ECE459: Programming for Performance Winter 2011

Lecture 23 — March 31, 2011

Patrick Lam *version 1*

High-performance Languages

In recent years, there have been three notable programming language proposals for high-performance computing: X10[1], Project Fortress[2], and Chapel[3]. I'll start with the origins and worldview of these projects, and then discuss specific features. You can find a good overview in [LY07].

Current Status

You can download sample implementations of these languages, and they seem to be actively-developed (for varying definitions of active). Fortress has the least web activity, but still has recent commits; there was a bit of excitement at Sun/Oracle in the not-too-distant past. Chapel and X10 seem to have more activity (i.e. releases in the last 6 months).

Context. The United States Defense Advanced Research Projects Agency (DARPA) started the "High Productivity Computing Systems"[4] project in 2002. This project includes both hardware and languages for high-performance computing; in this lecture, we focus on the languages and touch on the hardware that they were built to run on. It looks like HPCS hardware is due to go online in 2011[5][6].

Machine Model. We've talked about both multicore machines and clusters. The target HPCS machine is somewhere in the middle; it will include hundreds of thousands of cores (hundreds of thousands) and massive memory bandwidth (petabytes/second). The memory model is a variation of PGAS, or Partitioned Global Address Space, which is reminiscent of the GPU memory model: some memory is local, while other memory is shared. It is somewhere between the MPI message-passing model and the threading shared-memory model.

Unlike in MPI, in these languages you write one program with a global view on the data, and the compiler figures out (using your hints) how to distribute the program across nodes. (Recall that in MPI you had to write specific, and different, code for the server and client nodes.)

Base Languages. X10 looks like a Java variant. Fortress takes some inspiration from Fortran, but is meant to look like mathematical notation (possibly rendered to math). Chapel also doesn't

[1] `http://x10.codehaus.org`

[2] `http://java.net/projects/projectfortress`, `http://projectfortress.sun.com/Projects/Community`

[3] `http://chapel.cray.com`

[4] `http://www.hpcwire.com/features/17883329.html`

[5] `http://www.ncsa.illinois.edu/BlueWaters/`

[6] `http://www.genomeweb.com/blog/crays-cascade-coming-soon`

aim to look like any particular existing language. All of these language support object-oriented features, like classes.

Parallelism. All three of these languages require programmers to specify the parallelism structure of their code (in various ways). Fortress evaluates loops and arguments in parallel by default, or implicitly, (if it chooses to), while the other languages have optional constructs like `forall` and `async` to specify parallelism.

Visible Memory Model. As we've said before, all of these languages expose a single global memory; it may perhaps be costly to access some parts of the memory, but it's still transparent. Fortress programs divide the memory into locations, which belong to regions. Regions are classified into a hierarchy, and nearby regions would tend to have better communication. X10 has places instead of regions. Places run code and contain storage. Chapel uses locales.

As a developer, you get to control locality of your data structures in these languages (by allocating at the same place or at different places, for instance). They make it easy to write distributed data structures.

Software Transactions. These were quite popular when these languages were being developed. You get to specify an `atomic` block, and the compiler has to make sure that the block either runs to completion or aborts (and retries).

Bonus Link

I couldn't figure out where to put this, but I recommend that you look at it.

`http://x10.codehaus.org/Performance+Tuning+an+X10+Application`

References

[LY07] Ewing Lusk and Katherine Yelick. Languages for high-productivity computing: The DARPA HPCS language project. *Parallel Processing Letters*, 17(1):89–102, March 2007.