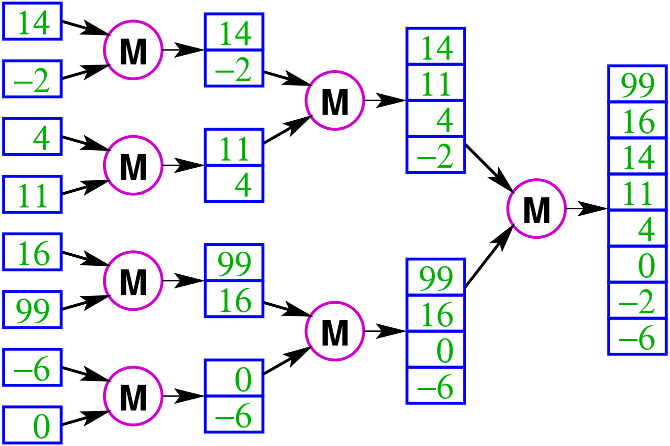
Sorting Networks

Mark Greenstreet

CpSc 418 – Feb. 8, 2017

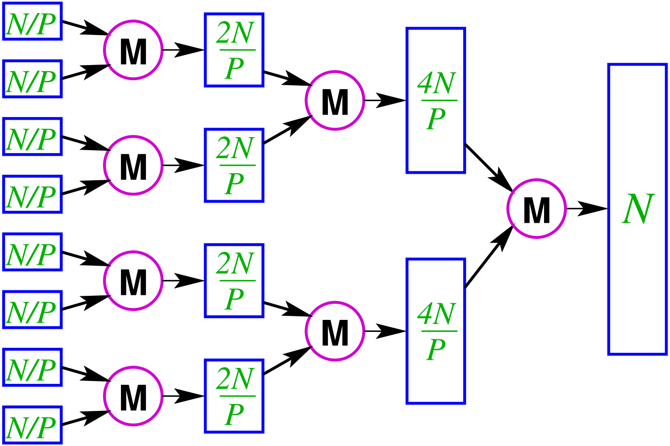
Parallelizing Mergesort

We could use reduce?



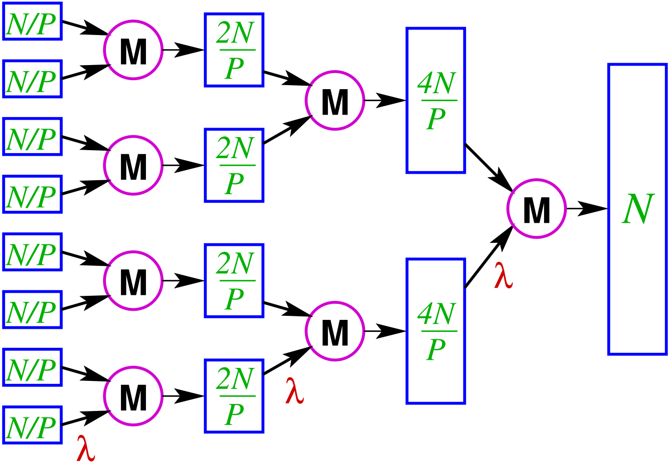
Parallelizing Mergesort

We could use reduce?



Parallelizing Mergesort

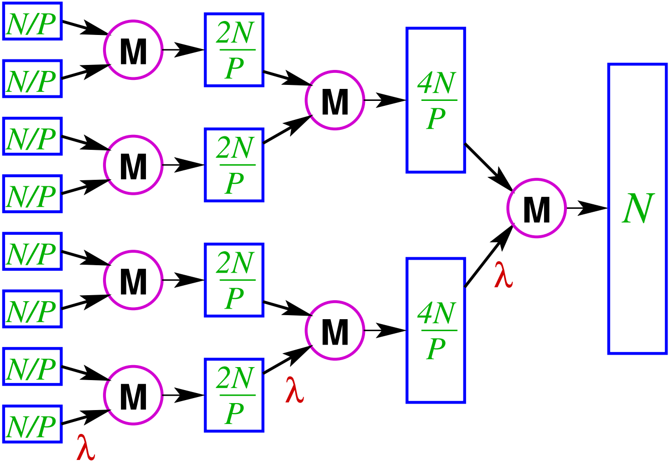
We could use reduce?



$\frac{N}{P} \log \frac{N}{P}$ $\frac{2N}{P}$ $\frac{4N}{P}$ ... N

Parallelizing Mergesort

We could use reduce?



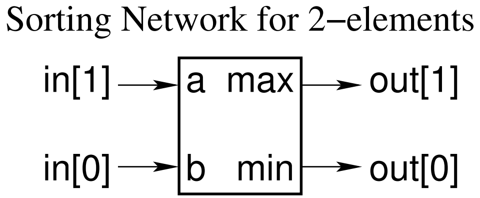
$\frac{N}{P}\log\frac{N}{P}$ $\frac{2N}{P}$ $\frac{4N}{P}$ ... N

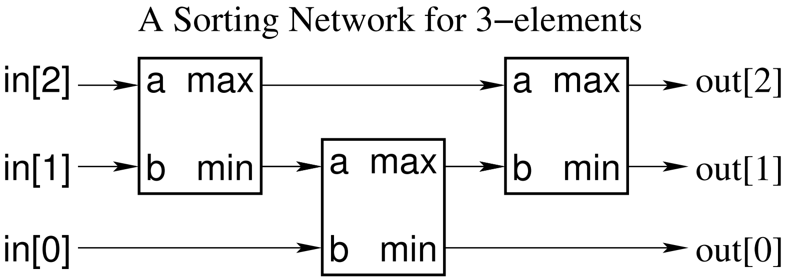
Total time: $\frac{N}{P}(\log N + 2(P-1) - \log P) + (\log P)\lambda$

Parallelizing Quicksort

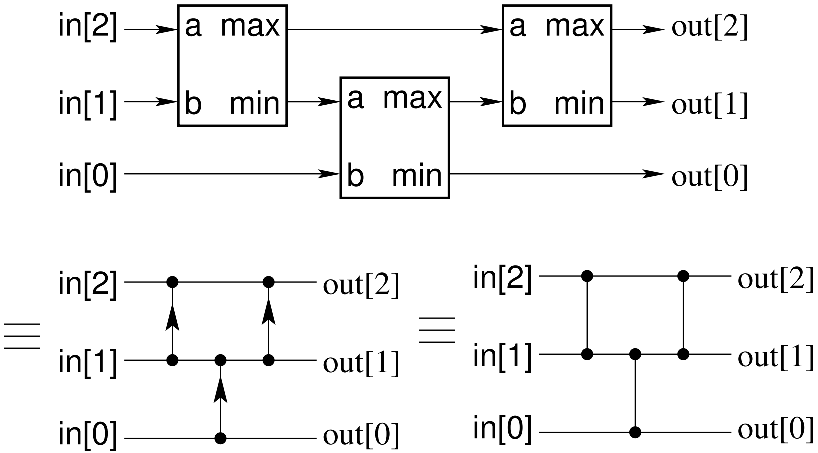
How would you write a parallel version of quicksort?

Sorting Networks

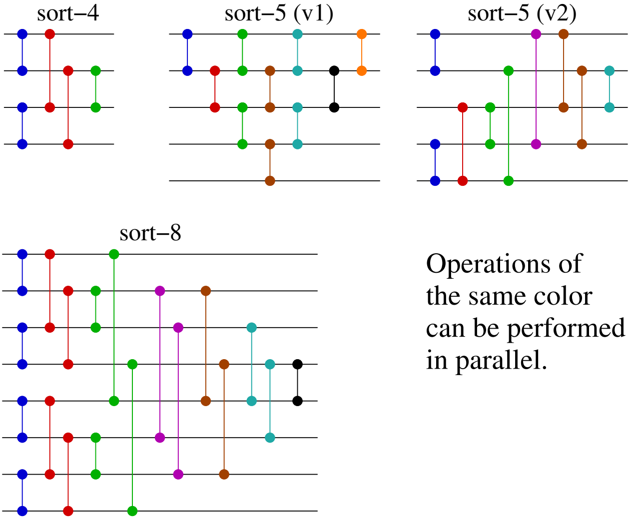




Sorting Networks – Drawing



Sorting Networks – Examples



Operations of the same color can be performed in parallel.

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

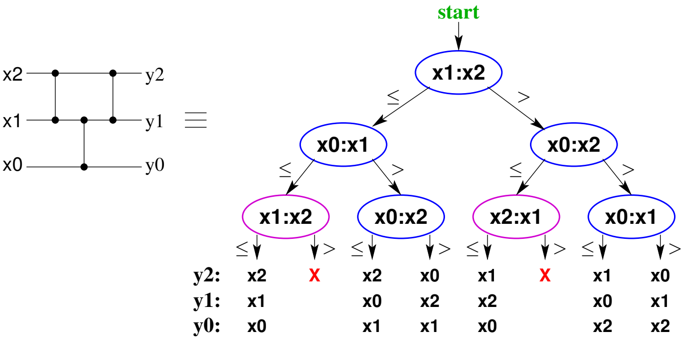
Sorting Networks: Definition

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.

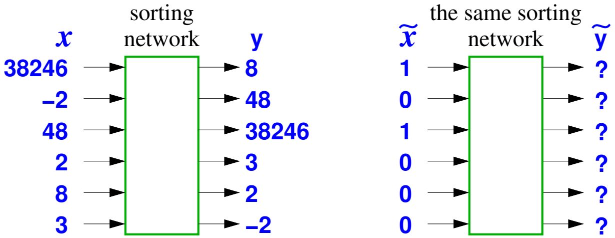
The 0-1 Principle

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.

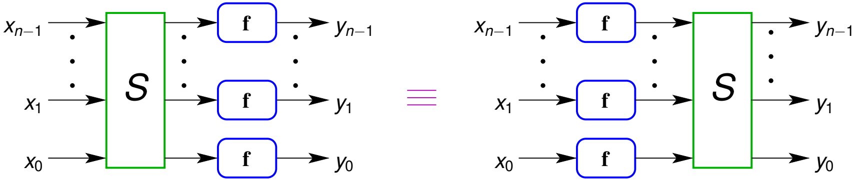
The 0-1 Principle: Proof Sketch

- 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.



- Choose $i < j$ such that $y_i > y_j$.
- Let $\tilde{x}_k = 0$ if $x_k < x_i$ and $\tilde{x}_k = 1$ otherwise.
  - Clearly $\tilde{x}$ consists only of 0s and 1s.
  - We will show that the sorting network does not sort correctly with input $\tilde{x}$.

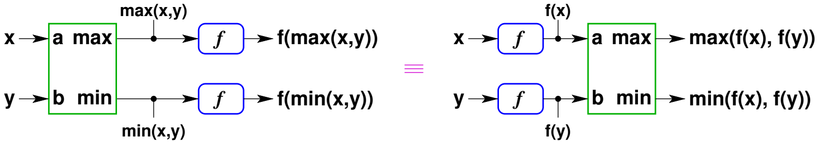
Monotonicity Lemma



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



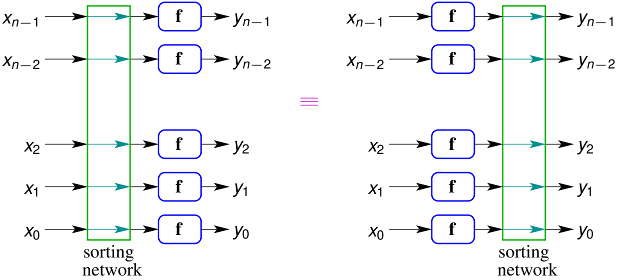
Compare-and-Swap commutes with monotonic functions.

- Case $x \le y$:

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

- Case $x \ge y$: equivalent to the $x \le y$ case.
- $\square$

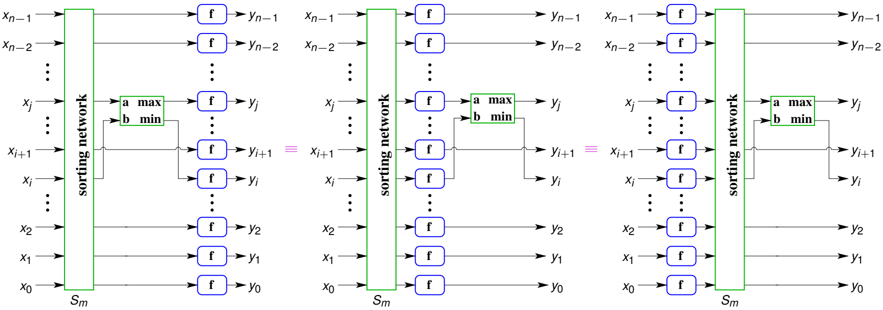
The monotonicity lemma – proof sketch



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 $0 \leq i < j < n$.
- 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 12).
- 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

February 10: Bitonic Sorting (part 1)
Reading: https://en.wikipedia.org/wiki/Bitonic_sorter
http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm

February 13: Family Day – no class

February 15: Bitonic Sorting (part 2)
Homework: HW 3 earlybird (11:59pm), HW 4 goes out.

February 17: Map-Reduce
Homework: HW 3 due (11:59pm).
HW 4 goes out

February 27: TBD

March 1: Midterm

March 3: GPU Overview
Reading The GPU Computing Era

March 6: Intro. to CUDA
Reading Kirk & Hwu Ch. 2

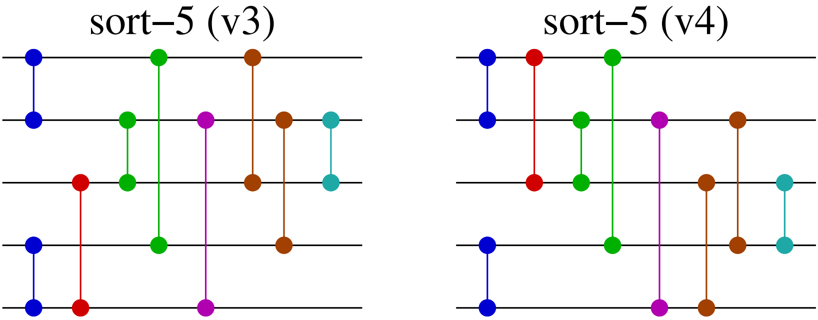
March 8: CUDA Threads, Part 1
Reading Kirk & Hwu Ch. 3
Homework: HW 4 earlybird (11:59pm)

March 8: CUDA Threads, Part 2
Homework: HW4 due (11:59pm).

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 6. 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/~cs418/2016-2/lecture/02-08/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.