Automated Empirical Optimization of High Performance Floating Point Kernels

R. Clint Whaley
whaley@cs.utsa.edu
www.cs.utsa.edu/~whaley

University of Texas at San Antonio
Department of Computer Science

February 19, 2013
Outline of Talk

1. Personal Background
2. Research philosophy & contributing with open source software
3. Introduction to my research
4. Motivation
5. Computational Optimization Approach
6. Intro to ATLAS
7. Intro to iFKO
8. Outline of research areas
9. Examples of ATLAS-style research
10. Examples of iFKO-style research
11. Current work
12. Future Directions
13. Funding History
14. Further info / Q&A
Personal Background

- **90-91**: *undergraduate researcher at ORNL*, distributed memory parallelization of nuclear collision models
- **91-94**: *GRA, UTK*, distributed memory parallelization with ScaLAPACK and BLACS
- **May 94**: M.S. in dist. mem. \(\parallel\) (advisor: Jack Dongarra)
- **94-99**: *Research Associate, UTK*: ScaLAPACK, HPL, HPF, clusters \(\rightarrow\) ATLAS
- **99-01**: *Senior Research Associate, UTK*: ATLAS
  \(\rightarrow\) Weakness: backend compilation
- **02-04**: *GRA, FSU*: empirical backend compilation (iFKO)
- **Dec 04**: Ph.D. in empirical compilation (advisor: David Whalley)
- **05-12**: Assistant Professor, UTSA
- **12-present**: Associate Professor wt tenure, UTSA
Research Philosophy

- I believe computer science should be approached as a science
- Reproducible experiments are a hallmark of science
- Papers w/o assoc software prevent true reproduction/verification

⇒ If software made usable and stable, software frameworks developed by CS researchers can make **strong** contributions to most fields in mathematics, science and engineering

My research goal is to maximize real-world impact:

- Always start research from real-world problem
- For every successful advance, embody it in open frameworks for all to use
- In this way, I have been able to have a large impact both inside and outside my field
An Introduction to my Research
My research oriented to increasing performance of fp computations

As student, my research embodied in widely-used distributed memory parallel linear algebra libraries:

- One of the core developers of ScaLAPACK (Scalable Linear Algebra PACKage), still in use today
- Main developer of its communication layer, the BLACS
- Contributer to HPL benchmark (Petitet)

As an independent researcher, main focus is on automated empirical optimization of computational kerns

- ATLAS (Automatically Tuned Linear Algebra Software) used by hundreds of thousands in almost every field worldwide
- iFKO (iterative Floating Point Kernel Optimizer): a way to extend my research’s impact beyond linear algebra to arbitrary HPC kernels
Ultimate goal of research is to make extensive hand-tuning unnecessary for HPC kernel production

- For many operations, no such thing as “enough” compute power
- Therefore, need to extract near peak performance even as hardware advances according to Moore’s Law
- Achieving near-optimal performance is tedious, time consuming, and requires expertise in many fields
- Such optimization is neither portable nor persistent

⇒ State-of-the-art approach uses automated empirical optimization
Key Idea
Probe machine empirically, keep transforms resulting in measurable improvements; Automate process so code is auto-tuned w/o advanced knowledge of architecture.

Goal
Optimized, portable kernel (wt associated library) available in hours rather than months or years (or never).

Package approaches
- ATLAS: use search-driven parameterization, multi-impl, & source gen to tune given kernels to arbitrary architectures
- iFKO: use search-driven empirical compilation to tune arbitrary kernels to a handful of select architectures
What is ATLAS?

- One of the pioneering packages in empirical tuning for HPC
- Provides high performance dense linear algebra routines:
  - BLAS (matmul, trislv), some LAPACK (linslv, eigen, SVD, etc.)
- Automatically adapts itself to differing architectures using empirical techniques
- Used in: Scientific simulation (physics, chemistry, biology, astronomy, engineering, math), medical imaging, machine learning; almost all supercomputers; built-in to many OSes (OS X, most Linux), used by many PSEs (Maple, MAGMA, Mathematica, Matlab, Octave), large numbers of packages (GSL, HPL, SciPy, R, come compilers, etc), and industry (IBM, Google, mobile computing oil & gas exploration, financial modeling and analysis, etc.)
iterative Floating Point Kernel Optimizer

iFKO is an iterative and empirical compiler framework

### iFKO composed of:
1. A collection of search drivers,
2. a compiler specialized for empirical floating point kernel optimization (FKO)
   - Specialized in analysis, HIL, type & flexibility of transforms

### Key design decisions:
- Iterative & empirical for auto-adaptation
- Transforms done at backend, allowing arch exploitation (eg. SIMD vect, CISC inst formats)
- Kernel annotation markup
- Narrow & deep focus

---

![Diagram of iFKO process](chart.png)
Outline of Research
Types of research arising from ATLAS and iFKO

ATLAS
- systems
- numerical analysis/scientific computing
- architectural exploitation for algorithmic development
- shared-memory parallelism & scalability

iFKO
- autotuning known kernels
- compiler transformation research
- new algorithms based on arbitrary kernels
Why has our parallel scalability dropped thru the floor?

- Found OS scheduling destroyed scaling due to thread spawn even on an $O(N^3)$ algorithm!
- Presented thread spawning algorithm that fixes the problem using thread affinity
- Perf of GEMM now scales with $p$; as much as 40% improvement in apps like factorizations (Linux/Windows)

→ This became ATLAS’s standard thread spawn strategy

Why can’t invert triangular mat as part of TRSM?
⇒ Pollutes back err wt cond number (textbook, G. W. Stewart)

But this got us interested in error control

- Dot prod basis of GEMM, and thus controls err of most LA
- Surveyed known err bounds for dot product, showed alg’s actual error follow sorting of bounds
- Presented a new class of alg called superblock subsuming known
- Superblock allows explicit trade-off bet storage/perf and err cont
⇒ Changed ATLAS’s GEMM for improved error control

How can we parallelize panel factorization?

- Used cache coherence mechanism to make parallel communication run at hardware speed
- Moved par overheads into inner loop to enable cache reuse
- Got superlinear speedups on algs where speedup is usually $\approx 2x$
- Presented mathematical trick leading to 100x speedups
- Documented allocation algorithm to avoid drastic perf loss on AMD’s HT-assist machines

$\rightarrow$ This became ATLAS’s standard panel factorization method

Can an empirical compiler beat hand-tuning?

- 7 *fundamental* (tunable) optimizations (loop unroll, acc exp, pref, NTW, etc)
- 9 *repeatable* optimizations (reg alloc, copy prop, etc.)
- Applied to simple single-loop kernels
- Beat hand-tuned (ATLAS & MKL) on all but a few kernels
  → Lost when we could not auto-vectorize past branches
Ongoing Work
Some of our present ATLAS and iFKO investigations

**ATLAS: How to scale fact. to $O(100)$ shared mem cores?**

- Rakib Hasan (PhD candidate): *parallelism and scaling*
- MKL/ACML facts better scaling than lapack/ATLAS
- There are *many* parallelization methods known for factorizations
- Which are the important ones in achieving these types of scaling?
  → have proto tying vendor libs for large probs, wins for small-med

**iFKO: How can we vectorize past branches**

- M. Sujon (PhD cand) & Qing Yi: *mainstream compiler research*
- Use novel technique when branch is strongly directional
- Use known techniques (redundant computation, etc.) when not

**ATLAS&iFKO: Novel algorithms using arbitrary kerns**

- Majedul Sujon & Rakib Hasan
- Increased performance (panel/Hess) via BLAS blending
Future Directions

**short term**

- ATLAS GEMM & threading redesign
- Extensions of PCA & related kernels to enable scaling even on small & medium-sized kernels as well as large
- Expansion of loop handling; use iFKO to tune more complex kernels

**longer term**

- Increased handling of scale, including distributed memory & accelerators (grant under review with PLASMA group to explore)
- Adapting to ongoing architectural trends (power/ARM, etc)
- Applying iFKO to new areas and apps; new algs using new kerns
- Release iFKO for general use
- Testing limits of empirical compilation (how many search types are required? How many backends needed? Can improve timer generation?, etc.)
Funding while at UTSA
Grant summary and active funding

Between 2005-2012 brought in more than $3.3 million:

- Almost $2.5 million as a sole investigator
- Obtained funding from: NSF, DoD, DoE and industry
- My software used by all these groups, which improves funding chances

Active, sole-PI grants

- NSF CAREER award, OCI-1149303, “Empirical Tuning for Extreme Scale”, $538,145, 03/01/12 – 02/28/17.
- SiCortex Research Gift, $10,000 (donated 2007)
sole-PI grants

- R. Clint Whaley, NSF REU supplement for CNS-0551504. 04/22/08 - 02/28/09, $12,000.
- R. Clint Whaley, “CRI, Community Resource Development: ATLAS Support and Development”, NSF CRI award, CNS-0551504. 03/01/06 - 02/28/09, $100,000.
Funding while at UTSA
Expired or soon-to-be-expired co-PI funding

**co-PI grants**

- Qing Yi, Daniel J. Quinlan and R. Clint Whaley, “A Multi-Language Environment For Programmable Code Optimization and Empirical Tuning”, DOE Office of Science award DE-SC0001770, 09/15/09 - 09/14/12, $360,000.

- Qing Yi, Daniel J. Quinlan and R. Clint Whaley, “Programmable Code Optimization and Empirical Tuning For High-end Computing”, NSF award CCF-0833203, 09/01/08 - 09/31/10, $462,000.
Further Information and Q&A

- Homepage: www.cs.utsa.edu/~whaley
- Publications: www.cs.utsa.edu/~whaley/papers.html
- ATLAS: math-atlas.sourceforge.net
- BLAS: www.netlib.org/blas
- LAPACK: www.netlib.org/lapack
- BLACS: www.netlib.org/blacs
- ScaLAPACK: www.netlib.org/scalapack/scalapack_home.html
## Courses Taught

### Graduate Courses
1. CS 5513 Computer Architecture
2. CS 6463 Fundamentals of High Performance Optimization
3. CS 6643 Parallel Processing

### Undergraduate Courses
1. CS 1073 Intro to Comp Prog for Scientific Apps
2. CS 1713 Intro to Comp Prog II
3. CS 3853 Computer Architecture
4. CS 4823 Parallel Programming
## Completed Thesis and Dissertations Supervised


## Completed Master’s Projects Supervised

1. Chad Zalkin, “SSE Code Generation for General Matrix Matrix Multiply“, Master’s project; successfully defended April, 2011.

## Master’s Co-advised with Qing Yi

- Mike Stiles
- Jichi Gua
- Faizur Rahman
- Ziaul Haque
- Akshatha Bhat


R. Clint Whaley, “Empirically Tuning LAPACK’s Blocking Factor for Increased Performance”, *Proceedings of the International Multiconference on Computer Science and Information Technology* (Computer Aspects of Numerical Algorithms), pages 303-310, Wisla, Poland, October 20-22, 2008. (*this was a refereed workshop, GSC: 13*)


Selected Other Publications

Invited Papers


Refereed Workshop Papers

1 Josh Magee, Qing Yi, and R. Clint Whaley, “Automated Timer Generation for Empirical Tuning”, Fourth Workshop on Statistical and Machine learning approaches to ARchitecture and compilaTion (SMART’10), January 24, 2010, Pisa, Italy. URL: http://ctuning.org/workshop-smart10

Books & Book Chapters


(b). Traditional Approach: Library Usage

Approach:
1. Isolate time-critical sections of code, define & agree on API (e.g., BLAS)
2. Get experts in required fields (type of computation, hardware platform and programming environment) to optimize kernels

Problems:
- Demand for experts far outstrips supply
- Even wt experts, by time lib fully optimized, target architecture well on its way towards obsolescence
- If an API is not perceived as general enough, will not be supported
Shortcomings of Compilation

Tradition compilers are not designed for this level of optimization:

- Opt general code to this degree counterproductive:
  - Comp dev & runtime time would explode
  - Would make no difference for 90% of code
- Many ISAs far from hardware (OOE, reg renaming, dynamic scheduling, etc) ⇒ effects of compiler transformation unpredictable
- Resource allocation is intractable approached a priori
- Modeling machine & operation at this level is intractable
  - Interaction between all PEs and shared sources (all lvels of cache & pipelines of all relevant PEs, along wt their usage restrictions)
- Differing kernel requires specialized models
  - How many/type of structures accessed
  - Usage pattern (use/set) of all operands
  - Context of operation (eg., in or out of cache)
Panel factorization:

- \( N \times N_b \) data elements
- \( O(N \cdot N_b^2) \) computations

Ideas behind PCA:

1. Use cache-coherence for hardware-speed syncs
2. Move parallel overheads into inner loop to exploit reuse
3. Copy data to local caches using standard distribution (eg., block)

→ fit entire prob in collective cache