Reduce – The Pattern

Mark Greenstreet and Ian M. Mitchell

CpSc 418 – January 12, 2018

- Surviving this Course
- The Reduce Pattern
- Examples

Survival: what I learned from piazza

From Piazza: "...HW1 Q3 took me 3–4 hours".

- Yikes! If Q3 took you 3–4 hours, then I'll guess 1-2 hours for each of Q1 and Q2, and 6 hours for Q4.
- Thats 12–13 hours for the HW.
- 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.

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.
    - ⋆ 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).

How reduce works

Using the sum example:

- In C/Java/Python if we write

  ```
  A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z
  ```

  The operations are performed left to right:

  ```
  ((((((((((((((((((((((((((((A+B)+C)+D)+E)+F)+G)+H)+I)+J)+
  ```

- If we do the same with reduce, we have each process do a sub-sequence of the original arguments:

  ```
  (([A+B+C+D+E+F] + [G+H+I+J+K+L+M]) + ([N+O+P+Q+R+S] +
  ```

- We have re-ordered the additions.
- Why is this OK?

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 is 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}`
- **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.

Examples

From the PIKA.