

Investigation of Programming Models for Emerging FPGA-Based High Performance Computing Systems

Andrew W. H. House and Paul Chow
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada M5S 3G4
{ahouse, pc}@eecg.toronto.edu

Abstract

This work proposes a set of requirements for programming emerging FPGA-based high performance computing systems, and uses them to evaluate a number of existing parallel programming models.

1. Introduction

As the rate of performance improvement in newer microprocessors levels off [1], system architects are forced to consider alternatives such as application-specific processors (ASPs) in high performance computing systems. Consequently, there is a pressing need for a flexible and easy programming model for these systems, and the purpose of this research is to identify the requirements of such a programming model and use them to evaluate existing approaches.

2. The HPC Landscape

The landscape of high performance computing (HPC) architecture is currently undergoing a period of rapid change as the approaches that have dominated in recent years are no longer sufficient to provide the desired performance improvements in the future. This has fostered an explosion of interest in the use of alternative processing elements such as FPGAs. Based on this, we can define four classes of massively-parallel HPC architectures:

- 1) *Multiprocessor Systems*, which use general-purpose microprocessors running in parallel to achieve high performance.
- 2) *Accelerated Multiprocessor Systems*, which use ASPs as co-processors to general-purpose processors to speed up parts of an application.
- 3) *Application-Specific Multiprocessor Systems*, which use ASPs as the main computing elements

instead of CPUs. An example would be a large multi-FPGA system.

- 4) *Heterogeneous Peer Multiprocessor Systems*, which combines CPUs and ASPs as first-class peers in a network where each has equal access to overall system resources.

These emerging heterogeneous computing systems share a common challenge – how to program them effectively and easily, and how to optimize the use of each of the different kinds of computational resources available. Finding the right programming model is the key to unlocking their power.

3. Programming Model Requirements

Creating a programming model that can be applied across all of the above architectures must necessarily limit its scope, and thus we choose to focus on the programming of HPC applications. This means that the programming model must be usable by scientists and researchers (rather than computer specialists) and should be able to effectively program any of the above classes of machines for HPC applications.

The generality of such a programming model means that it may not offer the best performance for each class of system, but if it can offer good performance and significant productivity and portability improvement across a large number of supported platforms, then it may find acceptance and even provide incentive to rewrite legacy code. With this in mind, we consider the constraints of the above architectures, as well as application and user constraints, and propose a set of nine requirements that can be used to evaluate existing and emerging programming models.

The programming model:

- 1) must support heterogeneous processing elements,

- 2) must be scalable,
- 3) must avoid features that are difficult to synthesize,
- 4) should assume only limited system services,
- 5) should support many types and classes of computation,
- 6) should allow the programmer to expose as much parallelism as possible, both coarse- and fine-grain,
- 7) should enforce separation of algorithm from implementation,
- 8) must be independent of the architecture it runs on,
- 9) should provide a degree of execution model transparency.

The first four requirements are derived from the architectural constraints; the next two from applications, as suggested in [1]. The final three user-centric requirements are largely guided by [1] and [2]. Collectively they are arranged in order of approximate importance. They also indicate the need for a set of robust back-end tools to handle the hidden complexity of the new programming model, or at least the means for users to access and influence those low-level details.

4. Programming Model Analysis

There is currently no standard model for programming these emerging classes of systems, but there are many other approaches to parallel programming that may be applicable. These include data-parallel programming, dataflow and functional programming, stream computing, communicating sequential processes, hardware description languages, shared memory programming, actor-based languages, and partitioned global address space (PGAS) languages.

A more comprehensive survey is in preparation that examines these models in-depth using the above requirements. The initial results suggest that no existing approach satisfies all of the requirements, and that a new programming model is needed. However, each of the existing models fails in different ways. From this, it is possible to derive a set of features that should be avoided in any new programming model, such as:

- An SPMD programming style
- Dynamic memory allocation and pointers
- Explicit introduction of all parallelism
- Explicit communication between processes
- Explicit mapping of work to processing elements
- Only one kind of parallelism/computation
- Complex or costly process synchronization

Similarly, from the strengths of existing programming

models, we can also derive a list of useful features that includes:

- Data-parallel operations
- Implicit communication and synchronization
- Emphasis on libraries and functions
- Functions without side effects
- Global view of memory
- Data streams and implicit parallelism (futures)
- Region-based array management

A programming model that incorporates these features with an eye toward programming reconfigurable hardware would be a strong candidate as a universal programming language for emerging accelerator-based HPC systems. Such a programming model could be a modification of an already-existing language (such as a stripped-down PGAS language to remove problem features, a version of Matlab extended to support different kinds and levels of parallelism, or something like Mitrion-C extended to a heterogeneous platform), or something entirely new.

5. Conclusion

Based on an evaluation of the emerging landscape of high performance computing and the types of applications it must serve, we have proposed a set of requirements that should be met by any new programming model meant to support these architectures. Based on an analysis of a variety of existing parallel programming models for high performance computing, sets of positive and negative features for a new language have been identified. A language that meets all of the requirements presented here would be useful in programming accelerator-based high performance computing systems.

Acknowledgments

This work was supported by NSERC, Xilinx, and the Walter C. Sumner Memorial Fellowship.

References

- [1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from Berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, December 2006. [Online]. Available: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html>
- [2] B. L. Chamberlain, D. Callahan, and H. P. Zima, "Parallel programmability and the Chapel language," *International Journal of High Performance Computing Applications*, vol. 21, no. 3, pp. 291–312, August 2007.