SOFT351 - MPI Communications

Date: 24-11-16

Connection Topologies

- If we have multiple processing cores
  - How can they be connected together?
- If on same PC, they can communicate via main memory bus
  - Effectively every core is connected directly to every other core

Networking hardware

- Once you have computations running in parallel across multiple machines
  - You need a network

Network topologies

- Let's assume connections are *direct*
  - I.e there is a physical wire or fibre between any two connected machines
- How should these connections be organised?
- Simplest option is a fully connected network: connect every node to every other node

Fully connected network

- 6 pcs all connected to every other pc

Properties of this network:

- **Diameter** is the maximum path length between a pair of nodes
  - In our example it is 1
- This is best possible diamter
  - Higher values cause more latency
- **Bisection width** of the network is the minmum number of links you need to cut to divide the network into two equal halves
  - Our example cut down the middle means you have to cut 9 connections
  - So bisection width = 9
- A high value like this is better: more links = more resilience

- The bisection bandwidth is the bisection width multiplied by the bandwidth of a link
  - So in our network, if links were 10GB/s then the bisection bandwidth would be 9 * 10GB/s = 90GB/s
- This is relatively high which is good, affects the performance of many parallel algorthims
- **valency** is how many connections each node makes
  - In this case, each node connects to 5 others, so valency = 5
  - in general, valency = N - 1
- High valency is generally bad - if machines are directly connected then you need this many network ports / cables for each machine
  - Maximum number of ports is 6 or 7
- This means fully connected directly connected network is pretty unfeasible on a large scale
- **Link count** is how many connections the network has in total
  - In this case **link count** = 15
  - In general, **link count** = valency * number of nodes / 2
- The fully connected network is also very bad in this area
  - Shows how many wires you need

Alternatives

- Fully connected network isn't practical
- **Alternatives**
  - Ring
  - Thin tree

Ring Topology

- Each node is connected to two neighbours, forming a closed loop
- What is the **diameter?**
  - In this case: 3
  - in general: N/2 (have to traverse half the network)
- **Not great!** For a large network, messages may have high latency
- What is the **bisection**?
  - In this case: 2, 2 * link bandwidth

  - In general, 2 if it is even

- **Not great!** Not resilient to broken links, low bandwidth
- What is the **Valency**?
  - In this case: 2
  - In general: 2
- **Great!** only need two ports per machine
- What is the **link count?**
  - In this case: 6
  - In general: N
- **Great!** Only need as many cables as there are machines

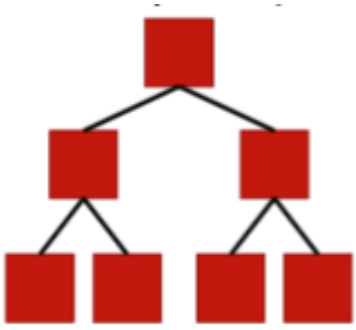
Summary: - Well on practical but poor on performance

Performance - Low bisection width and bisection bandwidth - high diameter

pratical - Cheap to implement - Only need two network ports per machine (low **valence**) - Only need one cable per machine (low **link count**)

Thin tree topology

- Arrange nodes into a tree where each parents has M children
  - Lets take M=2 binary tree



Note: N=8 now!

- What is the **diameter** - this case: 4 - in general: 2 log2 (n + 1) / 2 - **Not bad** grows less than linearly with N

- What is the **bisection**
  - **Cant cut exactly in half** (odd N)
  - In this case: 1
  - In general (for M=2): 1
- **bad** low resilience / bandwidth
- What is the **valency**

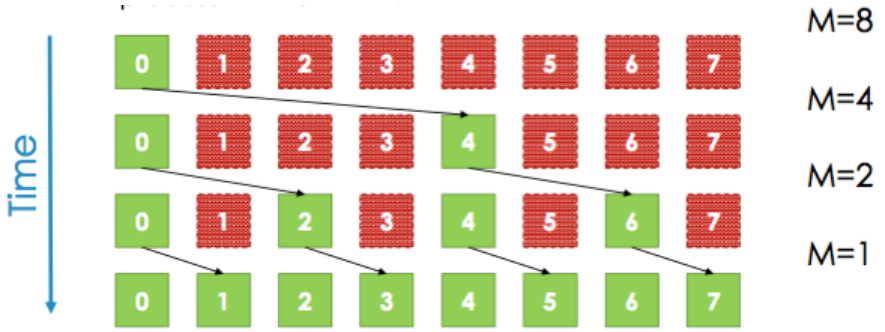
  - In this case: 3
  - In general: M for root, 1 for leaves, M+1 for others
- Only really interested in Maximum valency
- **Not bad**, for binary tree only need 3 network ports max
- What is the **Link count**
  - in this case: 6
- **Not bad**

Thin tree: summary

- Thin tree is compromise between a fully connected network and a ring
- **Good**: low diameter, low valency and low link count
- **Bad**: low bisection width and bisection bandwidth
- This means it will fairly good for latency and quite low cost to implement
  - but the bandwidth and resilience are limited

Distributing data

- Process 0 needs to transmit some data to every other process
- Simplest algorithm: send the data one by one to every process in turn

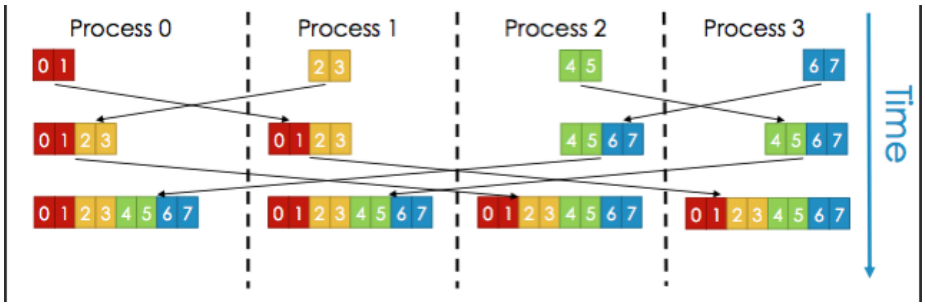


MPI_Bcast - What does it actually do? - It depends on what MPI implementation you're using and how your processes are connected - Some examples" - If processes are on the sample PC, could be *inter-process shared memory* - if using LAN: *Ethernet multicast*

Binary tree

- set M = number of processes

- At every time step:
  - Divide M by 2
  - If a process with rank i has the data, it sends it to the process with rank i+M



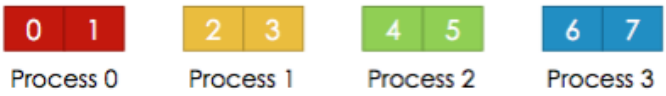
- This method is distributing data is *much* more efficient than the simple method - With N processes that need the data, if one transmit = one time step - **Simple method** takes N-1 time steps, root process does all the work - **Binary tree** takes log2N time steps, work is distributed amongst processes - **E.g.** N=1 million: - *simple method* takes 999,999 time steps - *Binary tree* takes 20

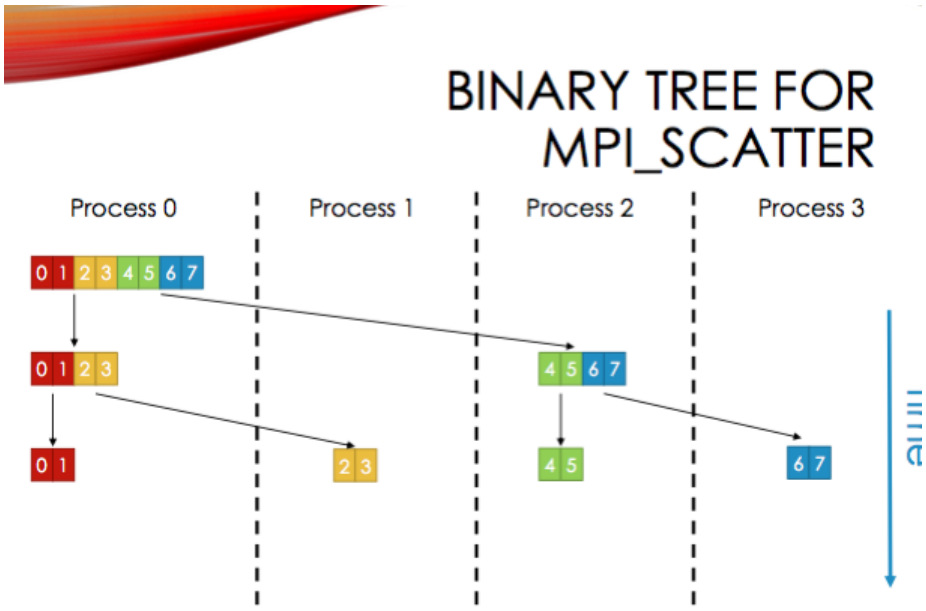
MPI_SCATTER

- MPI_Satter is similar to MP_Bcast but instead of sending the same data to every process it distributes the values in an array amongs the processes

- Example:
  - Process 0 (the root) has an array of eight values:

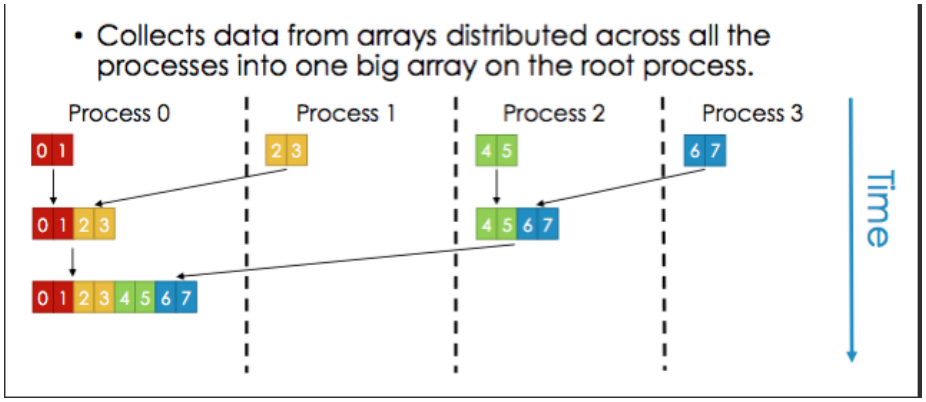
  - Use MPI_Scatter to distribute these values to four processes:





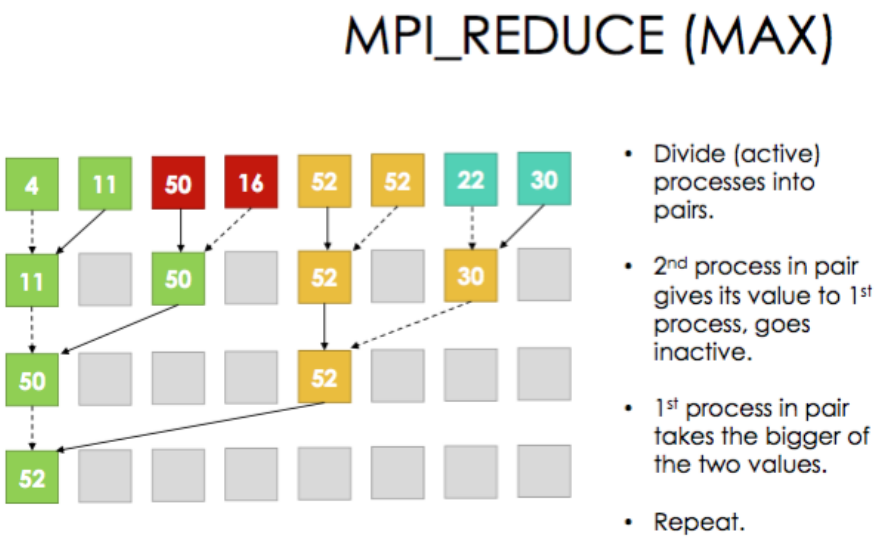
MPI_GATHER

- basically does exact opposite of MPI_Scatter



MPI_REDUCE

- Take an array of values and reduce it to a single value *e.g*
- Sum
- Min
- Product
- MPI_reduce can use a similar binary tree structure for communication



MPI_ALLGATHER

- Collects data from arrays distributed across all the processes into one big array that all processes have a copy of
- could be implemented using a butterfly arrangement