LLVM Optimizations for PGAS Programs

Case study: LLVM Wide Pointer Optimizations in Chapel

Akihiro Hayashi *, Rishi Surendran, Jisheng Zhao
Rice University
{ahayashi, rishi, jisheng.zhao}@rice.edu

Michael Ferguson
Laboratory for Telecommunication Sciences
mferguson@ltsnet.net

Vivek Sarkar
Rice University
vsarkar@rice.edu

Abstract

PGAS programming languages such as Chapel, Coarray Fortran, Habanero-C, UPC and X10 [3–6, 8] support high-level and highly productive programming models for large-scale parallelism. Unlike message-passing models such as MPI, which introduce non-trivial complexity due to message passing semantics, PGAS languages simplify distributed parallel programming by introducing higher level parallel language constructs that include operations on global / distributed arrays, distributed task parallelism, directed synchronization and mutual exclusion.

Past studies on program analysis and optimizations for PGAS programming languages have been specific to different languages, i.e. either built on special compilers (e.g [1]) or relied on recognizing language specific runtime APIs [2]. Since most PGAS programming languages have similar constructs for task management, synchronization, and communication, we believe that it is feasible to build a common compiler parallel intermediate representation (PIR [9]) that supports these features across multiple languages. A PIR-based approach enables compiler developers to write parallelism-aware program optimizations in a language-independent manner. We choose LLVM (Low Level Virtual Machine) [7] as the baseline infrastructure for implementing our PIR, since it is easy to extend, is widely used, and supports a wide range of hardware platforms.

The following steps summarize our approach to building PIR-based compiler infrastructure to optimize PGAS program:

1. Study PIR-related opportunities and challenges in existing compilers for PGAS programming language.
2. Extend LLVM IR to support Runtime-Independent and Runtime-Specific PIRs.
3. Add new semantic constraints to LLVM for PIR primitives.
4. Implement a core set of parallelism-aware optimizations for PGAS programs.

To take the first step towards the goal, we focus on optimization of Chapel programs[1], with an initial focus on communication optimizations. The Chapel compiler uses the address space feature in LLVM to distinguish between local and remote data, thereby providing a language and runtime-independent representation for the local/remote data communications. In particular, the *Wide Pointer Optimization* pass expresses remote data accesses as LLVM load/store/memcpy instructions that involve a designated address space (100) for remote accesses. Then, classic LLVM optimization passes (e.g. loop invariant code motion) can be applied to eliminate redundant data access instructions even if they involve different address spaces. Finally, instructions involving remote address space pointers are lowered to runtime communication APIs such as *get* and *put* that operate on the remote pointers.

* Presenting Author

[1] To the best of our knowledge, Chapel is the first PGAS programming language to use LLVM as a back-end.

In this talk, we demonstrate the impact of the wide pointer optimization pass using multiple Chapel programs. The result of the Smith-Waterman program shows a performance improvement of 194% on 16 locales (1 locale per node, 12 CPUs per locale) on an Intel Xeon X5660 cluster with a quad data rate InfiniBand interconnect. This experiment shows that the use of LLVM can effectively improve the performance of PGAS programs. Our study also shows that the PIR based parallel program optimization is a promising way to enhance the compiler construction for future PGAS parallel programming languages.

References

[1] R. Barik, Jisheng Zhao, D. Grove, I. Peshansky, Z. Budimlic, and V. Sarkar. Communication optimizations for distributed-memory x10 programs. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1101–1113, 2011.

[2] Christopher Mark Barton. Improving Access to Shared Data in a Partitioned Global Address Space Programming Model. 2009. Ph.D. Thesis.

[3] Chapel. The Chapel language specification version 0.93, April 2013.

[4] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '05, pages 519–538, New York, NY, USA, 2005. ACM.

[5] Sanjay Chatterjee, Sagnak Tasirlar, Zoran Budimlic, Vincent Cave, Milind Chabbi, Max Grossman, Vivek Sarkar, and Yonghong Yan. Integrating asynchronous task parallelism with mpi. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 712–725, Washington, DC, USA, 2013. IEEE Computer Society.

[6] Tarek El-Ghazawi, William W. Carlson, and Jesse M. Draper. UPC Language Specification v1.1.1, October 2003.

[7] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 75–, Washington, DC, USA, 2004. IEEE Computer Society.

[8] Robert W. Numrich and John Reid. Co-Array Fortran for parallel programming. *ACM SIGPLAN Fortran Forum Archive*, 17:1–31, August 1998.

[9] Jisheng Zhao and Vivek Sarkar. Intermediate Language Extensions for Parallelism. *5th Workshop on Virtual Machine and Intermediate Languages (VMIL'11)*, October 2011.