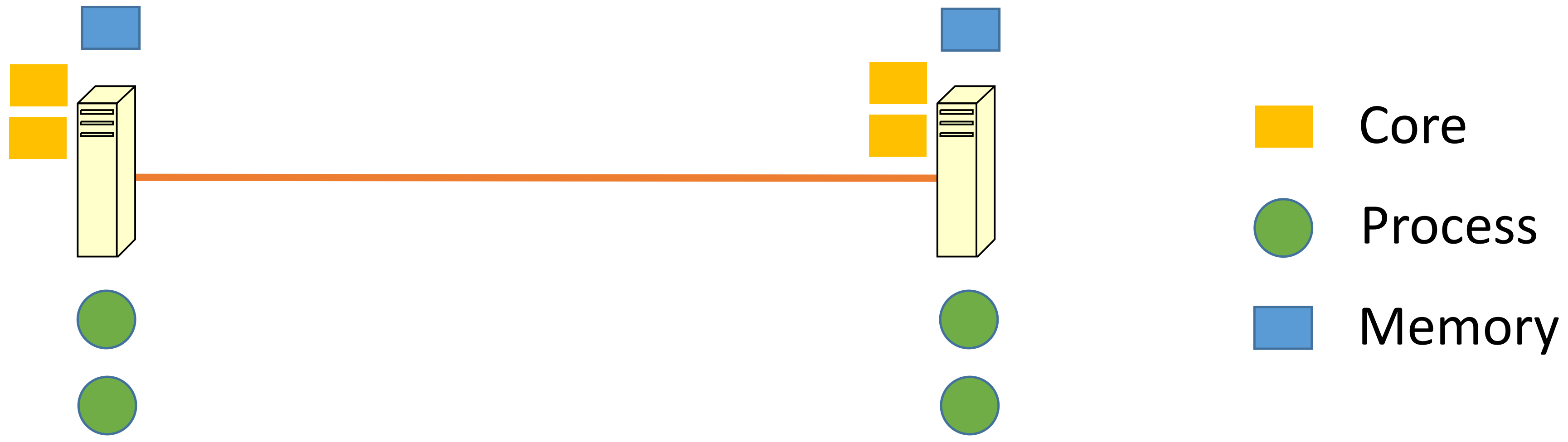
Communication

Feb 16, 2021

MPI Communication

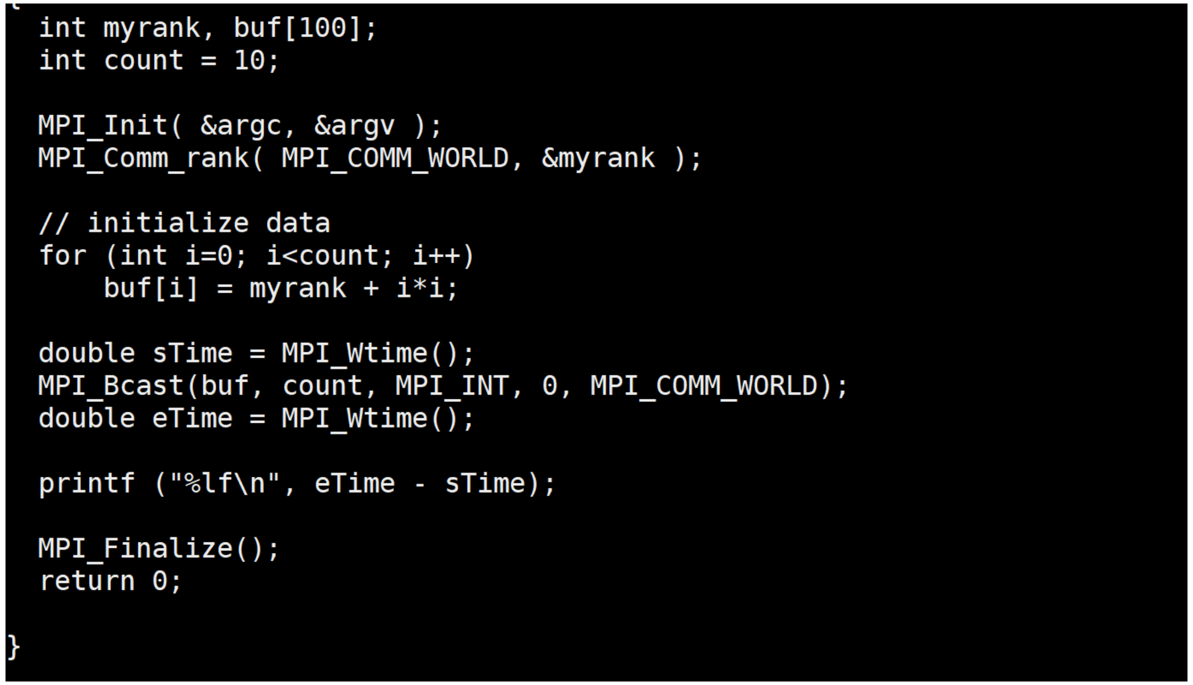
We'd like communication time to be negligible!

However, communication is often the bottleneck in parallel programs.

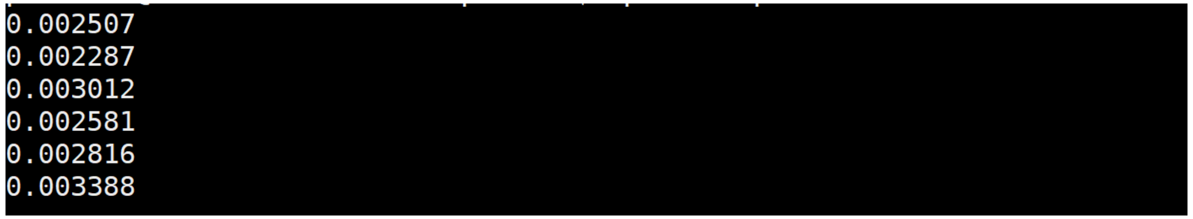


4 bytes
400 bytes
40000 bytes

Timing Parallel Codes



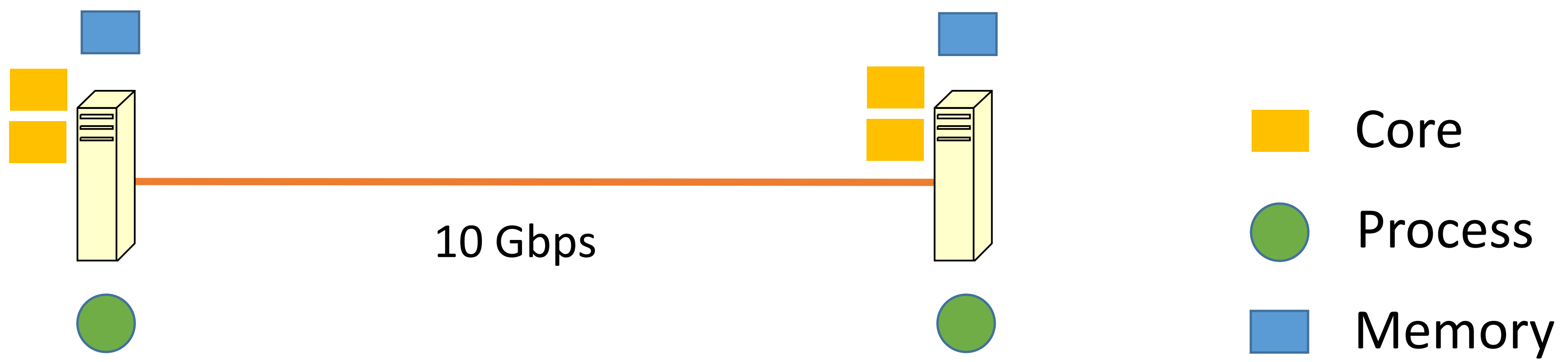
```
  int myrank, buf[100];
  int count = 10;

  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

  // initialize data
  for (int i=0; i<count; i++)
      buf[i] = myrank + i*i;

  double sTime = MPI_Wtime();
  MPI_Bcast(buf, count, MPI_INT, 0, MPI_COMM_WORLD);
  double eTime = MPI_Wtime();

  printf ("%lf\n", eTime - sTime);

  MPI_Finalize();
  return 0;

}
```



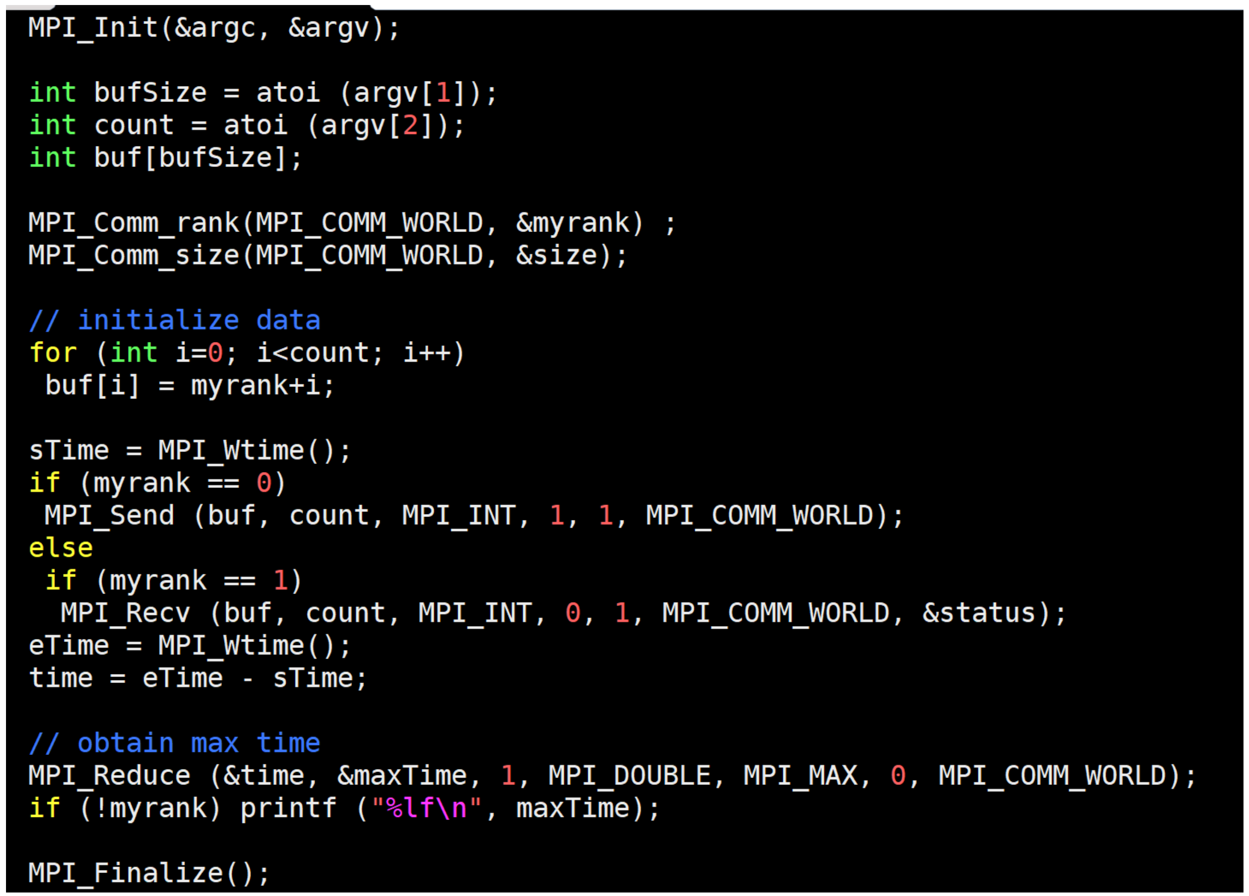
```
0.002507
0.002287
0.003012
0.002581
0.002816
0.003388
```

Bandwidth/Throughput

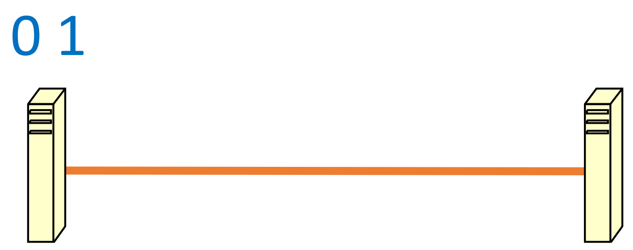
- Amount of data that can be transmitted in a certain amount of time
- Example: 200 bits in 2 ms → Achieved bandwidth = 10^5 bps

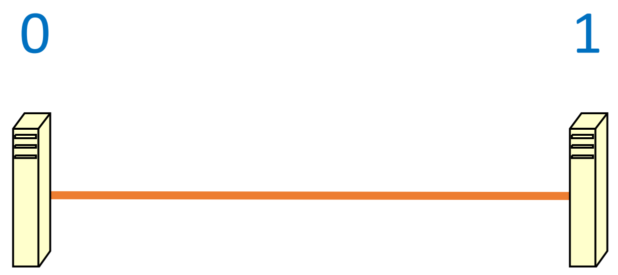


P2P (Send/Recv)



```
MPI_Init(&argc, &argv);

int bufSize = atoi (argv[1]);
int count = atoi (argv[2]);
int buf[bufSize];

MPI_Comm_rank(MPI_COMM_WORLD, &myrank) ;
MPI_Comm_size(MPI_COMM_WORLD, &size);

// initialize data
for (int i=0; i<count; i++)
 buf[i] = myrank+i;

sTime = MPI_Wtime();
if (myrank == 0)
 MPI_Send (buf, count, MPI_INT, 1, 1, MPI_COMM_WORLD);
else
 if (myrank == 1)
  MPI_Recv (buf, count, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
eTime = MPI_Wtime();
time = eTime - sTime;

// obtain max time
MPI_Reduce (&time, &maxTime, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
if (!myrank) printf ("%lf\n", maxTime);

MPI_Finalize();
```

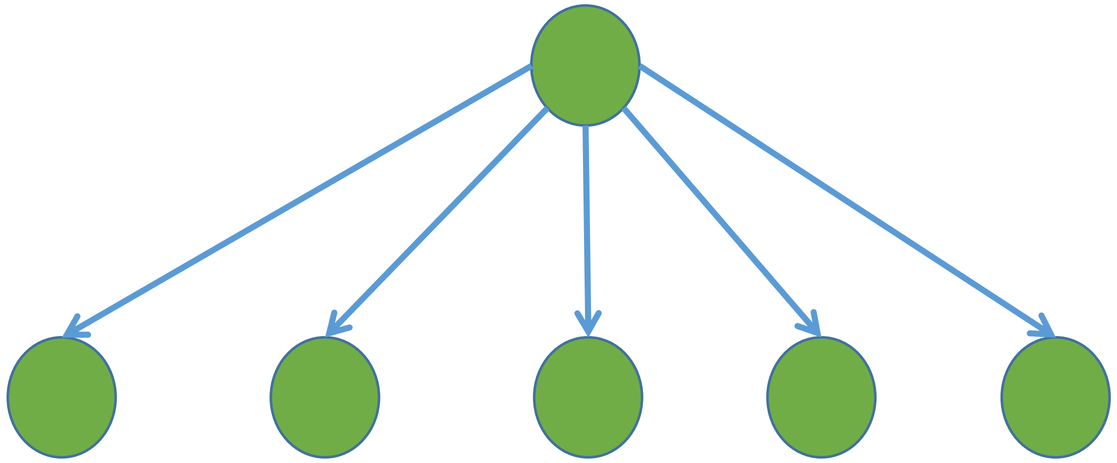




Homework

Measure the performance of P2P between different pairs of nodes (1 rank/node) for varying data sizes (1 KB – 128 MB)

1-to-many Sends



Timing Analysis for 1-to-many Sends

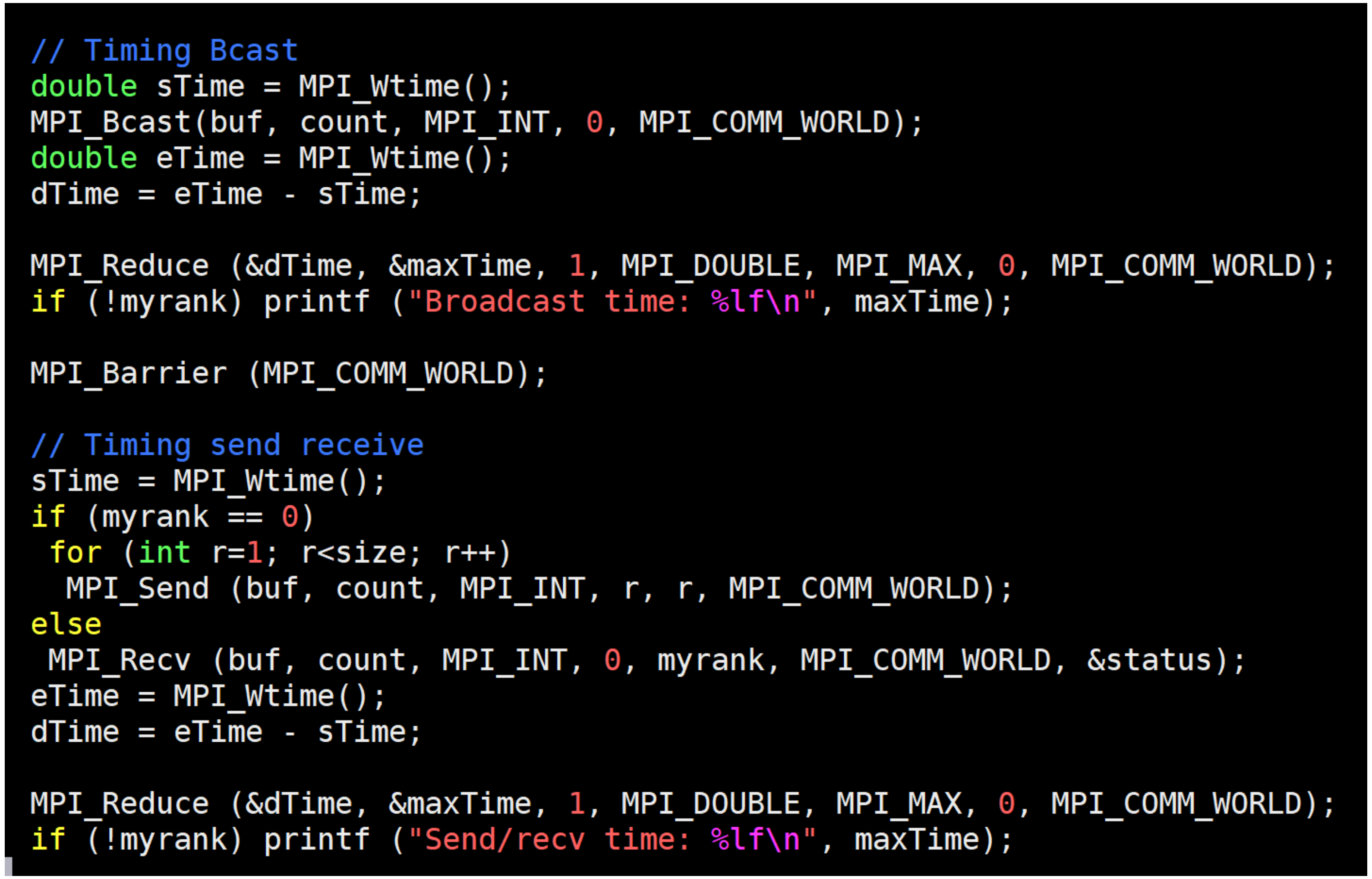
I Communication volume = 10000 integers to 29 ranks in 0.02 s

➔ Bandwidth achieved = 10000 * 4 * 29 bytes / 0.02 s = 55.3 Mbps

II Communication volume = 1000000 integers to 29 ranks in 0.8 s

➔ Bandwidth achieved = 1000000 * 4 * 29 bytes / 0.8 s = 138.3 Mbps

Bcast vs. Multiple P2P



```
// Timing Bcast
double sTime = MPI_Wtime();
MPI_Bcast(buf, count, MPI_INT, 0, MPI_COMM_WORLD);
double eTime = MPI_Wtime();
dTime = eTime - sTime;

MPI_Reduce (&dTime, &maxTime, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
if (!myrank) printf ("Broadcast time: %lf\n", maxTime);

MPI_Barrier (MPI_COMM_WORLD);

// Timing send receive
sTime = MPI_Wtime();
if (myrank == 0)
 for (int r=1; r<size; r++)
  MPI_Send (buf, count, MPI_INT, r, r, MPI_COMM_WORLD);
else
 MPI_Recv (buf, count, MPI_INT, 0, myrank, MPI_COMM_WORLD, &status);
eTime = MPI_Wtime();
dTime = eTime - sTime;

MPI_Reduce (&dTime, &maxTime, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
if (!myrank) printf ("Send/recv time: %lf\n", maxTime);
```

Timing Analysis – Bcast

- 10000 integers to 10 processes
  - Broadcast time: 0.005232
  - Send/recv time: 0.003686

10000 * 4 * 9 / 0.005 Bps = 68.7 MBps

10000 * 4 * 9 / 0.004 Bps = 85.8 MBps

- 10000 integers to 20 processes
  - Broadcast time: 0.007424
  - Send/recv time: 0.011729

10000 * 4 * 19 / 0.007 Bps = 103.5 MBps

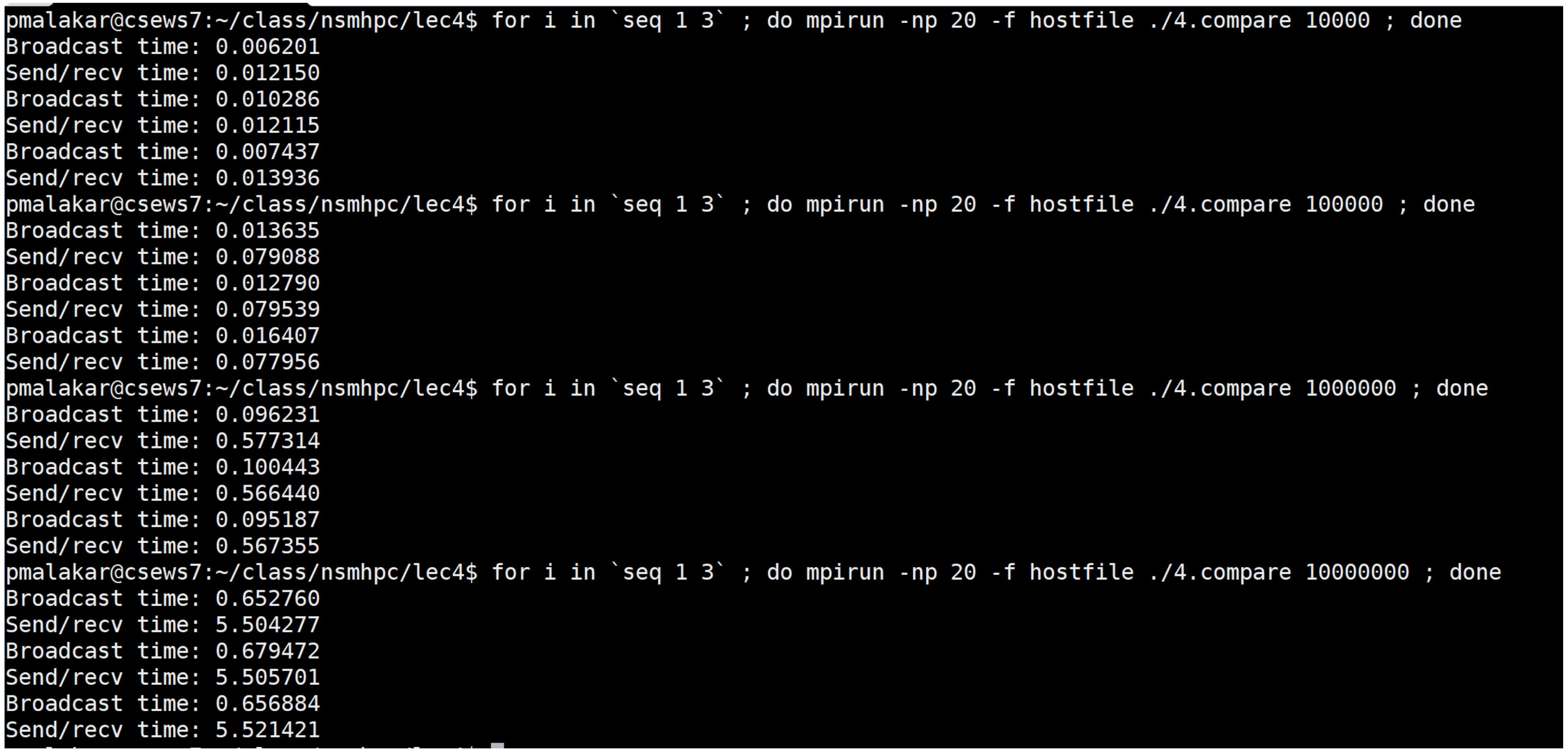
10000 * 4 * 19 / 0.011 Bps = 65.9 MBps

- 10000 integers to 40 processes
  - Broadcast time: 0.011361
  - Send/recv time: 0.025142

10000 * 4 * 39 / 0.011 Bps = 135.2 MBps

10000 * 4 * 39 / 0.025 Bps = 59.5 MBps

Timing – Effect of data size



```
pmalakar@csews7:~/class/nsmhpc/lec4$ for i in `seq 1 3` ; do mpirun -np 20 -f hostfile ./4.compare 10000 ; done
Broadcast time: 0.006201
Send/recv time: 0.012150
Broadcast time: 0.010286
Send/recv time: 0.012115
Broadcast time: 0.007437
Send/recv time: 0.013936
pmalakar@csews7:~/class/nsmhpc/lec4$ for i in `seq 1 3` ; do mpirun -np 20 -f hostfile ./4.compare 100000 ; done
Broadcast time: 0.013635
Send/recv time: 0.079088
Broadcast time: 0.012790
Send/recv time: 0.079539
Broadcast time: 0.016407
Send/recv time: 0.077956
pmalakar@csews7:~/class/nsmhpc/lec4$ for i in `seq 1 3` ; do mpirun -np 20 -f hostfile ./4.compare 1000000 ; done
Broadcast time: 0.096231
Send/recv time: 0.577314
Broadcast time: 0.100443
Send/recv time: 0.566440
Broadcast time: 0.095187
Send/recv time: 0.567355
pmalakar@csews7:~/class/nsmhpc/lec4$ for i in `seq 1 3` ; do mpirun -np 20 -f hostfile ./4.compare 10000000 ; done
Broadcast time: 0.652760
Send/recv time: 5.504277
Broadcast time: 0.679472
Send/recv time: 5.505701
Broadcast time: 0.656884
Send/recv time: 5.521421
```

Bandwidth Comparison

10000 integers (20 processes)

Broadcast: 103.5 MBps

Send/recv: 72.5 MBps

100000 integers (20 processes)

Broadcast: 517.7 MBps

Send/recv: 103.5 MBps

1000000 integers (20 processes)

Broadcast: 805.3 MBps

Send/recv: 120 MBps

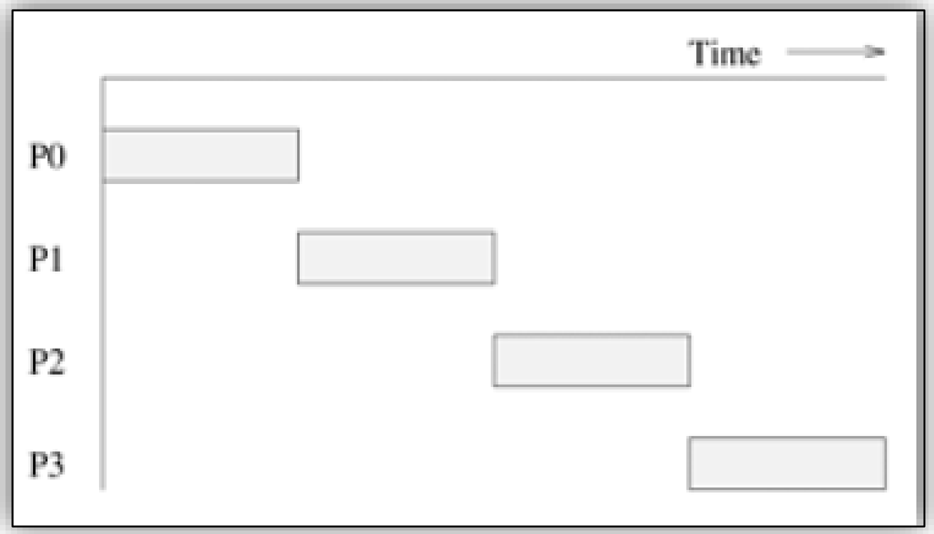
Communication Cost

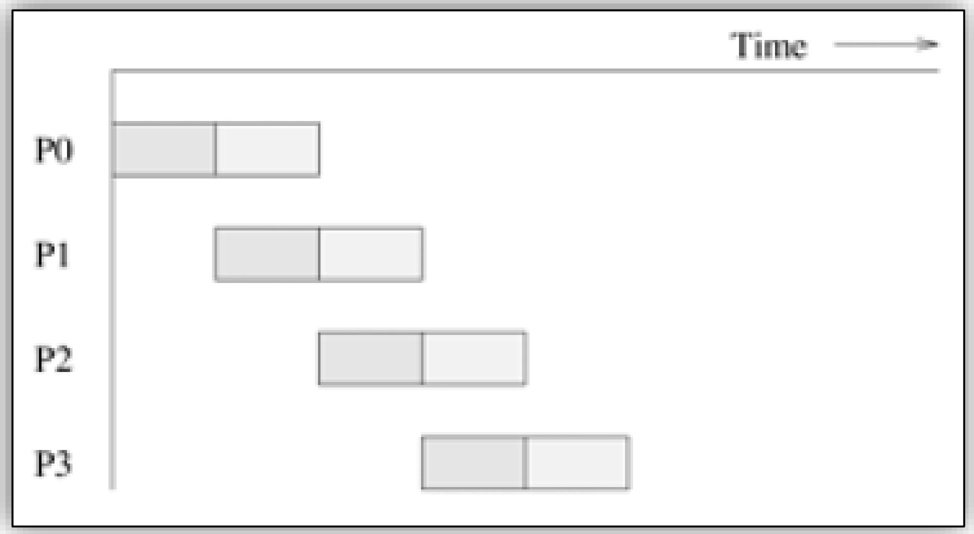
- Startup time (t_s)
- Per-hop time (t_h)
- Bandwidth (t_w)

Transfer time of n bytes on l links = $t_s + l.t_h + n/t_w$

Reference: Chapter 2 of "Introduction to Parallel Computing", Ananth Grama et al.

Routing





Store-and-forward

Cut-through

Ananth Grama et al.

Latency and Bandwidth

A Realistic Cost Model

Effective communication time

- L + n/B + contention

Reduce communication time by reducing

- Overhead (per-hop time)
- Contention

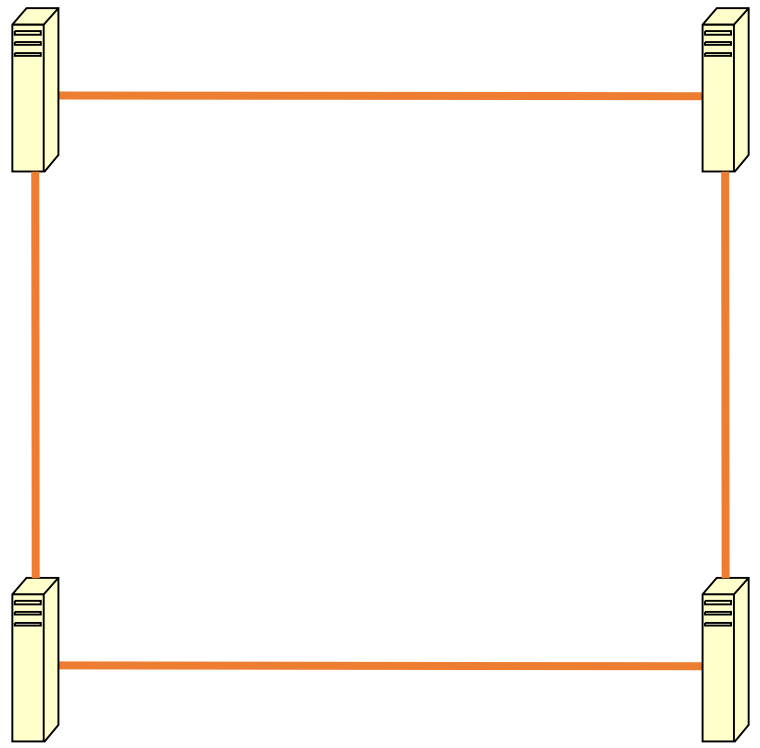
Most difficult to analyse

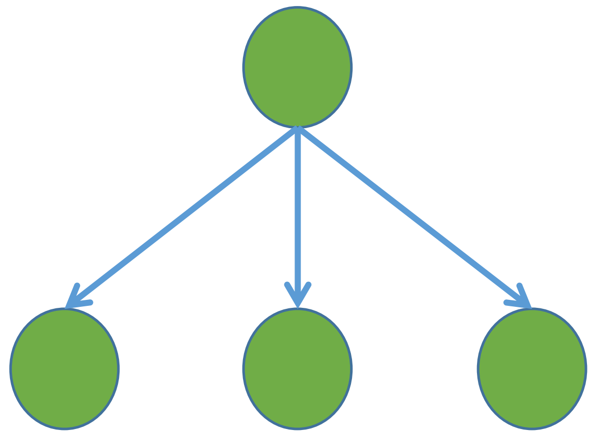
Hockney Model / Postal Model

- Communication time = $t_s + l.t_h + n/t_w$
  - Communicate in bulk
  - Communicate over fewer hops
  - Communicate less volume

- Simplified model (without congestion): $T_{comm} = L + n/B$
- Simplified model (with congestion): $T_{comm\_congest} = L' + n/B'$ [B' = B/?]

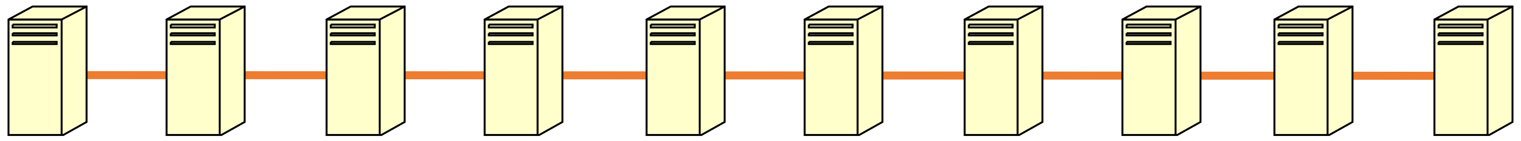
Effect of Network Topology





Interconnects

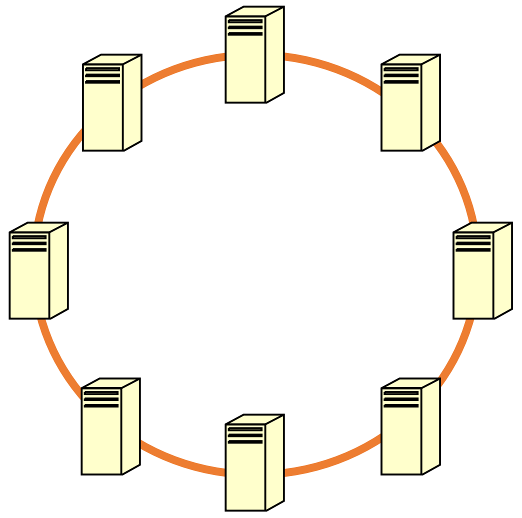
Linear Array



Attributes / Parameters

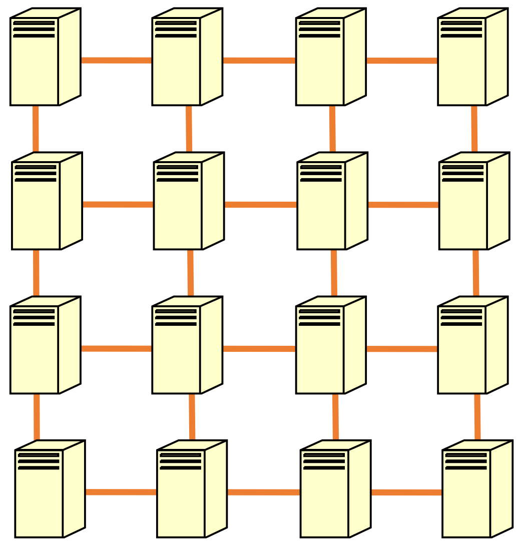
- Topology
- Diameter p-1
- Bisection width 1
- Cost p-1

Ring Interconnect



- Diameter p/2
- Bisection width 2
- Cost p

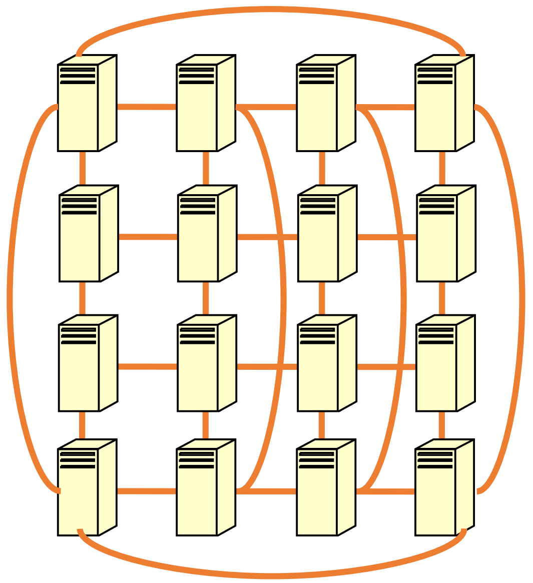
Mesh Interconnect



- Diameter $2(\sqrt{p} - 1)$
- Bisection width $\sqrt{p}$
- Cost $2(p - \sqrt{p})$

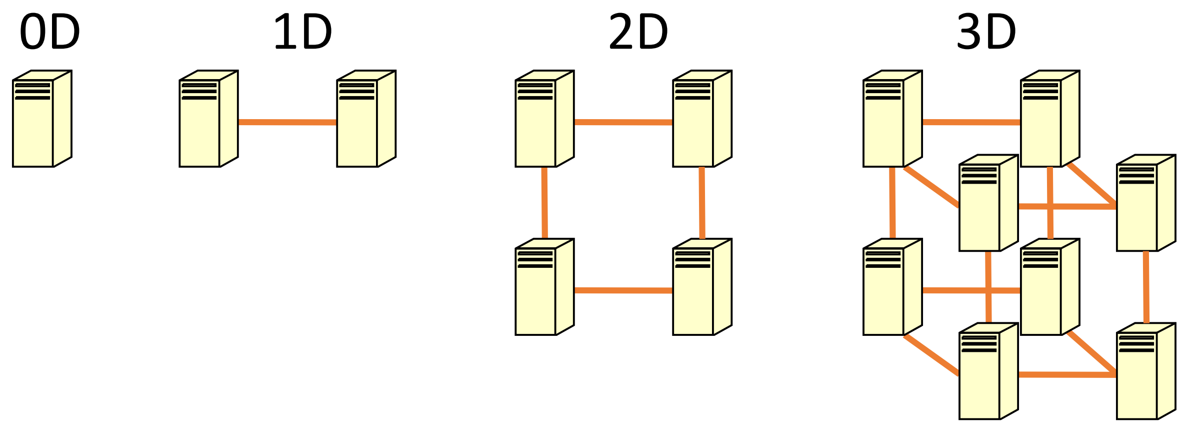
Simple improvement over this interconnect?

Torus Interconnect



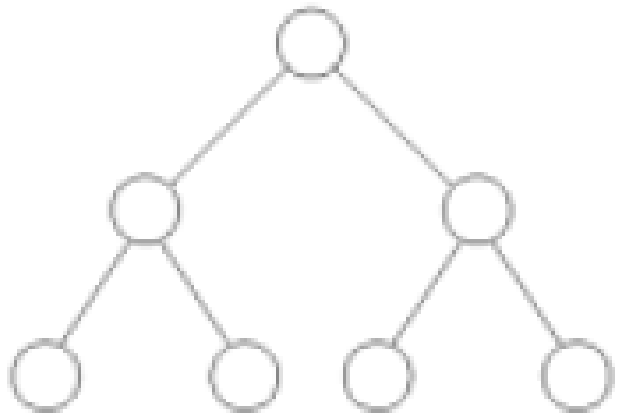
- Diameter 2(√p/2)
- Bisection width 2√p
- Cost 2p

Hypercube Interconnect



- Diameter log p
- Bisection width p/2
- Cost (p log p)/2

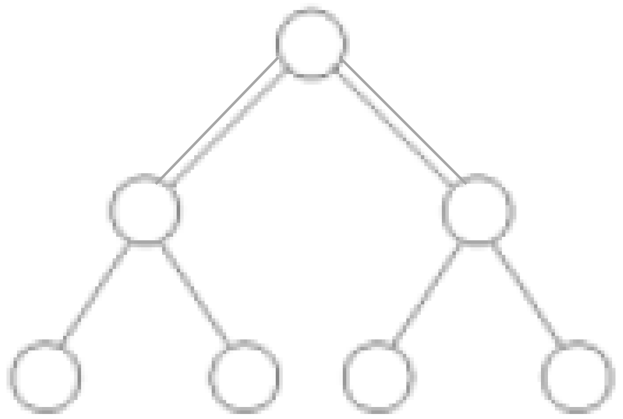
Tree Networks



Problem?

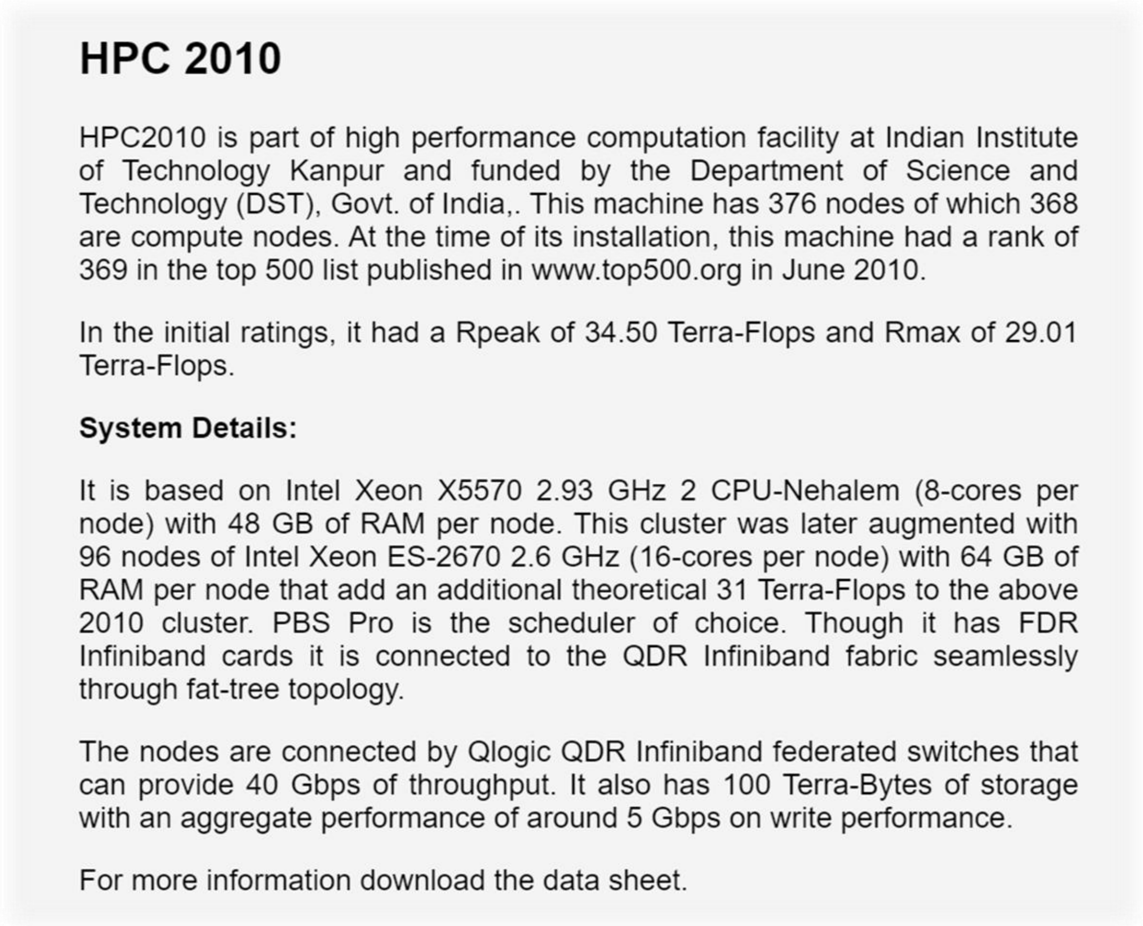
Communication bottlenecks at higher levels

Fat-tree Networks



C. E. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. In *Proceedings of the 1985 International Conference on Parallel Processing*, 393–402, 1985.

HPC2010 Supercomputer@IITK



HPC 2010

HPC2010 is part of high performance computation facility at Indian Institute of Technology Kanpur and funded by the Department of Science and Technology (DST), Govt. of India,. This machine has 376 nodes of which 368 are compute nodes. At the time of its installation, this machine had a rank of 369 in the top 500 list published in www.top500.org in June 2010.

In the initial ratings, it had a Rpeak of 34.50 Terra-Flops and Rmax of 29.01 Terra-Flops.

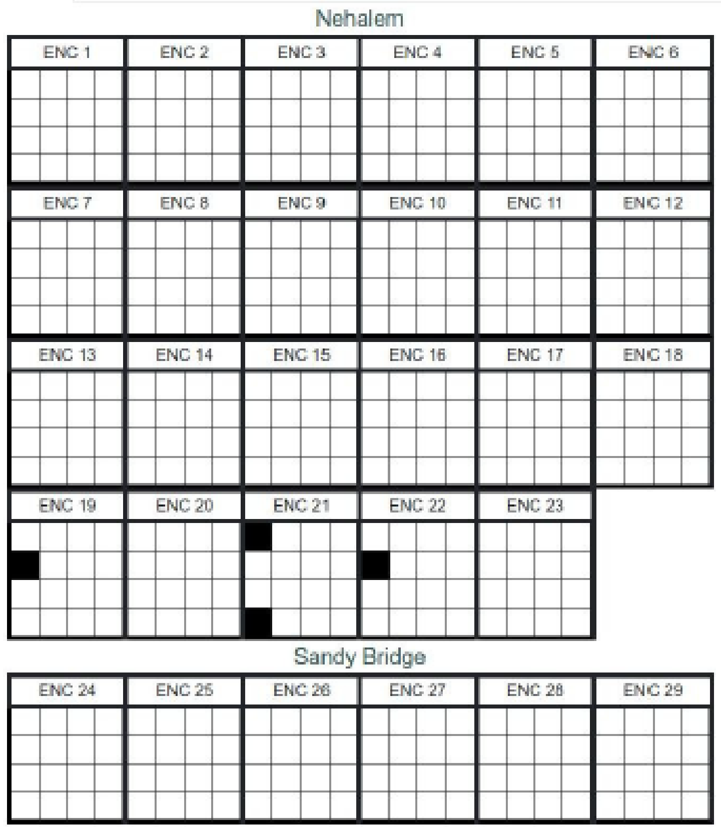
System Details:

It is based on Intel Xeon X5570 2.93 GHz 2 CPU-Nehalem (8-cores per node) with 48 GB of RAM per node. This cluster was later augmented with 96 nodes of Intel Xeon ES-2670 2.6 GHz (16-cores per node) with 64 GB of RAM per node that add an additional theoretical 31 Terra-Flops to the above 2010 cluster. PBS Pro is the scheduler of choice. Though it has FDR Infiniband cards it is connected to the QDR Infiniband fabric seamlessly through fat-tree topology.

The nodes are connected by Qlogic QDR Infiniband federated switches that can provide 40 Gbps of throughput. It also has 100 Terra-Bytes of storage with an aggregate performance of around 5 Gbps on write performance.

For more information download the data sheet.

https://www.hpc.iitk.ac.in



Source: A Pal et al.