Lecture 26: May 16, 2002

Today: Parallel algorithms

Final !!!

- May 22nd (Wed), 1:30-4:30PM, duPont
- Open-book, Open-notes
  (but no other books/bibles/photocopies)
- Material: everything excluding today's lecture. This includes (but is not limited to):
  - Algorithms for various problems: sorting & selection , various data structures (searching, disjoint sets, priority queues) , graph and network problems , computational geometry
  - Techniques for designing efficient algorithms: divide & conquer , randomization , binary search trees , hashing , greedy & dynamic programming , approximation algorithms
- Review session: 34-301, 34-302, 34-303 TODAY, 6-9 pm
- Office hours to be announced

Today

- Take advantage of *hardware* to solve problems
  - Multiple processing units
  - Solve problems by decomposing them into smaller pieces
- By proper parallelisation, can achieve $o(n)$ running times
- Example: sorting

Parallel computers

Many models for parallel computing:

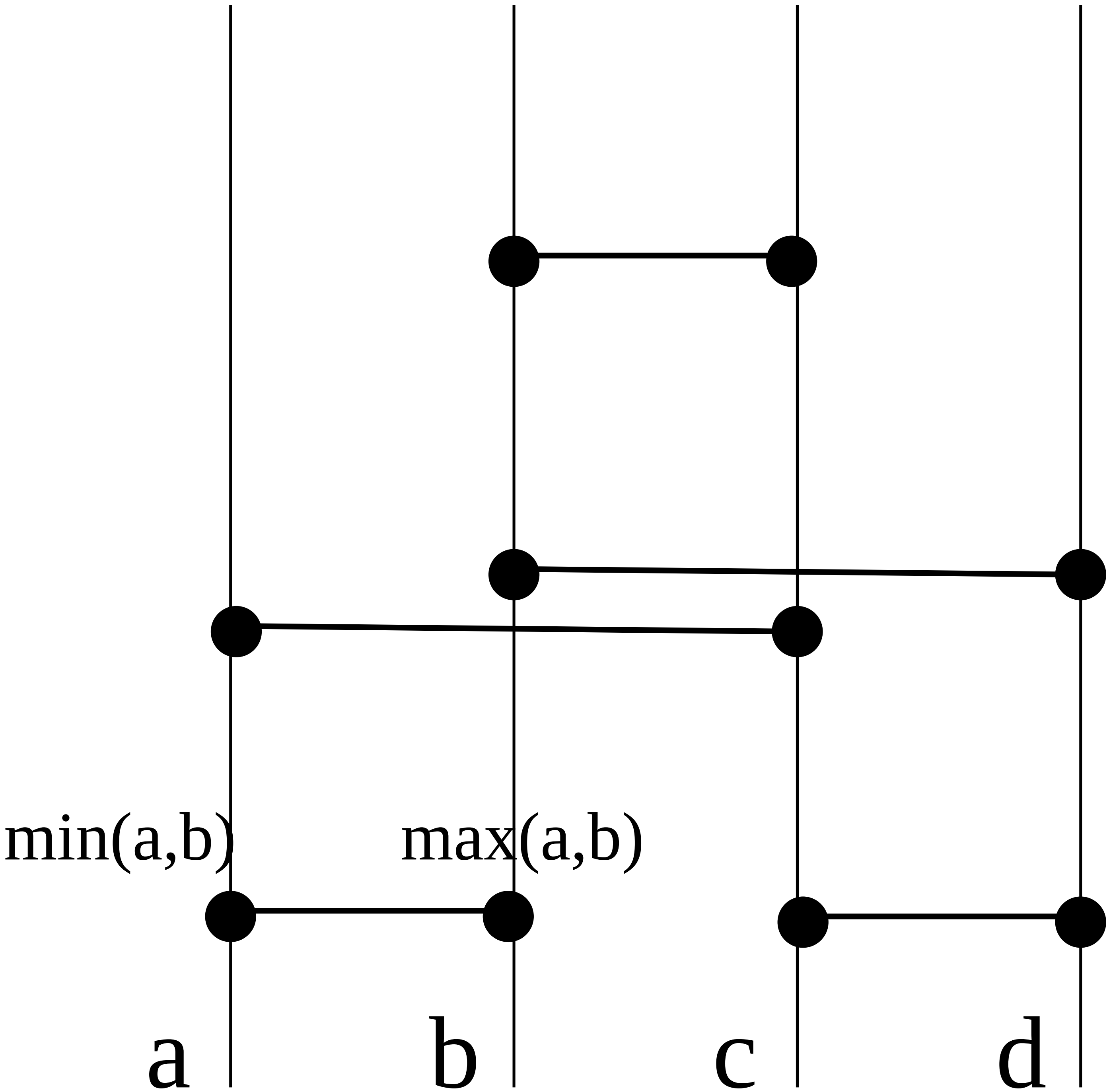
- Parallel RAM: many processors, shared memory

- Networks: processors connected e.g., in a tree

- This lecture: sorting networks

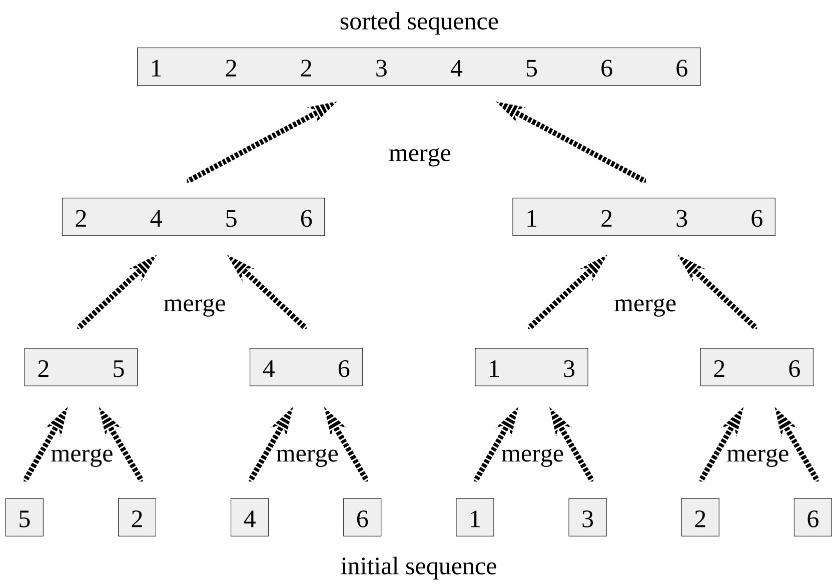
Sorting networks

Networks that are "born to sort".



Sorting

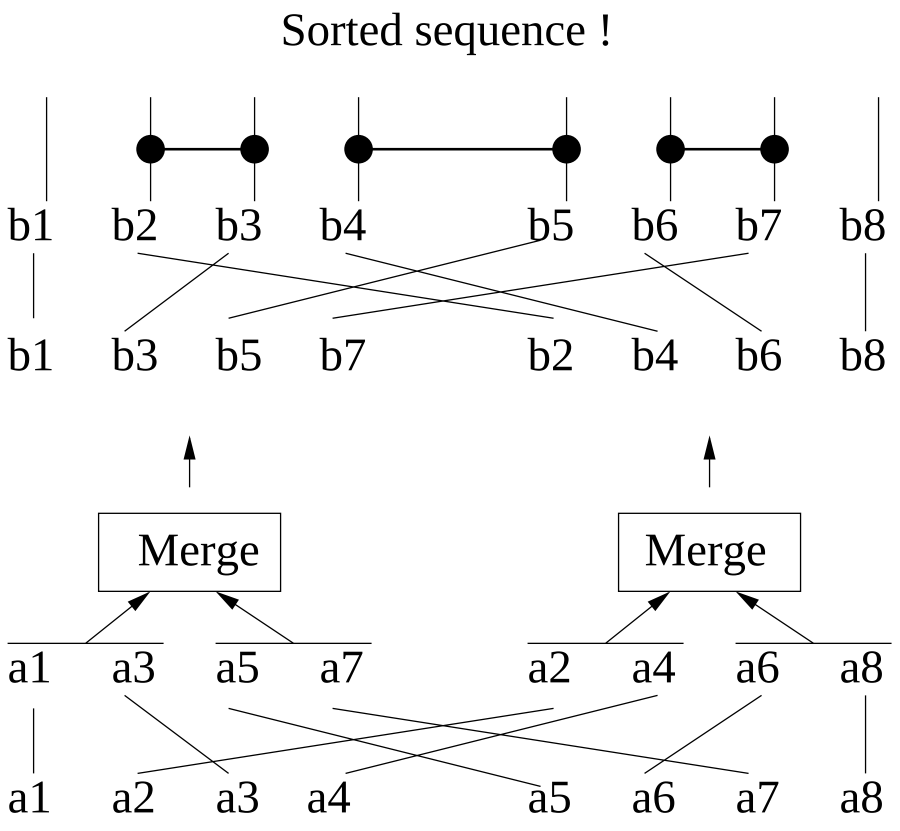
- We know how to sort $n = 4$ elements
- What about general n ?
  - Assume n is a power of 2
  - Mergesort



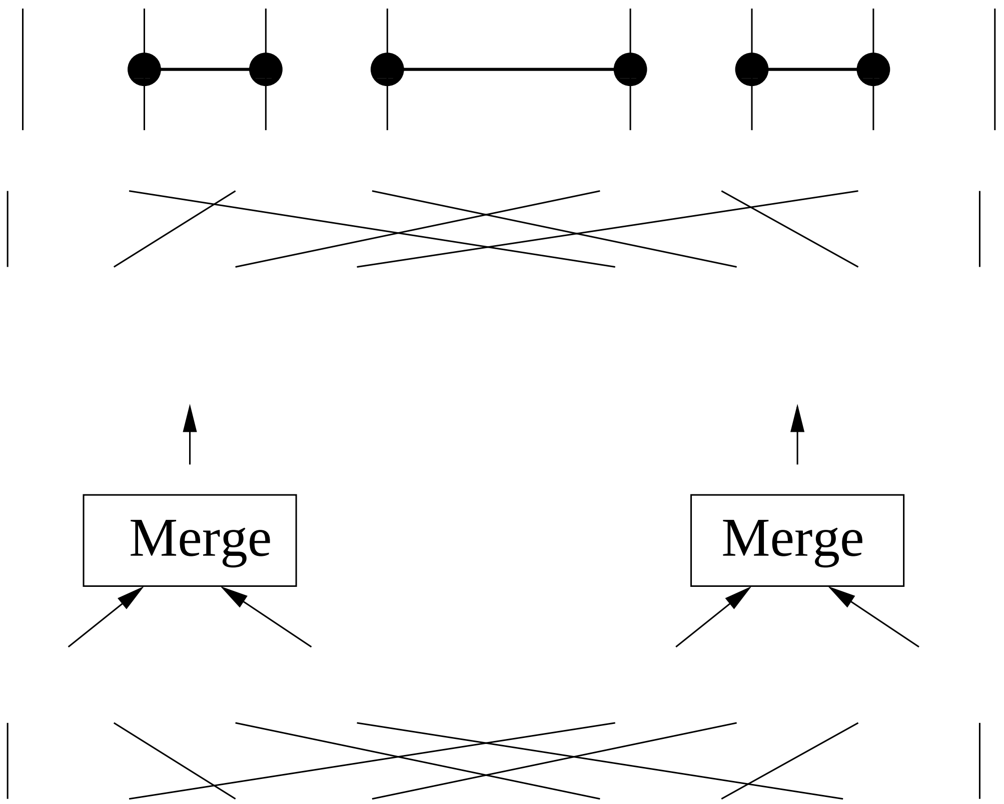
  - Sufficient to design a merging network
  - Parallel merging time $T \Rightarrow$ parallel sorting time $T \log n$

Merging

- Odd-even merging network (Batcher)
- Recursive design:
  - Shuffle & Split into 2 parts
  - Recursively merge each part
  - Re-Shuffle & Join the parts
  - Perform final comparisons



“Merge-it-yourself” Example

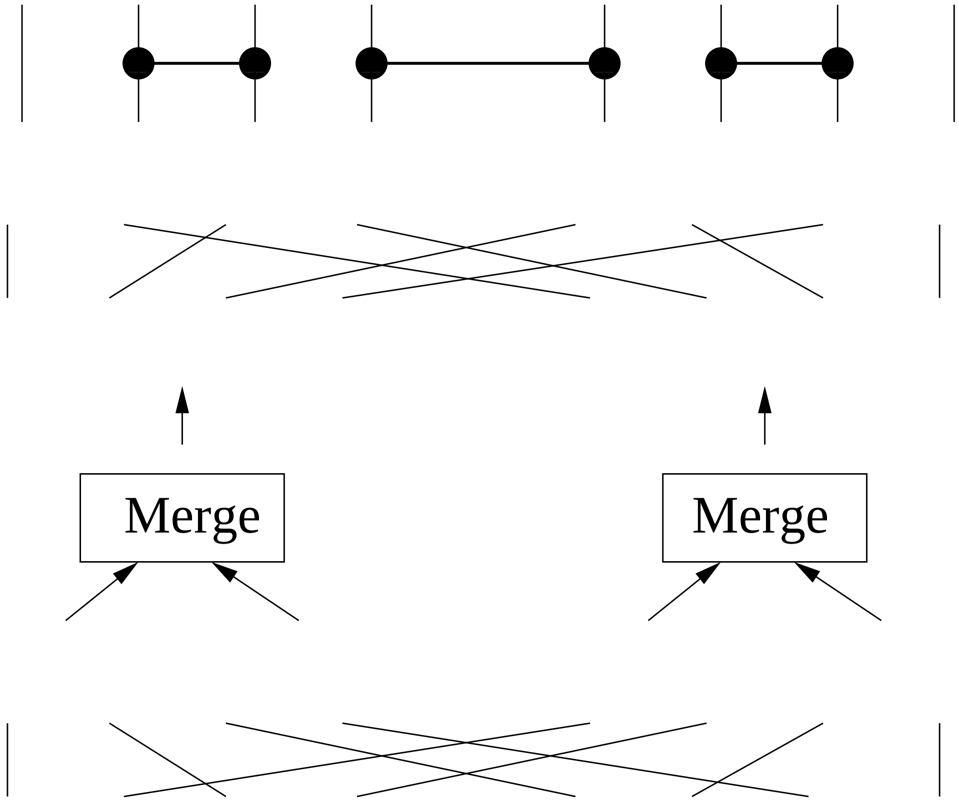


Analysis

Whatever it does, it does it fast !

- We perform
  - Parallel rearrangement (costs nothing)
  - Recursive merge (in parallel)
  - One final parallel step
- $T(n) = T(n/2) + 1 \Rightarrow T(n) = \log n$

Correctness (for 0's and 1's)



Sorting (for 0's and 1's)

- Merging in parallel time $O(\log n)$
- Sorting in parallel time $O(\log^2 n)$
- Total number of operations: $O(n \log^2 n)$

Can we sort something more than 0's and 1's ?

0-1 principle

If a sorting network correctly sorts every sequence of 0's and 1's, then it correctly sorts anything (i.e., is correct).

Proof: Assume that $a_1 \dots a_n$ is *not* sorted by our network NET. I.e., the sequence $b_1 \dots b_n = NET(a_1 \dots a_n)$ is not sorted. Recall that

$$COMP(a_1, a_2) = (\min(a_1, a_2), \max(a_1, a_2))$$

- $COMP(f(a_1), f(a_2)) = f(COMP(a_1, a_2))$, for f non-decreasing, with values in $\{0, 1\}$
- $NET(f(a_1) \dots f(a_n)) = f(NET(a_1 \dots a_n))$
- We construct f such that if $b_1 \dots b_n$ is not sorted, then $f(b_1) \dots f(b_n)$ is not sorted. Then

$$\begin{aligned} f(b_1) \dots f(b_n) &= f(NET(a_1 \dots a_n)) \\ &= NET(f(a_1) \dots f(a_n)) \end{aligned}$$

$$COMP(f(a_1), f(a_2)) = f(COMP(a_1, a_2))$$

Need to show

- $\min(f(a_1), f(a_2)) = f(\min(a_1, a_2))$.
  Assume $a_1 < a_2$.
  Then $\min(a_1, a_2) = a_1$.
  Moreover $f(a_1) \leq f(a_2)$ (by monotonicity).
  Thus $\min(f(a_1), f(a_2)) = f(a_1) = f(\min(a_1, a_2))$.

- $\max(f(a_1), f(a_2)) = f(\max(a_1, a_2))$ - same argument

Constructing monotone f

Recall $NET(a_1 \ldots a_n)$ not sorted. In other words, in $b_1 \ldots b_n$ there are $b_i > b_j$, $i < j$. Construct f such that

- $f(x) = 0$ if $x \leq b_j$
- $f(x) = 1$ if $x > b_j$

Note that $f(b_j) = 0, f(b_i) = 1$. But this means that a sequence $f(b_1) \ldots f(b_n)$ is not sorted.

Summing up

- Sorting in $O(\log^2 n)$ time
- Can be done in $O(\log n)$ time - incredibly complex
- Lots of other very interesting parallel algorithms
- Lots of very interesting algorithms in general
  - Advanced Algorithms
  - Randomized Algorithms (next Spring)
  - Algorithms for Massive Data Sets (this Fall)