Parallel Computer Organization

Classification of Parallel Architectures

Major Parallel Architectures

- SIMD computers (diminishing/coprocessors)
- Shared memory multiprocessors
- Message-passing multicomputers

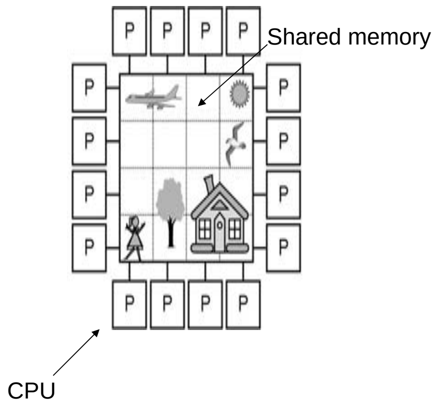
Full taxonomy given later.

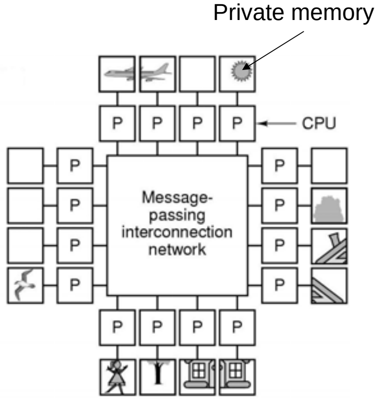
A Major Design Issue

- How are the CPUs and memory connected?
  - Multiprocessors (shared memory system):
    - all CPUs share common memory so processes on different CPUs communicate by accessing same memory location.
    - One, single virtual address space.
  - Multicomputers (distributed memory system):
    - each CPU has its own, private memory.
    - CPUs must pass messages to each other to communicate.
    - Separate virtual address spaces per CPU.

Shared vs Distributed Memory

Multiprocessor with 16 CPUs sharing memory holding image



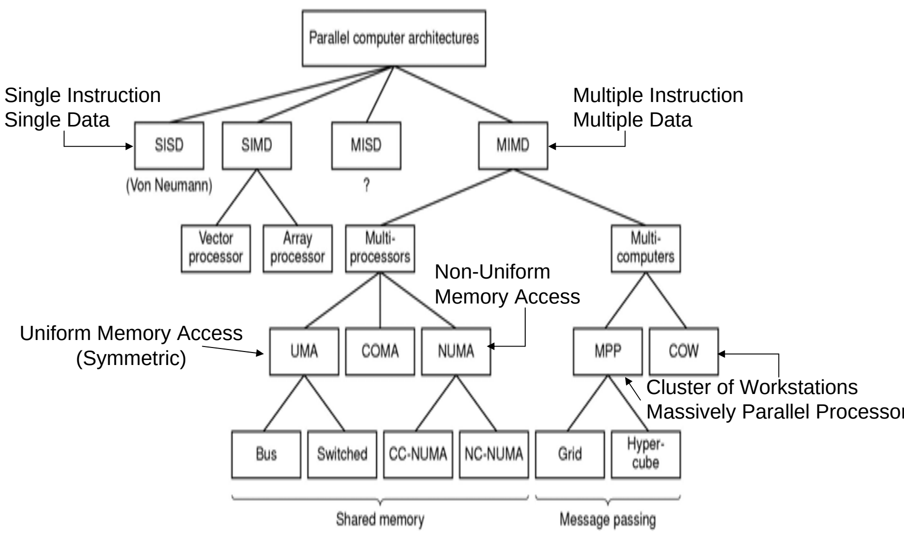


Multicomputer with 16 CPUs each with own, private memory holding part of image

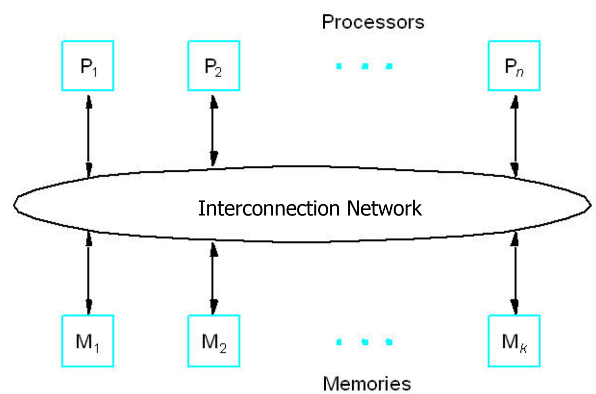
Distributed Shared Memory

- Multicomputers harder to program than multiprocessors because they have to code the sending and receiving of messages.
- Multicomputers ***much*** cheaper, easier to build than multiprocessors.
- One compromise is DSM (Distributed Shared Memory): multicomputer hardware with operating system that can simulate multiprocessor (shared) memory

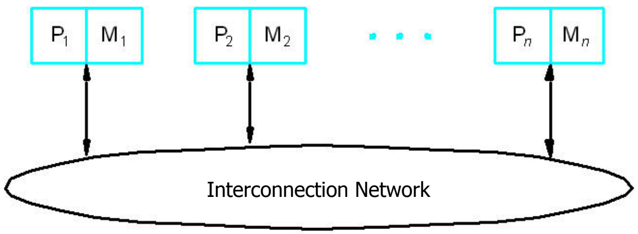
Flynn's Taxonomy of Parallel Computers



UMA Multiprocessor System



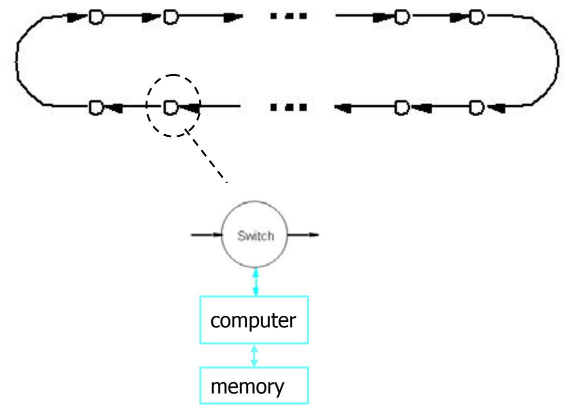
NUMA Multiprocessor System



Distributed Shared Memory

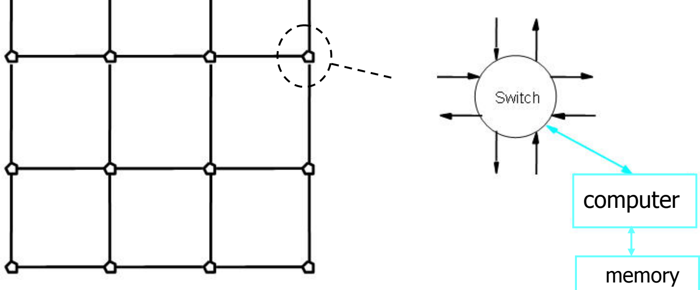
Ring Multicomputer System

Interconnection Network



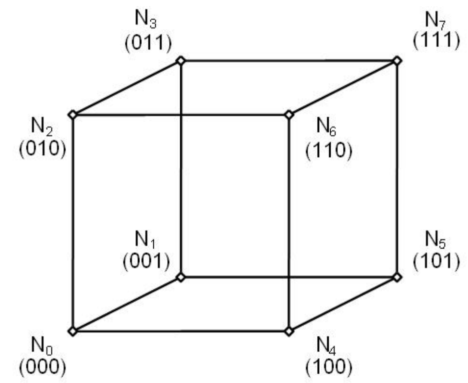
2-D Mesh Multicomputer System

Interconnection Network

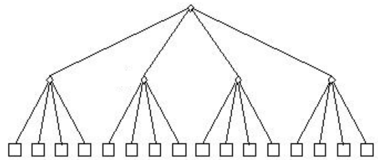


3-D Hypercube Multicomputer System

Interconnection Network



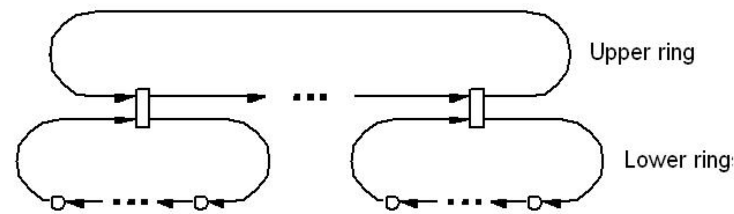
Tree Multicomputer System



4-Way Tree

Fat Tree

Hierarchy of Multicomputer Interconnection Networks



Parallel Virtual Machine (PVM)

- Multicomputer communication software: PVM (Parallel Virtual Machine)
  - Public-domain, UNIX-based for COWs
  - User-callable library and a daemon process that runs all the time to execute the function calls
  - Uses synchronous sends (blocking sends)
  - Also supports broadcasting

Message Passing Interface (MPI)

- Multicomputer communication software: MPI (Message Passing Interface)
  - More functions with more varied parameters than PVM. Has 4 basic concepts:
    - Communicators: group of processes that will communicate with each other
    - Message data types: type of data being sent (i.e., double)
    - Communication operations: functions for sending/receiving data. Has functions for synchronous, buffered and non-blocking
    - Virtual topologies: Processes can be arranged in tree, grid, torus, etc. so work well on different hardware and programs can specify paths to other programs by name

Performance Issues

Speedup

- Speedup: how much faster program runs on parallel machine compared to non-parallel machine.
- Amdahl's law: can't get linear speedup because of sequential parts of code
- Also can't get linear speedup because added communication time comes with additional processors

Amdahl's Law Applied to Parallel Computing [1]

- Let N be the number of processors
- S the amount of time spent (by a serial processor) on serial parts of the program
- P is the amount of time spent (by a serial processor) on parts of the program that can be done in parallel
- Then Amdahl's law says that speedup is given by

Amdahl's Law Applied to Parallel Computing [2]

Speedup = (S + P) / (S + P/N)

or = 1 / (s + p/N)

where s is fraction of time spent in serial computation and p is fraction of time spent in parallel computation (i.e., s + p = 1).

Example

- Suppose we have a code with s = .25 (thus, p = .75), and we apply 20 processors to the problem. What is the expected speedup?

$$Speedup = \frac{1}{.25 + \frac{.75}{20}} = 3.5$$

Amdahl's Law Revisited (by Gustafson-Barsis)

- Maybe the picture isn't as grim as first imagined
- Amdahl assumes as N increases, that problem size remains fixed
  - In practice, this is not usually the case
- More processors usually implies larger, more complex problems to be solved and bigger problems usually increase the parallel part and with less effect upon the serial part

Amdahl's Law Revisited "Scaled Speedup"

- Suppose problem involves data of size n and computation of size n^2.
- If we place 2 processors on the problem, then we can double the size of the problem to 2n, which would then involve $4n^2$ computational work.
- If serial part does not grow proportionally to parallel part, then $s_2 << s_1$ and $p_1 << p_2$

Example Revisited

- Suppose the code with original data n had $S_1 = 25$ and $P_1 = 75$, and we apply 20 processors to the problem with 20 times the data. Also, suppose the time for S_2 doubles, and the time for P grows as n^2.

$Time_2 \approx 2*25 + 400*75 = 30,050$

$s_2 = 50/30,050 = .0017$

$p_2 = 30,000/30,050 = .9983$

$$Speedup = \frac{1}{.0017 + \frac{.9983}{20}} = 19.4$$

Another Perspective

- This is the parallel programming equivalent of the old adage that while one woman can have a baby in nine months, nine woman can't have a baby in one month (Amdahl) – but they can have nine babies in nine months (Scaled Speedup).

Top 500 Computers

www.top500.org