



Introduction to Parallel Computing

National Tsing Hua University
Instructor: Jerry Chou
2019, Fall Semester



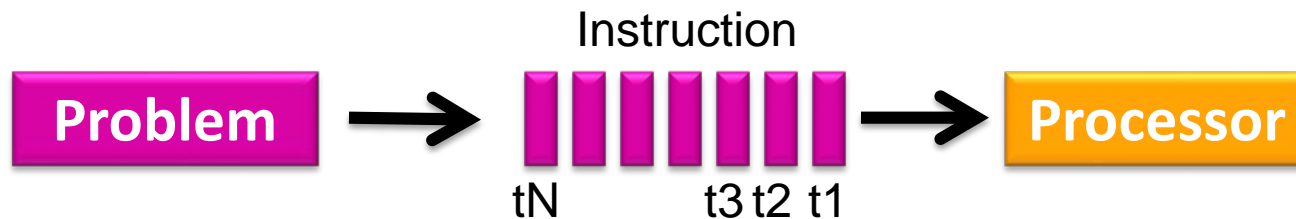
Outline

- Parallel Computing Introduction
 - What is parallel computing
 - Why need parallel computing
- Classifications of Parallel Computers & Programming Models
- Supercomputer & Latest technologies
- Parallel Program Analysis

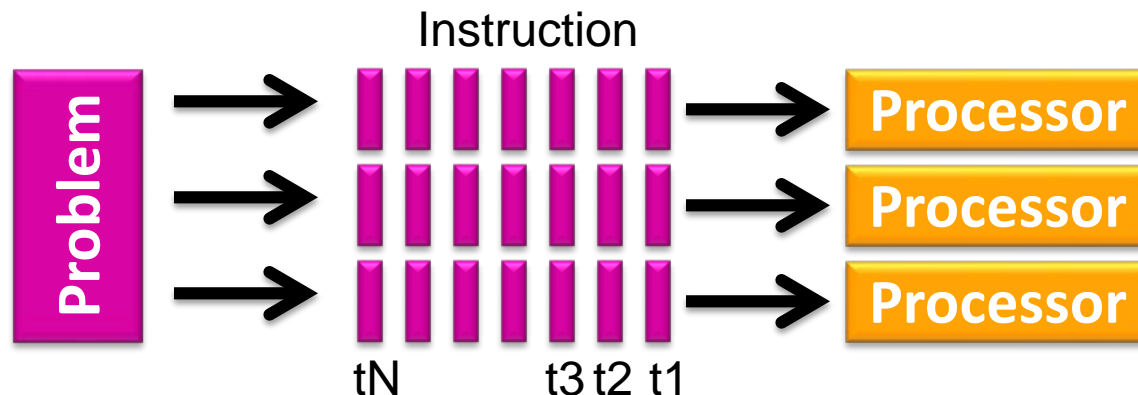
What is Parallel Computing?

“Solve a *single problem* by using *multiple processors* (i.e. *core*) working together”

- Traditionally, program has been written for **serial computation**



- In parallel computing**, use multiple computer resources to solve a computational problem



Difference between parallel computing & distributed computing

The two terminologies are very closely related.
But come from *different backgrounds*

■ Parallel computing ...

- Means different activities happen at the same time
- Spread out a single application over many cores/processors/processes to get it done bigger or faster
- Mostly used in scientific computing

■ Distributed computing...

- Activities across systems or distanced servers
- Focus more on concurrency and resource sharing
- From the business/commercial world

The Universe is Parallel

- Parallel computing is an **evolution of serial computing** that attempts to emulate what has always been the state of affairs in the natural world



Why need Parallel Computing

■ Save time

- Use more resources to shorten execution with potential cost saving

4 hours of work



Finish in 1 hour!!!



TIME IS MONEY



- Shorter execution time allows more runs or more tuning opportunity

	DUAL XEON CPU server	DGX-1 GPU server (8 GPUs)
FLOPS	3TF	170TF
Node Mem BW	76GB/s	768GB/s
Alexnet Train Time	150 Hr	2Hr
Train in 2Hr	>250Nodes	1Node

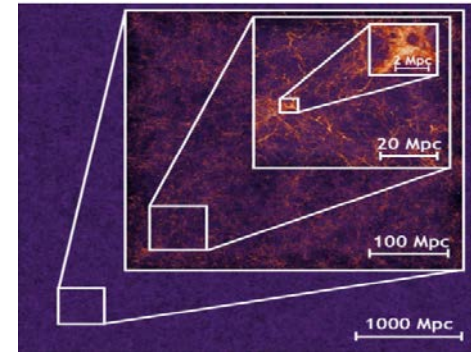
Why need Parallel Computing

■ Solve larger problem

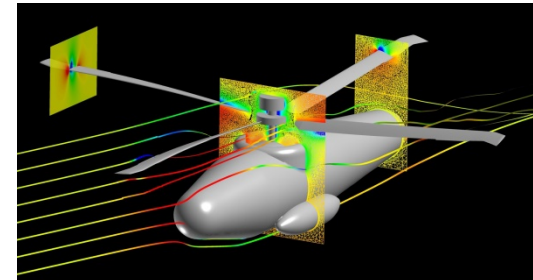
➤ Impossible or impractical to solve on a single computer

➤ Scientific computing:

- ◆ Trillion particles
- ◆ Tens and hundreds of parameters
- ◆ TBs of data to be processed/analyzed
- ◆ Several hours of execution
using millions of cores (PetaFLOPS)



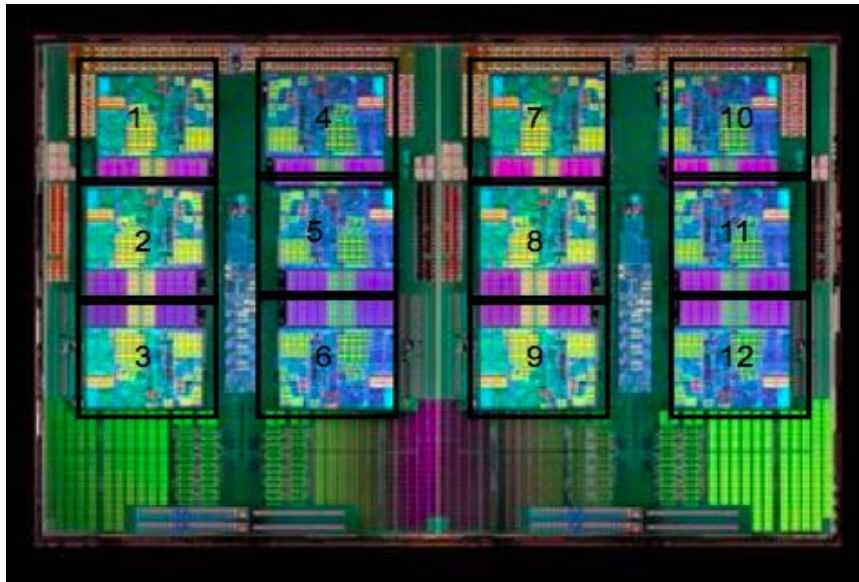
1 trillion particles, 4.225 Gpc box-size simulation, and 6 kpc force resolution.



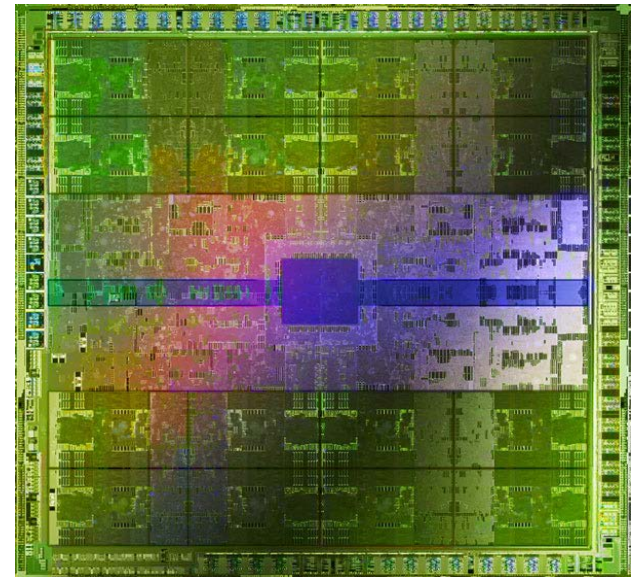
The world has been driven by science research!

Why need Parallel Computing

- Make better use of the underlying parallel hardware
 - Advance in computer architecture



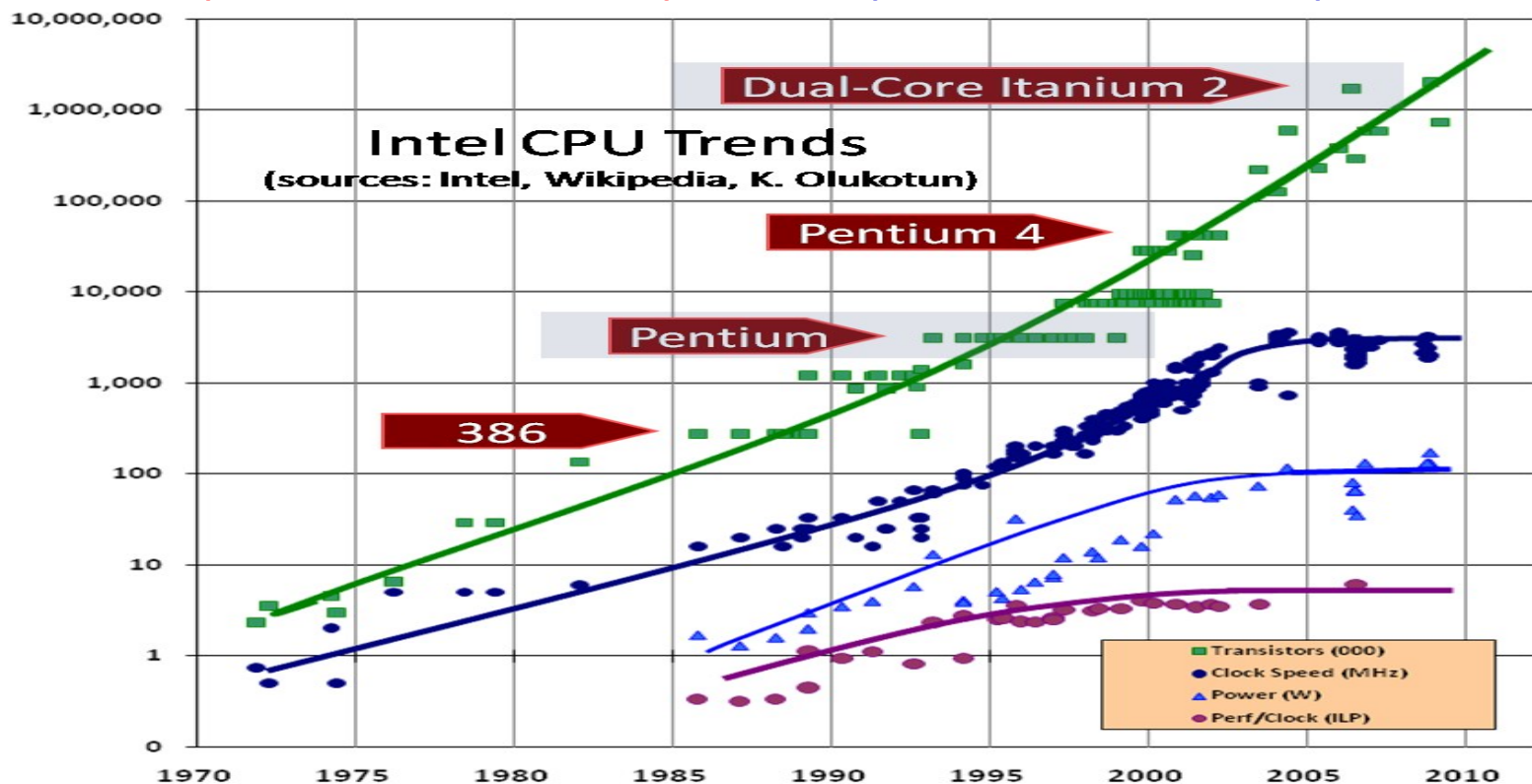
12 Cores IBM Blade Multi-core CPU



512 Cores NVIDIA Fermi GPU

The Death of CPU Scaling

- Increase of transistor density \neq performance
 - The power and clock speed improvements collapsed



“Parallel Computing is a trend and essential tools in today’s world!”

Trend of Parallel Computing

Single-Core Era

Enabled by:
Moore's Law
Voltage Scaling

Constraint by:
Power
Complexity

Assembly → C/C++ → Java ...

Heterogeneous Systems Era

Enabled by:
Abundant data
parallelism
Power efficient GPUs

Constraint by:
Programming
models
Comm. overhead

Shader → CUDA → OpenCL ...

Muti-Core Era

Enabled by:
Moore's Law
SMP

Constraint by:
Power
Parallel SW
Scalability

Pthread → OpenMP ...

Distributed System Era

Enabled by:
Networking

Constraint by:
Synchronization
Comm. overhead

MPI → MapReduce ...

Outline

- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
 - Flynn's classic taxonomy
 - Memory architecture classification
 - Programming model classification
- Supercomputer & Latest technologies
- Parallel Program Analysis

Parallel Computer Classification

■ Flynn's classic taxonomy

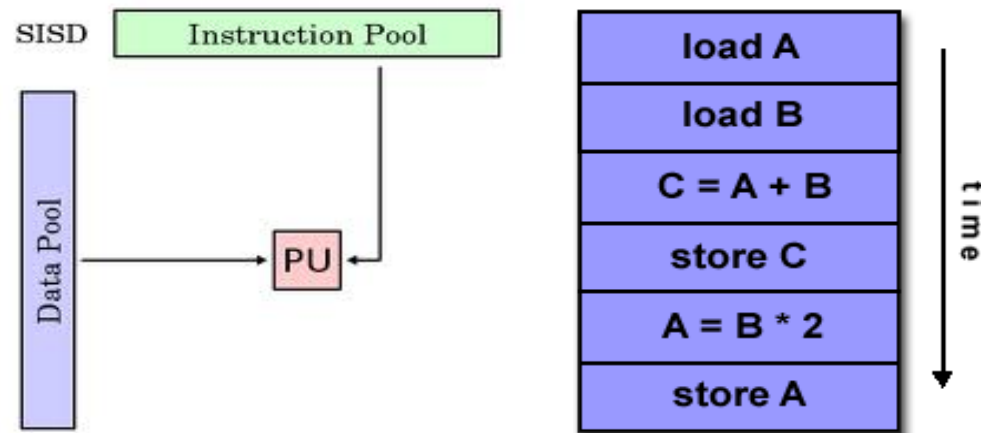
- Since **1966** (50 years ago ...)
- From the **process unit prospective**: Classify computer architecture based two independent dimensions: **Instruction & Data**

<u>SISD</u> Single Instruction Single Data	<u>SIMD</u> Single Instruction Multiple Data
<u>MISD</u> Multiple Instruction Single Data	<u>MIMD</u> Multiple Instruction Multiple Data

Flynn's classic taxonomy: SISD

■ Single Instruction, Single Data (SISD):

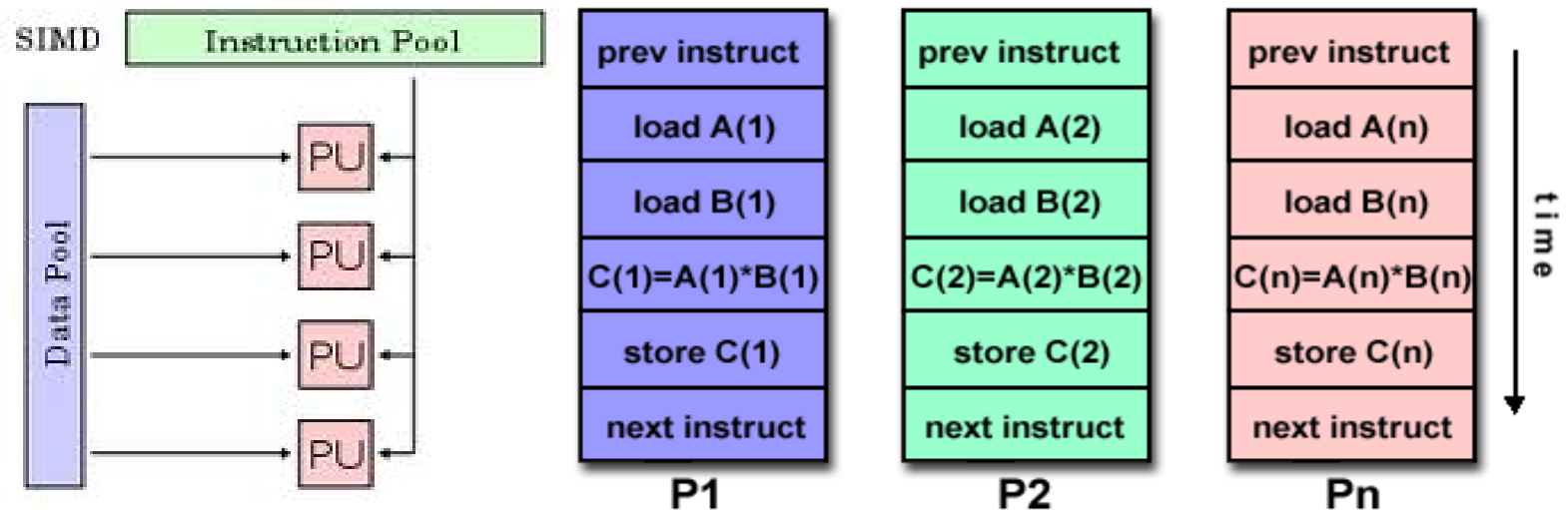
- A serial (**non-parallel**) computer
- **Single Instruction:** Only one instruction stream is being acted on by the CPU during any one clock cycle
- **Single Data:** Only one data stream is being used as input during any one clock cycle
- Example: Old mainframes, **single-core processor**



Flynn's classic taxonomy: SIMD

■ Single Instruction, Multiple Data (SIMD):

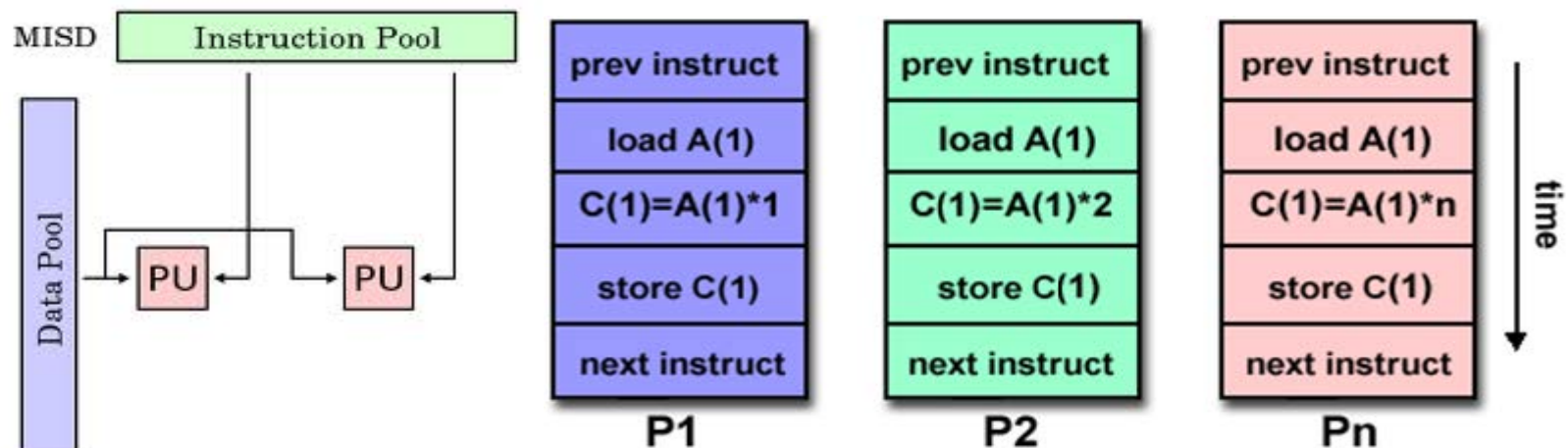
- **Single Instruction:** All processing units execute the same instruction at any given clock cycle
- **Multiple Data:** Each processing unit can operate on a different data element
- Example: GPU, vector processor (X86 AVX instruction)



Flynn's classic taxonomy: MISD

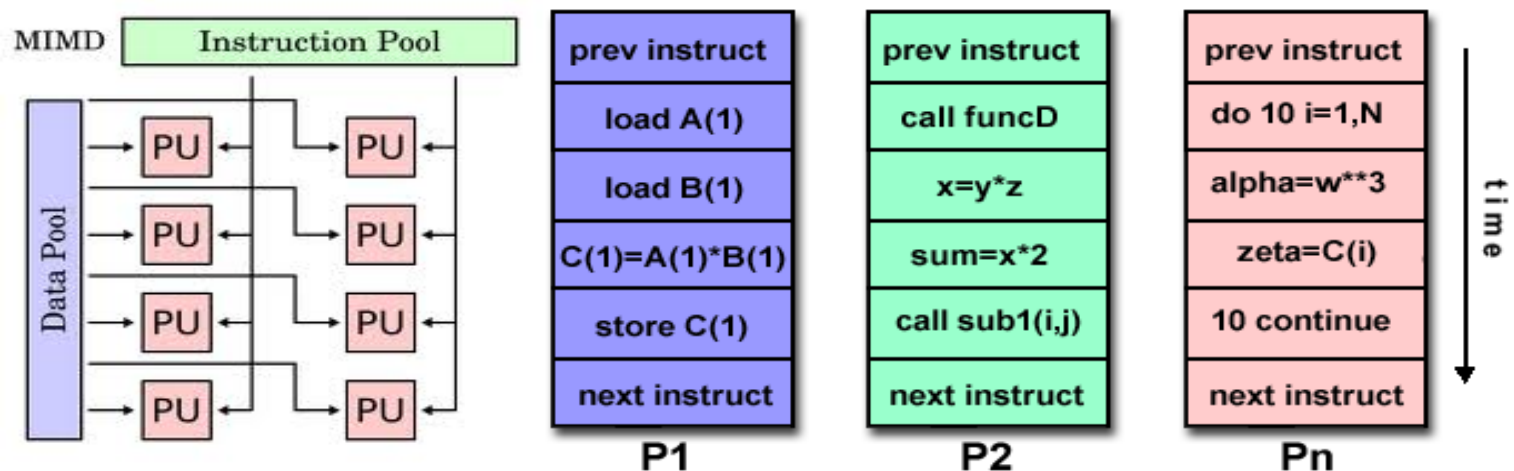
■ Multiple Instruction, Single Data (MISD):

- **Multiple Instruction:** Each processing unit operates on the data independently via separate instruction streams.
- **Single Data:** A single data stream is fed into multiple processing units.
- **Example:** Only experiment by CMU in 1971; Could be used for **fault tolerance**



Flynn's classic taxonomy: MIMD

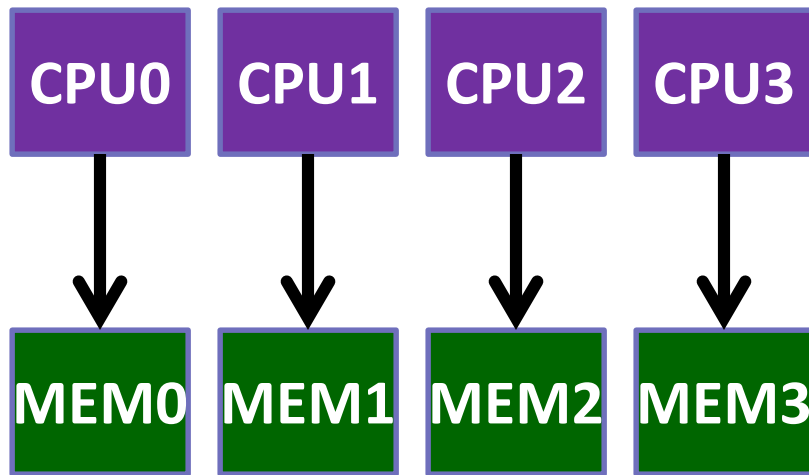
- Multiple Instruction, Multiple Data (MIMD):
 - **Multiple Instruction:** Every processor may be executing a different instruction stream
 - **Multiple Data:** Every processor may be working with a different data stream
 - Example: Most modern computers, such as **multi-core CPU**



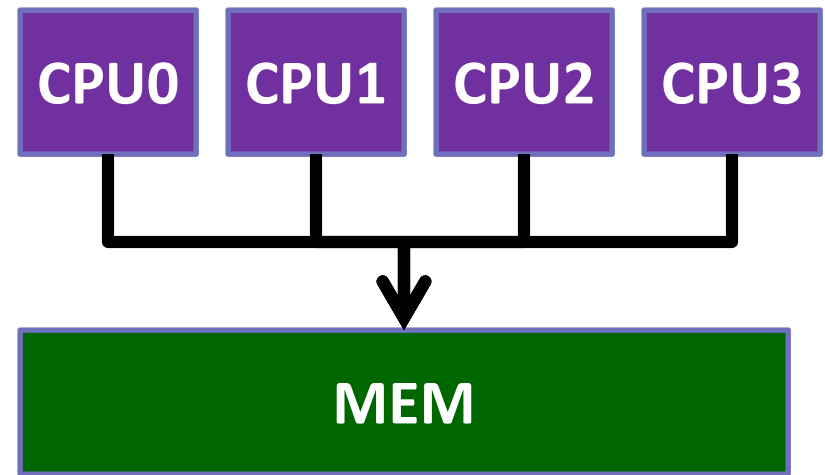
Outline

- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
 - Flynn's classic taxonomy
 - Memory architecture classification
 - Programming model classification
- Supercomputer & Latest technologies
- Parallel Program Analysis

Shared Memory vs. Distributed Memory Computer Architecture



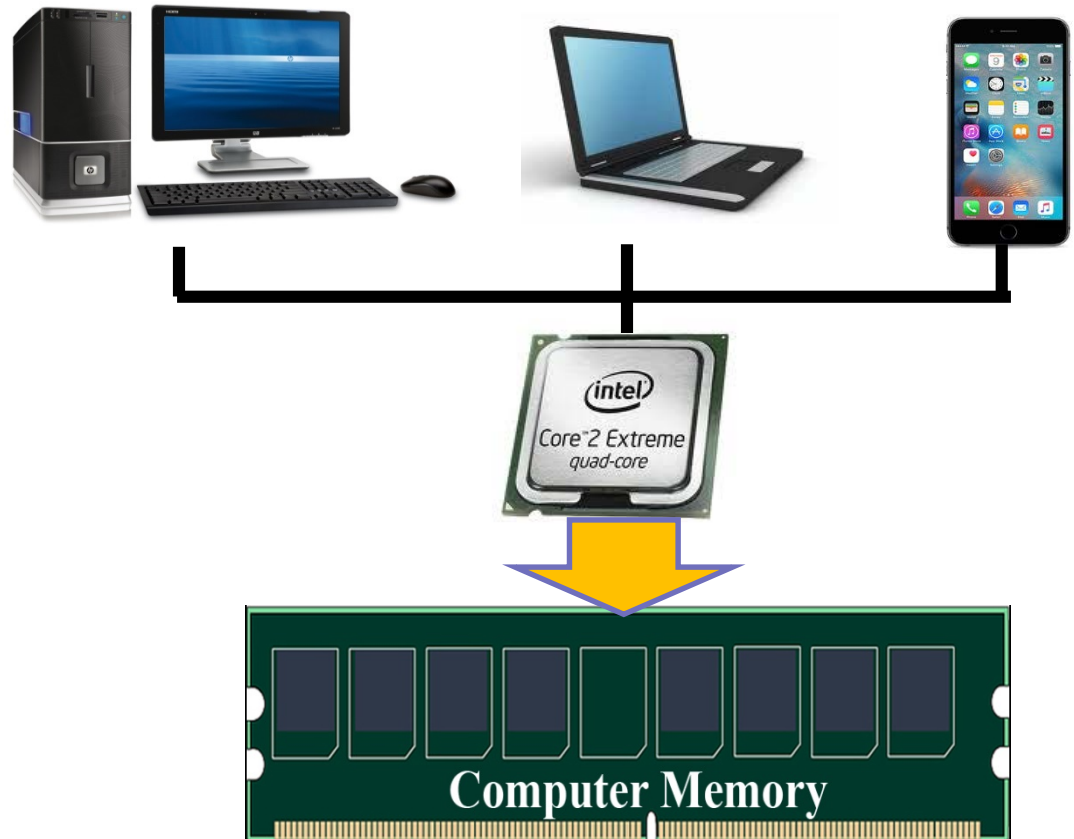
Distributed memory



Shared memory

Shared Memory Multiprocessor Computer System

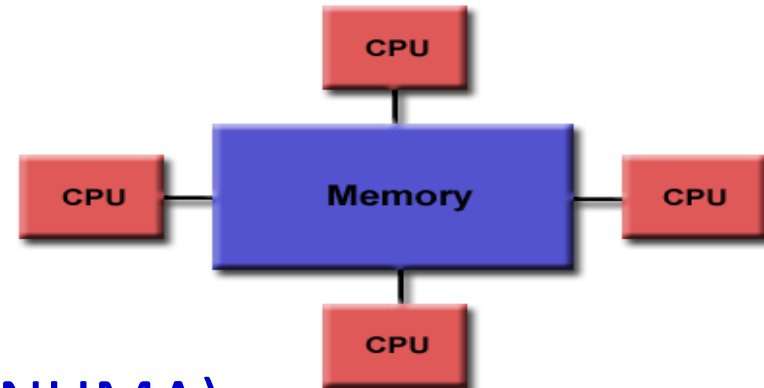
- Single computer with multiple internal multi-core processors



Shared Memory Computer Architecture

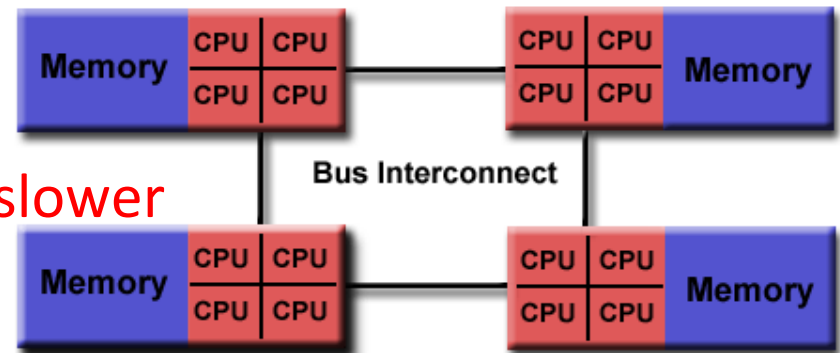
■ Uniform Memory Access (UMA):

- Most commonly represented today by Symmetric Multiprocessor (**SMP**) machines
- **Identical processors**
- **Equal access times to memory**
- Example: **commercial servers**



■ Non-Uniform Memory Access (NUMA):

- Often made by physically **linking two or more SMPs**
- One SMP can directly access memory of another SMP
- **Memory access across link is slower**
- Example: **HPC server**



Distributed Memory Multicomputer

- Connect multiple computers to form a computing platform without sharing memory



Cluster: tens of servers



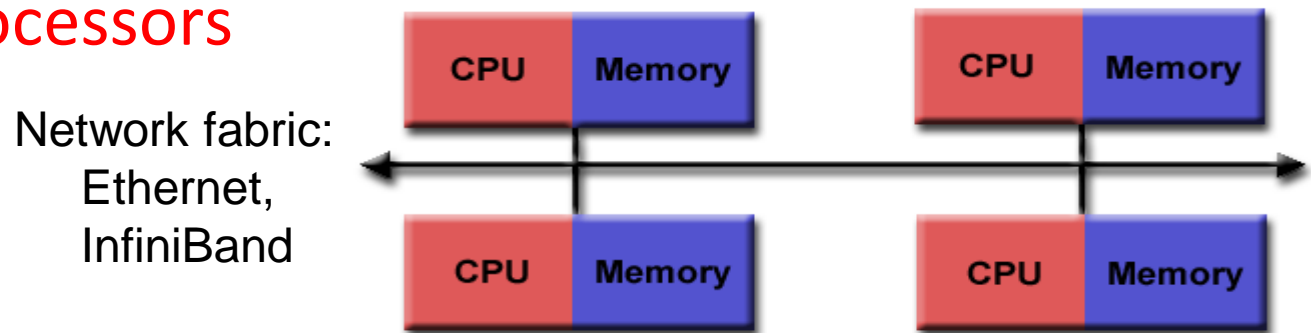
Supercomputer:
hundreds of servers



Datacenter: thousands of servers

Distributed Memory Multicomputer

- Require a communication network (i.e. not bus) to connect inter-processor memory
- Processors have their own memory & address space
- Memory change made by a processor has **NO effect** on the memory of other processors
- Programmers or programming tools are responsible to explicitly define how and when data is communicated between processors



Outline

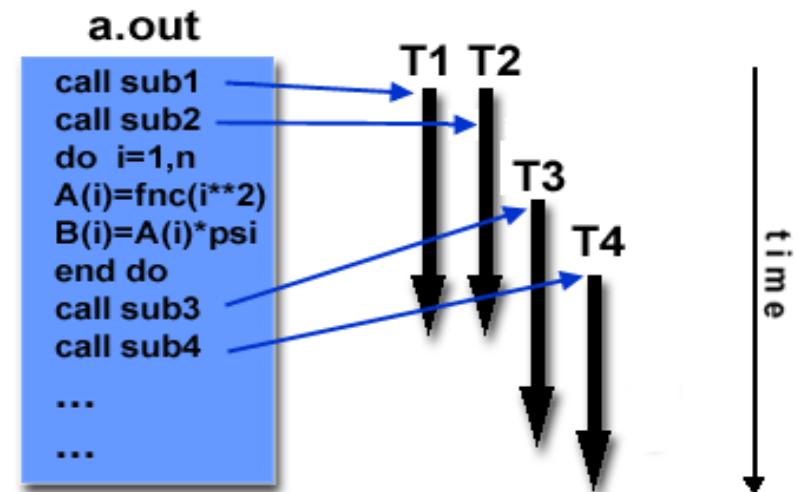
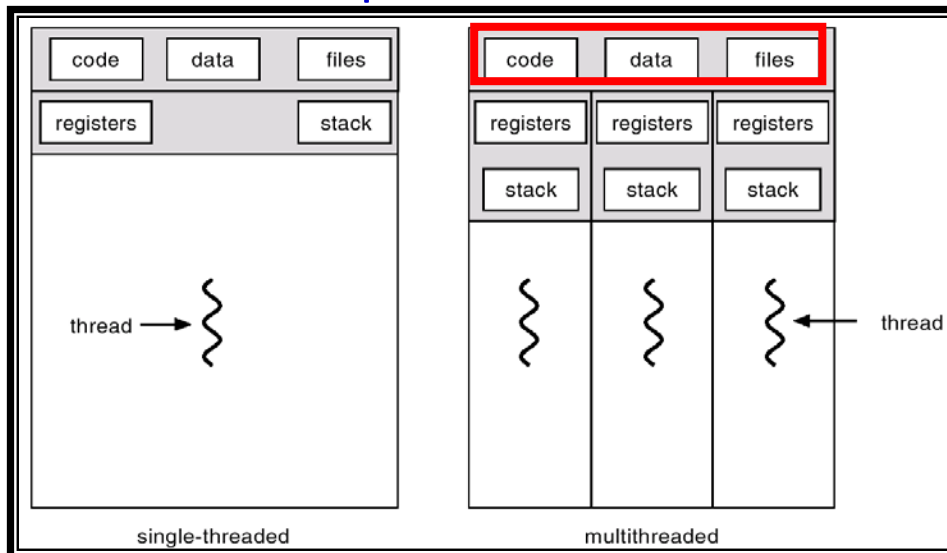
- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
 - Flynn's classic taxonomy
 - Memory architecture classification
 - Programming model classification
- Supercomputer & Latest technologies
- Parallel Program Analysis

Parallel Programming Model

- Parallel programming models exist as an **abstraction above hardware & memory architectures**
- In general programming models are designed to match the computer architecture
 - Shared memory prog. model for shared memory machine
 - Message passing prog. model for distributed memory machine
- But programming models are **NOT** restricted by the machine or memory architecture
 - **Message passing model** can be supported on **SHARED** memory machine: e.g., MPI on a single server
 - **Shared memory model** on **DISTRIBUTED** memory machine: e.g., Partitioned Global Address Space

Shared Memory Programming Model

- A single process can have **multiple, concurrent execution paths**
- Threads have **local data**, but also, **shares resources**
- Threads **communicate with each other through global memory**
- **Threads can come and go**, but the main program remains
 - to provide the necessary shared resources until the application has completed



Shared Memory Programming Model

■ Implementation

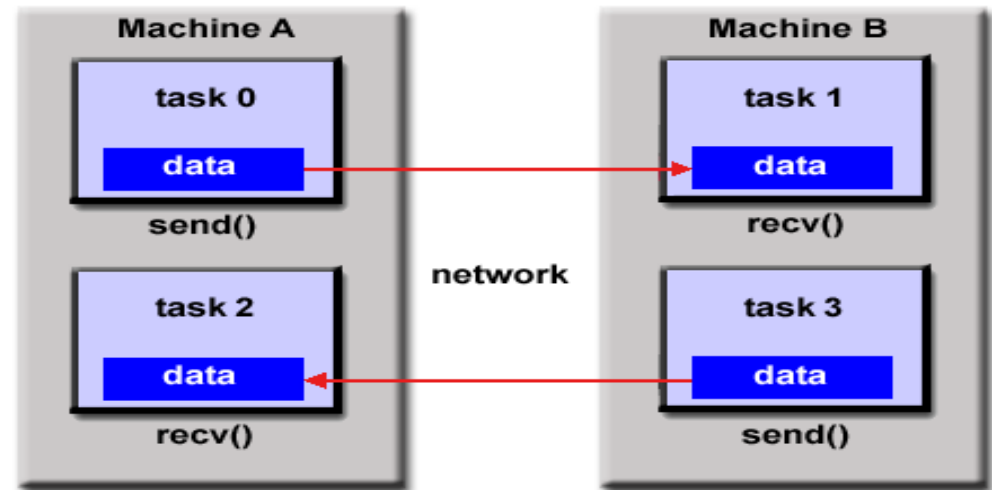
- A **library of subroutines** called from parallel source code
 - ◆ E.g.: **POSIX Thread (Pthread)**
- A set of **compiler directives** imbedded in either serial or parallel source code
 - ◆ E.g.: **OpenMP**

```
#include <pthread.h>
void print_message_function ( void *ptr ) {
    printf("Hello, world.\n");
}
int main() {
    pthread_t thread;
    pthread_create (&thread, NULL, (void *)
        &print_message_function, NULL);
    pthread_join(thread, NULL);
}
```

```
#include <omp.h>
int main() {
    #pragma omp parallel
    {
        printf("Hello, world.\n");
    }
}
```


Message Passing Programming Model

- A set of tasks that use their own **local memory** during computation
 - Multiple tasks can reside on the **same physical machine** and/or across an arbitrary number of machines
- Tasks exchange data through **communications by sending and receiving messages** (Memory copy)
- **MPI API:**
 - Send, Recv, Bcast, Gather, Scatter, etc.



Shared Memory vs. Message Passing

Shared Memory

■ Convenient:

- Can share data structures
- Just annotate loops
- Closer to serial code

■ Disadvantages

- No locality control
- Does not scale
- Race conditions

Message Passing

■ Scalable

- Locality control
- Communication is all explicit in code (cost transparency)

■ Disadvantage

- Need to rethink entire application/ data structures
- Lots of tedious pack/unpack code
- Don't know when to say "receive" for some problems

Summary

- The designs and popularity of **programming model** and **parallel systems** are highly influenced by each other
- **openMP, MPI, Pthreads, CUDA** are just some of the parallel languages for users to do **parallel programming**
- In reality, knowing what is **parallel computing** is more **IMPORTANT** than knowing how to do **parallel programming**, because that's how you can...
 - Learn a new parallel programming tools quickly
 - Understand the performance of your program
 - Optimize the performance of your program

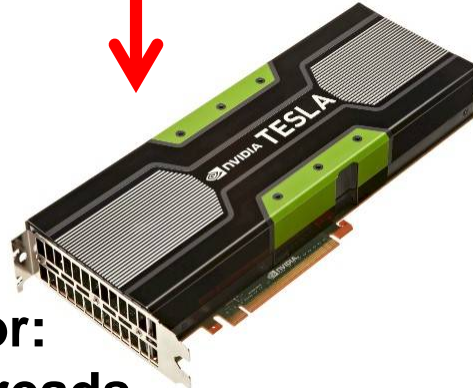
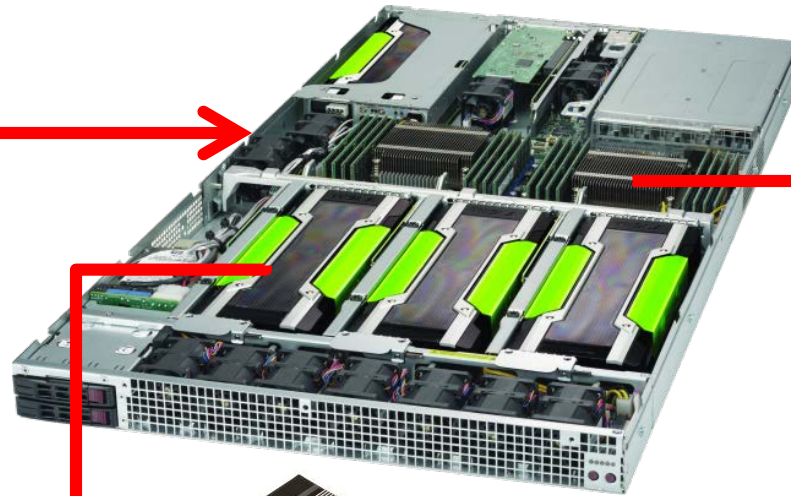
Outline

- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
- **Supercomputer & Latest technologies**
 - Supercomputer
 - Processor technology
 - Interconnect & Network technology
 - I/O & Storage technology
- Parallel Program Analysis

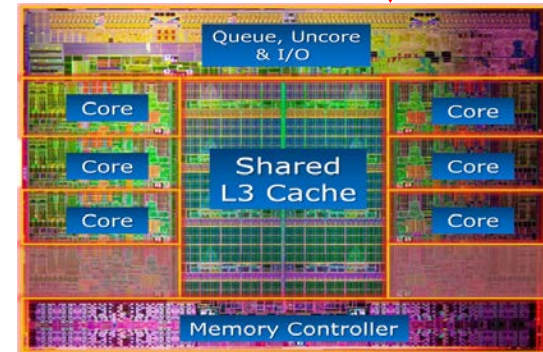
Today's Typical Parallel Computers

Racks: 16~42U

Node/Server: 1~4U



**Multi-core Processor:
100x cores/1000xthreads**



Co-Processor: 4~12 cores

Supercomputers

- Definition: A computer with a **high-level computational capacity** compared to a general-purpose computer
- Its **performance** is measured in floating-point operations per second (**FLOPS**) instead of million instructions per second (MIPS)
- Ranked by the **TOP500** list since 1993
 - According to the **HPL benchmark** results
 - Announced twice a year at ISC and SC conferences

HPL Benchmark

- A parallel implementation of Linpack library
 - Measure **floating point** rate of execution

- Computation:

- To solve **linear matrix equation**

$$Ax = b; \quad A \in \mathbf{R}^{n \times n}; \quad x, b \in \mathbf{R}^n$$

- **LU factorization** by Panel factorization.
 - Divide a matrix into many pieces.
 - **All parameters must be determined by user.**

HPL Benchmark

- A parallel implementation of Linpack library
 - Measure **floating point** rate of execution

What uses such benchmark?

- Computation:
 - To solve **linear matrix equation**

$$Ax = b; \quad A \in \mathbf{R}^{n \times n}; \quad x, b \in \mathbf{R}^n$$

- **LU factorization** by Panel factorization.
- Divide a matrix into many pieces.
- **All parameters must be determined by user.**

What makes it a supercomputer

- What makes it a supercomputer?
 - All the latest **hardware technologies**
 - Customized system **configurations**
 - Optimized **software and libraries**
 - Huge amount of cost in **money and energy**
- It represents a competition of technology and wealth among a countries



TOP500 List (2019 June)

	Country	System	Vendor	Power (kW)	#cores	Accelerator	Rmax	Rpeak (PFLOPS)
1	US	Summit	IBM	10,096	2.4M	Volta V100	148.6	200.8
2	US	Sierra	IBM	7,438	1.5M	Volta V100	94.6	125.7
3	China	TaihuLight	NRCPC	15,371	10M		93.0	125.4
4	China	Tianhe-2	NUDT	17,808	3M	Xeon Phi	33.9	54.9
5	US	Frontera	Cray		448K	Tesla P100	23.5	38.7

- Rpeak is a theoretic performance computed by HW spec
 - For example, an Intel Itanium 2 at 1.5 GHz can complete 4 floating point operations per cycle or a theoretical peak performance of 6 GFlop/s.
- Rmax is real performance measured by the HPL benchmark

TOP500 List (2019 June)

	Country	System	Vendor	Power (kW)	#cores	Accelerator	Rmax	Rpeak (PFLOPS)
1	US	Summit	IBM	10,096	2.4M	Volta V100	148.6	200.8
2	US	Sierra	IBM	7,438	1.5M	Volta V100	94.6	125.7
3	China	TaihuLight	NRCPC	15,371	10M		93.0	125.4
4	China	Tianhe-2	NUDT	17,808	3M	Xeon Phi	33.9	54.9
5	US	Frontera	Cray		448K	Tesla P100	23.5	38.7
6	Swiss	Piz Daint	Cray	2,384	387K	Tesla P100	21.2	27.1

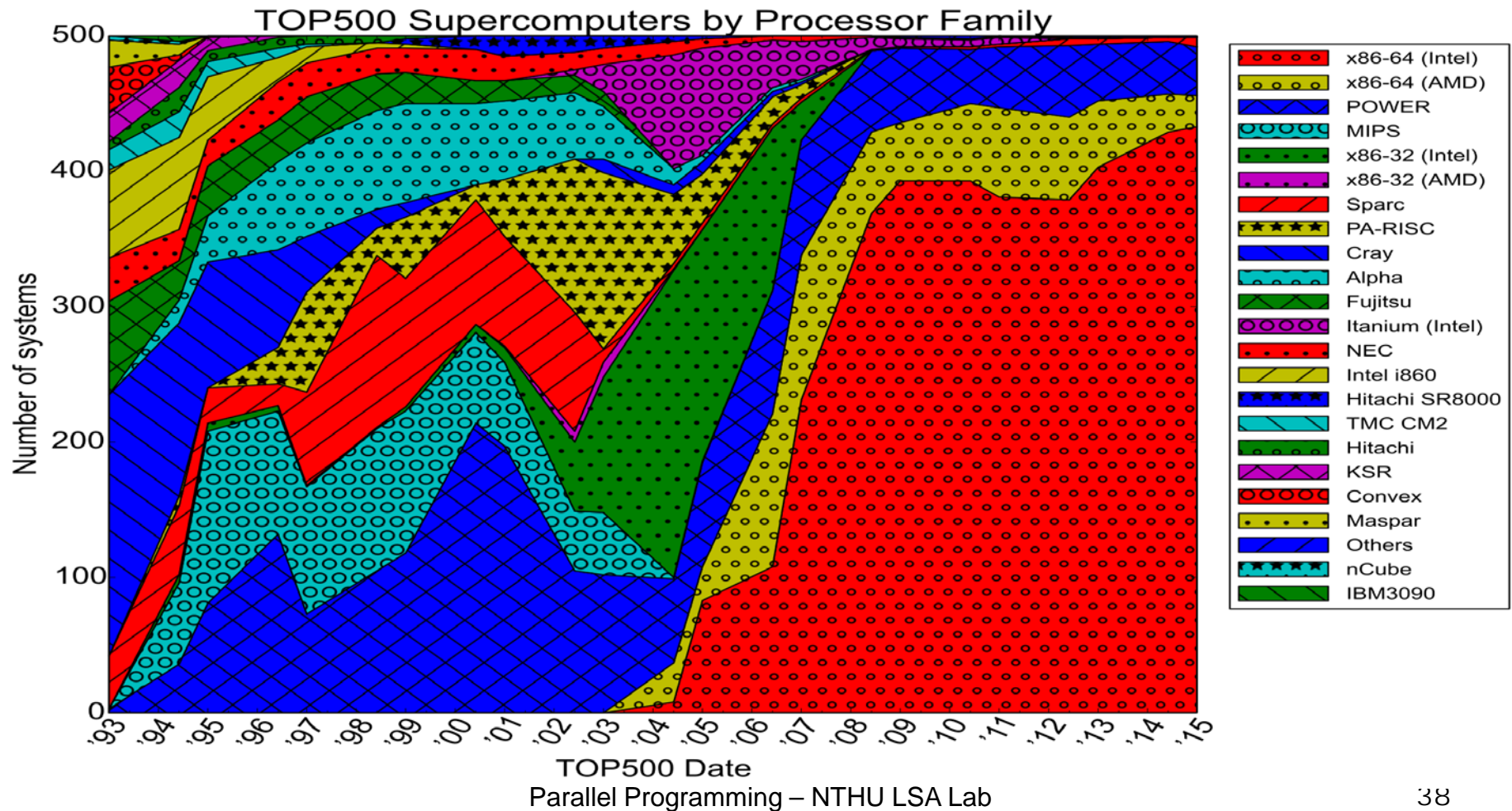
2016 June

8	Swiss	Piz Daint	Cray	2,325	116K	Tesla K20X	6.2	7.8
---	-------	-----------	------	-------	------	------------	-----	-----

- Accelerators provide huge computing power with better energy efficiency
- Power consumption is an increasing concern

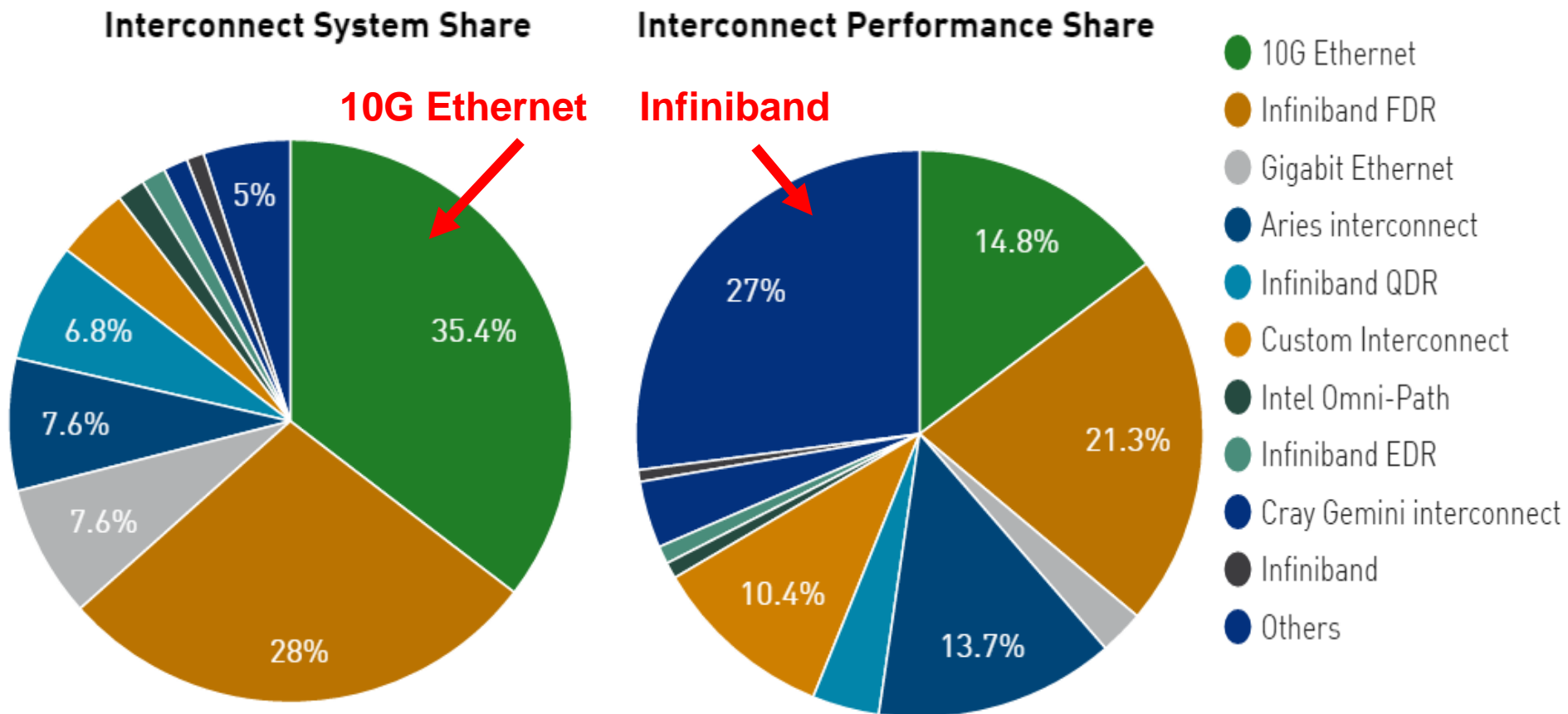
TOP500 Trend: CPU

■ Intel CPU counts for more than 80%



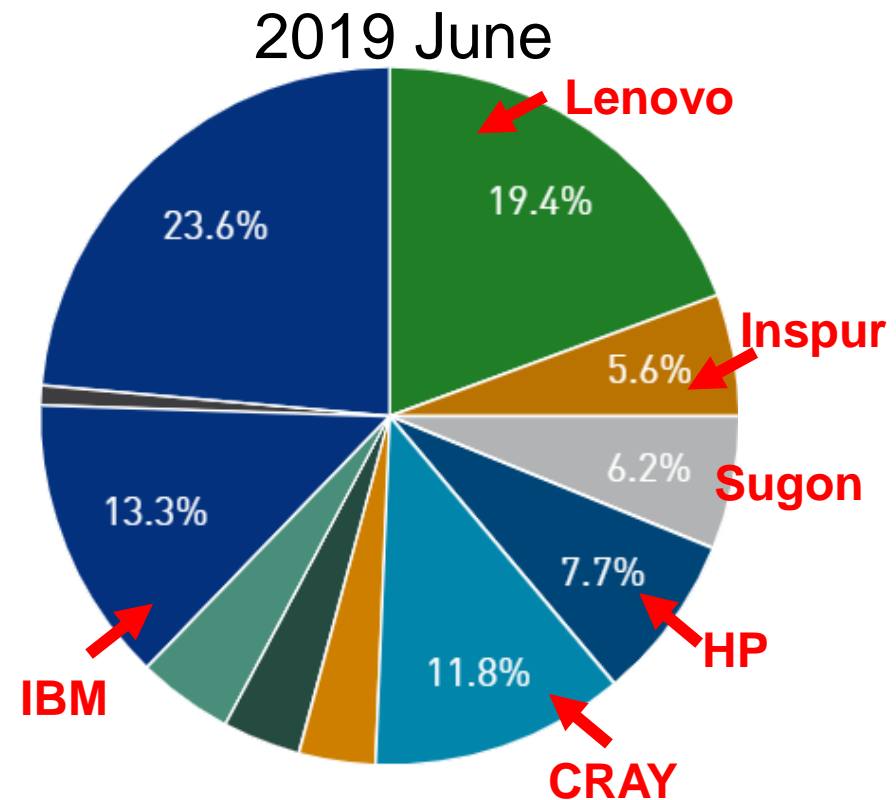
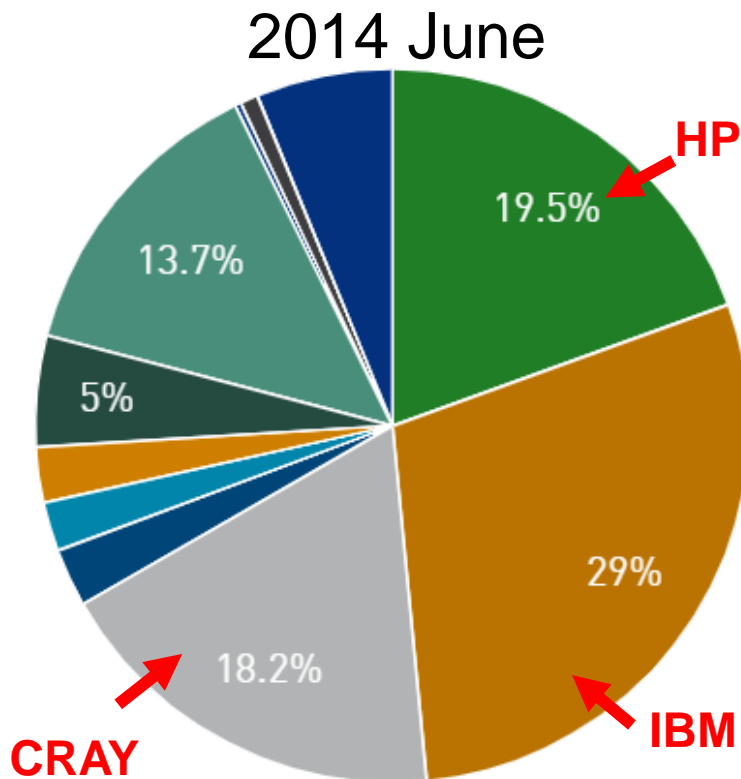
TOP500 Trend: Interconnect

- InfiniBand has much larger share in performance



TOP500 Trend: Vendor

- CRAY and IBM are the oldest vendors
- China is developing in a very fast pace

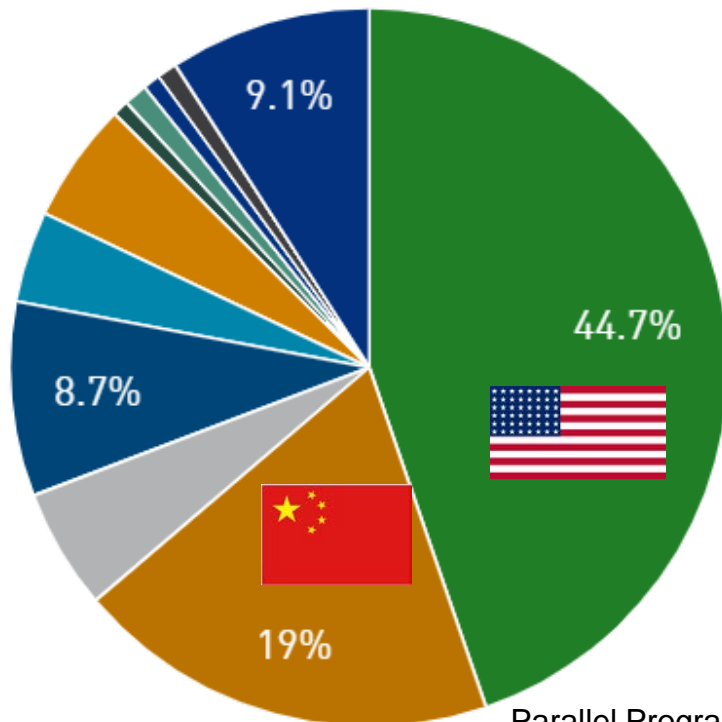


TOP500 Trend: Country

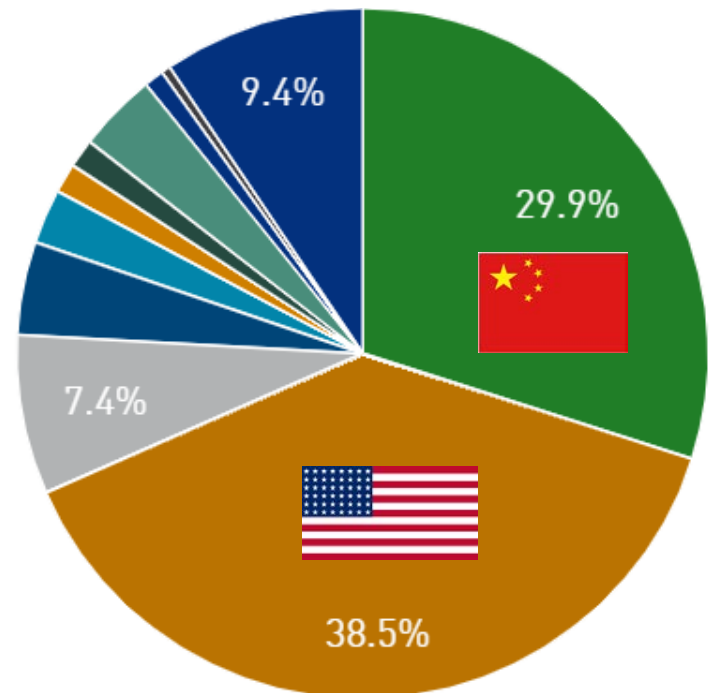
- China has a huge jump because of the new supercomputers

Countries Performance Share

2014 June

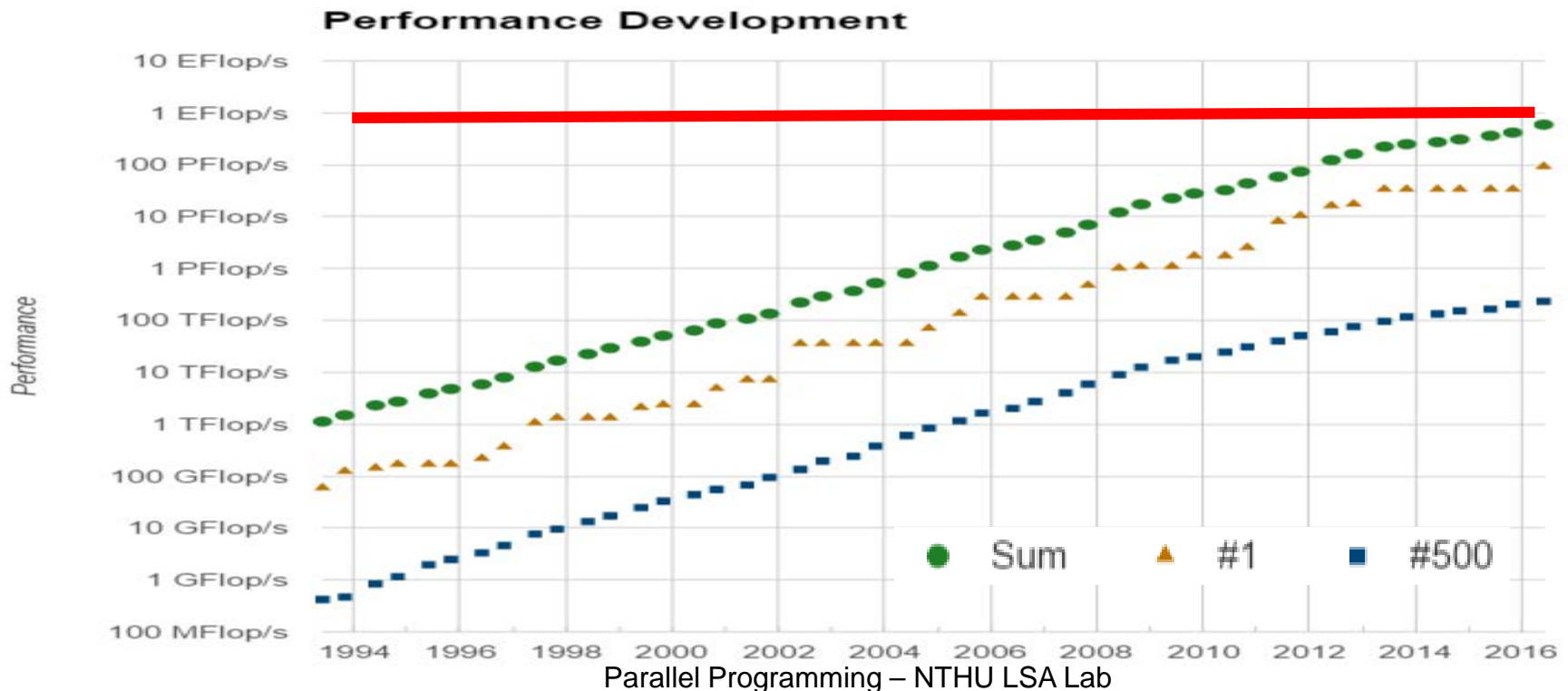


2019 June



TOP500 Trend: Computing power

- Goal is to reach **Exascale computing** 1EFlop (10^{18}) /s by 2020
 - It is estimated to be the order of processing power of the human brain at neural level
 - The target power of the Human Brain Project



Outline

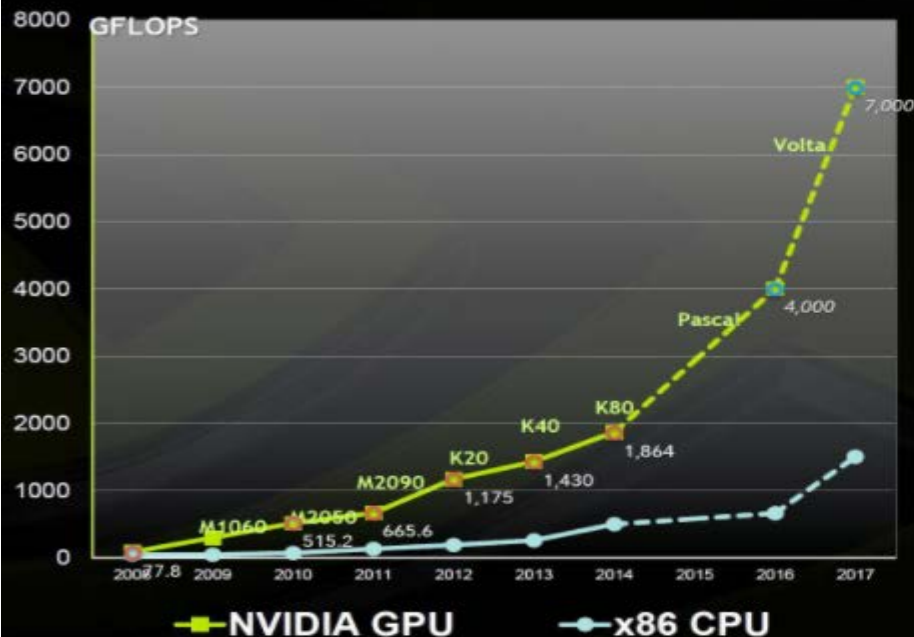
- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
- **Supercomputer & Latest technologies**
 - Supercomputer
 - **Processor technology**
 - Interconnect & Network technology
 - I/O & Storage technology
- Parallel Program Analysis

Limitation of CPU

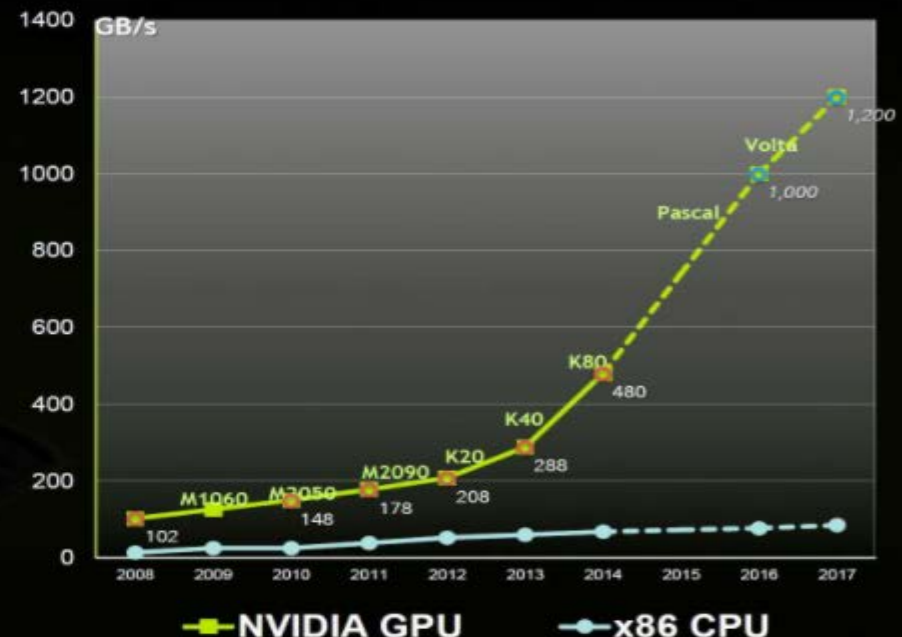
General Purpose Processor

- A general purpose CPU (central processing unit) can do anything, but its design is against the goal of achieving the best performance for a specific application.

Peak Double Precision FLOPS



Peak Memory Bandwidth



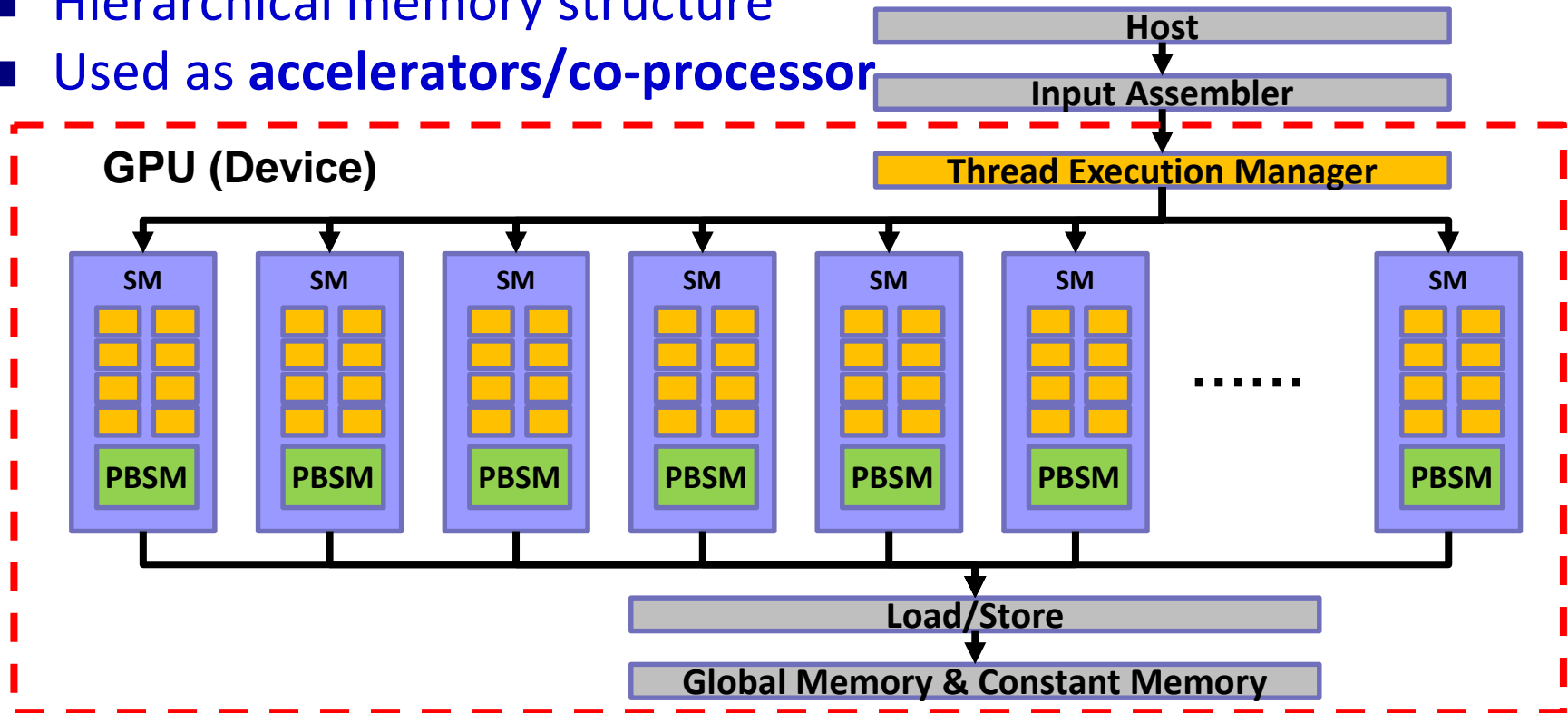
Comparison Numbers

	Intel Xeon E5-2697 v3 CPU (Haswell)	NVIDIA Tesla K80 GPU (Kepler)	Intel Xeon Phi 7120P (Knight's Corner)
Cores	2x14	2x13(SMX)	61
Logical Cores	2x28	2x2,496	244
Frequency	2.60GHz	562MHz	1.238GHz
GFLOPS(double)	2x583	2x1,455	1,208
Max memory	768GB	2x12GB	16GB
Max Mem BW	2x68GB/s	2x240GB/s (Internal)	352GB/s (Internal)
Price	2,700 USD	5,000 USD	4,000 USD

Source: https://www.xcelerit.com/computing-benchmarks/libor/haswell_k80_phi/

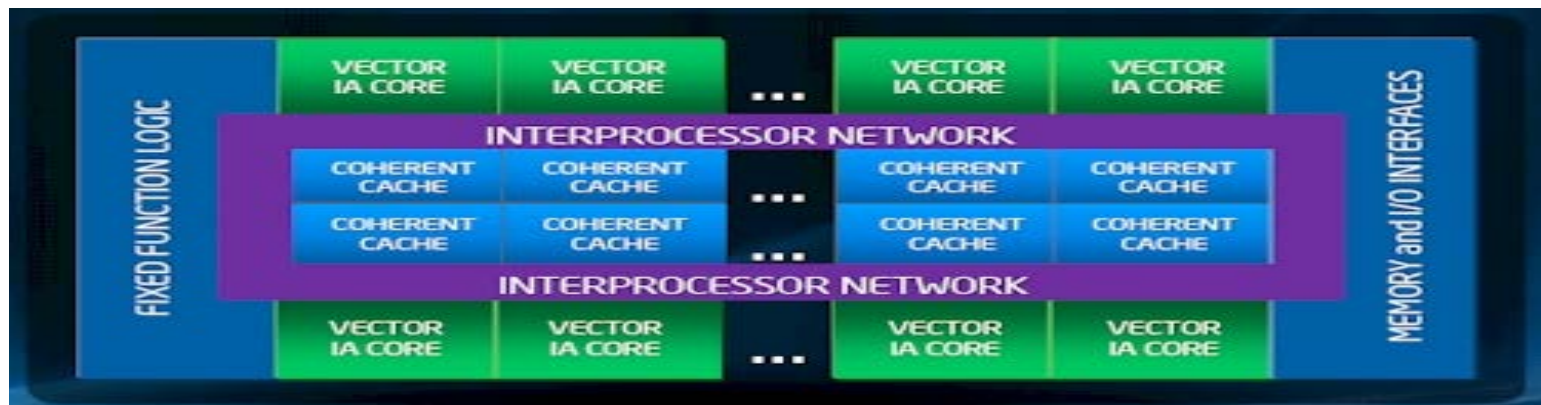
NVidia General Purpose GPU

- Extend GPU as a form of stream processor (or a **vector processor**) for general purpose computing
- Suited for **embarrassingly parallel** tasks and **vectorized operations**
- Hierarchical memory structure
- Used as **accelerators/co-processor**



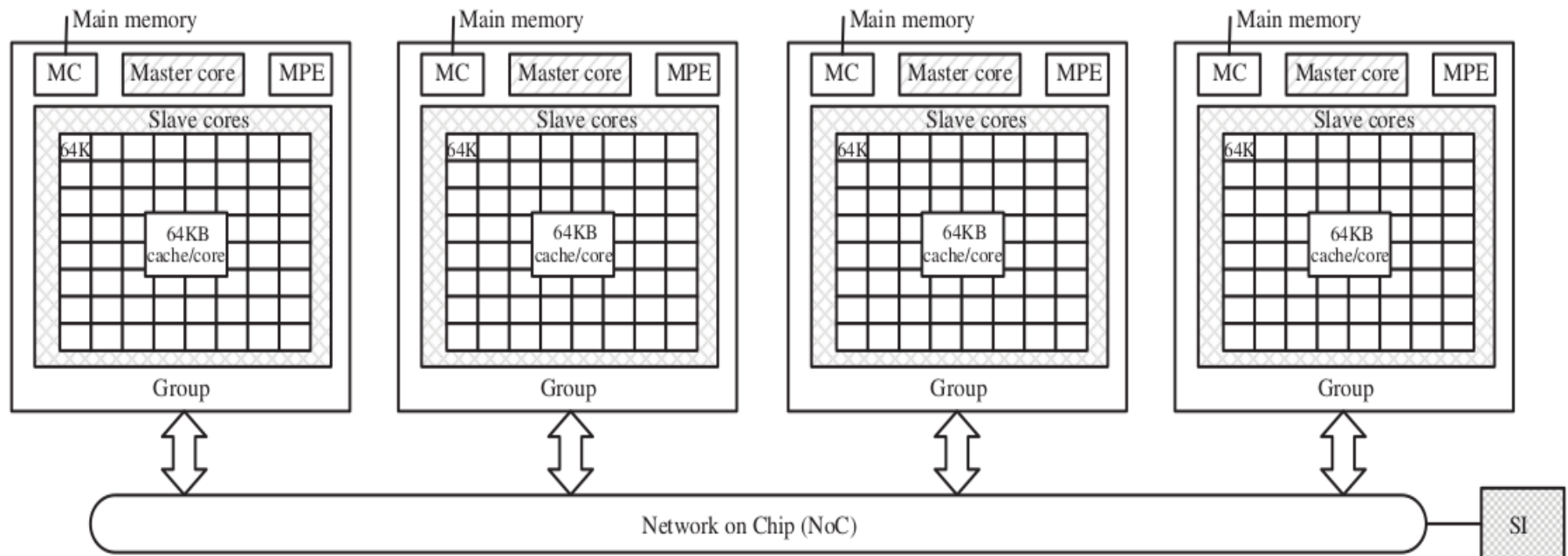
Intel Xeon Phi

- A brand name given to a series of manycore processors follows the Intel's MIC (Many Integrated Core) architecture
 - Typically it has **50-70 processors** on the die connected by a bidirectional Ring network
- More like a separate system
 - It runs **Intel assembly code** just like the main CPU in your computer
 - It has an **embedded linux**
 - Second generation chips (*Knights Landing*) could be used as a **standalone CPU**



Sunway TaihuLight SW26010

- Each node contains four clusters of 64 CPEs (SIMD)
- Each cluster is accompanied by a MPE (general purpose)



Google Tensor Processing Unit (TPU)

- Specifically for deep learning (tensorflow framework)
- 30–80X higher performance-per-watt than contemporary CPUs and GPUs
 - Only for **reduced precision computation** (e.g. 8-bit precision)
 - Matrix Multiplier Unit: use a to achieve hundreds of thousands of **matric operation** in a single clock cycle
 - Systolic array: The ALUs perform **only multiplications and additions in fixed patterns**
- Reference
 - <https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu>

Outline

- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
- **Supercomputer & Latest technologies**
 - Supercomputer
 - Processor technology
 - **Interconnect & Network technology**
 - I/O & Storage technology
- Parallel Program Analysis

Communication

- Communication has the most impact to the performance of parallel programs (Even more critical to computing or memory).
 - Network is generally much slower than CPU
 - Communication is common to parallel programs
 - Synchronization is expensive and could grow exponentially to the number of servers



Interconnection Networks

■ Network design considerations

➤ Scalability, Performance, Resilience and Cost

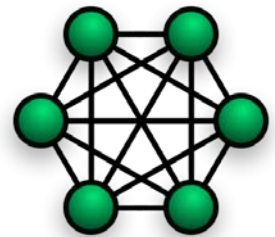
Application

- Communication pattern & protocol



Interconnection Network Topology

- Network diameter
- Re-routing path for fault tolerance
- # fan-in & fan-out degree per node



Network Devices (Cable, Switch, Adapter, etc.)

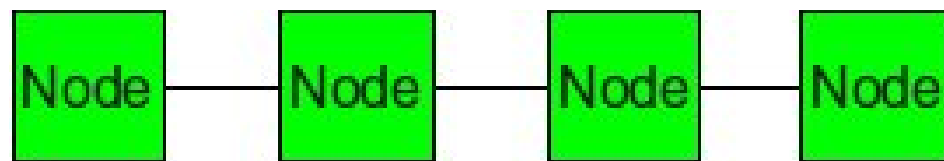
- Bandwidth: #bits transferred per second
- Latency: time to pack, unpack, and send a message
- Scalability: # of ports on the adapter and switch



Network Topology

	Diameter (latency)	Bisection (resilience)	#Links (cost)	Degree (scalability)
Linear array	P-1	1	P-1	2

- Cheapest solution, but not reliable and long latency

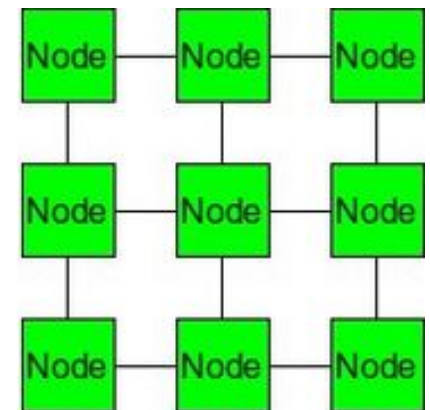


Linear Array

Network Topology

	Diameter (latency)	Bisection (resilience)	#Links (cost)	Degree (scalability)
Linear array	$P-1$	1	$P-1$	2
Ring	$p/2$	2	P	2
Tree	$2\log_2 p$	1	$2(p-1)$	3
2-D Mesh	$2(\sqrt{p} - 1)$	\sqrt{p}	$2\sqrt{p}(\sqrt{p} - 1)$	4

- Particularly suitable for some of the applications such as the ocean application and matrix calculation
- Can be extended to 3-D mesh



2-D Mesh

Network Topology

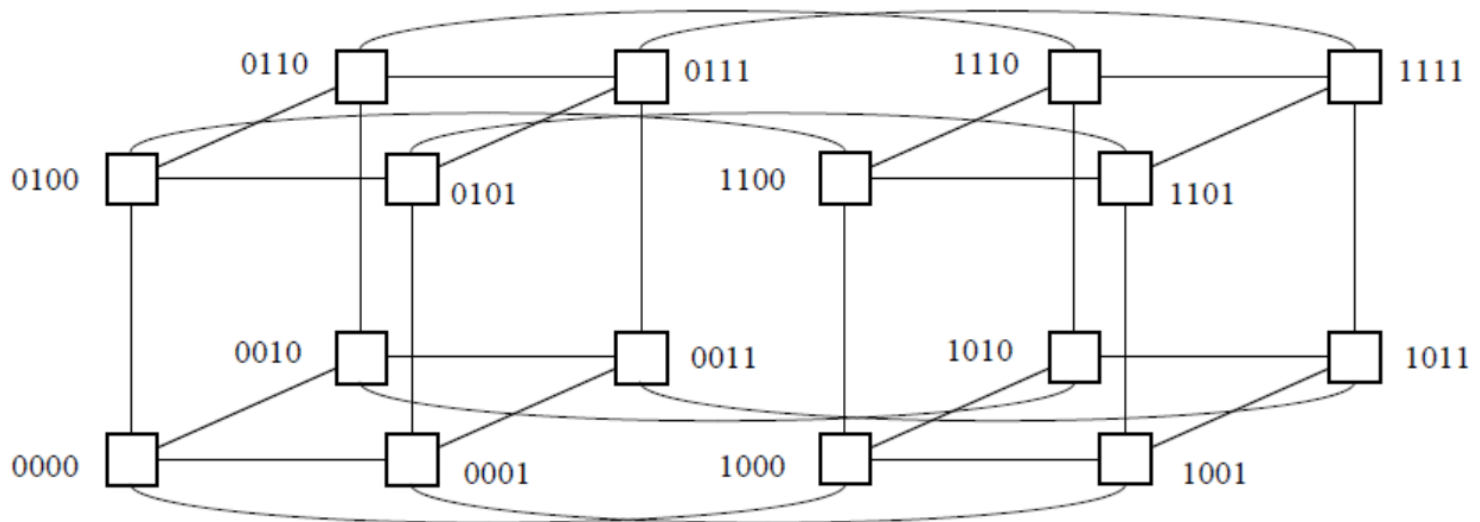
	Diameter (latency)	Bisection (resilience)	#Links (cost)	Degree (scalability)
Linear array	$P-1$	1	$P-1$	2
Ring	$p/2$	2	P	2
Tree	$2\log_2 p$	1	$2(p-1)$	3
2-D Mesh	$2(\sqrt{p} - 1)$	\sqrt{p}	$2\sqrt{p}(\sqrt{p} - 1)$	4
2-D Torus	$\sqrt{p}-1$	$2\sqrt{p}$	$2p$	4
Hypercube	$\log_2 p$	$p/2$	$p/2 \times \log_2 p$	$\log_2 p$

- Smaller diameter, more bisection, but also higher cost and degree than Mesh and Torus
- More suitable for smaller scale systems

Network Topology

■ 4-D hypercube

- Each node is numbered with a bitstring that is $\log_2(p)$ bits long.
- One bit can be flipped per hop so the diameter is $\log_2(p)$.



6-Dimensional Mesh/Torus on K-Computer

■ K-computer (Kei means “京”)

- Designed by FUJITSU, Japan
- World's #5 fastest supercomputer
- 80,000 compute nodes; 640,000 cores
- Network connection: Tofu



"6-dimensional mesh/torus" topology
(model)

■ Introduction video clip:

- <http://www.fujitsu.com/global/about/businesspolicy/tech/k/whatis/network/>

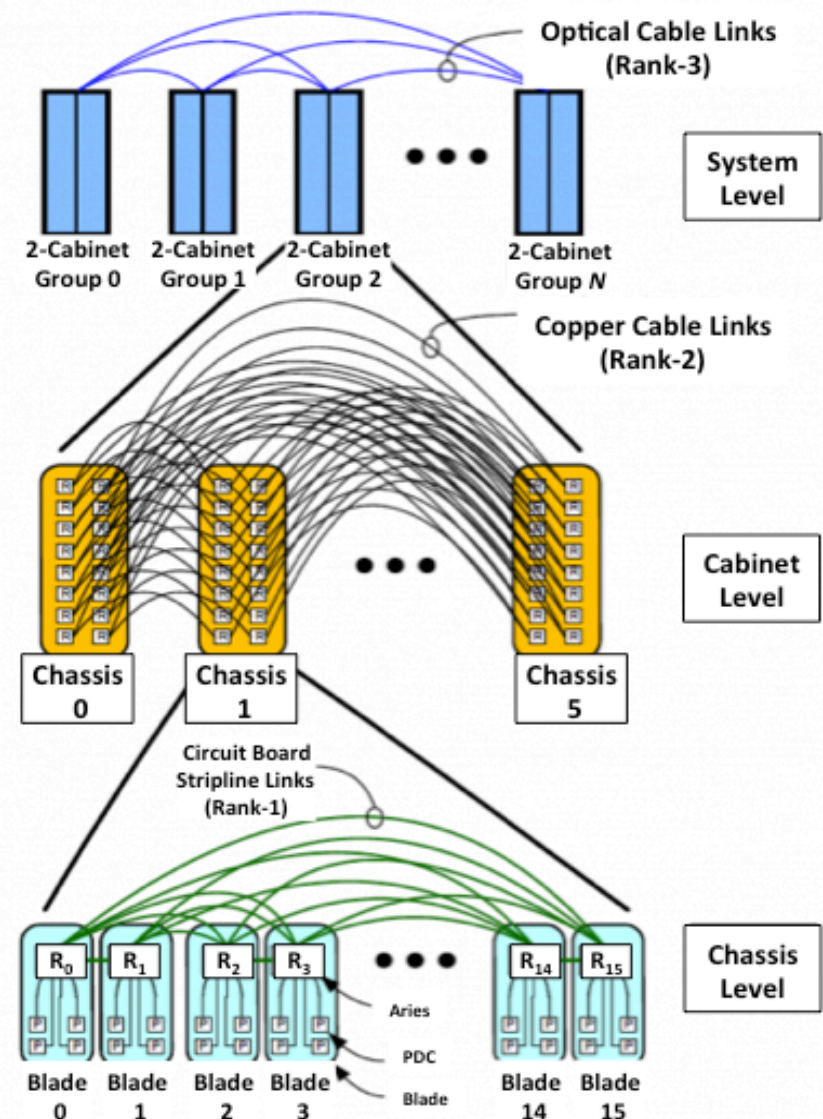


©RIKEN

Interconnection Network

- Dragonfly topology:
 - Take advantage of the technology advancement of **high-radix router, long channel optical fiber**
 - Start being used by the newest supercomputer today

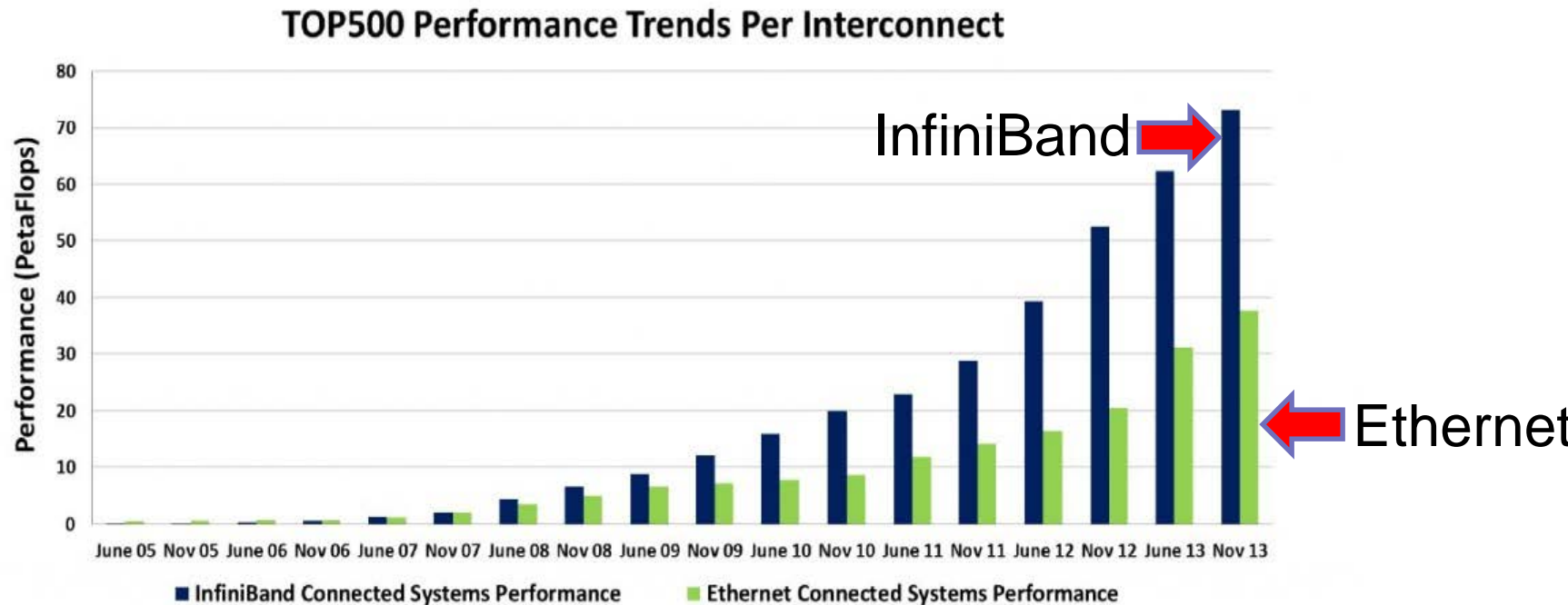
<http://www.nersc.gov/users/computational-systems/edison/configuration/interconnect/>



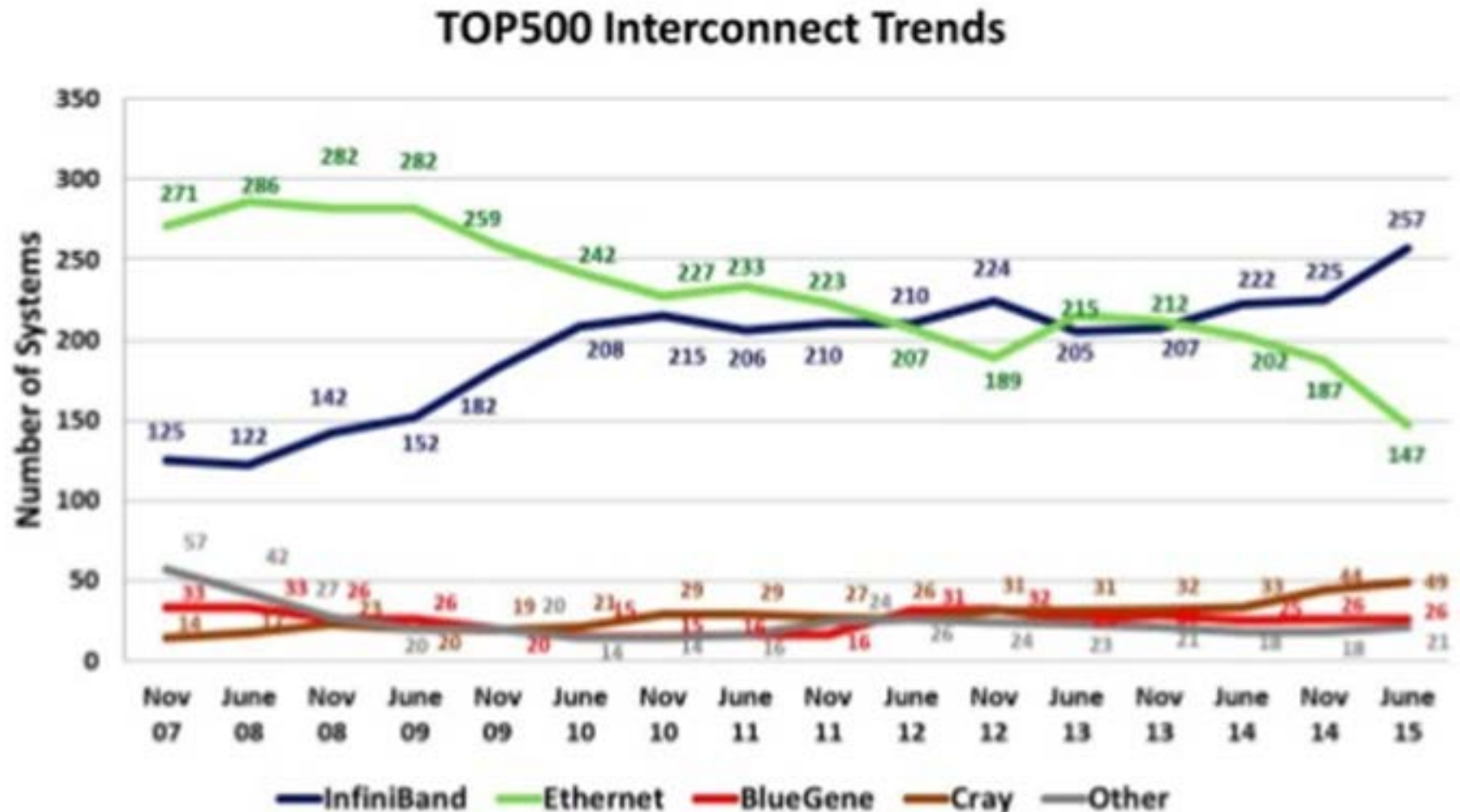
Network Device: InfiniBand



- A computer network communications link used in **high-performance computing** featuring very **high throughput**
- It is the most commonly used interconnect in supercomputers
- Manufactured by **Mellanox**

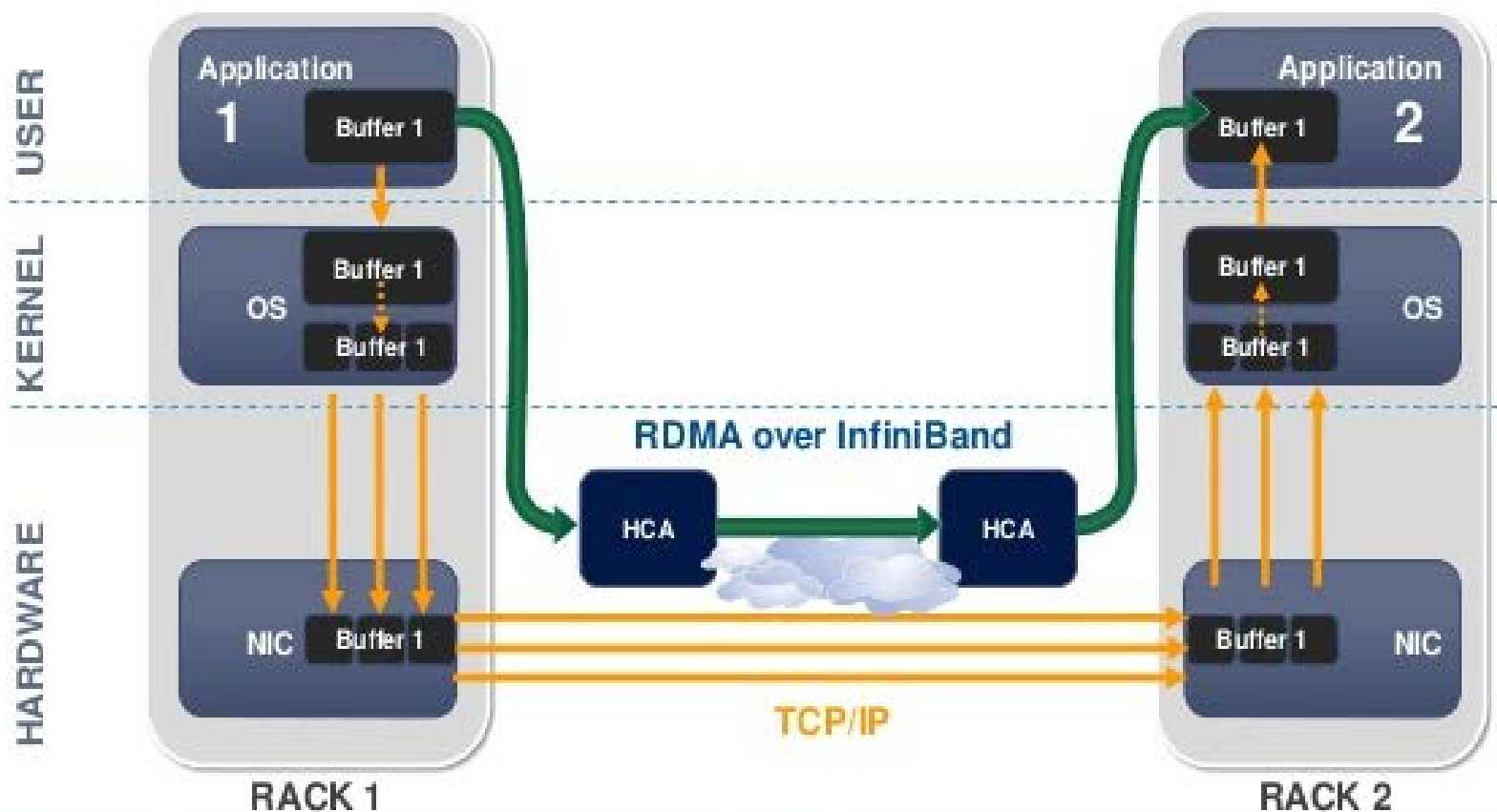


InfiniBand: Usage in TOP500



InfiniBand: RDMA

RDMA – How Does it Work



InfiniBand vs. Gigabit Ethernet

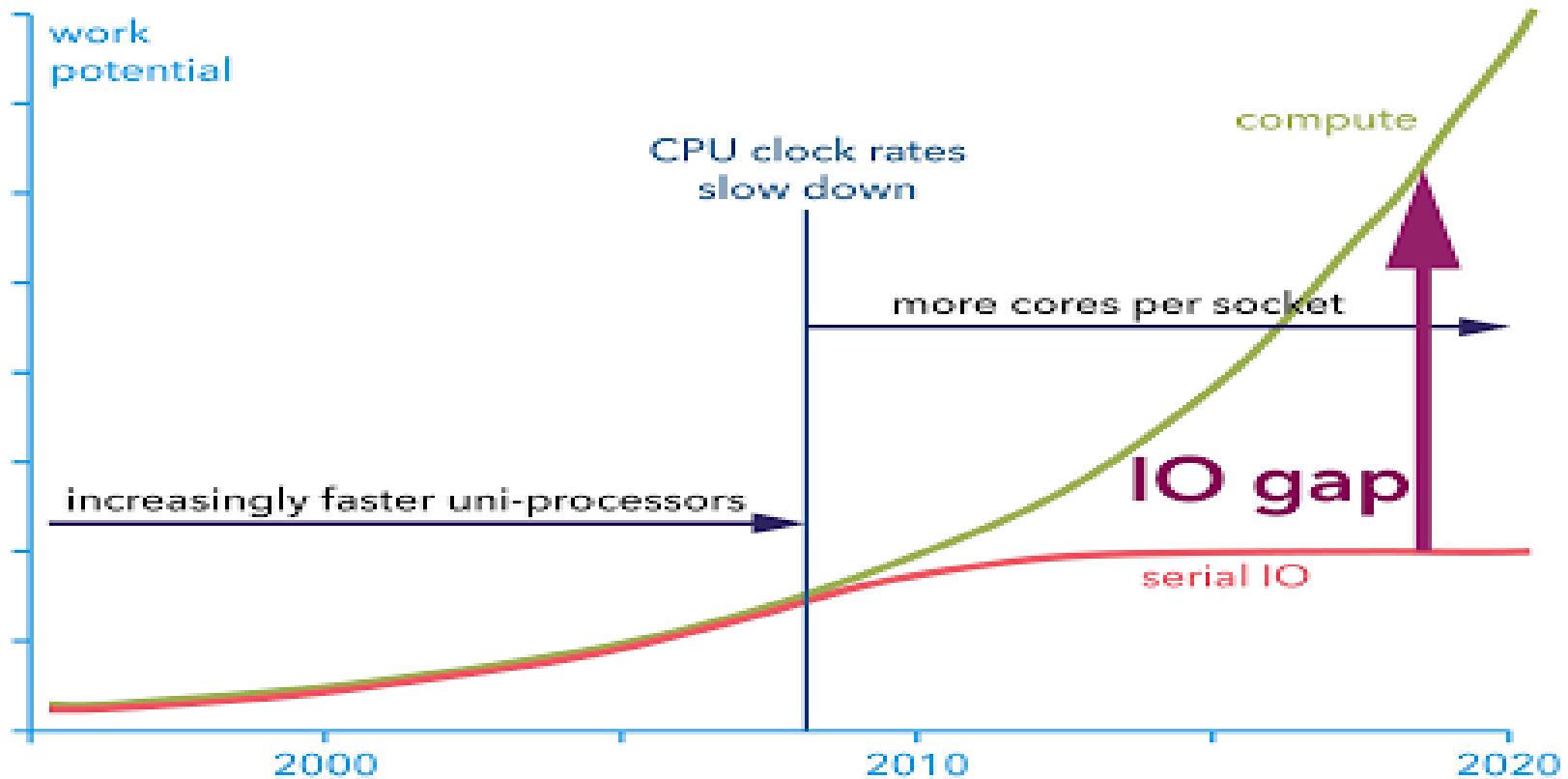
	InfiniBand	Ethernet
Protocol	Guaranteed credit based flow control	Best effort delivery
	End-to-End congestion management	TCP/IP protocol. Designed for L3/L4 switching
	Hardware based retransmission	Software based retransmission
RDMA	YES	NO (only now starting)
Latency	Low	High
Throughput	High	Low
Max cable length	4km	upto 70km
Price	36port switch: 25k USD QDR adapter: 500USD	36port switch: 1.5k USD Network card: 50 USD

Outline

- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
- **Supercomputer & Latest technologies**
 - Supercomputer
 - Processor technology
 - Interconnect & Network technology
 - **I/O & Storage technology**
- Parallel Program Analysis

How About I/O?

■ Not so great...

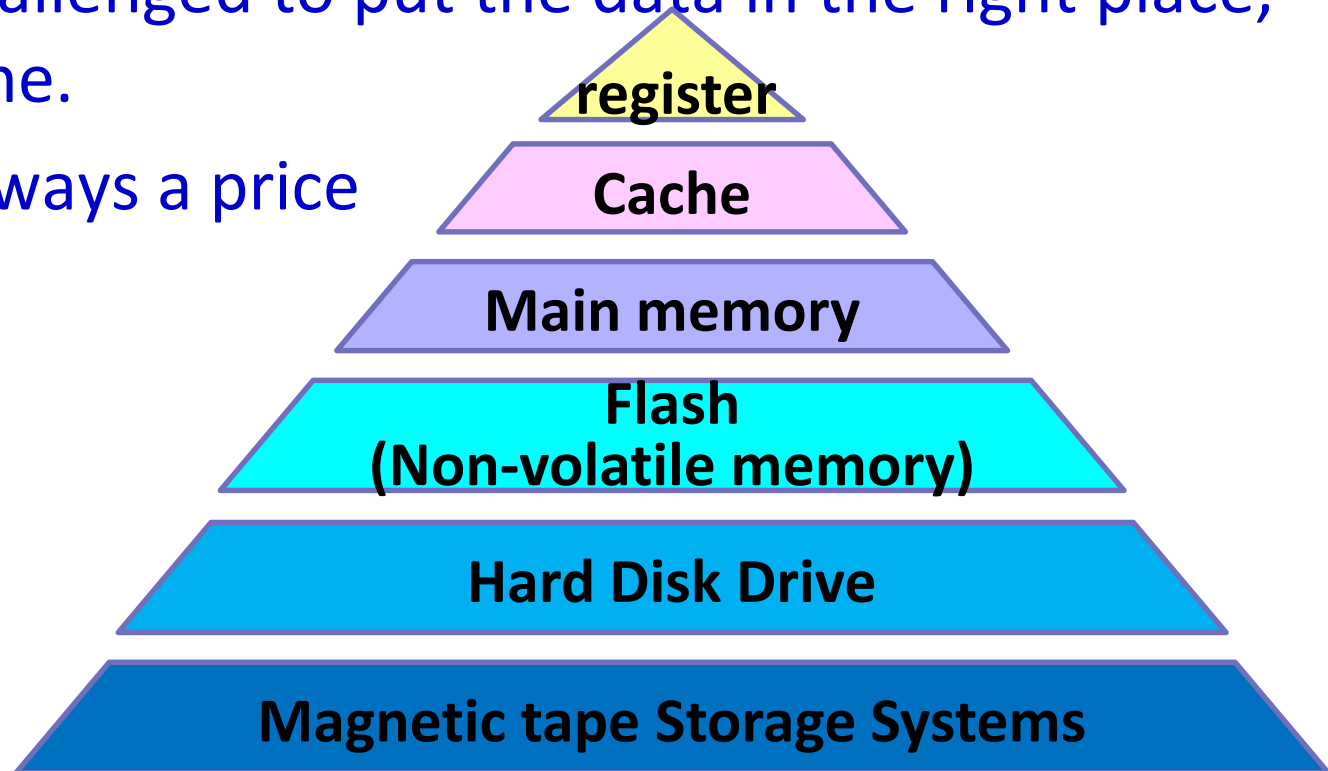


Source: http://www.mostlycolor.ch/2015_10_01_archive.html

Opportunity in I/O

■ Memory hierarchy

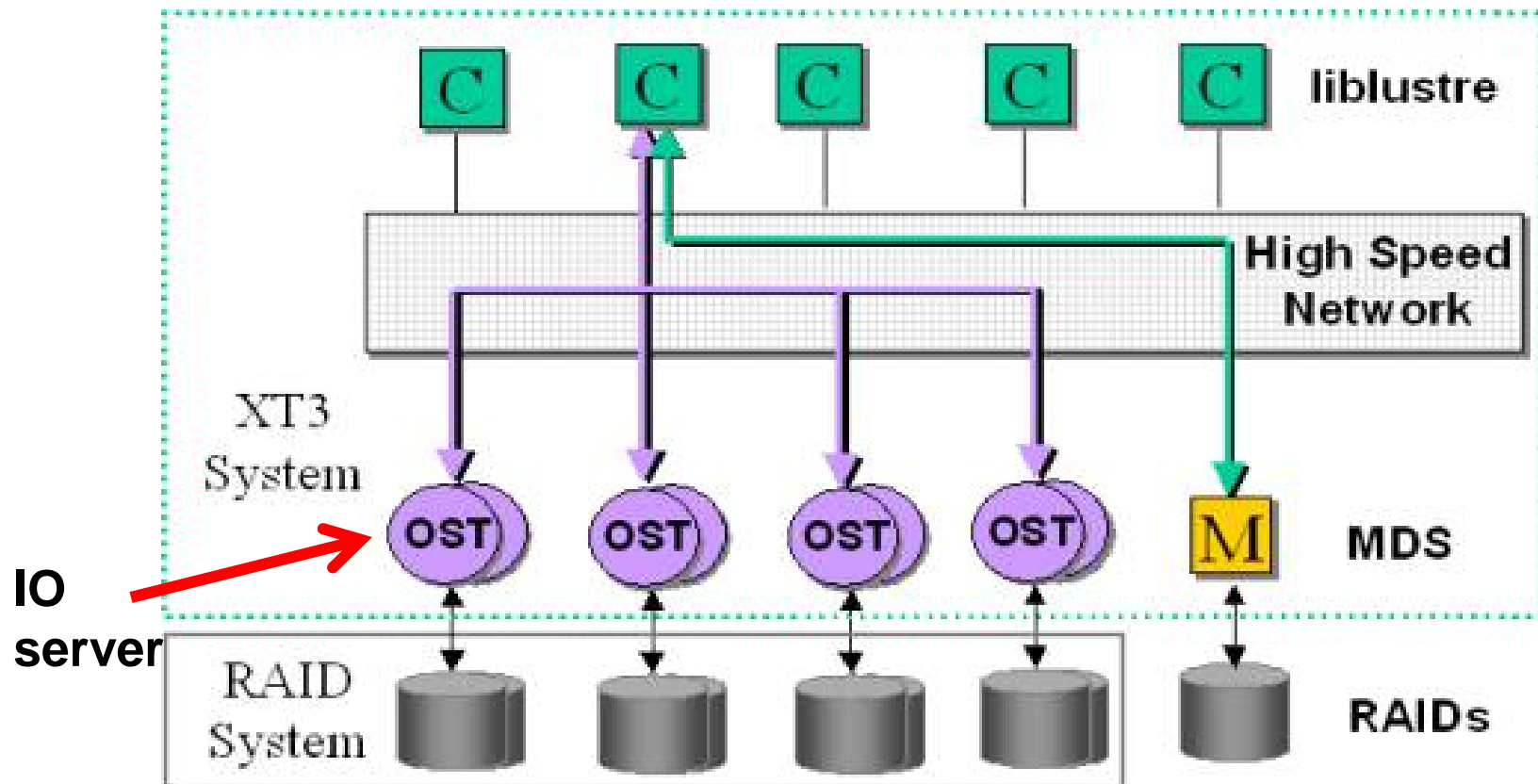
- New storage technology is coming: **Flash**
- It is still challenged to put the data in the right place, at right time.
- There is always a price to pay



Opportunity in I/O

■ Parallel file and IO systems

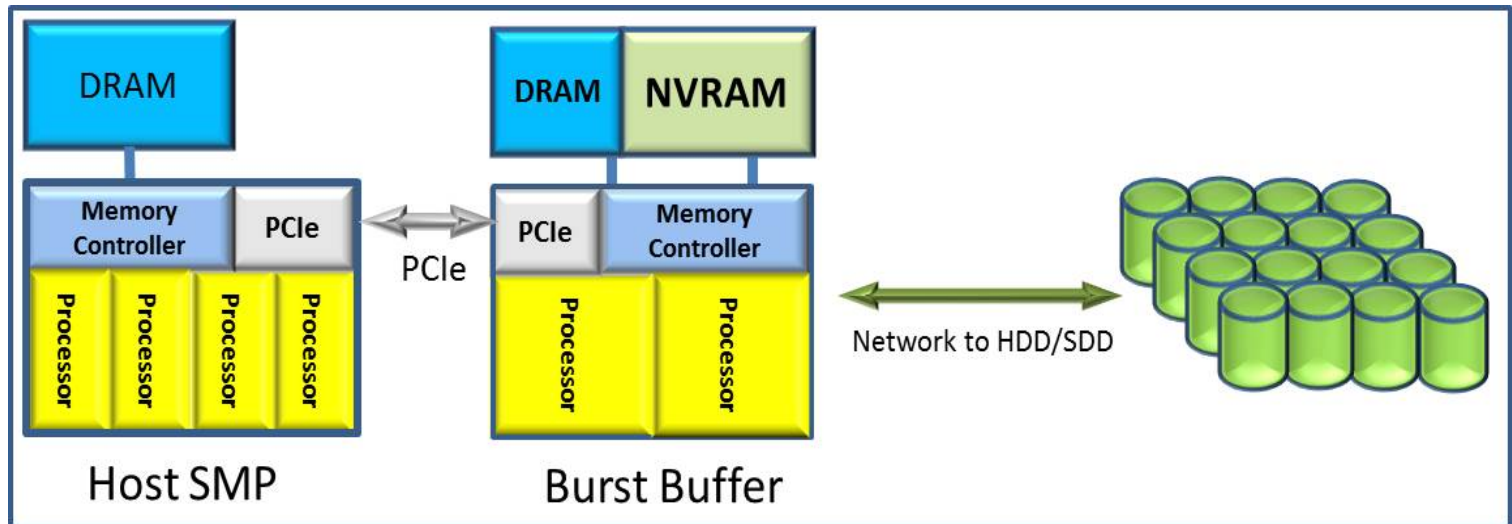
➤ Lustre file system, MPI-IO



Opportunity in I/O

■ Burst buffering

- Add non-volatile RAM at the IO server nodes as a buffer to smooth the burst traffic pattern for improving the IO performance of storage systems, and reduce the IO latency



Summary

- People has been and will always be able to find a way to keep the growth of computing
 - Technology: CPU scaling, distributed computing, new processor architecture
 - Optimization: algorithm, data management, compiler
 - System design: network topology, file system
- It is more than just computing
 - Networks and IO become greater concerns
- Does the performance report from supercomputers really meets the needs of applications?
 - People start re-thinking what should be the right **objective and benchmark for designing the next generation of supercomputers.**

Outline

- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
- Supercomputer & Latest technologies
- **Parallel Program Analysis**
 - Speedup & Efficiency
 - Strong scalability vs. Weak scalability
 - Time complexity & Cost optimality

Speedup Factor

■ Program **speedup factor**: $S(p) = \frac{T_s}{T_p}$

➤ T_s : execution time using the **BEST sequential algorithm**

➤ T_p : execution time using **p processor**

■ **Linear speedup**: $S(p) = p$

➤ Ideal maximum speedup in theory

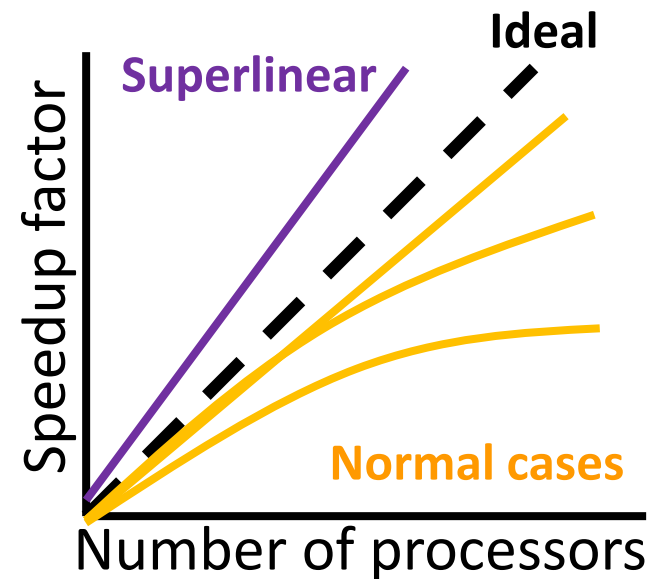
■ **Superlinear speedup**: $S(p) > p$

➤ Occasionally happen in practice

➤ Extra HW resource (e.g. memory)

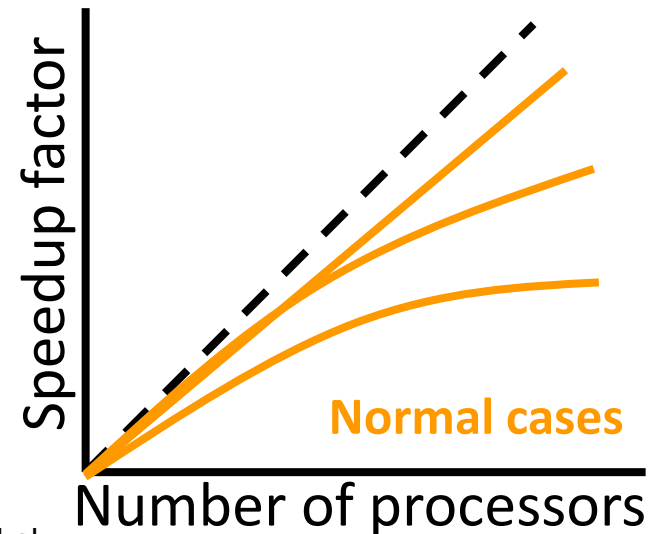
➤ SW or HW optimization (e.g. caching)

■ **System efficiency**: $E(p) = \frac{T_s}{T_p \times p} = \frac{S(p)}{p} \times 100\%$



Maximum Speedup

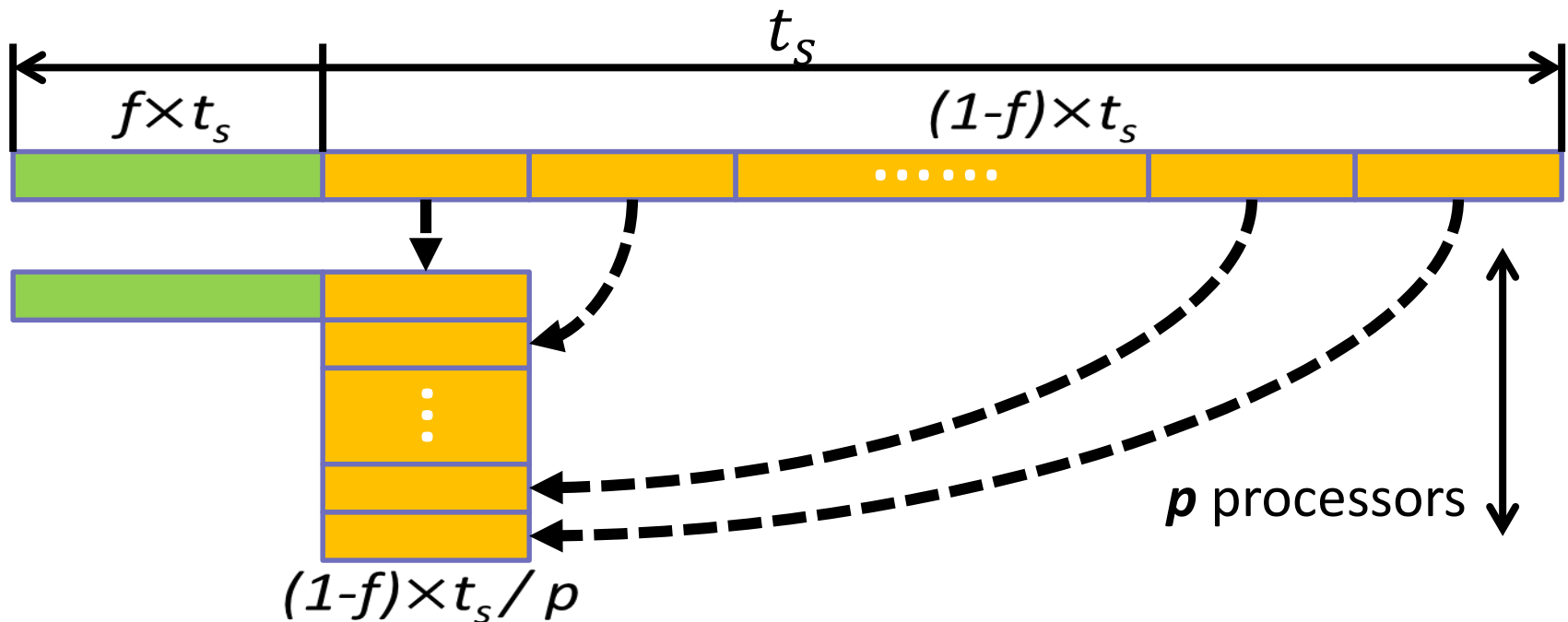
- Difficult to reach ideal max. speedup: $S(p)=p$
 - Not every part of a computation can be parallelized (results in **processor idle**)
 - Need **extra computations** in the parallel version (i.e. due to synchronization cost)
 - **Communication** time between processes (normally the major factor)



Maximum Speedup

- Let f be the fraction of computations that can **NOT** be parallelized

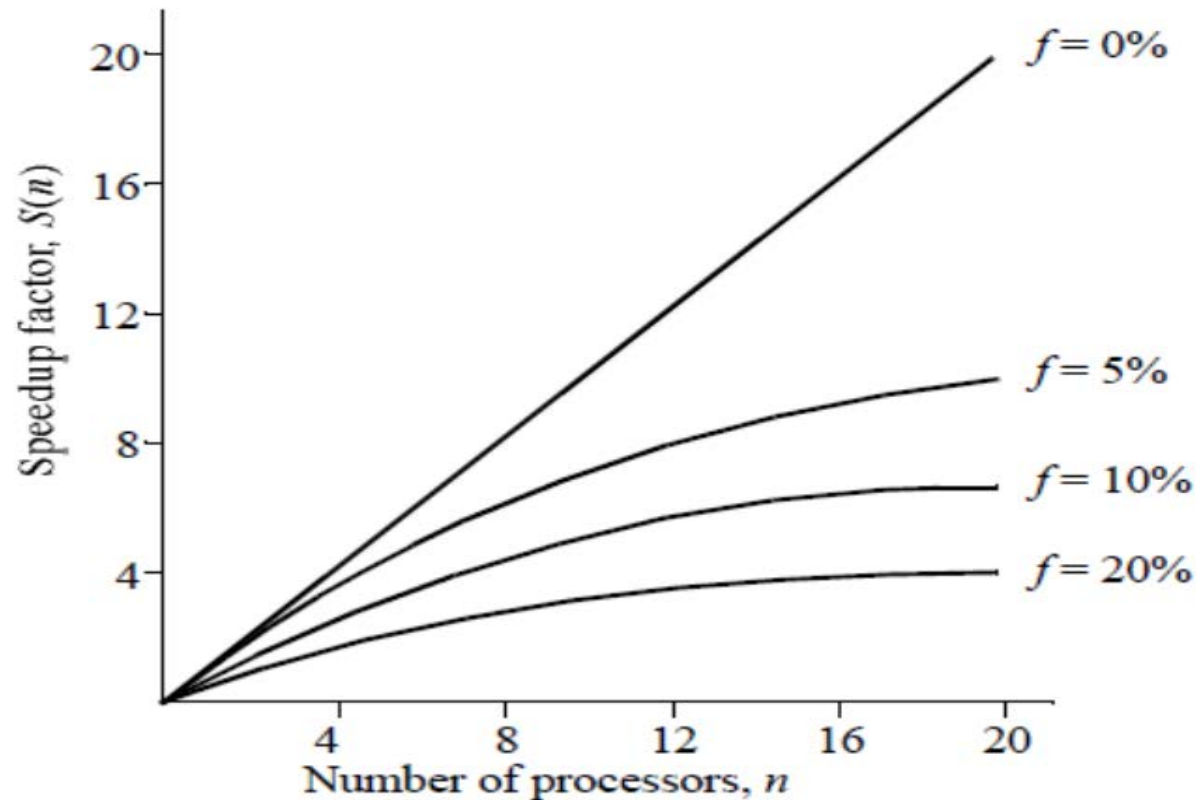
$$\triangleright S(p) = \frac{t_s}{ft_s + (1-f)t_s/p} = \frac{p}{1 + (p-1)f}$$



Maximum Speedup

- Even with infinite number of processors

➤ $S(p)_{p \rightarrow \infty} \frac{p}{1+(p-1)f} = \frac{1}{f}$

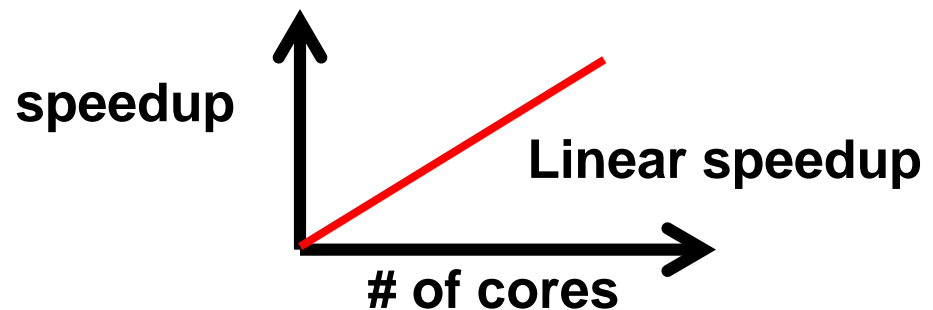
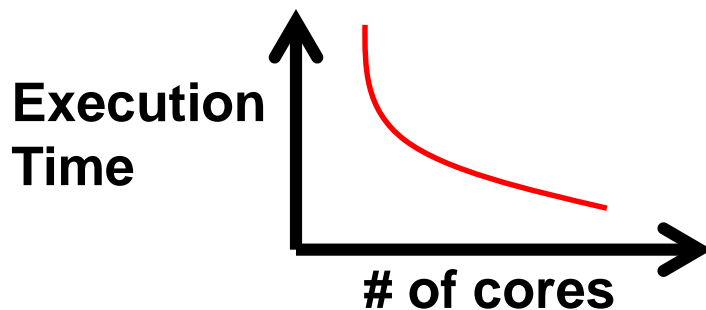


Outline

- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
- Supercomputer & Latest technologies
- **Parallel Program Analysis**
 - Speedup & Efficiency
 - **Strong scalability vs. Weak scalability**
 - Time complexity & Cost optimality

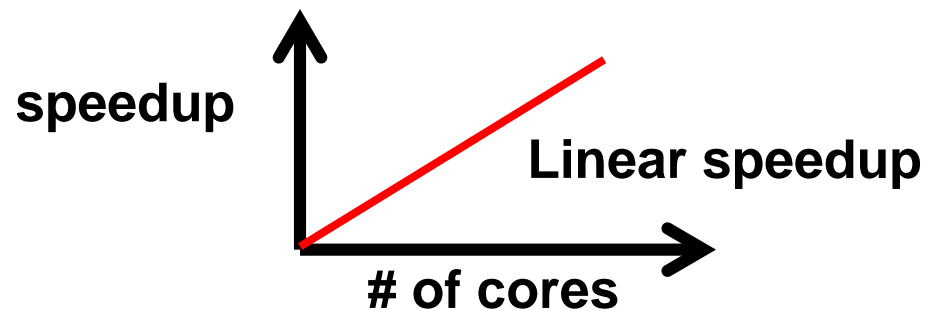
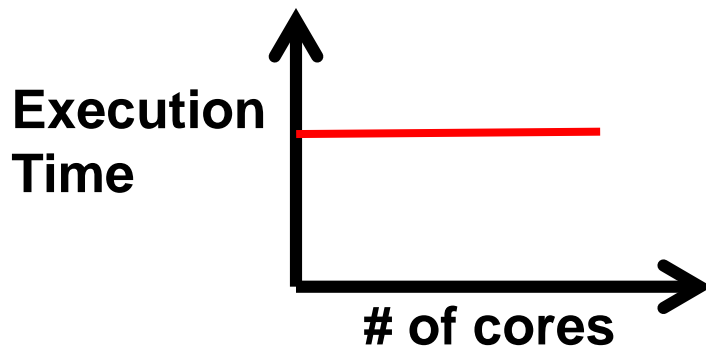
Strong Scaling

- The **problem size stays fixed** but the number of processing elements are increased.
- It is used to **find a "sweet spot"** that allows the computation to complete in a reasonable amount of time, yet does not waste too many cycles due to parallel overhead.
- **Linear scaling** is achieved if the **speedup is equal to the number of processing elements**.



Weak Scaling

- The problem size (workload) assigned to **each processing element** stays fixed and additional processing elements are used to solve a **larger total problem**
- It is a justification for programs that take a lot of memory or other system resources (e.g., a problem wouldn't fit in RAM on a single node)
- **Linear scaling is** achieved if the run time stays constant while the workload is increased



Strong Scaling vs. Weak Scaling

■ Strong scaling

- Linear scaling is harder to achieve, because of the communication overhead may increase proportional to the scale

■ Weak scaling

- Linear scaling is easier to achieve because programs typically employ nearest-neighbor communication patterns where the communication overhead is relatively constant regardless of the number of processes used

Outline

- Parallel Computing Introduction
- Classifications of Parallel Computers & Programming Models
- Supercomputer & Latest technologies
- **Parallel Program Analysis**
 - Speedup & Efficiency
 - Strong scalability vs. Weak scalability
 - **Time complexity & Cost optimality**

Time Complexity Analysis

- $T_p = T_{comp} + T_{comm}$

- T_p : Total execution time of a parallel algorithm

- T_{comp} : Computation part

- T_{comm} : Communication part

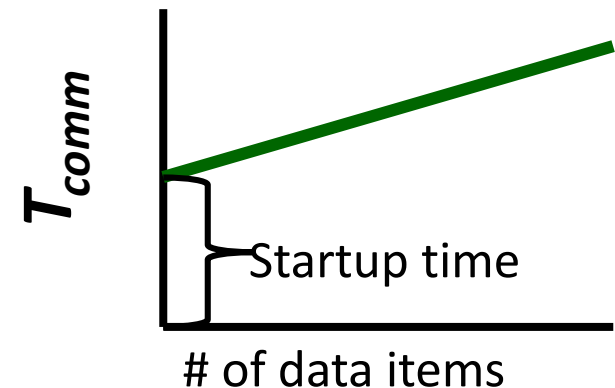
- $T_{comm} = q (T_{startup} + n T_{data})$

- $T_{startup}$: Message latency (assumed constant)

- T_{data} : Transmission time to send one data item

- n : Number of data items in a message

- q : Number of message



Time Complexity Example 1

■ Algorithm phase:

1. Computer 1 sends $n/2$ numbers to computer 2
2. Both computers add $n/2$ numbers simultaneously
3. Computer 2 sends its partial result back to computer 1
4. Computer 1 adds the partial sums to produce the final result

■ Complexity analysis:

➤ Computation (for step 2 & 4):

$$\blacklozenge T_{\text{comp}} = n/2 + 1 = O(n)$$

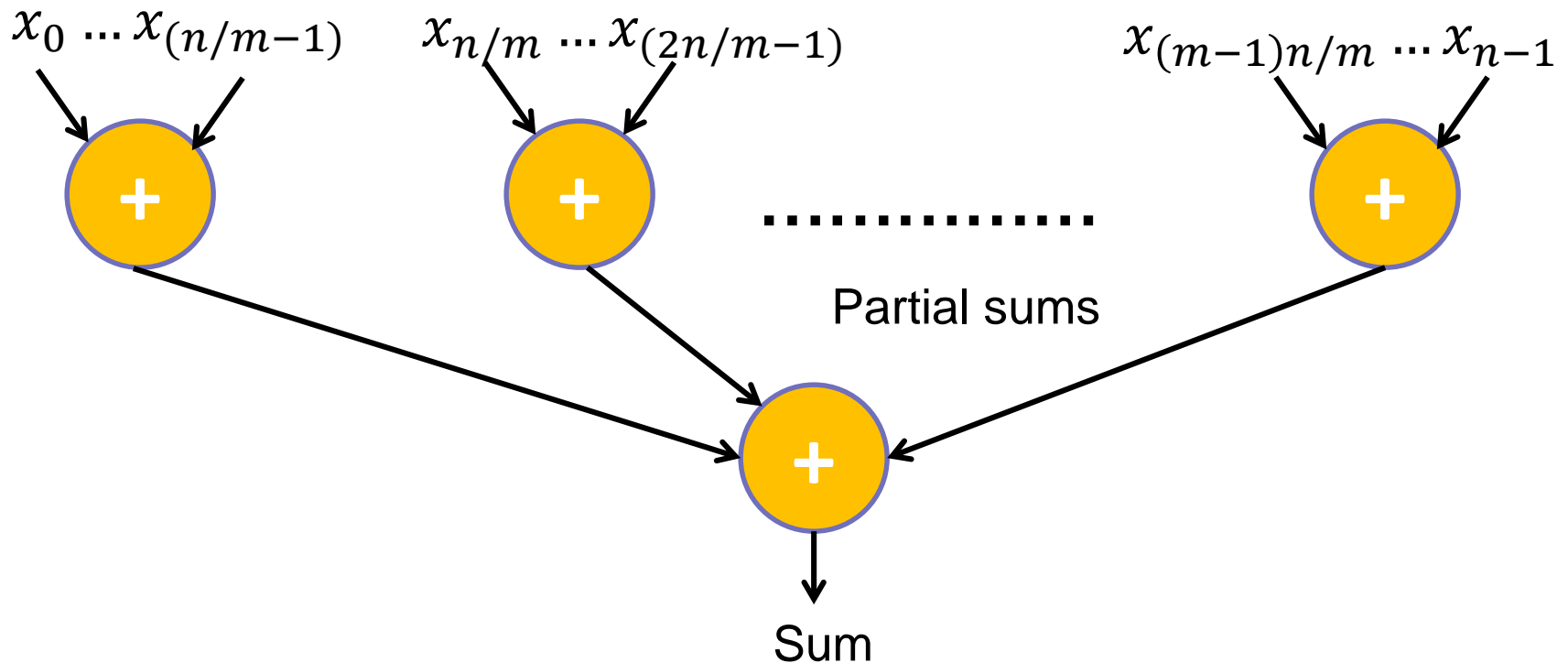
➤ Communication (for step 1 & 3):

$$\begin{aligned}\blacklozenge T_{\text{comm}} &= (T_{\text{startup}} + n/2 \times T_{\text{data}}) + (T_{\text{startup}} + T_{\text{data}}) \\ &= 2T_{\text{startup}} + (n/2 + 1) T_{\text{data}} = O(n)\end{aligned}$$

➤ Overall complexity: $O(n)$

Time Complexity Example 2

- Adding n numbers using m processes
 - Evenly partition numbers to processes



Time Complexity Example 2

■ Sequential: $O(n)$

Adding n numbers using m processes

■ Parallel:

- Phase1: Send numbers to slaves

$$t_{comm1} = m(t_{startup} + (n/m)t_{data})$$

- Phase2: Compute partial sum

$$t_{comp1} = n/m - 1$$

- Phase3: Send results to master

$$t_{comm2} = m(t_{startup} + t_{data})$$

- Phase4: Compute final accumulation

$$t_{comp2} = m - 1$$

- Overall:

$$t_p = 2mt_{startup} + (n + m)t_{data} + m + \frac{n}{m} - 2 = O(m + n/m)$$

Tradeoff
between
**computation &
communication**

Cost-Optimal Algorithm

■ Definition:

- Cost to solve a problem is proportional to the execution time on a single processor system
- $O(T_p) \times N = O(T_s)$

■ Example:

- Sequential algo: $O(N \log N)$
- Parallel algo1: uses N processor with $O(\log N)$
- Parallel algo2: uses N^2 processor with $O(1)$

Reference

- Textbook: Parallel Computing Chap1
- TOP500: <https://www.top500.org/>
- Blaise Barney, Lawrence Livermore National Laboratory, Introduction to Parallel Computing, https://computing.llnl.gov/tutorials/parallel_comp/
- Flynn's taxonomy, https://en.wikipedia.org/wiki/Flynn%27s_taxonomy
- K computer, <http://www.fujitsu.com/global/about/businesspolicy/tech/k/>
- InfiniBand, <http://www.infinibandta.org/>