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

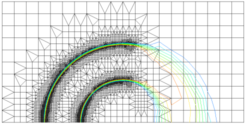
PARENG 11
Ajaccio, April 14, 2011

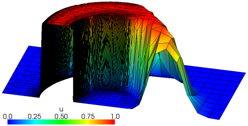
technische universität dortmund fakultät für mathematik

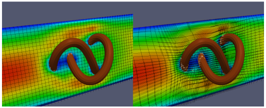
Motivation

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)





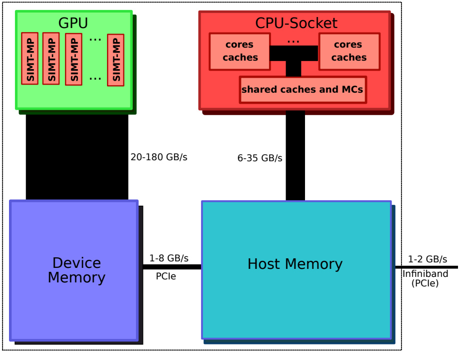


$\rightarrow$ Finite Element Geometric Multigrid enhances numerical efficiency.

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



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* $\rightarrow$ 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

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

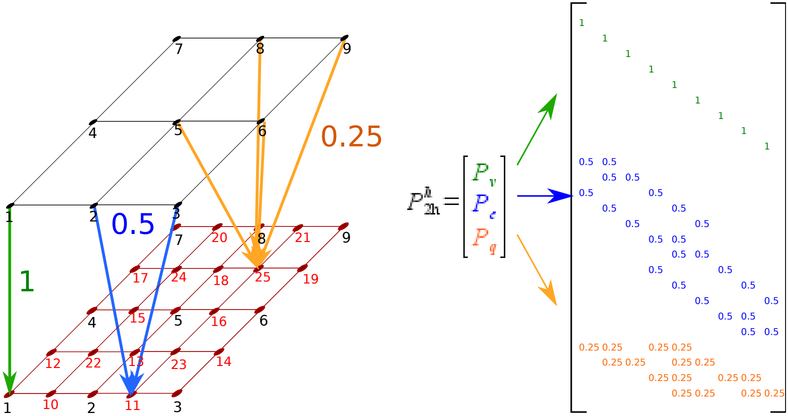
- for the basis functions of V_{2h} and $u_{2h} = \sum_{j=1}^{n} y_j \cdot \varphi_{2h}^{(j)}$ with coefficient vector y, we can write the prolongation as

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

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

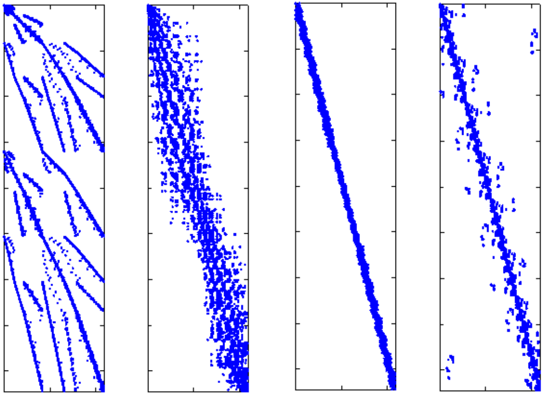
Solution approach

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



Solution approach

Grid transfer: Prolongation matrix examples



- 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 (#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

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

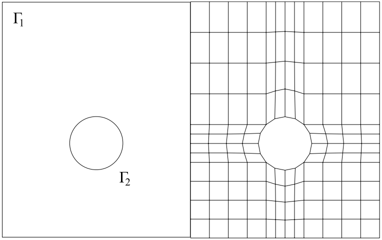
- based on the ELLPACK-R format
- $\mathbf{y} = A\mathbf{x}$ can be performed by computing each entry y_i of the result vector $\mathbf{y}$ independently (one GPU-thread per y_i)
- regular access pattern on data of $\mathbf{y}$ and A
- access pattern on $\mathbf{x}$ depends highly on sparsity pattern 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		Q_2	
	N	non-zeros	N	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 920 quadcore workstation (4 threads) / NVIDIA GeForce GTX 285 GPU

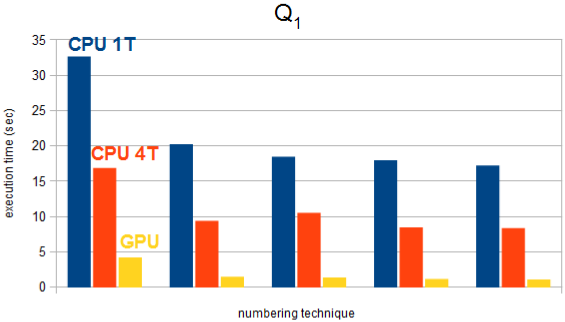
Results

Sparse matrix-vector multiply on the GPU

L	SSE	MCSSE	Q_1 speedup	CUDA	speedup	SSE	MCSSE	Q_2 speedup	CUDA	speedup
6	-	-	-	-	-	492	894	1.82	7985	8.93
7	870.81	2445.06	2.81	7441.65	3.04	467	612	1.31	8527	13.93
8	672.14	1163.98	1.73	8411.42	7.23	358	569	1.59	7942	13.96
9	506.85	988.19	1.95	7928.9	8.02	**323**	**528**	**1.63**	**7680**	**14.55**
10	**426.81**	**855.99**	**2.01**	**7925.34**	**9.26**	-	-	-	-	-
6	-	-	-	-	-	479	933	1.95	6168	6.61
7	790	2897	3.67	7247	2.5	458	883	1.93	7035	7.97
8	685	1268	1.85	8459	6.67	289	802	2.78	6470	8.07
9	445	1187	2.67	7539	6.35	**262**	**743**	**2.84**	**6288**	**8.46**
10	**399**	**1120**	**2.81**	**7314**	**6.53**	-	-	-	-	-
6	-	-	-	-	-	500	1096	2.19	6706	6.12
7	842	3299	3.92	8506	2.58	491	950	1.93	7677	8.08
8	760	1344	1.77	10403	7.74	334	897	2.69	7911	8.82
9	504	1369	2.72	11007	8.04	**330**	**836**	**2.53**	**8074**	**9.66**
10	**494**	**1372**	**2.78**	**11176**	**8.15**	-	-	-	-	-
6	-	-	-	-	-	416	841	2.02	5057	6.01
7	697	2048	2.94	5880	2.87	346	787	2.27	3820	4.85
8	497	981	1.97	4257	4.34	244	590	2.42	2468	4.18
9	348	843	2.42	2628	3.12	**160**	**366**	**2.29**	**1689**	**4.61**
10	**224**	**443**	**1.98**	**1794**	**4.05**	-	-	-	-	-
6	-	-	-	-	-	487	911	1.87	6454	7.08
7	809	3148	3.89	8049	2.56	482	852	1.77	7465	8.76
8	738	1313	1.78	9726	7.41	300	836	2.79	7776	9.3
9	471	1345	2.86	10342	7.69	**299**	**782**	**2.62**	**7903**	**10.11**
10	**465**	**1331**	**2.86**	**10553**	**7.93**	-	-	-	-	-

Results

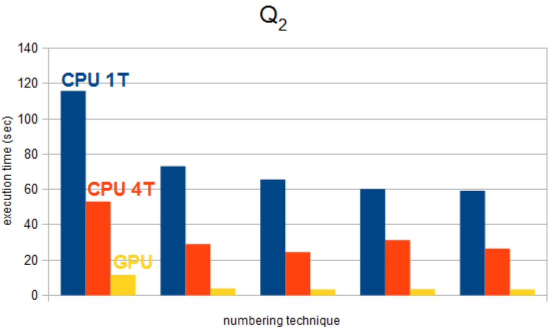
Geometric Multigrid with Jacobi preconditioning



- 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

	Q_1				Q_2			
	Jacobi		SPAI		Jacobi		SPAI	
L	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
6	-	-	-	-	0.50	0.18	0.33	0.11
7	0.11	0.09	0.10	0.07	1.63	0.38	1.22	0.25
8	0.47	0.18	0.39	0.13	8.27	0.97	5.69	0.79
9	2.30	0.42	1.68	0.34	-	-	-	-
6	-	-	-	-	0.31	0.17	0.23	0.11
7	0.12	0.10	0.10	0.07	1.35	0.36	0.94	0.24
8	0.45	0.19	0.37	0.12	6.10	1.04	3.56	0.68
9	1.97	0.45	1.69	0.37	-	-	-	-
6	-	-	-	-	0.25	0.15	0.16	0.08
7	0.09	0.09	0.09	0.07	1.10	0.32	0.61	0.16
8	0.44	0.17	0.36	0.12	**4.61**	**0.84**	**2.50**	**0.48**
9	**1.84**	**0.37**	**1.38**	**0.27**	-	-	-	-
6	-	-	-	-	0.40	0.20	0.27	0.12
7	0.12	0.09	0.12	0.07	1.63	0.51	1.10	0.31
8	0.53	0.21	0.47	0.14	**8.02**	**2.11**	**5.31**	**1.41**
9	**2.50**	**0.81**	**2.08**	**0.58**	-	-	-	-
6	-	-	-	-	0.33	0.17	0.20	0.09
7	0.14	0.10	0.11	0.07	1.31	0.34	0.95	0.21
8	0.69	0.18	0.43	0.12	5.63	0.91	3.38	0.58
9	3.88	0.39	1.91	0.34	-	-	-	-

- 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*.