

# CS8625

Shared Memory Multiprocessors

## CS8625 High Performance and Parallel Computing Dr. Ken Hoganson

*Class*

*Will*

*Start*

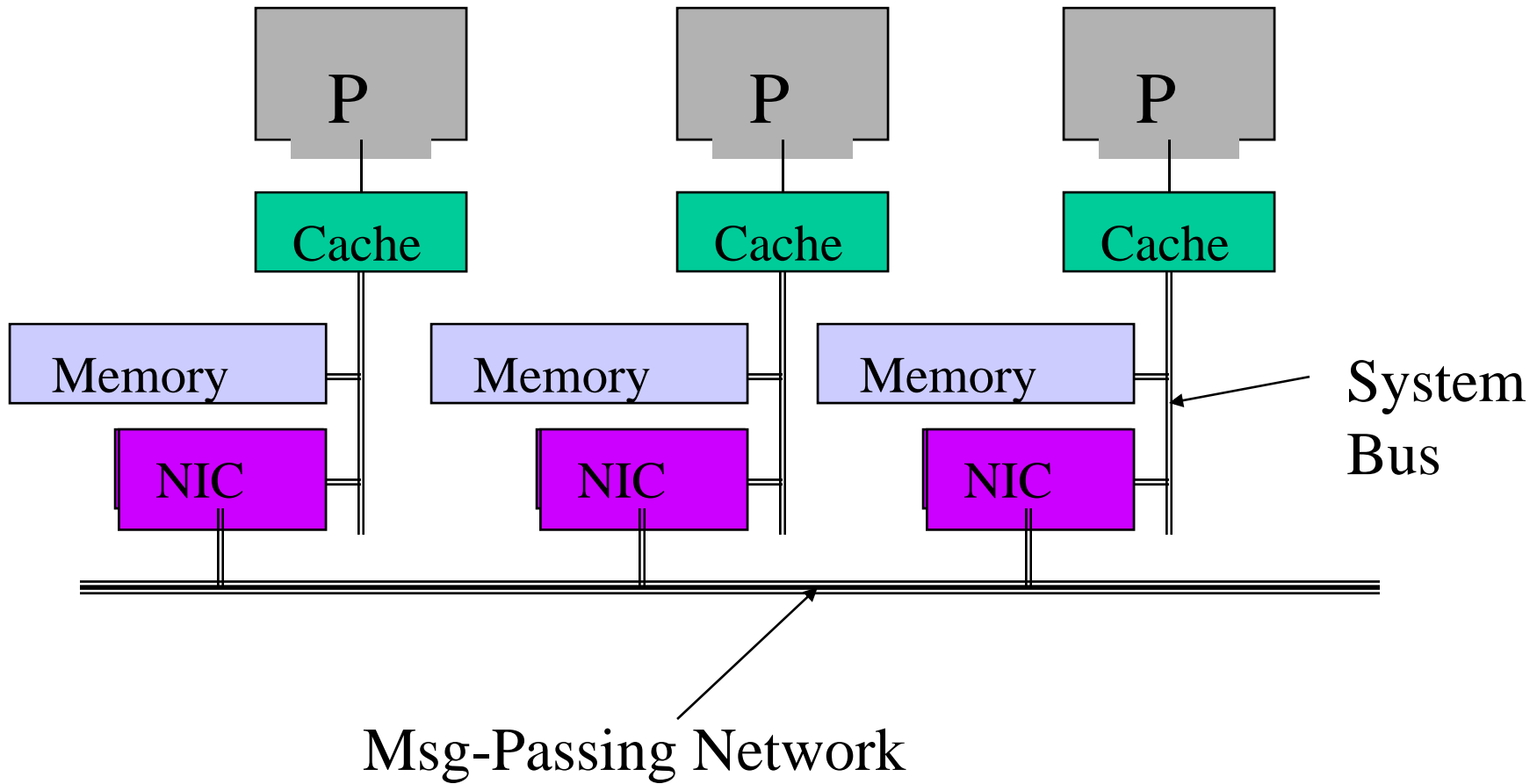
*Momentarily...*

# 1. Message Passing

## A. Discrete Separate memories

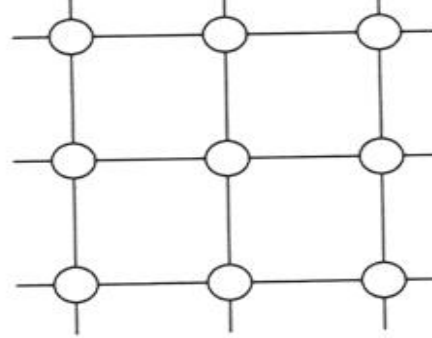
- Exchange data through passing messages
- Slow (high latency):
  1. Encapsulate data
  2. Transmit frame
  3. Extract data

- Autonomous capable
- Loosely coupled
- Longer latency for commo
- Message passing
  
- Appropriate for apps where much computation per unit of commo
- Or when don't care
- Distributed Internet apps example: calculating digits of pi – SPMD Single Program, Multiple Data
- More difficult to do dynamic load balancing (5.1) – how to partition and allocate data and programs (mapping ACT to PCCT)

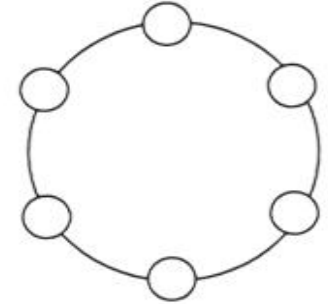




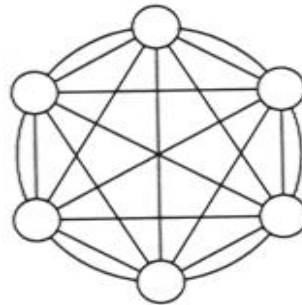
(a) Linear array



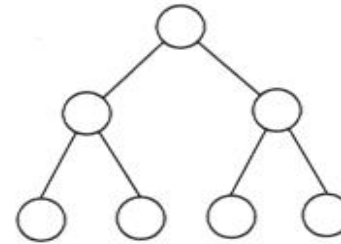
(b) Mesh



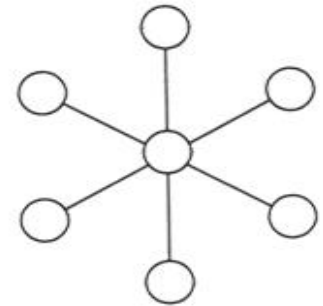
(c) Ring



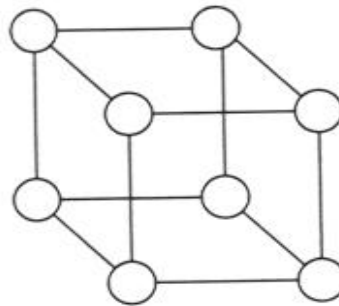
(c) Fully connected



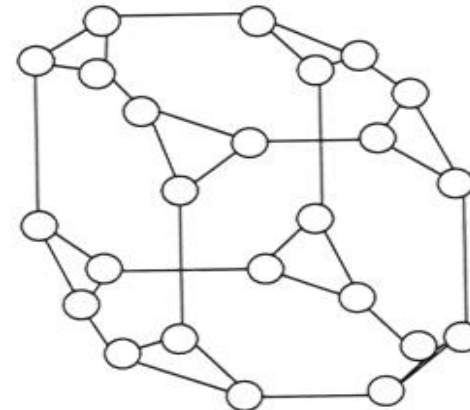
(d) Tree



(e) Star



(f) 3-Cube, hypercube

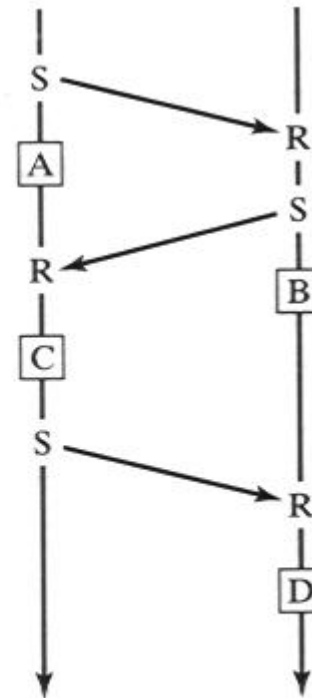


(e) 3-Cube-connected-cycle

Note that the graph shows more time spent in commo, than in computation

Time delay for commo can be non-deterministic

May required synchronization and time consistency



**Figure 5-1** Precedence imposed by interprocess communication.

## How to transmit a message

- One strategy: Store and Forward, Multiple “hops” from node to node
- “Tunneling” create a virtual pathway from source to destination, and send bits out as soon as they arrive – no store.
- Remember: communication is much slower than computation.

If time for communication is:

Number of bits to send/transmission rate

i.e.

10 megabytes = 80 million bits (plus some overhead)

@ 100 Mb/Sec → 0.8 seconds per hop

If 10 hops, then 8.0 seconds for the 80 million bits

- We transmit data in packets or frames, and some protocols acknowledge each frame.
- Lets assume acknowledgements from ultimate receiver, not intermediate nodes passing packets.
- Say for instance, a 1Mbit packet (1Mbit of data)
- Our 80Mbit message would be sent as 80 packets.
- Each packet at 100Mb/sec would take 1/100 second.
- Each packet acknowledged would take another 1/100 second.
- 2/100 second times 80 packets = 1.6 seconds per hop.
- 10 hops = **16.0 seconds.**

- Packet Burst Mode – Covered in CS8422?
- Send multiple packets without pause, then get acknowledge for multiple packets.
- Assumes commo works well.
- If problems with commo, then could be slower with larger retransmission delays.
- If no problems, then saves time.

- Say our "burst" is 10 packets of 1Mbit.
- 1 acknowledgement per 10 packets. (assume acknowledgement is the size of a packet.)
- Acknowledgement overhead is 1 for every 10.
- Our 1Mbyte message – 80Mbits is 80 packets.
- 10 acknowledgements (if no problems)
- Send 90 packets of 1Mbit at 100Mb/sec
- At 100Mb/sec, requires 0.9 seconds
- If 10 hops, **then 9.0 seconds**
- (better than the 16.0 seconds of previous)

- Piggybacking Data with Acknowledgments
- When sending an acknowledgment packet, data going in the reverse direction can ride with the acknowledgment.
- Better efficiency
- Doesn't directly affect our theoretical model performance
- Real-world – better performance because no waits while transmitting in the reverse direction.

- Apply parallel concepts to communication:
- Transmit in parallel: use pipeline concept
- When a communication requires multiple “hops”, don’t wait for each packet to go from sender to receiver.
- Each intermediate node is like a stage in a pipeline.
- Packets flow through the pipeline of node-stages.
- Pipeline startup cost: related to the number of “hops” in the communication  $H = (N-1)$
- After first packet, the remaining arrive without delay.
- Combine with packet burst mode concept.

- 11 nodes, 10 hops.
- 10Mbytes = 80Mbits
- 100Mbit/sec comms latency
- Data partitioned in 1Mbit packets (80 packets)
- Packets grouped in 10 packets per acknowledgment
- First packet is complete at the destination after 10 hops [ $10 * 1\text{Mbit}/100\text{Mbit} = 0.1$  second]
- Next 10 packets arrive in 1/100 second intervals
- 0.2 seconds for 10 packets
- Acknowledgement takes:  $10 * 1\text{Mbit}/100\text{Mbit} = 0.1$  second
- 0.3 seconds per group: 10 groups = **3.0 seconds**
- Much better than 9.0 or 16.0 seconds

- Lower Latency
- Say we have (a true) 1Gbit/second commo performance interconnect

- 11 nodes, 10 hops.
- 10Mbytes = 80Mbits
- 1Gb/sec = 1000Mbit/sec commo latency
- Data partitioned in 1Mbit packets (80 packets)
- Packets grouped in 10 packets per acknowledgment
- First packet is complete at the destination after 10 hops [10 \* 1Mbit/1000Mbit = 0.01 second]
- Next 10 packets arrive in 0.001 second intervals (0.009)
- About 0.02 seconds for 10 packets
- Acknowledgement takes: 10 \* 1Mbit/1000Mbit = 0.01 second
- 0.03 seconds per group: 10 groups = **0.3 seconds**
- Much better than 3.0 seconds at 100Mbit/sec

- Communication slows down the performance of parallel systems
- The slowest communication limits performance
- Previous example we had 0.3 seconds to transmit 10Mbytes at 1Gb/sec.
- The commo time should be a small fraction of the computation time as it is overhead.
- Say commo time is 0.3 seconds, then computation time AT EACH NODE should be about 3 seconds.
- On today's high-speed computers, that is a lot of processing.
- Very approximate – 3Ghz processor say 3 billion instructions per second – 3 seconds of computation is 9 billion instructions – that is a lot!

$$\text{Speedup} = \frac{\text{SerialTime}}{\text{ParallelTime}} =$$

$$= \frac{1}{(1 - \alpha) + \frac{\alpha}{n}}, \quad \text{where}$$

$\alpha$  = fraction of work that can be done in parallel

$n$  = number of processors

- Analytical model of parallel speedup from 1960s
- Parallel fraction ( $\alpha$ ) is run over  $n$  processors taking  $\alpha/n$  time
- The part that must be executed in serial ( $1 - \alpha$ ) gets no speedup
- Overall performance is limited by the fraction of the work that cannot be done in parallel ( $1 - \alpha$ )
- diminishing returns with increasing processors ( $n$ )
- Ignores commo slowdowns

1

-----

$$(1 - \alpha) + \alpha/n + \text{commo overhead}$$

Say commo overhead is 1/10 of parallel computation time  $(\alpha/n)$ , then our commo overhead is directly related to  $\alpha$  and  $n$ .

Our commo overhead works like additional serial time

1

-----

$$(1 - \alpha) + \alpha/n + nC_0(\alpha/n) \quad (\text{where } C_0 = 1/10 * \alpha/n)$$

1

-----

$$(1 - \alpha) + [\alpha(1 + nC_0)]/n$$

- Say our commo overhead is 1/10 the computation time on each node (where  $C_0 = 1/10 * \alpha / n$ )
- Say,  $N=11$  nodes, and alpha = 95%.

1

$$(1 - \alpha) + [\alpha(1 + nC_0)]/n$$

1

$$(0.05) + [0.95(1 + 10 * 1/10)]/11$$

1

$$= 1/0.223 = 4.48 \text{ Speedup}$$

$$(0.05) + 1.9/11$$

Without including commo overhead would be 7.59

- For multicomputers, the communication overhead is a factor to be considered.
- There must be a very large amount of computation compared to the communication time.
- There are numerous examples of science work that fits this criteria
- General business computing usually does not fit this category.

If time for communication is:

Number of bits to send/transmission rate

Without bit tunneling

i.e.

10 megabytes = 80 million bits (plus some overhead)

@ 100 Mb/Sec → 0.8 seconds per hop

If 10 hops, then 8.0 seconds for the 80 million bits

A virtual connection is created to go from one end to the other without any storage or acknowledgement pauses/slowdowns/hops.

As if a dedicated wire from end to end.

Time to transmit is the speed of the protocol plus time delay for signals to flow (say 3/4 speed of light).

i.e.

10 megabytes = 80 million bits (plus some overhead)

@ 100 Mb/Sec → 0.8 seconds for the signaling

Plus the delay for the first bit to appear at the end (like pipelining). SOL=670million miles per hour, 186,000 miles per second

@2000 miles, adds 0.01 seconds.

**End  
Of  
Today's  
Lecture.**