Reduce

Mark Greenstreet

CpSc 418 – September 12, 2018

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/

Objectives

- Understand how reduce combines values using a tree.
- Describe the performance issues for reduce: trade-offs of time for computation and time for communication
- Describe 2 or 3 examples or reduce.

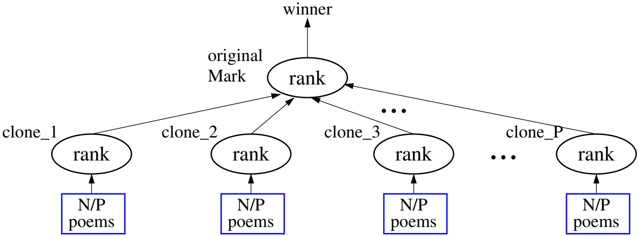
CPSC 418 Poetry Competition

- The competition:
  - Everyone writes a poem.
  - Everyone submits to poem to Mark (the contest judge).
  - Mark reads all of the poems, compares them, selects the best poem.
  - The winner receives an original manuscript of the complete poems of Li Bai, signed by the author.

Sequential Time for the Poetry Competition

- N students in the class.
- t_{rank} to read and rank two poems.
- Total time $(N - 1)t_{rank}$.
- Works fine until N becomes so large that we can't judge all the poems in a reasonable amount of time.

Parallel poetry: the procedure



- Clone P copies of Mark.
- Each Mark-clone reads and ranks N/P poems and sends the best poem to the original Mark.
- The original Mark receives P candidates for the best poem, and selects the best one.
- The winner receives the prize.

Parallel poetry: time

- The time for each of the P clones to select the best poem out of N/P:

  ____

- The time from when all P clones start until all P clones finish:

  ____

- The time for the original Mark to rank the P finalists:

  ____

- Simplify this to get ________________________.

- $SpeedUp = \frac{T_{\mathrm{seq}}}{T_{\mathrm{par}}} =$ ________________________.

Bureaucratic Overhead

- To satisfy UBC privacy policies, the messages between the Mark-clones and the original mark must be sent in special envelopes.
- There's lots of special procedure for handling these envelopes, takes time λ to send or receive a message.
- The original Mark receives P messages from the P clones. This takes time λP.

- The total time is now: ____________________.
- $\mathit{SpeedUp} = \frac{T_{\text{seq}}}{T_{\text{par}}} =$ ____________________.

Can we do better?

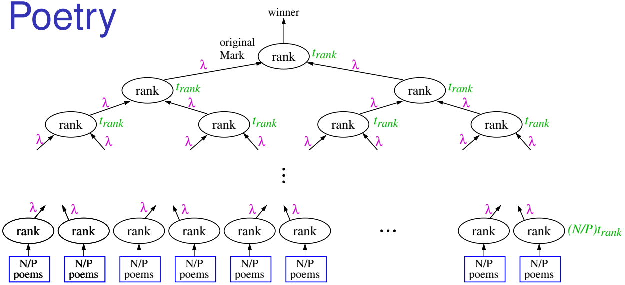
Revenge of the Clones

- While Mark is working through the pile of envelopes, some of the clones realize that they could pair up and combine their results.
  - This costs λ time, the clones have to follow the rules as well.
  - The original Mark ends up with half as many envelopes to handle.

- What is the total time? ____________________.

- What is the speed-up? ____________________.

Up a Tree with Poetry



- The optimization on the previous slide worked great.
  - We're computer scientists, let's apply the optimization recursively.
  - Viewed from another angle, this is an example of divide-and-conquer.
- Combine the results in a tree.
  - How many levels in the tree?
  - How much time at each level?
- What is the total time? ____________________.
- What is the speed-up? ____________________.

Is there more to life than poetry?

- Find the largest element in a list.
- Find the sum of the elements in a list.
- Count the number of 3s in a list.
- What to these all have in common?
- We'll look at more examples on Friday

The Reduce Pattern

- We have a problem that takes $T_{\text{seq}}(N)$ sequential time, where N is the "size" of the problem instance.
- We can divide this into P tasks with "perfect" speed-up:
  - Each task takes time $T_{\text{seq}}(N)/P$ time.
  - Combining the results takes $\lceil \log_2(P) \rceil \lambda$ time.
- $\mathit{SpeedUp} = \frac{T_{\text{seq}}(N)}{T_{\text{seq}}(N)/P + \lceil \log_2(P) \rceil \lambda}$
- What happens to *SpeedUp* as P goes large (for fixed N)?
- What happens to *SpeedUp* as N goes large (for fixed P)?
  - Assume $T(N)$ grows faster than $\log N$.

When can we use reduce?

- We have N values, V_1, V_2, V_3, $\ldots$ V_N.
- We want to compute them with some operator, $\circ$. I.e. we want:
  $$Total_{seq} = V_1 \circ V_2 \circ V_3 \circ \cdots \circ V_N$$
- I'll assume the sequential computation is left-to-right, so
  $$Total_{seq} = (\cdots((V_1 \circ V_2) \circ V_3) \circ \cdots) \circ V_N$$
- Using reduce, we do these operations in clusters of N/P, and then combine the results:

$$\begin{array}{l} Total_{reduce} \hfill = \\ \quad ((\cdots((V_1 \circ V_2) \circ V_3) \circ \cdots) \circ V_{N/P}) \\ \circ\, ((\cdots((V_{(N/P)+1} \circ V_{(N/P)+2}) \circ V_{(N/P)+3}) \circ \cdots) \circ V_{2N/P}) \\ \circ\, ((\cdots((V_{(2N/P)+1} \circ V_{(2N/P)+2}) \circ V_{(2N/P)+3}) \circ \cdots) \circ V_{3N/P}) \\ \ldots \\ \circ\, ((\cdots((V_{((P-1)N/P)+1} \circ V_{((P-1)N/P)+2}) \circ V_{((P-1)N/P)+3}) \circ \cdots) \circ V_N) \end{array}$$

- A sufficient condition is: $(X \circ Y) \circ Z = X \circ (Y \circ Z)$.
- What term describes an operator with this property?

Reduce is a Higher Order Function

- `reduce(WorkerTree, Leaf, Combine) -> Result`
  - `WorkerTree`: a collection of worker processes, organized as a tree.
    - "Organized" means each process knows who its parent and children processes are so it can send and receive the messages needed for reduce.
  - `Leaf`: a function – what to do at each leaf to produce a result to combine using reduce.
    - In our poetry example, `Leaf` finds the best poem in its subset of all poems submitted to the contest.
  - `Combine(Left, Right)`: a function – the operation to be applied at each node.
  - `Result`: the value computed at the root of the tree.

wtree:reduce

- Reduce as implemented in the CpSc 418 Erlang library.
- Example, `count3s` using `wtree:reduce`

```
count3s(WorkerTree, Key) ->
  wtree:reduce(WorkerTree,
    fun(ProcState) -> count3s_leaf(ProcState, Key) end,
    fun(Left, Right) -> count3s_combine(Left, Right) end
  ).

count3s_leaf(ProcState, Key) ->
  MyList = workers:get(ProcState, Key),
  length([E || E <- MyList, E =:= 3]).
count3s_combine(Left, Right) -> Left+Right.
```

- The code is available at reduce_intro.erl.

`count3s` notes (1/2)

- `WorkerTree`: a tree of workers.
  - To create a tree of `Nworker` workers, call `wtree:create(Nworkers)`.
  - When your done, you can clean up by calling `wtree:reap(WorkerTree)`.
- `ProcState`: Erlang is functional, how do workers remember anything?
  - Each worker executes a tail-recursive “get-a-task” function that is called with `ProcState` as a parameter.
  - This function does a `receive` to get a new `Task` (blocking if no task is ready).
    - ★ `NewProcState` = Task(ProcState).
    - ★ `Task` is called with `ProcState` as a parameter.
    - ★ `Task` returns an updated process state, `NewProcState`.
  - The worker recursively calls its get-a-task function, with `NewProcState` as the process state parameter.
  - `ProcState` is an Erlang key-list (i.e. a dictionary).

`count3s` notes (2/2)

- `workers:get(ProcState, Key)`: fetch the value for `Key` from `ProcState`. Called by a worker process.
- `NewProcState = workers:put(ProcState, Key, Value)`: create a new process-state where `Key` maps to `Value`, and all other mappings are the same as in `ProcState`. Called by a worker process.
- `workers:update(Workers, Key, ValueList)`: Called by the top-level process. `ValueList` should be a list with one element per worker. The elements of `ValueList` are stored in the `ProcState` of the corresponding workers with the key `Key`.
- `workers:retrieve(Workers, Key) -> ValueList`: fetch the value associate with `Key` for each worker in `Workers`.
  - Frequently, we have a list distributed across the workers. In this case, `workers:retrieve(Workers, Key)` returns a list of the form `[List1, List2, ...ListP]` where `P` is the number of workers, and `ListI` is the segement of the list held by worker `I`.
  - To merge these segements into one list:

    `lists:append(workers:retrieve(Workers, Key))`

Testing `count3s`

```
count3s_test(N_workers, N_values)
  when is_integer(N_workers), N_workers >= 0,
       is_integer(N_values), N_values >= 0 ->
  WorkerTree = wtree:create(N_workers),
  % create a random list of N_values integers chosen in [1, 10], distribute
  % it across the workers of WorkerTree and associate it with the key 'data'.
  workers:rlist(WorkerTree, N_values, 10, data),
  Par3s = count3s(WorkerTree, data),
  Data = lists:append(workers:retrieve(WorkerTree, data)),
  Seq3s = length([E || E <- Data, E == 3]),

  case Par3s =:= Seq3s of
    true ->
      io:format("passed:  N_values = ~b, Par3s = ~b~n",
                [N_values, Par3s]),
      ok;
    false ->
      io:format("failed:  N_values = ~b, Par3s = ~b, Seq3s = ~b~n",
                [N_values, Par3s, Seq3s]),
      fail
  end.
```

Preview

September 14: Reduce – The Pattern
Reading: Lin & Snyder, chapter 5, pp. 112–125

September 17: Scan
Homework: Homework 1 deadline for early-bird bonus (11:59pm)
Homework 2 goes out (due Oct. 1) – Reduce and Scan

September 19: Reduce & Scan Examples
Homework: Homework 1 due **11:59pm**

September 21 – 26: Parallel Architecture

Sept. 28 – Oct. 5: Performance Analysis

October 8 – 15: Sorting

October 17: Intro. to CUDA

October 19: Midterm review

October 22: Midterm

Oct. 24 – Nov. 30: Data Parallel Computing, GPUs, and CUDA