Лекция 1
Распределенные вычислительные системы

Курносов Михаил Георгиевич

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

Курс «Параллельное программирование»
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)
Осенний семестр, 2018

Классификация архитектур вычислительных систем
(по числу потоков команд и данных)

Классификация М. Флинна (M. J. Flynn, 1966)		
	Single instruction stream	**Multiple instruction stream**
Single data stream	SISD	MISD
Multiple data stream	SIMD	MIMD

- **SISD** – последовательная ВС; одно устройство управления работает с одним потоком инструкций в памяти, выполняя их на последовательном процессоре (работает с одним потоком данных): первые процессоры
- **SIMD** – вычислительная систем, в которой множество процессоров выполняют одну инструкцию над своими локальными данными: векторные ВС Cray, NEC; наборы векторных инструкций AVX, AltiVec, NEON SIMD; GPU
- **MISD** – вычислительная система типа “много потоков команд – один поток данных”: конвейерные ВС (частично) и систолические ВС (systolic arrays, частично)
- **MIMD** – совокупность процессорных элементов, работающих со своими локальными потоками команд и данных: вычислительные кластеры, MPP-системы

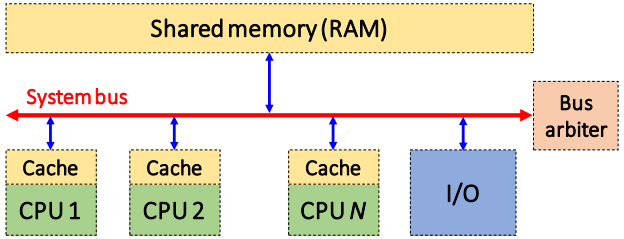
К какому классу можно отнести процессор Intel Xeon Core i5 6200U (Skylake)?

Классификация архитектур вычислительных систем

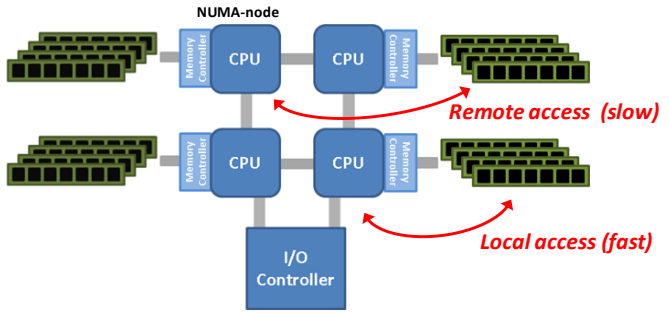
(структурно-функциональная – по способу организации оперативной памяти)

Класс 1. Системы с разделяемой процессорами оперативной памятью (shared memory systems)

- **Симметричные мультипроцессоры** (symmetric multiprocessor, SMP) – множество процессоров имеют одинаковые возможности по доступу к разделяемой оперативной памяти и функционируют под управлением одной операционной системы
  - ✓ Относительно простое создание параллельных программ (POSIX threads, OpenMP, …)
  - ✓ Контроллер памяти – узкое место, число процессоров <= 32



- **NUMA-системы** (non-uniform memory architecture) – множество процессоров имеют *неодинаковые* возможности по доступу к разделяемой оперативной памяти и функционируют под управлением одной операционной системы
  - ✓ Относительно простое создание параллельных программ (POSIX threads, OpenMP, libnuma, thread affinity, …)
  - ✓ Контроллер памяти и внутрисистемная шина (Intel QPI, HyperTransport) – узкое место, число процессоров <= 128

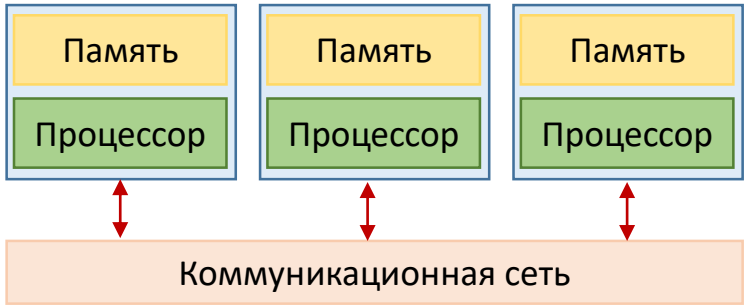


Классификация архитектур вычислительных систем

(структурно-функциональная – по способу организации оперативной памяти)

Класс 2. Системы с распределенной оперативной памятью (distributed memory systems)

- **Распределенная вычислительная система** – совокупность вычислительных узлов (элементарных машин, процессорных элементов), взаимодействующих через коммуникационную сеть (среду); каждый узел имеет свою оперативную память и функционирует под управлением свой операционной системы



- ✓ **Вычислительный кластер** (computer cluster) – распределенная ВС, построенная на базе *серийно* выпускаемого промышленностью оборудования
- ✓ **Массово параллельная система** (massively parallel system, MPP-system) – большемасштабная распределенная ВС; как правило, MPP-системы строятся на базе проприетарного (фирменного) оборудования и значительно эффективнее кластерных ВС (системы IBM BlueGene, Cray XK/XC и др.)

К какому классу можно отнести ноутбук на базе двухъядерного процессора Intel Xeon Core i5 6200U?

К какому классу можно отнести два связанных в сеть ноутбука на базе процессора Intel Xeon Core i5 6200U?

Рейтинги мощнейших вычислительных систем

- **Суперкомпьютер (суперВС, supercomputer)** – вычислительная система, обладающая рекордными для текущего уровня развития вычислительной техники, показателями производительности и/или надежности, технико-экономической эффективности

- www.top500.org – решение системы линейных алгебраических уравнений методом LU-факторизации (High-Performance Linpack, FLOPS – Floating-point Operations Per Seconds)
- www.graph500.org – алгоритмы на графах (построение графа, обход в ширину, TEPS – Traversed Edges Per Second)
- www.green500.org – главный критерий – энергоэффективность (объем потребляемой электроэнергии, kW)
- http://top50.supercomputers.ru – рейтинг мощнейших вычислительных систем СНГ (тест High-Performance Linpack)

- **Как создать свой тест производительности?**

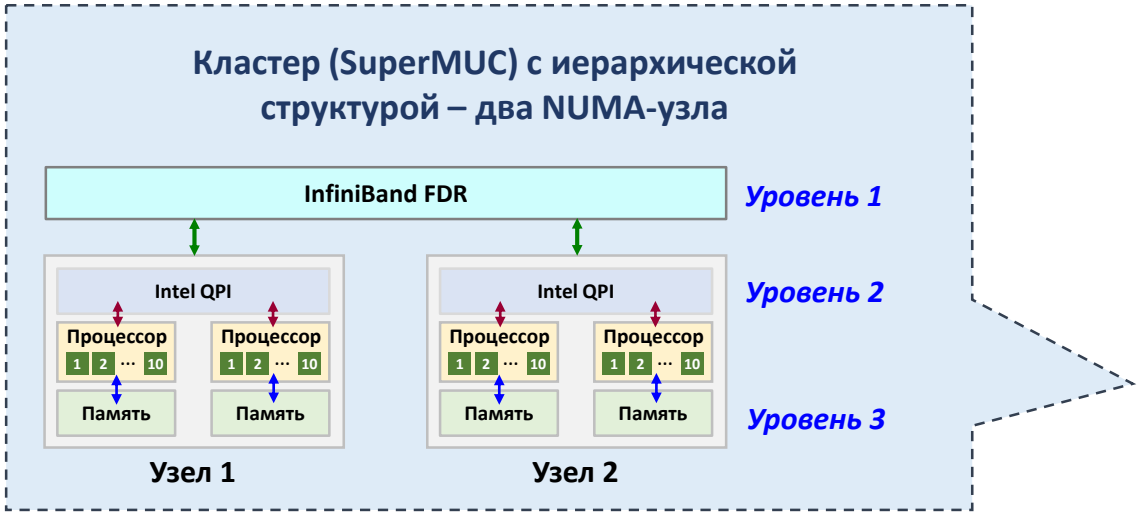
Top500 (июнь 2016)

	NAME	SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	Sunway TaihuLight	Shenwei SW26010 (260C 1.45 GHz) Custom interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
2	Tianhe-2 (Milkyway-2)	Intel Ivy Bridge (12C 2.2 GHz) & Xeon Phi (57C 1.1 GHz), Custom interconnect	NSCC in Guangzhou	China	3,120,000	33.9	17.8
3	Titan	Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect	DOE/SC/ORNL	USA	560,640	17.6	8.2
4	Sequoia	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	17.2	7.9
5	K computer	Fujitsu SPARC64 VIIIfx (8C 2.0 GHz), Custom interconnect	RIKEN AICS	Japan	705,024	10.5	12.7

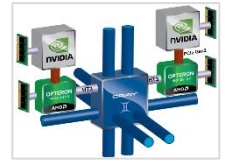
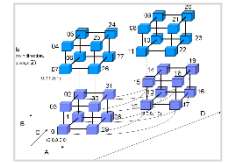
- Среднее количество вычислительных ядер в системе: **82 030**
- Среднее количество ядер на сокет (процессор): **11.4** (4, 6, 8, 10, 12, 14, 16, 32)
- Среднее энергопотребление: **1010.6** kW
- Коммуникационная сеть: InfiniBand (40.8%, суммарная производительность – 29.2%),
  10 Gigabit Ethernet (35.4%, суммарная производительность – 14.8%), Custom (14%, суммарная производительность – 50.8%)
- Процессоры: Intel (> 80%), IBM Power, AMD Opteron, SPARC64, ShenWei, NEC
- Ускорители (18% систем): Intel Xeon Phi, NVIDIA GPU, ATI/AMD GPU, PESY-SC
- Операционная система: GNU/Linux (497 систем), UNIX (3 системы)

Архитектурные свойства современных ВС

- Иерархическая организация коммуникационной среды
- Мультиархитектура вычислительных узлов
- Большемасштабность



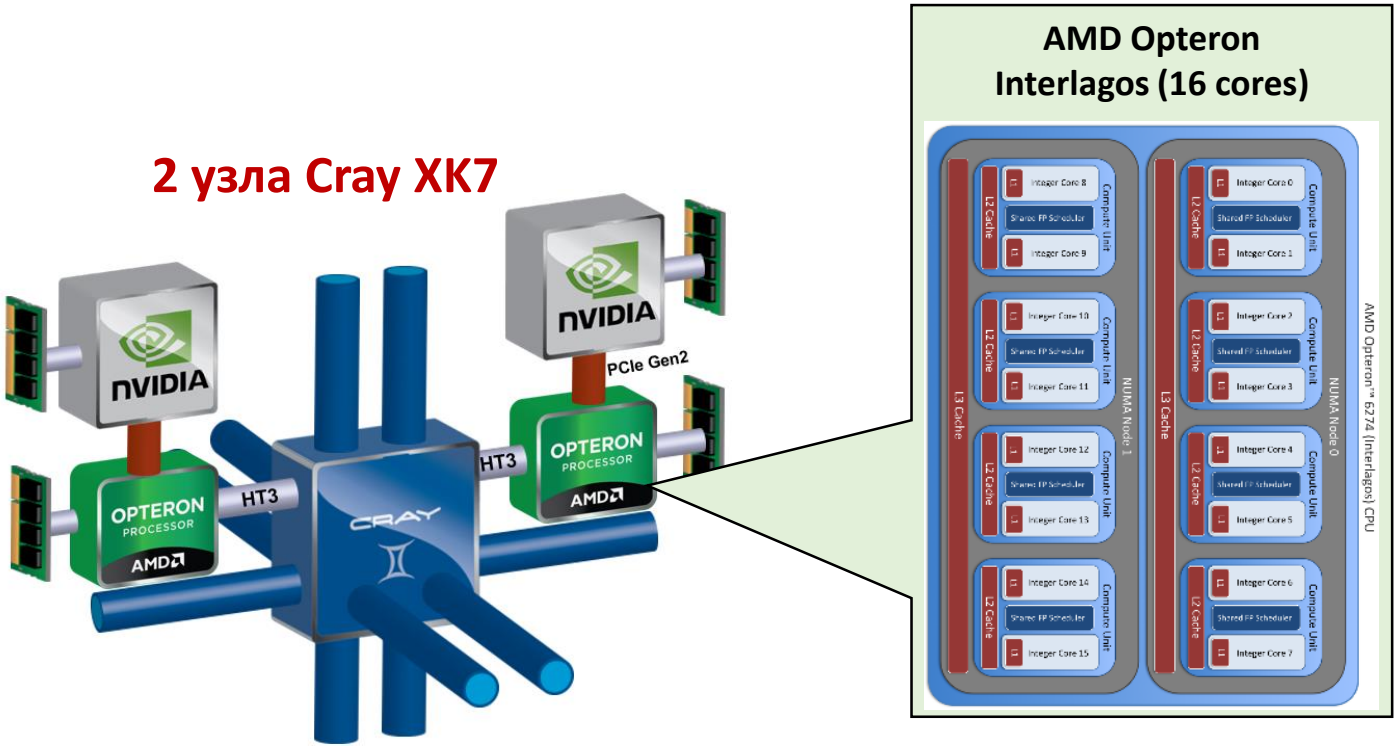
Системы Top500 (#46, 2015): 2, 3 уровня иерархии

№	Система	Коммуникационная среда		
		Уровень 1	Уровень 2	Уровень 3
1	**Tianhe-2** MilkWay-2 3 120 000 ядер	**TH Express-2** *fat tree* 16 000 узлов	**Intel QPI** 2 x Intel Xeon 3 x Xeon Phi	**Общая память** **DDR3** 16 ядер Intel Xeon
2	**Titan** Cray XK7 560 640 ядер	**Cray Gemini** *3D-mop* 18 688 узлов	**Общая память** **DDR3** 16 ядер AMD Opteron	
3	**Sequoia** IBM BlueGene/Q 1 572 864 ядер	***5D-mop*** 98 304 узлов	**Общая память** **DDR3** 16 ядер IBM PowerPC A2	
8	**Hazel Hen** Cray XC40 185 088 ядер	**Cray Aries** *Dragonfly* 7 712 узлов	**Intel QPI** 2 x Intel Xeon (NUMA-узел)	**Общая память** **DDR4** 12 ядер Intel Xeon
23	**SuperMUC** кластер 147 456 ядер	**InfiniBand FDR** *fat tree* 3 072 узлов	**Intel QPI** 2 x Intel Xeon (NUMA-узел)	**Общая память** **DDR** 10 ядер Intel Xeon

Система Cray XK7 Titan (#3 Top500, июнь 2016)

- **Titan Cray XK7** (MPP-система, https://www.olcf.ornl.gov/titan)
  - вычислительные узлы: 18 688 (NUMA – 2 AMD Opteron, 560 640 ядер)
  - коммуникационная сеть: Cray Gemini (3*D*-тор)
  - гибридная ВС: x86-64 AMD Opteron + NVIDIA GPU

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

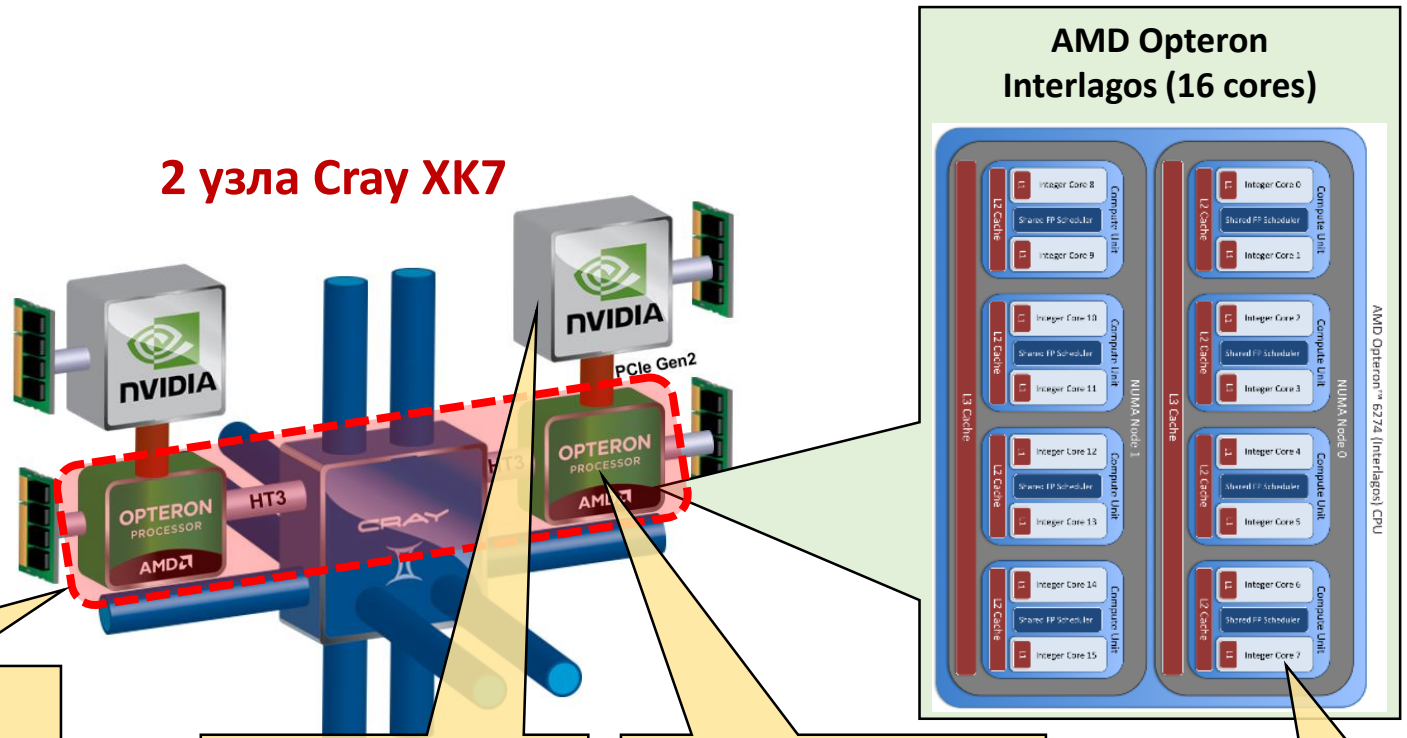


Cray Gemini (3*D*-тор)

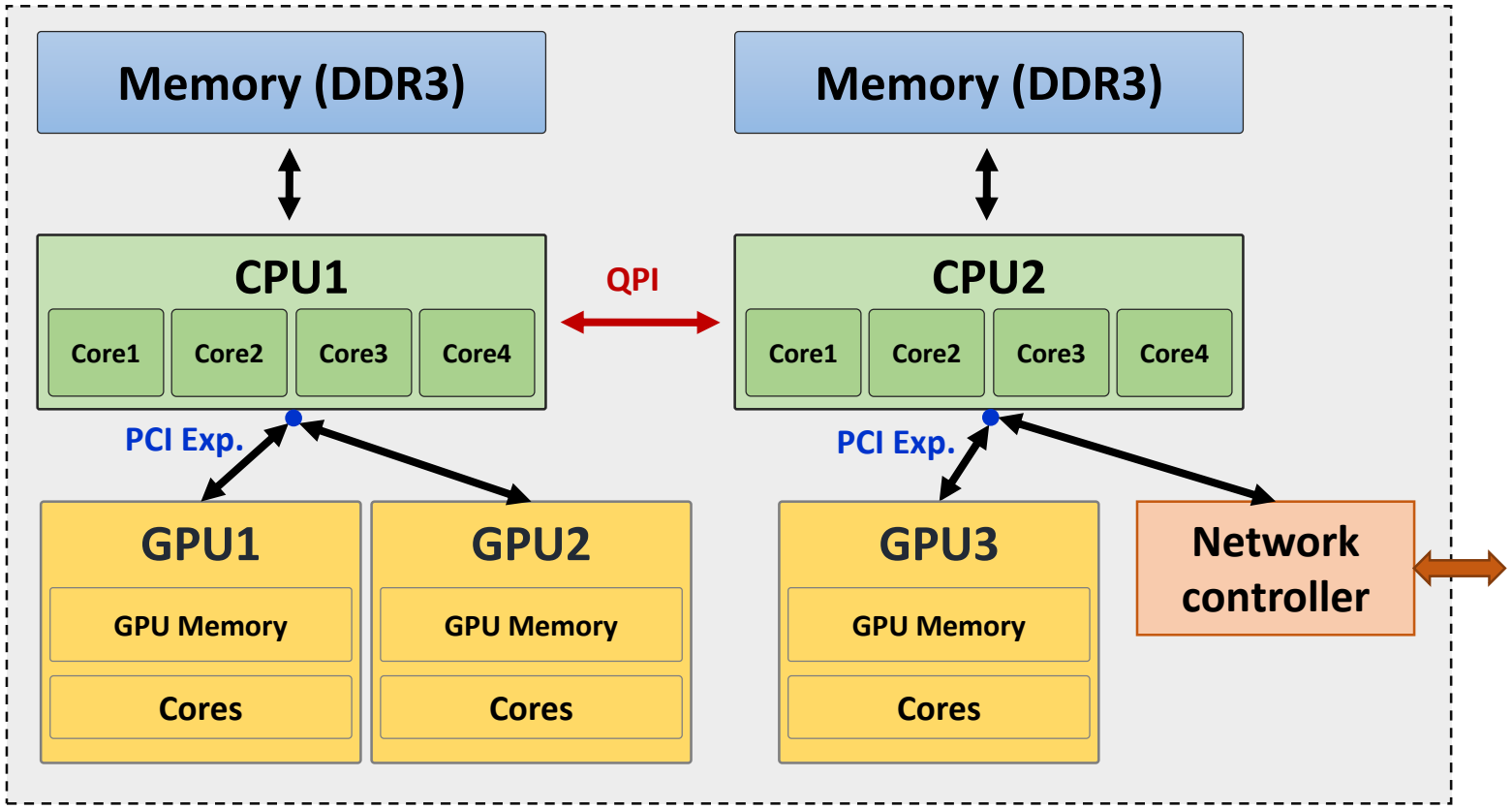
Система Cray XK7 Titan (#3 Top500, июнь 2016)

- **Titan Cray XK7** (MPP-система, https://www.olcf.ornl.gov/titan)
  - вычислительные узлы: 18 688 (NUMA – 2 AMD Opteron, 560 640 ядер)
  - коммуникационная сеть: Cray Gemini (3*D*-тор)
  - гибридная ВС: x86-64 AMD Opteron + NVIDIA GPU

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



Гибридные вычислительные узлы и ВС



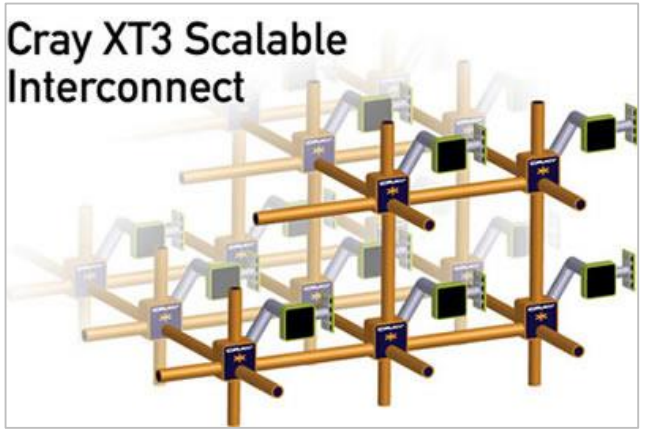
- **NUMA-узлы** –
  время доступа к памяти зависит от ее размещения в системе (NUMA-node)
- **Ускорители подключаются по шине PCI Express**
  - ❑ GPU (NVIDIA, AMD)
  - ❑ Intel Xeon Phi
  - ❑ FPGA-based accelerators

Коммуникационные сети ВС

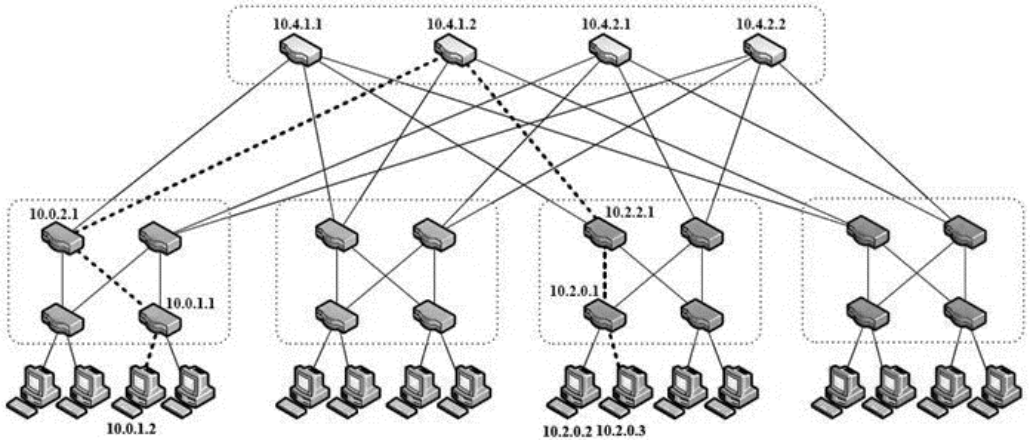
- **Задачи коммуникационных сетей ВС (communication network, interconnect)**
  - Реализация обменов информацией между ветвями параллельных программ:
    односторонние обмены (one-sided, RDMA: put/get),
    двусторонние (индивидуальные, дифференцированные, point-to-point: send/recv),
    коллективные операции (collectives: one-to-all broadcast, all-to-one gather/reduce, all-to-all)
  - Реализация обменов служебной информацией: контроль и диагностика состояния вычислительных узлов, барьерная синхронизация
  - Функционирования сетевых и параллельных файловых систем (доступ к дисковым массивам)
- **Требования к коммуникационной сети**
  - Высокая производительность реализации всех видов обменов (двусторонних, коллективных) – адекватность структуры ВС широкому классу параллельных алгоритмов
  - Масштабируемость (простое увеличение и уменьшение числа ЭМ в системе)
  - Живучесть и отказоустойчивость (функционирование при отказах отдельных подсистем)
  - Высокая технико-экономическая эффективность (цена/эффективность)

Виды коммуникационных сетей ВС

- **С фиксированной структурой межмашинных связей (direct network)**
  - ❑ Каждый вычислительный узел имеет *сетевой интерфейс* (системное устройство, маршрутизатор) с несколькими портами, через который он *напрямую* соединён с другими узлами



- **С динамической структурой (indirect network, switch-based) – на базе коммутаторов**
  - ❑ Каждый вычислительный узел имеет сетевой интерфейс с несколькими портами
  - ❑ Порты интерфейсов подключены к *коммутаторам* (switches), через которые происходит взаимодействие узлов

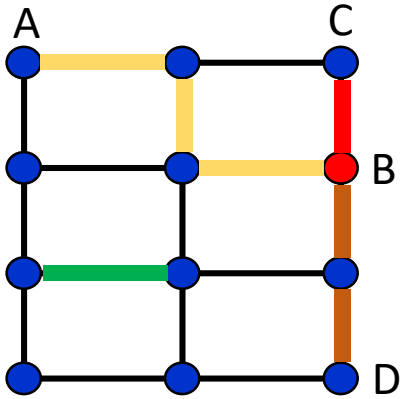


Выбор структуры коммуникационной сети (топологии)

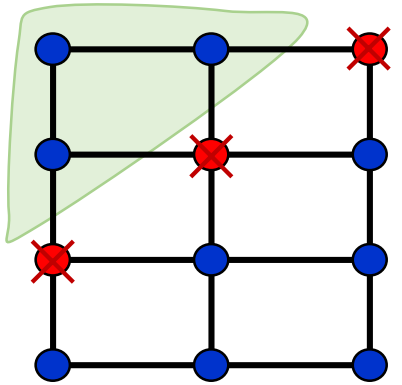
- **Структура ВС** (структура коммуникационной сети, topology) – граф, в котором вершинам соответствуют вычислительные узлы, а ребрам – межмашинные связи

- **Требования к структуре ВС (графу)**

- Минимизация времени выполнения межмашинных обменов и максимизация числа возможных одновременных обменов



- Максимизация вероятности сохранения связности структуры ВС при отказах ЭМ (вершин) и каналов связи (ребер)

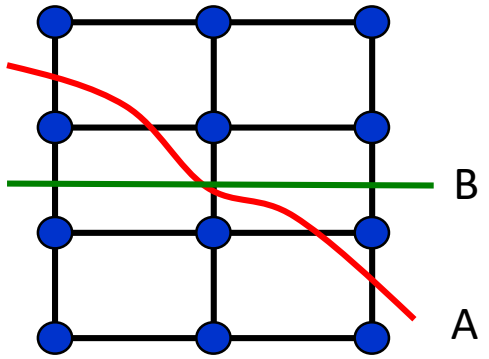


Образовались две компоненты связности

Показатели эффективности структуры ВС

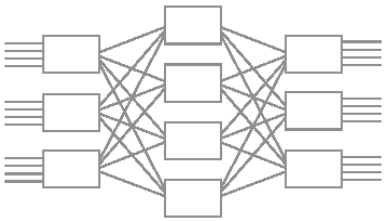
- **Диаметр графа** – длина максимального из кратчайших путей в графе (характеристика числа транзитных передач между ЭМ, hops)
- **Средний диаметр графа** – математическое ожидание расстояния между вершинами при их равновероятном выборе
- **Вектор-функция структурной живучести**
- **Бисекционная пропускная способность** (bisection bandwidth) – суммарная пропускная способность каналов связи между двумя непересекающимися подмножествами машин системы (для худшего разбиения, минимальное значение)
- **Аппаратная сложность** – число простейших коммутаторов (2 x 2, *n* x *n*) и каналов связи, необходимых для построения составного коммутатора сети
- **Метрическая сложность** – максимальная длина линии связи, требуемая для реализации выбранной топологии в трехмерном пространстве

Диаметр $d = 5$



A) бисекционная пропускная способность 5
B) бисекционная пропускная способность 3

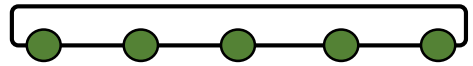
Коммутатор Клоза



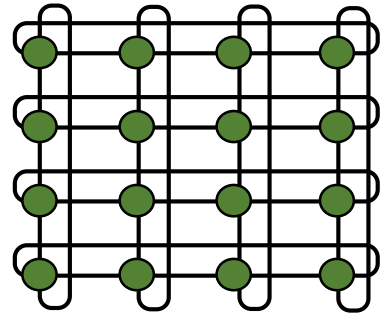
Структуры ВС с прямым соединением узлов

- **В *n*-мерной регулярной структуре каждая ЭМ связана с 2*n* соседями**
- **Тороидальные структуры**
  - ❑ Кольцо (1D-тор), тороидальная решетка (2D-тор), тороидальный куб (3D-тор)
  - ❑ Cray XK7 Titan (3D-тор), IBM BlueGene/Q (5D-тор), Fijitsu K Computer (6D-тор)
- **Гиперкубические структуры**
  - ❑ Линейка (1D-гиперкуб) , решетка (2D-гиперкуб), 3D-гиперкуб
  - ❑ Intel Paragon, ASCI Red (2D-куб), SGI Origin 2000 (3D-куб), МП-Х-Y (РФЯЦ-ВНИИЭФ)

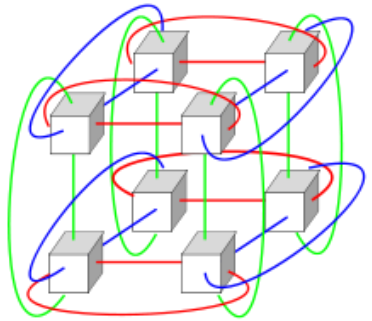
1*D*-тор

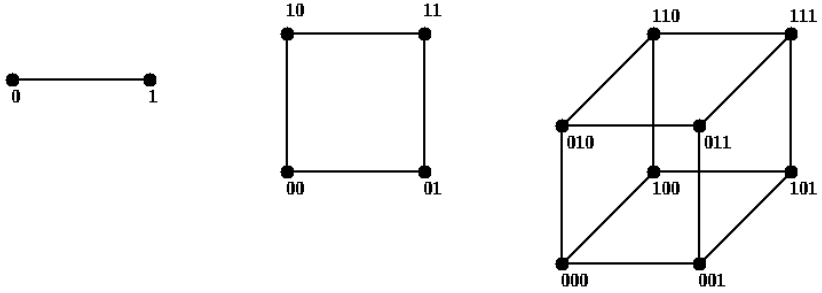


2*D*-тор

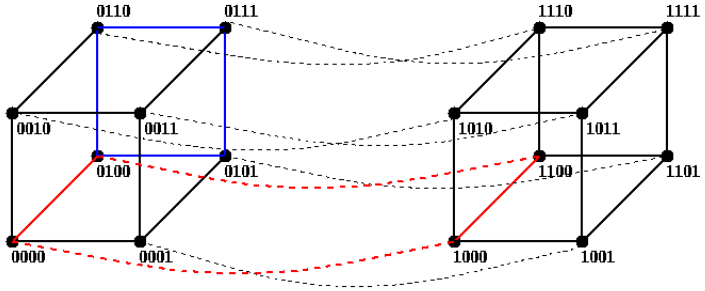


3*D*-тор





Гиперкубы: 1D, 2D, 3D



4D-гиперкуб

Структуры ВС с прямым соединением узлов (2)

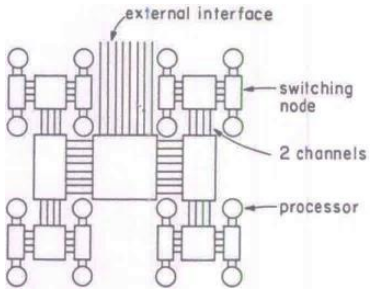
- Dragonfly
- HyperX/Hamming Graph
- D_n-графы, циркулянтные структуры (системы МИКРОС)
- Графы Кауца (Kautz network): система SiCortex SC5832 – 972 узла, диаметр 6, линков 2916
- Data Vortex Interconnect
- …

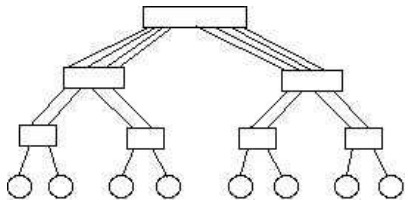
Структуры ВС на базе коммутаторов (indirect nets)

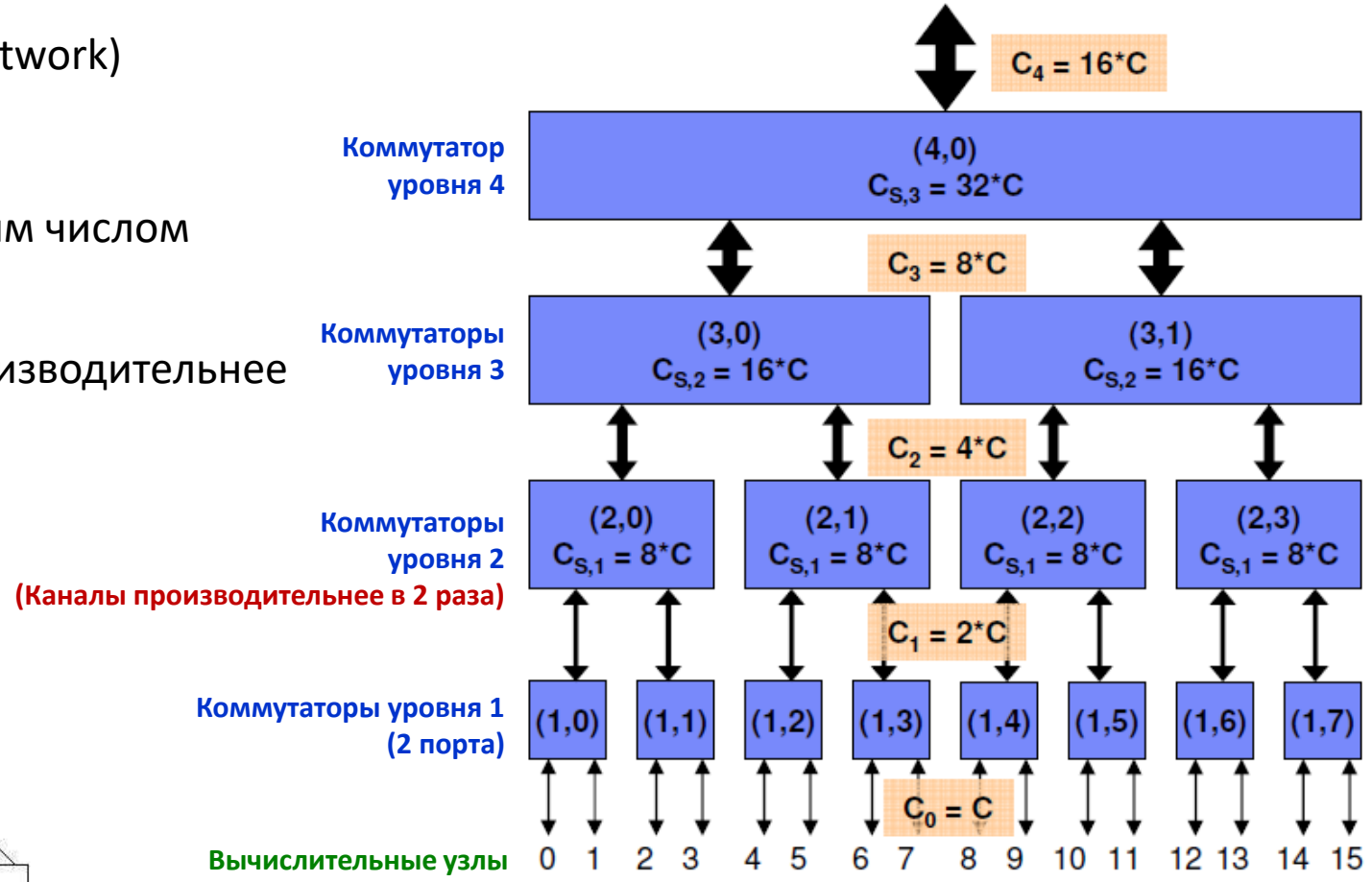
- **Деревья**
  - Толстое дерево (fat tree)
  - *k*-арные *n*-деревья (*k*-ary *n*-tree)
  - Extended generalized fat tree (XGFT)

Fat tree (толстое дерево)

- **Топология «толстое дерево» (fat tree)**
  Charles E. Leiserson. ***Fat-trees: universal networks for hardware-efficient supercomputing*** //
  IEEE Transactions on Computers, Vol. 34, No. 10, 1985
- Структура на базе коммутаторов (indirect network)
- Constant bisectional bandwidth (CBB)
- Сеть строится из коммутаторов с одинаковым числом *R* портов (линков, constant radix)
- Линки (каналы) коммутаторов уровня *i* производительнее линков коммутаторов уровня *i* – 1 в *R* раз (по числу портов)
- **Пример**: сети на базе InfiniBand, IBM RoadRunner

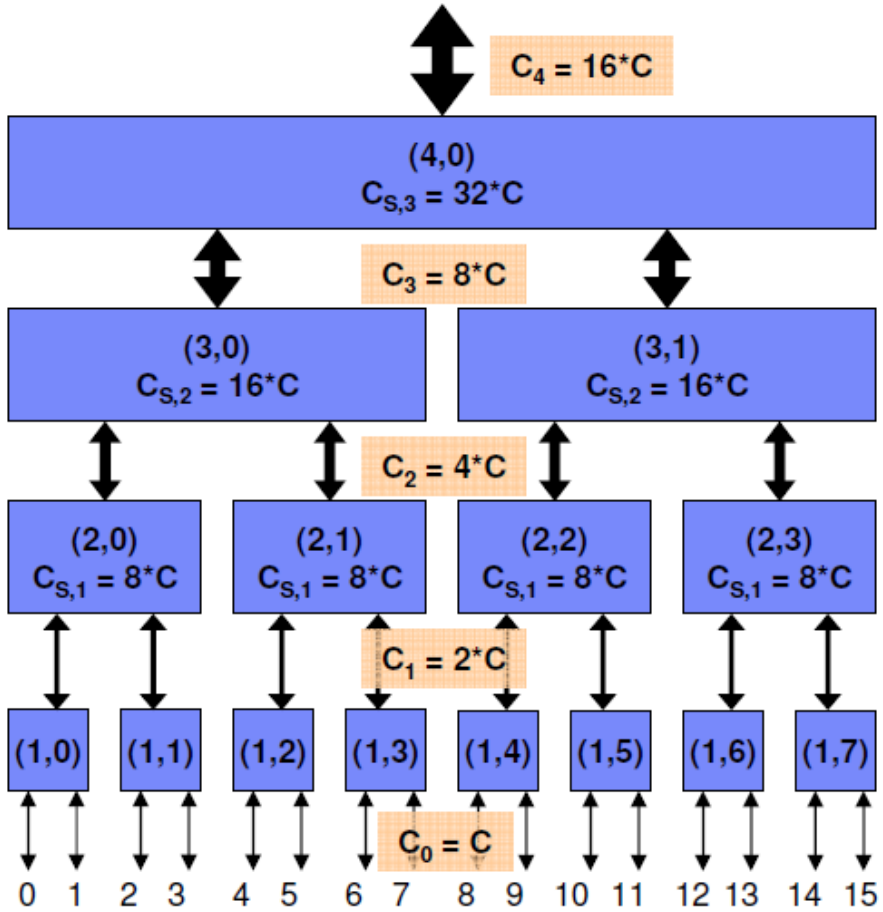






Fat tree (толстое дерево)

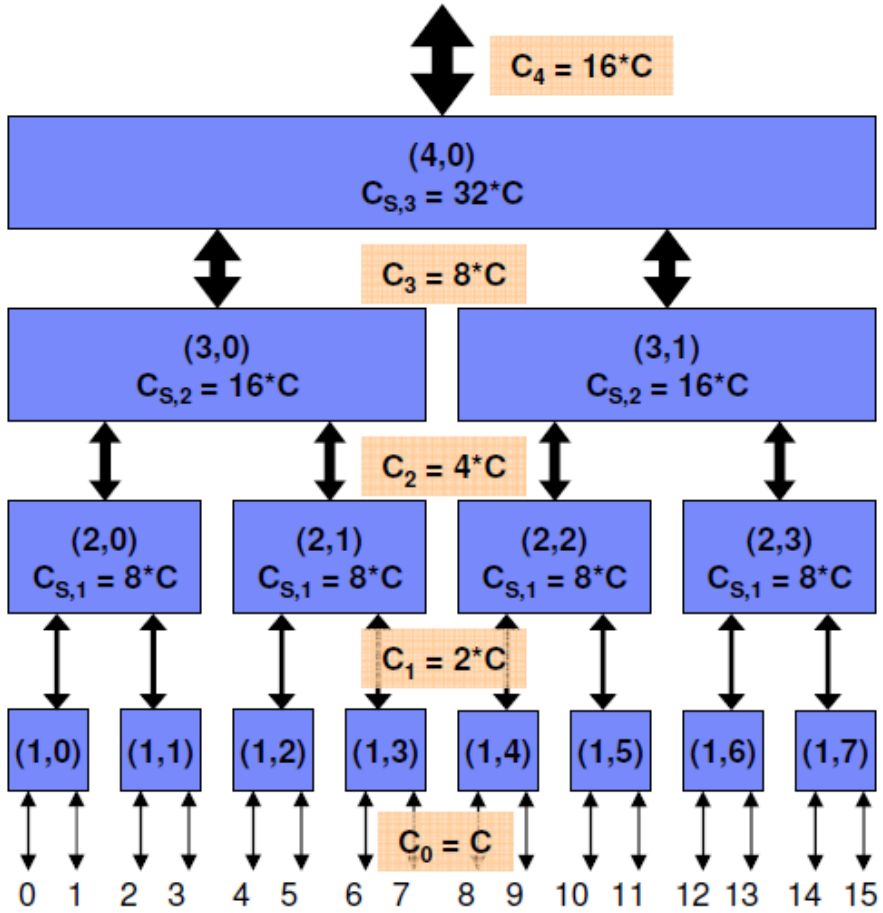
- Одинаковое число портов в коммутаторах
- Линки имеют **разную** производительность



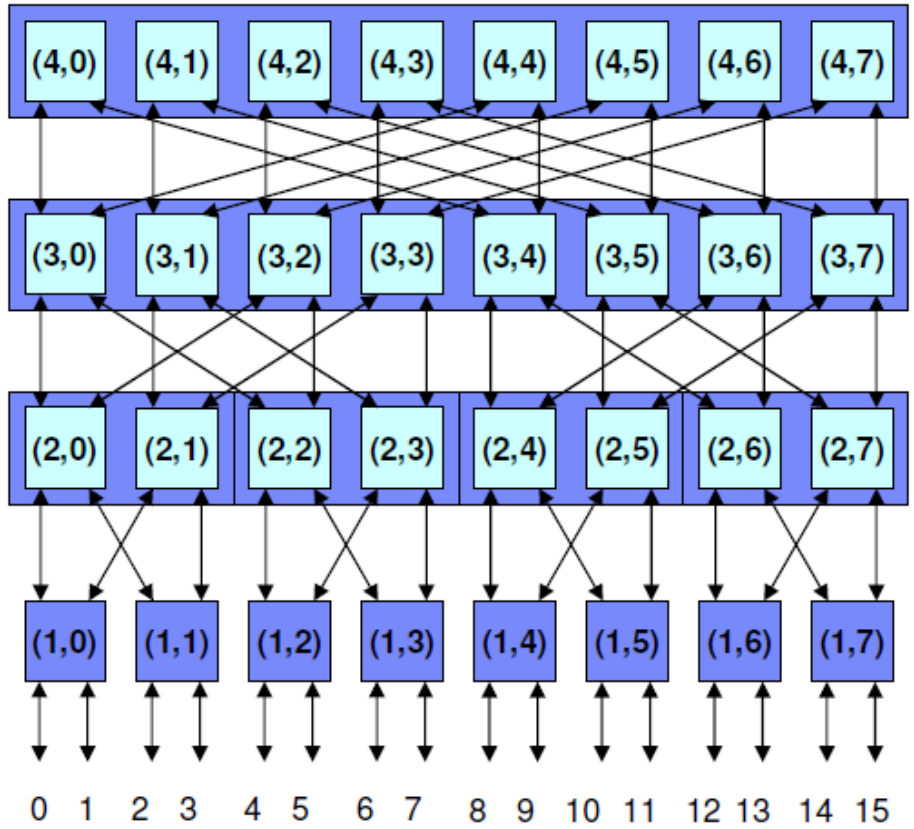
- Одинаковое число портов в коммутаторах
- Линки имеют **одинаковую** производительность

Fat tree (толстое дерево)

- Одинаковое число портов в коммутаторах
- Линки имеют **разную** производительность



- Одинаковое число портов в коммутаторах
- Линки имеют **одинаковую** производительность



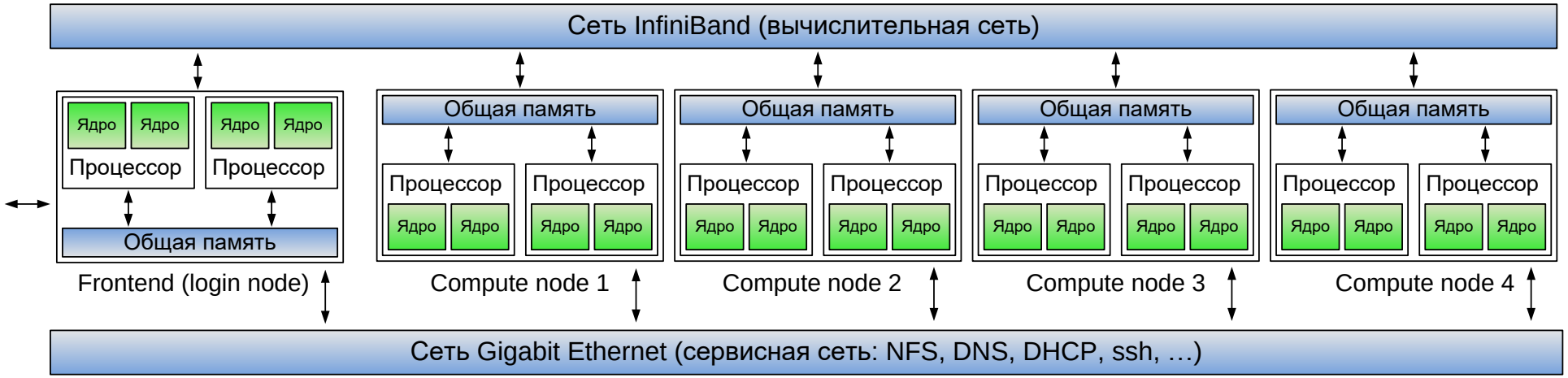
Что осталось “за кадром”

- **Выбор структуры для проблемно-ориентированной ВС**
  (для определенного класса задач)

- **Алгоритмы маршрутизации** (как доставить сообщение от узла A до узла B?
  Как учитывать загрузку каналов, отказы узлов и линков?)

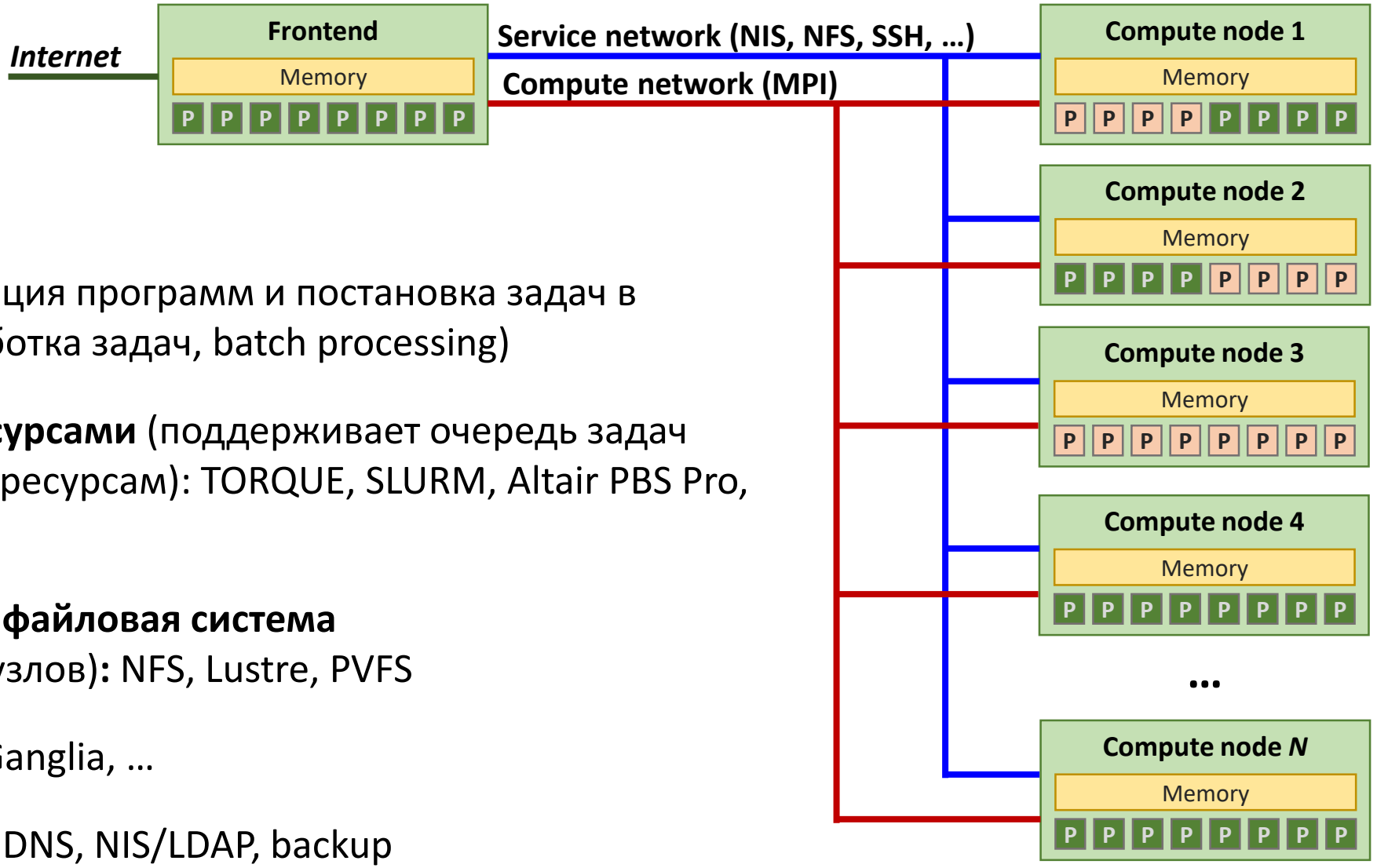
- **Вопросы технико-экономической эффективности**
  (учет длин кабелей, числа коммутаторов)

Вычислительные кластеры (computer cluster)



- **Вычислительные кластеры строятся на базе <u>свободно доступных компонентов</u>**
- Вычислительные узлы: 2/4-процессорные узлы, 1 – 8 GiB оперативной памяти на ядро (поток)
- Коммуникационная сеть (сервисная NFS/DNS/NIS и для обмена сообщениями MPI/SHMEM)
- 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

Программное обеспечение вычислительных кластеров



- **Узел Frontend** – компиляция программ и постановка задач в очередь (пакетная обработка задач, batch processing)
- **Система управления ресурсами** (поддерживает очередь задач и контролирует доступ к ресурсам): TORQUE, SLURM, Altair PBS Pro, IBM Load Leveler, …
- **Сетевая (параллельная) файловая система** (доступ к /home со всех узлов)**:** NFS, Lustre, PVFS
- **Система мониторинга:** Ganglia, …
- **Сетевые сервисы**: DHCP, DNS, NIS/LDAP, backup

Параллельные вычисления – введение

Разработка параллельного алгоритма

- **Поиск параллелизма в известном последовательном алгоритме, его модификация или создание нового алгоритма:** определения уровня распараллеливания – уровень инструкций (мелкозернистый параллелизм, fine grained), потоков/процессов (крупнозернистый параллелизм, coarse grained)

- Выбор класса целевой ВС: с общей или распределенной памятью

- Разработка алгоритма в терминах одной из моделей программирования целевой ВС:
  - Системы с общей памятью (SMP/NUMA): fork/join model, CSP, Actor model, передача сообщений
  - Системы с распределенной памятью (кластеры, MPP): явная передача сообщений (message passing: односторонние/двусторонние/коллективные обмены), BSP – Bulk synchronous parallel, MapReduce

- **Параллельная версия самого эффективного последовательного алгоритма решения задачи необязательно будет самой эффективной параллельной реализацией**

Реализация параллельного алгоритма (программы)

- Выбор инструментальных средств (MPI, OpenSHMEM; OpenMP, POSIX Threads, Cilk)
- 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 относительного ускорения (Rel. 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

Виды масштабируемости программ

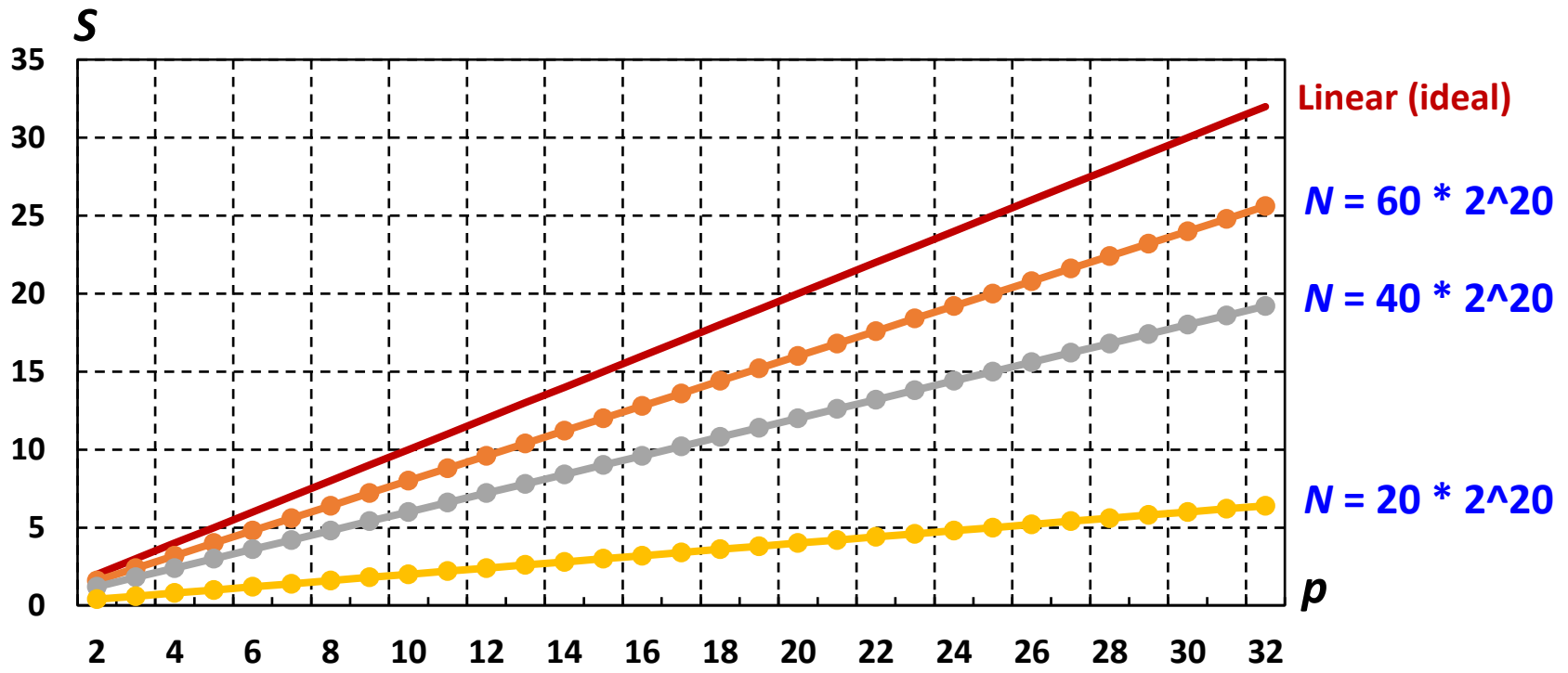
- **Масштабируемость параллельной программы** (scalability) – характеристика программы, показывающая как изменяются ее показатели производительности при варьировании числа параллельных процессов на конкретной ВС

- **Строгая/сильная масштабируемость** (strong scaling) – зависимость коэффициента ускорения от числа p процессов **при фиксированном размере *n*** входных данных (n = const)
  - Показывает как растут накладные расходы с увеличением p
  - Цель – минимизировать время решения задачи фиксированного размера

- **Слабая масштабируемость** (weak scaling) – зависимость коэффициента ускорения параллельной программы от числа процессов **при фиксированном размере входных данных на один процессор (**n / p = const**)**
  - Цель – решить задачу наибольшего размера на ВС

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

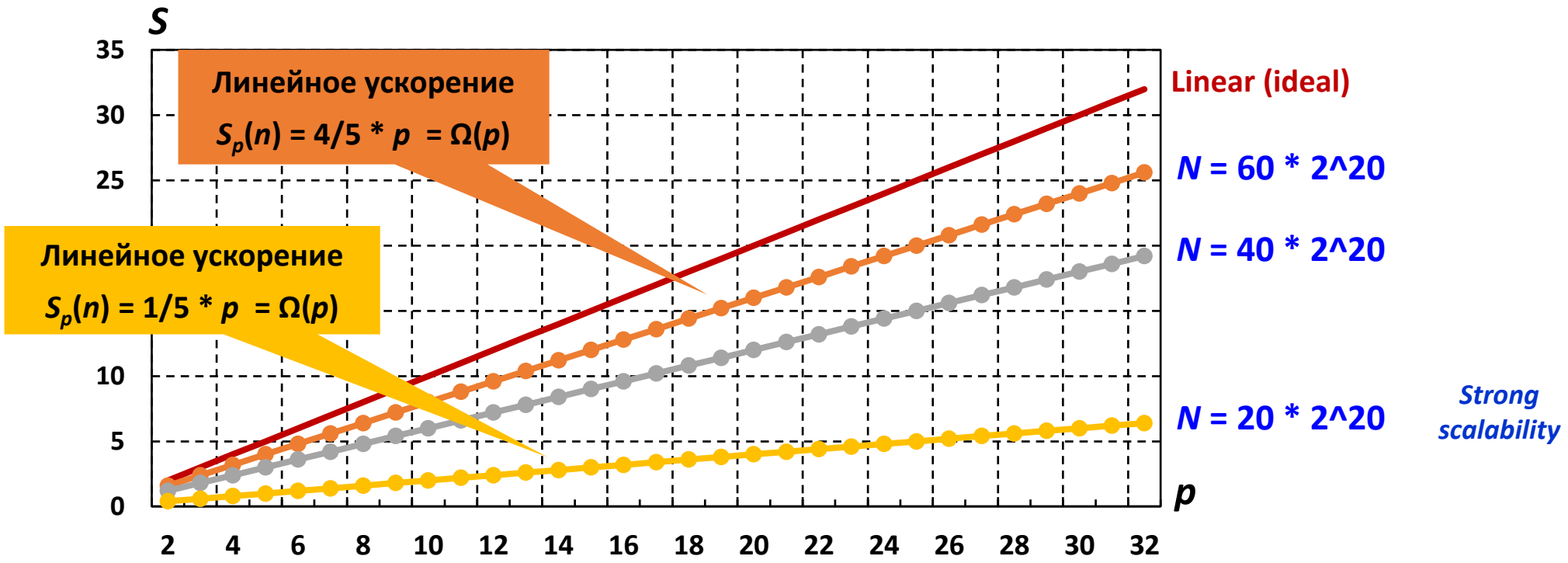
Коэффициент ускорения (Speedup)



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, …)

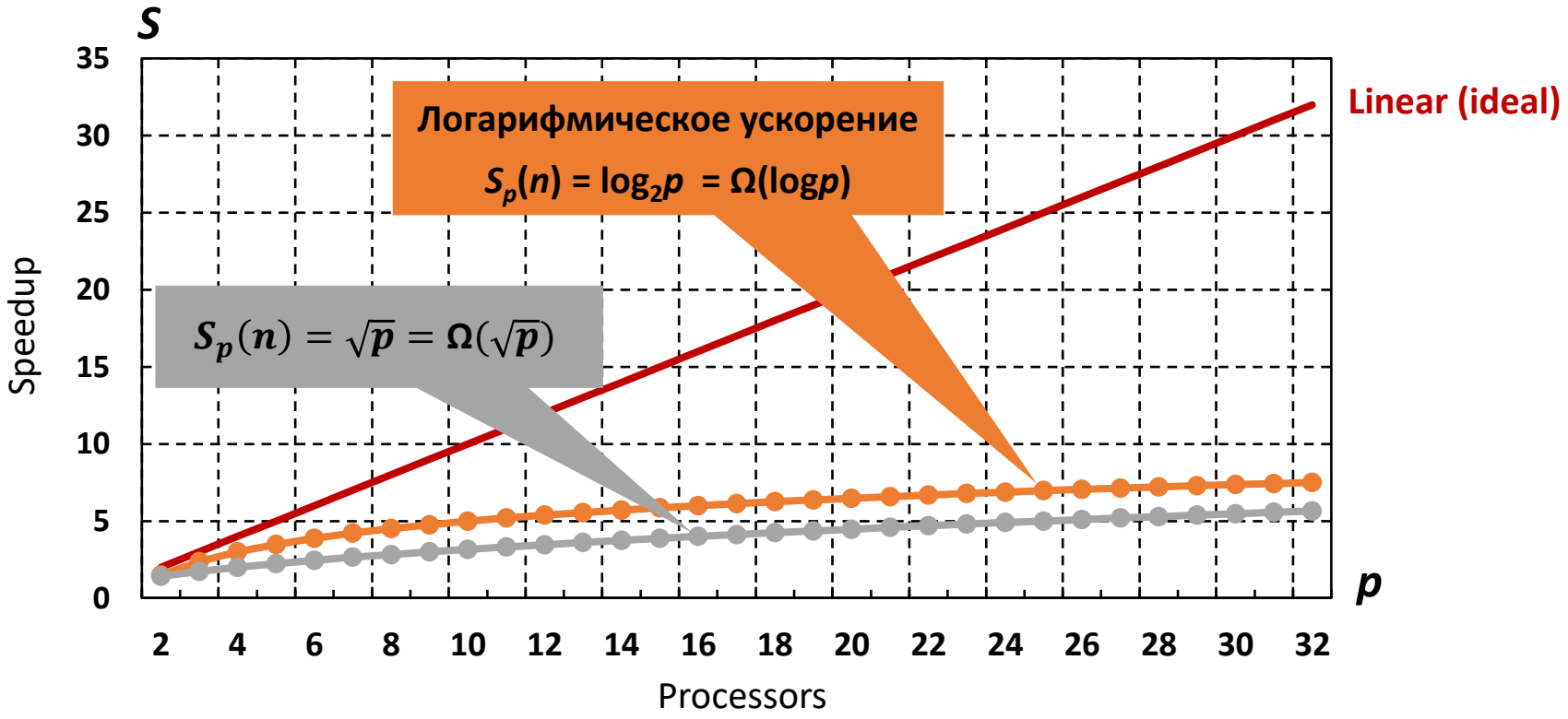
Коэффициент ускорения (Speedup)



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)



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

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

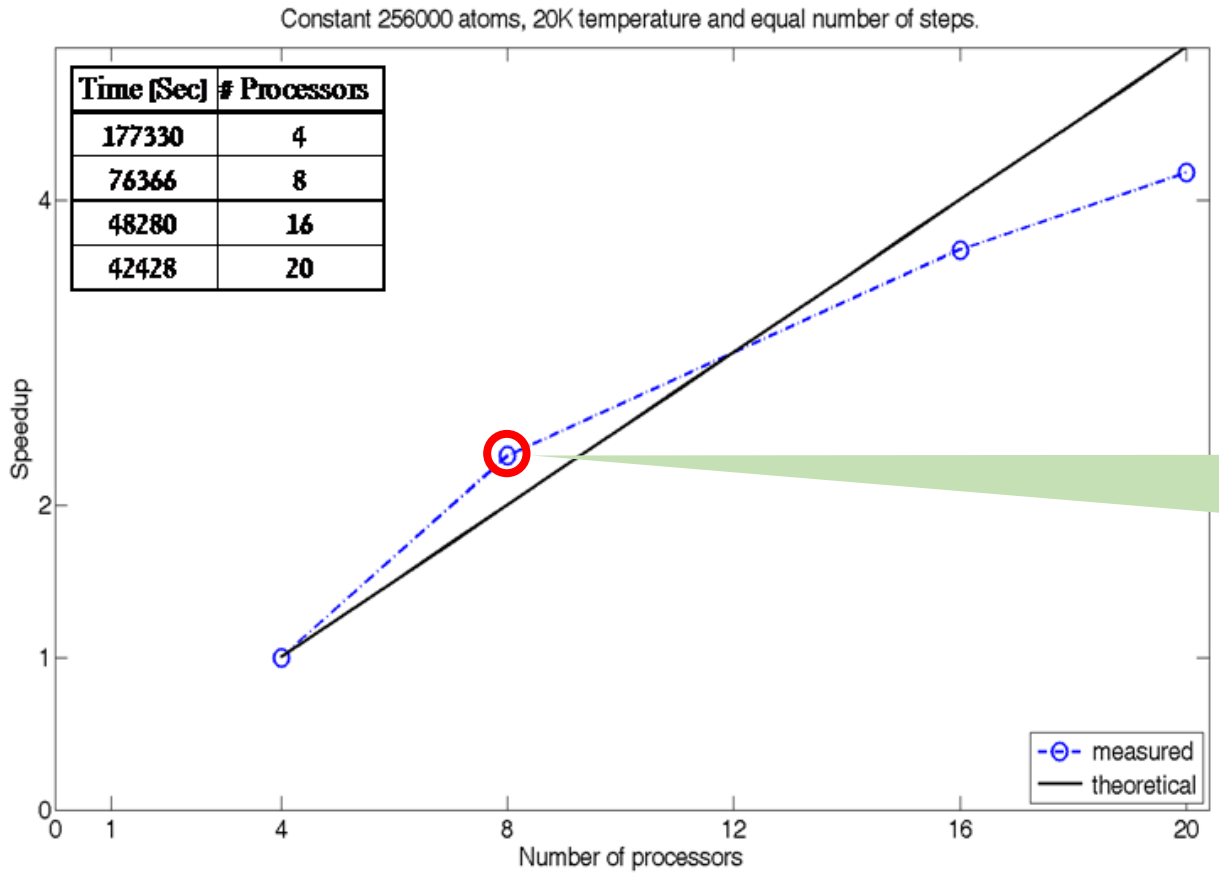
- Причина: иерархическая организация памяти:
  Cache – RAM – Local disk (HDD/SSD) – Network storage

- 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

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

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
- Коэффициент *V* вариации

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

- Отношение min/max

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

- Jain’s fairness index

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

Закон Дж. Амдала (Amdahl's law)

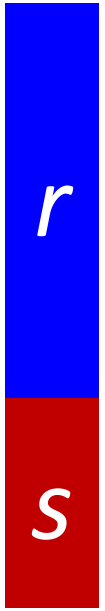
- Пусть имеется последовательная программа с временем выполнения $T(n)$
- Обозначим:
  - $\boldsymbol{r} \in [\mathbf{0}, \mathbf{1}]$ – часть программы, которая может быть распараллелена (perfectly parallelized)
  - $\boldsymbol{s} = \mathbf{1} - \boldsymbol{r}$ – часть программы, которая не может быть распараллелена (purely sequential)
- Время выполнения параллельной программы на p процессорах (время каждого потока) складывается из последовательной части s и параллельной r:

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

- Вычислим значение коэффициент ускорения (по определению)

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

- 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:
  - $\boldsymbol{r} \in [\mathbf{0}, \mathbf{1}]$ – часть программы, которая может быть распараллелена (perfectly parallelized)
  - $\boldsymbol{s} = \mathbf{1} - \boldsymbol{r}$ – часть программы, которая не может быть распараллелена (purely sequential)
- **Закон Дж. Амдала (Gene Amdahl, 1967) [1]:**

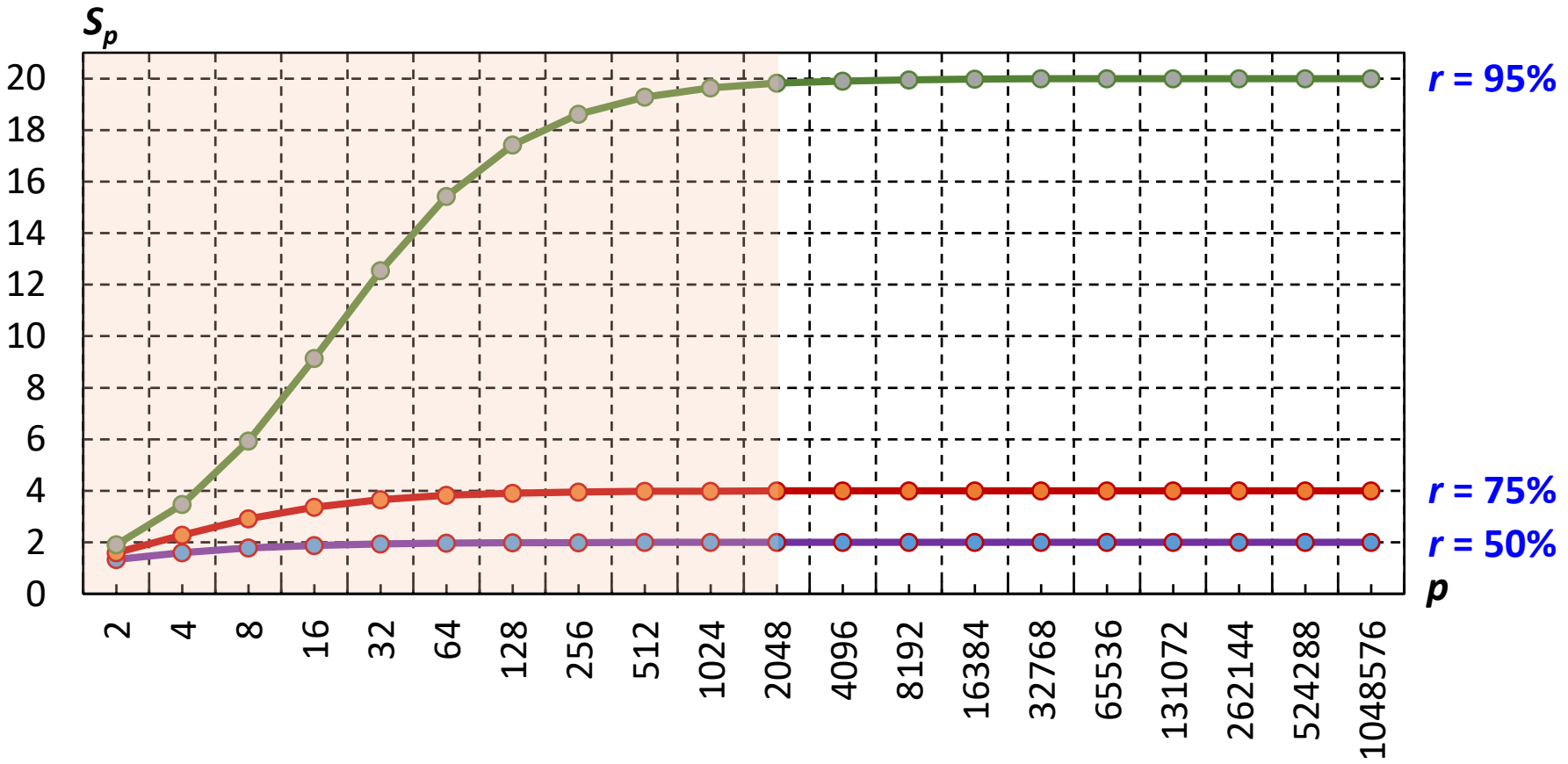
  Максимальное ускорение S_p программы на p процессорах равняется

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

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

❑ 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

Допущения закона Дж. Амдала (Amdahl's law)

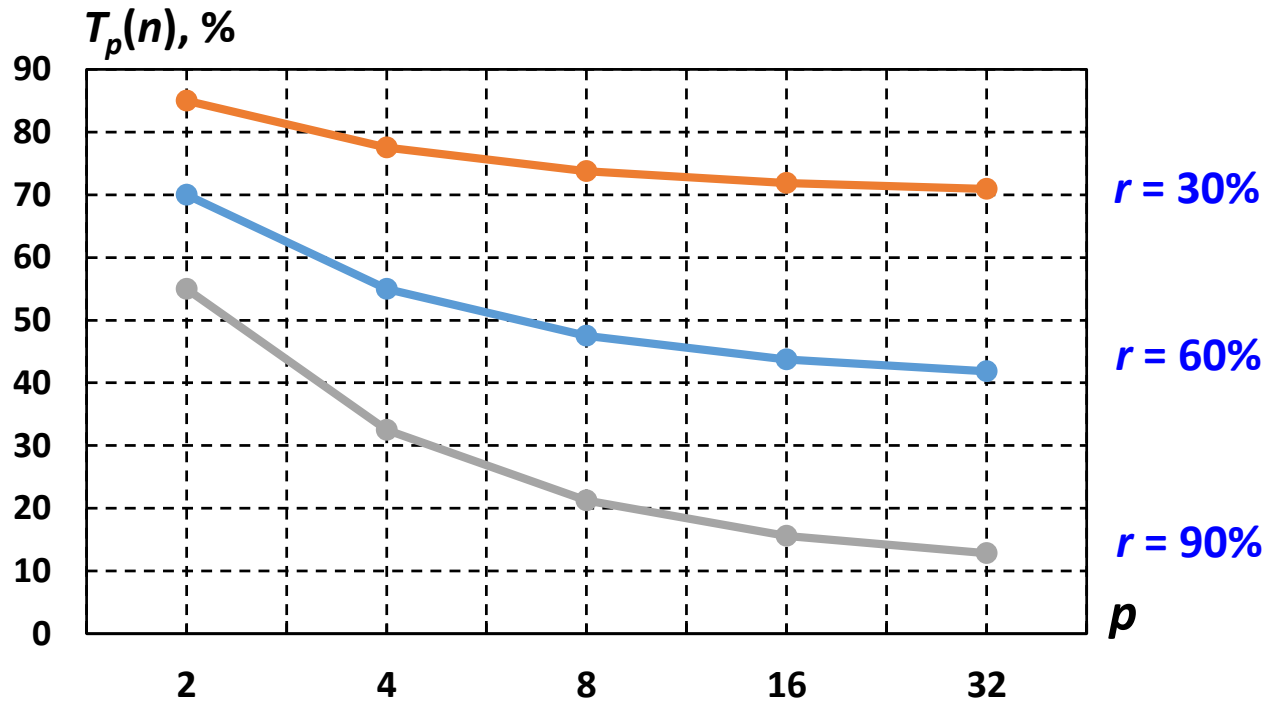
- **Последовательный алгоритм является наиболее оптимальным способом решения задачи**
- Возможны ситуации когда параллельная программа (алгоритм) эффективнее решает задачу (может эффективнее использовать кеш-память, конвейер, SIMD-инструкции, …)
- **Время выполнения параллельной программы оценивается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



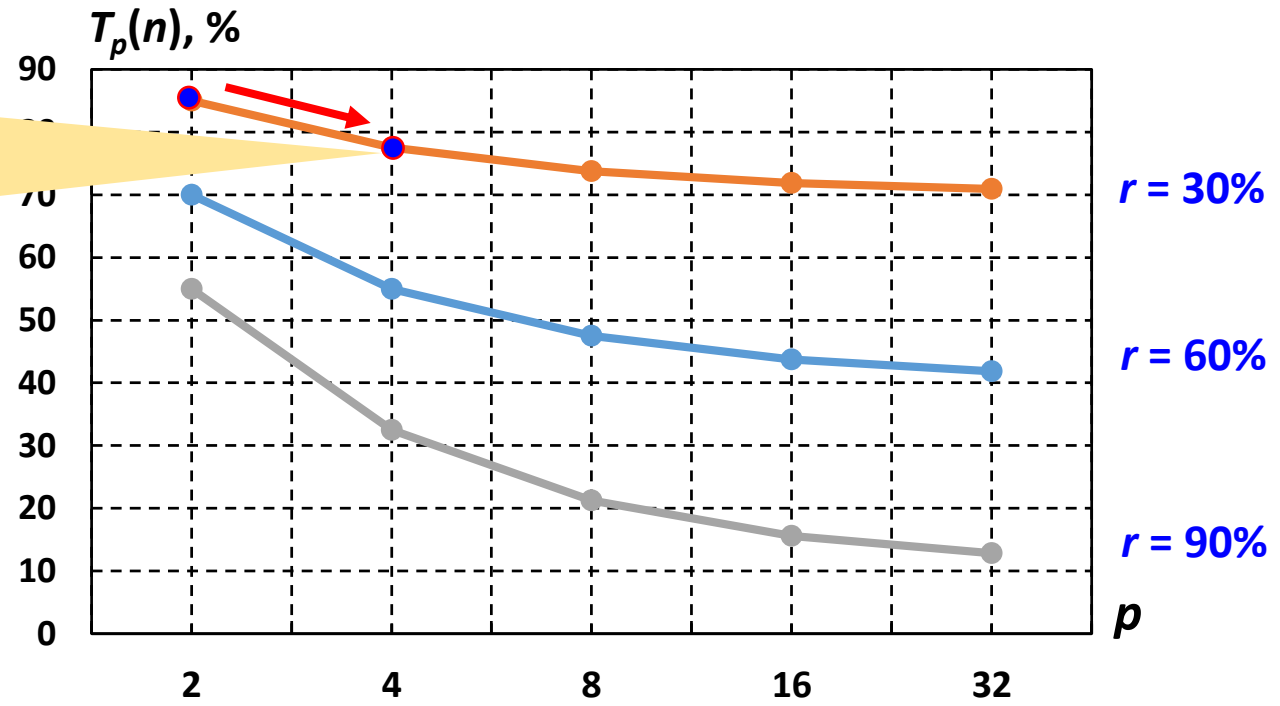
Зависимость времени $T_p(n)$ выполнения параллельной программы от количества *p* процессоров и доли *r* распараллеленного кода (время в % от времени $T_1(n)$)

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)

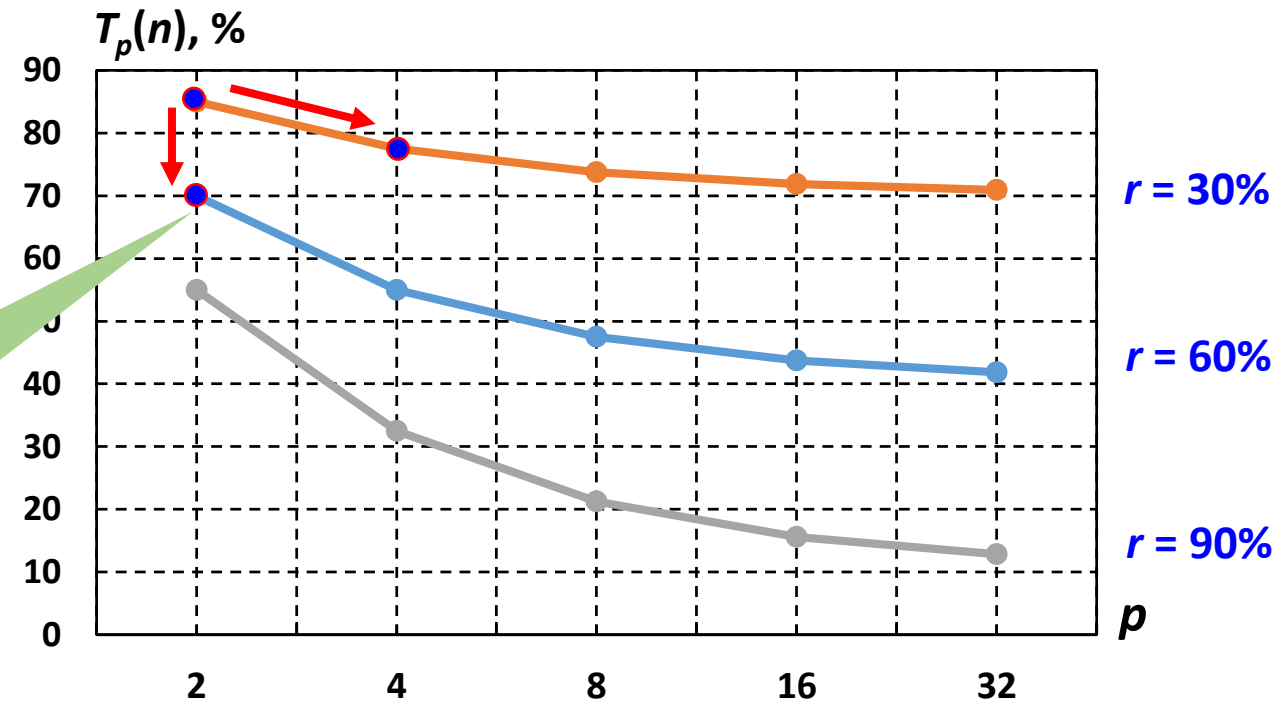
- **На что потратить ресурсы – на увеличение доли *r* параллельной части в программе или увеличение числа процессоров, на которых запускается программа?**

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 $s \in [0, 1]$ – часть параллельной программы, которая выполняется последовательно (purely sequential)
- **Закон Густафсона-Барсиса (Gustafson–Barsis' law) [1]:**

Масштабируемое ускорение S_p программы на p процессорах равняется

$$S_p = p - s(p - 1)$$

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

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

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

- **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.html

Литература

- Хорошевский В.Г. **Архитектура вычислительных систем**. – М.: МГТУ им. Н.Э. Баумана, 2008. – 520 с.
- Корнеев В.В. **Вычислительные системы**. – М.: Гелиос АРВ, 2004. – 512 с.
- Степаненко С.А. **Мультипроцессорные среды суперЭВМ. Масштабирование эффективности**. – М.: ФИЗМАТЛИТ, 2016. – 312 с.
- Эндрюс Г**. Основы многопоточного, параллельного и распределенного программирования**. – М.: Вильямс, 2003.