Midterm Review

Mark Greenstreet and Ian M. Mitchell

CpSc 418 – February 16, 2018

- Erlang: functional and message passing
- Reduce and Scan
- Parallel architecture: shared memory and message passing
- Parallel Performance: Speed-up, performance loss, Amdahl, Gustafson, dependencies, energy, PRAM, CTA, logP.
- Parallel sorting: sorting networks, the 0-1 principle, bitonic sort
- Data Parallel and GPUs: not on the midterm

Erlang is functional

- Implement simple recursive functions, e.g. fast_flatten from pika4.erl.
- Understand difference between head- and tail-recursive
- Avoid common ways to make inefficient code such as
  - using `++` to append one item at a time to a list.
  - using `length` in a guard
- Higher order functions: map, foldl, foldr, mapfoldl, mapfoldr, list comprehensions
  - able to use them for common patterns.

Erlang supports message passing

- Know how to spawn a process, send messages, receive messages.
- Message ordering constraints: the triangle inequality, nothing else.
- Receive uses pattern matching
  - tagging messages is good.
  - example: how could `reduce` fail if the implementation didn't tag messages?
- timeouts – use carefully and sparingly
- Example: the lock problem from HW2.

Reduce

- Reduce is a parallel version of foldl.
- The reduction operation needs to be
  - associative?
  - commutative?
  - reflexive?
  - transitive?
  - what do those “math words” mean?
- Often, we need to find an intermediate data structure to pass values from Leaf to Combine, between levels of Combine, and from Combine to Root.
  - Look at examples from HW2, lecture slides, and Lin & Snyder, Chapter 5 (handed out in class).
- If we can combine two value in unit time, how long does it take to combine N items using P processors, assuming that messages take time λ (total for a single send and the matching receive)?

Scan

- Scan is a parallel version of mapfoldl.
  - For every reduce problem, there is a corresponding scan version.
- Implementation involves a pass **down** the tree.
- But, we abstact/hide those details inside the `wtree:scan` function
  - What does `Leaf1` need to compute?
  - What does `Combine` need to compute?
  - What does `Leaf2` need to compute?
    - What is the `AccIn` parameter to `wtree:scan`?
    - What is the `AccIn` parameter to `Leaf2`?
- But, we abstact/hide those details inside the `wtree:scan` function
- Look at examples from HW2, lecture slides, and Lin & Snyder, Chapter 5 (handed out in class).

Shared Memory Architecture

- Caches
- The MESI Protocol
- Understand that MESI allows many caches to share a read-only copy of a cache line and guarantees that they all have the value of the most recent write.
- At most one cache can have a writeable copy.
- Understand how MESI combines write-through with write-back to achieve this.
- Able to define “sequential consistency”.
- Able to trace what happens for a short sequence of memory operations.
- Example see pika4.

Message Passing Architectures

Performance

Sorting

Proofs

Happy New Year