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

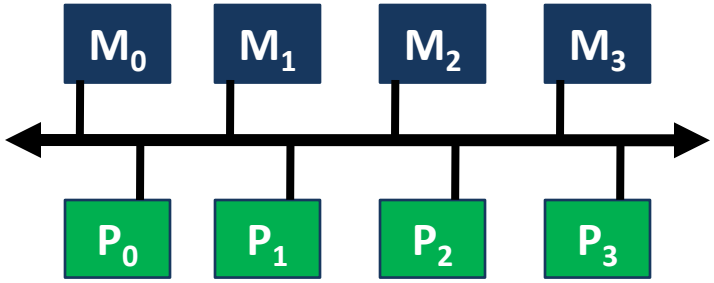
4. Shared Memory Parallel Architectures

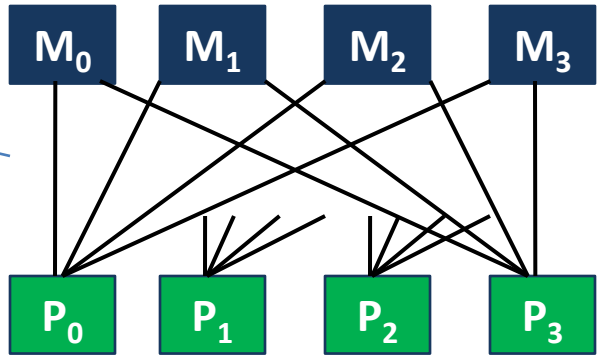
4.2. Interconnection Networks

Kinds of interconnection networks

- Main (contrasting) requirements:
  1. High connectivity
  2. High bandwidth, low latency
  3. Low number of links and limited pin count
- Two extreme cases:
  - **Bus** (**shared link**)
    - Minumum cost and pin count
    - No parallel communications
    - Latency O(n)
  - **Crossbar** (**full interconnection**)
    - Maximum cost $O(n^2)$ and pin count
    - Maximum parallelism
    - Minumum constant latency

For opposite reasons, bus and crossbar are not valid solutions for highly parallel architectures

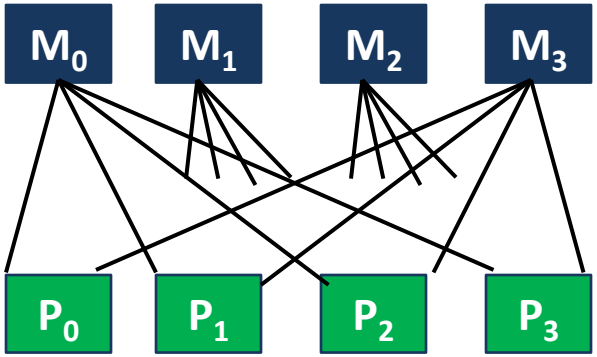




Limited degree networks

- Dedicated links only (no buses)
- Reduced pin count
  - network degree = number of links per node
- Base latency O(log N), or O($N^{1/k}$)
  - Note: *O(log N)* is the best result for structures with *N* arbitrarily large
  - **base latency** = latency in absence of conflicts. **Goal: mantain O(log N) latency in presence of conflicts too.**
- High bandwidth
  - Service time is not significantly increased wrt crossbar

Basic idea:
consider a crossbar-like structure composed of *m* **trees** having the memory modules in the respective roots (latency: O(log n))



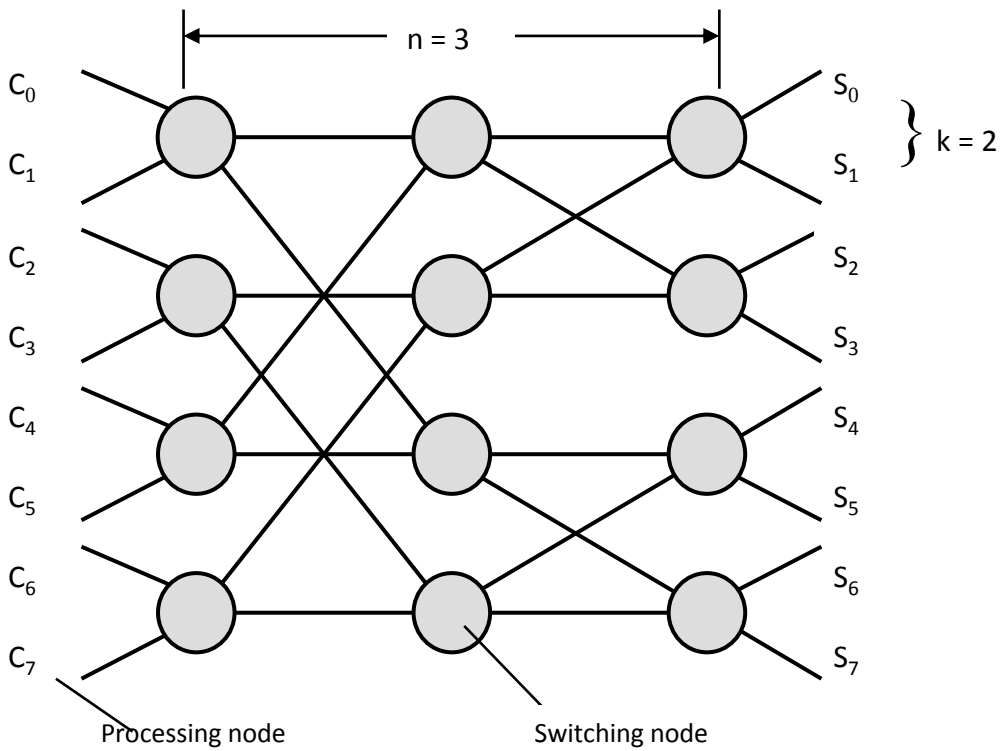
Merge all the trees in a limited degree structure, i.e. some sub-structures are in common to some trees.
Possibly, apply topologycal transformations (e.g. transform trees into cubes).

Kinds of limited degree networks

- Indirect / multistage networks
  - Paths between nodes consist of **intermediate nodes**, called **switching nodes** (or simply **switches**). *Any* communication requires a routing strategy through a path composed of switching nodes.
  - Typical examples:
    - **Butterfly (*k-ary n-fly*):** for networks connecting two distinct sets of nodes (processing nodes and memory modules in SMP)
    - **Tree, fat tree, generalized fat tree:** for networks connecting the same kind of processing nodes (NUMA)
- Direct networks
  - No intermediate switching nodes. *Some processing nodes are directly connected* to other nodes (of the same kind, e.g. NUMA), in this case no routing is required. Routing strategies are applied to nodes that are not directly connected.
  - Typical examples:
    - **Ring**
    - **Multidimensional mesh** (***k-ary n cube***)
- Direct networks and the varieties of trees are applied to **distributed memory architectures** as well.

k-ary n-fly networks

- Butterflies of *dimension* *n* and *ariety* *k*.
- Example: *2-ary 3-cube* for 8 + 8 nodes:



Number of processing nodes:

$$N = 2\,k^n$$

e.g., k^n processors (C_i), k^n memory modules (S_j) in SMP.

Network degree = **2k**.

Network distance (constant)**:**

$$d = n$$

proportional to the **base latency**.

Thus, base latency = **O(log N).**

There is **one and only one unique path** between any C_i and any S_j. It is exploited in **deterministic routing**.

Formalization: binary butterfly (2-ary n-fly)

- Connects 2^n processing nodes to 2^n processing nodes, through n levels of switching nodes
- Processing nodes are connected to the first and the last level respectively.
- Each level contains 2^{n-1} switching nodes
- Total number of switching nodes = $n\ 2^{n-1} = O(N \log N)$
- Total number of links = $(n - 1)\ 2^n = O(N \log N)$
  - *wrt $O(N^2)$ of crossbar.*

Note: the FFT (Fast Fourier Transform) algorithm has a binary butterfly topology.

It reduces the complexity of the Discrete Fourier Transform from $O(N^2)$ to $O(N \log N)$ steps. The data-parallel implementation of FFT has completion time $O(\log N)$ with $O(N)$ virtual processors. The stencil at *i-th* step is exactly described by the topology of the butterfly *i-th* level.

Formalization: binary butterfly (2-ary n-fly)

Connectivity rule:

- Each switching node is defined by the coordinates (x, y), where
  - $0 \leq x \leq 2^{n-1}$ is the *row* identifier,
  - $0 \leq y \leq n-1$ is the *column* identifier.
- Generic switching node (i, j), with $0 \leq j \leq n-2$, is connected to two switching nodes defined as follows:
  - $(i, j + 1)$ through the so called "straight link"
  - $(h, j + 1)$ through the so called "oblique link", where h is such that

$$abs\,(h - i) = 2^{n-j-2}$$

  i.e. the binary representation of h differs from the binay representation of i only for the j-*th* bit starting from the most significant bit.
- The connectivity rule defines the deterministic, minimal routing algorithm.

Formalization: binary butterfly (2-ary n-fly)

Recursive construction:

- for $n = 1$: butterly is just one switching node with 2 input links and 2 output links;
- given the n dimension butterfly, the $(n+1)$ dimension butterfly is obtained applying the following procedure:
  - two n dimension butterflys are considered, "one over the other", so obtaining the n final levels, each of them composed of 2^n switches;
  - a level of 2^n switches is added at the left;
  - the first level switches are connected to the second level switches according the *connectivity rule*, in order to grant the full reachability of the structure.
- Formal transformations of binary butterflies into binary *hypercubes* are known.
- Formalization can be generalized to any ariety k.
- Many other multistage networks (Omega, Benes, and so on) are defined as variants of *k-ary n-fly*.

Routing algorithm for k-ary n-fly

The **deterministic, minimal algorithm** derives from the connectivity rule. For *k = 2*, consider the binary representation of the source and destination processing node *identifiers*, e.g. C_3 = (011) and S_6 = (110). These identifiers are contained in the message sent through the network.

- Once identified the first switch, the algorithm evolves through *n* steps. At *i-th* step the switch compares the *i-th* bit of source and destination identifiers, starting from the most significant bit: if they are equal the message is sent to the *straight* link, otherwise to the *oblique* link.
- The last switch recognizes the destination just according to the least significant bit of destination identifier.
- For routing in the opposite direction (from S to C), the binary identifiers are analyzed in reverse order, starting from the *second* least significant bit.

The algorithm is generalized to any *k.*

Other non-minimal routing strategies can be defined for k-ary n –fly networks, notably **adaptive routing** according to *network load* and/or *link availability.*

k-ary n-cube networks

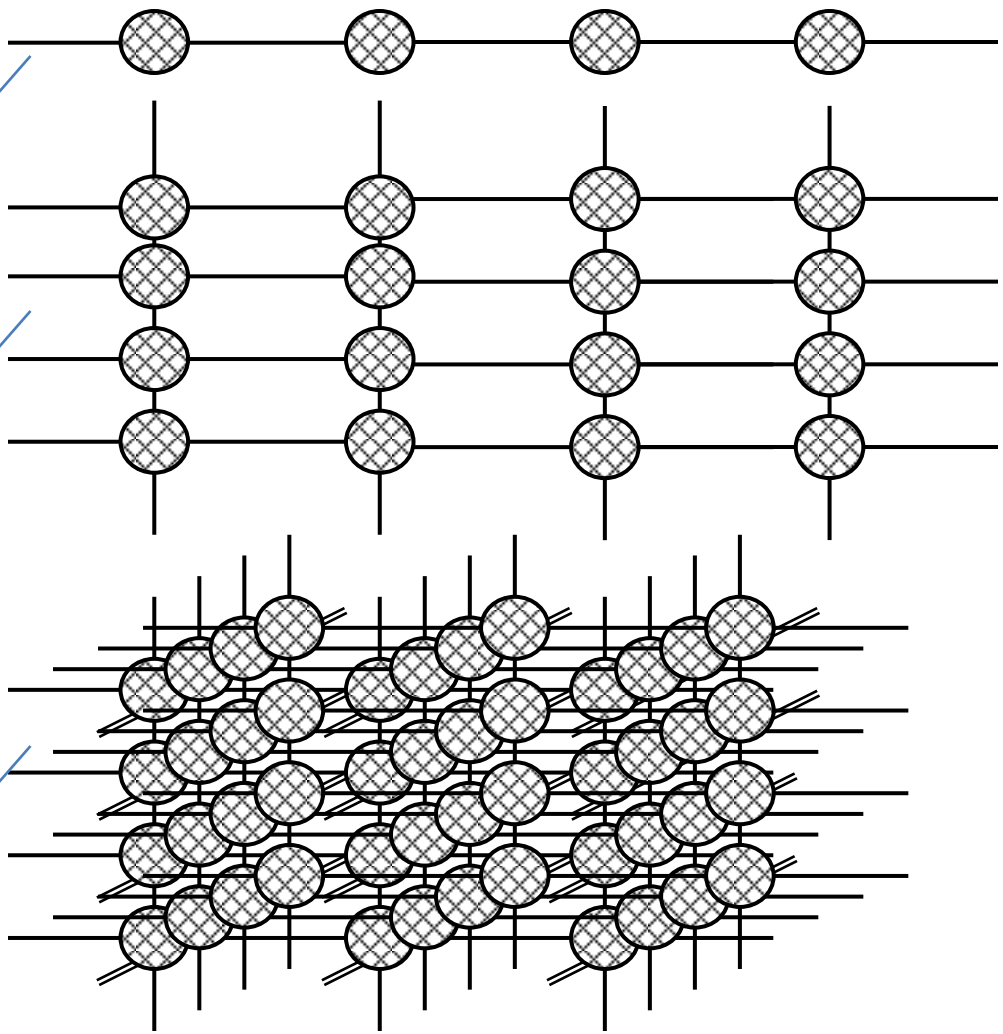
- Cubes of *dimension n* and *ariety k*.
- Generalization of *ring* (**toroidal**) structures.

k = 4, n = 1

k = 4, n = 2

No switch nodes. In principle, the processing nodes are the nodes of the network structure. In practice, the network nodes are *interfaces* of the processing nodes.

k = 4, n = 3



k-ary n-cubes

- Number of processing nodes: $N = k^n$.
- Network degree = $2n$.
- Average distance = $O(k\ n)$: proportional to base latency.
- For a given N (e.g. N = 1024), we have two typical choices:
  - n as low as possible (e.g. *32-ary 2-cube*)
  - n as high as possible (e.g. *2-ary 10-cube*)
- For low-dimension cubes: distance = $O(N^{1/k})$
- For high-dimension cubes: distance = $O(\log N)$
- *Despite this difference in the order of magnitude, the detailed evaluation of latency is in favour of low-dimension cubes:*
  - *High-dimension cubes* are critical from the **pin count and link cost** viewpoint: in practice, they are forced to use few-bit parallel links (*serial links* are common). This greatly increases the latency value (multiplicative constant is high).
  - *Low-dimension cubes* (***n = 2, 3***) are more feasible structures, and their latency tends to be lower (low values of the multiplicative constant) for many parallel programs written according to the known forms of parallelism (farm, data-parallel).

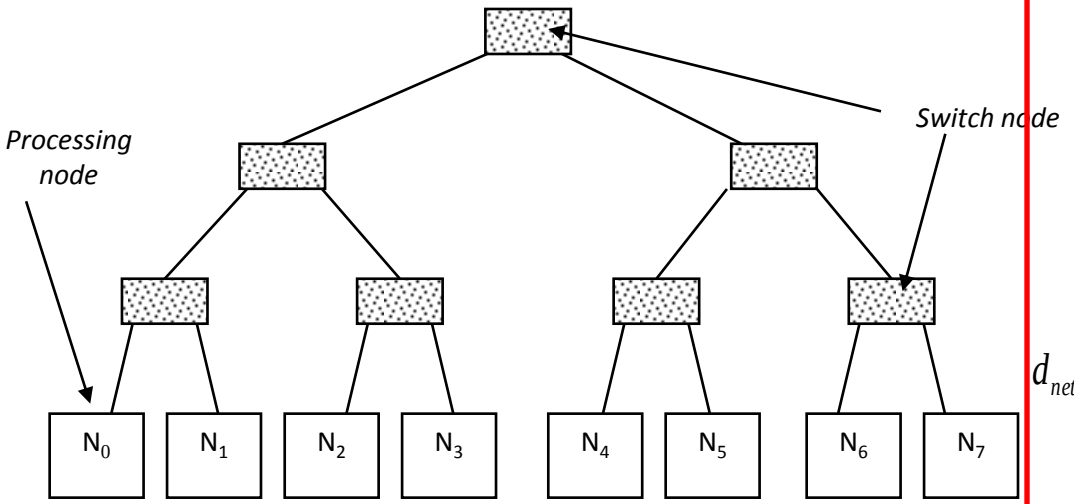
k-ary n-cubes

- Deterministic routing: dimensional
  - Example for *k = 2*: Sorce = (x1, y1), Destination (x2, y2)
  - Routing steps along the first dimension from (x1, y1) to (x2, y1)
  - Routing steps along the second dimension from (x2, y1) to (x2, y2)

- Application of k-ary n-cubes: NUMA

Trees

Consider a binary tree multistage network connecting *N* nodes of the same kind as *leafes* of the tree:



Ariety *k* (e.g. k = 2)
Dimension *n*
Number of nodes: $N = k^n$
Average distance:

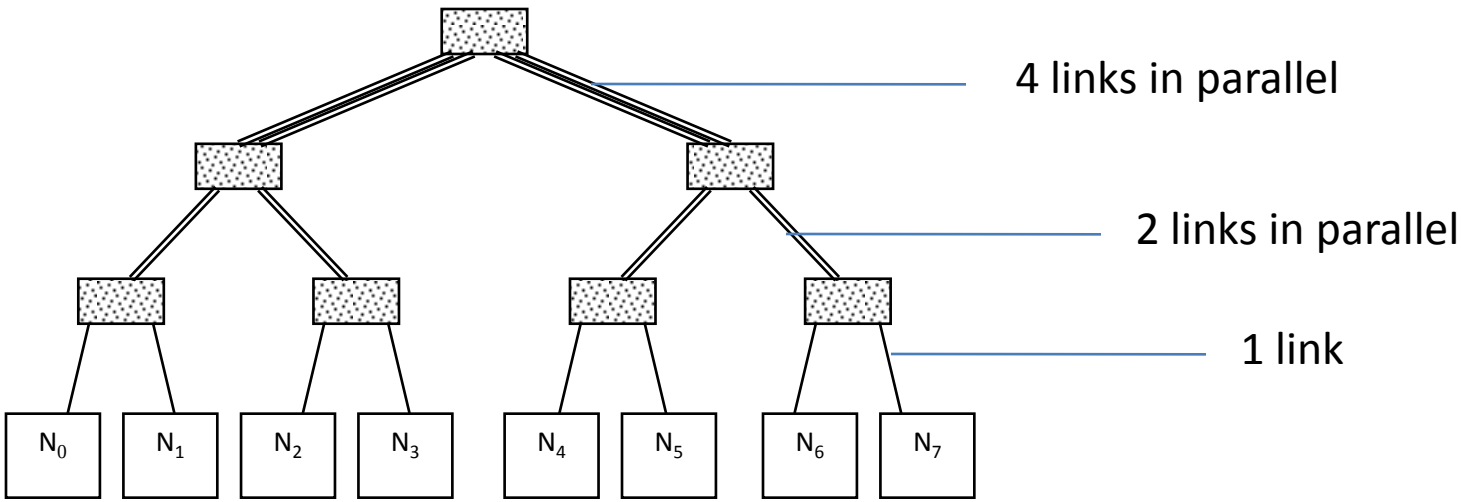
$$d_{net} = \frac{\sum_{i=1}^{n} \frac{N}{2^i} 2(n-i+1)}{N-1} + 1 = \frac{\sum_{i=1}^{n} (n-i+1)\, 2^{(n-i+1)}}{2^n - 1} + 1$$

$$d_{net} \sim 1{,}9 * n$$

Routing algorithm: tree visit

Base latency = *O(log N)*.
However, the number of conflicts is relatively high: as higher as the distance increases.
The problem can be solved by *increasing the bandwidth (i.e. parallelism of links)* as we move from the leaves towards the root: we obtain the so called *FAT TREE* network.

Fat trees



Links in parallel are used as **alternative links** for distinct messages flowing between the same switching nodes, thus reducing the conflicts.

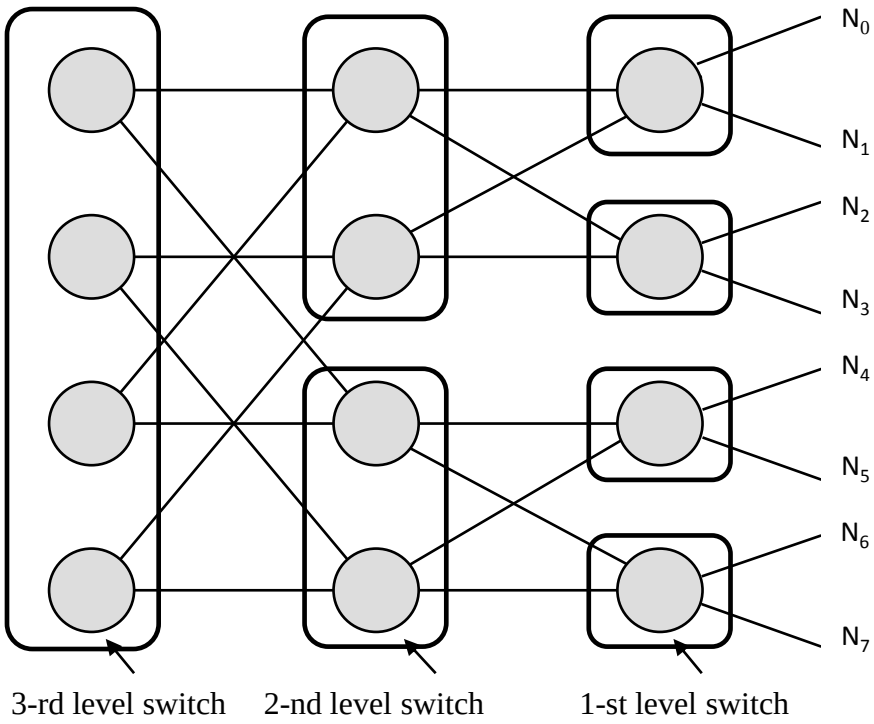
If the bandwidth doubles at each level from the leafs to the root, then the **conflict probability becomes negligible**,

provided that the switching node has a parallel internal behaviour.

The solution to the pin count and link cost problem, and to the switching node bandwidth problem, is the Generalized Fat Tree, based on k-ary n-fly structures.

Generalized Fat Tree

The requirement is that the *i-th* level behaves as a 2^i x 2^i crossbar, without pin count problems for high value of *i*. These *limited degree* crossbars are implemented by a *k-ary n-fly* structure with a single set of processing nodes:



Routing algorithm: tree routing algorithm + dynamic adaptivity to link availability.

Interesting network,
also because it can be **exploited, at the same time, as a Fat Tree and as a butterfly itself,**
provided that the switching node implements *both* routing algorithms.

Typical application: in SMP as processor-to-memory network (butterfly) and as processor-to-processor network (Fat Tree).

Basic scheme for the most interesting networks: Myrinet, Infiniband.

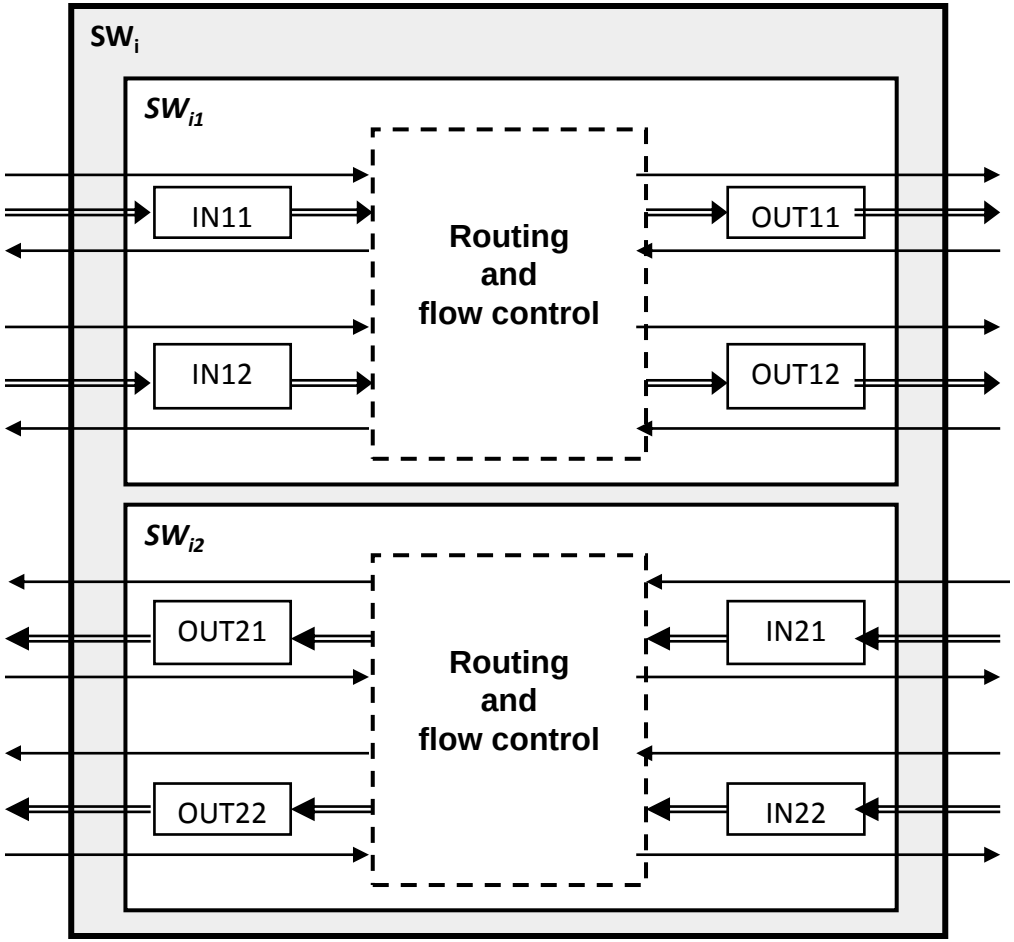
Flow control of interconnection networks for parallel architectures

- Flow control techniques: management of networks resources, i.e. links, switching nodes, internal buffers, etc.
- Packet-based flow control: at each switching node of the routing path, the whole packet must be received, and buffered, before it is sent towards the next switching node.
- Packets are the routing units, and distincts packets can follow distinct paths in parallel.
- Especially in parallel architectures, the single packet transmission can be further parallelized: **wormhole** flow control strategy:
  - packets are decomposed into **flits**,
  - flits of the same packet are propagated **in pipeline**, thus reducing the latency from *O(path length ∗ message size)* to *O(path length + message size)*, provided that this fine grain parallelism is efficiently exploited by the switching nodes.

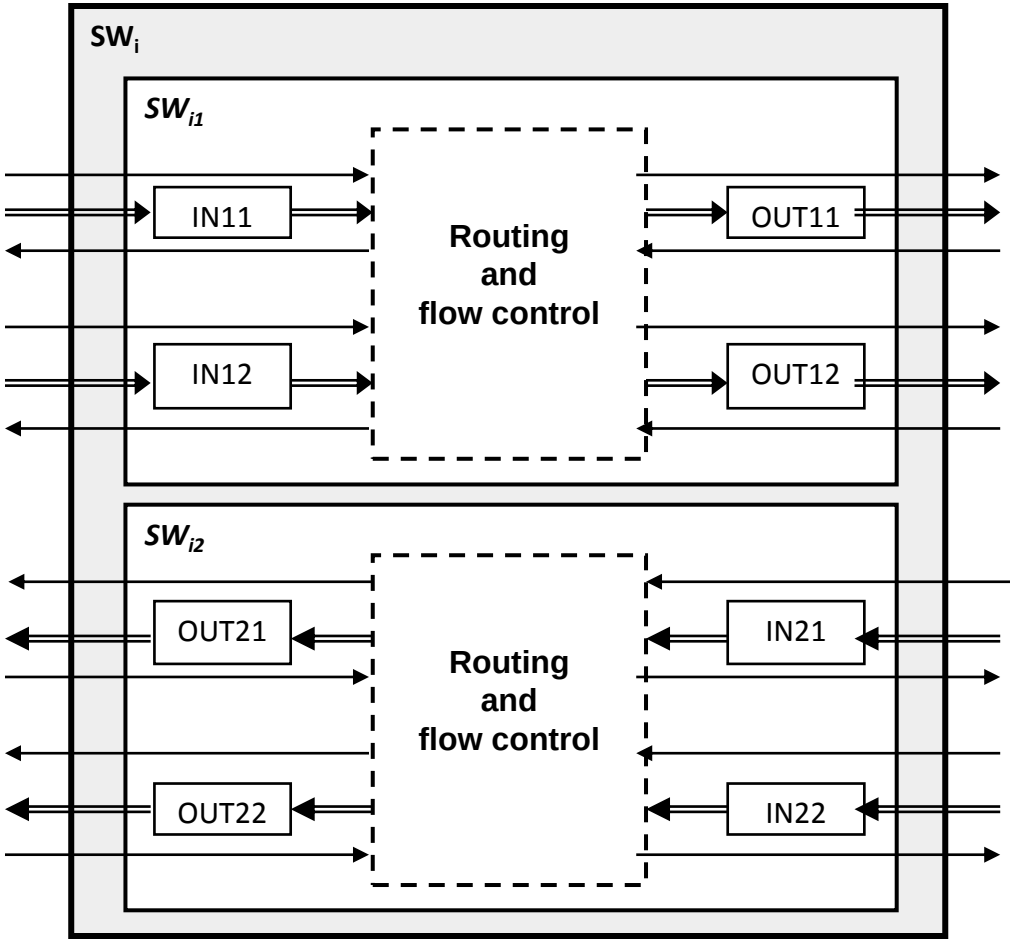
Wormhole flow control

- The routing unit is the packet as well: all the flits of the same packet follow exactly the same path.
- The *first flit* must contain the *routing information (packet heading)*.
- In general, with *w*-bit parallel links (e.g. w = 32 bit), the flit size is w bits (possibly, *2w* bits).
- The minimum *buffering capacity* inside a switching node is one flit per link (instead of a packet per link): this contributes to the very efficient firmware implementation of switching nodes (one clock cycle service time *and* internal latency).
- Implemented in most powerful networks.

Implementation of a wormhole switching node



Implementation of a wormhole switching node



Dedicated links with level-transition RDY-ACK interfaces (see firmware prerequisites)

Distinct units for the two network directions.

At every clock cycle, control the presence of incoming messages and, for heading-flits, determine the output interface according to the routing algorithm.
The heading-flit is sent if the output interface has not been booked by another packet.

Once the heading-flit of a packet has been sent to the proper output interface, then the rest of the packet follows the heading-flit.

Adaptive routing is implemented according to ACK availability (and possibly time-outs).

Exercize

Assume that a wormohole Generalized Fat Tree network, with ariety $k = 2$ and $n = 8$, is used in a SMP architecture with a double role: a) processor-to-memory interconnection, and b) processor-to-processor interconnection.

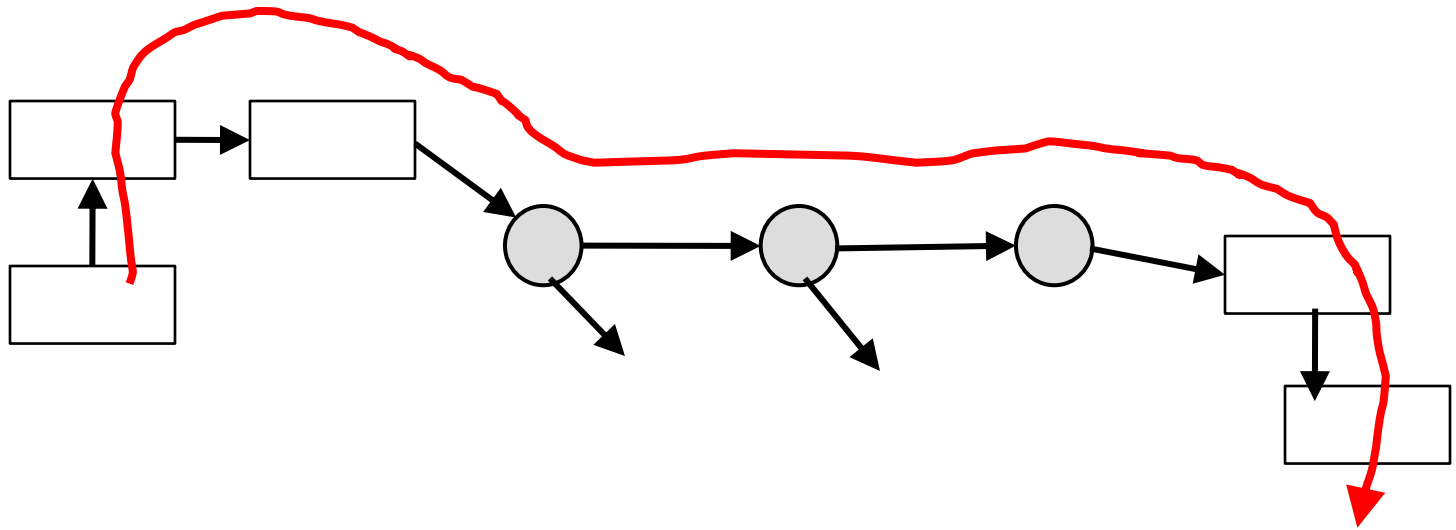
The firmware messages contain in the first word: routing information (source identifier, destination identifier), message type (a or b), message length in words.

Link and flits size is one word.

Describe the detailed (e.g., at the clock cycle grain) behaviour of a switching node.

Latency of communication structures with pipelined flow control

Pipelined communications occur in structures including wormhole networks and other computing structures (interface units, interleaved memories, etc).

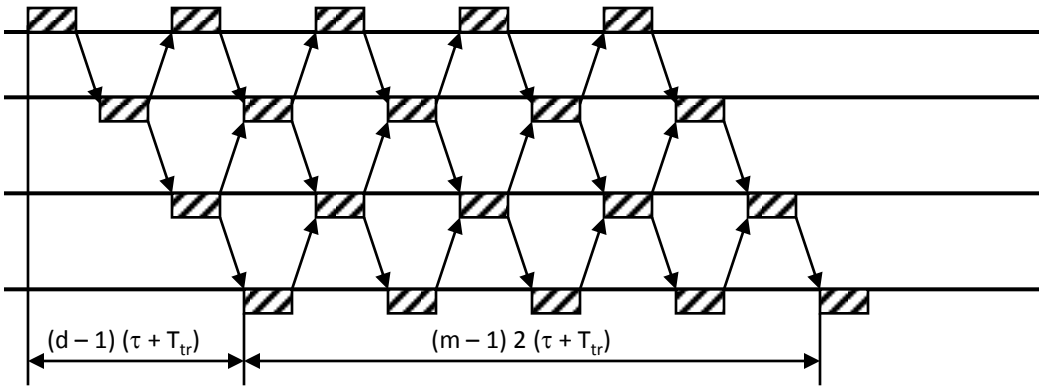


We wish to evaluate the latency of such a pipelined structure with ***d*** *units* for firmware messages of *length* ***m*** *words.*

Let's assume that

- wormhole *flit* is equal to a word,
- every units has clock cycle τ,
- every link has transmission latency T_{tr}.

Latency of communication structures with pipelined flow control

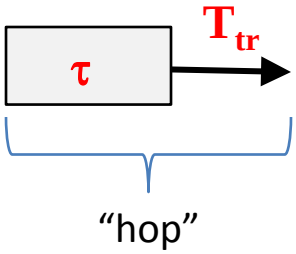


Example with d = 4 units and m = 5 words per message (packet)

$$T_{lat\text{-}pipe} = (m - 1)\, 2\, (\tau + T_{tr}) + (d - 1)\, (\tau + T_{tr}) = (2m + d - 3)\, (\tau + T_{tr})$$

Let's denote:

$$t_{hop} = \tau + T_{tr}$$



called "hop latency" (latency of a single "hop"):

$$T_{lat\text{-}pipe} = (2m + d - 3)\, t_{hop}$$

Memory access latency evaluation

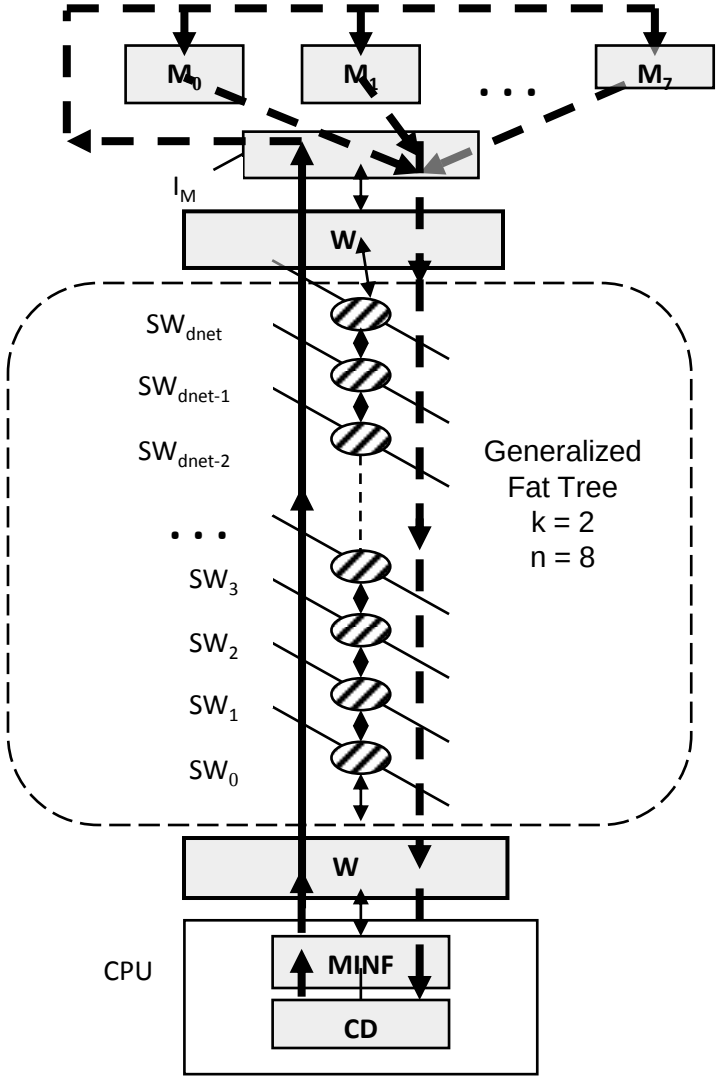
Assumptions:

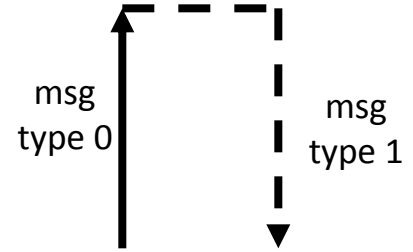
- All-cache architecture
- Cache block size σ
- Shared main memory, each memory macro-module is σ-way interleaved (or σ long word)
- Wormhole, *n* dimension, binary *Generalized Fat Tree* network
- Link size = flit size = one word
- D-RISC Pipelined CPU

Let's evaluate:

- *Base latency* of a cache block transfer (absence of conflicts)
- *Under-load latency* of a cache block transfer (presence of conflicts)

Scheme of the system to be evaluated





Request of a remote block transfer
(firmware msg 0) : **latency** T_0
and
block transfer
(firmware msg 1): **latency** T_1

Base latency for a remote block transfer:

$$t_{a.\text{-}base} = T_0 + T_1$$

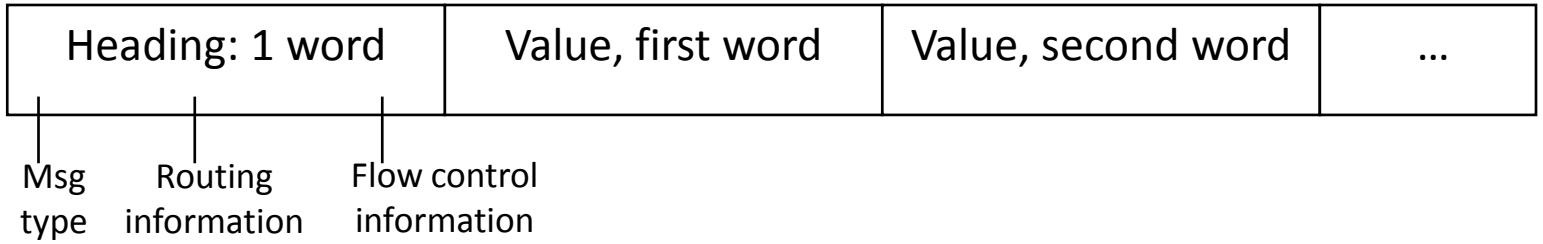
Let's denote:

d_{net} = *average network distance*

(e.g., with n = 8, d_{net} = 15 in the worst case, d_{net} = 8 in the best case)

Firmware messages

Message format: sequence of words



Heading is inserted or removed by the interface unit W between CPU and network:

4 bits: message type
8 bits: source node identifier
8 bits: destination node identifier; in this case, it is a memory macro-module, identified by the least 8 bits of physical address (in SMP), or the most 8 significant bit (in a NUMA architecture) – example for 256 nodes and memory macro-modules
8 bit: message length (number of words)
4 bits: other functions

Message types for this example:

Msg 0: request of a block transfer to the remote memory
Msg 1: block value from the remote memory

Firmware messages and their latencies

Msg 0:

- Value: Physical address relative to the memory module: e.g. 1 word
- Message length, including heading: $m = 2$ words

It propagates *in pipeline* through CPU-MINF, W_{CPU}, network, W_{MEM}, Memory Interface: distance

$$d = d_{net} + 4$$

Latency of message 0 (applying the pipelined communication formula):

$$T_0 = (2m + d - 3)\, t_{hop} = (5 + d_{net})\, t_{hop}$$

Firmware messages and their latencies

Msg 1:

- Value: Access outcome control information: 1 word
- Value: Block value: σ words
- Message length, including heading: $m = 2 + \sigma$ words

This message implies the reading from the interleaved memory module, which includes the following *sequential* paths:

- request from Memory Interface to modules: t_{hop}
- access in parallel to the s interleaved modules: τ_M
- σ words in parallel from modules to Memory Interface: t_{hop}

At this point, the result (outcome, block value) flows *in pipeline* through Memory Interface, W_{MEM}, network, W_{CPU}, CPU-MINF: distance

$$d = d_{net} + 4$$

Latency of message 1 (applying the pipelined communication formula):

$$T_1 = 2\, t_{hop} + \tau_M + (2m + d - 3)\, t_{hop} = 2\, t_{hop} + \tau_M + (2\sigma + 5 + d_{net})\, t_{hop}$$

Base latency

As a result, the base access latency is given by:

$$t_{a\text{-}base} = T_0 + T_1 = [\, c + 2(\sigma + d_{net}) \,]\; t_{hop} + \tau_M$$

where c is a system-dependent constant (= 12 in this example).

For a binary Fat Tree network:

$$d_{net} = \delta n$$

where $n = log_2 N$, and δ is an application-dependent parameter in the range

$$1 \leq \delta < 2$$

according to the locality of the internode communications, thus depending on the allocation of application processes.

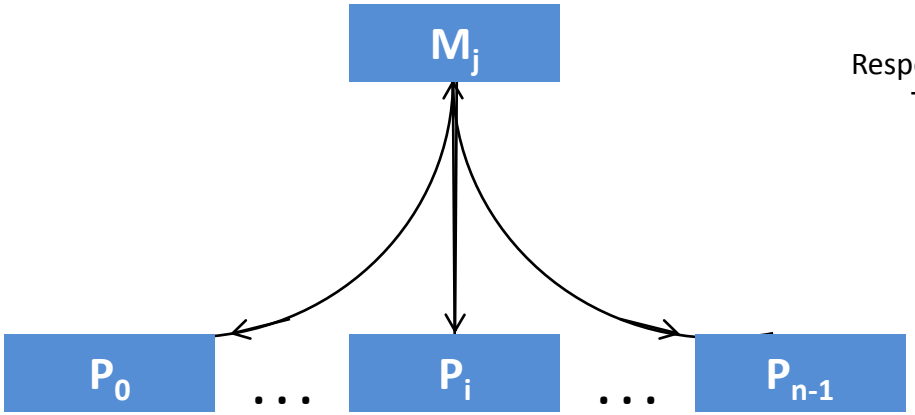
For example, with $d_{net} = 15$, $\sigma = 8$, $t_{hop} = 5\tau$, $\tau_M = 50\tau$: $t_{a\text{-}base} = 290\tau + 50\tau = 340\tau$.

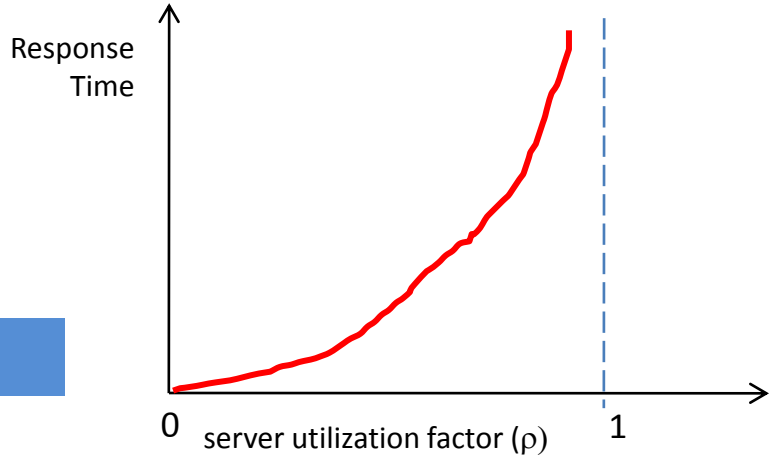
With $d_{net} = 8$, $\sigma = 8$, $t_{hop} = 5\tau$, $\tau_M = 50\tau$: $t_{a\text{-}base} = 240\tau + 50\tau = 290\tau$.

Even with a rather slow memory, the impact of the network latency on memory access latency is the most meaningful.

Under-load latency: queueing modeling

- The system is 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 low as possible: the importance of local accesses, thus the importance of the best exploitation of local memories (NUMA) and caches (SMP).

Under-load latency: queueing modeling

- Each server has an average number p of clients, where p is a parameter ($\leq N$) representing the average number of processing nodes that share the same memory module.
- In a SMP architecture, according to the costant probability that any processor accesses any memory macro-module:

$$p = N / m$$

- In a NUMA architecture, p is application-dependent, though

$$p < N$$

especially for structured parallel programs that are characterized by some communication locality.

Under-load latency: queueing modeling

- With a Fat Tree network, it is acceptable to assume that, especially for small p, the the effect of conflicts over networks links is negligible compared to the effect of conflicts on memory modules.
- Let T_p be the average processing time of the generic processor between two consecutive accesses to remote memory. Let's assume that the corresponding random variable is exponentially distributed.
- The memory access time, i.e. the server response time R_Q, is the solution of the following system of equations (see the client-server modeling in the ILP part of the course):

Under-load latency: queueing modeling

$$\begin{cases} T = T_p + R_Q \\ R_Q = W_Q(\rho, T_s, \sigma_s) + t_{a\text{-}base} \\ \rho = \dfrac{T_s}{T_A} \\ T_s = t_{a\text{-}base} \\ T_A = \dfrac{T}{p} \\ \rho < 1 \end{cases}$$

A good approximation for W_Q is given by the M/D/1 queue:

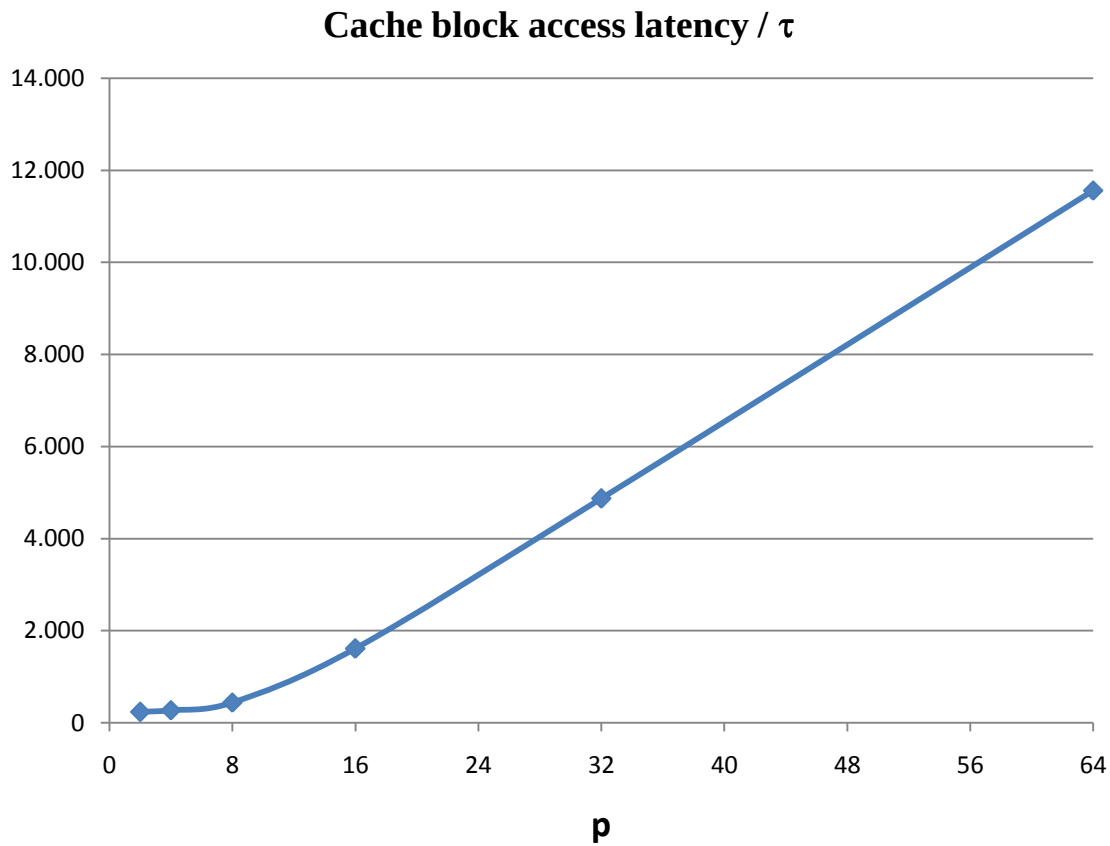
$$W_Q = T_s \frac{\rho}{2(1-\rho)}$$

R_Q is the one and only one real solution of a second degree equation with real coefficients.

In the following, the evaluation will be expressed graphically according to the parameters: p, T_p, N, σ, δ, t_{hop}, τ_M.

Under-load latency: evaluation

***p* variable**; $\sigma = 8$, $\delta = 1$, $t_{hop} = 5\ \tau$, $\tau_M = 10\ \tau$, $N = 64$, $T_p = 2000\ \tau$: $n = 6$, $t_{a\text{-}base} = 210\ \tau$



p is one of the most critical parameters.

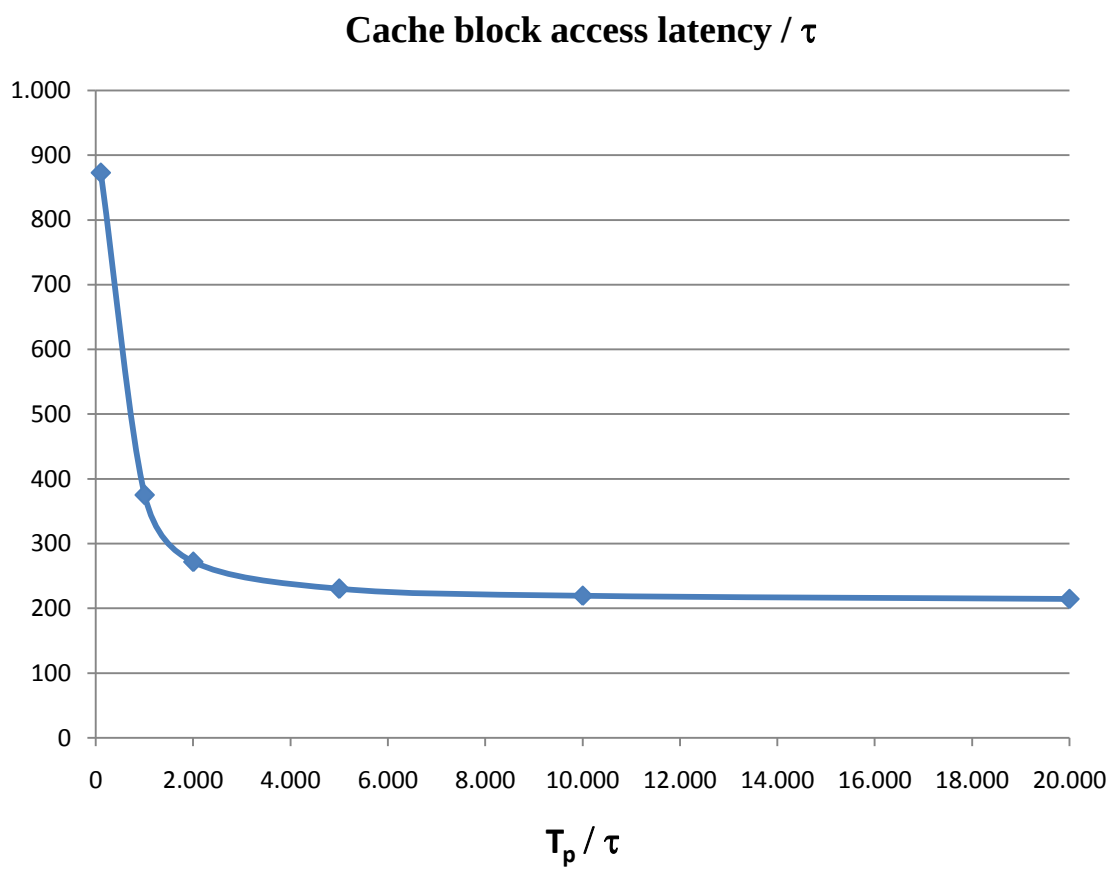
Good performances are achieved for $p < 8$:

in NUMA architectures if applications have high communication – locality,

in SMP for high ratio N/m.

Under-load latency: evaluation

T_p variable; σ = 8, δ = 1, t_{hop} = 5 τ, τ_M = 10 τ, N = 64, p = 4: *n = 6, $t_{a\text{-}base}$ = 210 τ*



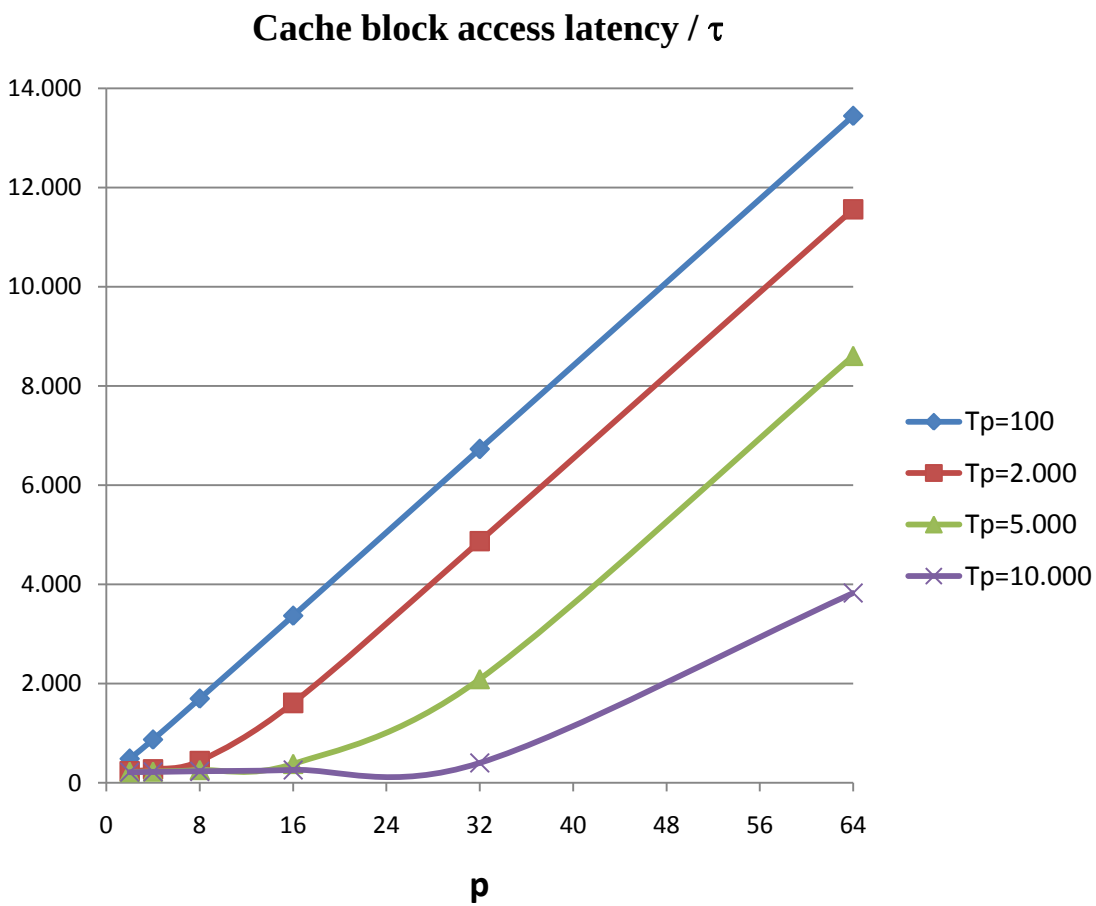
T_p is a meaningful parameter as well.

For *coarse grain* applications, R_Q tends to the $t_{a\text{-}base}$ value, i.e. the memory conflicts are negligible.

For *fine grain* applications, memory conflicts have a relevant negative impact.

Under-load latency: evaluation

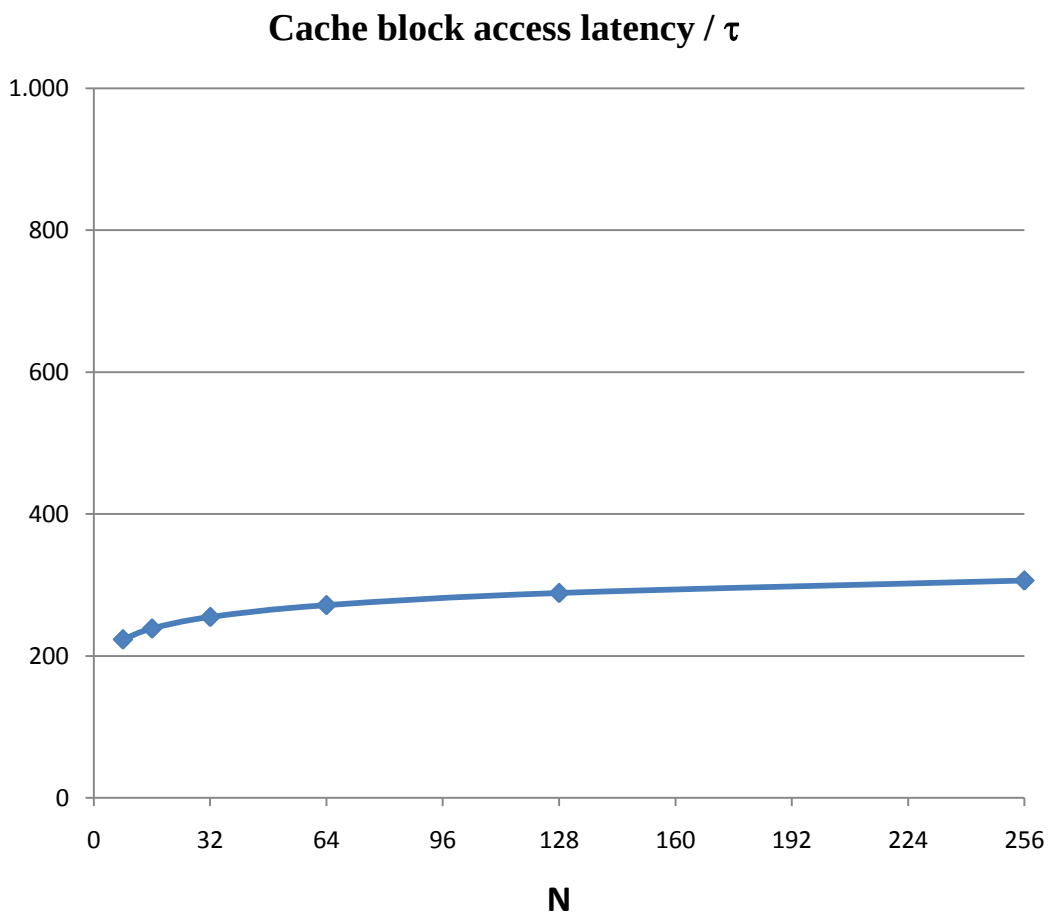
***p, T_p* variables**; **σ = 8, δ = 1, t_{hop} = 5 τ, τ_M = 10 τ, N = 64**: *n = 6, $t_{a\text{-}base}$ = 210 τ*



Combined effect of *p* and T_p.

Under-load latency: evaluation

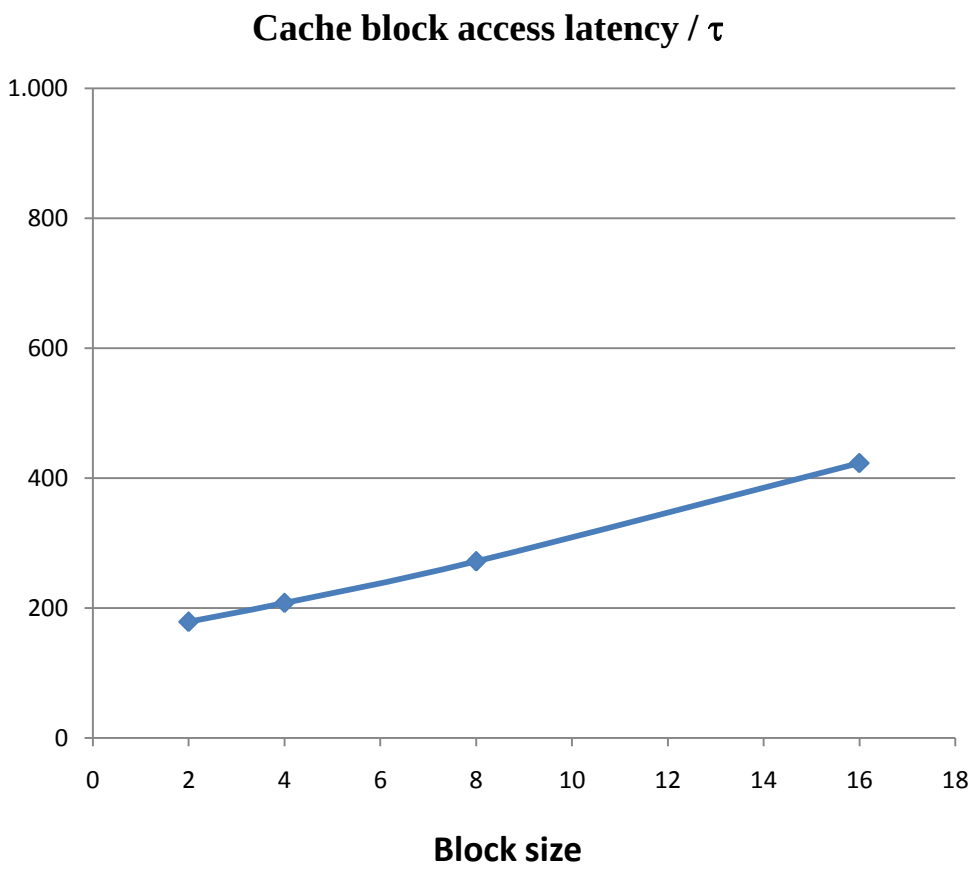
***N* variable**; **σ = 8, δ = 1, t_{hop} = 5 τ, τ_M = 10 τ, N = 64, p = 4, T_p = 2000 τ**: *for $8 \le N \le 256$, $3 \le n \le 8$, $180\tau \le t_{a\text{-}base} \le 230\tau$.*



In this evaluation, the true parallelism degree is *p*, not *N*.

Under-load latency: evaluation

<u>σ variable</u>; δ = 1, t_{hop} = 5 τ, τ_M = 10 τ, N = 64, p = 4, T_p = 2000 τ: *for $2 \le \sigma \le 16$, $150\tau \le t_{a\text{-}base} \le 290\tau$; $n = 6$.*



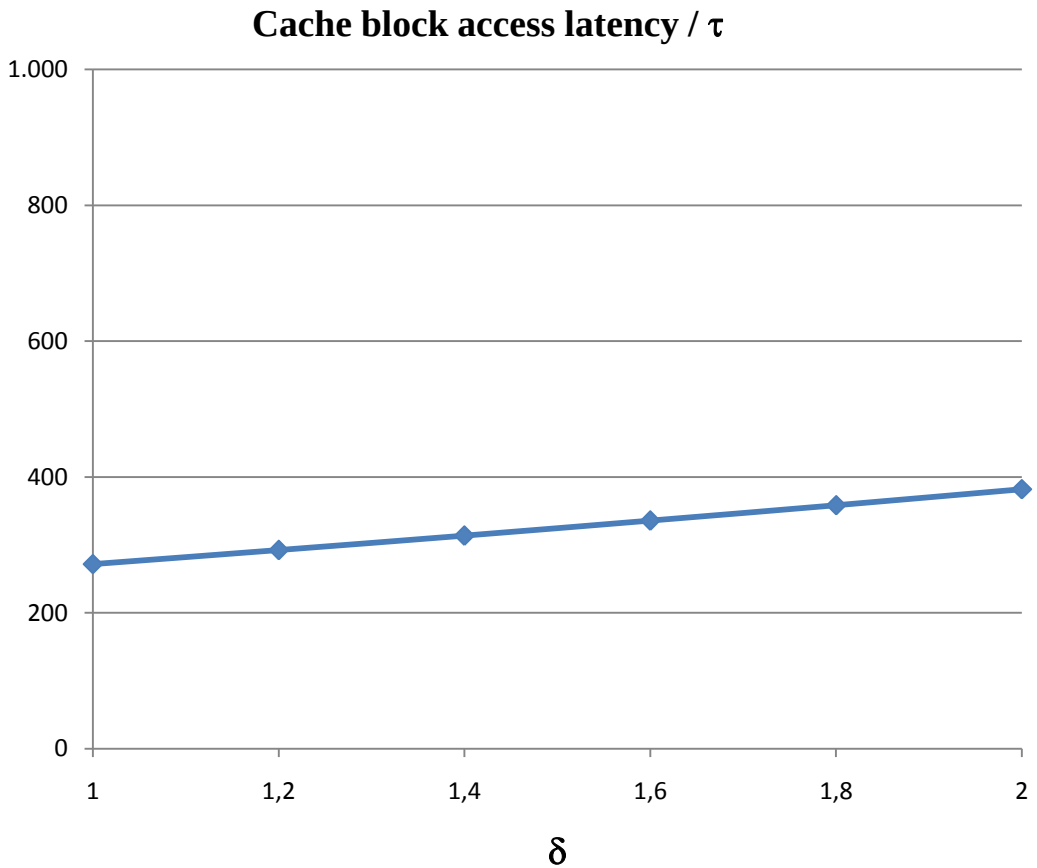
Though large blocks increase R_Q, they can be beneficial:

a double value of R_Q is compensated by a less number of remote accesses.

The positive impact of wormhole flow control is remarkable in this evaluation.

Under-load latency: evaluation

<u>δ variable</u>; σ = 8, t_{hop} = 5 τ, τ_M = 10 τ, N = 64, p = 4, T_p = 2000 τ: *for $1 \le \delta \le 2$, $210\tau \le t_{a\text{-}base} \le 270\tau$; $n = 6$.*

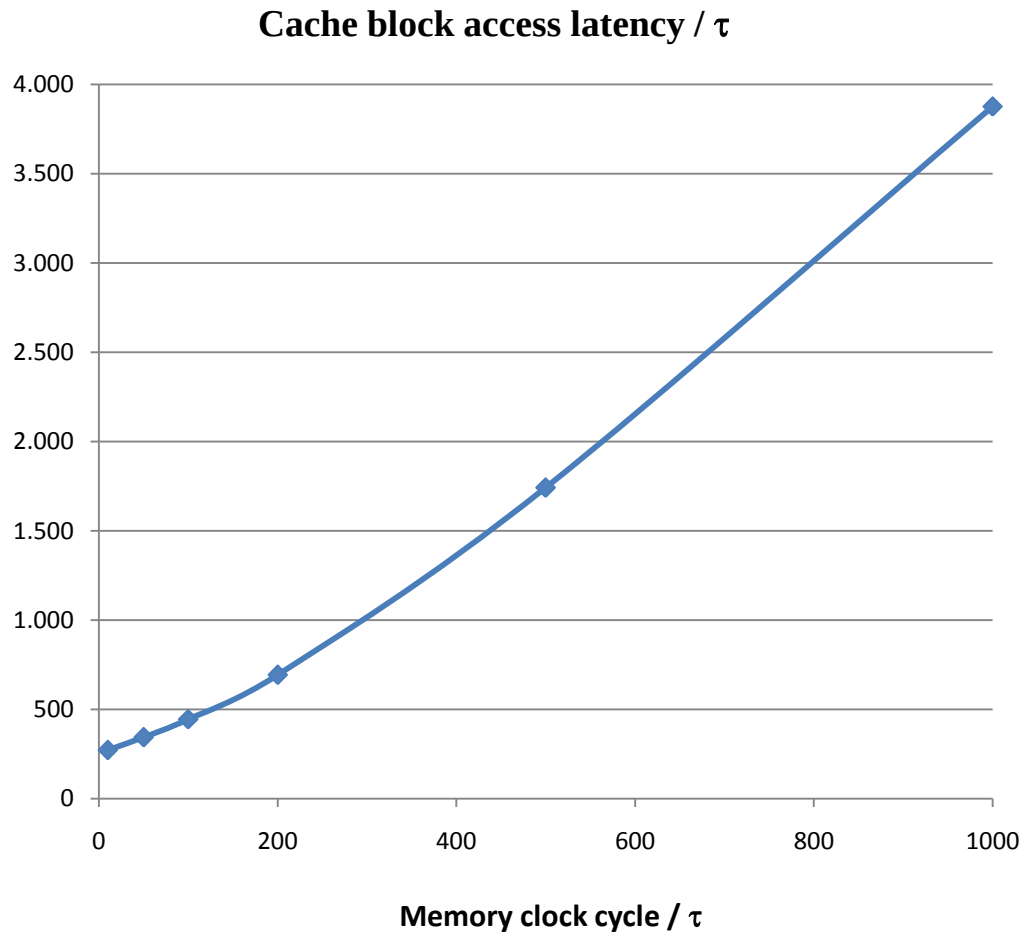


δ has not a meaningful impact for low values of *p*.

The network path length tends to have a negligible impact wrt the impact of the number of nodes in conflict.

Under-load latency: evaluation

τ_M variable; σ = 8, δ = 1, t_{hop} = 5 τ, N = 64, p = 4, T_p = 2000 τ: *n = 6; for 10 $\tau \leq \tau_M \leq$ 1000 τ, 210$\tau \leq t_{a\text{-}base} \leq$ 1200τ.*



"Slow" memories have a relevant, negative impact,

while the impact is limited for memory clock cycle of few 10s τ.