Research Statement

Amir Kamil

January 31, 2015

My research interests encompass programming models and tools for parallel computing, particularly in the segment of high-performance computing (HPC). Fundamental challenges in this area include reasoning about the sequence of operations in a parallel program, managing distribution and locality of data and execution, and optimizing for the varying communication costs across a distributed machine. My research has involved program analysis to reason about these issues in a compiler or runtime, designing language constructs to make it easier for programmers to express computation on hierarchical machines, and developing libraries to introduce new parallel concepts into existing languages.

Program Analysis

My work in program analysis addresses parallel programs written in the widespread Single Program, Multiple Data (SPMD) style of execution, where a fixed set of threads runs the same program. In addition, I have focused on programs that use the Partitioned Global Address Space (PGAS) memory model, which provides the abstraction of shared memory across a distributed machine. This work includes concurrency and pointer analyses of distributed programs, with applications to memory-consistency models and locality and synchronization optimizations, and runtime analysis of the synchronization in a parallel program.

I developed a new algorithm for computing concurrent pairs of operations in SPMD programs with properly aligned synchronization operations [10][1], which applies to the vast majority of SPMD programs. This algorithm uses graph analysis techniques to reduce the computational cost of concurrency analysis from $O(n^3)$ to $O(n^2)$, where n is the number of operations in a program. Along with formally proving the correctness and complexity of the algorithm, I implemented it in the compiler for the Titanium language.

I also wrote an algorithm for pointer analysis of PGAS programs [8], proving it sound on a simplified version of the Titanium language and analyzing its running time. In addition to computing the set of allocation sites that each variable can reference as in traditional pointer analysis, the analysis also computes a measure of the distance between the executing thread and the thread that owns each referenced allocation site using a hierarchical machine model. I then implemented the analysis in the Titanium compiler, extending it to handle the full language.

With both analyses implemented in the Titanium compiler, I introduced client analyses and optimizations to take advantage of compile-time concurrency and pointer information. These included efficiently enforcing a sequentially consistent memory model [9], static race detection [8],

[1] All references are to the publication list in my CV.

and locality and data privacy analysis [16] to improve performance of distributed applications. Separately, I implemented a hybrid static and dynamic analysis to detect and report improper use of synchronization constructs in SPMD programs [7].

Hierarchical Computation

Hierarchical computation has been a major focus of my research. My work on pointer analysis led to the Hierarchical Partitioned Global Address Space (HPGAS) memory model, and I also introduced the Recursive Single Program, Multiple Data (RSPMD) model of execution. These models are now being used across many projects including Titanium at UC Berkeley, UPC++ at Lawrence Berkeley Lab, Hierarchical Coarray Fortran at Rice, and DASH at Ludwig-Maximilians-University Munich.

The RSPMD model organizes execution among threads arranged in a hierarchy of *teams*, or groups of cooperating threads. As part of my work, I introduced data structures for creating and manipulating team hierarchies, runtime support for automatically constructing team hierarchies from the structure of a machine, and language constructs for expressing computation across teams. These constructs were designed to allow a programmer to express hierarchical computation and to facilitate composition of multiple algorithms or libraries, but at the same time protect against deadlock using a combination of scoping rules and runtime checks. In addition to designing the model, I implemented it in the Titanium compiler and runtime and demonstrated on multiple benchmarks that the model enables better productivity and performance on hierarchical machines [6].

Parallel Programming Libraries

More recently, I have worked on taking ideas from parallel language research and implementing them in more accessible and maintainable libraries. In my time at Lawrence Berkeley Lab, I have been a key contributor to the UPC++ project, an implementation of a SPMD and PGAS programming model as a standard C++ library [5]. This includes porting and optimizing a multidimensional array library to UPC++ [4] and evaluating performance and productivity with an application benchmark [3]. I am also currently doing research into augmenting UPC++ with a model of hierarchical execution and data distribution. Other ongoing research includes implementing a distributed programming model in the Python language [18].

Future Directions

My current research interests include the following:

> **Programming models and tools for hierarchical machines.** Much work remains to be done in facilitating programming for hierarchical machines. For example, an abstract machine model is needed to represent the layout and communication cost of a physical machine without encumbering the programmer with too many details. Other research areas include support for distributed hierarchical data structures, improved compiler and runtime performance, and libraries for building hierarchical applications. Also crucial is a unified model of hierarchy that includes all aspects of a program, including execution, data distribution, resilience, and work stealing.

Dynamic languages for parallel computing. PGAS languages have demonstrated increased productivity over message-passing models in parallel computing. At the same time, dynamic languages such as Python and Ruby enable greater productivity in sequential programming than static languages. Combining dynamic languages with PGAS should similarly improve productivity in parallel programming.

Dynamic program analysis. SPMD and PGAS languages enable more precise compile-time program analyses than unstructured task-parallel languages. The runtime information available to dynamic analyses can likely result in even greater analysis precision. Compilers for high-level PGAS languages can be augmented to allow some degree of runtime introspection and optimization.

Specialization for distributed machines. Many projects have shown that domain-specific specializers can provide excellent performance on multicore machines while maintaining the expressiveness of a high-level language. An open area of research is whether or not it is possible to write *communication* specializers that optimize common communication patterns on large-scale distributed machines.