Sparse Networks: balance of processing and communication

Ville Leppänen (University of Turku)
Martti Penttonen (University of Kuopio)

August 12, 2007

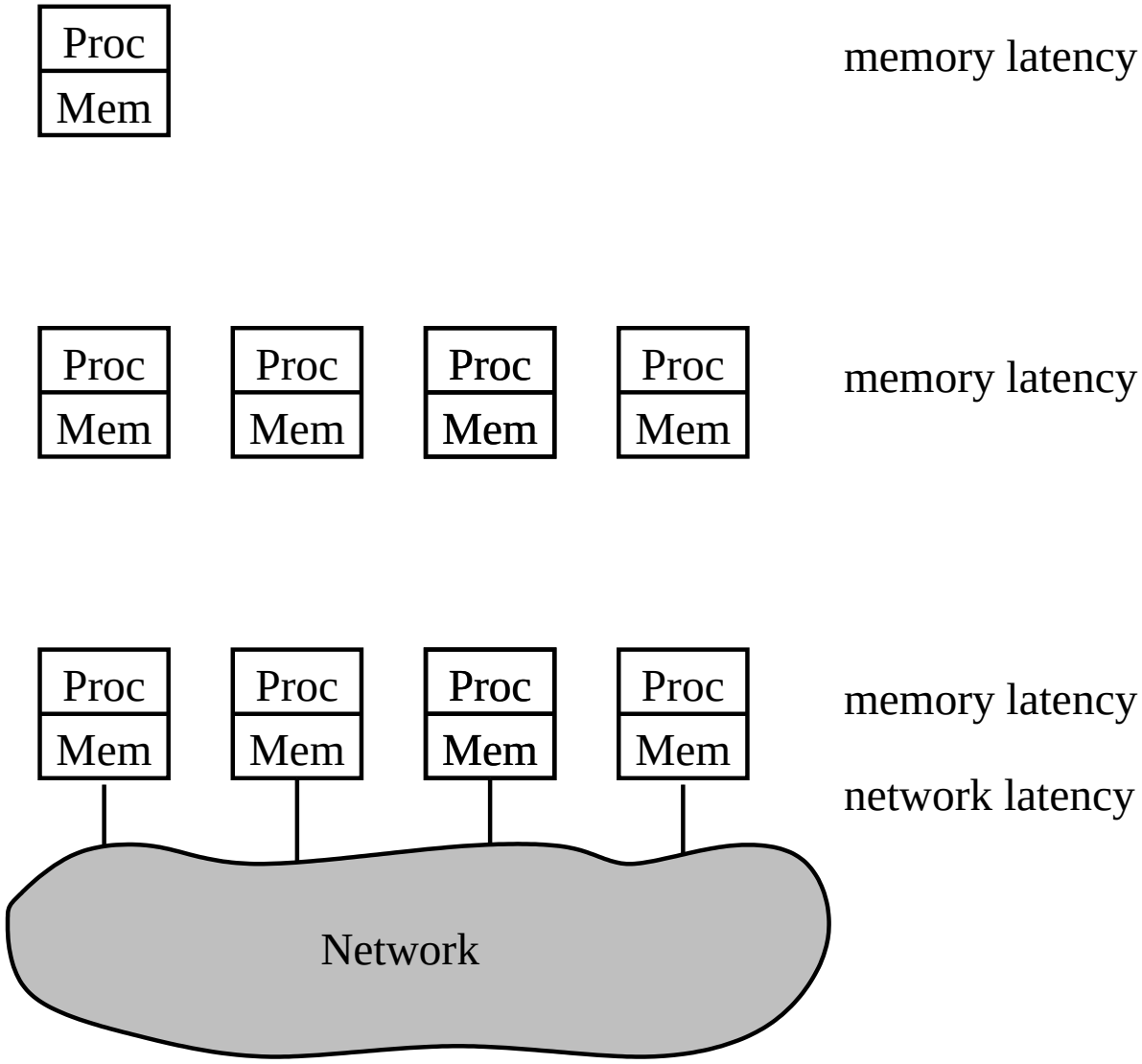
Problem setup

Computation consists of

- processor processing data
- data in processor or in storage
- communicating data between processor and storage

Possible problem: How to move data for processing without too much delay.

Monoprocessor and Multiprocessor



Real world facts

By laws of Nature

- Speed of electrons/light is limited: 1 GHz = 30 cm
- When the number p of processors grows, the network diameter grows by factor $\Omega(\sqrt[3]{p})$
- In practise communication is much slower
- A node in a network is connected to a bounded number d of other nodes

Structure of network

We consider networks consisting of the following elements:

- Processor-Memory nodes connected with d neighbors
- Instead of Processor-Memory node there may be a Router node, also connected with d neighbors
- Connecting links for a node to its neighbors.

Definition. If the number of routers is higher than the number of processors, the network is sparse.

Properties of network

Consider a network of

- n nodes (processors or routers) of degree d
- p processors sending packets every k'th step of computation
- diameter of ϕ hops (from node to node)

Capacity requirement condition:

$$p\phi/k \leq dn$$

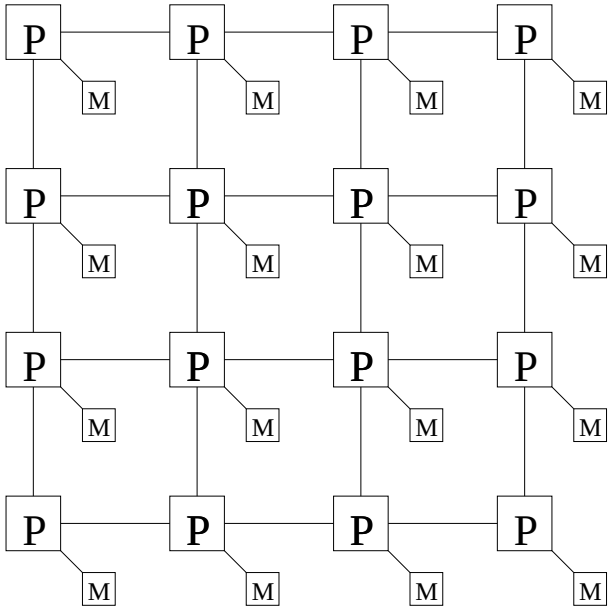
A fundamental observation

Due to $\phi \in \Omega(\sqrt[3]{p})$, either network must be sparse, or data communication must be sparse, i.e. k is not a constant.

Example 1. Mesh

Consider an s-sided mesh of processors:

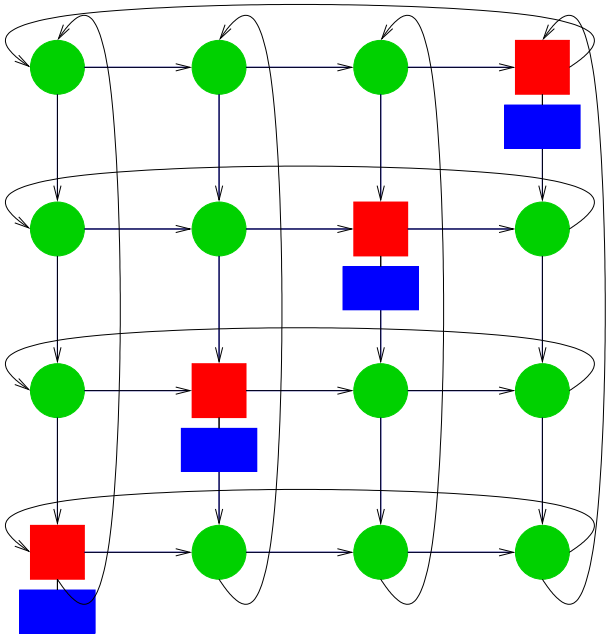
MESH



In mesh, $n = s^2, d = 2, p = s^2, \phi = 2s$.
Thus, $p\phi = 2s^3$, and $dn = 2s^2$.
Hence $p\phi/k \leq dn$ is satisfied only for sparse communication.

Example 2. Sparse mesh/torus

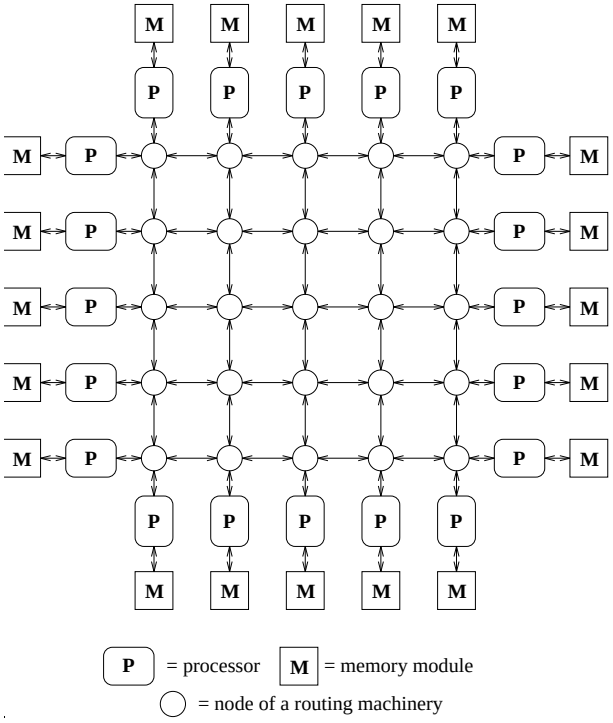
Consider an s-sided sparse torus:



In sparse torus, $n = s^2, d = 2, p = s, \phi = s$.
Thus, $p\phi = s^2$, and $dn = 2s^2$.
Hence $p\phi/k \leq dn$ is satisfied.

Example 3. Coated mesh

Consider an s-sided coated torus:



In coated mesh, $n = s^2, d = 2, p = 4s - 2, \phi = 2s$.
Thus, $p\phi = 8s^2 - 4s$, and $dn = 2s^2$.
Hence $p\phi/k \leq dn$ is satisfied for $k \geq 4$.

Efficient routing

If communication is dense, the network must be sparse. Even then, is efficient communication possible?

We present a scheduled, hot-potato routing algorithm for sparse torus that routes in time O(1) per prosessor.

Slackness principle

In a multiprocessor network latency necessarily grows. It can still use resources efficiently.

Bicycle mechanist's problem. One never knows beforehand, what parts are required. They are ordered when needed. What to do while waiting for the parts? Answer: Start to repair another bike.

Parallel slackness. Assume a parallel algorithm on sp "virtual" processors. When the algorithm is run on a computer of p "real" processors, there is parallel slackness s.

Application of slackness. When sp "virtual" processes are distributed to the p "real"processors, each processor gets s processes. When these are run in turn, there is a gap of $s-1$ steps between two successive steps of computation in a virtual process. This time can be used for fetchin data.

Hot potato routing

In Hot potato routing

- routers do not have memory
- all incoming data must be forwarded immediately, to "right" or "wrong" direction
- protocol determines which data packets go to "right" direction and which go to "wrong" direction

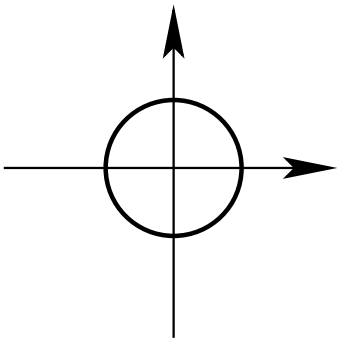
Hot-potato routing is useful in optical communication, because it is difficult to build optical memory.

Scheduled routing

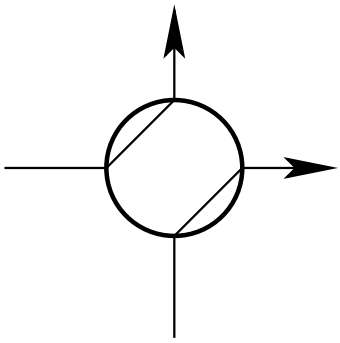
Routing decision at each node and at every moment depends on the time moment only.

Scheduled routing is useful in optical communication, because there is no obvious way for optical addressing and electrical control would require electro-optical transformation, which is slow and costly.

Optical router



direct state

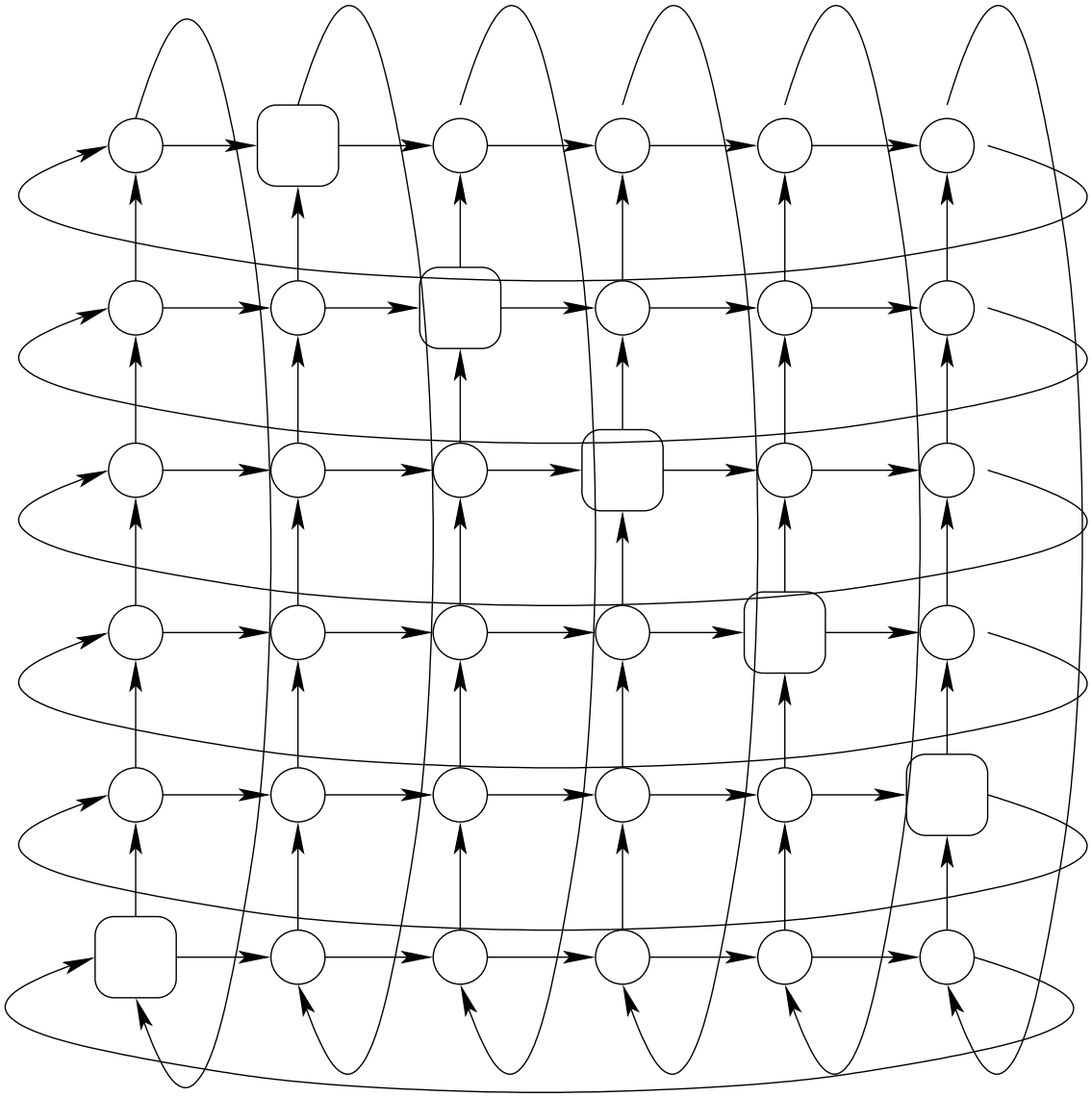


crossing state

dim i to
dim i+1
(mod d)

general crossing

2-dimensional sparse optical torus



Routing in 2-dimensional case

- Let all routers of $ST(2, n)$ be in crossing state at moments $0, n, 2n, \ldots$
- Processor $(x_1, n - x_1)$ sends a packet for $(x_2, n - x_2)$ along x-axis at moment $n - x_2 + x_1$.
- It can send another packet along y-axis respectively.

Multidimensional sparse torus ST(d,n)

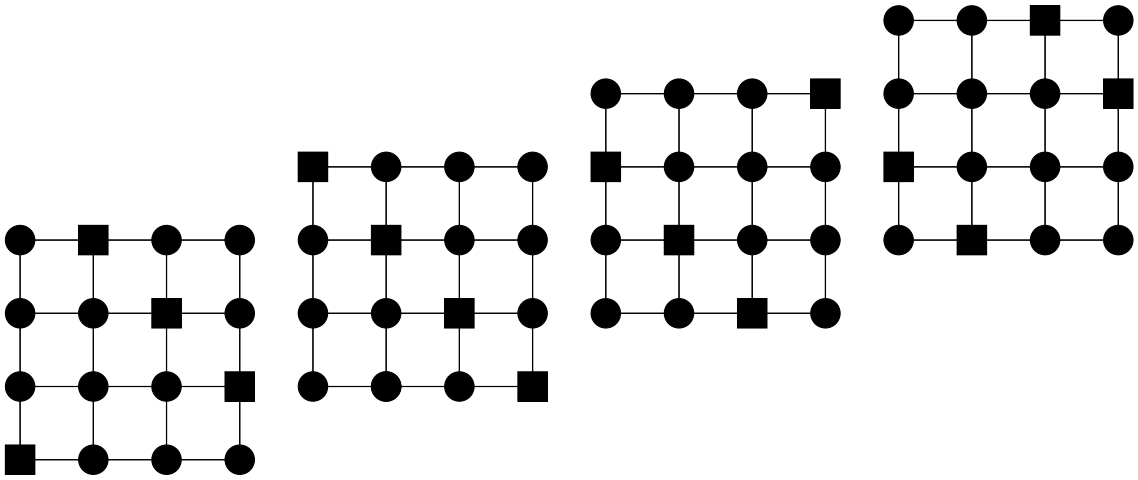
Nodes at coordinates $\{0, 1, \ldots, n-1\}^d$

A node is a *processor* or a *router*.

Processors at postitions $(x_0, x_1, \ldots, x_{d-1})$, where

$$x_0 + x_1 + \ldots + x_{d-1} = 0 \pmod{n}$$

Sparse torus 4**3



Processors: x+y+z=0 (mod 4)

1 proc at level 0, 12 proc at level 4, 3 proc at level 8

Basic properties of sparse tori

Lemma 1. *For $ST(d, n)$,*

(i) The number of processors is n^{d-1}.

(ii) Processors, when projected to the surfaces of $ST(d, n)$, cover the whole surface.

(iii) Average distance from processors to the origin $(0, 0, \ldots, 0)$ is $d(n-1)/2$

Level sizes

Level $L_{d,n}(k)$: nodes at distance k from the origin.

Unfortunately levels are not of equal size.

Lemma 2. *(i)* $|L_{d,n}(0)| = 1$,

(ii) $|L_{d,n}(k)| = \binom{k+d-1}{d-1} - \sum_{i=1}^{\lfloor k/n \rfloor} \binom{d-1+i}{d-1} \times |L_{d,n}(k - i \cdot n)|$ when $0 < k < d \cdot n$.

Scheduled routing

In scheduled routing, routing of packets bases on the moment of sending. Thus, packets need no address information on the route.

Routers follow a certain time dependent pattern, a boolean function $f_r(t)$ that for each time moment t determines, whether router r is in direct or crossing state.

Some observations:

- For symmetry, we can consider sending packets from the processor at the origin, to the other processors.
- At each moment, packets at level l move to level $l + 1$. Hence, packets that were sent at the same time, cannot collide with each others.
- If processors that were sent at the same time, arrive at crossing routers at the same time, they do not collide.

Path patterns

Paths from processor to processor can be expressed by *path patterns* $(D_0, D_1, \ldots, D_{d-1})$, where $0 \leq D_i < n$, and $D_1 + D_2 + \ldots + D_{d-1} = 0 \pmod{n}$.

For a path pattern $(D_0, D_1, \ldots, D_{d-1})$, its *associated* path patterns are $(D_k, D_{k+1 \bmod d}, \ldots, D_{(k+d-1) \bmod d})$.

Note that all associated path patterns are not necessarily different: For example (3,2,3,2), (2,3,2,3), (3,2,3,2), (2,3,2,3) has duplicates.

Routing in $ST(d, n)$

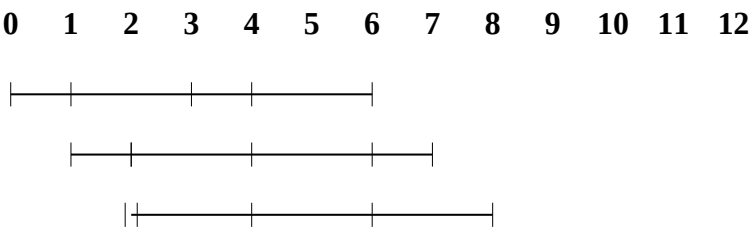
Consider a routing task, where each processor has exactly one packet to route to all processors. We call this a *strict h-relation*. How can we route a strict h-relation?

Main idea. Each processor is reached from the origin by a path pattern. Divide processors (or packets) to groups that have associated path patterns. At every moment, send to d dimensions, packets of associated path patterns. They do not collide with each others!

A minor problem. In some rare cases, there are not d associated path patterns.

Example. Schedule for sparse torus 3^4

time	slice	paths / time
0	**0**	1212 2121
1	**1**	1221 2211 2112 1122
2	**2**	0222 2220 2202 2022
3	**0**	
4	**1**	
5	**2**	
6	**0**	0111 1110 1101 1011
7	**1**	0102 1020 0201 2010
8	**2**	0021 0210 2100 1002
9	**0**	0012 0120 1200 2001
10	**1**	0000



# free slots	2	1	0	0	0	0	0	0	0	0	1	2

Scheduling

- Note that scheduling is preprocessing that is done once before the packets are moved. As the result of scheduling, routers get their control function. During routing, the state of a router depends on its control function and time moment, only.

- Packets have different distances to pass: $0, n, 2n, \ldots$ or $(d-1)n$. The average distance is about $dn/2$

- By greedy principle, send the long distance packets first. (Another way to handle long distance packets is to split a long path to subpaths of length n. During routing phase the processor needs to decide whether it has reached the target or needs to be forwarded at suitable time.)

- A packet needs $dn/2$ link time on the average. Each of the n^{d-1} processors has n^{d-1} packets. Altogether, $n^{d-1} \times n^{d-1} \times dn/2$ link time is needed. There are dn^d links in the network, hence total time is $n^{d-1}/2$. Thus, about $1/2$ time per packet is needed.

Result

Theorem 1. *A strict all-to-all routing task can be routed in time* $0.5 + o(1/p)$*, where* p *is the number of processors.*