AESOP:

Adaptive Environment for Supercomputing with Optimized Parallelism

BAE SYSTEMS

SRC

Parakinetics

CRAY

Objective: Automated transformation of sequential code, written in standard languages, into continuously optimized parallelized code for a wide range of heterogeneous computing systems.

MOTIVATION

The increasing complexity of modern computing systems makes it difficult to achieve even a reasonable fraction of a system's available performance. System builders currently employ a variety of techniques to boost peak system performance including multi-core architectures, heterogeneous systems, and accelerators built from nontraditional processing elements. Unfortunately, the level of sophistication and expertise required to develop and tune a program is growing with the complexity of the underlying systems. As system complexity continues to grow, this problem creates an increasingly serious bottleneck which inhibits the ability of programmers to achieve the performance potential of modern computer systems.

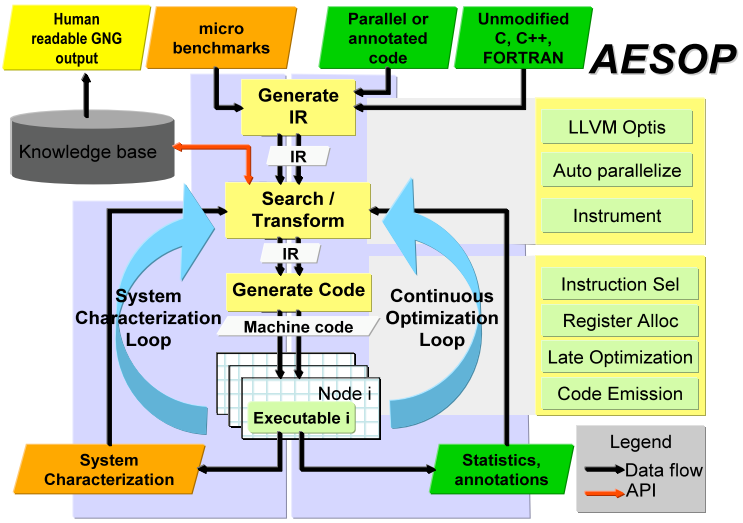
Programming language compilers that map an application onto the underlying hardware are the principal moderators of performance. It is typical for a user to provide substantial assistance in this process through source-level directives, direct specification of low-level application programming interfaces (APIs), and many other mechanisms. The level of application performance achieved is limited by the compiler's ability to make full use of the complex functionality of the underlying hardware.

Compilers have become large complex monolithic software systems that can have upwards of two million lines of code and take many years to develop. Optimized production compilers are typically released several years after the initial release of the computer systems they target. As a consequence, the optimized compiler is frequently released around the time that its targeted computer is eclipsed by the next generation computer platform.

An Architecture-Aware Compiler Environment (AACE[1]) such as AESOP, is a computationally efficient compiler that incorporates learning and reasoning methods to drive compiler optimizations for a broad spectrum of computing system configurations that can keep pace with the variety and rate of deployment of new parallel computer systems.

OUR SOLUTION

We are developing novel compiler technology that brings practical automatic parallelization into mainstream software development reflecting the fact that people program using conventional serial programming languages. Three key innovations drive our development. **System Characterization** automatically extracts key descriptive elements about the computing platform to inform the compiler's optimizations (e.g. memory architecture, core configuration). **Automatic Parallelization** will automatically parallelize sequential codes, will extract fin-grained parallelization from explicitly parallelized code and lightweight semantic annotations from the programmer will enable even more significant levels of parallelization. Finally, **Continuous Optimization & Learning** will exploit parallel hardware to conduct multiple experiments to learn how to estimate performance from system characteristics. These key elements are divided into three main areas shown in the diagram here:



1. **The System Characterization Loop** is highlighted on the left in the diagram. System characterization is seeded by micro benchmarks (input from top into LLVM[2]'s Generate Intermediate Representation step which also takes as input the source code either unmodified serial or annotated parallel code in upper right) and iterates, driven by a nonlinear optimization search engine. Derived system characteristics (bottom left) drive and increase the efficiency of the Continuous Optimization Loop for auto-parallelizing, compiling and running parallel applications.

2. **The Continuous Optimization Loop** (diagram right) encompasses our dynamic runtime system, which includes applying transformations (e.g. loop unrolling, code hoisting, application of concurrent threads) and instrumentation to code, monitoring the results of the transformations and instrumentation, and then applying further transformations. Knowledge-based search chooses the next set of optimizations to explore the massive search space of program transformations.

3. **The Learning and Reasoning** element consists of the persistent Knowledge Base (upper left), its inputs—which

---

[1] Architecture-Aware Compiler Environment proposed and funded by DARPA.

2 Low-Level Virtual Machine. An open-source complier framework from The University of Illinois and now maintained by Apple. http://llvm.org

come from both system characterization and application instrumentation—and its query interface API (connecting line). The Knowledge Base tracks relations between code fragment features (e.g. operation count), optimization parameters (e.g. loop unrolls) and system characteristics (e.g. L1 cache size) and continuously refines learned classifications. The learned correlations among machine, code, transform characteristics and performance results are inputs to the Search/Transform (center) to inform the next positions in the search space of program transformations.

These technologies combine to allow AESOP to automate the optimization and parallelization of both scientific and general purpose applications, to increase programmer productivity through automation, to quickly produce detailed system characterizations in order to seed the optimization search, and to adapt automatically to novel system configurations and future parallel hardware architectures.

INNOVATIONS

1. Linear programming approach to system characterization minimizes the number of micro-benchmark runs needed to reach >90% accuracy.
2. Automatic parallelization of general-purpose sequential code enables >10X productivity for full software life-cycle and range of system configurations.
3. Run-time system uses efficient heuristic search to adapt parallelization and optimization to the system to achieve >20% performance increase.
4. Learning from results of optimization seeds iterative compilation and transfers results across runs, programs and even hardware.
5. Design combines all these elements seamlessly for full AACE vision including adapting codes to new hardware systems with zero code modifications.
6. AESOP's architecture and implementation is designed to promote broad applicability and user acceptance. By being based on the LLVM compiler framework and adopting the same unencumbered open source licensing, combined with LLVM's plug-in architecture, AESOP is an ideal platform on which researchers and commercial entities can base their work and products.

GO/NO-GO METRICS

Processing system characterization (> 75%/90% accuracy phase I/II). Building on the mature Active Harmony framework, AESOP will auto-adapt to never-before-seen computer systems through highly accurate system characterization.

10X development productivity with 20% runtime performance gains. Early indications from SPEC 2000 benchmarks indicates AESOP will achieve >80% performance improvements when applied in a completely automated fashion to general-purpose (serial) code in C/C++/Fortran. Even more substantial (>500%) performance benefits will accrue if programmers also provide sparse (<1/100% source lines) semantic annotations.

TEAM

Member	Affiliation	Task areas
Dr. Gregory Sullivan	BAE Systems, AIT	PI Knowledge base, System Architecture
Dr. Jothy Rosenberg	BAE Systems, AIT	Assist PI Integration, Open Source & commercialization prep
Dr. David August	Princeton University Parakinetics	AESOP architecture Automatic parallelization
Dr. Scott Mahlke	U of Michigan Parakinetics	Automatic parallelization LLVM modifications
Dr. Rajeev Barua	U of Maryland	Affine and non-affine transformations
Dr. Alan Sussman	U of Maryland	MPI-level optimization System characterization
Dr. Rance Cleaveland	U of Maryland	System characterization Optimization parameter space exploration
Dr. Dan Poznanovic	Cray	Supercomputer expert, advisor on preliminary and final design

PHASES§ AND GOALS

Phase 1	18 Months
• Develop preliminary design of AESOP concept and architecture.	
• Specify AESOP source languages and extensions.	
• Prototype the AESOP tool environment to support end-to-end system experimentation and to assess performance of the system characterization features.	
Phase 2	**18 Months**
• Develop final detailed design for AESOP based on feedback, reviews and prototyping efforts.	
• Develop an open source release plan.	
• Develop source-to-source language translator based on the language specification established at PDR.	
• Develop a prototype capability including run-time system to automatically optimize unmodified C/C++ or Fortran codes for efficient execution on single-node, multi-note and heterogeneous systems.	
Phase 3	**12 Months**
• Complete the implementation of AESOP. Continue the spiral development process, including internal gate metric evaluations.	
• Execute the open source release plans.	
• Support system evaluations and AACE bake-offs.	

§ Start date March 9, 2009