A robust geometric multigrid solver within the WaLBerla framework

Dr.-Ing. H. Köstler

15th Copper Mountain Conference on Multigrid Methods
28.3.2011
Contents

- WaLBerla framework: Software design for HPC clusters
- Numerical Results
  - Geometric Multigrid on CPU-clusters
  - Geometric Multigrid on GPU
  - Robust Multigrid
- Future Work
WaLBerla

SOFTWARE AND PERFORMANCE ENGINEERING
WaLBerla framework
WaLBerla

Applications

Free-Surface Flows

Particulate Flows

Hardware
WaLBerla

Applications  Free-Surface Flows  Particulate Flows

Hardware

Performance  Parallelization  Efficient Kernels  Performance Modeling

Software Engineering

Sweeps  Patches & Blocks  Meta Data
WaLBerla: Basic idea

**waLBerla (C++)**

Code management, standard implementations

**Low-level kernels** for optimized architecture-specific computations (in C++, CUDA, Assembler)
WaLBerla structure

The Core

- Responsible for sequence control and data management

Modules

- Common functionality, used by several applications

Applications

- Each user works on an own application
WaLBerla framework

- **Main goal**: provide a massive parallel and efficient software framework for multi-physics simulations
- WaLBerla is mainly designed for HPC clusters
- Software design concepts
  - Patch, block, and sub-block concept
  - Functionalities
  - Sweep concept
  - Communication concept
WaLBerla: Patch concept

- **Simulation Domain**
- **Patch**
- **Unknown / Cell**
- **Block**

**Block Info:**
- Application
- Rank
- Is Allocated
- AABB
- BlockID

**Block Data:**
- Simulation Data:
  - Cartesian Data
  - Unstructured Data
- Configurable:
  - Data Structures
  - Algorithms
  - Optimizations
WaLBerla: Blocks

- Pure LBM
- Free-Surface
- Particulate Flows
- Free-Surface + Particulate Flows
WaLBerla: Functionalities

- Different granularities
- Realized by adding meta data to each functionality consisting of three unique identifiers (UID)

<table>
<thead>
<tr>
<th>UID</th>
<th>Name</th>
<th>Granularity</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>fs</td>
<td>Functionality Selector</td>
<td>Simulation</td>
<td>Gravity on/off, collision model</td>
</tr>
<tr>
<td>hs</td>
<td>Hardware Selector</td>
<td>Process</td>
<td>CPU, GPU, CPU+GPU</td>
</tr>
<tr>
<td>bs</td>
<td>Block Selector</td>
<td>Block</td>
<td>Application</td>
</tr>
</tbody>
</table>
WaLBerla: Sweep concept

Sweep Concept

Sweep Chain I (Time loop)
- Sweep I
- Sweep II
- Sweep III

Sweep Chain II
- Sweep I
- Sweep II

Sweep
- Preprocessing
  - Communication
  - Timing
- Block Sweep
- Global Sweep
- Postprocessing
  - Visualization
  - Timing

↓: Execution Order
→: Iteration
←: Dependency
WaLBerla: Communication concept

**GPU**
- GPU Buffers
- Local Communication
- Swap
- GPU - GPU Copy Operations

**CPU**
- MPI Buffers
- MPI_Irecv
- PCI Express Transfer
- InfiniBand Transfer

**Neighboring Process**
- MPI Communication
Multigrid on CPU clusters

NUMERICAL RESULTS
Blue Gene/P in Jülich (Jugene)

- Compute node: 4-way SMP processor
- Processortype: 32-bit PowerPC 450 core 850 MHz
- Cores: 294 912
- Overall peak performance: 1 Petaflops
- Main memory: 2 Gbytes per node (aggregate 144 TB)
How many unknowns are possible today?

<table>
<thead>
<tr>
<th>Cores</th>
<th>DoF (waLBerla)</th>
<th>Time (s) (waLBerla)</th>
<th>DoF (HHG)</th>
<th>Time (s) (HHG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>2.15E+09</td>
<td></td>
<td>9.06</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>1.72E+10</td>
<td></td>
<td>9.17</td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>2.75E+11</td>
<td></td>
<td>9.21</td>
<td></td>
</tr>
<tr>
<td>131072</td>
<td>2.20E+12</td>
<td>scheduled</td>
<td>9.33</td>
<td></td>
</tr>
<tr>
<td>262144</td>
<td>4.40E+12</td>
<td>scheduled</td>
<td>1.10E+12</td>
<td>6.52</td>
</tr>
<tr>
<td>294912</td>
<td>4.95E+12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- HHG (Hierarchical Hybrid Grids): 3D FEM multigrid solver, constant 15-point stencil, double accuracy (→ talk U. Rüde, B. Gmeiner)
- WaLBerla: 3D FV multigrid solver, constant 7-point stencil, float accuracy → 7.4 Mu/s (per compute node with 4 cores, CPU+Poisson+3D)
Multigrid on GPU

NUMERICAL RESULTS
## Test Setup

<table>
<thead>
<tr>
<th></th>
<th>Xeon 5550</th>
<th>Tesla M1060</th>
<th>Tesla C2050</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor Frequency</strong></td>
<td>2.66 GHz</td>
<td>1.3 GHz</td>
<td>1.15 GHz</td>
</tr>
<tr>
<td><strong>Memory Frequency</strong></td>
<td>1.3 GHz</td>
<td>800 MHz</td>
<td>1.5 GHz</td>
</tr>
<tr>
<td><strong>Memory Size</strong></td>
<td>12 GB</td>
<td>4 GB</td>
<td>3 GB</td>
</tr>
<tr>
<td><strong># Streaming Units / Cores</strong></td>
<td>4</td>
<td>240</td>
<td>448</td>
</tr>
<tr>
<td><strong>FLOP/s (SP)</strong></td>
<td>85.12 GFLOP/s</td>
<td>933 GFLOP/s</td>
<td>1030 GFLOP/s</td>
</tr>
<tr>
<td><strong>FLOP/s (DP)</strong></td>
<td>42.56 GFLOP/s</td>
<td>78 GFLOP/s</td>
<td>515 GFLOP/s</td>
</tr>
<tr>
<td><strong>Memory Bandwidth</strong></td>
<td>32 GB/s</td>
<td>102 GB/s</td>
<td>144 GB/s</td>
</tr>
</tbody>
</table>
## Performance for one V(2,2)-cycle

<table>
<thead>
<tr>
<th></th>
<th>PowerPC 450</th>
<th>Xeon 5550</th>
<th>M1060</th>
<th>C2050</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D const stencil (5p)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>798.9 Mu/s</td>
<td></td>
</tr>
<tr>
<td>3D const stencil (7p)</td>
<td>7.4 Mu/s</td>
<td>26.8 Mu/s</td>
<td>93.2 Mu/s</td>
<td></td>
</tr>
<tr>
<td>3D variable stencil (7p)</td>
<td>11.2 Mu/s</td>
<td>32.9 Mu/s</td>
<td>88.3 Mu/s</td>
<td></td>
</tr>
</tbody>
</table>

We solve: \[ \text{div}(g \nabla u) = f \quad \text{in} \ \Omega \]

\[ u = 0 \quad \text{on} \ \partial \Omega \]

for more details on multigrid performance → talk M. Stürmer
Single node performance

- Tesla C2050, single precision
- Tesla C2050, double precision
- Tesla M1060, single precision
- Tesla M1060, double precision

Runtime for a (2,2)-V-Cycle [s] vs Number of unknowns

- 128x128x128
- 256x128x128
- 256x256x128
- 256x256x256
Weak scaling, variable stencil

Weak scaling for $2 \times 256^3$ unknowns per node, single precision.

Runtime for a (2,2)-V-Cycle [s]

Number of used nodes

- Xeon 5550: 2 threads per node, 1 per socket
- Xeon 5550: 4 Threads per node, 2 per socket
- Tesla M1060: 2 GPUs per node
- Xeon 5550: 2 Threads per node, 2 per socket
Strong scaling, variable stencil

Strong Scaling: Runtime for $256^3$ unknowns in total

- Xeon 5550, single precision
- Xeon 5550, double precision
- Tesla M1060, single precision
- Tesla M1060, double precision

Number of processing units vs. Runtime for a (2,2)-V-Cycle [s]

0 1 2 3 4 5 6 7 8 9

0 0.5 1 1.5 2 2.5
Robust Multigrid on GPU

NUMERICAL RESULTS
A Robust Multigrid solver

- Requirements:
  - Arbitrary mesh sizes, block-structured grids
  - Varying stencils, jumping coefficients
  - Various boundary conditions, 2D and 3D

- Ingredients:
  - Various smoothers
  - Collocation Coarse Approximation (CCA) for coarse grid operators
  - Matrix-dependent transfer operators


Construction of coarse grid operator via CCA

- Generalize standard Galerkin coarsening

\[ A^c = RAP \]

- by allowing an arbitrary no. of entries \( S_c \) per row (stencil) in the coarse grid operator and fulfilling for certain basis functions \( b_k, k = 1 \ldots S_c \)

\[ A^c b_k^c = RAP b_k^c \]

- With fine grid basis functions we have

\[ A^c \hat{R}b_k = RAb_k \]

- Results in small linear system for each row of the coarse grid operator

- for fixed basis \( b \) the inverse of the system matrix can be precomputed

- and only \( A \) and \( R \) have to be applied to the resulting canonical basis functions
We require for additional basis functions

$$Sb_k = P\hat{R}b_k$$

where $S$ denotes smoothing

Again considering canonical basis functions this approach is related to smoothed aggregation.

With the „optimal“ smoother

$$S = \begin{bmatrix} 0 & -A_{FF}^{-1}A_{FC} \\ 0 & I \end{bmatrix}$$

$$P = \begin{bmatrix} P_F \\ I \end{bmatrix}$$

$$A = \begin{bmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{bmatrix}$$

we obtain

$$-A_{FF}^{-1}A_{FC}b_k^c = P_Fb_k^c$$

$$A_{FF}P_Fb_k^c = -A_{FC}b_k^c$$
We choose additional basis functions that vanish on coarse points, therefore

\[ A^c \hat{R} b_k = R A b_k = 0 \]

With

\[
R = \begin{bmatrix}
R_{FF} \\
I
\end{bmatrix}
\]

this results in

\[ R A b_k = (R_F A_{FF} + A_{CF}) b_k = 0 \]

\[ R_F A_{FF} b_k = -A_{CF} b_k \]
CCA estimates (coarse operator, two-grid, 2D)

<table>
<thead>
<tr>
<th>P, R</th>
<th>stores</th>
<th>loads</th>
<th>flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>$N_c S_c$</td>
<td>NS</td>
<td>$(18S+18) N_c S_c$</td>
</tr>
<tr>
<td>variable</td>
<td>$N_c S_c$</td>
<td>NS</td>
<td>$(18S+82) N_c S_c$ + construct P, R</td>
</tr>
</tbody>
</table>

- $N_c$, $N$ no. of grid points on fine and coarse grid
- $S_c$, $S$ no. of stencil entries for fine and coarse operator
- $P$, $R$ are assumed to have a 9-point stencil on each grid

Assumptions for GPU implementation
- Basis functions are stored in constant memory
- $P$, $R$ are stored in shared memory
- $A$, $A_c$ are stored in global memory
Performance for one V(2,2)-cycle and CCA

<table>
<thead>
<tr>
<th></th>
<th>setup 2D</th>
<th>V(2,2) 2D</th>
<th>setup 3D</th>
<th>V(2,2) 3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>const P, R</td>
<td>4.19 Mu/s</td>
<td>27.26 Mu/s</td>
<td>1.86 Mu/s</td>
<td>11.95 Mu/s</td>
</tr>
<tr>
<td>variable P, R</td>
<td>1.98 Mu/s</td>
<td>17.03 Mu/s</td>
<td>0.49 Mu/s</td>
<td>7.27 Mu/s</td>
</tr>
</tbody>
</table>

- Experiments are performed on a Xeon X5670 @ 2.93 Ghz (OpenMP parallel)
- In 2D for constant P the pure compute kernel within the setup achieves 130 Mu/s on Nvidia GTX 480
Future Work

- Performance Engineering
  - CPU-GPU clusters
- CCA
  - Evaluation of different matrix-dependent transfer operators
- Full integration of parallel, robust multigrid solver in WaLBerla
Acknowledgements

- Regional Computing Center of Erlangen (support for computer systems)

- Funded by
  - SKALB, German project,
  - Bundesministerium für Bildung und Forschung
  - KONWIHR, Bavarian project