Review of

Languages for High-Productivity Computing: The DARPA HPCS Language Project by Ewing Lusk and Katherine Yelick

Michelle Strout

September 2, 2010

What problem does the paper address?

Big picture problem

- Developing software that takes advantage of highly parallel machines is an overly difficult and error-prone process.
- New languages face adoption barriers.

Why the problem is hard

- Architectural evolution
- Highly portable and efficient programming models exist and are strongly entrenched, even though they have programming productivity issues.

Specific Problem

- How should we evaluate such languages?
- How could the three HPCS languages: X10, Chapel, and Fortress, be merged into a single new language.

Why should we care?

A significant amount of time, effort, and money has already been put into applications that use the current programming models.

– Full rewrites are not practical, so incremental approaches are necessary.
– Realizing the value of previous programming models is important when developing new programming models.

A significant effort and resources are being put into the development of new parallel programming languages.

– We need some way of evaluating these new languages.
– We need some inkling of how they might eventually be adopted.
– Will it even be possible to merge/evolve three languages into a new one?

What is the approach used to solve this problem?

Partial survey of existing parallel programming models

- MPI+Fortran, message passing interface
- PGAS languages: UPC (based on C), CAF (based on Fortran), Titanium (based on Java)

Discussion of features that parallel programmers want and expect

- Portability, being able to compile and run on multiple architectures
- Performance, that is the point of parallel after all
- Performance transparency, the programmer wants to have control over the performance details
  - Sub-set collectives and synchronization, which suggests an explicit parallel model
  - User-defined distributions

A qualitative evaluation of current status (2007) of the HPCS effort

- The X10, Chapel, and Fortress efforts are diverse.
- They are making good progress and not ready to merge yet.

How does the paper support the conclusions it reaches?

Some of their conclusions

- "Portability, performance, and incrementality seem to have mattered more in the recent past than elegance of language design, power of expression, or even ease of use, at least when it comes to programming large scientific applications"
- "… portability, completeness, support for modularity, and at least some degree of performance transparency"
- Standardization of a run-time interface and a new language, standardization effort should follow the MPI standardization model.

Conclusions that I make

- They indicate that subset collectives and synchronization have often been requested. MPI-2 and the HPCS languages have these, so I think some form of these concepts are here to stay.
- Programmers want a straight-forward execution and programming model, but they also want performance. Sometimes these goals conflict.

Future Research Questions

How can a programming model provide locality information to the compiler and run-time system?

Would the creation of dynamic memory locales be useful?

– Reminds me of data-structure specific cache regions work out of UT Austin.

Can we develop a standardized runtime?

What if we could semi-automatically convert code between different programming models? Like refactoring tools.

How does the HPCS effort relate to model driven environments?

Critique

Background of authors

- Authors have made significant contributions to MPI and PGAS programming models.
- Given their background they give the new HPCS languages a fair description.
- No mention of OpenMP seems odd due to its prevalence.

Evaluating the paper as a survey

- They do NOT give feature-by-feature descriptions of each programming model. That is a good thing.
- The focus is the features that programmers expect and how those are met by current languages.
- Their evaluation of the status and future of the HPCS languages is vague.

What I learned from the paper

- Fortress has hierarchical regions
- Looked-up MPI-2 abstractions: put, get, accumulate, MPI_comm_spawn, connect, join, MPI_File_set_view, open, read, write, close

Relation to CS653

Shares some aspects of some of the main parallel programming models in use today.

Provides some criteria for evaluation of parallel programming models

Related possible projects

- The Programming Model DataBase PMDB project is attempting to provide concrete examples of language features implementing important parallel patterns so as to evaluate the programmer control and tangling aspects of programming models.
  - Programmer control is related to “performance transparency”.
  - Tangling is related to “support for modularity
  - Students could evaluate any of these programming models.