Evaluating the Productivity of Next Generation Parallel Programming Languages via Scientific Applications

By: Jessica Vallejo
Adviser: Dr. Tim Stitt
Background
Petascale Computing

• 1 petaflop = \(10^{15}\) floating point operations per second

• Abundance of hardware advancements
  – Jaguar supercomputer from Oak Ridge Labs
    • Upgrade from 2.3 petaflops to 3.3 petaflops in 2012

• Less advancement on software side
  – C, Fortran, MPI used traditionally
DARPA

- **2001:** HPCS project started
  - End goal: develop new generation high productivity computing systems with 10 times the productivity of traditional systems by 2010

- **2012:** systems created include
  - Chapel by Cray Supercomputing
  - UPC by University of Berkeley
  - X10 by IBM
  - CoArray Fortran
Evaluation Challenges

• Recode entire community-standard applications into new languages
  – Thousand to millions of lines of source code
• Test languages using simple benchmarks
  – Matrix Matrix Multiply
Mini-Applications Alternative

- Mini-Applications (miniapps for short)
  - Complete, standalone, proxy applications that contain the performance of community standard applications
    - Hundreds of lines of source code instead of thousands
Sandia National Labs

• Mantevo project
  – Develop tools to assist in design of high performance computers and applications
  – Performance of large applications is often dominated by a subset of lines
  – A variety of high performance miniapps are available as open-source
REU Project
Evaluation of Next Gen Languages

- How? Rewrite Mantevo minapps in Chapel and UPC
- MiniFE
  - Available in C++, OpenMP, CUDA it will be easier to do comparison testing.
- MiniMD
  - Miniapp for molecular dynamics application LAMMPS.
Chapel

- Identifying parallelism and locality is user’s job, not compiler’s
- No pointers and limited opportunities for aliasing
- Not an extension of an existing language
- Interactive Parallel language
Chapel Inspiration

- C, Modula: basic syntax
- CRAY MTA C/Fortran: task parallelism, synchronization
- ZPL, HPF: data parallelism, index sets, distributed arrays
- CLU (see also Ruby, Python, C#): iterators
- Scala (see also ML, Matlab, Perl, Python, C#): type inference
- Java, C#: OOP, type safety
- C++: generic programming/templates (but with a different syntax)
Tradeoffs

• Chapel allows for condensed programming, making programmability easier.

• Chapel is slow at creating executables from source code at this time.
  – Similar to how python provides ease of programmability but is slower in execution compared to C.
Goals for Project

• Provide valuable feedback to Chapel and UPC development teams
• Provide analysis of the progress and potential of these next-general parallel programming languages