Programming Exascale Supercomputers

Mary Hall

SC12

November 2012

*** This work has been partially sponsored by DOE SciDAC, DOE Office of Science, the National Science Foundation, DARPA and Intel Corporation.**

Three Goals for Talk

1. Introduction and personal history
2. Setting expectations from a 20 year career retrospective
3. Key issues and opportunities in future programming models

Personal History

- B.A. Computer Science and Mathematical Sciences, Rice University, 1985
  - Planned to go on to business school to be an engineering manager
- Ph.D. Computer Science, Rice University, 1991
  - Had planned to get a Masters degree
- Research scientist positions at Rice, 1991-1992, and Stanford, 1992-1995
- Visiting Professor, Caltech, 1995-1996
- Research faculty (USC) and project leader (USC/ISI), 1996-2008
- Professor, University of Utah, since 2008
- Personal:
  - Youngest of five, native Texan, mother taught math and computer literacy, father was a journalist
  - Married 25 years, two daughters 12 and 16

Research Timeline

2005-present: Auto-tuning compiler technology (memory hierarchy, multimedia extensions, multi-cores and GPUs)

2007-present: Reports on compiler, exascale software and archiving research directions

2001-2006: Compilation for multimedia extensions (DIVA, AltiVec and SSE)

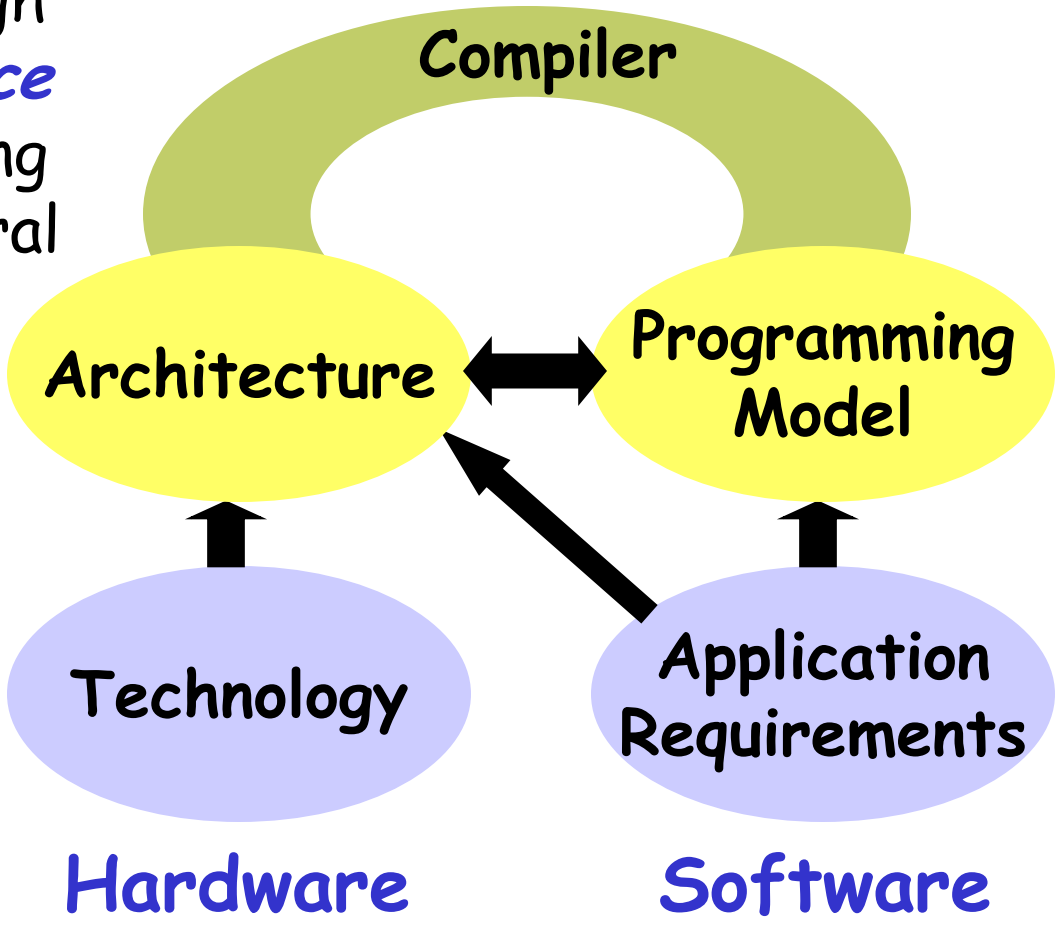
1998-2005: DIVA Processing-in-memory system architecture (HP Itanium-2 architecture)

1998-2004: DEFACTO design environment for FPGAs (C to VHDL)

1986-2000: Interprocedural Optimization and Automatic Parallelization, Rice D System and Stanford SUIF Compiler

Introduction: What Drives the Research

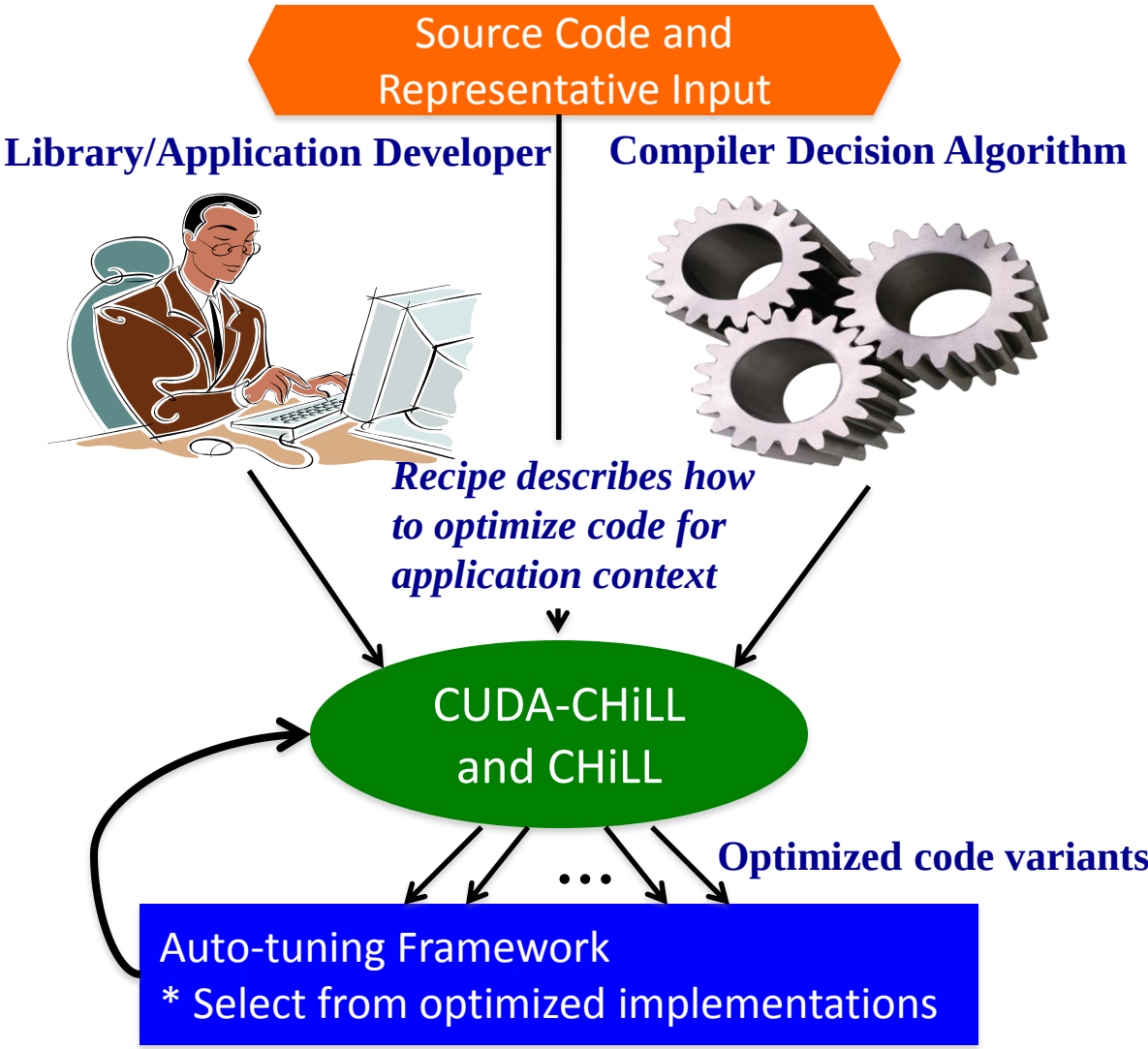
Achieve high *performance* by exploiting architectural features …



… while freeing programmers from managing low-level details (*productivity*).

Compiler and Autotuning Technology

- Increase compiler effectiveness through ***autotuning*** and ***specialization***
- Provide high-level interface to code generation (***recipe***) for library or application developer to suggest optimization
- Bridge domain decomposition and single-socket locality and parallelism optimization
- Autotuning for different optimization goals: performance, energy, reliability



Current Projects

- X-TUNE from DOE X-Stack program
  - Design autotuning framework to produce high-performance, energy-efficient, reliable software for the exascale software stack of 2018
  - Utah leads in collaboration with Argonne and Berkeley National Laboratories and USC
- Osprey from DARPA PERFECT program
  - Design an energy-efficient, high-performance embedded system targeting signal processing applications
  - Utah leads autotuning software system technology in collaboration with Nvidia (overall lead), Virginia Tech and others
- SUPER, a DOE SciDAC Institute
  - Develop programming system technology for high-performance, energy-efficient, reliable scientific applications over the next 5 years
  - Utah leads performance optimization area, in collaboration with USC (overall lead), University of Maryland, University of North Carolina, University of Oregon, University of Tennessee, University of Texas-El Paso, Argonne, Berkeley, Livermore and Oak Ridge National Laboratories
- NSF Projects
  - A Compiler-Based Autotuning Framework for Many-Core Code Generation
  - Hardware/Software Management of Large Multi-Core Memory Hierarchies
  - Correctness Verification Tools for Extreme Scale Hybrid Computing

Top 10 Reasons to Work in this Area

1. Algorithms and abstractions in compilers are mathematically and logically elegant.
2. The concrete realization of these algorithms and abstractions in working, faster code is tangible.
3. Tracking current and future hardware is cool.
4. Impacting science is rewarding.
5. Working with scientists offers a human element.
6. We work on problems critical to the nation's and earth's future.
7. We get to work with the absolute best people across a bunch of fields.
8. We get to use the absolute best hardware, including supercomputers.
9. The area is sufficiently broad that all sorts of different skill sets and backgrounds are valuable.
10. There are short-term and long-term benefits, so new students can impact practice while setting up for long-term research.

Getting to Exascale

- Before 2020, exascale systems will be able to compute a quintillion operations per second!
- Scientific simulation will continue to push on system requirements:
  - To increase the precision of the result
  - To get to an answer sooner (e.g., climate modeling, disaster modeling)
- The U.S. will continue to acquire systems of increasing scale
  - For the above reasons
  - And to maintain competitiveness
- A similar phenomenon in commodity machines
  - More, faster, cheaper

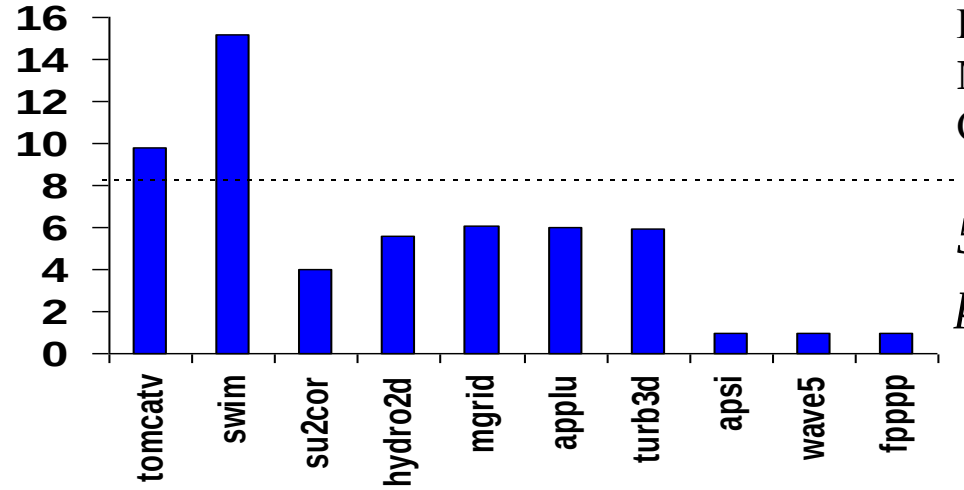
Exascale Challenges Will Force Change in How We Write Software

- Exascale architectures will be fundamentally different
  - Power management becomes fundamental
  - Reliability (h/w and s/w) increasingly a concern
  - Memory reduction to .01 bytes/flop
  - Hierarchical, heterogeneous
- Basic rethinking of software
  - Express and manage locality and parallelism for ~billion threads
  - Create/support applications that are prepared for new hardware (underlying tools map to h/w details)
  - Manage power and resilience
    - Locality is a big part of power/energy
    - Resilience should leverage abstraction changes

"Software Challenges in Extreme Scale Systems," V. Sarkar, B. Harrod and A. Snavely, SciDAC 2009, June, 2009.
Summary of results from a DARPA study entitled, "Exascale Software Study," June 2008 through Feb. 2009.

Can programming language and compiler technology automatically solve the programming challenges?

Previous Work in Automatic Parallelization



8-processor Speedups--Digital AlphaServer 8400

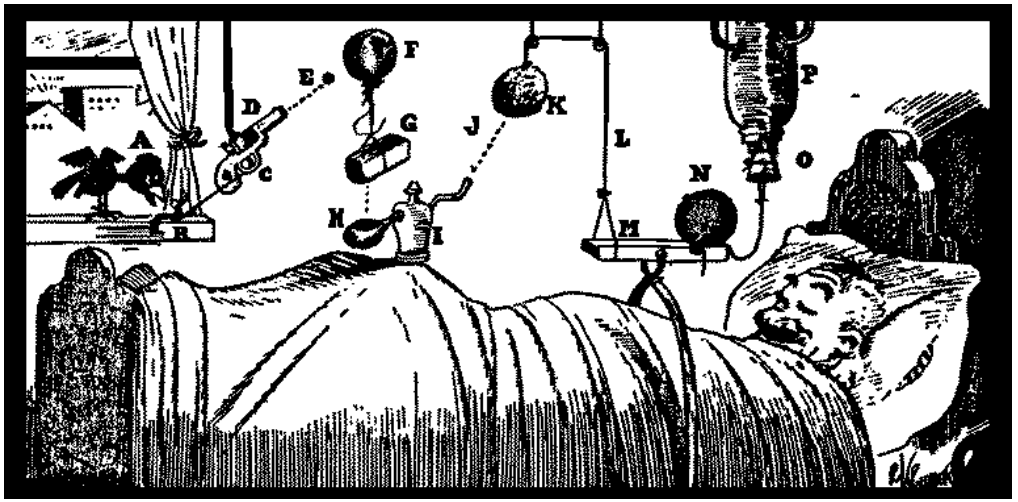
From Hall et al, "Maximizing Multiprocessor Performance with the SUIF Compiler", IEEE Computer, Dec. 1996.

50% higher Specfp95 ratio than previously reported

- Old approaches to compilers mapping parallelism
  - Limited to loops and array computations
  - Difficult to find sufficient granularity (parallel work between synchronization)
  - Very restricted mapping strategy
  - Success but from fragile, complex software

1990s View

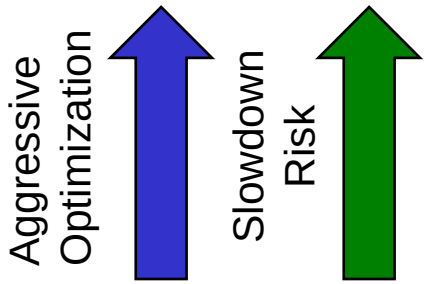
- Programmer writes code at high level
  - Much or all complexity managed by compiler



- But doing everything in the compiler is hard!
  - Expert programmers have knowledge that should be exploited.
  - Compiler development cycle is slow.
  - Application scientists will find expedient solutions.

Historical Organization of Compilers, Users' Perspective

- What's not working
  - Optimizations often applied in isolation, but significant interactions as architectures get more complex
  - Static compilers must anticipate all possible execution environments
  - Potential to slow code down
  - Users write low-level code to get around compiler which makes things even worse



Bottom line: *Known compiler techniques capable of much better performance than they are delivering, but solutions don't generalize across applications and complexity of system is difficult to maintain.*

Future Parallel Programming

- It seems clear that for the next decade architectures will continue to get more complex, and achieving high performance will get harder.
- Most people in the research community agree that different kinds of parallel programmers will be important to the future of computing.
  - Programmers that understand how to write software, but are naïve about parallelization and mapping to architecture (Joe programmers)
  - Programmers that are knowledgeable about parallelization, and mapping to architecture, so can achieve high performance (Stephanie programmers)
  - Intel/Microsoft say there are three kinds (Mort, Elvis and Einstein)
- Programming abstractions will get a whole lot better by supporting specific users.

A Broader View in 2012

Thanks to exascale reports and workshops

- Multiresolution programming systems for different users
  - Joe/Stephanie/Doug **[Pingali, UT]**
  - Elvis/Mort/Einstein **[Intel]**
- Specialization simplifies and improves efficiency
  - Target specific user needs with domain-specific languages/libraries
  - Customize libraries for application needs and execution context
- Interface to programmers and runtime/hardware
  - Seamless integration of compiler with programmer guidance and dynamic feedback from runtime
- Toolkits rather than monolithic systems
  - Layers support different user capability
  - Collaborative ecosystem
- Virtualization (over-decomposition)
  - Hierarchical, or flat but construct hierarchy when applicable?

What is Autotuning?

- Definition:
  - Automatically generate a "search space" of possible implementations of a computation
    - A ***code variant*** represents a unique implementation of a computation, among many
    - A ***parameter*** represents a discrete set of values that govern code generation or execution of a variant
  - Measure execution time and compare
  - Select the best-performing implementation (for exascale, tradeoff between performance/energy/reliability)
- Key Issues:
  - Identifying the search space
  - Pruning the search space to manage costs
  - Off-line vs. on-line search

Three Types of Autotuning Systems

a. Autotuning libraries
   - Library that encapsulates knowledge of its performance under different execution environments
   - Dense linear algebra: **ATLAS, PhiPAC**
   - Sparse linear algebra: **OSKI**
   - Signal processing: **SPIRAL, FFTW**

Current/ Future Work

b. Application-specific autotuning
   - **Active Harmony** provides parallel rank order search for tunable parameters and variants
   - **Sequoia** and **PetaBricks** provide language mechanism for expressing tunable parameters and variants

c. Compiler-based autotuning (this talk!)
   - Other examples: Saday et al., Swany et al., Eignenmann et al.
   - Related concepts: iterative compilation, continuous compilation, learning-based compilation

Differences: Present and Future

Who/What	Present	Future
Application programmer writes	A single implementation of a computation, or perhaps a few guarded by run-time tests	A compact search space of parameterized variants
Library developer writes	Numerous implementations of a computation, guarded by run-time tests	A compact search space of parameterized variants
Compiler generates	A single implementation of a computation, or perhaps a few guarded by run-time tests	A compact search space of parameterized variants
System executes	Compiled code as provided	A synthesis of variants and their parameter values meeting optimization criteria

Compiler-Based Autotuning: My Philosophy

- ***Foundational Concepts***
  - Identify search space through a high-level description that captures a large space of possible implementations
  - Prune space through compiler domain knowledge and architecture features
  - Provide access to programmers with **transformation recipes** (controversial)
  - Uses source-to-source transformation for portability, and to leverage vendor code generation
  - Requires ***restructuring of the compiler***
- ***Impact***
  - Developers write less and higher-level code, more automatically generated/managed
  - Systematic characterization and analysis

Transformation Recipes for Autotuning: Incorporate the Best Ideas from Manual Tuning

Nvidia GTX-280 implementation
Mostly corresponds to CUBLAS 2.x and Volkov's SC08 paper

```
1 tile_by_index({"i","j"},{TI,TJ},{l1_control="ii",l2_control="jj"},
                {"ii","jj","i","j"})
2 tile_by_index({"k"},{TK},{l1_control="kk"},
                {"ii","jj","kk","i","j","k"},strided)
3 tile_by_index({"i"},{TJ},{l1_control="tt",l1_tile="t"},
                {"ii","jj","kk","t","tt","j","k"})
4 cudaize("mm_GPU",{a=N*N,b=N*N,c=N*N},
          {block={"ii","jj"},thread={"t","tt"}})
5 copy_to_shared("tx","b",-16)
6 copy_to_registers("kk","c")
7 copy_to_texture("b")
8 unroll_to_depth(2)
```

Nvidia TC2050 Fermi implementation
Mostly corresponds to CUBLAS 3.2 and MAGMA

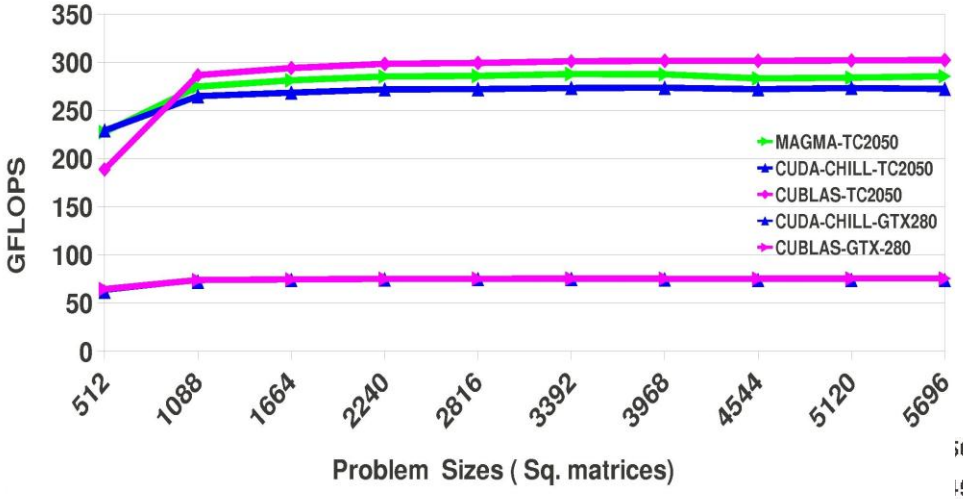
```
1  tile_by_index({"i","j"},{TI,TJ},{l1_control="ii",l2_control="jj"},
                 {"ii","jj","i","j"})
2  tile_by_index({"k"},{TK},{l1_control="kk"},
                 {"ii","jj","kk","i","j","k"},strided)
3  tile_by_index({"i"},{TK},{l1_control="t",l1_tile="tt"},
                 {"ii","jj","kk","tt","t","j","k"})
4  tile_by_index({"j"},{TK},{l1_control="s",l1_tile="ss"},
                 {"ii","jj","kk","tt","t","ss","s","k"})
5  cudaize("mm_GPU",{a=N*N,b=N*N,c=N*N},
           {block={"ii","jj"},thread={"tt","ss"}})
6  copy_to_shared("tx","b",-16)
7  copy_to_texture("b")
8  copy_to_shared("tx","a",-16)
9  copy_to_texture("a")
10 copy_to_registers("kk","c")
11 unroll_to_depth(2)
```

Different computation decomposition leads to additional tile command

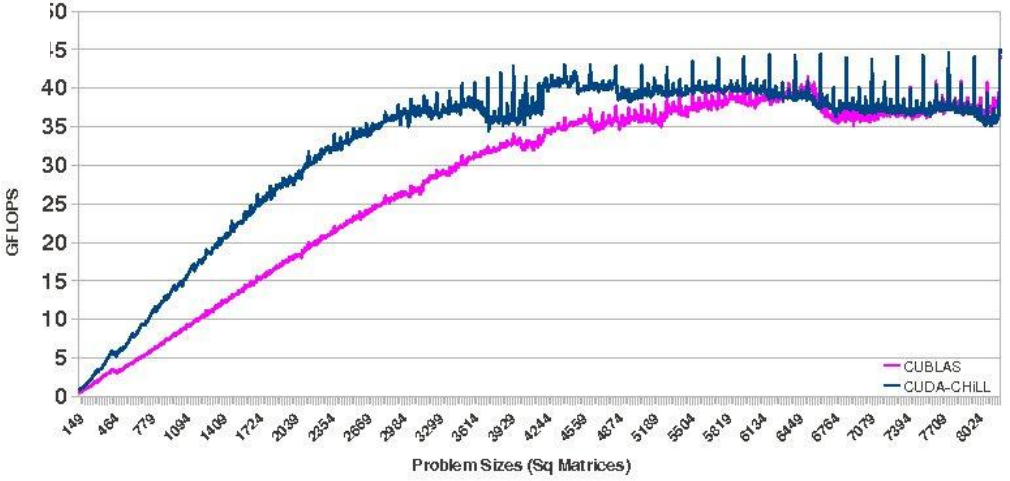
a in shared memory, both a and b are read through texture memory

Compiler + Autotuning can yield comparable and even better performance than manually-tuned libraries

Matrix-Matrix Multiply (dgemm)



Matrix-Vector Multiply (sgemv)

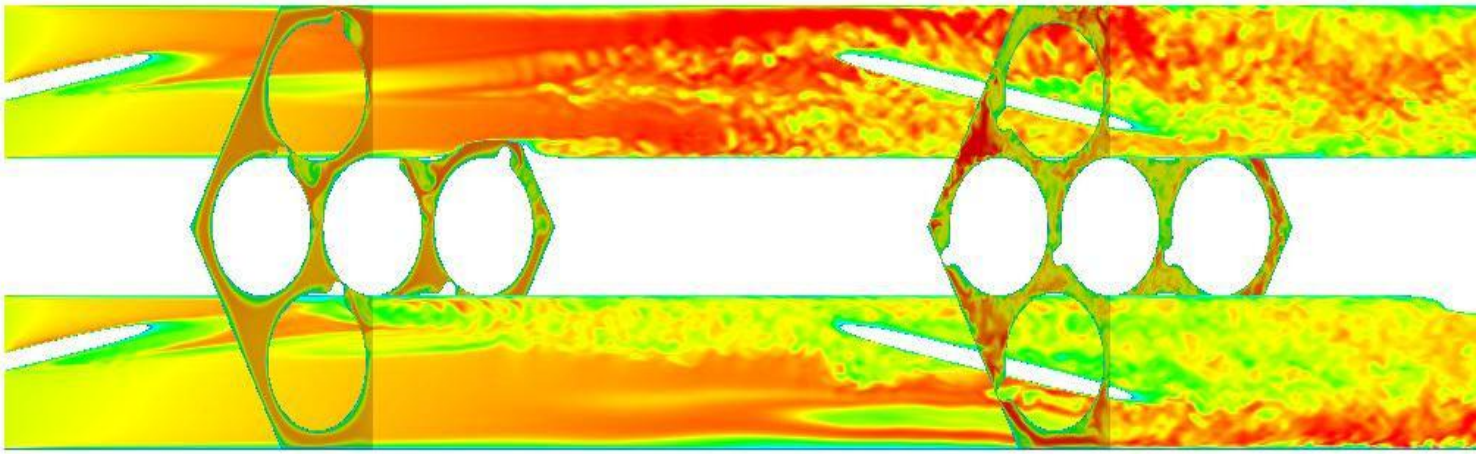


- Performance comparison with CUBLAS 3.2

"Autotuning, Code Generation and Optimizing Compiler Technology For GPUs," M. Khan, PhD Dissertation, University of Southern California, May 2012.

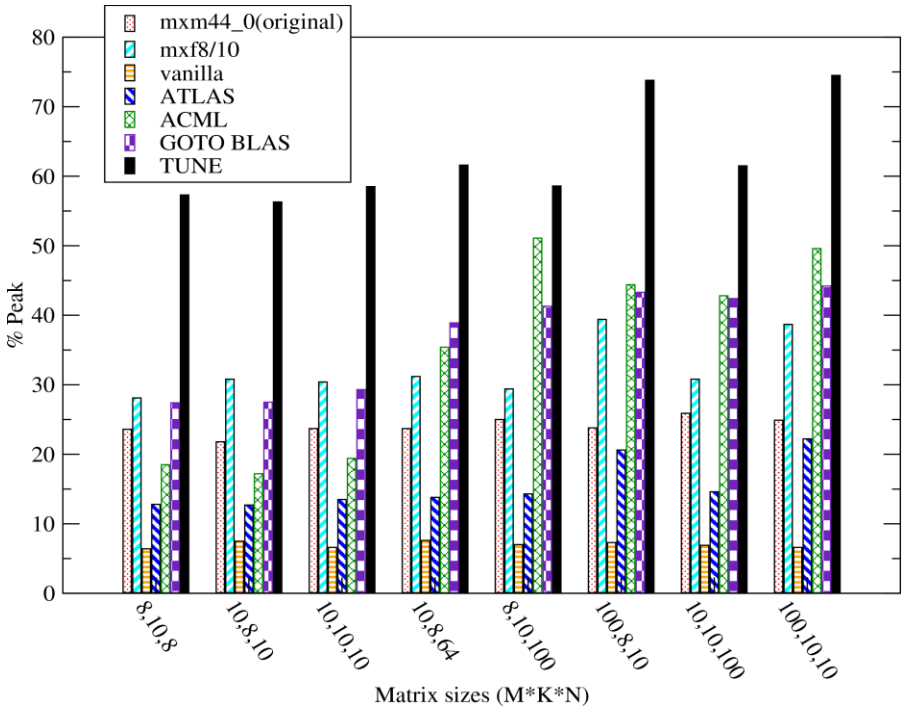
Autotuning and Specialization for Nek5000

Spectral element code: turbulence in wire-wrapped subassemblies



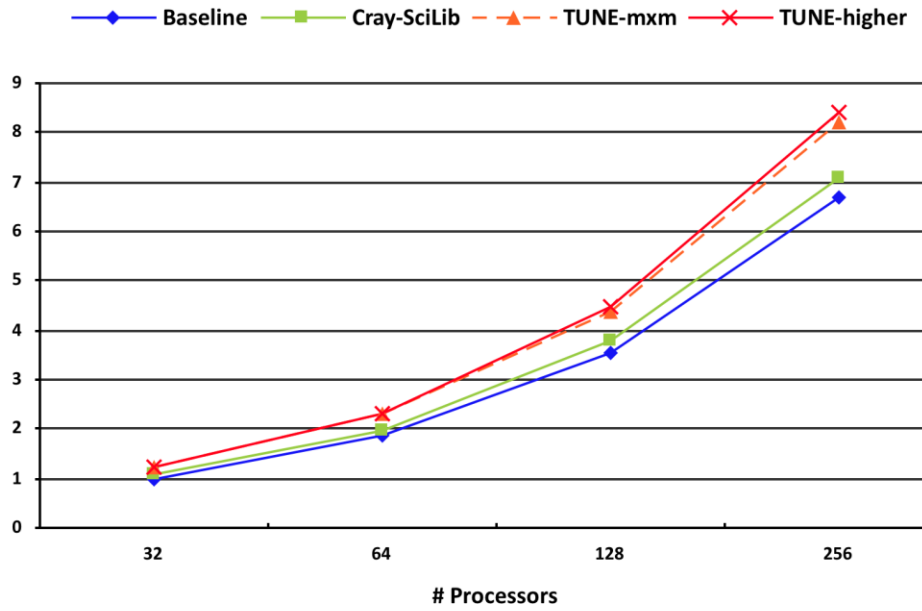
- Applications: nuclear energy, astrophysics, ocean modeling, combustion, bio fluids, ...
- Scales to P > 10,000 (Cray XT5, BG/P)
- > 75% of time spent on manually optimized mxm
  - matrix multiply of very small, rectangular matrices
  - matrix sizes remain the same for different problem sizes

nek5000: Automatically-Generated BLAS Code is Faster than Manually-Tuned Libraries



Library:
2.2X speedup for specialized DGEMM

Application:
26% performance gain on Jaguar



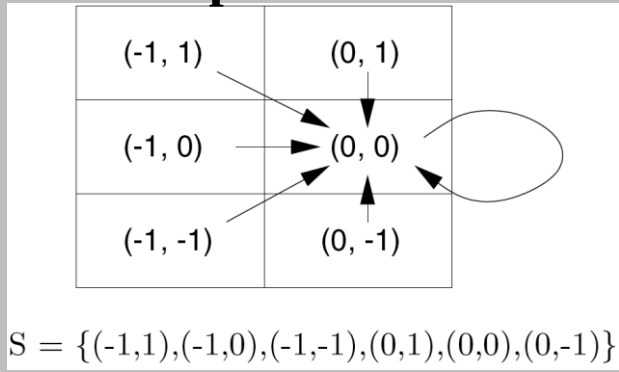
"Autotuning and Specialization: Speeding up Nek5000 with Compiler Technology," J. Shin, M. W. Hall, J. Chame, C. Chen, P. Fischer, P. D. Hovland, International Conference on Supercomputing, June, 2010.

Application example from PERI: SMG2000 Optimization

- Semi-coarsening multigrid on structured grids
  - Residual computation contains sparse matrix-vector multiply bottleneck, expressed in 4-deep loop nest
  - Key computation identified by HPCToolkit

```
for si = 0 to NS-1
  for k = 0 to NZ-1
    for j = 0 to NY-1
      for i = 0 to NX-1
        r[i + j*JR + k*KR] -=
            A[i + j*JA + k*KA + SA[si]]
          * x[i + j*JX + k*KX + Sx[si]]
```

2D 6-point Stencil



$S = \{(-1,1),(-1,0),(-1,-1),(0,1),(0,0),(0,-1)\}$

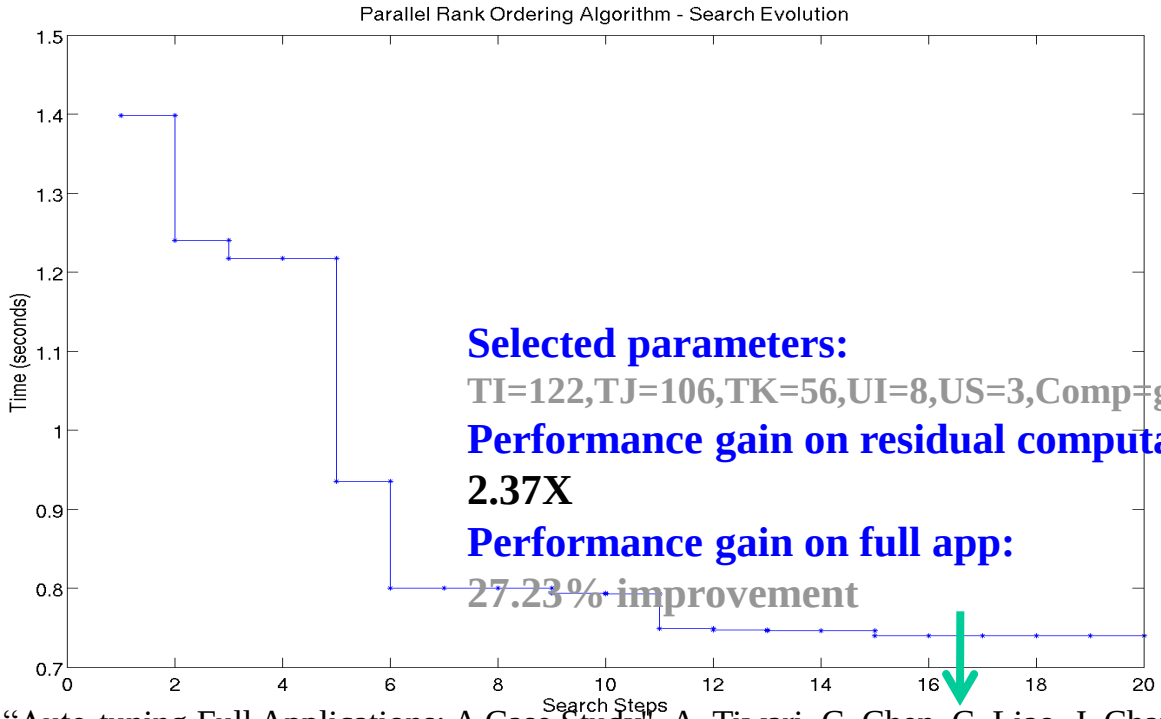
Parallel Heuristic-Based Search for SMG2000 Converges Rapidly

Outlined Code (from ROSE outliner)

```
for (si = 0; si < stencil_size; si++)
  for (kk = 0; kk < hypre__mz; kk++)
    for (jj = 0; jj < hypre__my; jj++)
      for (ii = 0; ii < hypre__mx; ii++)
        rp[((ri+ii)+(jj*hypre__sy3))+(kk*hypre__sz3)] -=
          ((Ap_0[((ii+(jj*hypre__sy1))+ (kk*hypre__sz1))+
           (((A->data_indices)[i])[si])])*
           (xp_0[((ii+(jj*hypre__sy2))+(kk*hypre__sz2))+(( *dxp_s)[si])]));
```

CHiLL Transformation Recipe

permute([2,3,1,4])
tile(0,4,TI)
tile(0,3,TJ)
tile(0,3,TK)
unroll(0,6,US)
unroll(0,7,UI)



Selected parameters:
TI=122,TJ=106,TK=56,UI=8,US=3,Comp=gcc
Performance gain on residual computation:
2.37X
Performance gain on full app:
27.23% improvement

Optimization search space has 581M points!

Parallel search (Active Harmony) evaluates 490 points, converges in 20 steps

"Auto-tuning Full Applications: A Case Study", A. Tiwari, C. Chen, C. Liao, J. Chame, J. Hollingsworth, M. Hall and D. Quinlan, International Journal of High Performance Computing Applications, 25(3):286-294, Aug. 2011.

Future: X-TUNE (DOE X-Stack)

A unified autotuning framework that seamlessly integrates programmer-directed and compiler-directed autotuning,

- Expert programmer and compiler work collaboratively to tune a code.
  - Unlike previous systems that place the burden on either programmer or compiler.
  - Provides access to compiler optimizations, offering expert programmers the control over optimization they so often desire.
- Design autotuning to be encapsulated in domain-specific tools
  - Enables less-sophisticated users of the software to reap the benefit of the expert programmers' efforts.
- Focus on Adaptive Mesh Refinement Multigrid (Combustion Co-Design Center,BoxLib,Chombo) and tensor contractions (TCE)

Summary: Autotuning Challenges

- **Conceptual:** Rethink the development process as a way of expressing a search space rather than a fixed implementation
  - What are the right abstractions to expose to programmer
  - Integrate into multiresolution system
- Navigating prohibitively large search space
  - Includes performance, power and reliability
  - Models and pruning are critical
  - Parallel search algorithms can be effective
  - Tuning multiple computations simultaneously still an open problem
- Managing overhead (performance, storage and energy)