An Overview of SIMD and MIMD Architectures

The Physical and Theoretical Models

Tópico 2

Parallel Architecture Design

The relationships to the program parallelism that need to be supported have to be taken into consideration in the architecture design which in itself involves effort in:

1. Replicating *processing hardware*; and
2. Partitioning *memory* while maintaining well-matched processing and memory bandwidths.
3. Furthermore, components 1. and 2. must be connected into an integrated structure – therefore requiring an *interconnection network*.

Types of Architectures

Generally speaking there are 2 basic types of parallel architecture:

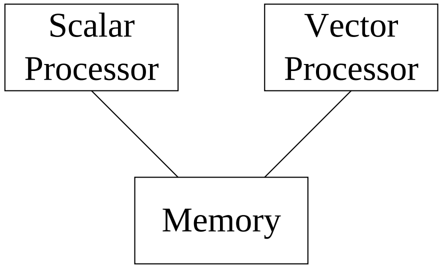
- SIMD computers
  - synchronous execution ("lock step")
  - although "synchronous" is **not** equivalent to "SIMD", existing array processors are SIMD.
- Multiprocessors and Multicomputers
  - Asynchronous execution
  - MIMD

Other "non-mainstream" parallel machines are more difficult to classify.

Vector Processors

It might be considered that vector processors generalise to higher dimensions:

- conceptually, maybe
- but still “essentially” von Neumann machine.



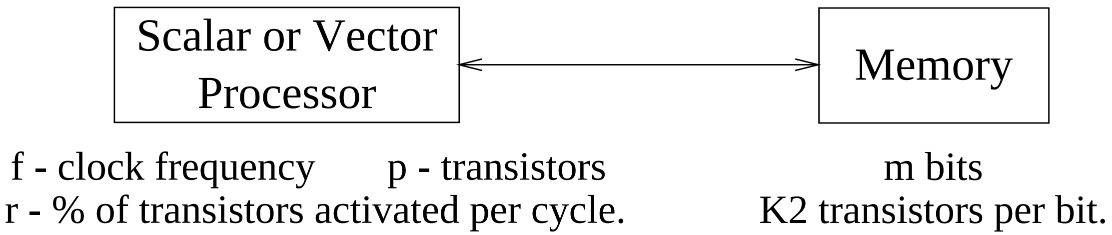
Anyway, vector architectures can do pretty well running standard sequential code.

Which raises the following question......

Why SIMD Arrays (or multiprocessors) instead of Vector Processors?

- Problem 1: Von Neumann Bottleneck. [See Backus "Can Programming be liberated from the von Neumann style?" CACM, 1978

- Problem 2: Inefficiency of von Neumann architectures.



It is known (empirically) that the memory size should be proportional to the processor speed: $m = O(f)$ or $m = K_1 \cdot f$ where K_1 is a constant.

Thus the total machine complexity, in terms of transistors, is: $p + K_1 \cdot K_2 \cdot f$

Since each instruction activates $O(1)$ memory bits, i.e. a constant K_3 bits, so the number of transistors activated per cycle (or instruction) is:
$r \cdot p + K_2 \cdot K_3$

and thus the utilisation of the machine is:

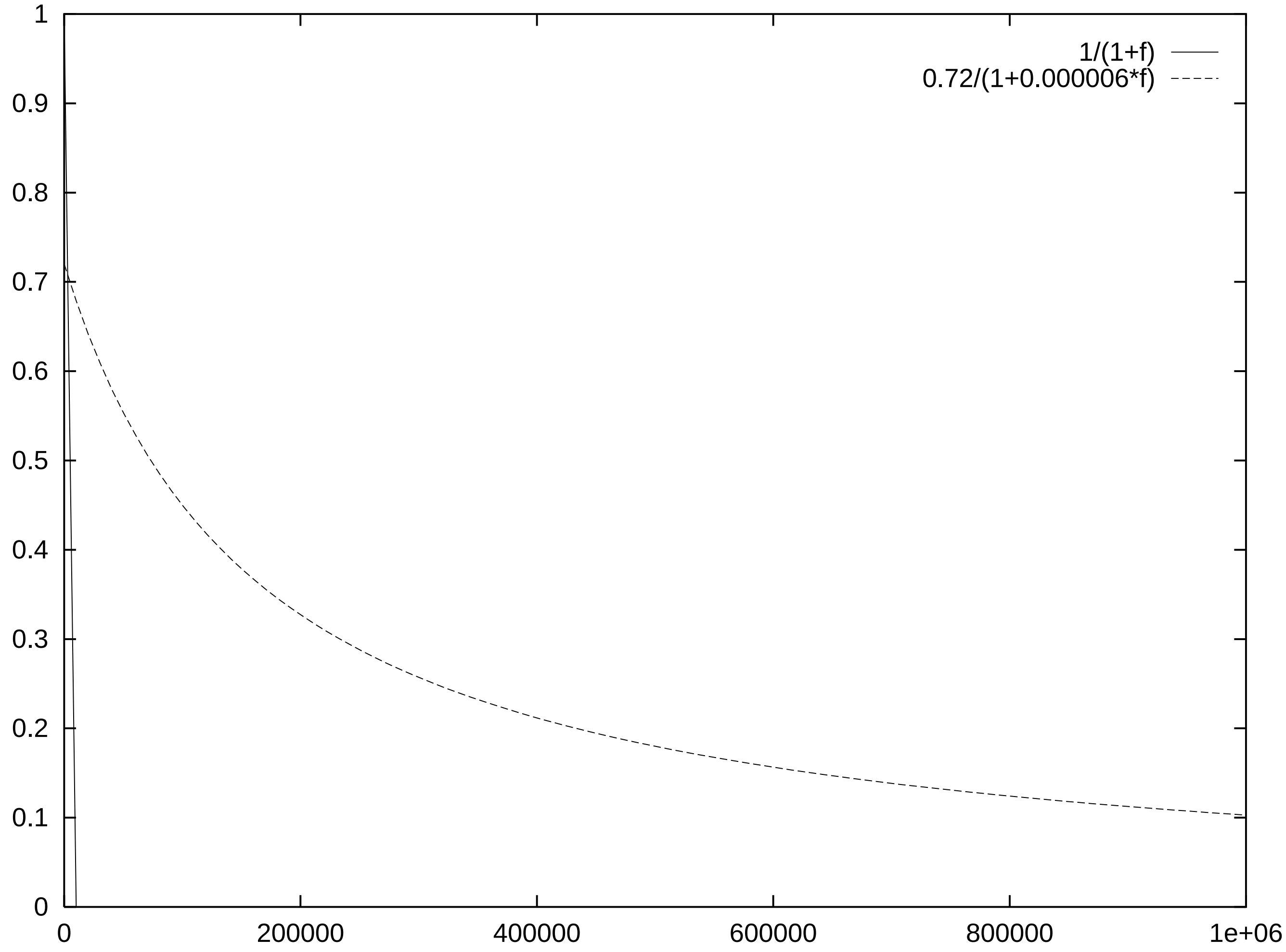
$$U = \frac{r \cdot p + K_2 \cdot K_3}{p + K_1 \cdot K_2 \cdot f}$$

Setting $A = r + \frac{K_2 \cdot K_3}{p}$ and $B = \frac{K_1 \cdot K_2}{p}$ (both being constants) gives:

$$U = \frac{A}{1 + B \cdot f}$$

which is a simple expression of how utilisation changes with the cycle time. (i.e. how utilisation drops as the instruction rate or clock frequency increases!)

Utilisation Vs Clock Frequency



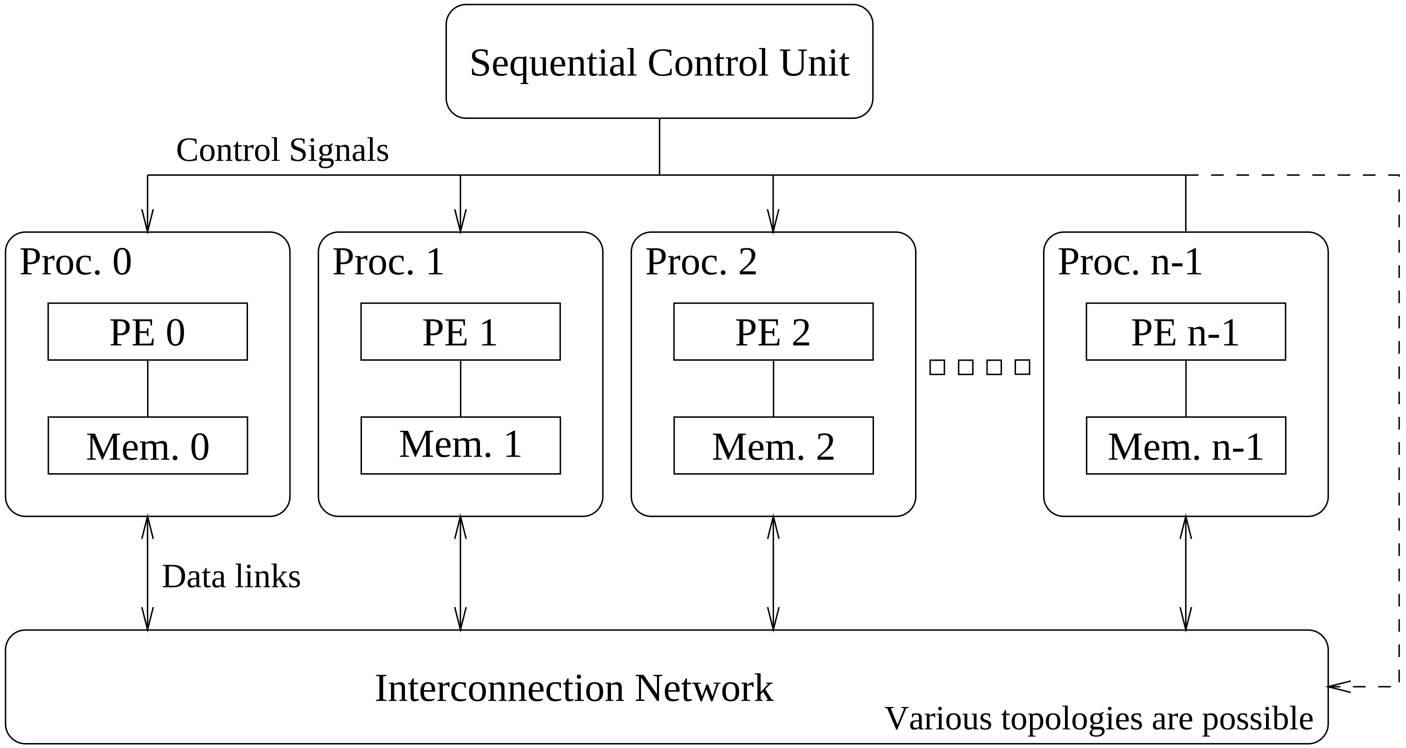
What can we conclude?

- There is a price to be paid for high performance
- Is something "wrong" with von Neumann architectures?

Solution (to both problems)

- Replicate the memory, i.e. partition the memory
  - "logic-in-memory" machines
  - "multiple data" machines
  - SIMD (and also MIMD)

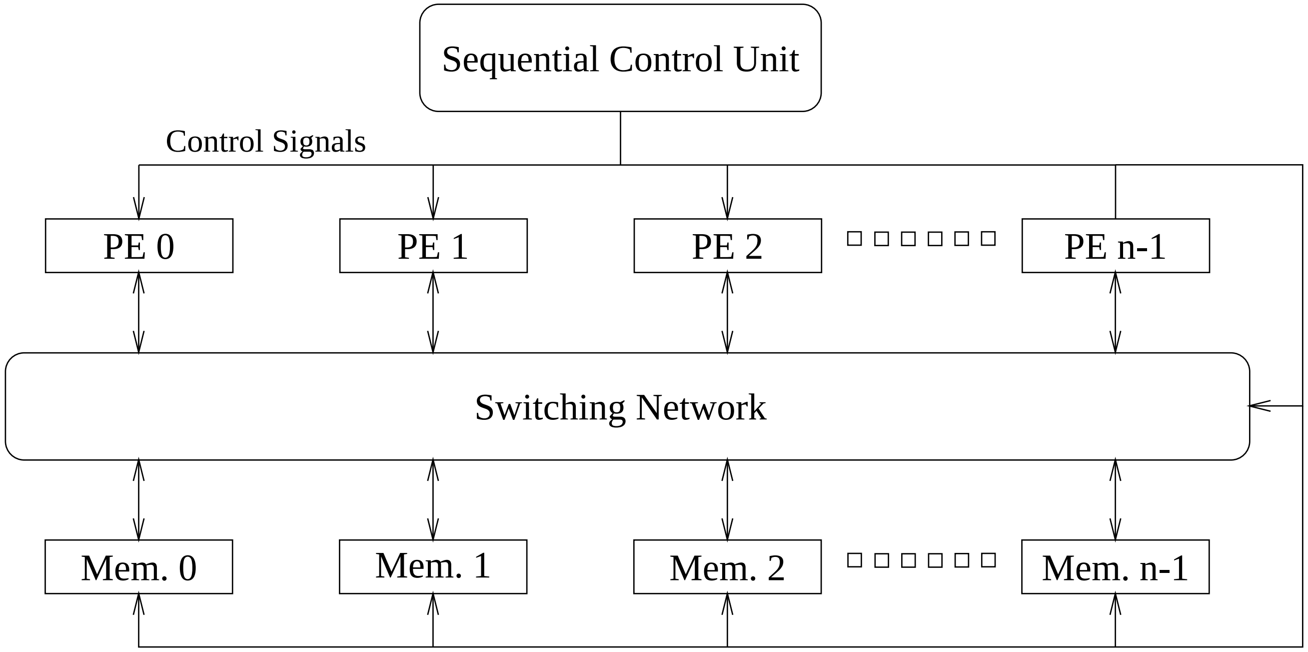
SIMD Array Processor (Ver. A)



BASIS : - "distributed memory" or "private memory"

- communication either via non-local memory access or direct messages

SIMD Array Processor (Ver. B)



Total Memory size = m; Each memory has m/n locations. (Normally m>n.)

BASIS : - PEs and Memory are separated.
- Equal access to all memory modules from all PEs (usually).
- Exploits regularity in the data access patterns.

SHARED MEMORY

SIMD - Things to remember

- Instruction issue is sequential; each instruction applies to all processors in lockstep; instructions are broadcast from the control unit.

- Processors (or PEs) are simple "data processors"; they have little or no instruction decode ability.

- Processors have "activity flags" or masks which enable them to be turned on or off selectively (i.e. they may "sit out" some instructions, and "participate" in others).

SIMD - Things to remember$_{(cont)}$

- Switching, or interconnection networks may be:
  - partial : (cheap, simple, restricted)
  - complete : (expensive, complex, unrestrictive)

- Data can be loaded into memory either by a broadcast from the control unit, or from an external source via a system bus (if so equiped), and analogously for storing data.

Characterising SIMD machines

A SIMD computer can basically be specified by 5 characteristics.

1. The number of Processing Elements (PEs).

2. The set of instructions for the "front end" control unit.

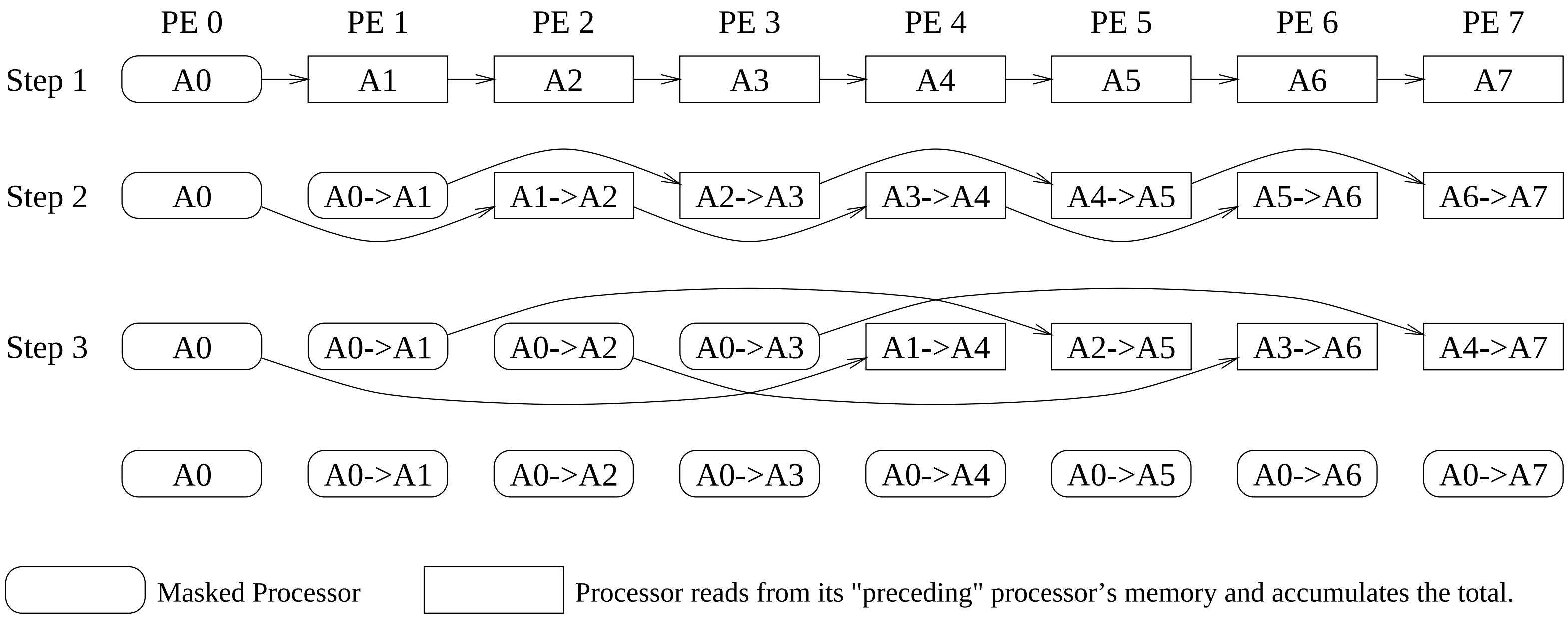
3. The set of (decoded) instructions broadcast by the control unit to all of the PEs - only active PEs will execute instructions.

4. The set of Masking schemes which partition PEs into “activated” and “deactivated” sets for particular instructions.

5. The set of routing functions which work on the interconnection network to implement inter-PE communication.

Parallel Prefix Sums

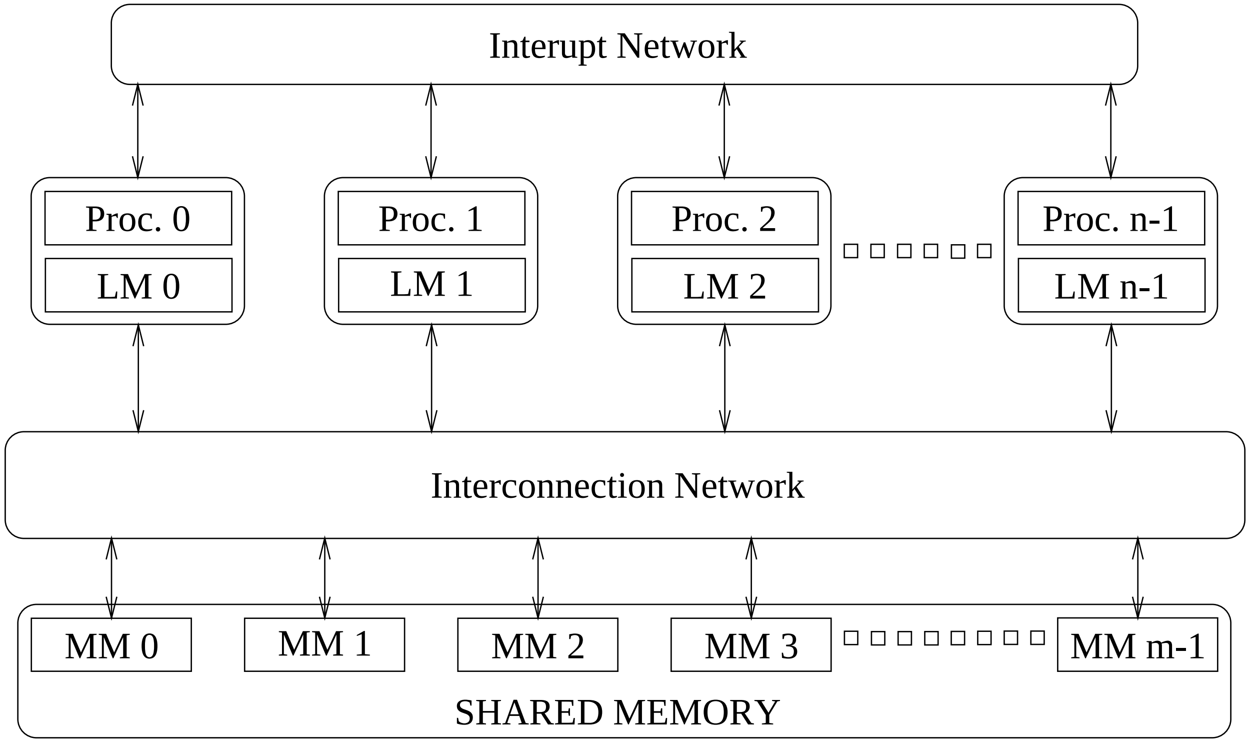
- Sequence of numbers $A_i, 0 \leq i \leq n-1$
- n processors (or PEs)
- A_k is in memory of processor PE_k
- For each i, $0 \leq i \leq n-1$,
  we want to compute $\sum_{j=0}^{i} A_j$
- Assume $n = 8$ and x->y denotes $\sum_{j=x}^{y} A_j$


PE 0
PE 1
PE 2
PE 3
PE 4
PE 5
PE 6
PE 7
Step 1
A0
A1
A2
A3
A4
A5
A6
A7
Step 2
A0
A0->A1
A1->A2
A2->A3
A3->A4
A4->A5
A5->A6
A6->A7
Step 3
A0
A0->A1
A0->A2
A0->A3
A1->A4
A2->A5
A3->A6
A4->A7
A0
A0->A1
A0->A2
A0->A3
A0->A4
A0->A5
A0->A6
A0->A7
Masked Processor
Processor reads from its "preceding" processor's memory and accumulates the total.

MIMD Architectures

- *Shared memory multiprocessors*
  - uniform memory access (UMA)
  - non-uniform memory access (NUMA)
  - cache-only memory access (COMA)
- *Distributed memory multicomputers*

MIMD Multiprocessors



Interconnection Network is DYNAMIC (e.g. bus, X-bus, multistage).

Multiprocessors are sometimes referred to as "tightly coupled systems", whereas multicomputers are "loosely coupled". (Obsolete terminology)

UMA variant

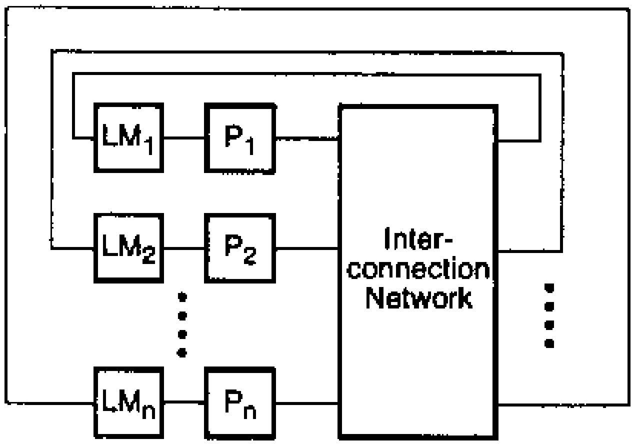
- Physical structure (the hardware) is the same as the logical structure - Local Memories (LMs) may include caches in addition to registers.

- Access time to memory from any processor is uniform.

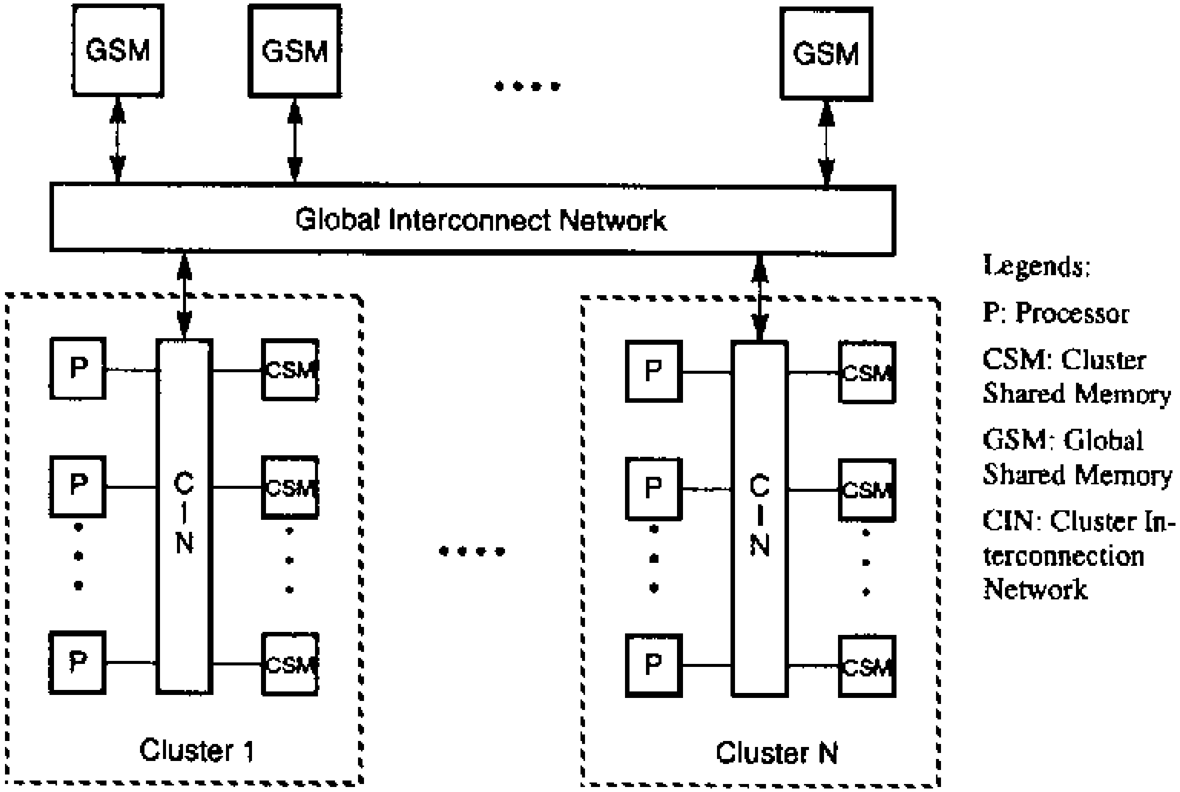
NUMA variant

- Physical structure has each MM_i $(= LM_i)$ attached to P_i, i.e. a "logically shared, physically distributed" memory.
- Access time to memory is non-uniform.
- This permits a variety of memory configurations e.g. a mixture of shared and private memories, global shared memory in addition to distributed memory.
- Multiprocessors can be structured hierarchically into clusters of processors. Each cluster being a UMA or NUMA multiprocessor.

NUMA variant



(a) Shared local memories (e.g., the BBN Butterfly)



(b) A hierarchical cluster model (e.g., the Cedar system at the University of Illinois)

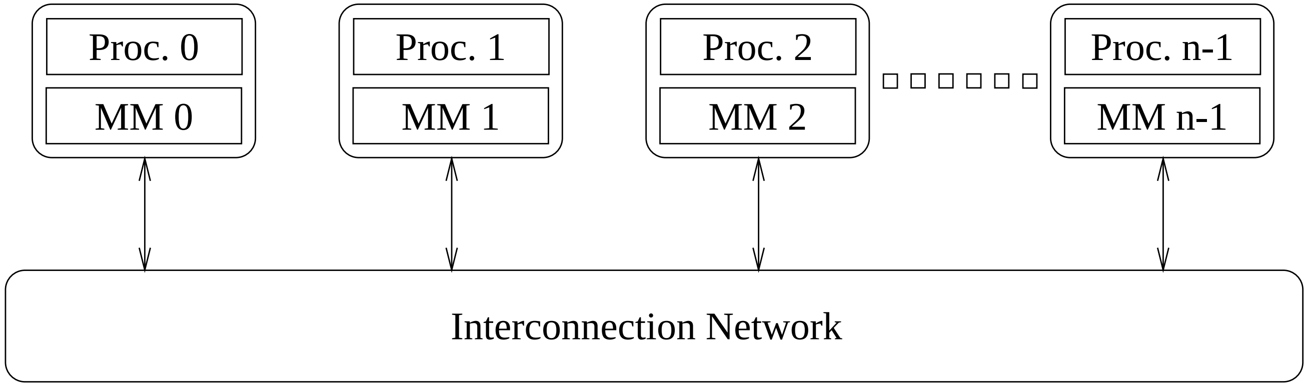
COMA variant

- This is a special case of NUMA: but all of the LM_i ($= MM_i$) are now caches. Results in “dynamic memories”.

- New technology; looks promising (e.g. KSR-1 architecture), very sophisticated caching protocols obviously.

MIMD Multicomputer

- Autonomous, “multiple computers”, inter-networked.
- also known as “loosely coupled”.
- Memory is private, being physically and logically distributed (unlike NUMA).



The Interconnection Network is STATIC (e.g. mesh, hypercube, ring, butterfly, e.t.c.) and may perform other functions in addition to routing.

Who Needs Theory?

So far, we have looked briefly at both hardware and software aspects. But these ideas need to be formalised in order to reason about the *principles* involved.

- Theory is always important! (This is computer science not computer hacking).
- Parallelism is complicated.
- How do we deal with complex things ?
  – Use **Abstraction** to achieve "theoretical results".

The Sequential Computing Model

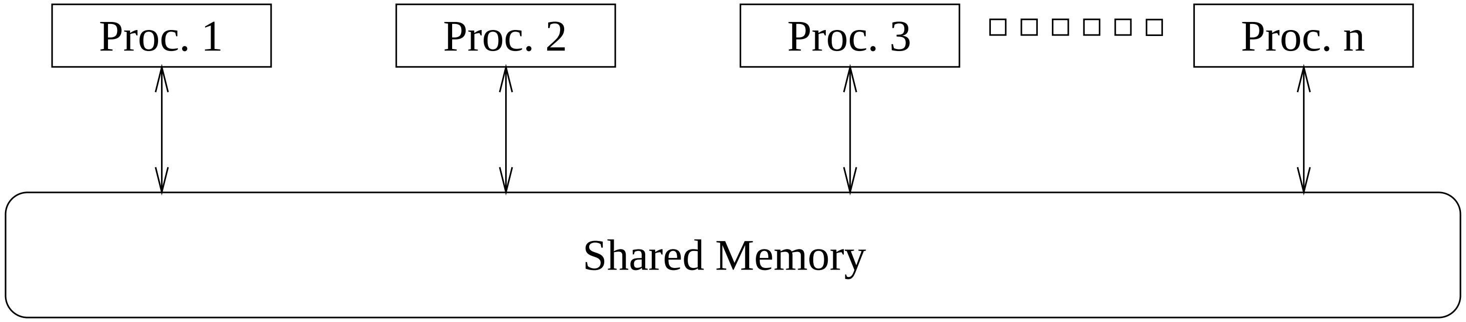
In conventional sequential uniprocessors, hardware and software have been "connected" to theory through the *Random Access Machine* (RAM) model (or the von Neumann Model of Computation). It offers programmers:

- Predictability of performance, and
- Portability, of software.

Ideal models provide a convenient framework for developing algorithms without worrying about implementation details or physical constraints.

An ideal model for parallel computing – the parallel RAM??

PRAM Model



Each processor P_i knows its own ID, i.

Processors execute synchronously (although, may have different instruction streams, a feature that is not usually exploited in algorithms).

Technically, synchronous MIMD, but usually (always?) used in SIMD fashion.

Key Properties

Synchronous execution, with no extra cost charged for synchronisation steps.

Unit time communication to ANY location in memory.

PRAM Model Variants

Given shared memory is assumed, the model must also specify how reads from and writes to memory are handled:

- Exclusive Read (ER);
- Exclusive Write (EW);
- Concurrent Read (CR);
- Concurrent Write (CW) requires some policy to resolve conflicts (e.g. Common, Arbitrary, Minimum, or Priority).

Combinations lead to variants of the PRAM model (EREW, CREW, ERCW, and CRCW).

PRAM Characteristics

- Simplicity
  - algorithm analysis by counting the total number of steps in the algorithm.
  - "universal" data availability to processors.
  - synchronisation is automatic.
- Practicality/Realisability
  - synchronisation and communication take time (i.e. have cost) in any parallel architecture.

This is an example of the tradeoff we will always have between the ease of use and the practicality of building the architecture.

Program Principles

PRAM is good for focusing on principles (theory) of parallel problem solving.

Example: Summing

- Array A of size $n = 2^k$ in shared memory
- $P = n$ processors, indexed by $P_i, 1 \leq i \leq n$
- Put $\sum_{i=1}^{n} A[i]$ in $A[1]$

for $h = 1$ to $log\ n$ do

    for-all P_i, $1 \leq i \leq \frac{n}{2^h}$

        read $A[2i\text{-}1]$ into register x;

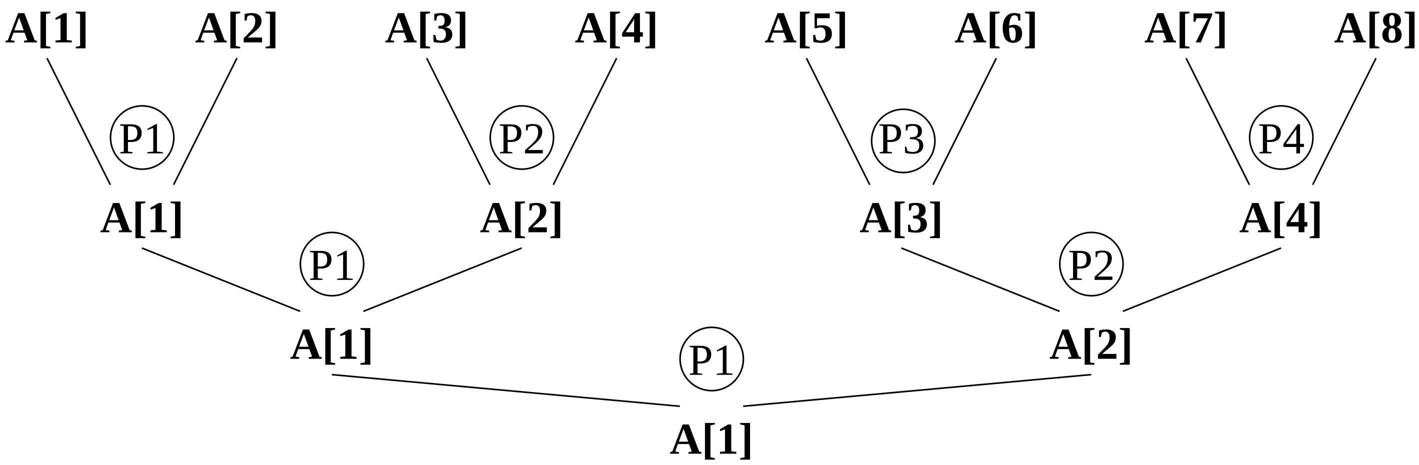
        read $A[2i]$ into register y;

        $z := x + y$;

        write z to $A[i]$

    end-for-all

end-for



Summary

- Two major classes (models) of parallel machines: SIMD and MIMD.
- Organisation models can allow programmers to predict the complexity (and thus roughly the performance/efficiency) of their (parallel) algorithms.
- PRAM and algorithm theory connects parallel architecture to problem solving, programming and *users*.
- Unfortunately, ideal models like PRAM are not realistic and so the search for the "holy grail" continues.

Had the overview of the "big picture", next we add in some details .....