U.S. Department of Energy
Office of Science

ASYNCHRONOUS ITERATIVE SOLVERS FOR EXTREME-SCALE COMPUTING

DOE-UTK-DE-SC0016513-1

Jack Dongarra
Stanimire Tomov
Hartwig Anzt

Date Prepared: 02/04/2021
Reporting Period: 09/1/2016–08/31/2020

Prepared by
INNOVATIVE COMPUTING LABORATORY
UNIVERSITY OF TENNESSEE
Knoxville, TN 37996
under contract DE-SC0016513

EXECUTIVE SUMMARY

The Asynchronous Iterative Solvers for Extreme-Scale Computing (AsyncIS) project aims to explore more efficient numerical algorithms by decreasing their overhead. AsyncIS does this by replacing the outer Krylov subspace solver with an asynchronous optimized Schwarz method, thereby removing the global synchronization and bulk synchronous operations typically used in numerical codes.

AsyncIS—a U.S. Department of Energy (DOE)-funded collaboration between Georgia Tech, the University of Tennessee, Knoxville, Temple University, and Sandia National Laboratories—also focuses on the development and optimization of asynchronous preconditioners (i.e., preconditioners that are generated and/or applied in an asynchronous fashion). The novel preconditioning algorithms that provide fine-grained parallelism enable preconditioned Krylov solvers to run efficiently on large-scale distributed systems and manycore accelerators like GPUs.

1. INTRODUCTION

Extreme-scale supercomputers, like upcoming exascale machines, will have massive amounts of parallelism, both at the node level and within nodes. This massive parallelism will pose challenges to the efficient execution of numerical codes that use global synchronization or a bulk synchronous parallel model of computation. This is because any load imbalance or nonuniformity in hardware performance will cause all processing units to idle at a synchronization point, waiting for the slowest unit. Irregular and adaptive DOE simulation codes are particularly at risk, and this problem is exacerbated by non-application irregularities such as OS jitter, different rates of memory error corrections due to variability in chip manufacturing, and CPU speeds unpredictably being throttled depending on local cooling characteristics. Balance can be very difficult to achieve, especially for large numbers of processors and relatively small workloads per processor. A significant bottleneck will be the Krylov subspace iterative methods, which have become a mainstay of DOE's implicit simulation codes. The inner products in the Krylov methods are global synchronization points that generally involve every single processing unit that is participating in the computation.

A key idea of this project is to replace the outer Krylov subspace solver with the asynchronous optimized Schwarz (AOS) method, thereby removing the global synchronization and bulk synchronous requirements. Subdomain solves in the optimized Schwarz method use a preconditioned Krylov subspace method. Thus, the Krylov subspace method is moved to an inner level, acting on subdomains the sizes of which are chosen such that performance non uniformities are small or can be tolerated. Such subdomains can span multiple compute nodes, corresponding to the type of distributed-memory solvers we use today. Within a compute node, however, massive thread-level parallelism for the preconditioner must still be addressed. For this, we studied fine-grained parallel preconditioners where asynchrony can also play a part.

We further developed and analyzed asynchronous preconditioners, i.e., preconditioners that are based on asynchronous iterations. To accelerate convergence, we needed an alternative to preconditioners for many large-scale problems that decouple subdomains, communication across subdomains for elliptic partial differential equations (PDEs), and others. Asynchronous methods can be designed to adaptively perform communication when necessary. For this, we proposed to study a new relaxation method that we call "Distributed Southwell,'" which can be naturally implemented in an asynchronous fashion. Distributed Southwell algorithms could be very effective as multigrid smoothers and as components of physics-based block preconditioners.

Convergence of these new methods were analyzed and established mathematically. The methods were also explored in a software framework for asynchronous iterative methods. Leadership-class DOE computers were targeted, as well as large-scale DOE applications in fluid flow and ice-sheet simulation. To measure success, the AOS method with various types of subdomain solvers were benchmarked against preconditioned Krylov subspace methods—including pipelined (non-blocking) GMRES, recent hierarchical and nested Krylov subspace methods, and also preconditioned inexact Chebyshev methods. We anticipated that the asynchronous methods' competitiveness would depend on the degree of load imbalance and other imbalances in the system; however, we hoped to see a trend toward better performance, for example, of AOS over standard Krylov subspace methods as machine configurations increase in size, even if theoretical load balance is perfect. As a contingency, we note that AOS could also be used as a preconditioner within a Krylov subspace method. In this case, the subdomain solves in the AOS method can be approximate. By shifting the burden of work between GMRES and AOS through tuning the accuracy of the subdomain solves, we can control the number of iterations, and thus the amount of global synchronization.

2. MAJOR ACTIVITIES

Year 1

The role of the University of Tennessee, Knoxville (UTK), in the first year of the AsyncIS project focused on supporting the project partners' efforts to implement parallel versions of Southwell iterations on HPC systems and advance fine-grained subdomain preconditioners. With the project partners from Georgia Tech making quick progress in the development of parallel Southwell techniques, the UTK team primarily worked on the node-level preconditioners providing fine-grained parallelism. In the last months, we developed, implemented, tested, and presented new preconditioning strategies to the scientific community.

Southwell Implementation

In the first months of the project, the team members at Georgia Tech made significant advances in the development, analysis, and implementation of parallel versions of asynchronous Southwell iterations. Due to the obvious progress in this research direction, UTK's research efforts concentrated on the fine-grained subdomain solvers.

Robust ILU

The fine-grained ILU algorithm based on fixed-point iterations succeeds in generating incomplete factorizations if the bilinear fixed-point equations are not too ill conditioned, and the iterations do not violate the limits of the floating point format used. This work package aims to increase the numerical stability of the fixed-point iterations and enhance the accuracy of the generated incomplete factors. Together with our project partners, we advanced the fixed-point iterations that generate the incomplete factors; however, we realize more work is needed to attain the level of robustness we are targeting. The research efforts moved in this direction in the following months. To enhance the quality of incomplete factorization preconditioners generated via fixed point iterations, we developed a new class of parallel threshold ILU. Conversely to existing thresholding techniques, our strategy does not arise from decomposing the global system into subproblems (a strategy that only allows for coarse-grained parallelism), but from combining the fixed-point-based ILU algorithm with a dynamically changing sparsity pattern. This way we can leverage the fine-grained parallelism available in manycore accelerators equipped with shared memory. From the theoretical perspective, we modified the fixed-point ILU algorithm from minimizing the nonlinear residual form to minimizing the ILU residual norm. The latter is expected to be a better metric for the preconditioner quality. Numerical experiments confirm that the new strategy is, in terms of preconditioner quality, competitive with existing threshold-ILU strategies. At the same time, the parallelization potential makes the new algorithm much faster when running on parallel architectures. We presented the new parallel threshold ILU at community events and had a high-profile publication [1, 2].

Hybrid Triangular Solver

The goal of this work package is the development of new types of sparse triangular solves that allow leveraging of the high concurrency levels that are typical of manycore accelerators like GPUs. With this objective, we derived a new strategy for incomplete factorization preconditioning. Instead of generating sparse triangular factors along with level scheduling information, we approximated the inverse of the sparse triangular factors. For the approximate inverse, we used the same nonzero pattern shared by the incomplete factor. This motivated naming the strategy "incomplete sparse approximate inverses (ISAI)" [3]. In the preconditioner application, the exact sparse triangular solves that were parallelized using level scheduling are replaced by multiplication with the sparse approximate inverses. Compared with exact triangular solves, sparse matrix vector multiplications typically provide a much higher level of parallelism. Additionally, for most manycore technologies, highly optimized implementations exist for a sparse matrix vector product as part of vendor libraries like Intel's MKL or NVIDIA's cuSPARSE. Although the ISAI preconditioning strategy solves the incomplete factors only approximately, the significantly faster preconditioner application can accelerate the overall solution process. The accuracy of the ISAI preconditioning can also be enhanced by using a few stationary iterations (Jacobi relaxations). We demonstrated the effectiveness of ISAI preconditioning using a large number of test problems with different origins [1]. For the preconditioner generation, we developed efficient routines that make heavy use of batched operations. We presented the ISAI preconditioning strategy at community events and reported the achieved performance on Intel's Xeon Phi architecture and NVIDIA GPUs in scientific publications [3, 4, 5, 6]. We made the implementation publically available in the MAGMA Sparse open-source software library. This accelerator-focused, node-level software module contains a collection of sparse solvers, preconditioners, and eigensolvers and will become a key component in the sparse linear algebra software ecosystem, as the DOE Exascale Computing Project 1.3.3.11 STMS11-PEEKS aims to develop a generic interface to the Trilinos software infrastructure.

The project team at UTK also supported the project efforts that aim to replace exact sparse triangular solves with block-Jacobi relaxations. Together with colleagues from Georgia Tech, we addressed the challenge of deriving efficient blocking techniques and reported promising strategies in scientific publications [7]. Furthermore, we addressed the challenge of the efficient computation of block-Jacobi matrices on manycore accelerators by deriving techniques that involve batched factorization and inversion. We considered Gauss-Jordan, Gauss-Huard, and LU factorization (the de facto standard) and compared the implementations for the distinct algorithms with respect to performance, numerical stability, and their efficiency when used for block-Jacobi preconditioning [8, 9, 10, 11]. As we did for the ISAI preconditioning, we made the implementation of block Jacobi-based sparse triangular solves publically available in the MAGMA Sparse software module.

Year 2

Synchronous and Asynchronous Schwarz Software

In the second year of the project, we developed a software framework to study the performance of the asynchronous Schwarz method. The major results are provided below.

We used our software to examine the potential of the AOS method on current and future distributed-memory computers. Previous work has already demonstrated the asynchronous method's potential to improve the performance of the synchronous method by removing global synchronization points. In contrast to previous work, however—which used non-blocking, two-sided communication—we studied the use of several one-sided (remote memory access) communication mechanisms that are truly asynchronous (with passive target completion). In addition, previous work used the imbalanced workloads and heterogeneous compute nodes to emphasize the advantages of asynchronous iterative methods. To extend those previous studies, we studied performance with 2-D regular meshes that are evenly distributed on the Cori supercomputer at the National Energy Research Scientific Computing Center (NERSC). Our performance results using the Message Passing Interface (MPI) or Symmetric MEMory (SHMEM) for the asynchronous communication on the Intel Haswell or Knights Landing (KNL) CPUs of the Cori supercomputer demonstrate that when asynchronous communication is well supported (e.g., the communication progresses behind the local computation), the asynchronous method can outperform the synchronous method—even for the balanced problem using homogeneous nodes [12].

We extended the software to solve a general linear system of equations using the classical Schwarz method and to use an iterative method for solving the local subdomain problems. We are studying the relationship of the stopping criteria to the inner and outer iterations and the potential of the asynchronous iteration to overcome the load imbalances among the processes (e.g., differences in the required number of iterations for solving the local problems). We plan to use this framework to utilize both CPUs and GPUs on the node.

Production-Ready Pipelined or Communication-Avoiding Krylov Solvers

For the second year of the project, we also continued our development of the pipelined and communication-avoiding Krylov solvers, which can be used to compare the performance of the asynchronous iterative solvers developed on a large scale by the UTK or Sandia group. The solver will become available through the Trilinos public GitHub repository, so that it will be readily available to other groups. The target applications mentioned in the proposal (e.g., the NaLu low-Mach fluids simulation code and the Albany ice-sheet model) currently rely on the Trilinos linear solvers.

Asynchronous Parallel Threshold ILU Generation

In the first year of the project, the UTK team put a great deal of effort into developing an asynchronous iterative algorithm for generating a threshold ILU preconditioner. We continued these efforts in Year 2 and finally developed a robust parallel threshold ILU (ParILUT) that has good scalability properties on multicore and manycore architectures [4]. Furthermore, the generic design of the algorithm enables easy control of the fill-in, and the user can benefit from updating a previously generated preconditioner to a similar system.

We realized an implementation of the ParILUT algorithm for multicore based on OpenMP and an implementation for manycore that targets NVIDIA GPUs. Both implementations are well documented and ready-to-use components of the MAGMA Sparse open-source software package. Aside from the general ILU, we also developed multiple Cholesky variants for symmetric, positive-definite systems. As the implementation is the first GPU implementation of a threshold ILU preconditioner, the UTK team published a scientific paper detailing the ParILUT algorithm and the performance achieved on modern GPUs [6].

Parallel Triangular Solver
In Year 2, the UTK team continued developing new types of sparse-triangular solvers that allow for leveraging the high concurrency levels typical of manycore accelerators like GPUs. We also devised a new strategy for incomplete factorization preconditioning. As stated for Year 1, instead of generating sparse-triangular factors along with level scheduling information, we approximated the inverse of the sparse-triangular factors—ISAI [13]. We pushed production code implementations of this ISAI strategy for parallel triangular solvers on GPUs and made the code publicly available in the MAGMA Sparse open-source software package.

Furthermore, the UTK team completed the joint research effort with Georgia Tech on blocking techniques in triangular solves based on block-Jacobi [3]. In the context of quickly generating block-Jacobi matrices for iterative triangular solves, we investigated the option of mixed-precision algorithms [14]. The idea here is to store part of the preconditioner in a lower-than-working precision to reduce the cost of accessing the data in main memory, and thereby accelerate the triangular solve.

Year 3

Asynchronous Parallel Threshold ILU Generation
We further advanced the parallel generation of threshold-based ILU preconditioners. In particular, as the quick generation of a threshold separating the *k* smallest entries of a matrix, we focused on designing selection and sorting algorithms for GPUs that are highly parallel, efficient, and allow for reducing the threshold accuracy in favor of a reduced run time. The technology alone became relevant outside of the threshold-ILU research topic [10]. The ParILUT algorithm based on the new selection algorithm has been deployed as production-ready functionality for NVIDIA GPUs in the Ginkgo open-source software package. The technology is now used in scientific simulations based on the MFEM and deal.II finite element packages.

Parallel Triangular Solver
Despite the advantages in developing efficient iterative triangular solves [3], the UTK team continued research on designing fast and accurate triangular solves. Based on the strategy of using block-Jacobi iterations for solving the triangular systems arising in ILU factorization preconditioning, this algorithm was enhanced by reducing the floating-point precision where the numerical properties allow [8]. We deployed the technology for iterative triangular solves in the Ginkgo open-source software package.

Integration
The use of the asynchronous parallel threshold ILU preconditioners in scientific applications is efficient, but not always so right out of the box. In particular, many applications allow for some tuning of the ParILUT parameter—such as sweep count, fill-in, and thresholds—to the specific problem. The UTK team worked closely with a team of researchers from Lawrence Livermore National Laboratory to deploy ParILU and ParILUT technology in an MFEM application.

Asynchronous Subdomain Solves

The UTK team, in cooperation with Pratik Nayak (KIT, Germany), investigated the use of asynchronous subdomain solves in a restricted additive Schwarz (RAS) solver on a multi-GPU supercomputer. The focus was on quantifying the benefits of (1) the use of asynchronous subdomain solves and (2) the use of asynchronous communication strategies (asynchronous, GPU-aware MPI). The findings were published in a scientific paper [9].

Asynchronous Stochastic Gradient Descent (SGD) Solvers

The SGD algorithm is widely employed for training machine learning models such as deep neural networks (DNNs). One of the most popular variants, the mini-batch SGD, not only offers good convergence properties but is also easily parallelizable. However, parallel implementations of mini-batch SGD can be inefficient and can have poor speedups due to the need for synchronization. Therefore, we investigated the use of asynchronous SGD for training DNNs. We developed new asynchronous SGD algorithms and implemented them in the open-source MagmaDNN library [15]. Different from previous asynchronous algorithms, we considered the case where the gradient updates are not particularly sparse, and we compared the parallel efficiency of the asynchronous implementation with that of the traditional synchronous implementation. The results show that the asynchronous SGD not only offers good convergence but is able to outperform the synchronous variant on multicore CPUs and GPUs [16].

Furthermore, we investigated the current challenges in designing deep learning artificial intelligence (AI) and integrating it with traditional high-performance computing (HPC) simulations. We evaluated existing packages for their ability to efficiently run deep learning models and applications on large-scale HPC systems, the identified challenges, and the proposed new asynchronous parallelization and optimization techniques for current large-scale heterogeneous systems and upcoming exascale systems. The approaches and future challenges were illustrated in materials science, imaging, and climate applications, and were presented and published in a position paper at the 2020 Smoky Mountains Computational Sciences and Engineering Conference (SMC 2020) [17].

3. CONCLUSIONS AND RESULTS

Summary of Conclusions

In this project, we advanced the technology in terms of asynchronous algorithms for generating incomplete factorizations and applying sparse-triangular solves like they occur in incomplete factorization preconditioning. In particular, we developed the first algorithm that can generate a threshold-based, incomplete-factorization preconditioner on GPUs. We also advanced the technology for iterative triangular solves. We successfully deployed the new algorithms as production-ready software in the MAGMA Sparse and Ginkgo software ecosystems, and the new technology is already used by finite-element applications. The achievements and the new technology have been presented and advertised at several conferences and detailed in scientific papers.

Significant Results

Software

The developed ParILUT algorithm for the asynchronous generation of an incomplete factorization preconditioner and the ISAI algorithm for the parallel sparse triangular solves (preconditioner application) are available in the MAGMA Sparse and the Ginkgo open-source software packages.

In the context of the ECP effort to efficiently disseminate scientific software, the MAGMA Sparse and Ginkgo software packages were integrated into the xSDK software ecosystem (https://xsdk.info/). This makes the developed methods available on all supercomputing facilities that install the xSDK software package by default.

Knowledge

The ParILUT algorithm is a natural starting point for further research on algorithms that adapt matrix values and sparsity patterns dynamically for a given problem. Furthermore, the need for a fast and accurate sorting algorithm for GPUs has resulted in significant research advances in the field of parallel sorting algorithms.

Integration

Within the US Exascale Computing Project, the Ginkgo and MAGMA Sparse software packages, along with the asynchronous factorization and trisolver algorithms, were integrated into the xSDK software ecosystem. The technology is now available and used within the deal.II and MFEM software packages.

4. OPPORTUNITIES FOR PROFESSIONAL DEVELOPMENT

Dr. Dalal Sukkari worked on the project as a Post-Doctoral Research Associate. Dr. Sukkari studied asynchronous SGD methods and their use in DNN frameworks.

Dr. Stephen Wood worked on the project as a Post-Doctoral Research Associate. Dr. Wood investigated convergence and robustness of asynchronous ILU preconditioner generation via the ParILU algorithm.

5. PROJECT PARTICIPANTS

Below is a table of participants and the total hours they worked on the AsynchIS project.

Participant	Role	Hours
Ahmad Ahmad	Research Scientist	104
Hartwig Anzt	Research Scientist	1287
Jack Dongarra	PI	133
Azzam Haidar	Research Scientist	78
Dalal Sukkari	PostDoc	43
Stanimire Tomov	Research Scientist	156
Stephen Wood	PostDoc	87
Ichitaro Yamazaki	Research Scientist	823

6. PRODUCTS

Papers

1. Yamazaki, I., E. Chow, A. Bouteiller, and J. Dongarra, "Performance of Asynchronous Optimized Schwarz with One-sided Communication," Parallel Computing, vol. 86, pp. 66-81, August 2019. DOI: 10.1016/j.parco.2019.05.004
2. Chow, E., H. Anzt, J. Scott, and J. Dongarra, "Using Jacobi Iterations and Blocking for Solving Sparse Triangular Systems in Incomplete Factorization Preconditioning," Journal of Parallel and Distributed Computing, vol. 119, pp. 219–230, November 2018. DOI: 10.1016/j.jpdc.2018.04.017
3. Anzt, H., J. Dongarra, G. Flegar, N. J. Higham, and E. S. Quintana-Orti, "Adaptive Precision in Block-Jacobi Preconditioning for Iterative Sparse Linear System Solvers," Concurrency and Computation: Practice and Experience, vol. 31, no. 6, pp. e4460, March 2019. DOI: 10.1002/cpe.4460
4. Anzt, H., T. Huckle, J. Bräckle, and J. Dongarra, "Incomplete Sparse Approximate Inverses for Parallel Preconditioning," Parallel Computing, vol. 71, pp. 1–22, January 2018. DOI: 10.1016/j.parco.2017.10.003
5. Anzt, H., J. Dongarra, G. Flegar, and E. S. Quintana-Orti, "Variable-Size Batched LU for Small Matrices and Its Integration into Block-Jacobi Preconditioning," 46th International Conference on Parallel Processing (ICPP), Bristol, United Kingdom, IEEE, August 2017. DOI: 10.1109/ICPP.2017.18
6. Anzt, H., J. Dongarra, G. Flegar, E. S. Quintana-Orti, and A. E. Thomas, "Variable-Size Batched Gauss-Huard for Block-Jacobi Preconditioning," International Conference on Computational Science (ICCS 2017), vol. 108, Zurich, Switzerland, Procedia Computer Science, pp. 1783-1792, June 2017. DOI: 10.1016/j.procs.2017.05.186
7. Anzt, H., J. Dongarra, G. Flegar, and E. S. Quintana-Orti, "Batched Gauss-Jordan Elimination for Block-Jacobi Preconditioner Generation on GPUs," Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores, New York, NY, USA, ACM, pp. 1–10, February 2017. DOI: 10.1145/3026937.3026940
8. Anzt, H., E. Chow, T. Huckle, and J. Dongarra, "Batched Generation of Incomplete Sparse Approximate Inverses on GPUs," Proceedings of the 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, pp. 49–56, November 2016. DOI: 10.1109/ScalA.2016.11
9. Anzt, H., E. Chow, and J. Dongarra, "ParILUT - A New Parallel Threshold ILU," SIAM Journal on Scientific Computing, vol. 40, issue 4: SIAM, pp. C503–C519, July 2018. DOI: 10.1137/16M1079506
10. Chow, E., H. Anzt, J. Scott, and J. Dongarra, "Using Jacobi Iterations and Blocking for Solving Sparse Triangular Systems in Incomplete Factorization Preconditioning," Journal of Parallel and Distributed Computing, vol. 119, pp. 219–230, November 2018. DOI: 10.1016/j.jpdc.2018.04.017
11. Anzt, H., T. Huckle, J. Bräckle, and J. Dongarra, "Incomplete Sparse Approximate Inverses for Parallel Preconditioning," Parallel Computing, vol. 71, pp. 1–22, January 2018. DOI: 10.1016/j.parco.2017.10.003

Presentations and Invited Talks

1. "ParILUT - A Parallel Threshold ILU for multicore and GPUs, SIAM Conference on Parallel Processing for Scientific Computing (PP20)," Seattle, Washington, U.S., 2020
2. "Algorithm Design in the Advent of Exascale Computing," 4th International Symposium on Research and Education of Computational Science (RECS), Tokyo, October 2019.
3. "ParILUT - A Parallel Threshold ILU for GPUs," 33rd IEEE International Parallel and Distributed Computing Symposium, Rio de Janeiro, May 2019
4. "Approximate and Exact Selection on GPUs," 9th International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), Rio de Janeiro, May 2019
5. "ParILUT - A new Parallel Threshold ILU," 9th Joint Laboratory for Extreme Scale Computing (JLESC) workshop, Knoxville, April 2019
6. "Batched Factorization and Inversion Routines for Block-Jacobi Preconditioning on GPUs," Workshop on Batched, Reproducible, and Reduced Precision BLAS 2017, Atlanta.
7. "Preconditioning on Parallel and Hybrid Architectures, SIAM Conference on Computational Science & Engineering," SIAM CSE 2017, Atlanta.
8. "Batched Routines in Preconditioning – The Future of Incomplete Factorization Preconditioners," Workshop on Batched, Reproducible, and Reduced Precision BLAS 2016, Knoxville.
9. "ParILUT – A New Parallel Threshold ILU," SIAM Conference on Parallel Processing, SIAM PP 2016, Paris.
10. "Asynchronous SGD for DNN Training on Shared-Memory Parallel Architectures," IPDPSW 2020.
11. "Integrating Deep Learning in Domain Sciences at Exascale," 2020 Smoky Mountains Computational Sciences and Engineering Conference (SMC 2020), August 2020.
12. "How to Build Your Own Deep Neural Network," PEARC20, July 2020.
13. "Integrating Deep Learning in Domain Sciences at Exascale (MagmaDNN)," DOD HPCMP virtual seminar, December 8, 2020.

Software Releases

- Gingko release 1.2.0 incorporates the ParILUT preconditioner for multicore and GPUs
  https://github.com/ginkgo-project/ginkgo/releases/tag/v1.2.0
- MagmaDNN v1.2 release
  https://bitbucket.org/icl/magmadnn/get/release-magmadnn-v1.2.tar.gz

7. ACKNOWLEDGMENT AND DISCLAIMERS

This material is based upon work supported by the U.S. Department of Energy, Office of Science under Award Number DE-SC0016513.

8. WORKS CITED

1. Anzt, H., T. Ribizel, G. Flegar, E. Chow, and J. Dongarra, “ParILUT – A Parallel Threshold ILU for GPUs,” IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rio de Janeiro, Brazil, IEEE, May 2019. DOI: 10.1109/IPDPS.2019.00033
2. “Approximate and Exact Selection on GPUs,” presentation at the 9th International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), Rio de Janeiro, May 2019
3. Chow, E., H. Anzt, J. Scott, and J. Dongarra, “Using Jacobi Iterations and Blocking for Solving Sparse Triangular Systems in Incomplete Factorization Preconditioning,” Journal of Parallel and Distributed Computing, vol. 119, pp. 219–230, November 2018. DOI: 10.1016/j.jpdc.2018.04.017
4. Anzt, H., E. Chow, and J. Dongarra, “ParILUT - A New Parallel Threshold ILU,” SIAM Journal on Scientific Computing, vol. 40, issue 4: SIAM, pp. C503–C519, July 2018. DOI: 10.1137/16M1079506
5. “Algorithm Design in the Advent of Exascale Computing,” presentation at the 4th International Symposium on Research and Education of Computational Science (RECS), Tokyo, October 2019.
6. “ParILUT - A Parallel Threshold ILU for GPUs,” presentation at the 33rd IEEE International Parallel and Distributed Computing Symposium, Rio de Janeiro, May 2019
7. Anzt, H., J. Dongarra, G. Flegar, and E. S. Quintana-Orti, “Variable-Size Batched LU for Small Matrices and Its Integration into Block-Jacobi Preconditioning,” 46th International Conference on Parallel Processing (ICPP), Bristol, United Kingdom, IEEE, August 2017. DOI: 10.1109/ICPP.2017.18
8. Goebel, F., H. Anzt, T. Cojean, G. Flegar, and E. S. Quintana-Orti, “Multiprecision Block-Jacobi for Iterative Triangular Solves,” European Conference on Parallel Processing (Euro-Par 2020): Springer, August 2020. DOI: 10.1007/978-3-030-57675-2_34
9. Nayak, P., T. Cojean, and H. Anzt, “Evaluating Asynchronous Schwarz Solvers on GPUs,” International Journal of High Performance Computing Applications, August 2020. DOI: 10.1177/1094342020946814
10. Ribizel, T., and H. Anzt, “Parallel Selection on GPUs,” Parallel Computing, vol. 91, March 2020, 2019. DOI: 10.1016/j.parco.2019.102588
11. “ParILUT - A Parallel Threshold ILU for Multicore and GPUs,” presentation at the SIAM Conference on Parallel Processing for Scientific Computing (PP20), Seattle, Washington, 2020.
12. Yamazaki, I., E. Chow, A. Bouteiller, and J. Dongarra, “Performance of Asynchronous Optimized Schwarz with One-sided Communication,” Parallel Computing, vol. 86, pp. 66-81, August 2019. DOI: 10.1016/j.parco.2019.05.004
13. Anzt, H., T. Huckle, J. Bräckle, and J. Dongarra, “Incomplete Sparse Approximate Inverses for Parallel Preconditioning,” Parallel Computing, vol. 71, pp. 1–22, January 2018. DOI: 10.1016/j.parco.2017.10.003
14. Anzt, H., J. Dongarra, G. Flegar, N. J. Higham, and E. S. Quintana-Orti, “Adaptive Precision in Block-Jacobi Preconditioning for Iterative Sparse Linear System Solvers,” Concurrency and Computation: Practice and Experience, vol. 31, no. 6, pp. e4460, March

2019. DOI: 10.1002/cpe.4460

15. Nichols, D., N-S. Tomov, F. Betancourt, S. Tomov, K. Wong, and J. Dongarra, "MagmaDNN: Towards High-Performance Data Analytics and Machine Learning for Data-Driven Scientific Computing," ISC High Performance, Frankfurt, Germany, Springer International Publishing, June 2019. DOI: 10.1007/978-3-030-34356-9_37
16. Lopez, F., E. Chow, S. Tomov, and J. Dongarra, "Asynchronous SGD for DNN Training on Shared-Memory Parallel Architectures," Workshop on Scalable Deep Learning over Parallel And Distributed Infrastructures (ScaDL 2020), May 2020.
17. Archibald, R., E. Chow, E. D'Azevedo, J. Dongarra, M. Eisenbach, R. Febbo, F. Lopez, D. Nichols, S. Tomov, K. Wong, et al., "Integrating Deep Learning in Domain Sciences at Exascale," 2020 Smoky Mountains Computational Sciences and Engineering Conference (SMC 2020), August 2020.