Parallel Architectures

Frédéric Desprez

INRIA

Some references

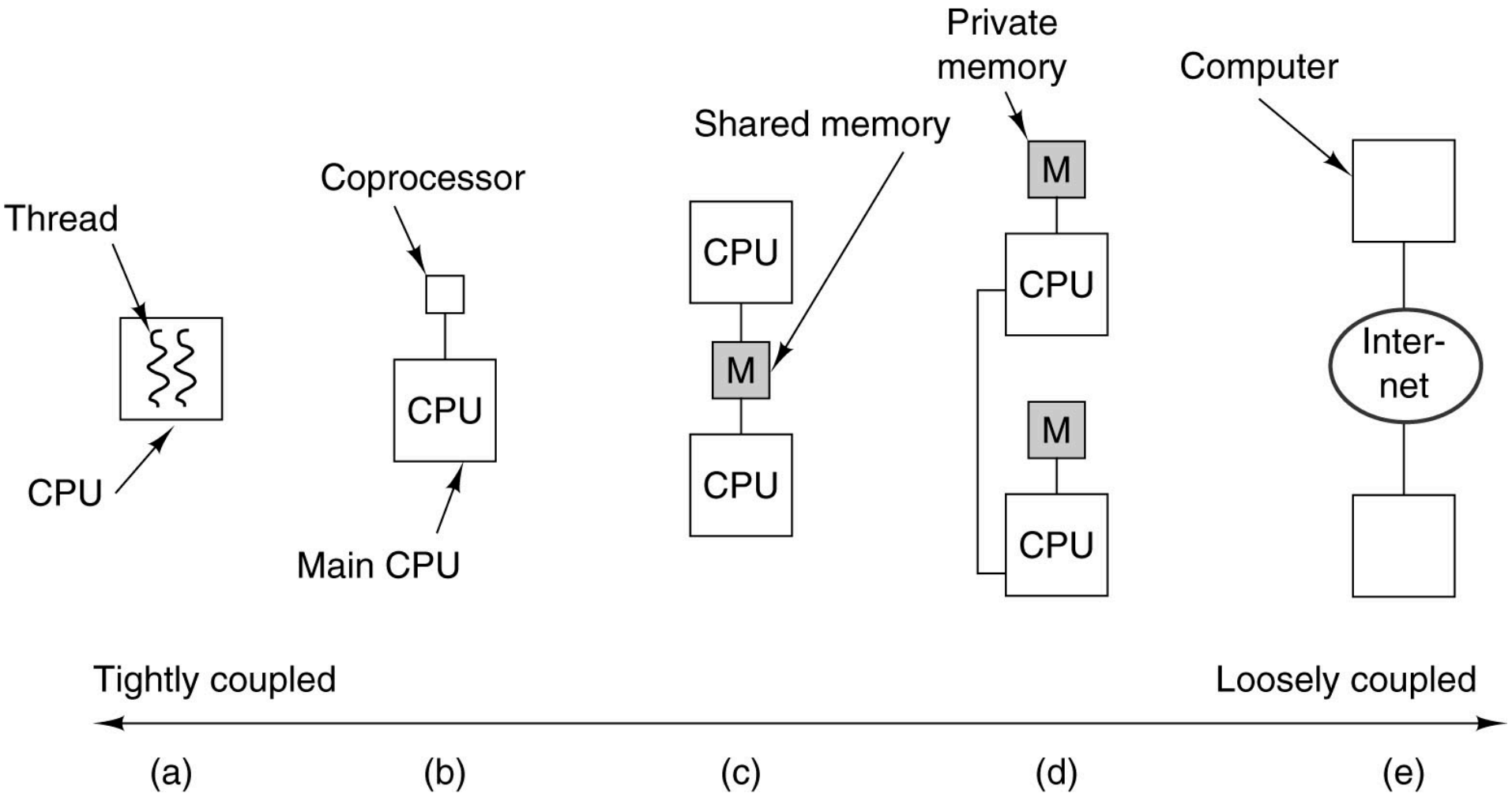
- Lecture "**Calcul hautes performance – architectures et modèles de programmation**", Françoise Roch, Observatoire des Sciences de l'Univers de Grenoble Mesocentre CIMENT
- **4 visions about HPC - A chat**, X. Vigouroux, Bull
- **Parallel Programming – For Multicore and Cluster System,** T. Rauber, G. Rünger

Lecture summary

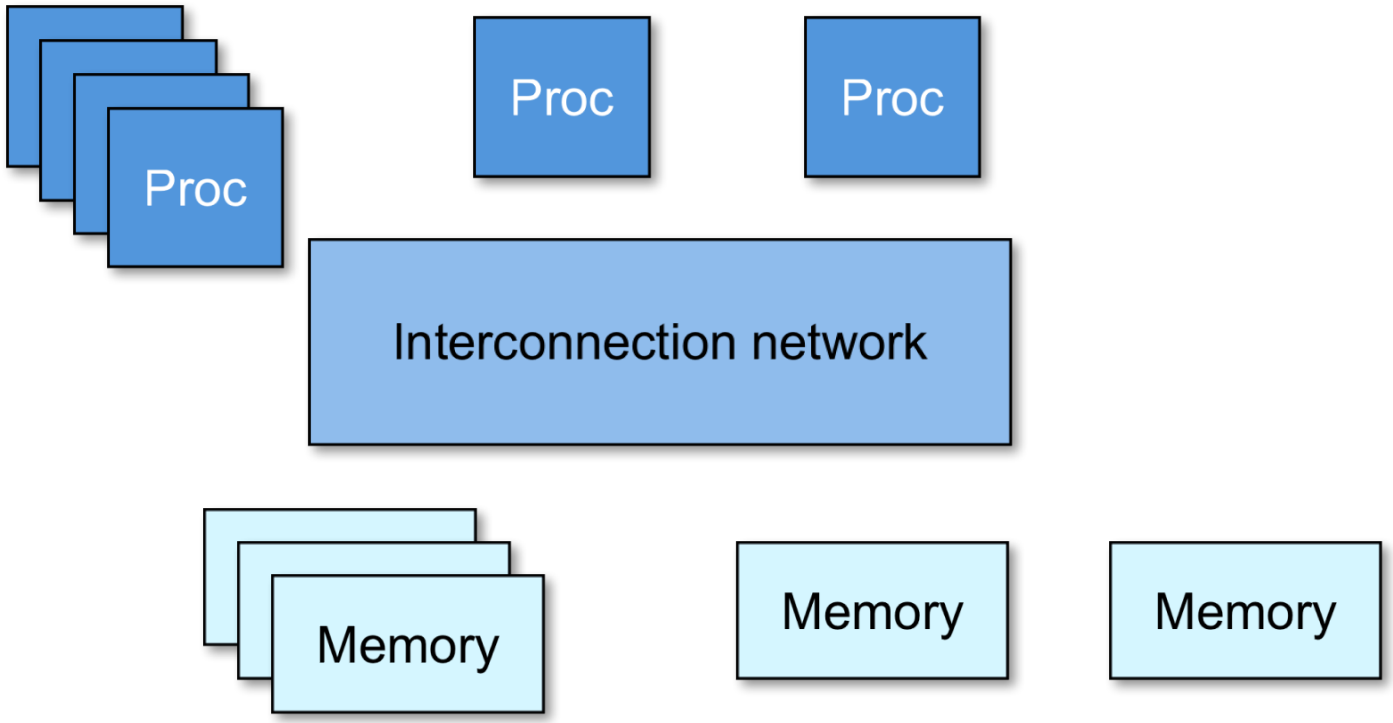
- Introduction
- Models of parallel machines
- Multicores/GPU
- Interconnection networks

MODELS OF PARALLEL MACHINES

Parallel architectures



A generic parallel machine

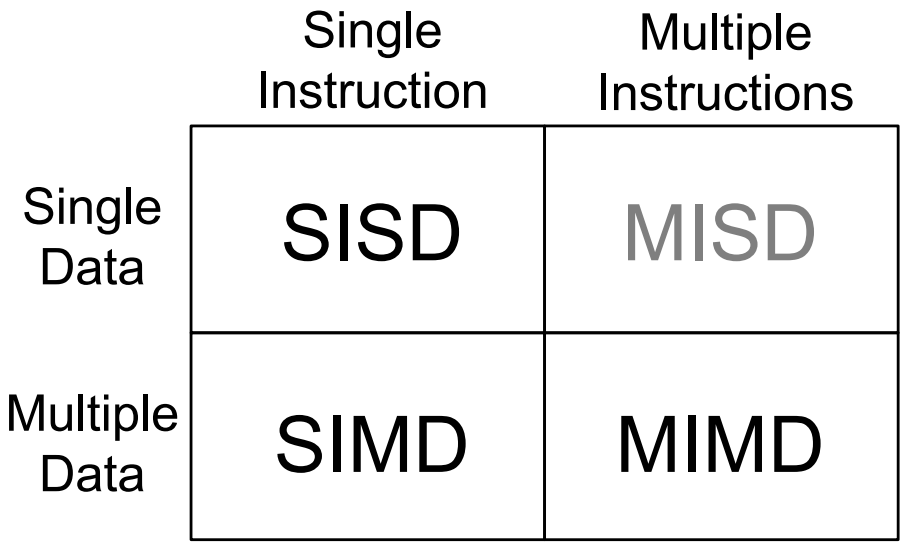


- Where is the memory?
- Is it connected directly to the processors?
- What is the processor connectivity?

Parallel machines models

Flynn's classification

- Characterizes machines according to their flow of data and instructions

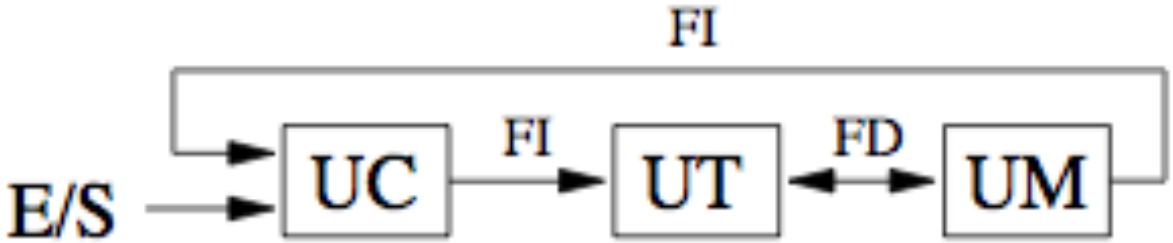


Flynn, M., "**Some Computer Organizations and Their Effectiveness**". IEEE Trans. Comput. C-21: 948., 1972.

SISD: Single Instruction, Single Data stream

"Classical" sequential machines

Each operation is performed on one data at a time



UC = Control Unit (responsible for the sequencing of instructions)

UT = Processing Unit (performs the operations)

FI = Instructions Flow

UM = Memory Unit (contains instructions and data)

FD = Data Flow

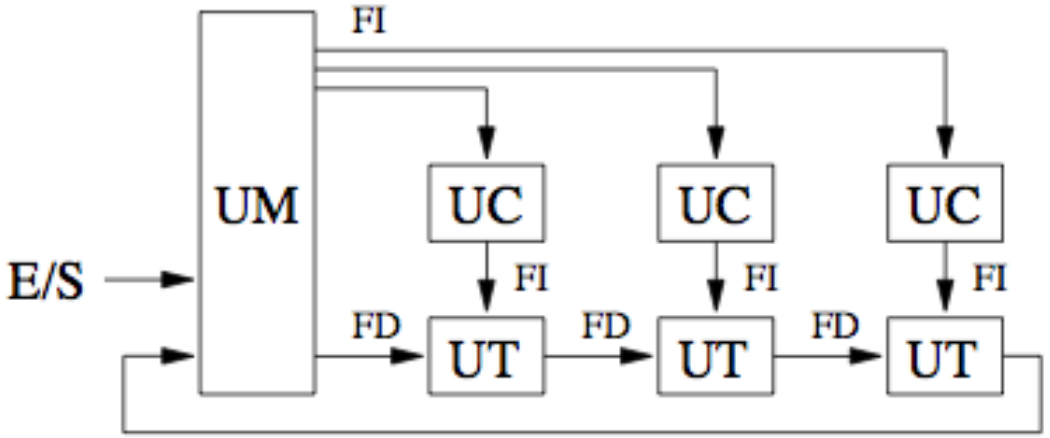
Von Neuman's model (1945)

MISD: Multiple Instruction stream, Single Data stream

Specialized "systolic" type machines

Processors arranged with a fixed topology

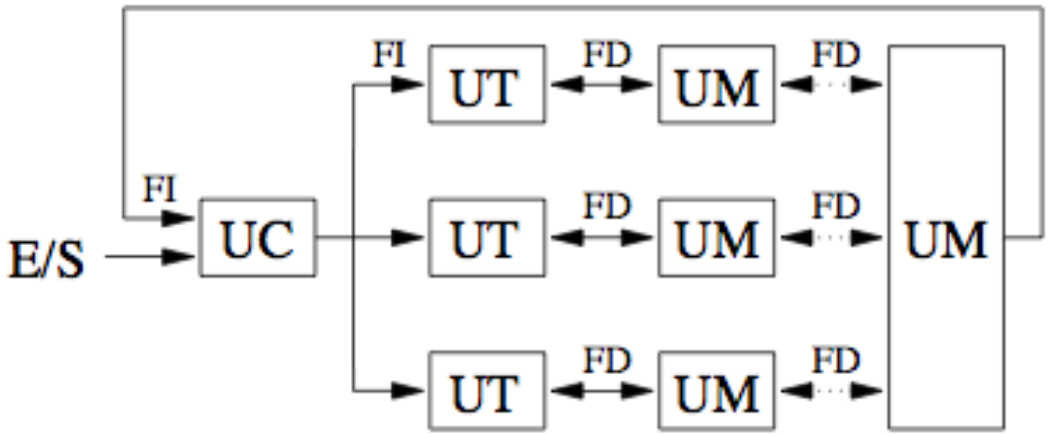
Strong synchronization



SIMD: Single Instruction stream, Multiple Data stream

Totally **synchronized** calculation units

Conditional execution with masking flag



• Machines adapted to very regular processing (matrix operations, FFT, image processing)

• Not adapted at all to irregular operations

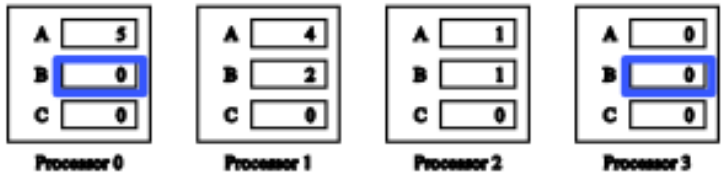
Conditionals in SIMD

- **Masking flag**
  - Used to prevent processors from performing some operations

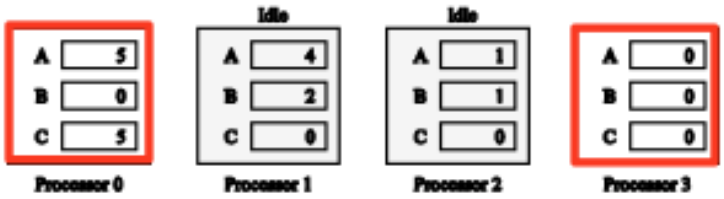
conditional statement

```
if (B == 0)
then C = A
else C = A/B
```

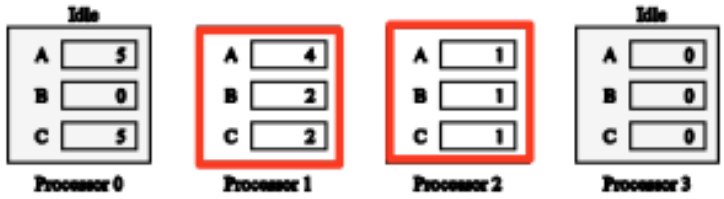
initial values



execute "then" branch



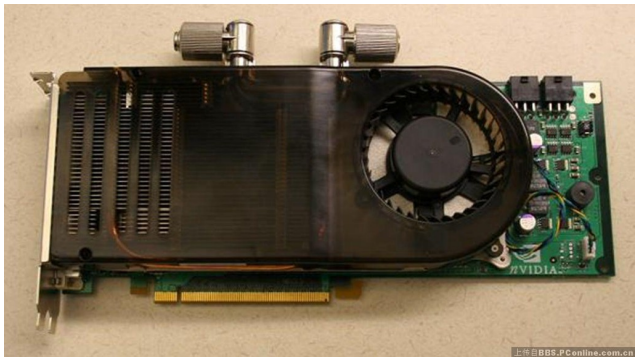
execute "else" branch



Some examples of SIMD machines

- **80's/90's parallel machines**
  - Illiac IV, MPP, DAC, Connection Machine CM-1/2, MasPar MP-1/2

- **A great return today**
  - Intel processors and SSE / SSE-2 mode (vector units)
  - 128-bit vector registers
    - 16 floats (8 bits), 8 short integers (16 bits), 4 integers (32 bits)
    - 2 floats (64 bits) for SSE-2
  - Altivec (Velocity Engine, VMX)
  - Co-processors
    - GPGPU nVidia G80
    - ClearSpeed array processor (2 control processors + 192 processors)



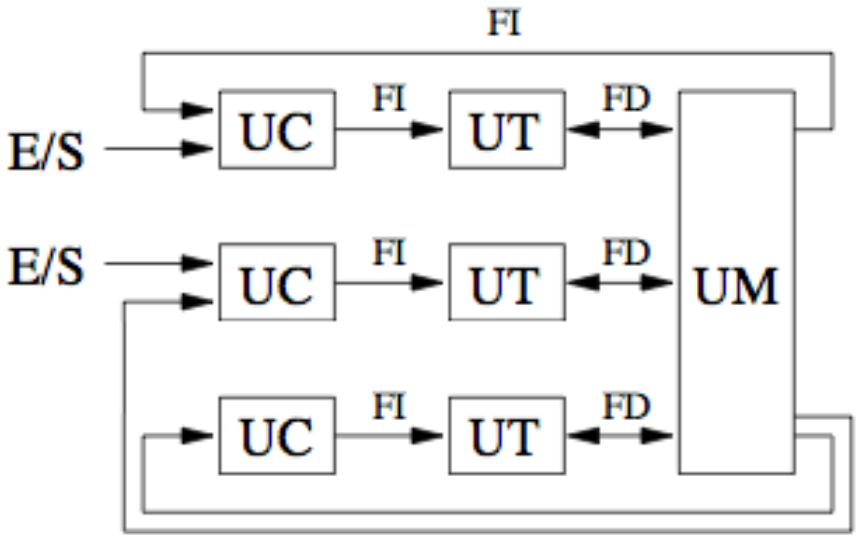
MIMD: Multiple Instructions stream, multiple data stream

Multi-Processor Machines

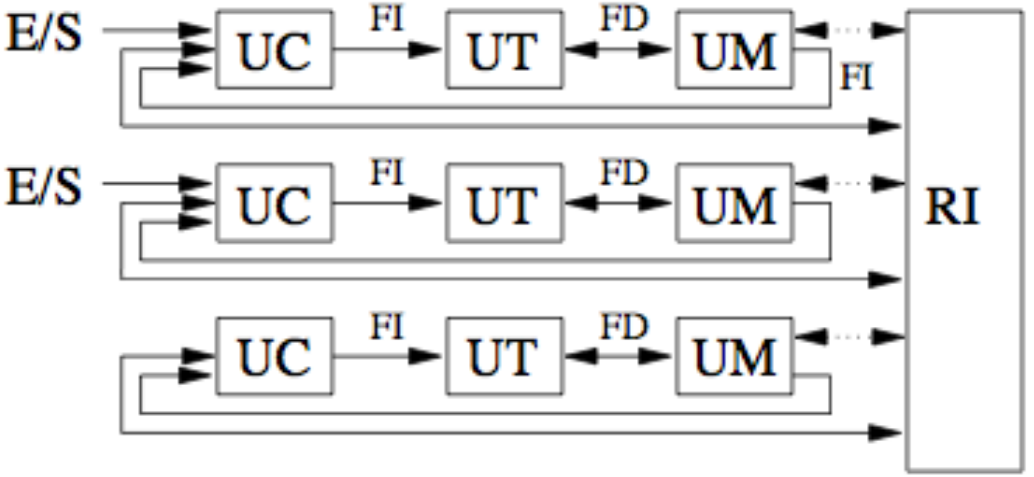
Each processor runs its own code asynchronously and independently

Two sub-classes

Shared memory



Distributed memory

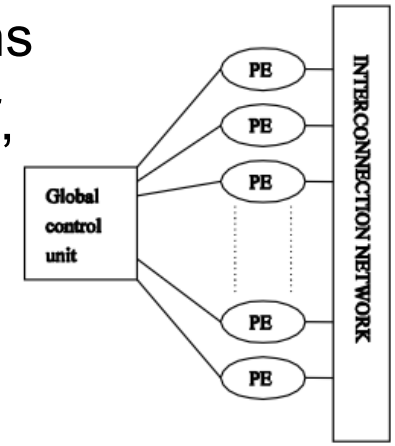


A mix between SIMD and MIMD: SPMD (**Single Program, Multiple Data**)

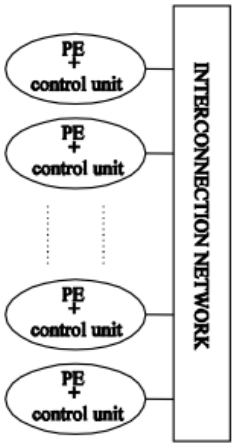
SIMD vs MIMD

- **SIMD Platforms**
  - Designed for specific applications
  - Complicated (and long) design, no "on-shelf" processors
  - Less equipment (one control unit)
  - Need less memory for instructions (single program)
  - Used heavily for current co-processors

- **MIMD Platforms**
  - Works for a wide variety of applications
  - Less expensive (components on shelf, short design)
  - Need more memory (OS and program on each processor)



SIMD architecture



MIMD architecture

Raina's classification

Taking into account the address space

- **SASM** (Single Address space, Shared Memory)
  Shared memory

- **DADM** (Distributed Address space, Distributed Memory)
  Distributed memory, without access to remote data. The exchange of data between processors is necessarily effected by passing messages, by means of a communication network

- **SADM** (Single Address space, Distributed Memory)
  Distributed memory, with global address space, possibly allowing access to data located on other processors

Raina's classification, contd.

The type of memory access implemented

NORMA (No Remote Memory Access)

No means of access to remote data, requiring the message passing

UMA (Uniform Memory Access)

Symmetric access to memory, identical cost for all processors

NUMA (Non-Uniform Memory Access)

The access performance depends on the location of the data

CC-NUMA (Cache-Coherent NUMA)

Type of NUMA architecture integrating caches

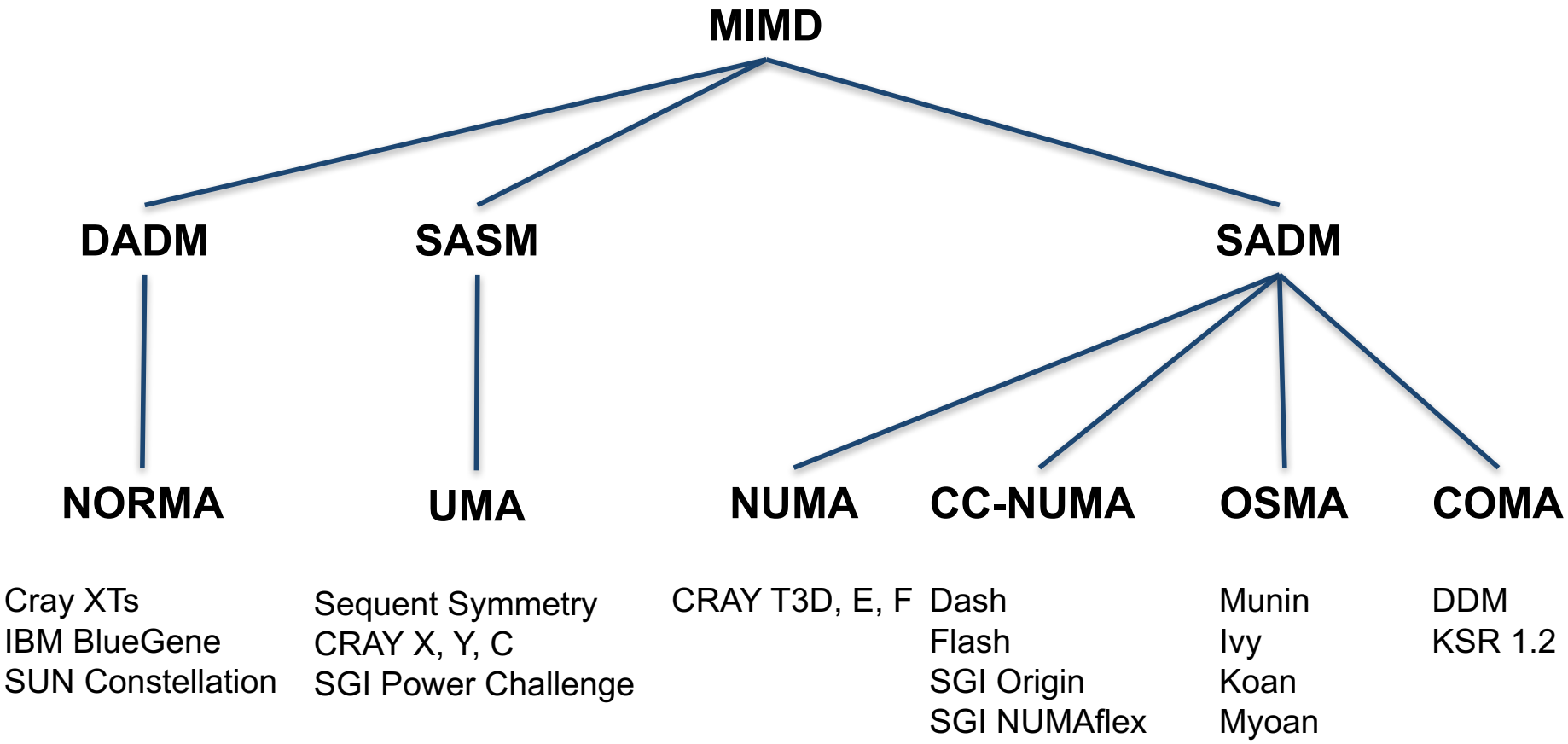
OSMA (Operating System Memory Access)

The remote data accesses are managed by the operating system, which handles page faults at the software level and handles remote copy/send requests

COMA (Cache Only Memory Access)

The local memories behave like caches, so that a data item has neither a proprietary processor nor a determined location in memory

Raina's classification, contd.



Parallel Programming Models

The **programming model** consists of the languages and libraries that will allow to have an **abstraction** of the machine

Control

- How is parallelism created (implicit or explicit)?
- What are the sequences between operations (synchronous or asynchronous)?

Data

- What are the private and shared data?
- How are these data accessed and / or communicated?

Synchronization

- What operations can be used to coordinate parallelism?
- What are atomic (indivisible) operations?

Cost

- How can we calculate the cost of each previous item?

A simple example: the sum

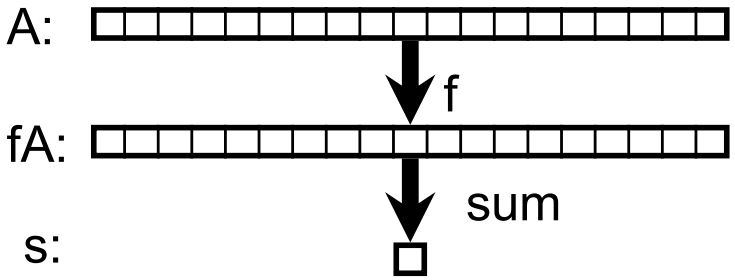
A function f is applied to the elements of an array A and the sum

$$\sum_{i=0}^{n-1} f(A[i])$$

Questions

- Where is A? In a central memory? Distributed?
- What will be the work done by the processors?
- How will they coordinate themselves to achieve a single outcome?

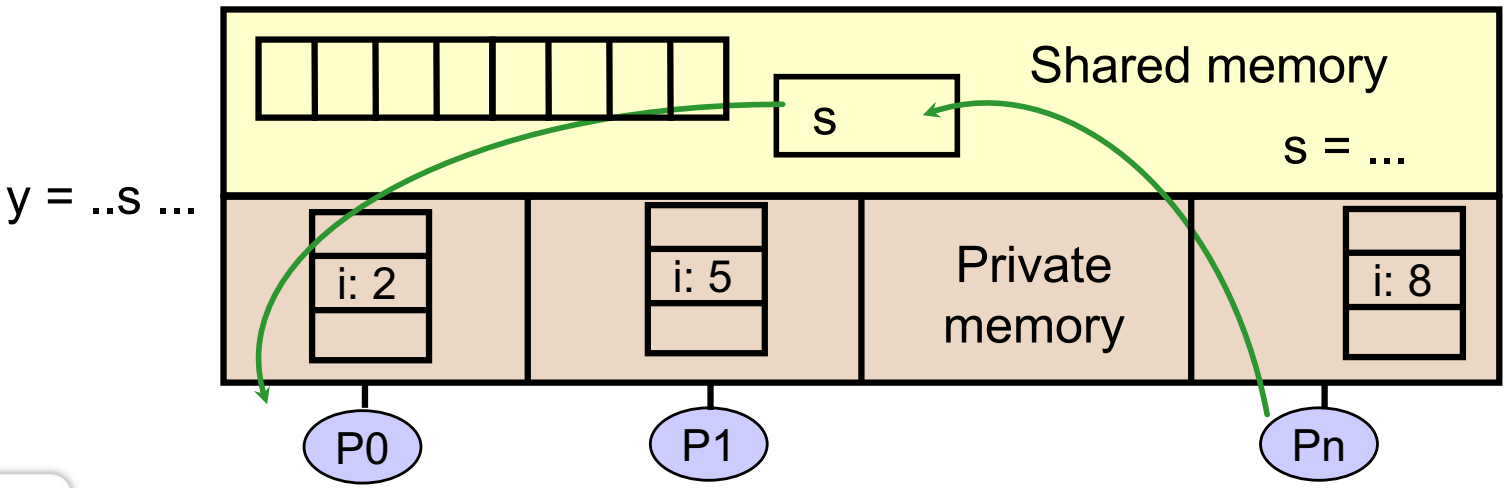
A = data array
fA = f(A)
s = sum(fA)



Shared memory

The program is a set of control threads

- They can sometimes be created dynamically during execution in some languages
- Each thread has its own private data set (local stack variables)
- Set of shared variables (static variables, shared blocks, global stack)
- Threads communicate by writing and reading shared variables
- They synchronize on shared variables



Parallelization strategy

$$\sum_{i=0}^{n-1} f(A[i])$$

Shared Memory strategy

- Small number of processors (p << n = size(A))
- Connected to a single central memory

Parallel decomposition

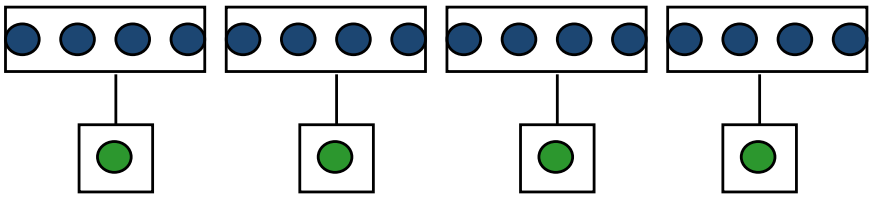
- Each evaluation and each partial sum is a task

Assign n / p numbers to each processor p

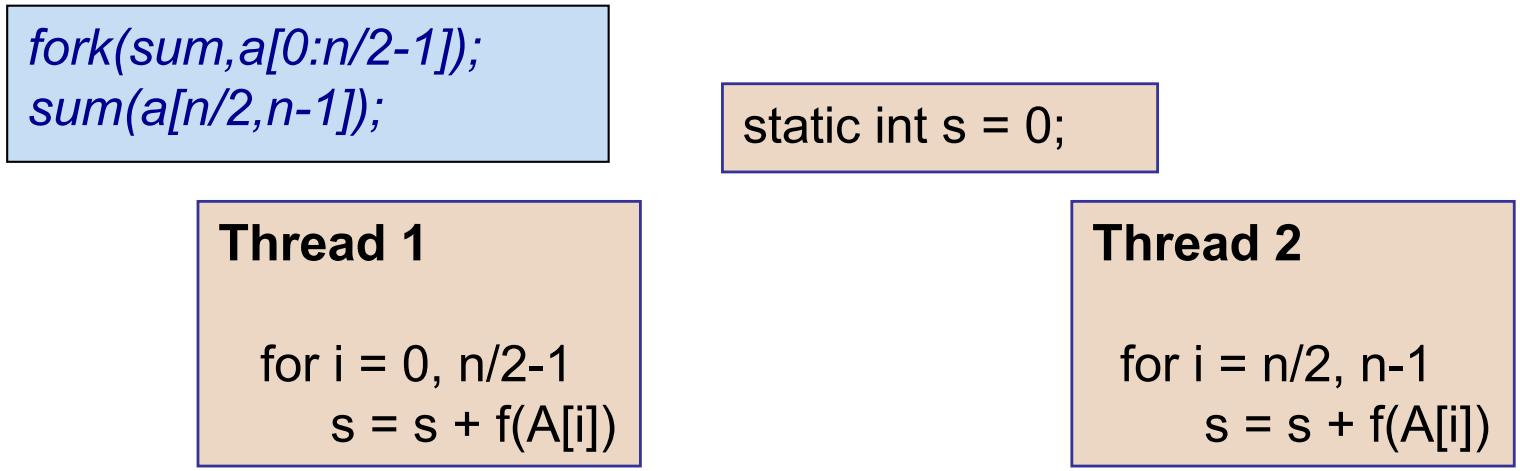
- Each of them calculates private results and a partial sum
- Gather the p local sums and calculate the total sum

Two classes of data

- Shared (logically)
  - The n numbers, the global sum
- Private (logically)
  - Local evaluations of functions



Shared memory "code" for the computation of the sum



- **What is the problem with this program?**
- **A race condition occurs when**
  - Two processors (or two threads) access the same variable (and at least one of them performs a write)
  - The accesses are competing (not synchronized) and they can appear at the same time

Shared memory "code" for the computation of the sum, contd.

A= | 3 | 5 |

$f(x) = x^2$

```
static int s = 0;
```

Thread 1		Thread 2	
….		…	
compute f([A[i]) and put in reg0	9	compute f([A[i]) and put in reg0	25
reg1 = s	0	reg1 = s	0
reg1 = reg1 + reg0	9	reg1 = reg1 + reg0	25
s = reg1	9	s = reg1	25
…		…	

- Suppose that A = [3,5], $f(x) = x^2$ and s=0 at the start
- For the result to be correct we need to have s = $3^2 + 5^2 = 34$ at the end
  - But here it can be 34, 9, or 25
- Atomic operations are read and write
  - We will not see a mixture of numbers but the operation + = is not atomic
  - All computations take place in private registers

Improved code for the sum

```
static int s = 0;
static lock lk;
```

Thread 1

```
local_s1= 0
for i = 0, n/2-1
    local_s1 = local_s1 + f(A[i])
lock(lk);
s = s + local_s1
unlock(lk);
```

Thread 2

```
local_s2 = 0
for i = n/2, n-1
    local_s2= local_s2 + f(A[i])
lock(lk);
s = s +local_s2
unlock(lk);
```

- Since the addition is associative, one can change the order
- Most computations take place on private variables
  - The frequency of sharing is also reduced, which can improve the speed
  - But there is always competition for updating s
  - It can be deleted with locks (only one thread can have a lock at one time, the other waits)

Shared memory machine model

Processors are connected to a large shared memory

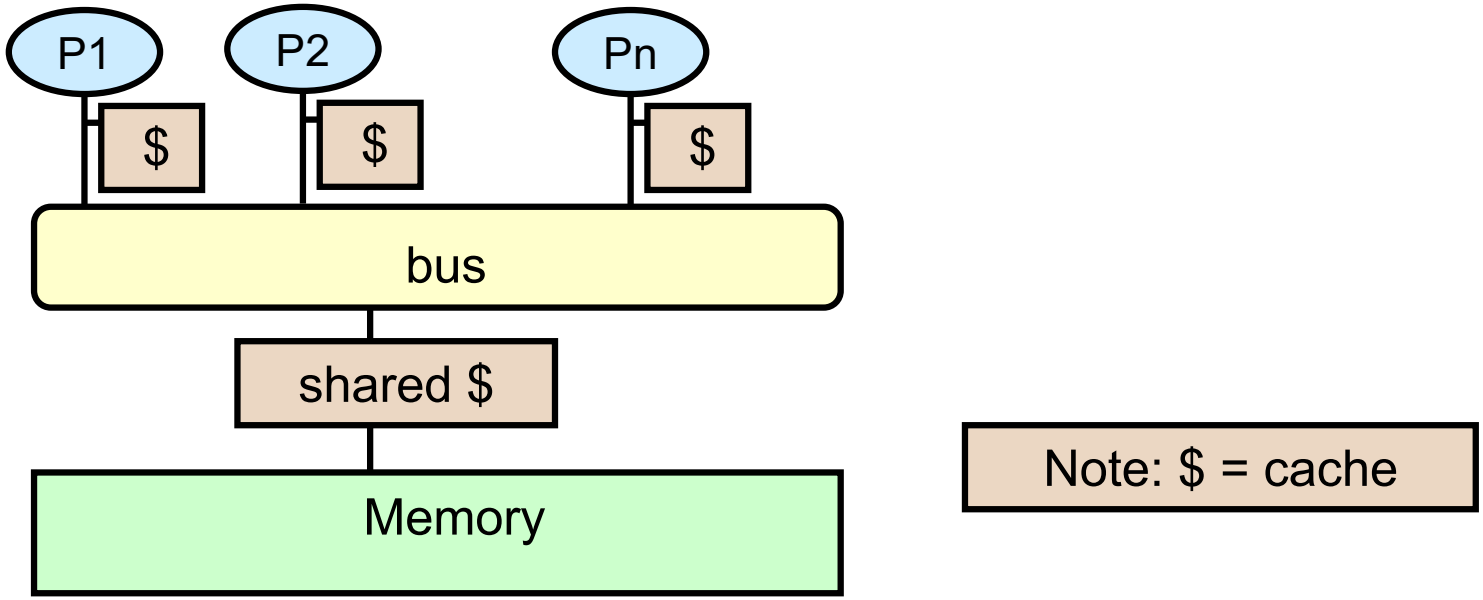
- Also known as *Symmetric Multiprocessors* (SMPs)
- SGI, Sun, HP, Intel, SMPs IBM
- Multicore processors (except that caches are shared)

Scalability issues for large numbers of processors

- Usually <= 32 processors

Advantage: Uniform memory access (*Uniform Memory Access*, UMA)

Access code: lower cost for caches compared to the main memory



Extensibility Issues for Shared Memory Architectures

Why not put more processors (with larger memory)?

- Memory bus becomes a bottleneck
- Caches must remain consistent

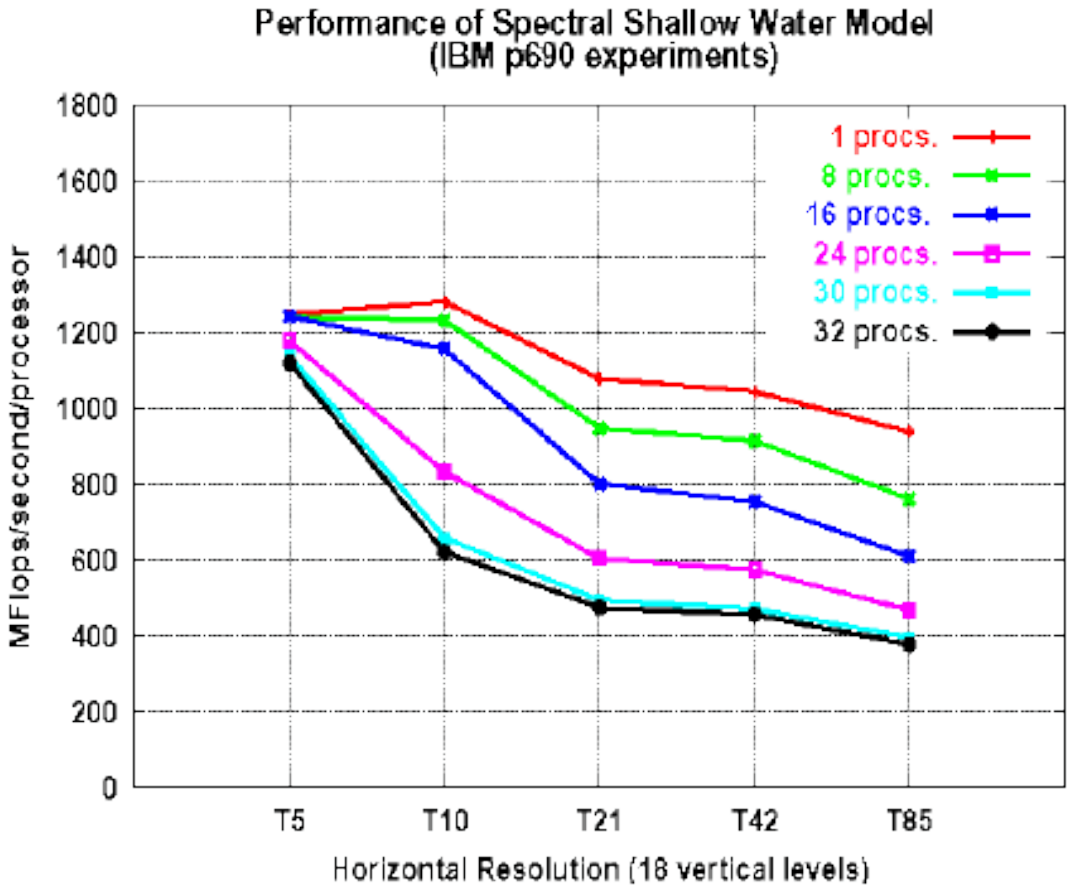
Example: Parallel Spectral Transform Shallow Water Model (PSTSWM)

- Experimental results of Pat Worley (ORNL)
- Important core of atmospheric models
  - 99% of the floating operations are additions or multiplications
  - But the code uses data on all the memory with low re-use of the loaded data (bus use and frequent shared memory)
- Experiments with sequential performance (a copy of the code running independently by increasing the number of processors used)
  - Normally the best case for shared memory: no sharing
  - But the data do not all fit in the registers and caches

Scalability Issues for Shared Memory Architectures, contd.

- Performance degradation is a function of the number of processors involved
- No data sharing between codes so perfect parallelism

- Code executed for 18 vertical levels with several horizontal sizes



Process scaling on IBM p690

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

UT-BATTELLE

28

Crédits: Pat Worley, ORNL

Distributed Shared Memory

Memory is logically shared but physically distributed

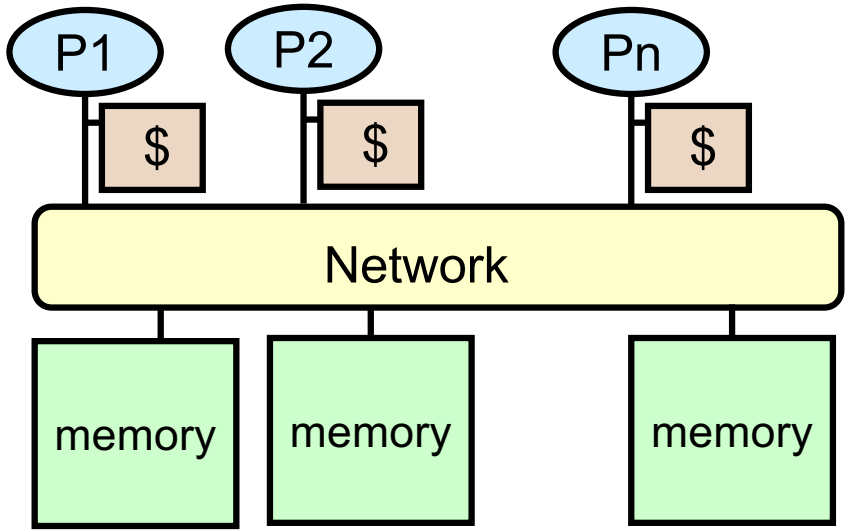
- Any processor can access any address in memory
- The lines (or pages) of cache lines are exchanged in the machine

Example: SGI platforms

- Scalable to 512 nodes (SGI Altix (Columbia) @ NASA / Ames)

Problem

- Cache Coherence Protocols
- How to maintain consistency between copies of the same memory area



The cache lines (or pages) must be large enough to cushion the overhead

➔ Locality of data critical for performance

NUMA

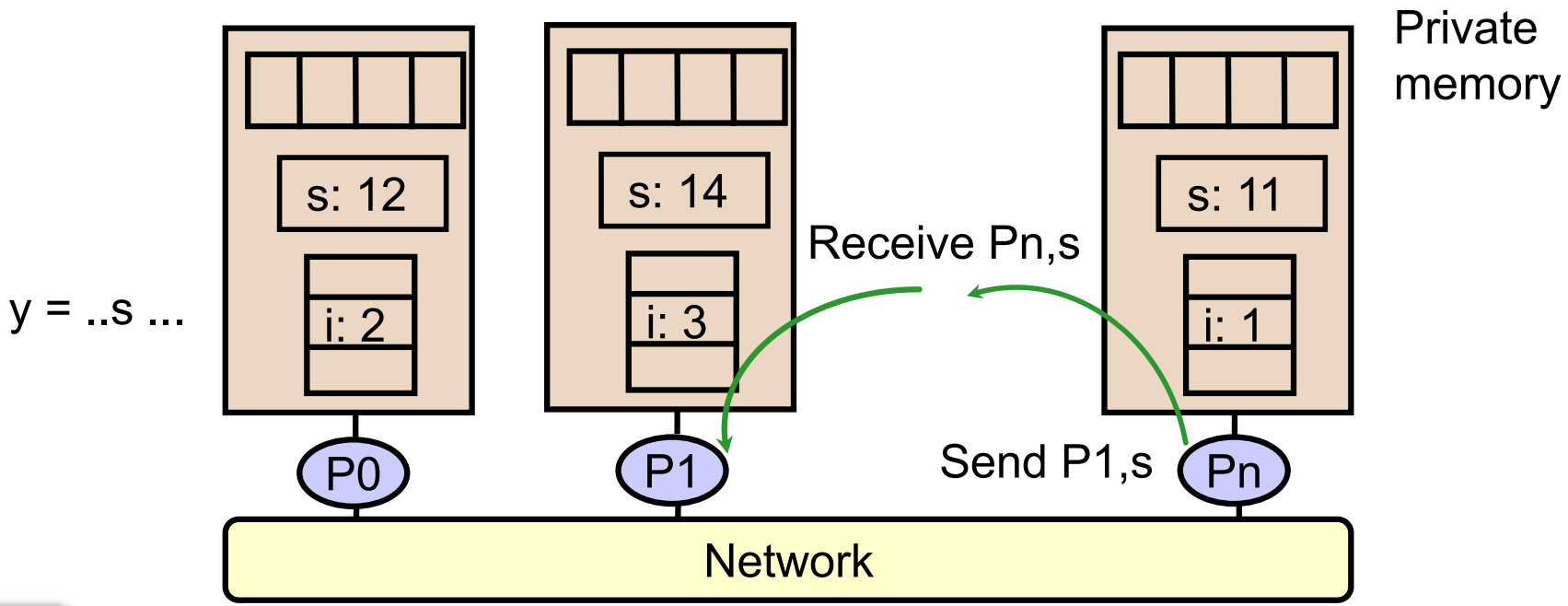
Programming model: message passing

The program consists of a set of named processes

- Generally at the start of the program
- No data sharing: a control thread and a local address space
- Data is partitioned between local processes

Processes communicate with explicit send / receive pairs

- Coordination is implicit in each communication event
- MPI (Message Passing Interface) is the most used API



Compute s = A[1]+A[2] on each processor

° First possible solution - what can crash?

```
Processor 1
    xlocal = A[1]
    send xlocal, proc2
    receive xremote, proc2
    s = xlocal + xremote
```

```
Processor 2
    xlocal = A[2]
    send xlocal, proc1
    receive xremote, proc1
    s = xlocal + xremote
```

° If the send / receive behave like the telephone system?
° Like the surface mail system?

° Second possible solution

```
Processor 1
    xlocal = A[1]
    send xlocal, proc2
    receive xremote, proc2
    s = xlocal + xremote
```

```
Processor 2
    xlocal = A[2]
    receive xremote, proc1
    send xlocal, proc1
    s = xlocal + xremote
```

° What happens if we have more processors?

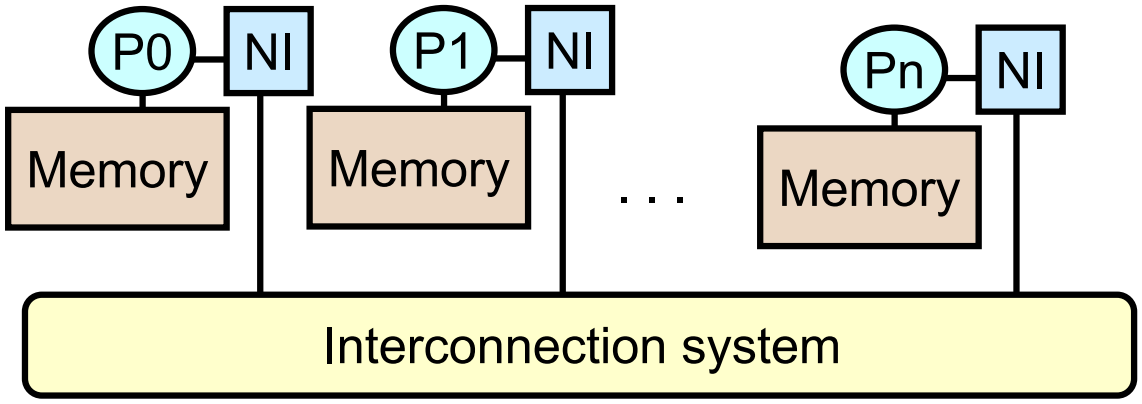
Distributed memory

Examples

- Cray XT4, XT 5
- PC clusters (Berkeley NOW, Beowulf)

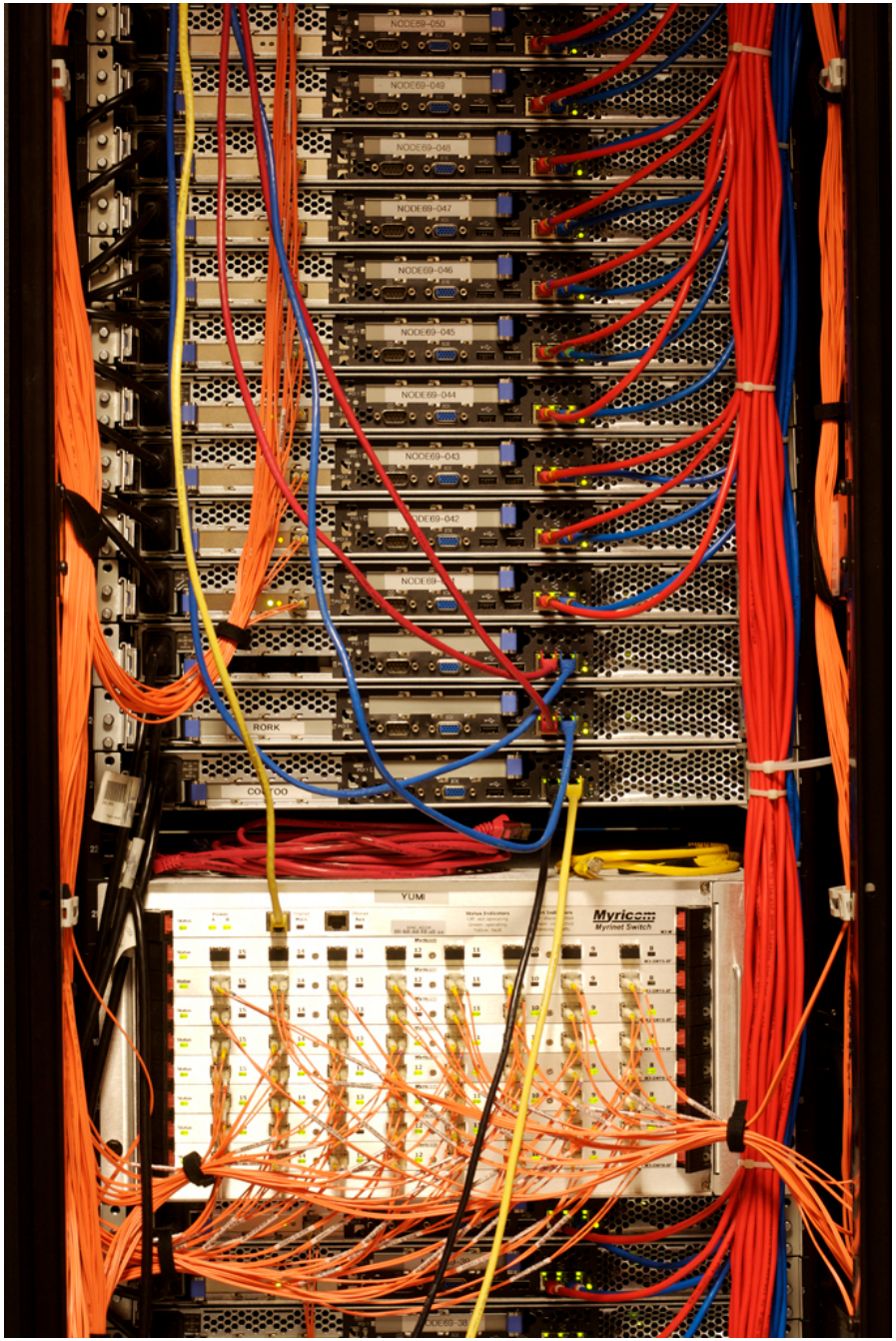
• Each processor has its own memory and cache, but can not access the memory of others

• Each "node" has its own network interface (NI) for all communications and synchronizations



Beowulf (T. Sterling)

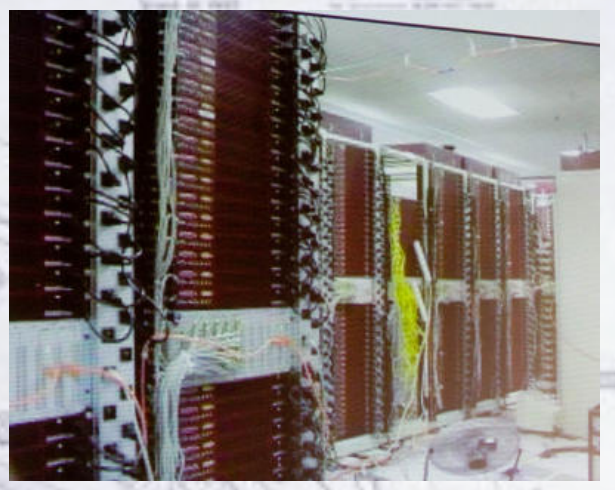

A24
10.69.137.23
OS booting
ID
select
A23
10.69.137.22
Main power on
ID
select
A22
10.69.137.21
OS booting
ID
select

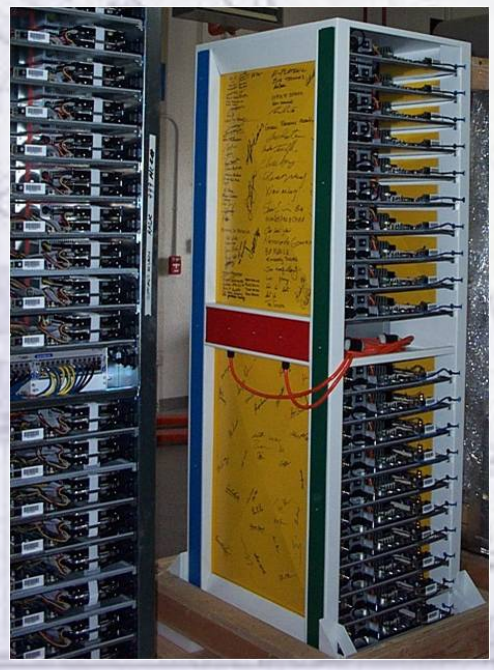

YUMI
Myricom
Myrinet Switch

Google cluster 1997

Google Data centers

- ~ 20 data centers containing more than one million servers around the world
- 40 servers / rack





facebook

Adresse électronique | Mot de passe | Se conn

Garder ma session active | Mot de passe oublié ?

Inscription Facebook vous permet de rester en contact et d'échanger avec les personnes qui vous entourent.

Articles de Facebook Engineering

Articles sur Facebook Engineering

Abonnement

Articles de Facebook Engineering

Building Efficient Data Centers with the Open Compute Project

par Jonathan Heiliger, jeudi 7 avril 2011, 10:45

OPEN Compute Project

A small team of Facebook engineers spent the past two years tackling a big challenge: how to scale our computing infrastructure in the most efficient and economical way possible.

Working out of an electronics lab in the basement of our Palo Alto, California headquarters, the team designed our first data center from the ground up; a few months later we started building it in Prineville, Oregon. The project, which started out with three people, resulted in us building our own custom-designed servers, power supplies, server racks, and battery backup systems.

Because we started with a clean slate, we had total control over every part of the system, from the software to the servers to the data center. This meant we could:

- Use a 480-volt electrical distribution system to reduce energy loss
- Remove anything in our servers that didn't contribute to efficiency
- Reuse hot aisle air in winter to both heat the offices and the outsi the data center.
- Eliminate the need for a central uninterruptible power supply.

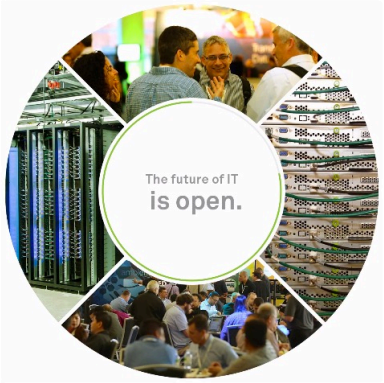
The result is that our Prineville data center uses 38 percent less energy as Facebook's existing facilities, while costing 24 percent less.

OPEN Compute Project

About | Learn | Buy | Participate | Projects | News | Contact | Sign In

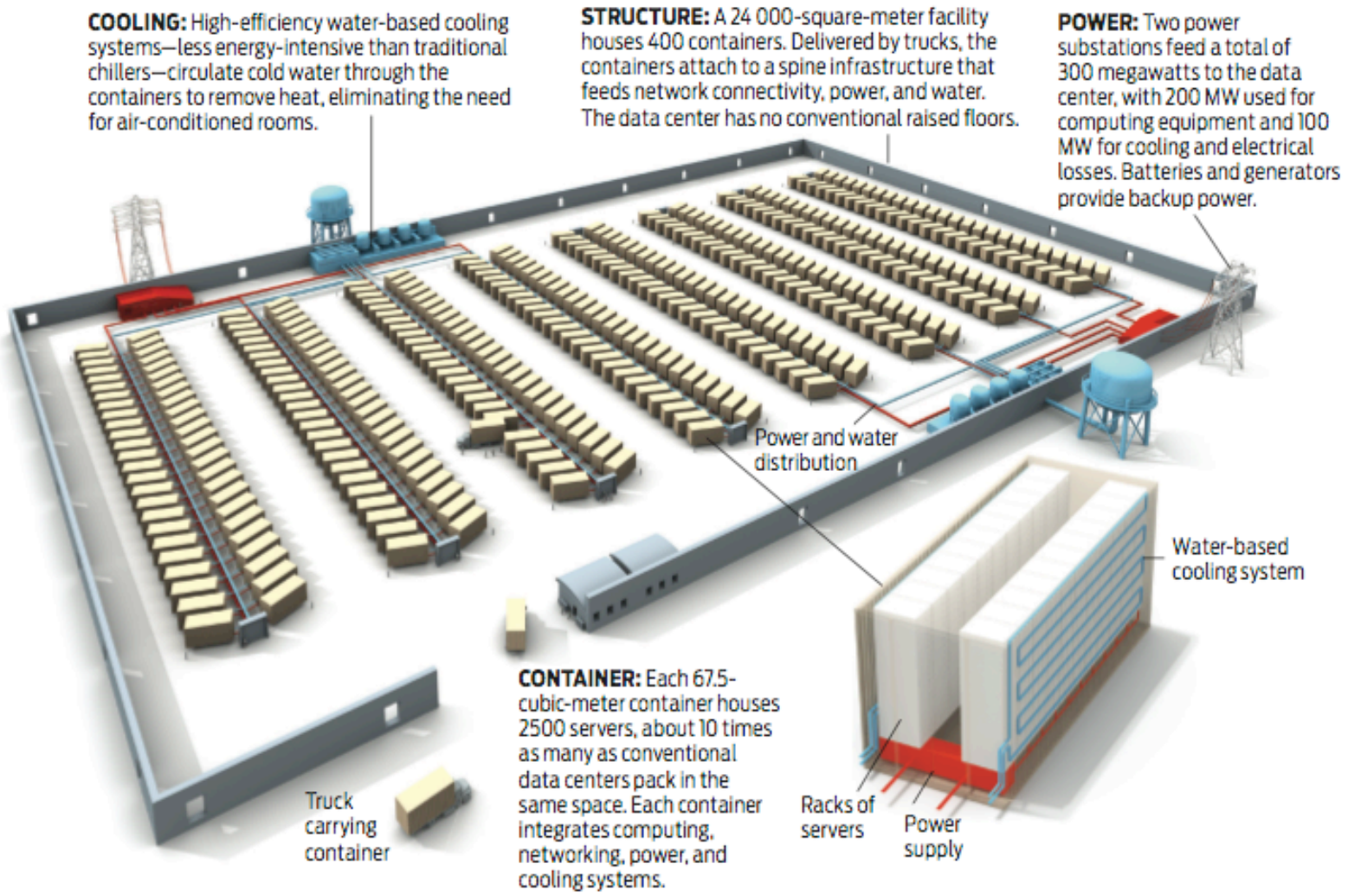
Take control of your technology future

The Open Compute Project (OCP) is reimagining hardware, making it more efficient, flexible, and scalable. Join our global community of technology leaders working together to break open the black box of proprietary IT infrastructure to achieve greater choice, customization, and cost savings.



http://opencompute.org/

The Million-Server Data Center



http://spectrum.ieee.org/tech-talk/semiconductors/devices/what-will-the-data-center-of-the-future-look-like

IBM Roadrunner (2008)

First computer to reach the Petaflops (10^{15} flops)

Roadrunner runs on

- 6,948 dual-core AMD Opteron chips on IBM Model LS21 blade servers
- 12,960 Cell engines (same as PS3) on IBM Model QS22 blade servers

With 80 terabytes of memory, the Roadrunner system and is housed in 288 IBM BladeCentre racks occupying 6,000 square feet.

10,000 connections, both Infiniband and gigabit Ethernet, with 57 miles of fiber-optic cable.