Reduce – The Pattern

Mark Greenstreet

CpSc 418 – September 14, 2018

- Surviving this Course
- The Reduce Pattern
- Examples

Survival: what I learned from piazza

From Piazza in a previous term: "...HW1 Q3 took me 3–4 hours".

- Yikes! If one question takes you 3–4 hours, then I'll guess 12 or more hours for the assignment.
- Add lectures, reading, and a PIKA, and we're looking at 20 hours for the week.
- If you're taking five classes, that's 100 hours/week – no time for eating, sleeping, brushing your teeth, or parties.
- Not sustainable, Brian fails.

How to survive

- Piazza lets me know that there **might** be a problem, but it doesn't let me know if there **is** a problem.
  - Is everyone drowning in the workload?
  - Are there just a few students who need some help to catch-up?
  - Are there just a few students who will complain about the workload no matter how easy it is?
- The solution: office hours and tutorial
  - You outnumber the instructors and TAs.
  - Use this to your advantage.
  - If it is taking you 3-4 hours to solve one HW problem, you can save time by going to office hours or tutorial and asking questions.
- This solves the instructors dilemma
  - If 80% of the class is overwhelmed, I'll have 20–30 or more students at office hours. I'll find out where you're stuck, and I'll adjust the course to match.
  - If a few of you need a bit of help to get going with Erlang, parallel programming, timing measurements, or other stuff, we'll get it taken care of.
  - Either way, if **you** are finding the workload too high, go to office hours and/or tutorials.

Objectives

- Understand the reduce pattern.
- Solve simple problems using reduce.
- Understand how to write `Combine` functions.

Reduce Review

- The basic idea:
  - We have a task that can be divided over P processes.
  - We need to combine the results from the sub-tasks to get the main result.
  - This involved communication between processes.
    - Communication is slow. We write λ for the communication time.
    - If each worker sends its result to the master process, this takes λP time.
    - If the workers combine their results using a tree, it takes $\lambda \log_2 P$ time.
  - Reduce reduces the communication overhead.
    - Parallel approaches can be used efficiently for smaller problems using reduce than using the brute-force approach.
    - If N is the problem size, we can make effective use of a bigger P for a smaller N.

Beyond Poetry

Some examples we will consider:

- Finding the largest element in a list or array distributed across P processes.
- Finding the sum of the elements in a list or array distributed across P processes.
- Finding the average of the elements in a list or array distributed across P processes.
- Removing adjacent duplicates (see PIKA2).

Associative (and Commutative) Operators

- An operation is associative if we can re-arrange the parentheses while preserving the left-to-right order of the operands and get the same result.
  - Addition is associative if you're a mathematician.
  - Addition is almost associative if you're working with floating point numbers.
  - Addition is associative if you're working with integers.
  - Similar remarks for multiplication, finding the maximum, and many other operations.
- What about commutative?
  - We're at a university, so "associative and commutative" just rolls off the tongue because it makes us sound so mathematical and therefore scholarly.
  - An operator, $\circ$ is commutative if $A \circ B = B \circ A$ for all A and B.
  - Commutative is nice because:
    - ⋆ We can re-order the operations however we like.
    - ⋆ We don't need to preserve left-to-right order.

Do we care about commutativity?

- **No:** while being able to re-order more may seem like a good idea, e.g., use results as they become available, in practice this often isn't worth it.
  - Figuring out which results are available requires synchronization.
  - This incurs the λ cost for global actions.
- **Maybe:** if the operator is associative but not commutative, then we care about the left-to-right order of the data.
  - The summaries that we pass through combine will say something about the left-to-right order.
  - Often these summaries have the form of:

    `{LeftSummary, OverallSummary, RightSummary}`
  - Reduce tends to be **simpler** to implement when the function is associative and commutative.
- **Yes:** if the underlying hardware shuffles the data ordering (we'll see this in CUDA), then we are much happier if the operation for the reduce is commutative.

Count 3s: the code

- We kind of rushed it on Wednesday. Let's go through the details

```
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.

Count 3s: Let's try it

```
bash$ erl
Erlang/OTP 19 blah blah blah ...
Eshell V8.3 (abort with ∧G)
1> c(reduce_intro).
{ok,reduce_intro}
2> W = wtree:create(8).
** exception error:  undefined function wtree:create/1
% We need to tell Erlang the path to the course library.
% I'll show this for running Erlang on the ugrad.cs.ubc.ca machines.
3> code:add_path("/home/c/cs418/public_html/resources/erl").
true
4> W = wtree:create(8).
% wtree:create returns a list of pids
[<0.71.0>,<0.72.0>,<0.73.0>,<0.74.0>,<0.75.0>,<0.76.0>,<0.77.0>,<(
% Create a list of 100 random integers in [1,10]. The list is
% distributed over the workers of W and associated with the key data.
5> workers:rlist(W, 100, 10, data).
ok
6> reduce_intro:count3s(W, data).
4
% Let's check
```

The course library

- If you are on a `ugrad.cs.ubc.ca` linux machine:
  - From the Erlang shell:

    ```
    code:add_path("/home/c/cs418/public_html/resources
    ```
  - Or, add the following to your `~/.bashrc`.

    ```
    function erl {
      /usr/bin/erl erl -eval 'code:add_path("/home/c/cs418/p
    ```

    and you will have the path set-up every time you run Erlang.
- Do try this at home: download the library from:

  http://www.ugrad.cs.ubc.ca/~cs418/resources/erl/

  You will need to compile the modules. I should add a `Makefile` to the archive that does that for you.
- The library comes with documentation.

  http://www.ugrad.cs.ubc.ca/~cs418/resources/erl/

Count 3s: Let's time it (1 of 3)

We need a sequential version. See reduce_examples.erl.

```
count3s(List) -> count3s_tr(List, 0).
count3s_tr([3 | Tl], Acc) -> count3s_tr(Tl, Acc+1);
count3s_tr([_ | Tl], Acc) -> count3s_tr(Tl, Acc);
count3s_tr([], Acc) -> Acc.  count3s_time(N_values) -> % time the sec
  Data = misc:rlist(N_values, 10),
  time_it:t(fun() -> count3s(Data) end).
```

Count 3s: Let's time it (2 of 3)

We need timing measurement function:

```
count3s_time(seq, N_values) -> count3s_time(N_values);
count3s_time(N_workers, N_values)
  when is_integer(N_workers), N_workers >= 0,
       is_integer(N_values), N_values >= 0 ->
  % time the parallel version
  WorkerTree = wtree:create(N_workers),
  workers:rlist(WorkerTree, N_values, 10, data),
  workers:retrieve(WorkerTree, fun(_) -> ok end), % make sure that rlis
  T = time_it:t(fun() -> count3s(WorkerTree, data) end),
  wtree:reap(WorkerTree),
  T.
```

Count 3s: Let's time it (3 of 3)

`N_values = 1000000`. Running on `thetis.ugrad.cs.ubc.ca`.

N_workers	Time (seconds)	SpeedUp
seq	6.54e-3	1
2	3.83e-3	1.7
4	2.01e-3	3.25
8	1.40e-3	4.69
16	7.95e-4	8.23
32	5.27e-4	12.42
64	4.62-4	14.17
128	4.28-4	15.29
256	4.28-4	12.41

Demystifying `ProcState`

Generalizing Reduce: `max`