CS855: Parallel Linear Algebra (I)

Marc Moreno Maza

February 19, 2008

Outline

- Routing algorithms and switching techniques
- Permutation routines in mesh-based networks
- Basic matrix mapping schemes
- Matrix transposition
- Matrix-vector multiplication
- Matrix-matrix multiplication

- Communication problems.
- Router architecture.
- One-to-one routine algorithms.
- Basic switching techniques: Performance metrics.
- Circuit switching (CS)
- Store-and-forward (SF) switching.

One-to-one communication: Information is only exchanged (not duplicated) between one or several pairs of nodes.

single-pair communication: just one isolated communicating pair.

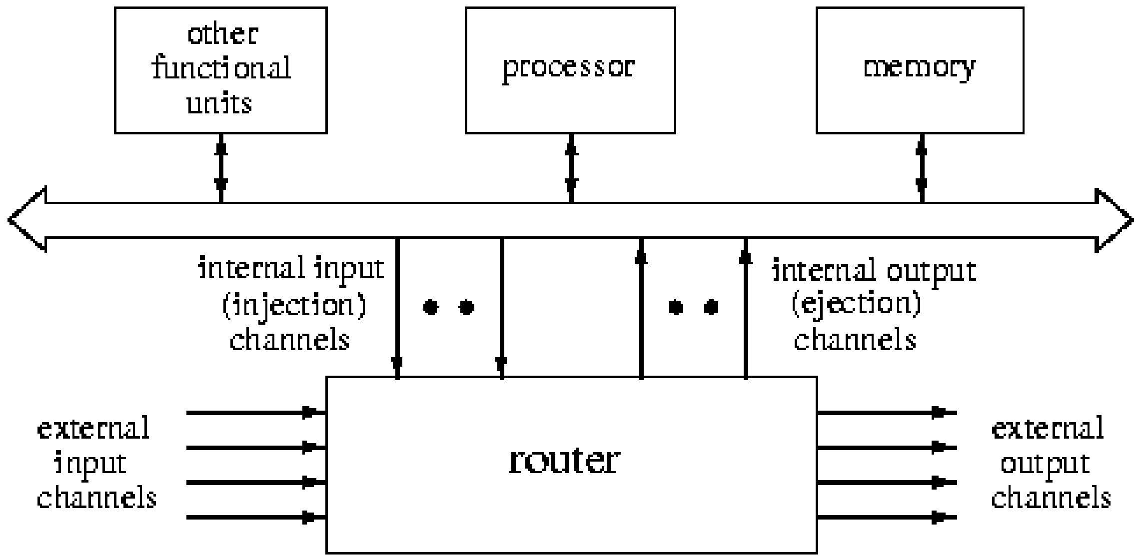
many one-to-one communications: several pairs exchange information simultaneously.

permutation routing: every node is a source of one message and a destination for another message also.

One-to-many communication: One node is the sender and several or all nodes are receivers.

- If the same information is disseminated in multiple copies among the other nodes, these patterns are known as multicast and **one-to-all broadcast** communication
- If the information sent to each destination node is different, the pattern is called **one-to-all scatter**.

All-to-all communication: Again, all-to-all broadcast or scatter.

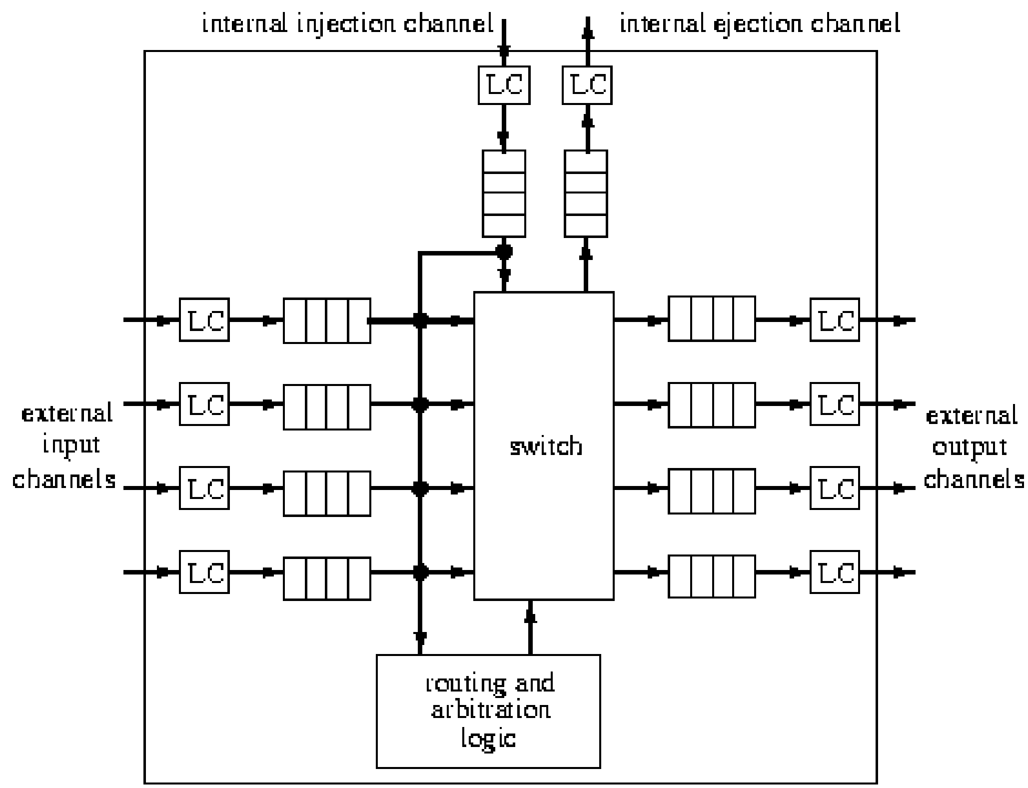


processing node: standard bus-based computer augmented with a router.

router: HW coprocessor implementing lower levels of communication protocol. Comprises of a switch, buffers, and routing-arbitration unit.

buffer: FIFO memory for storing one or several units of communication in transit.

switch: interconnects input buffers to output buffers (channels).



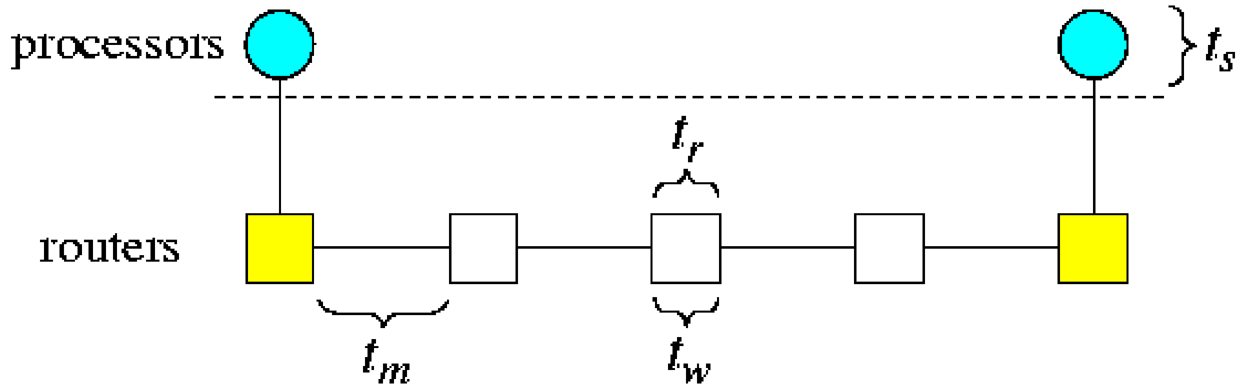
routing and arbitration unit: implements the routing algorithm and packet flow control protocol and sets the switch accordingly. For example, it resolves conflicts between simultaneous requests for the same output link.

routing algorithm: determines the path selected by a packet to reach its destination. It must decide within each intermediate router which output channel(s) are to be selected for exiting packets.

switching mechanism: determines how network resources are allocated for data transmission, i.e., how and when the input channel is connected to the output channel selected by the routing algorithm. It is the actual mechanism that removes data from input channels and places them on output channels.

flit: small unit of information at link layer, with size one or several words. Flits can be of several types and flit exchange protocol typically requires several cycles.

phit: the smallest physical unit of information at the physical layer, which is transfered across one physical link in one cycle.



channel width w**:** # of bits that a physical channel can transmit simultaneously between two adjacent routers. Clearly, it is exactly the size of a phit.

channel rate μ**:** the peak rate in bits/second at which bits can be transfered over each individual wire of a physical channel.

channel bandwidth B **:** μ expressed in phits/second.

channel latency t_m**:** the inter-router latency of one phit, $t_m = 1/B$.

startup latency t_s**:** the time required for packet framing/unframing, copying data between memory and router buffers, validation and so on, at both source and destination side. It depends mainly on the design of system SW, on its interface to user programs, and on the physical interface between processing nodes and routers. It is a static value, independent of network traffic or distance between nodes. It depends only on the size of the message.

network latency: the elapsed time after the head flit of a packet has entered the network at the source router until the tail flit is ejected. The network latency consists of two components: a conflict-free packet transmission time, which strongly depends on the switching technique, and the blocking time, which includes all possible delays encountered during the lifetime of a packet.

In the remaining of our discussion on architecture:

- we will not be mentioning startup time t_s, since it is a static component
- we give formulae for the conflict-free network latency, i.e., assuming a

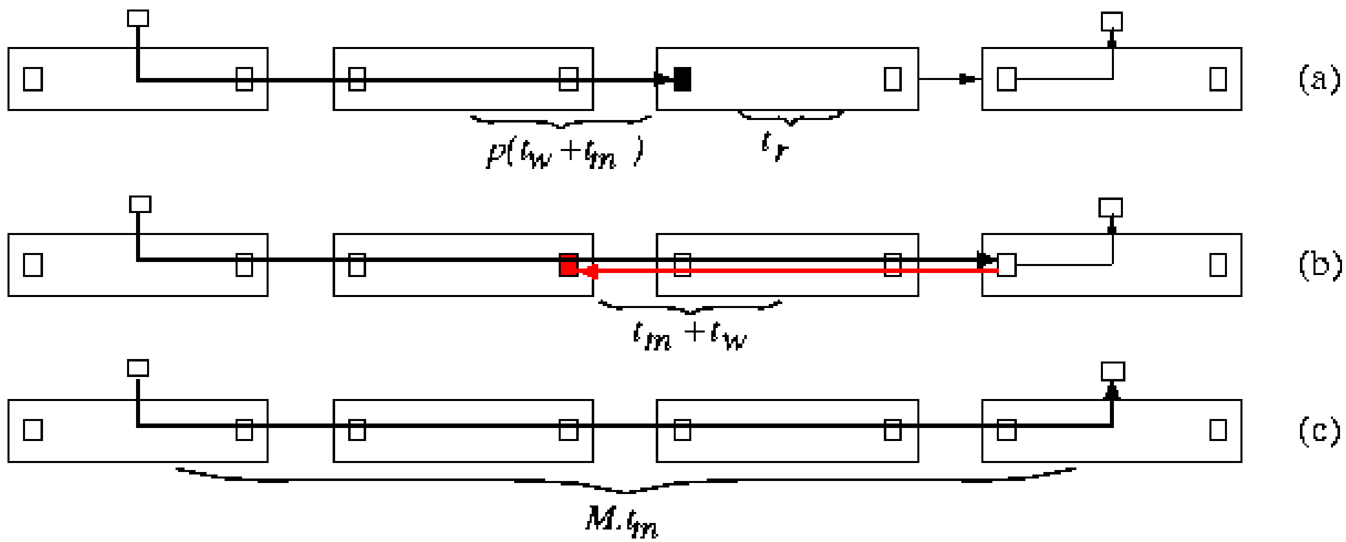
network without any other traffic. We will call this latency the **base communication latency**.

- We will use the following notations:
  - w = phit size = channel width (in bits),
  - M = packet size (in flits),
  - t_r = time of routing decision within one router during building a routing path (in seconds),
  - t_w = intra-router switching latency once the router has made the routing decision and a path has been set up through the router (in seconds/phit),
  - t_m = 1/B = inter-router channel latency (in seconds/flit). The transmission latency of a packet of M flits between two adjacent routers is $M\,t_m$ seconds.
  - d = the length of a routing path.

- The communication between a source and destination has two phases:
  - circuit establishment phase and,
  - message transmission phase.
- A physical path from the source to the destination is reserved prior to the transmission of data by injecting a routing probe, which contains destination address and some control information. It usually needs more that one phit. Let p be the size of the probe in phits.
- The probe progresses towards the destination node, reserving physical links as it is transmitted through intermediate routers.
- The path is set up after the probe has reached the destination.
- Then an acknowledgment flit is sent back.
- Upon reception of the acknowledgment, the sender transmits the whole message at the full bandwidth of the path.
- The path forms a HW circuit that is reserved for the whole time of the message transmission.

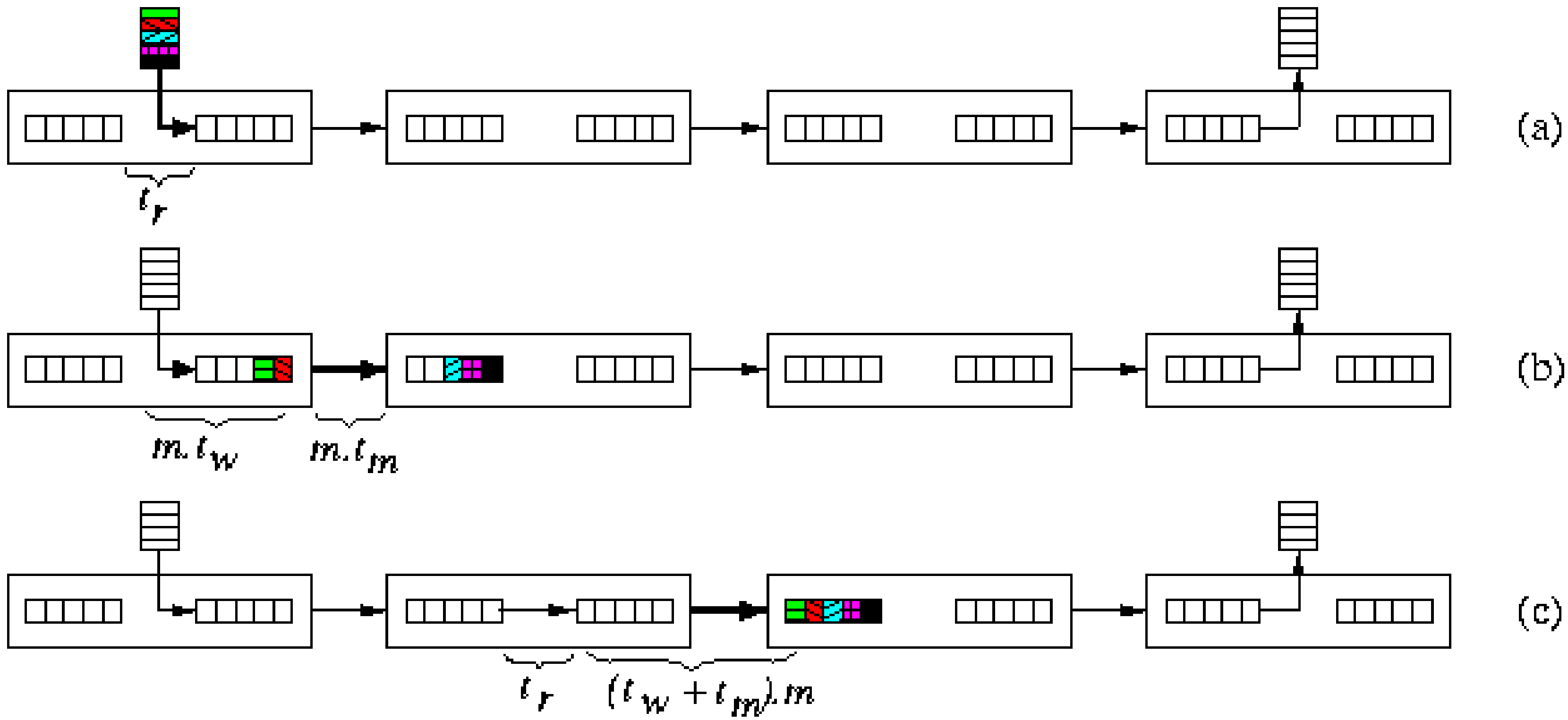
- The circuit is released either by destination node or by the last bits of the message.
- **Base communication latency:** A message of length M transmitted to distance d takes time

$$t_{CS} = d(t_r + (p+1)(t_w + t_m)) + M\, t_m.$$



- This technique is alternatively called packet switching.
- The message is split into fixed-length packets, every packet consists of flits, starting with the header flit.
- Every channel has input and output buffers for one entire packet.
- Every packet is individually routed from the source to the destination.
- One step of the SF switching is called **hop**: It consists of copying the whole packet from one output buffer to the next input buffer.
- Routing decisions are made by each intermediate router only after the whole packet was completely buffered in its input buffer.
- **Base communication latency:** A message of length M transmitted to distance d takes time

$$t_{SF} = d(t_r + (t_w + t_m)M).$$



- (SF) is advantageous, when messages are short and frequent, whereas (CS) is advantageous if messages are infrequent and long.
- (CS) does not need to break messages into fixed-length packets.

- We consider both complete and partial permutation routings: each node can be a source and/or destination of at most one packet.
- We investigate store-and-forward switching meshes only.
- We discuss
  - Farthest-first routing in 1-D meshes.
  - XY greedy permutation routing in 2-D meshes.

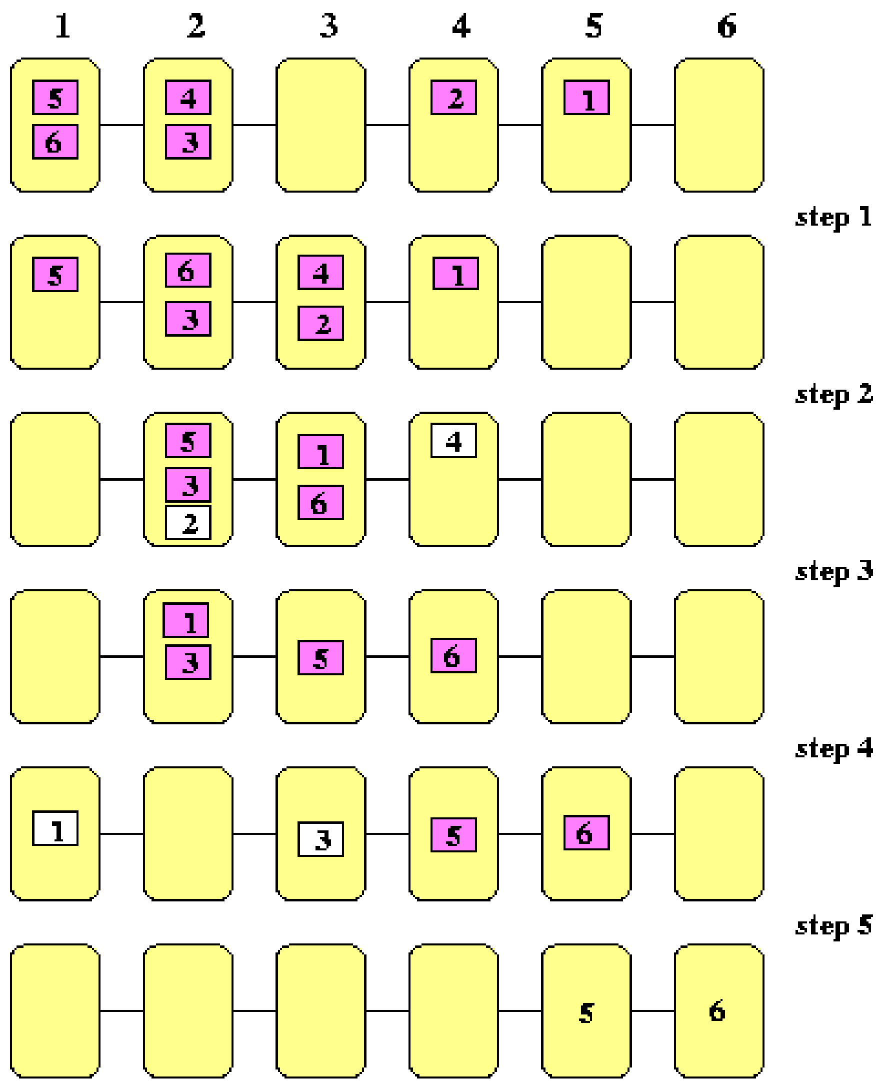
Lemma 1 *Consider a linear array $M(n)$ of n nodes with full-duplex all-port links and (SF) switching, and with $O(n)$-packet buffers. Assume a one-to-one routing problem where each node is a source of any number of packets, but globally, there exists at most one packet destined for each node. Then farthest-first greedy routing strategy needs at most $n-1$ steps (hops) to deliver all packets.*

PROOF. (Sketch of)

- Since the channels are full-duplex, packets moving in opposite directions can never interfere so that each node (except the corner ones) can inject two packets and receive two packets at each step.
- So we will examine only packets moving in the same direction, say **rightward**.
- Routers implement the farthest-first strategy to resolve possible channel contention: if they have more than one packet to be sent in the same direction, the priority is given to the packet that has the longest path ahead.

- The inductive proof is based on the fact that packets destined for the rightmost nodes in the array keep moving once they start.

□


1
2
3
4
5
6
5
6
4
3
2
1
step 1
5
6
3
4
2
1
step 2
5
3
2
1
6
4
step 3
1
3
5
6
step 4
1
3
5
6
step 5
5
6

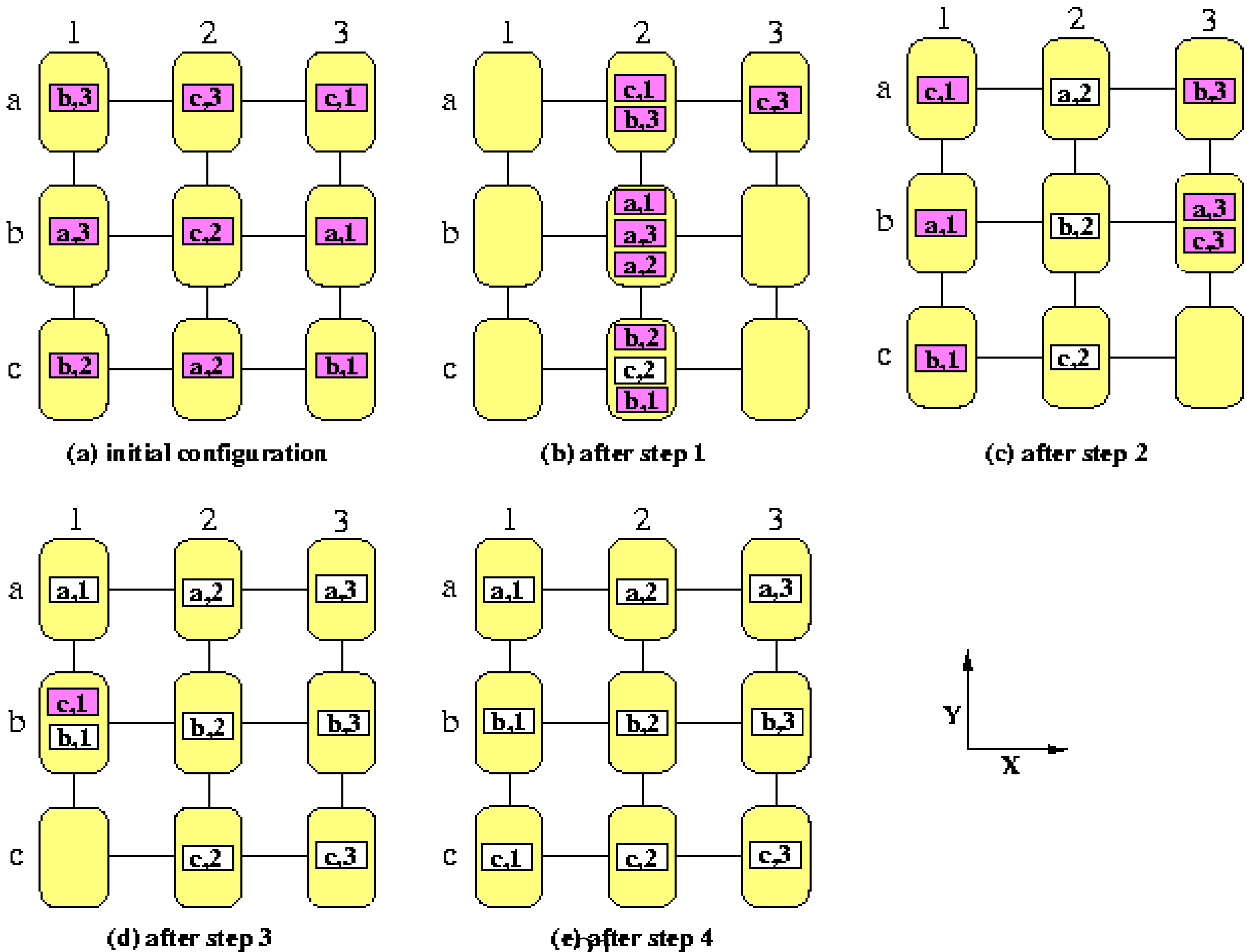
- The previous Lemma has one important implication: If we take any N-node 2-D mesh, we can perform any permutation of N packets between nodes in the optimum number of steps, but to achieve that, we will need to store $O(N)$ packets in one router at one time in the worst case
- In general, we will assume that each router has auxiliary buffers for β packets in addition to the standard ones. Our desire of course is to minimize β.
- A permutation is *memory-optimal* if $\beta = O(1)$.

Lemma 2 *Consider store-and-forward all-port mesh $M(n, n)$, $n \geq 2$, with full-duplex channels and any* <u>*permutation routing problem*</u> *on $M(n, n)$. Using greedy XY routing, at most $2n - 2$ steps is needed to complete the permutation (which is optimal, since the diameter of $M(n, n)$ is $2n - 2$), but $\beta = max(2, 2n/3 - 1)$.*

PROOF. (Sketch of)

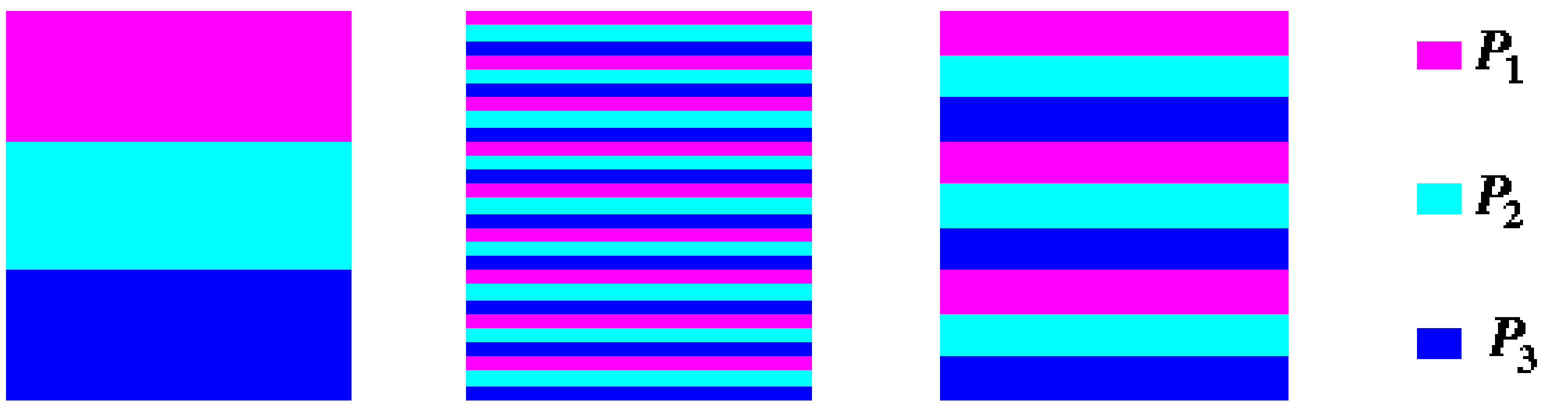
- In the first step, all packets which are not in their destination columns, start moving horizontally.
- Since each node has initially only one packet to send and channels are full-duplex, packets keep moving horizontally without contention, until they reach their destination columns.
- Hence, each packet will reach its correct column in at most $n-1$ steps.
- Within columns, the farthest-first routing used in Lemma applies to packets, gathered in buffers of column nodes. By this lemma, this will take again at most $n-1$ steps.
- It remains to show that the buffers in each router must be for at least $\max(3, 2n/3-1)$ packets.

$\square$

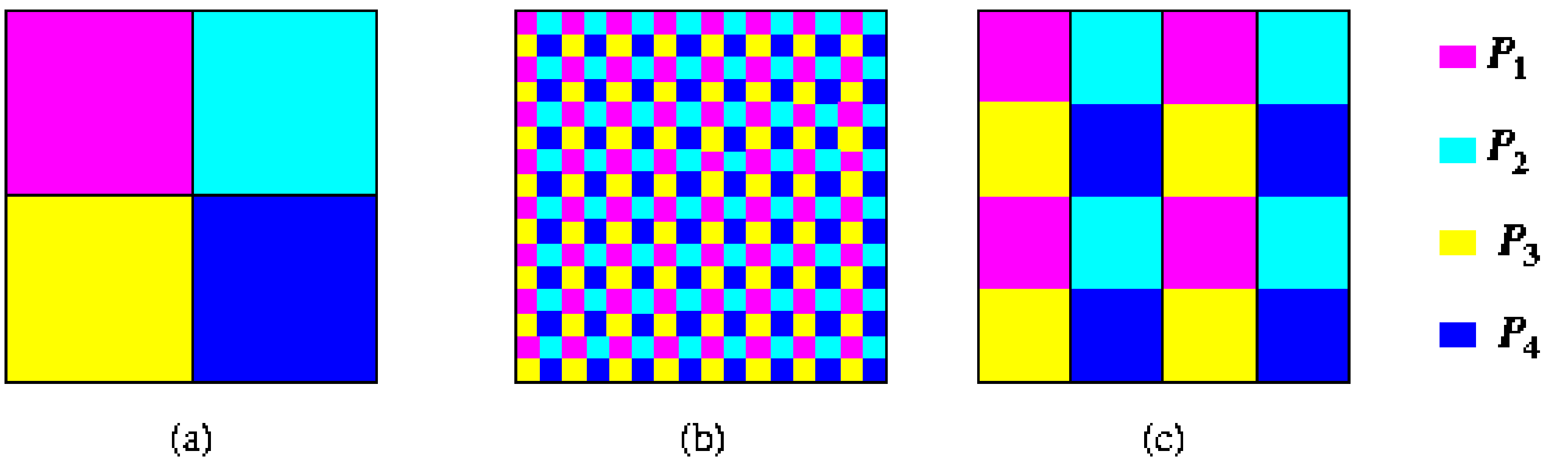

1
2
3
a
b
c
b,3
c,3
c,1
a,3
c,2
a,1
b,2
a,2
b,1
(a) initial configuration
c,1
b,3
c,3
a,1
a,3
a,2
b,2
c,2
b,1
(b) after step 1
c,1
a,2
b,3
a,1
b,2
a,3
c,3
b,1
c,2
(c) after step 2
a,1
a,2
a,3
c,1
b,1
b,2
b,3
c,2
c,3
(d) after step 3
a,1
a,2
a,3
b,1
b,2
b,3
c,1
c,2
c,3
(e) after step 4
Y
X

- Performance and elegance of matrix computations in parallel distributed memory machines depends strongly on the way matrices are mapped or partitioned among processors.
- Choosing the most appropriate mapping scheme is an inevitable part of a parallel matrix algorithm design.

Striped mapping Consider an $n \times n$ matrix and p processors $P_1, \ldots Pp$ with $p < n$. Processors are virtually arranged into a linear array. Assume for simplicity that p divides n and let $k = n/p$.

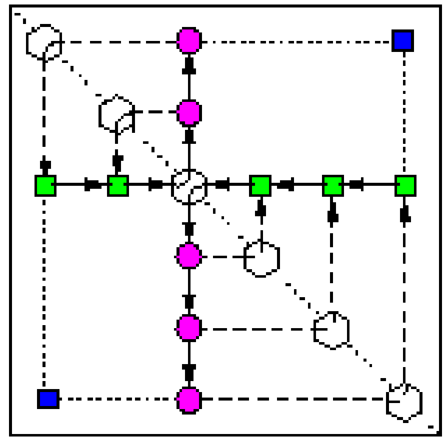


Checkerboard mapping Checkerboard mapping views p processors virtually arranged into a 2D mesh $M(r, r)$, with $r = \sqrt{p}$, denoted by $P_{i,j}, 1 \le i, j \le r$. Assume that r is integer. Consider an $n \times n$ matrix A with $n^2 \ge p$.



The transposition is the simplest matrix operation. Consider an $n \times n$ matrix A in Checkerboard mapping.

- Assume again for simplicity that $r = \sqrt{p} \in \mathbb{Z}$.
- We view p processors arranged into a virtual 2D mesh $M(r, r)$.
- Then a transposition is equivalent to **pairwise exchanges** of elements or submatrices between pairs of nondiagonal processors symmetric along the main diagonal.
- This “diagonal” matching defines a “partial” permutation.
- The implementation depends on the real physical network and communication hardware; we will consider (SF) switching on a (real) 2D mesh with XY routing.



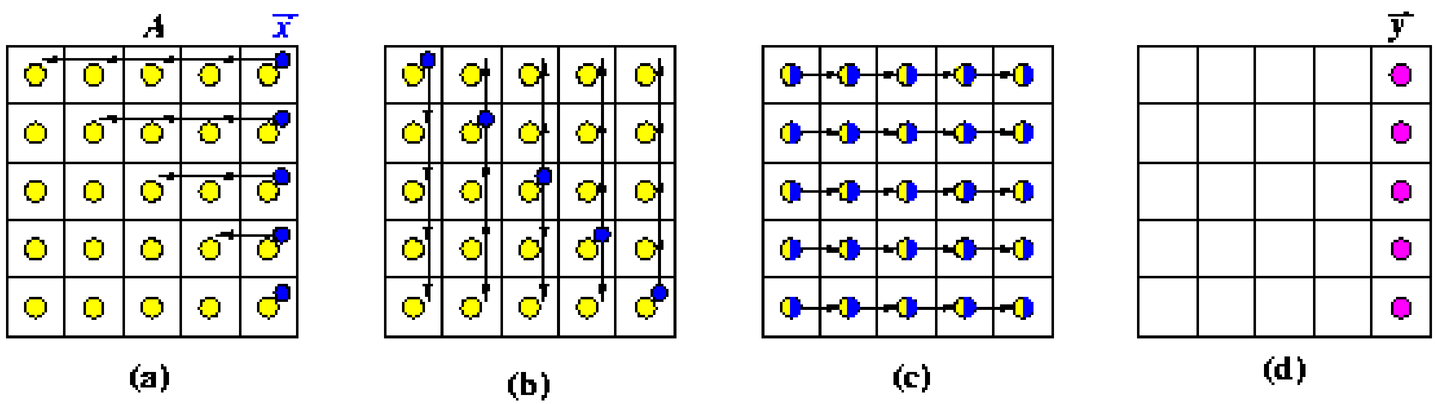
1. One processor owns an $n/r \times n/r$ submatrix.
2. All processors $P_{i,j}$ send horizontally their submatrices to $P_{i,i}$ on the main diagonal so that each row hosts two horizontal pipelines moving from the boundaries toward the main diagonal.
3. $P_{i,i}$ forwards a packet upward (downward) if arrived from left (right).
4. All the packets start moving from the very first round and keep moving until they reach their destinations.

- The total time is determined by the communication latency between the two (blue) processors in the opposite corners of $M(r, r)$.
- The total time includes the time for local sequential transpositions of submatrices in the destination processors:

$$T_{TR}(n, p) = t_s + 2\frac{n^2}{p}t_m(\sqrt{p} - 1) + O(\frac{n^2}{p}).$$

Assume an $n \times n$ matrix A in Checkerboard mapping and an $n \times 1$ vector x. We are to compute the product $y = Ax$ on a 2D mesh $M(r, r)$ of $p = r^2$ processors. Assume for simplicity that n is divisible by p and let $k = \frac{n}{p}$.

- Assume further that the input vector x is block-partitioned among the rightmost processors of $M(r, r)$ initially, i.e., for all $1 \leq i \leq r$, Processors $P_{i,r}$ owns subvector $x_{(i-1)k+1}, \ldots, x_{ik}$.
- We will also require that at the end, the resulting vector is stored equally as the input vector was, in the rightmost column of processors. Under these assumptions, the algorithm needs four steps:



1. Vector x has to be aligned diagonally. For all $i = 1, \ldots, r$ **in parallel**, Processor $P_{i;r}$ sends its subvector of x to $P_{i,i}$. It is just a singlenode routing within each row of $M(r;r)$.
2. For all $i = 1, \ldots, r$ **in parallel**, Processor $P_{i,i}$ does *one-to-all broadcast* within its column.
3. All processors of $M(r;r)$ **in parallel** compute local scalar products of their submatrices of A and subvectors of x.
4. These partial values must be summed within rows. To do that, we have to perform reductions in all rows in parallel, with the roots being the processors in the rightmost column of $M(r;r)$.

The number of communication steps of this algorithm is $\Theta(r)$ on $M(r,r)$.

Consider $C = AB$, where A, B, C are square $n \times n$ matrices, on a p-processor 2D mesh $M(r;r)$, with $r = \sqrt{p}$ Assume the block checkerboard mapping of both input A, B and output C on $M(r;r)$. An intuitively obvious approach could be first to broadcast submatrices so that before the computation starts, each processor $P_{i,j}$ gets the whole rows of A and the whole columns of B, needed for computing $C_{i;j} = \Sigma_{i=1}^{r} A_{i,k} B_{k,j}$. Such a parallel algorithm consists of 3 phases, each requiring $O(r)$ steps on $M(r;r)$.

1. Each processor $P_{i,j}$ broadcasts its submatrix $A_{i,j}$ to all other processors in the ith row of $M(r;r)$. This leads to independent All-to-All broadcasts within each of the r row in parallel.

2. Each processor $P_{i,j}$ broadcasts $B_{i,j}$ to all other processors in the jth column of $M(r;r)$.

3. At the end of the second phase, each processor $P_{i,j}$ holds all it needs to compute $C_{i,j}$.

Assuming an all-port model, Phases 1 and 2 can be done in parallel and also, the communication can be interleaved with computation of Phase 3. Hence, the actual implementation can be made more efficient. This algorithm, however, is **inefficient in terms of memory use**. Each element of both the input matrices exists in r copies in the distributed memory, hence in total, the parallel algorithm requires r times more memory space than the sequential one. There exist memory efficient algorithms, such as Cannon's algorithm described below.

Cannon's algorithm shifts rows of A and columns of B so that matching submatrices $A_{i,k}$ and $B_{k,j}$ meet in $P_{i,j}$ at the same time, for all $1 \leq i, j, k \leq r$.

for $i = 1, \ldots, r$ **do_in_parallel**

rotate submatrices of A in row i by $i - 1$ positions leftward

for $j = 1, \ldots, r$ **do_in_parallel**

rotate submatrices of B in column j by $j - 1$ positions upward

repeat r times

for all processors in $M(r; r)$ **do_in_parallel**

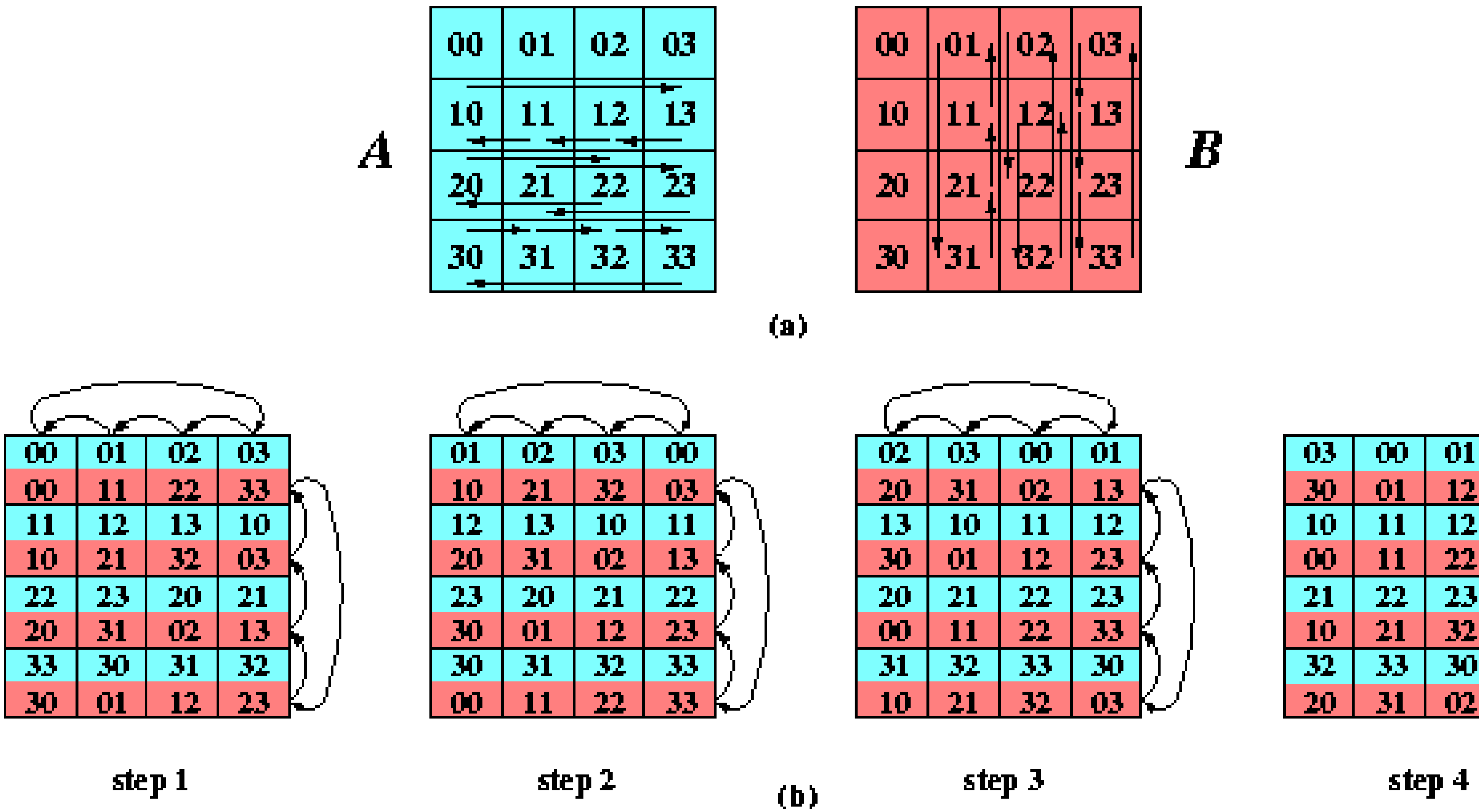
multiply current submatrix of A and B and add the result to C

for $i = 1, \ldots, r$ **do_in_parallel**

rotate submatrices of A in row i by 1 position leftward;

for $j = 1, \ldots, r$ **do_in_parallel**

rotate submatrices of B in column j by 1 position upward;


A
B
(a)
step 1
step 2
step 3
step 4
(b)

References and Acknowledgments

Most of the material of these course notes are inspired by the following.

- Parallel Algorithms (WISM 459, 2005/2006) by Rob Bisseling.
  `http://www.math.uu.nl/people/bisselin/pas1.html`
- Topics in Parallel Computing (CS 838, Univ. of Wisconsin-Madison, Spring 1999) by Pavel Tvrdik.
  `http://www.cs.wisc.edu/ tvrdik/cs838.html`
- Parallel Algorithms (CS 662, San Diego Univ., Spring 1996) by Roger Whitney.
  `http://www.eli.sdsu.edu/courses/spring96/cs662/notes/`