HYPERQUICK SORT

CSE 633: Parallel Algorithms

Guide: Dr. Russ Miller

Presenter: Gaurav Nathani

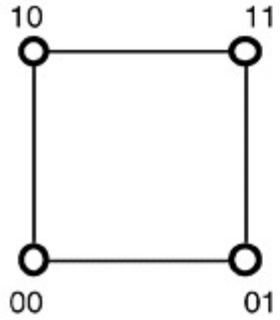
University at Buffalo The State University of New York

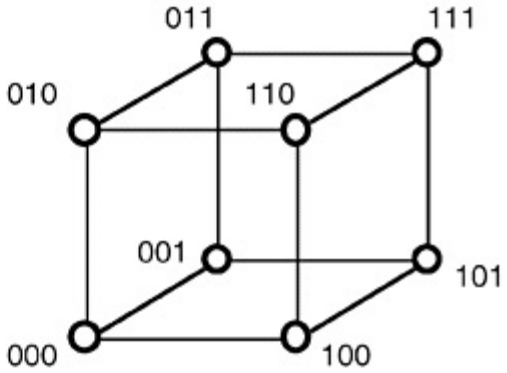
Hypercube

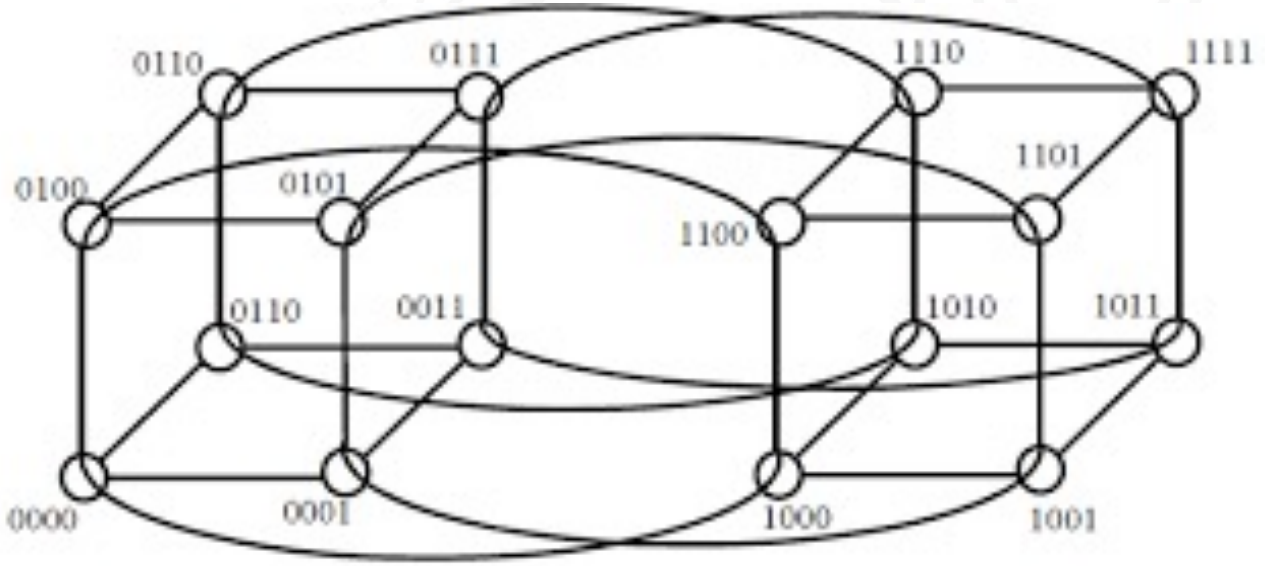
- Multi-dimensional mesh network
- Has 2^d processors or processing elements (d – degree)
- Degree = 0 – just a point (single processor)
- Degree = 1 – line (2 processors connected)
- Degree = 2 – square (Each processor connected to 2 other processors)
- Degree = 3 – cube (Each processor connected to 3 other processors)

Hypercube - Network

- Network connections only between processors that differ at exactly 1 bit in the binary representation of their processor ids

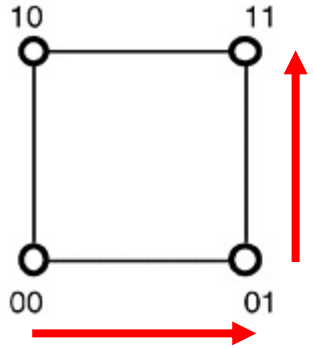


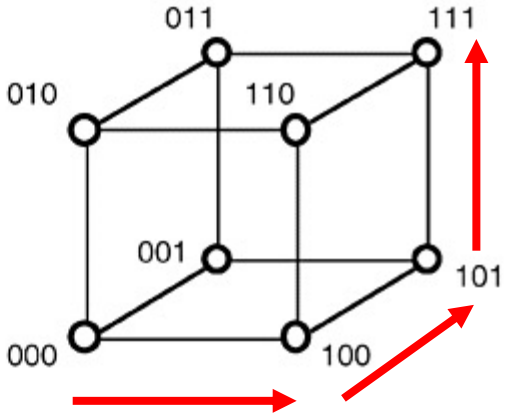


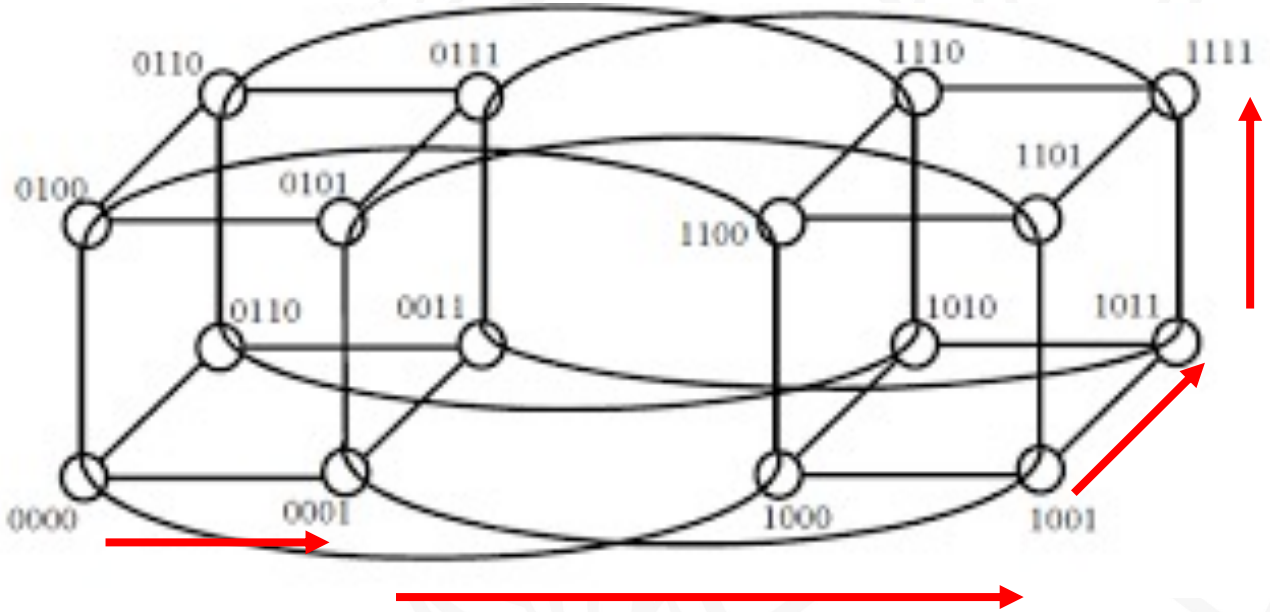


Hypercube – Communication Diameter

- Minimum number of hops to connect most distant processors
- For hypercube – $\log_2(P) = d$
- Observe multiple shortest paths present

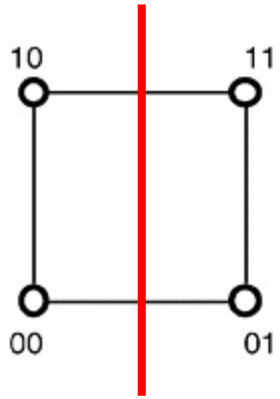


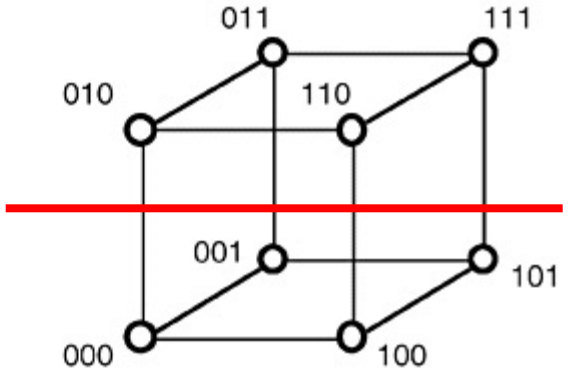


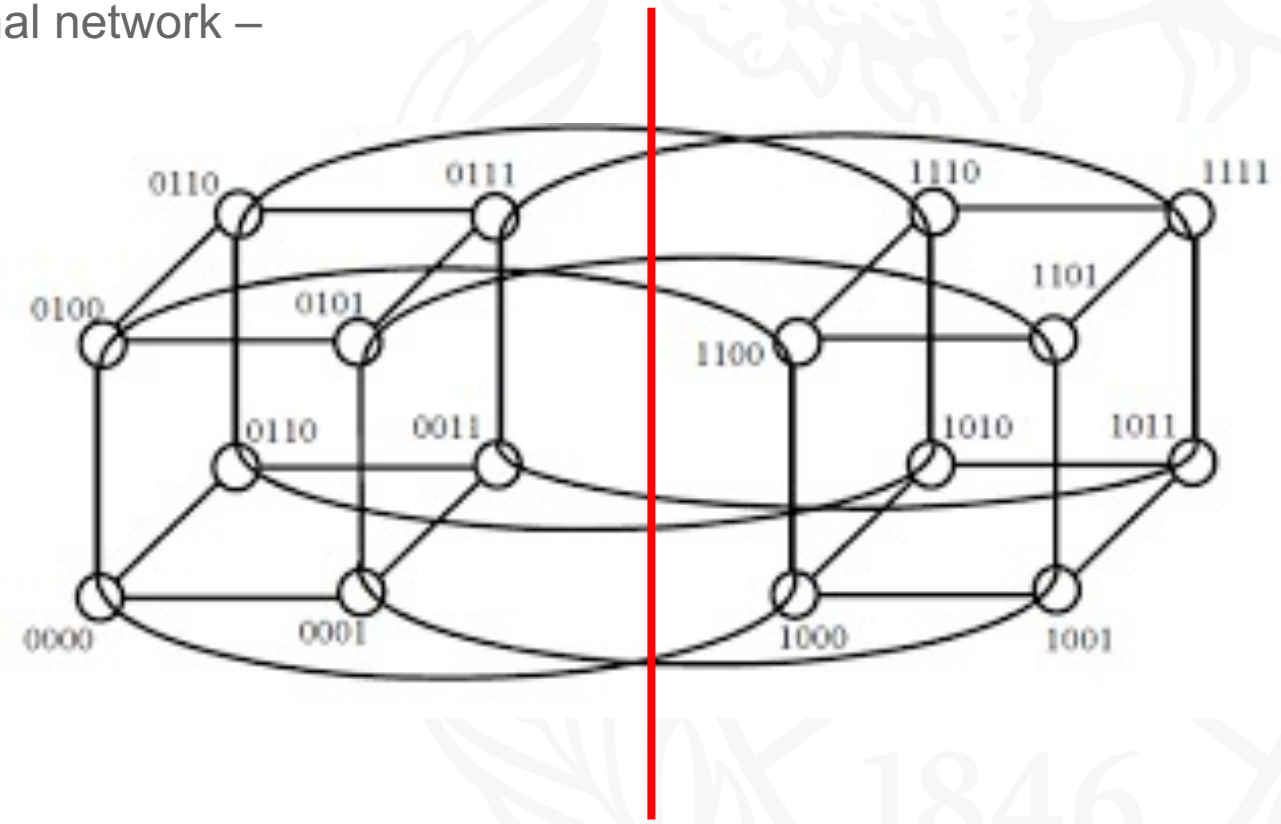


Hypercube – Bisection Width

- Number of connections we need to cut in order to create 2 networks of equal size - half the size of the original network – $2^{(d-1)}$







Quick Sort

- Divide and conquer algorithm
- Choose pivot/splitter
- Divide list: less than or equal to pivot & greater than pivot/splitter
- Recurse on the 2 sub lists
- Time Complexity O(n log(n))

Hypercube Quick Sort

1. Divide data equally on each processor
2. Sort data individually on each processor
3. Pick median from the lowest numbered processor/s in current sub cube and broadcast to all processors in the sub cube
4. Based on this median, create a less than equal to median list & a greater than median list
5. Along dimension 0, each pair of neighbors exchange their greater & lesser list
6. Merge the 2 lists at each processor to create sorted list
7. Create 2 sub-networks or 2 sub-cubes along the lowest dimension and repeat steps 3 to 7 until only one processor in the sub-cube.

Hyperquick Sort – Divide and Conquer

d – degree of hypercube
P – number of processors = 2^d
Number of Phases = $\log_2(P) = d$
Total Number of Broadcast Communications = $2^d - 1$

- Example: d=3, P=8
  Number of phases = 3
  Total number of broadcast steps = 7

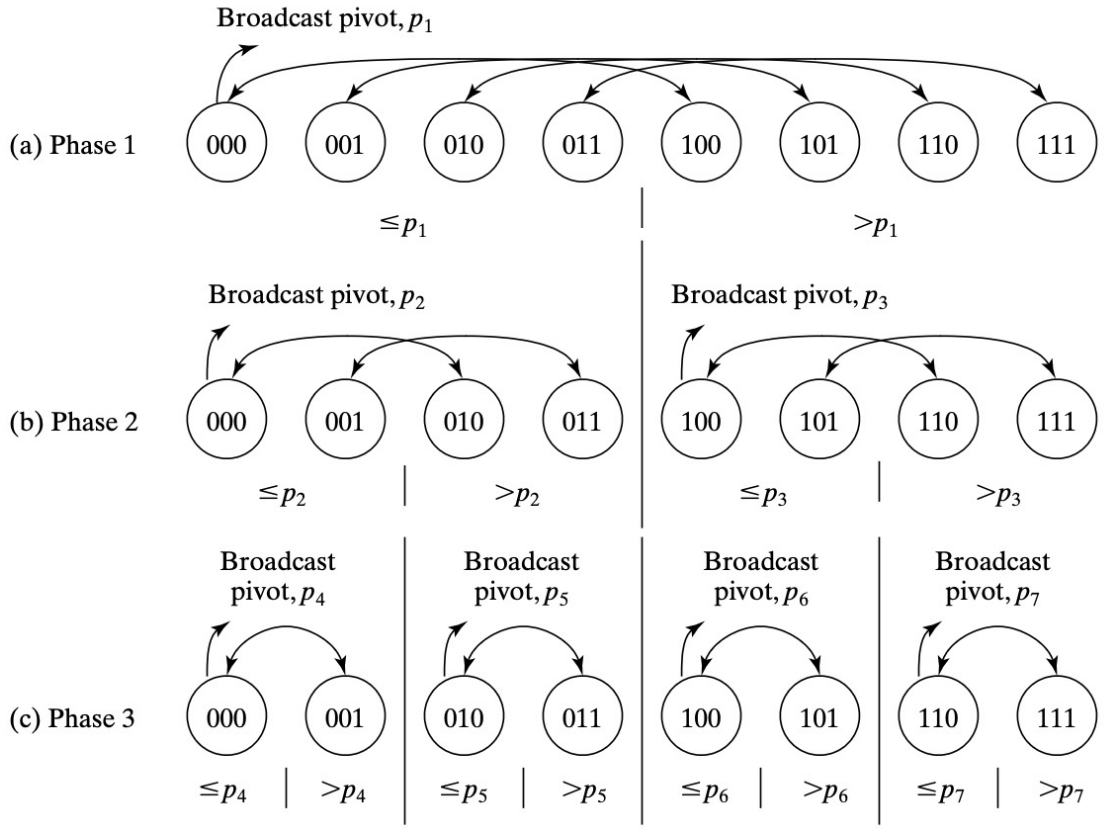


Figure 10.23 Hypercube quicksort algorithm when numbers are distributed among nodes.

Hyperquick Sort – Analysis

n – number of data points, d – degree of hypercube, p – number of processors = 2^d

- $(n/p)\log_2(n/p)$ – time for sequential sort on each processor [only done at the beginning]
- $(2^d-1)(t_b)$ – time for all broadcast communication where t_b is the time required for broadcast
- $d(t_e)$ – time for all data exchange communication where t_e is the time required for data exchange
- $d(n/p) + d(n/p)$ – time for creating sublists and merging sublists

Expected Running Time for Hyperquick Sort :

$$\Theta((n/p)\log_2(n/p) + (2^d-1)(t_b) + d(t_e) + d(n/p) + d(n/p)) \quad \text{[Non-simplified]}$$

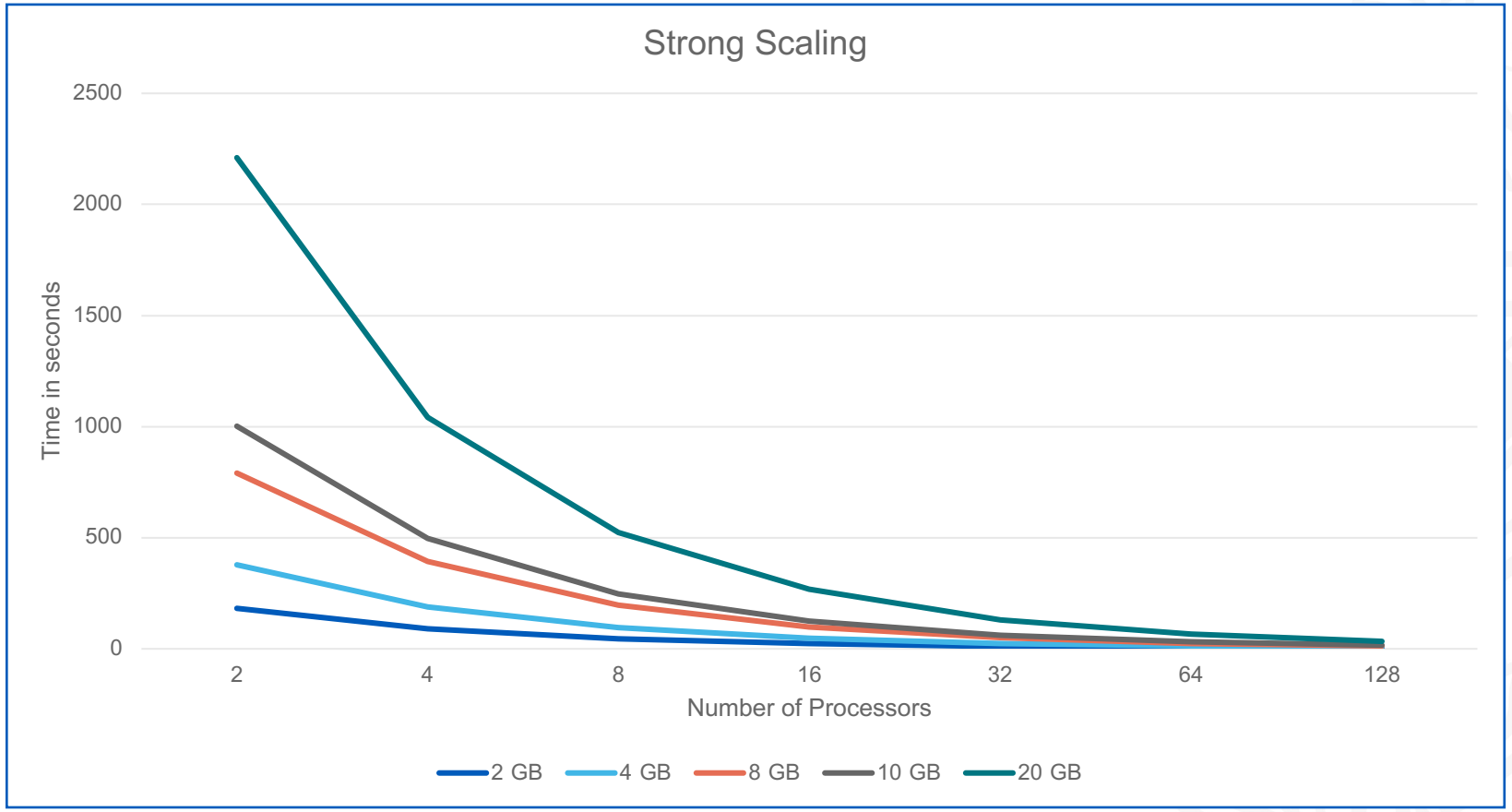
* t_b & t_e depend on message size, p, and network latency & bandwidth

Strong Scaling

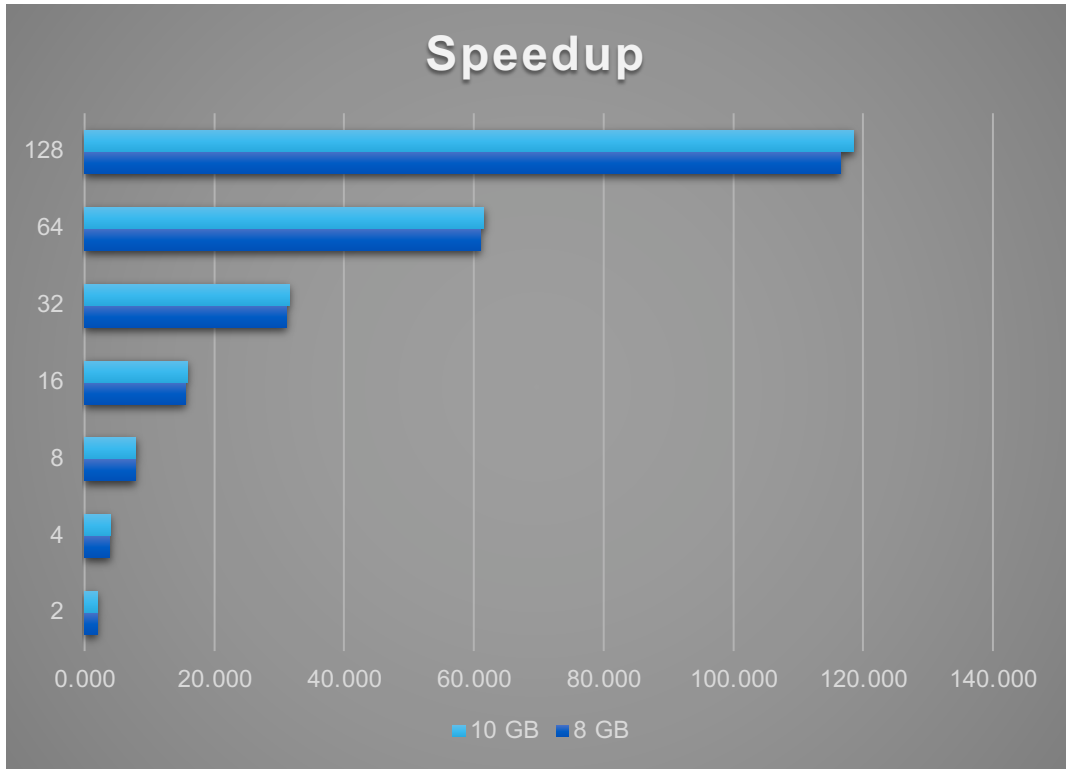
p/n	2 GB	4 GB	8 GB	10 GB	20 GB
1	358.049	744.424	1532.8	1956.08	4140
2	182.204	378.132	790.97	1002.09	2210.49
4	90.7163	188.477	392.078	496.728	1042.77
8	45.7771	94.9527	197.01	247.982	522.592
16	23.2691	47.5236	98.8261	123.863	268.314
32	11.6798	24.073	49.287	62.0191	130.855
64	6.26326	13.5721	25.1438	31.7966	65.8802
128	3.42971	6.60675	13.1622	16.512	33.715

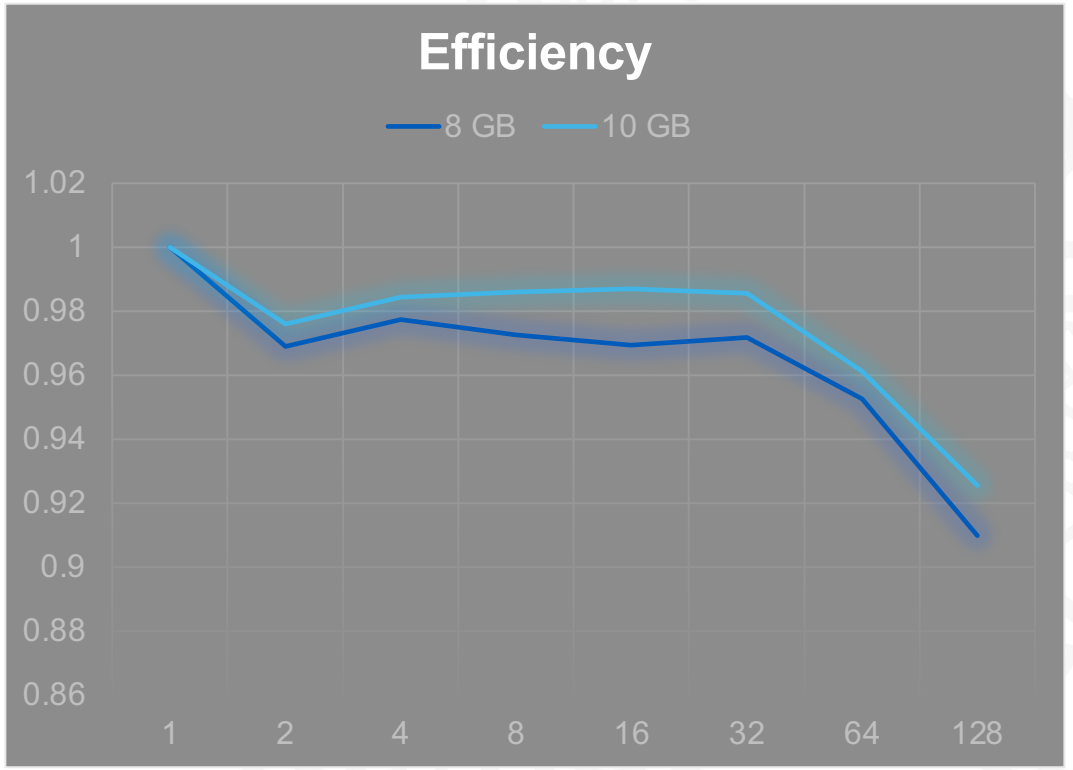
We observe considerable speedup for constant data by increasing the number of processors.

Strong Scaling



Speedup vs Efficiency: Strong Scaling



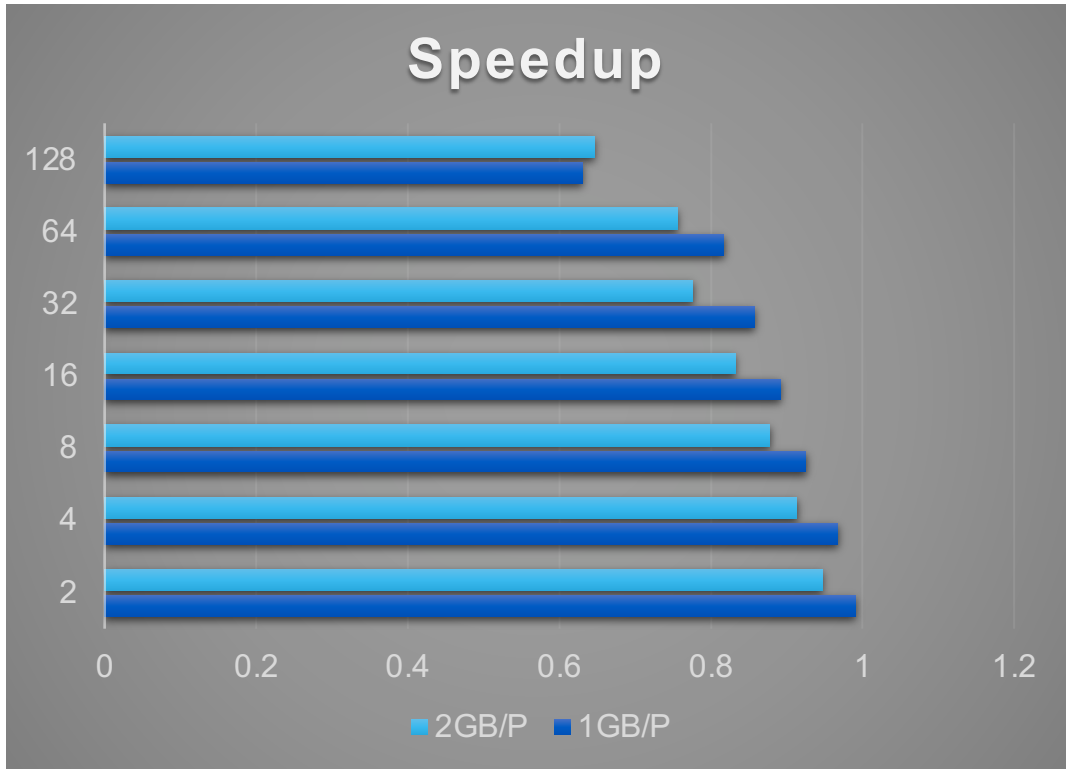


Weak Scaling

p/n	1 GB Per Node	2 GB Per Node
1	173.309	358.049
2	184.062	378.132
4	188.477	392.078
8	197.008	407.917
16	204.384	430.394
32	212.644	461.355
64	223.135	473.859
128	289.025	553.869

For same ratio of data per processor we see sort of similar times.

Speedup vs Efficiency: Weak Scaling



Pros

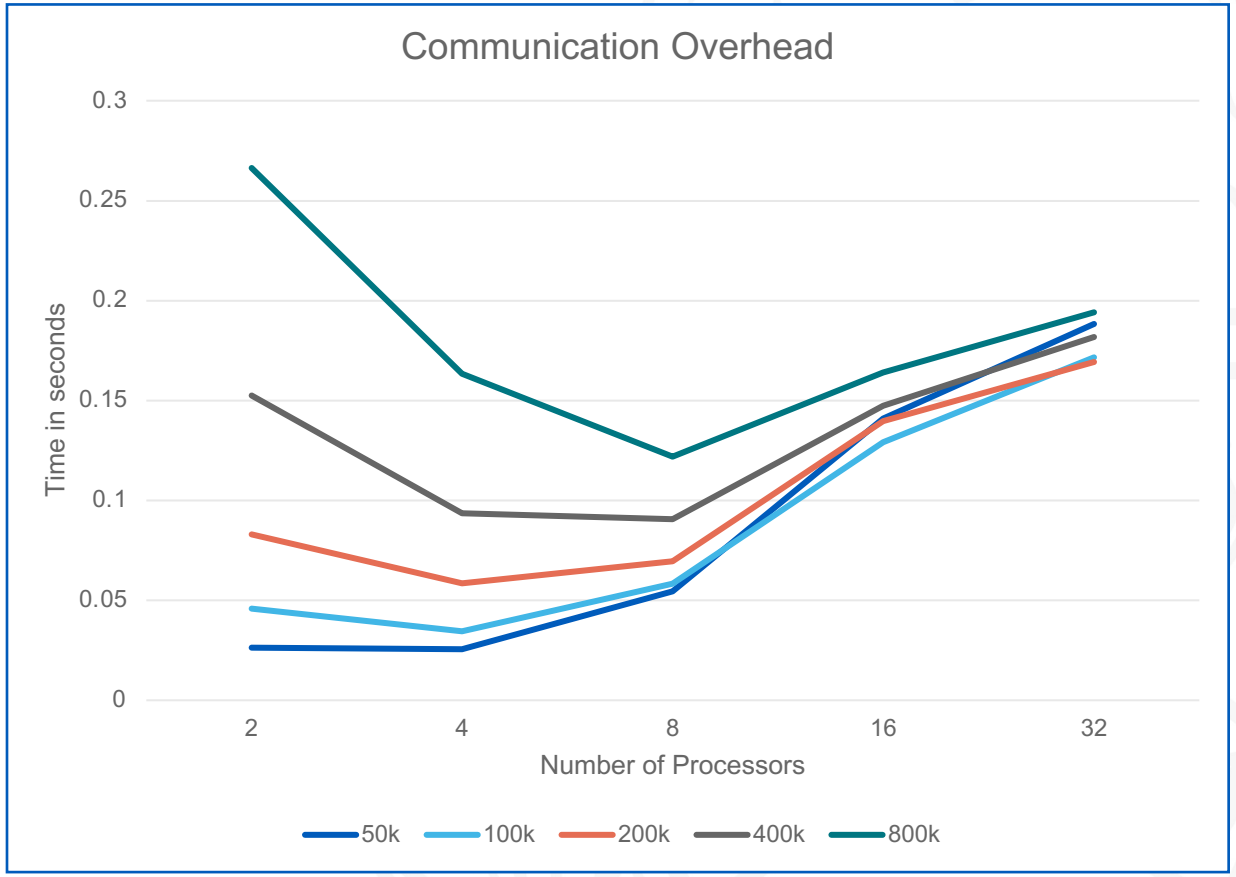
- Hypercube properties – high bisection width & low communication diameter
- With uniform distribution of data, we get good median/splitter and hence good speedup possible

Cons

- Number of nodes must be power of 2
- High communication overheads
- Load imbalance due to bad splitter affects performance
- If some prior knowledge of data – choose better sort options

High Communication Overheads

n/p	2	4	8	16	32
50k	0.02642	0.02555	0.054602	0.1408	0.1883
100k	0.0459	0.03452	0.05832	0.1291	0.1716
200k	0.0831	0.05852	0.069626	0.1397	0.1693
400k	0.15262	0.09348	0.090608	0.1473	0.1818
800k	0.26634	0.1635	0.121906	0.164	0.194



Load Imbalance

P	Neighbor	Phase 1	Broadcast
0	2	1 1 1 1 1 **1** 1 1 1 1 1 1	splitter 1
1	3	1 1 1 1 1 1 1 1 1 2 5 6	
2	0	2 3 4 5 6 6 6 20 24 24 26 28	
3	1	1 2 3 8 9 21 22 29 29 32 39 41 52	
P		Phase 2	Broadcast
0	1	1 1 1 1 1 **1** 1 1 1 1 1 1	splitter 1
1	0	1 1 1 1 1 1 1 1 1 1	
2	3	2 3 4 5 6 **6** 6 20 24 24 26 28	splitter 6
3	2	2 2 3 5 6 8 9 21 22 29 29 32 39 41 52	
P		Final	Broadcast
0		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
1			
2		2 2 2 3 3 4 5 5 6 6 6 6	
3		8 9 20 21 22 24 24 26 28 29 29 29 32 39 41 52	

Questions?

THANK YOU!

University at Buffalo The State University of New York