Scuola Superiore Sant'Anna
di Studi Universitari e di Perfezionamento

Università di Pisa

Master Program (Laurea Magistrale) in Computer Science and Networking

High Performance Computing Systems and Enabling Platforms

Marco Vanneschi

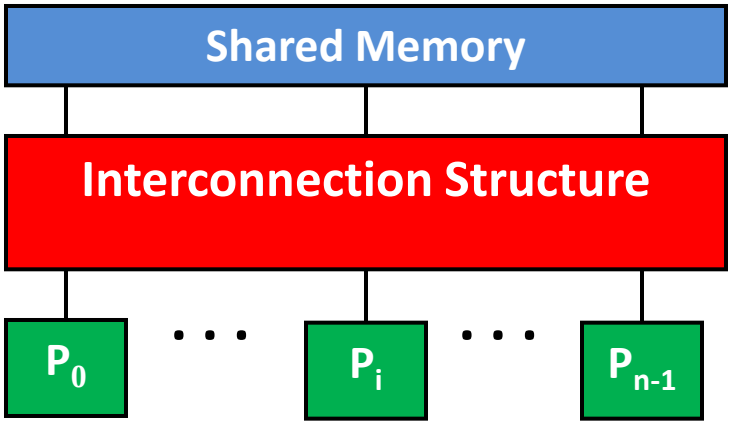
4. Shared Memory Parallel Architectures

4.1. Multiprocessor: organizations and issues

Basic characteristics

- *Shared memory parallel architecture = Multiprocessor*: MIMD (Multiple Instruction Stream Multiple Data Stream) general-purpose architecture
  - nowadays, large diffusion for high-performance servers, medium/high-end workstations
  - *multicore* evolution / revolution
- Homogeneous: *n* identical processors
  - In general, *n* **processing nodes** (**CPU, local memory/caches, local I/O**)
- Processing nodes share the Main Memory physical space:
  - At the firmware level, any processor can access any location of Main Memory
  - In presence of a memory hierarchy, some Cache levels (notably, Secondary,or Tertiary Cache) can be shared
  - Shared information are allocated in shared memory supports, as well as private information

Abstract scheme



Several issues have to be pointed out to study multiprocessor architectures in detail, notably:

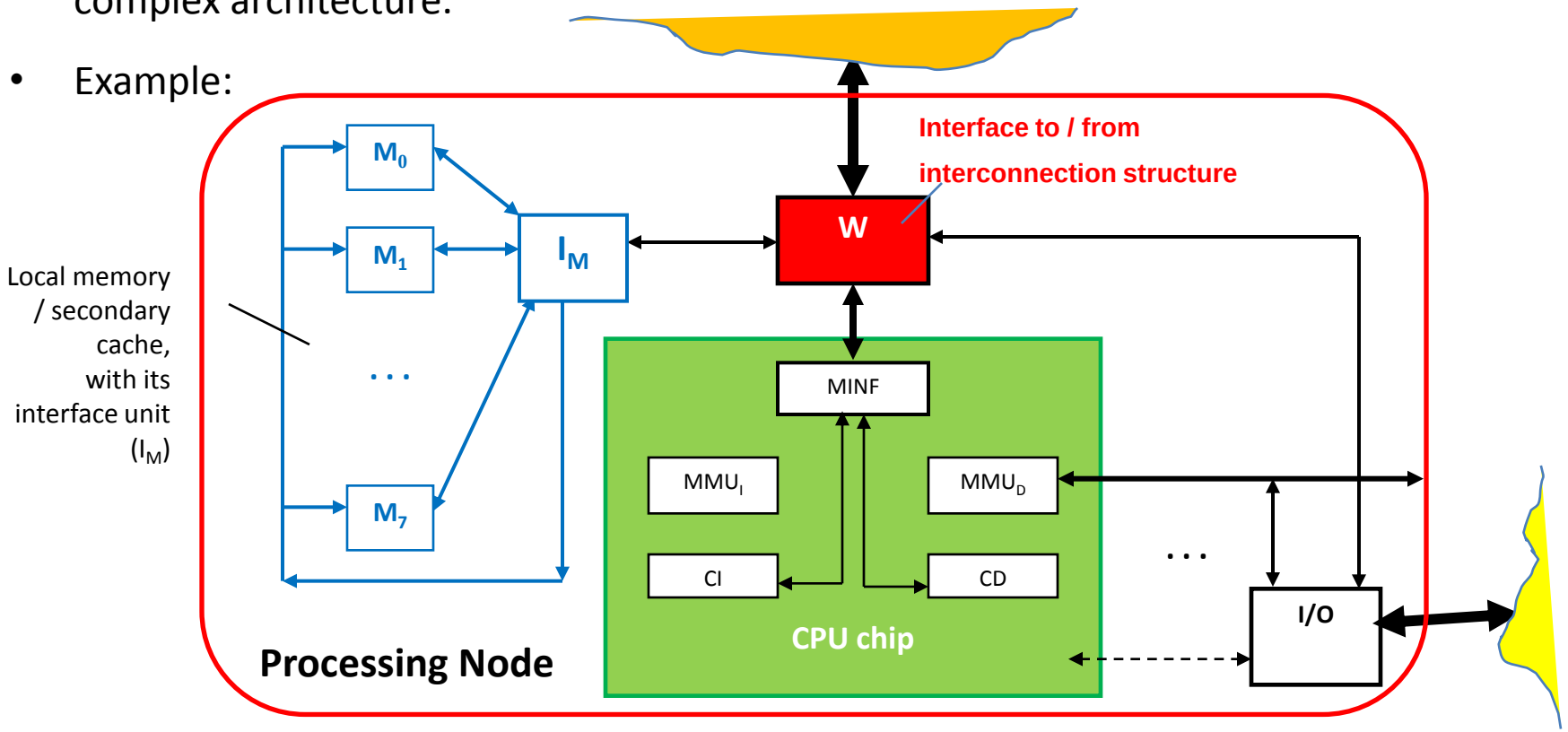
- Processing nodes
- Shared memory organization, and I/O
- Cache management
- Processes-to-processors mapping
- Interconnection structure

Interconnection structure (communication network) between processing nodes and shared memory

- If shared memory has a modular structure, any processor can "reach" any memory module, either *directly* (crossbar network, bus) or *indirectly* (limited degree network)
- Communication networks for multiprocessors are **entirely managed at the firmware level**, i.e., no additional level of protocol exists on top of the primitive firmware protocol (*routing*, *flow control*).

Processing nodes

- We'll assume off-the-shelf CPUs and other standard / existing resources
- Processing nodes may include local memories and I/O
- Proper **interface units** are provided to connect an existing CPU into a more complex architecture.
- Example:



In the example

- Interface unit W is in charge of performing cache block transfers (or other memory accesses), masking to the CPU the structure of the shared memory and communication network
  - Of course, the firmware protocol implemented in the CPU to request a block transfer (memory access) *cannot* be modified wrt the uniprocessor architecture
  - Unit W *adapts* this firmware protocol to the features of the shared memory
    - e.g., individuates the referred memory module, and the path to reach it

    and to the features of the communication network
    - e.g., inserts the CPU request into a message with proper **format** (source, destination node identifier, information about the routing and flow control strategy, etc) and **size** (one or more *packets*, according to the link width of the interconnection network)
- Though the main mode for nodes cooperation is shared memory, some cooperation actions can be done also through direct communications by value
  - in this case, the **I/O subsystem** must be used
  - an I/O unit is in charge of interfacing the direct communications between nodes, adapting the CPU request to the rest of the systems
  - often, **the same interconnection structure** for shared memory is used for direct communications too
  - in the figure: this unit exploits also the DMA mode inside the processing node.

Physical addressing spaces

- Multiprocessor memory as a whole has high capacity
  - e.g., expandable to Tera-words or more
  - physical address of 40 bits or more.
- CPU must be able to generate such *physical* addresses
  - an impact on the design of CPU
  - *not the only one impact case (other meaningful cases will be met)*
  - i.e., though we wish to adopt standard CPUs and other structures, these must be *prone* to some multiprocessor requirements and peculiarities
    - e.g., support of indivisible sequences of memory accessess
    - e.g., cache-coherence
- *Note:* remind that there is no a-priori relation between the size of physical addresses and of logical addresses (e.g. a 32-bit logical address machine)

Classes of multiprocessor architectures

- Two distinctions according to:

1. ***processes-to-processors mapping***
   - dynamic or static correspondence between processes and processors
   - ***anonymous*** *vs* ***dedicated processors***

2. ***organization of modular memory as "seen" by the processors***
   - uniform or non uniform access time to parts (modules) of the shared memory
   - ***uniform memory access*** (**SMP** or **UMA**) *vs* ***non-uniform memory access*** (**NUMA**)

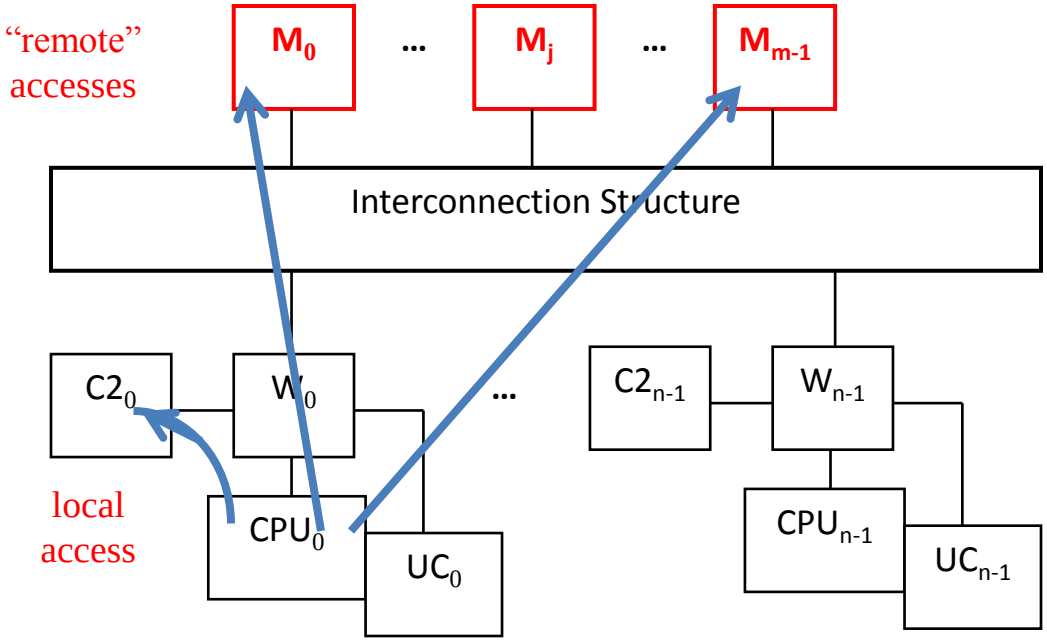
Architecture according to process-to-processor mapping

a. **Anonymous processors** architecture:
   - any process can be executed by any processor
   - in general, when a waiting process is waked-up, its execution is resumed on a different processor
   - **dynamic** mapping according to the *low-level scheduling* functionalities
   - a unique **Ready List** exists for all processors: shared by all processes.

b. **Dedicated processors** architecture:
   - **static** association of disjoint subsets of active processes to processors
   - decided at *loading time*
   - multiprogrammed nodes
   - each node has its own **Ready List**, not shared by processes allocated to other nodes
   - **re-allocation** of processes to nodes can be done sporadically, e.g. for fault-tolerance or load-balancing: conceptually, it is considered static mapping.

Architecture according to memory organization: UMA / SMP

- **UMA** (Uniform Memory Access), also called **SMP** (Symmetric MultiProcessor)
  - the **base** (i.e., measured in absence of conflicts) **memory access latency** of processor P_i to access memory module M_j is constant and independent of i, j.



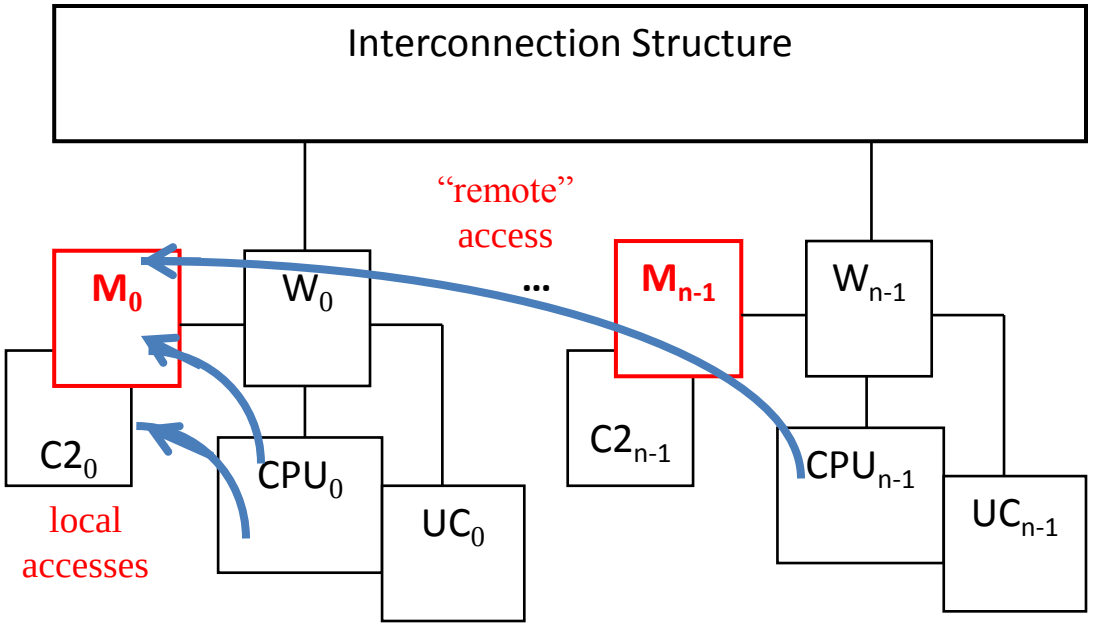
In this example:

- secondary caches are private of the nodes,
- main memory (or tertiary cache) is shared.

In general, shared memory is organized in **macro-modules**, each macro-module being *interleaved* or *long-word*

Architecture according to memory organization: NUMA

- **NUMA** (Non Uniform Memory Access)
  - the **base** (i.e., measured in absence of conflicts) **memory access latency** of processor P_i to access memory module M_j *depends on* i, j.
  - typically: **the shared memory is the union of the all local memories of the nodes** at a certain level of the memory hierarchy.



In this example:

- secondary caches are private of the nodes,
- **local memories of the nodes are all shared**

Every local memory can be interleaved (the shared memory, as a whole, is *not*) or long-word

Variant: **COMA** (Cache Only Memory Access)

Combined features

- According to the distinctions described till now, all the possible combinations are feasible architectures:
  1. anonymous processors + UMA
  2. anonymous processors + NUMA
  3. dedicated processors + UMA
  4. dedicated processors + NUMA
- Most “natural” (most popular) combinations: 1 and 4.
- For this reason, unless otherwise stated, we’ll use the term SMP to denote combination 1, and NUMA to denote combination 4.
- However, combinations 2, 3 are interesting as well.

Exercize

The goal of this exercize is to reason about, and to acquire familiarity with, the features of multiprocessor architectures and memory hierarchies.

Issue to be discussed: memory hierarchies tend to smooth the difference between architectural combinations 1 and 2 (anonymous processors + UMA, anonymous processors + NUMA) and between 3 and 4 (dedicated processors + UMA, dedicated processors + NUMA).

That is, although combinations 1, 4 appear “natural”, the proper use of memory hierarchies could render the other combinations acceptable as well.

Use general reasonings and/or examples about the execution of parallel programs.

Local memories and memory hierachies in multiprocessors

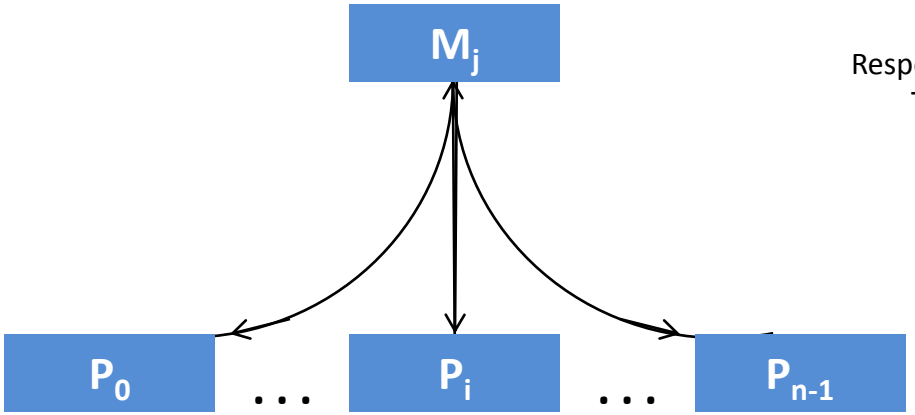
- In multiprocessors the proper use of local memories and memory hierachies is a main issue for performance optimization:
  i. **minimizing the access latency**
  ii. **minimizing the shared memory conflicts**

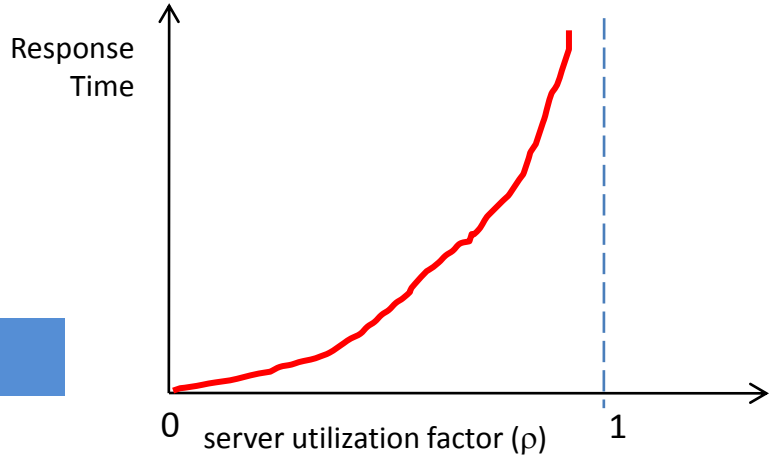
Minimizing the access latency

- Interconnection networks latency is dependent of the number of nodes
  - linear (bus)
  - logarithmic (butterflies, high dimension cubes, trees)
  - square-root (low dimension cubes)
- *Shared memory access are "expensive":*
  - in UMA all accesses to shared memory are "remote",
  - in NUMA some accesses to shared memory are "remote", other ones are local
- UMA goal: try to dynamically allocate useful information in local caches (C1, C2), as usually
- NUMA goal: try to statically allocate useful information in **local memories**, and (as usually) to dynamically allocate local memory information in local caches
  - **in a *dedicated processors* architecture, all the *private* information of processes mapped to the a node are allocated in the local memory of such node** (code, data), as well as some *shared* information
  - **remote accesses: only for (the remaining) *shared* information.**

Minimizing the shared memory conflicts

- The system can be modeled as a queueing system, where
  - processing nodes are clients
  - shared memory modules are servers, including the interconnection structure
  - e.g. M/D/1 queue for each memory module





- The memory access latency is the server Response Time.
- Critical dependency on the server utilization factor: a measure of memory modules congestion, or a measure of processors "conflicts" to access the same memory module
- The interarrival time has to be taken as high as possible: the importance of local accesses, thus the importance of the best exploitation of local memories (NUMA) and caches (SMP).

Some simplified results about multiprocessor performance

- In this section we introduce some initial results about multiprocessor performance evaluation.
- To understand the importance of memory hierarchies, we start with an approximate cost model of an SMP architecture.
- This analysis can be done by evaluating the bandwidth of an interleaved memory.
- Given an interleaved memory with *m* modules, it is statistically reasonable (and it is validated by experiments) that

  probability (processor *i* accesses module *j*) = *1/m*, independently from i, j

- Thus, the distibution of

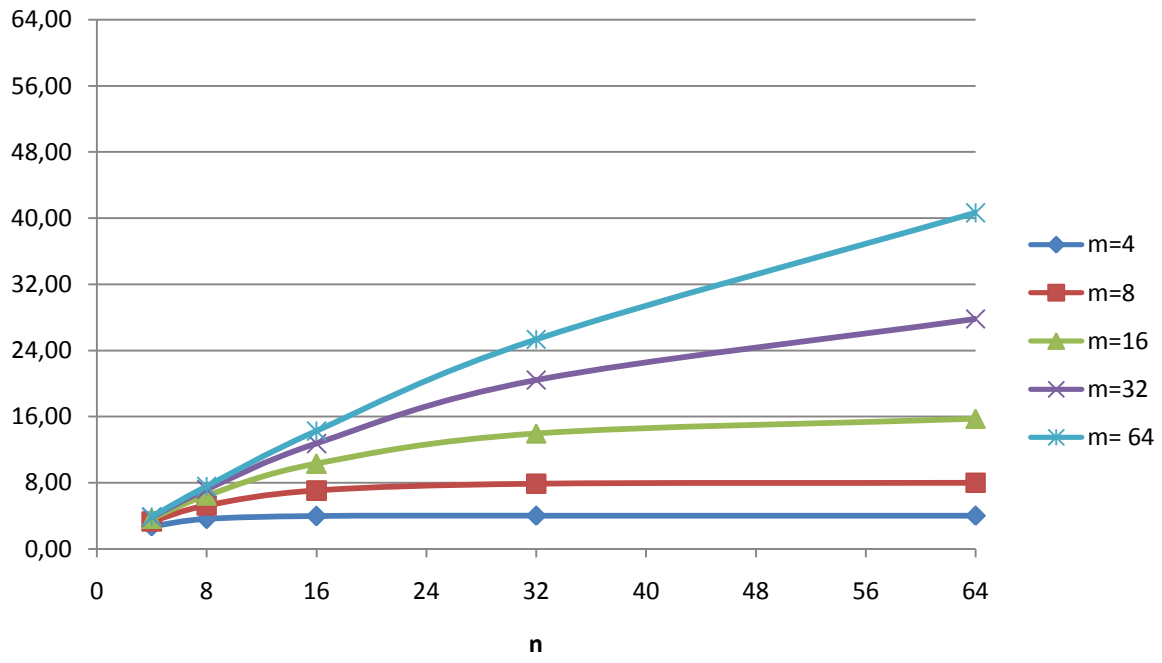
  p(k) = probability (*k* processors | $1 \leq k \leq n$ are in conflict trying to access the same memory module)

  is the binomial distribution. With some manipulations we achieve:

Interleaved memory bandwidth

$$B_M = m\left[1 - \left(1 - \frac{1}{m}\right)^n\right]$$

expressed as average number of shared memory accesses per second, where accesses can be single words or cache blocks. Graphically:



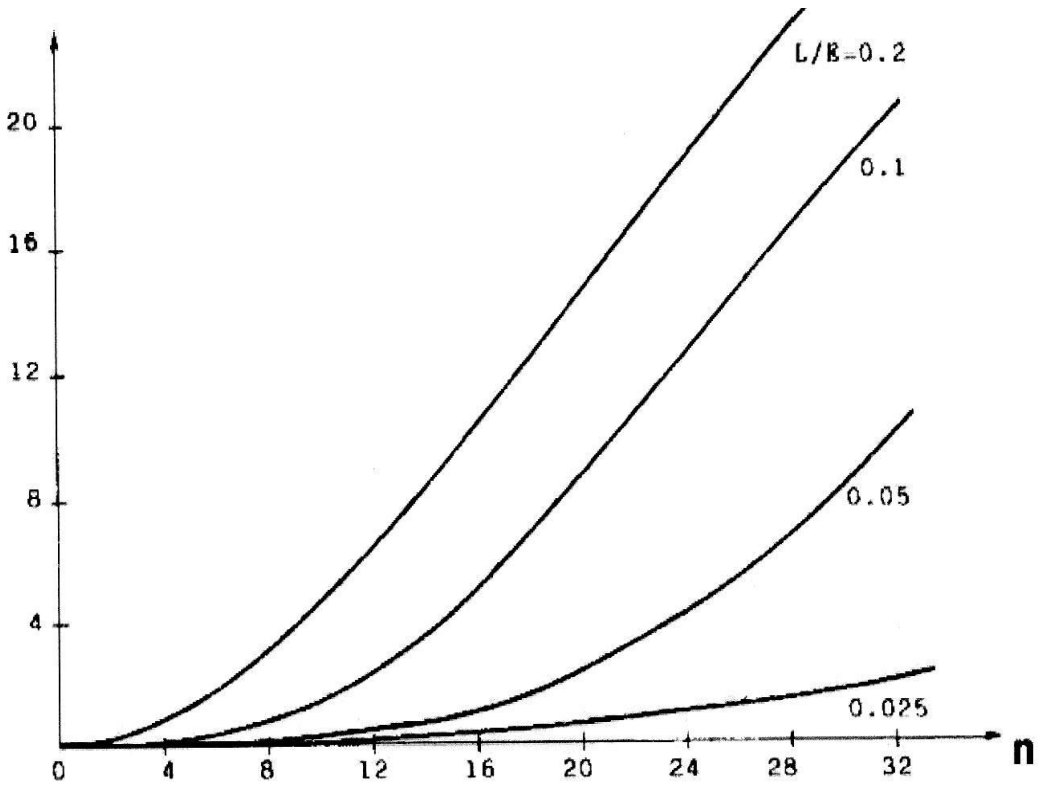
Though not usable for a quantitative analysis of a parallel program performance on a specific real machine,

this result shows qualitatively how the performance degradation due to memory conficts is critical, even with very parallel memories,

unless proper caching techniques are applied to minimize the conflicts.

Software lockout

- Another preliminary result about what we can expect on multiprocessor performance is the average number of processors that are blocked (*busy waiting*) during the execution of indivisible sequences on mutually exclusive - *locked* - shared data structures:



L = average duration of critical sections,

E = average processing time not in critical sections.

For a given L/E, a value n_c exists such that, for $n > n_c$, all the added processors are blocked.

This result shows that also the *critical sections design* is an important issue: for achieving good performances critical sections must be *as short as possible.*

Typical values di L/E for commercial uniprocessor operating systems (> 0.2) show that such OS versions cannot be utilized to build a scalable multiprocessor (they have to be fully re-designed).

Local memories and memory hierachies in multiprocessors

- In multiprocessors the proper use of local memories and memory hierachies is a main issue for performance optimization:
  i. **minimizing the access latency**
  ii. **minimizing the shared memory conflicts**

- Solutions to both issues have a counterpart: the presence of writable shared information in caches introduces the problem of cache coherence
  - how to grant that the contents of distinct caches are consistent.

Cache coherence

Two approaches:

1. cache coherence is mantained automatically by the firmware architecture
2. cache coherence is solved entirely by program (for each application or in the run-time support design), without any firmware dedicated support

(+ intermediate approaches).

"All-cache" architecture? (i.e. *all* information of a process are accessed through caching *vs* some information are not cacheable).

Critical problems for the programmability *vs* performance trade-off.

Intensive debate about the future multicore products.

Subjects to be studied

1. Interconnection structures, and memory access latency
2. Multiprocessor run-time support of concurrency mechanisms (interprocess communication), including cache coherence, and interprocess communication cost model