X-TUNE
Autotuning for Exascale: Self-Tuning Software to Manage Heterogeneity

Mary Hall
May 28, 2014
Participants

• University of Utah
  • Mary Hall, Manu Shantharam, Protonu Basu, Axel Rivera, Bob Wheeler, Derrick Huth

• Lawrence Berkeley National Laboratory
  • Sam Williams, Lenny Oliker, Brian van Straalen

• Argonne National Laboratory
  • Paul Hovland, Sri Krishna Narayanan, Jeff Hammond, Prasanna Balaprakash, (Stefan Wild), Thomas Nelson (Colorado)

• USC/ISI
  • Jacqueline Chame
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
X-TUNE Goals

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 Geometric Multigrid (ExaCT, BoxLib, Chombo), Nekbone (CESAR) and tensor contractions (NWCHEM)
Existing software all implemented using ROSE AST.
X-TUNE Vision and Status Overlay

Overlay for joint funding, power/energy not part of X-TUNE, remainder in progress

Existing software all implemented using ROSE AST.
X-TUNE Approach at a Glance

• When available, start with manually-tuned code or work with developer of new code
  • What are the performance bottlenecks, inherent and on specific architectures?
  • What transformations are needed to target specific architectures?
  • What performance questions can be addressed by autotuning?

• Attempt to automate
  • Develop new transformations and required analysis and code generation support
  • Develop modeling and decision algorithms

• Collect application code from collaborators, Co-Design Centers and other DOE application teams
  • Generalize from experiments with manually-tuned code
Outline

• Technical Approach
  • Communication-Avoiding Geometric Multigrid (use case)
  • OCTOPI: Tensor Computations and Tensor Contraction (use case)
    • Modeling and Decision Algorithms

• Summary of Interactions

• Remainder
  • Comparison with state-of-the-art
Geometric Multigrid

- Multigrid solves elliptic PDEs in $O(N)$ computational complexity by using a hierarchical approach.

- As a result, the degree of Parallelism decreases exponentially...
  - N-way parallelism, $N/8$, $N/64$, … 1-way parallelism across the entire machine … $N/64$, $N/8$, $N$
  - This is major worry for exascale machines 1000’s of cores per node

- Geometric Multigrid (GMG) is specialization in which the operator (A) is simply a stencil on a structured grid (i.e. matrix-free)
miniGMG Benchmark

- **miniGMG proxies the MG solves in BoxLib/Chombo codes**
- Cubical domain decomposed among processes into **boxes**.
- Fine-grid box dimension is configurable.
  - smaller boxes mimic AMR MG challenges
  - fewer boxes per process can be used to mimic combustion code constraints.
- **operator** is configurable
  - 7pt variable coefficient **proxies LMC**
  - 7pt constant coefficient is simpler
  - 27pt/13pt high-order stencils are available.
- **smoother** in the v-cycle is configurable
  - Gauss Seidel, Red-Black (“GSRB”) = **proxies LMC**
  - Jacobi (mathematically weaker)
- **bottom solver** is configurable
  - multiple GSRB’s
  - Krylov solver like **BiCGStab**, CG, CA-BiCGStab, CA-CG, etc…
Compiler Optimization of miniGMG (Smooth)

**Optimization Using Known Transformations**
- Loop skew, permute and tiling to create a parallel wavefront

**New Domain-Specific Transformations**
- Loop fusion in presence of fusion-preventing dependences
- Eliminating temporaries
- Adding ghost zones (comm. avoiding) to Multigrid operators

**High-Performance OpenMP Code Generation**
- Vary parallelism (intra-box) for different box sizes.

**Optimizations Built into CHiLL**
- CHiLL = loop transformations and code generation

**Autotuner**

**CHiLL**

**Omega+**

**Codegen+**

**Smooth Variants**

---

[Logos of collaborating institutions]
Inter-Box Parallelism
Thread Configuration
<6,1>

Best parallel code generation strategy depends on box size and machine!

Nested Parallelism
Thread Configuration
<2,3>

Parallel Decomposition

Intra-Box Parallelism
Thread Configuration
<1,6>
Highlight – April 2014

Compiler Autotuning for Geometric Multigrid

- **Problem**
  - Geometric multi-grid (GMG), is one of the most popular methods for solving partial differential equations, but is very difficult to optimize on evolving CPU architectures

- **Solution**
  - Leverage communication-avoiding optimizations which reduce communication overhead
  - Apply CHiLL compiler technology, using a set of novel transformations to derive performance comparable to hand-written optimizations
  - Make the approach portable, via autotuning system that explores tradeoffs between reduced communication and increased computation, as well as tradeoffs in threading schemes

- **Recent results**
  - Improved overall multi-grid solve execution by over 4x on NERSC Edison vs. reference version (Basu et al., HIPC 2013 & WOSC 2013)
  - Improved smooth at finest level by over 4x - *CHiLL-generated code outperforms hand-tuned*
  - Demonstrated comparable performance for low-level OpenMP threads & higher level *Habanero C phasers*

- **IMPACT**
  - Achieves comparable performance to hand-tuned code without sacrificing programmer productivity
  - Demonstrates capability of compiler-directed autotuning, with broad impact on important numerical methods for the DOE Office of Science
Recent Work: Compiler Optimization of miniGMG (Smooth+Residual+Restrict)

- CHiLL can tune and generate the best implementation for a given combination of operator (7pt or 27pt) and smoother (Jacobi).
  - Fusion may include smooth+residual+restriction
  - Partial sums optimization reduces computation, exposes reuse in cache and registers, improves SIMD code generation
- This choice of optimization, ghost zone depth, and threading strategy is made for each box size at each level in a MG V-cycle.
Tensor Products and Tensor Contractions

• Develop autotuning strategy for tensor computations such as Nekbone (CESAR) and NWCHEM (SciDAC)
  • Express tensors in mathematical notation (borrowing from Build-to-Order BLAS)
  • Decision algorithm maps to CHiLL recipes
  • Use Orio to explore autotuning search space

• Builds on prior work for small matrix-multiply kernels in Nek5000

• Leverages and integrates existing tools
Example: Spectral Element Method from nek5000/nekbone (CESAR)

\[ C = A \otimes B u \]

- A and B are square matrices
- \( u \) is a component vector
- In 2-d, \( C \) can be computed:

\[
c_{i,j} = \sum_{l} \sum_{k} a_{j,l} b_{i,k} u_{k,l}
\]

Order \( O(n^4) \)

Optimize by rewriting to the following:

\[ C = (A \otimes I)(I \otimes B)u \]

Partial Results: \( w = (I \otimes B)u \)

\[
w_{i,j} = \sum_{l} u_{i,l} b_{i,j}^T
\]

Order \( O(n^3) \), Can use DGEMM

Final Results: \( C = (A \otimes I)w \)

\[
c_{i,j} = \sum_{k} a_{i,k} w_{k,j}
\]

Order \( O(n^3) \), Can use DGEMM
Prior Work (TUNE): Matrix Multiply for Small Matrices using Autotuning and Specialization

Net result: 1.36 speedup in overall Nek5000 performance
Toward a High-Level Representation

• Prior work ignored tensor structure

```fortran
subroutine local_grad3(ur, us, ut, u, n, D, Dt)

Output: ur, us, ut
Input: u, n, D, Dt

real ur(0:n,0:n,0:n), us(0:n,0:n,0:n), ut(0:n,0:n,0:n)
real u(0:n,0:n,0:n), D(0:n,0:n), Dt(0:n,0:n)
integer e, i1, j1

m1 = n+1
m2 = m1*m1

call mxm(D, m1, u, m1, ur, m2)
do k=0, n
    call mxm(u(0,0,k), m1, Dt, m1, us(0,0,k), m1)
endo do
call mxm(u, m2, Dt, m1, ut, m1)

return
end
```
subroutine local_grad3(ur, us, ut, u, n, D)

c     Output: ur, us, ut               Input: u, n, D

real ur(0:n,0:n,0:n), us(0:n,0:n,0:n), ut(0:n,0:n,0:n)
real u(0:n,0:n,0:n), D(0:n,0:n)

UR_{ijk} = D_{il} U_{ljk}
US_{ijk} = D_{il} U_{ilk}
UT_{ijk} = D_{il} U_{ijl}

return
end
Experimental Framework

OCTOPI

High Level representation → Parser → C-Code → Decision Algorithm → Recipes Generated

User Input

CUDA, CUDA-CHiLL

Choose one recipe

Code Transformation

CUDA Code

Test new code

Best Optimized code

Finish?
Preliminary Results

- Speedup on GPU: 1.95x Nekbone and 9.94x NWChem
- Speedup over OMP: 1.01x Nekbone and 1.95x NWChem
- Speedup tuning OpenACC: 2.45x Nekbone and 13.68x NWChem

Experimental Setup:

- CPU: Intel i7 with 2.8 GHz
- RAM: 4 GB
- GPU: NVIDIA TESLA C2050 (FERMI)
- OS: LINUX Mint 13 64 bits
- COMPILER: PGI 14.3 with OpenACC support
- CUDA: Version 5.5
Model-Guided Compiler Decision Algorithms

• Goal: Automate the generation of code variants by compiler decision algorithms
• Models and analysis derive information about application
  • data dependences, data reuse, instruction counts, performance bounds
• Application and architecture information guide decisions
  • transformations, data placement
Modeling and Compiler Decision Algorithm

**PBound** analyzes code to derive reuse distance, and data footprint, integration with roofline

**Decision Algorithm** examines dependences and data reuse to generate a set of CHiLL transformation recipes

**CHiLL** performs transformations and code generation as specified by parameterized recipes

Maps PBound data structures to Decision Algorithm queries

Maps output from Decision Algorithm to CHiLL transformation recipes
Modeling and Decision Algorithm Status

• A new data reuse algorithm with more precise identification of reuse types added to PBound.
• A new locality decision algorithm was implemented and integrated with PBound.
• A new algorithm targeting GPUs was developed and integrated with PBound.
• NWCHEM
  - locality algorithm generates scripts that are used by CHiLL to generate code variants.
Interaction with X-Stack and Co-Design Projects

- Sam Williams - ExaCT and DEGAS
- Brian van Straalen – D-TEC
- Paul Hovland – CESAR
- Also interfacing with other X-Stack software
  - Orio/Active Harmony and OpenTuner planned
  - Habanero C
  - ROSE
- Additional code excerpts
  - TiDA (LBNL), S3D (LANL), HPGMG (LBNL)
Naïve (Inter-Box) Threading

Habanero Phasers

Naïve (Inter-Box) Threading

Phasers

OMP Spin-locks

Edison Phase(II), 12 cores per chip, 2 chips per node

Increasing number of threads inside a box

Widens gap between OMP Barrier and spin locks

Speedup over Naive Inter-Box Threading

<table>
<thead>
<tr>
<th></th>
<th>Phasers</th>
<th>OMP + Spin Locks</th>
<th>Naive Threading</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Raising Run-Time Level of Abstraction with Habanero C for miniGMG
Connection to State-of-the-Art (MPI+OpenMP)

• miniGMG uses MPI for domain decomposition and OpenMP for thread parallelism

• X-TUNE is agnostic about code outside its purview but introduces thread-level parallelism
  • Goal is to find right abstraction for compiler
  • Compatible with a variety of run-time systems

• Autotuning and communication-avoiding optimizations complementary to run-time and communication support
Papers and Presentations

• Papers

• Presentations
  • Tiling Dense and Sparse Computations for Parallelism and the Memory Hierarchy of GPUs, Mary Hall, SIAM Parallel Processing Symposium, Feb. 2014.

• Thesis and Dissertations
  • Other (PhD) students: Thomas Nelson (Colorado), Protonu Basu (Utah)