X-TUNE

Autotuning for Exascale: Self-Tuning Software to Manage Heterogeneity

Principal Investigator: Mary Hall (University of Utah)
Other Investigators:
Jeff Hammond, Paul Hovland, Sri Hari Krishna Narayanan and Stefan Wild (ANL);
Leonid Oliker, Brian Van Straalen and Samuel Williams (LBNL);
Jacqueline Chame (USC/ISI)
http://ctop.cs.utah.edu/x-tune/

Objectives

We will develop the first unified autotuning framework that seamlessly integrates programmer-directed and compiler-directed autotuning, so that a programmer and the compiler system can work collaboratively to tune a code, unlike previous systems that place the entire tuning burden on either programmer or compiler. The proposed system will dramatically improve generality and usability of autotuning technology through an integrated, composable collection of tools, including an autotuning compiler framework, language extensions, a code transformation framework, compiler decision algorithms and performance models. To maximize productivity impact of autotuning and make it approachable by many users, it should be encapsulated in domain-specific tools developed by expert users and made available to others. To this end, we will demonstrate autotuning on computations from AMR MG, Combustion Co-Design Center, TCE and Nek5000, and will work with DOE to define a small number of other mini-app demonstrations. We will identify opportunities for integration, software reuse and demonstrations with other X-stack projects.

X-TUNE Framework

The X-TUNE framework is described in the following figure.

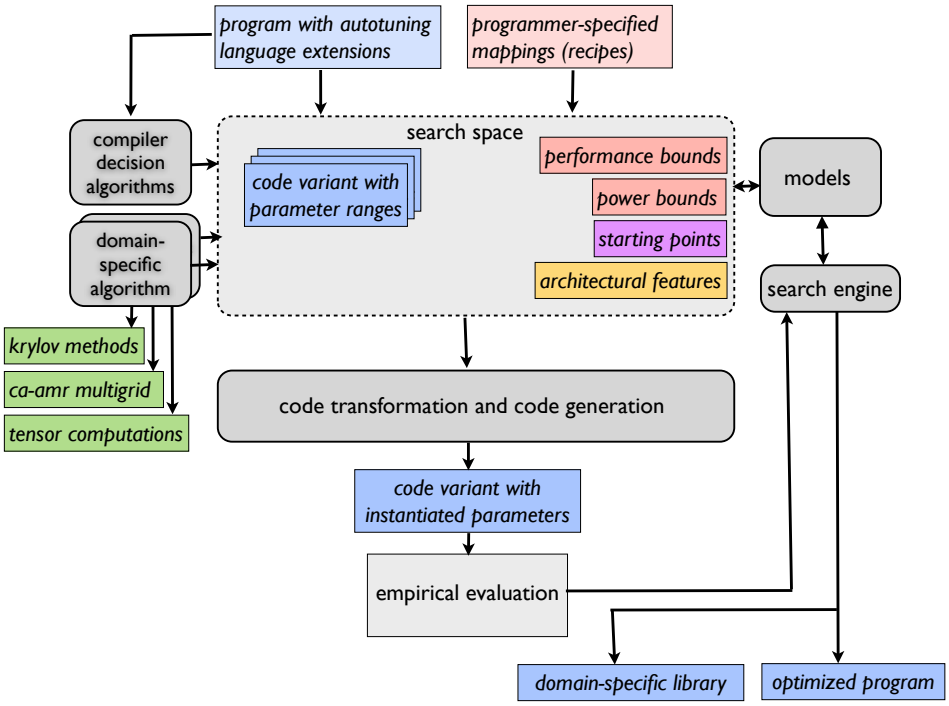

Figure: X-TUNE End-to-End Autotuning System.

Overview

- Expert developers employ autotuning under their control by expressing ***code variants*** and ***optimization parameters,*** arising in program specification or in directing compiler optimization. Code variants may include different algorithms for implementing a computation, or implementations partially customized to different target processors (e.g., in today's systems, this might be OpenMP code for a conventional multicore and CUDA code for a GPU). Optimization parameters refer to discrete values that govern optimization or code generation (e.g., number of threads to apply to a computation).
- An autotuning transformation and code generation framework called CHiLL employs a polyhedral representation to transform computation and manage complex memory hierarchies. ***Transformation recipes*** describe the mapping of the source code to a particular architecture.
- By focusing on particular application domains, we can customize the programmer guidance, compiler decision algorithms and recipes to the needs of the domains.
- The language constructs and transformation recipes describe a search space of possible implementations (search algorithms funded by other efforts).
- The compiler decision algorithms and autotuning search will consult performance models using the PBound modeling software.
- Additional end-to-end support includes specialization and dynamic selection of optimized variants, along with triage to identify where in the program to focus autotuning efforts.
- Domains include adaptive multigrid and tensor contractions, and similar problems from the Exascale Co-Design Centers.
- Near-term architectures serve as proxies for exascale platforms, including the Xeon Phi (MIC), BGQ, and GPUs.

Case study: Adaptive Mesh Refinement Multigrid

Multi-grid codes exhibit:

- few stencil (operator) executions and limited reuse of data within each sweep.
- exponentially decreasing parallelism and high communication costs.

Adaptive Mesh Refinement adds complexity through inclusion of

- many small (possibly irregularly shaped) boxes, complex (coarse-fine) boundary conditions and complex communication patterns.

Real applications demand solvers for particularly complex systems/operators

- productive operator descriptions lend themselves to inefficient implementations, while efficient implementations are rarely portable.

Focus on data movement (since it is the bottleneck)

- support communication-avoiding algorithms, stencil fusion, optimize coarse-fine boundary conditions, express and tune hierarchical parallelism within a node, optimize high-order stencils.

Impact

- Expert programmer and compiler work collaboratively to tune a code.
- Programmer guides optimization, but relies on system to carry out tedious details including code generation and management of autotuning experiments.
- The autotuning process is designed to be encapsulated in domain-specific tools so that other less-sophisticated users of the software can reap the benefit of the expert programmers' efforts.
- New optimization strategies can be added to an existing code base, and machine-specific details are isolated from the high-level source code.