

---

## Improving Code Quality

### Developing & Deploying New Ideas

**Keith D. Cooper**  
Rice University  
Department of Computer Science

[http://lacs.rice.edu/review/slides\\_2006](http://lacs.rice.edu/review/slides_2006)

*Participants:*

Linda Torczon, Devika Subramanian, Tim Harvey, Steve Reeves  
Alex Grosul, Todd Waterman, Anshuman Dasgupta, Jason Eckhardt, Jeff Sandoval, and Yi Guo



---

## Background

- Our group is funded under both performance & components
  - Better compiler techniques for microprocessor based systems
  - Automatic application tuning through adaptive compilation
- Focus on development & deployment
  - Invent new code optimization techniques
    - Improve runtime performance
    - Broaden suite of codes that achieve “good” performance
  - Transfer technology into important compilers
    - Widely used open source systems & vendor compilers
    - Model implementations to guide commercial
- Three examples
  - Instruction mix, register allocation, adaptive inline substitution



---

## Changing Instruction Balance

**Motivation:** Some of the performance critical loops in Sweep3D have a low ratio of useful (floating point) ops to total ops

**Question:** Can we transform them to improve performance?

- Mellor-Crummey rewrote loops in SAGE to cure similar problems
- Can we develop automated techniques to achieve similar results that are suitable for use in compilers such as gcc or Intel’s compilers?
  - Low-level automatic techniques would help a variety of codes

- Launched a study that tried novel combinations of

- Algebraic reassociation
- Strength reduction
- Code motion

Based on prior experience, we believed that we could reduce the overhead from addressing & control flow

- We have not had much success
  - Subject of continuing research

---

Developing new techniques



---

## Register Allocation

**Motivation:** High-quality register allocation plays a critical role in single-processor performance

- Open-source systems have a history of using weak algorithms

**Goal:** Move best practices into important open-source systems

- Direct performance benefits for most programs

**Activities in the Last 18 Months**

- Advised implementor of gcc’s “new-ra”
  - Uses many of our algorithms & data structures
- Built two new global allocators for LLVM
  - LLVM is a strong contender for next-generation gcc backend
  - Implementations show how to adapt allocators for x86 ISA
  - IP issues complicate the distribution process (working on it)
- Made fundamental algorithmic improvements
  - Better coalescing, faster graph updates for JIT environments

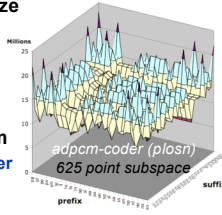
---

Technology transfer



## Adaptive Compilation (Background)

- Today's compilers apply fixed set of passes in a fixed order
  - No relationship to mathematical notion of optimization
- Our systems look for "good" optimization sequences
  - Feedback-driven search over large, complex spaces *1 trillion points of interest*
  - 20 to 40% improvement over fixed sequence optimizers
  - Reduced cost for "good" sequence from 20,000 evaluations to  $\leq 250$
- Developed search techniques that capitalize on properties of the search spaces
  - Large scale studies to learn properties
  - Efficient searches for different cost points
  - Randomized restart avoids local minima
- Sequence finding is a hard search problem
  - Finding loop blocking factors is much easier
- Spawned interest in academia & industry



New techniques  $\Rightarrow$  new tools



## Adaptive Inline Substitution

- Inline substitution is a well-known transformation
  - Replace procedure call with body of called procedure
  - Eliminates overhead & creates larger scope for optimization
  - Important in OO languages, also improves C or FORTRAN
- Difficult problem is deciding which call sites to inline
  - Decisions interact with each other *And, inlining can have negative side effects*
  - Profitability depends heavily on context
  - Underlying graph changes as decisions are made
- Current state of the art for the decision procedure
  - A single heuristic applied at each call site in each application
  - We have shown that program-specific inlining heuristics can produce significantly better code than any general scheme

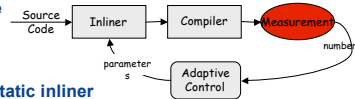
New techniques  $\Rightarrow$  new tools



## Adaptive Inline Substitution

Inline substitution is a natural application for adaptive behavior

- Built a demonstration system for ANSI C programs
  - Analyzes whole program and collects data on program properties
    - Nesting depth, code size, constants at call, call frequency, etc.
    - Experimented with 12 properties in Waterman's thesis
  - Apply tunable heuristic at each call site
    - Compare actual values against parameter values
    - Use search to select best parameter values
  - Produce transformed source
  - Compile, run, evaluate
- Improvements of 20% over static inliner and 30% over original (PowerPC & Pentium)
- Heuristics vary by application and by target architecture



New techniques  $\Rightarrow$  new tools



## Adaptive Inline Substitution

Key design issues

- Finding a good way to parameterize the problem & the software
  - Takes a "condition string" in CNF where each clause is a program property and a constant, e.g.,
 

```
inliner -c "sc < 25 | lnd > 0, sc < 100" foo.c
```
  - Search produces a condition string that can be used repeatedly
- Search space is huge
  - Range of values depends on input program
    - Estimate the range & discretize it into 20 intervals
  - Condition string syntax admits too many choices
  - Designed a single format for condition strings in our experiments

$sc < A \mid sc < B, lnd > 0 \mid sc < C, scc = 1 \mid$   
 $clc < D \mid cpc > E, sc < F \mid dcc > G$

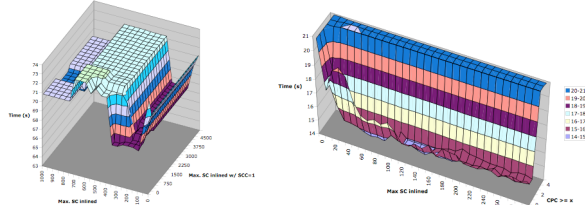
} Fixes the search space's "shape"

New techniques  $\Rightarrow$  new tools



## Adaptive Inline Substitution

Search spaces are much smoother than in sequence finding problem



*bzip, varying sc and sc for single-call procedures*

*vortex, varying sc and constants per call*

- Designed search techniques for these spaces
  - Impatient hill-climber and random restart
- And validated them experimentally

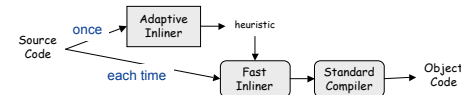
New techniques ⇒ new tools



## Adaptive Inline Substitution

How can we deploy these results?

- Source-to-source inliner
  - Runs for a while and produces a CNF expression that describes a program-specific heuristic
  - Use the inliner on subsequent compilations with that heuristic
    - If code properties change “enough”, re-run the search



- Tools
  - We are implementing these ideas in Rose and in Phoenix
  - For FORTRAN 90, we are looking for a front end *(build one?)*
  - Expect to have (& distribute) working tools later this year

New techniques ⇒ new tools



## Adaptive Optimization

Further work with adaptive inlining

- Robust implementations applied to large programs
- Experiment with other properties *(obj. code size)*

Apply knowledge & insight to other hard problems

- Expression refactoring
  - Balance for ILP, depth for registers, affinity for folding & CSE
  - Another extremely complex problem
- Register allocation
  - Reformulate coalescing, recoloring, rematerialization, Ir splitting
  - Evaluate variants on same optimized code & interference graph

*Attack on instruction balance*

Key issues

- Developing expressive parameterizations
- Designing effective search strategies



## Summary

- Better compiler techniques for microprocessor based systems
- Automatic application tuning through adaptive compilation
- Invent new code optimization techniques
  - Improve runtime performance
  - Broaden suite of codes that achieve “good” performance
- Transfer technology into important compilers
  - Widely used open source systems & vendor compilers
  - Model implementations to guide commercial
  - Ph.D. students into industrial compiler groups



## Students & Papers

---

### Students

- Alex Grosul, Ph.D. (Search techniques for sequence finding) May 2005
- Todd Waterman, Ph.D. (Adaptive inline substitution), January 2006

### Publications

- “ACME: Adaptive Compilation Made Easy”, *ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, June 2005 (Cooper, Grosul, Harvey, Reeves, Subramanian, & Torczon)
- “Improved Passive Splitting”, *Int'l Conference on Programming Languages and Compilers*, June 2005 (Cooper & Eckhardt)
- “Revisiting Graph-coloring Register Allocation: A Study of the Chaitin-Briggs and Callahan-Koblenz Algorithms”, *18<sup>th</sup> International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, October 2005 (Cooper, Dasgupta, Eckhardt)
- “Adaptive Inlining”, *in review*
- Plus, two Ph.D. theses