Communication Networks

Oguz Kaya

université
PARIS-SACLAY

Outline

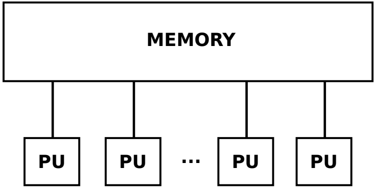
Outline

Objectives

- Introduce the communication cost in to the parallel machine model
- Explore different communication network topologies
- Study communcation algorithms in certain networks

Communication in PRAM

In a PRAM machine, PUs are connected by a shared memory



- If data exchange is needed, we can perform read/write to the same memory location.
- Access to each memory case is in constant time.
- Therefore, communication cost is constant for each pair of PUs.
- This is not a realist model of a real parallel supercomputer.

Parallel machine

In a real parallel machine, the PUs are connected by a communication network

- A shared memory is no longer possible due to hardware constraints.
- The memory is distributed; each PU has its own local memory.
- PUs are connected by links, which constitute the communication network.
- Data exchange is done by explicit communication routines executed on the network
- Data exchange cost between two PUs vary in terms of their position in the network as well as network parameters (bandwidth, latency, topology of connection).

Network topology

The connectivity graph of PUs form the **network topology**.

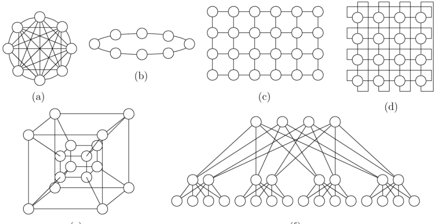


FIGURE 3.1: A few examples of interconnection network topologies: (a) clique; (b) ring; (c) grid; (d) torus; (e) hypercube; (f) fat-tree.

- **Static network:** Links are establish beforehand and never change.
- Examples: Clique, ring, mesh, torus, hypercube.
- **Dynamic networks:** Links can be configured in runtime with switches.
- Examples: Fat-tree, butterfly.
- In general, more links = more efficient and cheaper communication.
- More links = more expensive network cost.
- It is a compromise.

Network parameters

The connectivity graph of PUs form the **topology** of the network that we can classify with certain parameters:

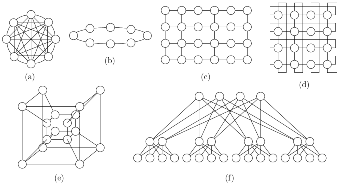


FIGURE 3.1: A few examples of interconnection network topologies: (a) clique; (b) ring; (c) grid; (d) torus; (e) hypercube; (f) fat-tree.

- **Number of nodes/PUs (p):** Number of processors in the network.
- **Degree (k):** Number of links connected to each node/PU. If it is not same for all, we specify (k_{min}, k_{max}) among all nodes.
- **Diameter (D):** Maximum distance among all node pairs.
- **Number of links (N_l):** Total number of links in the network.
- **Bisection width ((L_B):** Number of links to remove in order to divide the network into two

Network parameters

Network parameters of certain networks:

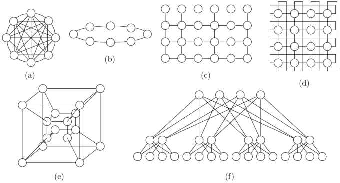


FIGURE 3.1: A few examples of interconnection network topologies: (a) clique; (b) ring; (c) grid; (d) torus; (e) hypercube; (f) fat-tree.

TABLE 3.1: Main characteristics of classical topologies.

Topology	**Num. of proc.** p	**Degree** k	**Diameter** D	**Num. of links** N_l	**Bisec. Width** L_B
Clique	p	$p-1$	1	$p(p-1)/2$	$(p/2)^2$
Ring	p	2	$\lfloor p/2 \rfloor$	p	2
2-D Grid	$\sqrt{p}\sqrt{p}$	$2 \rightarrow 4$	$2(\sqrt{p}-1)$	$2p - 2\sqrt{p}$	$\sqrt{p}$
2-D Torus	$\sqrt{p}\sqrt{p}$	4	$2\lfloor \sqrt{p}/2 \rfloor$	$2p$	$2\sqrt{p}$
Hypercube	$p = 2^d$	$d = \log(p)$	$d = \log(p)$	$p \log(p)/2$	$p/2$

Communication cost

Sending/receiving a message of m bytes by a link is $L + m/B = L + mb$

.

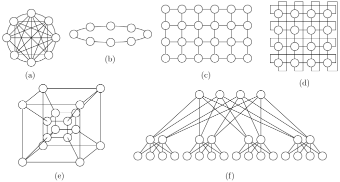


FIGURE 3.1: A few examples of interconnection network topologies: (a) clique; (b) ring; (c) grid; (d) torus; (e) hypercube; (f) fat-tree.

- L: Latency for the preparation and transmission of the message, independent of the message size (in seconds).
- m: Size of the message (in bytes)
- B: Link bandwidth (in bytes/second)
- $b = 1/B$: Inverse link bandwidth (in seconds/byte)
- **Example:** $L = 0.05s$, $B = 10GB/s$ $b = 0.1s/GB$, $m = 1GB$
  - Cost $= L + m/B = 0.05 + 1/10 = 0.15$ seconds.

Outline

Unidirectional ring (cont.)

A topology with P processors, each processor P_i has a link to $P_{(i+1)\%P}$ and is able to execute following routines:

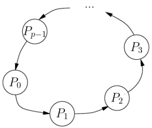


FIGURE 3.7: A unidirectional ring with p processors.

- $\textsc{My\_Num}$: Returns the rank/identifier of each processor, $0 \leq \textsc{My\_Num} < P$.
- $\textsc{Num\_Procs}$: Returns the total number of processors P.
- $\textsc{Send}$ (addr, m): Sends m elements starting from the address *addr* to the processor $P_{(i+1)\%P}$.
- $\textsc{Recv}$ (addr, m): Receives m elements starting from the address *addr* from the processor $P_{(i-1)\%P}$
- Each $\textsc{Send}$ must correspond to a $\textsc{Recv}$ (otherwise a bug)

Unidirectional ring (cont.)

A topology with P processors, each processor P_i has a link to $P_{(i+1)\%P}$ and is able to execute following routines:

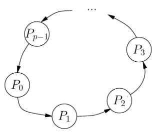


FIGURE 3.7: A unidirectional ring with p processors.

- *addr* is the address of an array in the local memory of the processor.
- We typically suppose that a call to SEND (addr, m) does not block the processor (it continues with the rest of the computation while SEND is executed in the background).
- RECV (addr, m) blocks the processor.

Broadcast on a unidirectional ring

Given the index of a processor k, send a message of size m to all processors

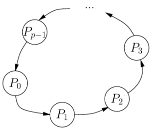


FIGURE 3.7: A unidirectional ring with p processors.

- The idea is to relay the message from left to right while keeping a local copy.

Broadcast on a unidirectional ring (cont.)

Given the index of a processor k, send a message of size m to all processors

```
BROADCAST(k, addr, m)
    q ← MY_NUM()
    p ← NUM_PROCS()
    if q = k then
        SEND(addr, m)
    else
        if q = k − 1 mod p then
            RECEIVE(addr, m)
        else
            RECEIVE(addr, m)
            SEND(addr, m)
```

- The idea is to relay the message from left to right while keeping a local copy.
- Root processor performs one SEND and no RECV.
- Processor before root performs one RECV and no SEND.
- All others perform a RECV followed by a SEND.
- Complexity: $(p-1)(L+mb)$

Scatter on a unidirectional ring

Given a processor index k, send the message at the address $addr[i]$ of size m to each processor $0 \leq l < p$

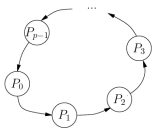


FIGURE 3.7: A unidirectional ring with p processors.

- Idea is still relaying messages from left to right.
- We should try to avoid transfering all messages each time! (slow)
- Send the message from the root in the reverse processor index order.
- The message of $P_{(k-1)\%p}$, then $P_{(k-2)\%p}$, etc.
- **Warning**: Each processor performs a different number of sends and receives!

Scatter on a unidirectional ring (cont.)

Given a processor index k, send the message at the address $addr[i]$ of size m to each processor $0 \leq l < p$

```
SCATTER(k, msg, addr, m)
    q ← MY_NUM()
    p ← NUM_PROCS()
    if q = k then
        for i = 1 to p − 1 do
            SEND(addr[k + p − i mod p], m)
        msg ← addr[k]
    else
        RECEIVE(tempR, m)
        for i = 1 to k − 1 − q mod p do
            tempS ↔ tempR
            SEND(tempS, m) || RECEIVE(tempR, m)
        msg ← tempR
```

- $||$ indicates parallel execution.
- Complexity: $(p-1)(L+mb)$.
- Same as BROADCAST despite different message sent to each processor. Why?
    - Because the network is better utilized.
    - Can we do better BROADCAST then using the same idea? We will see in the exercise session.

All-to-all on a unidireectional network

This time, each processor k has a message to address *my_message* of size m to send to all processors, to put in $addr[k]$ of each processor.

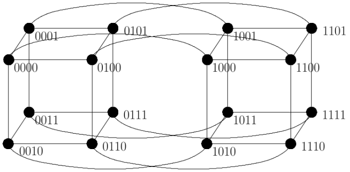
```
ALL_TO_ALL(my_message, addr, m)
    q ←MY_NUM()
    p ←NUM_PROCS()
    addr[q] ←my_message
    for i = 1 to p − 1 do
        SEND(addr[q − i + 1 mod p], m) ||
        RECEIVE(addr[q − i mod p], m)
```

- Equivalent to p BROADCAST s, but we can do better.
- Complexity: $(p-1)(L+mb)$.
- Same as BROADCAST and SCATTER despite transfering more/different messages. Why?
  - Because the network is better utilized.

Outline

Hypercube

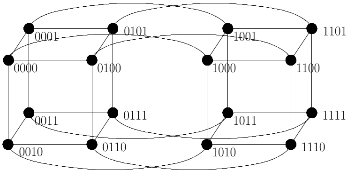
A d-dimensional hypercube , called$d$-cube, consists of



- $p = 2^d$ nodes and $p \log_2 p = 2^d d$ links,
- two $(d-1)$-cubes whose each pair of corresponding nodes i $(0 \leq i < 2^{d-1})$ connected with a link.
- Nodes having exactly $d = \log_2 p$ connections.
- It suffices to flip the bit i $(0 \leq i < d)$ in the binary representation of the rank of a node to find its i.th neighbor.
- **Example:** Neighbors(0000) = {0001, 0010, 0100, 1000}.

Communication on a hypercube

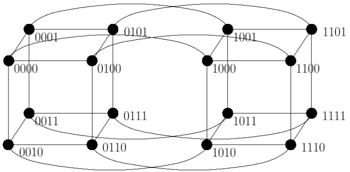
Message transmission on a d-cube:



- Send a message to the neighbor i, SEND (i, addr, m)
- Receive a message from neighbor i, RECV (i, addr, m)
- **Cost** : $L + m/B = L + mb$
- Possible to send a message to any node in at most $d = \log_2 p$ steps maximum.

Broadcast on a hypercube

Idea: Recursive doubling of the message



- Suppose that we broadcast from P_0.
- Initially, BROADCAST is already done on a 0-cube.
- In phase t, BROADCAST will be done on a t-cube.
    - by using messages in the $(t-1)$-cube
    - each node sends its message to its neighbor $(t-1)$
- In phase d, entire BROADCAST will be done.

Broadcast on a hypercube

Idea: Recursive doubling of the message

```
Broadcast(k, addr, m)
    q ← My_Num()
    n ← log(Tot_Proc_Num())
        { Update pos to work as if P_0 was the root of the broadcast }
    pos ← q XOR k
        { Find the rightmost 1 }
    first1 ← 0
    while ((Bit(pos, first1) = 0) And (first1 < n)) do
        first1 ← first1 + 1
        { Core of the algorithm }
    for phase = n − 1 to 0 do
        if (phase = first1) then Receive(phase, addr, m)
        else if (phase < first1) then Send(phase, addr, m)
```

- If *root* is not 0, we can XOR all ranks by k.
- Find the first left bit 1, then Recv the message in that phase.
- Then, in all successive phases, Send the message
- **Cost:** $d(L+mb) = \log_2 p(L+mb)$.

Contact

Oguz Kaya
Université Paris-Saclay and LRI, Paris, France
oguz.kaya@lri.com
www.oguzkaya.com