Autotuning for High Performance Computing

David Padua
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N. Goodwin Avenue
Urbana, IL 61801-2302
+1 (217) 333-4223
padua@illinois.edu

ABSTRACT

Program performance depends not only on the algorithms and data structures implemented in the program but also on *coding parameters*. These parameters include frequency and size of messages, shape of loop tiles, and minimum number of iterations required for parallel execution of a loop. Making the right selection of algorithms, data structures and coding parameters for a given target machine can be an onerous task in part because of the many *machine parameters* that must be taken into account and the interaction between these parameters. Important machine parameters include cache size, memory bandwidth, communication costs, and overhead. Furthermore, some of the selections must often be reassessed when porting to a different machine even when this machine does not differ significantly from the original target.

It is clearly advantageous to make use of tools and techniques that help reduce the initial effort of programming for performance as well as the cost of porting. The tool that comes first to mind is the *compiler*. Compilers were developed to enable machine independent programming and, to this end, apply powerful code generation and optimization strategies that take into account machine parameters. However, compilers not always suffice. They operate almost exclusively at the coding level and even at this low level they are not always effective. For example, compilers often fail to reorganize loops in the best manner when generating code for microprocessor vector extensions. Good use of these extensions today requires manual intervention.

Autotuning programs are those capable of generating one or several versions of a program. These versions could be derived from a parameterized program, or from descriptions at a higher level of abstraction that could take the form of algorithms or even problem specification. It is also desirable to take into account target machine parameters and the characteristics of the input data in the generation process.

When multiple versions are generated, one is selected at compile-time or at run time by carrying out an empirical search that executes the versions with representative data and measures program performance to guide the selection.

Autotuning programs can be written in conventional code such as Fortran, C, C++, or java, annotated with transformations that can be applied to the whole program or to code segments. Alternatively, autotuning programs can be written in a very high level declarative notation that represent algorithms or problems to be solved.

Although the initial cost of developing an autotuning program is higher than that of developing a conventional program, it has the advantage that much of the analysis required for the first target machine is done automatically and that it can be ported across machines and machine classes maintaining good performance.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *complexity measures, performance measures, product metrics.* D.3.4 [**Programming Languages**]: Processors – *optimization.*

General Terms

Measurement, Performance, Experimentation.

Keywords

Autotuning, Program synthesis.