Sorting Networks

Mark Greenstreet and Ian M. Mitchell

CpSc 418 – February 2, 2018

Pause, Breath, Where are We?

We are ahead of schedule! ☺

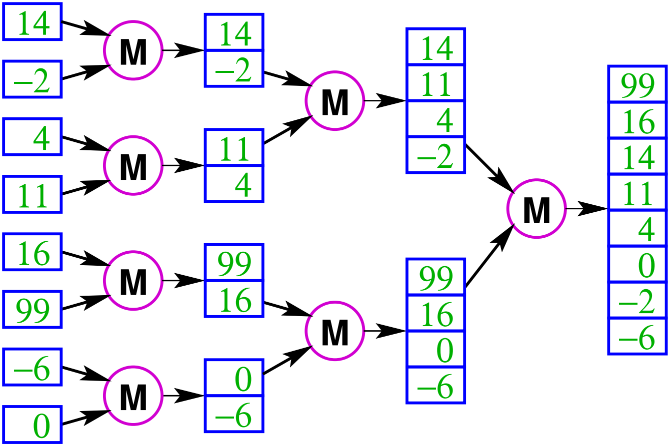
- Introduction to Erlang: **done.**
- Reduce & Scan: **done.**
- Parallel Architecture: **done.**
- Parallel Performance: **done.**
- Parallel Sorting: *starting today*
- Data Parallel Computation: probably start on Feb. 14.

Pause, Breath, Where are We?

- I got the feedback that Wednesday's lecture was kind of fast.
  - *Don't Panic*
  - What do you need to know?
    - There are many models for parallel computation.
    - RAM – the model used for nearly all **sequential algorithm analysis**.
    - PRAM – a parallel version of RAM that is unrealistic because it ignores communication costs and memory access costs. **Primarily of theoretical interest.**
    - Work-Span – models **dependencies**. Provides a clear basis for Amdahl's Law and Gustafson's law. Ignores communication costs. Trying to add communication to Work-Span is messy.
    - CTA – the model we use in class. You can call it the "λ is big" model. **Directly addresses communication costs.** Need to refine the model if we are considering implementations with big messages.
    - logP – **CTA with a few more parameters**. Better known that CTA.
- I'm open to having an "Ask Me Anything" lecture sometime in the next two weeks if that will help
  - If you'd like such a lecture, post questions and topics to piazza. If there is demand, I'll give the lecture.

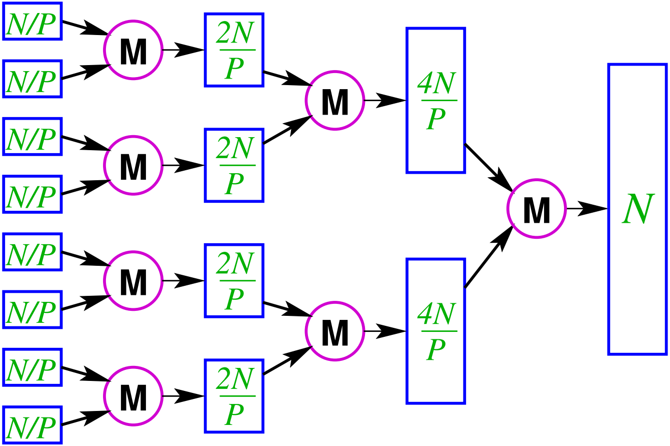
Parallelizing Mergesort

We could use reduce?



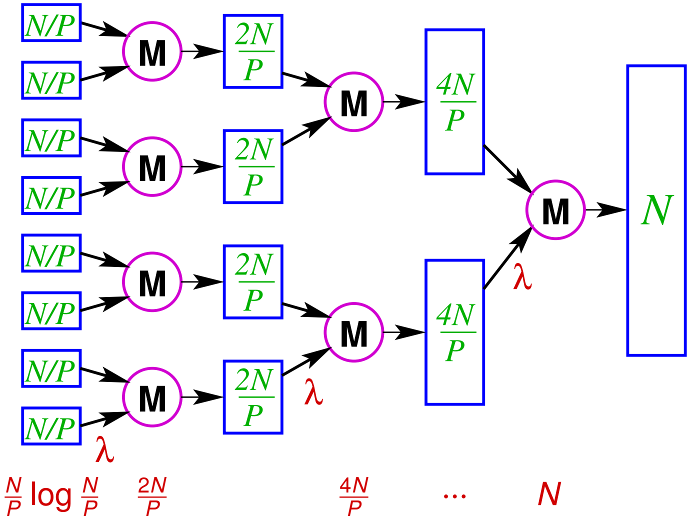
Parallelizing Mergesort

We could use reduce?



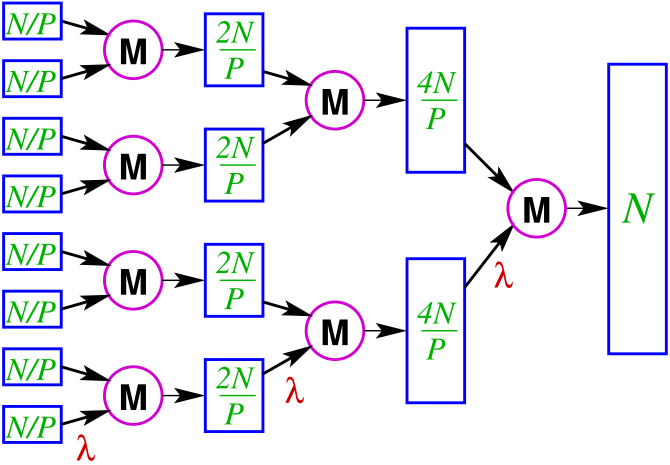
Parallelizing Mergesort

We could use reduce?



Parallelizing Mergesort

We could use reduce?



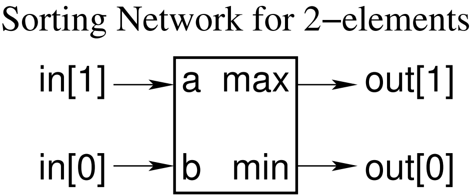
$\frac{N}{P} \log \frac{N}{P}$ $\frac{2N}{P}$ $\frac{4N}{P}$ ... N

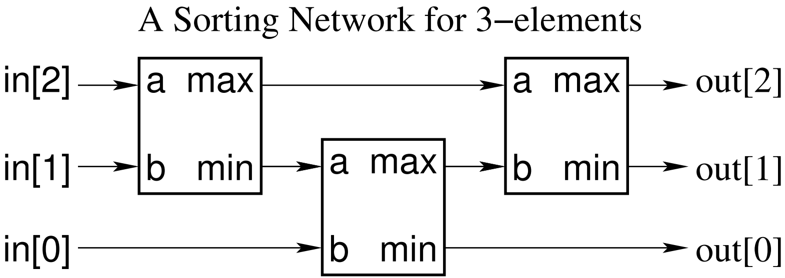
Total time:

Parallelizing Quicksort

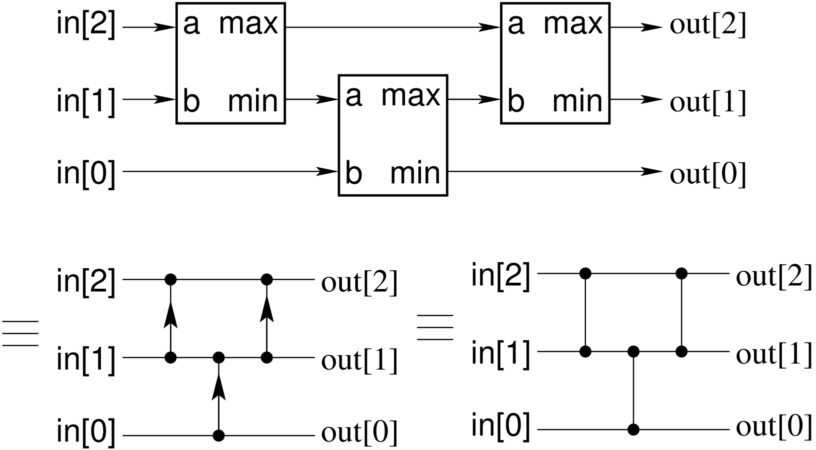
How would you write a parallel version of quicksort?

Sorting Networks



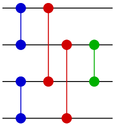


Sorting Networks – Drawing

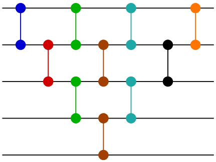


Sorting Networks – Examples

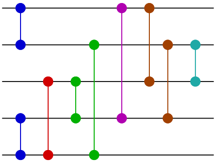
sort–4



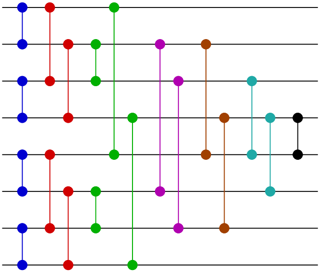
sort–5 (v1)



sort–5 (v2)



sort–8



Operations of the same color can be performed in parallel.

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

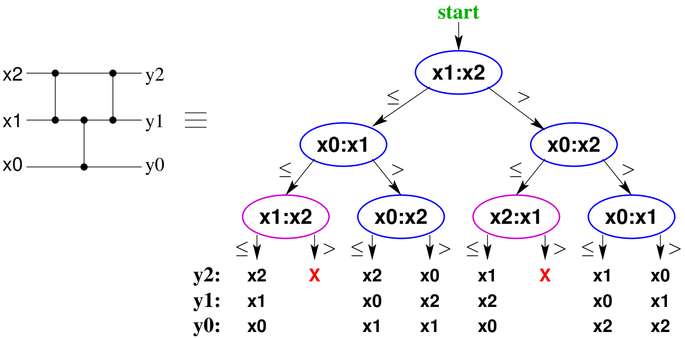
Sorting Networks: Definition

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.

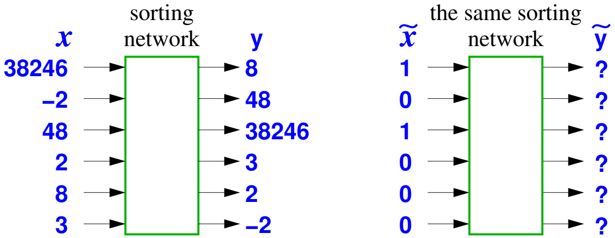
The 0-1 Principle

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.

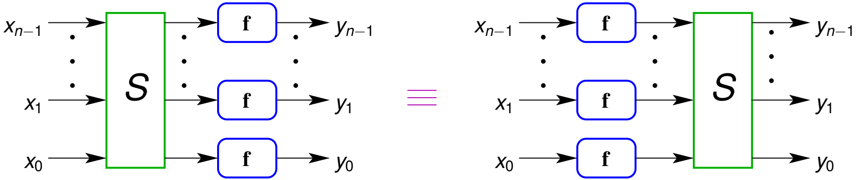
The 0-1 Principle: Proof Sketch

- 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 $\tilde{x}$.

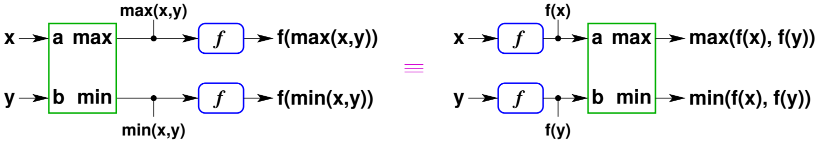
Monotonicity Lemma



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



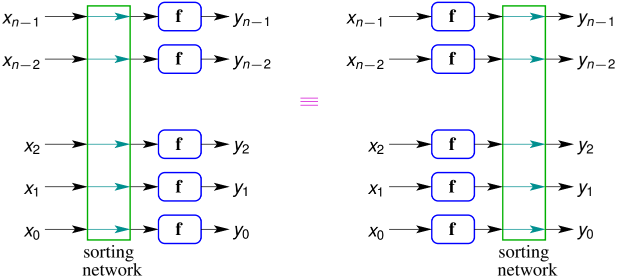
Compare-and-Swap commutes with monotonic functions.

- Case $x \leq y$:

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

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

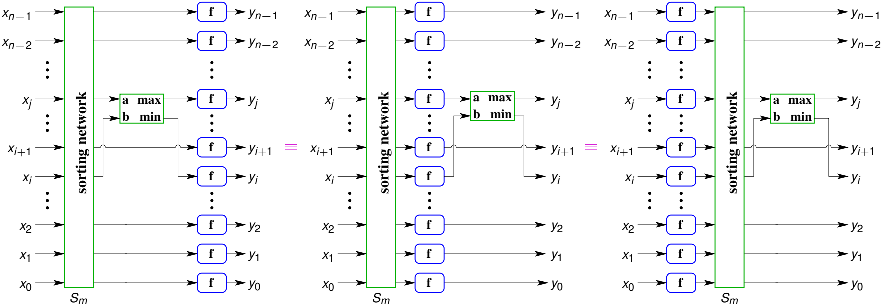
The monotonicity lemma – proof sketch



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 $0 \leq i < j < n$.
- 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 14).
- 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
$$\begin{aligned} f(x) &= 0, \quad \text{if } x < y_i \\ &= 1, \quad \text{if } x \geq y_i \\ \tilde{y} &= S(f(x)) \end{aligned}$$
- 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 5: Bitonic Sorting		
February 7: Implementing Bitonic Sorting		
February 9: Intro. to GPU computing or **"Ask Me Anything"**		
	Reading:	Kirk & Hwu – Chapter 2
February 13: Tuesday – Mark's office hours		
	Homework:	HW 3 earlybird (11:59pm).
		HW 4 goes out – midterm review, maybe some simple CUDA
February 14: GPUs & CUDA		
	Reading:	Kirk & Hwu – Chapter 3
	Homework:	HW 3 due (11:59pm).
February 16: GPus & CUDA		
February 19-23: break week		
February 28: midterm – see next slide		

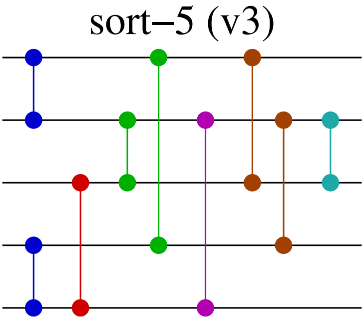
The Midterm:

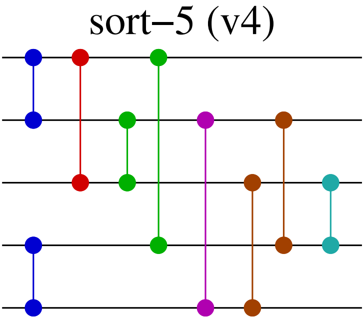
- February 28
- Open book, open notes, open anything on paper, calculators allowed
  - This is a “do you understand the concepts?” test, not a “did you memorize a bunch of trivia?” test.
  - I’ll aim for zero to minimal need for calculators, but won’t guarantee zero.
  - Previous midterms are on the 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.