Towards a complete FEM-based simulation toolkit on GPUs: Geometric Multigrid solvers

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

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

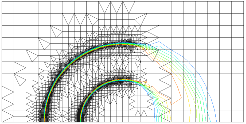
ParCFD 11
Barcelona, May 18, 2011

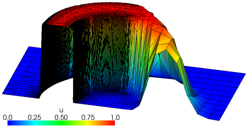
technische universität dortmund
fakultät für mathematik

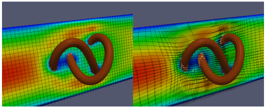
Motivation

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 indepent of mesh width h
    - superlinear convergence effect possible ($\rightarrow$ high order FE spaces)





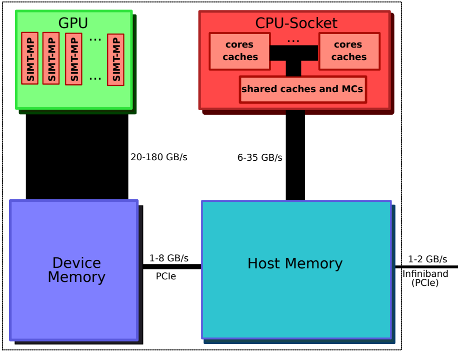


$\rightarrow$ Finite Element Geometric Multigrid enhances numerical efficiency.

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 → SPAI, ILU
- complete Geometric Multigrid solvers haven't had much attention yet



But: bare 'MachoFlop'-performance does not count! Today: Realising FE-gMG on the GPU → *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

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

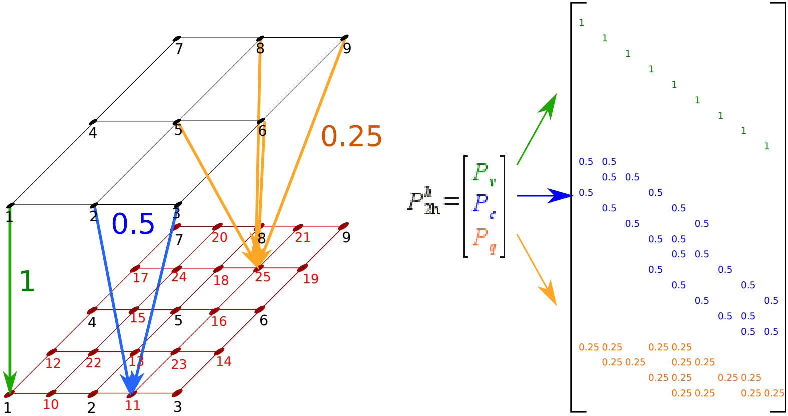
- for the basis functions of V_{2h} and $u_{2h} = \sum_{j=1}^{n} y_j \cdot \varphi_{2h}^{(j)}$ with coefficient vector y, we can write the prolongation as

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

- restriction matrix $R_h^{2h} = (P_{2h}^{h})^T$

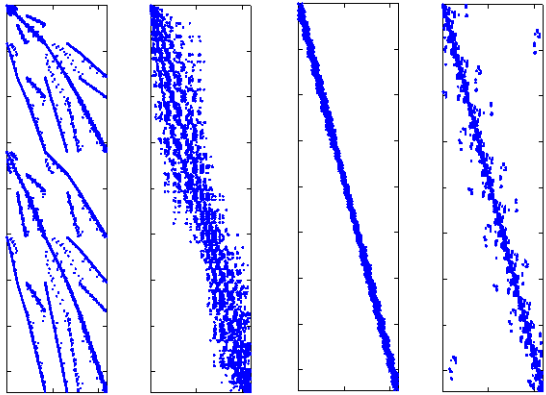
Solution approach

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



Solution approach

Grid transfer: Prolongation matrix examples



- left to right: 2-Level, Cuthill McKee, Coordinate-based and Hierarchical orderings
- sparsity pattern (and bandwidth) depends on DOF numbering technique → performance

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 (#rows in S) $\times$ (maximum number of non-zeros in any row of S)
- shorter rows are padded with zeros
- additional array `rl` to store effective count of non-zeros in every row without the padding-zeros (stop computation on a row after the actual non-zeros)

$$S = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \quad \Rightarrow \quad \mathtt{A} = \begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \mathtt{j} = \begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 2 & 3 \\ 1 & 3 & * \end{bmatrix} \mathtt{rl} = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 2 \end{bmatrix}$$

Implementation

Sparse matrix-vector multiply on the GPU

$$y_i = \sum_{nz=0}^{rl_i} A_{i,nz} * x_{j_{nz}}$$

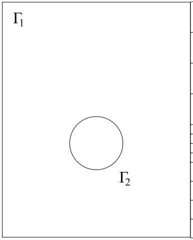
- based on the ELLPACK-R format
- $\mathbf{y} = A\mathbf{x}$ can be performed by computing each entry y_i of the result vector $\mathbf{y}$ independently (one GPU-thread per y_i)
- regular access pattern on data of $\mathbf{y}$ and A
- access pattern on $\mathbf{x}$ depends highly on sparsity pattern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

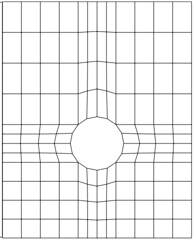
Results

Benchmark setup

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

L	Q_1 N	Q_1 non-zeros	Q_2 N	Q_2 non-zeros
4	576	4552	2176	32192
5	2176	18208	8448	128768
6	8448	72832	33280	515072
7	33280	291328	132096	2078720
8	132096	1172480	526336	8351744
9	526336	4704256	2101248	33480704
10	2101248	18845696	-	-





- Poisson problem as a fundamental component in many practical situations
- different FE spaces
- different DOF numbering techniques
- Jacobi preconditioning, V-cycle
- Intel Core i7 980 Gulftown hexacore workstation / NVIDIA Fermi GPU (Tesla C2070)

Results

In addition: stronger preconditioning with SPAI

$$\| I - MA \|_F^2 \;=\; \sum_{k=1}^{n} \| e_k^{\mathrm{T}} - m_k^{\mathrm{T}} A \|_2^2 \;=\; \sum_{k=1}^{n} \| A^{\mathrm{T}} m_k - e_k \|_2^2$$

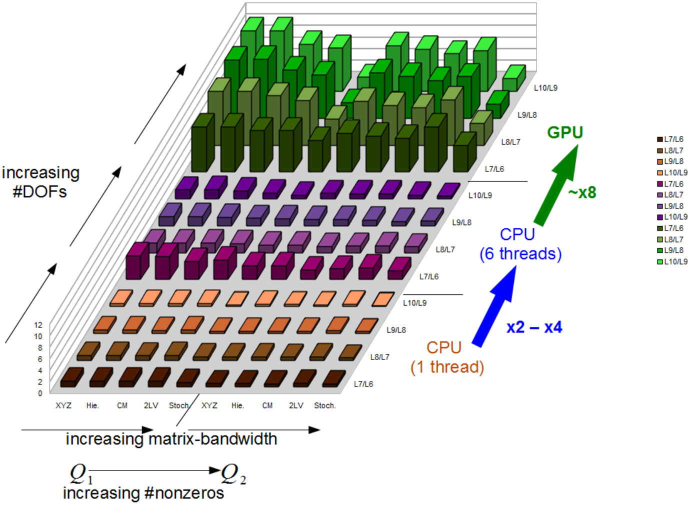
where e_k is the k-th unit-vector and m_k is the k-th row of M. $\rightarrow$ for n columns of M $\rightarrow$ n *least squares* opt.-problems:

$$min_{m_k} \| A^{\mathrm{T}} m_k - e_k \|_2, \;\; k = 1, \ldots n.$$

- sparsity-pattern of the stiffness-matrix is used for pattern of preconditioner

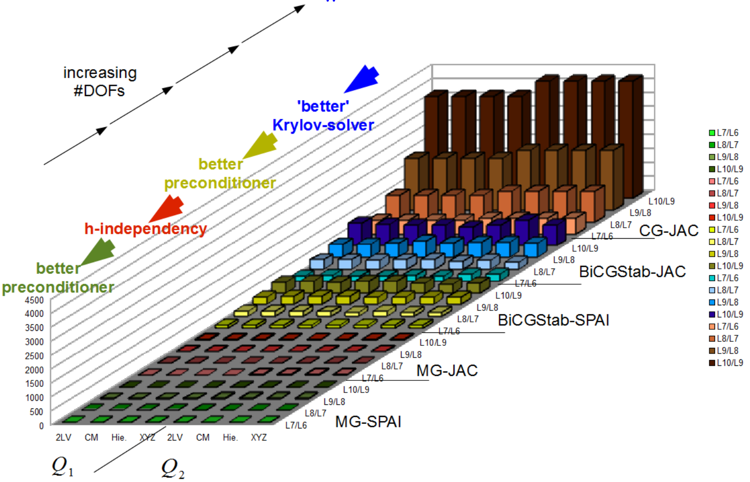
Results

Sparse matrix-vector multiply on the GPU



Results

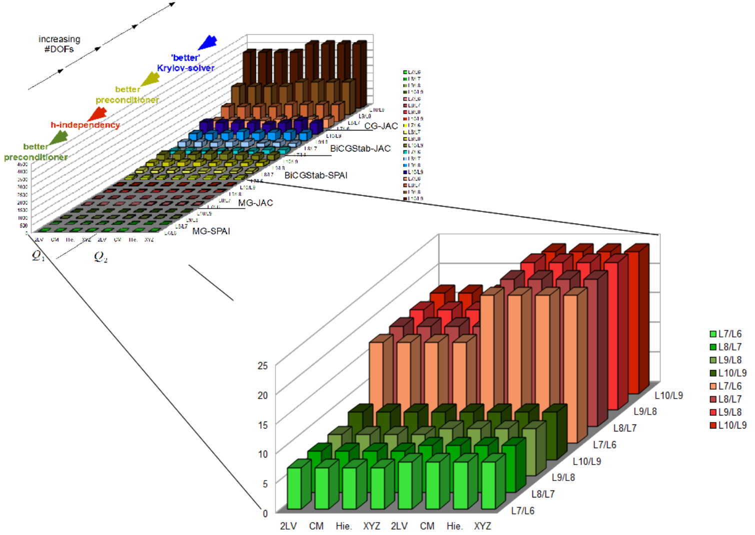
Its numerics, that counts: #iterations



- → potential degradation of $\times 1/1000$
- → hardware may offer an order of magnitude speedup

Results

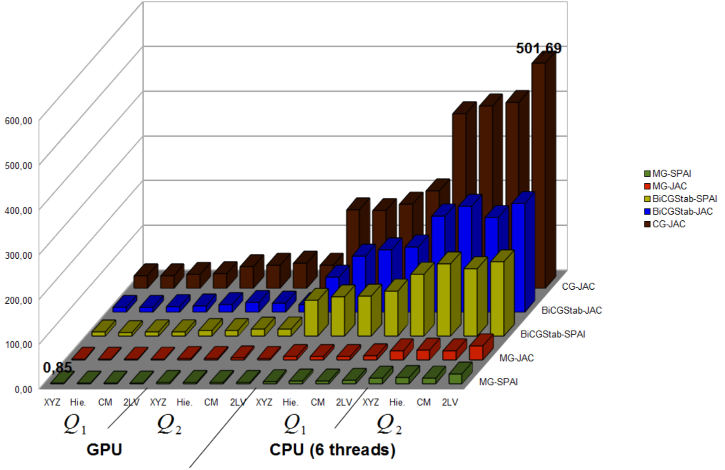
FE-gMG: a closer look at preconditioning



- → SPAI offers $\times 1/2$; SPAI+Q_2 works well

Results

Execution times for finest discretisation and reasonable numbering-techniques: CPU, GPU



Results

Geometric Multigrid

- mission accomplished: SpMV performance transported to solver level
- clever sorting may pay off
- gap between
  - weak solvers + unthoughtful DOF-ordering + unoptimised kernels (with respect to hardware) and
  - FE-gMG + clever reordering + hardware-acceleration
- is huge
- current design oblivious of FE-spaces, domain-dimension, preconditioning, grid properties, ...

Conclusion

Summary

- FE-gMG is efficient and flexible
- GPU vs. multicore CPU: close to one order of magnitude speedup
- sophisticated (sparse) preconditioners make the difference
- single-node hardware-oriented FE-gMG is ready from the solver-side, but ...

Future work

- assembly of preconditioners, system matrices, transfer-matrices still unresolved, especially for unstructured grids
- cross-effects with resorting the degrees of freedom in combination with a specific matrix storage format and associated SpMV kernel
- other related data-parallel operations: adaptive grid-deformation, ...

Acknowledgements

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