Efficient Finite Element Geometric Multigrid Solvers for Unstructured Grids on GPUs

Markus Geveler,
Dirk Ribbrock, Dominik Göddeke, Peter Zajac, Stefan Turek

Institut für Angewandte Mathematik
TU Dortmund, Germany
markus.geveler@math.tu-dortmund.de

PARENG 11
Ajaccio, April 14, 2011
Motivation

FEM

- highly accurate for solving PDEs:
  - high order (non-conforming) FEs
  - arbitrarily unstructured grids to resolve complex geometries
  - grid adaptivity
  - Pressure-Schur-Complement Preconditioning
  - ...

- in connection with Geometric Multigrid solvers:
  - convergence rates independent of mesh width $h$
  - superlinear convergence effect possible (→ high order FE spaces)

→ Finite Element Geometric Multigrid enhances numerical efficiency.
Motivation

**GPUs**

- high on-chip memory bandwidth
- maximisation of the overall throughput of a large set of tasks
- parallelisation techniques for FEM software are being explored
- stronger smoothers are still an issue \(\rightarrow\) SPAI, ILU
- complete Geometric Multigrid solvers haven’t had much attention yet

**Today: Realising FE-gMG on the GPU** \(\rightarrow\) *hardware-oriented numerics*
Solution approach

**Idea:** One performance-critical kernel: SpMV
- coarse-grid solver: Conjugate Gradients
- smoothers: based on preconditioned Richardson iteration
- defect calculations

**What’s left**
- some BLAS-1 (dot-product, norm, ...)
- *grid transfer* → can be reduced to SpMV too (later)

**Benefits**
- solver must be implemented only once
- oblivious of FE space and domain dimension
- performance tuning reduced to one kernel
Solution approach

Grid transfers

- chose the standard Lagrange bases for two consecutively refined $Q_k$ finite element spaces $V_{2h}$ and $V_h$
- function $u_{2h} \in V_{2h}$ can be interpolated in order to prolongate it

$$u_h := \sum_{i=1}^{m} x_i \cdot \varphi_h^{(i)}, \quad x_i := u_{2h}(\xi_h^{(i)})$$

- for the basis functions of $V_{2h}$ and $u_{2h} = \sum_{j=1}^{n} y_j \cdot \varphi_{2h}^{(j)}$ with coefficient vector $y$, we can write the prolongation as

$$u_h := \sum_{i=1}^{m} x_i \cdot \varphi_h^{(i)}, \quad x := P_{2h}^h \cdot y$$

- restriction matrix $R_{2h}^h = (P_{2h}^h)^T$
Solution approach

Grid transfer: Simplified example - 2D, $Q_1$ on regular grid

$$P_{2h}^h = \begin{bmatrix} P_v \\ P_{\epsilon} \\ P_q \end{bmatrix}$$
Solution approach

Grid transfer: Prolongation matrix examples

- sparsity pattern (and bandwidth) depends on DOF numbering technique → performance
- same for the stiffness matrices
Implementation

Sparse matrix-vector multiply on the GPU: ELLPACK-R

- store sparse matrix $S$ in two arrays $A$ (non-zeros in column-major order) and $j$ (column index for each entry in $A$)
- $A$ has size ($\#\text{rows in } S$) $\times$ (maximum number of non-zeros in any row of $S$)
- shorter rows are padded with zeros
- additional array $r1$ to store effective count of non-zeros in every row without the padding-zeros (stop computation on a row after the actual non-zeros)

$$S = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \quad \Rightarrow \quad A = \begin{bmatrix} 1 & 7 & \ast \\ 2 & 8 & \ast \\ 5 & 3 & 9 \\ 6 & 4 & \ast \end{bmatrix} \quad j = \begin{bmatrix} 0 & 1 & \ast \\ 1 & 2 & \ast \\ 0 & 2 & 3 \\ 1 & 3 & \ast \end{bmatrix} \quad r1 = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 2 \end{bmatrix}$$
Sparse matrix-vector multiply on the GPU

\[ y_i = \sum_{nz=0}^{rl_i} A_{i,nz} \times x_{j_nz} \]

- based on the ELLPACK-R format
- \( y = A x \) can be performed by computing each entry \( y_i \) of the result vector \( y \) independently (one GPU-thread per \( y_i \))
- regular access pattern on data of \( y \) and \( A \)
- access pattern on \( x \) depends highly on sparsity pattern of \( A \)
- data access to all three arrays is fully coalesced due to column-major ordering
- \( x \)-values can be cached (texture-cache or L2 on FERMI)
- no synchronisation between threads necessary
- no branch divergence
Results

Benchmark setup

\[
\begin{cases}
-\Delta u = 1, & x \in \Omega \\
u = 0, & x \in \Gamma_1 \\
u = 1, & x \in \Gamma_2
\end{cases}
\]

- Poisson problem as a fundamental component in many practical situations
- different FE spaces
- different DOF numbering techniques
- Jacobi preconditioning, V-cycle
- Intel Core i7 920 quadcore workstation (4 threads) / NVIDIA GeForce GTX 285 GPU
## Sparse matrix-vector multiply on the GPU

<table>
<thead>
<tr>
<th>L</th>
<th>SSE</th>
<th>MCSSE</th>
<th>$Q_1$ speedup</th>
<th>CUDA speedup</th>
<th>SSE</th>
<th>MCSSE</th>
<th>$Q_2$ speedup</th>
<th>CUDA speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>870.81</td>
<td>2445.06</td>
<td>2.81</td>
<td>7441.65</td>
<td>492</td>
<td>894</td>
<td>1.82</td>
<td>7985</td>
</tr>
<tr>
<td>8</td>
<td>672.14</td>
<td>1163.98</td>
<td>1.73</td>
<td>8411.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>506.85</td>
<td>988.19</td>
<td>1.95</td>
<td>7928.9</td>
<td>323</td>
<td>528</td>
<td>1.63</td>
<td>7680</td>
</tr>
<tr>
<td>10</td>
<td>426.81</td>
<td>855.99</td>
<td>2.01</td>
<td>7925.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>790</td>
<td>2897</td>
<td>3.67</td>
<td>7247</td>
<td>458</td>
<td>883</td>
<td>1.93</td>
<td>7035</td>
</tr>
<tr>
<td>8</td>
<td>685</td>
<td>1268</td>
<td>1.85</td>
<td>8459</td>
<td>289</td>
<td>802</td>
<td>2.78</td>
<td>6470</td>
</tr>
<tr>
<td>9</td>
<td>445</td>
<td>1187</td>
<td>2.67</td>
<td>7539</td>
<td>262</td>
<td>743</td>
<td>2.84</td>
<td>6288</td>
</tr>
<tr>
<td>10</td>
<td>399</td>
<td>1120</td>
<td>2.81</td>
<td>7314</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>842</td>
<td>3299</td>
<td>3.92</td>
<td>8506</td>
<td>491</td>
<td>950</td>
<td>1.93</td>
<td>7677</td>
</tr>
<tr>
<td>8</td>
<td>760</td>
<td>1344</td>
<td>1.77</td>
<td>10403</td>
<td>334</td>
<td>897</td>
<td>2.69</td>
<td>7911</td>
</tr>
<tr>
<td>9</td>
<td>504</td>
<td>1369</td>
<td>2.72</td>
<td>11007</td>
<td>330</td>
<td>836</td>
<td>2.53</td>
<td>8074</td>
</tr>
<tr>
<td>10</td>
<td>494</td>
<td>1372</td>
<td>2.78</td>
<td>11176</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>697</td>
<td>2048</td>
<td>2.94</td>
<td>5880</td>
<td>346</td>
<td>787</td>
<td>2.27</td>
<td>3820</td>
</tr>
<tr>
<td>8</td>
<td>497</td>
<td>981</td>
<td>1.97</td>
<td>4257</td>
<td>244</td>
<td>590</td>
<td>2.42</td>
<td>2468</td>
</tr>
<tr>
<td>9</td>
<td>348</td>
<td>843</td>
<td>2.42</td>
<td>2628</td>
<td>160</td>
<td>366</td>
<td>2.29</td>
<td>1689</td>
</tr>
<tr>
<td>10</td>
<td>224</td>
<td>443</td>
<td>1.98</td>
<td>1794</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>809</td>
<td>3148</td>
<td>3.89</td>
<td>8049</td>
<td>482</td>
<td>852</td>
<td>1.77</td>
<td>7465</td>
</tr>
<tr>
<td>8</td>
<td>738</td>
<td>1313</td>
<td>1.78</td>
<td>9726</td>
<td>300</td>
<td>836</td>
<td>2.79</td>
<td>7776</td>
</tr>
<tr>
<td>9</td>
<td>471</td>
<td>1345</td>
<td>2.86</td>
<td>10342</td>
<td>299</td>
<td>782</td>
<td>2.62</td>
<td>7903</td>
</tr>
<tr>
<td>10</td>
<td>465</td>
<td>1331</td>
<td>2.86</td>
<td>10553</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results
Results

**Geometric Multigrid with Jacobi preconditioning**

- mission accomplished: SpMV performance transported to solver level
- clever sorting pays off
Results

**Geometric Multigrid with Jacobi preconditioning**

- mission accomplished: solver oblivious of FE-space
Results

Prospects of even better numerics - Geometric Multigrid with stronger smoothing: SPAI

<table>
<thead>
<tr>
<th>L</th>
<th>Jacobi $Q_1$ CPU</th>
<th>SPAI CPU</th>
<th>Jacobi $Q_2$ CPU</th>
<th>SPAI CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0.11</td>
<td>0.09</td>
<td>0.07</td>
<td>0.33</td>
</tr>
<tr>
<td>8</td>
<td>0.47</td>
<td>0.18</td>
<td>0.13</td>
<td>0.59</td>
</tr>
<tr>
<td>9</td>
<td>2.30</td>
<td>0.42</td>
<td>0.34</td>
<td>0.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>Jacobi $Q_1$ GPU</th>
<th>SPAI GPU</th>
<th>Jacobi $Q_2$ GPU</th>
<th>SPAI GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0.11</td>
<td>0.09</td>
<td>0.07</td>
<td>0.11</td>
</tr>
<tr>
<td>8</td>
<td>0.47</td>
<td>0.18</td>
<td>0.13</td>
<td>0.59</td>
</tr>
<tr>
<td>9</td>
<td>2.30</td>
<td>0.42</td>
<td>0.34</td>
<td>0.99</td>
</tr>
</tbody>
</table>

- but: assembly of SPAI-matrix on GPU still unresolved
Conclusion

Summary of the results

- FE-gMG is efficient and flexible
- GPU vs. multicore CPU: close to one order of magnitude speedup
- DOF numbering may be critical
- sophisticated (sparse) preconditioners make the difference

Future challenges

- stronger smoothers for unstructured problems
- cross-effects with resorting the degrees of freedom in combination with a specific matrix storage format and associated SpMV kernel
- assembly of transfer-, stiffness- and preconditioner-matrices
- other related data-parallel operations: adaptive grid-deformation, ...
Acknowledgements

Supported by BMBF, *HPC Software für skalierbare Parallelrechner: SKALB project 01IH08003D.*