Performance of Block Jacobi-Davidson Eigensolvers

Melven Röhrig-Zöllner[†] Jonas Thies[†] Moritz Kreutzer[‡] Andreas Alvermann[§] Andreas Pieper[§] Achim Basermann[†]
Georg Hager[‡] Gerhard Wellein[‡] Holger Fehske[§]

***Abstract*—Jacobi-Davidson methods can efficiently compute a few eigenpairs of a large sparse matrix. Block variants of Jacobi-Davidson are known to be more robust than the standard algorithm, but they are usually avoided as the total number of floating point operations increases. We present the implementation of a block Jacobi-Davidson solver and show by detailed performance engineering and numerical experiments that the increase in operations is typically more than compensated by performance gains on modern architectures, giving a method that is both more efficient and robust than its single vector counterpart.**

I. Introduction

We consider the problem of finding a small number of extremal eigenvalues and corresponding eigenvectors of a large, sparse matrix $A \in \mathbb{C}^{n\times n}$,

$$Av_i = \lambda_i v_i, \qquad i = 1, \ldots, l, \quad l \ll n. \tag{1}$$

Such eigenproblems arise in many scientific and engineering applications; here we present examples from quantum physics.

We investigate a block Jacobi-Davidson method that performs matrix-vector products and vector-vector operations with several vectors at once. In [1] Stathopoulos and McCombs look into Jacobi-Davidson and Generalized Davidson methods for symmetric (respectively Hermitian) eigenvalue problems and also address block methods briefly. Their results suggest that block methods do not "pay off" in the sense that the performance gains do not justify the overall increase in the number of operations. We seek to demonstrate the opposite by extending the performance model for the sparse matrix-vector product given in [2] to the case of multiple vectors.

II. Contents

A. Numerical method

First, we introduce a block variant of the Jacobi-Davidson method, show the correction equation that has to be solved in every iteration, and relate the method to the standard (single vector) algorithm. To motivate the research, we present the results of some numerical experiments, which are based on a set of eigenvalue problems from a wide range of applications: as expected, the block method leads to an increase in the total number of operations in almost all cases. However, if more than 10–20 eigenvalues are sought, the relative overhead remains small enough, such that the performance benefits of blocking can prevail. This is shown in Figure 1.

This work was supported by the German Research Foundation (DFG) through the Priority Program 1648 "Software for Exascale Computing" (SPPEXA) under project ESSEX.

[†]German Aerospace Center (DLR), Simulation and Software Technology

[‡]Erlangen Regional Computing Center, Friedrich-Alexander-Universität Erlangen-Nürnberg

[§]Institut für Physik, Ernst-Moritz-Arndt-Universität Greifswald, Germany

B. Blocked linear solvers

To obtain optimal blocking of the required operations in the inner linear solver, one needs to group together similar operations appropriately. Figure 2 illustrates the behavior of our blocked GMRES implementation for a general setting where one needs to solve several independent linear systems concurrently.

C. Performance engineering of the key operations

As demonstrated in Figures 3 and 4, block vector operations and in particular sparse matrix-multiple-vector multiplications (spMMVMs) can be much faster on current computer hardware than corresponding single vector operations. More insight into the performance of the required operations is gained from performance models: all operations required here are memory bound for large problems and blocking improves the code balance (memory transfers divided by floating point operations). Thus, from the roofline model [**?**] one can predict the potential speedup of the spMMVM compared to n_b single sparse matrix-vector multiplications

$$S^* = n_b \cdot \frac{6n_{nzr} + 8}{6n_{nzr} + 8n_b}. \tag{2}$$

Here, double-precision calculations are assumed and n_{nzr} denotes the average number of non-zero matrix entries per row. The actual speedup is slightly smaller because the model does not account for additional memory transfers caused by non-consecutive memory accesses. E.g. for the matrix $\mathsf{Spin}_{\mathsf{SZ}}[26]$ and a block size of 4 the model predicts a potential speedup of 3.1 while our experiments indicate a speedup of 2.6 as shown in Figure 3.

In addition, based on the performance model we assure that our implementation achieves (at least almost) the highest possible node-level performance. As illustrated in Figure 4, our implementation of the required block operations is much faster than common sparse numerical algebra libraries that use column-wise storage for blocks of vectors such as the Trilinos [3] package Epetra. Here, we use the SELL-C-σ sparse matrix format presented in [2] with a row-wise storage for blocks of vectors. The latter also improves the spMMVM performance of the CRS format (central plot of Figure 4).

D. Applications from quantum physics

For our performance tests we consider typical eigenvalue problems from quantum physics: the matrices represent the Hamilton operator for a Heisenberg spin chain model (Figure 5) and their dimensions grow exponentially with the number of spins; in Figure 6 we can see the sparsity pattern of one of the test matrices.

E. Block performance in strong scaling experiments

Figures 7 and 8 present timing measurements of our complete algorithm on a computing cluster for an eigenvalue problem from quantum physics. Our implementation employs OpenMP for the parallelization on the node-level and MPI for the inter-node communication. In these tests the overall performance increases by a factor of about 1.3 as shown in Figure 7.

F. Software

The software developed in the ESSEX project is organized in several packages, which will eventually be published as open source libraries and can be made available for early testing on request. The PHIST package implements iterative solvers for eigenvalue problems and linear systems based on an abstract interface layer for the core linear algebra operations. This layer is implemented for a wide range of hardware presently available (multi-core CPUs, GPUs, Intel MIC architecture) in the GHOST library. The application layer (quantum physics) is implemented in a separate package which provides scalable test cases and actual applications of scientific interest.

III. Results

The key operation in the block method consists of a sparse matrix-vector product followed by an orthogonal projection. By performance engineering and benchmarking we demonstrate that applying this operation to blocks of vectors, as in our proposed algorithm, has significant performance advantages over the single vector case. An important implementation detail is the row-wise storage of blocks of vectors. This design choice, which is hardly ever found in implementations of block algorithms, is the key to achieving the speedup we have shown for the sparse matrix-vector products.

Our numerical results indicate that the method works well for a wide range of matrices, both symmetric and non-symmetric. The performance results show that the hybrid parallel approach we take (MPI+OpenMP) gives good scalability on a modern cluster, and that the block variant outperforms its single-vector counterpart even for fairly large problems on up to 1280 cores.

References

[1] A. Stathopoulos and J. R. McCombs, "Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues," *SIAM Journal on Scientific Computing*, vol. 29, no. 5, pp. 2162–2188, Jan 2007.

[2] M. Kreutzer, G. Hager, G. Wellein, G. 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." *Accepted for publication in the SIAM Journal on Scientfic Computing (SISC)*, Mar. 2014, preprint: arXiv:1307.6209.

[3] S. W. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-134, 2008.

[4] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the Trilinos project," *ACM Trans. Math. Softw.*, vol. 31, no. 3, pp. 397–423, 2005.