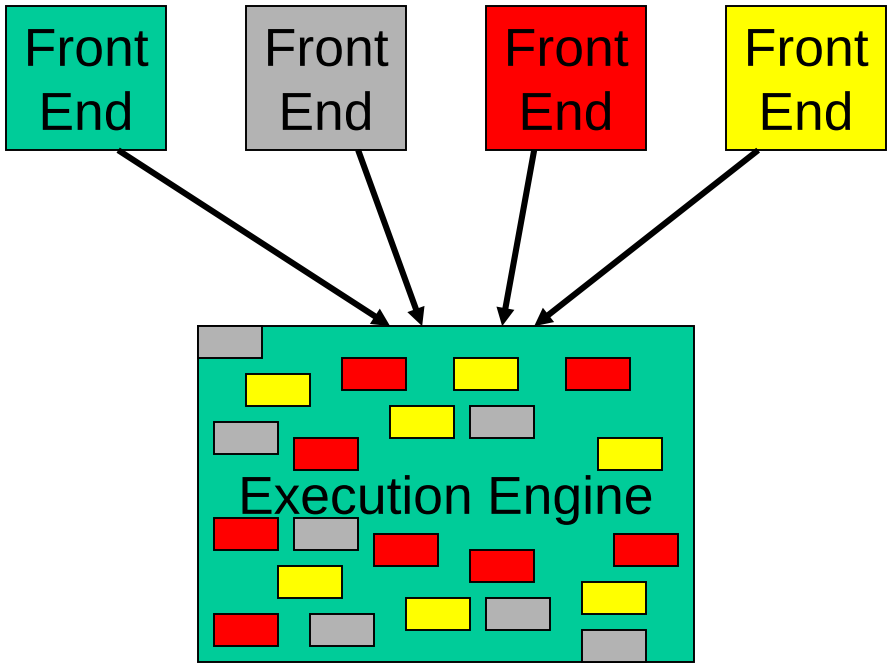
Lecture 18: Core Design, Parallel Algos

- Today: Innovations for ILP, TLP, power and parallel algos
- Sign up for class presentations

SMT Pipeline Structure



Private/ Shared Front-end

I-Cache

Bpred

Private Front-end

Rename

ROB

Shared Exec Engine

Regs

IQ

DCache

FUs

SMT maximizes utilization of shared execution engine

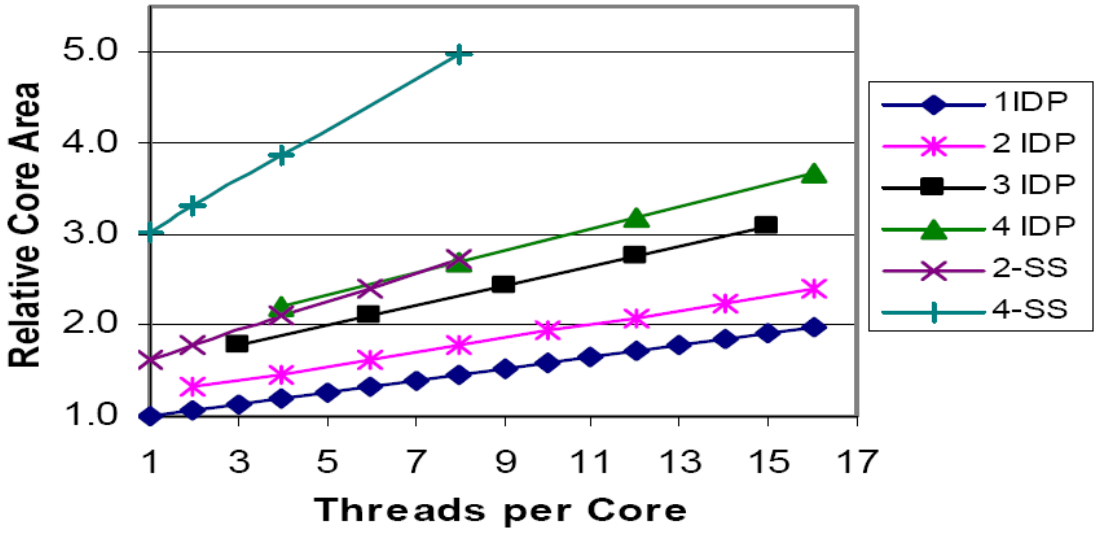
SMT Fetch Policy

- Fetch policy has a major impact on throughput: depends on cache/bpred miss rates, dependences, etc.

- Commonly used policy: ICOUNT: every thread has an equal share of resources
  - faster threads will fetch more often: improves thruput
  - slow threads with dependences will not hoard resources
  - low probability of fetching wrong-path instructions
  - higher fairness

Area Effect of Multi-Threading

- The curve is linear for a while
- Multi-threading adds a 5-8% area overhead per thread (primary caches are included in the baseline)



From Davis et al., PACT 2005

Single Core IPC

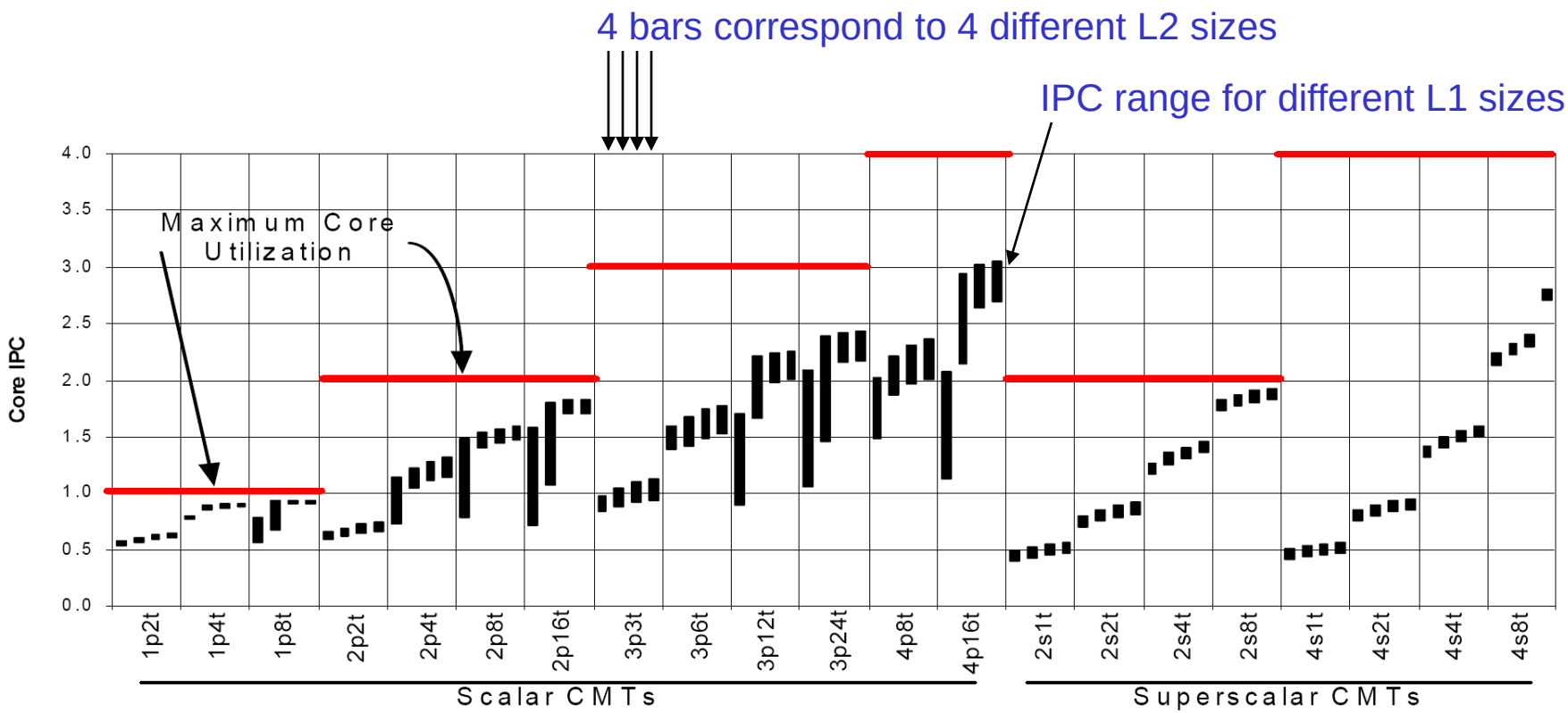


Figure 4: SPEC JBB average core IPC range (maximum to minimum) for medium-scale CMTs. The secondary cache size range is 1.5MB, 2.5MB, 3.5MB, and 4.5MB from left to right for each core.

Maximal Aggregate IPCs

Table 3: Maximum AIPC for medium-scale CMTs for SPEC JBB, TPC-C, TPC-W, and XML Test.

Core Config	SPEC JBB 2000				TPC-C				TPC-W				XML Test			
	L1	L2	Cores	AIPC	L1	L2	Cores	AIPC	L1	L2	Cores	AIPC	L1	L2	Cores	AIPC
1p2t	16/32	1.5/12	20	9.8	16/32	2.5/10	16	5.8	16/32	1.5/12	20	8.6	16/32	1.5/12	20	11.8
1p4t	16/32	1.5/12	17	13.2	16/32	2.5/10	14	8.2	16/32	1.5/12	17	10.6	16/32	1.5/12	17	14.8
1p8t	16/32	2.5/10	12	11.7	32/32	1.5/12	14	8.9	32/32	1.5/12	14	13.0	16/32	1.5/12	14	13.8
2p2t	16/32	1.5/12	16	8.6	16/32	1.5/12	16	5.1	16/32	1.5/12	16	7.5	16/32	1.5/12	16	10.5
2p4t	32/32	1.5/12	14	12.9	32/32	2.5/10	12	7.8	32/32	1.5/12	14	10.6	16/32	1.5/12	14	15.2
2p8t	16/32	1.5/12	12	16.5	32/32	2.5/10	9	9.5	32/32	1.5/12	12	13.6	32/32	1.5/12	12	18.9
2p16t	32/64	2.5/10	7	13.3	64/64	2.5/10	7	**11.8**	64/64	1.5/12	9	**15.2**	32/64	1.5/12	9	16.9
3p3t	32/32	1.5/12	13	10.3	32/32	2.5/10	10	5.9	32/32	1.5/12	13	8.5	16/32	1.5/12	13	12.7
3p6t	32/32	1.5/12	11	14.4	32/32	2.5/10	9	8.5	32/32	1.5/12	11	11.3	32/32	1.5/12	11	16.5
3p12t	32/64	1.5/12	9	**17.3**	32/64	2.5/10	7	10.7	64/64	1.5/12	9	14.6	32/64	1.5/12	9	**20.1**
3p24t	32/64	2.5/10	5	13.6	32/64	2.5/10	5	10.9	32/64	1.5/12	6	14.0	32/64	1.5/12	6	15.5
4p8t	32/32	1.5/12	9	14.9	32/32	2.5/10	7	8.5	64/64	1.5/12	9	11.5	16/32	1.5/12	9	16.6
4p16t	32/64	1.5/12	7	16.8	32/64	2.5/10	5	9.8	64/64	1.5/12	7	14.4	32/64	1.5/12	7	18.5
2s1t	64/64	1.5/12	11	4.4	64/64	1.5/12	11	2.8	64/64	1.5/12	11	3.7	64/64	1.5/12	11	5.5
2s2t	64/64	1.5/12	10	7.0	64/64	1.5/12	10	4.3	64/64	1.5/12	10	5.8	64/64	1.5/12	10	8.6
2s4t	64/64	1.5/12	9	10.5	64/64	1.5/12	9	6.4	64/64	1.5/12	9	8.7	64/64	1.5/12	9	12.4
2s8t	64/64	1.5/12	7	12.1	64/64	1.5/12	7	8.1	64/64	1.5/12	7	10.6	64/64	1.5/12	7	12.7
4s1t	64/64	1.5/12	7	2.9	64/64	1.5/12	7	1.9	64/64	1.5/12	7	2.6	64/64	1.5/12	7	3.7
4s2t	64/64	1.5/12	6	4.5	64/64	1.5/12	6	2.9	64/64	1.5/12	6	3.9	64/64	1.5/12	6	5.8
4s4t	64/64	1.5/12	5	6.6	64/64	1.5/12	5	4.1	64/64	1.5/12	5	5.6	64/64	1.5/12	5	7.8
4s8t	64/64	1.5/12	4	8.5	64/64	1.5/12	4	5.5	64/64	1.5/12	4	7.2	64/64	1.5/12	4	9.1

Note: The L1 refers to the primary data/instruction cache size. The L2 cache configuration size (MB)/set associativity (SA) are provided along with the total number of cores for that CMT configuration.

Power/Energy Basics

- Energy = Power x time
- Power = Dynamic power + Leakage power
- Dynamic Power = α C V^2 *f*

  α switching activity factor
  C capacitances being charged
  V voltage swing
  f processor frequency

Guidelines

- Dynamic frequency scaling (DFS) can impact power, but has little impact on energy
- Optimizing a single structure for power/energy is good for overall energy only if execution time is not increased
- A good metric for comparison: ED^2 (because DVFS is an alternative way to play with the E-D trade-off)
- Clock gating is commonly used to reduce dynamic energy, DFS is very cheap (few cycles), DVFS and power gating are more expensive (micro-seconds or tens of cycles, fewer margins, higher error rates)

Criticality Metrics

- Criticality has many applications: performance and power; usually, more useful for power optimizations

- QOLD – instructions that are the oldest in the issueq are considered critical
  - can be extended to oldest-N
  - does not need a predictor
  - young instrs are possibly on mispredicted paths
  - young instruction latencies can be tolerated
  - older instrs are possibly holding up the window
  - older instructions have more dependents in the pipeline than younger instrs

Other Criticality Metrics

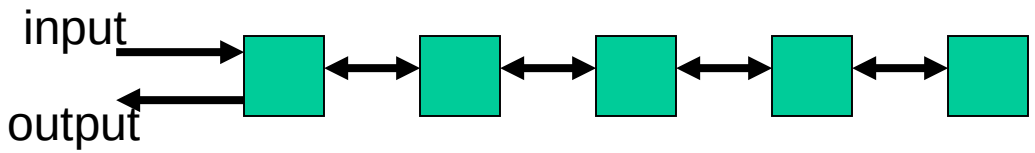
- QOLDDEP: Producing instructions for oldest in q
- ALOLD: Oldest instr in ROB
- FREED-N: Instr completion frees up at least N dependent instrs
- Wake-Up: Instr completion triggers a chain of wake-up operations
- Instruction types: cache misses, branch mpreds, and instructions that feed them

Parallel Algorithms – Processor Model

- High communication latencies → pursue coarse-grain parallelism (the focus of the course so far)
- Next, focus on fine-grain parallelism
- VLSI improvements → enough transistors to accommodate numerous processing units on a chip and (relatively) low communication latencies
- Consider a special-purpose processor with thousands of processing units, each with small-bit ALUs and limited register storage

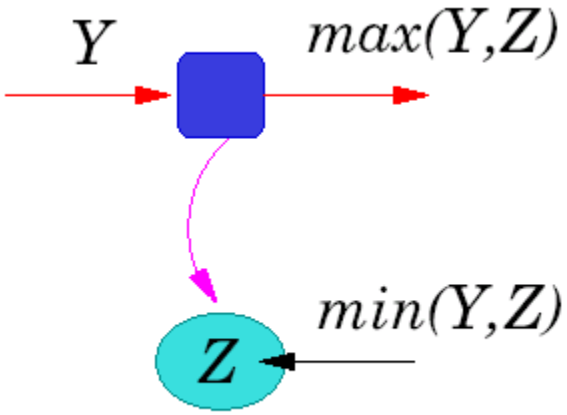
Sorting on a Linear Array

- Each processor has bidirectional links to its neighbors
- All processors share a single clock (asynchronous designs will require minor modifications)
- At each clock, processors receive inputs from neighbors, perform computations, generate output for neighbors, and update local storage

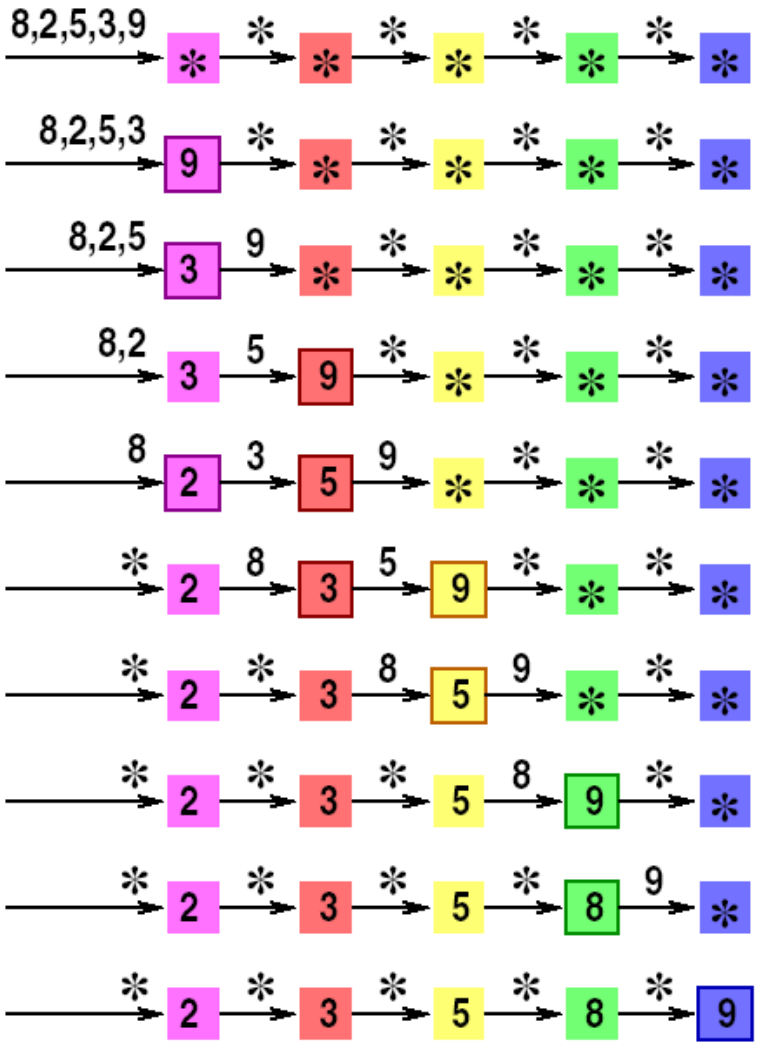


Control at Each Processor

- Each processor stores the minimum number it has seen
- Initial value in storage and on network is "∗", which is bigger than any input and also means "no signal"
- On receiving number Y from left neighbor, the processor keeps the smaller of Y and current storage Z, and passes the larger to the right neighbor



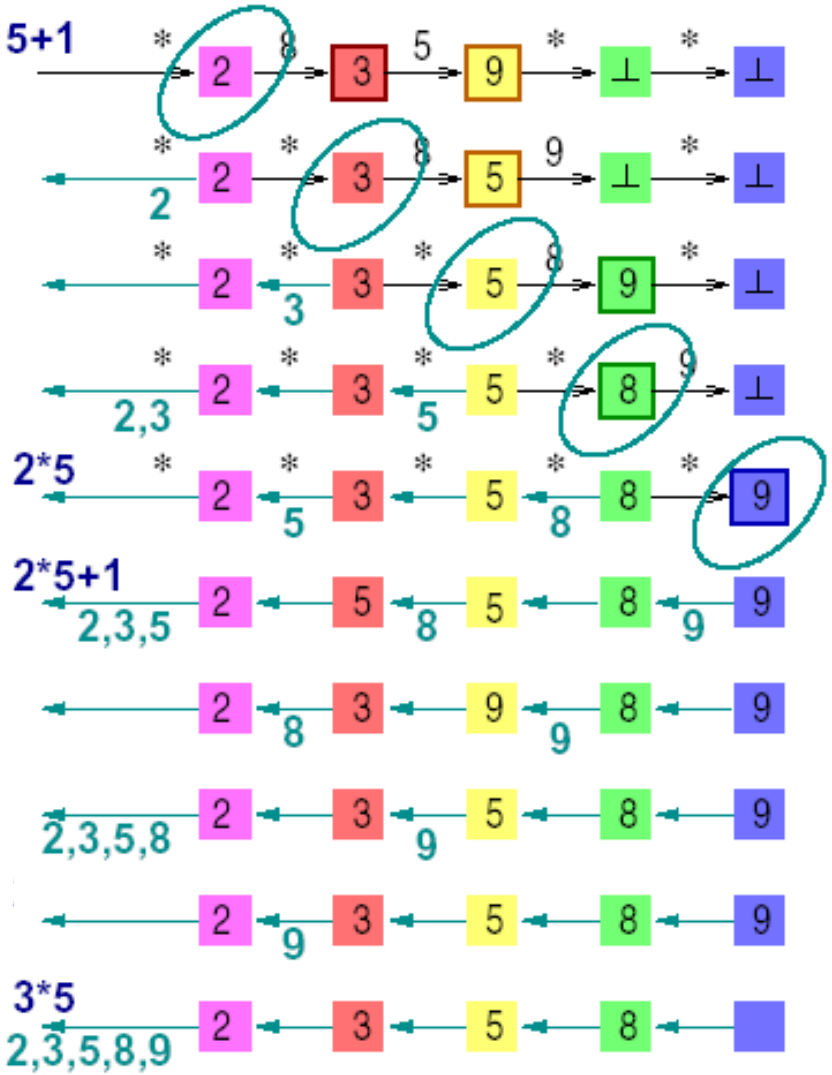
Sorting Example



Result Output

- The output process begins when a processor receives a non-∗, followed by a “∗”
- Each processor forwards its storage to its left neighbor and subsequent data it receives from right neighbors
- How many steps does it take to sort N numbers?
- What is the speedup and efficiency?

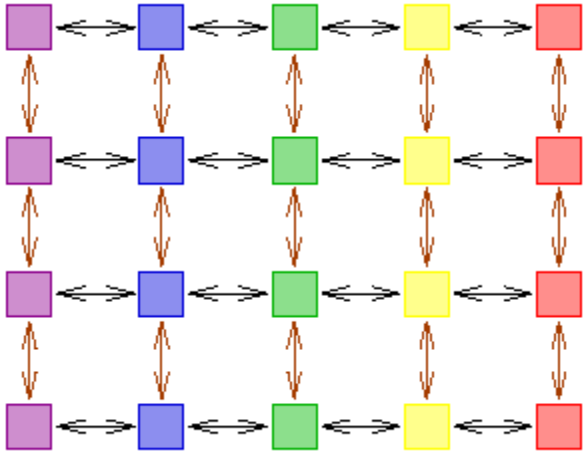
Output Example



Bit Model

- The bit model affords a more precise measure of complexity – we will now assume that each processor can only operate on a bit at a time

- To compare N k-bit words, you may now need an N x k 2-d array of bit processors



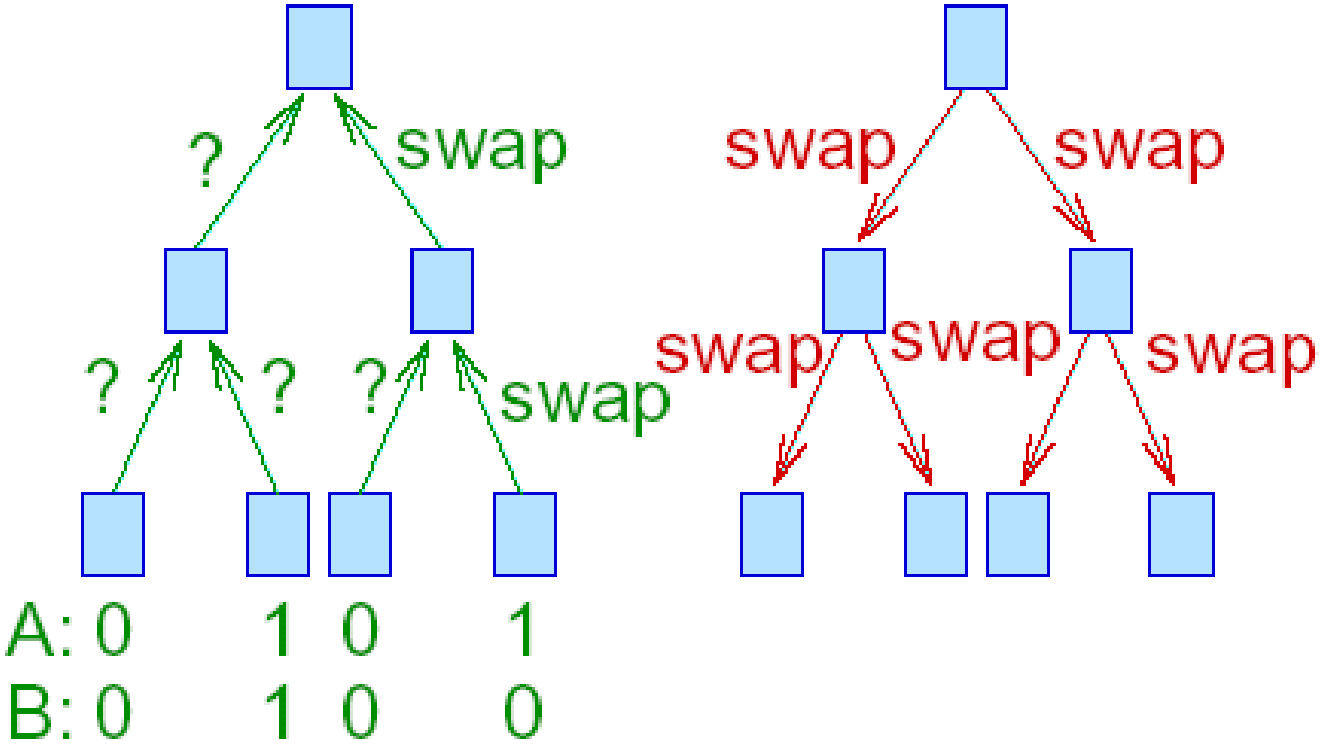
Comparison Strategies

- Strategy 1: Bits travel horizontally, keep/swap signals travel vertically – after at most 2k steps, each processor knows which number must be moved to the right – 2kN steps in the worst case

- Strategy 2: Use a tree to communicate information on which number is greater – after 2logk steps, each processor knows which number must be moved to the right – 2Nlogk steps

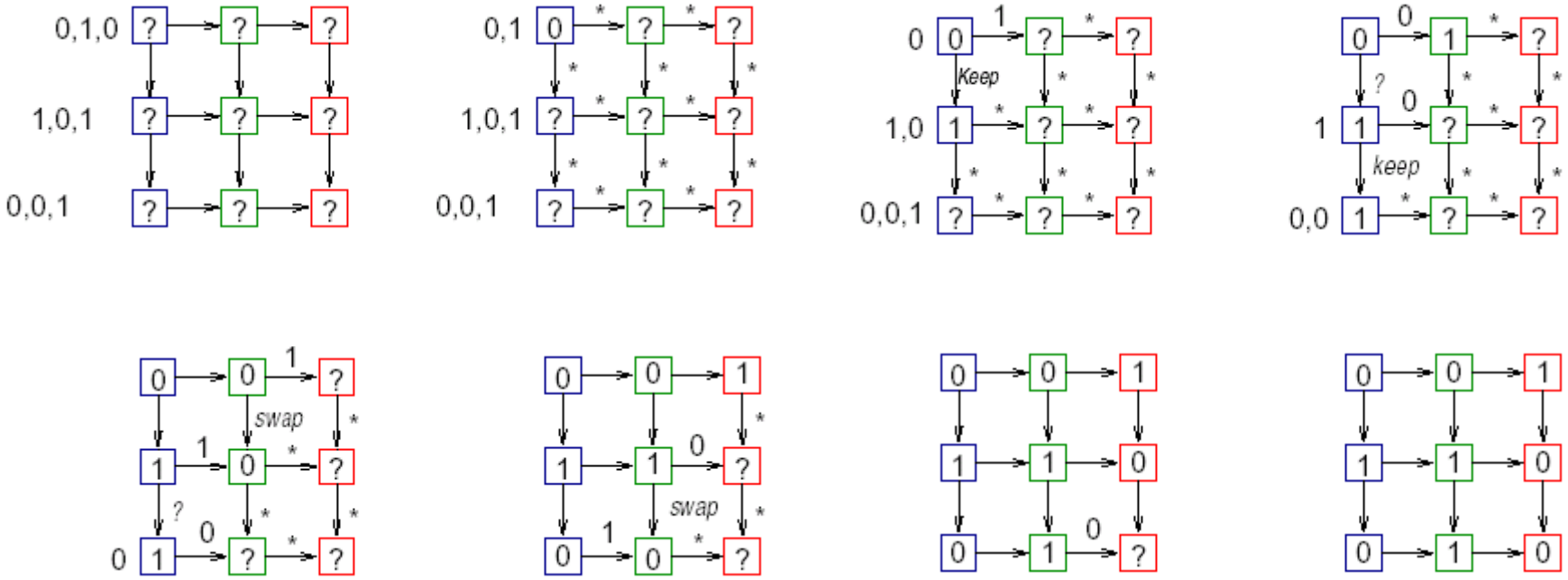
- Can we do better?

Strategy 2: Column of Trees



Pipelined Comparison

Input numbers: 3 4 2
0 1 0
1 0 1
1 0 0



Complexity

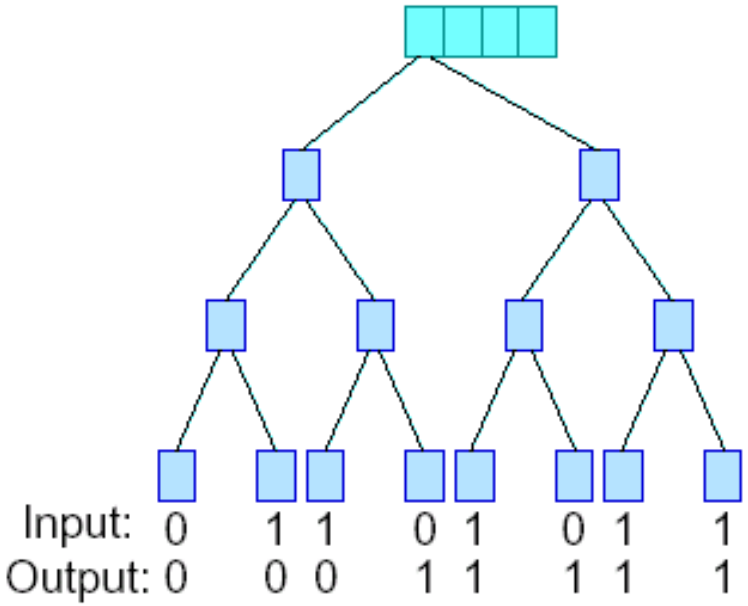
- How long does it take to sort N k-bit numbers?
  (2N – 1) + (k – 1) + N (for output)
- (With a 2d array of processors) Can we do even better?
- How do we prove optimality?

Lower Bounds

- Input/Output bandwidth: Nk bits are being input/output with k pins – requires $\Omega(N)$ time
- Diameter: the comparison at processor (1,1) influences the value of the bit stored at processor (N,k) – for example, N-1 numbers are 011..1 and the last number is either 00…0 or 10…0 – it takes at least N+k-2 steps for information to travel across the diameter
- Bisection width: if processors in one half require the results computed by the other half, the bisection bandwidth imposes a minimum completion time

Counter Example

- N 1-bit numbers that need to be sorted with a binary tree
- Since bisection bandwidth is 2 and each number may be in the wrong half, will any algorithm take at least N/2 steps?



Counting Algorithm

- It takes O(logN) time for each intermediate node to add the contents in the subtree and forward the result to the parent, one bit at a time
- After the root has computed the number of 1's, this number is communicated to the leaves – the leaves accordingly set their output to 0 or 1
- Each half only needs to know the number of 1's in the other half (logN-1 bits) – therefore, the algorithm takes Ω(logN) time
- Careful when estimating lower bounds!

Title

- Bullet