

PERFORMANCE AUTOTUNING WITH ORIO

Azamat Mametjanov
Mathematics and Computer Science Division
Argonne National Laboratory

INTRODUCTION

A program is an algorithm that operates on structured data. An algorithm is implemented in a structured programming language. The grammatical structure of a programming language can be used to treat programs as data, where a meta-program transforms other, first-order programs. Compilers (the best-known application of meta-programming) transform programs/data in a high-level (easy to develop) source language to programs/data in a low-level (machine-executable) target language. The primary goal of compilation is to create an executable for a particular instruction set architecture: e.g. VLIW for embedded processors, CISC for Intel's x86 processors and RISC for IBM's PowerPC processors. Other goals of meta-programming are to (1) optimize code for performance and/or energy, and (2) refactor code for resiliency, maintainability and readability. While compilers are good at program translation, they are notoriously bad at optimization and refactoring objectives. Active research and development is underway to create optimizing compilers and refactoring tools that assist programmers in quickly developing functionally correct, fast and resilient codes.

Prototyping and development of a scientific application requires a deep component stack. It typically includes design choices on hardware architecture, OS, programming language, compiler and libraries. While libraries may provide significant functional capabilities for an application, they may in turn rely on other libraries and increase the depth of dependencies (e.g. BLAS, NetCDF, MPI).

Having chosen a particular component stack, an application can become a monolithic "stove-pipe" that is vulnerable to configuration changes such as version updates. More importantly, the choices of components can lock-in the application making it hard to port to other component configurations.

While some tool suites intend to provide portability across platforms (GNU, PGI, LLVM), some platforms require an application to use a specific tool chain: e.g. IBM Blue Gene P/Q, Cray XT/XE/XK, Intel MIC systems. The heterogeneity that is induced by multiple choices at each layer of a component stack not only creates compatibility issues, which can be handled by configuration tools, but also performance and refactoring variability, which have not been systematically addressed.

METHOD

The goal of autotuning is to automatically tune the performance of a code for a given platform configuration. Because the targeted platforms can be heterogeneous, the accompanying benefit of autotuning is performance portability. This is accomplished by annotating existing code with performance directives in the form of source code comments. The annotation-based approach does not modify the semantics of a given program, which acts as a reference implementation that can be compiled and executed to obtain reference results. The annotated code is transformed according to performance directives, compiled and executed to obtain its performance metrics and outputs for comparison with reference results. Depending on the number of parameter variations within the directives, a number of resultant code variants are compared and the highest performing variant is selected as the result of autotuning.

RESULTS

We have extended Orio's annotation-based approach to generate CUDA GPU code from an existing C code. Performance tuning parameters include number of threads in a block, number of blocks in a grid of CUDA threads, number of asynchronous streams, size of L1 cache and loop unroll factor among others. The result is an auto-generated CUDA code that performs better than hand-tuned cuSPARSE library code. Figure 1 illustrates how the autotuned code consistently outperforms the library code [1].

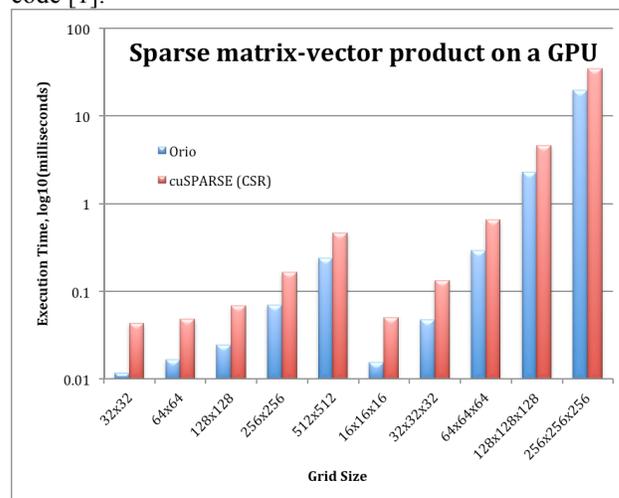


Figure 1: Performance (lower execution time is better) of autotuned vs. special-purpose CUDA GPU library code on sparse matrices from 5- and 7-point stencils on grids of various sizes.

REFERENCES

1. A. Mametjanov, D. Lowell, C.-C. Ma, and B. Norris. Autotuning Stencil-Based Computations on GPUs. In *Proceedings of IEEE International Conference on Cluster Computing (Cluster'12)*, September 2012.