Affine Loop Optimization Using Modulo Unrolling in CHAPEL

Aroon Sharma

April 4, 2014

Abstract

Compilation of shared memory programs for distributed memory architectures is a vital task with high potential for speedups over existing techniques. The partitioned global address space (PGAS) parallel programming model, one such implementation, exposes locality of reference information to the programmer thereby improving programmability and allowing for compile-time performance optimizations on top of a shared memory programming model. In particular, shared memory programs compiled to message passing hardware can improve in performance by aggregating messages and eliminating dynamic locality checks for affine array accesses in the PGAS model.

This research presents a loop optimization for message passing programs that use affine array accesses in Chapel, a PGAS parallel programming language. Each message in Chapel incurs some non-trivial run-time overhead. Therefore, aggregating messages improves performance. The optimization is based on a technique known as modulo unrolling where the locality of any affine array access can be deduced at compile time. First pioneered by Barua et al for tiled architectures, we adapt modulo unrolling to the problem of efficiently compiling PGAS languages to message-passing computers. When applied to loops and distributed data, modulo unrolling can decide when to aggregate messages thereby reducing the overall message count and run time for a particular loop. Compared to other methods, modulo unrolling greatly simplifies the very complex problem of automatic code generation of message-passing code from a non-message passing language like Chapel. It also results in substantial performance improvement compared to the un-optimized Chapel compiler.

To implement this optimization in Chapel, we modify the leader and follower iterators in the Cyclic and Block Cyclic data distribution modules. Results were collected that compare the performance of our optimized Chapel programs with programs using the existing Chapel data distributions. For a ten-locale machine tested on three benchmarks, we see on average a 92

1 Introduction

Hello. I want to add to this document now. Please refer to section 1.