Efficiency of general Krylov methods on GPUs – An experimental study

Hartwig Anzt, Jack Dongarra
University of Tennessee
Knoxville, TN, USA
{*hanzt,dongarra*}*@icl.utk.edu*

Moritz Kreutzer, Gerhard Wellein
Friedrich-Alexander University of
Erlangen-Nuremberg
Erlangen, Germany
{*moritz.kreutzer,gerhard.wellein*}*@fau.de*

Martin Köhler
Max Planck Institute for
Dynamics of Complex Technical Systems,
Magdeburg, Germany
koehlerm@mpi-magdeburg.mpg.de

***Abstract*—This paper compares different Krylov methods based on short recurrences with respect to their efficiency when implemented on GPUs. The comparison includes BiCGSTAB, CGS, QMR, and IDR using different shadow space dimensions. These methods are known for their good convergence characteristics. For a large set of test matrices taken from the University of Florida Matrix Collection, we evaluate the methods' performance against different target metrics: convergence, number of sparse matrix-vector multiplications, and execution time. We also analyze whether the methods are "orthogonal" in terms of problem suitability. We propose best practices for choosing methods in a "black box" scenario, where no information about the optimal solver is available.**

***Keywords*-Krylov solver, GPU, IDR(s), BiCGSTAB, CGS, QMR, algorithmic bombardment**

I. Introduction

Krylov methods are a popular choice for iteratively solving large, sparse linear systems. Their often superior convergence properties compared to component-wise relaxation methods, and their ability to benefit from preconditioning make them attractive from the theoretical point of view. At the same time, their generic construction as a combination of sparse matrix vector products, vector operations, and reductions makes them attractive for parallel execution, e.g., on manycore architectures like GPUs. Therefore, linear algebra software libraries like cuSPARSE, MAGMA, Paralution, or ViennaCL offer a large variety of Krylov solvers to users [1], [2], [3], [4].

In recent decades, significant advances were made in designing efficient Krylov methods. Some of these methods are optimized for specific matrix properties, with the Conjugate Gradient (CG [5]) algorithm suitable for symmetric, positive definite problems being the most popular example. Other Krylov solvers work well for a wide range of problems. However, for a problem with unknown origin it is difficult to identify the best method. One strategy to overcome this challenge is to run multiple Krylov methods simultaneously [6]. The advantage of this "algorithmic bombardment" is that convergence is determined by the iteration count of the most suitable solver included in the multi-iteration. Krylov methods sharing the same algorithmic structure can be combined efficiently into a multi-iteration method. More precisely, the individual sparse matrix vector products, generating the Krylov spaces, can be combined into a sparse matrix times block-vector product, and the reduction operations like dot product can be interleaved for minimizing the number of synchronization points.

This paper intends to explore, experimentally, the Krylov solver landscape, and provide an overview of how well the distinct Krylov methods work on GPUs. In particular, we are interested in investigating the robustness, and in identifying methods that can be considered "orthogonal" in terms of suitability for different problem classes. The software basis for this study is the MAGMA open source software library [2] containing a large variety of Krylov solvers implemented on GPUs. Test matrices are taken from the University of Florida Matrix Collection [7]. Although we focus exclusively on one single NVIDIA GPU, the findings on convergence and stability carry beyond that architecture. In terms of the solvers' hardware efficiency, similar results can be expected for all relevant modern hardware architectures, with the performance scaled to the respectively higher or lower memory bandwidth.

The rest of the paper is structured as follows. In Section II we give an overview of related work and introduce the Krylov methods we include in our evaluation. Section III gives details about how we define our matrix test suite, the libufget tool we use to access the matrices in the University of Florida Matrix Collection, the MAGMA software package we employ for our evaluation, and the hardware we target. Section IV is the main contribution of this paper, presenting the experimental results along with an analysis. We conclude in Section V.

II. Background

A. Related work

Designing Krylov methods that are efficient in solving non-symmetric systems is an active field of research. A comprehensive survey on Krylov solvers developed before 1991 can be found in [8]. Since then, a large number of new methods has been developed, often with improved convergence and stability properties for a certain problem class. New methods often arise as a combination of already existing algorithms. Unfortunately, however, it looks like no overall best Krylov solver exists, as —for each method—

it is possible to find a problem class where a different solver is superior [9]. And the growing variety of solvers to choose from presents the challenge of selecting a suitable method to the user. One possible workaround is to base the selection on a theoretical analysis, matching the problem characteristics to the algorithm properties. This, however, requires thorough knowledge of the mathematical theory. A very comprehensive overview of Krylov methods, their algorithmic design, and mathematical properties is given in [10]. For problems with an origin in partial differential equations, Saad outlines in [5] the complete simulation path: from the partial differential equations with their numerical properties, via the discretization methods generating linear systems of equations, to the efficiency of Krylov methods when solving these. However, a deep theoretical analysis may only be justified if a significant amount of work is spent in the solution process, e.g., if an application requires solving a sequence of linear systems with the same characteristics. If the origin of the problem and some key characteristics are known, a different approach is to base the selection on empirical knowledge: There exist multiple studies comparing the efficiency of different Krylov methods for specific problems, see [11], [12], [13], [14].

In a "black-box scenario," there is no information about the problem characteristics available, permitting the reuse of a theoretical analysis. The only goal is to make a good guess for this single run. For this black-box scenario, we compare a set of Krylov solvers with respect to their convergence properties, and their efficiency when implemented on GPU hardware. In contrast to previous studies, we base the analysis not on a specific problem class, but on a large set of test matrices available in the University of Florida Matrix Collection (UFMC, [7]). Our goal is to provide suggestions for which method to choose if little or no information about the linear system is available.

B. General Krylov solvers attractive for GPU implementation

Among the most popular Krylov solvers for general systems are GMRES, BiCG, BiCGSTAB, CGS, QMR, TFQMR, and IDR(s). Except for GMRES, all these are based on short recurrence formulation [15]. This means that in each iteration, only a few of the latest basis vectors are required to generate the new basis vector. For GMRES, each new basis vector is orthogonalized against all previous basis vectors. Its computational cost is thus increasing with the iteration count. Also, GMRES requires storing all previous basis vectors. This can be unattractive for problems requiring a high number of iterations. A workaround that avoids the explosion in computational cost and memory footprint is the restarted GMRES. It truncates the orthogonalization process to a few, last basis vectors that are stored explicitly. The restart parameter bounding the number of orthogonalizations is a trade-off between the numerical properties and the computational / storage cost. However, restart parameters larger than 20 are often advisable for smooth convergence, and a small variance in the restart parameter can have a significant impact on the solver's convergence. Hence, the restarted GMRES usually has a memory footprint significantly larger than the other methods listed. Given that the memory size of GPUs is often much smaller than main memory of the host system, GMRES is only attractive for GPU acceleration if the restart parameter is matched to the characteristics of the hardware and the linear system. This motivates one to exclude GMRES in an analysis targeting a large set of problems with different characteristics.

BiCG arises as a non-symmetric variant of the CG algorithm, which is known to be very efficient for symmetric, positive definite (spd) systems [5]. BiCG, however, carries some unattractive properties: it requires multiplication with the transpose of the system matrix, often has non-smooth convergence, and does not implicitly compute an iterative residual. In terms of stability, BiCG suffers from two potential breakdown scenarios: pivot breakdown and, in case of a non-symmetric system matrix, Lanczos breakdown [16]. This has motivated efforts to modify the BiCG method in favor of the desirable properties: avoiding breakdown; avoiding use of the transpose; smooth convergence; and an implicit residual. An effort to avoid pivot breakdown is the "quasi-minimal residual" (QMR) method developed by Freud et al. [17]. QMR does not resolve the Lanczos breakdown that can occur for non-symmetric systems. Avoiding Lanczos breakdown is much more challenging, and although numerous research efforts exist, see e.g. [18], [19], most of the proposed ideas are not very popular in the scientific computing community. Much more successful was the search for a workaround avoiding the transposed system matrix. Sonneveld developed the very efficient "conjugate gradient square" algorithm (CGS, [20]). Squaring the BiCG polynomial removes the need for the transposed system matrix, and for linear systems where BiCG converges, CGS is often an attractive, faster alternative. Unfortunately, CGS inherits the non-smooth convergence properties, and the breakdown conditions from BiCG [20]. For enhanced numerical stability, and smoother convergence, Van der Vorst developed the BiCGSTAB method [21]. This algorithm can be seen as a combination of BiCG and GMRES using the restart parameter 1 [22]. BiCGSTAB offers an attractive balance between numerical stability and fast convergence. There are also efforts to combine QMR, CGS and BiCGSTAB to obtain a method sharing multiple enhancements, see TFQMR as a combination of QMR and CGS [23], and QMRBiCGSTAB as a combination of QMR and BiCGSTAB [24]. The relation between these solvers, all based on on a Bi-Lanczos process generating the Krylov space, is visualized in Figure 1.

A much more recent Krylov solver is the "induced dimension reduction" algorithm with a flexible shadow space dimension (IDR(s)) developed by Sonneveld et al. [25]. For

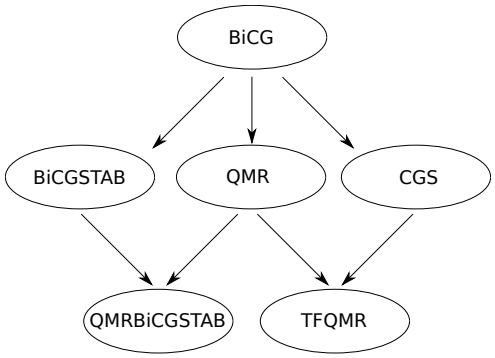

Figure 1: Outlining dependencies between some Krylov methods based on a Bi-Lanczos process.

different shadow space dimensions s, IDR(s) is a robust and efficient short recurrence Krylov subspace method. At the same time it can be seen as generalization of different Krylov solvers, as it can be shown that for the shadow space dimension $s = 1$, IDR(1) becomes very similar to BiCGSTAB [26]. Unfortunately, for shadow space dimensions $s > 1$, the structure of a single IDR(s) iteration is very different from the other Krylov solvers based on short recurrences. Each iteration then consists of a two-way loop nest. This complicates, and practically prevents, the integration of IDR(s) for $s > 1$ into an algorithmic bombardment scheme. Hence, for $s > 1$, IDR(s) can merely be applied as a standalone method. Similar to the previous discussion on the storage cost of the GMRES algorithm, we limit the IDR(s) analysis in this study to the shadow space dimensions $s = 2$, $s = 4$, and $s = 8$ (which was also proposed by Sonneveld et al. [25]). A variant of the IDR(s) that is not included in this study is the Ritz-IDR proposed by Simoncini et al. [27]. Motivated by the possibility to regard the IDR(s) as a Petrov-Galerkin method, the poles of the rational function are chosen as Ritz values [27]. The convergence of Ritz-IDR is competitive, and for small shadow space dimensions s often superior, to the initial algorithm proposed in [25].

III. Test framework

A. Test matrices

We evaluate the efficiency of the Krylov solvers' GPU implementations using a subset of the non-symmetric matrices available in the University of Florida Matrix Collection (UFMC, [7]). More precisely, we include all matrices in the test suite that fulfill the following conditions:

- The matrix is square.
- The matrix is non-symmetric.
- The matrix has more than 1,000 rows.
- The matrix has less than 5,000,000 rows and less than 100,000,000 nonzeros.
- At least one of the Krylov solvers included in the study converges.

This scenario reflects the situation where no information about the linear system is known, except for being structurally or numerically non-symmetric. We consider this a realistic "black-box" scenario where no information about the system characteristics – except the non-symmetry – is provided. We handle the symmetry differently as this information is usually available, and for symmetric, positive definite (spd) systems, the Conjugate Gradient is well-known to be a very efficient alternative [5]. In Figure 2, we visualize the size distribution (left) and the nonzero distribution (right) of the 94 matrices included in the test suite. In the Appendix we give a list of the matrix IDs that allows for identifying the matrices in the UFMC.

B. The libufget library

The UFMC, where we source our example matrices, is normally interfaced by a MATLAB interface or a Java application. Both interfaces allow one to search and download matrices by their name or their ID. Additionally, the Java interface has the ability to search for matrices using meta data information, but once identified each matrix has to be downloaded manually. Having our conditions from the previous subsection in mind, this is not applicable for large matrix sets. In the case of the MATLAB interface we we have an even worse situation because we have to check each matrix for fulfilling the required properties after downloading it. Furthermore, both interfaces cannot be easily used from C or integrated into standalone programs.

We solved this issue by developing a C-library named *libufget* which allows us to access the UFMC directly from a C program. The library downloads the MATLAB file which contains the meta-data from the UFMC and converts it into an SQLite database. Like the MATLAB interface the library provides a by-name and a by-id interface to download matrices. Furthermore, the SQLite database allows us to search for matrices using SQL queries on their meta-data. The search result is obtained as an iterator over all matrices matched by the query. This iterator is used to execute a piece of code for each matrix returned by the query. By translating our conditions from Subsection III-A into the following SQL query: `SELECT * FROM matrices WHERE rows==cols && numerical_symmetry!=1.0 && rows > 1000 && rows < 5000000 && nnz < 100000000`, we perform all benchmarks automatically without any interruption or manual downloading of the test matrices. The *libufget* library enables us to check the influence of different matrix properties like symmetry, positive-definiteness, or bandwidth on the algorithms by only changing the SQL query. This allows for rapid and easy-to-use penetration testing of matrix related algorithms.

C. MAGMA software package

MAGMA [2] is an accelerator-focused linear algebra library developed at the University of Tennessee. It provides

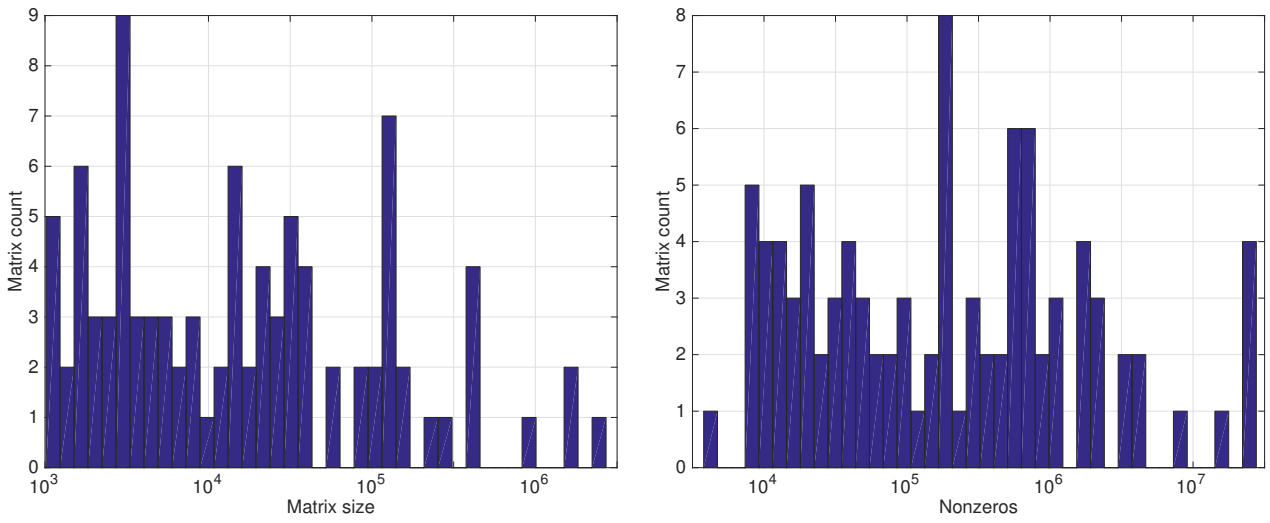

Figure 2: Histograms reflecting the size distribution (left) and the nonzero distribution (right) of the test matrices contained in the test suite.

back-ends for NVIDIA GPUs, Intel's Xeon Phi manycore accelerators (MIC), and any OpenCL-compatible system such as AMD GPUs. Originally focused on dense linear algebra routines, MAGMA now also contains a large variety of solvers, preconditioners, and eigensolvers for sparse linear systems. Comprehensive support for NVIDIA GPUs is provided, some basic routines and functionalities are also available in OpenCL and for the Xeon Phi. The performance of the Krylov solvers available in MAGMA is highly competitive to other GPU-accelerated packages [4]. The subset of Krylov methods we choose from the solvers available in MAGMA are expected to work well for a large range of problems, and have proven to efficiently exploit the compute power of modern GPUs [28], [29].

D. GPU hardware setup

The GPU target architecture is a NVIDIA Tesla K40 GPU (Kepler microarchitecture) with a theoretical peak performance of 1,682 GFlop/s (double precision). The 12 GB of GPU main memory can be accessed at a theoretical bandwidth of 288 GB/s. In a bandwidth analysis using large data-streams, we achieved values around 193 GB/s. The Krylov solvers we use keep all matrix and vector data and most of the scalar values in the GPU memory. Given this background, all vector operations are handled by the accelerator. For completeness, we nevertheless want to mention the host being an Intel Xeon E5 processor (Sandy Bridge). The MAGMA implementation is using CUDA and cuSPARSE in version version 7.5 [30].

E. Solver parameters

The individual linear systems $Ax = b$ used for the experimental solver analysis are all composed of the different test matrices from the University of Florida Matrix Collection, and a right-hand side $b \equiv 1$. All Krylov solvers are started with an initial guess $x \equiv 0$. Convergence in iteration k is defined as the residual norm $\|b - Ax^k\|$ for the iteration vector x^k being at least 10 orders of magnitude smaller than the norm of the right-hand side:

$$\|b - Ax^k\| < 10^{-10}\|b\|.$$

We impose an additional stopping criterion to avoid unbound execution times. Theoretically, any Krylov solver is converged once the Krylov subspace spans the the complete system space, and no early breakdown due to numerical issues has occurred [5]. For a linear system of size n, this is fulfilled once n Krylov basis vectors are generated, i.e., after n sparse matrix vector products. The Krylov methods we consider differ with respect to how many basis vectors are generated in each iteration: BiCGSTAB, CGS, and QMR all have 2 sparse matrix-vector multiplications (`SpMV`s) per iteration; IDR(s) has $2s+1$ `SpMV`s in every outer iteration, respectively. For a fair comparison, we set the upper iteration bound for the distinct solvers with respect to the `SpMV` count and the matrix dimension. To account for numerical effects related to the finite precision of floating point numbers, we allow the methods to execute up to $2n$ `SpMV`s.

IV. Experimental results

In a first experiment we analyze the robustness and efficiency for the distinct solvers. The height of the bars in Figure 3 reflects for how many linear systems the distinct methods achieved convergence. Additionally, we color a part of the bars in yellow. This part corresponds to the matrix count where a certain solver is the overall winner with respect to the runtime metric. Finally, a small grey bar

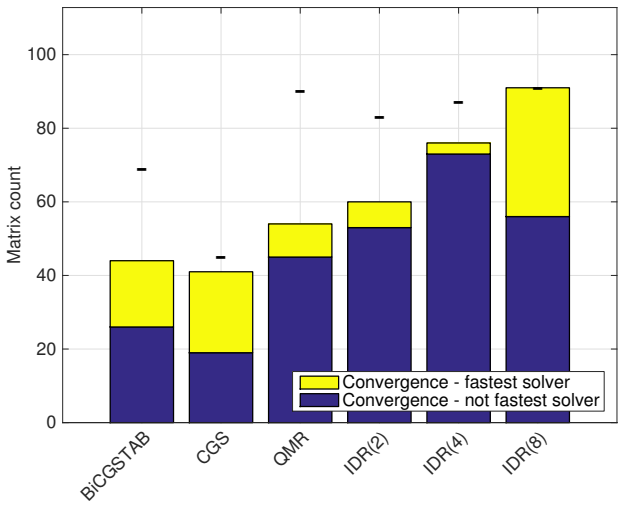

Figure 3: Successful convergence and fastest solver analysis. The grey bars indicate problems that did not converge within the iteration limit, but did not experience an early breakdown either.

indicates the number of problems where convergence was not achieved within the iteration limit, but no breakdown has occurred to this point. The methods may be unable converge with the demanded accuracy, converge for a larger iteration limit, or fail.

We observe that for the general test suite, QMR and IDR(s) are the most robust solvers. Also, the number of problems that can be solved with IDR(s) increases with the shadow space dimension. CGS fails for almost 60% of the test cases, but if it converges, it is very fast.

Next we investigate whether it is possible to identify methods that are "orthogonal" in the sense of problem suitability. Motivation is the strategy presented in [6], where a set of methods is interleaved, resulting in a poly-iterative solver. The key idea of this "algorithmic bombardment" is to successively drop the methods that break down, and benefit from the fast convergence of the most suitable method included in the set. Although the poly-iterative approach has some overhead compared to running a single method, it can be implemented very efficiently for a set of Krylov-based methods having a similar algorithm structure. The central, and often computationally most expensive, building block of all Krylov methods is a sparse matrix vector product needed to generate the Krylov subspace. Aside from that, the algorithms are usually composed of inherently parallel vector updates, global reduction operations that require synchronization, and some scalar computations. The most attractive feature of the poly-iterative approach is to block the sparse matrix vector products generating the distinct subspaces into one blocked sparse matrix vector product that reads the sparse matrix only once, independent of the number of generated Krylov subspaces. Also, merging the reduction phases helps maintain a low number of synchronization points. As a consequence, interleaving a set of algorithmically similar Krylov methods often results in small runtime overhead compared to running only one iterative solver. The relative overhead decreases with an increasing number of non-zero elements per matrix row. Choosing only one Krylov solver, the successful outcome and the time-to-solution performance is unknown. Opposed to that, the poly-iterative approach not only increases the chance of successful completion, but also provides the best convergence rate among the methods included in the set. Algorithmic bombardment is particularly attractive if the methods included are "orthogonal" in the sense of problem suitability.

In Figure 4 we show a "head-to-head" comparison of the distinct solvers we evaluate in this paper. For each solver combination, there is a figure showing how many systems can be solved with both methods (green bar), which method is superior (location of green bar), and how many systems can be solved by one but not the other solver (red and blue bar, respectively). As previously observed, IDR(s) converges for all shadow space dimensions for a larger set of problems than any of the other methods. Comparing the different shadow space dimensions for IDR(s), a smaller shadow space dimension s may offer faster convergence for some of the problems, but robustness increases with the shadow space dimension. More precisely, we did not find a single problem where choosing a larger shadow space dimension destroys convergence of IDR(s). We note again that the upper iteration limit is adapted to the number of `SpMV`s, i.e., a larger shadow space dimension has a lower iteration limit.

When combined with BiCGSTAB, CGS, or QMR, IDR(s) is usually slower for the systems where both methods converge. In terms of robustness, the combination IDR(8) and QMR is the overall winner, converging for all 94 test cases. However, the structural difference between IDR(s) and the other Krylov solvers, makes it hard to combine them efficiently in poly-iteration fashion. Ignoring IDR(s), QMR works best in combination with BiCGSTAB. For the systems where both, BiCGSTAB and CGS converge, CGS is usually faster. BiCGSTAB is more robust than CGS, but not as fast, and therefore a less attractive counterpart to QMR. Although impossible to visualize in this fashion, we want to mention that the combination BiCGSTAB, CGS, and QMR, as proposed in [6], converges for 63 of the 94 test cases (67%). This is a lower success rate than the 96% convergence of IDR(8).

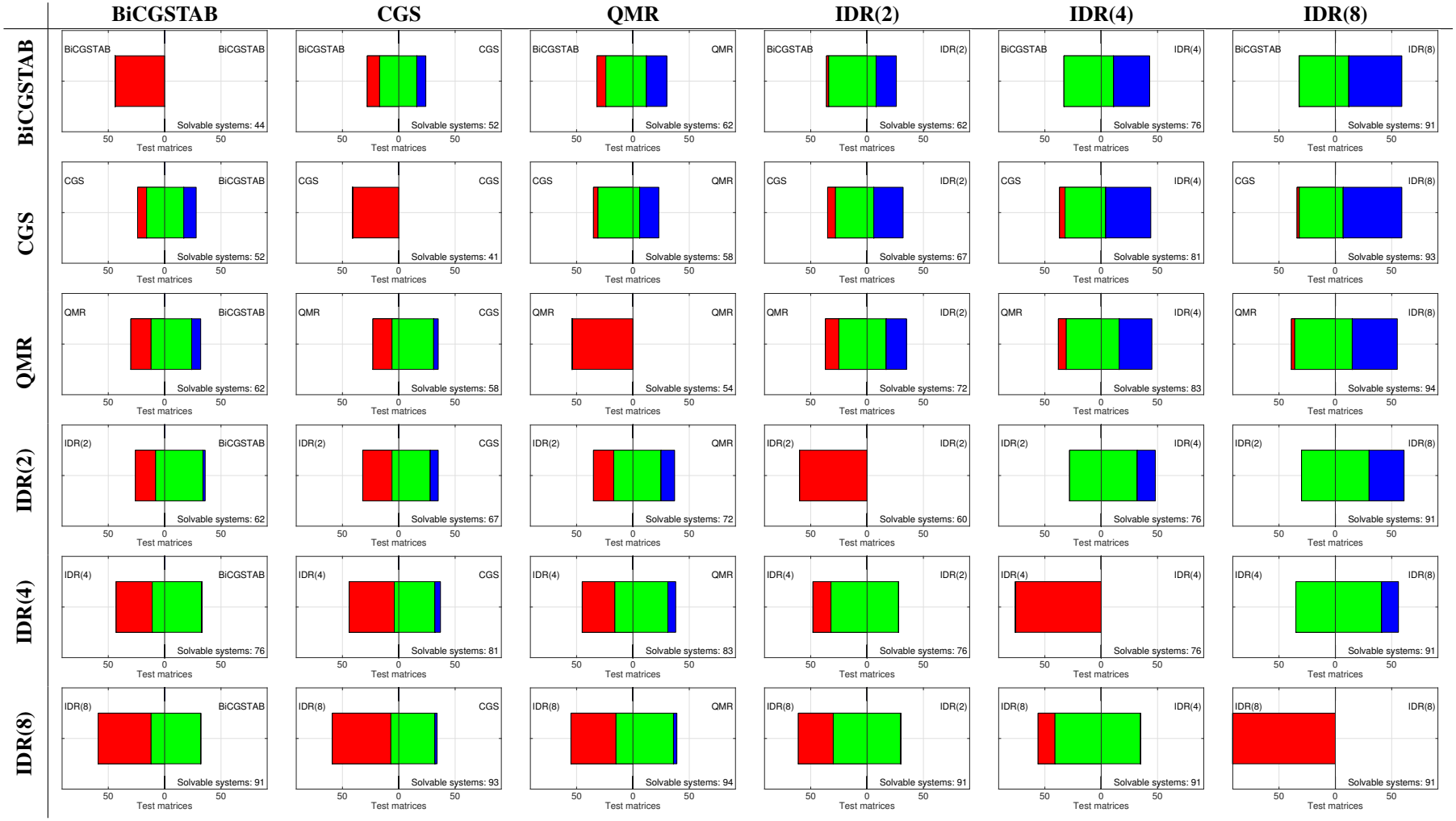

Figure 4: Krylov solver comparison: The green bar reflects the linear systems that can be solved by both methods, the location indicates which method converges faster. The red and blue bars reflect test systems that can be handled by only one of the methods. An interactive visualization of the data can be found at `http://www.icl.utk.edu/~hanzt/solver_ortho/`.

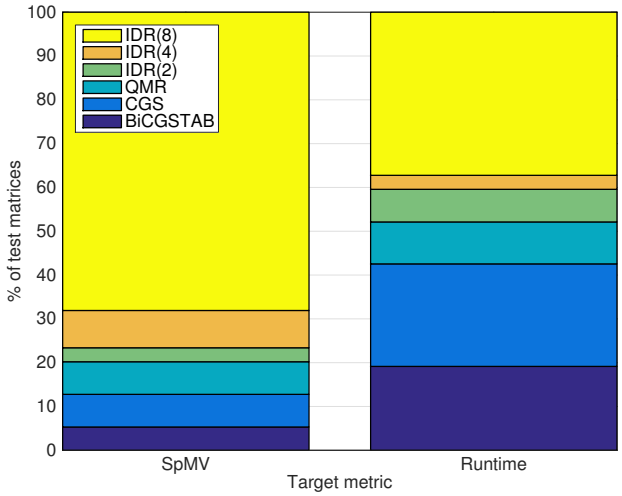

Figure 5: Performance comparison for the different target metrics SpMV count and execution time.

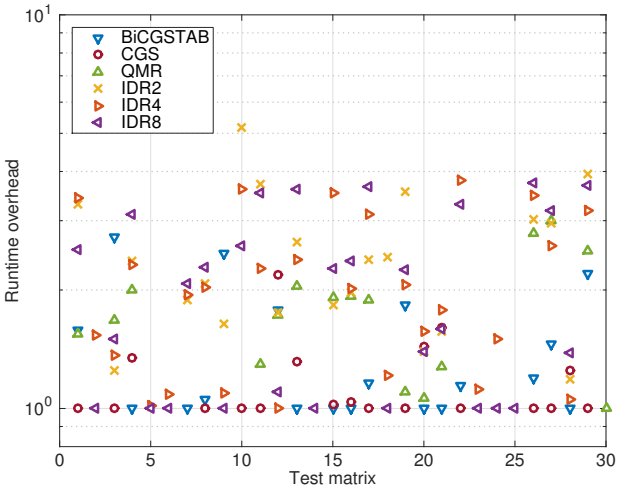

Figure 6: Normalized runtimes: for each matrix, the fastest solver is identified, and all solver execution times are normalized by the runtime of the fastest solver. Test problems are the first 30 matrices listed in the Appendix.

Execution time is the metric of interest when using the plain Krylov solvers. A popular strategy for improved convergence properties is to enhance the Krylov methods with a preconditioner. The efficiency analysis focusing exclusively on execution time is then no longer valid, as each basis vector is then generated for the preconditioned system. Depending on the problem and the preconditioner properties, the preconditioner application can be very expensive compared to the rest of the Krylov solver.

If we assume that an expensive preconditioner provides the same convergence improvement to each of the considered methods, we should focus the analysis on the number of generated basis vectors. The latter corresponds to the number of executed sparse matrix vector products. This motivates the extension of the survey by a second target metric: number of executed SpMVs. Figure 5 compares the solver's superiority with respect to both metrics. Considering the runtime metric, IDR(8) was primarily attractive due to its robustness. Looking at SpMV count, the more expensive orthogonalization in IDR(8) is not reflected, and IDR(8) wins in most cases. Consequently, for an expensive preconditioner providing the same convergence improvement to all methods, IDR(8) would also win most problems in the runtime metric.

Turning back to the basic Krylov solvers, we are interested in the cost we have to pay for the higher robustness of IDR(8). For each matrix, there is a fastest solver, but which one depends on test matrix. We normalize for each matrix the solver runtimes to the runtime of the fastest solver for this particular problem. In Figure 6, we visualize this metric we call "normalized runtime" for a subset containing the first 30 test matrices listed in the Appendix. For each solver, we then compute the average of the normalized runtimes over all matrices, however, only consider converging combinations. If a solver did not converge for a certain test matrix, its average runtime is not affected. This quantity is visualized in Figure 7, taking all 94 test cases into account. For the robust IDR(8), the average runtime is around 1.9 times longer than when choosing the fastest solver for every problem. BiCGSTAB, CGS, and QMR have lower values, but again, this metric does not reflect the fact that those methods fail for a significant portion of the test matrices. The combination of converging for 96% of the problems and an average runtime less than twice slower than the fastest method makes IDR(8) attractive for a black-box scenario. Although not scientifically relevant, we want to mention that it takes IDR(8) about 1 hour and 15 minutes to solve 91 of the considered 94 problems.

V. Summary

In this paper, we evaluate the efficiency of different Krylov solvers based on short recurrences when being implemented on GPUs. For BiCGSTAB, CGS, QMR, and IDR(s) we take the respective implementations from the MAGMA software library and compare their efficiency for different target metrics. The study is based on a large set of test matrices available in the University of Florida Matrix Collection.

The analysis reveals the superiority of IDR(s) in terms of robustness, in particular when using large shadow space dimensions. In terms of time-to-solution performance, BiCGSTAB, CGS, and QMR are often faster for converging cases. Interleaving these solvers in terms of a poly-iterative algorithm results in a fast solver for many problems. In terms of robustness, IDR(8) is still superior, converging for about

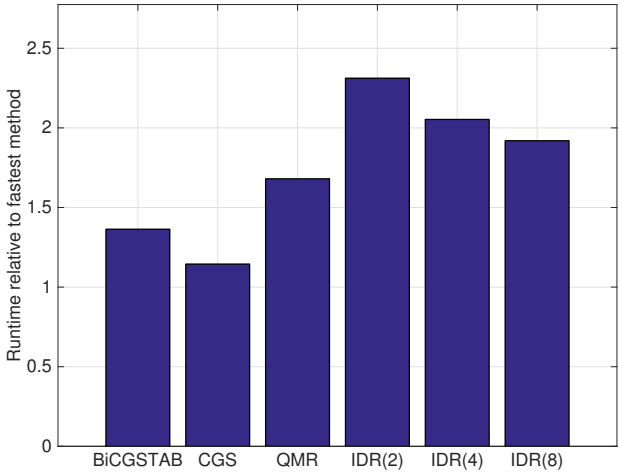

Figure 7: Runtime normalized to the fastest method. The average is computed considering only convergent problems.

96% of the test cases. Compared to the respectively fastest solver, IDR(8) is on average less than twice slower. This good balance between robustness and performance makes IDR(8) an attractive choice in a black-box scenario where no information about the optimal solver is available.

Future work will address the benefits coming from preconditioning, and try to correlate the solvers' superiority with the problems' origins.

APPENDIX

The matrices included in this study can be accessed in the University of Florida Matrix Collection under the following IDs: 227, 235, 237, 245, 246, 287, 288, 289, 290, 291, 370, 371, 377, 396, 467, 468, 540, 542, 543, 814, 815, 820, 823, 825, 826, 828, 829, 833, 834, 835, 864, 895, 897, 898, 909, 910, 911, 912, 913, 914, 915, 925, 927, 928, 930, 931, 932, 934, 982, 984, 1053, 1054, 1106, 1107, 1108, 1109, 1170, 1187, 1188, 1196, 1197, 1319, 1320, 1321, 1322, 1323, 1324, 1416, 1790, 1854, 1858, 1859, 1868, 1869, 1898, 1941, 2233, 2234, 2240, 2241, 2242, 2245, 2274, 2278, 2279, 2562, 2563, 2564, 2565, 2566, 2647, 2648, 2649, 2655.

ACKNOWLEDGMENT

The authors would like to acknowledge support from the U.S. Department of Energy (Award Number DE-SC-0010042), the German Research Foundation (DFG) through the Priority Program 1648 "Software for Exascale Computing" (SPPEXA) under project ESSEX ("Equipping Sparse Solvers for Exascale"), and NVIDIA. The authors would also like to thank Daniel B. Szyld for sharing his knowledge of Krylov methods.

REFERENCES

[1] *cuSPARSE Toolkit v7.0*, v7.0 ed., NVIDIA Corporation, March 2015.

[2] Innovative Computing Lab, "Software distribution of MAGMA version 2.0," http://icl.cs.utk.edu/magma/, 2016.

[3] "PARALUTION," http://www.paralution.com/, 2015.

[4] "ViennaCL," http://viennacl.sourceforge.net/, 2015.

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

[6] R. Barrett, M. Berry, J. Dongarra, V. Eijkhout, and C. Romine, "Algorithmic bombardment for the iterative solution of linear systems: A poly-iterative approach," *Journal of Computational and Applied Mathematics*, vol. 74, no. 1-2, pp. 91 – 109, 1996.

[7] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, pp. 1–25, 2011.

[8] R. W. Freund, G. H. Golub, and N. M. Nachtigal, "Iterative solution of linear systems," *Acta Numerica*, vol. 1, pp. 57–100, 1 1992.

[9] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, "How Fast are Nonsymmetric Matrix Iterations?" *SIAM Journal on Matrix Analysis and Applications*, vol. 13, no. 3, pp. 778–795, 1992.

[10] V. Simoncini and D. B. Szyld, "Recent computational developments in Krylov subspace methods for linear systems," *Numerical Linear Algebra with Applications*, vol. 14, no. 1, pp. 1–59, 2007.

[11] L. Giraud, D. Ruiz, and A. Touhami, "A Comparative Study of Iterative Solvers Exploiting Spectral Information for SPD Systems," *SIAM Journal on Scientific Computing*, vol. 27, no. 5, pp. 1760–1786, 2006.

[12] F. Perini, E. Galligani, and R. D. Reitz, "A study of direct and Krylov iterative sparse solver techniques to approach linear scaling of the integration of chemical kinetics with detailed combustion mechanisms," *Combustion and Flame*, vol. 161, no. 5, pp. 1180 – 1195, 2014.

[13] S. Sundar and B. Bhagavan, "Comparison of Krylov subspace methods with preconditioning techniques for solving boundary value problems," *Computers and Mathematics with Applications*, vol. 38, no. 11–12, pp. 197 – 206, 1999.

[14] D. C. Fong and M. Saunders, "CG versus MINRES: An empirical comparison," Stanford University, Tech. Rep., 2011.

[15] J. Liesen and Z. Strakoš, "On Optimal Short Recurrences for Generating Orthogonal Krylov Subspace Bases," *SIAM Review*, vol. 50, no. 3, pp. 485–503, 2008.

[16] C. Brezinski, M. Redivo-Zaglia, and H. Sadok, "Breakdowns in the implementation of the Lánczos method for solving linear systems," *Computers and Mathematics with Applications*, vol. 33, no. 1–2, pp. 31 – 44, 1997.

[17] R. W. Freund and N. M. Nachtigal, "QMR: a quasi-minimal residual method for non-Hermitian linear systems," *Numerische Mathematik*, vol. 60, no. 1, pp. 315–339.

[18] C. Brezinski, M. R. Zaglia, and H. Sadok, "Avoiding breakdown and near-breakdown in Lanczos type algorithms," *Numerical Algorithms*, vol. 1, no. 2, pp. 261–284.

[19] C. Brezinski and M. Redivo-Zaglia, "Treatment Of Near-Breakdown In The Cgs Algorithm," in *Numer. Algorithms*, 1994, pp. 33–73.

[20] P. Sonneveld, "CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 1, pp. 36–52, 1989.

[21] H. A. van der Vorst, "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 2, pp. 631–644, 1992.

[22] M. H. Gutknecht, "Variants of BICGSTAB for Matrices with Complex Spectrum," *SIAM Journal on Scientific Computing*, vol. 14, no. 5, pp. 1020–1033, 1993.

[23] R. W. Freund, "A Transpose-free Quasi-minimal Residual Algorithm for non-Hermitian Linear Systems," *SIAM J. Sci. Comput.*, vol. 14, no. 2, pp. 470–482, Mar. 1993.

[24] T. F. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, and C. H. Tong, "A Quasi-Minimal Residual Variant of the Bi-CGSTAB Algorithm for Nonsymmetric Systems," *SIAM Journal on Scientific Computing*, vol. 15, no. 2, pp. 338–347, 1994.

[25] P. Sonneveld and M. B. van Gijzen, "IDR(s): A Family of Simple and Fast Algorithms for Solving Large Nonsymmetric Systems of Linear Equations," *SIAM Journal on Scientific Computing*, vol. 31, no. 2, pp. 1035–1062, 2009.

[26] G. L. Sleijpen, P. Sonneveld, and M. B. van Gijzen, "Bi-CGSTAB as an induced dimension reduction method," *Applied Numerical Mathematics*, vol. 60, no. 11, pp. 1100 – 1114, 2010, special Issue: 9th {IMACS} International Symposium on Iterative Methods in Scientific Computing (IISIMSC 2008).

[27] V. Simoncini and D. B. Szyld, "Interpreting IDR as a Petrov-Galerkin method," *SIAM Journal on Scientific Computing*, pp. 1898–1912, 2010.

[28] H. Anzt, W. Sawyer, S. Tomov, P. Luszczek, and J. Dongarra, "Acceleration of GPU-based Krylov Solvers via Data Transfer Reduction," *International Journal of High Performance Computing*, 2015.

[29] H. Anzt, E. Ponce, G. D. Peterson, and J. Dongarra, "GPU-accelerated Co-design of Induced Dimension Reduction: Algorithmic Fusion and Kernel Overlap," in *Proceedings of the 2Nd International Workshop on Hardware-Software Co-Design for High Performance Computing*, ser. Co-HPC '15. New York, NY, USA: ACM, 2015, pp. 5:1–5:8.

[30] *CUDA Toolkit v7.5*, NVIDIA Corporation, September 2015.