Self-Adaptive Multiprecision Preconditioners on Multicore and Manycore Architectures

Hartwig Anzt[1], Dimitar Lukarski[2], Stanimire Tomov[1], and Jack Dongarra[1]

[1] Innovative Computing Lab, University of Tennessee, Knoxville, USA.
hanzt@icl.utk.edu, tomov@cs.utk.edu, dongarra@eecs.utk.edu
[2] Department of Information Technology, Uppsala University, Sweden.
dimitar.lukarski@it.uu.se

Abstract. Based on the premise that preconditioners needed for scientific computing are not only required to be robust in the numerical sense, but also scalable for up to thousands of light-weight cores, we argue that this two-fold goal is achieved for the recently developed self-adaptive multi-elimination preconditioner. For this purpose, we revise the underlying idea and analyze the performance of implementations realized in the PARALUTION and MAGMA open-source software libraries on GPU architectures (using either CUDA or OpenCL), Intel's Many Integrated Core Architecture, and Intel's Sandy Bridge processor. The comparison with other well-established preconditioners like multi-coloured Gauss-Seidel, ILU(0) and multi-colored ILU(0), shows that the twofold goal of a numerically stable cross-platform performant algorithm is achieved.

1 Introduction

When solving sparse linear systems iteratively, e.g. via Krylov subspace solvers, using preconditioners is often the key to reducing the time needed to obtain a sufficiently accurate solution approximation. For this reason, significant effort is spent on the development of efficient preconditioners, usually optimized for one particular problem. However, the theoretical derivation of methods improving the convergence characteristics is often not sufficient, as the algorithms have to be implemented and parallelized on the respectively used hardware platform. The use of accelerator technology, like graphics processing units (GPUs) or Intel Xeon Phi Coprocessors (known also as Many Integrated Core Architectures, or MIC), in scientific computing centers requires a combination of deep mathematical background knowledge and software engineering skills to develop suitable methods. The challenge is to combine the robustness and efficiency of the preconditioner scheme with the scalability of the implementation up to hundreds and thousands of light-weight computing cores. The non-uniformity of the high-performance computing landscape introduces additional complexity to this endeavor, and complex sparse linear algebra algorithms that are designed to efficiently exploit one specific architecture often fail to leverage the computing power of other technologies. In this paper we show that, for the recently developed self-adapting and multi-precision preconditioner [10], the two-fold goal of deriving a

numerically robust method featuring cross-platform scalability is achieved. While the use of different floating point precision formats, and the combination of dense and sparse linear algebra operations, may challenge cross-platform suitability, we show that the self-adaptive mixed precision multi-elimination method can efficiently exploit different hardware architectures and is highly competitive to some of the most commonly used preconditioners. While the implementation of the algorithm is realized using the PARALUTION [8] and MAGMA [5] open source software libraries, both known to be able to efficiently exploit the computing power of accelerators, the hardware systems used in our experiments represent some of the most popular technologies used in current HPC platforms. The rest of the paper is structured as follows. First, we provide some details about the self-adaptive mixed precision multi-elimination preconditioner and the implementation we use. Next, we summarize some characteristics of the many-core accelerators we target in our experiments and introduce the test matrices we use for benchmarking. We then evaluate the performance of the mixed precision multi-elimination preconditioner, embedded in a Conjugate Gradient solver on the different hardware systems, and compare against other well-known preconditioners. Finally, we summarize some key findings and provide ideas for future research.

2 Self-Adaptive Multi-Elimination Preconditioner

Among the most popular preconditioners is the class based on the incomplete LU factorization (ILU) [15]. Although using ILU without fill-ins can lead to appealing convergence improvement to the top-level iterative method, it may also fail due to its rather rough approximation properties, e.g., when solving linear systems arising from complex applications like computational fluid dynamics [14]. To enhance the accuracy of the preconditioner, one can allow for additional fill-in in the preconditioning matrix, resulting in the (ILU(m) scheme, see [15]). Additional fill-in usually reduces the amount of parallelism in ILU(m) compared to ILU(0), but there are a number of techniques designed to retain it, such as the level-scheduling techniques [15, 11] or the multi-coloring algorithms for the ILU factorization with levels based on the power(q)-pattern method [9]. Another workaround is given by the idea of multi-elimination [14, 16], which is based on successive independent set coloring [6]. The motivation is that in a step of the Gaussian elimination, there usually exists a large set of rows that can be processed in parallel. This set is called the independent set. For multi-elimination, the idea is to determine this set, and then eliminate the unknowns in the respective rows simultaneously, to obtain a smaller reduced system. To control the sparsity of the factors, multi-elimination uses an approximate reduction based on a standard threshold strategy. Recursively applying this step, one obtains a sequence of linear systems with decreasing dimension and increasing fill-in. On the lowest level, the system must be solved, e.g., either by an iterative method, or by a direct solver based on an LU factorization. Recently, a multi-elimination preconditioner, using an adaptive level depth in combination with a direct solver

based on LU factorization , was proposed in [10]. The advantage of this approach is that the once computed LU factorization for the bottom-level system can be reused in every iteration step, and the ability to utilize a lower precision format in the triangular solves allows for leveraging the often superior single precision performance of accelerators like GPUs. While we only shortly recall the central ideas of the multi-elimination concept, a detailed derivation can be found in [14]. The underlying scheme is to use permutations P to bring the original matrix A, of the system $Ax = b$ that we want to solve, into the form

$$PAP^T \equiv \begin{pmatrix} D & F \\ E & C \end{pmatrix},$$

where D is preferably a diagonal or at least an easy to invert matrix, so that

$$PAP^T \equiv \begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ ED^{-1} & I \end{pmatrix} \times \begin{pmatrix} D & F \\ 0 & \hat{A} \end{pmatrix} \text{ with } \hat{A} = C - ED^{-1}F \qquad (1)$$

is easy to compute [10]. One way to achieve this is by using an independent set ordering [6, 7, 18, 13], where non-adjacent unknowns of the original matrix A are determined. Recursively applying this idea and using some threshold strategy to control the fill-in one obtains a sequence of successively smaller problems. To control the increasing density of $\hat{A}$, we propose a self-adapting algorithm which determines an appropriate sequence depth and a fill-in threshold based on the average of all non-zero entries of $\hat{A}$. In the iteration phase (see Figure 1) the sequence of transformations must also be applied to the right-hand side and to the solution approximation. This is achieved by applying the decomposition [14]

$$x := \begin{pmatrix} \hat{y} \\ \hat{x} \end{pmatrix}$$

and computing, according to the partitioning in (1), the forward sweep as [14]: $\hat{x} := \hat{x} - ED^{-1}y$. Consequently, backward solution for y_j hence becomes $y := D^{-1}(y - F\hat{x})$. On the lowest level the linear system must be solved, either again via an iterative method, or, like suggested in [10] via triangular solves (in single precision), using a beforehand computed factorization. Algorithmic details, as well as a comparison between single and double precision triangular solves, can be found in [10]. As the level-depth is not preset but determined during the recursive factorization sequence using thresholds for drop-off and the direct solve size, the algorithm is self-adapting to a specific problem.

3 Hardware and Software Issues

Target Platforms. The trend to introduce accelerator technology into high performance computers is reflected in the top-ranked computer systems in both the performance-oriented TOP500, and the resource-aware Green500 list (see [3] and [1], respectively). While in recent years the usage of GPUs from different vendors drew attention, Intel responded with the development of the MIC

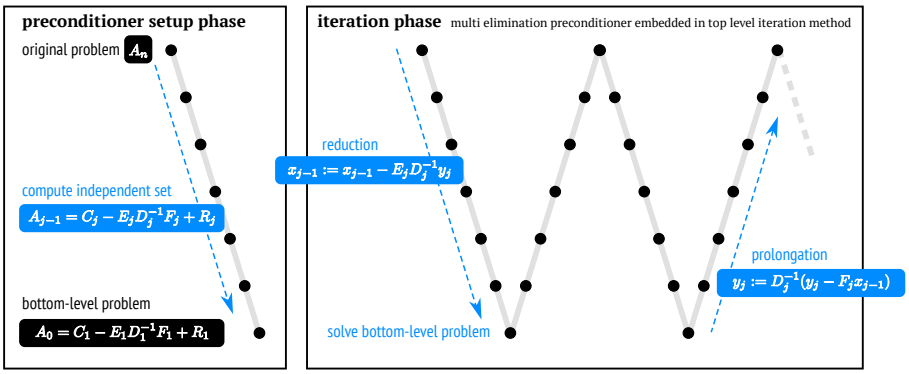

Fig. 1: Visualization of the multi-elimination scheme denoting the system matrix of the original problem A_n and a sequence of successively smaller problems down to the bottom-level system matrix A_0.

architecture (and in the November 2013 Top500 list, the number one ranked supercomputer was based on MICs). For the future, even more diversity may be expected as precise plans for building systems based on the low-power ARM technology already exist [2]. Despite attempts like OpenCL [17] and OpenACC [4], unfortunately no cross-platform language that allows for efficient usage of the different accelerator architectures currently exists. Therefore, it usually remains a burden to the software developer to implement algorithms for a specific target architecture using a suitable programming language for the respective hardware. Especially for numerical linear algebra algorithms, the algorithm-specific properties often make the implementation on different architectures challenging. To determine whether the challenge of deriving a cross-platform performant method is achieved for the recently developed self-adaptive multi-elimination preconditioner we introduced in the last section, we benchmark it on different multi- and many-core systems listed along with some key characteristics in Table 1.

Acronym	System	Performance Peak	Memory	Bandwidth
ISB	2× Intel Xeon E5-2670 (Sandy Bridge)	333 GFlop/s	65 GB	2× 25.5 GB/s
K40	NVIDIA Tesla K40c	1,682 GFlop/s	12 GB	288 GB/s
AMD	AMD Radeon HD 7970 (Tahiti)	947 GFlop/s	3 GB	264 GB/s
MIC	Intel Xeon Phi 7110P	1,238 GFlop/s	16 GB	352 GB/s

Table 1: Key characteristics of the target architectures.

The implementation of the preconditioner, as well as the other methods we compare against in Section 4, is realized using the PARALUTION [8] (version 0.4.0) and MAGMA [5] (version 1.4) open-source software libraries. The framework and the CPU solver implementations are based on C/C++, while the GPU-

accelerated implementations use either CUDA [12] version 5.5 for the NVIDIA GPUs, or OpenCL [17], version 1.2 and clAmdBlas 1.11.314 for AMD GPUs. The MIC implementation, similar to GPU's, treats the MIC as an accelerator/coprocessor and is based on OpenMP and the BLAS operations provided in Intel's MKL 11.0, update 5.

Solver Parameters. All experiments solve the linear system $Ax = b$ where we set the initial right-hand-side to $b \equiv 1$, start with the initial guess $x \equiv 0$ and run the iteration process until we achieve a relative residual accuracy of $1e-6$. In the preprocessing phase of the multi-elimination, the identification of an independent set via a graph algorithm is handled by the CPU of the host system; the factorization process itself, including the permutation and the generation of the lower-level systems via a sparse matrix-matrix multiplication is implemented on the GPU.

Test Matrices. For the experiments, we use a set of symmetric, positive definite (SPD) test matrices taken either from the University of Florida matrix collection (UFMC)[3], Matrix Market[4], or generated as finite difference discretization (LAPLACE). The test matrices are listed along with some key characteristics in Table 2. Although we target only SPD systems, we use ME-ILU factorization due to the fact that the IC requires non-zero diagonal elements. Positive diagonal entries for the IC can be obtained with non-symmetric permutation. This is not applicable because the multi-elimination uses maximal independent set (MIS) algorithm which produces a symmetric permutation.

matrix	#nonzeros (nnz)	Size (n)	nnz/n
APACHE	4,817,870	715,176	6.74
ECOLOGY	4,995,991	999,999	5.00
G2_CIRC	726,674	150,102	4.83
G3_CIRC	7,660,826	1,585,478	4.83
LAPLACE	4,996,000	1,000,000	4.99
OFFSHORE	4,242,673	259,789	16.33
STOCF	21,005,389	1,465,137	14.34
THERMAL	8,580,313	1,228,045	6.99

Table 2: Description and properties of the test matrices.

4 Performance on Emerging Hardware Architectures

In Table 3 we list the runtime of the iteration phase of the self-adaptive mixed precision multi-elimination implementation on different hardware platforms. With the number of iterations constant over the architectures, the performance is determined by the available computing power and the efficiency of the programming model to exploit it. The results reveal that the best performance is achieved

[3] UFMC; see `http://www.cise.ufl.edu/research/sparse/matrices/`

[4] see `http://math.nist.gov/MatrixMarket/`

matrix	#iters	ISB	K40	AMD	MIC
APACHE	293	15.43	3.04	15.46	8.35
ECOLOGY	799	63.57	10.98	-	23.17
G2_CIRC	359	11.11	2.49	15.99	5.29
G3_CIRC	512	20.30	5.42	18.23	16.18
LAPLACE	338	9.13	3.41	14.31	10.01
OFFSHORE	1314	93.67	9.59	58.23	14.88
STOCF	4388	178.56	52.06	-	115.05
THERMAL	916	57.41	13.93	59.20	35.73

Table 3: Iteration count and runtime (in seconds) of the Conjugate Gradient solver preconditioned with the self-adaptive mixed precision multi-elimination (MPME) preconditioner for different test matrices and hardware architectures.

using the CUDA implementation on the NVIDIA Kepler architecture. The MIC implementation fails to achieve the K40 performance, but is in most cases superior to ISB . Switching from the CPU to the OpenCL programming model on the AMD platform accelerates the solver execution only for some problems, and even for those, the performance is significantly lower than on the NVIDIA GPU. Furthermore, the smaller memory size of the AMD architecture prevents it from handling all problems. While this performance drop may suggest that mixed precision multi-elimination is not suitable for OpenCL on AMD architectures, the runtime results for other preconditioner choices in Table 4 indicate that this behavior is not a singularity. None of the implementations using the OpenCL-AMD framework achieves performance competitive to the CUDA results on the Kepler K40. Finally, we want a comparison between the different preconditioners. In Figure 2 we compare the performance of the plain CG with the implementations preconditioned by multi-colored Gauss-Seidel, ILU(0), multi-colored ILU(0), and the developed mixed precision multi-elimination with the runtime normalized to the respective best implementation. From the results we can determine that the mixed precision multi-elimination is not suitable for the small G2_CIRC problem, but reduces the runtime significantly in the STOCF case. Overall, the developed self-adaptive preconditioner is competitive compared to the well-established methods.

5 Summary and Future Research

In this paper we have analyzed the cross-platform suitability of the recently developed mixed precision multi-elimination preconditioner using self-adaptive level depth. We have analyzed the method's performance characteristics using different hardware platforms and compared the runtime with some of the most popular preconditioners. The numerical robustness combined with platform-independent scalability makes the method a competitive candidate when choosing a preconditioner for solving linear problems in scientific computing. Future research will target the question of how to leverage the computing power of platforms equipped with multiple, not necessarily uniform, accelerators.

	CG					MCGS-CG				
matrix	#iters	ISB	K40	AMD	MIC	#iters	ISB	K40	AMD	MIC
APACHE	3971	16.60	5.02	15.39	10.12	1677	15.45	5.22	14.90	12.56
ECOLOGY	5392	24.17	8.20	19.74	15.35	2784	27.50	8.94	19.83	19.17
G2_CIRC	8911	5.32	3.76	13.83	10.27	907	1.61	1.29	5.47	4.80
G3_CIRC	12658	107.56	29.67	60.86	77.15	1329	28.55	9.01	15.82	22.35
LAPLACE	1633	8.03	2.53	5.87	4.73	817	9.00	2.63	5.76	5.60
OFFSHORE		*– no convergence –*				628	10.19	4.92	15.03	16.79
STOCF		*– no convergence –*				66042	2200.46	1187.59	2679.99	2678.97
THERMAL	4589	53.06	9.63	28.44	30.52	2151	39.27	18.33	36.28	52.68

	ILU0-CG					MCILU0-CG				
matrix	#iters	ISB	K40	AMD	MIC	#iters	ISB	K40	AMD	MIC
APACHE	643	25.56	9.63	-	-	1438	16.37	4.07	11.55	9.80
ECOLOGY	1700	74.86	64.03	-	-	2854	38.18	8.35	18.72	18.09
G2_CIRC	481	3.28	6.15	-	-	857	1.54	1.12	4.37	3.94
G3_CIRC	680	51.73	33.77	-	-	1242	25.06	7.71	13.20	19.24
LAPLACE	537	23.18	19.30	-	-	817	8.49	2.37	5.29	5.29
OFFSHORE	365	13.83	23.22	-	-	487	6.88	3.57	8.54	11.72
STOCF	2364	368.36	158.37	-	-	16740	544.91	290.38	634.35	624.89
THERMAL	1945	188.58	54.13	-	-	2095	42.33	16.79	30.57	49.53

Table 4: Iteration count and runtime (in seconds) of the unpreconditioned Conjugate Gradient solver (labelled CG) and the implementations using a multi-coloured Gauss-Seidel preconditioner (labelled MCGS-CG), a ILU-0 and a multi-colored ILU-0 preconditioner (labelled ILU0-CG and MCILU0-C, respectively) for different test matrices and hardware architectures.

Acknowledgments

This work has been supported by the Linnaeus centre of excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center, DOE grant #DE-SC0010042, NVIDIA, and the NSF grant # ACI-1339822.

References

1. The green 500 list, `http://www.green500.org/`.
2. The Mont Blanc Project, `http://montblanc-project.eu`.
3. The top 500 list, `http://www.top.org/`.
4. O. Corp. Openacc 2.0a spec - revised august 2013, June 2013.
5. I. C. Lab. Software distribution of MAGMA version 1.4. `http://icl.cs.utk.edu/magma/`, 2013.
6. M. R. Leuze. Independent set orderings for parallel matrix factorization by gaussian elimination. *Parallel Computing*, 10(2):177 – 191, 1989.
7. M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.
8. D. Lukarski. PARALUTION project. http://www.paralution.com/.
9. D. Lukarski. *Parallel Sparse Linear Algebra for Multi-core and Many-core Platforms - Parallel Solvers and Preconditioners*. PhD thesis, Karlsruhe Institute of Technology (KIT), Germany, 2012.

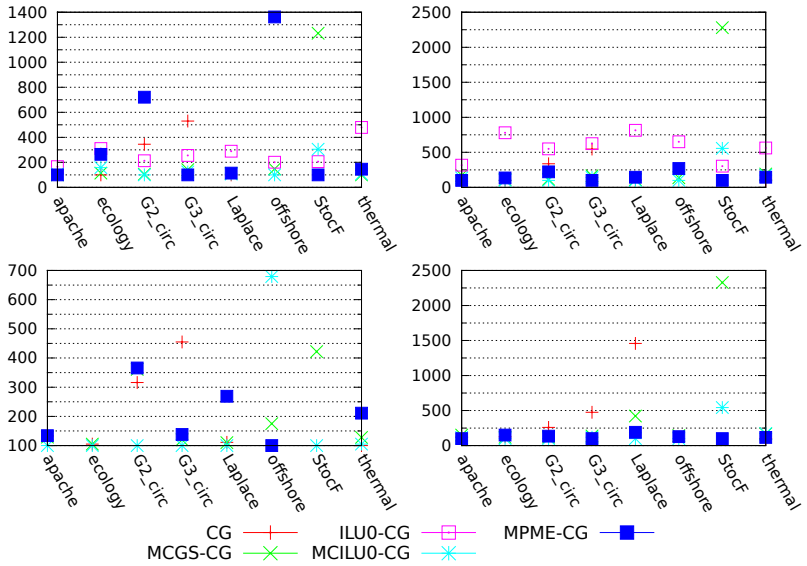

Fig. 2: Relative runtime [%] of the diffferent implementations with respect to the best method on the Intel Sandy Bridge CPU (left top) Kepler K40 GPU (right top), AMD Radeon 7900 and Intel's Many Integrated Core Architecture (left and right bottom).

10. D. Lukarski, H. Anzt, S. Tomov, and J. Dongarra. Multi-Elimination ILU Preconditioners on GPUs. Technical Report UT-CS-14-723, Innovative Computing Laboratory, University of Tennessee, 2014.
11. M. Naumov. Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU. Technical report, NVIDIA, 2011.
12. NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, 2.3.1 edition, August 2009.
13. J. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425 – 440, 1986.
14. Y. Saad. Ilum: A multi-elimination ilu preconditioner for general sparse matrices. *SIAM J. Sci. Comput*, 17:830–847, 1999.
15. Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
16. Y. Saad and J. Zhang. Bilum: Block versions of multi-elimination and multi-level ilu preconditioner for general sparse linear systems. *SIAM J. SCI. COMPUT*, 20:2103–2121, 1997.
17. J. E. Stone, D. Gohara, and G. Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test*, 12(3):66–73, May 2010.
18. L. Yao, W. Cao, Z. Li, Y. Wang, and Z. Wang. An improved independent set ordering algorithm for solving large-scale sparse linear systems. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2010 2nd International Conference on*, volume 1, pages 178–181, 2010.