Parallel Computer Architectures

Zhengxiong Hou

Fall 2022

II Hardware and Parallel architectures

- **Uniprocessor Architecture** *(von Neumann)*

  -CPU, Memory, I/O and Networking

- **Parallel Architectures**

  -CPU parallelism

  (SIMD, PVP, SMP, MPP, DSM, COW; Multi-core/Many-core)

  -Memory parallelism

  Shared Memory (UMA, NUMA, CCNUMA, COMA, NORMA); Distributed Memory; Hybrid Distributed-Shared Memory

  -Interconnects/Communication Network

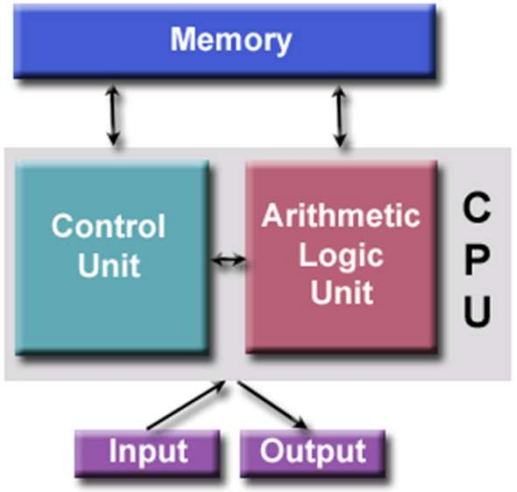
  -Parallel I/O and Networking

  -Parallel Architecture Design Tradeoffs

von Neumann Architecture

John von Neumann circa 1940s
(Source: LANL archives)

- When we talk about computer systems at large, we always have a certain architectural concept in mind. This concept was conceived by Turing in 1936, and first implemented in a real machine (EDVAC) in 1949 by Eckert and Mauchly
- Named after the Hungarian mathematician/genius John von Neumann who first authored the general requirements for an electronic computer in his 1945 papers.
- Also known as "**stored-program computer architecture**" - **both program instructions and data** are kept in electronic memory. Differs from earlier computers which were programmed through "hard wiring".
- Since then, virtually all computers have followed this basic design



- Comprised of four main components:
  - Memory
  - Control Unit
  - Arithmetic Logic Unit
  - Input/Output
- Read/write, random access memory is used to store both program instructions and data
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
- Control unit fetches instructions/data from memory, decodes the instructions and then ***sequentially*** coordinates operations to accomplish the programmed task.
- Aritmetic Unit performs basic arithmetic operations
- Input/Output is the interface to the human operator

- Programming a stored-program computer amounts to modifying instructions in memory, which can in principle be done by another program;

- a *compiler* is a typical example, because it translates the constructs of a high-level language like C or Fortran into instructions that can be stored in memory and then executed by a computer.

So what? Who cares?

- Well, parallel computers still follow this basic design, **just multiplied in units**.
- The basic, fundamental architecture remains the same.

- This blueprint is the basis for all mainstream computer systems today, and its inherent problems still prevail:

  -Instructions and data must be continuously fed to the control and arithmetic units, so that the ***speed of the memory interface*** poses a limitation on compute performance. This is often called the *von Neumann bottleneck*.

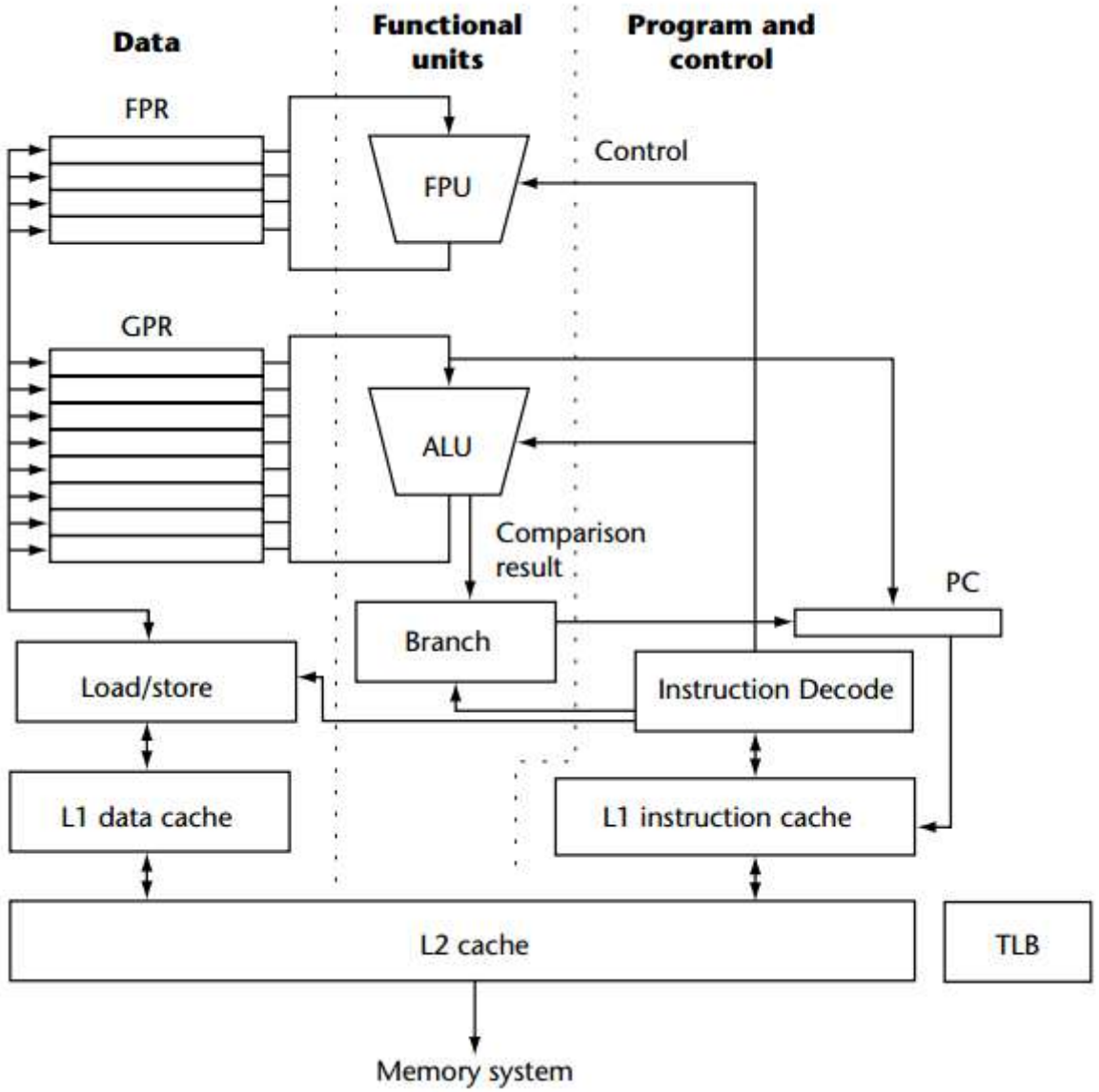
  -The architecture is inherently ***sequential***, processing a single instruction with (possibly) a single operand or a group of operands from memory. The term SISD (Single Instruction Single Data) has been coined for this concept.

- Despite these drawbacks, no other architectural concept has found similarly widespread use in nearly 70 years of electronic digital computing!

CPU

- The CPU is the heart of the computer; it is responsible for all calculations and for controlling or supervising the other parts of the computer. A typical CPU contains the following

  - *Arithmetic logic unit (ALU).* Performs computations such as addition and comparison.
  - *Floating-point unit (FPU).* Performs operations on floating-point numbers.
  - *Load/store unit.* Performs loads and stores for data.
  - *Registers.* Fast memory locations used to store intermediate results. These are often subdivided into floating-point registers (FPRs) and general purpose registers (GPRs).
  - *Program counter (PC).* Contains the address of the instruction that is executing.
  - *Memory interface.* Provides access to the memory system. In addition, the CPU chip often contains the fastest part of the memory hierarchy (the top-level



- Generic CPU diagram. This example has a separate L1 cache for data and for program instructions and a unified (both data and instructions) L2 cache. Not all data paths are shown.

Memory

- While a computer is running, active data and programs are stored in memory. Memory systems are quite complex, introducing a number of design issues. Among these are the following:

  - *Memory size*. Users never have enough computer memory, so the concept of *virtual memory* was introduced to fool programs into thinking that they have large amounts of memory just for their own use.
  - *Memory latency and hierarchy*. The time to access memory has not kept pace with CPU clock speeds. Levels or *hierarchies* of memory try to achieve a compromise between performance and cost.
  - *Memory bandwidth*. The rate at which memory can be transferred to and from the CPU (or other devices, such as disks) also has not kept up with CPU speeds.
  - *Memory protection*. Many architectures include hardware support for memory protection, aimed primarily at preventing application software from modifying (intentionally or inadvertently) either system memory or memory in use by other programs.

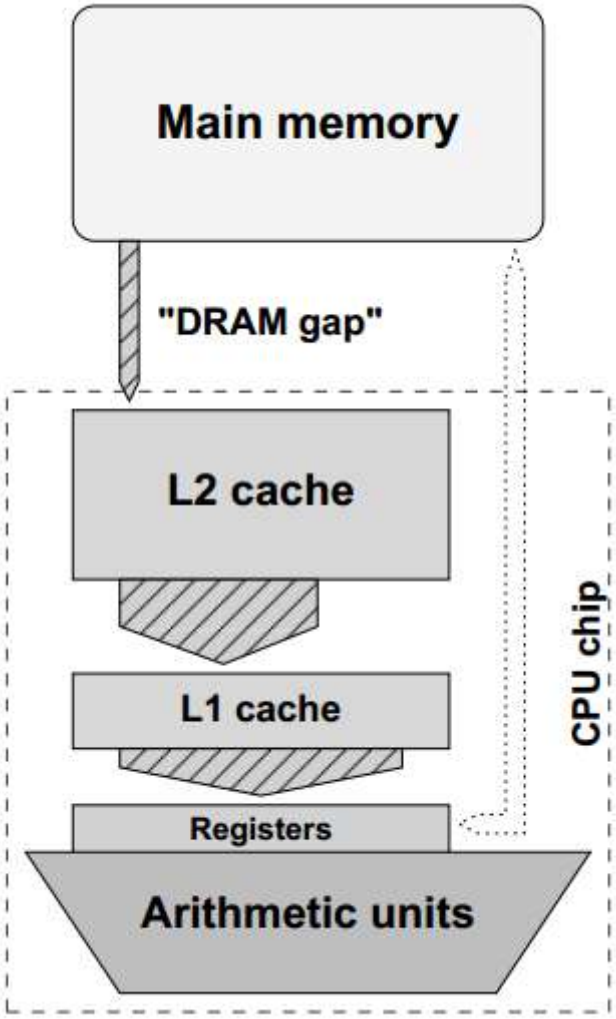
I/O

- I/O, particularly to the disks that store files and swap space for supporting virtual memory, has followed a path similar to that of main memory. That is, ***densities and sizes*** **have increased enormously, but** ***latencies*** **have remained relatively unchanged**.

- To address this issue, some of the same techniques used for memory have been adopted, particularly the use of ***caches*** (typically using DRAM memory) to improve performance.

- **Simplified data-centric memory hierarchy in a cache-based microprocessor**

  (direct access paths from registers to memory are not available on all architectures). There is usually a separate L1 cache for instructions.
- **Cache-Reducing Memory Access Times**



Caches and programs: an example

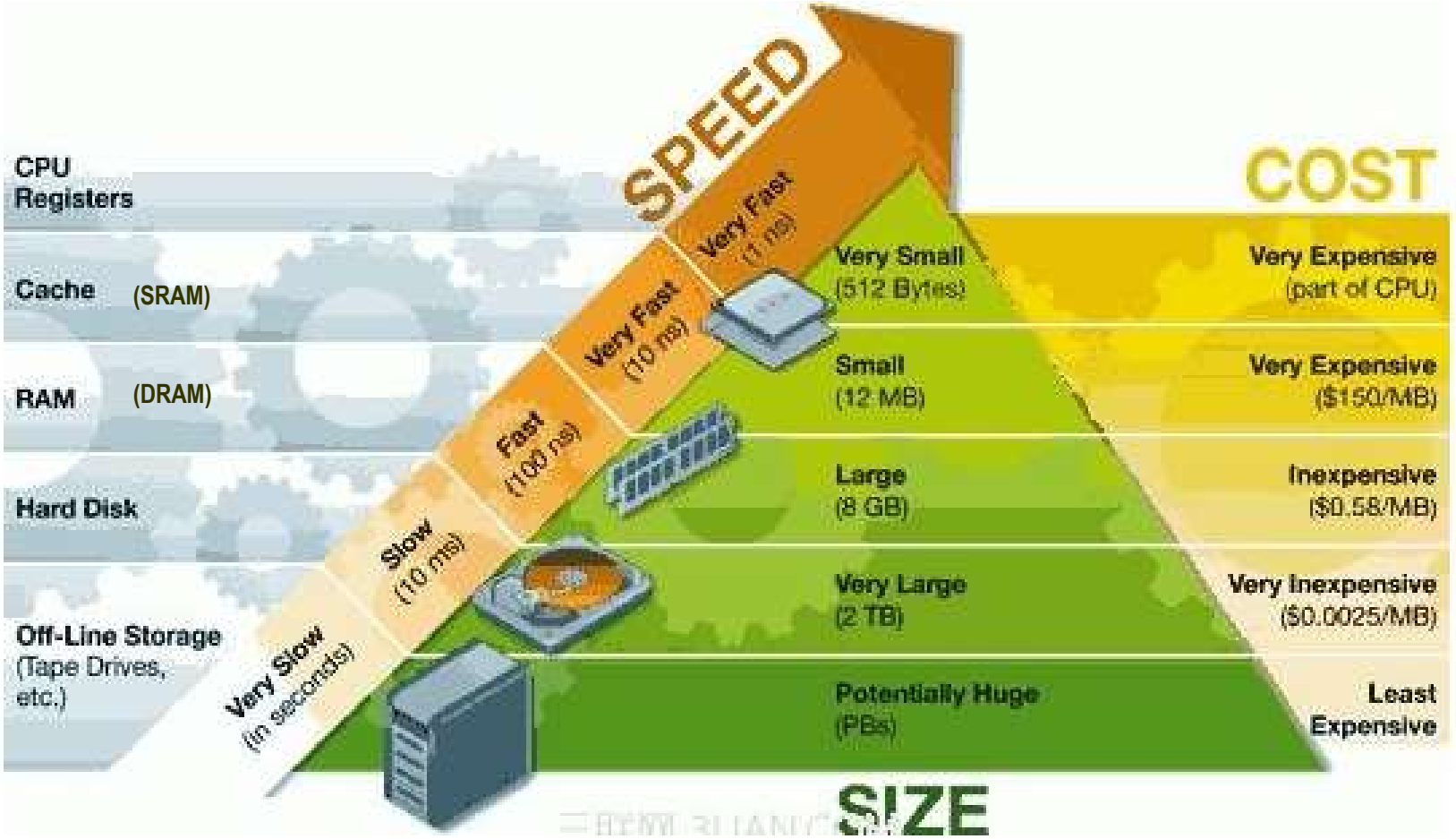
```
double A[MAX][MAX], x[MAX], y[MAX];
. . .
/* Initialize A and x, assign y = 0 */
. . .
/* First pair of loops */
for (i = 0; i < MAX; i++)
    for (j = 0; j < MAX; j++)
        y[i] += A[i][j]*x[j];
. . .
/* Assign y = 0 */
. . .
/* Second pair of loops */
for (j = 0; j < MAX; j++)
    for (i = 0; i < MAX; i++)
        y[i] += A[i][j]*x[j];
```

much faster

- suppose MAX is four, a cache line stores four doubles, and the elements of A are stored in memory as follows:

Cache Line	Elements of A			
0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
3	A[3][0]	A[3][1]	A[3][2]	A[3][3]

Memory Hierarchy

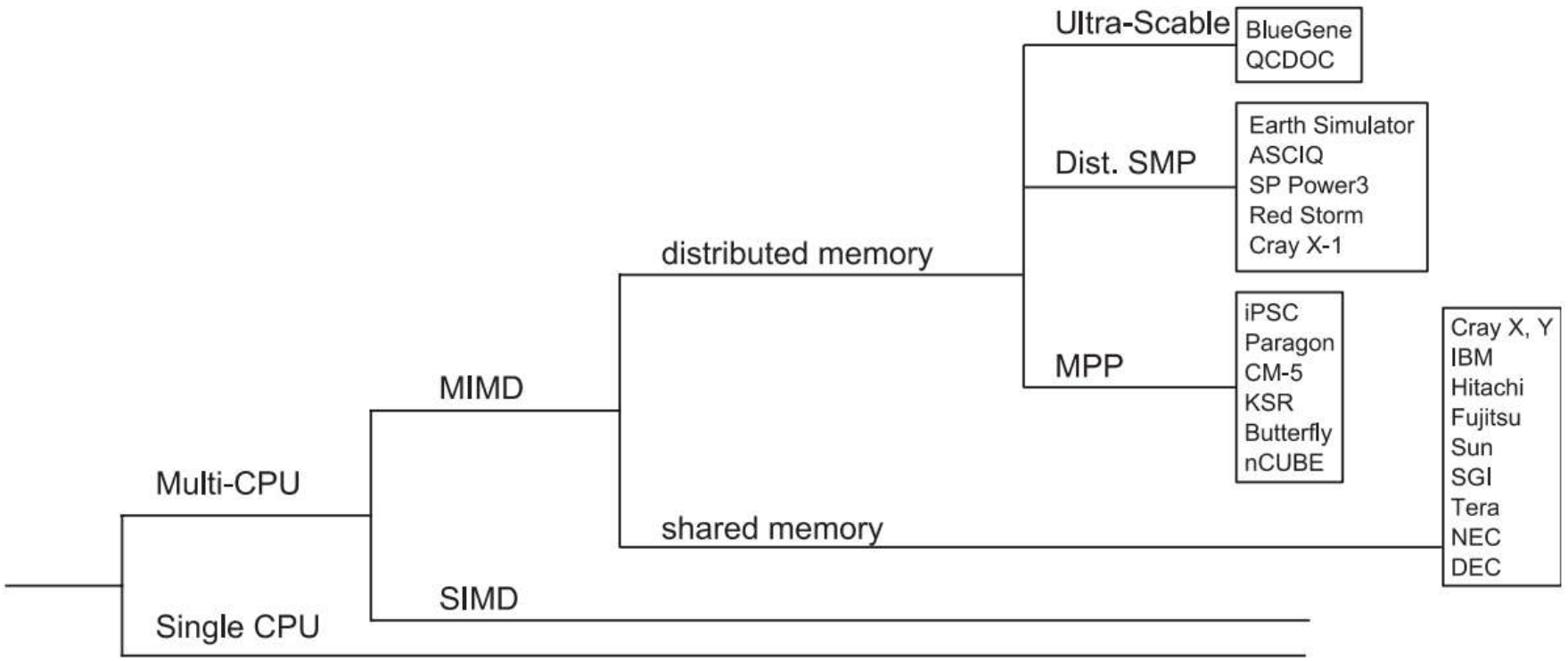


From von Neumann Architecture to Parallel Architectures

- How **von Neumann Architecture** can be modified and extended to support parallelism in many different flavors?

- How such a parallel machine can be efficiently used?

Evolution of computer architectures



History of Parallel Computers

- 1946(ENIAC-the first computer)
- 1971(Intel 4004-the first CPU)

- 1972-ILLIAC IV (First parallel computer-32 PEs-SIMD)
- 1983-CRAY-1G Flops；Yinhe-I（100M Flops）
- 1993(Top 500)-CM5-59.7 Gflops(1)--422M Flops(500)
- 1996-Intel-1T Flops
- 2006 (CMP-Dual-core-the First)
- 2008-1P Flops
- 2020-1E Flops
- 2025…?
- …?

CM-5

Parallel Architectures

- CPU Parallelism
- Memory Parallelism
- Interconnects
- Parallel I/O and Networking
- Parallel Architecture Design Tradeoffs

CPU parallelism

- Parallelism at the level of the CPU is more difficult to implement than simple replication of CPUs and memory, even when the memory presents a single shared address space.
- However, modest parallelism in the CPU provides the easiest route to improved performance for the majority of applications because little needs to be done by the programmer to exploit this kind of parallelism.

CPU parallelism (II)

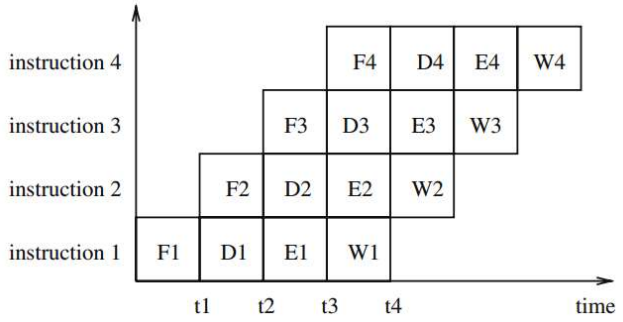
- Parallelism within a
  single processor or processor core

- SIMD, PVP (Parallel Vector Processor), SMP (symmetric multiprocessor), MPP (Massively Parallel Processor) , DSM(Distributed Shared Memory), COW (Cluster of Workstations)

- Multi-core/Many-core

Parallelism within a single processor or core

- **Parallelism at bit level**
  the word size used by processors for operations
  (4 bits->32 bits->64bits)



- **Parallelism by pipelining**

Overlapping execution of four independent instructions by pipelining. The execution of each instruction is split into four stages: *fetch* (F), *decode* (D), *execute* (E), and *write-back* (W)

- **Parallelism by multiple functional units**

They use multiple, independent functional units like ALUs (arithmetic logical units), FPUs (floating-point units), load/store units, or branch units

-Superscalar Processing/instruction-level parallelism (ILP)

-VLIW (very long instruction word)

- **Multithreading**

-Simultaneous multithreading (SMT); Fine-grained multithreading uses a single thread at a time but allows the CPU to change threads in a single clock cycle

- **SIMD and Vectors**

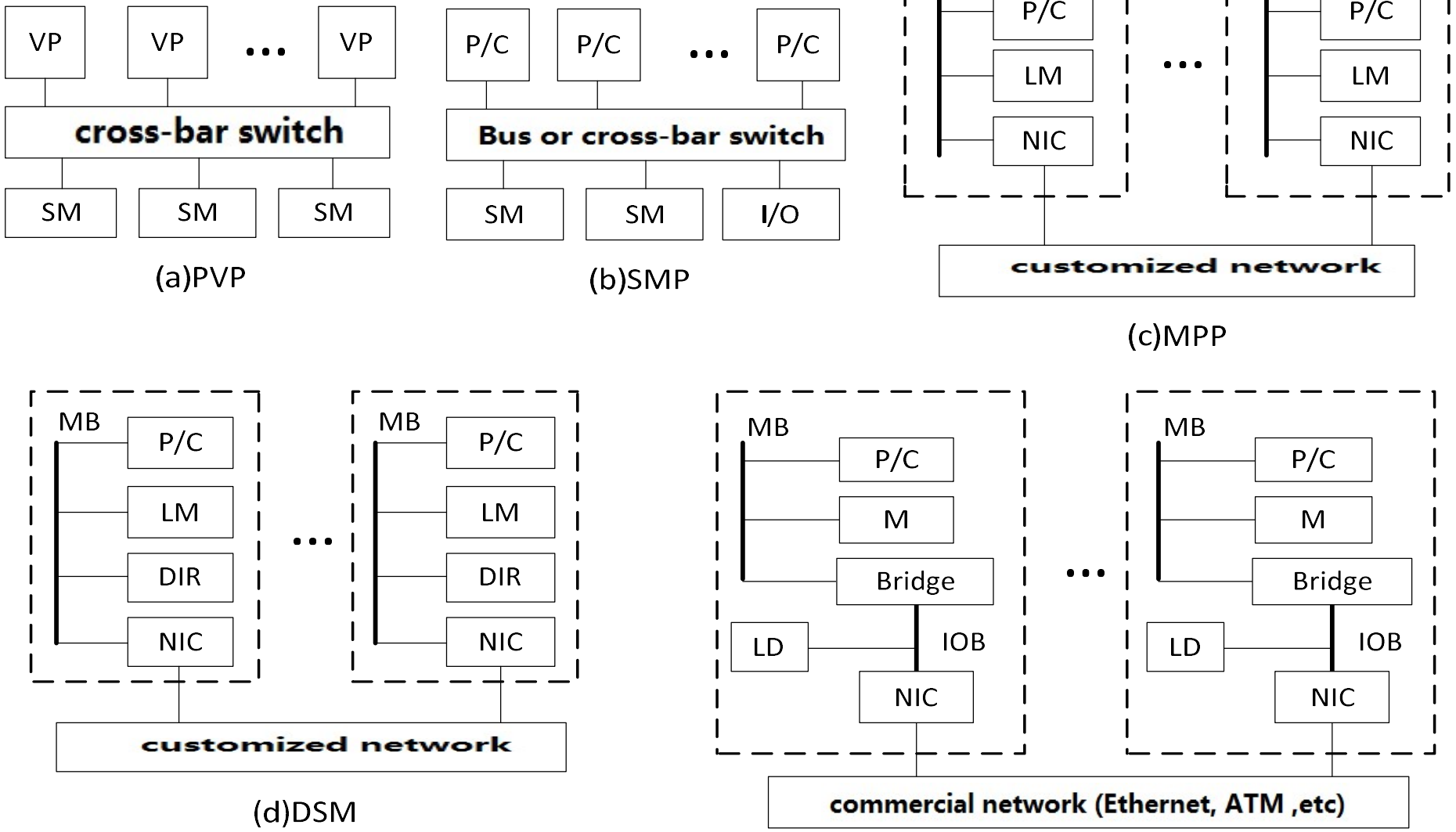
  -apply the same operation to several different data values, using multiple functional units;

  -Vector computer

- **Parallelism at process or thread level**

  -multicore processors

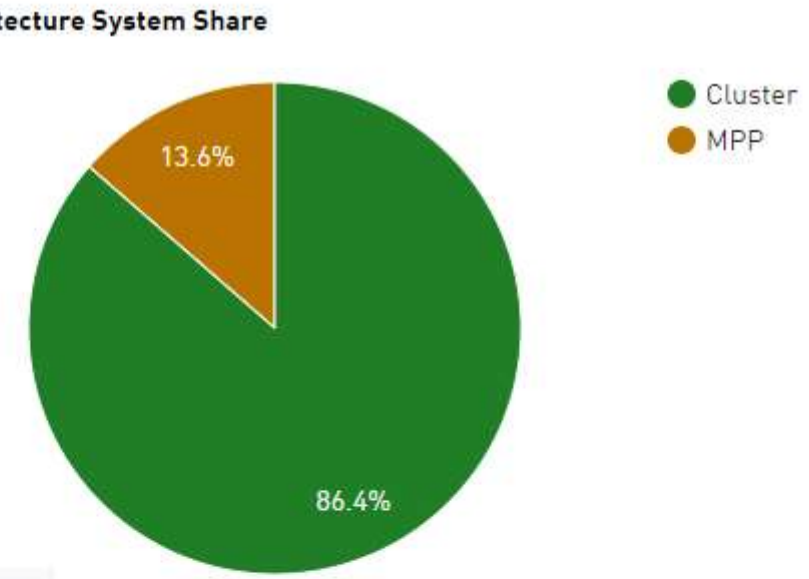
Parallel Computer Architecture



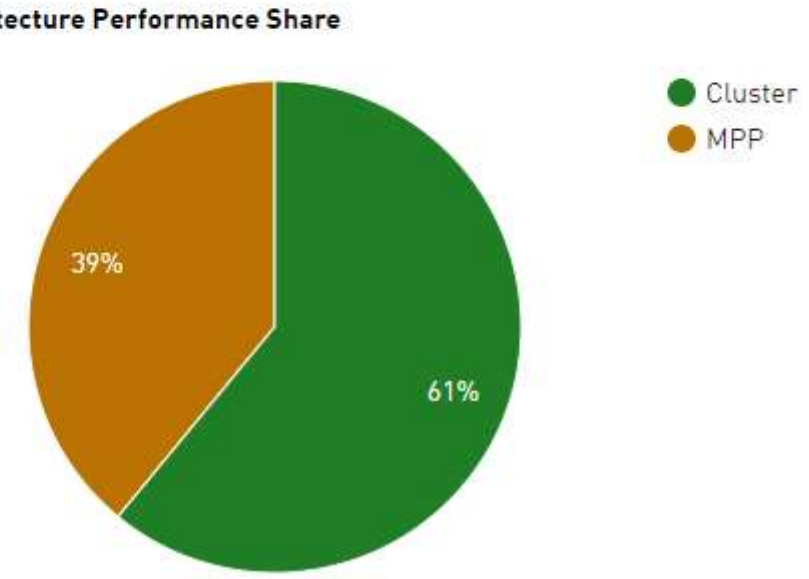
Comparison of the architectures

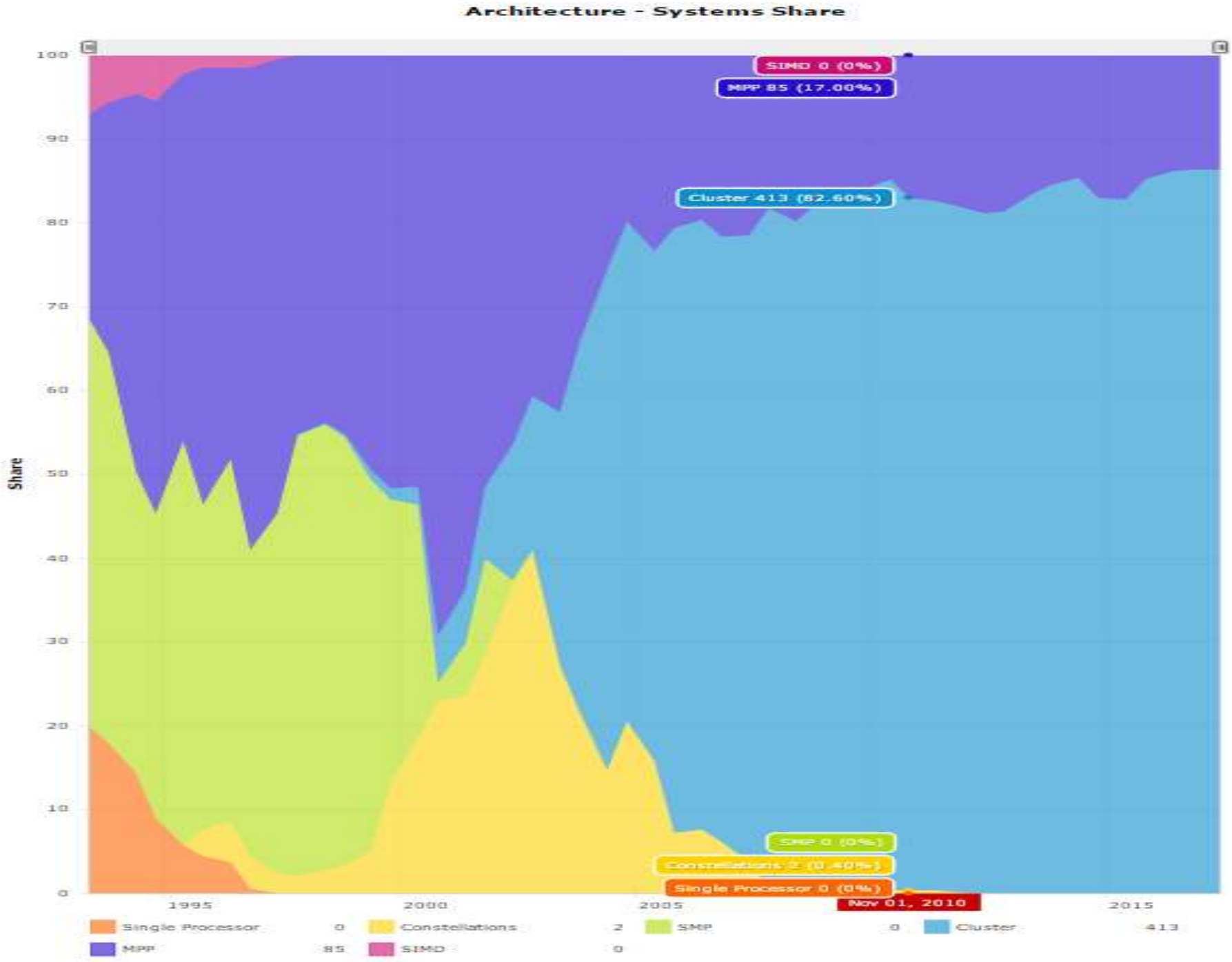
Property	PVP	SMP	MPP	DSM	COW
Arch. Type	MIMD	MIMD	MIMD	MIMD	MIMD
Processor Type	customized	commercial	commercial	commercial	commercial
Interconnect Network	customized cross-bar switch	Bus, cross-bar switch	customized network	customized network	commercial (Ethernet ATM）
Comm. Mechanism	Shared variable	Shared variable	Message passing	Shared variable	Message passing
Address Space	single	single	multiple	single	multiple
System Storage	Centralized sharing	Centralized sharing	Distributed Non-sharing	Distributed sharing	Distributed Non-sharing
Memory Access	UMA	UMA	NORMA	NUMA	NORMA
Representative Machine	Cray C-90，Cray T-90，Yinhe 1	IBM R50，SGI Power Challenge，Sugon 1	Intel Paragon，IBMSP2，Sugon 1000/2000	Stanford DASH，Cray T 3D	Berkeley NOW，Alpha Farm

Top500-2017.6



Architecture	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
Cluster	432	86.4	456,325,585	734,745,036	27,337,874
MPP	68	13.6	292,046,303	397,496,983	20,741,916




Architecture - Systems Share
Share
100
90
80
70
60
50
40
30
20
10
0
SIMD 0 (0%)
MPP 85 (17.00%)
Cluster 413 (82.60%)
SMP 0 (0%)
Constellations 2 (0.40%)
Single Processor 0 (0%)
Nov 01, 2010
1995
2000
2005
2015
Single Processor 0
Constellations 2
SMP 0
Cluster 413
MPP 85
SIMD 0

SMP\MPP\Cluster

System - characteristic	SMP	MPP	Cluster
# of nodes (N)	≤O(10)	O(100)-O(1000)	≤O(100)
Complexity of nodes	middle or fine	Fine or middle	Middle or coarse
Inter-nodes communication	Memory Sharing	Message passing or Memory sharing（DSM）	Message Passing
Node operating system	1	N(micro-kernel) and 1 OS(single)	N (homogeneous)
Single system image	ever	part	hope
Address space	Single	Multiple or single（DSM）	Multiple
Job scheduling	Single queue	Single queue on host	collaborative multiple queue
Network protocol	Non-standard	Non-standard	Standard/ Non-standard
Availability	low	Low-middle	High or fault tolerant
Perf./Cost	normal	normal	high
interconnect network	Bus/Cross-bar switch	Customized	Commercial

Memory parallelism

- **Shared Memory**

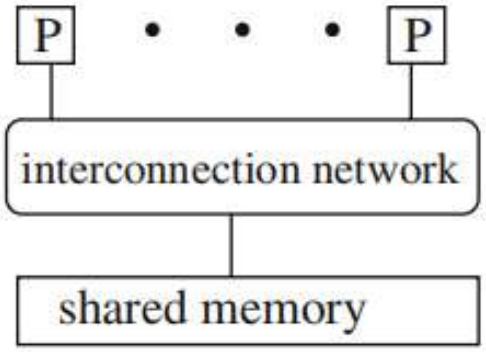
  UMA(Uniform Memory Access), NUMA(Nonuniform Memory Access), CCNUMA (Cache Coherence NUMA), COMA (Cache-Only Memory Access), NORMA (No-Remote Memory Access)

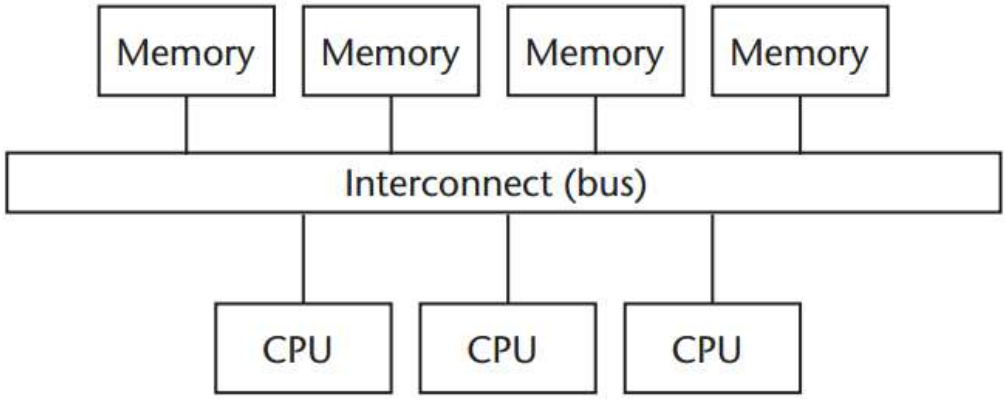
- **Distributed Memory**

- **Hybrid Distributed-Shared Memory**

Shared Memory

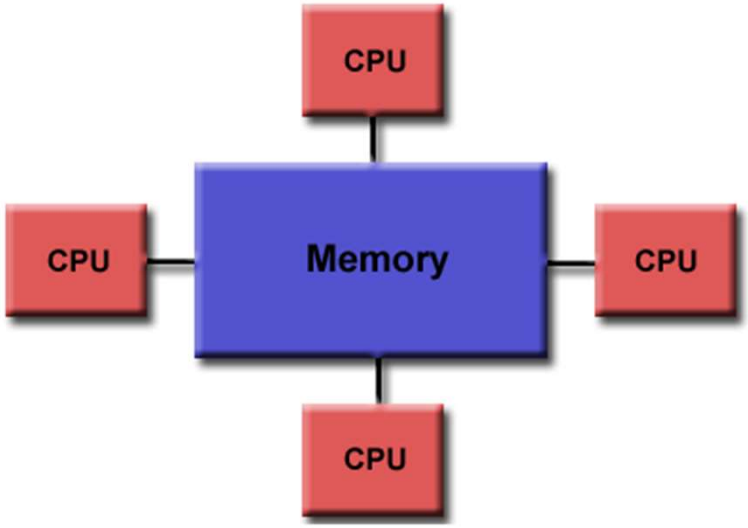
- **General Characteristics:** Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
  - Multiple processors can operate independently but share the same memory resources.
  - Changes in a memory location effected by one processor are visible to all other processors.
- Historically, shared memory machines have been classified as ***UMA*** and ***NUMA***, based upon memory access times.

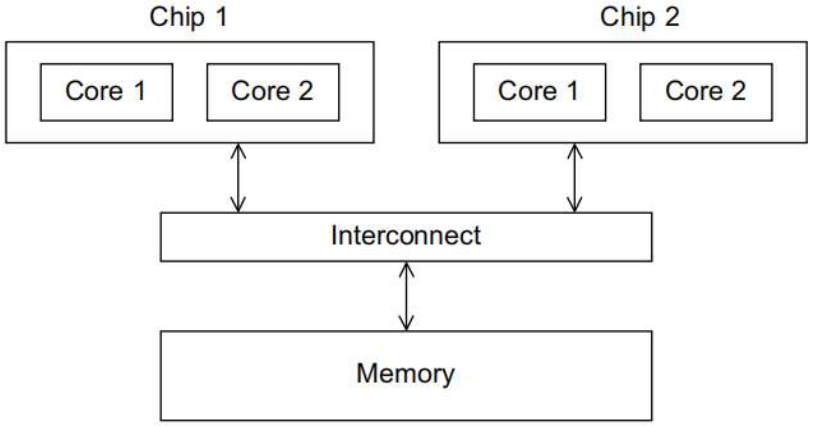


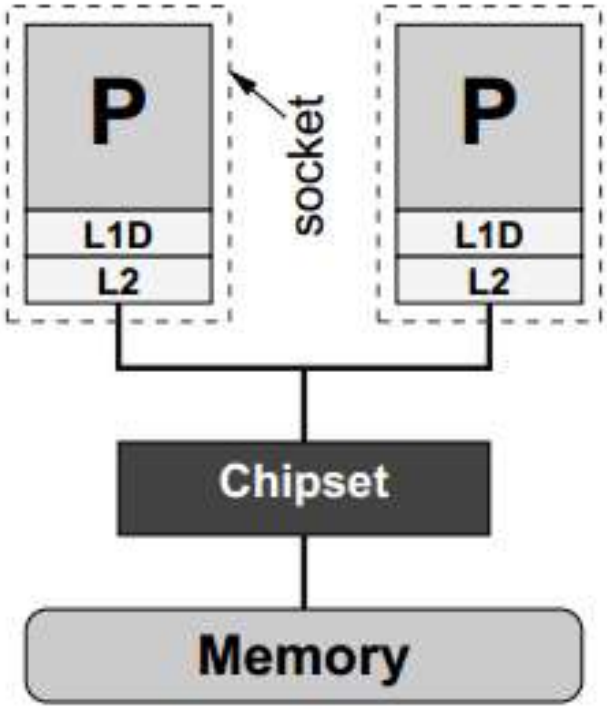


Uniform Memory Access (UMA):

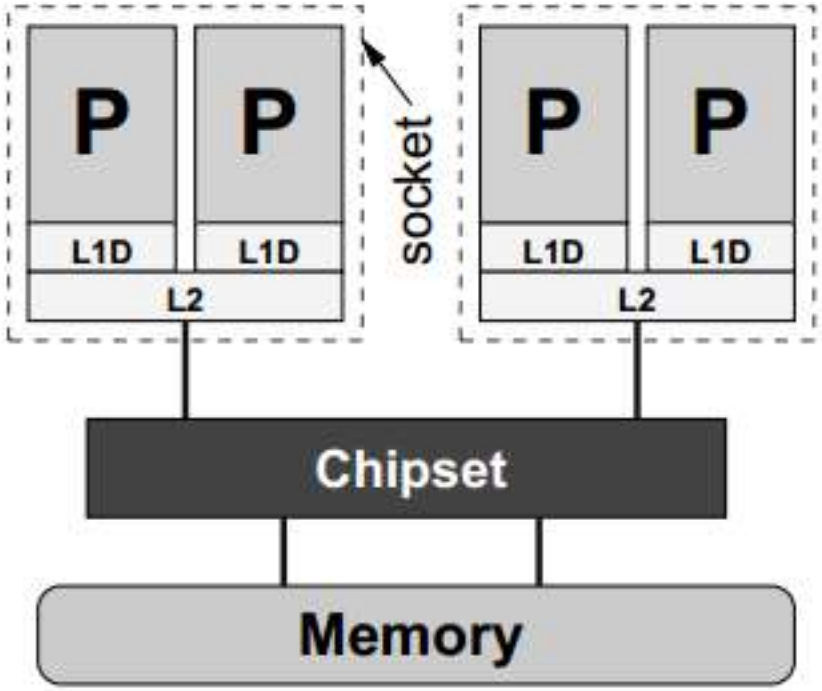
- Most commonly represented today by ***Symmetric Multiprocessor (SMP)*** machines
  - Identical processors
  - Equal access and access times to memory
- Sometimes called CC-UMA - **Cache Coherent UMA**. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level.







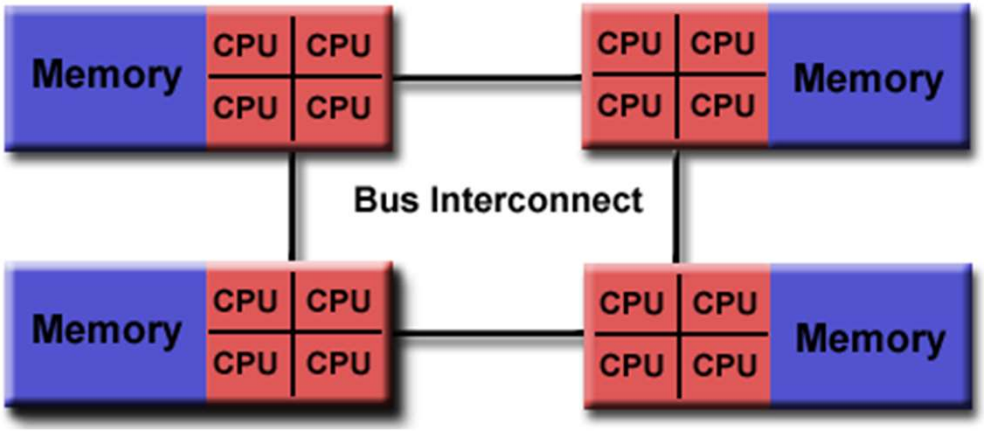
- A UMA system with two singlecore CPUs that share a common frontside bus (FSB)

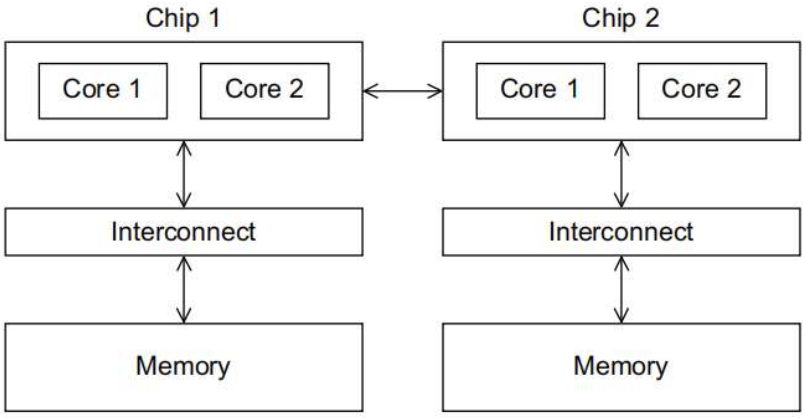


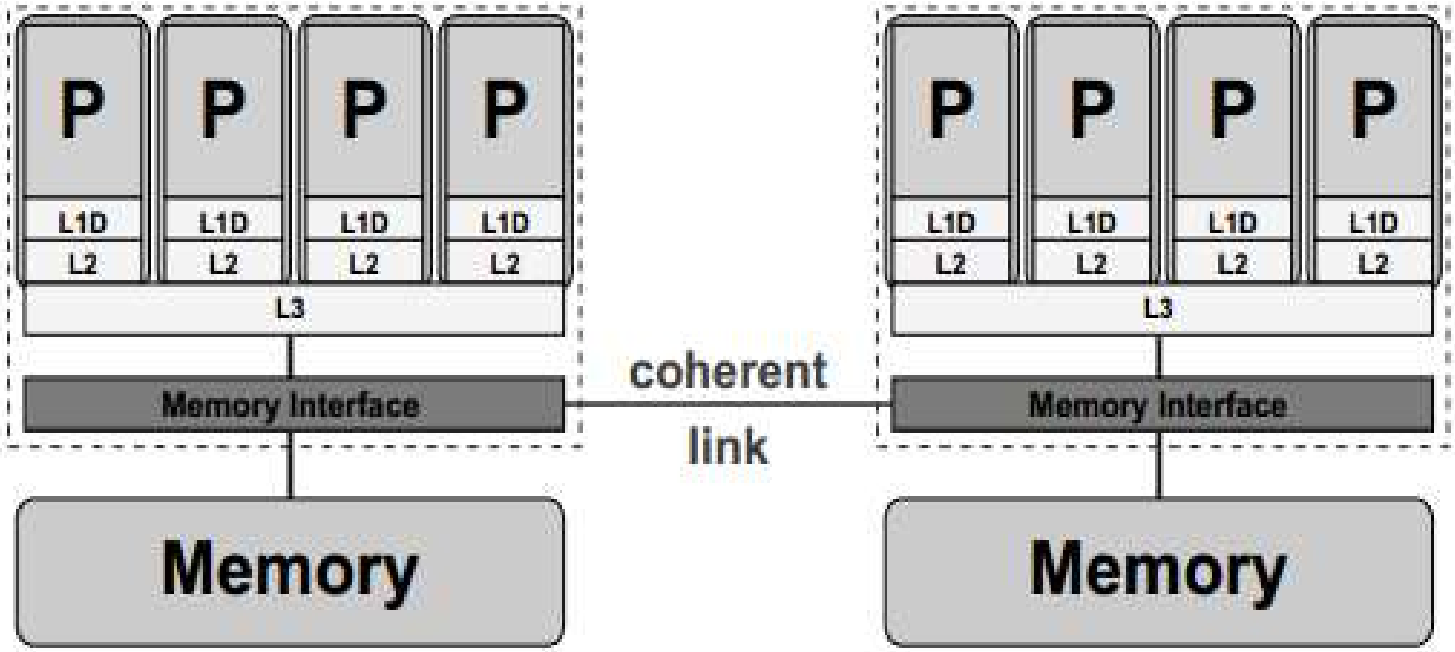
- A UMA system in which the FSBs of two dual-core chips are connected separately to the chipset

Non-Uniform Memory Access (NUMA):

- Often made by physically linking two or more SMPs
  - One SMP can directly access memory of another SMP
  - ***Not*** all processors have equal access time to all memories
  - Memory access across link is slower
- If cache coherency is maintained, then may also be called **CC-NUMA** - Cache Coherent NUMA

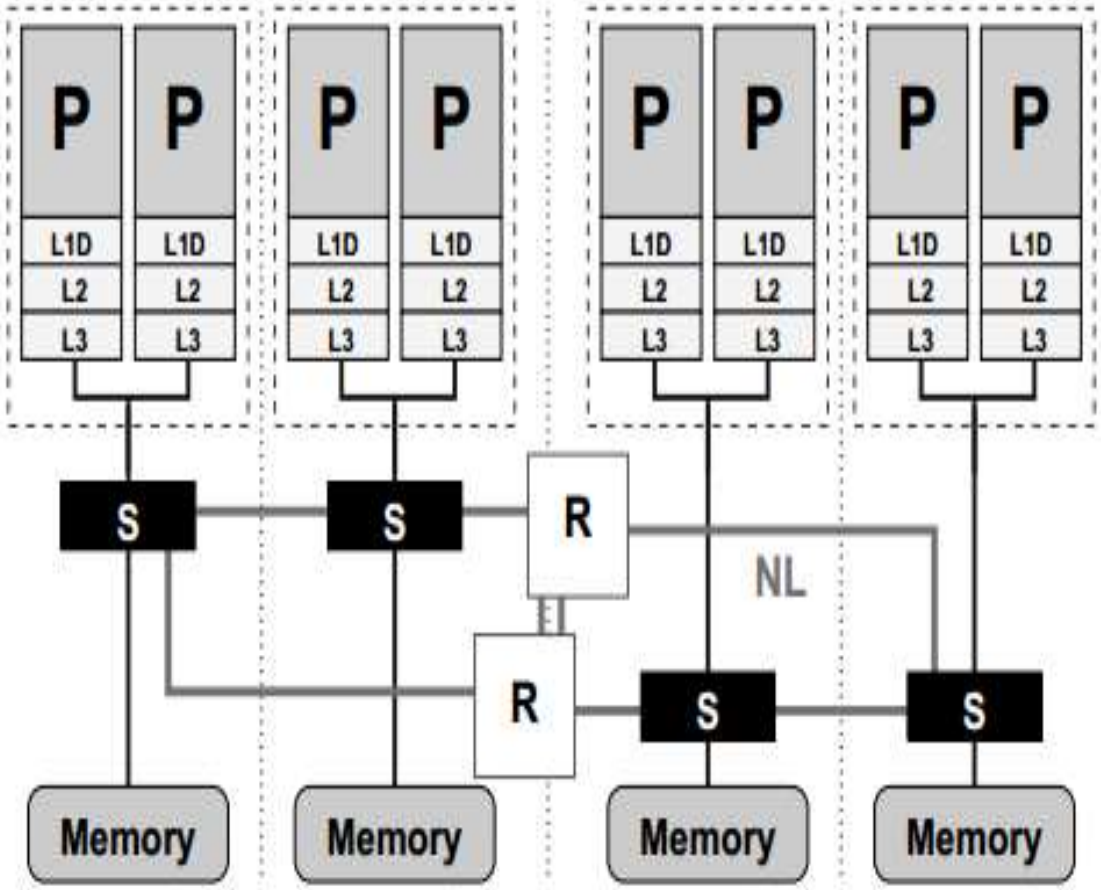


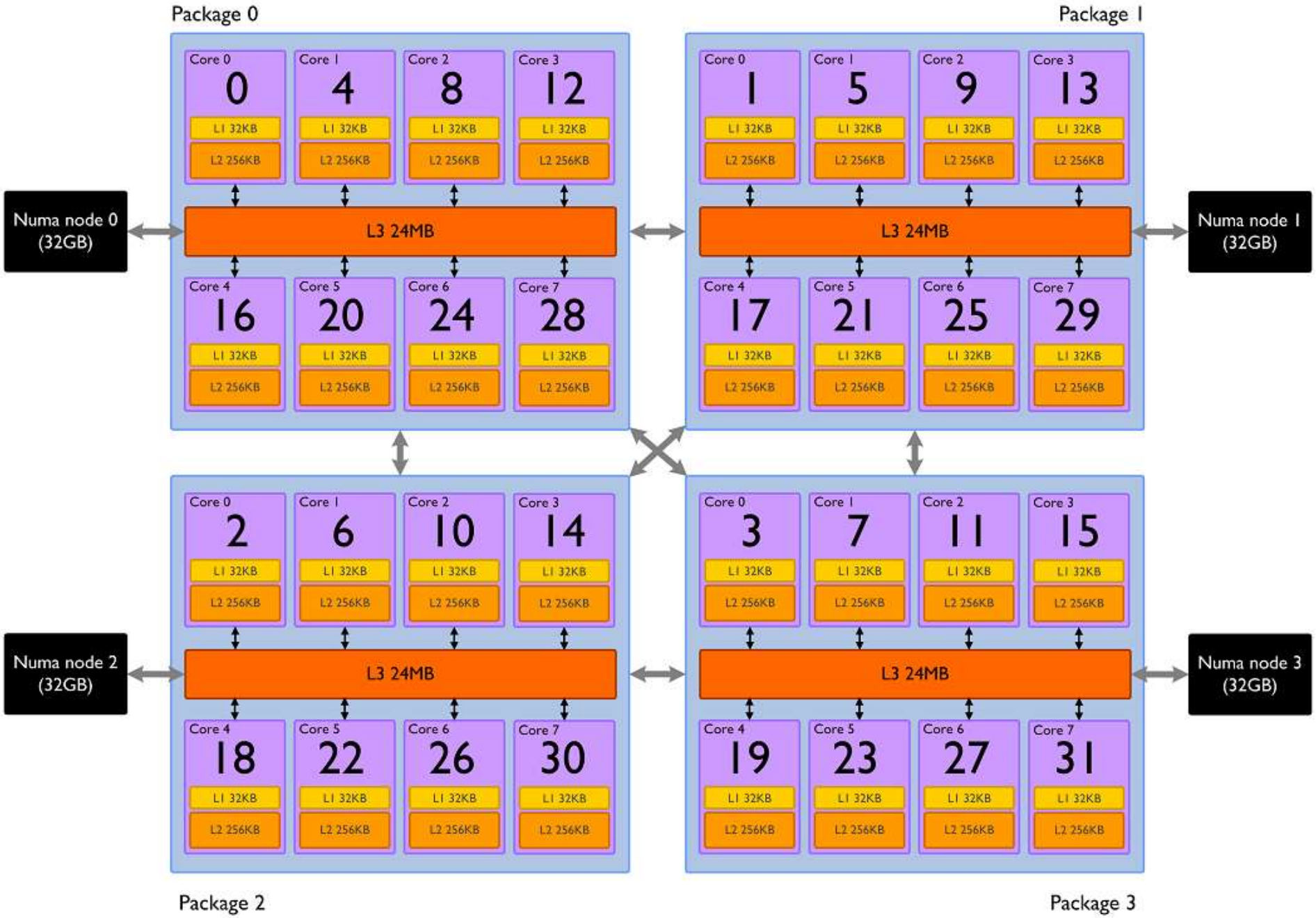




- A ccNUMA system with two locality domains (one per socket) and eight cores

- A ccNUMA system (**SGI Altix**) with four locality domains, each comprising one socket with two cores. The LDs *(locality domain)* are connected via a routed NUMALink (NL) network using routers (R)





64-threaded Intel Nehalem-EX

Cache coherence

- Cache coherence mechanisms are required in all cache-based multiprocessor systems, whether they are of the UMA or the ccNUMA kind.
- This is because copies of the same cache line could potentially reside in several CPU caches. If, e.g., one of those gets modified and evicted to memory, the other caches' contents reflect outdated data.
- Cache coherence **protocols** ensure a consistent view of memory under all circumstances.

MESI coherence protocol

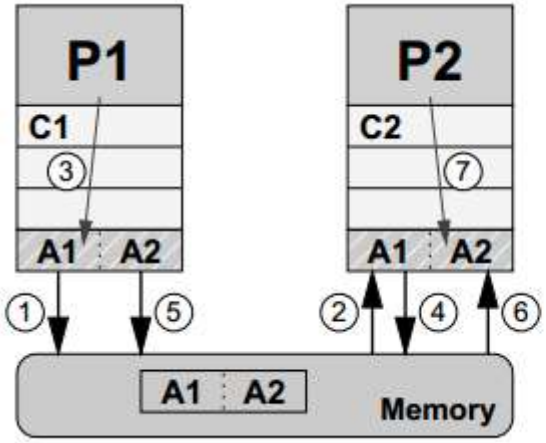
M *modified:* The cache line has been modified in this cache, and it resides in no other cache than this one. Only upon eviction will memory reflect the most current state.

E *exclusive:* The cache line has been read from memory but not (yet) modified. However, it resides in no other cache.

S *shared:* The cache line has been read from memory but not (yet) modified. There may be other copies in other caches of the machine.

I *invalid:* The cache line does not reflect any sensible data. Under normal circumstances this happens if the cache line was in the shared state and another processor has requested exclusive ownership.

- an example on two processors P1 and P2 with respective caches C1 and C2. Each cache line holds two items. Two neighboring items A1 and A2 in memory belong to the same cache line and are modified by P1 and P2, respectively



1. C1 requests exclusive CL ownership
2. set CL in C2 to state I
3. CL has state E in C1 → modify A1 in C1 and set to state M
4. C2 requests exclusive CL ownership
5. evict CL from C1 and set to state I
6. load CL to C2 and set to state E
7. modify A2 in C2 and set to state M in C2

- Two processors P1, P2 modify the two parts A1, A2 of the same cache line in caches C1 and C2. The **MESI coherence protocol** ensures consistency between cache and memory.

- Without cache coherence, each cache would read the line from memory, A1 would get modified in C1, A2 would get modified in C2 and some time later both modified copies of the cache line would have to be evicted. As all memory traffic is handled in chunks of cache line size, there is no way to determine the correct values of A1 and A2 in memory.

- Under control of cache coherence logic this discrepancy can be avoided. As an example we pick the **MESI protocol**, which draws its name from the four possible states a cache line can assume

- **COMA**

  -Cache-Only memory access

- **NORMA**

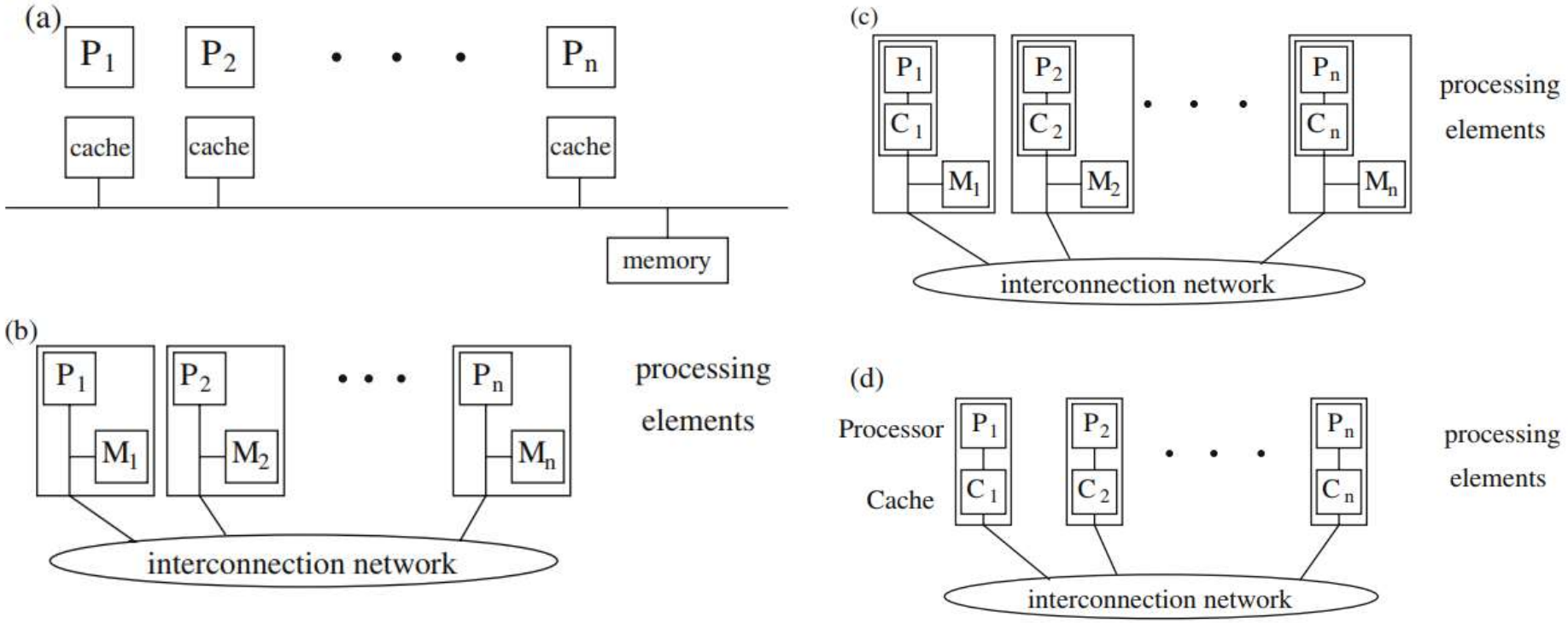
  -No Remote Memory Access (NORMA)

COMA (Cache-Only memory access)

- COMA model is a special case of NUMA in which the distributed memories are converted to caches.
- All caches form a global address space.
- Remote cache access is assisted by the distributed cache directories.

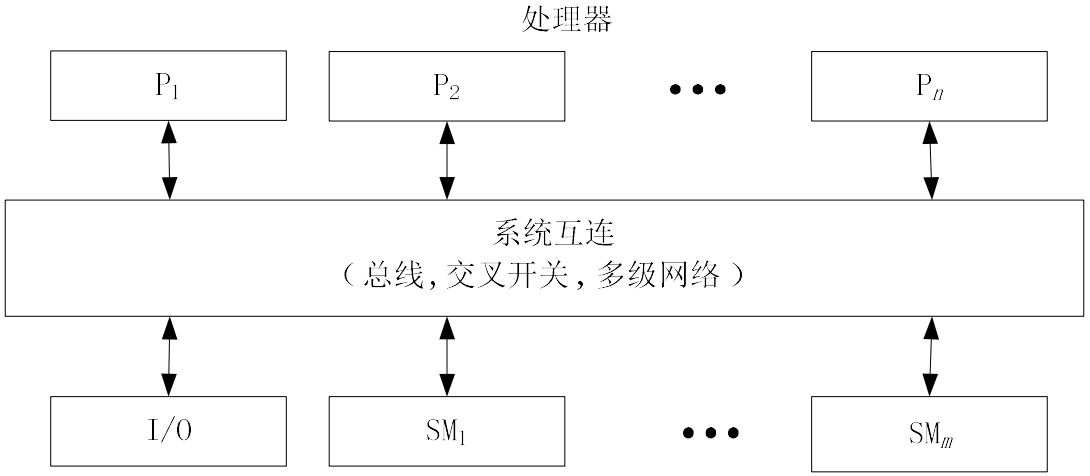
NORMA *(No Remote Memory Access)*

- A distributed memory multicomputer system is called NORMA model if all memories are private and accessible only by local processors.
- Most of NUMA model supports no-remote memory access.
- In DSM system, NORMA will disappear.

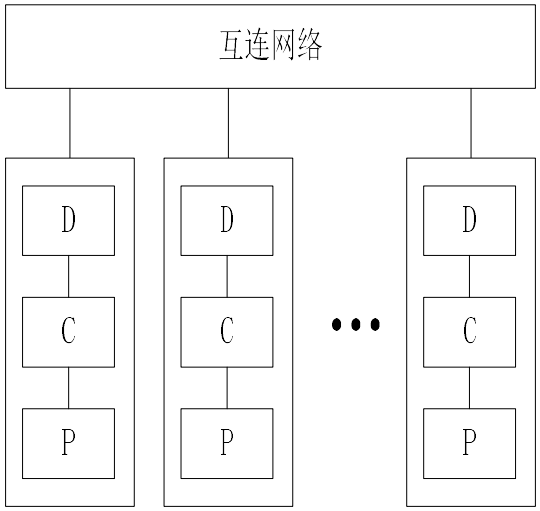


- Illustration of the architecture of computers with shared memory: (**a**) SMP – symmetric multiprocessors, (**b**) NUMA – non-uniform memory access, (**c**) CC-NUMA – cache-coherent NUMA, and (**d**) COMA – cache-only memory access

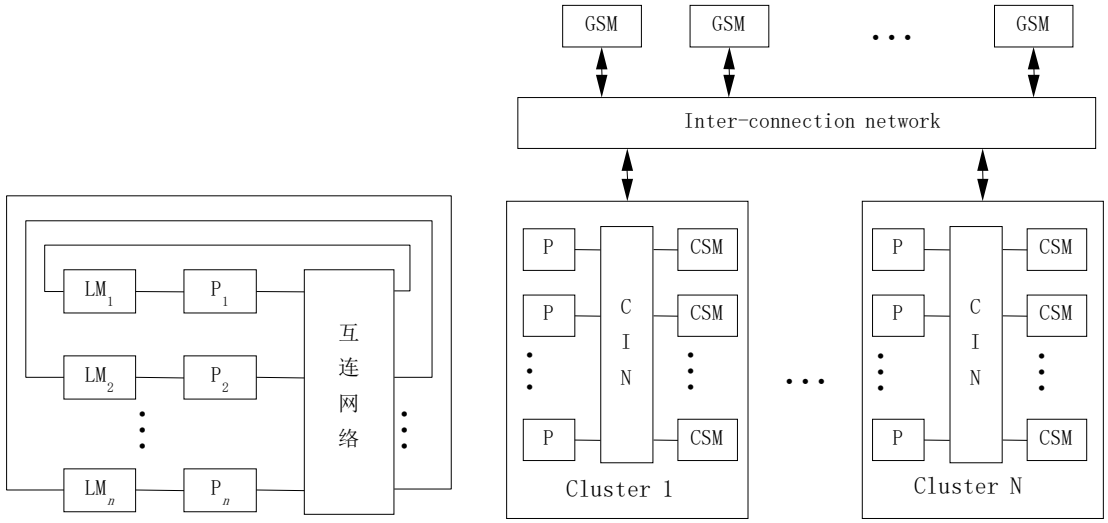
UMA NUMA COMA NORMA



UMA



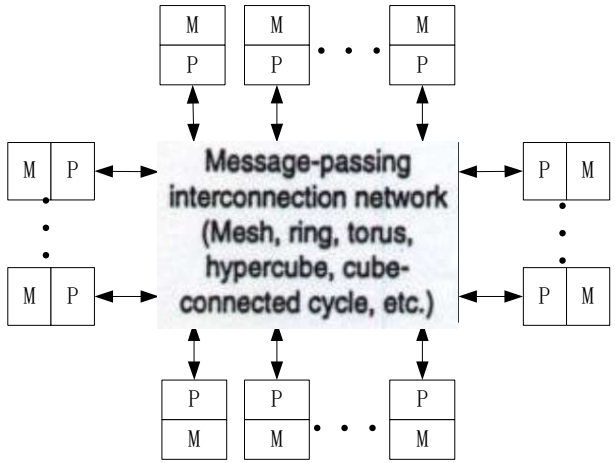
COMA



(a)Local memory sharing model

NUMA

(b)Layered cluster model



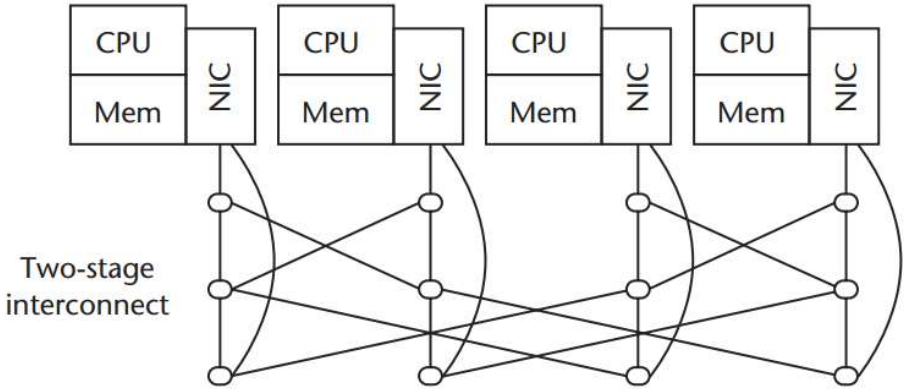
NORMA

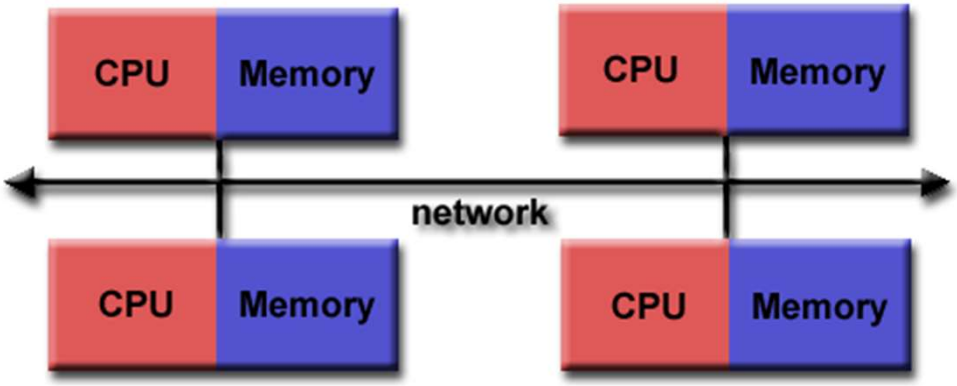
Shared Memory- Pros. & Cons.

- **Advantages:**
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
- **Disadvantages:**
  - Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
  - Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

Distributed Memory

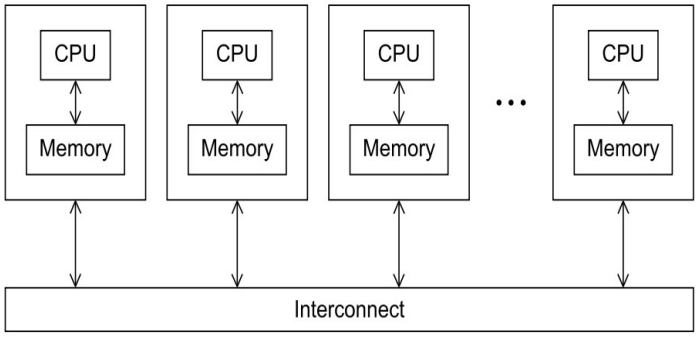
- Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a **communication network** to connect inter-processor memory.
- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.

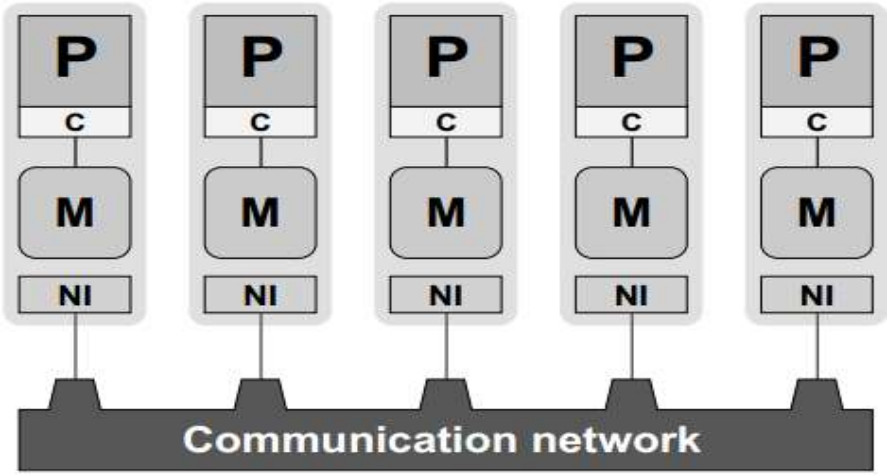


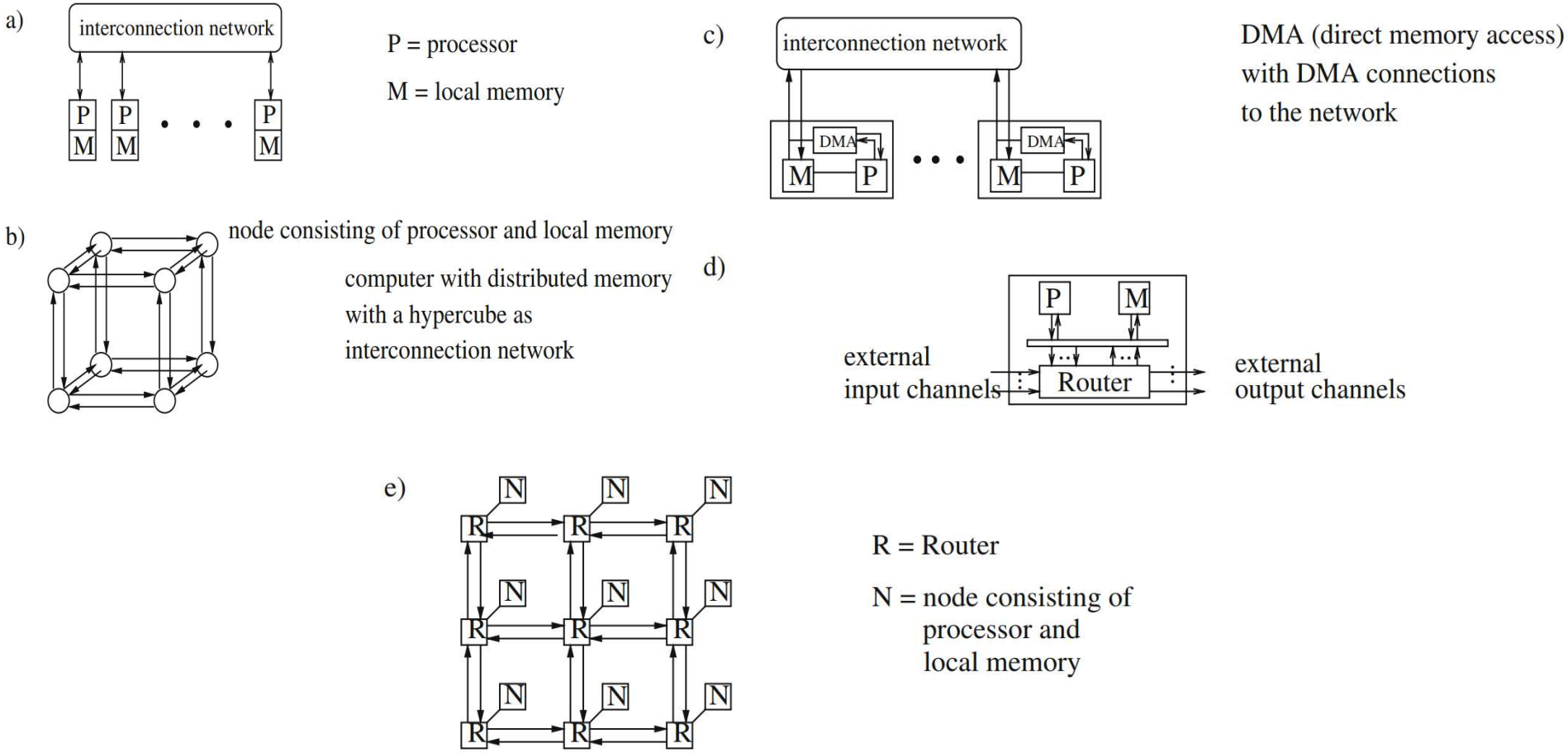


- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
- The network "fabric" used for data transfer varies widely, though it can can be as simple as Ethernet.

- **MPP, Cluster**







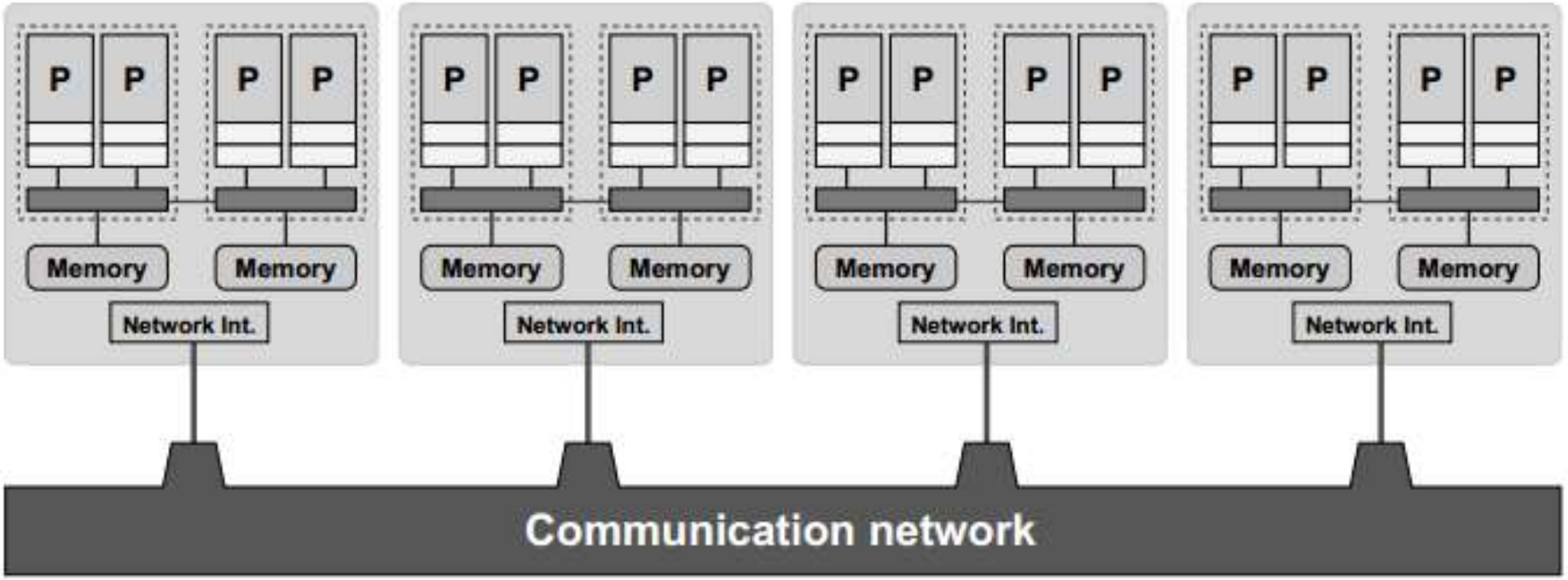
- Illustration of computers with distributed memory: (**a**) abstract structure, (**b**) computer with distributed memory and hypercube as interconnection structure, (**c**) DMA (direct memory access), (**d**) processor–memory node with router, and (**e**) interconnection network in the form of a mesh to connect the routers of the different processor–memory nodes

Distributed Memory- Pros. & Cons.

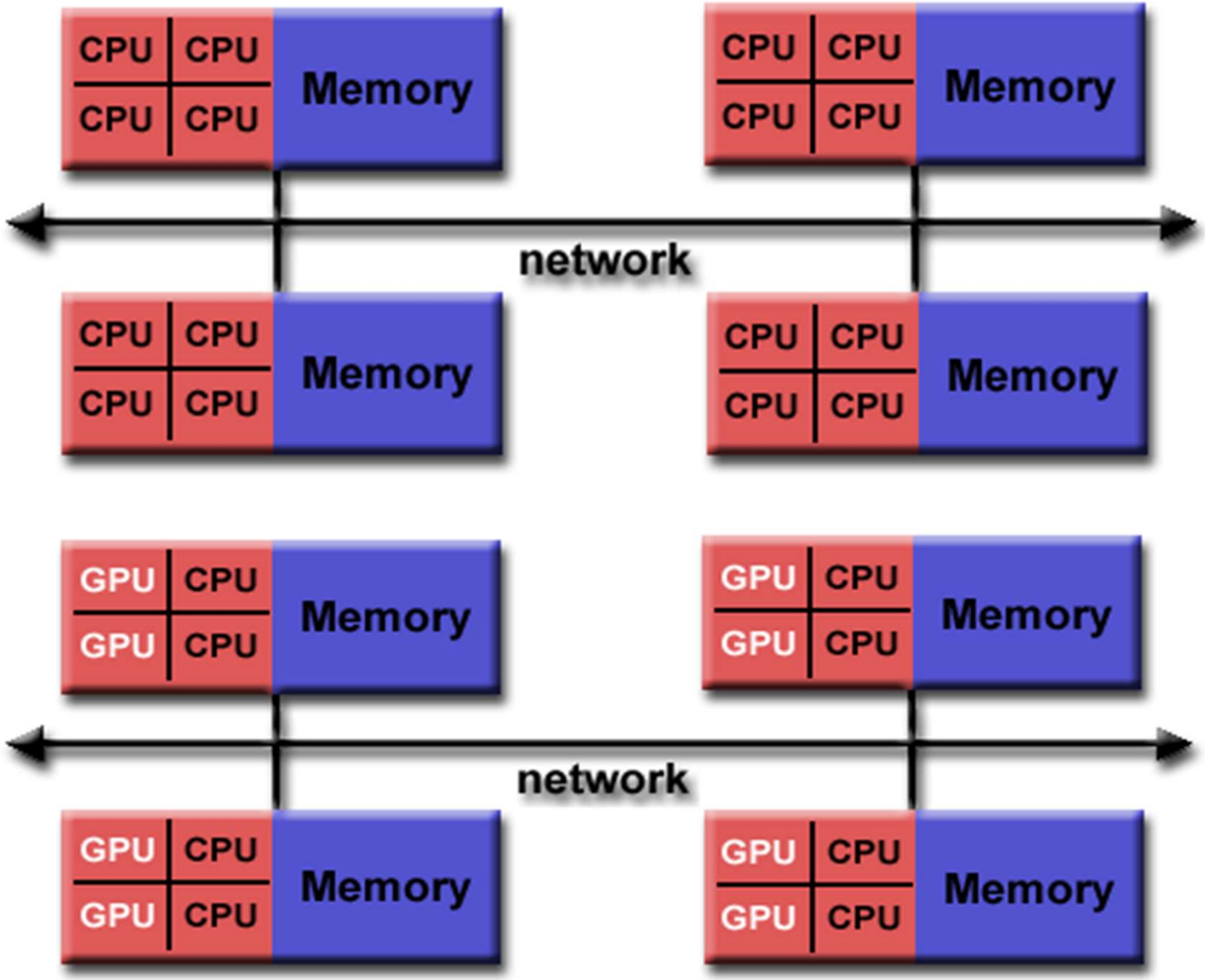
- **Advantages:**
  - Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately.
  - Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- **Disadvantages:**
  - The programmer is responsible for many of the details associated with data communication between processors.
  - It may be difficult to map existing data structures, based on global memory, to this memory organization.
  - Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.

Hybrid Distributed-Shared Memory

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.
  - The shared memory component can be a shared memory machine and/or graphics processing units (GPU).
  - The distributed memory component is the networking of multiple shared memory/GPU machines, which know only about their own memory - not the memory on another machine. Therefore, network communications are required to move data from one machine to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.



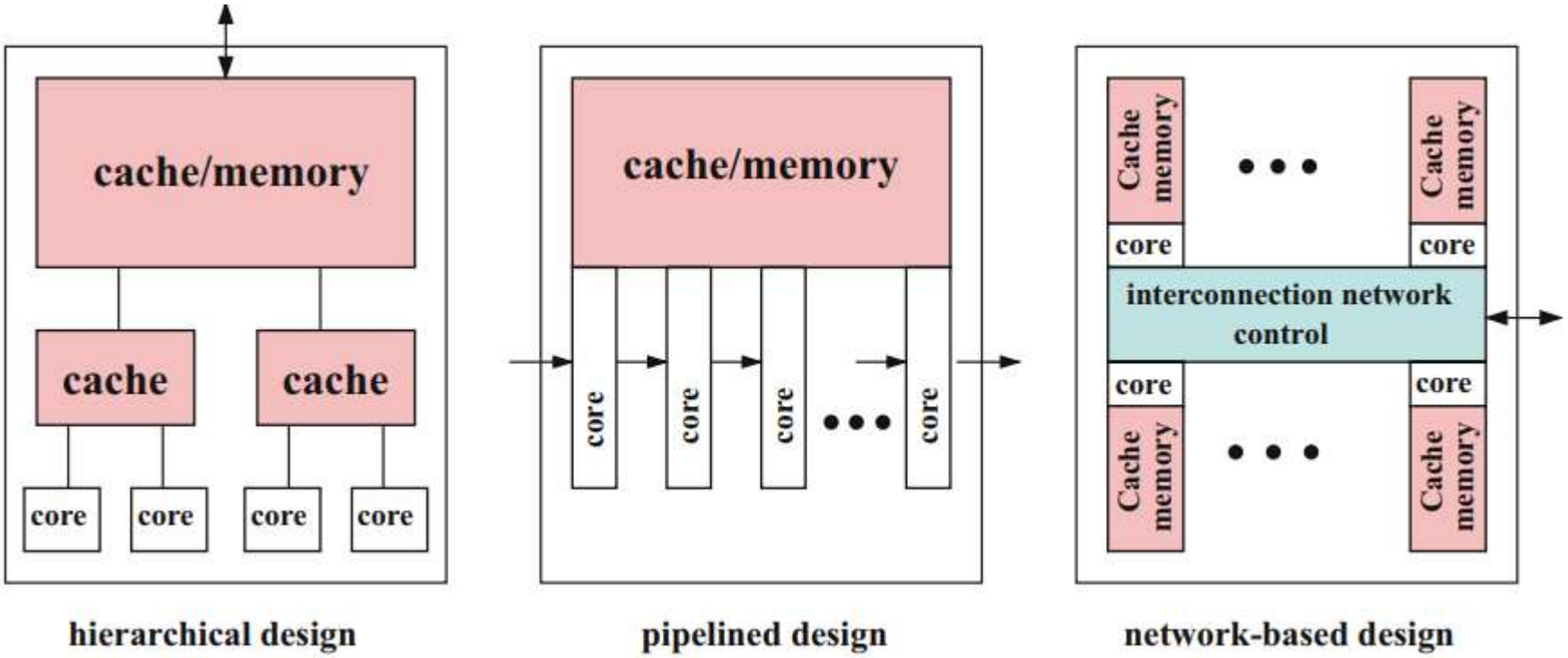
- Typical hybrid system with shared-memory nodes (ccNUMA type). Two-socket building blocks represent the price vs. performance “sweet spot” and are thus found in many commodity clusters


CPU
CPU
CPU
CPU
Memory
network
GPU
CPU
GPU
CPU
Memory

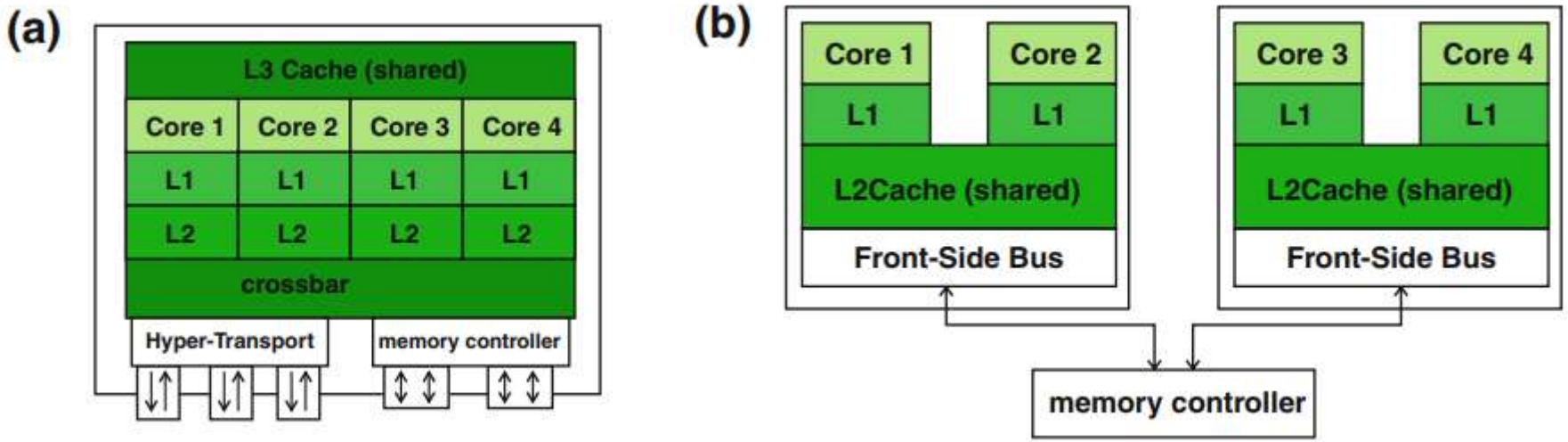
Hybrid Distributed- Pros. & Cons.

- **Advantages and Disadvantages:**
  - Whatever is common to both shared and distributed memory architectures.
  - Increased scalability is an important advantage
  - Increased programmer complexity is an important disadvantage

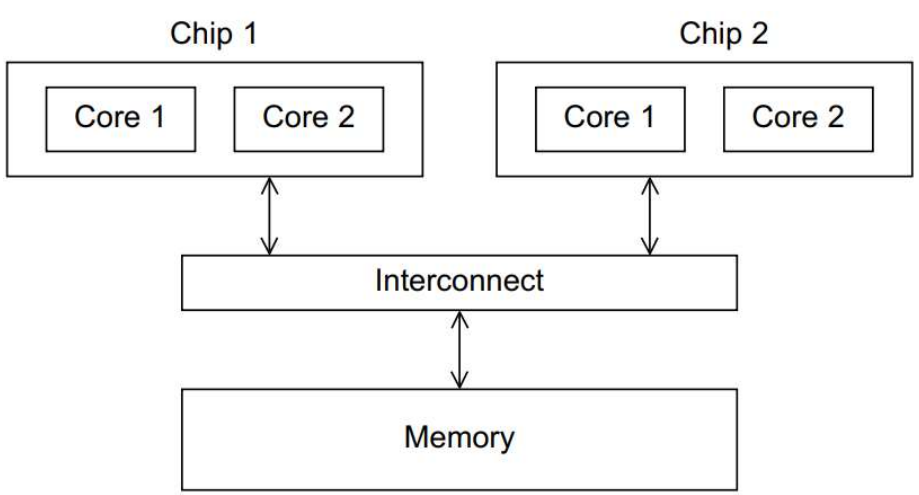
Multi-core



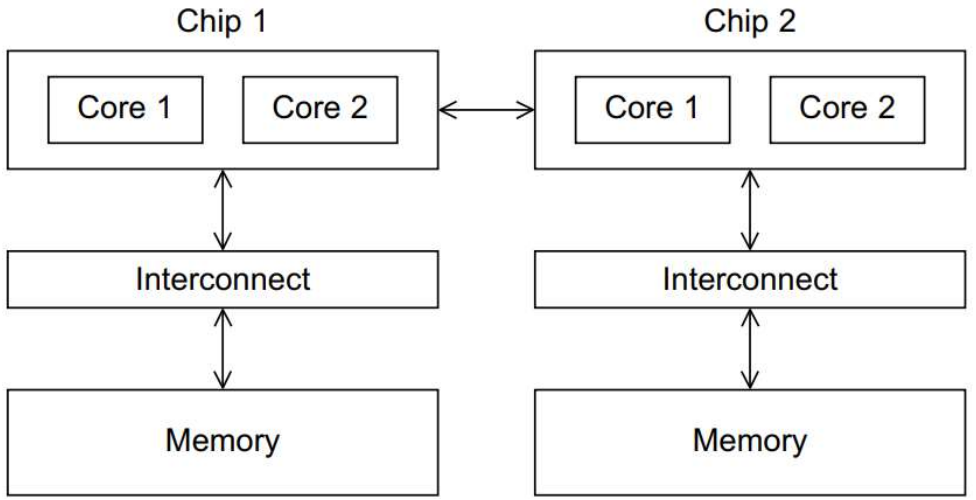
- Design choices for multicore chips
  (P.M. Kogge. 2005.)



- Quad-Core AMD Opteron (*left*) vs. Intel Quad-Core Xeon architecture (*right*) as examples for a hierarchical design



- A UMA multicore system



- A NUMA multicore system

lscpu

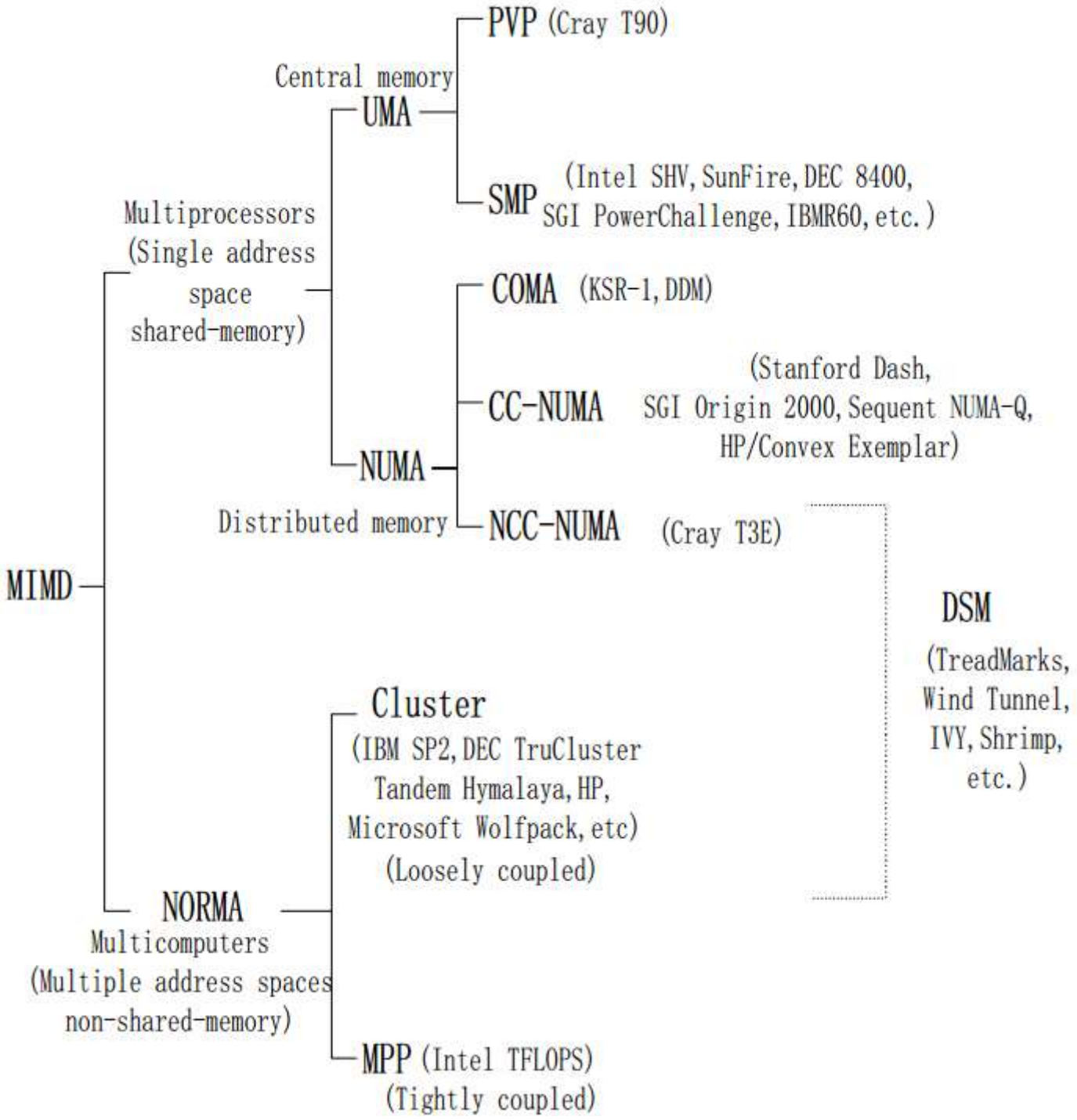
```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                56
On-line CPU(s) list:   0-55
Thread(s) per core:    1
Core(s) per socket:    14
Socket(s):             4
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
Stepping:              4
CPU MHz:               1000.000
CPU max MHz:           2601.0000
CPU min MHz:           1000.0000
BogoMIPS:              5200.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              19712K
NUMA node0 CPU(s):     0-13
NUMA node1 CPU(s):     14-27
NUMA node2 CPU(s):     28-41
NUMA node3 CPU(s):     42-55
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdp
e1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 ds_cpl vmx smx est tm2 sss
e3 fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb cat_l3 cdp_l3 int
el_pt tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm mpx rdt_a avx512f avx512dq rdseed adx smap clflu
shopt clwb avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida arat pln pts hwp hwp_act_window hw
p_epp hwp_pkg_req
```

likwid-topology

```
--------------------------------------------------------------------------------
CPU name:       Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
CPU type:       Unknown Intel Processor
CPU stepping:   4
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                4
Cores per socket:       14
Threads per core:       1
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               0               1               0               *
2               0               2               0               *
3               0               3               0               *
4               0               4               0               *
5               0               5               0               *
6               0               6               0               *
7               0               7               0               *
8               0               8               0               *
9               0               9               0               *
10              0               10              0               *
11              0               11              0               *
12              0               12              0               *
13              0               13              0               *
14              0               0               1               *
15              0               1               1               *
16              0               2               1               *
17              0               3               1               *
18              0               4               1               *
19              0               5               1               *
20              0               6               1               *
21              0               7               1               *
22              0               8               1               *
23              0               9               1               *
24              0               10              1               *
25              0               11              1               *
26              0               12              1               *
27              0               13              1               *
28              0               0               2               *
29              0               1               2               *
30              0               2               2               *
31              0               3               2               *
32              0               4               2               *
33              0               5               2               *
34              0               6               2               *
35              0               7               2               *
36              0               8               2               *
37              0               9               2               *
38              0               10              2               *
39              0               11              2               *
40              0               12              2               *
41              0               13              2               *
42              0               0               3               *
43              0               1               3               *
44              0               2               3               *
45              0               3               3               *
46              0               4               3               *
47              0               5               3               *
48              0               6               3               *
49              0               7               3               *
50              0               8               3               *
51              0               9               3               *
52              0               10              3               *
53              0               11              3               *
54              0               12              3               *
55              0               13              3               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 )
Socket 1:               ( 14 15 16 17 18 19 20 21 22 23 24 25 26 27 )
Socket 2:               ( 28 29 30 31 32 33 34 35 36 37 38 39 40 41 )
Socket 3:               ( 42 43 44 45 46 47 48 49 50 51 52 53 54 55 )
--------------------------------------------------------------------------------
```

```
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 ) ( 11 ) ( 12 ) ( 13 ) ( 14 ) ( 15 ) ( 16 ) ( 17 ) ( 18 ) ( 19 ) ( 2
0 ) ( 21 ) ( 22 ) ( 23 ) ( 24 ) ( 25 ) ( 26 ) ( 27 ) ( 28 ) ( 29 ) ( 30 ) ( 31 ) ( 32 ) ( 33 ) ( 34 ) ( 35 ) ( 36 ) ( 37 ) ( 38 ) ( 39 ) ( 40 ) ( 41 ) ( 42 )
 ( 43 ) ( 44 ) ( 45 ) ( 46 ) ( 47 ) ( 48 ) ( 49 ) ( 50 ) ( 51 ) ( 52 ) ( 53 ) ( 54 ) ( 55 )
--------------------------------------------------------------------------------
Level:                  2
Size:                   1 MB
Cache groups:           ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 ) ( 11 ) ( 12 ) ( 13 ) ( 14 ) ( 15 ) ( 16 ) ( 17 ) ( 18 ) ( 19 ) ( 2
0 ) ( 21 ) ( 22 ) ( 23 ) ( 24 ) ( 25 ) ( 26 ) ( 27 ) ( 28 ) ( 29 ) ( 30 ) ( 31 ) ( 32 ) ( 33 ) ( 34 ) ( 35 ) ( 36 ) ( 37 ) ( 38 ) ( 39 ) ( 40 ) ( 41 ) ( 42 )
 ( 43 ) ( 44 ) ( 45 ) ( 46 ) ( 47 ) ( 48 ) ( 49 ) ( 50 ) ( 51 ) ( 52 ) ( 53 ) ( 54 ) ( 55 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   19 MB
Cache groups:           ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 ) ( 14 15 16 17 18 19 20 21 22 23 24 25 26 27 ) ( 28 29 30 31 32 33 34 35 36 37 38 39 40 41 ) ( 42
43 44 45 46 47 48 49 50 51 52 53 54 55 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           4
--------------------------------------------------------------------------------
Domain:                 0
Processors:             ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 )
Distances:              10 21 21 21
Free memory:            93004.6 MB
Total memory:           96940.1 MB
--------------------------------------------------------------------------------
Domain:                 1
Processors:             ( 14 15 16 17 18 19 20 21 22 23 24 25 26 27 )
Distances:              21 10 21 21
Free memory:            95273.1 MB
Total memory:           98304 MB
--------------------------------------------------------------------------------
Domain:                 2
Processors:             ( 28 29 30 31 32 33 34 35 36 37 38 39 40 41 )
Distances:              21 21 10 21
Free memory:            95204.8 MB
Total memory:           98304 MB
--------------------------------------------------------------------------------
Domain:                 3
Processors:             ( 42 43 44 45 46 47 48 49 50 51 52 53 54 55 )
Distances:              21 21 21 10
Free memory:            95569.3 MB
Total memory:           98304 MB
--------------------------------------------------------------------------------
```

Architectures & Memory Access Models



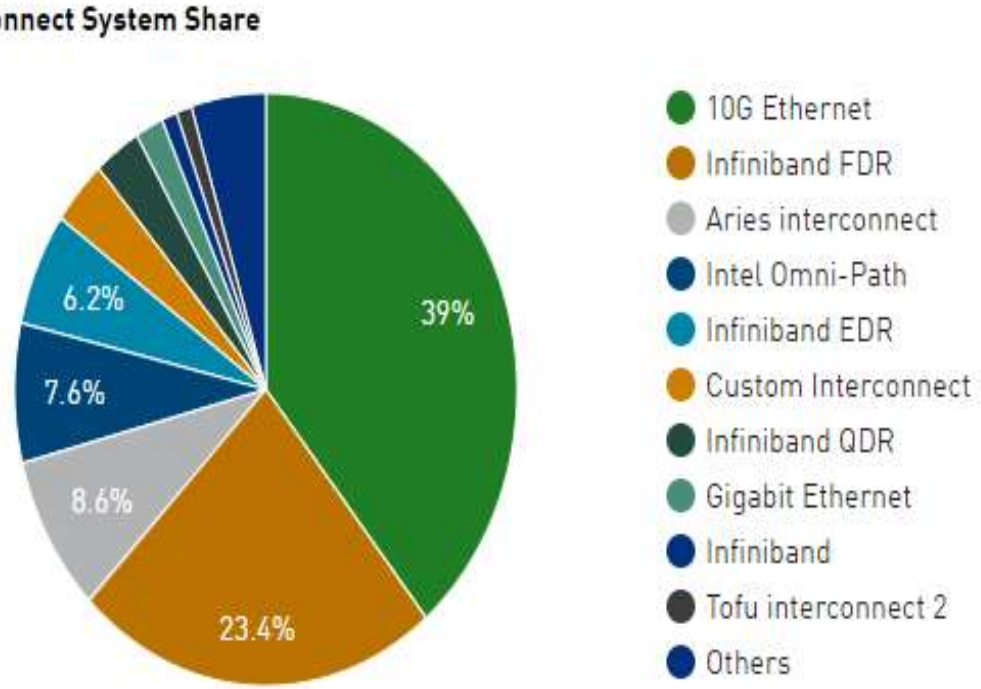
Networking/Interconnects

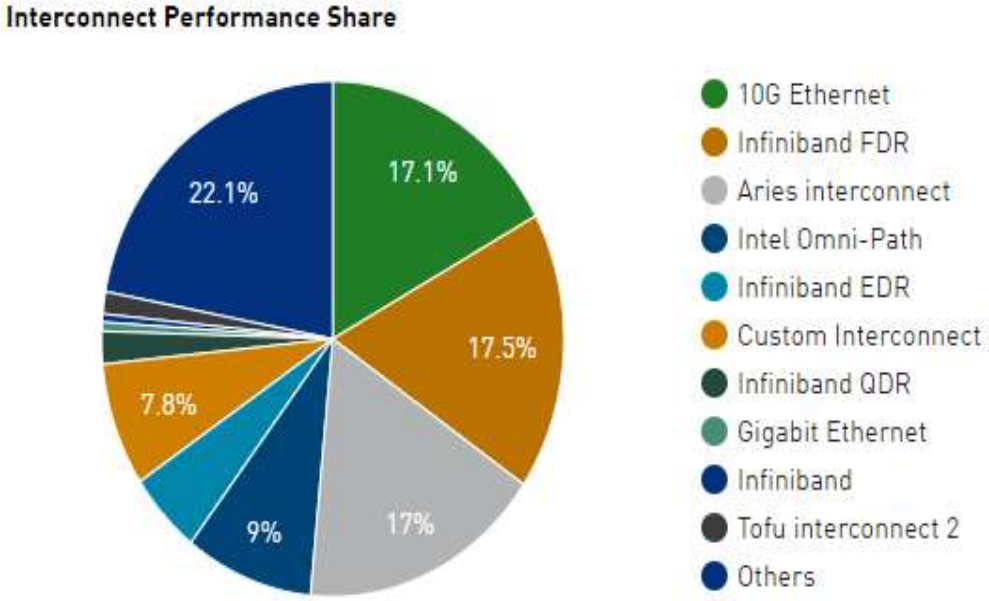
- A physical connection between the different components of a parallel system is provided by an **interconnection network**
- Important design criteria of networks are

-the **topology** describing the interconnection structure used to connect different processors or processors and memory modules and

-the **routing** technique describing the exact message transmission used within the network between processors or processors and memory modules

Communication Network-Top500-2017.6





Interconnect	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
10G Ethernet	195	39	127,779,818	253,903,182	7,174,560
Infiniband FDR	117	23.4	130,764,424	180,509,923	6,706,402
Aries interconnect	43	8.6	127,102,756	178,488,648	4,616,516
Intel Omni-Path	38	7.6	67,110,097	105,131,151	2,375,804
Infiniband EDR	31	6.2	38,724,581	52,471,007	4,219,264
Custom Interconnect	18	3.6	58,071,153	67,108,818	4,978,656
Infiniband QDR	15	3	15,286,050	23,403,893	940,312
Gigabit Ethernet	9	1.8	4,263,780	16,419,251	918,392
Infiniband	5	1	3,794,105	5,697,797	204,524
Tofu interconnect 2	5	1	10,422,200	11,530,310	354,384
Cray Gemini interconnect	5	1	21,030,100	31,700,246	962,552
40G Ethernet	3	0.6	1,632,600	2,393,453	28,008
Infiniband EDR/FDR	3	0.6	2,532,236	4,351,258	89,456
Infiniband FDR14	3	0.6	3,772,773	4,952,026	129,888
Proprietary	2	0.4	3,337,700	6,043,751	239,616
TH Express-2	2	0.4	35,934,090	57,976,934	3,294,720
Sunway	1	0.2	93,014,594	125,435,904	10,649,600
Intel Truscale	1	0.2	532,900	615,347	29,584
Intel TrueScale Infiniband	1	0.2	596,010	681,574	32,768
100G Ethernet	1	0.2	613,200	920,000	25,000
56G Infiniband FDR	1	0.2	1,013,721	1,372,134	32,984
Tofu interconnect	1	0.2	1,043,000	1,135,411	76,800

- **Static interconnection networks** connect nodes (processors or memory modules) *directly* with each other by fixed physical links;

- **Dynamic interconnection networks** connect nodes *indirectly* via switches and links

- **Static interconnection networks** can be described by a connection graph **G = (V, E)** where V is a set of nodes to be connected and E is a set of direct connection links between the nodes

Properties of Interconnection Networks

- number of nodes
- degree of the nodes
- diameter of the network

  the maximum distance between any pair of nodes
- bisection bandwidth

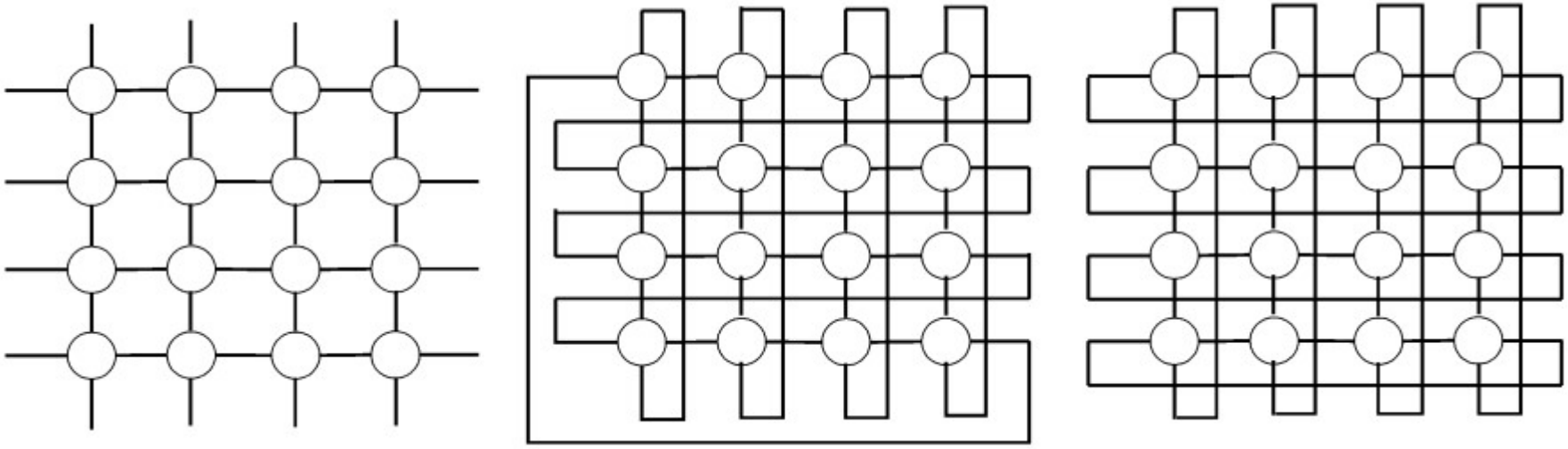
  the minimum number of edges that must be removed to partition the network into two parts of equal size without any connection between the two parts
- node and edge connectivity of the network

  the number of nodes or edges that must fail to disconnect the network
- flexibility of embeddings into other networks
- the embedding of other networks.

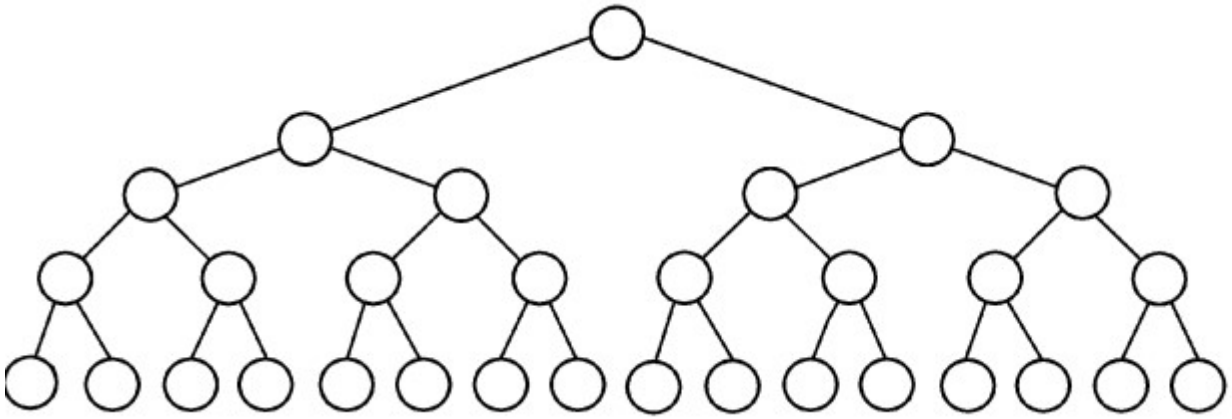
- a small diameter to ensure small distances for message transmission,
- a small node degree to reduce the hardware overhead for the nodes,
- a large bisection bandwidth to obtain large data throughputs,
- a large connectivity to ensure reliability of the network,
- embedding into a large number of networks to ensure flexibility, and
- easy extendability to a larger number of nodes.

- Many types of networks have been used for constructing parallel systems, ranging from relatively simple buses, to 2-D and 3-D meshes, to Clos networks, and to complex hypercube network topologies

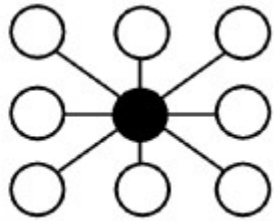
- There are some industry efforts, such as Inifiniband, to accelerate the rate of improvement in network bandwidth.

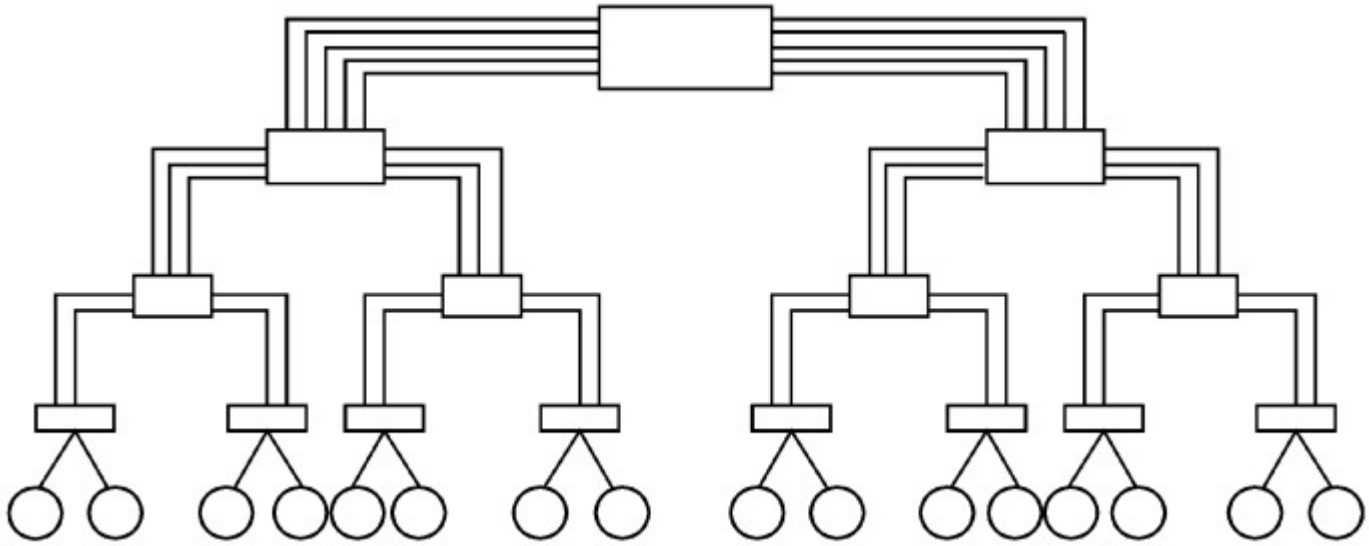


- **2D-mesh** **2D-Illiac** **2D-Torus**



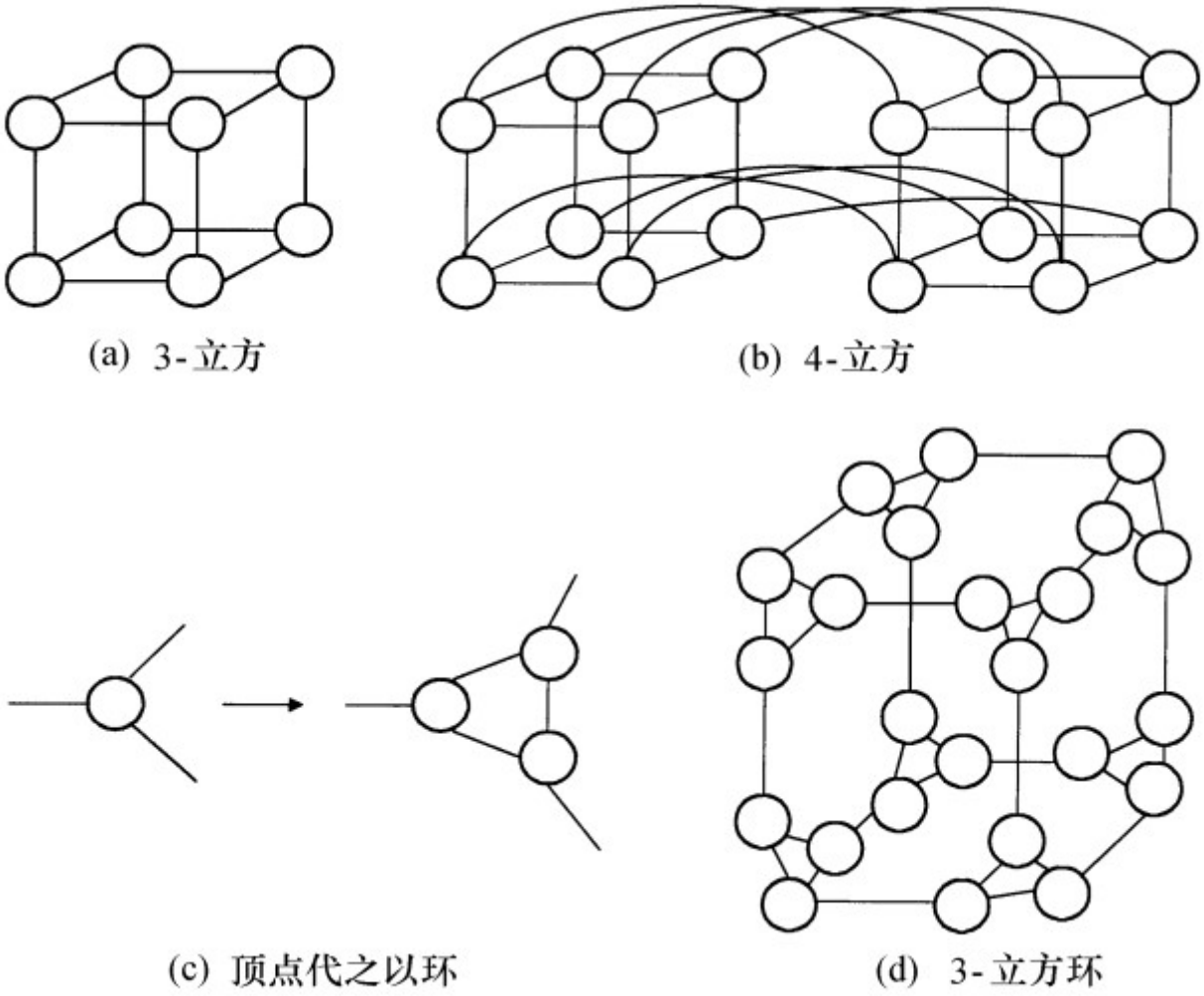
X — T r e e



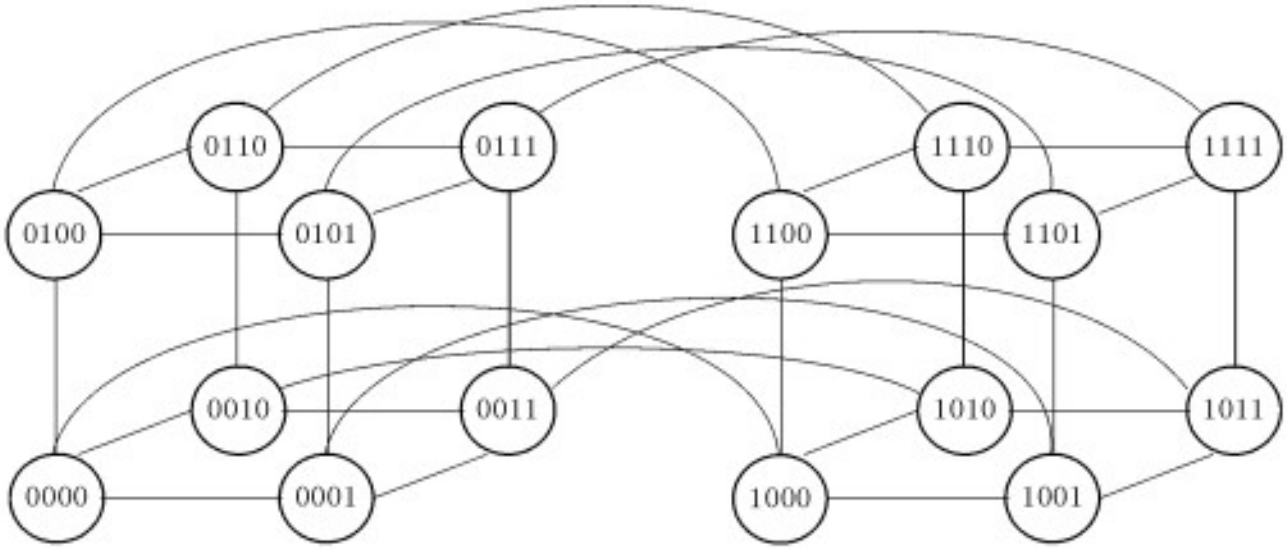


F a t T r e e

(a) 3-cube
(b) 4-cube
(c) V to Cycle
(d) 3-Cube Connected Cycle



Hypercube network



Basic performance characteristics of networks

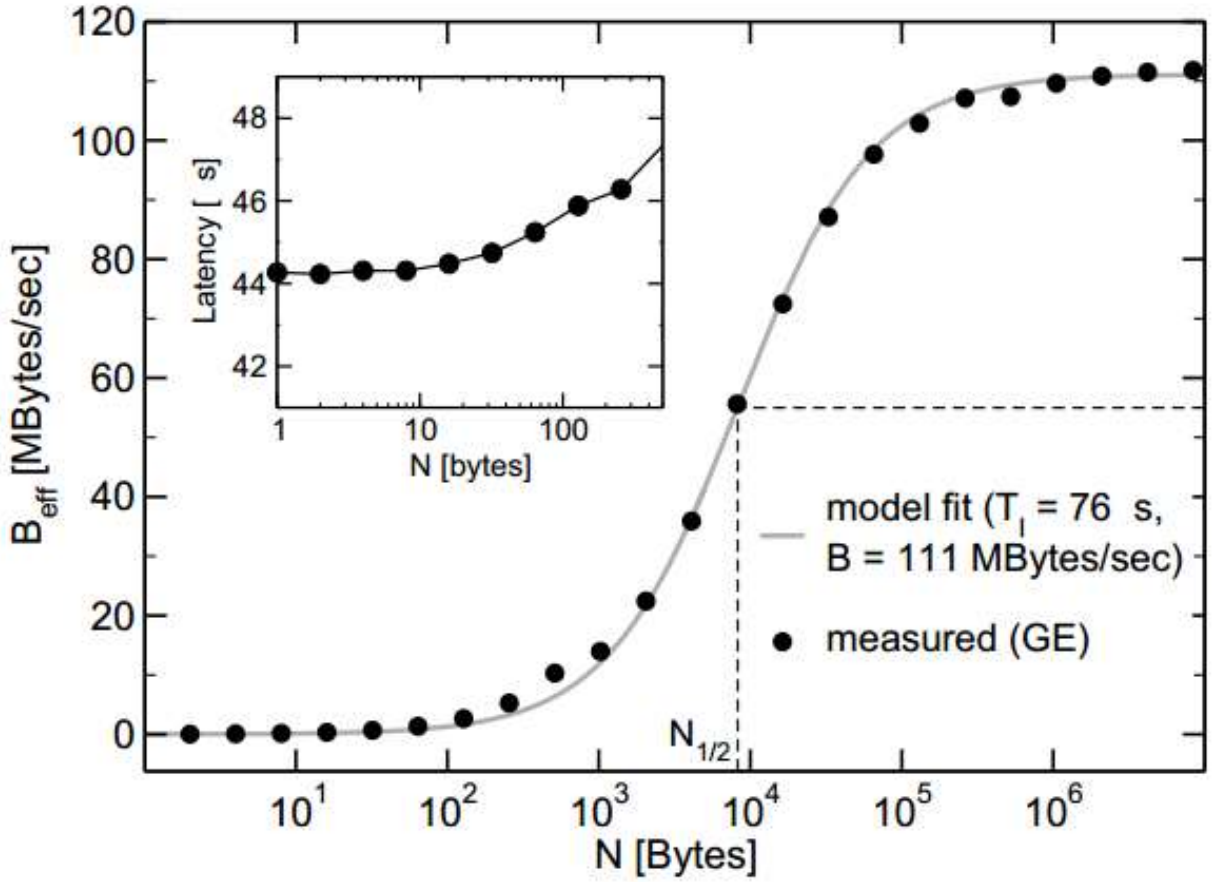
- **Point-to-point connections**

-Assuming that the total transfer time for a message of size *N* [bytes] is composed of latency and streaming parts,

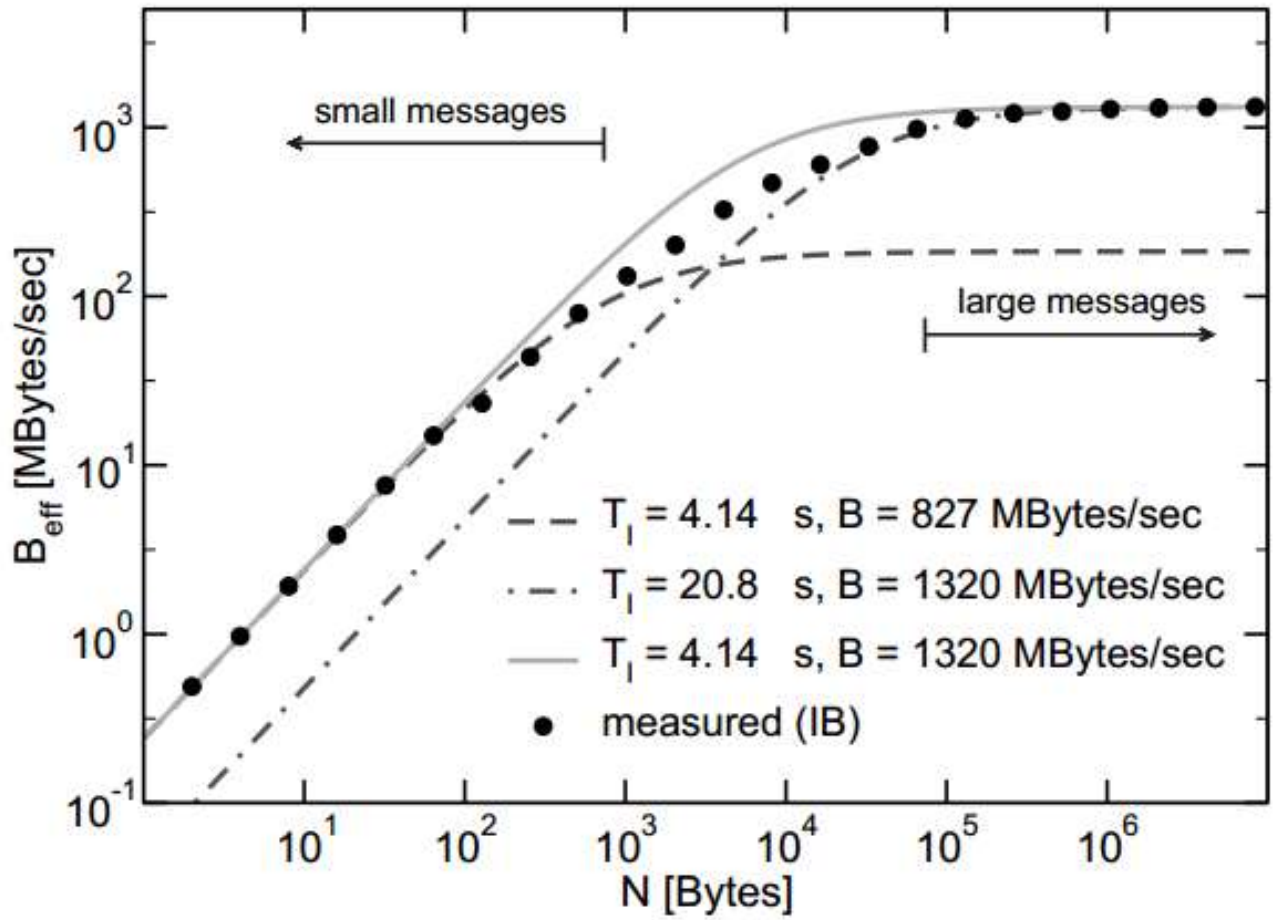
$$T = T_\ell + \frac{N}{B}$$

-*B* being the maximum (asymptotic) network bandwidth in MBytes/sec, the effective bandwidth is

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

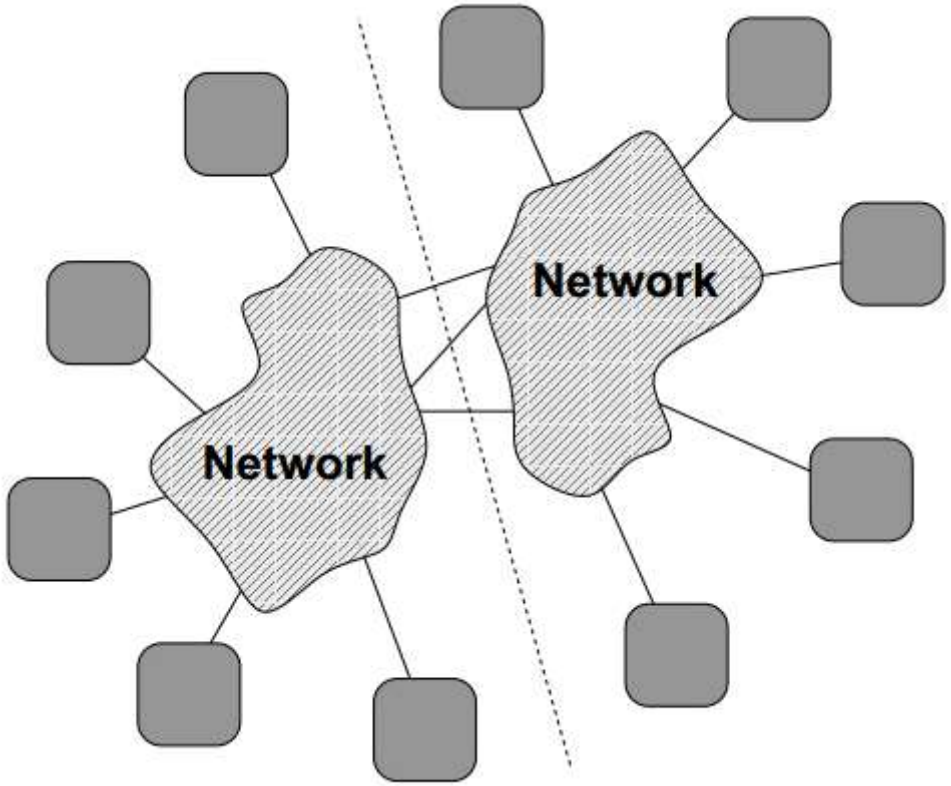


- Fit of the model for effective bandwidth to data measured on a GigE network. The fit cannot accurately reproduce the measured value of T_ℓ , $N_{1/2}$ is the message length at which half of the saturation bandwidth is reached



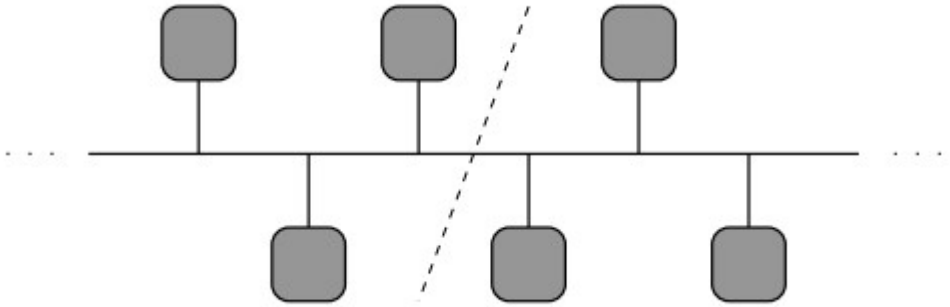
- Fit of the model for effective bandwidth to data measured on a DDR InfiniBand network. “Good” fits for asymptotic bandwidth (dotted-dashed) and latency (dashed) are shown separately, together with a fit function that unifies both (solid)

Bisection bandwidth

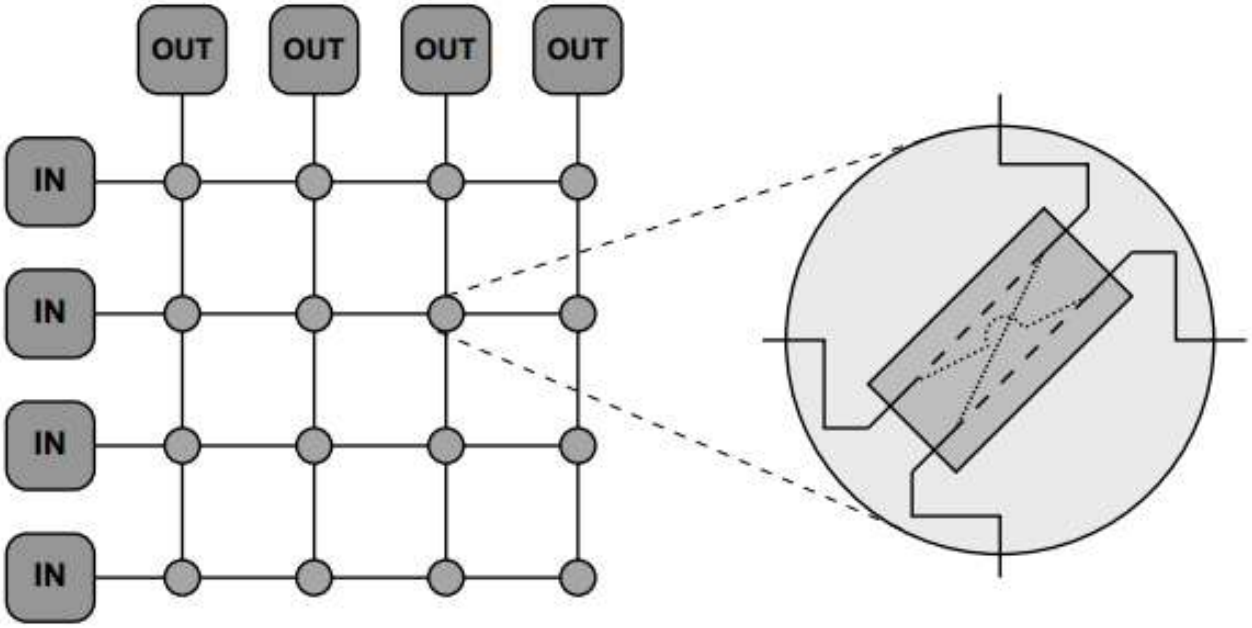


- The bisection bandwidth *B*b is the sum of the bandwidths of the minimal number of connections cut (three in this example) when dividing the system into two equal parts.

Buses



- 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

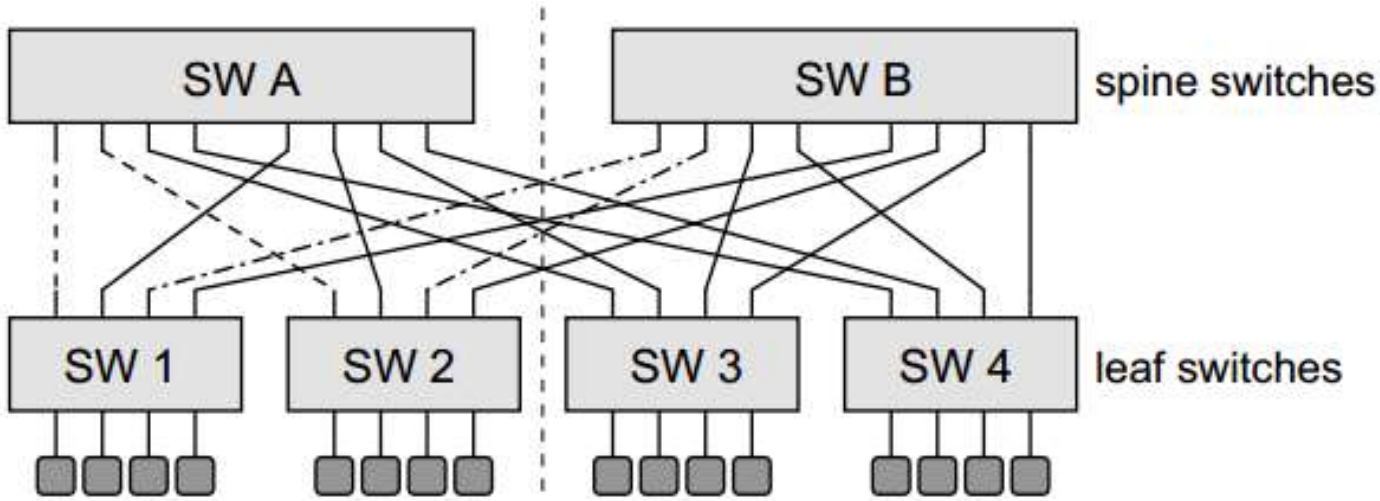


- A flat, fully nonblocking two-dimensional crossbar network. Each circle represents a possible connection between two devices from the “IN” and “OUT” groups, respectively, and is implemented as a 2×2 switching element. The whole circuit can act as a four-port nonblocking switch.

Switched and fat-tree networks

- A switched network subdivides all communicating devices into groups. The devices in one group are all connected to a central network entity called a ***switch*** in a star-like manner.
- Switches are then connected with each other or using additional switch layers. In such a network, the distance between two communicating devices varies according to how many "**hops**" a message has to sustain before it reaches its destination. Therefore, a multiswitch hierarchy is necessarily heterogeneous with respect to latency.
- The maximum number of hops required to connect two arbitrary devices is called the ***diameter*** of the network. For a bus, the diameter is one.

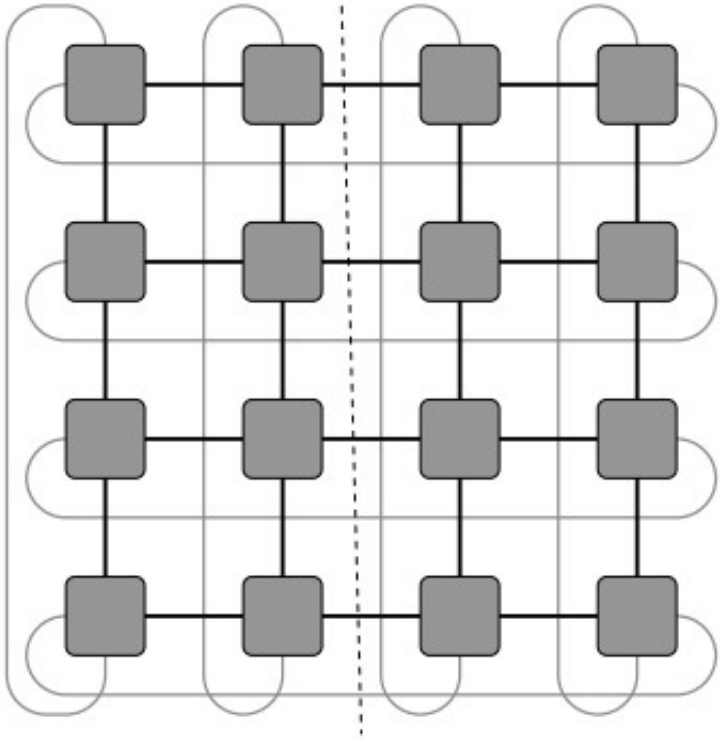
- A fully nonblocking full-bandwidth fat-tree network with two switch layers. The switches connected to the actual compute elements are called *leaf switches*, whereas the upper layers form the *spines* of the hierarchy.



Mesh networks

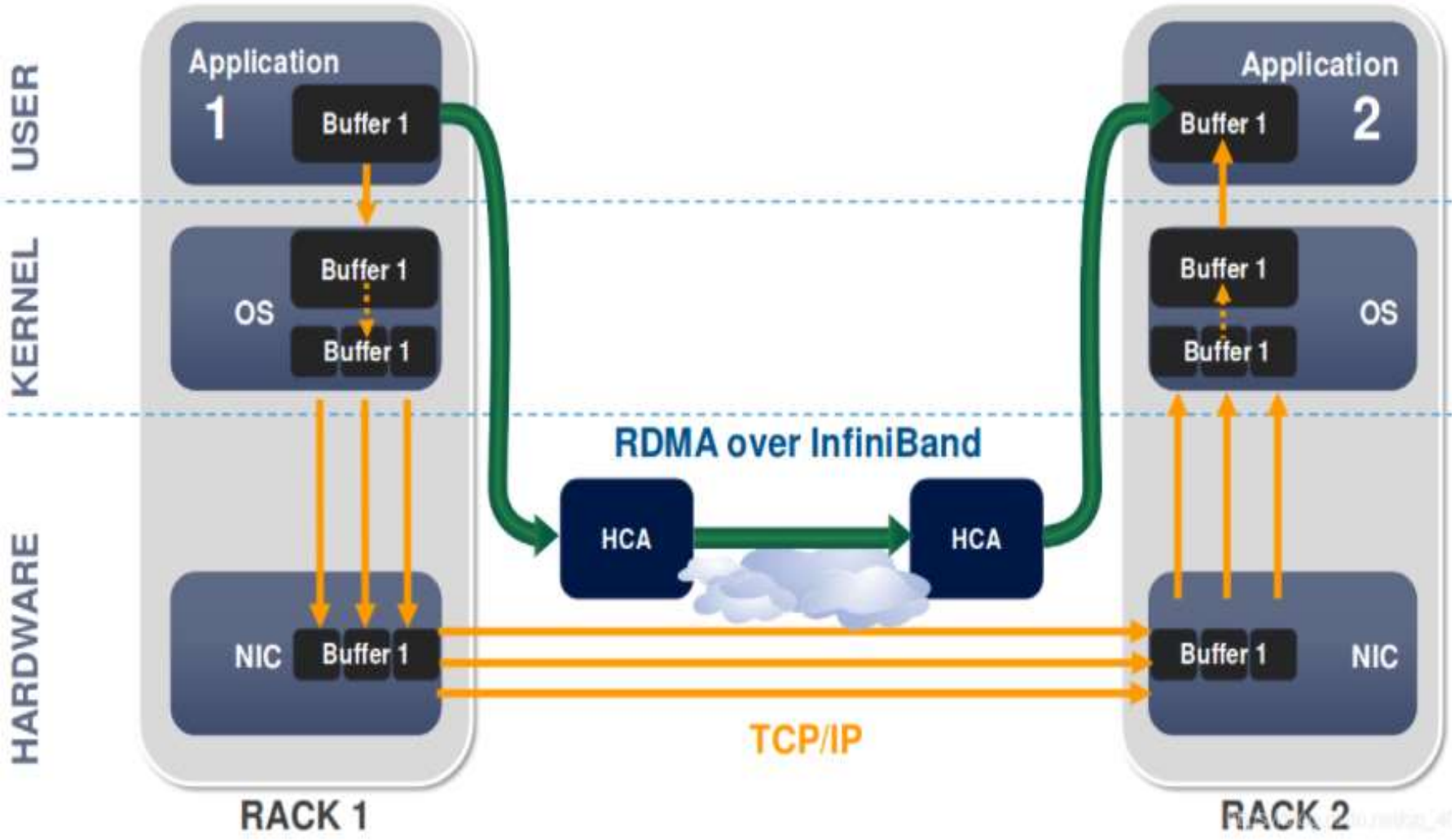
- Fat-tree switch hierarchies have the disadvantage of limited scalability in very large systems, mostly in terms of price vs. performance.
- In order to overcome those drawbacks and still arrive at a controllable scaling of bisection bandwidth, large MPP machines like the IBM Blue Gene or the Cray XT use ***mesh networks,*** usually in the form of multidimensional (hyper-) cubes

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



- Networks, are, however, fundamentally constrained by the speed of light. Latencies can never be less than 3 ns per meter.
- Another constraint is the way in which the network is used by the software. The approaches that are currently used by most software involve the operating system (OS) in most networking operations, including most data transfers between the main memory and the network.

  -OS bypass/User mode/Zero Copy **-RDMA**


USER
KERNEL
HARDWARE
Application 1
Buffer 1
OS
Buffer 1
Buffer 1
NIC
Buffer 1
RDMA over InfiniBand
HCA
HCA
TCP/IP
Application 2
Buffer 1
Buffer 1
OS
Buffer 1
Buffer 1
NIC
RACK 1
RACK 2

Attributes of SIX switching network topologies

	Processor nodes	Switch nodes	Diameter	Bisection width	Edges/ nodes	Constant edge length
2-D mesh	$n = d^2$	n	$2(\sqrt{n} - 1)$	$\sqrt{n}$	4	Yes
Binary tree	$n = 2^d$	$2n - 1$	$2 \log n$	1	3	No
4-ary hypertree	$n = 4^d$	$2n - \sqrt{n}$	$\log n$	$n/2$	6	No
Butterfly	$n = 2^d$	$n(\log n + 1)$	$\log n$	$n/2$	4	No
Hypercube	$n = 2^d$	n	$\log n$	$n/2$	$\log n$	No
Shuffle-exchange	$n = 2^k$	n	$2 \log n - 1$	$\approx n/\log n$	2	No

Topology	Bypass networks	Dimensions	Node degree	Diameter (hop)	Bisection width (×1024) links)	Average distance (hop)
3D Torus	N/A	$32 \times 32 \times 16$	6	40	1	20.001
4D Torus	N/A	$16 \times 16 \times 8 \times 8$	8	24	2	12.001
6D Torus	N/A	$8 \times 8 \times 4 \times 4 \times 4 \times 4$	12	16	4	8.000
3D SRT	$L = 1$; $l_1 = 4$	$32 \times 32 \times 16$	10	24	9	12.938
2D SRT	$l_{max} = 6$	128×128	8	13	1.625	8.722
3D MSRT	$L = 2$; $l_1 = 2$, $l_2 = 4$	$32 \times 32 \times 16$	8	16	2.5	9.239 9.413

Network G with n nodes	Degree $g(G)$	Diameter $\delta(G)$	Edge- connectivity $ec(G)$	Bisection bandwidth $B(G)$
Complete graph	$n-1$	1	$n-1$	$\left(\frac{n}{2}\right)^2$
Linear array	2	$n-1$	1	1
Ring	2	$\lfloor \frac{n}{2} \rfloor$	2	2
d-Dimensional mesh $(n = r^d)$	$2d$	$d(\sqrt[d]{n}-1)$	d	$n^{\frac{d-1}{d}}$
d-Dimensional torus $(n = r^d)$	$2d$	$d\left\lfloor \frac{\sqrt[d]{n}}{2} \right\rfloor$	$2d$	$2n^{\frac{d-1}{d}}$
k-Dimensional hyper-cube $(n = 2^k)$	$\log n$	$\log n$	$\log n$	$\frac{n}{2}$
k-Dimensional CCC network $(n = k2^k$ for $k \geq 3)$	3	$2k-1+\lfloor k/2 \rfloor$	3	$\frac{n}{2k}$
Complete binary tree $(n = 2^k - 1)$	3	$2\log \frac{n+1}{2}$	1	1
k-ary d-cube $(n = k^d)$	$2d$	$d\lfloor \frac{k}{2} \rfloor$	$2d$	$2k^{d-1}$

Parallel I/O and Networking

- The redundant arrays of inexpensive disks (RAID) approach is an example of the benefits of parallelism in I/O
- The simplest form of parallelism in networks is the use of multiple paths, each carrying part of the traffic.

Parallel Architecture Design Tradeoffs

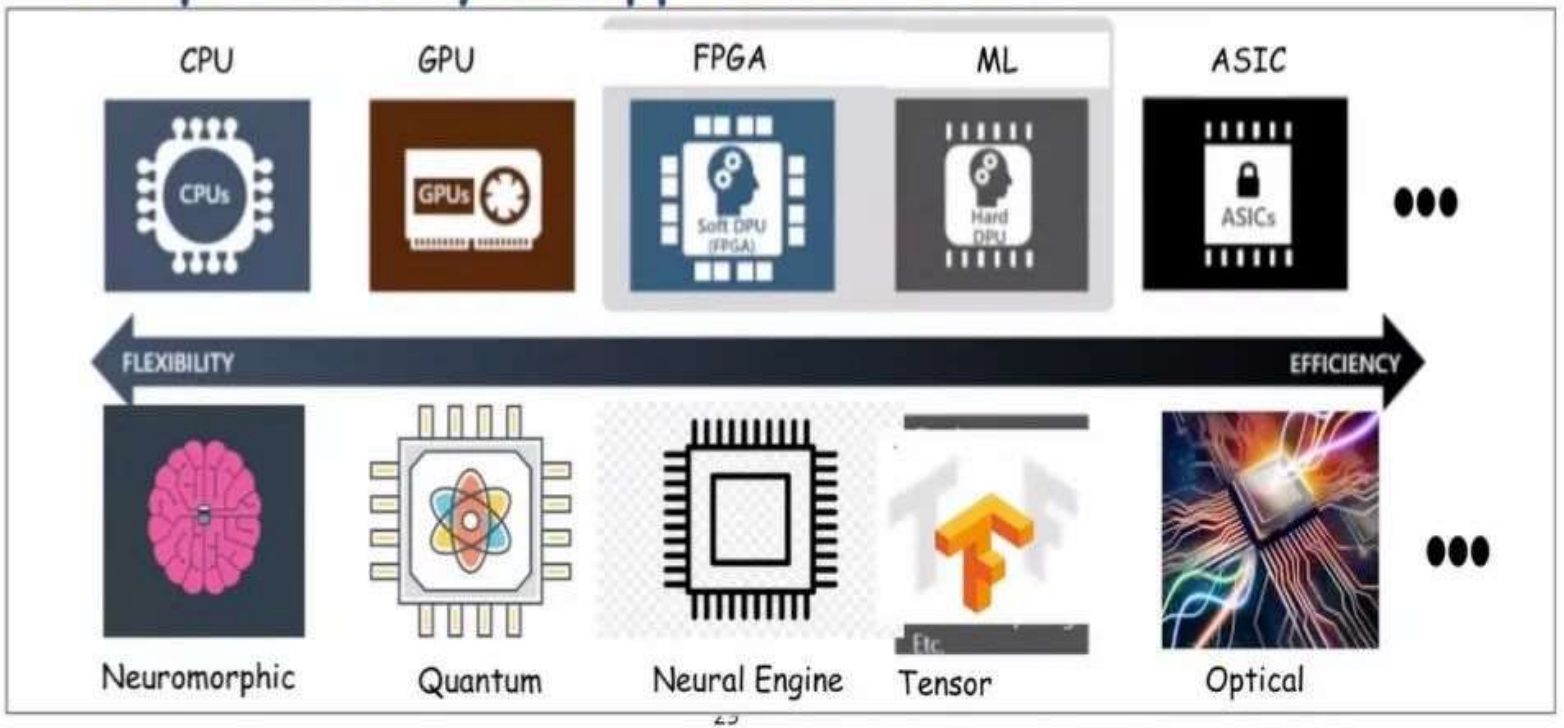
- Parallelism is a powerful approach to improving the performance of a computer system.
  - All systems employ some degree of parallelism, even if it is only parallel data paths between the memory and the CPU.
  - Parallelism is particularly good at solving problems related to bandwidth or throughput; it is less effective at dealing with latency or start-up costs
  - one of the major problems in designing any computer is providing a **high-bandwidth, low-latency path between the CPU and memory**
- **Combined approaches for greater performance**

- HPC Hardware is Constantly Changing
  - Scalar
  - Vector
  - Distributed
  - Accelerated
  - Mixed precision
- Three computer revolutions
  - High performance computing
  - Deep learning
  - Edge & AI

- Algorithm / Software advances follows hardware.
  - And there is “plenty of room at the top”

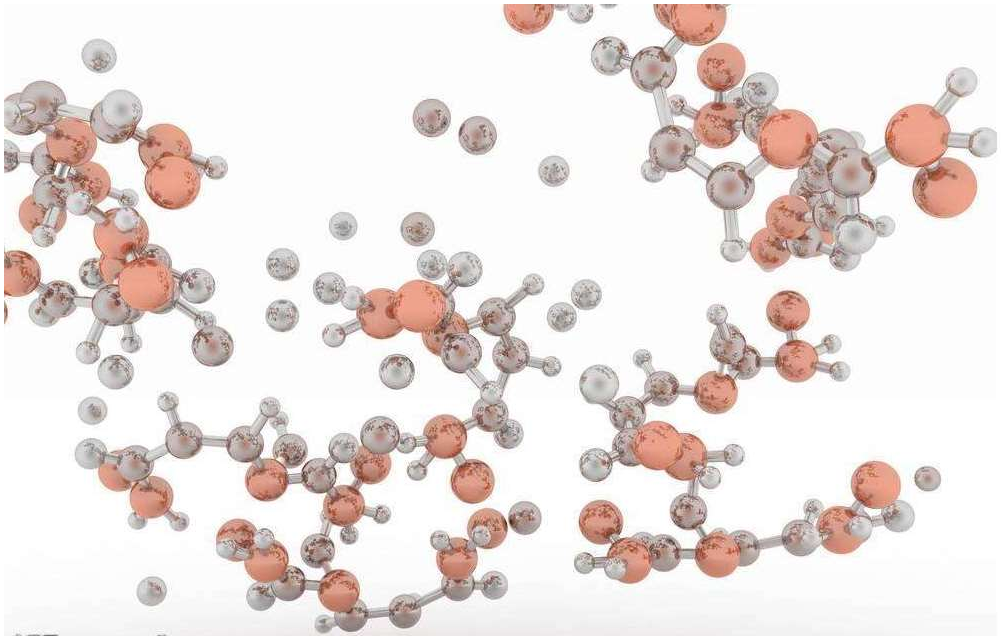
Future HPC Systems Will be Customized…

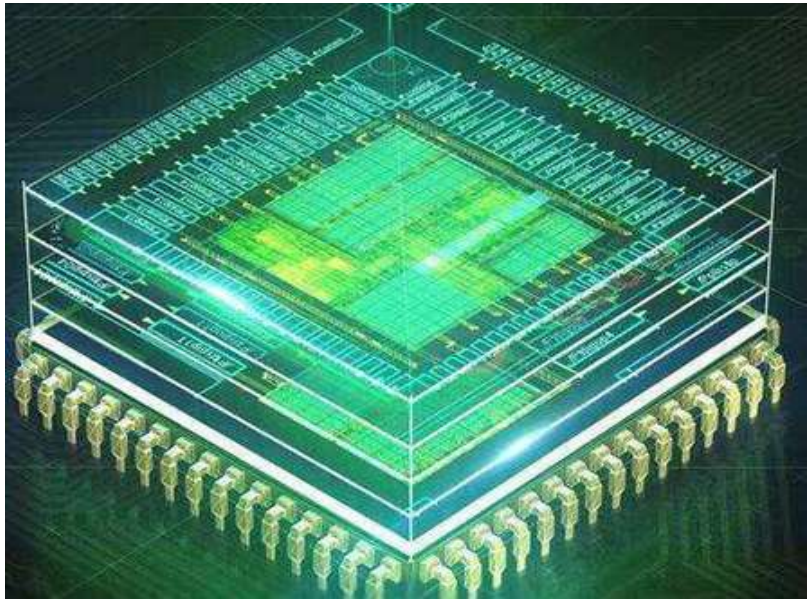
- You will be able to dial up what you need in your computer for your application mix …



- Courtesy to Prof. Jack Dongarra

- **Biological computing,** computing based on biological elements often seeks to exploit parallelism by using **molecules** as processing elements
- **Quantum computing**, particularly quantum computing based on exploiting the superposition principle, is a fundamentally different kind of parallelism





Thank you!