Auto-tuning within the R-Stream Compiler

Kaushik Datta, Nicolas Vasilache, and Richard Lethin
{datta, vasilache, lethin}@reservoir.com

Reservoir Labs, Inc., 632 Broadway, Suite 803, New York, NY 10012

Abstract. The R-Stream compiler is an automatically parallelizing compiler based upon the polyhedral model. This model is a linear algebraic representation of programs; its power, however, lies in the fact that it can perform advanced code restructuring based on precise data dependence analysis. This paper introduces a non-traditional approach to auto-tuning in the R-Stream compiler.

In 2009, Datta (the first author of this paper) published his thesis showing the benefits of automatic tuning for stencil codes across a variety of cache-based multicore architectures [2]. The thesis exhibited speedups of up to 5.4× over straightforward implementations of the stencil code. While the results show great promise for auto-tuning, we now ask the question: how much of this technology is already captured in any of today's production compilers? We find that the R-Stream compiler [3] is often able to complement or broaden the capabilities of his code generator by being able to:

- encapsulate many of the code generator's stencil-specific transformations
- easily apply these transformations to other stencil kernels
- maintain program semantics after every optimization
- apply these optimizations without programmer specification
- also run on distributed memory and local store architectures

One of the largest hurdles facing the stencil auto-tuning system was the fact that the generated stencil code variants were produced using a heterogeneous software system of C files and Perl scripts. Moreover, each script was essentially customized to a single stencil kernel. In most cases, several optimizations were layered together into a single script. However, for optimizations requiring drastic code changes (e.g. SIMDization), additional scripts would be developed for that particular kernel. Clearly, programmer productivity was low; the introduction of new optimizations, as well as changes to the stencil shape, size, or dimensionality would usually require new Perl scripts.

The R-Stream compiler is able to handle this situation much more elegantly. The compiler encapsulates many of the optimizations used in the stencil auto-tuning environment, including parallelization, different levels of tiling, loop unrolling, and SIMDization. Moreover, in order to boost programmer productivity, it accepts sequential C code as input. As a result, the programmer is able to optimize a different stencil kernel by merely changing the serial C code, *not* by creating a new script.

R-Stream is a production compiler, not a code generator, so it is also able to preserve program semantics. Unfortunately, code generators provide no such guarantees on correctness; the resultant code needs to be manually verified by the programmer, which is often both time-consuming and fallible. R-Stream, on the other hand, uses an intermediate representation (IR) that can be transformed by merely finding the solution to a system of constraints generated from the *polyhedral model* [3]. As a result, preserving the program semantics after each transformation is much simpler.

However, perhaps the most impressive aspect of R-Stream is that it is able to apply many powerful stencil-specific transformations without human intervention. For instance, time skewing is a stencil optimization introduced by Wonnacott [4] to preserve data locality when performing multiple Jacobi sweeps. Remarkably, the algorithm falls naturally out of the polyhedral model, since the model schedules statement executions based on their iteration spaces and dependencies. Similarly, Gauss-Seidel stencil sweeps, which are harder to parallelize than Jacobi sweeps, are easily parallelized through the polyhedral model.

More generally, R-Stream is able to handle most loop-based codes with regular accesses, not merely stencils. It can also compile for platforms beyond the standard shared memory architectures; this includes heterogeneous local store architectures like GPU [1] and STI Cell. For these systems, R-Stream is automatically able to generate DMA calls and perform array placement.

In order to generate the highest-performing code, R-Stream currently uses cost models that trade-off optimizations targeting specific architectural features. These models allow us to automatically extract coarse-grained or fine-grained parallelism, increase or decrease locality, and restructure loops to obtain contiguous memory accesses. The key difference between R-Stream and other approaches (including other polyhedral approaches) is that we optimize these metrics jointly, in a single pass. As a result, we are able to trade-off the very complex and subtle interactions between these desired properties. For instance, using loop fusion to augment the amount of reuse in a program frequently reduces the available parallelism. On the other hand, reducing the parallelism may be beneficial if we are optimizing for a single core. The cost models allow R-Stream to know and understand these high-level program properties.

However, the cost models may still need refinement when compiling on a new platform. This is where auto-tuning comes in. We do not plan to use traditional search-based auto-tuning, where many different, but equivalent, code variants are executed so as to find the best-performing one; this "black-boxing" usually does not shed light on the attained performance. Instead, we envision that by running certain code variants, we can assign appropriate costs to features like contiguity, parallelism, locality, and communication via machine learning techniques. Such an approach would let us better understand the platform, not abstract it away.

References

1. A. Leung et al. A mapping path for multi-gpgpu accelerated computers from a portable high level programming abstraction. In *GPGPU-3*, Pittsburgh, PA, 2010.
2. K. Datta. *Auto-tuning Stencil Codes for Cache-Based Multicore Platforms.* PhD thesis, University of California, Berkeley, Berkeley, CA, USA, December 2009.
3. Reservoir Labs. *R-Stream Parallelizing C Compiler Power User Guide*, 2010.
4. D. Wonnacott. Using time skewing to eliminate idle time due to memory bandwidth and network limitations. In *IPDPS '00*, page 171, Cancun, Mexico, 2000.