Efficient Large-scale Sparse Eigenvalue Computations on Heterogeneous Hardware

Moritz Kreutzer
Erlangen Regional Computing Center
Friedrich-Alexander University of Erlangen-Nuremberg
Martensstraße 1
91058 Erlangen, Germany
moritz.kreutzer@fau.de

Andreas Pieper
Institute of Physics
Ernst Moritz Arndt University of Greifswald
Felix-Hausdorff-Straße 6
17487 Greifswald, Germany
pieper@physik.uni-greifswald.de

Andreas Alvermann
Institute of Physics
Ernst Moritz Arndt University of Greifswald
Felix-Hausdorff-Straße 6
17487 Greifswald, Germany
alvermann@physik.uni-greifswald.de

Holger Fehske
Institute of Physics
Ernst Moritz Arndt University of Greifswald
Felix-Hausdorff-Straße 6
17487 Greifswald, Germany
fehske@physik.uni-greifswald.de

Georg Hager
Erlangen Regional Computing Center
Friedrich-Alexander University of Erlangen-Nuremberg
Martensstraße 1
91058 Erlangen, Germany
georg.hager@fau.de

Gerhard Wellein
Department of Computer Science
Friedrich-Alexander University of Erlangen-Nuremberg
Martensstraße 1
91058 Erlangen, Germany
gerhard.wellein@fau.de

Alan R. Bishop
Theory, Simulation, and Computation Directorate
Los Alamos National Laboratory
P.O. Box 1663
Los Alamos, NM 87545, USA
arb@lanl.gov

Computing the eigenvalue density and inner eigenvalues of large sparse matrices are important tasks in quantum physics and chemistry. In these communities numerical schemes based on the evaluation of Chebyshev polynomials are used to solve the corresponding sparse eigenvalue problem: The Kernel Polynomial Method (KPM, [16]) for the former and Chebyshev Filter Diagonalization (ChebFD, [13, 3]) for the latter. Conceptually, both algorithms have a lot in common and are equally well suited for high-performance large scale computations. In their basic formulation, the most time consuming operation is sparse matrix-vector multiplication (SpMV), accompanied by several vector-vector operations. In comparison to KPM, the ChebFD scheme requires one additional vector-vector operation and a block vector orthogonalization step (comprising dense matrix-matrix operations of tall and skinny dense matrices) of the search space after a number of Chebyshev steps.

Firstly, we identify the main memory bandwidth as the relevant architectural bottleneck for the algorithms. Based on this knowledge, we then reformulate the algorithms using kernel/loop fusion and vector blocking. Kernel fusion is an ever-relevant optimization technique to avoid unnecessary data transfers. Recent research in the context of GPU programming can be found in [14, 15, 12]. Park et al. have employed kernel fusion in recent work on the High-performance Conjugate Gradient benchmark [10] and Nelson et al. are developing a domain-specific compiler for automatic kernel fusion of BLAS calls [9]. Vector blocking entails the combination of multiple (dense) vectors into a vector block. Fundamental research on performance implications of this optimization technique in the context of SpMV has been conducted by Gropp et al. [4] and recently attained renewed attention ([7, 1, 2]). The computational intensity of the algorithm increases with the number of vectors put in the block. Applying both optimizations, the asymptotic (with respect to the number of vectors in the block) computational intensity of the KPM algorithm can be improved by a factor of ten. The resulting compute kernel, which we call "augmented sparse matrix-multiple vector multiplication", shows the expected increase of computational intensity and allows decoupling from main memory bandwidth while shifting the performance bottleneck towards on-chip resources.

As the optimized algorithm comprises custom kernels we can no longer use existing libraries but need to implement it manually. We present an implementation which is part of

our software library GHOST ("General, Hybrid, and Optimized Sparse Toolkit", [6]). GHOST is designed with highly efficient execution on heterogeneous compute architectures in mind. Unlike most of the existing heterogeneous software ([11, 8]) it features data-parallel execution of sparse linear algebra computations across different devices. It uses the SELL-C-σ sparse matrix storage format as proposed in [5] which shows good performance on all relevant compute architectures. Further capabilities like automatic code generation and manually vectorized and fused kernels lead to very high efficiency. Our software development process is guided by suitable Roofline-like [17] performance models to validate the efficiency of our implementations.

We present performance data for the application scenario of novel topological materials. We illustrate how the algorithmic optimizations increase the single-device and heterogeneous CPU+GPU performance on a single node. The CPU/GPU-only performance can be enhanced by 4.8/2.3x due to algorithmic optimizations and an efficient implementation. The parallel efficiency of single-node heterogeneous runs (i.e., one Sandy Bridge host equipped with an Nvidia Tesla K20X GPU) is close to 90%. Heterogeneous execution especially pays off for the fully optimized compute kernel, where we achieve a 40% performance gain compared to GPU-only execution. For the orthogonalization step of ChebFD we present performance data of our custom dense matrix-matrix multiplication (for tall and skinny dense matrices) in comparison to the BLAS libraries Intel MKL and ATLAS. Our implementation exhibits substantial performance gains for small block sizes. Finally, we demonstrate the scalability of our KPM code on up to 4096 heterogeneous nodes of Piz Daint, Europe's largest TOP500 machine with over 5000 nodes (each equipped with an Intel Sandy Bridge CPU and an Nvidia Kepler GPU). For an originally sparse linear algebra algorithm, we achieve over 0.5 Pflop/s of performance which corresponds to 11% of LINPACK efficiency. In our largest run of the KPM, the underlying sparse system matrix holds more than a hundred billion complex double precision values. For ChebFD, we performed experiments to compute the 100 innermost eigenpairs of a matrix with 1 billion rows on up to 512 Haswell nodes of SuperMUC, achieving a sustained performance of 85 Gflop/s per node for our sparse matrix eigenvalue solver.

1. ADDITIONAL AUTHORS

2. REFERENCES

[1] H. M. Aktulga, A. Buluç, S. Williams, and C. Yang. Optimizing sparse matrix-multiple vectors multiplication for nuclear configuration interaction calculations. In *Proceedings of the 2014 IEEE International Parallel and Distributed Processing Symposium, May 2014*. IEEE Computer Society, 2014.

[2] H. Anzt, S. Tomov, and J. Dongarra. Accelerating the LOBPCG method on GPUs using a blocked sparse matrix vector product. *University of Tennessee Innovative Computing Laboratory Technical Report UT-CS-14-731*, October 2014.

[3] E. Di Napoli, E. Polizzi, and Y. Saad. Efficient estimation of eigenvalue counts in an interval. Preprint avilable: `http://arxiv.org/abs/1308.4275`, 2014.

[4] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. Towards realistic performance bounds for implicit CFD codes. In *Proceedings of Parallel CFD'99*, pages 233–240. Elsevier, 1999.

[5] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. R. Bishop. A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. *SIAM J. Sci. Comput.*, 36(5):C401–C423, 2014.

[6] M. Kreutzer, J. Thies, M. Röhrig-Zöllner, A. Pieper, F. Shahzad, M. Galgon, A. Basermann, H. Fehske, G. Hager, and G. Wellein. GHOST: Building blocks for high performance sparse linear algebra on heterogeneous systems. Preprint avilable: `http://arxiv.org/abs/1507.08101`, 2015.

[7] X. Liu, E. Chow, K. Vaidyanathan, and M. Smelyanskiy. Improving the performance of dynamical simulations via multiple right-hand sides. In *Proceedings of the 2012 IEEE International Parallel and Distributed Processing Symposium, May 2012*, pages 36–47. IEEE Computer Society, 2012.

[8] MAGMA: Matrix algebra on GPU and multicore architectures. `http://icl.cs.utk.edu/magma/`. Accessed: June 2015.

[9] T. Nelson, G. Belter, J. G. Siek, E. Jessup, and B. Norris. Reliable generation of high-performance matrix algebra. *ACM Transactions on Mathematical Software*, 41(3), 2015.

[10] J. Park, M. Smelyanskiy, K. Vaidyanathan, A. Heinecke, D. D. Kalamkar, X. Liu, M. M. A. Patwary, Y. Lu, and P. Dubey. Efficient shared-memory implementation of high-performance conjugate gradient benchmark and its application to unstructured matrices. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 945–955, Piscataway, NJ, USA, 2014. IEEE Press.

[11] K. Rupp, F. Rudolf, and J. Weinbub. ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. In *Intl. Workshop on GPUs and Scientific Applications*, pages 51–56, 2010.

[12] K. Rupp, J. Weinbub, A. Jüngel, and T. Grasser. Pipelined iterative solvers with kernel fusion for graphics processing units. *CoRR*, abs/1410.4054, 2014.

[13] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. SIAM, revised edition, 2011.

[14] S. Tabik, G. Ortega, and E. Garzón. Performance evaluation of kernel fusion BLAS routines on the GPU: iterative solvers as case study. *The Journal of Supercomputing*, 70(2):577–587, 2014.

[15] M. Wahib and N. Maruyama. Scalable kernel fusion for memory-bound GPU applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 191–202, Piscataway, NJ, USA, 2014. IEEE Press.

[16] A. Weiße, G. Wellein, A. Alvermann, and H. Fehske. The kernel polynomial method. *Rev. Mod. Phys.*, 78:275–306, Mar 2006.

[17] S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, Apr. 2009.