VIII. Communication costs, routing mechanism, mapping techniques, cost-performance tradeoffs

April 6th, 2009

Message Passing Costs

- Major overheads in the execution of parallel programs: from communication of information between processing elements.
- The cost of communication is dependent on a variety of features including:
  - programming model semantics,
  - network topology,
  - data handling and routing, and
  - associated software protocols.
- Time taken to communicate a message between two nodes in a network

  = time to prepare a message for transmission + time taken by the message to traverse the network to its destination.

Parameters that determine the communication latency

- **Startup time** (t_s):
  - The startup time is the time required to handle a message at the sending and receiving nodes.
  - Includes
    1. the time to prepare the message (adding header, trailer & error correction information),
    2. the time to execute the routing algorithm, and
    3. the time to establish an interface between the local node and the router.
  - This delay is incurred only once for a single message transfer.
- **Per-hop time** (t_h):
  - After a message leaves a node, it takes a finite amount of time to reach the next node in its path.
  - The time taken by the header of a message to travel between two directly-connected nodes in the network
  - It is also known as node latency.
  - Is directly related to the latency within the routing switch for determining which output buffer or channel the message should be forwarded to.
- **Per-word transfer time** (t_w):
  - If the channel bandwidth is r words per second, then each word takes time $t_w = 1/r$ to traverse the link.
  - This time includes network as well as buffering overheads.

Store-and-Forward Routing

- When a message is traversing a path with multiple links, each intermediate node on the path forwards the message to the next node after it has received and stored the entire message.
- Suppose that a message of size m is being transmitted through such a network. Assume that it traverses *l* links.
- At each link, the message incurs a cost t_h for the header and $t_w m$ for the rest of the message to traverse the link.
- Since there are l such links, the total time is $(t_h + t_w m)l$.
- Therefore, for store-and-forward routing, the total communication cost for a message of size m words to traverse l communication links is

$$t_{comm} = t_s + (m t_w + t_h) l.$$

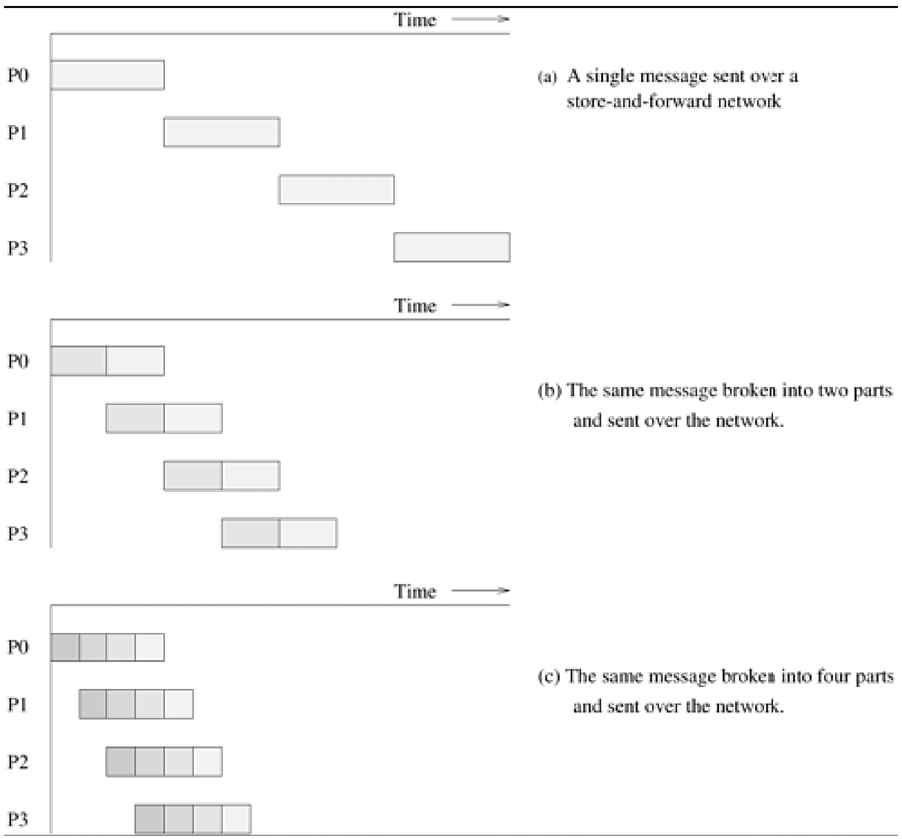
- In current parallel computers, the per-hop time t_h is quite small.
- For most parallel algorithms, it is less than $t_w m$ even for small values of m and thus can be ignored.
- For parallel platforms using store-and-forward routing, the time given by the above equation can be simplified to

$$t_{comm} = t_s + m l t_w.$$

Packet Routing

- Store-and-forward: message is sent from one node to the next only after the entire message has been received
- Consider the scenario in which the original message is broken into two equal sized parts before it is sent.
  - An intermediate node waits for only half of the original message to arrive before passing it on.
- A step further: breaks the message into four parts.
- In addition to better utilization of communication resources, this principle offers other advantages:
  - lower overhead from packet loss (errors),
  - possibility of packets taking different paths, and
  - better error correction capability.
- This technique is the basis for long-haul communication networks such as the Internet, where error rates, number of hops, and variation in network state can be higher.
- The overhead here is that each packet must carry routing, error correction, and sequencing information.

Store-and-forward vs. packet routing



- Passing a message from node P0 to P3
  (a) through a store-and-forward communication network;
  (b) extending the concept to cut-through routing.
- The shaded regions represent the time that the message is in transit.
- The startup time associated with this message transfer is assumed to be zero.

Cost communication in packet routing

- Consider the transfer of an *m* word message through the network.
- Assume:
  - routing tables are static over the time of mes. transfer - all packets traverse the same path
  - The time taken for programming the network interfaces & computing the routing info, is independent of the message length and is aggregated into the startup time t_s
  - Size of a packet: $r + s$, r - original message, s - additional information carried in the packet
  - Time for packetizing the message is proportional to the length of the message: mt_{w1}.
  - The network is capable of communicating one word every t_{w2} seconds,
  - Incurs a delay of t_h on each hop, and
  - The first packet traverses l hops,
- Then the packet takes time $t_h l + t_{w2}(r + s)$ to reach the destination.
- The destination node receives an additional packet every $t_{w2}(r + s)$ seconds.
- Since there are $m/r - 1$ additional packets, the total communication time is given by:

$$
\begin{aligned}
t_{comm} &= t_s + t_{w1}m + t_h l + t_{w2}(r+s) + \left(\frac{m}{r} - 1\right) t_{w2}(r+s) \\
&= t_s + t_{w1}m + t_h l + t_{w2}m + t_{w2}\frac{s}{r}m \\
&= t_s + t_h l + t_w m,
\end{aligned}
$$

$$
t_w = t_{w1} + t_{w2}\left(1 + \frac{s}{r}\right).
$$

- Packet routing suited to networks with highly dynamic states & higher error rates,
  - such as local- and wide-area networks.
  - Individual packets may take different routes &retransmissions can be localized to lost packets.

Cut-Through Routing

- Aim: to further reduce the overheads associated with packet switching.
  - Forcing allpackets to take the same path, we can eliminate the overhead of transmitting routing information with each packet.
  - By forcing in-sequence delivery, sequencing information can be eliminated.
  - By associating error information at message level rather than packet level, the overhead associated with error detection and correction can be reduced.
  - Since error rates in interconnection networks for parallel machines are extremely low, lean error detection mechanisms can be used instead of expensive error correction schemes.
- Routing scheme resulting from these optimizations:cut-through routing.
  - A message is broken into fixed size units called flow control digits or flits.
  - Flits do not contain the overheads of packets => much smaller than packets.
  - A tracer is sent from the source to the destination node to establish a connection.
  - Once a connection has been established, the flits are sent one after the other.
  - All flits follow the same path in a dovetailed fashion.
  - Intermediate node does not wait for entire message to arrive before forwarding it.
  - As soon as a flit is received at intermediate node, it is passed on to the next node.
- No necessary a buffer at each intermediate node to store the entire message.

  => cut-through routing uses less memory at intermediate nodes, and is faster.

Cost of cut-through routing

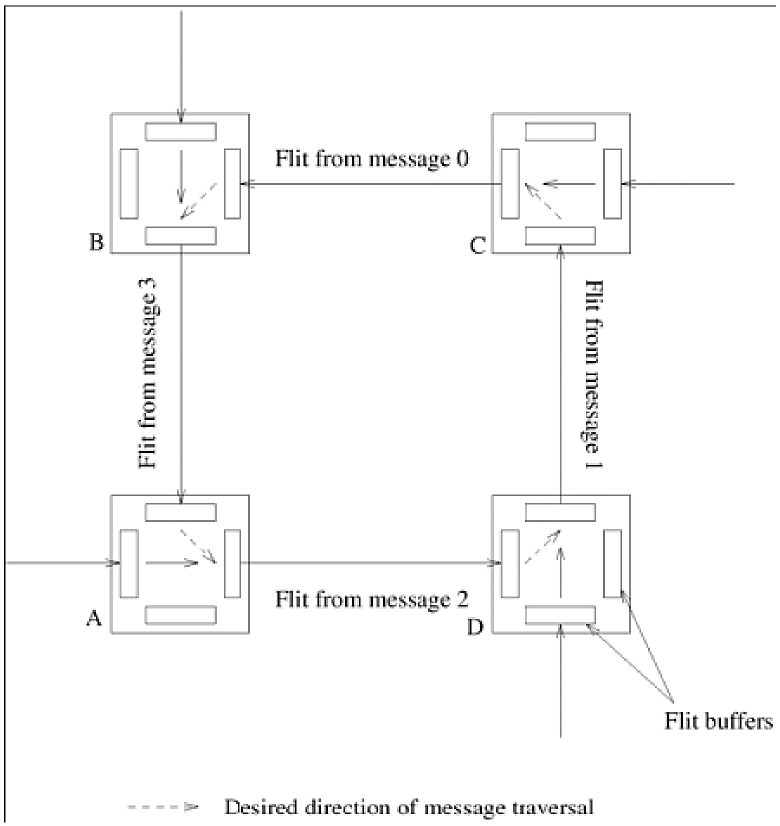
- Assume:
  - the message traverses *l* links, and
  - *th* is the per-hop time => the header of the message takes time lt_h to reach the destination.
  - the message is m words long => the entire message arrives in time $t_w m$ after the arrival of the header of the message.
- The total communication time for cut-through routing is

$$t_{comm} = t_s + l t_h + t_w m.$$

- Improvement over store-and-forward routing
- If the communication is between nearest neighbors (that is, $l = 1$), or if the message size is small, then the communication time is similar for store-and- forward
- Most current parallel computers & many LANs support cut-through routing.
  - The size of a flit is determined by a variety of network parameters.
  - The control circuitry must operate at the flit rate.
  - Select a very small flit size, for a given link bandwidth, the required flit rate becomes large.
  - As flit sizes become large, internal buffer sizes increase (and the latency of message transfer)
  - Flit sizes in recent cut-through interconnection networks range from four bits to 32 bytes.
- In many parallel programming paradigms that rely predominantly on short messages (such as cache lines), the latency of messages is critical.
- Routers are using multilane cut-through routing.
  - In multilane cut-through routing, a single physical channel is split into a no.f virtual channels.

Deadlocks in cut-through routing

- While traversing the network, if a message needs to use a link that is currently in use, then the message is blocked.
  - This may lead to deadlock.
- Fig. illustrates a deadlock in a cut-through routing network.
  - The destinations of messages 0, 1, 2, and 3 are A, B, C, and D, respectively.
  - A flit from message 0 occupies the link CB (and the associated buffers).
  - Since link BA is occupied by a flit from message 3, the flit from message 0 is blocked.
  - Similarly, the flit from message 3 is blocked since link AD is in use.
  - No messages can progress in the network and the network is deadlocked.
- Can be avoided by using appropriate routing techniques & message buffers.



Reducing the Cost

- The equation of cost of communicating a message between two nodes *l* hops away using cut-through routing implies that in order to optimize the cost of message transfers:

1. Communicate in bulk:
   - instead of sending small messages and paying a startup cost *ts* for each, aggregate small messages into a single large message and amortize the startup latency across a larger message.
   - Because on typical platforms such as clusters and message-passing machines, the value of t_s is much larger than those of t_h or t_w.
2. Minimize the volume of data.
   - To minimize the overhead paid in terms of per-word transfer time t_w, it is desirable to reduce the volume of data communicated as much as possible.
3. Minimize distance of data transfer.
   - Minimize the number of hops *l* that a message must traverse.

- First 2 objs are relatively easy to achieve, 3 is difficult (unnecessary burden alg.designer)
  - In mess-pass lib. (e.g MPI), the programmer has little control on the mapping of processes onto physical processors.
    - In such paradigms, while tasks might have well defined topologies and may communicate only among neighbors in the task topology, the mapping of processes to nodes might destroy this structure.
  - Many architectures rely on randomized (two-step) routing,
    - A message is first sent to a random node from source and from this intermediate node to the destination.
    - This alleviates hot-spots & contention on the network.
    - Minimizing number of hops in a randomized routing network yields no benefits.
  - The per-hop time (t_h) is typically dominated either by the startup latency (t_s)for small messages or by perword component ($t_w m$) for large messages.
    - Since the max no. hops (l) in most networks is relatively small, the per-hop time can be ignored

Simplified cost model

- Cost of transferring a message between two nodes on a network is given by:

$$t_{comm} = t_s + t_w m$$

- It takes the same amount of time to communicate between any pair of nodes => it corresponds to a completely connected network.
- Instead of designing algs for each specific arch (a mesh, hypercube, or tree), we design algs with this cost model in mind & port it to any target parallel comp.
- Loss of accuracy (or fidelity) of prediction when the alg is ported from our simplified model (for a completely connected netw) to an actual machine arch.
  - If our initial assumption that the t_h term is typically dominated by the t_s or t_w terms is valid, then the loss in accuracy should be minimal.
- Valid only for uncongested networks.
  - Architectures have varying thresholds for when they get congested;
  - a linear array has a much lower threshold for congestion than a hypercube.
- Valid only as long as the communication pattern does not congest the network.
  - Different communication patterns congest a given network to different extents.

Effect of congestion on communication cost

- Consider a sqrt(*p*)xsqrt(*p*) mesh in which each node is comm.with its nearest neighbor.
  - Since no links in the network are used for more than one communication, the time for this operation is $t_s + t_w m$, where *m* is the number of words communicated.
  - This time is consistent with our simplified model.
- Consider a scenario in which each node is communic. with a randomly selected node.
  - This randomness implies that there are *p*/2 communications (or *p*/4 bi-directional communications) occurring across any equi-partition of the machine.
  - A 2-D mesh has a bisection width of sqrt(*p*).
  - Some links would now have to carry at least sqrt(*p*)/4 mess.on bidirectional communic.channels
  - These messages must be serialized over the link.
  - If each message is of size *m*, the time for this operation is at least $t_s + t_w m$ x sqrt(*p*)/4.
  - This time is not in conformity with our simplified model.

⇒ For a given arch., some communic. patterns can be non-congesting & others congesting

⇒ This makes the task of modeling communic. costs dependent not just on the architecture, but also on the communication pattern.

⇒ To address this, we introduce the notion of *effective bandwidth*.

  - For communication patterns that do not congest the network, is identical to the link bandwidth.
  - For communication operations that congest the network,is the link bandwidth scaled down by the degree of congestion on the most congested link.
    - Difficult to estimate: it is a fct.of process to node mapping, routing algorithms, & communic. schedule.
    - Therefore, we use a lower bound on the message communication time:
      The associated link bandwidth is scaled down by a factor *p/b, b* is the bisection width of the network.

A Performance Model to Prove the Scalability of MPP

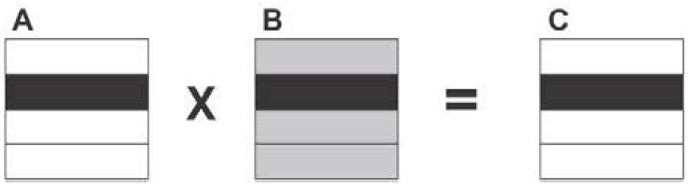
- Goals of communication network of a MPP:
  - provide a communication layer that is fast, well balanced with the no. & performance of processors, and homogeneous.
  - should cause no degradation in communication speed even when al processors of the MPP simultaneously perform intensive data transfer operations.
  - should also ensure the same speed of data transfer between any two processors of the MPP.

- Case study: parallel matrix-matrix multiplication
  - $C = A \times B$ on a p-processor MPP, where A, B are dense square n x n matrices,
  - the matrix-matrix multiplication involves $O(n^3)$ operations.
  - the total execution time of the parallel algorithm is

$$t_{\text{comp}} = \frac{t_{\text{proc}} \times n^3}{p},$$

where t_{proc} characterizes the speed of a single processor.

Matrix-matrix multiplication with matrices evenly partitioned in one dimension



- Each element c_{ij} in C is computed as $c_{ij} = \Sigma_{k=0}^{n-1} a_{ik} \times b_{kj}$.
- The A, B, and C matrices are evenly (and identically) partitioned into p horizontal slices (for simplicity we assume that n is a multiple of p). There is one-to-one mapping between these slices and the processors. Each processor is responsible for computing its C slice (see Figure).
- In order to compute elements of its C slice, each processor requires all elements of the B matrix. Therefore, during the execution of the algorithm, each processor receives from each of $p - 1$ other processors n^2/p matrix elements (shown in gray in Figure).

- Assume that the time of transfer of a data block is a linear function of the size of the data block.
  - the cost of transfer of a single horizontal slice between two processors is

$$t_{\text{slice}} = t_s + t_e \times \frac{n^2}{p},$$

  - where *ts* is the start-up time and *te* is the time per element.
- Each proc sends its slice to *p* - 1 other procs as well as receives their slices
  - Assume that the proc can be simultaneously sending a single message & receiving another single message (double-port model).
  - Pessimistic assumption that the processor sends its slice to other processors in *p* - 1 sequential steps
  - The estimation of the per-processor communication cost is

$$t_{\text{comm}} = (p-1) \times t_{\text{slice}} \approx t_s \times p + t_e \times n^2.$$

…

Matrix-matrix multiplication with matrices evenly partitioned in one dimension

- Assume that communications and computations do not overlap.
  - All of the communications are performed in parallel, and then all of the computations are performed in parallel.
- The total execution time of the parallel algorithm is

$$t_{\text{total}} \approx t_{\text{proc}} \times \frac{n^3}{p} + t_s \times p + t_e \times n^2 .$$

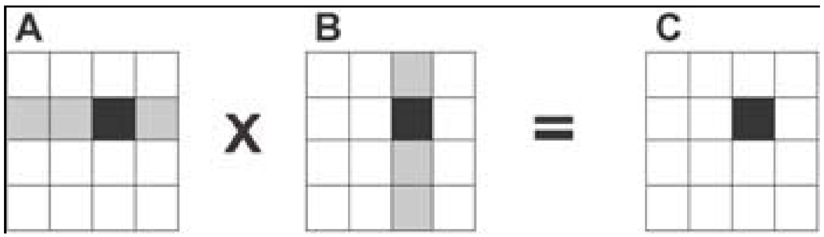
- Scalability: how to ensure faster execution of the alg on a (p + 1)-proc. configuration compared with the p-processor configuration (p = 1, 2, . . .)?
  - The algorithm must ensure speedup at least while upgrading the MPP from a one-processor to a two-processor configuration.
    - It means that $t_{\text{proc}} \times n^3 - \left(t_{\text{proc}} \times \frac{n^3}{2} + t_s + t_e \times n \frac{n^2}{2} \right) = t_{\text{proc}} \times \frac{n^3}{2} - t_s - t_e \times \frac{n^2}{2} > 0.$

    Typically $t_s/t_{\text{proc}} \sim 10^3$ and $t_e/t_{\text{proc}} \sim 10$.
    The following inequality will be comfortably satisfied if $n > 100$: $n^3 > 2 \times \frac{t_s}{t_{\text{proc}}} + \frac{t_e}{t_{\text{proc}}} \times n^2$
  - The algorithm will be scalable, if t_{total} is a monotonically decreasing function of p, that is, if

    $$\frac{\partial t_{\text{total}}}{\partial p} = t_s - t_{\text{proc}} \times \frac{n^3}{p^2} < 0, \quad \text{or} \quad \frac{t_s}{t_{\text{proc}}} \times \left(\frac{p}{n} \right)^2 \times \frac{1}{n} < 1.$$

    The inequality above will be true if n is reasonably larger than p.

2D decomposition of matrices instead of 1D



- Each element c_{ij} in C is computed as $c_{ij} = \Sigma_{k=0}^{n-1} a_{ik} \times b_{kj}$.
- The A, B, and C matrices are identically partitioned into p equal $n/\sqrt{p} \times n/\sqrt{p}$ squares so that each row and each column contain $\sqrt{p}$ squares (for simplicity we assume that p is a square number and n is a multiple of $\sqrt{p}$). There is one-to-one mapping between these squares and the processors. Each processor is responsible for computing its C square (see Figure 4.3).
- To compute elements of its C square, each processor requires the corresponding row of squares of the A matrix and column of squares of the B matrix (shown in gray in Figure 4.3). Therefore, during the execution of the algorithm, each processor receives from each of its $\sqrt{p} - 1$ horizontal and $\sqrt{p} - 1$ vertical neighbors n^2/p matrix elements.

- Total per-processor communication cost,

$$t_{\mathrm{comm}} = 2 \times (\sqrt{p} - 1) \times \left(t_s + t_e \times \frac{n^2}{p} \right) \approx 2 \times t_s \times \sqrt{p} + 2 \times t_e \times \frac{n^2}{\sqrt{p}},$$

- Total execution time of that parallel alg,

$$t_{\mathrm{total}} \approx t_{\mathrm{proc}} \times \frac{n^3}{p} + 2 \times t_s \times \sqrt{p} + 2 \times t_e \times \frac{n^2}{\sqrt{p}},$$

- Considerably less than in the 1D alg
- Further improvements can be made
  - to achieve overlapping communications and computations
  - better locality of computation during execution of the algorithm.
- 2D alg is efficient and scalable for any reasonable task size and no. processors.
- Conclusion:
  - MPP scalable when executing carefully designed and highly efficient parallel algs.

Communication Costs in Shared-Address-Space Machines

- Difficulty reasons:
  - Memory layout is typically determined by the system.
  - Finite cache sizes can result in cache thrashing.
  - Overheads associated with invalidate and update operations are difficult to quantify.
  - Spatial locality is difficult to model.
  - Prefetching can play a role in reducing the overhead associated with data access.
  - False sharing is often an important overhead in many programs.
  - Contention in shared accesses is often a major contributing overhead in shared address space machines.
- Building these into a single cost model results in a model that is
  - too cumbersome to design programs for and
  - too specific to individual machines to be generally applicable.
- A simplified model presented above accounts primarily for remote data access but does not model a variety of other overheads (see the textbook)

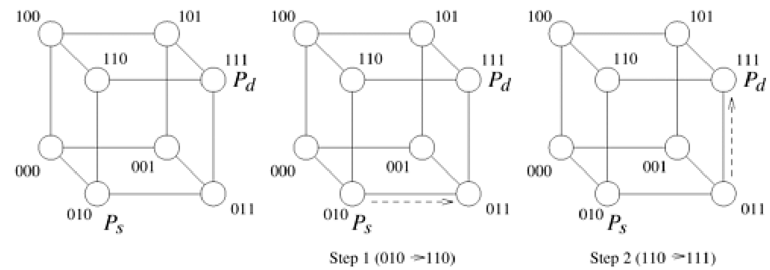
Routing Mechanisms for Interconnection Networks

- Critical to the performance of parallel computers.
- A *routing mechanism*
  - Det. the path a mess. takes through the netw to get from source to destination.
  - It takes as input a message's source and destination nodes.
  - It may also use information about the state of the network.
  - It returns one or more paths through the netw from the source to the destination
- Classification based on route selection:
  - A *minimal* routing mechanism
    - always selects one of the shortest paths between the source and the destination.
    - each link brings a message closer to its destination,
    - can lead to congestion in parts of the network.
  - A *nonminimal* routing scheme
    - may route the message along a longer path to avoid network congestion.
- Classification on the basis on information regarding the state of the network:
  - A *deterministic routing* scheme
    - determines a unique path for a message, based on its source and destination.
    - It does not use any information regarding the state of the network.
    - may result in uneven use of the communication resources in a network.
  - An *adaptive routing* scheme
    - uses information regarding the current state of the netw to determine the path of the mes
    - detects congestion in the network and routes messages around it

Dimension-ordered routing

- Commonly used deterministic minimal routing technique
- Assigns successive channels for traversal by a message based on a numbering scheme determined by the dimension of the channel.
- For a two-dimensional mesh is called *XYrouting*
- For a hypercube is called *E-cube routing*.
- XY-routing:
  - Consider a two-dimensional mesh without wraparound connections.
  - A message is sent first along the X dimension until it reaches the column of the destination node and then along the Y dimension until it reaches its destination.
  - Let $P_{Sy,Sx}$ represent the position of the source node and $P_{Dy,Dx}$ represent that of the destination node.
  - Any minimal routing scheme should return a path of length $|Sx - Dx| + |Sy - Dy|$.
  - Assume that $Dx >= Sx$ and $Dy >= Sy$.
  - The message is passed through intermediate nodes $P_{Sy,Sx+1}$, $P_{Sy,Sx+2}$, ..., $P_{Sy,Dx}$ along the X dimension and
  - Then through nodes $P_{Sy+1,Dx}$, $P_{Sy+2,Dx}$, ..., $P_{Dy,Dx}$ along the Y dimension to reach the destination.

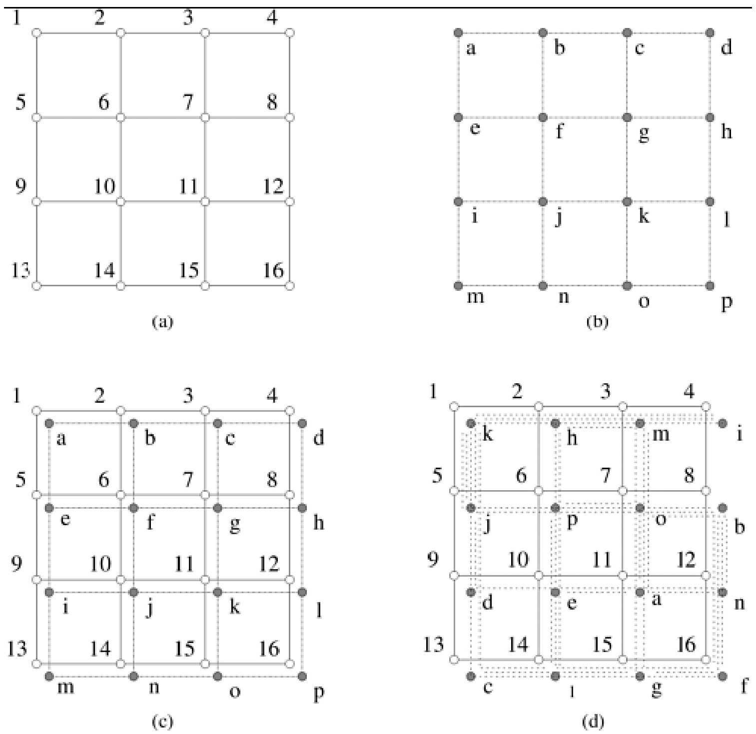
E-cube routing



- Consider a *d*-dimensional hypercube of *p* nodes.
- Let P_s and P_d be the labels of the source and destination nodes
- We know that the binary representations of these labels are *d* bits long.
- The minimum distance between these nodes is given by the number of ones in P_s o P_d , where o represents the bitwise exclusive-OR operation.
- Node P_s computes P_s o P_d and sends the message along dimension *k*, where *k* is the position of the least significant nonzero bit in P_s o P_d .
- At each intermediate step, node P_i , which receives the message, computes P_i o P_d and forwards the message along the dimension corresponding to the least significant nonzero bit.
- This process continues until the message reaches its destination.
- Example – Fig.
  - Let P_s = 010 and P_d = 111 represent the source and destination nodes for a message.
  - Node P_s computes 010 o 111 = 101.
  - In the first step, P_s forwards the message along the dimension corresponding to the least significant bit to node 011.
  - Node 011 sends the message along the dimension corresponding to the most significant bit (011 o 111 = 100).
  - The message reaches node 111, which is the destination of the message.

Impact of process-processor mapping

- Problem:
  - A programmer often does not have control over how logical processes are mapped to physical nodes in a network.
    - even communication patterns that are not inherently congesting may congest the netw.
- Example – fig.
  - (a) underlying architecture;
  - (b) processes and their interactions;
  - (c) an intuitive mapping of processes to nodes:
    - a single link in the underlying architecture only carries data corresponding to a single communication channel between processes.
  - (d) a random mapping of processes to nodes:
    - each link in the machine carries up to six channels of data between processes.
    - considerably larger communication times if the required data rates on communication channels between processes is high

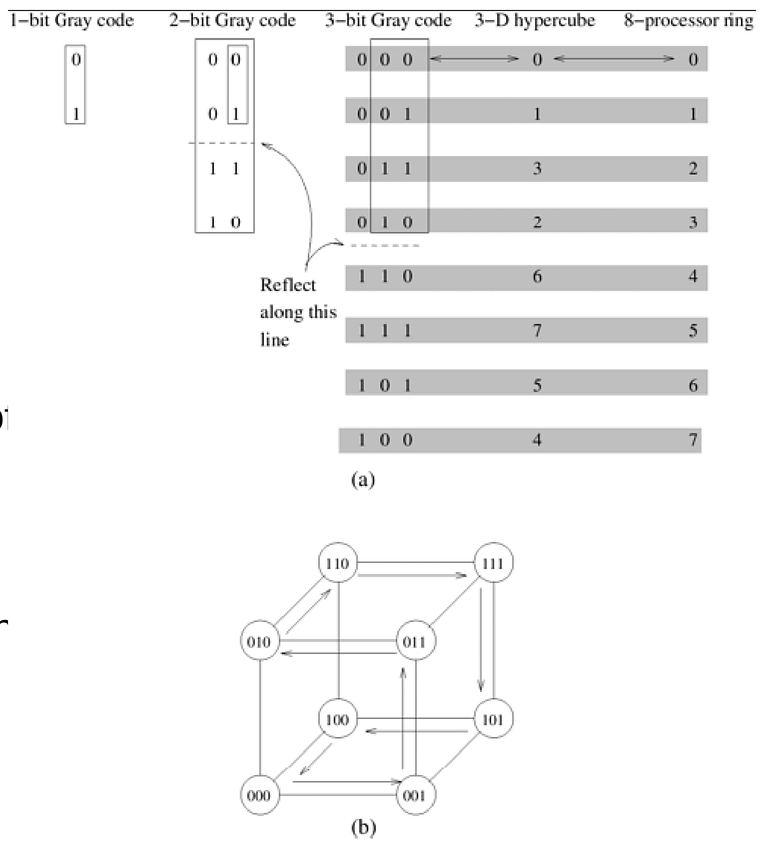


Mapping Techniques for Graphs

- Given 2 graphs, *G*(*V, E*), *G*'(*V*', *E*'), mapping graph *G* into graph *G*' maps
  - each vertex in the set *V* onto a vertex (or a set of vertices) in set *V*' and
  - each edge in the set *E* onto an edge (or a set of edges) in *E*'.
- 3 parameters are important:
  - Congestion of the mapping:
    - The maximum number of edges mapped onto any edge in *E*'
      - it is possible that more than one edge in *E* is mapped onto a single edge in *E*'.
  - Dilatation of the mapping:
    - The maximum number of links in *E*‘ that any edge in *E* is mapped onto
      - An edge in *E* may be mapped onto multiple contiguous edges in *E*'.
      - This is significant because traffic on the corresponding communication link must traverse more than one link, possibly contributing to congestion on the network.
  - Expansion of the mapping:
    - The ratio of the number of nodes in the set *V*' to that in set *V* is called the.
      - Third, the sets *V* and *V*' may contain different numbers of vertices. In this case, a node in *V* corresponds to more than one node in *V*'.
      - The expansion of the mapping must be identical to the ratio of virtual&physical procs.

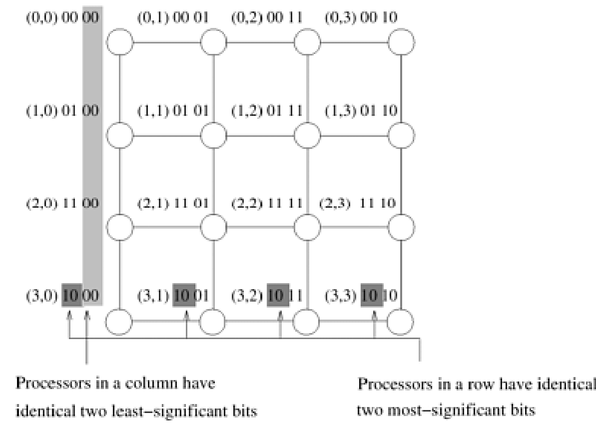
Embedding a Linear Array into a Hypercube

- A linear array/ring composed of 2^d nodes ($0: 2^d - 1$) can be embedded into a *d*-dim. hypercube by mapping node *i* onto node *G*(*i*, *d*)
- *G* : the binary reflected Gray code (RGC).
  - The entry *G*(*i*, *d*) denotes the *i*th entry in the sequence of Gray codes of d bits.
  - Gray codes of *d* + 1 bits are derived from Gray codes of *d* bits by reflecting the table & prefixing the reflected entries with a 1 & the original entries with a 0.
  - Adjoining entries (*G*(*i, d*) and *G*(*i* + 1, *d*)) differ from each other at only one bit position.
- Node *i* in the linear array is mapped to node *G*(*i, d*), and node *i* + 1 is mapped to *G*(*i* + 1, *d*) => is a direct link in the hypercube that corresponds to each direct link in the linear array.
  - Mapping specified by the function *G* has a dilation of one and a congestion of one.
- Figure (b) : the embedding of an eight-node ring into a three-dimensional hypercube.

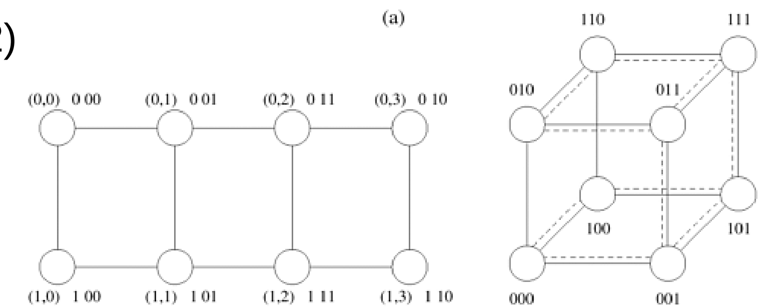


Embedding a Mesh into a Hypercube

- Natural extension of embedding a ring into a hypercube.
  - Embed a 2r x 2s wraparound mesh into a 2r+s -node hypercube by mapping node (*i,* j) of the mesh onto node *G*(*i, r* - 1)||*G*(*j, s* - 1) of the hypercube (where || denotes concatenation of the two Gray codes).
  - Immediate neighbors in the mesh are mapped to hypercube nodes whose labels differ in exactly one bit position => mapping has a dilation of one and a congestion of one.
- Example: a 2 x 4 mesh into an eight-node hypercube.
  - Node (*i, j*) of the mesh is mapped to node *G*(*i*, 1)||*G*(*j*, 2) of the hypercube
  - Node (0, 0) of the mesh is mapped to node 000 of the hypercube, because *G*(0, 1) is 0 and *G*(0, 2) is 00;
  - Node (0, 1) of the mesh is mapped to node 001 of the hypercube
- Figure illustrates embedding meshes into hypercubes:
  (a) a 4 x 4 mesh to the nodes in a four-dimensional hypercube; and
  (b) a 2 x 4 mesh embedded into a three-dimensional hypercube.



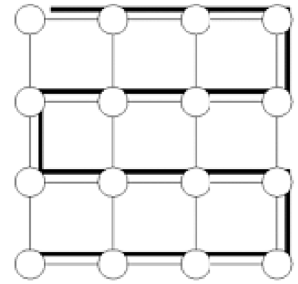
(a)



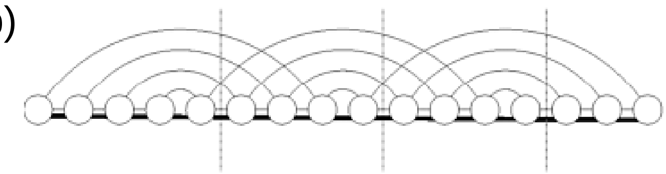
(b)

Embedding a Mesh into a Linear Array

- An intuitive mapping of a linear array into a mesh is illustrated in Figure (a):
  - the solid lines correspond to links in the linear array and
  - normal lines to links in the mesh.
  - congestion-one, dilation-one mapping of a linear array to a mesh is possible.
- Consider now the inverse of this mapping,
  - given a mesh, map vertices of the mesh to those in a linear array using the inverse of the same mapping function-Fig. (b)
  - solid lines correspond to edges in the linear array and normal lines to edges in the mesh.
  - the congestion of the mapping in this case is five – i.e., no solid line carries more than five normal
  - In general: the congestion of this (inverse) mapping is sqrt(*p*)+1 for a general p-node mapping (one for each of the sqrt(*p*) edges to the next row, and one additional edge).
- We can do better?
  - Congestion of any mapping is lower bounded by sqrt(*p*)
    - see textbook - why
  - In general:
    - the lower bound on congestion of a mapping of network *S* with *x* links into network *Q* with y links is *x/y*.
    - In the case of the mapping from a mesh to a linear array, this would be 2*p/p*, or 2.



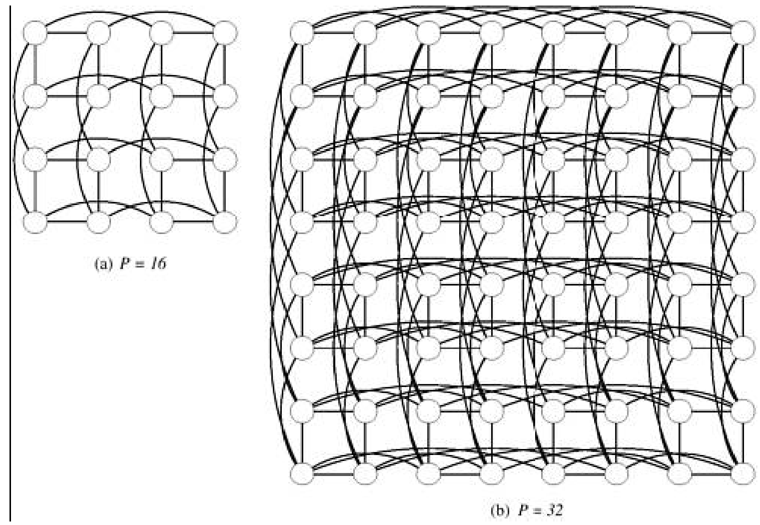
(a) Mapping a linear array into a 2D mesh (congestion 1).



(b) Inverting the mapping – mapping a 2D mesh into a linear array (congestion 5)

Embedding a Hypercube into a 2-D Mesh

- p-node hypercube into a p-node 2-D mesh, *p* is an even power of two.
- visualize the hypercube as sqrt(*p*) subcubes, each with sqrt(*p*) nodes.
  - let *d* = log *p* be the dim of the hypercube.
  - take the *d*/2 least significant bits and use them to define individual subcubes of sqrt(*p*) nodes.
  - For example, in the case of a 4D hypercube, we use the lower two bits to define the subcubes as (0000, 0001, 0011, 0010), (0100, 0101, 0111, 0110), (1100, 1101, 1111, 1110), and (1000, 1001, 1011, 1010).
- The (row) mapping from a hypercube to a mesh can now be defined as follows:
  - Each sqrt(*p*) node subcube is mapped to a sqrt(*p*) node row of the mesh.
  - We do this by simply inverting the linear-array to hypercube mapping.
  - The congestion: sqrt(*p*)/2
- Fig: *p* = 16 and *p* = 32
- Column mapping:
  - We map the hypercube nodes into the mesh in such a way that nodes with identical *d*/2 least significant bits in the hypercube are mapped to the same column.
  - a congestion of sqrt(*p*)/2.



Cost-Performance Tradeoffs

- Remark: it is possible to map denser networks into sparser networks with associated congestion overheads.
  - a sparser network whose link bandwidth is increased to compensate for the congestion can perform as well as the denser network (modulo dilation effects).
  - Example: a mesh whose links are faster by a factor of sqrt(*p*)/2 will yield comparable performance to a hypercube => call it fat-mesh.
    - A fat-mesh has the same bisection-bandwidth as a hypercube;
    - However it has a higher diameter.
    - Using appropriate message routing technqs, the effect of node distance can be minimized.
- Analyzing the performance of a mesh & a hypercube netw with identical costs:
  - Cost of a network is proportional to the number of wires,
    - a square p-node wraparound mesh with (log *p*)/4 wires per channel costs as much as a p-node hypercube with one wire per channel.
  - Average communication times of these two networks.
    - See textbook
  - For *p* > 16 and sufficiently large message sizes, a mesh outperforms a hypercube of the same cost.
    - For large enough messages, a mesh is always better than a hypercube of the same cost, provided the network is lightly loaded.
    - Even when the network is heavily loaded, the performance of a mesh is similar to that of a hypercube of the same cost.