An overview of dense eigenvalue solvers for distributed memory systems

Davor Davidović*
* Centre for Informatics and Computing, Ruđer Bošković Institute, Zagreb, Croatia, davor.davidovic@irb.hr
† department, organization, city, country, email address

***Abstract*—Solving large-scale eigenvalue problems is a central problem in many research areas, such as electronic structure calculations, macromolecular simulations, solid states, theoretical physics, and combinatorial optimization. The computation of the required eigenvalues and the corresponding eigenvectors of the large matrices is a challenging task that requires considerable computational time. Therefore, the computation of such problems is usually performed on large computational resources consisting of a large number of computational nodes interconnected by fast network and often equipped with accelerators, such as graphic processing units. Nowadays, when the whole world is vying for the first exascale supercomputer and the computational appetite of researchers is greater than ever, the need for scalable and powerful eigenvalue solvers capable of utilising such large machines with distributed memory is crucial for further breakthroughs in research. This paper reviews existing numerical linear algebra packages and libraries that implement solvers for dense eigenvalue problems and are tailored to distributed-memory systems. The survey analysis has shown that there are numerous eigenvalue solvers for distributed memory systems. However, not many of them are able to exploit the full potential of modern, heterogeneous, GPU-based machines with complex memory hierarchies.**

***Keywords*—eigenvalue solvers, high-performance computing, distributed-memory, large-scale systems**

I. Introduction

Computing the eigenvalues and eigenvectors of a given system is a fundamental computational problem in many research fields and often the main computational bottleneck in many scientific codes. The problem becomes particularly challenging in the fields of structural dynamics, quantum chemistry, and control theory when eigenvalues and eigenvectors of extremely large matrices are sought-after. To tackle such problems, numerous computational libraries have been developed, tailored for specific classes of eigenvalue problems. The most significant ones, globally accepted as a standard in both academia and industry, are the LAPACK [1], developed for the shared-memory systems, and the ScaLAPACK [2], a version of the LAPACK for the distributed-memory systems. These two libraries are widely used as basic building blocks for other computational libraries, in various scientific codes but also in computational software platforms like R, Matlab, Octave, and SciLab.

Nowadays, we witness a rapid development of never larger computational systems that consist of a huge number of computational nodes. It is not uncommon that each computational node has two or more multi-core processors accompanied by graphical processing units (GPUs) and a complex memory hierarchy. These new architectures pose a challenging task to the existing eigenvalue solver packages in terms of portability, scalability, and efficiency of the code. It is of the most significant importance that the development of the new computing architectures is followed by the development of the novel eigenvalue solvers that can exploit the full computational potential of such systems.

This research aims to give a brief overview of the existing high-performance eigenvalue solver solutions and how efficient are they in solving large eigenvalue problems on modern, distributed and heterogeneous computing systems.

This paper is organized as follows. A brief introduction to the eigenvalue problems is given in Section II. Section III describes the available computational libraries implementing eigensolvers for shared-memory and distributed-memory parallel systems. An overview of the performance and comparison of distributed eigenvalue solvers is presented in Section IV. The paper is concluded in Section V with the discussion on the state-of-the-art eigensolvers for distributed-memory systems.

II. Eigenvalue problem formulation

The standard eigenvalue problem is defined, in matrix representation, as:

$$AX = X\Lambda, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is a square matrix, $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \in \mathbb{C}^{n\times n}$ is a diagonal matrix with the sought-after eigenvalues, and the columns of $X \in \mathbb{R}^{n\times n}$ contains the associated eigenvectors [3], with n being the matrix dimension. The eigenvalue λ_i and its corresponding eigenvector x_i form an eigenpair (λ_i, x_i). If matrix A is complex, then the resulting eigenvectors are also complex. In the case A is symmetric or Hermitian (A is complex and is equal to its conjugate-transpose), then the eigenvalues are real numbers ($\lambda_i \in \mathbb{R}^n$).

However, in many real-world cases, one has to compute the eigenvalues (Λ) and the corresponding eigenvectors (X) of the systems:

$$AX = BX\Lambda, \tag{2}$$

where both matrices A and B are squared. This type of eigenvalue problem is called a generalized eigenproblem. Note that the standard eigenproblem is a special case of the generalized eigenproblem in which matrix B is the identity matrix.

To solve an eigenvalue problem on a computer, one must choose an appropriate numerical method and computational approach that will ensure both the performance and accuracy of the solution obtained. Which method or approach to choose depends mainly on the properties of the given eigenvalue problem (i.e., the properties of the matrices A and B in (1) and (2)) and these properties are: 1) the type of matrix, which can be symmetric/unsymmetric, Hermitian/non-Hermitian, general or unitary, 2) the structure of the matrix - dense, band, sparse, structured sparsity, Toeplitz and others - and 3) the number of eigenvalues required - all eigenvalues, inner/outer spectrum, smallest/largest eigenvalues and whether or not the eigenvectors are also computed. Since the focus of this research is on dense eigenvalue problems, only methods and approaches for dense eigenvalue problems are discussed in the following text.

The first (and very rough) division of eigenvalue solvers is whether all eigenvalues and the corresponding eigenvectors are needed or only a subset. In the first case, the so-called direct eigenvalue solvers are used, while in the second case, the iterative eigenvalue solvers are the better choice. In principle, all eigenvalue solvers are iterative processes, including the direct solvers. However, the direct solvers have the first steps in which the matrix A is reduced to a canonical form in a predetermined number of steps by applying orthogonal transformations (e.g., to the tridiagonal form for symmetric matrices or to the upper/lower Hessenberg form for non-symmetric ones). In the following steps, the eigenpairs are computed from the obtained canonical form using fast iterative eigensolvers such as MRRR [4], bisection and inverse iteration [5], Divide-and-Conquer [6] or QR iteration [3]. To obtain the eigenvectors of the initial matrix A, the accumulated reduction transformations are applied to the computed eigenvectors of the canonical form. This approach is sometimes referred to as the one-step approach.

The computational cost of obtaining the eigenpairs from a tridiagonal or Hessenberg matrix is $O(n^2)$ in the worst case, while computing the eigenpairs directly from A costs $O(n^3)$ (and is entirely cast in the form of the memory-bound BLAS -2 operations). The computational bottleneck now becomes the reduction phase (with a cost of $O(n^3)$), where only half of the computations of the reduction phase computations can be performed in terms of optimized, compute-bound BLAS-3 operations. To further improve performance (increase the ratio of BLAS-3 operations), a two-stage (or multi-stage) [7] approach is proposed that successively reduces the dense matrix A to the band matrix form (i.e a diagonal matrix with a few non-zero sub/super diagonals) and then reduces the band matrix to the tridiagonal or Hessenberg matrix form.

The generalized eigenvalue problems are usually solved by first reducing them to the standard form. This is usually done by reducing either the matrix A or B to diagonal/triangular form. The other approach is to simultaneously reduce A and B to upper/lower Hessenberg or tridiagonal form using the QZ algorithm [8] [3].

On the other hand, iterative solvers (working on the dense matrix A) are a common choice when a smaller subset of eigenvalues is sought. One of the most popular eigenvalue solvers is the QR algorithm [3], Krylov subspace iterations (such as Lanczos/Arnoldi methods), Jacobi methods, and power iterations [3].

III. High-performance eigenvalue solvers

In this section, we present the most popular high-performance numerical linear algebra packages that implement parallel eigenvalue solvers for dense eigenvalue problems. Nowadays, one can find a large number of different eigenvalue solvers, however, most of them are the result of single research papers, which are often not properly maintained, outdated, or whose developers have stopped further maintenance of the code. Therefore, in the rest of the article we will only consider eigenvalue solvers (and numerical packages) that are regularly maintained, freely available and mature.

Eigensolvers are divided into two categories, shared-memory and distributed-memory eigensolvers. The first category contains the eigensolvers tailored for shared-memory systems, i.e., eigensolvers that can exploit parallelism at the level of multi-core CPUs and optionally GPUs of a single machine. The second category comprises the eigensolvers that can distribute their workload across multiple compute nodes (machines), each consisting of multi-core CPUs and (optionally) GPU accelerators.

The shared memory solutions are usually based on the OpenMP and pthread programming models for exploiting parallelism at CPU level and GPU-specific models (CUDA, OpenCL) for GPU acceleration. The distributed memory models are mostly based on the Messing Passing Interface (MPI), a standardized and portable inter-machine communication standard.

A. Shared-memory solutions

The best known and most widely used package for linear algebra is called **LAPACK** [1]. It provides routines for solving standard and advanced numerical problems in linear algebra such as linear systems, least-square problem, eigenvalue, and singular value problems, and related matrix factorizations such as Schur, Cholesky, QR, and LU. It has been adopted and used by numerous software packages and forms the basis for almost all other linear algebra packages and libraries. LAPACK is known for its high performance and scalability on shared memory computer systems with multi-layered memory hierarchies. The algorithms are organized into matrix blocks (which act as basic building blocks) whose size can be tailored to a particular memory tier to achieve high performance on a variety of modern machines. The building blocks (basic matrix operations) depend on the highly tuned library - Basic Linear Algebra Subprograms (BLAS) [9].

LAPACK provides a large number of solvers for almost all types of eigenproblems. Dense eigensolvers are divided into three phases: 1) reducing the dense initial matrix into a condensed matrix form, 2) computing the eigenvalues and eigenvectors of the obtained matrix in condensed

form, 3) (if eigenvectors are needed) back-transforming the eigenvectors of the condensed form into those of the dense initial matrix. The dense symmetric (Hermitian) eigenproblems are solved by calling the driver routines xSYEV (xHYEV) or xSYEVD (xHYEVD), which compute all eigenvalues and optionally eigenvectors. The xSY,HEEVR and xSY,HEEVX routines, which use the MRRR and QR algorithms, respectively, compute a subset of eigenvalues and corresponding eigenvectors. The above routines, which are implemented as a one-step approach, are also available as two-step variants. LAPACK also implements the variants for solving symmetric and non-symmetric generalized eigenvalue problems.

LAPACK also provides computational routines for reducing a dense symmetric and rectangular matrix into tridiagonal form (xSYTRD and xGEBRD, respectively) and a nonsymmetric matrix into Hessenberg form (xGEHRD). The eigenpairs are computed directly from the obtained condensed forms using the routines xSTEDC and xSTEVR to compute all or selected eigenpairs of the symmetric tridiagonal matrix. The eigenvectors of the dense initial matrix can be obtained by applying the routine xGEMM (matrix-matrix multiplication) (back transformation).

One of the major drawbacks of LAPACK is its lack of support for modern computer architectures equipped with GPU accelerators. The **MAGMA** library [10]–[12] overcomes some of these shortcomings by redesigning and reimplementing most LAPACK routines for heterogeneous computer systems equipped with multi-core CPUs and one or more GPUs. The main idea is to offload and balance the computational load between CPUs and GPUs by selecting the best algorithms for each processing unit. MAGMA also targets only shared memory systems, but provides support for a wide range of different computer architectures and parallel models (CUDA, OpenCL, OpenMP). The library implements solvers for non-symmetric, symmetric (prefix *SY*) and Hermitian (prefix *HE*) standard and generalized symmetric/Hermitian eigenproblems. The routines (suffix with *x*) compute only selected eigenpairs and three different algorithms can be used to compute eigenpairs. For the standard symmetric/Hermitian eigenvalue problems, divide and conquer (routines xSY,HEEVDX), QR (xHEEVX) or MRRR (xHEEVR) can be used, and the counterparts for the generalized eigenvalue problem xSY/HEGVDX, xSY/HEGVX and xSY/HEGVR respectively. All routines internally reduce the input matrices into condensed forms (one-stage and two-stage variants) and then compute the eigenvalues of the obtained condensed matrices.

The input matrices can be stored in main memory before calling an eigenvalue solver, or in GPU memory. In the former case, the routines internally manage the data movement into GPU memory. The latter allows for easier integration into existing codes, especially if the input matrices are already in GPU memory, reducing the costly data movement from GPU to main memory.

cuSolver [13] is a vendor–specific library developed by NVIDIA and is tailored specifically for NVIDIA's GPUs. The package provides a set of the most commonly used numerical linear algebra routines from LAPACK. Both sparse and dense eigenproblems can be addressed on the multi-GPU machine (shared-memory system). Using the cuSolver both generalized and standard symmetric eigenvalue problems can be solved, however, the input matrices have to be in the GPU memory before a call of the routines *cuSolverDn<t>SYEVDX* (standard) and *cuSolver<t>SYGVDX* (generalized eigenproblem). Optional letter *X* at the end of the routine name says that the routine computes a selection of the eigenvalues and optionally eigenvectors. By the time of writing this paper, only symmetric eigenproblems can be solved using multi-GPU support, and a 1-D column block-cyclic data layout is used for that purpose.

Compared to the other linear algebra packages, **Eigen** [14] is a C++ template library that defines matrices and numerical solvers as class objects. Therefore, the library allows for easy programming that is closer to standard mathematical terminology, rather than calling a function (e.g., to compute general matrix-matrix calculations) and worrying about input/output arguments, sizes of the matrix, etc. The library provides routines for computing dense generalized/standard eigenproblems of general and symmetric/Hermitian matrices. The routines for standard eigenproblems first reduce the input matrix to Schur form and then compute the eigenpairs. In the case of the generalized eigenvalue problem, the QZ algorithm (for the general matrix) or the Cholesky decomposition (in the case of the symmetric/Hermitian matrix B) is applied to reduce the generalized problem to a standard eigenvalue problem.

B. Distributed memory solutions

The most popular linear algebra package and industry standard for distributed memory systems **ScaLAPACK** [2] (Scalable Linear Algebra PACKage) is an extension of the LAPACK library. ScaLAPACK does not support offloading of computations to GPU accelerators. The package provides routines for solving standard symmetric/Hermitian and generalized symmetric-definite eigenproblems when all eigenvalues and (optionally) the eigenvectors or a certain subset of the eigenspectrum (routine with the suffix *x*) are required. Routines PxSYEV (QR algorithm) and PxSYEVD (Divide-and-Conquer algorithm) compute all eigenvalues (and eigenvectors) of the dense symmetric or Hermitian eigenproblems, and routine PxSYGVX computes the eigenpairs of the dense generalized eigenproblems. In addition, ScaLAPACK implements a number of computational routines for a one-stage approach, including reduction to tridiagonal form (PxSYTRD), application of the implicitly stored orthogonal matrix Q (PxORMTR), and computation of the eigenvalues and eigenvectors of the symmetric tridiagonal matrix using look-ahead QR (xSTEQR2), bisection (PxSTEBZ), and inverse iteration (PxSTEIN). Internally, the ScaLAPACK routines assume that the input matrices

are distributed among the processes in the two-dimensional block-cyclic data distribution[1].

ELPA [15] [16] is a specialized library for computing the eigenvalues and eigenvectors of large dense symmetric matrices commonly encountered in quantum chemistry, computational materials science, biological network theory, and other fields. The goal of ELPA is to efficiently solve extremely large eigenvalue problems, too large to fit in the main memory of a single machine, on parallel distributed-memory systems.

The library contains two direct eigenvalue solvers to compute all or a subset of the eigenvalues and eigenvectors. The standard one-stage solver (ELPA1) first reduces the symmetric/Hermitian input matrix to tridiagonal form by Householder transformations, and then computes the eigenvalue pairs of the obtained tridiagonal system using the divide-and-conquer algorithm. When the eigenvectors are required, a back transformation of the eigenvectors is performed. As discussed in Section I), a direct reduction from dense to tridiagonal form quickly becomes a bottleneck for large matrices. Therefore, the second approach (referred to as ELPA2) is used, which reduces the dense input matrix to the tridiagonal form via an intermediate band matrix form. ELPA2 provides high performance on large eigenproblems and shows very good scaling on a large number of computational cores [16]. The current version of the ELPA library efficiently solves standard and generalized symmetric/Hermitian eigenvalue problems in double and double-complexity precission. The latest version of the ELPA library implements a distributed GPU-based two-stage eigensolver (ELPA2) [17] based on the cuBLAS library and special CUDA kernels to speed up the back transformation of eigenvectors.

In the one-stage approach, the major computational bottleneck is the reduction from dense symmetric to tridiagonal form. To overcome this bottleneck, a two-stage approach is used that reduces the computational cost by first reducing to the band matrix form. However, in this approach, the bottleneck is shifted from the reduction phase to the back-transformation phase. The library **EigenEXA** [18] overcomes this bottleneck by introducing a novel one-step approach in which the eigenpairs are computed directly from the band matrix, eliminating the need to back-transform the eigenvectors of the tridiagonal eigenproblem into those of the band matrix. The parallelism is based on highly tuned LAPACK and BLAS libraries for shared memory systems and ScaLAPACK for distributed memory systems.

FEAST package [19] is a high-performance library for distributed memory systems, ideal for large sparse eigenproblems when a subset of (interior) eigenpairs is needed. FEAST provides solvers for standard and generalized symmetric/Hermitian and non-Hermitian eigenvalue problems with interfaces for dense, banded, and sparse matrices. The core of the library is the FEAST algorithm [20], which is based on contour integration and density-matrix representation. Moreover, the solvers support the reverse communication interface [21] and can therefore be linked to any (external) MPI-based linear systems solver that uses a customized data distribution pattern. The only FEAST requirement is the availability of highly optimized BLAS and LAPACK packages. The library does not provide support for hybrid or GPU execution.

Intel **MKL** [22] is another vendor-specific library that implements a large number of mathematical routines, including BLAS, LAPACK, Fast Fourier Transformations, vectorized mathematical functions, and random number generators. MKL is a widely used and very popular math library because it provides very good performance on all Intel-based processors. It implements most of the ScaLAPACK routines for distributed memory systems and provides the same interface (see the ScaLAPACK section for more details). Currently, Intel MKL supports only Intel GPU accelerators, but some third-party efforts have been made to support GPUs from other vendors as well. MKL provides driver routines for matrices stored in 3 different storage types: Full, Packed and Band and can be stored in column or row major memory format for better compatibility with other programming models (e.g. Fortran). The library provides routines for generalized and standard, symmetric/Hermitian and non-symmetric eigenvalue problems. MKL implements routines from the ScaLAPACK package for solving symmetric and non-symmetric eigenvalue problems on distributed memory architectures.

Elemental [23] is a C++ library first announced in 2013 for dense and sparse linear algebra computations on distributed memory systems. Unfortunately, it has not been maintained since 2016, but an extension was developed by Lawrence Livermore National Lab, called Hydrogen [24]. Hydrogen added support for GPU execution (based on the CUDA architecture). The library provides several one-step routines for solving real symmetric or Hermitian eigenvalue problems. The input matrix is reduced to tridiagonal form (HermitianTridiag) and then a parallel MRRR (PMRRR) [25]) is applied to find all eigenpairs of the tridiagonal system.

The Software for Linear Algebra Targeting Exascale (**SLATE**) [26] project aims to provide modern basic linear algebra capabilities for HPC and future exascale systems. The software library, similar to LAPACK, provides the basic operations for dense matrices, including linear system solvers, least square solvers, and singular value and eigenvalue solvers. The plans for SLATE are very ambitious, as it aims to replace the very popular ScaLAPACK library, whose main drawback is the lack of support for modern hardware accelerators. The package implements the communication-avoiding solvers for symmetric eigenproblems. The solvers are based on the two-stage reduction to the tridiagonal and bidiagonal form.

(P_)ARPACK [27] [28] is a computational library that computes a subset of the eigenpairs of sparse or structured non-symmetric, symmetric (Hermitian) and generalized eigenproblems. The package is best suited for computing a smaller subset of eigenpairs of large sparse matrices, since

[1]https://www.netlib.org/scalapack/slug/node75.html

the algorithm is based on the matrix-vector product and reverse communication. However, it is shown [29] that the ARPACK library can also be very competitive when only a few eigenpairs of the dense generalized eigenproblem are needed. The distributed version (P_ARPACK) is based on BLACS and MPI message-passing interfaces.

LIS [30] [31] is a software library of iterative solvers for linear systems and eigenvalue problems. The library was originally developed to provide a scalable and efficient numerical solver for computing partial differential equations. It provides basic linear algebra operations for both dense and sparse matrices and iterative solvers for dense eigenvalue problems. However, due to the lack of documentation, it is not known which iterative eigenvalue solvers are implemented and which types of eigenvalue problems can be solved. The library supports inter-node parallelization (MPI) and intra-node parallelization (OpenMP), but does not support GPUs.

Table I summarizes the high performance libraries described above. The matrix structures *dense*, *sparse*, and *band* in the column `Sparsity` are denoted by *d*, *s*, and *b*, respectively. The `Eigenproblem` column lists the types of eigenproblems that the libraries can solve. *std* and *gen* denote standard and generalized eigenproblems, respectively, and *nsym* and *sym* stand for non-symmetric and symmetric/Hermitian eigenproblems, respectively.

IV. Performance overview

The analysis and performance results presented in this section were not conducted by this study. The information comes from previously published work and is discussed here. The goal is to provide initial insight into the performance and scalability of the distributed eigensolvers.

Analysis of the currently available computational packages for solving dense eigenvalue problems shows that of the 13 parallel eigenvalue solvers analyzed, 9 support execution on distributed memory architectures. Of the distributed memory libraries, 4 packages implement eigenvalue solvers that can use modern GPU accelerators, 3 of which support distributed execution on GPUs (ELPA, Hydrogen - a derivative of the Elemental library and SLATE). The last GPU-based distributed memory eigensolver, provided by the Intel MKL library, supports execution on Intel GPUs only for BLAS routines and a subset of the available LAPACK functions.

From the performance analysis point of view, it is almost impossible to collect the performance and scalability results of the target eigensolvers, although numerous research works (for example, on the performance of ScaLAPACK [32] from 2004) have been done to analyze the performance of distributed memory systems. Most of the papers that analyzed the performance of eigensolvers are more than 5 years old and the benchmarks were performed on now obsolete computer systems. As far as the author is aware, only a few performance analyzes have been performed recently and on state-of-the-art computers. One of the most recent researches benchmarked the ScaLAPACK, ELPA and EigenEXA [33] packages on the Oakforest- PACS [2] supercomputer in Japan. The authors showed that ScaLAPACK-only eigensolvers are inferior to those of a more modern ELPA library when solving large eigenproblems (matrix size up to $90,000$) on state-of-the-art machines. This finding is somewhat surprising since ScaLAPACK is still widely used, but understandable since ScaLAPACK was designed and constructed in the 1990s and tailored to the machines of the time, and no major redesigns have been made since then. The benchmark kernels are available on GitHub[3].

Another recent study from 2020 [34] reports the performance of SLATE library solvers for the SVD and generalized dense Hermitian eigenvalue problems on the Summit supercomputer[4]. The eigenvalue solver used was a two-stage algorithm using the QR algorithm to compute eigenvalue pairs of the tridiagonal system. The authors reported a $3\times$ speedup of the SLATE SVD solver without using accelerators and a nearly $4\times$ speedup when using NVIDIA GPUs, on 1 node, compared to ScaLAPACK. However, the performance converting the generalized to the standard Hermitian eigenproblem on 18 Summit nodes (without accelerators) is closely matching the ScaLAPACK, and achieves only a modest speedup (about 20%) with accelerators. Since the conversion can be fully casted in terms of compute-bound BLAS -3 operations, the performance gain from using the accelerators should be much higher. So much more research needs to be done on the SLATE library.

The performance and scalability of the latest GPU implementation of the ELPA library has been demonstrated at Summit (Oak Ridge Computing Facility, USA) and Talos[5] (Max Planck Computing and Data Facility, Garching, Germany) supercomputers. Performance analysis on eigenproblems ranging from $40,000$ to $100,000$ and on 64 compute nodes shows that two-stage approaches outperform their single-stage counterparts on both CPU and GPU. Unfortunately, the authors did not compare the performance of the ELPA solvers with other eigensolvers from other libraries.

V. Conclusions

Today, many eigenvalue solvers can use distributed memory systems, but only a few of them can exploit the full potential of modern computing architectures equipped with GPU accelerators. However, many of them are still under development, require additional performance tuning (such as SLATE), or cover only a limited number of eigenvalue problems, so their full performance is yet to be achieved in the coming years.

From the user's point of view, the lack of a more detailed performance analysis and side-by-side comparison of the available state-of-the-art eigenvalue solvers is a major drawback that could lead to an incomplete overview of the overall performance of the observed libraries. Often

[2]https://www.cc.u-tokyo.ac.jp/en/supercomputer/ofp/system.php
[3]https://github.com/eigenkernel/eigenkernel
[4]https://www.olcf.ornl.gov/summit/
[5]https://www.mpcdf.mpg.de/services/computing/linux/TALOS

TABLE I
HIGH-PERFORMANCE LIBRARIES FOR SOLVING EIGENPROBLEMS

Library	Distributed	GPU	Hybrid	Parallel model	Sparsity	Eigenproblem
LAPACK	×	×	×	OpenMP/pthreads	d/b	std/gen nsym/sym
MAGMA	×	yes (multi-GPU)	yes	OpenMP/pthreads/CUDA	d/s/b	std/gen nsym/sym
cuSolver	×	yes (multi-GPU)	×	CUDA	d/s	std/gen sym
EIGEN	×	×	×	OpenMP	d	std/gen nsym/sym
ScaLAPACK	yes	×	×	MPI/BLASC	d	
ELPA	yes	yes (GPU)	×	MPI/OpenMP/CUDA	d	std/gen sym
EigenEXA	yes	×	×	MPI/OpenMP	d	std sym
FEAST	yes	×	×	MPI	d/s/b	std/gen nsym/sym
Intel MKL	yes	yes (Intel GPU)	×	MPI/OpenMP/pthreads	d/b/s	std/gen nsym/sym
Elemental/Hydrogen	yes	yes (Hydrogen)	yes (Hydrogen)	MPI/OpenMP/(CUDA)	d	std sym
SLATE	yes	yes	yes	MPI/OpenMP/CUDA	d	std sym
P_ARPACK	yes	×	×	MPI/BLACS	s	std/gen nysm/sym
LIS	yes	×	×	MPI/OpenMP	d/s	

the performance of new libraries is compared with the ScaLAPACK library or different variants of the solvers from the same package are observed, but rarely compared with other modern libraries.

Future work will focus on performing a detailed performance analysis of distributed memory eigensolvers on modern computer architectures. The goal of future work will be to determine how well existing eigensolvers can handle scalability and performance challenges on current systems and to predict their behavior on future computing systems, which will most likely be very heterogeneous. Moreover, we believe that a fair performance comparison of different eigensolvers with the same input eigenproblems on a dedicated distributed-memory system would give a huge boost to the application of eigensolvers in solving concrete real-world problems, but would also show developers how to further improve eigensolver packages.

ACKNOWLEDGMENT

This work was supported by the Croatian Science Foundation under grant number HRZZ-UIP-2020-02-4559 and the European Regional Development Fund under the grant KK.01.1.1.01.009 - DATACROSS.

REFERENCES

[1] E. C. Anderson, Z. Bai, C. H. Bischof, L. S. Blackford, J. W. Demmel, J. Dongarra, J. J. Du Croz, A. Greenbaum, A. Hammarling, S. McKenney, and D. C. Sorensen, *{LAPACK} Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[2] J. Choi, J. Dongarra, R. Pozo, and D. Walker, "ScaLAPACK: a scalable linear algebra library for distributed memory concurrent computers," in *[Proceedings 1992] The Fourth Symposium on the Frontiers of Massively Parallel Computation*. IEEE Comput. Soc. Press, 1 2003, pp. 120–127. [Online]. Available: http://ieeexplore.ieee.org/document/234898/

[3] G. H. Golub and C. F. Van Loan, "Matrix Computations," p. 48, 1996.

[4] I. S. Dhillon, B. N. Parlett, and C. Vömel, "The design and implementation of the MRRR algorithm," *ACM Transactions on Mathematical ...*, vol. 32, no. 4, pp. 533–560, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1186788

[5] I. C. F. Ipsen, "Computing an Eigenvector with Inverse Iteration," *SIAM Review*, vol. 39, no. 2, pp. 254–291, 1 1997. [Online]. Available: http://epubs.siam.org/doi/10.1137/S0036144596300773

[6] M. Gu and S. C. Eisenstat, "A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem," *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 1, pp. 172–191, 1 1995.

[7] C. H. Bischof, B. Lang, and X. Sun, "Algorithm 807: The SBR Toolbox—software for successive band reduction," *ACM Transactions on Mathematical Software*, vol. 26, no. 4, pp. 602–616, 2000.

[8] C. B. Moler and G. W. Stewart, "An Algorithm for Generalized Matrix Eigenvalue Problems," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 241–256, 4 1973. [Online]. Available: http://epubs.siam.org/doi/10.1137/0710024

[9] F. T. Krogh, "Blas- Basic Linear Algebra Subprograms," vol. 5, pp. 1–22, 1994. [Online]. Available: http://www.netlib.org/blas/

[10] S. Tomov, J. Dongarra, and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Computing*, vol. 36, no. 5-6, pp. 232–240, 6 2010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167819109001276

[11] F. Song, S. Tomov, and J. Dongarra, "Enabling and scaling matrix computations on heterogeneous multi-core and multi-GPU systems," in *Proceedings of the 26th ACM international conference on Supercomputing - ICS '12*, 2012, p. 365. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2304576.2304625

[12] The University of Tenneesse, "{MAGMA} Matrix Algebra on GPU and Multicore Architectures," 2014. [Online]. Available: https://icl.cs.utk.edu/magma/

[13] NVIDIA, "cuSOLVER :: CUDA Toolkit Documentation," 2019. [Online]. Available: https://docs.nvidia.com/cuda/cusolver/

[14] G. Guennebaud, B. Jacob, and others, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[15] ELPA, "Eigenvalue SoLvers for Petaflop-Applications (ELPA)," 2014. [Online]. Available: https://elpa.mpcdf.mpg.de/

[16] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, "The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science," *Journal of Physics: Condensed Matter*, vol. 26, no. 21, p. 213201, 5 2014. [Online]. Available: https://iopscience.iop.org/article/10.1088/0953-8984/26/21/213201

[17] V. W. z. Yu, J. Moussa, P. Kůs, A. Marek, P. Messmer, M. Yoon, H. Lederer, and V. Blum, "GPU-acceleration of the ELPA2 distributed eigensolver for dense symmetric and hermitian eigenproblems," *Computer Physics Communications*, vol. 262, p. 107808, 5 2021.

[18] T. Fukaya, T. Imamura, and Y. Yamamoto, "A case study on modeling the performance of dense matrix computation: Tridiagonalization in the eigenexa eigensolver on the k computer," in *Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018*. Institute of Electrical and Electronics Engineers Inc., 8 2018, pp. 1113–1122.

[19] J. Kestyn, V. Kalantzis, E. Polizzi, and Y. Saad, "PFEAST: A High Performance Sparse Eigenvalue Solver Using Distributed-Memory Linear Solvers," in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, vol. 0. IEEE, 11 2016, pp. 178–189.

[20] E. Polizzi, "Density-matrix-based algorithm for solving eigenvalue problems," *Physical Review B - Condensed Matter and Materials Physics*, vol. 79, no. 11, p. 115112, 3 2009. [Online]. Available: https://journals.aps.org/prb/abstract/10.1103/PhysRevB.79.115112

[21] J. Dongarra, V. Eijkhout, and A. Kalhan, "Reverse communication interface for linear algebra templates for iterative

methods," *UT, CS-95-291, May*, 1995. [Online]. Available: ftp://ftp.artfiles.org/netlib.org/lapack/lawnspdf/lawn99.pdf

[22] "Intel Math Kernel Library. https://software.intel.com/en-us/mkl [10 December 2017]." [Online]. Available: https://software.intel.com/en-us/mkl

[23] J. Poulson, B. Marker, R. A. Van De Geijn, J. R. Hammond, and N. A. Romero, "Elemental: A new framework for distributed memory dense matrix computations," *ACM Transactions on Mathematical Software*, vol. 39, no. 2, pp. 1–24, 2 2013. [Online]. Available: https://dl.acm.org/doi/10.1145/2427023.2427030

[24] LLNL, "GitHub - LLNL/Elemental: Distributed-memory, arbitrary-precision, dense and sparse-direct linear algebra, conic optimization, and lattice reduction." [Online]. Available: https://github.com/LLNL/Elemental

[25] M. Petschow, E. Peise, and P. Bientinesi, "High-Performance Solvers for Dense Hermitian Eigenproblems," *SIAM Journal on Scientific Computing*, vol. 35, no. 1, pp. C1–C22, 1 2013. [Online]. Available: http://epubs.siam.org/doi/10.1137/110848803

[26] M. Gates, J. Kurzak, A. Charara, A. Yarkhan, and J. Dongarra, "SLATE: Design of a modern distributed and accelerated linear algebra library," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*. New York, NY, USA: IEEE Computer Society, 11 2019, pp. 1–18. [Online]. Available: https://dl.acm.org/doi/10.1145/3295500.3356223

[27] R. Lehoucq, D. C. Sorensen, and C. Yang, *Arpack User's Guide: Solution of Large-Scale Eigenvalue Problems With Implicityly Restorted Arnoldi Methods*, 1998.

[28] K. J. Maschhoff and D. C. Sorensen, "P_ARPACK: An efficient portable large scale eigenvalue package for distributed memory parallel architectures," in *Applied Parallel Computing in Industrial ...*, 1996, pp. 478–486.

[29] J. I. Aliaga, P. Bientinesi, D. Davidović, E. Di Napoli, F. D. Igual, and E. S. Quintana-Ortí, "Solving dense generalized eigenproblems on multi-threaded architectures," *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11 279–11 289, 7 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S009630031200505X

[30] "Lis: a Library of Iterative Solvers for Linear Systems." [Online]. Available: https://www.ssisc.org/lis/ http://www.ssisc.org/lis/index.en.html

[31] A. Nishida, "Experience in Developing an Open Source Scalable Software Infrastructure in Japan," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, 2010, vol. 6017 LNCS, no. PART 2, pp. 448–462.

[32] E. Breitmoser and A. G. Sunderland, "A performance study of the PLAPACK and ScaLAPACK Eigensolvers on HPCx for the standard problem," Technical Report HPCxTR0406, The HPCx Consortium, Tech. Rep., 2004.

[33] K. Tanaka, H. Imachi, T. Fukumoto, A. Kuwata, Y. Harada, T. Fukaya, Y. Yamamoto, and T. Hoshi, "EigenKernel: A middleware for parallel generalized eigenvalue solvers to attain high scalability and usability," *Japan Journal of Industrial and Applied Mathematics*, vol. 36, no. 2, pp. 719–742, 7 2019. [Online]. Available: https://doi.org/10.1007/s13160-019-00361-7

[34] M. Gates Mohammed Al Farhan Ali Charara Jakub Kurzak Dalal Sukkari Asim YarKhan Jack Dongarra and A. Farhan, "Implementing Singular Value and Symmetric/Hermitian Eigenvalue Solvers," Tech. Rep., 2020.