Automatic Generation of Algorithms and Data Structures for Geometric Multigrid

Harald Köstler, Sebastian Kuckuk
Siam Parallel Processing
02/21/2014
Introduction
Multigrid

- Goal: Solve a partial differential equation approximately by solving a discretized form of said PDE

\[ \Delta u = f \quad \text{in} \; \Omega \]
\[ u = 0 \quad \text{in} \; \partial \Omega \]

- An efficient method to solve such discretized PDEs in O(N) is multigrid

- Basic idea: Treat high frequency and low frequency errors separately by smoothing and solving for coarse grid representations respectively
Multigrid

Smoothing of High Frequency Errors

Coarsened Representation of Low Frequency Errors
Multigrid V-Cycle

- Smoothing
- Restriction
- Prolongation & Correction

Coarse Grid Solving
Our Scope

- Uniform grids

- Block-Structured grids
Goals

● **What do we want?**
  ● Efficient and robust multigrid solvers
  ● Performance portability
  ● Easy to adapt to new settings and concepts (e.g. hardware)
  ● Easy to extend
  ● …

● **Solutions?**
  ● Extensive Libraries?
  ● Optimizing by hand?
  ● Auto-Tuning?
Problem – Variance

- There is a lot of variance in the MG domain:
  - **Hardware**: CPU, GPU or both? Number of nodes, sockets and cores? Cache characteristics? Network characteristics?
  - **Software**: MPI, OpenMP or both? CUDA or OpenCL? Which version?
  - **MG components**: Cycle Type? Which smoother(s)? Which coarse grid solver? Which inter-Grid operators?
  - **MG parameters**: Relaxation? Number of smoothing steps? Other component dependent parameters?
  - **Optimizations**: Vectorization? (Software) Prefetching? Tiling? Temporal Blocking? Loop transformations?
  - **Problem description**: Which PDE? Which boundary conditions?
  - **Discretization**: Finite Differences, Finite Elements or Finite Volumes?
  - **Domain**: Uniform or block-structured? How to partition?
  - …
Possible Solutions

● What do we want?
  ● Efficient and robust multigrid solvers
  ● Performance portability
  ● Easy to adapt to new hardware
  ● Easy to extend
  ● …

● Solutions?
  ● Extensive Libraries?
  ● Optimizing by hand?
  ● Auto-Tuning?
  ● Code generation?
The ExaStencils Project
Project ExaStencils

- Sebastian Kuckuk
- Harald Köstler
- Ulrich Rüde

- Christian Schmitt
- Frank Hannig
- Jürgen Teich

- Hendrik Rittich
- Matthias Bolten

- Alexander Grebhahn
- Sven Apel

- Stefan Kronawitter
- Armin Größlinger
- Christian Lengauer
ExaStencils Vision

- Generate exa-scalable C++ code for GMG solvers from
  - a high-level problem description specified by domain experts and
  - a target hardware architecture specification
ExaStencils Overview

- DSL as intuitive interface to the user
- Automatic deduction of configuration if desired
- Prediction and Optimization of the configuration’s performance using SPL and LFA
- Code generation in Scala
- Automatic hardware-specific optimizations
ExaStencils Workflow

End-User

Domain Expert

Mathematician

Software Specialist

Hardware Expert

DSL Program

Discretization and Algorithm Selection

Software Component Selection via SPL

Polyhedral Optimization

Code Generation

Tuning towards Target Hardware

Exascale C++
ExaStencils Vision

- Generate exa-scalable C++ code from
  - a high-level problem description specified by domain experts and
  - a target hardware architecture specification

- Further visions: Provide different levels of abstraction that can be used as testing environments for
  - Mathematicians researching multigrid methods and components
  - Software Specialists researching programming languages, efficient communication strategies and program optimizations
  - Hardware Experts researching low-level and hardware-specific optimizations
State of the Project
Current State – LFA

- Convergence rate prediction for 2D/3D Jacobi, Gauss-Seidel, Red-Black Gauss-Seidel
- Hybrid GS and RBGS are predictable for small blocks as well
- Supports all cycle types
Current State – SPL

- First experiments in applying SPL techniques to our domain have been conducted [2]

<table>
<thead>
<tr>
<th>Heuristic</th>
<th># M</th>
<th># M [%]</th>
<th>Time [ms]</th>
<th>Faultrate distribution</th>
<th>$\mu \pm \sigma$ [%]</th>
<th>$\Delta$ [ms]</th>
<th>$\delta$ [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW</td>
<td>22</td>
<td>2.5</td>
<td>51 773.21</td>
<td></td>
<td>51.9 ± 59.3</td>
<td>93.81</td>
<td>8.3</td>
</tr>
<tr>
<td>PW</td>
<td>192</td>
<td>22.2</td>
<td>518 721.48</td>
<td></td>
<td>12.2 ± 14.7</td>
<td>184.80</td>
<td>16.3</td>
</tr>
<tr>
<td>HO</td>
<td>636</td>
<td>73.6</td>
<td>2 037 253.25</td>
<td></td>
<td>8.5 ± 17</td>
<td>2474.11</td>
<td>217.5</td>
</tr>
<tr>
<td>HS</td>
<td>864</td>
<td>100.0</td>
<td>2 230 326.94</td>
<td></td>
<td>0.2 ± 0.6</td>
<td>5.06</td>
<td>0.4</td>
</tr>
<tr>
<td>FL</td>
<td>88</td>
<td>10.2</td>
<td>209 415.50</td>
<td></td>
<td>11.2 ± 10.9</td>
<td>695.31</td>
<td>61.1</td>
</tr>
<tr>
<td>BF</td>
<td>864</td>
<td>100.0</td>
<td>2 230 326.94</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

FW: feature-wise, PW: pair-wise, HO: higher-order, HS: hot-spot, FL: function learning, BF: brute force, # M: number of measurements required for the heuristic, Time: runtime for all measurements neglecting compilation times, $\mu$: average error rate, $\sigma$: standard deviation, $\Delta$: absolute difference between the measured performance of the optimal configuration and the measured performance of the configuration predicted to perform best, $\delta$: percentage share of $\Delta$ on measured performance of the optimal configuration.
Current State – HW Optimizations

- Experiments with basic optimizations (vectorization, address pre-calculation) and temporal/ spatial blocking on different hardware architectures [3]

Speedups for Jacobi Smoothers on Ivy Bridge

Speedups for Jacobi Smoothers on BlueGene/Q
Current State – DSL(s)

- Different levels
  1. Continuous model (PDE, Domain)
  2. Discrete model (Stencils, Fields)
  3. Algorithmic components & parameters
  4. Pseudo-code for critical functions

- Prototype DSLs for each level
- First work on deriving levels from previous configurations
Current State – Code Generation (Multigrid)

- Multigrid
  - Scala prototype capable of generating fully working multigrid solvers for FD discretizations of Poisson’s equation in 2D and 3D

- Domain Generation
  - Currently only uniform grids, i.e. no HHG (Hierarchical Hybrid Grids) data structures
  - Domain is divided into rectangular blocks
  - Each block is composed of one or more fragments
  - Domain is setup at runtime
  - This includes memory for data fields, neighborhood connections, temporary memory for communication, …
Current State – Code Generation

- Parallelization
  - Uniform grids in 2D or 3D
  - Different communication schemes (6P/26P in 3D and 4P/8P in 2D)
  - Pure MPI or hybrid OpenMP-MPI parallelization
  - OpenMP parallelization by replacing MPI communication with local communication or by agglomeration of fragments and parallelizing the stencil kernels directly
  - Optional usage of MPI data types for sending and receiving field data in most cases
  - Variable number of ghost layers
  - …
JuQueen

- 28,672 Nodes (458,752 Cores)
- Compute Node: IBM PowerPC A2, 1.6 GHz, 16+1+1 cores
- Main memory: 16 GB per node (aggregate 448 TB)
- Overall peak performance: 5.9 PetaFLOP/s
(Very) Preliminary Results for 3D FD Poisson

- Weak scaling for a $V(3,3)$ cycle with Gauss-Seidel as smoother
- Coarse-grid solver is not implemented yet; thus, we use the smoother as CGS with the number of iterations according to
  a) the squared maximum of the number of fragments per dimension or
  b) a fixed number of iterations

![Weak Scaling for Different Smoothers](image-url)
Next Steps
Next Steps

- **Multigrid**
  - Integrate missing multigrid components to allow for comparison with our old multigrid codes [1]
  - This mainly includes coarse-grid solvers

- **Data structures**
  - Generate HHG data structures and the necessary stencil application codes
Next Steps

- Low-level optimization
  - Setup an interface between the code generator and the polyhedron model
  - Express transformations in polyhedron model
- Runtime prediction and optimization (LFA & SPL)
  - Develop a more precise model for feature interactions
  - Extend the LFA tool
  - Combine the two approaches to yield an efficient and robust optimization
References

(1) Sebastian Kuckuk, Björn Gmeiner, Harald Köstler, and Ulrich Rüde. *A generic prototype to benchmark algorithms and data structures for hierarchical hybrid grids.* Accepted at ParCo2013.


Thank you for your Attention!

Questions?