Scan

Mark Greenstreet and Ian M. Mitchell

CpSc 418 – January 15, 2018

- What is Scan?
- Dependencies
- Implementing Scan

Scan: overview

- What is scan?
  - Given a list, X, with N elements, produde a list Y where the I^{th} element of Y is the sum of the first I elements of X, for $1 \leq I \leq N$.
  - Generalizes to any associative operator, just like reduce.
- Why scan?
  - It's useful.
  - It's our first "non-obvious" parallel algorithm – scan is an "aha!" for parallel computing.
  - It illustrates the importance of reasoning about dependencies.

map, foldl, and foldr

We've learned about higher order functions in Erlang:

- `map(Fun, List1) -> List2`
  - `length(List2)` = `length(List1)`
  - for all 1 ≤ `I` ≤ `length(List1)`:
    `lists:nth(I, List2)` = `Fun(lists:nth(I, List1))`
- `foldl(Fun, Acc0, List1) -> AccOut`
  - Combine the elements of `List1` in left-to-right order, i.e. first element of `List1` to the last element.
  - Start the accumulator what `Acc0`.
  - Return the final value of the accumulator.
- `foldr(Fun, Acc0, List1) -> AccOut`
  - Like `foldl` but accumulates the value in right-to-left order

`mapfold` and `scan`

- `mapfoldl(Fun, Acc0, List1) -> {List2, AccOut}`
  - `nth(I, List2)` is the result of folding the first codel elements of `List1` using `Fun`.
  - `AccOut` is the same as for `foldl(Fun, Acc0, List1)`.
- `scan`: a parallel function similar to `mapfoldl`.
  - If `Fun` is associative, we can do `mapfoldl` in parallel using a tree-pattern similar to reduce.
  - **Every** reduce problem has a corresponding scan version, and vice-versa.

Dependencies

Scan: if you're a theoretician

- Let `List2` be the list produced by `scan(Fun, Acc0, List1)`.
- Each element of `List2` can be computed using a reduce.
  - Element `I` has a reduce tree with `I-1` nodes.
  - Total number of tree nodes is $O(N^2)$ where $N =$ `length(List1)`.
  - Time is $O(\log N)$.
  - Time is polylog N, and number of processors is polynomial in N.
  - $\therefore$ scan is in NC
- NC is a class of problems that are highly parallelizable in theory.
  - If a problem is not in NC, it's probably not a good candidate for parallel computing.
  - If a problem is in NC, it's worth considering a parallel approach, but the algorithm that achieved polylog time is probably not practical.
  - There won't be any questions about NC on the homework or exams – for this class, NC is poetry.

Scan: Kogge-Stone

Re-use replicated result from the brute-force method.

Scan: Schwartz

Use a separate upward and downward pass.