

---

**CSE 412/CS 454/MATH 486**  
**Parallel Numerical Algorithms**  
**5. Interprocessor Communication**

Prof. Michael T. Heath

Department of Computer Science  
University of Illinois at Urbana-Champaign

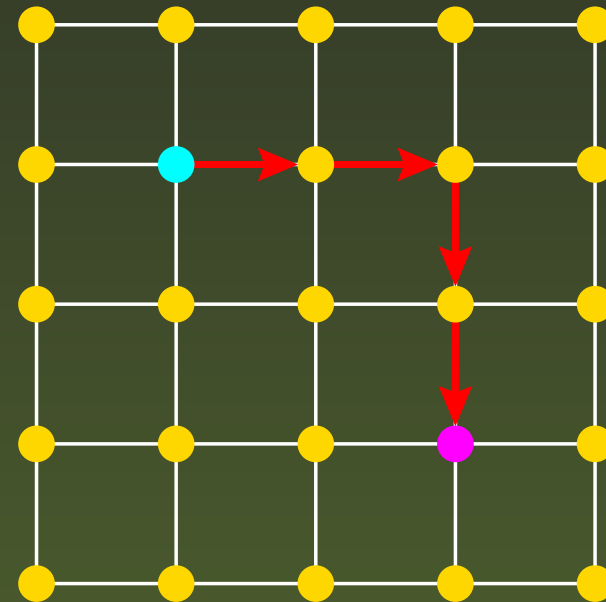
# Message Routing

---

- If message is sent between processors that are not directly connected, then it must be routed through intermediate processors
- Message routing algorithms can be
  - minimal or nonminimal
  - static or dynamic
  - deterministic or randomized
  - circuit switched or packet switched
- Most regular network topologies admit relatively simple routing schemes that are static, deterministic, and minimal

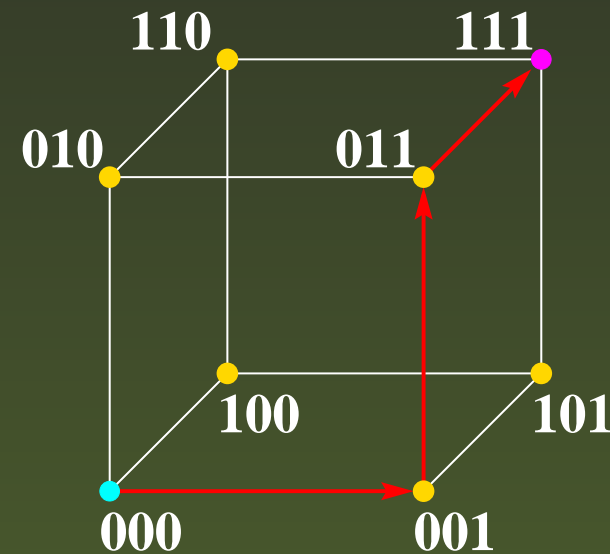
# Example: Routing in Mesh

- In 2-D mesh, message is forwarded along row (or column) of sending node until column (or row) of destination node is reached, then forwarded along destination column (or row) until destination node is reached
- In 3-D mesh, forwarding takes place similarly along each dimension until destination node is reached



# Example: Routing in Hypercube

- In hypercube, if current node number differs from that of destination node in  $i$ th bit, then message is forwarded to adjacent node with opposite value in  $i$ th bit
- Message reaches destination node in  $k$  steps, where  $k$  is number of bit positions in which source and destination node numbers differ, which is at most  $\log(p)$



# Message Routing

---

- There is often considerable freedom in choosing routing scheme
  - In 2-D or 3-D mesh one can take respective dimensions in any order
  - In hypercube, bits that differ between source and destination nodes can be “corrected” in any order
- Thus, there are often multiple possible paths for any given message, and this freedom can sometimes be exploited for improved performance or fault tolerance

# Cut-Through Routing

---

- Early distributed-memory multicomputers used *store-and-forward* routing: at each node along path from source to destination, entire message is received and stored before being forwarded
- Modern communication networks use *cut-through* (or *wormhole*) routing, in which message is broken into smaller segments that are pipelined through network
- Each node on path forwards each segment as soon as it is received, which improves performance and reduces buffer space requirements

# Store-and-Forward vs Cut-Through

## store-and-forward



## cut-through



# Cut-Through Routing

---

- In effect, cut-through routing establishes virtual circuit between source and destination nodes
- Care must be taken in designing routing algorithm to avoid potential deadlock when multiple messages contend for same link
- Cut-through routing makes network distance less important for individual messages, so matching problem topology to network topology is less crucial
- Aggregate bandwidth constraints still necessitate some attention to locality, however

# Communication Concurrency

---

- We have thus far considered only *point-to-point* communication, in which one pair of processors communicate with each other
- If many processors communicate simultaneously, overall performance is affected by degree of concurrency supported by communication system
- It may or may not be possible for processor to
  - send and receive on same link simultaneously
  - send on one link and receive on another link simultaneously
  - send and/or receive on multiple links simultaneously

# Communication Concurrency

---

- We can usually allow for these distinctions by appropriately defining what we mean by “step” of given communication pattern
- Effect is to multiply overall cost by constant factor in network whose degree does not vary with number of processors
- Corresponding factor may grow with number of processors in network having variable degree

# Collective Communication

---

- *Collective communication* involves multiple nodes simultaneously
- Examples occurring frequently include
  - *broadcast*: one-to-all
  - *reduction*: all-to-one
  - *multinode broadcast*: all-to-all
  - *scatter/gather*: one-to-all/all-to-one
  - *total exchange*: personalized all-to-all
  - *scan* or *prefix*
  - *circular shift*
  - *barrier*

# Broadcast

---

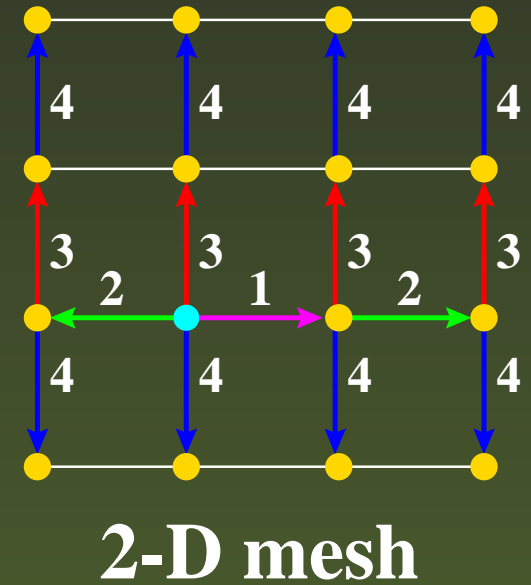
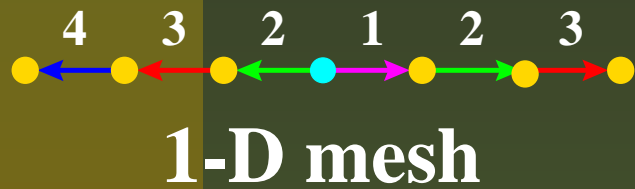
- In *broadcast*, source node communicates single message to  $p - 1$  other nodes
- Source node could send  $p - 1$  separate messages serially, one to each of other nodes
- Efficiency can be improved by exploiting parallelism and fact that messages often need to be routed through intermediate nodes anyway

# Generic Broadcast Algorithm

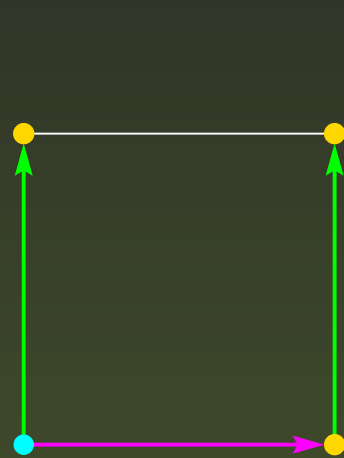
---

1. If source  $\neq$  me, receive message
2. Send message to each of my direct neighbors who have not already received it

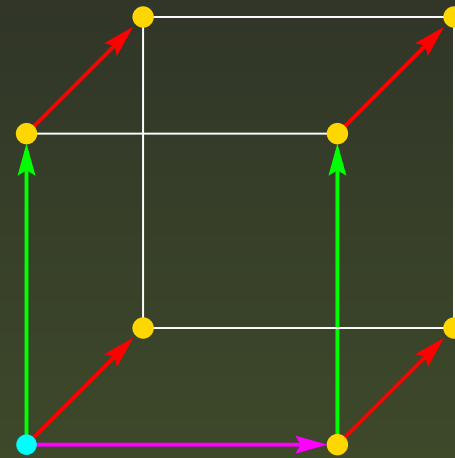
# Broadcast in Mesh or Torus



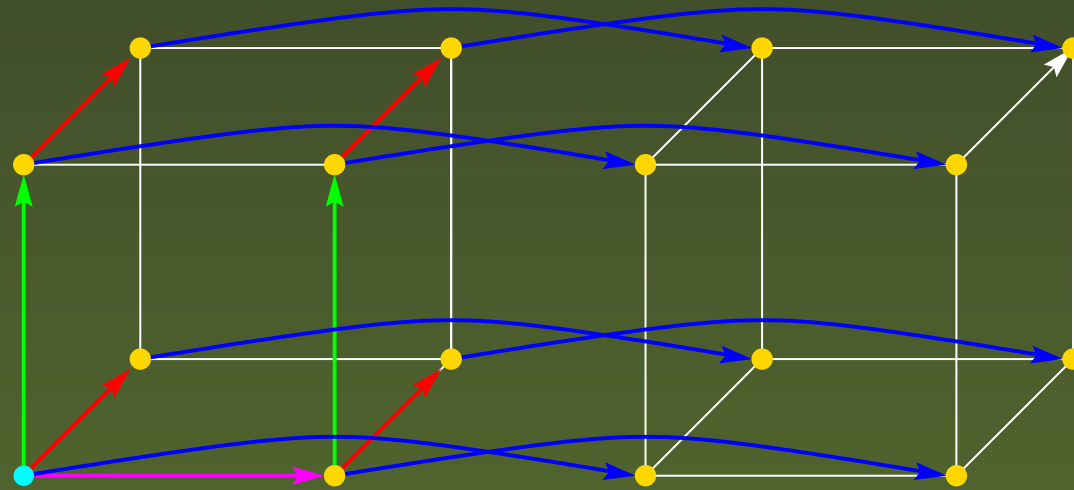
# Broadcast in Hypercube



2-cube



3-cube



4-cube

# Cost of Broadcast

- Broadcast algorithm generates spanning tree for given network, with source node as root
- Height of spanning tree determines total number of steps required
- Cost of broadcast for message of length  $L$  is
  - 1-D mesh:  $(p - 1) (t_s + t_w L)$
  - 2-D mesh:  $2(\sqrt{p} - 1) (t_s + t_w L)$
  - Hypercube:  $\log(p) (t_s + t_w L)$

# Enhanced Broadcast

---

- For long message for which bandwidth dominates latency, network bandwidth may be better exploited by breaking message into pieces and either
  - pipeline pieces along single spanning tree, or
  - send each piece along different spanning tree with same root
- In hypercube with  $2^k$  nodes, with any given node as root, there are  $k$  edge-disjoint spanning trees, all of which can potentially be exploited simultaneously in broadcast

# Reduction

---

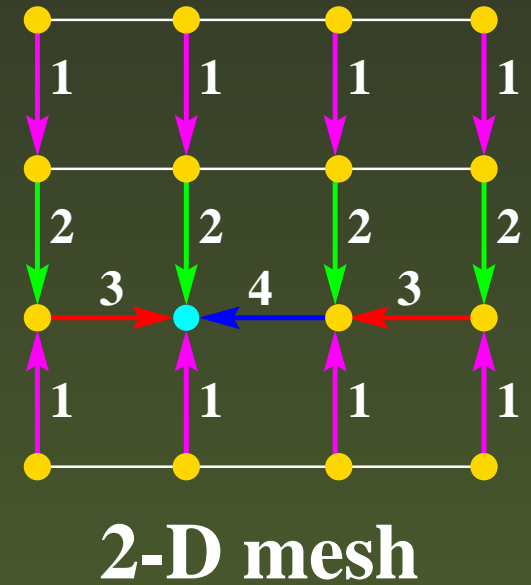
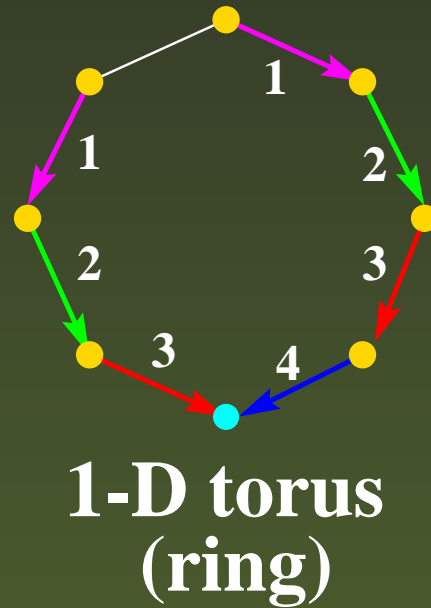
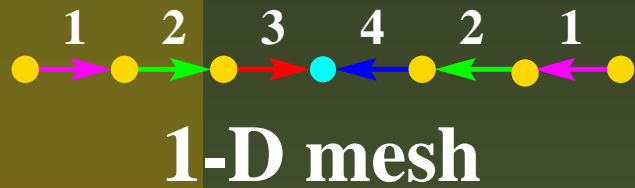
- In *reduction*, data from all  $p$  nodes are combined by applying specified associative operation (e.g., sum, product, max, min, logical OR, logical AND) to produce overall result
- As with broadcast, reduction uses spanning tree for given network, but data flow is in opposite direction, from leaves to root
- Incoming results are combined with receiving node's value before forwarding to its parent
- Final result ends up at root node; if it is also needed by other nodes, final result can be broadcast

# Generic Reduction Algorithm

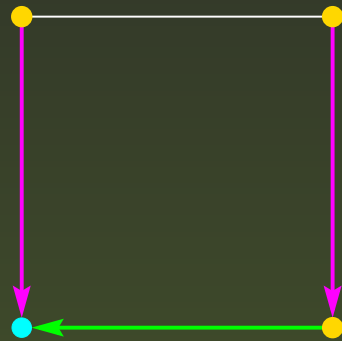
---

1. Receive message from each of my children in spanning tree, if any
2. Combine received values with my own using specified associative operation
3. Send result to my parent, if any

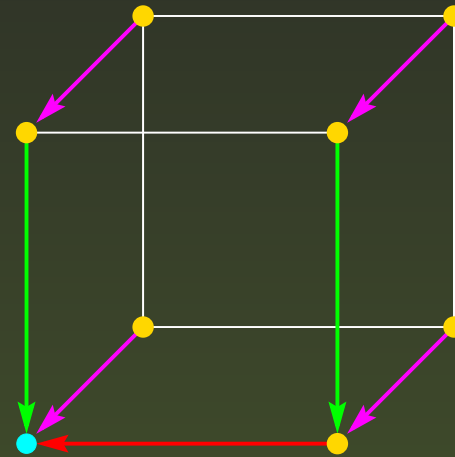
# Reduction in Mesh



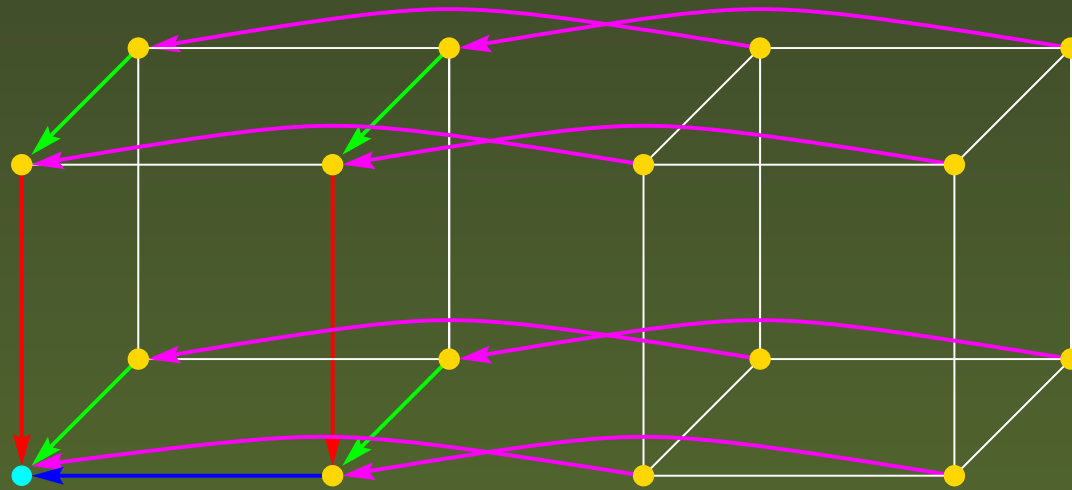
# Reduction in Hypercube



2-cube



3-cube



4-cube

# Cost of Reduction

- Reduction algorithm uses same spanning tree as broadcast, but messages flow in reverse direction
- Height of spanning tree determines total number of steps required
- Cost of reduction for message of length  $L$  is
  - 1-D mesh:  $(p - 1) (t_s + (t_w + t_c)L)$
  - 2-D mesh:  $2(\sqrt{p} - 1) (t_s + (t_w + t_c)L)$
  - Hypercube:  $\log(p) (t_s + (t_w + t_c)L)$

where  $t_c$  is cost per word of associative reduction operation

# Multinode Broadcast

---

- In *multinode broadcast*, each node sends message to all other nodes
- This all-to-all operation is logically equivalent to  $p$  broadcasts, one from each node, and could be implemented that way
- Efficiency can often be improved by overlapping separate broadcasts
- Total cost of multinode broadcast depends strongly on degree of overlap supported by target system
- Multinode broadcast need be no more costly than standard broadcast if aggressive overlapping of communication is supported

# Multinode Broadcast in Torus

---

- In 1-D torus, broadcast can be initiated from each node simultaneously in same direction around ring
- After  $p - 1$  steps, each node has received data from all other nodes, and multinode broadcast is complete, and cost is same as standard broadcast
- In 2-D torus, ring algorithm can be applied first in each row, then in each column (or vice versa)
- There are  $2(\sqrt{p} - 1)$  steps for square 2-D torus
- Messages for second phase are larger by factor of  $\sqrt{p}$ , so total amount of data transferred is still proportional to  $p$

# Multinode Broadcast in Hypercube

---

- In hypercube with  $2^k$  nodes, multinode broadcast can be implemented by successive pairwise exchanges in each of  $k$  dimensions, with messages concatenated at each stage
- There are  $\log(p)$  steps for hypercube, but growth in message sizes means that total communication volume is still proportional to  $p$

# Reduction via Multinode Broadcast

---

- If instead of concatenating messages they are combined using specified associative operation, multinode broadcast can be used to implement reduction
- Since all nodes receive final result, this approach avoids root node having to broadcast it after reduction, thereby saving factor of up to two in cost if result is needed by all nodes

# Personalized Collective Comm.

---

- In broadcast or multinode broadcast, given node sends *same* message to all other nodes
- In analogous *personalized* versions, *distinct* message is sent to each other node
  - *Scatter*: analogous to broadcast, but root sends *distinct* message to each other node
  - *Gather*: analogous to reduction, but data received by root are concatenated rather than combined using associative operation
  - *Total exchange*: analogous to multinode broadcast, but each node exchanges *distinct* message with each other node

# Personalized Collective Comm.

---

- Scatter uses spanning tree algorithm similar to standard broadcast, but multiple messages are transmitted together at each stage
  - Root node sends messages to each of its children containing data for entire subtree of which that child is root
  - Each child retains its own portion of data and forwards appropriate subsets of remainder to each of its children
  - Eventually every node receives its distinct message

# Personalized Collective Comm.

---

- Gather uses algorithms similar to reduction, except data are concatenated at each stage rather than combined using associative operation
- Total exchange uses algorithm similar to multinode broadcast, except broadcasts are replaced by scatter operations, which are overlapped as in multinode broadcast

# Scan or Prefix

- In *scan* or *prefix* operation, data values

$$x_0, x_1, \dots, x_{p-1}$$

are given, one per node, along with specified associative operation  $\oplus$

- Sequence of partial results

$$s_0, s_1, \dots, s_{p-1}$$

is to be computed, where

$$s_k = x_0 \oplus x_1 \oplus \dots \oplus x_k$$

and  $s_k$  is to reside on node  $k$ ,  $k = 0, \dots, p - 1$

# Scan or Prefix

---

- Scan operation can be implemented by algorithms similar to those for multinode broadcast, except that intermediate results received by each node are selectively combined, depending on sending node's numbering, before forwarding

# Circular Shift

---

- In *circular  $k$ -shift*, with  $0 < k < p$ , node  $i$  sends data to node  $(i + k) \bmod p$
- Such operations arise in some finite difference and matrix computations and string matching problems
- Circular shift can be implemented quite naturally in ring network
- Implementing circular shift in other networks can be considerably more complicated, but basically it involves embedding ring or series of rings in given network

# Barrier

---

- *Barrier*: synchronization mechanism in which all processors must reach barrier before any processor is allowed to proceed beyond it
- Implementation of barrier depends on underlying memory architecture and network
- In distributed-memory systems, barrier is usually implemented by message passing, using algorithms similar to those for all-to-all communication
- In shared-memory systems, barrier is usually implemented using test-and-set, semaphore, or other mechanism for enforcing mutual exclusion

# References

---

- M. Barnett, R. Littlefield, D. Payne, and R. van de Geijn, Global combine algorithms for 2-D meshes with wormhole routing, *J. Parallel Distrib. Comput.* 24:191-201, 1995
- M. Barnett, D. Payne, R. van de Geijn, and J. Watts, Broadcasting on meshes with wormhole routing, *J. Parallel Distrib. Comput.* 35:111-122, 1996
- D. P. Bertsekas, C. Ozveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis, Optimal communication algorithms for hypercubes, *J. Parallel Distrib. Comput.* 11:263-275, 1991
- G. E. Blelloch, Scans as primitive operations, *IEEE Trans. Comput.* 38:1526-1538, 1989

# References

---

- V. G. Cerf, Networks, *Scientific American* 265(3):72-81, September 1991
- W. J. Dally and C. L. Seitz, Deadlock-free routing in multiprocessor interconnection networks, *IEEE Trans. Comput.* 36:547-553, 1987
- A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd. ed., Addison-Wesley, 2003
- D. Hensgen, R. Finkel, and U. Manber, Two algorithms for barrier synchronization, *Internat. J. Parallel Prog.* 17:1-17, 1988

# References

---

- S. L. Johnson and C.-T. Ho, Optimum broadcasting and personalized communication in hypercubes, *IEEE Trans. Comput.* 38:1249-1268, 1989
- P. Kermani and L. Kleinrock, Virtual cut-through: a new communication switching technique, *Computer Networks* 3:267-286, 1979
- C. P. Kruskal, L. Rudolph, and M. Snir, The power of parallel prefix, *IEEE Trans. Comput.* C-34:965-968, 1985
- R. E. Ladner and M. J. Fischer, Parallel prefix computation, *J. ACM* 27:831-838, 1980

# References

---

- O. McBryan and E. F. Van de Velde, Hypercube algorithms and implementations, *SIAM J. Sci. Stat. Comput.* 8:s227-s287, 1987
- L. M. Ni and P. K. McKinley, A survey of wormhole routing techniques in direct networks, *IEEE Computer* 26(2):62-76, 1993
- S. Ranka, Y. Won, and S. Sahni, Programming a hypercube multicomputer, *IEEE Software* 69-77, September 1988
- Y. Saad and M. H. Schultz, Data communication in hypercubes, *J. Parallel Distrib. Comput.* 6:115-135, 1989

# References

---

- Y. Saad and M. H. Schultz, Data communication in parallel architectures, *Parallel Computing* 11:131-150, 1989
- Q. F. Stout and B. Wagar, Intensive hypercube communication, *J. Parallel Distrib. Comput.* 10:167-181, 1990
- R. van de Geijn, On global combine operations, *J. Parallel Distrib. Comput.* 22:324-328, 1994