Fine-grained Parallel ILU Preconditioners with Fill-ins for Multi-core CPUs and GPUs

List of authors:

D. Lukarski [1]

V. Heuveline [2] J.-P. Weiss [3]

Numerical simulation and its huge computational demands require a close coupling between efficient mathematical methods and their hardware-aware implementation on emerging and highly parallel computing platforms. The paradigm shift towards manycore parallelism not only offers a high potential of computing capabilities but also comes up with urgent challenges in designing scalable, portable, and flexible software solutions. The latter point is closely related to adaptation, variation, and restructuring of algorithms and numerical schemes in order to be compliant with coarse- and fine-grained parallelism, hierarchical memory subsystems, heterogeneous platforms, and any kind of communication bottleneck. Numerical codes should not only be efficient and robust, but also future-proof with respect to the current dynamic landscape of hardware platforms and parallel programming environments. The goal of our work is to treat problems with great impact on science and humanities, e.g. in the domain of meteorology, medical engineering and energy research, can be modeled and solved. With efficient preconditioners the bounds of solvability can be pushed further beyond.

Complex problem domains and the need for high accuracy typically result in huge systems that are sparse, closely coupled, and with bad condition numbers. A classical choice for iterative solvers are Krylov subspace methods like *conjugate gradient* (CG) for symmetric and positive definite systems and the *generalized minimal residual* (GMRES) method. In both cases, the number of iterations depends on the condition number and grows polynomially in the problem size. Good preconditioners should fullfil several properties. First, they should mitigate the costs in terms of necessary iterations by restructuring the problem matrix and affecting its spectrum and condition number. Second, since in each iteration step an additional linear system has to be solved, the additional effort should not outweigh achieved benefits. Third – and this point is becoming much more important due to the development towards manycore computing platforms – the preconditioner has to comprise a high degree of parallelism. Fourth, the preconditioner should be applicable to a large class of problems where only minimal additional information is available on specific matrix properties. The latter point is particularly important for the inclusion of preconditioners into widely applicable solver suites. As experience shows, parallelism for preconditioners based on reduced couplings comes at the expense of reduced preconditioning efficiency. So preconditioning also means to find a trade-off between several aspects.

The HiFlow[3] finite element software tackles productivity and performance issues by its con-

[1]SRG New Frontiers in High Performance Computing, Engineering Mathematics and Computing Lab (EMCL), Karlsruhe Institute of Technology (KIT), Germany

[2]Engineering Mathematics and Computing Lab (EMCL), Karlsruhe Institute of Technology (KIT), Germany

[3]SRG New Frontiers in High Performance Computing, Engineering Mathematics and Computing Lab (EMCL), Karlsruhe Institute of Technology (KIT), Germany

ceptual approach [1]. Efficient numerical solvers are built on the basis of unified interfaces to different hardware platforms and parallel programming approaches in order to obtain modular and portable software solutions on emerging systems with heterogeneous configuration. With the concept of object-orientation in C++, the HiFlow^3 finite element software provides a flexible, generic, and modular framework for building efficient parallel solvers and preconditioners for partial differential equations of various types. It comprises a complete set of BLAS 1 and 2 linear algebra routines and data access routines for all platforms. For scalable and portable parallelism all solvers are built upon a two-level library. The linear algebra toolbox (LAtoolbox) orchestrates computations and communications across nodes by an MPI layer. Underneath, the local multi-platform linear algebra toolbox (lmpLAtoolbox) provides highly efficient and optimized implementations with backends to various platforms (e.g. CUDA, OpenMP, OpenCL) where parallelism is expressed by means of data parallelism. Further platforms can be added easily. The user can build modular solutions by utilizing unified interfaces on an abstract level. The same source code for the solver and the preconditioners can be used on several platforms including GPUs. The final choice of platform and particular implementation is taken at run-time. This approach enables user-friendly and scalable solutions in heterogeneous environments.

In this work we describe our efforts for building efficient preconditioning schemes for Krylov subspace iterative solvers in the context of the HiFlow^3 parallel finite element solver package [2]. Our main intention is to construct preconditioners with a high degree of parallelism. Structure and organisation of the HiFlow^3 software into several modules and communication layers with unified interfaces for the programmer allows to write a single code base that can be run across a wide range of parallel platforms including multi-core CPUs, graphics processing units (GPUs), and OpenCL-capable accelerators. Since GPU code should be scalable to thousands of threads, our approach is to identify parallelism on the level of blocks within block-decompositions and not only on the level of non-scalable parallel execution of blocks.

We consider preconditioners in block form based on additive matrix splittings like e.g. Gauß-Seidel and SOR, and multiplicative decompositions like incomplete LU (ILU). All our considered preconditioners are based on the block-wise decomposition into small sub-matrices. In both scenarios, typically a large amount of forward and backward sweeps in triangular solvers need to be performed. In order to harness parallelism within each block of the decomposition we use matrix reordering techniques like multi-coloring. For splitting-based methods (Gauß-Seidel, SOR) and ILU(0) without fill-ins the original matrix pattern is preserved. But before solving the preconditioned system, the matrix is reorganized such that diagonal blocks are diagonal itself. Then, inversion of the diagonal blocks is just an easy vector operation [3]. In the multi-coloring approach sets of independent nodes – called colors – are identified. The number of colors depends on the mesh, the choice of finite element spaces, and on the problem dimension. A performance analysis for a convection-diffusion problem solved by Q1 and Q2 elements in two and three dimensions is presented in [4]. The scalability and efficiency of our approach has been proven in [3, 5].

Furthermore, we allow fill-ins in the ILU(p) method for achieving higher level of coupling with increased efficiency. Here, we consider two algorithms for parallelism: *level-scheduling*

method [6] and our *adding auxiliary colors* algorithm. In our experiments, we apply both techniques to matrices with a symmetric pattern of non-zero elements. The first method is used as a postprocessing method following the factorization where the level of parallelism for the elimination processes in the forward and backward sweeps is determined and utilized. However, this method produces very small blocks for many problem classes, i.e. the degree of parallelism is low. An improvement with respect to the level of parallelism can by obtained by applying multi-coloring to the original matrix in first place. As a second approach we have developed our new technique that adds further colors to the original matrix resulting in diagonal matrices on the diagonal blocks combined with ILU(p) with additional fill-in. This approach provides a much higher degree of parallelism compared to the level-scheduling method.

We present a detailed performance analysis on several multi-core platforms and GPU configurations – proofing viability and benefit of our solution. In combination with HiFlow^3's cross-platform portable concept, we provide flexible and scalable iterative solvers combined with efficient parallel preconditioners based on splitting methods and parallel ILU(p).

References

[1] H. Anzt, et. al, HiFlow^3 – A Flexible and Hardware-Aware Parallel Finite Element Package *EMCL Preprint http://www.emcl.kit.edu/preprints/emcl-preprint-2010-06.pdf* 2010

[2] HiFlow^3 – Parallel Finite Element Package, EMCL, Karlsruhe Institute of Technology, http://www.hiflow3.org, 2010

[3] V. Heuveline and D. Lukarski and J.-P. Weiss Scalable Multi-Coloring Preconditioning for Multi-core CPUs and GPUs *UCHPC'10, Euro-Par 2010 Parallel Processing Workshops*

[4] V. Heuveline, C. Subramanian, D. Lukarski, J.-P. Weiss Parallel Preconditioning and Modular Finite Element Solvers on Hybrid CPU-GPU Systems submitted, 2010

[5] V. Heuveline, C. Subramanian, D. Lukarski, J.-P. Weiss A Multi-Platform Linear Algebra Toolbox for Finite Element Solvers on Heterogeneous Clusters *PPAAC'10, IEEE Cluster 2010 Workshops*

[6] Y. Saad Iterative Methods for Sparse Linear Systems *Society for Industrial and Applied Mathematics* Philadelphia, PA, USA 2003