IN3200/IN4200: Chapter 4
Parallel computers

Textbook: Hager & Wellein, *Introduction to High Performance Computing for Scientists and Engineers*

Parallel computers

Goals:

- be able to describe shared-memory parallel architectures, including concepts of *cache coherence* and *non-uniform memory access*
- be able to describe distributed-memory parallel computer architectures and some basic communication networks
- be able to reason about performance in distributed computing, including *point-to-point messaging* and *bisection bandwidth*

Overview of the chapter

- An introduction to the fundamental variants of parallel computers
  - The *shared-memory* type
  - The *distributed-memory* type
- A glimpse at basic design rules and performance characteristics for communication networks

What is *parallel computing*?

Parallel computing—using multiple "*compute elements*" (processor cores) to solve a problem in a cooperative way.

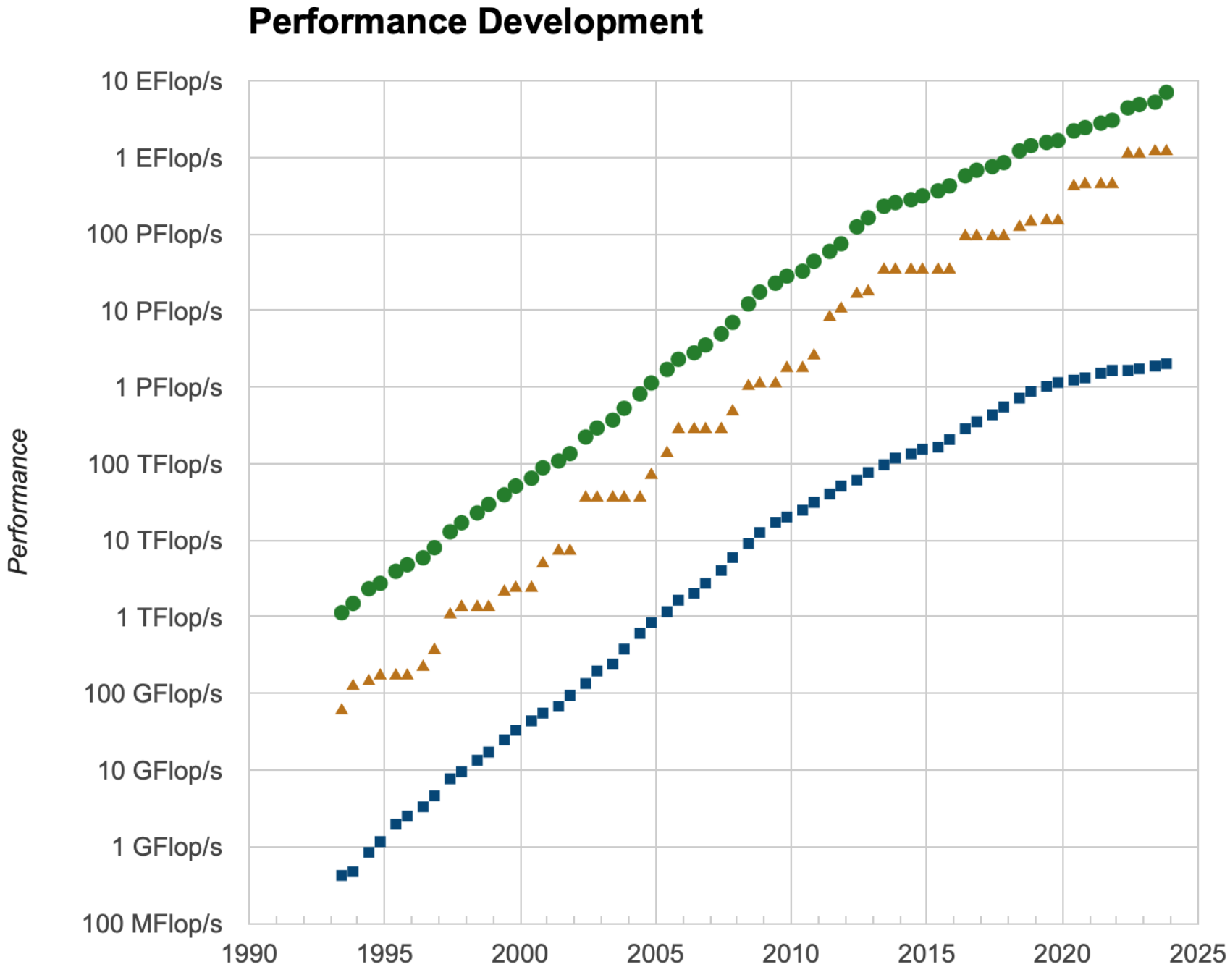
All modern supercomputer architectures depend heavily on parallelism—a large number of interconnected compute elements.

A "peek" into supercomputers through **Top500**

The Top500 list (https://www.top500.org/)

- A list of the world's 500 most powerful supercomputers
- Ranking by the measured performance of the LINPACK benchmark
  - Solve a dense system of linear equations (the system size freely adjustable)
  - Metric: number of floating-point operations executed per second
  - Mostly reflect the floating-point capability of a supercomputer
  - *Relevance of LINPACK is debatable*
- The list is updated twice a year

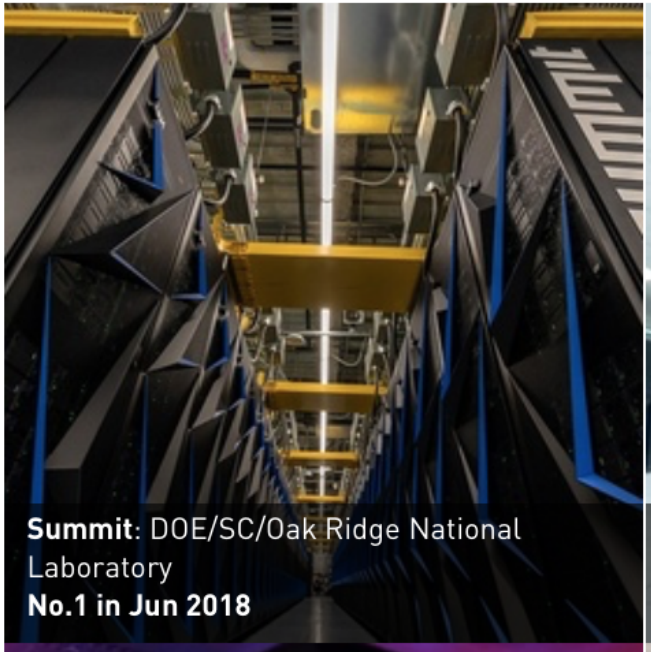
History of **Top500**



Top supercomputers of today (November 2023)

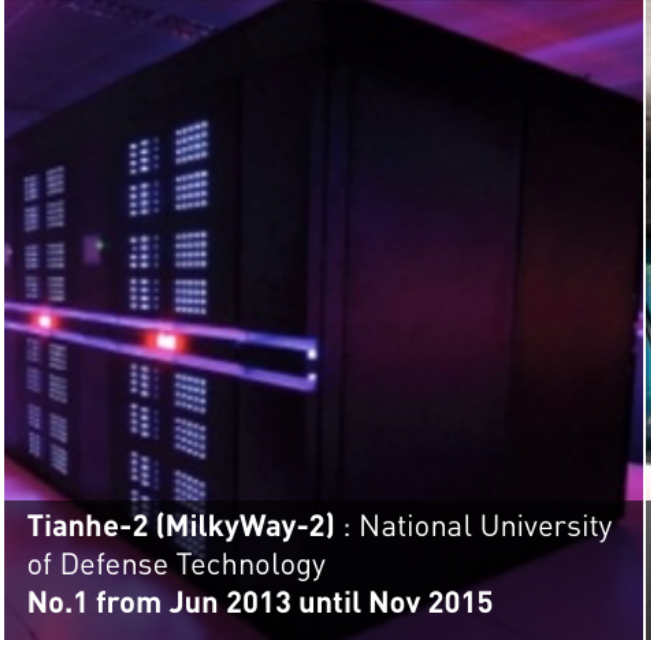
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	4,742,808	585.34	1,059.33	24,687
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Microsoft Azure United States	1,123,200	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107

Some of the previous Top1 systems

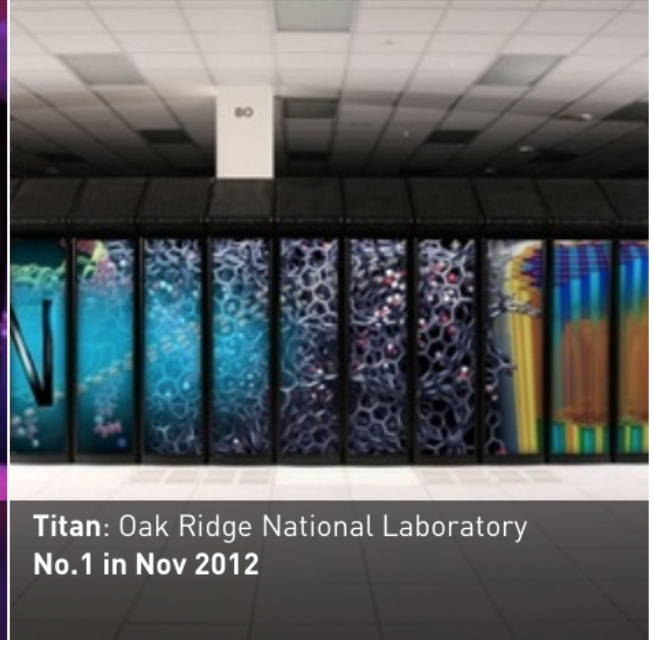


Summit: DOE/SC/Oak Ridge National Laboratory
No.1 in Jun 2018

Sunway TaihuLight: National Supercomputing Center in Wuxi
No.1 from Jun 2016 until Nov 2017



Tianhe-2 (MilkyWay-2) : National University of Defense Technology
No.1 from Jun 2013 until Nov 2015



Titan: Oak Ridge National Laboratory
No.1 in Nov 2012

Taxonomy of parallel computing paradigms

Dominating concepts:

- **SIMD** (*Single Instruction, Multiple Data*)—A single instruction stream, either on a single processor (core) or on multiple computing elements, provides parallelism by operating on multiple data streams concurrently. (Hardware examples: vector processors, SIMD-capable modern superscalar microprocessors and GPUs.)
- **MIMD** (*Multiple Instruction, Multiple Data*)—Multiple instructions streams on multiple processor (cores) operate on different data items concurrently. (Hardware examples: shared-memory and distributed-memory parallel computers.)

The focus of this chapter is on multiprocessor MIMD parallelism.

Shared-memory computers

A *shared-memory parallel computer* has a number of CPUs (cores) that work on a shared virtual address space.

Two varieties:

- *Uniform Memory Access* (UMA) systems have a "flat" memory model: latency and bandwidth are the same for all processors and all memory locations. (Typically, single multicore processor chips are "UMA machines".)
- *Cache-coherent Nonuniform Memory Access* (ccNUMA) systems have a physically distributed memory that is *logically shared*. The aggregated memory appears as one single address space. Memory access performance depends on which CPU (core) accesses which parts of memory ("local" vs. "remote" memory access).

Caches are not (completely) shared

A shared-memory system, no matter UMA or ccNUMA, has multiple CPU cores.

Although there is a single address space (shared memory), there are private caches, or partially shared caches, for the different CPU cores.

Therefore, copies of the same cache line may reside in several local caches.

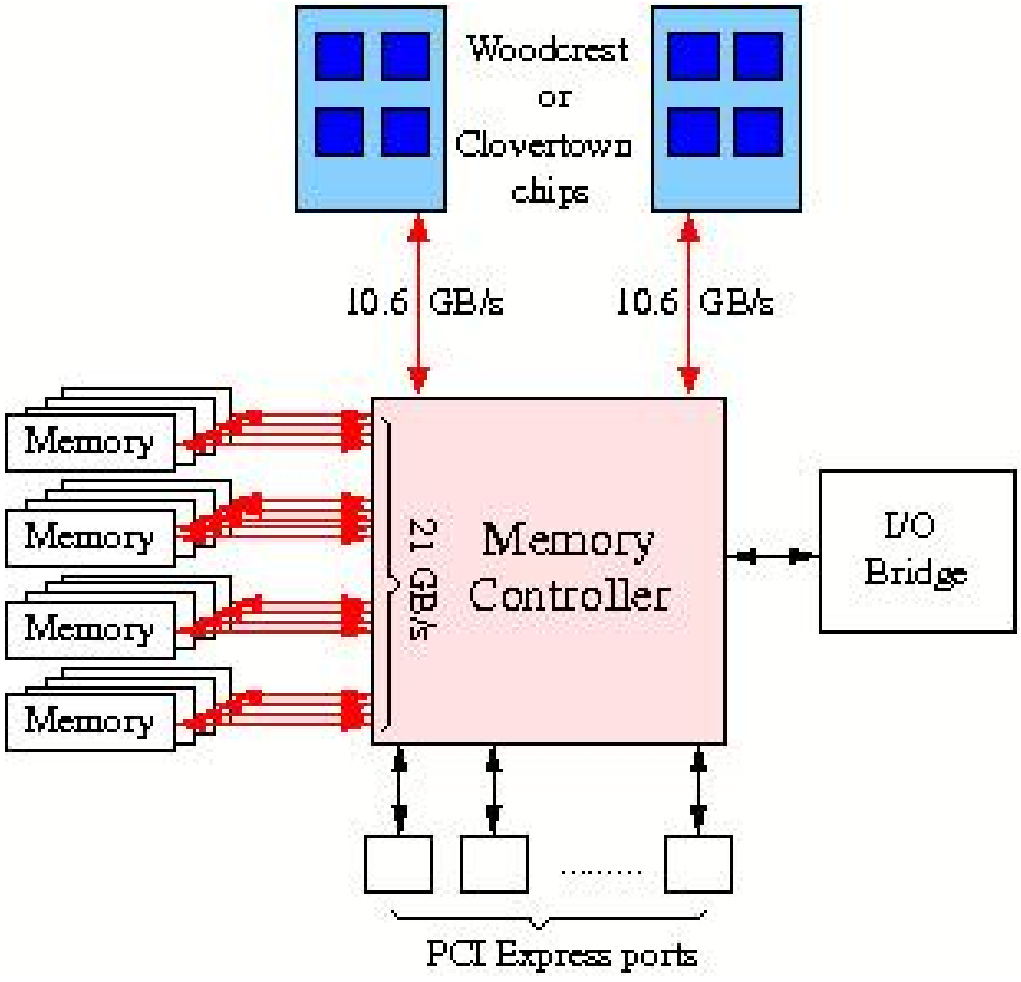
Cache coherence

Problematic situations when the same cache line resides in several caches:

- If the cache line in one of the caches is modified, the other caches' contents are *outdated* (thus invalid).
- If different parts of the same cache line are modified by different processors in their local caches $\rightarrow$ no one has the correct cache line anymore.

Cache coherence protocols (supported in hardware) guarantee *consistency* between cached data and data in the shared memory at all times, **but may bring a negative impact on performance**.

Example of UMA



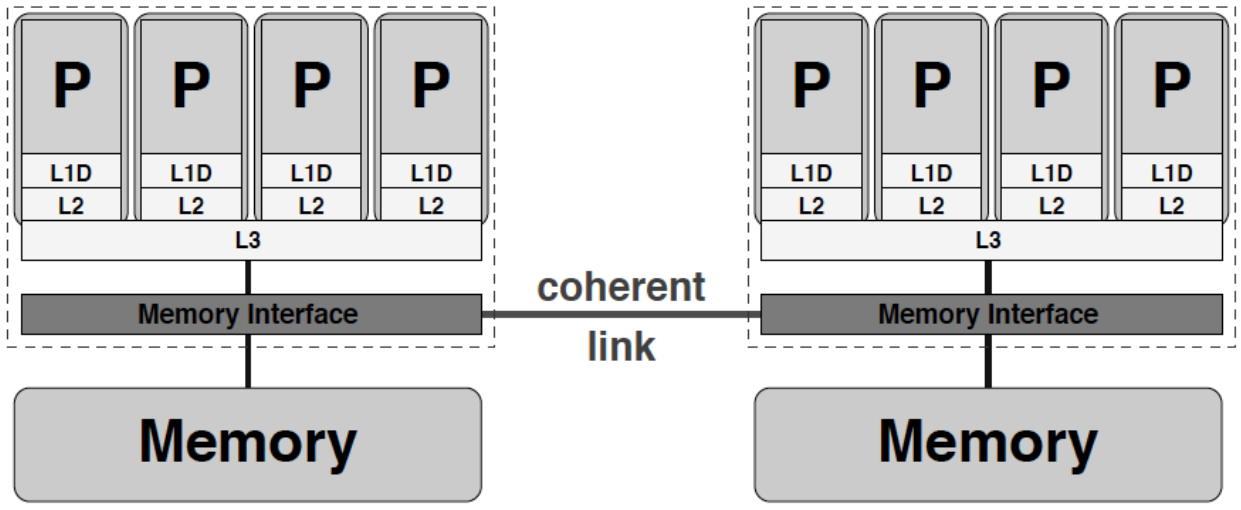
Dual-socket Xeon Clovertown CPUs

ccNUMA for scalable memory bandwidth

- A *locality domain* (LD) is a set of processor cores together with locally connected memory. This "local" memory can be accessed by the set of processor cores in the most efficient way, without resorting to a network of any kind.
- Each LD is a UMA building block.
- Multiple LDs are linked via a coherent interconnect, which can mediate cache-coherent memory accesses. (This mechanism is transparent for the programmer.)
- The whole ccNUMA system has a shared address space (memory), runs a single OS instance.

Example of ccNUMA

Figure 4.5: A ccNUMA system with two locality domains (one per socket) and eight cores.



Penalty for non-local transfers

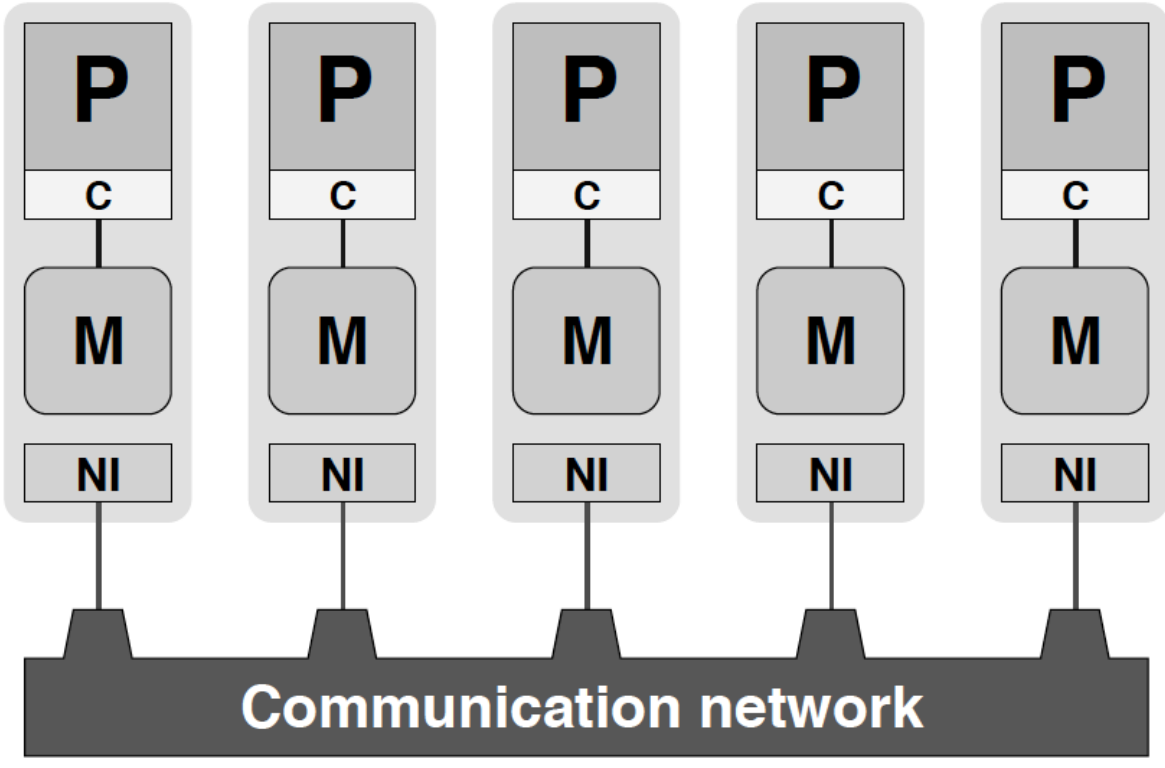
The *locality problem*: Non-local memory transfers (between LDs) are more costly than local transfers (within a LD).

The *contention problem*: If two processors from different LDs access memory in one LD, they will fight for the same memory bandwidth.

Both problems can be "solved" (alleviated) by carefully observing the data access patterns of an application and restricting data access of each processor (mostly) to its own LD, through proper programming.

A "purely" distributed-memory computer

Figure 4.7: Simplified programmer's view, or "programming model," of a distributed-memory parallel computer: Separate processes run on processors (P), communicating via interfaces (NI) over some network. No process can access another process' memory (M) directly, although processors may reside in shared memory.



"A programmer's view": Each processor is connected to its exclusive local memory (not shared by any other CPUs).

No such "purely" distributed-memory computer today.

Typical modern distributed-memory systems

A cluster of shared-memory *"compute nodes"*, interconnected via a *communication network*.

Each node comprises at least one network interface (NI) that mediates the connection to the communication network.

A serial process runs on each CPU (core). Between the nodes, processes can communicate by means of the network.

The layout and speed of the network has a considerable impact on application performance.

Hierarchical hybrid systems

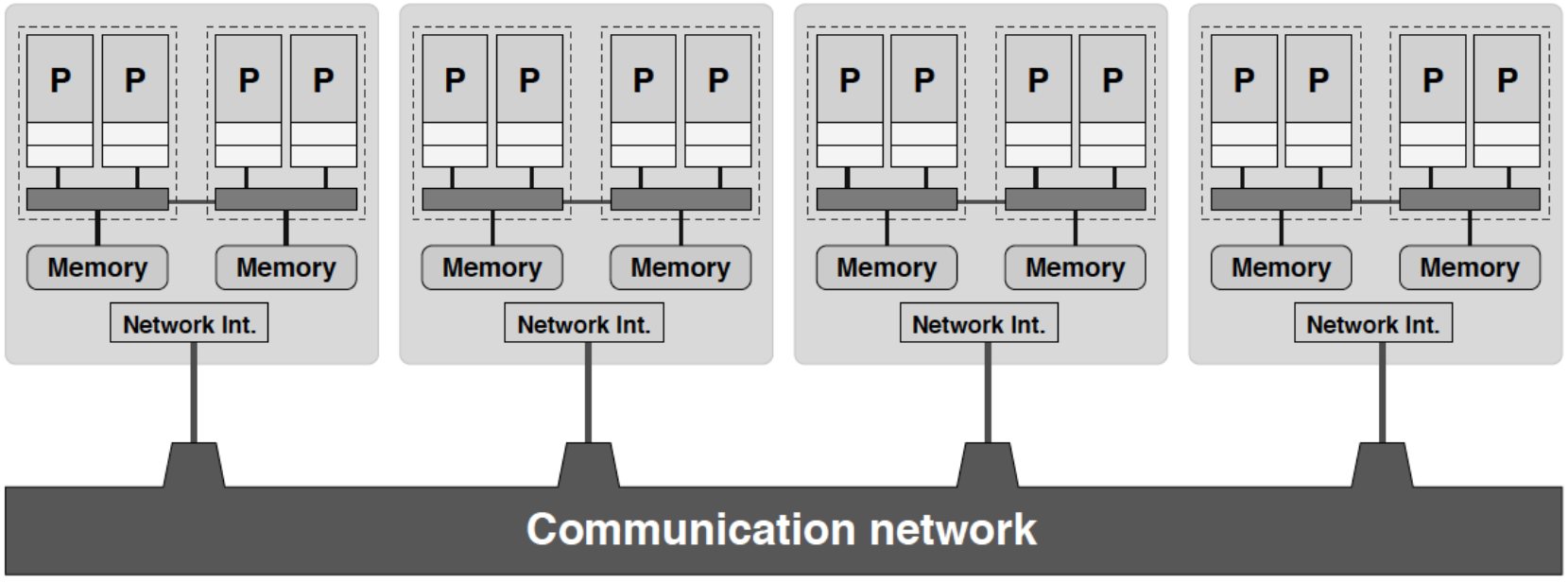


Figure 4.8: Typical hybrid system with shared-memory nodes (ccNUMA type). Two-socket building blocks represent the price vs. performance "sweet spot" and are thus found in many commodity clusters.

Networks

There are different network technologies and topologies for connecting the compute elements.

The following is a *very* brief overview of the topological and performance aspects of different types of communication networks.

Basic performance characteristics of networks

- Point-to-point communication (from one compute element to another)
- Bisection bandwith (a measure of the “whole” network)

Simple model of point-to-point communication

Time spent on transferring a message of size N [bytes] from a "sender" process to a "receiver" process:

$$T = T_\ell + \frac{N}{B}$$

This is a simplified performance model:

- T_ℓ: latency
- B: maximum network point-to-point bandwith [bytes/sec]

T_ℓ and B are considered as constants, but in reality they can both depend on N (message size), as well as on the locations of the two processes.

Effective bandwidth

Due to latency T_ℓ, the actual data transfer rate will be lower than B:

$$B_{\text{eff}} = \frac{N}{T_\ell + \frac{N}{B}}$$

The effective bandwidth B_{eff} approaches B when N is large enough.

"Ping-pong" benchmark

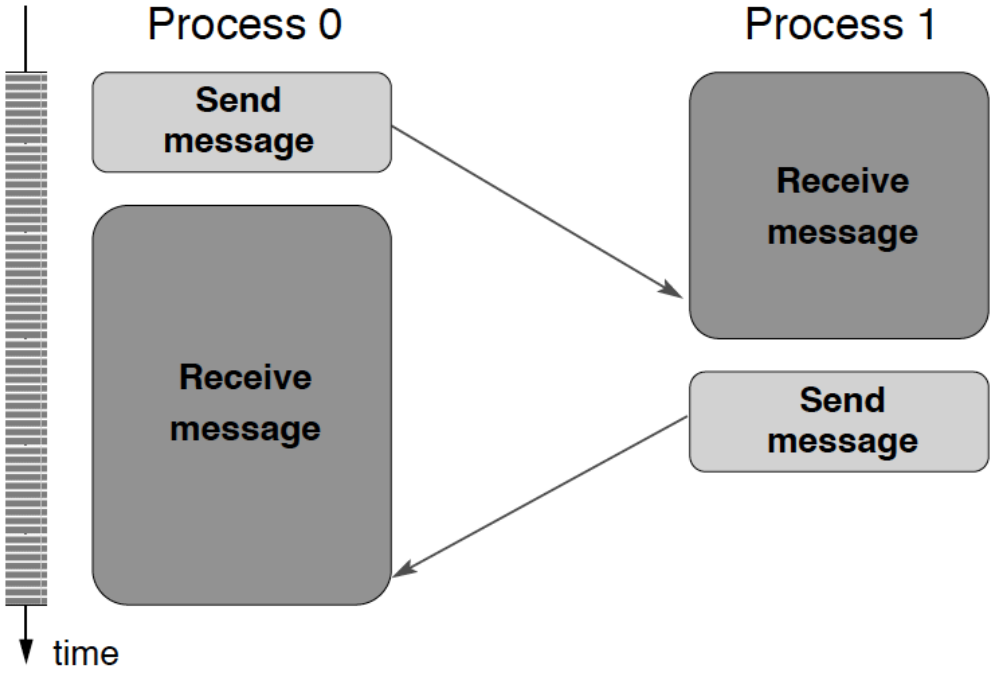


Figure 4.9: Timeline for a "Ping-Pong" data exchange between two processes. PingPong reports the time it takes for a message of length *N* bytes to travel from process 0 to process 1 and back.

"Ping-pong" benchmark (cont'd)

Pseudo code:

```
1  myID = get_process_ID()
2  if(myID.eq.0) then
3    targetID = 1
4    S = get_walltime()
5    call Send_message(buffer,N,targetID)
6    call Receive_message(buffer,N,targetID)
7    E = get_walltime()
8    MBYTES = 2*N/(E-S)/1.d6      ! MBytes/sec rate
9    TIME   = (E-S)/2*1.d6        ! transfer time in microsecs
10                                ! for single message
11 else
12   targetID = 0
13   call Receive_message(buffer,N,targetID)
14   call Send_message(buffer,N,targetID)
15 endif
```

The same code is simultaneously run by two processes.

Example of "ping-pong" measurements

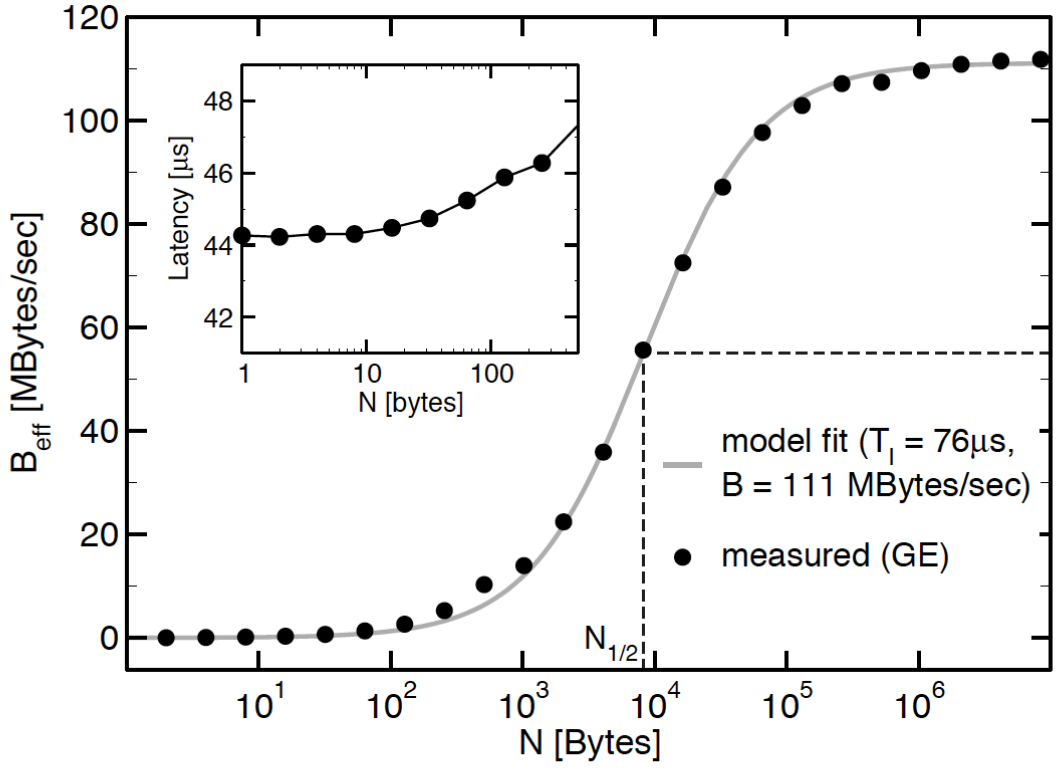


Figure 4.10: Fit of the model for effective bandwidth (4.2) to data measured on a GigE network. The fit cannot accurately reproduce the measured value of T_ℓ (see text). $N_{1/2}$ is the message length at which half of the saturation bandwidth is reached (dashed line).

B_{eff} is measured for different values of N; The values of T_ℓ and B can be *deduced* by "fitting" the measurements with the theoretical model.

Bisection bandwidth

How to quantify the "total" communication capacity of a network?

When all the compute elements are sending or receiving data at the same time:

- "competition" (even collision) may lead to that the *aggregated bandwidth*, the sum of all effective bandwidths for all point-to-point connections, is lower than the theoretical limit.

Bisection bandwidth of a network, B_b, is the sum of the bandwidths of the minimal number of connections cut when splitting the system into two *equal-sized* parts.

Illustration of bisection bandwidth

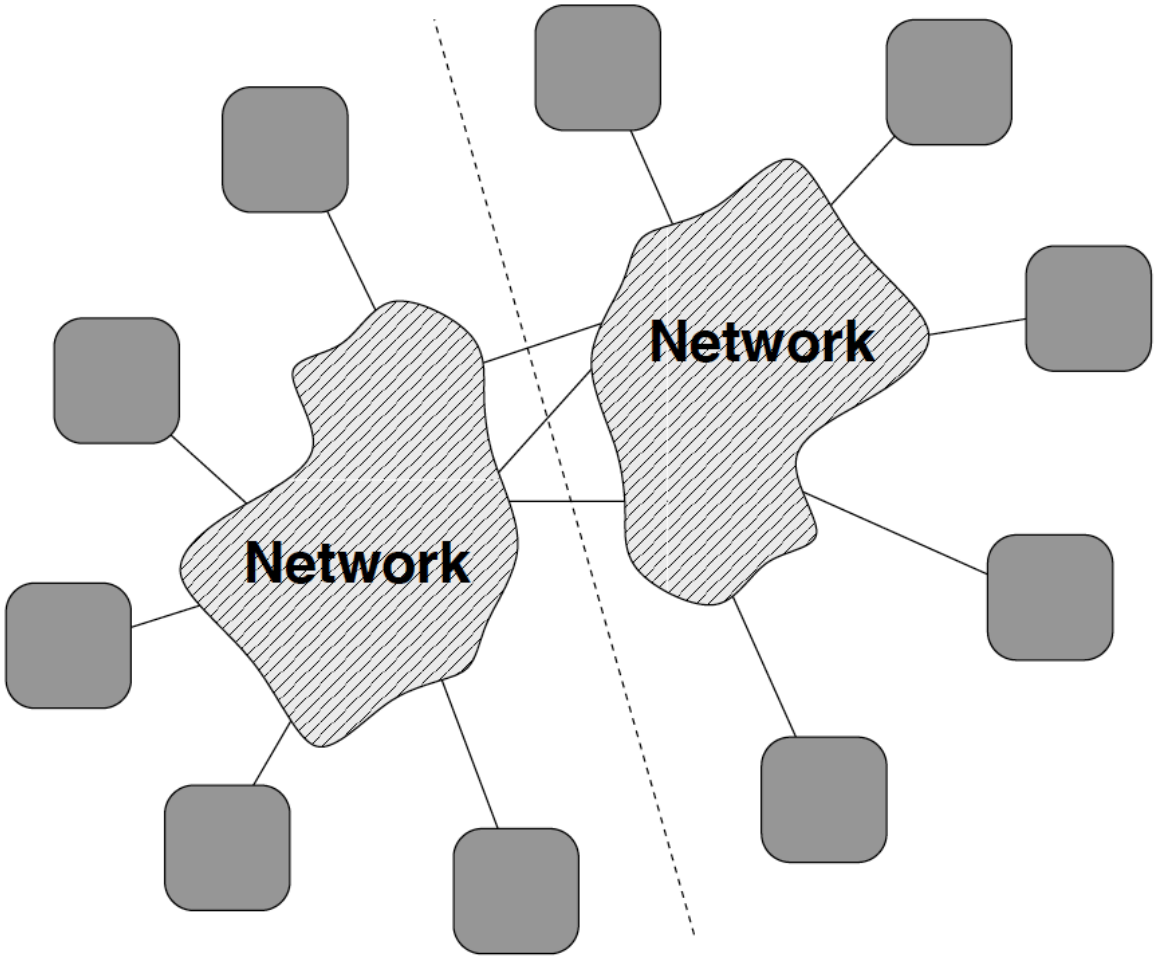


Figure 4.12: The bisection bandwidth B_b is the sum of the bandwidths of the minimal number of connections cut (three in this example) when dividing the system into two equal parts.

Different types of a communication network

- Buses
- Switched and fat-tree networks
- Mesh networks

Buses

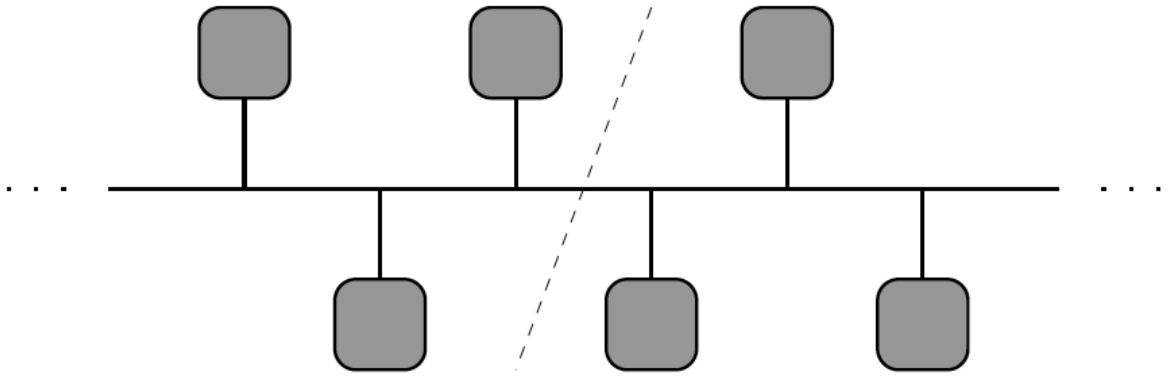


Figure 4.13: A bus network (shared medium). Only one device can use the bus at any time, and bisection bandwidth is independent of the number of nodes.

- Can be used by exactly **one** communicating device at a time.
- Easy to implement, featuring lowest latency at small utilization.
- The most important drawback is *blocking*.
- Buses are susceptible for failures.

Switched and fat-tree networks

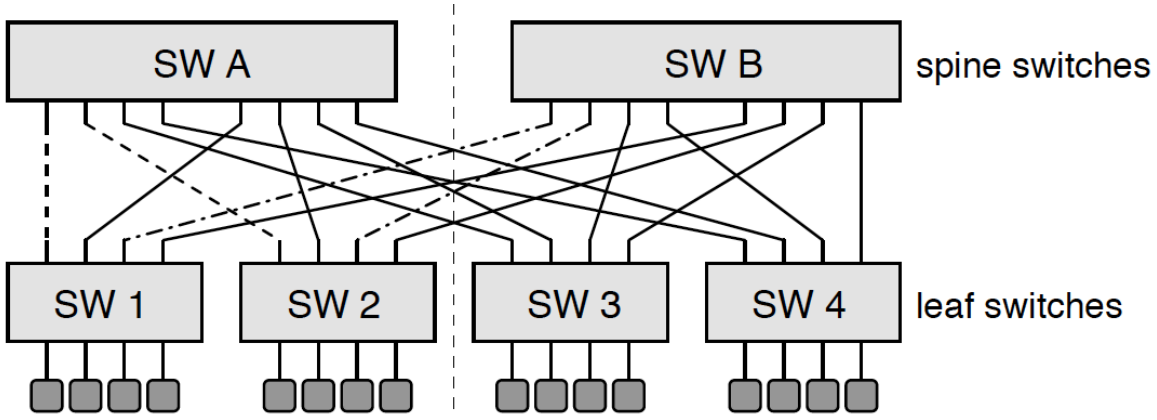


Figure 4.15: A fully nonblocking full-bandwidth fat-tree network with two switch layers. The switches connected to the actual compute elements are called *leaf switches*, whereas the upper layers form the *spines* of the hierarchy.

- All communicating devices are organized into groups.
- The devices in one group are connected to a *switch*.
- Switches are connected with each other (as a *fat-tree* hierarchy)
- The "distance" between two communicating devices—number of "hops".

Mesh networks

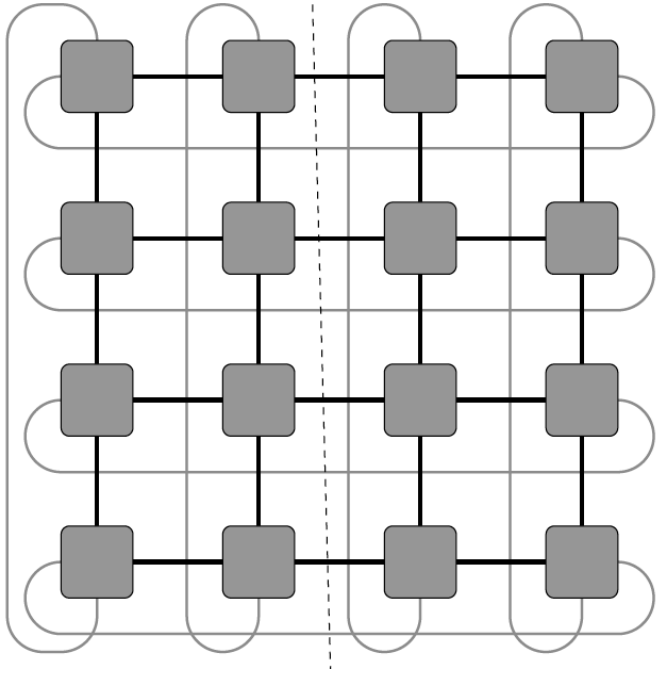


Figure 4.18: A two-dimensional (square) torus network. Bisection bandwidth scales like $\sqrt{N}$ in this case.

- In form of a multidimensional (hyper)cubes.
- Each compute element is located at a Cartesian grid intersection.
- Connections can be wrapped around the boundaries, to form a torus topology.