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

M. Geveler[*], D. Ribbrock[*], D. Göddeke[*], P. Zajac[*], S. Turek[*]
Corresponding author: markus.geveler@math.tu-dortmund.de

[*] Institute of Applied Mathematics, TU Dortmund University, Germany.

Abstract: We describe a GPU- and multicore-oriented implementation technique for a key component of finite element based simulation toolkits for partial differential equations on unstructured grids: Geometric Multigrid solvers. We use efficient sparse matrix-vector multiplications throughout the solver pipeline: within the coarse-grid solver, smoothers and even grid transfers. Our implementation can handle several low- and high-order finite element spaces in 2D and 3D, and for representative benchmark problems, we achieve close to an order of magnitude speedup on a single GPU over a multithreaded CPU code. In addition we present preliminary results for experiments with strong smoothers for unstructured problems on the GPU, aiming at augmenting numerical and computational efficiency simultaneously.

Keywords: unstructured grids, multigrid solvers, sparse matrices, finite elements, preconditioners, GPU computing

1 Introduction

Finite element methods (FEM) offer a number of highly accurate instruments for solving partial differential equations (PDEs) ranging from high-order and non-conforming elements over arbitrarily unstructured geometries and adaptivity up to Pressure-Schur-Complement Preconditioning [1]. Geometric Multigrid solvers can solve the arising sparse linear systems in a number of iterations that is independent of the grid size and in combination with high-order Finite Elements, superlinear convergence-effects can be obtained [2].

Over the past several years, graphics processors (GPUs) have made the transition to a valuable and increasingly accepted general purpose computing resource, both on standalone workstations and in large-scale HPC installations. The main reason why GPUs excel at many HPC workloads that provide ample parallelism is that their design is fundamentally different from commodity CPU architectures: Instead of minimising the latency of a single task, they maximise the overall throughput of a large set of tasks and the chip's ratio of functional units to control logic is much more favourable. For memory-bound problems, the GPU boards' more hard-wired memory lanes allow for a higher signal quality, and thus more aggregated memory bandwidth. We refer to a recent article by Garland and Kirk [3] for technical details and a concise description of the hardware-software model of *throughput-oriented computing.* Parallelisation techniques (on the core-, chip-, and node-level) for FEM software and the hardware-aware implementation of their components have been recently studied, especially targeting acceleration by GPUs [4] but the tailoring of preconditioners for unstructured problems to fine grained parallel execution remains an open issue.

2 Solution approach

We evaluate the performance of an implementation technique for FE-gMG (finite element Geometric Multigrid) solvers for PDE problems discretised on *unstructured* grids. Our target architectures are fine-grained (manycore) GPUs - at this point, we focus entirely on the solver performance, evaluating it for different unstructured grids and finite element spaces. This is justified since in many practical scenarios, the linear solver often dominates the total execution time.

The solver's performance-critical components are based on cascades of sparse matrix-vector multiplications (SpMV): As a key component for multigrid, our *smoother* is a preconditioned Richardson iteration, using SpMV to plug in the preconditioner and for defect calculation. Furthermore, a preconditioned Conjugate Gradient *coarse grid solver* is employed which is also composed solely of SpMV and vector-vector operations. Finally, *grid transfer* can be implemented by means of SpMV, if we chose the standard Lagrange bases for two consecutively refined Q_k finite element spaces V_{2h} and V_h. In this case, we can interpolate any function $u_{2h} \in V_{2h}$ in order to prolongate it to V_h and the interpolant u_h can be calculated by evaluating u_{2h} in the corresponding nodal points $\xi_h^{(i)}$ of the basis function $\varphi_h^{(i)}$:

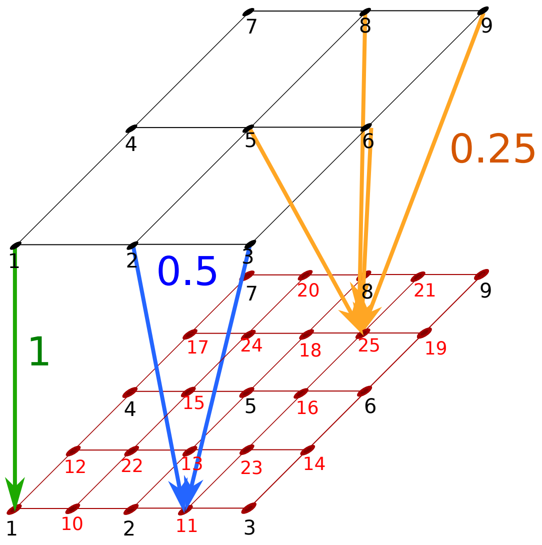
$$u_h := \sum_{i=1}^{m} x_i \cdot \varphi_h^{(i)}, \quad x_i := u_{2h}(\xi_h^{(i)})$$

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 analogously as

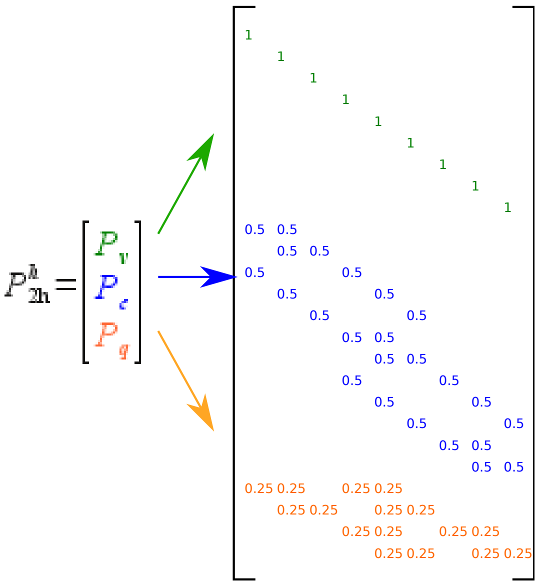
$$u_h := \sum_{i=1}^{m} x_i \cdot \varphi_h^{(i)}, \quad x := P_{2h}^{h} \cdot y$$

which results in an $m \times n$ *prolongation matrix* $(P_{2h}^{h})_{ij} = \varphi_{2h}^{(j)}(\xi_h^{(i)})$ which can be transposed to retrieve the corresponding restriction matrix. Figure 1(a) and (b) depict a basic (structured) example for a 2D quadrilateral grid and a two-level numbering scheme. In addition, Figure 1(c) exemplarily depicts the sparsity patterns of real-world prolongation matrices stemming from an unstructured grid. The storage demand of the prolongation matrix is bounded by the memory requirements of the discrete Laplace Operator on V_h and, as can be seen, the bandwidth strongly depends on the numbering technique used to label the degrees of freedom, which of course also holds true for the associated system matrices.

This simple reduction to one performance-critical kernel within the multigrid solver has surprisingly many beneficial properties: The multigrid solver needs to be implemented only once, and is completely oblivious of the underlying finite element space, and even oblivious of the dimension of the computational domain (2D, 3D). Furthermore, our implementation replaces many specialised kernels with one central, well-understood and well-optimised parallel kernel, which is favourable in terms of software maintainability and the adoption of GPUs in multigrid and finite element codes.

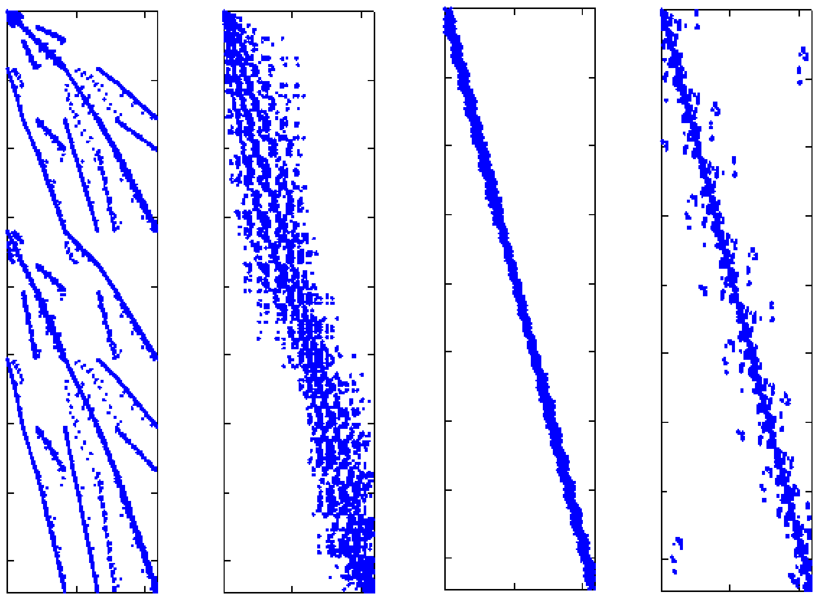


(a) 2D Q_1 interpolation: Each vertex of the fine grid corresponds to either a vertex, an edge midpoint or a quadrilateral midpoint in the coarse grid



$P_{2h}^{h} = \begin{bmatrix} P_v \\ P_e \\ P_q \end{bmatrix}$

(b) resulting block prolongation matrix for two-level numbering: Node vertices of the coarse grid need no interpolation resulting in an identity matrix block; coarse edges' corner vertices contribute with weight 1/2 to interpolation for coarse edge midpoints; corner vertices of coarse quadrilaterals are represented with weight 1/4 in the prolongation matrix



(c) sparsity patterns of real-world prolongation matrices for an unstructured grid and different strategies for the numbering of the degrees of freedom (from left to right: **2LV**, **CM**, **XYZ** and **HIE**, see Section 4)

Figure 1: Construction of prolongation matrices and sparsity pattern examples

3 Implementation

We do not utilise the 'standard' CSR format, but rather the ELLPACK-R format proposed by Vazques et al. [5]. In our experience, ELLPACK(-R) leads to significantly higher computational throughput, even for sequential code.

Based on the ELLPACK-R format, the sparse matrix-vector multiplication $\mathbf{y} = A\mathbf{x}$ can be performed by computing each entry y_i of the result vector $\mathbf{y}$ independently. In general, this results in a comparatively regular access pattern on the data of $\mathbf{y}$ and A. In contrast, the access pattern on $\mathbf{x}$ depends highly on the non-zero structure of A.

The ELLPACK-R based SpMV kernel is mapped to the GPU architecture by launching one device thread for the calculation of an entry y_i, resulting in fully coalesced memory access to the matrix and the vector $\mathbf{y}$ due to the column-major ordering used. The access to the array $\mathbf{x}$ can be cached via the texture cache on the GPU to improve efficiency. On the FERMI generation of GPUs, the device-wide L2-cache is well utilised. No synchronisation between threads is necessary. The threads in one CUDA warp do not diverge because flow instructions are not necessary which would cause serialisation. Every warp finishes execution directly when all non-zero entries in the rows of its threads are completely processed. Because of this, only warps with a high relative non-zero count in their rows execute longer compared to average warps.

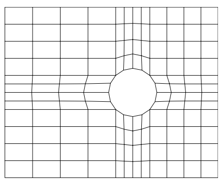
4 Benchmark setup and performance results

As a representative of an elliptic PDE, which is a key component in many practical situations, we employ a Poisson problem on a domain with two boundary components: One rectangular outer boundary component Γ_1 and one for the inner boundary wrapping a circle (Γ_2) covered by an unstructured grid, see Figure 2(a). We configure our multigrid solver to perform a V-cycle always traversing the full mesh hierarchy. Coarse grid problems are treated with a Conjugate Gradient solver using Jacobi preconditioning, it is configured to reduce the initial residual by two digits. In addition, we use two fundamentally different smoothers: Simple Jacobi smoothing on the one hand and, experimentally, preassembled *Sparse Approximate Inverses* [6] as strong preconditioners. All benchmarks are performed on an Intel Core i7 920 quadcore workstation including an NVIDIA GeForce GTX 285 GPU and we use different numbering techniques [7], two-level numbering (**2LV**), the popular Cuthill McKee numbering, **CM**, the **XYZ** technique, randomly permuted numbering (**STO**) and finally, a hierarchical approach **HIE** is used, see Figure 1. Problem sizes and resulting execution times are collected in the tables in Figure 2(b) and (c).

Our results show that the solution times can be dropped by almost one order of magnitude when using the GPU instead of the multicore CPU. In addition, the ordering of the degrees of freedom has a clear impact on the overall performance. For instance, using the best numbering technique (for this problem the **XYZ** sorting) provides a speedup factor of two over the **HIE** technique. Finally, using a sophisticated SPAI preconditioner gives a good return in reducing the computation time regardless of the higher computational costs due to the 'denser' preconditioning matrix (by almost halving the needed number of iterations).

5 Conclusion and future work

With our current work, We have shown that the approach of relying on SpMV within the Geometric Multigrid solver is efficient and flexible. Our results show that future efforts should include the theoretical and practical examination of strong smoothers for unstructured problems



(a) grid of the benchmark problem

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	-	-

(b) #degrees of freedom (N) and #non-zeros for the different refinement levels

		Q_1				Q_2			
		Jacobi		SPAI		Jacobi		SPAI	
Sort	L	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
2LV	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	-	-	-	-
CM	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	-	-	-	-
XYZ	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	-	-	-	-
STO	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	-	-	-	-
HIE	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	-	-	-	-

(c) Total execution time (sec) for Q_1 and Q_2 elements and different numbering techniques and preconditioners; the CPU results comprise four threads

Figure 2: Settings and results for the benchmark problem

and the cross-effects with resorting the degrees of freedom in combination with a specific matrix storage format and associated SpMV kernel. Other formats for sparse matrices are going to be examined in order to enhance the technique. In a further step, the assembly of the stiffness- and interpolation matrices as well as the preconditioners will be taken into account and optimised for the GPU, as well as other data parallel operations connected to the solution process, like adaptive grid deformation routines which include ray-tracing-like search per vertex.

References

[1] S. Turek, *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*, Springer, Berlin, 1999, ISBN 3-540-65433-X.

[2] M. Köster, S. Turek, “The influence of higher order FEM discretisations on multigrid convergence”, *Computational Methods in Applied Mathematics*, 6(2): 221–232, 2006.

[3] M. Garland, D.B. Kirk, “Understanding throughput-oriented architectures”, *Commununications of the ACM*, 53(11): 58–66, 2010.

[4] D. Göddeke, *Fast and Accurate Finite-Element Multigrid Solvers for PDE Simulations on GPU Clusters*, PhD thesis, TU Dortmund, Faculty of Mathematics, 2010.

[5] F.M. Vazquez, G. Ortega, J.J. Fernandez, E.M. Garzon, “Improving the Performance of the Sparse Matrix Vector Product with GPUs”, in *International Conference on Computer and Information Technology (CIT 2010)*, pages 1146–1151, 2010.

[6] O. Bröker, M.J. Grote, “Sparse approximate inverse smoothers for geometric and algebraic multigrid”, *Applied Numerical Mathematics*, 41(1): 61–80, 2002.

[7] S. Turek, “On Ordering Strategies in a Multigrid Algorithm”, in *Proc. 8th GAMM–Seminar*, Volume 41 of *Notes on Numerical Fluid Mechanics*, 1992.