Lecture 30 - High-Performance Languages

ECE 459: Programming for Performance

Jon Eyolfson

University of Waterloo

March 30, 2012

Assignment 4

- There's only one OpenCL platform now, so you shouldn't have to worry
- Here are the following expected results:

```
jon@ece459-1 assignment-04 % time ./nbody-seq > nbody-1.out
./nbody-seq > nbody-1.out  11.08s user 0.00s system
                            99% cpu 11.088 total
jon@ece459-1 assignment-04 % time ./nbody > nbody-2.out
./nbody > nbody-2.out  0.16s user 0.16s system
                   99% cpu 0.318 total

jon@ece459-1 assignment-04 % diff -u nbody-1.out nbody-2.out
jon@ece459-1 assignment-04 %
```

Hint: let local do it's thing

Introduction

- DARPA began a supercomputing challenge in 2002
- Purpose to create multi petaflop systems (floating point operations)
- Three notable programming language proposals
  - X10 (IBM) [Looks like Java]
  - Chapel (Cray) [Looks like Fortran/math]
  - Fortress (Sun/Oracle)

Machine Model

- We've used multicore machines and talked about clusters (MPI, MapReduce) these languages are targeted somewhere in the middle
- They have thousands of cores and massive memory bandwidth
- They used a Partitioned Global Address Space (PGAS) memory model
  - Each process has a view of the global memory
  - The memory is distributed across the nodes, however the processors explicitly know what global memory is local

Parallelism

- These languages require you to specify the parallelism structure
- Fortress evaluates loops and arguments in parallel by default
- Others use an explicit construct like `forall` and `async`
- Fortress divides memory into locations, which belong to regions (which are in a hierachy, the closer the better communication)
- Called different names: places (X11) and locacles (Chapel)
- These langauges make it easier to control the locality of data structures and have distributed (fast) data

X10 Example

```
import x10.io.Console;
import x10.util.Random;

class MontyPi {
  public static def main(args:Array[String](1)) {
    val N = Int.parse(args(0));
    val result=GlobalRef[Cell[Double]](new Cell[Double](0));
    finish for (p in Place.places()) at (p) async {
      val r = new Random();
      var myResult:Double = 0;
      for (1..(N/Place.MAX_PLACES)) {
        val x = r.nextDouble();
        val y = r.nextDouble();
        if (x*x + y*y <= 1) myResult++;
      }
      val ans = myResult;
      at (result) atomic result()() += ans;
    }
    val pi = 4*(result()())/N;
  }
}
```

X10 Example Explained

- It's the same problem as Assignment 1, but gives a distrubuted solution
- We could replace `for (p in Place.places())` with `for (1..P)` (where P is a number) for a parallel solution (you would also remove `GlobalRef`)
- `async` creates a new child activity which executes the statements
- `finish` waits for all the child `async`s to finish
- `at` performs the statement at the place specified, in this example the processor that is holding the result increments its value

Summary

- For supercomputers there are three notable languages: X10, Fortress and Chapel
- The use the Partitioned Global Adress Space memory model, which allows distrubited memory with explicit locality
- The parallel programming aspect to the languages are very similar to everything else we've seen in the course
- Let's do a quick course summary

Bandwidth and Latency

- These are the measures of performance we saw at the start of the course
- Improving Bandwidth (tasks per unit time)
  - Parallelism
- Improving Latency (time per task)
  - Approximation algorithms
  - Saving work on algorithms
  - Better utilization of underlying hardware (brief review)
  - Using better data structures
- All improvements require profiling to make sure they're actually improvements!

Limits to Parallelization

- There's always some serial part which is going to limit the amount of parallelization
- Amdahl's Law provides an estimation for a fixed problem size
- Gustafson's Law provides an estimation for a variable problem size

Dependencies

- The main barrier for parallelization

- Different types of dependencies
  - RAW - read after write
  - WAR - write after read
  - WAW - wrtie after write

- WAR and WAW can be broken by renaming/copying

- Speculation could also be used

Race Conditions and Synchronization

- Recall a race condition is when the same memory location could be accessed at the same time, which one of the accesses being a write

- We can protect against them with sychronization primitives
  - Locks
  - Semaphores
  - Barriers

Parallel Programming

- We focused on thread programming using:
  - Pthreads
  - OpenMP
- Looked at detached threads, OpenMP sections, for loops, etc.
- Automatic parallelization
- Had to be aware of the memory model (different then locking)
  - Memory fences

Compiler Optimizations

- Ran through a list of what optimizations are available and what they do
- The simpler your code the better the optimizations the compiler can make, and may be very complex
- Expected values of if statements, loop unrolling, SIMD instructions
- The compiler is good at figuring out what to inline as well (with profiler guided optimizations)
- Steps to PGO:
  - Compile with `-fprofile-generate`
  - Run your program with your test input
  - Compile with `-fprofile-use`

GPU Programming

- We went over how to use OpenCL
- How to convert parallelizable code to being parallelized by the GPU
- This is the focus of Assignment 4, and should show a huge improvement depending on the problem

Distributed Systems

- These are massive computing systems running in a cluster (or on the cloud)

- There's approaches with shared-memory and messaging passing

- We looked at two major
  - MPI
  - MapReduce

- Although we didn't use them in an Assignment, we got a high-level idea of their operation

Final Word

- Thanks! Hopefully it was enjoyable

- Although we didn't use them in an Assignment, we got a high-level idea of their operation

Monday's Plan

- Monday will be a review session were I'll be here to answer any questions

- As always, you can also e-mail me or a TA to set up office hours