Algorithm and software development for efficient multigrid methods on modern HPC system

H. Köstler

30.04.2013
Computational Science and Engineering

Applications
- Multiphysics
- fluid, structure
- medical imaging
- laser

Computer Science
- HPC / hardware
- Performance engineering
- software engineering

Applied Math
- LBM
- multigrid
- FEM
- numerics
real-time simulation  
e.g. medicine

Large-scale simulation  
e.g. multi-physics
Problems

- **Hardware:** Modern HPC clusters are massively parallel
  - Intra-core, intra-node, and inter-node

- **Software:** Applications become more complex with increasing computational power
  - More complex (physical) models
  - Code development in interdisciplinary teams

- **Algorithm:** Multigrid is a general idea
  - Components and parameters depend on grid, type of problem, …
State of the Art: Application-driven Projects

User from application field  $\rightarrow$ Description of application  $\rightarrow$ Solution method  $\rightarrow$ Parallel implementation and framework  $\rightarrow$ Efficient implementation on specific hardware

- Mathematician
- Software specialist
- Hardware specialist

Efficient implementation on specific hardware
Proposed: Domain-driven Projects

Users from different application fields

→

Description of application in domain specific language

Domain expert

→

Mathematician

Automatic selection of multigrid components

Software specialist

→

Code generation for specific application

Hardware specialist

→

Automatic Tuning on specific hardware

PDE {
  Operators::Laplacian(Data::solution) = Data::rhs
}
A unique, tool-assisted, domain-specific co-design approach for the class of stencil codes

http://www.exastencils.org/
Toolbox Approach

Application layer

Algorithmic layer

Hardware layer

Software layer
Domain Scoping: Block-structured grids
Domain Scoping: PDEs and Discretization

- Type of PDEs

\[ Lu = -\nabla \cdot a\nabla u + b \cdot \nabla u + cu = f \text{ in } \Omega \]

- Finite differences, finite volumes, and finite elements

\[
\frac{1}{h^2} \begin{bmatrix}
-1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1
\end{bmatrix}
\]
Block smoothers for hybrid parallelization

Collocation coarse approximation (CCA) in order to control the stencil sizes on coarser grids

\[ A_C b_C = RAPb_C \]

Construction of prolongation operators in order to deal with jumping coefficients within the PDEs via

\[
Ax = \begin{pmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{pmatrix} \begin{pmatrix} x_F \\ x_C \end{pmatrix} \quad \quad S_{opt} = \begin{pmatrix} 0 & -A_{FF}^{-1} \\ 0 & I_{CC} \end{pmatrix} \quad \quad P_{opt} = \begin{pmatrix} -A_{FF}^{-1}A_{FC} \\ I_{CC} \end{pmatrix}
\]

\[
S_{opt} \begin{pmatrix} b_F \\ b_C \end{pmatrix} = Pb_C
\]

Toolbox Approach

Application layer

Algorithmic layer

Hardware layer

Software layer

DSL

Domain knowledge

Optimization of Multigrid Components and Parameters

Polyhedral Optimization

Generated Code

User interface

(Interactive) Visualization
Hardware { 

MPI {
    name= my_cluster
    id= HYBRID_CLUSTER
    nodes= 32

    components {
        XEON_CPU[2]
        GTX_680_GPU[2]
    }
}

activeConfiguration= HYBRID_CLUSTER
}
DSL: Computational domain

Domain {

Elements {  // required primitives
    Quad;
}

BoundaryConditions {
    NEU; // neumann
    NONE; // no boundary, inner primitive
}

meshfile= mesh.txt
}
DSL: Grid Data

Data {

  Solution {

    type= double
    dataLocation= VERTICES  // the data is located at the vertex positions
    gridLevels= ALL  // is defined on all grid levels
    ghostLayer= true  // has a ghostlayer for parallel data exchange
    init= RANDOM(0,1)  // initialize with random numbers

    boundaryConditions = …

  }

}
DSL: Operators and PDE

Operators {

Laplacian {

\textit{discretization} = FD \ // discretization strategy
\textit{Stencil\textless\text{NONE}\textgreater} = \{\{0, -1, 0\}, \{-1, 4, -1\}, \{0, -1, 0\}\} \ // stencil for inner data points
\textit{discretizationorder} = 2

}

}

PDE {

\texttt{Operators::Laplacian(Data::solution) = Data::rhs}

}
Toolbox Approach

Application layer

Algorithmic layer

Hardware layer

Software layer

- DSL
- Domain knowledge
- Optimization of Multigrid Components and Parameters
- Polyhedral Optimization
- Generated Code
- User interface
- (Interactive) Visualization
- LFA Toolbox
- Performance and Communication Model
Application: High Dynamic Range Compression

Data: Siemens AG, Healthcare Sector

- Sequences of 2D x-ray images
- Computational Steering

HDR Compression

- Idea: Modify magnitude of image gradient by position-dependent attenuating function $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}$

$$C = \nabla I \cdot \Phi$$

Fattal/Lischinski/Werman, *Gradient Domain High Dynamic Range Compression*, SIGGRAPH, 2002

- Energy functional

$$E(u) = \min_u \int_{\Omega} \left\| u(\mathbf{x}) - C \right\|^2 d\mathbf{x}$$

- Solve by multigrid the Euler-Lagrange equation

$$\Delta u = f \quad \text{in } \Omega$$

$$u = 0 \quad \text{on } \partial \Omega$$
Goal: minimize time $T$ to reach prescribed accuracy $\kappa$

$$T = t_V \cdot \log_{\rho_\infty} \kappa$$

Asymptotic convergence rates are estimated via LFA

<table>
<thead>
<tr>
<th></th>
<th>$V(1,1)$</th>
<th>$V(2,1)$</th>
<th>$V(2,2)$</th>
<th>$V(3,2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>0.063</td>
<td>0.033</td>
<td>0.025</td>
<td>0.019</td>
</tr>
<tr>
<td>$\rho_{2L}$</td>
<td>$R_f$</td>
<td>0.074</td>
<td>0.052</td>
<td>0.041</td>
</tr>
<tr>
<td>$\rho_{2L}$</td>
<td>$R_h$</td>
<td>0.104</td>
<td>0.074</td>
<td>0.056</td>
</tr>
<tr>
<td>$\rho_{3L}$</td>
<td>$R_f$</td>
<td>0.125</td>
<td>0.034</td>
<td>0.024</td>
</tr>
<tr>
<td>$\rho_{3L}$</td>
<td>$R_h$</td>
<td>0.161</td>
<td>0.053</td>
<td>0.034</td>
</tr>
</tbody>
</table>
\[ T = t_V \cdot \log_{\rho_{\infty}} \kappa \]

\[ t_V = \sum_{l=1}^{L} t_l \]

last pre-smoothing with restriction
rest of smoother

\[ t_1^{\text{block}} = t(2s_1 + s_2) + (\nu - 1) \cdot t(2s_1) \]

first pre-smoothing
last pre-smoothing with restriction
last post-smoothing with correction

\[ t_l^{\text{block}} = t(1.5s_l) + t(2s_l + s_{l+1}) + t(1.5s_l + s_{l-1}) \]

for \( 1 < l < (L - 4) \)

rest of smoother

\[ \sum_{l=L-4}^{L} t_l^{\text{block}} = t_{31 \times 31} + t(1.5s_{L-4} + s_{L-5}). \]

correction of level (L-5)
Optimized HDR Compression (size 2048x2048)

- half of an NVIDIA GTX 295
- 112 GB/s peak bandwidth
- compute capability 1.3
- NVIDIA GTX 480
- 177 GB/s peak bandwidth
- compute capability 2.0 (Fermi)

<table>
<thead>
<tr>
<th>GPU Type</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 295/2</td>
<td>60</td>
</tr>
<tr>
<td>GTX 480</td>
<td>100</td>
</tr>
<tr>
<td>GTX 480 wavefront</td>
<td>140</td>
</tr>
</tbody>
</table>

model:
- "opt." kernel time
- profiler measurement:
  - kernel time
  - bandwidth

mixed: 0.33 ms, 106 GB/s
split: 0.17 ms, 112 GB/s
optimized: 0.17 ms, 132 GB/s
### Prediction vs. Measurement

<table>
<thead>
<tr>
<th></th>
<th>$V(2,1)$</th>
<th>$t_V$</th>
<th>$V(3,2)$</th>
<th>$V(2,1)$</th>
<th>$t_{1E-4}$</th>
<th>$V(2,2)$</th>
<th>$V(3,2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1023$^2$</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basic layout</td>
<td>1.03</td>
<td>1.31</td>
<td>1.59</td>
<td>3.2</td>
<td>3.6</td>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>split layout</td>
<td>0.68</td>
<td>0.85</td>
<td>1.03</td>
<td>2.1</td>
<td>2.3</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>blocking</td>
<td>0.43</td>
<td>0.54</td>
<td>0.64</td>
<td>1.4</td>
<td>1.5</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>measurement</td>
<td>0.90</td>
<td>1.10</td>
<td>1.30</td>
<td>2.8</td>
<td>3.0</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td><strong>2047$^2$</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basic layout</td>
<td>3.4</td>
<td>4.4</td>
<td>5.4</td>
<td>10.7</td>
<td>12.0</td>
<td>13.7</td>
<td></td>
</tr>
<tr>
<td>split layout</td>
<td>2.0</td>
<td>2.5</td>
<td>3.0</td>
<td>6.1</td>
<td>6.7</td>
<td>7.6</td>
<td></td>
</tr>
<tr>
<td>blocking</td>
<td>1.3</td>
<td>1.6</td>
<td>2.0</td>
<td>4.1</td>
<td>4.4</td>
<td>5.0</td>
<td></td>
</tr>
<tr>
<td>measurement</td>
<td>2.1</td>
<td>2.6</td>
<td>3.2</td>
<td>6.6</td>
<td>7.2</td>
<td>8.1</td>
<td></td>
</tr>
<tr>
<td><strong>4095$^2$</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basic layout</td>
<td>12.9</td>
<td>16.5</td>
<td>20.2</td>
<td>40.3</td>
<td>45.0</td>
<td>51.5</td>
<td></td>
</tr>
<tr>
<td>split layout</td>
<td>7.0</td>
<td>8.8</td>
<td>10.7</td>
<td>21.9</td>
<td>24.1</td>
<td>27.3</td>
<td></td>
</tr>
<tr>
<td>blocking</td>
<td>4.7</td>
<td>5.9</td>
<td>7.1</td>
<td>14.6</td>
<td>16.1</td>
<td>18.2</td>
<td></td>
</tr>
<tr>
<td>measurement</td>
<td>6.5</td>
<td>8.3</td>
<td>10.0</td>
<td>20.5</td>
<td>22.5</td>
<td>25.4</td>
<td></td>
</tr>
</tbody>
</table>
Toolbox Approach

Application layer

Algorithmic layer

Hardware layer

Software layer
Tsubame 2.0 in Japan

- Compute nodes: 1442
- Processor: Intel Xeon X5670
- GPU: 3 x Nvidia Tesla M2050
- LINPACK performance: 1.2 Petaflops
- Power consumption: 1.4 MW
- Interconnect: QDR Infiniband
waLBerla: parallel block-structured grid framework
Optimal problem size on 1029 GPUs

- **6 levels**
  - Runtime in ms:
    - 69055 unknowns in million: 3000 ms
    - 34528 unknowns in million: 2500 ms
    - 17264 unknowns in million: 2000 ms
    - 8632 unknowns in million: 1500 ms
    - 4316 unknowns in million: 1000 ms
    - 2158 unknowns in million: 500 ms
    - 1079 unknowns in million: 0 ms
    - 539 unknowns in million: 0 ms
    - 270 unknowns in million: 0 ms
- **5 levels**
  - Runtime in ms:
    - 69055 unknowns in million: 3000 ms
    - 34528 unknowns in million: 2500 ms
    - 17264 unknowns in million: 2000 ms
    - 8632 unknowns in million: 1500 ms
    - 4316 unknowns in million: 1000 ms
    - 2158 unknowns in million: 500 ms
    - 1079 unknowns in million: 0 ms
    - 539 unknowns in million: 0 ms
    - 270 unknowns in million: 0 ms
Data sets for 3D HDR Compression

MRI data provided by Universitätsklinikum Erlangen

Tetrahedral finite element mesh used in HHG
Blue Gene/P in Jülich (Jugene)

- Compute node: 4-way SMP processor
- Processor: 32-bit PowerPC 450 core 850 MHz
- Cores: 294 912
- Overall peak performance: 1 Petaflops
- Main memory: 2 Gbytes per node (aggregate 144 TB)
Strong Scaling for Multigrid Solver on Jugene

Time [ms]

HHG
walberla

40% mesh
33% mesh
33% mesh

5646 cores
8192 cores
37158 cores
16384 cores
37158 cores
32758 cores

512x512x288
1024x1024x576
2048x2048x1152
Input

- Algorithm: V-cycle on (block)structured grid
- Generic Implementation
- Hardware information (bandwidth, peak performance)

Assumption

\[ t_{\text{total}} = t_{\text{comp,outer}} + \max(t_{\text{comp,inner}}, t_{\text{buffer}} + t_{\text{comm,GPUCPU}} + t_{\text{comm,MPI}}) \]

- Computation time limited by memory bandwidth and instruction throughput
- Communication time limited by network bandwidth and latency (for direct and collective communication)
Toolbox Approach

Application layer

Algorithmic layer

Hardware layer

Software layer

DSL

Domain knowledge

Optimization of Multigrid Components and Parameters

Polyhedral Optimization

Generated Code

User interface

(Interactive) Visualization

LFA Toolbox

Performance and Communication Model
Generic Grid Data Structures

- Memory representation
- Geometric representation
- Inner point
- Ghost point
- Volume
- Edge
- Vertex
Polyhedral Optimization

for (i=1; i<=n; i++)
  for (j=1; j<=n-i; j++)

for (t=1; t<=n; t++)
  parfor (p=1; p<=t; p++)
    A[t-p+1][p] = ...

Analysis

Transformation(s)

Code generation

Dependences:
(i, j) → (i+1, j)
(i, j) → (i, j+1)

1 ≤ i ≤ n
1 ≤ j ≤ n - i

Loop bounds and array indices are linear (affine) expressions.
→ Polyhedron model

1 ≤ t ≤ n
1 ≤ p ≤ t

(t, p) → (t+1, p)
(t, p) → (t+1, p+1)
Toolbox Approach

Application layer

Algorithmic layer

Hardware layer

Software layer

Domain knowledge

DSL

Optimization of Multigrid Components and Parameters

Polyhedral Optimization

Generated Code

User interface

(Interactive) Visualization
HiPacc (Heterogeneous Image Proc. Acceleration)

Domain Knowledge
Architecture Knowledge

Source-to-Source Compiler
Clang/LLVM

C++ embedded DSL

CUDA (GPU)
OpenCL (x86/GPU)
C/C++ (x86)
Renderscript (x86/ARM/GPU)

CUDA/OpenCL/Renderscript Runtime Library

Computer Science X - System Simulation Group
Harald Köstler (harald.koestler@fau.de)
<table>
<thead>
<tr>
<th></th>
<th>Tesla C2050 Manual</th>
<th>Tesla C2050 OpenCL</th>
<th>Tesla C2050 CUDA</th>
<th>Quadro FX 5800 Manual</th>
<th>Quadro FX 5800 OpenCL</th>
<th>Quadro FX 5800 CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1: smooth</td>
<td>0.53</td>
<td>0.58</td>
<td>0.79</td>
<td>1.35</td>
<td>1.50</td>
<td>1.01</td>
</tr>
<tr>
<td>L1: smooth</td>
<td>0.57</td>
<td>0.79</td>
<td>1.48</td>
<td>0.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1: residual</td>
<td>0.67</td>
<td>0.57</td>
<td>0.79</td>
<td>1.65</td>
<td>1.62</td>
<td>0.93</td>
</tr>
<tr>
<td>L1: restrict</td>
<td>0.28</td>
<td>0.28</td>
<td></td>
<td>0.59</td>
<td>0.53</td>
<td></td>
</tr>
<tr>
<td>L2: smooth</td>
<td>0.12</td>
<td>0.16</td>
<td>0.26</td>
<td>0.35</td>
<td>0.44</td>
<td>0.26</td>
</tr>
<tr>
<td>L2: smooth</td>
<td>0.16</td>
<td>0.26</td>
<td></td>
<td>0.44</td>
<td>0.27</td>
<td></td>
</tr>
<tr>
<td>L2: residual</td>
<td>0.19</td>
<td>0.16</td>
<td>0.25</td>
<td>0.44</td>
<td>0.46</td>
<td>0.26</td>
</tr>
<tr>
<td>L2: restrict</td>
<td>0.08</td>
<td>0.12</td>
<td></td>
<td>0.18</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>L3–L6</td>
<td>0.70</td>
<td>0.63</td>
<td>1.85</td>
<td>1.33</td>
<td>1.73</td>
<td>1.34</td>
</tr>
<tr>
<td>L2: interpolate</td>
<td></td>
<td>0.21</td>
<td>0.17</td>
<td>0.29</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>L2: smooth</td>
<td>0.15</td>
<td>0.16</td>
<td>0.27</td>
<td>0.34</td>
<td>0.45</td>
<td>0.27</td>
</tr>
<tr>
<td>L2: smooth</td>
<td>0.34</td>
<td>0.16</td>
<td>0.27</td>
<td>0.86</td>
<td>0.44</td>
<td>0.27</td>
</tr>
<tr>
<td>L1: interpolate</td>
<td>0.53</td>
<td>0.83</td>
<td>0.48</td>
<td>0.96</td>
<td>0.61</td>
<td></td>
</tr>
<tr>
<td>L1: smooth</td>
<td>0.53</td>
<td>0.57</td>
<td>0.89</td>
<td>1.35</td>
<td>1.48</td>
<td>1.01</td>
</tr>
<tr>
<td>L1: smooth</td>
<td>0.53</td>
<td>0.57</td>
<td>0.88</td>
<td>1.35</td>
<td>1.49</td>
<td>1.01</td>
</tr>
<tr>
<td>(\Sigma) V-cycle</td>
<td>3.90</td>
<td>5.75</td>
<td>8.31</td>
<td>9.02</td>
<td>13.54</td>
<td>9.07</td>
</tr>
</tbody>
</table>

Execution times in for an image of size 2048x2048 pixels.

Toolbox Approach

Application layer

Algorithmic layer

Hardware layer

Software layer
Future Work

- Integration of all parts
- Finish Toolboxes
- Apply to real applications