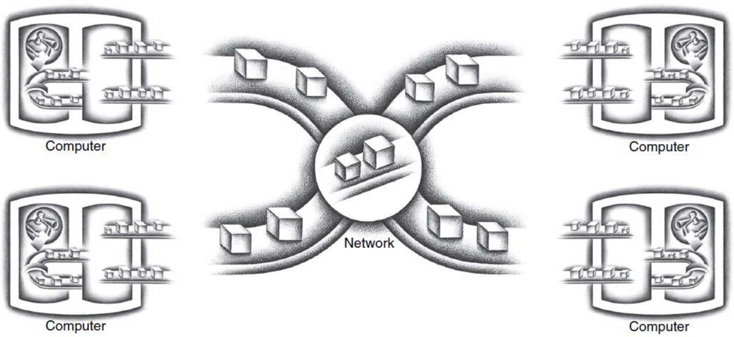
Chapter 7: Introduction to Multi-processing



Ngo Lam Trung

[with materials from *Computer Organization and Design,* Patterson & Hennessy, *Computer Organization and Architecture*, William Stallings, and *M.J. Irwin's presentation*, PSU 2008]

Introduction

- Overall goal: increasing performance
  - For a large software
  - For a large number of small individual software
  - With energy efficiency
- Approaches in previous chapters?
  - Pipeline
  - Super scaler
  - Multi-threaded

    → increase performance of a single CPU core

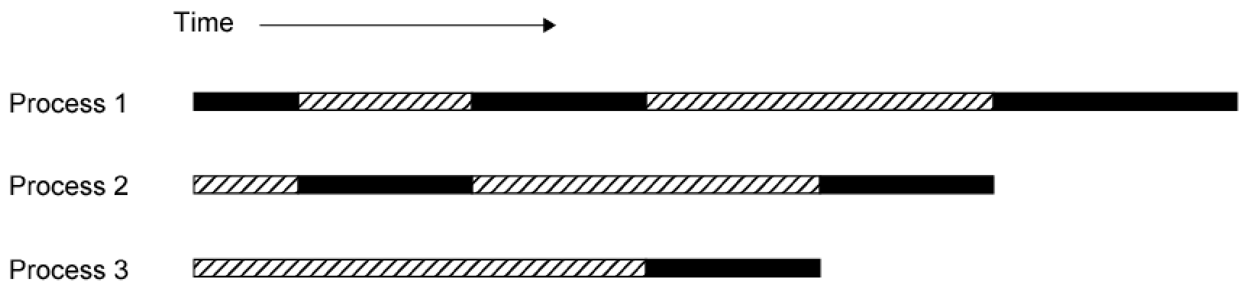
Difficulty in parallelism

- The difficulty with parallelism is not the hardware!
- It is difficult to write soft ware that uses multiple processors to complete one task faster, and the problem gets worse as the number of processors increases.
  - Conventional algorithm have been designed as sequential
  - Divide the job to multiple processors may lead to large communication overhead

		Software	
		Sequential	Concurrent
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Core i7	Windows Vista Operating System running on an Intel Core i7

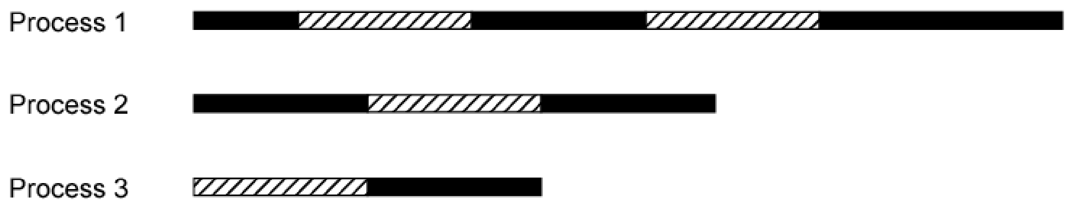
Multi-thread vs multi-processing

- Multi-thread



(a) Interleaving (multiprogramming, one processor)

- Multi-processing



(b) Interleaving and overlapping (multiprocessing; multiple processors)

- Comparison?

Multiprocessing

- Pollack:
  - "Performance is roughly proportional to square root of increase in complexity"
  - double the logic in a processor core, then it delivers only 40% more performance
- The use of multiple cores has the potential to provide near-linear performance improvement with the increase in the number of cores

Multiprocessing

- Replacing large inefficient processors with many smaller, efficient processors

➔better performance per joule

➔Improved energy efficiency joins scalable performance

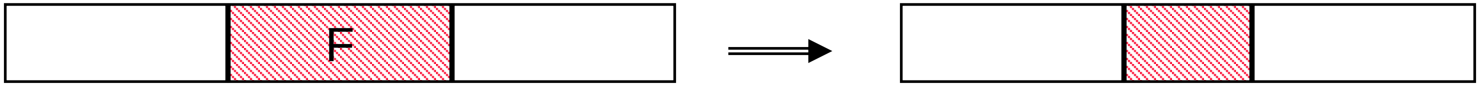
- ***Task-level/Process-level parallelism***: multiple independent tasks running on multiple processors
- ***Parallel processing program***: single program running on multiple processors simultaneously

Encountering Amdahl's Law

- Speedup due to enhancement E is

$$\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}$$

- Suppose that enhancement E accelerates a fraction F (F <1) of the task by a factor S (S>1) and the remainder of the task is unaffected



$$\text{ExTime w/ E} = \text{ExTime w/o E} \times$$

$$\text{Speedup w/ E} =$$

Example 1: Amdahl's Law

Speedup w/ E =

- Consider an enhancement which runs 20 times faster but which is only usable 25% of the time.

Speedup w/ E =

- What if its usable only 15% of the time?

Speedup w/ E =

- Amdahl's Law tells us that to achieve linear speedup with 100 processors, none of the original computation can be scalar!

- To get a speedup of 90 from 100 processors, the percentage of the original program that could be scalar would have to be 0.1% or less

Speedup w/ E =

Example 2: Amdahl's Law

Speedup w/ E = 1 / ((1-F) + F/S)

- Consider summing 10 scalar variables and two 10x10 matrices (matrix sum) on 10 processors

  Speedup w/ E =

- What if there are 100 processors ?

  Speedup w/ E =

- What if the matrices are 100x100 (or 10,010 adds in total) on 10 processors?

  Speedup w/ E =

- What if the matrices are 100x100 and there are 100 processors?

  Speedup w/ E =

Scaling

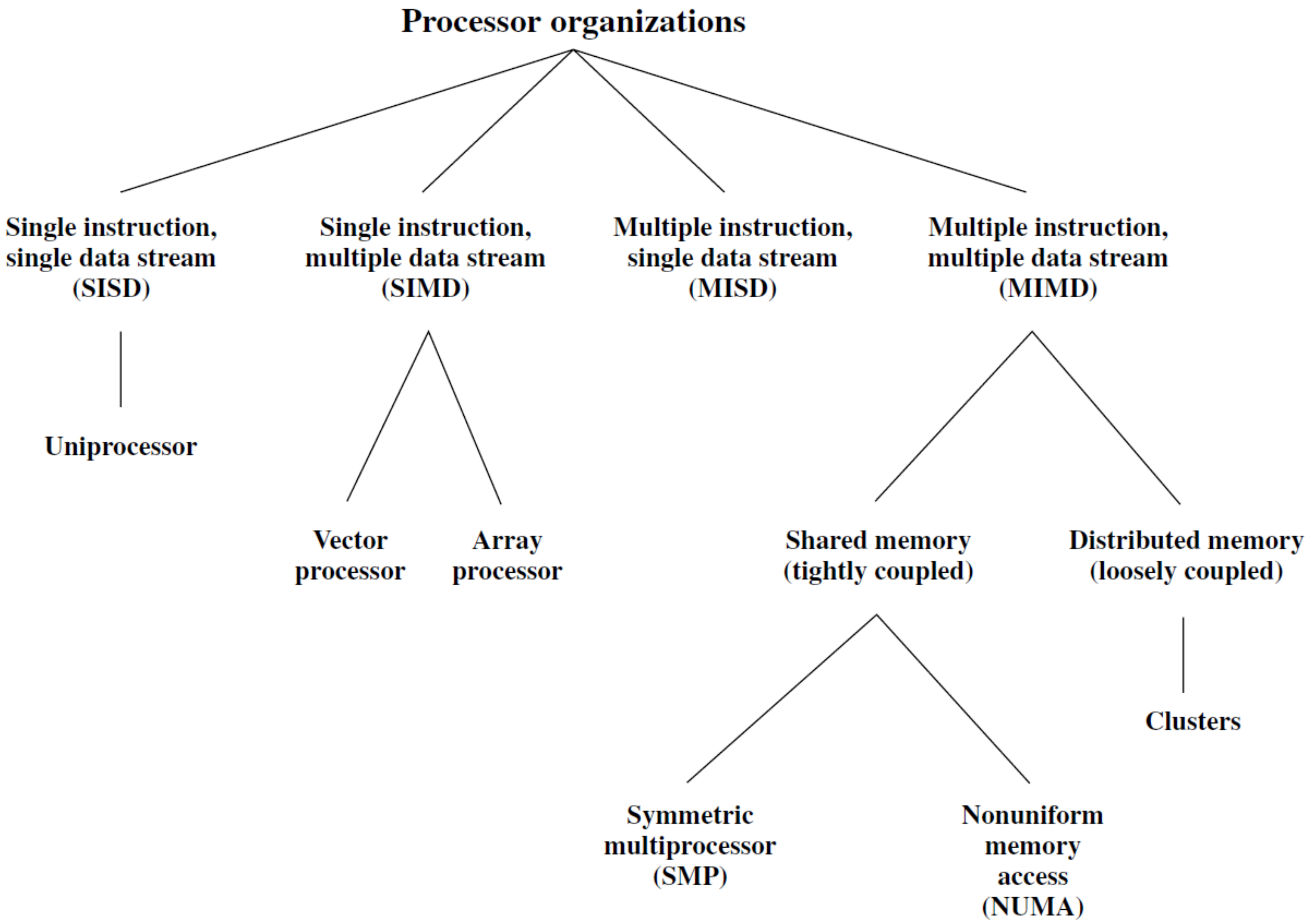
- To get good speedup on a multiprocessor while keeping the problem size fixed is harder than getting good speedup by increasing the size of the problem.
  - Strong scaling – when speedup can be achieved on a multiprocessor without increasing the size of the problem
  - Weak scaling – when speedup is achieved on a multiprocessor by increasing the size of the problem proportionally to the increase in the number of processors

- Load balancing is another important factor. Just a single processor with twice the load of the others cuts the speedup almost in half

Multiprocessor Key Questions

- Q1 – How do they share data?
- Q2 – How do they coordinate?
- Q3 – How scalable is the architecture? How many processors can be supported?

Types of Parallel Processor Systems



Types of Parallel Processor Systems

- Single instruction, single data (**SISD**) stream: A single processor executes a single instruction stream to operate on data stored in a single memory.
- Single instruction, multiple data (**SIMD**) stream: A single machine instruction controls the simultaneous execution of a number of processing elements
- Multiple instruction, single data (**MISD**) stream: not implemented
- Multiple instruction, multiple data (**MIMD**) stream: A set of processors simultaneously execute different instruction sequences on different data sets.

Types of Parallel Processor Systems

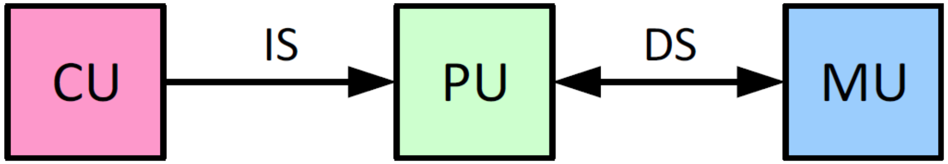
- SISD/SIMD/MISD/MIMD

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Core i7

- SPMD: Single Program, Multiple Data streams.

➔The conventional MIMD programming model, where a single program runs across all processors.

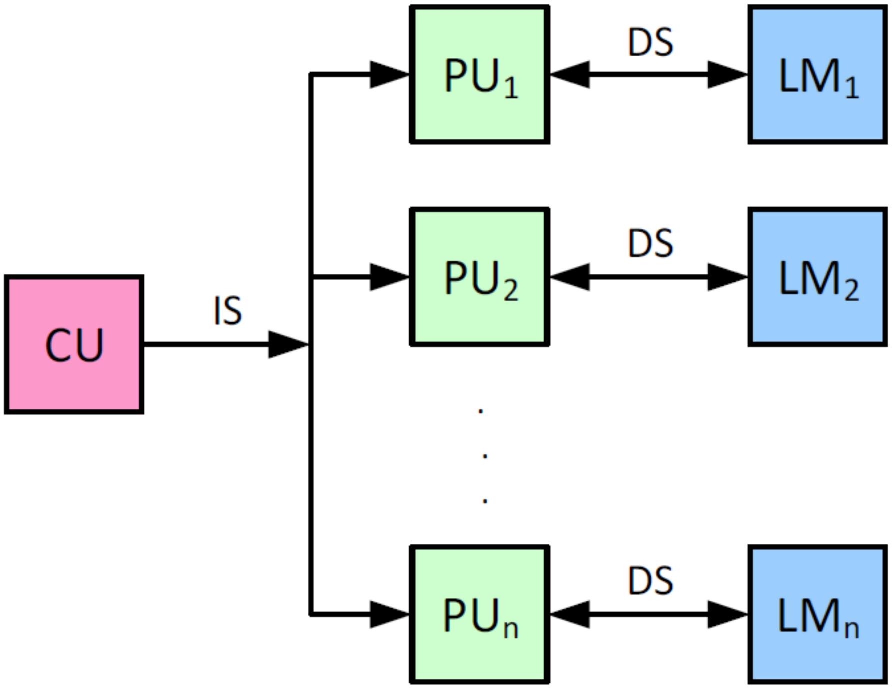
SISD



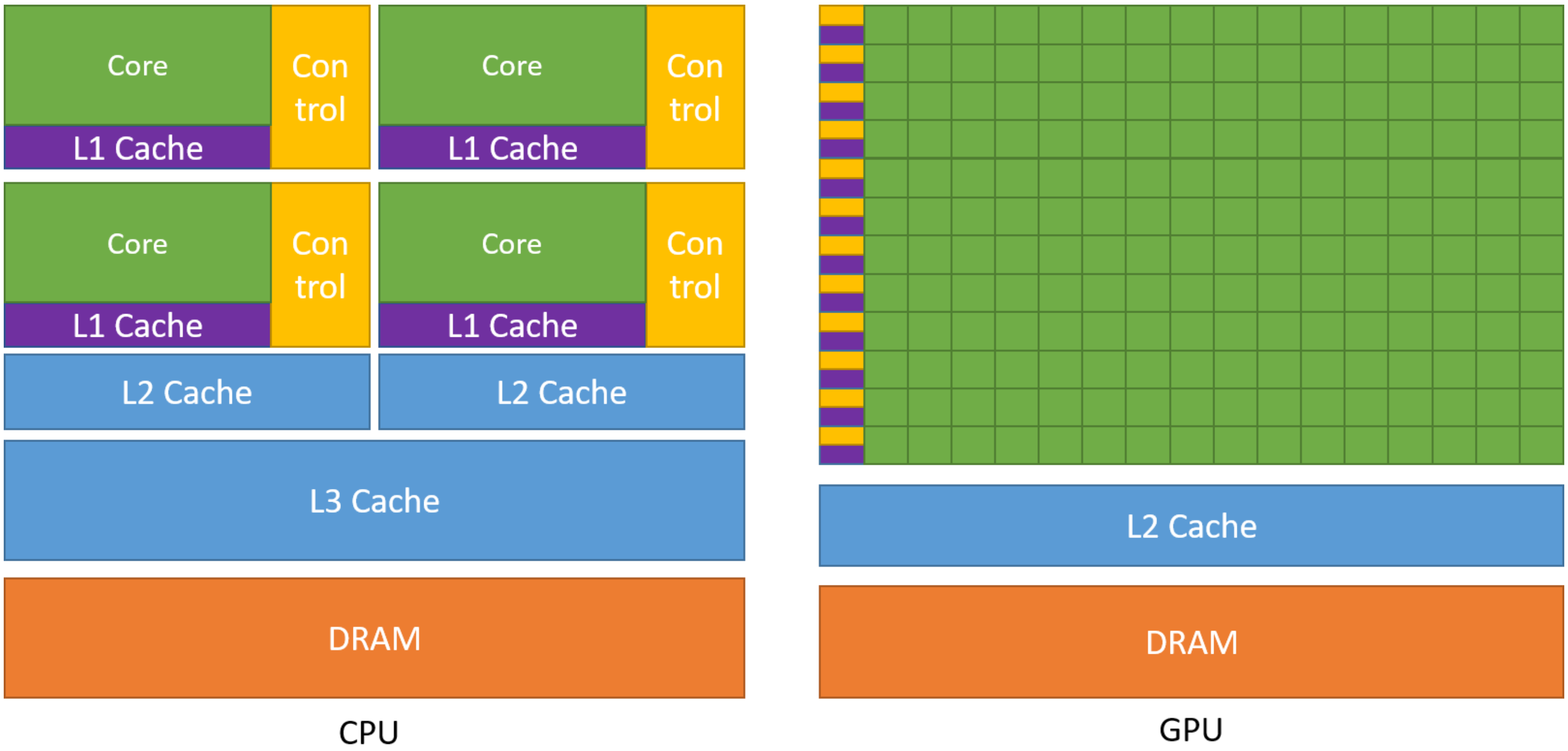
- CU: Control Unit
- PU: Processing Unit
- MU: Memory Unit
- Sequential execution
- Data stored in a single main memory
- ➔ Uniprocessor computer

SIMD

- 1 instruction stream
- Multiple processing units
- Each PC processes data from a separate memory
- All PUs execute the same instruction stream from CU
- Example: GPU

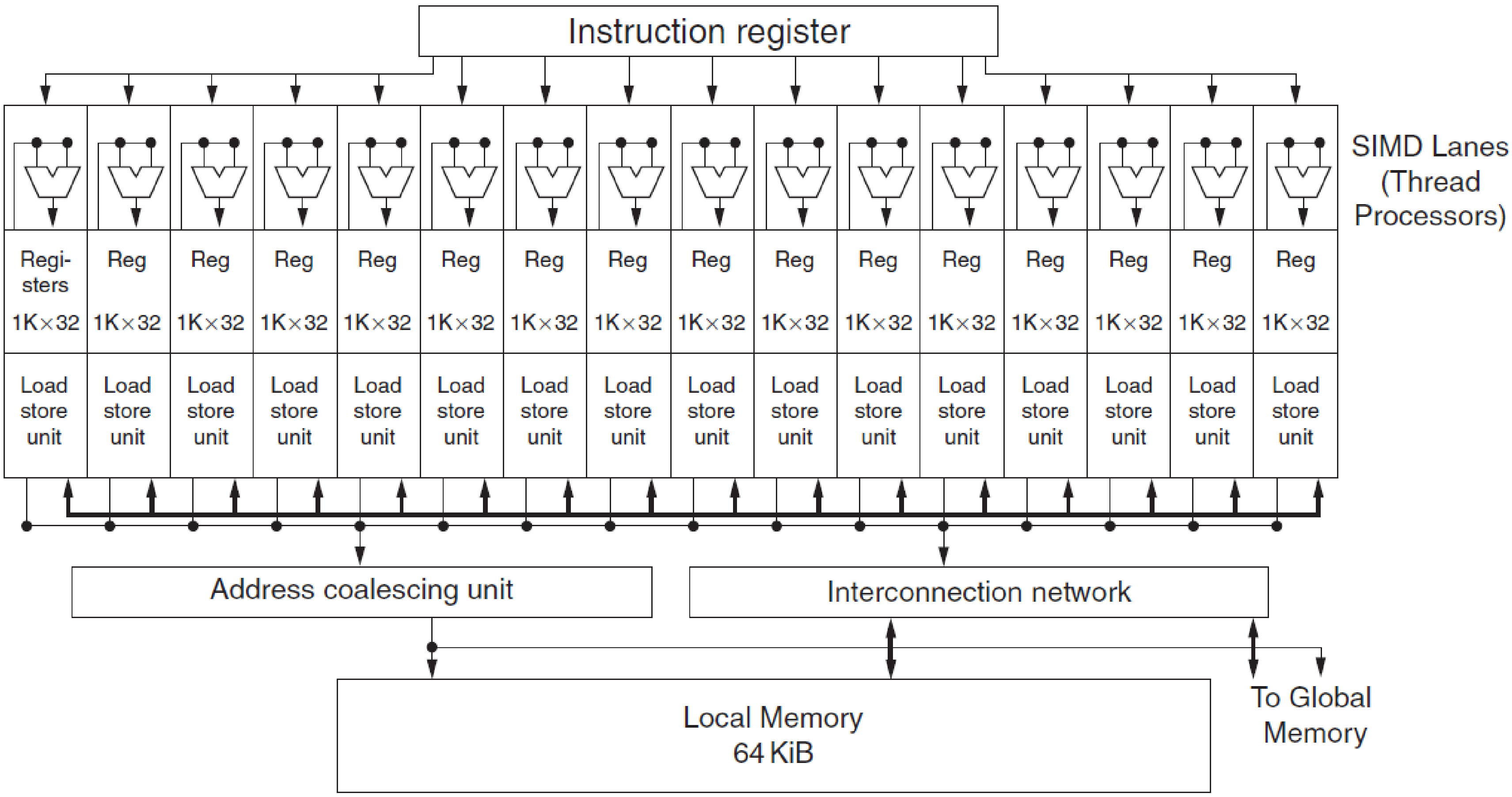


CPU vs GPU



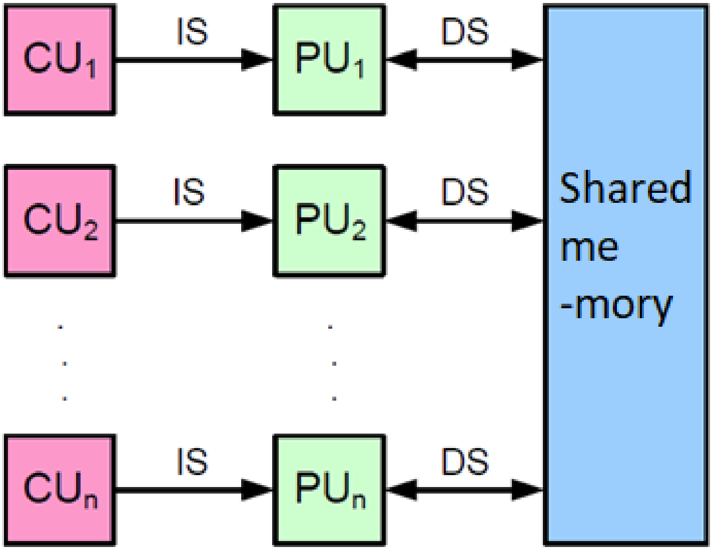
https://docs.nvidia.com/cuda/cuda-c-programming-guide/graphics/gpu-devotes-more-transistors-to-data-processing.png

Simplified block diagram of a SIMD Processor

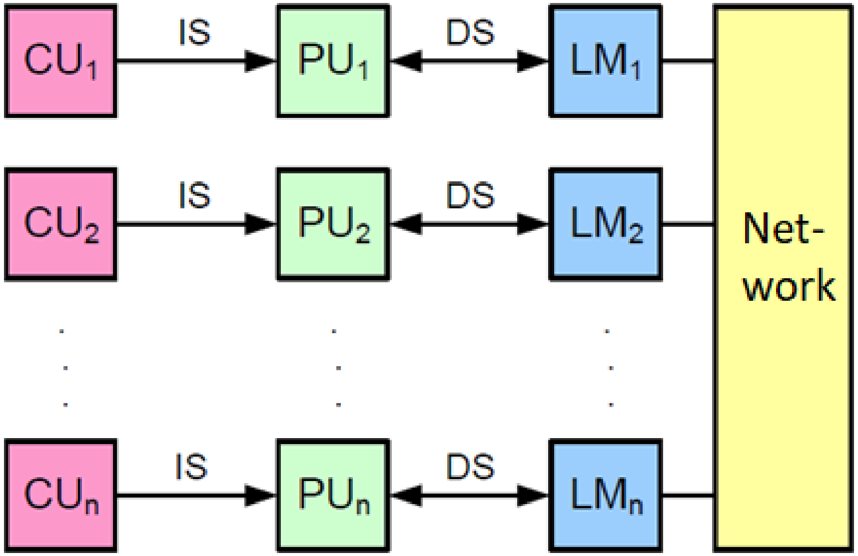


MIMD

- Multiple instruction, multiple data
- Require multiple CUs and PUs
- Shared or distributed memory



MIMD with shared memory



MIMD with distributed memory

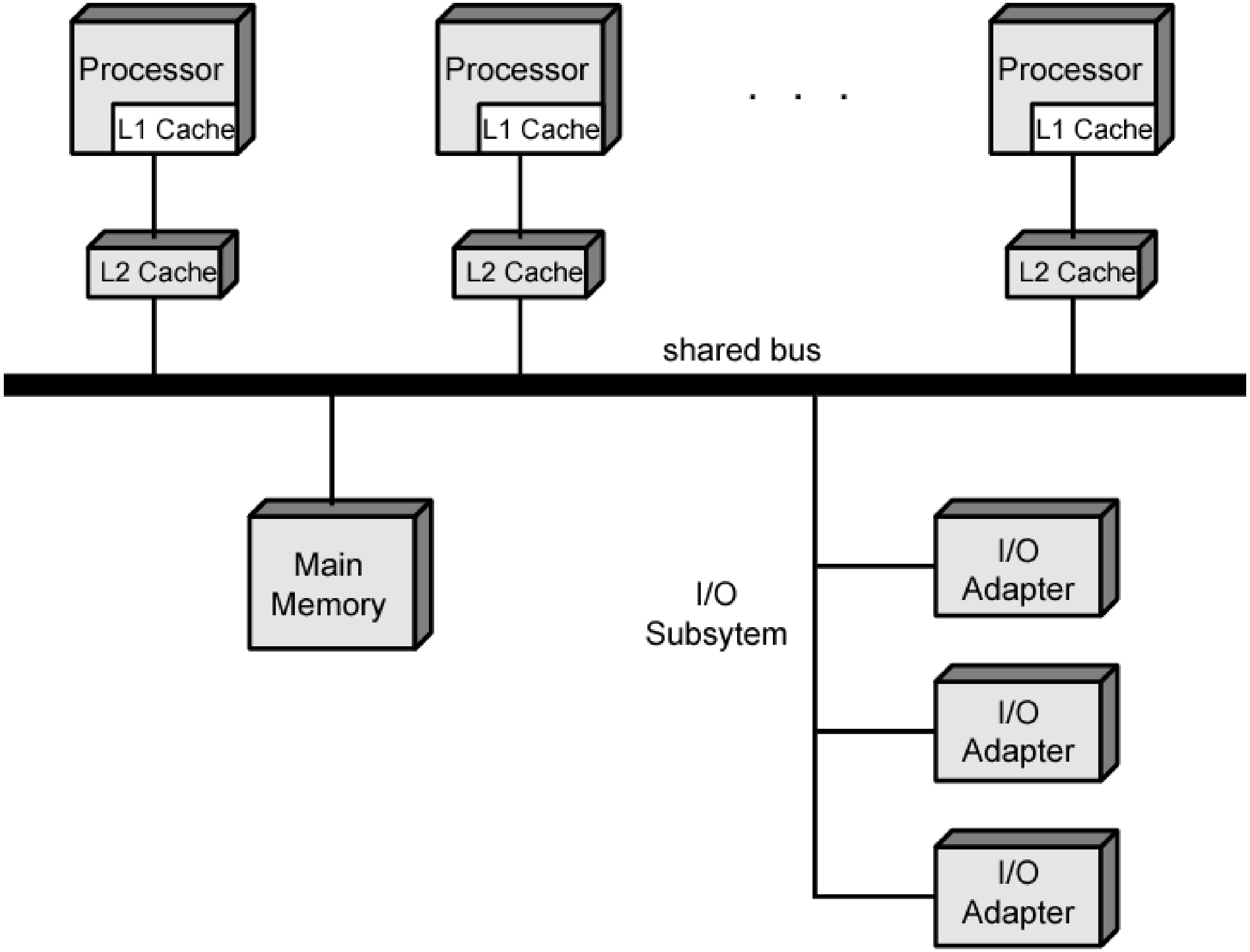
Types of MIMD

- Tightly coupled: standalone CPUs connect to memory and IOs by system buses.
  - Symmetric Multiprocessor: e.g. multi-core CPU.
  - Non-uniform Memory Access.
- Loosely coupled: CPUs are connected via high speed network connections

Symmetric Multiprocessor (SMP)

- Two or more similar processors of comparable capability
- These processors share the same main memory and I/O facilities
- All processors share access to I/O devices
- All processors can perform the same functions (*symmetric*)
- The system is controlled by an integrated operating system
  - Provides interaction between processors and system resources

Symmetric Multiprocessor Organization



SMP Design Considerations (1)

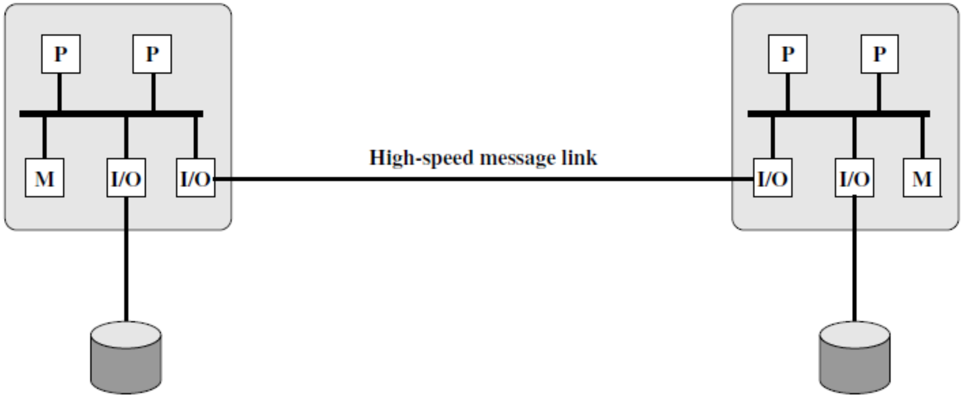
- Hardware
  - ***Cache coherence***: Each processor has a separate cache. A single data on main memory can be mapped to multiple cache on different CPUs.
- Software (OS):
  - Simultaneous concurrent processes.
  - Multi-processer scheduling.
  - Synchronization.
  - Memory management
  - Reliability and fault tolerance

Cluster

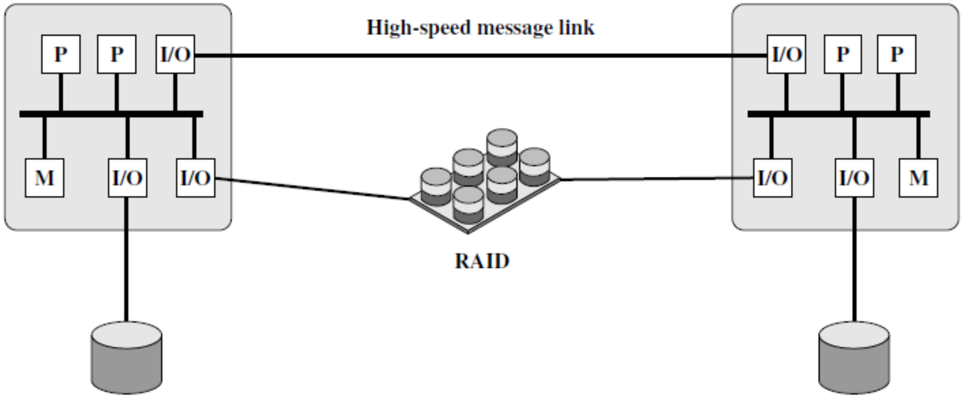
- Group of interconnected, whole computers working together as a unified computing resource. Each computer in a cluster is typically referred to as a node.

- Absolute scalability
- Incremental scalability
- High availability
- Superior price/performance

Cluster Configurations

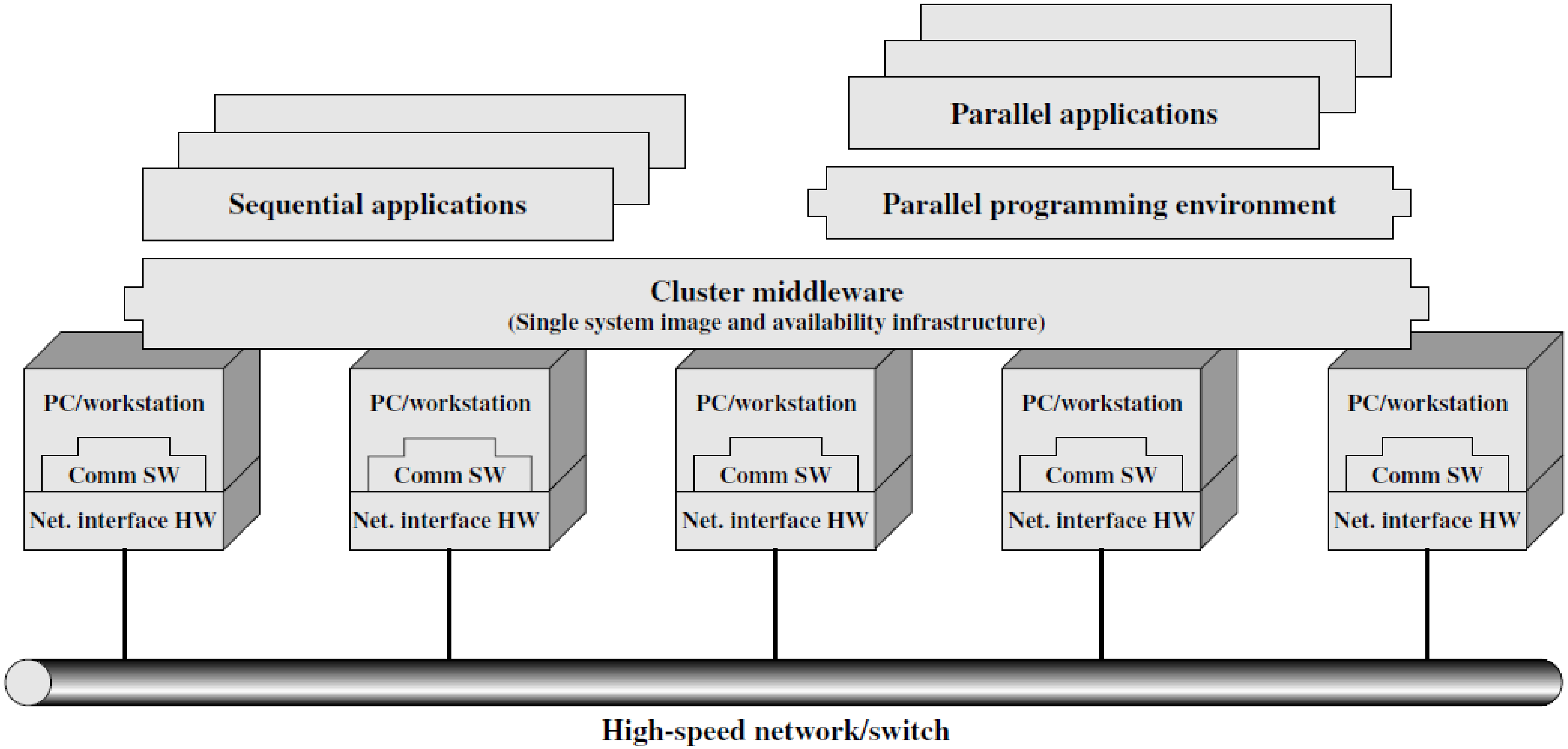


(a) Standby server with no shared disk

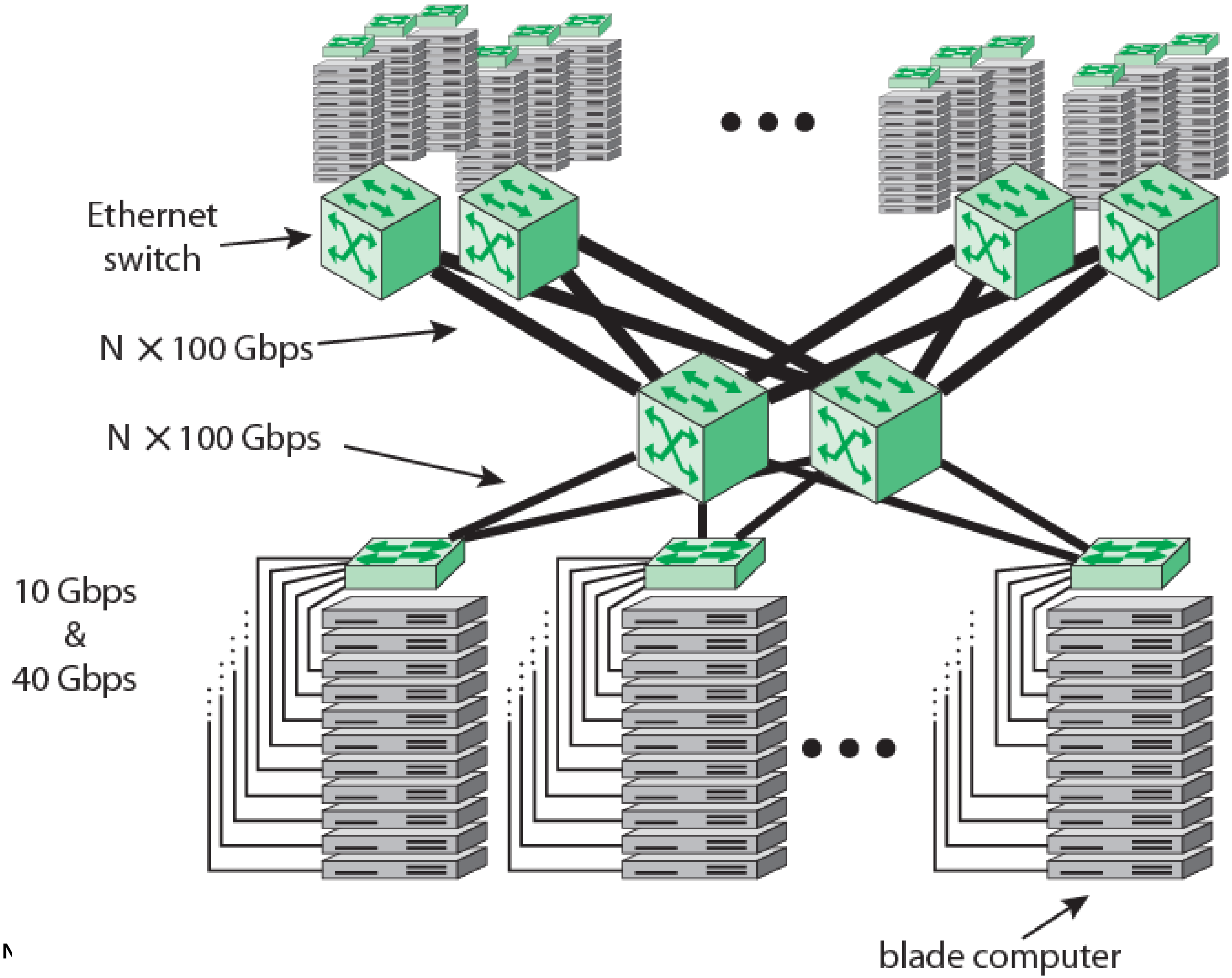


(b) Shared Disk

Cluster Computer Architecture



Example: blade server



SMP vs cluster

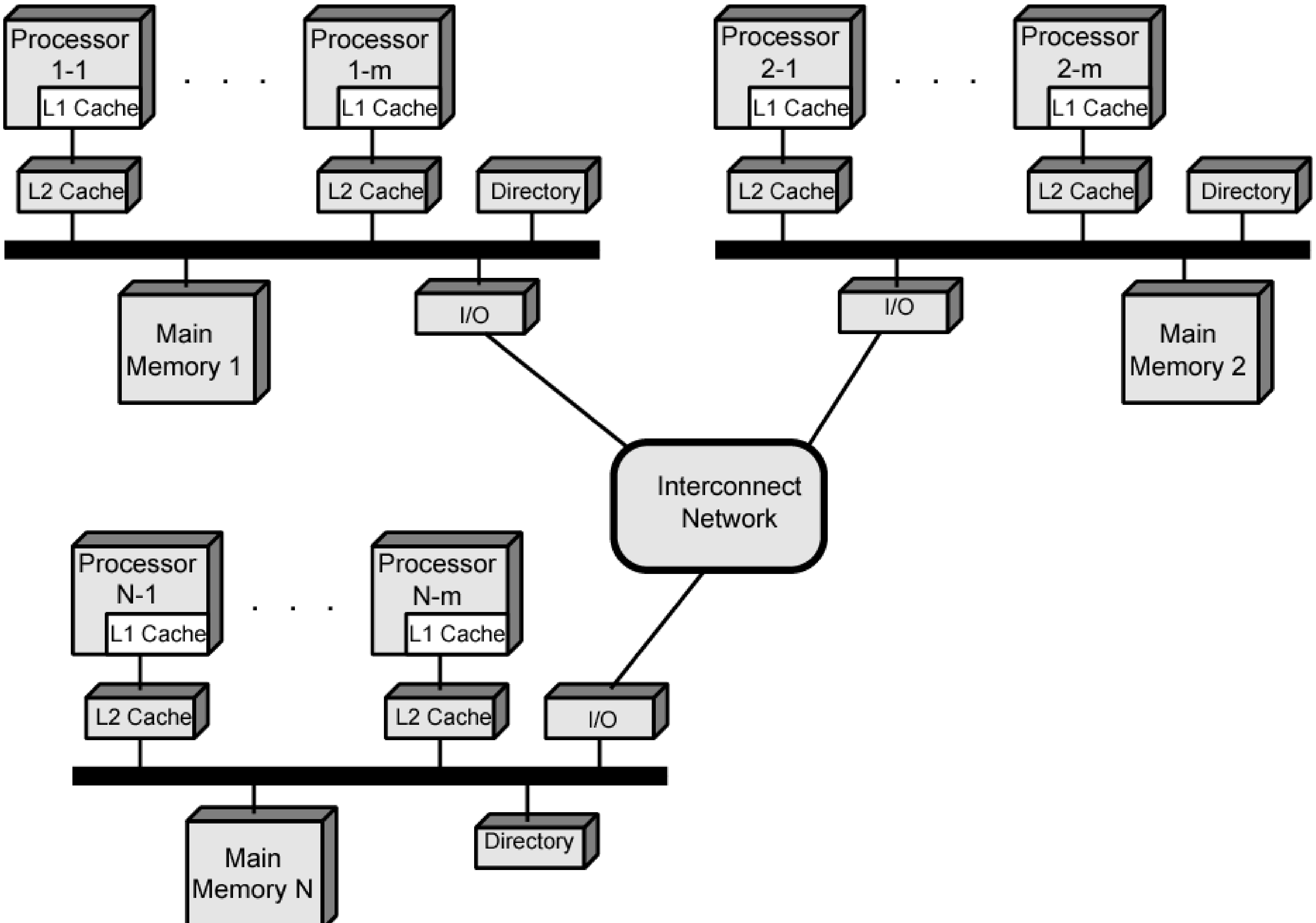
- Both are high performance computer architecture
- SMP
  - Easy to use and maintenance
  - Closer to uniprocessor system
  - Small size and low power consumption
- Cluster
  - High computing capability
  - Scalability
  - Hight dependability and availability

SMP vs cluster

- SMP
  - Limited capability.
- Cluster
  - Separated memory on each node
  - Complicated software

➔ combining advantages of SMP and cluster: NUMA

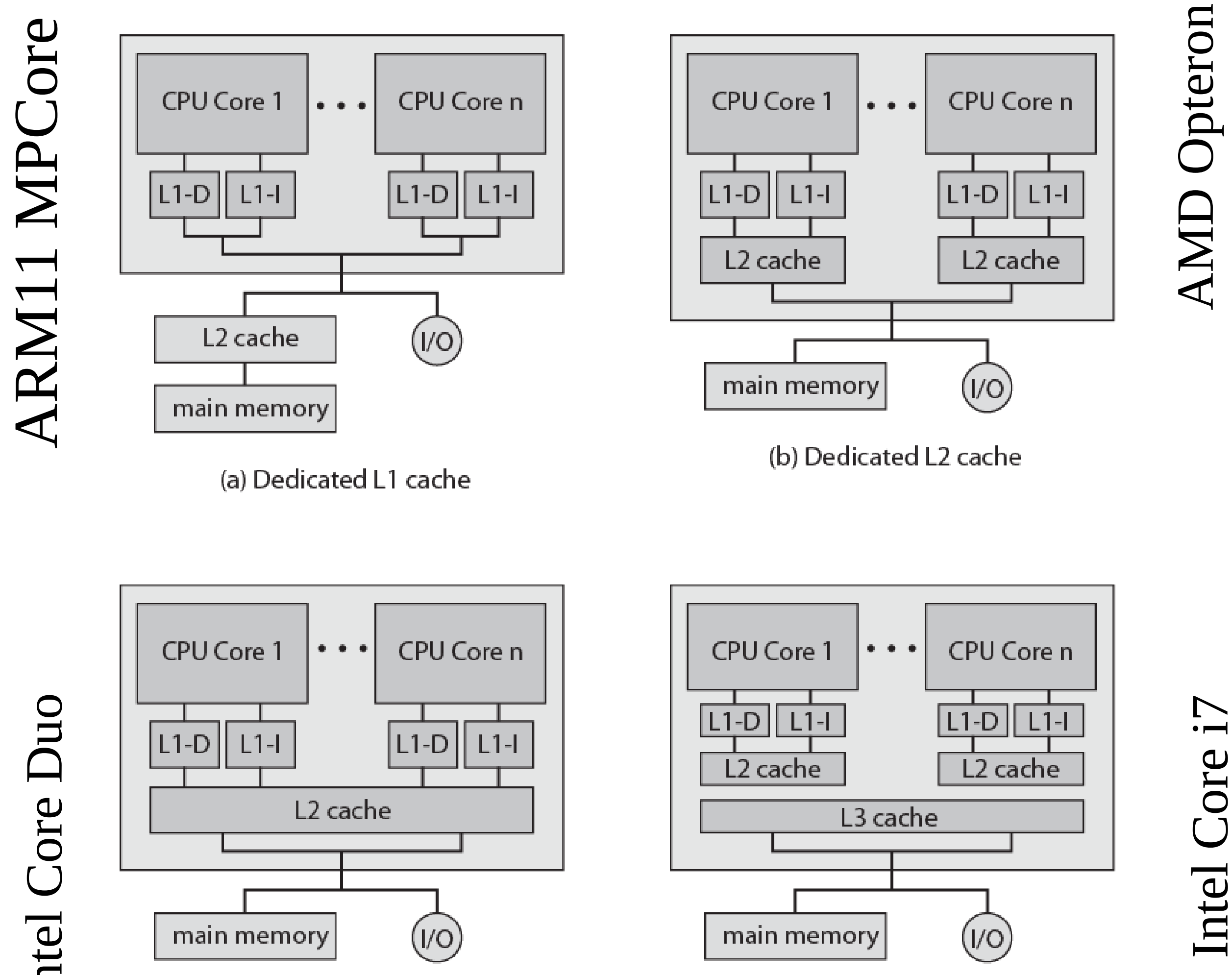
Cache-coherence NUMA



Multicore Processors

- Conventional performance improvement
  - Pipelining
  - Superscalar
  - Multithreading
  - ➔ increasingly difficult engineering challenge in CPU design
- Pollack:
  - “Performance is roughly proportional to square root of increase in complexity”
  - double the logic in a processor core, then it delivers only 40% more performance
- The use of multiple cores has the potential to provide near-linear performance improvement with the increase in the number of cores

Multicore Organization Alternatives



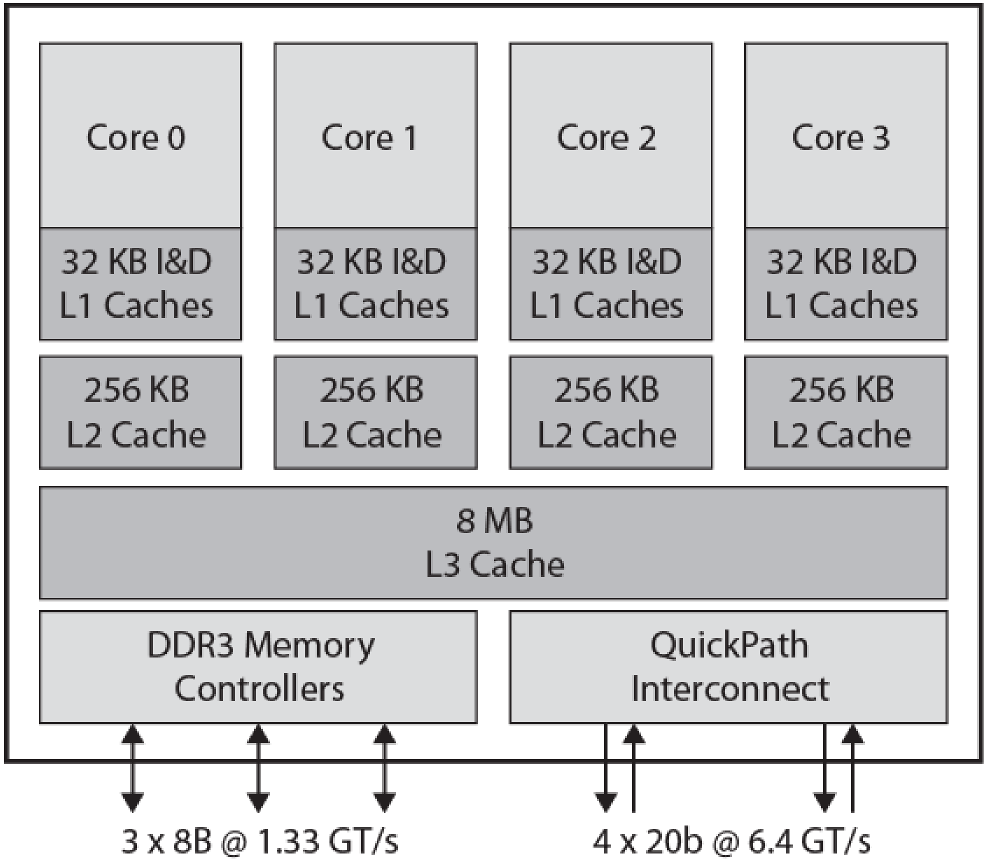
(a) Dedicated L1 cache

(b) Dedicated L2 cache

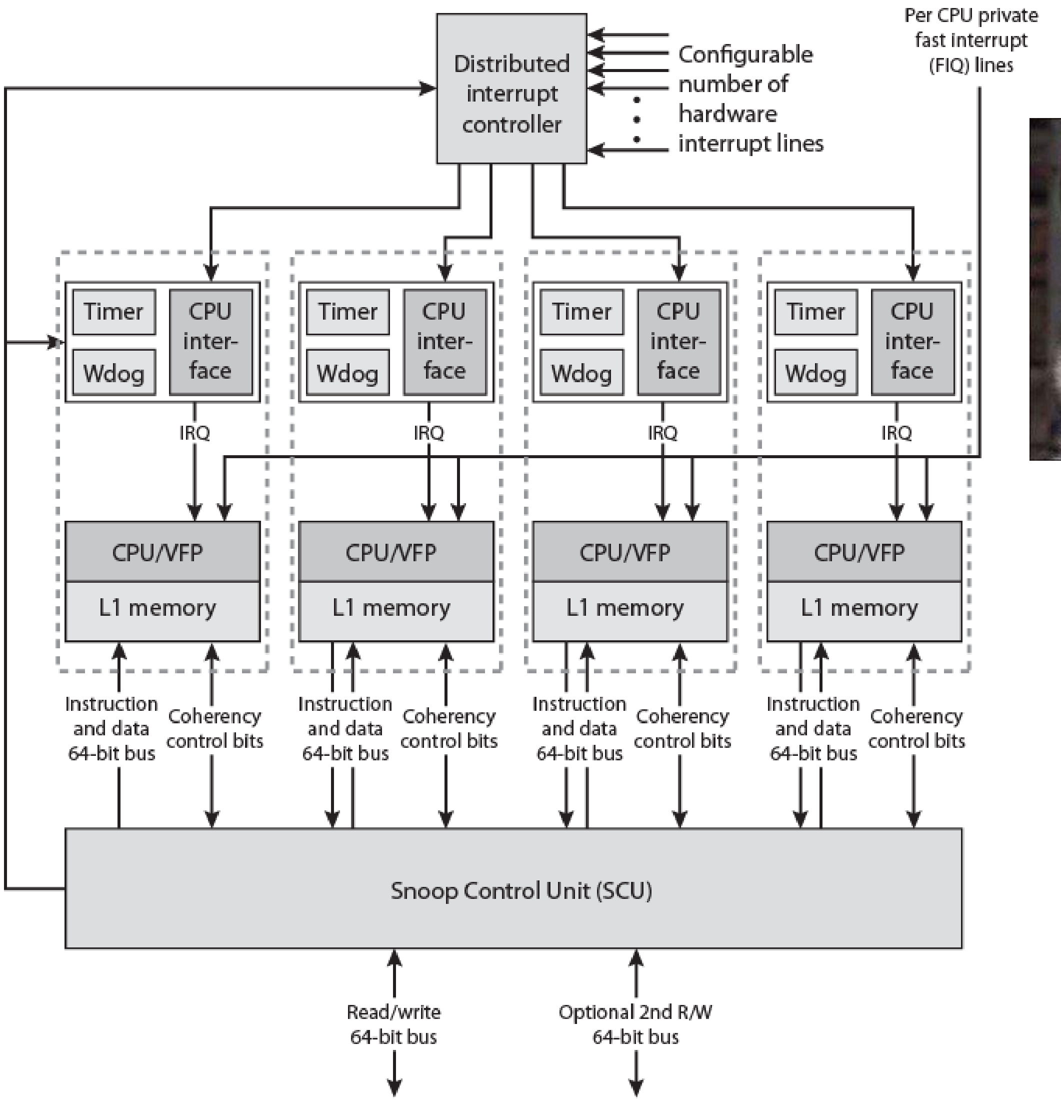
(c) Shared L2 cache

(d) Shared L3 cache

Intel Core i7

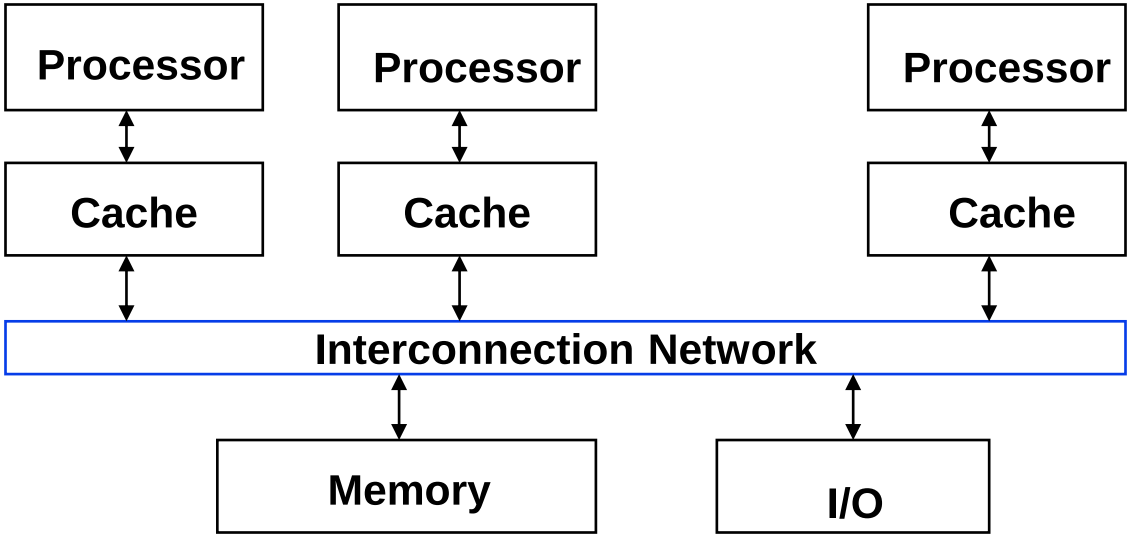


ARM11 MPCore



The Big Picture: Where are We Now?

- Multiprocessor – a computer system with at least two processors



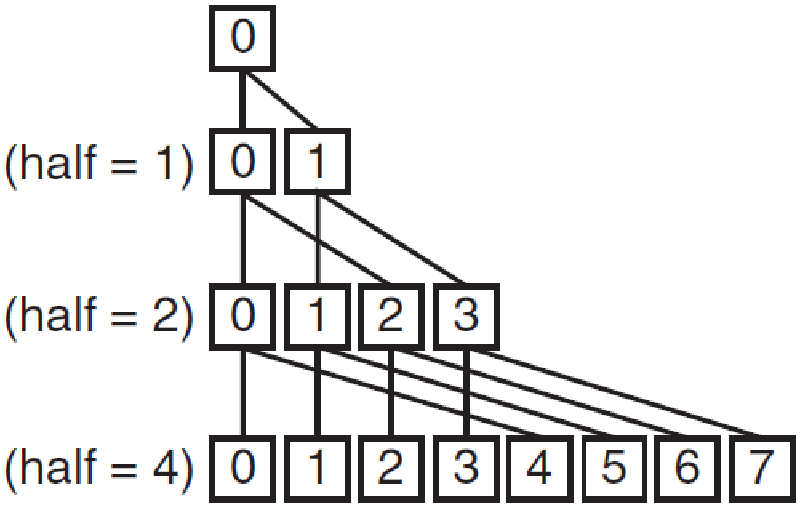
- Can deliver high throughput for independent jobs via job-level parallelism or process-level parallelism
- And improve the run time of a *single* program that has been specially crafted to run on a multiprocessor - a parallel processing program

Reduction process

- Adding 64000 numbers in a 64 processors SMP

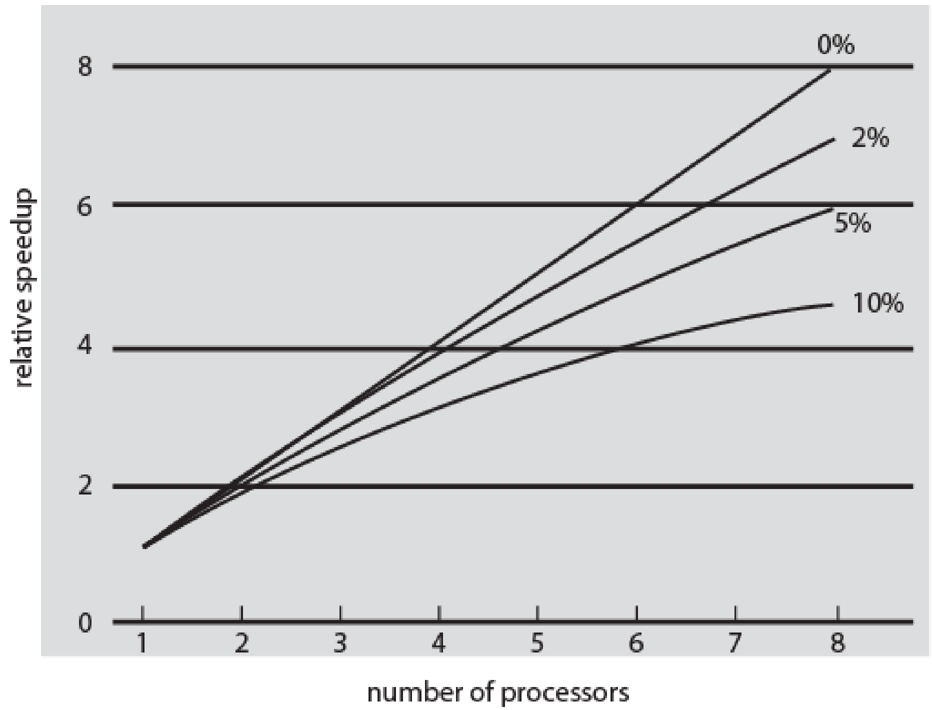
```
sum[Pn] = 0;
for (i = 1000*Pn; i < 1000*(Pn+1); i += 1)
    sum[Pn] += A[i]; /*sum the assigned areas*/
```

- Finally, the total sum is calculated across all processors
  - 32 adds
  - 16 adds
  - …



How many core is enough?

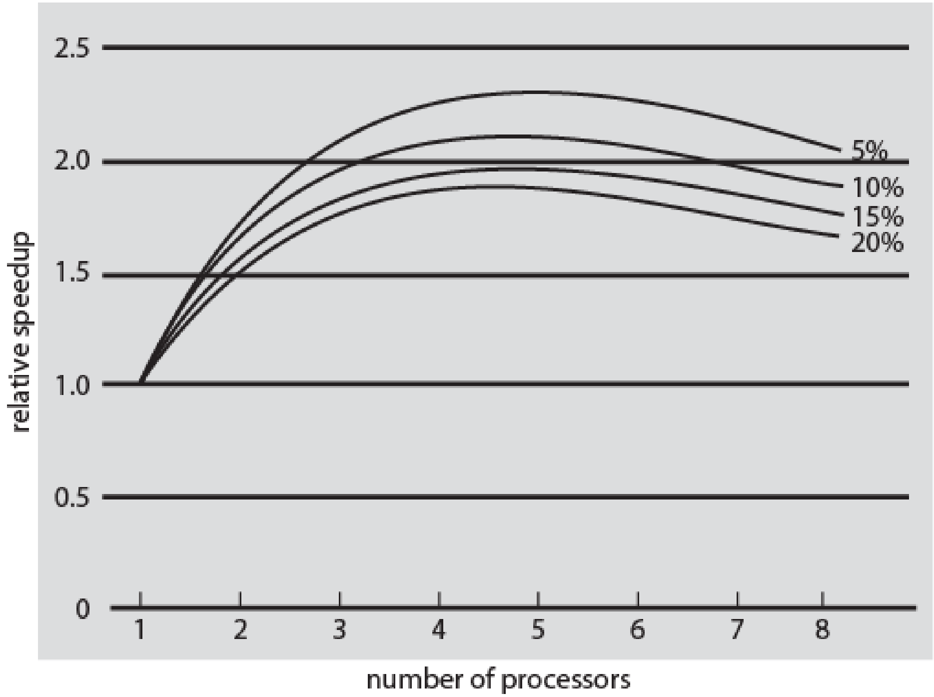
- The more core the higher performance?
  - Not really, it depends on the sequential portion of code
- Amdahl's law



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions

How many core is enough?

- Overhead

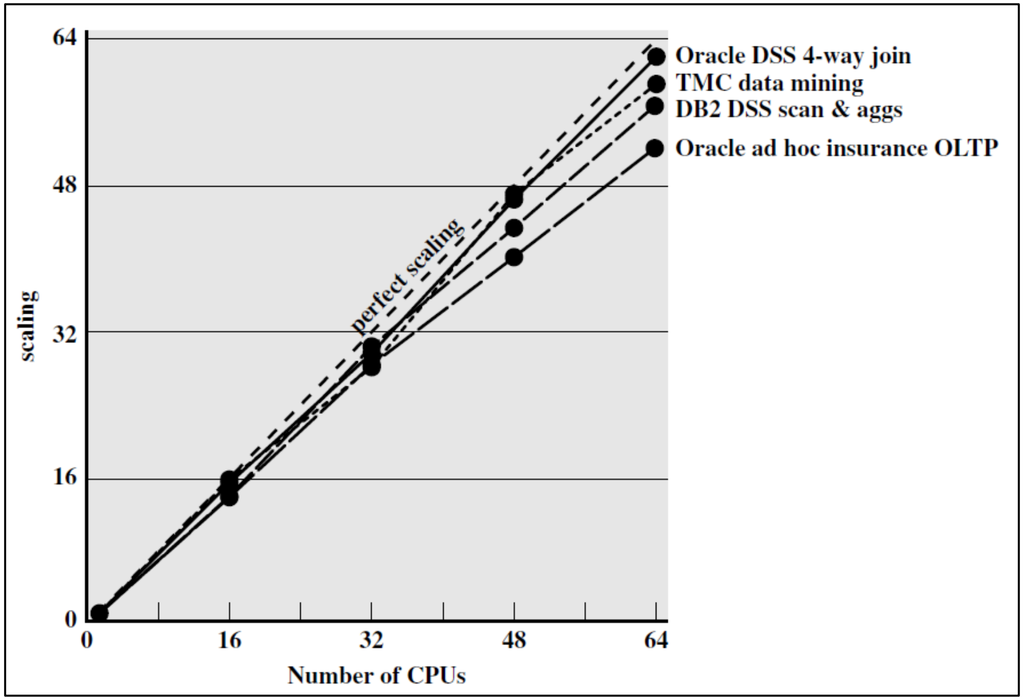


(b) Speedup with overheads

- How many core is best suited for end-user PC?

When a large number of cores is necessary?

- Database server. Ex: SELECT * FROM …
- Multithreaded native applications
- Multiprocess applications
- Java applications



Scaling of Database Workloads on Multiple-Processor Hardware

The end