Parallelism

Master 1 International

Andrea G. B. Tettamanzi
Université de Nice Sophia Antipolis
Département Informatique
andrea.tettamanzi@unice.fr

Lecture 5

Parallel Architectures – Describing Concurrent and Parallel Algorithms

Plan

- Part ***a***: Parallel Architectures
  - Instruction-Level Parallelism vs. Multiprocessors
  - Historical Perspective
  - Models of Parallelism
- Part ***b***: Describing Parallel and Distributed Algorithms
  - “Informal” Modeling Tools
  - Methodical Design of Parallel Algorithms

Parallel Architectures

Source: Yan Solihin, Fundamentals of Parallel Computer Architecture, 2008

Key Points

- Increasingly more and more components can be integrated on a single chip
- Speed of integration tracks Moore’s law: doubling every 18–24 months.
- Performance tracks speed of integration up until recently
- At the architecture level, there are two techniques
  - Instruction-Level Parallelism
  - Cache Memory
- Performance gain from uniprocessor system so significant that making multiprocessor systems is not profitable

Illustration

- 100-processor system with perfect speedup vs. single CPU
  - Year 1: 100x faster
  - Year 2: 62.5x faster
  - Year 3: 39x faster
  - …
  - Year 10: 0.9x faster
- Single-processor performance catches up in just a few years!
- Even worse
  - It takes longer to develop a multiprocessor system
  - Low volume means prices must be very high
  - High prices delay adoption
  - Perfect speedup is unattainable

Why Did Uniprocessor Performance Grow so Fast?

- ~½ from circuit improvement (smaller transistors, faster clock, etc.)
- ~½ from architecture/organization:

- Instruction-Level Parallelism (ILP)
  - Pipelining: RISC, CISC with RISC back-end
  - Superscalar
  - Out of order execution
- Memory hierarchy (Caches)
  - Exploiting spatial and temporal locality
  - Multiple cache levels

Uniproc. Performance Growth is Stalling

- Source of uniprocessor performance growth: instruction level parallelism (ILP)
  - Parallel execution of independent instructions from a single thread
- ILP growth has slowed abruptly
  - Memory wall: Processor speed grows at 55%/year, memory speed grows at 7% per year
  - ILP wall: achieving higher ILP requires quadratically increasing complexity (and power)
- Power efficiency
- Thermal packaging limit vs. cost

Why Instr-Level Parallelism is Slowing

- Branch prediction accuracy is already > 90%
  - Hard to improve it even more
- Number of pipeline stages is already deep (~20-30 stages)
  - But critical dependence loops do not change
  - Memory latency requires more clock cycles to satisfy
- Processor width is already high
  - Quadratically increasing complexity to increase the width
- Cache size
  - Effective, but also shows diminishing returns
  - In general, the size must be doubled to reduce miss rate by a half

Current Trend: Multicore and Manycore

Aspects	Intel Clovertown	AMD Barcelona	IBM Cell
# cores	4	4	8+1
Clock Freq	2.66 GHz	2.3 GHz	3.2 GHz
Core type	OOO Superscalar	OOO Superscalar	2-issue SIMD
Caches	2x4MB L2	512KB L2 (private), 2MB L3 (shd)	256KB local store
Chip power	120 Watts	95 Watts	100 Watts

Historical Perspective

"If the automobile industry advanced as rapidly as the semiconductor industry, a Rolls Royce would get ½ million miles per gallon and it would be cheaper to throw it away than to park it."

Gordon Moore,

Intel Corporation

Historical Perspective

- 80s: Prime Time for parallel architecture research
- 90s: emergence of distributed (vs. parallel) machines
  - Progress in network technologies
  - Connects cheap uniprocessor systems into a large distributed machine: Clusters, Grid
- 00s: parallel architectures are back
  - Transistors per chip >> microproc transistors
  - Harder to get more performance from a uniprocessor
  - SMT (Simultaneous multithreading), CMP (Chip Multi-Processor), ultimately *Massive CMP*

What is a Parallel Architecture?

- A parallel computer is a **collection of processing elements** that can **communicate** and **cooperate** to **solve a large problem fast**. [Almasi&Gottlieb]

- "collection of processing elements"
  - How many? How powerful each? Scalability?
  - Few very powerful vs. many small ones
- "that can communicate"
  - Shared memory vs. message passing
  - Interconnection network (bus, multistage, crossbar, …)
  - Evaluation criteria: cost, latency, throughput, scalability, and fault tolerance

What is a Parallel Architecture?

- "cooperate"
  - Issues: granularity, synchronization, and autonomy
  - Synchronization allows sequencing of operations to ensure correctness
  - Granularity up => parallelism down, communication down, overhead down
  - Autonomy
    - SIMD (single instruction stream) vs. MIMD (multiple instruction streams)

What is a Parallel Architecture?

- “solve a large problem fast”
  - General- vs. special-purpose machine?
  - Any machine can solve certain problems well

  What domains?
  - Highly (embarassingly) parallel applications
    - Many scientific codes
  - Medium parallel apps
    - Many engineering apps (finite-elements, VLSI-CAD)
  - Non-parallel applications
    - Compilers, editors (do we care?)

Why Parallel Computers?

- Absolute performance: Can we afford to wait?
  - Folding of a single protein takes years to simulate on the most advanced microprocessor. It only takes days on a parallel computer
  - Weather forecast: timeliness is crucial
- Cost/performance
  - Harder to improve performance on a single processor
  - Bigger monolithic processor vs. many, simple processors
- Power/performance

Loop-Level Parallelism

- Each iteration can be computed independently

```
for (i=0; i<8; i++)
  a[i] = b[i] + c[i];
```

- Each iteration cannot be computed independently, thus does not have loop level parallelism

```
for (i=0; i<8; i++)
  a[i] = b[i] + a[i-1];
```

+ Very high parallelism > 1K
+ Often easy to achieve load balance
- Some loops are not parallel
- Some applications do not have many loops

Task-Level Parallelism

- Arbitrary code segments in a single program
- Across loops:

```
…
for (i=0; i<n; i++)
  sum = sum + a[i];
for (i=0; i<n; i++)
  prod = prod * a[i];
…
```

- Subroutines:

```
Cost = getCost();
A = computeSum();
B = A + Cost;
```

- Threads: e.g. editor: GUI, printing, parsing

+ Larger granularity => low overheads, communication

- Low degree of parallelism

- Hard to balance

Program-Level Parallelism

- Various independent programs execute together
- gmake:
  - gcc –c code1.c // assign to proc1
  - gcc –c code2.c // assign to proc2
  - gcc –c main.c // assign to proc3
  - gcc main.o code1.o code2.o

+ no communication

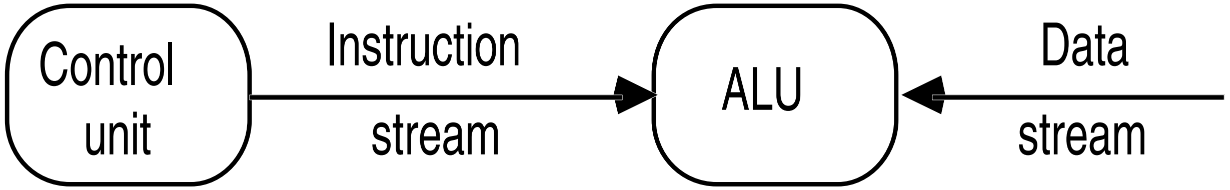
- Hard to balance

- Few opportunities

Taxonomy of Parallel Computers

The Flynn taxonomy:

- *Single* or *multiple instruction* streams.
- *Single* or *multiple data* streams.
- **1. SISD machine (Most desktops, laptops)**
  - Only one instruction fetch stream
  - Most of today's workstations or desktops



SIMD

- Examples: Vector processors, SIMD extensions (MMX)
- A single instruction operates on multiple data items.

```
SISD:
for (i=0; i<8; i++)
  a[i] = b[i] + c[i];
```

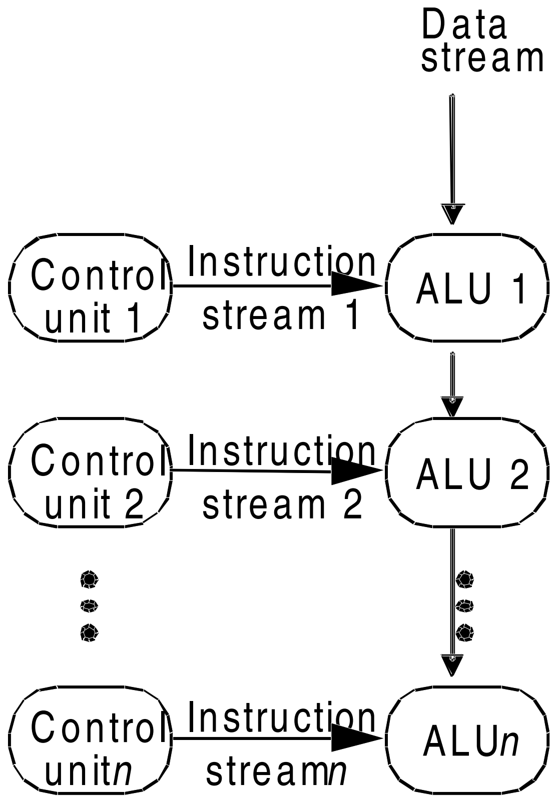
```
SIMD:
a = b + c;  // vector addition
```

𝔪

- Pseudo-SIMD popular for multimedia extension

MISD Machine

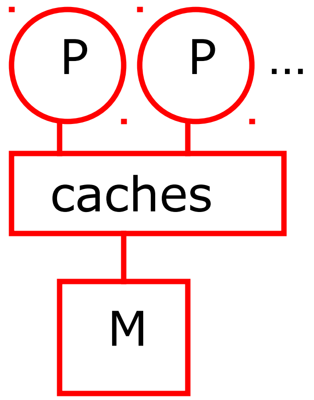
- Example: CMU Warp
- Systolic arrays



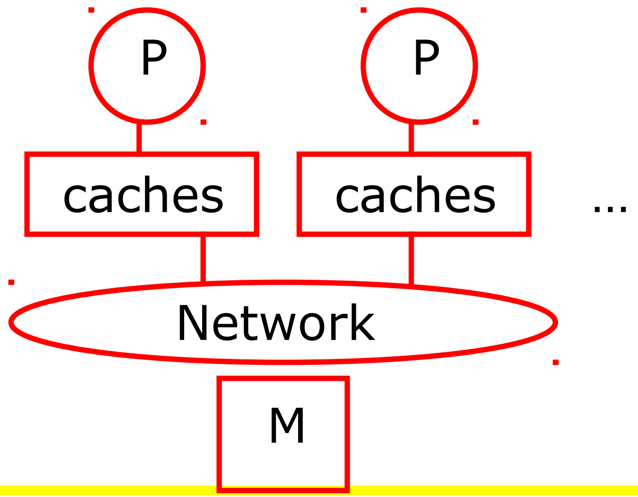
MIMD Machine

- Independent processors connected together to form a *multiprocessor* system.
- Physical organization: which memory hierarchy level is shared
- Programming abstraction:
  - Shared Memory:
    - On a chip: Chip Multiprocessor (CMP)
    - Bus interconnection: Symmetric multiprocessors (SMP)
    - Point-to-point interconnection: Distributed Shared Memory (DSM)
  - Distributed Memory:
    - Clusters, Grid

MIMD Physical Organization

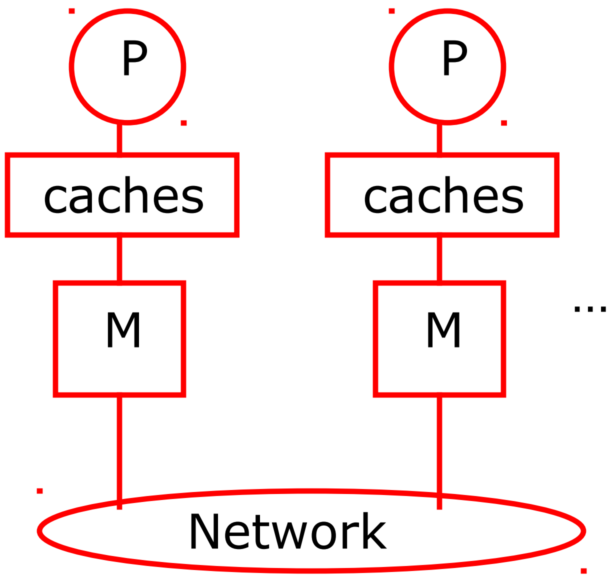


Shared Cache Architecture:
- CMP (or Simultaneous Multi-Threading)
- e.g.: Pentium4 chip, IBM Power4 chip, SUN Niagara, Pentium D, etc.
- Implies shared memory hardware



UMA (Uniform Memory Access) Shared Memory :
- Pentium Pro Quad, Sun Enterprise, etc.
- What interconnection network?
  - Bus
  - Multistage
  - Crossbar
  - etc.
- Implies shared memory hardware

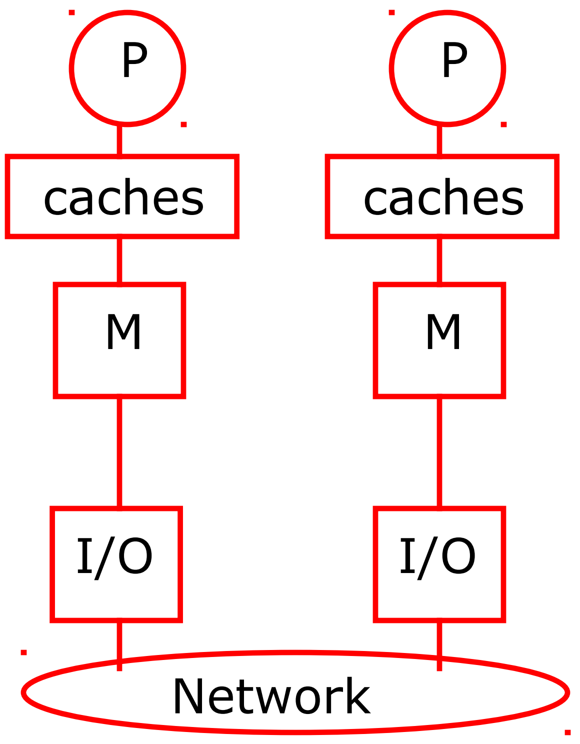
MIMD Physical Organization



NUMA (Non-Uniform Memory Access) Shared Memory :

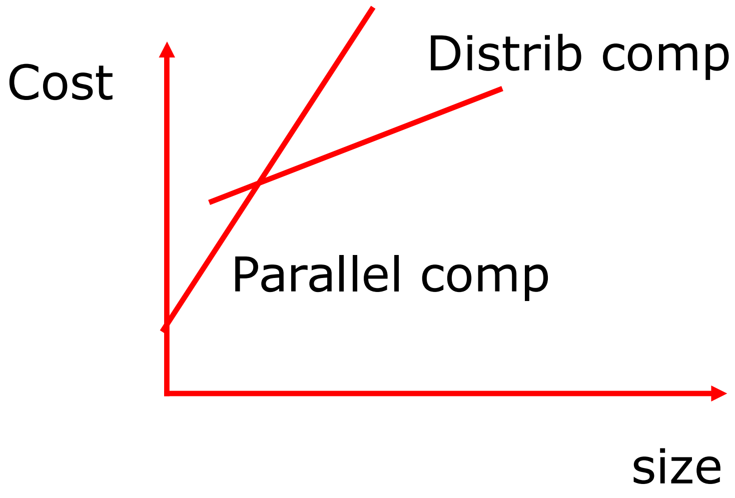
- SGI Origin, Altix, IBM p690, AMD Hammer-based system
- What interconnection network?
  - Crossbar
  - Mesh
  - Hypercube
  - etc.
- Also referred to as *Distributed Shared Memory*

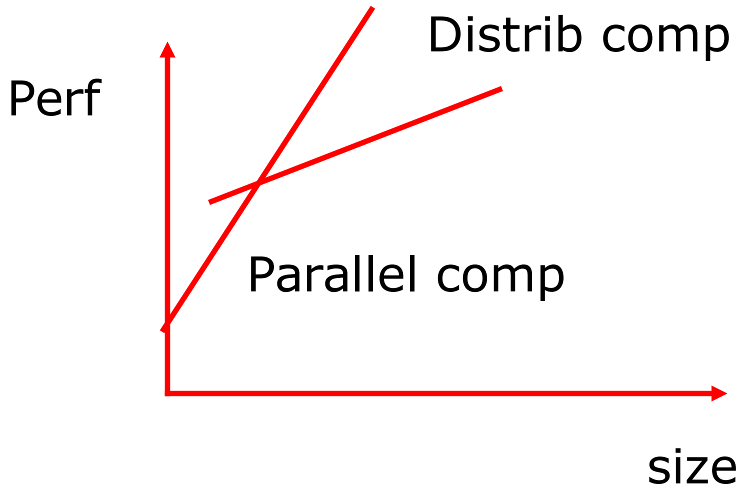
MIMD Physical Organization



Distributed System/Memory:
- Also called clusters, grid
- Don't confuse it with *distributed shared memory*

Parallel vs. Distributed Computers





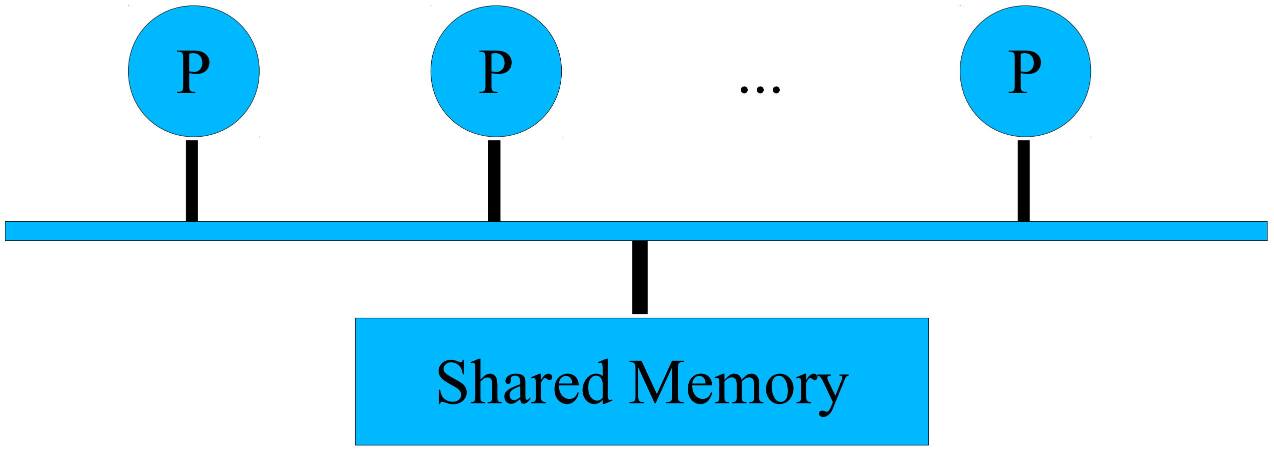
- Small scale machines: parallel system cheaper
- Large scale machines: distributed system cheaper

- Performance: parallel system better (but more expensive)
- System size: parallel system limited, and cost grows fast

- However, must also consider software cost

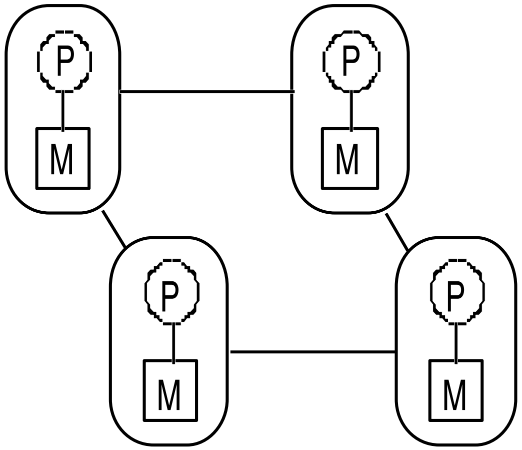
Programming Models: Shared Memory

- Shared Memory / Shared Address Space:
  - Each processor can see the entire memory
  - Programming model = thread programming in uniprocessor systems



Programming Models: Distributed Memory

- Distributed Memory / Message Passing / Multiple Address Space:
  - a processor can only directly access its own local memory. All communication happens by explicit messages.



Shared Mem compared to Msg Passing

+ Can easily be automated (parallelizing compiler, OpenMP)
+ Shared vars are not communicated, but must be guarded
– How to provide shared memory? Complex hardware
– Synchronization overhead grows fast with more processors
± Difficult to debug, not intuitive for users

Top 500 Supercomputers

http://www.top500.org