Sorting Networks

Mark Greenstreet

CpSc 418 – October 2, 2018

- Parallelizing mergesort and/or quicksort
- Sorting Networks
- The 0-1 Principle
- Summary

Unless otherwise noted or cited, these slides are copyright 2018 by Mark Greenstreet and are made available under the terms of the Creative Commons Attribution 4.0 International license
http://creativecommons.org/licenses/by/4.0/

Pause, Breath, Where are We?

We are ahead of schedule! ☺

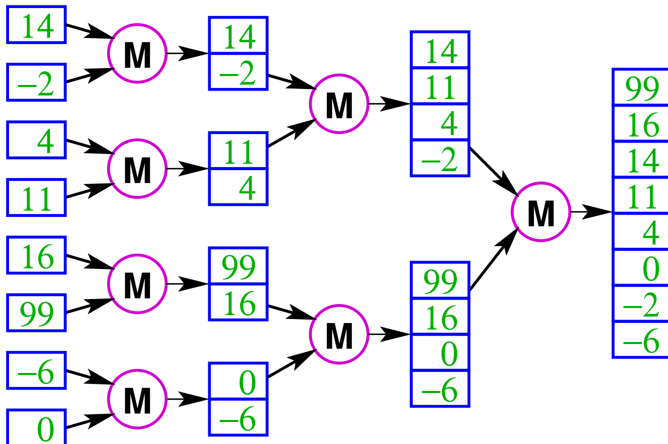
- Introduction to Erlang: **done.**
- Reduce & Scan: **done.**
- Parallel Architecture: **done.**
- Parallel Performance: **done.**
- Parallel Sorting: *starting today*
- Data Parallel Computation: probably start Oct. 29.

Midterm: Oct. 22

- Open book, notes, homework, solutions
  - Anything printed on paper.
  - No communication devices: e.g. computers, phones, tablets, etc.
  - Calculators allowed: programmable, graphing, whatever.
  - This is not a memorization exam
    - Bring stuff so you don't stress about making sure you have every obscure formula memorized.
    - Know the concepts we've been covering in class and homework – don't plan to study during the exam.
- Other details
  - 50 minutes
  - Covers Sept. 5 through Oct. 12.
  - Probably 4 questions, probably "do all four"
  - Two students have contacted me about late arrival at the exam.
    - The exam will start promptly at 1pm, and end at 1:50pm.
    - If you contact me by e-mail prior to the exam stating why you cannot be sure to be in DMP 301 by 1pm,
      - You can arrive between 1pm and 1:10pm.
      - I will mark your arrival time on your exam.
      - You will have 50 minutes to complete the exam.

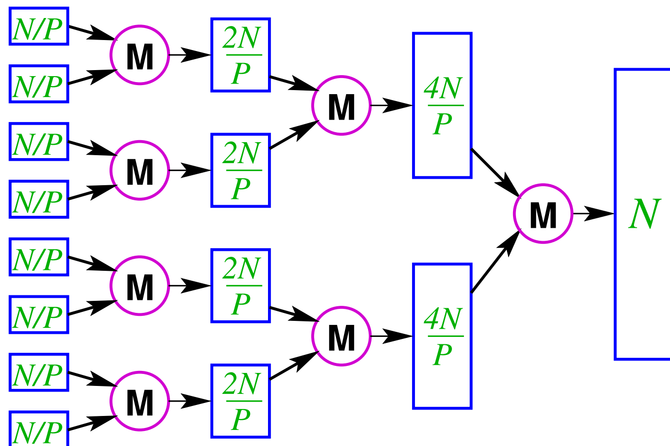
Parallelizing Mergesort

We could use reduce?



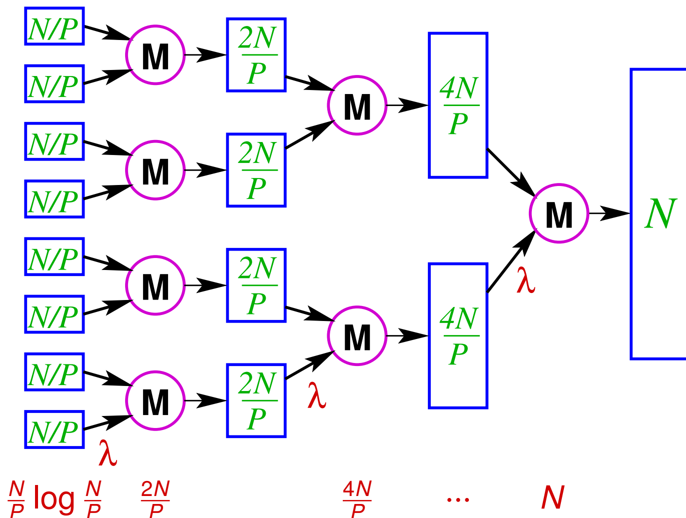
Parallelizing Mergesort

We could use reduce?



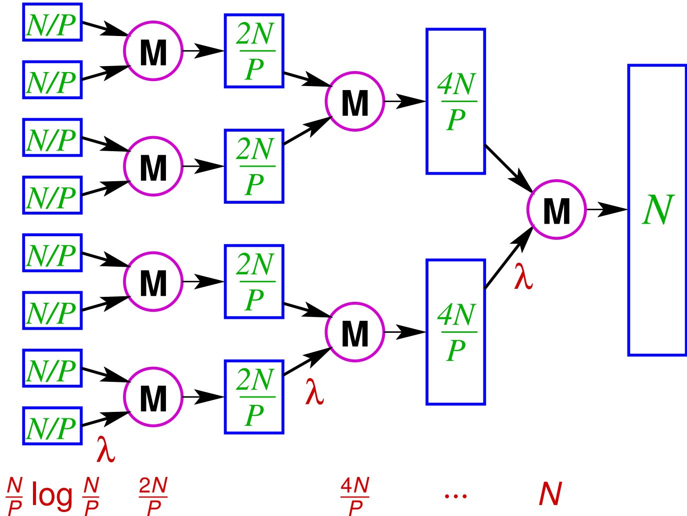
Parallelizing Mergesort

We could use reduce?



Parallelizing Mergesort

We could use reduce?

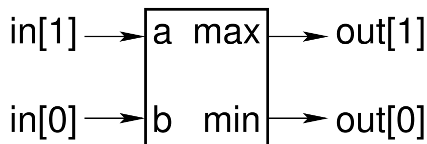


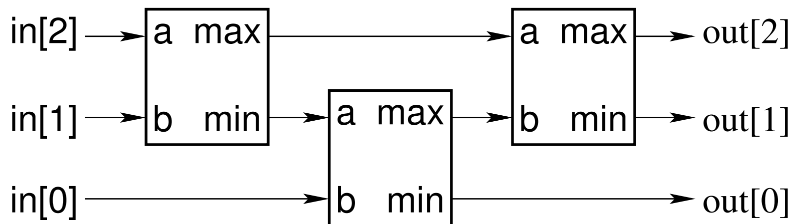
Total time:

Parallelizing Quicksort

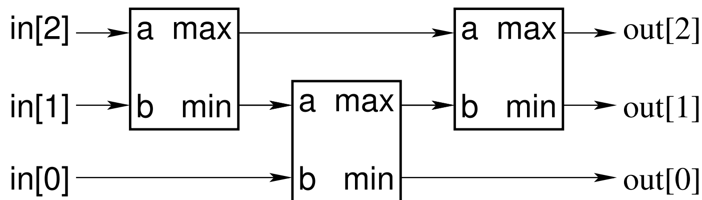
How would you write a parallel version of quicksort?

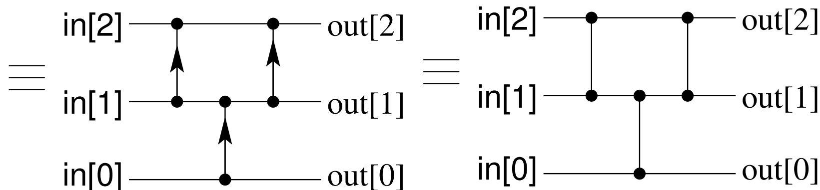
Sorting Networks



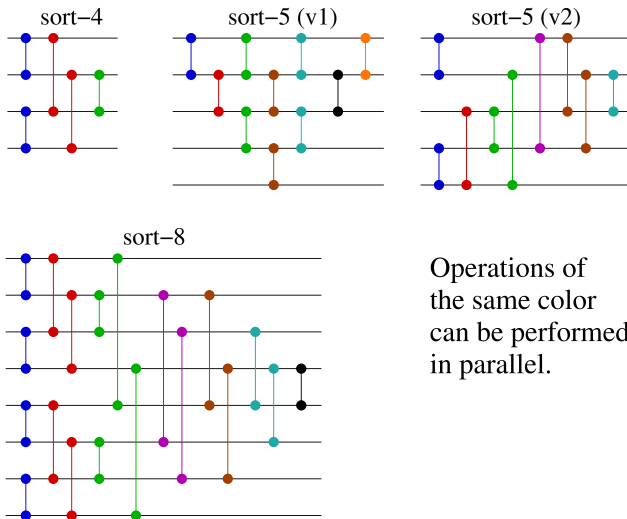


Sorting Networks – Drawing





Sorting Networks – Examples



Operations of the same color can be performed in parallel.

See: http://pages.ripco.net/~jgamble/nw.html

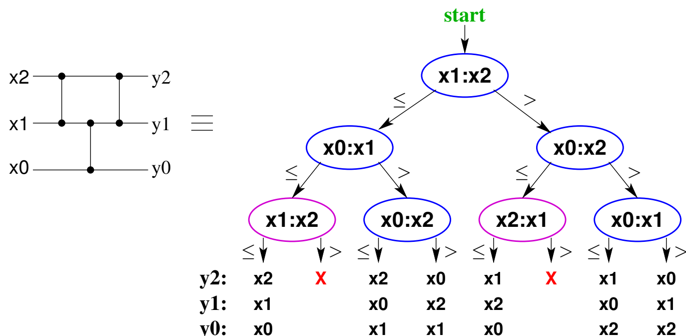
Sorting Networks: Definition

Structural version:

- A sorting network is an acyclic network consisting of compare-and-swap modules.
  - Each primary input is connected either to the input of exactly one compare-and-swap module or to exactly one primary output.
  - Each compare-and-swap input is connected either to a primary input or to the output of exactly one compare-and-swap module.
  - Each compare-and-swap output is connected either to a primary output or to the input of exactly one compare-and-swap module.
  - Each primary output is connected either to the output of exactly one compare-and-swap module or to exactly one primary input.
- More formally, a sorting network is either
  - the identity network (no compare and swap modules).
  - a sorting network, S composed with a compare-and-swap module such that two outputs of S are the inputs to the compare-and-swap, and the outputs of the compare-and-swap are outputs of the new sorting network (along with the other outputs of the original network).

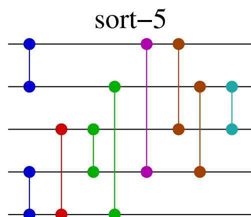
Sorting Networks: Definition

Decision-tree version:



- Let v be an arbitrary vertex of a decision tree, and let x_i and x_j be the variables compared at vertex v.
- A decision tree is a sorting network iff for every such vertex, the left subtree is the same as the right subtree with x_i and x_j exchanged.

Sorting Networks: Connection with Work-Flow Graphs



Operations of the same color can be performed in parallel.

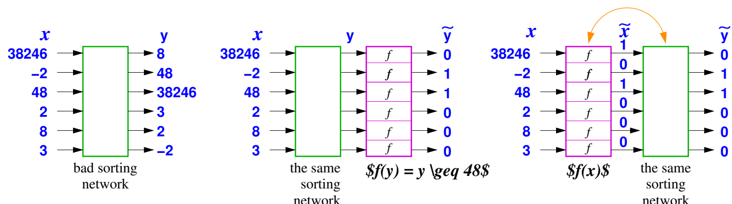
- Each compare-and-swap is a vertex of the work-flow graph.
- Connections between compare-and-swaps are edges in the work-flow graph
- Work: number of compare-and-swap elements.
- Span: number of "levels" in the "most-compact" drawing of the network.

The 0-1 Principle

If a sorting network correctly sorts all inputs consisting only of 0s and 1s, then it correctly sorts inputs consisting of arbitrary (comparable) values.

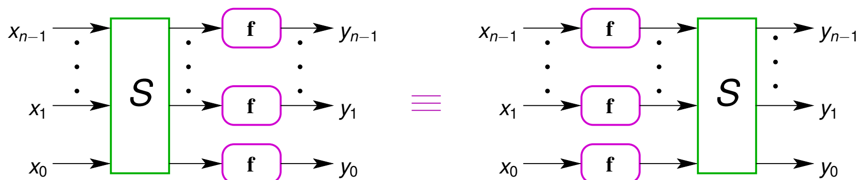
- The 0-1 principle doesn't hold for arbitrary algorithms:
  - Consider the following linear-time "sort"
  - In linear time, count the number of zeros, *nz*, in the array.
  - Set the first *nz* elements of the array to zero.
  - Set the remaining elements to one.
  - This correctly sorts any array consisting only of 0s and 1s, but does not correctly sort other arrays.
- By restricting our attention to sorting networks, we can use the 0-1 principle.

The 0-1 Principle: Proof Sketch



- We will show the contrapositive: if y is not sorted properly, then there exists an $\tilde{x}$ consisting of only 0s and 1s that is not sorted properly.
- Key idea: let f be any monotonic function
  - We will show that applying f to every input of a sorting network, S, and then performing the sorting network, produces the same results as applying the sorting network first and then applying f to each output of the sorting network.
  - More succinctly, we will show $S \circ f \equiv f \circ S$
  - If a sorting network does not sort correctly:
    - Choose $i < j$ such that $y_i > y_j$.
    - Let $f(x) = x \geq y_i$
    - Then $S \circ f$ is all 0s and 1s and not sorted correctly.
    - Thus, $f \circ S$ produces an unsorted output, and the input to S is all 0s and 1s.

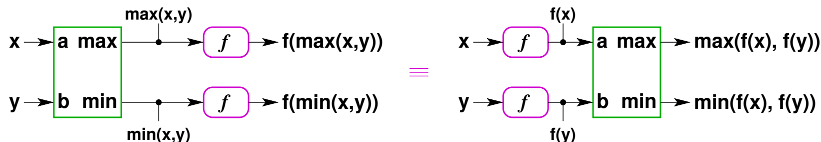
Monotonicity Lemma



Lemma: sorting networks commute with monotonic functions.

- Let S be a sorting network with n inputs an N outputs.
  - I'll write $x_0, \ldots, x_{n-1}$ to denote the inputs of S.
  - I'll write $y_0, \ldots, y_{n-1}$ to denote the outputs of S.
- Let f be a monotonic function.
  - If $x \leq y$, then $f(x) \leq f(y)$.
- The monotonicity lemma says
  - applying S and then f produces the same result as
  - applying f and then S.
- Observation: `f(X) when X <` X_i `-> 0; f(_) -> 1.` is monotonic.

Compare-and-Swap Commutes with Monotonic Functions



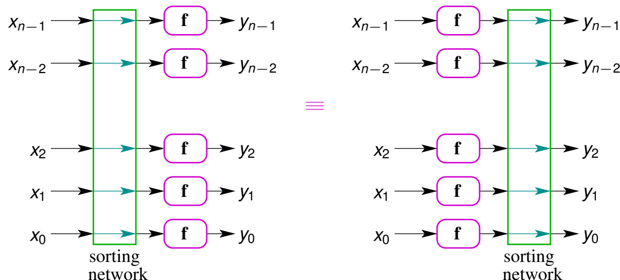
Compare-and-Swap commutes with monotonic functions.

- Case $x \leq y$:

$$\begin{array}{rcll} f(x) & \leq & f(y), & \text{because } f \text{ is monotonic.} \\ \max(f(x), f(y)) & = & f(y), & \text{because } f(x) \leq f(y) \\ \max(f(x), f(y)) & = & f(max(x, y)), & \text{because } x \leq y \end{array}$$

- Case $x \geq y$: equivalent to the $x \leq y$ case.
- $\square$

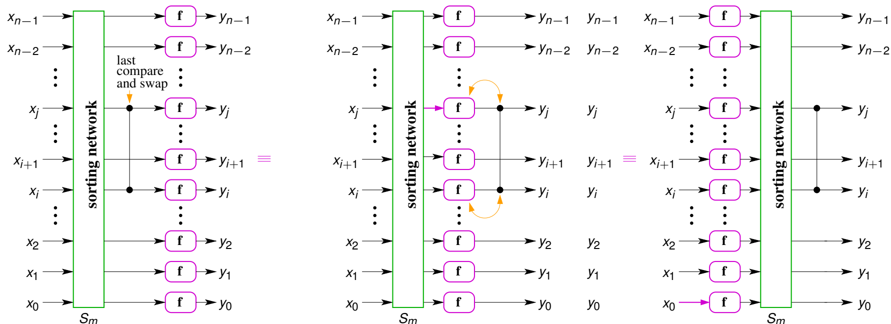
The monotonicity lemma – proof sketch



Induction on the structure of the sorting network, S.
Base case:

- The simplest sorting network, S_0 is the identity function.
- It has 0 compare-and-swap modules.
- Because S_0 is the identity function, $S_0(f(x)) = f(x) = f(S_0(x))$.

The monotonicity lemma – induction step



- Let S_m be a sorting network with n inputs and let $0 \le i < j < n$.
- Let S_{m+1} be the sorting network obtained by composing a compare-and-swap module with outputs i and j of S_m.
- We can "move" the f operations from the outputs of the new compare-and-swap to the inputs (see slide 15).
- We can "move" the f operations from the outputs S_m to the inputs (induction hypothesis).
- Therefore, S_{m+1} commutes with f.

The 0-1 Principle

If a sorting network correctly sorts all inputs consisting only of 0s and 1s, then it correctly sorts inputs of any values.

I'll prove the contrapositive.

- If a sorting network does not correctly sort inputs of any values, then it does not correctly sort all inputs consisting only of 0s and 1s.
- Let S be a sorting network, let x be an input vector, and let $y = S(x)$, such that there exist i and j with $i < j$ such that $y_i > y_j$.
- Let
$$\begin{array}{rcll} f(x) & = & 0, & \text{if } x < y_i \\ & = & 1, & \text{if } x \geq y_i \\ \tilde{y} & = & S(f(x)) & \end{array}$$
- By the definition of f, $f(x)$ is an input consisting only of 0s and 1s.
- By the monotonicity lemma, $\tilde{y} = f(y)$. Thus,
$$\tilde{y}_i \;=\; f(y_i) \;=\; 1 \;>\; 0 \;=\; f(y_j) \;=\; \tilde{y}_j$$
- Therefore, S does not correctly sort an input consisting only of 0s and 1s.
- $\square$

Summary

- Sequential sorting algorithms don't parallelize in an "obvious" way because they tend to have sequential bottlenecks.
  - Later, we'll see that we can combine ideas from sorting networks and sequential sorting algorithms to get practical, parallel sorting algorithms.
- Sorting networks are a restricted class of sorting algorithms
  - Based on compare-and-swap operations.
  - The parallelize well.
  - They don't have control-flow branches – this makes them attractive for architectures with large branch-penalties.
- The zero-one principle:
  - If a sorting-network sorts all inputs of 0s and 1s correctly, then it sorts all inputs correctly.
  - This allows many sorting networks to be proven correct by counting arguments.

Preview

October 17: Bitonic Sorting

October 19: Midterm Review
Homework: HW 3 Due (12 noon).

October 22: Midterm

October 24: Implementing Bitonic Sorting

October 26: Intro. to GPU computing
Reading: Kirk & Hwu – Chapter 2

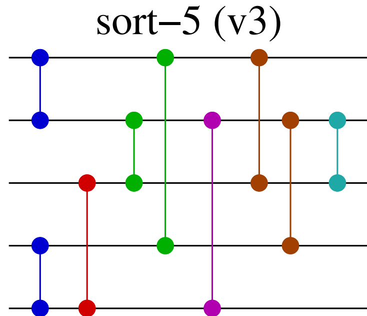
The Midterm:

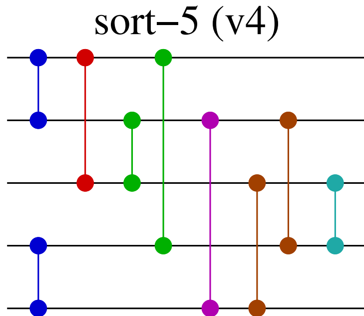
- February 28
- Open book, open notes, open anything on paper, calculators allowed
  - This is a “do you understand the concepts?” test, not a “did you memorize a bunch of trivia?” test.
  - I’ll aim for zero to minimal need for calculators, but won’t guarantee zero.
  - Previous midterms are on the resources page. We’ll cover the same range of concepts, but the questions won’t just be repeats of previous questions with different numbers.
- Two venues: this course is slightly over-subscribed.
  - We expect to hold the midterm in two rooms at the same time.
  - I’ll announce the details when I know them.

Review 1

- Why don't traditional, sequential sorting algorithms parallelize well?
- Try to parallelize another sequential sorting algorithm such as heap sort? What issues do you encounter?
- Consider network sort-5(v2) from slide 8. Use the 0-1 principle to show that it sorts correctly?
  - What if the input is all 0s?
  - What if the input has exactly one 1?
  - What if the input has exactly two 1s?
  - What if the input has exactly three 1s? Note, it may be simpler to think of this the input having exactly two 0s.
  - What if the input has exactly four 1s? Five ones?

Review 2





Consider the two sorting networks shown above. One sorts correctly; the other does not.

- Identify the network that sorts correctly, and prove it using the 0-1 principle.
- Show that the other network does not sort correctly by giving an input consisting of 0s and 1s that is not sorted correctly.

Review 3

I claimed that `max` and `min` can be computed without branches. We could work out the hardware design for a compare-and-swap module. Instead, consider an algorithm that takes two "words" as arguments – each word is represented as a list of characters. The algorithm is supposed to output the two words, but in alphabetical order. For example:

```
% See: http://www.ugrad.cs.ubc.ca/~cs418/2017-2/lecture/src/cas.erl
compareAndSwap(L1, L2) when is_list(L1), is_list(L2) ->
  compareAndSwap(L1, L2, []).
compareAndSwap([], L2, X) ->
  {lists:reverse(X), lists:reverse(X, L2)};
compareAndSwap(L1, [], X) ->
  {lists:reverse(X), lists:reverse(X, L1)};
compareAndSwap([H1 | T1], [H2 | T2], X) when H1 == H2 ->
  compareAndSwap(T1, T2, [H1 | X]);
compareAndSwap(L1=[H1 | _], L2=[H2 | _], X) when H1 < H2 ->
  {lists:reverse(X, L1), lists:reverse(X, L2)};
compareAndSwap(L1, L2, X) ->
  {lists:reverse(X, L2), lists:reverse(X, L1)}.
```

Show that `compareAndSwap` can be implemented as a scan operation.