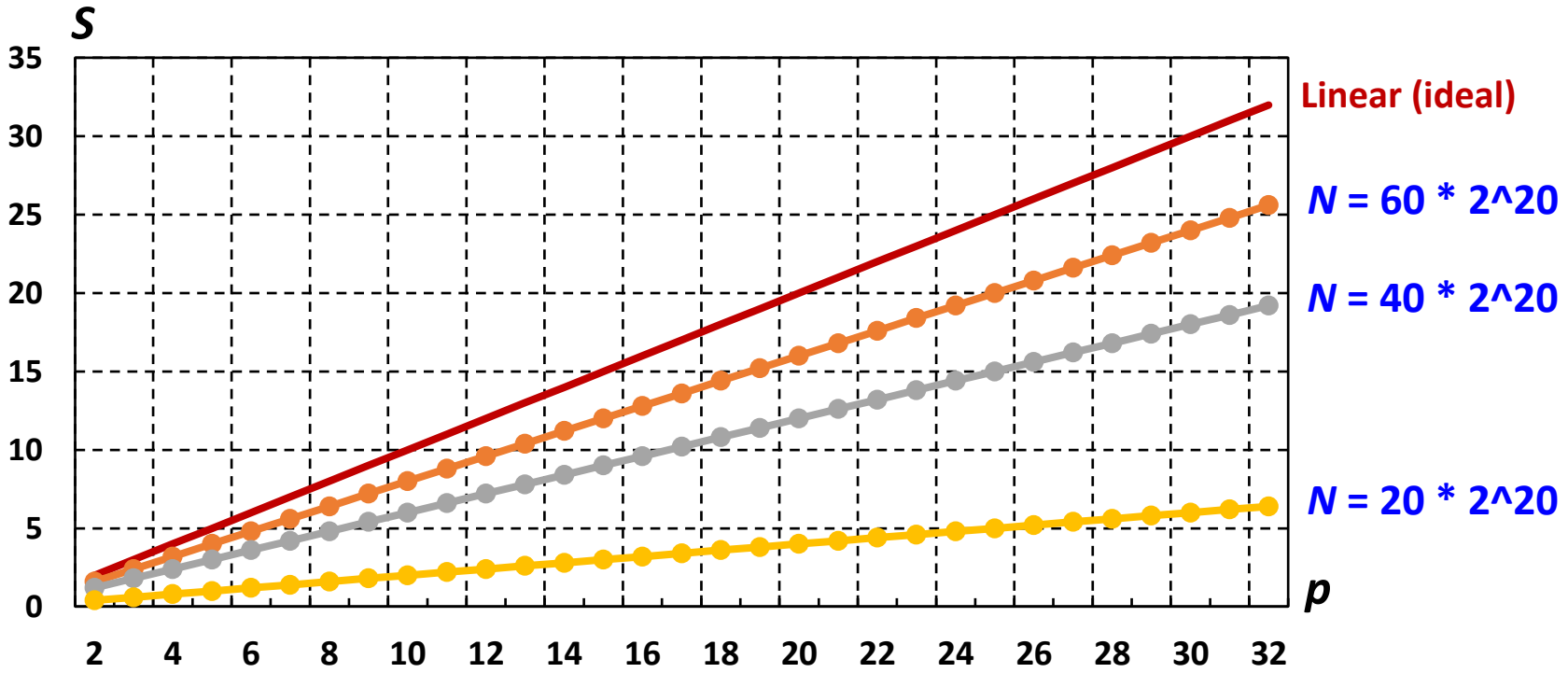
$$E_p(n) = \frac{S_p(n)}{p} = \frac{T(n)}{pT_p(n)} \in [0,1]$$

- **Коэффициент накладных расходов (Overhead)**

$$\varepsilon(p,n) = \frac{T_{Sync}(p,n)}{T_{Comp}(p,n)} = \frac{T_{Total}(p,n) - T_{Comp}(p,n)}{T_{Comp}(p,n)}$$

- $T_{Sync}(p,n)$ – время создания, синхронизации и взаимодействия *p* потоков
- $T_{Comp}(p,n)$ – время вычислений в каждом из *p* потоков

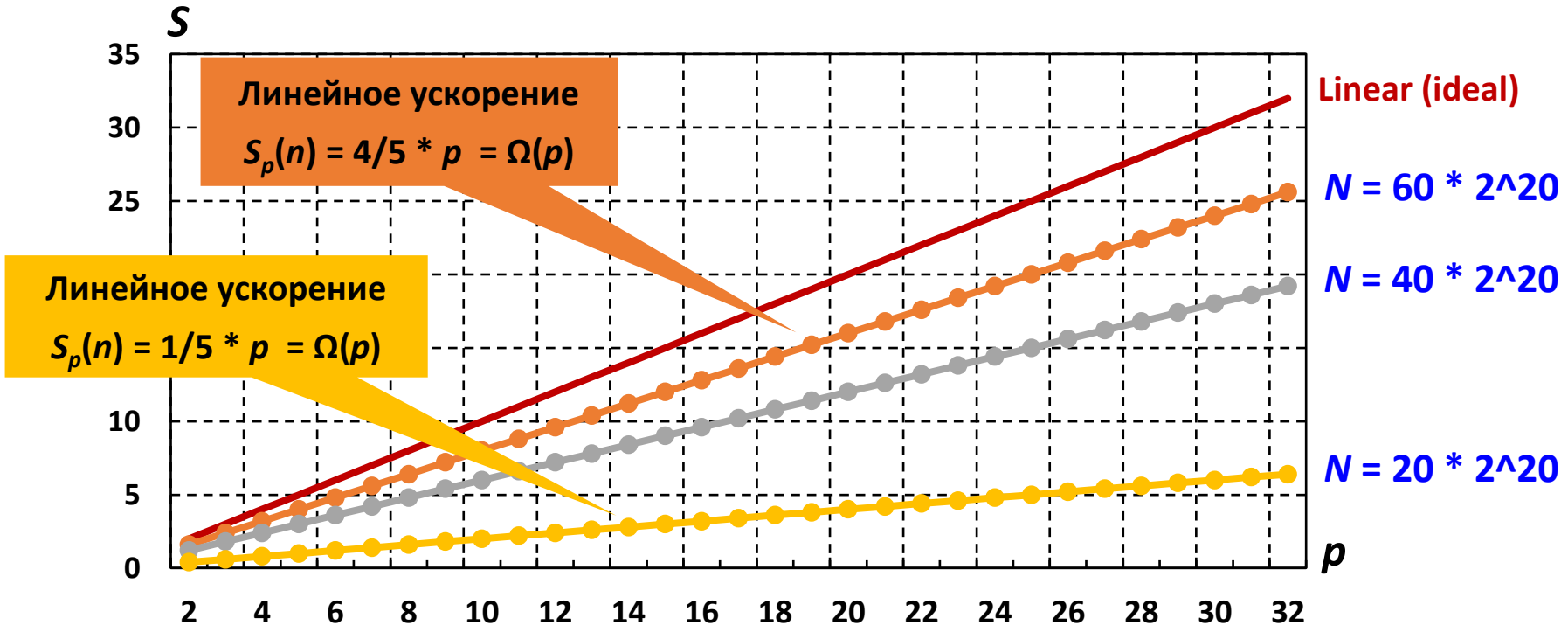
Коэффициент ускорения (Speedup)



Зависимость коэффициента ускорения *S*
параллельного алгоритма X от количества *p* процессоров

- Ускорение программы может расти с увеличением размера входных данных
- Время вычислений превосходит накладные расходы на взаимодействия потоков (управление потоками, синхронизацию, обмен сообщениями, …)

Коэффициент ускорения (Speedup)



Зависимость коэффициента ускорения *S*
параллельного алгоритма X от количества *p* процессоров

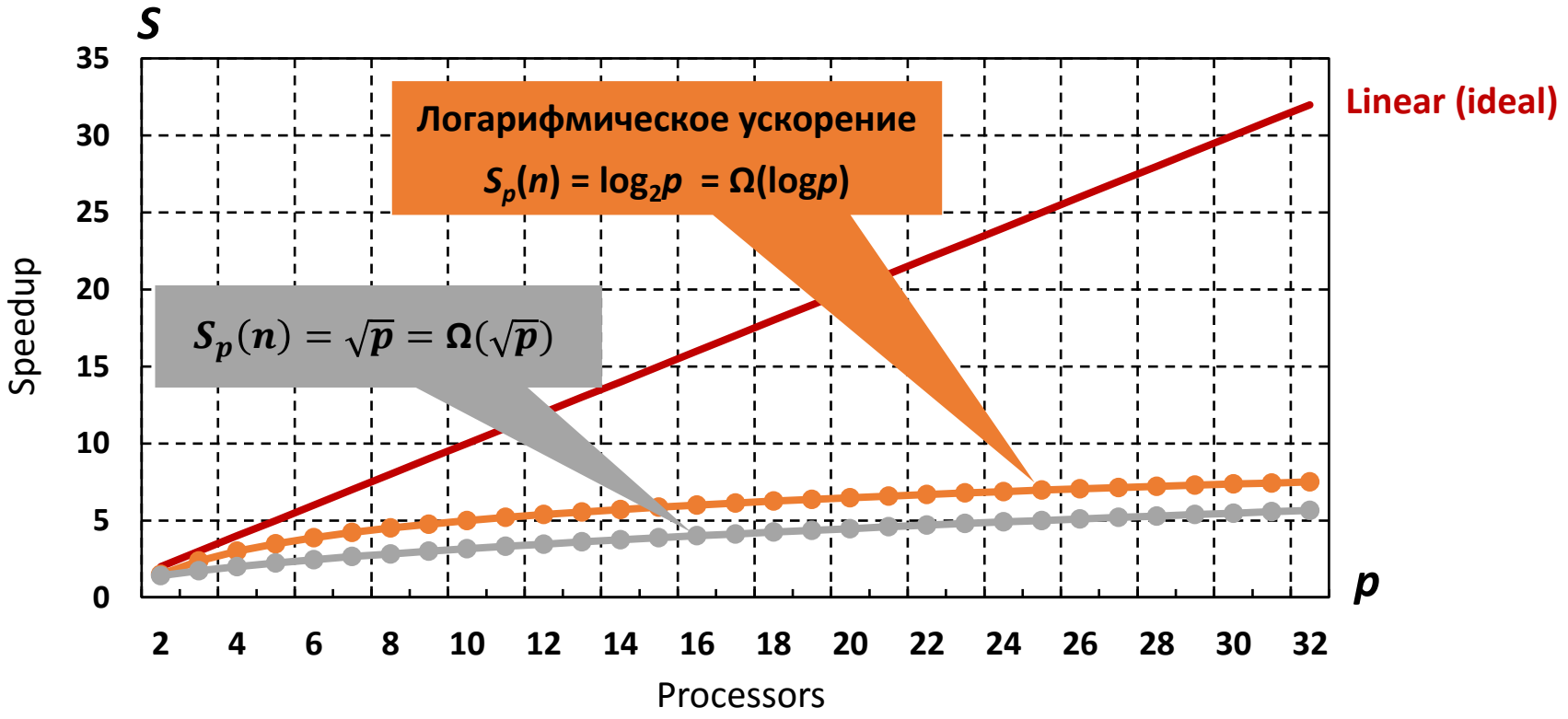
- Ускорение программы может расти с увеличением размера входных данных
- Время вычислений превосходит накладные расходы на взаимодействия потоков (управление потоками, синхронизацию, обмен сообщениями, …)

Коэффициент ускорения (Speedup)

- Параллельная программа (алгоритм) коэффициент ускорения, которой линейной растет с увеличением *p* называется линейно масштабируемой или просто **масштабируемой** (scalable)

- Масштабируемая параллельная программа допускает эффективную реализацию на различном числе процессоров

Коэффициент ускорения (Speedup)



Зависимость коэффициента ускорения *S*
параллельных алгоритмов Y и Z от количества *p* процессоров

Суперлинейное ускорение (Superlinear speedup)

- Параллельная программа может характеризоваться **суперлинейным ускорением** (Superlinear speedup) – коэффициент ускорения $S_p(n)$ принимает значение больше p

$$\boldsymbol{S_p(n) > p}$$

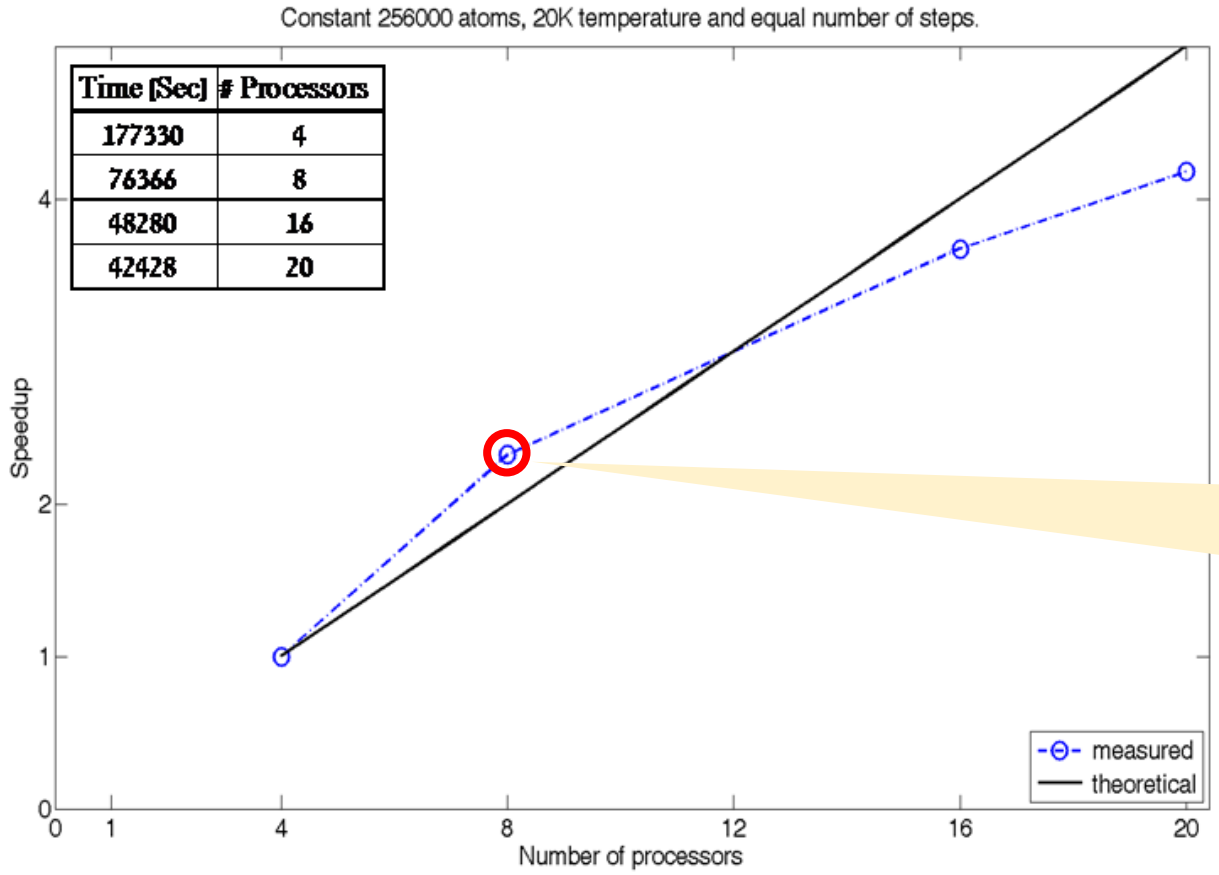
- Причина: иерархическая организация памяти:
  Cache – RAM – Local disk (HDD/SSD) – Network storage

- Последовательная программ выполняется на одном процессоре
  и обрабатывает данные размера n

- Параллельная программа имеет p потоков на p процессорах, каждый поток работает со своей частью данных, большая часть которых может попасть в кеш-память,
  в результате в каждом потоке сокращается время доступа к данным

- Тот же самый эффект можно наблюдать имя два уровня иерархической памяти:
  диск – память

Суперлинейное ускорение (Superlinear speedup)



Superlinear speedup (relative)

$$S_8 = \frac{T_4}{T_8} = 2.32$$

Parallel Molecular Dynamic Simulation

MPI, Spatial decomposition; Cluster nodes: 2 x AMD Opteron Dual Core; InfiniBand network

http://phycomp.technion.ac.il/~pavelba/Comp_Phys/Project/Project.html

Равномерность распределения вычислений

- **По какому показателю оценивать равномерность времени выполнения потоков/процессов параллельной программы?**

- Известно время выполнения потоков t_0, t_1, …, t_{p-1}

- Коэффициент *V* вариации

$$V = \frac{\sigma[t_i]}{\mu[t_i]}$$

- Отношение min/max

$$M = \frac{\min\{t_i\}}{\max\{t_i\}}$$

- Jain’s fairness index

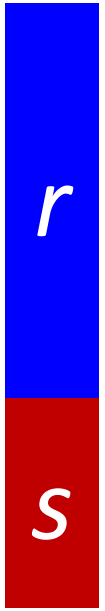
$$f = \frac{\left(\sum_{i=0}^{p-1} t_i\right)^2}{n \sum_{i=0}^{p-1} t_i^2} \in [0, 1]$$

“Последовательные” части в программах

- Инициализация и завершение работы
- Чтение входных данных и запись
- Инициализация данных
- Синхронизация, критические секции
- Пул потоков обрабатывает независимые задания
  - Извлечение заданий из очереди
  - Обработка результатов
    - ❑ Запись результатов в общую структуру данных
    - ❑ Слияние результатов из локальных структур данных

Закон Дж. Амдала (Amdahl's law)

- Пусть имеется последовательная программа с временем выполнения $T(n)$
- Обозначим:
  - $r \in [0, 1]$ – часть программы, которая может быть распараллелена (perfectly parallelized)
  - $s = 1 - r$ – часть программы, которая не может быть распараллелена (purely sequential)



- Время выполнения параллельной программы на p процессорах (время каждого потока) складывается из последовательной части s и параллельной r:

$$T_p(n) = T(n)s + \frac{T(n)}{p}r$$

- Вычислим значение коэффициент ускорения (по определению)

$$S_p(n) = \frac{T(n)}{T_p(n)} = \frac{T(n)}{T(n)s + \frac{T(n)}{p}r} = \frac{1}{s + \frac{r}{p}} = \frac{1}{(1 - r) + \frac{r}{p}}$$

- Полученная формула по значениям r и s позволяет оценить максимальное ускорение

Закон Дж. Амдала (Amdahl's law)

- Пусть имеется последовательная программа с временем выполнения $T(n)$
- Обозначим:
  - $r \in [0, 1]$ – часть программы, которая может быть распараллелена (perfectly parallelized)
  - $s = 1 - r$ – часть программы, которая не может быть распараллелена (purely sequential)
- **Закон Дж. Амдала (Gene Amdahl, 1967) [1]:**

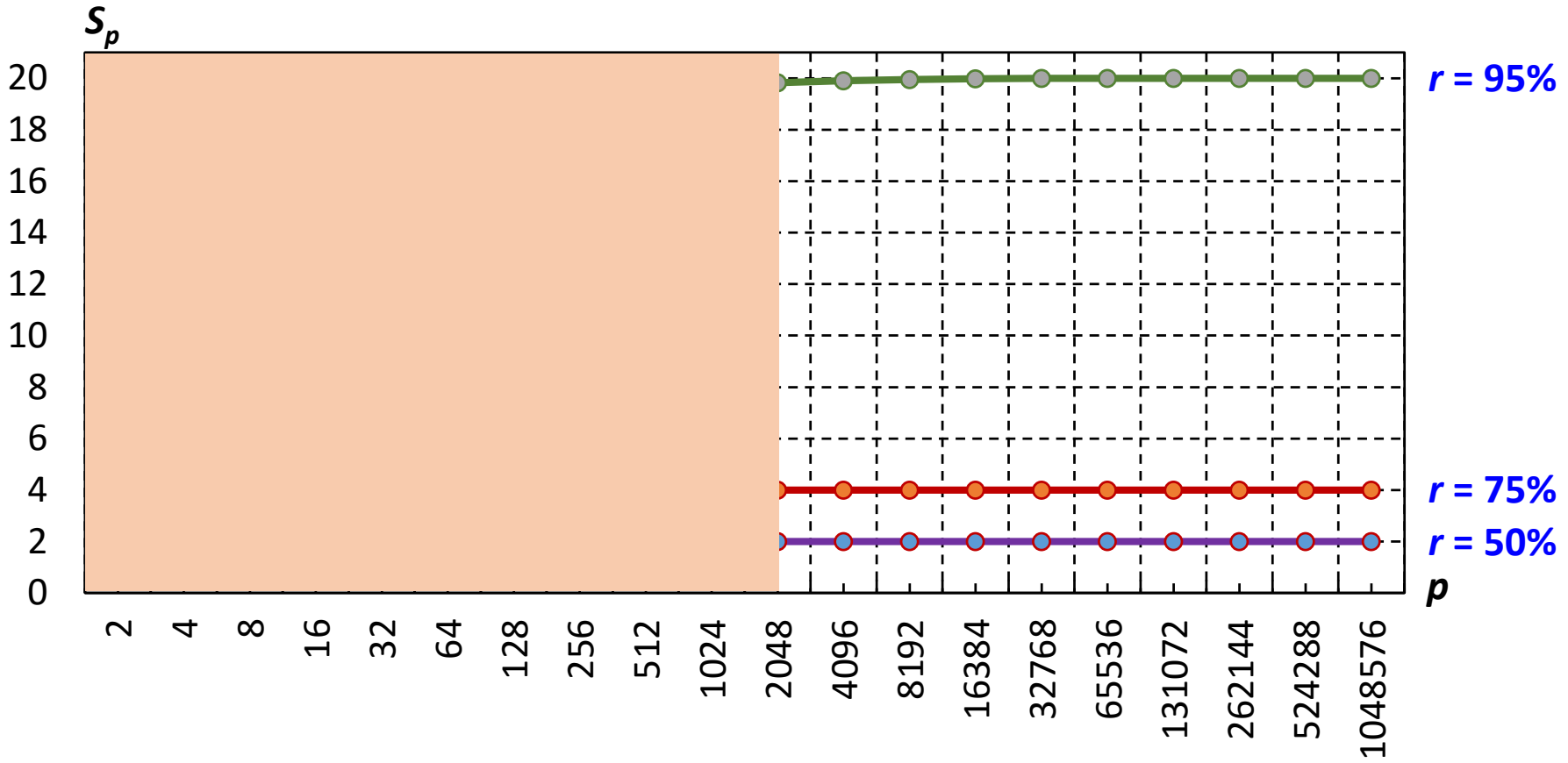
  Максимальное ускорение S_p программы на p процессорах равняется

$$S_p = \frac{1}{(1 - r) + \frac{r}{p}}$$

$$S_\infty = \lim_{p \to \infty} S_p = \lim_{p \to \infty} \frac{1}{(1 - r) + \frac{r}{p}} = \frac{1}{1 - r} = \frac{1}{s}$$

❑ Amdahl Gene. **Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities** // AFIPS Conference Proceedings, 1967, pp. 483-485, http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf

Закон Дж. Амдала (Amdahl's law)



Зависимость коэффициента S_p ускорения параллельной программы от количества p процессоров

Имеет ли смысл создавать системы с количеством процессоров > 1024?

Пример

```
// Последовательная часть: инициализация
x = (int *)calloc(n, sizeof(int));

// Распараллеливаемая часть
do {
    for (i = 0; i < n; i++) {
        x[i] = f(i);  // O(1)
    }

    // Проверка сходимости
    done = ... ;      // O(1)
} while (!done)
```

- Пусть для определенности, цикл **do** завершается после *k* итераций
- Цикл **for** можно эффективно распараллелить
- $T_{seq}(n) = \boldsymbol{n} + k + kn$
- $T_{par}(n) = \boldsymbol{n} + k + kn/p$
- $s = \frac{n+k}{n+k+kn}$, $r = \frac{kn}{n+k+kn}$

- Тогда доля последовательного кода при $n \to \infty$:

$$\lim_{n\to\infty} s = \lim_{n\to\infty} \frac{n+k}{n+k+kn} = \frac{1}{1+k}$$

- Ускорение $\boldsymbol{S_\infty = 1/s = 1 + k}$

Пример

```
// Последовательная часть: инициализация
x = (int *)malloc(n * sizeof(int));

// Распараллеливаемая часть
do {
    for (i = 0; i < n; i++) {
        x[i] = f(i);  // O(1)
    }

    // Проверка сходимости
    done = ... ;      // O(1)
} while (!done)
```

- Пусть для определенности,
  цикл **do** завершается после k итераций
- Цикл **for** можно эффективно распараллелить
- $T_{seq}(n) = \mathbf{1} + k + kn$
- $T_{par}(n) = \mathbf{1} + k + kn/p$
- $s = \frac{1+k}{1+k+kn}$, $r = \frac{kn}{1+k+kn}$

Ускорение $\boldsymbol{S \to p}$, **при** $\boldsymbol{n \to \infty}$

Допущения закона Дж. Амдала (Amdahl's law)

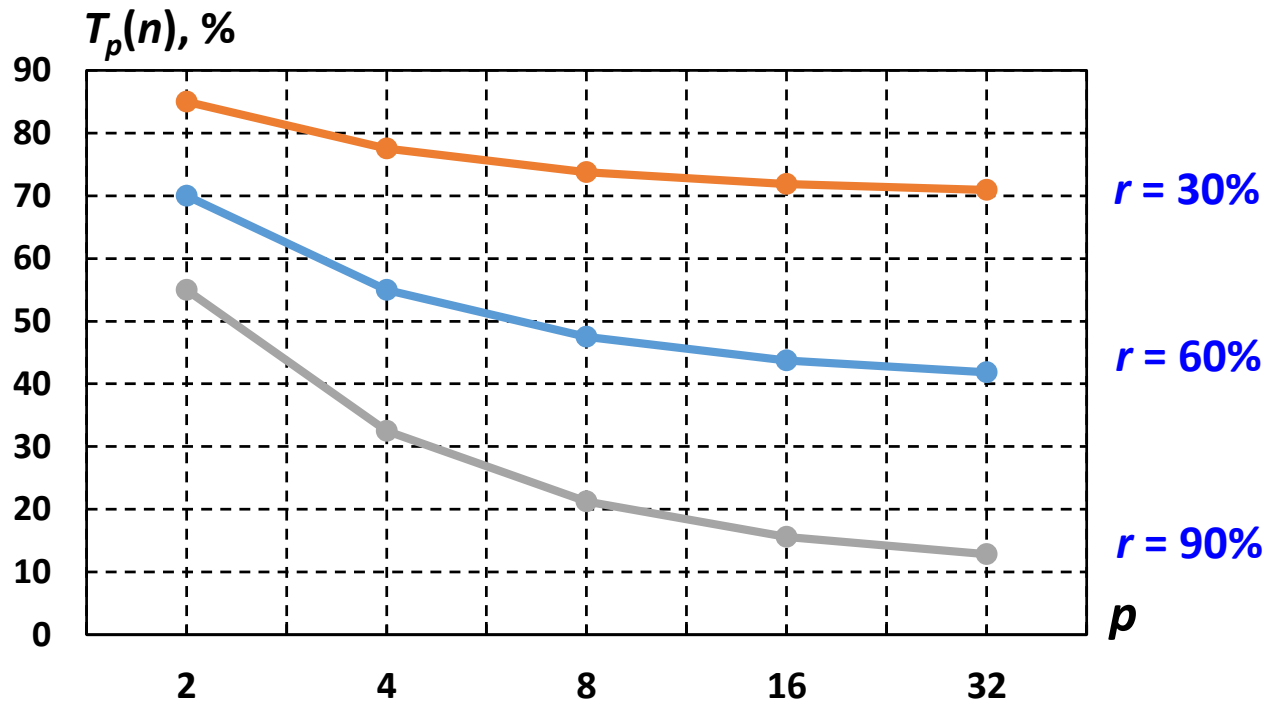
- **Последовательный алгоритм является наиболее оптимальным способом решения задачи**
- Возможны ситуации когда параллельная программа (алгоритм) эффективнее решает задачу (может эффективнее использовать кеш-память, конвейер, SIMD-инструкции, …)
- **Время выполнения параллельной программы оценивается через время выполнения последовательной**, однако потоки параллельной программы могут выполнятся эффективнее

$$T_p(n) = T(n)s + \frac{T(n)}{p}r, \quad \text{на практике возможна ситуация } \frac{T(n)}{p} > T_p(n)$$

- **Ускорение $S_p(n)$ оценивается для фиксированного размера n данных при любых значениях p**
- В реальности при увеличении числа используемых процессоров размер n входных данных также увеличивают, так как может быть доступно больше памяти

Закон Дж. Амдала (Amdahl's law)

- **На что потратить ресурсы – на увеличение доли *r* параллельной части в программе или увеличение числа процессоров, на которых запускается программа?**



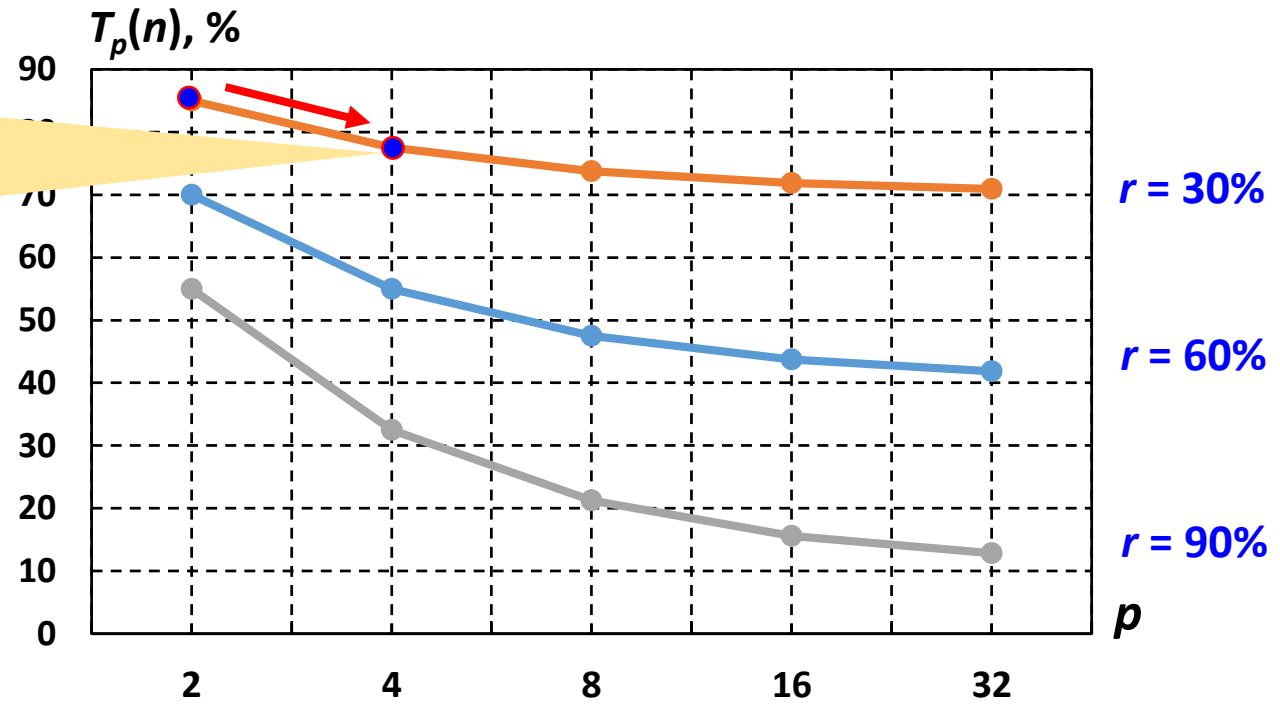
Зависимость времени $T_p(n)$ выполнения параллельной программы от количества *p* процессоров и доли *r* распараллеленного кода (время в % от времени $T_1(n)$)

Закон Дж. Амдала (Amdahl's law)

- **На что потратить ресурсы – на увеличение доли *r* параллельной части в программе или увеличение числа процессоров, на которых запускается программа?**

Увеличили число процессоров с 2-х до 4-х (программу не меняли)

Время выполнения сократилось с 85% до 77,5%



Зависимость времени $T_p(n)$ выполнения параллельной программы от количества *p* процессоров и доли *r* распараллеленного кода (время в % от времени $T_1(n)$)

Закон Дж. Амдала (Amdahl's law)

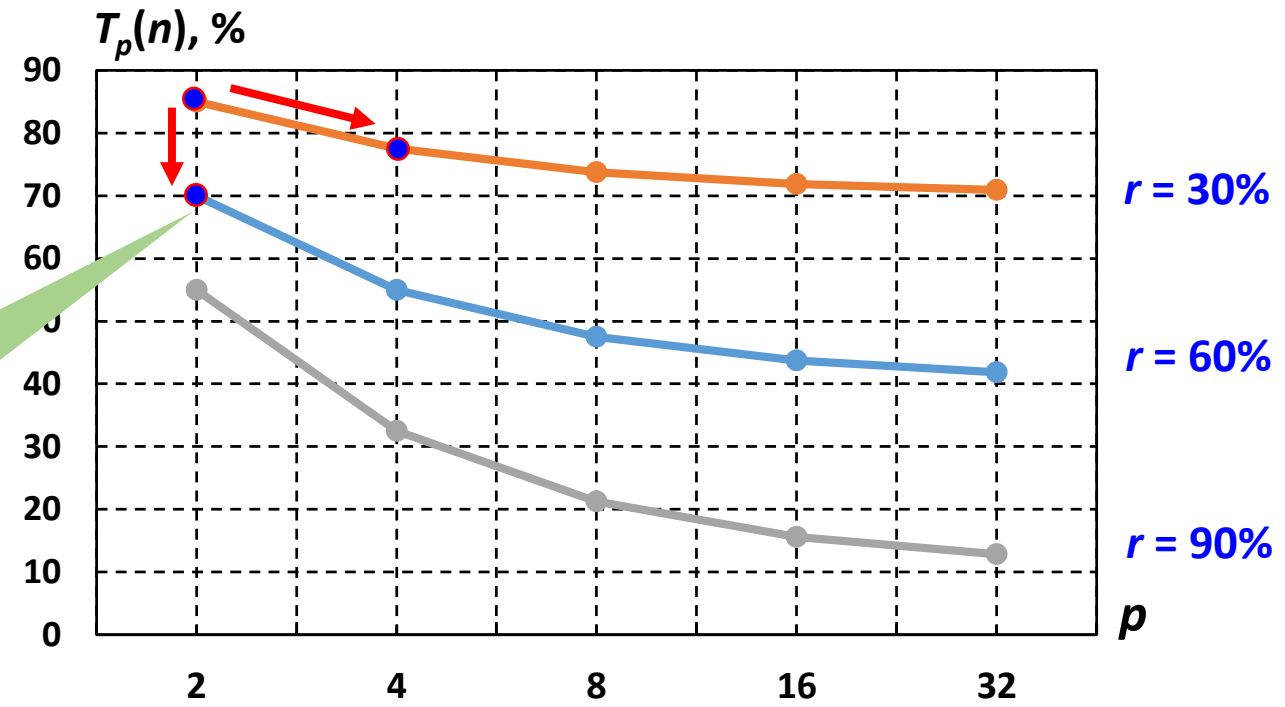
- **На что потратить ресурсы – на увеличение доли *r* параллельной части в программе или увеличение числа процессоров, на которых запускается программа?**

Увеличили число процессоров с 2-х до 4-х (программу не меняли)

Время выполнения сократилось с 85% до 77,5%

Увеличим в 2 раза долю параллельного кода

Время выполнения сократилось с 85% до 70%



Зависимость времени $T_p(n)$ выполнения параллельной программы от количества *p* процессоров и доли *r* распараллеленного кода (время в % от времени $T_1(n)$)

Закон Густафсона-Барсиса

- Пусть имеется последовательная программа с временем выполнения $T(n)$
- Обозначим $\boldsymbol{s} \in [\mathbf{0}, \mathbf{1}]$ – часть параллельной программы, которая выполняется последовательно (purely sequential)
- **Закон Густафсона-Барсиса (Gustafson–Barsis' law) [1]:**

Масштабируемое ускорение S_p программы на p процессорах равняется

$$\boldsymbol{S_p = p - s(p-1)}$$

- **Обоснование:** пусть a – время последовательной части, b – время параллельной части

$$\boldsymbol{T_p(n) = a + b, \qquad T(n) = a + pb}$$

$$\boldsymbol{s = a/(a+b), \qquad S_p(n) = s + p(1-s) = p - s(p-1)}$$

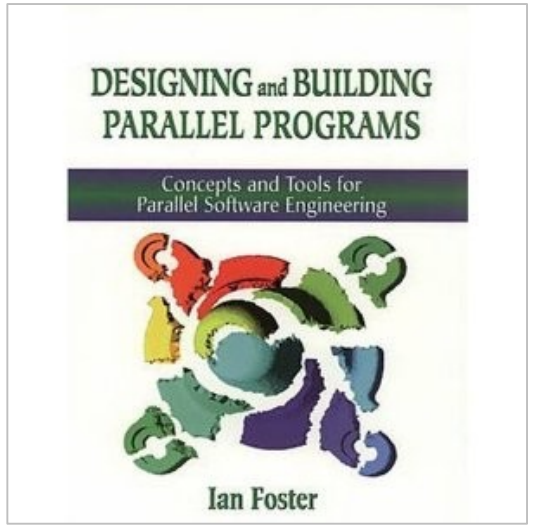
- **Время выполнения последовательной программы выражается через время выполнения параллельной**

❑ **Reevaluating Amdahl's Law**, John L. Gustafson, Communications of the ACM 31(5), 1988. pp. 532-533 // http://www.scl.ameslab.gov/Publications/Gus/AmdahlsLaw/Amdahls.htmlЯ

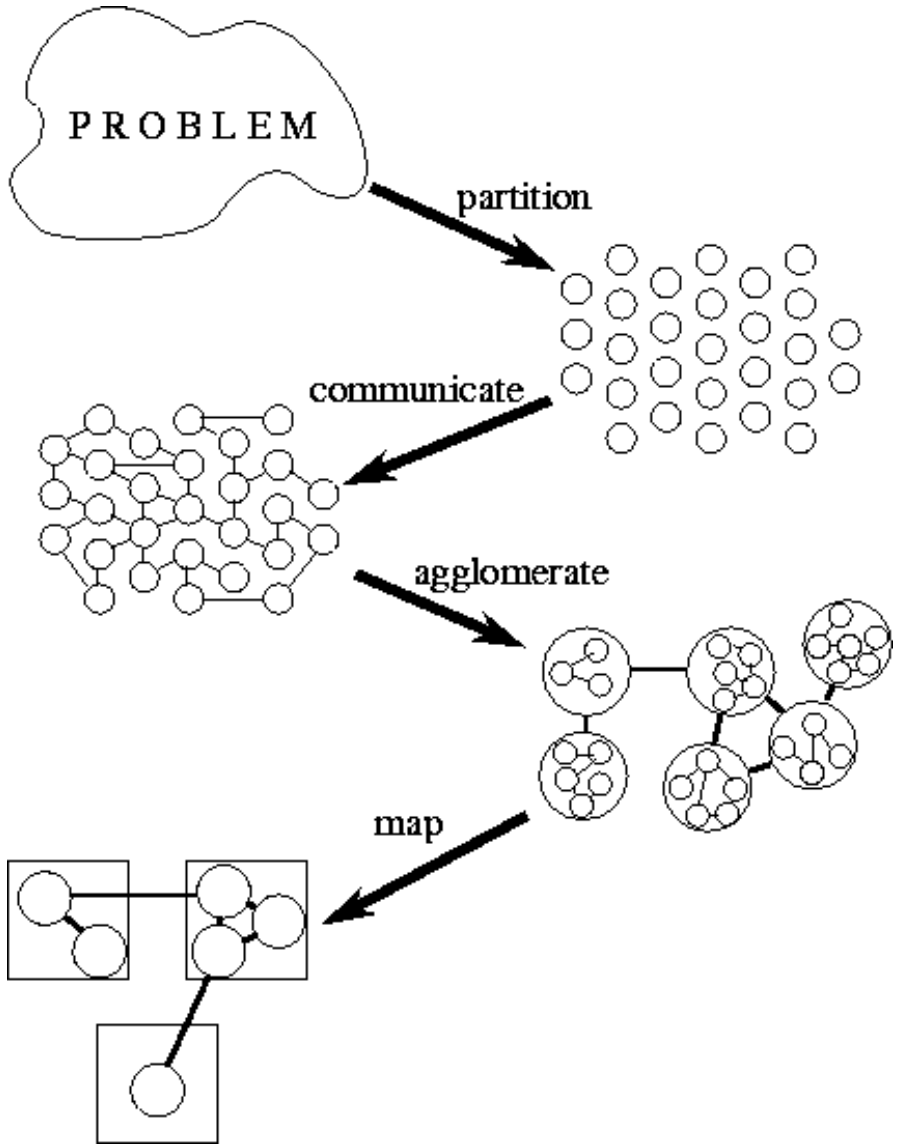
Принципы разработки параллельных алгоритмов

Методология PCAM

- **Методология PCAM** описывает общий подход к процессу разработки параллельного алгоритма для решения заданной задачи
- **PCAM** = Partitioning, Communication, Agglomeration, Mapping
- Foster I. **Designing and Building Parallel Programs: Concepts and Tools for Software Engineering**. Reading, MA: Addison-Wesley, 1995 // http://www.mcs.anl.gov/~itf/dbpp/



Методология PCAM



- **Декомпозиция** (Partition) вычислений и данных на параллельные подзадачи (архитектура ВС игнорируется)
- **Анализ зависимостей и разработка алгоритма взаимодействия** (Communicate) параллельных подзадач
- Выбор целевой вычислительной системы (класса)
- Масштабирование подзадач (Agglomerate) с учетом архитектуры выбранной ВС
- Распределение подзадач (Map) между процессорами (статическое или динамическое)

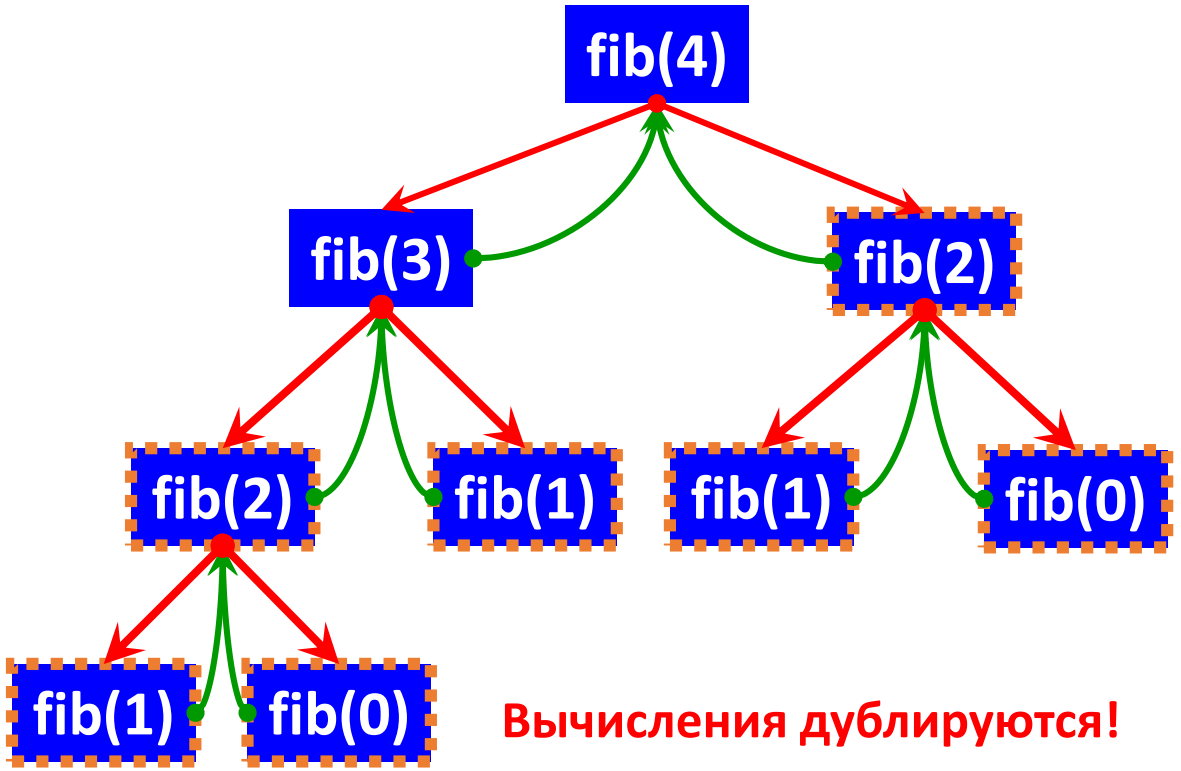
Декомпозиция на подзадачи

- **Выявление возможностей для параллельного выполнения**
- Разбиение задачи на подзадачи минимального размера – *fine-grained decomposition* (мелокзернистая декомпозиция)
- **Виды декомпозиции**
  - **По данным (Domain decomposition)** – распределение данных по подзадачам
  - **Функциональная декомпозиция (Functional decomposition)** – распределение вычислений по подзадачам
- Необходимо избегать дублирования вычислений и данных

Функциональная декомпозиция

```
function fib(int n)
    if n < 2 then
        return n
    x = fork task fib(n - 1)
    y = fork task fib(n - 2)
    join threadX
    join threadY
    return x + y
end function
```

- **Параллельное вычисление *n*-ого члена последовательности Фибоначчи**
- Функциональная декомпозиция:
  каждый рекурсивный вызов – это отдельная подзадача



Выбор структуры алгоритма

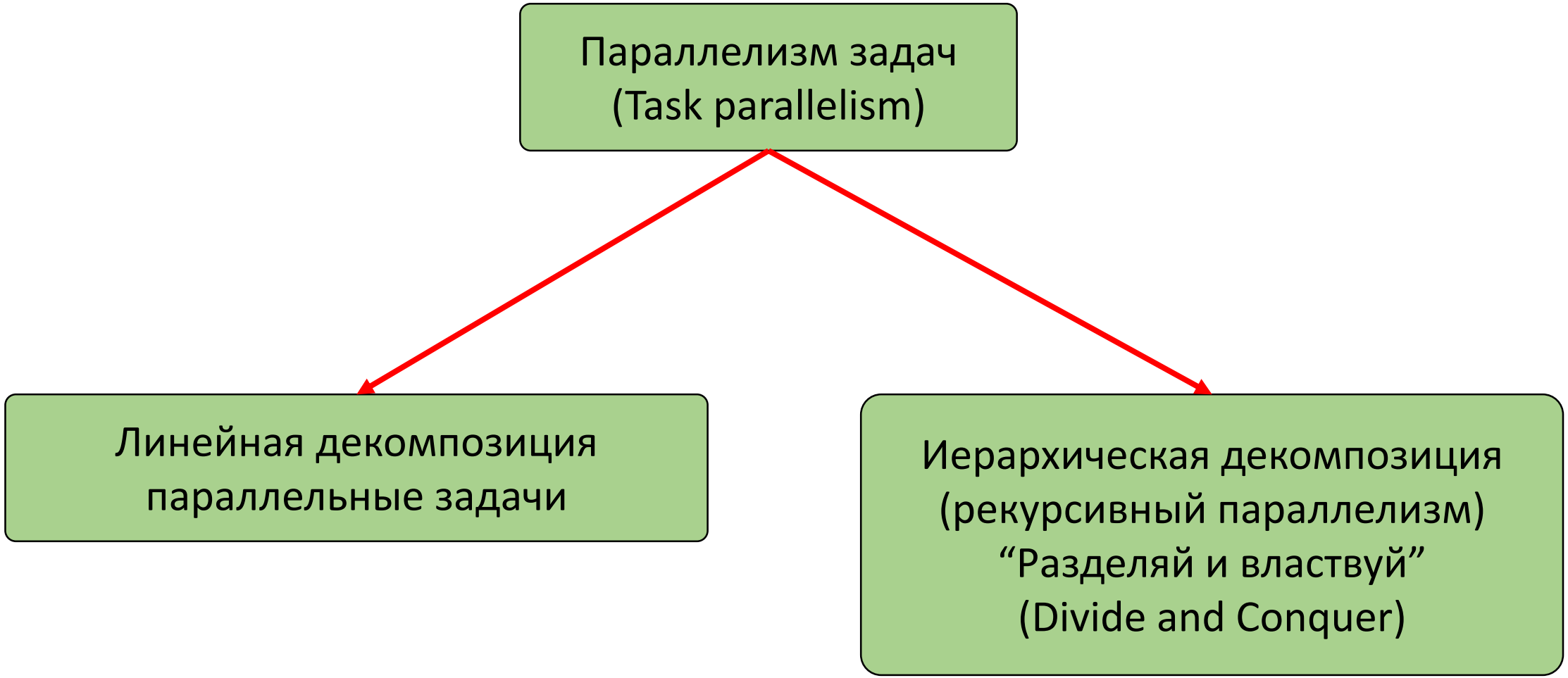
- **Существуют типовые структуры (паттерны) параллельных алгоритмов**

- Mattson T., Sanders B., Massingill B. **Patterns for Parallel Programming**. – Addison-Wesley, 2004

- Krste Asanovic, Ras Bodik, Bryan Christopher et. al. **The Landscape of Parallel Computing Research: A View from Berkeley** // http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

- **Dwarf Mine** // http://view.eecs.berkeley.edu/wiki/Dwarfs

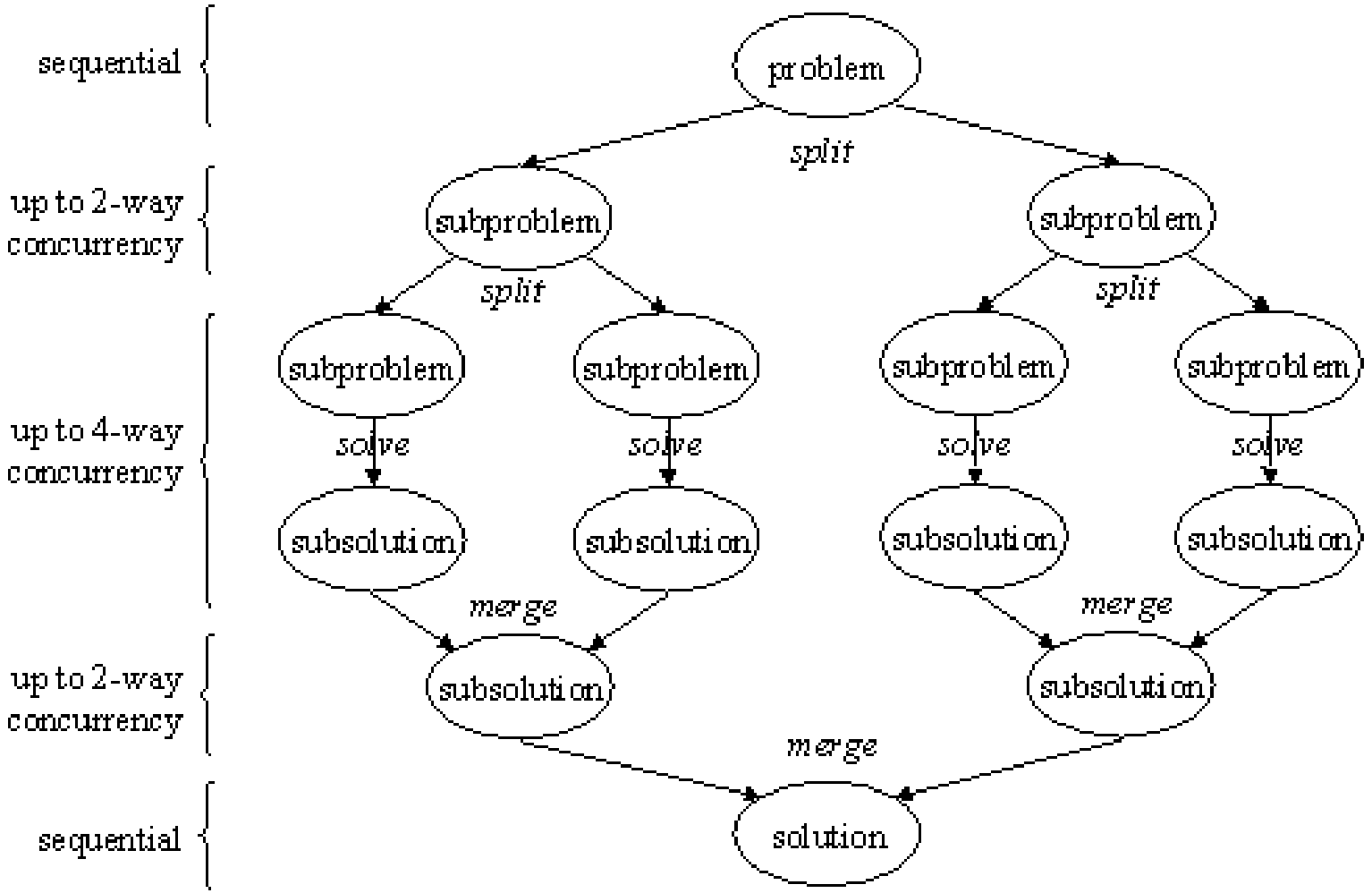
Функциональная декомпозиция



Параллелизм задач (Task parallelism)

- Многовариантный счет, методы Монте-Карло, рендеринг графических сцен
  - Большое количество параллельных подзадач, между задачами нет зависимостей по данным (embarrassingly parallel)
- Молекулярная динамика (система из *n* взаимодействующих атомов)
  - Параллельное вычисление сил, действующих на атом
- Метод «ветвей и границ» (branch and bound)
  - Обход и разбиение множества решений в соответствии с правилами отсева и ветвления
  - Динамическое порождение заданий

Рекурсивный параллелизм (разделяй и властвуй)

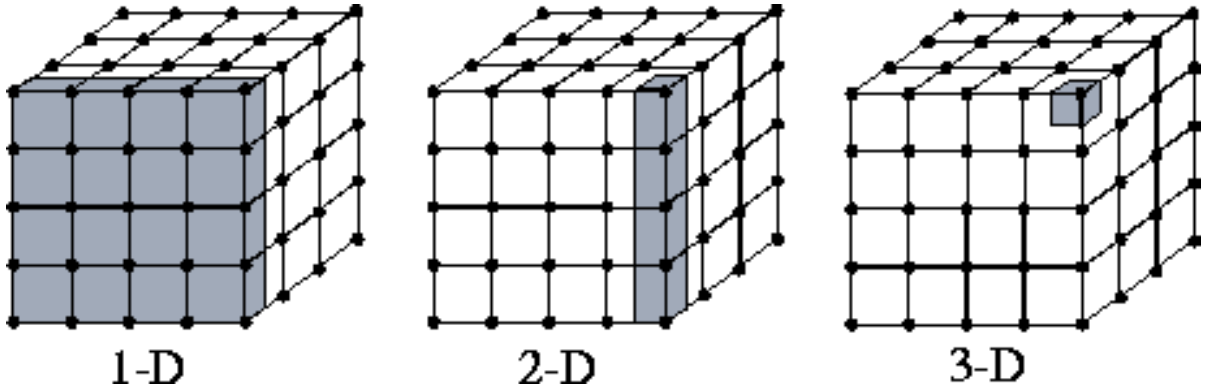


- Parallel Merge Sort
- Parallel Quick Sort

Рекурсивный параллелизм (разделяй и властвуй)

- Степень параллелизма изменяется в ходе выполнения алгоритма
- Операции Split и Merge могут стать узким местом (выполняются последовательно, см. закон Амдала)
- Задания порождаются динамически (балансировка загрузки потоков)
- Очень большое количество заданий может привести к значительным накладным расходам

Геометрическая декомпозиция (Domain decomposition)



- **Данные задачи разбиваются на области** (желательно равного размера)
- С каждой областью данных ассоциируются алгоритм её обработки
- **Вычисления локализованы внутри области?**
  - Да: независимые подзадачи
  - Нет: требуется разделение данных между областями

Геометрическая декомпозиция (Domain decomposition)

- **Декомпозиция структуры данных на области**
  - Размер подзадач обычно подбирается эмпирически
  - Форма области влияет на накладные расходы (отношение объема к площади поверхности)
  - Дублирование соседних точек
- **Реализация обмена данными**
  - Перед операцией обновления
  - Параллельно с операцией обновления

Геометрическая декомпозиция (Domain decomposition)

- **Декомпозиция структуры данных на области**
  - Размер подзадач обычно подбирается эмпирически
  - Форма области влияет на накладные расходы (отношение объема к площади поверхности)
  - Дублирование соседних точек
- **Реализация обмена данными**
  - Перед операцией обновления
  - Параллельно с операцией обновления

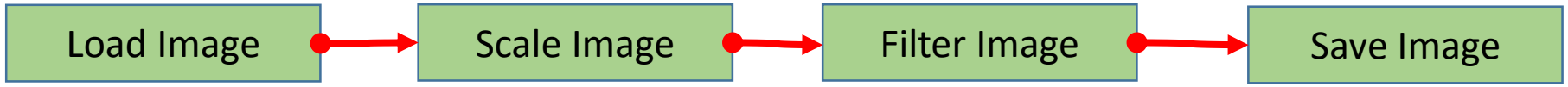
Рекурсивная декомпозиция

- Алгоритм работает с рекурсивной структурой данных (список, дерево, граф)
- Часто кажется, что единственный способ решения – последовательный обход структуры
- Однако иногда возможно перестроить алгоритм так, что операции над отдельными элементами можно выполнять одновременно

- Vivek Sarkar. **Parallel Graph Algorithms** // http://www.cs.rice.edu/~vs3/comp422/lecture-notes/comp422-lec24-s08-v2.pdf

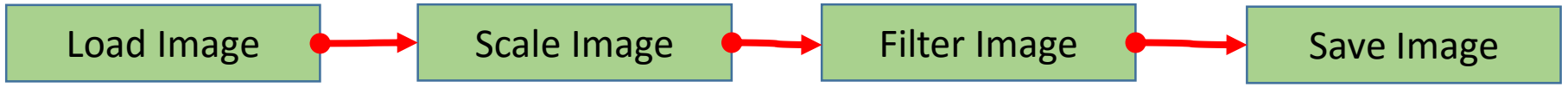
Конвейерная обработка (Pipeline)

- Вычисления производятся над набором элементов данных, каждый из которых проходит несколько стадий обработки – этапы/блоки конвейера
- Регулярный, односторонний, стабильный поток данных
- Подзадачи – применение операции "стадия N" к каждому элементу данных
- Примеры
  - Конвейерная обработка команд процессором
  - Обработка сигналов, фильтры, графика



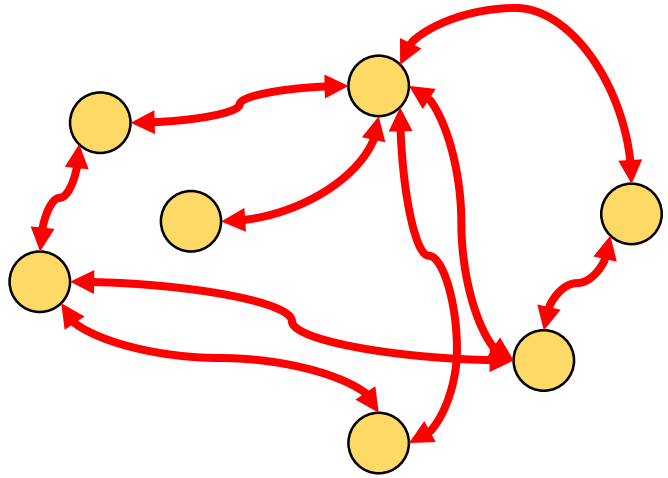
Конвейерная обработка (Pipeline)

- Параллелизм ограничен числом стадий
- В идеале времена работы каждой стадии должны быть одинаковыми
  - Самая медленная стадия становится узким местом
  - Комбинирование и декомпозиция стадий
  - Распараллеливание медленной стадии
- Времена заполнения и опустошения конвейера



Координация на основе событий

- Декомпозиция на слабосвязанные компоненты, взаимодействующие нерегулярным образом
- Двусторонние потоки данных
- Нерегулярные, непредсказуемые взаимодействия
- Высокий риск возникновения взаимной блокировки
- **Примеры**
- Моделирование с дискретными событиями
- Координация между заданиями в других шаблонах
- Распределенные системы

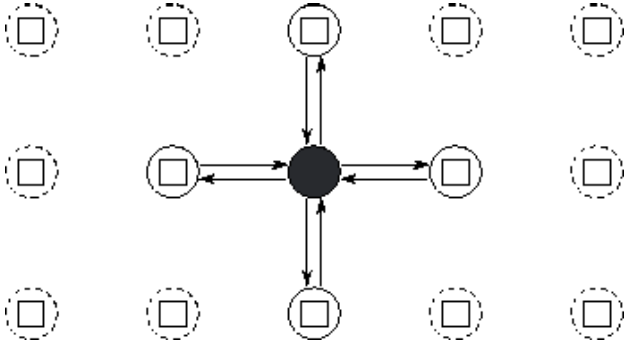


Граф взаимодействий подзадач может быть недетерминированным

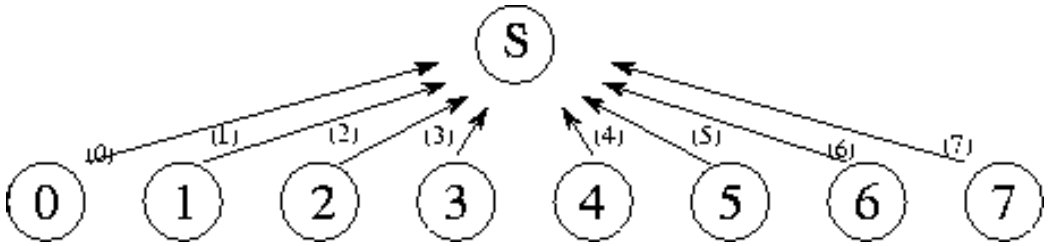
Взаимодействия между подзадачами

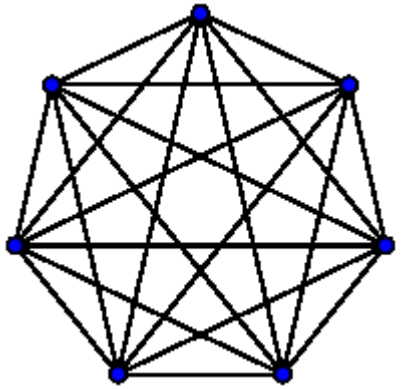
- Локальные и глобальные
- Структурированные и неструктурированные
- Статические и динамические
- Синхронные и асинхронные

Взаимодействия между подзадачами

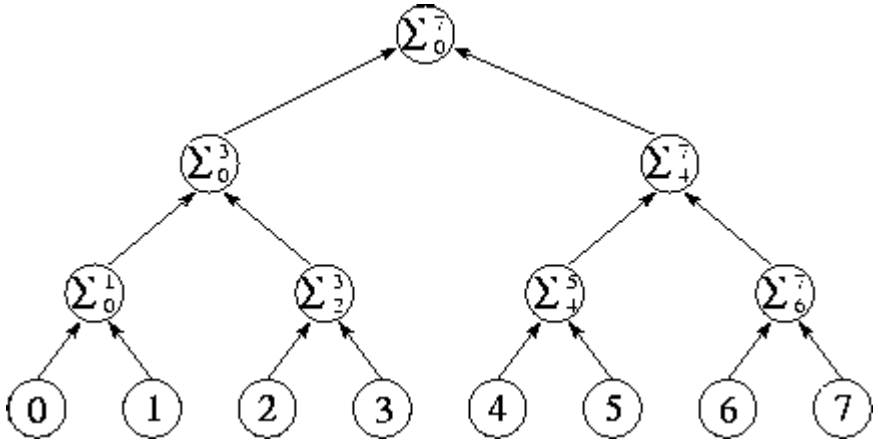


Локальные взаимодействия





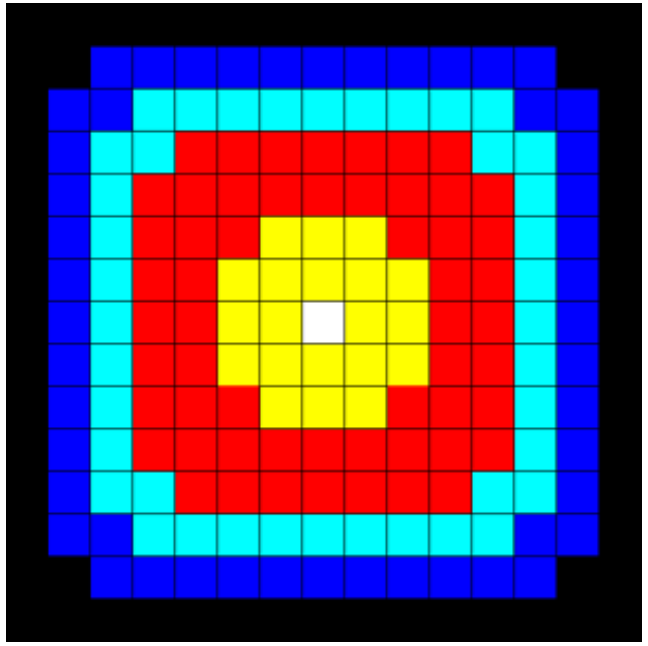
Коллективные операции
All-to-all

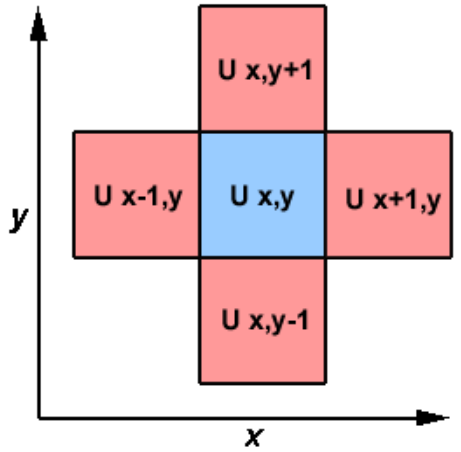


Коллективные (глобальные) операции
Reduction - сборка

Heat 2D (serial code)

- Уравнение теплопроводности описывает изменение температуры в заданной области с течением времени
- Приближенное решение можно найти методом конечных разностей
- Область покрывается сеткой
- Производные аппроксимируются конечными разностями
- Известна температура на границе области в начальный момент времени



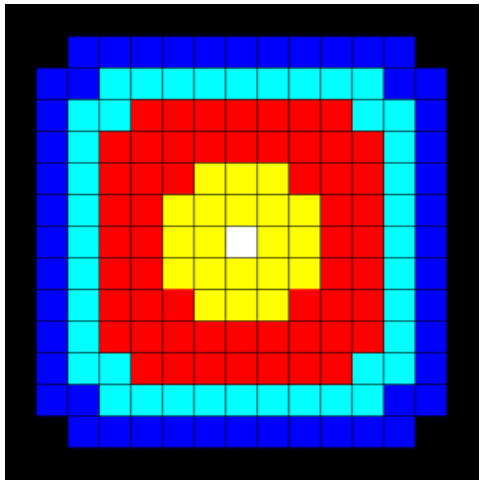


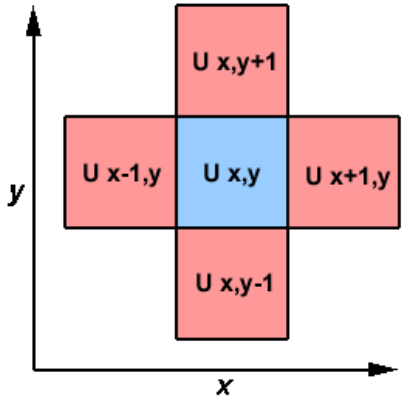
[*] Blaise Barney. Introduction to Parallel Computing (LLNL)

Heat 2D (serial code)

- Температура хранится в двумерном массиве – расчетная область

```
do iy = 2, ny - 1
  do ix = 2, nx - 1
    u2(ix, iy) = u1(ix, iy) +
                 cx * (u1(ix + 1, iy) +
                 u1(ix - 1, iy) – 2.0 * u1(ix, iy)) +
                 cy * (u1(ix, iy + 1) +
                 u1(ix, iy - 1) – 2.0 * u1(ix, iy))
  end do
end do
```



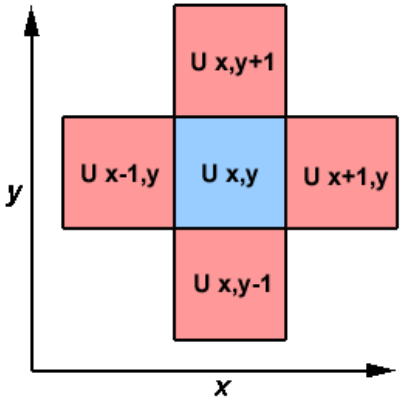


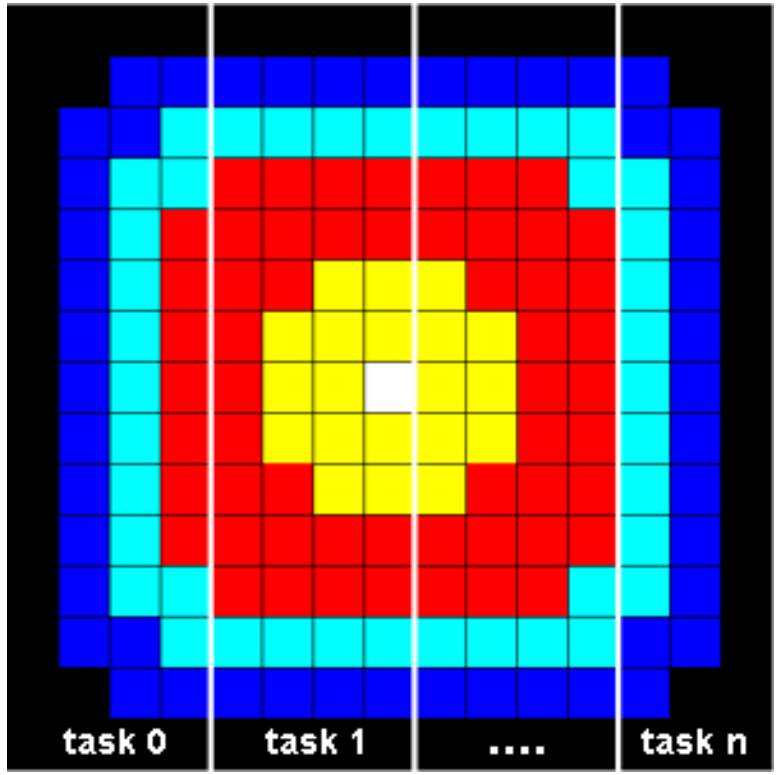
[*] Blaise Barney. Introduction to Parallel Computing (LLNL)

Heat 2D (serial code)

- Каждый процесс будет обрабатывать свою часть области – **1D domain decomposition**

```
for t = 1 to nsteps
  1. Update time
  2. Send neighbors my border
  3. Receive from neighbors
  4. Update my cells
end for
```





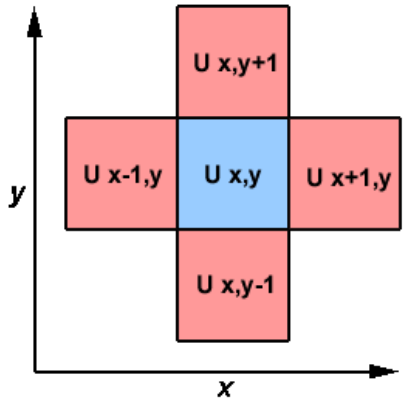
Расчетная область разбита на вертикальные полосы – массив распределен

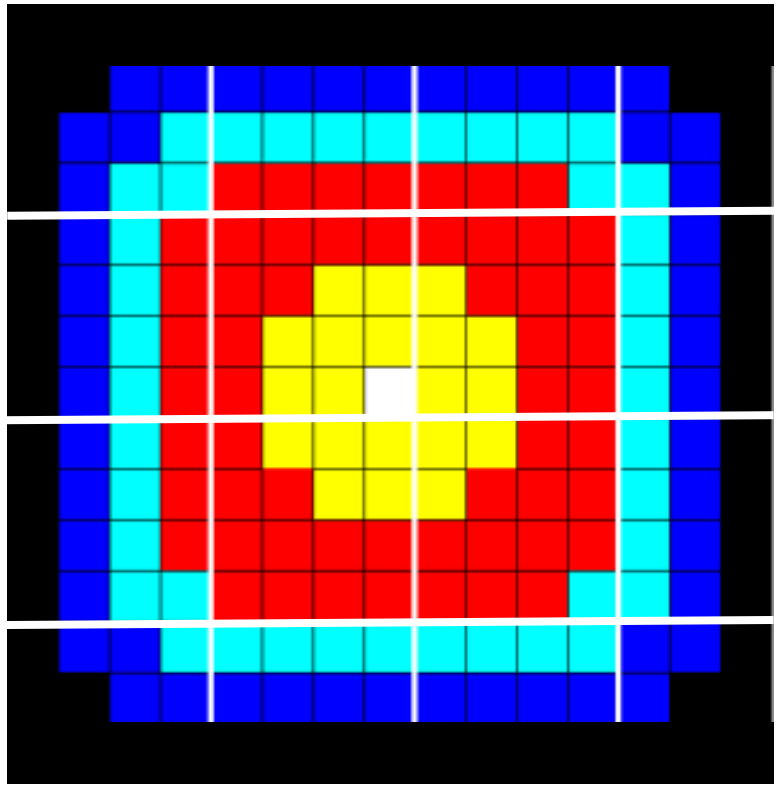
[*] Blaise Barney. Introduction to Parallel Computing (LLNL)

Heat 2D (serial code)

- Каждый процесс будет обрабатывать свою часть области – **2D domain decomposition**

```
for t = 1 to nsteps
  1. Update time
  2. Send neighbors my border
  3. Receive from neighbors
  4. Update my cells
end for
```





Расчетная область разбита на прямоугольные области (2D)

Сколько надо выполнить обменов на каждом шаге?

[*] Blaise Barney. Introduction to Parallel Computing (LLNL)

Литература

- Эндрюс Г. **Основы многопоточного, параллельного и распределенного программирования**. – М.: Вильямс, 2003.

- Foster I. **Designing and Building Parallel Programs: Concepts and Tools for Software Engineering** – http://www.mcs.anl.gov/~itf/dbpp/

- Herb Sutter. **Welcome to the Jungle** – http://herbsutter.com/welcome-to-the-jungle/