Equipping Sparse Solvers For Exascale

Christie L. Alappat, Andreas Alvermann, Achim Basermann, Holger Fehske, Yasunori Futamura, Martin Galgon, Georg Hager, Sarah Huber, Akira Imakura, Masatoshi Kawai, Moritz Kreutzer, Bruno Lang, Kengo Nakajima, Melven Röhrig-Zöllner, Tetsuya Sakurai, Faisal Shahzad, Jonas Thies, and Gerhard Wellein

Abstract The ESSEX project has investigated programming concepts, data structures, and numerical algorithms for scalable, efficient, and robust sparse eigenvalue solvers on future heterogeneous exascale systems. Starting without the burden of legacy code, a holistic performance engineering process could be deployed across the traditional software layers to identify efficient implementations and guide sustainable software development. At the basic building blocks level, a flexible MPI+X programming approach was implemented together with a new sparse data structure (SELL-C-σ) to support heterogeneous architectures by design. Furthermore, ESSEX focused on hardware-efficient kernels for all relevant architectures and efficient data structures for block vector formulations of the eigensolvers. The algorithm layer addressed standard, generalized, and nonlinear eigenvalue problems and provided some widely usable solver implementations including a block Jacobi-Davidson algorithm, contour-based integration schemes, and filter polynomial approaches. Adding to the highly efficient kernel implementations, algorithmic advances such as adaptive precision, optimized filtering coefficients, and preconditioning have further improved time to solution. These developments were guided by

Gerhard Wellein, Christie L. Alappat, Georg Hager, Moritz Kreutzer, Faisal Shahzad
Department of Computer Science and Erlangen Regional Computing Center (RRZE), Friedrich-Alexander Universität Erlangen-Nürnberg, e-mail: gerhard.wellein@fau.de

Melven Röhrig-Zöllner, Jonas Thies, Achim Basermann
Simulation and Software Technology, German Aerospace Center

Martin Galgon, Sarah Huber, Bruno Lang
Angewandte Informatik, Bergische Universität Wuppertal

Andreas Alvermann, Holger Fehske
Institute of Physics, Universität Greifswald

Masatoshi Kawai, Kengo Nakajima
Information Technology Center, University of Tokyo

Yasunori Futamura, Akira Imakura, Tetsuya Sakurai
Department of Computer Science, University of Tsukuba

the field of quantum physics applications, and especially by current topics such as topological insulator systems or problems from graphene research. For these, ScaMaC, a scalable matrix generation framework for a broad set of quantum physics problems, was developed. As the central software core of ESSEX, the PHIST library for sparse linear and eigenvalue problems has been established. It abstracts algorithmic developments from low-level optimization. Finally, central ESSEX software components and solvers have demonstrated scalability and hardware efficiency on up to 256 K cores using million-way process/thread-level parallelism.

1 Introduction

The efficient solution of linear systems or eigenvalue problems involving large sparse matrices has been an active research field in parallel and high performance computing for many decades. Software packages like Trilinos [33] or PETSc [9] have been developed to great maturity, and algorithmic improvements were accompanied by advances in programming abstractions addressing, e.g., node-level heterogeneity (cf. Kokkos [19]). Completely new developments such as Ginkgo[1] are rare and do not focus in large-scale applications or node-level efficiency.

Despite projections from the late 2000s, hardware architectures have not developed away from traditional clustered multicore systems. However, a clear trend of increased node-level parallelism and heterogeneity has been observed. Although several new architectures entered the field (and some vanished again), the basic concepts of core-level code execution and data parallelism have not changed. This is why the MPI+X concept is still a viable response to the challenge of hardware diversity.

Performance analysis of highly parallel code typically concentrated on scalability, but provably optimal node-level performance was rarely an issue. Moreover, strong abstraction boundaries between linear algebra building blocks, solvers, and applications made it hard to get a holistic view on a minimization of time to solution, encompassing optimizations in the algorithmic and implementation dimensions.

In this setting, the ESSEX project took the opportunity to start from a clean slate, deliberately breaking said abstraction boundaries to investigate performance bottlenecks together with algorithmic improvements from the core to the highly parallel level. Driven by the targeted application fields, bespoke solutions were developed for selected algorithms and applications. The experience gained in the development process will lead the way towards more generic approaches rather than compete with established libraries in terms of generality. The overarching motif was a consistent performance engineering process that coordinated all performance-relevant activities across the different software layers [1–3, 20, 53–56, 63, 66].

Consequently, the ESSEX parallel building blocks layer implemented in the GHOST library [57] supports MPI+X, with X being a combination of node-level

[1] `https://github.com/ginkgo-project/ginkgo`

programming models able to fully exploit hardware heterogeneity, functional parallelism, and data parallelism. Despite fluctuations in hardware architectures and new programming models hitting the market every year, OpenMP or CUDA is still the most promising and probably most sustainable choice for X, and ESSEX-II adhered to it. In addition, engineering highly specialized kernels including sparse-matrix multiple-vector operations and appropriate data structures for all relevant compute architectures provided the foundation for hardware- and energy-efficient large-scale computations.

Building on these high-performance building blocks, one focus of the algorithm layer was put on the block formulation of Jacobi-Davidson [66] and filter diagonalization [53] methods, the hardware efficiency of preconditioners [46–48], and the development of hardware-aware coloring schemes [1]. In terms of scalability, the project has investigated new contour-based integration eigensolvers [24, 26] that can exploit additional parallelism layers beyond the usual data parallelism. The solvers developed in ESSEX can tackle standard, generalized, and nonlinear eigenvalue problems and may also be used to extract large bulks of extremal and inner eigenvalues.

The applications layer applies the algorithms and building blocks to deliver scalable solutions for topical quantum materials like graphene, topological insulators, or superconductors, and nonlinear dynamical systems like reaction-diffusion systems. A key issue for large-scale simulations is the scalable (in terms of size and parallelism) generation of the sparse matrix representing the model Hamiltonian. Our matrix generation framework ScaMaC can be integrated into application code to allow the on-the-fly, in-place construction of the sparse matrix. Beyond the ESSEX application fields, matrices from many other relevant areas can be produced by ScaMaC.

The PHIST library [81] is the sustainable outcome of the performance-centric efforts in ESSEX. It is built on a rigorous software and performance engineering process, comprises diverse solver components, and supports multiple backends (e.g., Trilinos, PETSc, ESSEX kernels). It also interfaces to multiple languages such as C, C++, Fortran 2003, and Python. The CRAFT library [75] provides user-friendly access to fault tolerance via checkpoint/restart and automatic recovery for iterative codes using standard C++.

Scalabilty, performance, and portability have been tested on three top-10 supercomputers covering the full range of architecures available during the ESSEX project time frame: Piz Daint[2] (heterogeneous CPU-GPU), OakForest-PACS[3] (many-core), and SuperMUC-NG[4] (standard multi-core).

This review focuses on important developments in ESSEX-II. After presenting a brief overview of the most relevant achievements in the first project phase ESSEX-I in Section 2, Section 3 details algorithmic developments in ESSEX-II, notably with respect to preconditioners and projection-based methods for obtaining

[2] `https://www.cscs.ch/computers/piz-daint/`

[3] `https://www.cc.u-tokyo.ac.jp/en/supercomputer/ofp/service/`

[4] `https://doku.lrz.de/display/PUBLIC/SuperMUC-NG`

inner eigenvalues. Moreover, we present the RACE (Recursive Algebraic Coloring Engine) method, which delivers hardware-efficient graph colorings for parallelization of algorithms and kernels with data dependencies. In Section 4 we showcase performance and parallel efficiency numbers for library components developed in ESSEX-II that are of paramount importance for the application work packages: GPGPU-based tall and skinny matrix-matrix multiplication and the computation of inner eigenvalues using polynomial filter techniques. Section 5 describes the software packages that were developed to a usable and sustainable state, together with their areas of applicability. In Section 6 we show application results from the important areas of quantum physics and nonlinear dynamical systems. Finally, in Section 7 we highlight the collaborations sparked and supported by SPPEXA through the ESSEX-II project.

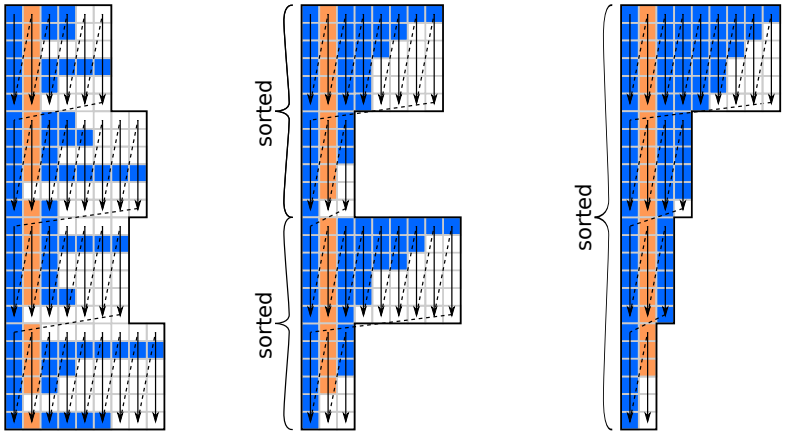
2 Summary of the ESSEX-I software structure

The Exascale-enabled Sparse Solver Repository (ESSR) was developed along the requirements of the algorithms and applications under investigation in ESSEX. It was not intended as a full-fledged replacement of existing libraries like Trilinos[5] [33], but rather as a toolbox that can supply developers with blueprints as a starting point for their own developments. In ESSEX-I, the foundations for a sustainable software framework were laid. See Section 5 for developments in ESSEX-II.

The initial version of the ESSR [80] comprised four components:

- GHOST (General, Hybrid and Optimized Sparse Toolkit) [57], a library of basic sparse and dense linear algebra building blocks that are not available in this form in other software packages. The development of GHOST was strictly guided by performance engineering techniques; implementations of standard kernels such as sparse matrix-vector multiplication (spMVM) and sparse matrix-multiple-vector multiplication (spMMVM) as well as tailor-made fused kernels, for instance those employed in the Kernel Polynomial Method (KPM) [84], were modeled using the roofline model. GHOST supports, by design, strongly heterogeneous environments using the MPI+X approach. See [52] for a comprehensive overview of GHOST and its building blocks.
- ESSEX-Physics, a collection of prototype implementations of polynomial eigensolvers such as the KPM and Chebyshev Filter Diagonalization (ChebFD). These were implemented on top of GHOST using tailored kernels and were shown to perform well on heterogeneous CPU-GPU systems [55].
- PHIST (Pipelined Hybrid-parallel Iterative Solver Toolkit), which comprises Jacobi-Davidson type eigensolvers and Krylov methods for linear systems. One important component is a test framework that allows for continuous integration (CI) throughout the development cycle. PHIST can not only use plain GHOST as

[5] `https://trilinos.org/`



(a) SELL-6-1, $\beta = 0.51$ (b) SELL-6-12, $\beta = 0.66$ (c) SELL-6-24, $\beta = 0.84$

Fig. 1: Variants of the SELL-C-σ storage format. Arrows indicate the storage order of matrix values and column indices. Image from [54].

its basic linear algebra layer; it is also equipped with fallback kernel implementations and adapters for the Trilinos and Anasazi libraries. A major achievement in the development of PHIST was an efficient block Jacobi-Davidson eigenvalue solver, which could be shown to have significant performance advantages over nonblocked versions when using optimized building blocks from GHOST [66].

- BEAST (Beyond fEAST), which implements innovative projection-based eigensolvers motivated by the contour integration-based FEAST method [24]. The ESSEX-I project has contributed to improving FEAST in two ways: by proposing techniques for solving or avoiding the linear systems that arise, and by improving robustness and performance of the algorithmic scheme.

A pivotal choice for any sparse algorithm implementation is the sparse matrix storage format. In order to avoid data conversion and the need to support multiple hardware-specific formats in a single code, we developed the SELL-C-σ format [54]. It shows competitive performance on the dominating processor architectures in HPC: standard multicore server CPUs with short-vector single instruction multiple data (SIMD) capabilities, general-purpose graphics processing units (GPGPUs), and many-core designs with rather weak cores such as the Intel Xeon Phi. SELL-C-σ circumvents the performance penalties of matrices with few nonzero entries per row on architectures on which SIMD vectorization is a key element for performance even with memory-bound workloads.

In order to convert a sparse matrix to SELL-C-σ, its rows are first sorted according to their respective numbers of nonzeros. This sorting is performed across row blocks of length σ. After that, the matrix is cut into row blocks of length C. Within each block, rows are padded with zeros to equal length and then stored in

column-major order. See Figure 1 for visualizations of SELL-C-σ with $C = 6$ and $\sigma \in \{1, 12, 24\}$. Incidentally, known and popular formats can be recovered as corner cases: SELL-1-1 is the well-known CSR storage format and SELL-N-1 is ELLPACK. The particular choice of C and σ influences the performance of the spMVM operation; optimal values are typically matrix- and hardware-dependent. However, in practice one can usually find parameters that yield good performance across architectures for a particular matrix. A roofline performance model was constructed in [54] that sets an upper limit for the spMVM performance for any combination of matrix and architecture. This way, "bad" performance is easily identified. SELL-C-σ was quickly adopted by the community and is in use, in pure or adapted form, in many performance-oriented projects [5, 6, 28, 61, 82].

3 Algorithmic Developments

In this section we describe selected developments within ESSEX-II on the algorithmic level, in particular preconditioners for the solution of linear systems that occur in the eigensolvers, a versatile framework for computing inner eigenvalues, and a nonlinear eigensolver. We also cover a systematic comparison of contour-based methods. We close the section with the introduction of RACE, which is an algorithmic development for graph coloring guided by the constraints of hardware efficiency.

3.1 Preconditioners (ppOpen-SOL)

Two kinds of solvers have been developed: a preconditioner targeting the ill-conditioned large scale problems arising in the BEAST-C method (cf. Sect. 3.2) and a multigrid solver targeting problems arising from finite difference discretizations of partial differential equations (PDEs).

3.1.1 Regularization

The BEAST-C method leads to a large number of ill-conditioned linear systems with complex diagonal shifts [26]. Furthermore, in many of our quantum physics applications, the system matrices have small (and sometimes random) diagonal elements. In order to apply a classic incomplete Cholesky (IC) factorization preconditioner, we used two types of regularization to achieve robustness: a blocking technique (BIC) and an additional diagonal shift [47]. Using this approach, we solved a set of 120 prototypical linear systems from this context (e.g., BEAST-C applied to quantum physics applications). Due to the complex shift, the system matrix is symmetric but not Hermitian. Hence we use an adaptation of the Conjugate Gradient (CG) method

for complex symmetric matrices called COCG (conjugate orthogonal conjugate gradient [83]).

The blocking technique is a well-known approach for improving the convergence rate. In this study, we apply the technique not only for better convergence but also for more robustness. The diagonal entries in the target equations are small. By applying the blocking technique, the diagonal blocks to be inverted include larger off-diagonal entries.

The diagonal shifting is a direct measure for transforming the ill-conditioned matrices to be more diagonally dominant before performing the incomplete factorization. On the other hand, this may deteriorate the convergence of the overall method. We therefore investigate the best value for the diagonal shifting for our applications.

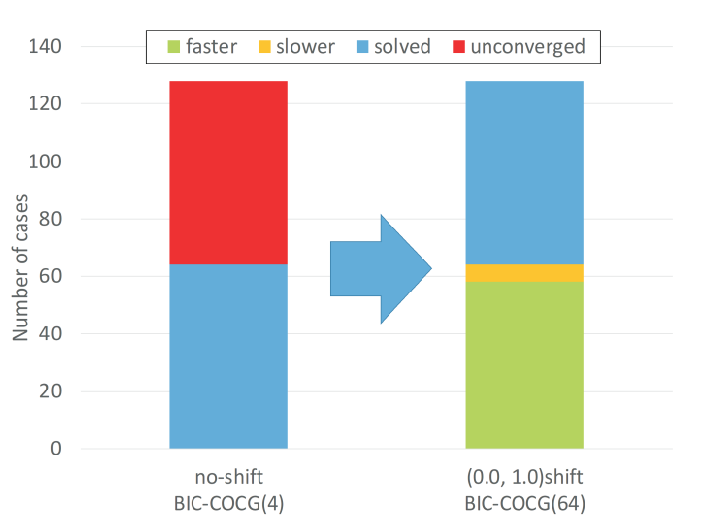


Fig. 2: Effect of the regularized IC preconditioner with the COCG method. By using the diagonal shifted block IC-COCG (BIC-COCG), we can solve all test problems from our benchmark set.

Figure 2 shows the effect of the regularized IC preconditioner with the COCG method. By using the diagonal shifted block IC-COCG (BIC-COCG), we solve all target linear systems.

3.1.2 Hierarchical parallel reordering

In this section, we present scalability results for the BIC preconditioner parallelized by a hierarchical parallel graph coloring algorithm. This approach yields an almost

constant convergence rate with respect to the number of compute nodes, and good parallel performance.

Node-wise multi-coloring (with domain decomposition between nodes) is widely used for parallelizing IC preconditioners on clusters of shared memory CPUs. Such "localized" multi-coloring leads to a loss of robustness of the regularized IC-COCG method, and the convergence rate decreases at high levels of parallelism. To solve this problem, we parallelize the block IC preconditioner for the hybrid-parallel cluster system. In addition, we proposed the hierarchical parallelization for the multi-coloring algorithms [46]. This versatile scheme allows us to parallelize almost any multi-coloring algorithm.

Figure 3 shows the number of iterations and computational time of the BIC-COCG method on the Oakleaf-FX cluster, using up to 4,800 nodes. The benchmark matrix is the Hamiltonian of a graphene sheet simulation with more than 500 million linear equations, for which interior eigenvalues are of interest [26]. Hierarchical parallelization yields almost constant convergence with respect to the number of nodes. The computational time with 4,600 nodes is 30 times smaller than with 128 nodes, amounting to a parallel efficiency of 83.5% if the 128-node case is taken as the baseline.

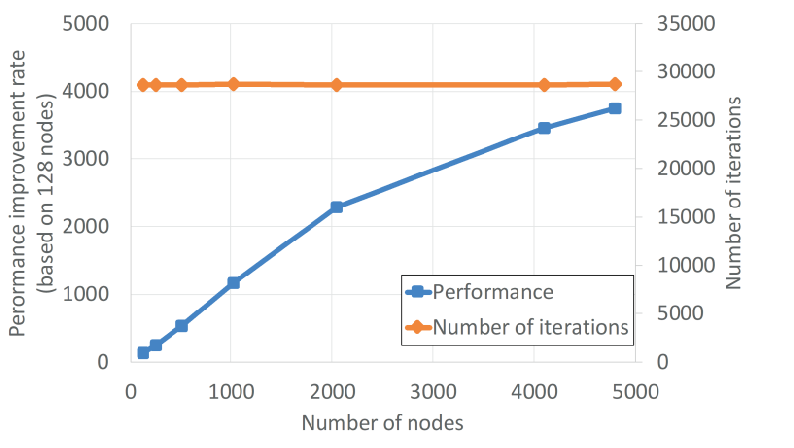


Fig. 3: Computational time and convergence of BIC-COCG for a graphene benchmark problem (strong scaling).

3.1.3 Multiplicative Schwarz-type block red-black Gauß-Seidel smoother

Multigrid methods are among the most useful preconditioners for elliptic PDEs. In [49] we proposed a multiplicative Schwarz block red/black Gauß-Seidel (MS-BRB-GS) smoother for geometric multigrid methods. It is a modified version of

the block red-black Gauß-Seidel (BRB-GS) smoother that improves convergence rate and data locality by applying multiple consecutive Gauß-Seidel sweeps on each block.

The unknowns are divided into blocks so that the amount of data for processing each block fits into the cache, and α Gauß-Seidel iterations are applied to the block per smoother step. The computational cost for the additional iterations is much lower than for the first iteration because of data locality.

Figure 4 shows the effect of the MS-BRB-GS(α) smoother on a single node of the ITO system (Intel Xeon Gold 6154 (Skylake-SP) Cluster at Kyushu University). By increasing the number of both pre- and post-smoothing steps, the number of iterations is decreased. In the best case, MS-BRB-GS is 1.64× faster than BRB-GS.

		Post-smoothing									
		1	2	3	4	5	6	7	8	9	10
Pre-smoothing	1	2.74/10	2.25/8	2.30/8	2.39/8	2.21/7	2.29/7	2.38/7	2.48/7	2.59/7	2.72/7
	2	2.25/8	2.01/8	2.11/7	1.82/6	1.90/6	1.99/6	2.08/7	2.17/6	2.28/6	2.36/6
	3	2.02/7	2.08/7	1.82/6	1.87/6	1.95/6	2.03/6	2.12/6	2.17/6	2.28/6	2.36/6
	4	2.08/7	1.82/6	1.87/6	1.90/6	1.67/5	1.74/5	1.82/5	1.89/5	1.96/5	2.03/5
	5	2.15/7	1.89/6	1.97/6	1.66/5	1.72/5	1.74/5	1.84/5	1.95/5	2.00/5	2.09/5
	6	2.25/7	1.97/6	1.68/5	1.66/5	1.79/5	1.85/5	1.84/5	1.95/5	2.08/5	2.16/5
	7	2.36/7	2.07/6	1.77/5	1.79/5	1.85/5	1.93/5	1.99/5	2.07/5	2.08/5	2.24/5
	8	2.10/6	2.15/6	1.82/5	1.88/5	1.94/5	2.01/5	2.08/5	2.17/5	2.24/5	2.33/5
	9	2.22/6	2.15/6	1.82/5	1.96/5	2.02/5	2.08/5	2.17/5	2.17/5	2.32/5	2.41/5
	10	2.31/6	2.36/6	2.01/5	2.04/5	2.10/5	2.18/5	2.24/5	2.33/5	2.40/5	2.47/5

fast

slow

Fig. 4: Computational time and number of iterations of a geometric multigrid solver with the MS-BRB-GS(α) smoother.

3.2 The BEAST framework for interior definite generalized eigenproblems

The BEAST framework targets the solution of interior definite eigenproblems

$$AX = BX\Lambda \ ,$$

i.e., for finding all eigenvectors and eigenvalues of a definite matrix pair (A,B), with A and B Hermitian and B additionally positive definite, within a given interval $[\underline{\lambda},\overline{\lambda}]$. The framework is based on the Rayleigh-Ritz subspace iteration procedure, in particular the spectral filtering approach: Arbitrary continuous portions of the spectrum may be selected for computation with appropriate filtering functions that are applied via an implicit approximate projector to compute a suitable subspace

basis. Starting with an initial subspace Y, the following three main steps are repeated until a suitable convergence criterion is met:

Compute a subspace U by approximately projecting Y
Rayleigh-Ritz extraction: solve the reduced eigenproblem $A_U V = B_U V \Lambda$,
where $A_U = U^H A U$, $B_U = U^H B U$, and let $X = UV$
Obtain new Y from X or U

In the following we highlight some of BEAST's algorithmic features, skipping other topics such as locking converged eigenpairs, adjusting the dimension of the subspace, and others.

3.2.1 Projector types

BEAST provides three variants of approximate projectors. First, polynomial approximation (BEAST-P) using Chebyshev polynomials, which only requires matrix vector multiplications but is restricted to standard eigenproblems. Second, Cauchy integral-based contour integration (BEAST-C), as in the FEAST method [64]. As a third method, an iterative implementation of the Sakurai-Sugiura method [69] is available (BEAST-M), which shares algorithmic similarities with FEAST. In the following we briefly elaborate on the algorithmic ideas.

- In BEAST-P, we have $U = p(A) \cdot Y$ with a polynomial $p(z) = \sum_{k=0}^{d} c_k T_k(z)$ of suitable degree d. Here, T_k denotes the kth Chebyshev polynomial,

$$T_0(z) \equiv 1, \quad T_1(z) = z, \quad T_k(z) = 2z \cdot T_{k-1}(z) - T_{k-2}(z),\ k \geq 2.$$

  Due to the use of the T_k, this method is also known as Chebyshev filter diagonalization.
  In addition to well-known methods for computing the coefficients c_k [18, 63], BEAST also provides the option of using new, improved coefficients [25]. Their computation depends on two parameters, μ and σ, and for suitable combinations of these, the filtering quality of the polynomial can be improved significantly; see Figure 5, which shows the "gain," i.e., the reduction of the width of those λ values *outside* the search interval, for which a damping of corresponding eigenvectors by at least a factor 100 cannot be guaranteed. For some combinations (σ, μ), marked red in the picture, this "no guarantee" area can be reduced by a factor of more than 2, which in turn allows using lower-degree polynomials to achieve comparable overall convergence. A parallelized method for finding suitable parameter combinations and computing the c_k is included with BEAST.
- In BEAST-C, the exact projection

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z\ (zB - A)^{-1} BY$$

  (integration is over a contour Γ in the complex plane that encloses the eigenvalues $\lambda \in [\underline{\lambda}, \overline{\lambda}]$, but no others) is approximated using an N-point quadrature

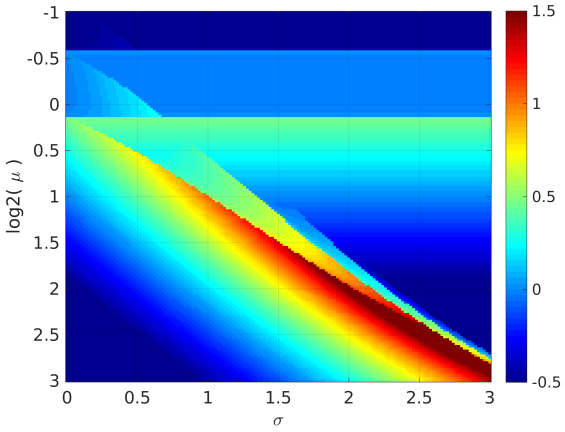


Fig. 5: Base-2 log of the "gain" from using modified coefficients with parameters (σ,μ) for the interval $[\underline{\lambda},\overline{\lambda}] = [-0.584,-0.560]$ (matrix scaled such that $\text{spec}(A) = [-1,+1]$) and degree $d = 1600$.

rule,

$$U = \sum_{j=1}^{N} \omega_j (z_j B - A)^{-1} BY,$$

leading to N linear systems, where the number of right-hand sides (RHS) corresponds to the dimension of the current subspace U (and Y).

- BEAST-M is also based on contour integration, but moments are used to reduce the number of RHS in the linear systems. Taking M moments, we have

$$U = [U_0,\ldots,U_{M-1}] \quad \text{with} \quad U_k = \sum_{j=1}^{N} \omega_j z_j^k (z_j B - A)^{-1} BY,$$

and thus an M times smaller number of RHS (dimension of Y) is sufficient to achieve the same dimension of U.

The linear systems in the contour-based schemes may be ill-conditioned if the integration points z_j are close to the spectrum (this happens, e.g., for narrow search intervals $[\underline{\lambda},\overline{\lambda}]$); cf. also 3.1 and 3.4 for approaches to address this issue.

3.2.2 Flexibility, adaptivity and auto-tuning

The BEAST framework provides flexibility at the algorithmic, parameter, and working precision levels, which we describe in detail in the following.

Algorithmic level

The projector can be chosen from the three types described above, and the type may even be changed between iterations. In particular, an innovative subspace-iterative version of Sakurai-Sugiura methods (SSM) has been investigated for possible cost savings in the solution of linear systems via a limited subspace size and the overall reduction of number of right hand sides over iterations by using moments. Given, however, the potentially reduced convergence threshold with a constrained subspace size, we support switching from the multi-moment method, BEAST-M, to a single-moment method, BEAST-C. The efficiency, robustness, and accuracy of this approach in comparison with traditional SSM and FEAST has been explored [35].

We further studied this scheme along with another performance-based implementation of SSM, z-PARES [68, 70]. These investigations considered the scaling and computational cost of the libraries as well as heuristics for parameter choice, in particular with respect to the number of quadrature nodes. We observed that the scaling behavior improved when the number of quadrature nodes increased, as seen in Figure 6. As the linear systems solved at each quadrature node are independent and the quality of numerical integration improves with increased quadrature degree, exploiting this property makes sense, particularly within the context of exascale computations. However, it is a slightly surprising result, as previous experiments with FEAST showed diminishing returns for convergence with increased quadrature degree [24], something we do not observe here.

Parameter level

In addition to the projector type, several algorithmic parameters determine the efficiency of the overall method, most notably the dimension of the subspace and the degree of the polynomial (BEAST-P) or the number of integration nodes (BEAST-C and BEAST-M).

With certain assumptions on the overall distribution of the eigenvalues, clear recommendations for optimum subspace size (as a multiple of the number of expected eigenvalues) and the degree can be given, in the sense that overall work is minimized. For more details, together with a description of a performance-tuned kernel for the evaluation of $p(A) \cdot Y$, the reader is referred to [63].

If such information is not available, or for the contour integration-type projectors, a heuristic has been developed that automatically adjusts the degree (or number of integration nodes) during successive iterations in order to achieve a damping of the unwanted components by a factor of 100 per iteration, which leads to close-to-optimum overall effort; cf. [23].

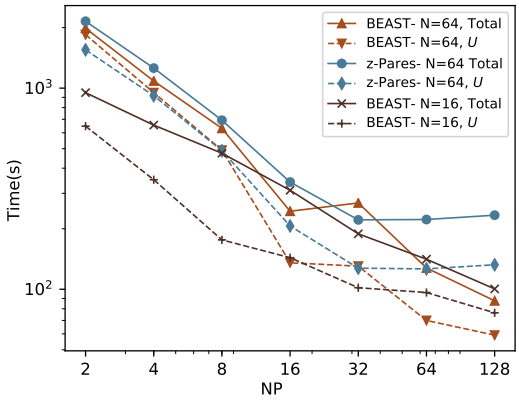
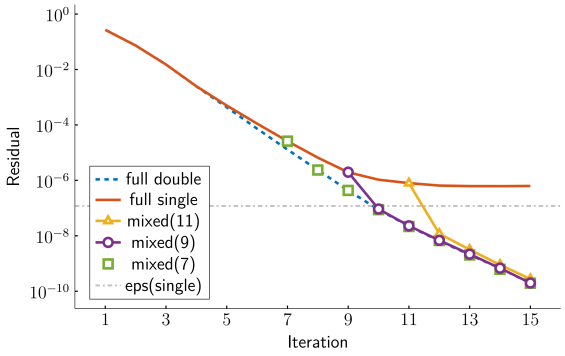


Fig. 6: Strong scaling of BEAST and z-Pares for a 1M $\times$ 1M standard eigenproblem based on a graphene sheet of dimension 2000×500. Both solvers found 260 eigenpairs in the interval $[-0.01, 0.01]$ to a tolerance of 1×10^{-8}. Both methods used 4 moments and began with random initial block vector Y. For BEAST, Y contained 100 columns; for z-Pares, 130. Testing performed on the Emmy HPC cluster at RRZE. MUMPS was used for the direct solution of all linear systems. N refers to the number of quadrature nodes along a circular contour, NP to the number of processes.

Working precision level

Given the iterative nature of BEAST, with one iteration being comparatively expensive, the possibility to reduce the cost of at least some of these iterations is attractive. We have observed that before a given residual tolerance is surpassed, systematic errors in the computation of the projector and other operations do not impair convergence speed per se, but impose a limit on what residual can be reached before progress stagnates. One such systematic error is the finite accuracy of floating-point computations, which typically are available in single and double precision. In the light of the aforementioned behavior, it seems natural to perform initial iterations in single precision and thereby save on computation time before a switch to double precision becomes inevitable; cf. Figure 7.

Therefore, mixed precision has been implemented in all BEAST schemes mentioned above, allowing an adaptive strategy to automatically switch from single to double precision after a given residual tolerance is reached. A comprehensive description and results are presented in [2]. These results and our initial investigations also suggest that increased precision beyond double precision (i.e., quad precision) will have no benefit for the convergence rate until a certain double precision specific threshold is reached; convergence beyond this point would require all operations to be carried out with increased precision.



Run / precision	Time total	Time projection
double	420 s	305 s
mixed(7)	333 s	265 s
mixed(9)	316 s	257 s
mixed(11)	301 s	245 s
single	266 s	225 s

Fig. 7: Left: average residual over the BEAST iterations for using double or single precision throughout, and for switching from single to double precision in the 7th, 9th, or 11th iteration, respectively. Right: time (in seconds) to convergence for a size 1 048 576 topological insulator (complex values) with a search space size of 256 and a polynomial degree of 135 on 8 nodes of the Emmy-cluster at RRZE. Convergence is reached after identical numbers of iterations (with the exception of pure single precision, of course). The timings can vary for different ratios of polynomial degree and search space size and depend on the single precision performance of the underlying libraries.

3.2.3 Levels of parallelism

The BEAST framework exploits multiple levels of parallelism using an MPI+X paradigm. We rely on the GHOST and PHIST libraries for efficient sparse matrix/dense vector storage and computation; cf. Sect. 5. The operations implemented therein are themselves hybrid parallel and constitute the lowest level of parallelism in BEAST. Additional levels are addressed by parallelizing over blocks of vectors in Y and, for BEAST-C and BEAST-M, over integration nodes during the application of the approximate projector. A final level is added by exploiting the ability of the method to subdivide the search interval $[\underline{\lambda}, \overline{\lambda}]$ and to process the subintervals independently and in parallel. Making use of these properties, however, may lead to non-orthogonal eigenvectors, which necessitates postprocessing as explained in the following.

3.2.4 A posteriori cross-interval orthogonalization

Rayleigh-Ritz-based subspace iteration algorithms naturally produce a B-orthogonal set of eigenvectors X, i.e., $\mathrm{orth}(X)$ is small, where

$$\mathrm{orth}(X) = \max\left\{\mathrm{orth}(x_i, x_j) | i \neq j\right\} \quad \text{with} \quad \mathrm{orth}(x,y) = \frac{\langle y, x\rangle}{\|x\|\,\|y\|}.$$

By contrast, the orthogonality

$$\mathrm{orth}(X,Y) = \max\left\{\mathrm{orth}(x_i,y_j)\right\}$$

between two or more independently computed sets of eigenvectors may suffer if the distance between the involved eigenvalues is small [50, 51]. Simultaneous re-orthogonalization of evolving approximate eigenvectors during subspace iteration has proven ineffective unless the vectors have advanced reasonably far. A large scale re-orthogonalization of finished eigenvector blocks, on the other hand, requires a careful choice of methodology in order to not diminish the quality of the previously established residual.

Orthogonalization of multiple vector blocks implies Gram-Schmidt style propagation of orthogonality, assuming $\mathrm{orth}(X,Y)$ can be arbitrarily poor. In practice, the independently computed eigenvectors will exhibit multiple grades of orthogonality, but rarely will there be no orthogonality (in the sense above) at all. This, in turn, allows for the use of less strict orthogonalization methods. While, in theory, the orthogonalization of p blocks requires at least $p(p-1)/2+(p-1)$ block-block or intra-block orthogonalizations and ensures global orthogonality, an iterative scheme allows for more educated choices on the ordering of orthogonalizations in order to reduce losses in residual and improve the communication pattern, eliminating the need for broadcasts of vector blocks at the cost of additional orthogonalization operations in the form of multiple sweeps. In practice, very few sweeps (~ 2) are sufficient in most cases.

Every block-block orthogonalization $X = X - Y(Y^H BX)$ disturbs the orthogonality $\mathrm{orth}(X)$ of the modified block, as well as its residual. Local re-orthogonalization of X disturbs the residual further. We have identified orthogonalization patterns and selected orthogonalization algorithms that reduce the loss of residual accuracy to a degree that essentially eliminates the need for additional post-iteration.

The implementation of an all-to-all interaction of many participating vector blocks can be performed in multiple ways with different requirements regarding storage, communication, runtime, and with different implications on accuracy and loss of residual. Among several such strategies and algorithms that have been implemented and tested, the most promising is a purely iterative scheme, both for global and local orthogonalization operations. It is based on a comparison of interval properties, most notably the achieved residual from the subspace iteration. We are continuing to explore the possibility to detect certain orthogonalizations as unnecessary without computing the associated inner products in order to further reduce the workload without sacrificing orthogonality.

3.2.5 Robustness and resilience

In the advent of large scale HPC clusters, hardware faults, both detectable an undetectable, have to be expected.

Detectable hardware faults, e.g., the outage of a component that violently halts execution, can typically only be mitigated by frequent on-the-fly storage of the most vital information. In the case of subspace iteration, as is used in BEAST, almost all required information for being able to resume computation is encoded in the iterated subspace basis in form of the approximate eigenvectors, besides runtime information about the general program flow. Relying on the CRAFT library [76], a per-iteration checkpointing mechanism has been implemented in BEAST.

Additionally, for also being able to react to "silent" computation errors that merely distort the results but do not halt execution, the most expensive operation (application of the approximate projector) has been augmented to monitor the sanity of the results. This can be done in two ways: A checksum-style entrainment of additional vectors, linear combinations of the right-hand sides, can be checked during and after the application of the projector to detect errors and allow for the re-computation of the incorrect parts. The comparison of approximate filter values obtained form the computed basis and the expected values obtained from the scalar representation of the filter function, on the other hand, gives an additional a posteriori test for the overall plausibility of the basis.

Practical tests have shown that small distortions of the subspace basis have not enough impact on the overall process in order to justify expensive measures. If the error is not recurring, just continuing the subspace iteration is often the best and most cost-efficient option. This is particularly true in early iterations, where small errors have no effect at all.

3.3 Further progress on contour integral-based eigensolvers

3.3.1 Relationship among contour integral-based eigensolvers

The complex moment-based eigensolvers such as the Sakurai-Sugiura method can be regarded as projection methods using a subspace constructed by the contour integral

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z \, z^k (zB - A)^{-1} BY.$$

The property of the subspace is well analyzed by using a filter function

$$f(\lambda) := \sum_{j=1}^{d} \frac{\omega_j}{z_j - \lambda},$$

which approximates a band-pass filter for the target region where the wanted eigenvalues are located. Using the filter function, error analyses of the complex moment-based eigensolvers were shown in [30,39,40,67,79]. By using the results of the error analyses, an error resilience technique and an accuracy deterioration technique have also been given in [32, 41].

The relationship between typical complex moment-based eigensolvers was also analyzed in [40] focusing on the subspace. The block SS-RR method [36] and the FEAST algorithm [79] are projection methods for solving the target generalized eigenvalue problem, whereas the block SS-Hankel method [37], Beyn [12], the block SS-Arnoldi methods [38] and its improvements [42] are projection methods for solving an implicitly constructed standard eigenvalue problem; see [40] for details. Figure 8 shows a map of the relationships among the contour integral-based eigensolvers.

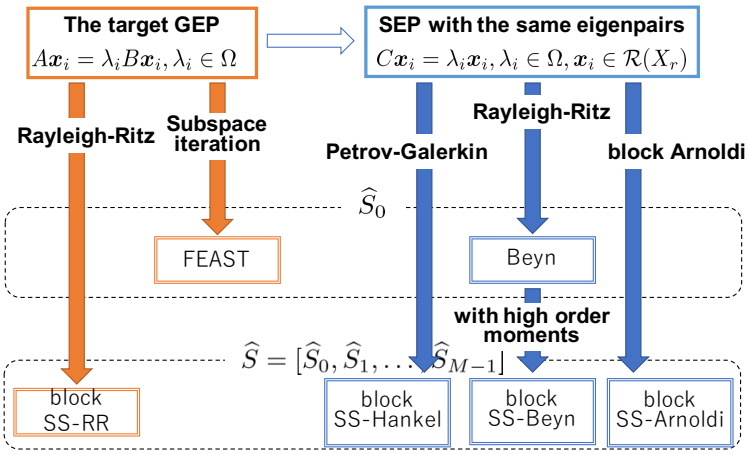


Fig. 8: A map of the relationships among the contour integral-based eigensolvers.

3.3.2 Extension to nonlinear eigenvalue problems

The complex moment-based eigensolvers were extend to nonlinear eigenvalue problems (NEPs):

$$T(\lambda_i)x_i = 0, \quad x_i \in \mathbb{C}^n \setminus \{0\}, \quad \lambda_i \in \Omega \subset \mathbb{C},$$

where the matrix-valued function $T : \Omega \to \mathbb{C}^{n\times n}$ is holomorphic in an open domain Ω. The projection for a nonlinear matrix function $T(\lambda)$ is given by

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z \, z^k T(z)^{-1} Y.$$

This projection is approximated by

$$U_k = \sum_{j=1}^{N} \omega_j z_j^k T(z_j)^{-1} Y, \quad k = 0, 1, \ldots, m-1.$$

The block SS-Hankel [7, 8], block SS-RR [86], and block SS-CAA methods [43] are simple extensions of the GEP solvers. A technique for improving the numerical stability of the block SS-RR method for NEP was developed in [14, 15].

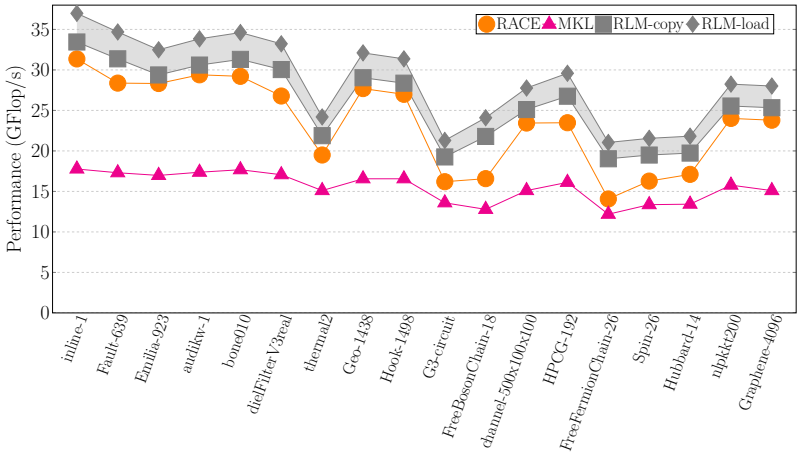
Beyn proposed a method using Keldysh's theorem and the singular value decomposition [12]. Van Barel and Kravanja proposed an improvement of the Beyn method using the canonical polyadic (CP) decomposition [10].

3.4 Recursive Algebraic Coloring Engine (RACE)

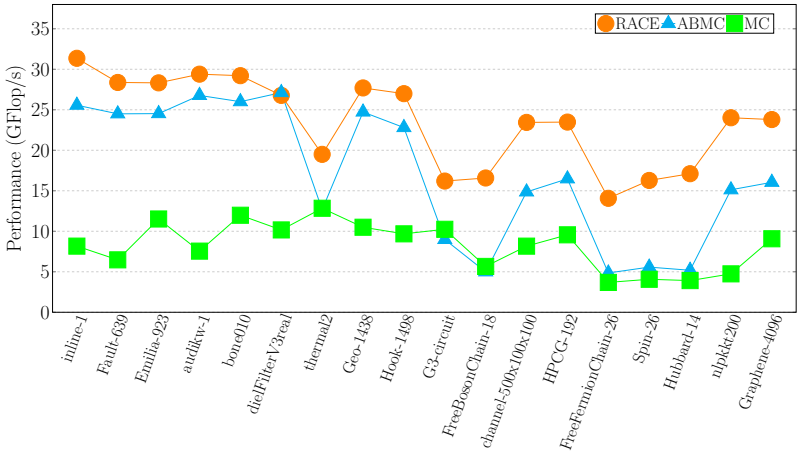
The standard approach to solve the ill-conditioned linear systems arising in BEAST-C or FEAST is to use direct solvers. However, in [26] it was shown that the Kaczmarz iterative solver accelerated by a Conjugate Gradient (CG) method (the so-called CGMN solver [29]) is a robust alternative to direct solvers. Standard multicoloring (MC) was used in [29] for the parallelization of the CGMN kernels. After analyzing the shortcomings of this strategy in view of hardware efficiency, we developed in collaboration with the EXASTEEL-II project the Recursive Algebraic Coloring Engine (RACE) [1]. It is an alternative to the well-known MC and algebraic block multicoloring (ABMC) algorithms [44], which have the problem that their matrix reordering can adversely affect data access locality. RACE aims at improving data locality, reducing synchronization, and generating sufficient parallelism while still retaining simple matrix storage formats such as compressed row storage (CRS). We further identified distance-2 coloring of the underlying graph as an opportunity for parallelization of the symmetric spMVM (SymmSpMV) kernel.

RACE is a sequential, recursive, level-based algorithm that is applicable to general distance-k dependencies. It is currently limited to matrices with symmetric structure (undirected graph), but possibly nonsymmetric entries. The algorithm comprises four steps: level construction, permutation, distance-k coloring, and load balancing. If these steps do not generate sufficient parallelism, recursion on subgraphs can be applied. Using RACE implies a pre-processing and a processing phase. In pre-processing, the user supplies the matrix, the kernel requirements (e.g., distance-1 or distance-2) and hardware settings (number of threads, affinity strategy). The library generates a permutation and stores the recursive coloring information in a level tree. It also creates a pool of pinned threads to be used later. In the processing phase, the user provides a sequential kernel function which the library executes in parallel as a callback using the thread pool.

Figure 9 shows the performance of SymmSpMV on a 24-core Intel Xeon Skylake CPU for a range of sparse symmetric matrices. In Figure 9a we compare RACE against Intel's implementation in the MKL library, and with roofline limits obtained via bandwidth measurements using array copy and read-only kernels, respectively. RACE outperforms MKL by far. In Figure 9b we compare against standard multicoloring (MC) and algebraic block multicoloring (ABMC). The advantage of RACE is especially pronounced with large matrices, where data traffic and locality of access is pivotal. One has to be aware that some algorithms may exhibit a change in



(a) Performance of RACE compared with MKL

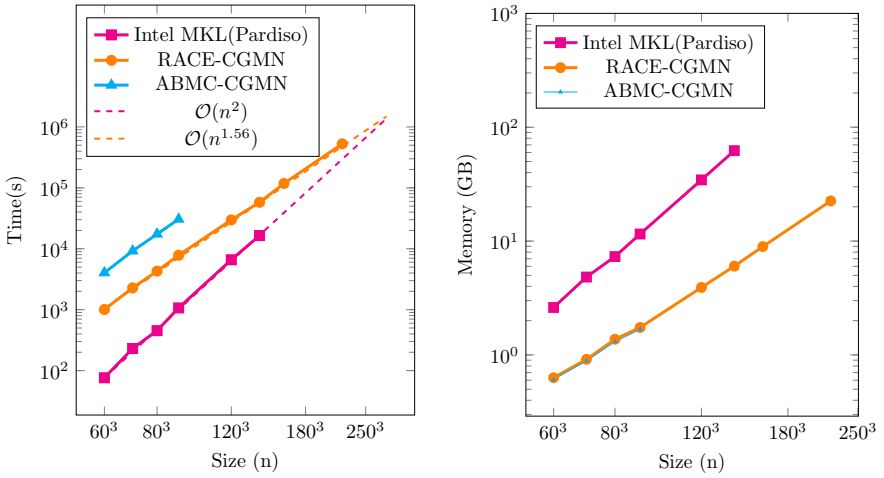


(b) Performance of RACE compared to other coloring approaches

Fig. 9: SymmSpMV performance of RACE compared to other methods. The roofline model for SymmSpMV is shown in Figure 9a for reference. Representative matrices from [17] and ScaMaC 5.5 were used. Note that the matrices are ordered according to increasing number of rows. (One Skylake Platinum 8160 CPU [24 threads])

convergence behavior due to the reordering. This has to be taken into account when benchmarking whole program performance instead of kernels. Details can be found in [1].

In order to show the advantages of RACE in the context of a relevant algorithm, we chose FEAST [65] for computing inner eigenvalues. The hot spot of the algorithm (more than 95%) is a solver for shifted linear systems ($A - \sigma I = b$). These



(a) Time to solution

(b) Memory requirement

Fig. 10: Comparison of FEAST with default MKL direct solver and iterative solver CGMN, parallelized using RACE. (One Skylake Platinum 8160 CPU [24 threads])

systems are, however, highly ill-conditioned, posing severe convergence problems for most linear iterative solvers. We use the FEAST implementation of Intel MKL, which by default employs the PARDISO direct solver [71], but its Reverse Communication Interface (RCI) allows us to plug our CGMN implementation instead. In the following experiment we find ten inner eigenvalues of a simple discrete Laplacian matrix to an accuracy of 10^{-8}. Figure 10 shows the measured time and memory footprint of the default MKL version (using PARDISO) and the CGMN versions parallelized using both RACE and ABMC for different matrix sizes. ABMC is a factor of $4\times$ slower than RACE. The time required by the default MKL with PARDISO is smaller than with CGMN using RACE for small sizes; however, the gap gets smaller as the size grows due to the direct solvers having a higher time complexity (here $\approx \mathcal{O}(n^2)$) compared to iterative methods ($\approx \mathcal{O}(n^{1.56})$). Moreover, the direct solver requires more memory, and the memory requirement grows much faster (see Figure 10b) than with CGMN. In our experiment the direct solver ran out of memory at problem sizes beyond 140^3, while CGMN using RACE used less than 10% of space at this point. Thus, CGMN with RACE can solve much larger problems compared to direct solvers, which is a major advantage in fields like quantum physics.

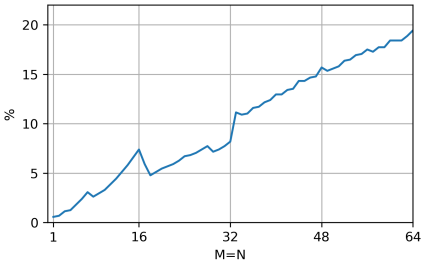


Fig. 11: Percentage of roofline predicted performance achieved by cuBLAS for the range $M = N \in [1,64]$ on a Tesla V100 with 16 GB of memory. (From [20])

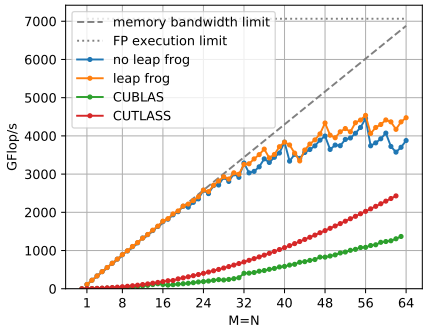


Fig. 12: Best achieved performance for each matrix size with $M = N$ in comparison with the roofline limit, cuBLAS and CUTLASS, with $K = 2^{23}$. (From [20])

4 Hardware Efficiency and Scalability

In this section we showcase performance and parallel efficiency numbers for library components developed in ESSEX-II that are of paramount importance for the application work packages: GPGPU-based tall & skinny matrix-matrix multiplication and the computation of inner eigenvalues using polynomial filter techniques.

4.1 Tall & skinny matrix-matrix multiplication (TSMM) on GPGPUs

Orthogonalization algorithms frequently require the multiplication of matrices that are strongly nonsquare. Vendor-supplied optimized BLAS libraries often yield sub-optimal performance in this case. "Sub-optimal" is a well-defined term here since the multiplication of an $M \times K$ matrix A with an $K \times N$ matrix B with $K \gg M,N$ and small M,N is a memory-bound operation: At $M = N$, its computational intensity is just $M/8$ flop/byte. In ESSEX-I, efficient implementations of TSMM on multicore CPUs were developed [52].

The naive roofline model predicts memory-bound execution for $M \lesssim 64$ on a modern Volta-class GPGPU. See Figure 11 for a comparison of optimal (roofline) performance and measured performance for TSMM on an Nvidia Tesla V100 GPGPU using the cuBLAS library[6]. We have developed an implementation of TSMM for GPGPUs [20], investigating various optimization techniques such as different thread mappings, overlapping long-latency loads with computation via

[6] `https://docs.nvidia.com/cuda/cublas` (May 2019)

leapfrogging[7] and unrolling, options for global reductions, and register tiling. Due to the large and multi-dimensional parameter space, the kernel code is generated using a python script.

Figure 12 shows a comparison between our best implementations obtained via parameter search (labeled "leap frog" and "no leap frog," respectively) with cuBLAS and CUTLASS[8], which is a collection of CUDA C++ template abstractions for high-performance matrix multiplications. Up to $M = N = 36$, our implementation stays within 95% of the bandwidth limit. Although the performance levels off at larger M, N, which is due to insufficient memory parallelism, it is still significantly better than with cuBLAS or CUTLASS.

4.2 BEAST performance and scalability on modern hardware

4.2.1 Node-level performance

Single-device benchmark tests for BEAST-P were performed on an Intel Knights Landing (KNL), an Nvidia Tesla P100, and an Nvidia Tesla V100 accelerator, comparing implementations based on vendor libraries (MKL and cuBLAS/cuSPARSE, respectively) with two versions based on GHOST: one with and one without tailored fused kernels. The GPGPUs showed performance levels expected from a bandwidth-limited code, while on KNL the bottleneck was located in the core (see Figure 13a). Overall, the concept of fused optimized kernels provided speedups of up to $2\times$ compared to baseline versions. Details can be found in [53].

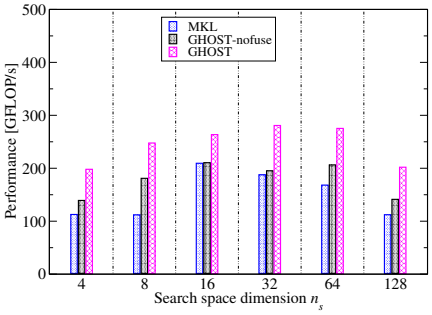
4.2.2 Massively parallel performance

Scaling tests for BEAST-P were performed on the "Oakforest-PACS" (OFP) at the University of Tokyo, "Piz Daint" at CSCS in Lugano, and on the "SuperMUC-NG" (SNG) at Leibniz Supercomputing Centre (LRZ) in Garching.[9] While the OFP nodes comprise Intel "Knights Landing" (KNL) many-core CPUs, SNG has CPU-only dual-socket nodes with Intel Skylake-SP, and Piz Daint is equipped with single-socket Xeon "Haswell" nodes, each of which has an Nvidia Tesla P100 accelerator attached. Weak and strong scaling tests were done with topological insulator (TI) matrices generated by the ScaMaC library. Flops were calculated for the computation of the approximate eigenspace, U, averaged over the four BEAST iterations it took to find the 148 eigenvalues in each interval to a tolerance of 1×10^{-10}. The

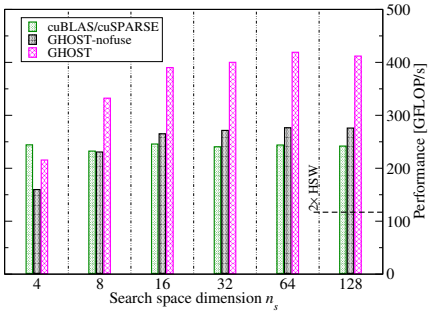
[7] Leapfrogging in this context means that memory loads to operands are initiated one loop iteration before the data is actually needed, allowing for improved overlap between data transfers and computations.

[8] `https://github.com/NVIDIA/cutlass` (May 2019)

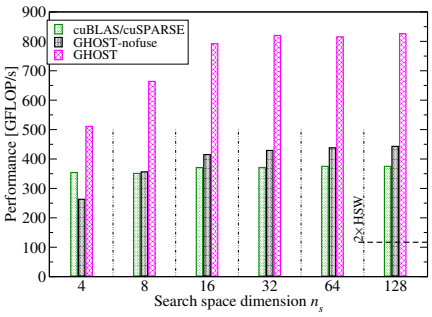
[9] Runs on OFP and SNG were made possible during the "Large-scale HPC Challenge" Project on OFP and the "Friendly-User Phase" of SNG.



(a) KNL



(b) P100



(c) V100

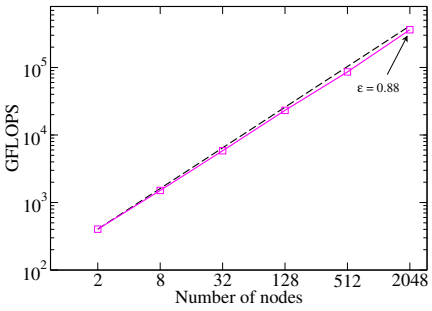
Fig. 13: BEAST-P performance for a topological insulator problem of dimensions $128\times64\times64$ with $n_p = 500$ using different implementations on KNL, P100, and V100. Performance of a dual Xeon E5-2697v3 node (Haswell) is shown for reference. Note the different y axis scaling of the V100 results. (From [53]; for details see therein)

subspace contained 256 columns, and spMMVs were performed in blocks of size 32 for best performance. Optimized coefficients [25] were used for the Chebyshev polynomial approximation, resulting in a lower overall required polynomial degree. Weak and strong scaling results are shown in Figures 14a through 14d.

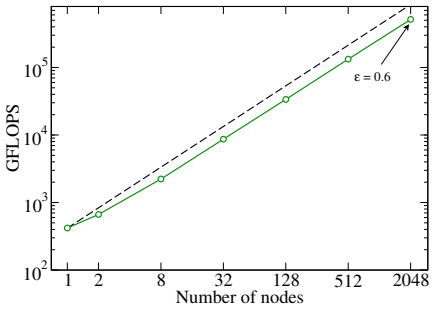
OFP and SNG show similar weak scaling efficiency due to comparable single-node performance and network characteristics. Piz Daint, owing to its superior single-node performance of beyond 400 Gflop/s, achieves only 60% of parallel efficiency at 2048 nodes. A peculiar observation was made on the CPU-only SNG system: Although the code runs fastest with pure OpenMP on a single node (223 Gflop/s), scaled performance was observed to be better with one MPI process per socket. The ideal scaling and efficiency numbers in Figures 14a–14c use the best value on the smallest number of nodes in the set as a reference. The largest matrix on SNG had 6.6×10^9 rows.

5 Scalable and Sustainable Software

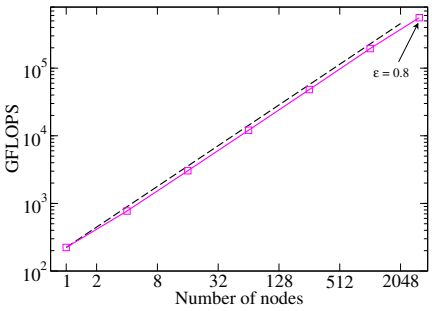
It was a central goal of the ESSEX-II project to consolidate our software efforts and provide a library of solvers for sparse eigenvalue problems on extreme-scale HPC systems. This section gives an overview of the status of our software, most of which



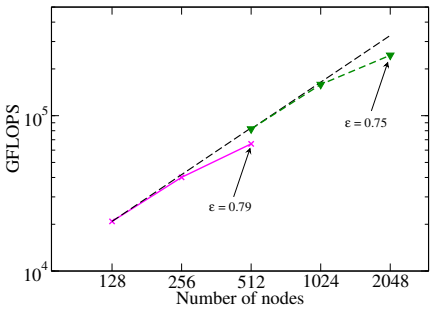
(a) Weak scaling of BEAST-P on OFP for problems of size 2^{20} (2 nodes) to 2^{30} (2048 nodes, about one quarter of the full machine).



(b) Weak scaling of BEAST-P on Piz Daint for problems of size 2^{21} (1 node) to 2^{32} (2048 nodes). (From [53])



(c) Weak scaling of BEAST-P on SNG for problems of size 2^{21} (1 node) to 1.53×2^{32} (3136 nodes, about half of the full machine).



(d) Strong scaling of BEAST-P on SNG for problems of size 2^{28} (crosses) and 2^{30} (triangles).

Fig. 14: Weak scaling of BEAST-P on OFP, Piz Daint, and SNG, and strong scaling on SNG. Dashed lines denote ideal scaling with respect to the smallest number of nodes in the set.

is now publicly available under a three-clause BSD license. Many of the efforts have been integrated in the PHIST library so that they can easily be used together, and we made part of the software available in larger contexts like Spack [27] and the extreme-scale scientific software development kit xSDK [11]. The xSDK is an effort to define common standards for high-performance, scientific software in terms of software engineering and interoperability.

The current status of the software developed in the ESSEX-II project is summarized as follows.

- BEAST is available via bitbucket[10], and can be compiled either using the PHIST kernel interface or the GHOST library directly. The former allows using it with any backend supported by PHIST.

[10] `https://bitbucket.org/essex/beast/`

- CRAFT is available stand-alone[11] or (in a fixed version) as part of PHIST.
- ScaMaC is available stand-alone[12] or (in a fixed version) as part of PHIST.
- GHOST is available via bitbucket[13]. The functionality which is required to provide the PHIST interface can be tested via PHIST. Achieving full (or even substantial) test coverage of the GHOST-functionality would require a very large number of tests (in addition to what the PHIST interface provides, GHOST allows mixing data types, and it uses automatic code generation, which leads to an exponentially growing number of possible code paths with every new kernel, supported processor and data type). It is, however, possible to create a basic GHOST installation via the Spack package manager (since March 2018, commit `bcde376`).
- PHIST is available via bitbucket[14] and Spack (since commit `2e4378b`). Furthermore, PHIST 1.7.5 is part of xSDK 0.4.0. The version distributed with the xSDK is restricted to use the Tpetra kernels to maximize the interoperability of the package.

5.1 PHIST and the Block-ILU

In ESSEX-I we addressed mostly node-level performance [80] on multi-core CPUs. The main publication of ESSEX-II concerning the PHIST library [81] presents performance results for the block Jacobi-Davidson QR (BJDQR) solver on various platforms, including recent CPUs, many-core processors and GPUs. It was also shown in this work that the block variant has a clear performance advantage over the single-vector algorithm in the strong scaling limit. The reason is that, while the number of matrix-vector multiplications increases with the block size (see also [66]), the total number of reductions decreases. In order to demonstrate the performance portability of PHIST, we show in figure 15 a weak scaling experiment on the recent SuperMUC-NG machine.

For the block size 4, we roughly match the performance it achieves in the memory-bounded HPCCG benchmark (207 TFlop/s),[15] but using only half of the machine. This gives a clear indication that our node-level performance engineering and multi-node implementation are highly successful: after all, we do not optimize for the specific operator application (a simple structured grid, 3D Laplace operator), which the HPCCG code does. On the other hand, we have an increased computational intensity for some of the operations due to the blocking, which increases the performance over a single-vector CG solver. The single-vector BJDQR solver achieves 98 TFlop/s on half of the machine.

[11] `https://bitbucket.org/essex/craft/`

[12] `https://bitbucket.org/essex/matrixcollection/`

[13] `https://bitbucket.org/essex/ghost/`

[14] `https://bitbucket.org/essex/phist`

[15] see `https://www.top500.org/system/179566`

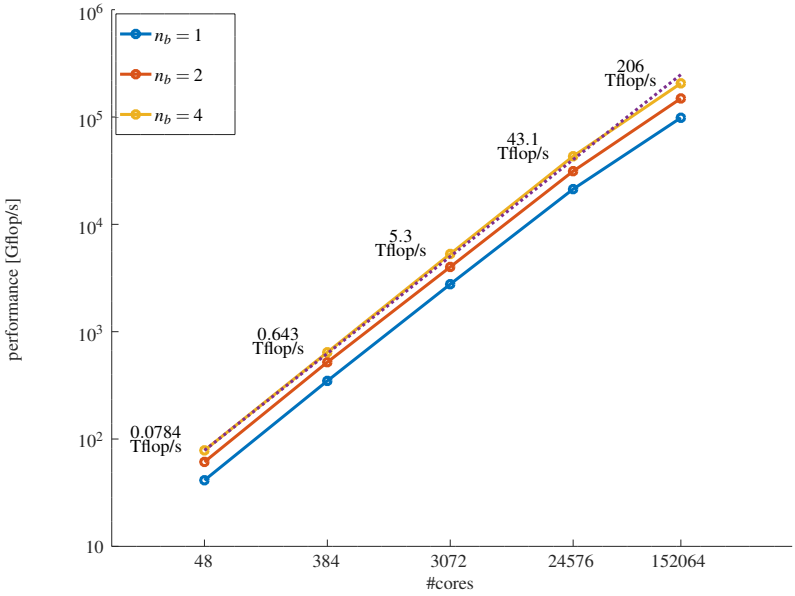


Fig. 15: Weak scaling behavior of the PHIST BJDQR solver for a symmetric PDE benchmark problem and different block sizes.

5.1.1 Integration of the Block-ILU preconditioning technique

Initial steps have been taken to make the Block-ILU preconditioner (cf. Section 3.1) available via the PHIST preconditioning interface. At the time of writing, there is an experimental implementation of a block CRS sparse matrix format in the PHIST builtin kernel library, including parallel conversion and matrix-vector product routines and the possibility to construct and apply the block Cholesky preconditioner. Furthermore, the interfaces necessary to allow using the preconditioner within the BJDQR eigensolver have been implemented. These features are available for experimenting in a branch of the PHIST git repository because they do not yet meet the high demands on maintainability (especially unit testing) and documentation of a publicly available library. Integration of the method with the BEAST eigensolver is not yet possible because the builtin kernel library does not support complex arithmetic. As mentioned in Section 3.2, the complex version will be integrated directly into the BEAST software, instead.

5.2 BEAST

BEAST combines implementations of spectral filtering methods for Rayleigh-Ritz type subspace iteration in a generalized framework to provide facilities for improving performance and robustness. The algorithmic foundation allows for the solution of interior Hermitian definite eigenproblems of standard and generalized form via an iterative eigensolver, unveiling all eigenpairs in one or many specified intervals. The software is designed as hybrid parallel library, written in C/C++, and relying on GHOST and PHIST to provide basic operations, parallelism, and data types. Beyond the excellent scalability of the underlying kernel libraries, multiple additional levels of parallelism allow for computing larger portions of the spectrum and/or utilizing a larger number of computing cores. The inherent ability of the underlying algorithm to compute separate intervals independently offers wide potential but requires careful handling of cross-interval interactions to ensure the desired quality of results, which is well supported by BEAST.

The BEAST library interface comes in variations for the common floating point formats (real and complex, single and double precision) for standard and generalized eigenproblems. Additionally, the software offers the possibility to switch precisions on-the-fly, from single to double precision, in order to further improve performance. While BEAST offers an algorithm for standard eigenproblems that completely bypasses the need for linear system solves, other setups typically require a suitable linear solver. Besides a builtin parallel sparse direct solver for banded systems, BEAST includes interfaces to MUMPS and Strumpack, as well as a flexible callback-driven interface for the inclusion of arbitrary linear solvers. It also interfaces with CRAFT and ScaMaC, which provide fault tolerance and dynamic matrix generation, respectively. While working out of the box for many problems, BEAST offers a vast amount of options to tweak the software for the specific problem at hand. A builtin command line parser allows for easy modification. The included application bundles the several capabilities of BEAST in form of a stand-alone tool that reads or generates matrices and solves the specified eigenproblem. As such, it acts as comprehensive example for the usage of BEAST.

The library is still in a development state, and interface and option sets may change. A more comprehensive overview over a selection of features is provided in Section 3.2.

5.3 CRAFT

The CRAFT library [75] covers two essential aspects of fault tolerance namely communication, and data recovery of an MPI application in case of process-failures.

In the Checkpoint/Restart part of the library, it provides an easier and extensible interface for making application-level checkpoint/restart. A CRAFT-checkpoints can be defined simply by defining a `Checkpoint` object and adding the restart-relevant data in it, as shown in Listing 1. By default, the `Checkpoint::add()`

```
#include <mpi.h>
#include <craft.h>
int main(int argc, char* argv[]){
  ...
  size_t n=5, myrank, iteration=1, cpFreq=10;
  double dbl = 0.0;
  int * dataArr    = new int[n];
  MPI_Comm FT_Comm;
  MPI_Comm_dup (MPI_COMM_WORLD, &FT_Comm);
  AFT_BEGIN (FT_Comm, &myrank, argv);
  ********************************************
  Checkpoint myCP ("myCP", FT_Comm);          * //define checkpoint
  myCP.add ("dbl", &dbl);                     *
  myCP.add ("iteration", &iteration);         *
  myCP.add ("dataArr", dataArr, &n);          AFT Zone
  myCP.commit();                              *
  myCP.restartIfNeeded (&iteration);          *
  for(; iteration <= 100 ; iteration++){      *
    Computation_communication();              *
    modifyData(&dbl, dataArr);                *
    myCP.updateAndWrite (iteration, cpFreq);  *
  }                                           *
  ...                                         *
  ********************************************
  AFT_END();
}
```

Listing 1: A toy-code that demonstrates the simplicity of CRAFT's checkpoint/restart and automatic fault tolerace features in a typical iterative-style scientific application.

function supports the most frequently used data formats, e.g., "plain old data" (POD), i.e., `int`, `double`, `float`, etc., POD 1D- and 2D-arrays, MPI data-types, etc.. However, it can be easily extended to support any user defined data-types. The `Checkpoint::read()`, `write()` and `update()` methods can then be used to read-/write all added checkpoint's data. The library supports asynchronous-checkpointing as well as node-level checkpointing using the SCR library [45]. Moreover it supports multi-staged, nested-, and signal-checkpointing.

The Automatic Fault Tolerance (AFT) part of CRAFT provides an easier interface for a dynamic process-failure recovery and management. CRAFT uses the ULFM-MPI implementation for process-failure detection, propagation, and communication recovery procedures, however it considerably reduces the user's effort by hiding these details behind `AFT_BEGIN()` and `AFT_END()` functions as shown in Listing 1. After a process failure, the library recovers the broken communicator (shrinking or non-shrinking by process-spawning), and returns the control back to the program at `AFT_BEGIN()`, where the data can be recovered. Both of these CRAFT functionalities are designed to complement each other, however they can be used independently as well. For detailed explanation of the features included in CRAFT, check [75]. Moreover, the library is available at [74].

Lanczos parameters

Matrix	Graphene-3000-3000	num. rows & cols.	$9.0 \cdot 10^8$
number of non-zeros	$11.7 \cdot 10^9$	**global checkpoint size**	≈ 14.4 GB
num. of iterations	3000	Checkpoint frequency	500

Jacobi-Davidson parameters (using Phist)

Matrix	spinSZ30	num. of rows & columns	$1.6 \cdot 10^8$
Number of non-zeros	$2.6 \cdot 10^9$	num. of sought eigenvalues	20
num. of sought eigenvalues	20	num. of checkpoints	10
global checkpoint size	≈ 32 GB	Backend support library	Ghost

Beast parameters

Matrix	tgraphene: 12000,12000,0	num. rows & cols.	$1.44 \cdot 10^8$
Beast iterations	9	checkpoint frequency	2
global checkpoint size	≈ 65 GB	Backend support library	Ghost

Table 1: The parameter values for Lanczos, JD, and Beast benchmarks.

5.4 CRAFT Benchmark Application

Within the scope of ESSEX, we have integrated CRAFT in the GHOST and PHIST libraries, and the BEAST algorithm.

Figure 16 shows a benchmark comparing the overhead of three different checkpointing strategies for the Lanczos algorithm (GHOST-based eigensolver) , Jacobi-Davidson (PHIST-based eigensolver), and the BEAST algorithm. The important parameters for these benchmarks are listed in Table 1. The benchmark shows that the node-level and asynchronous checkpointing significantly reduces the checkpoint overhead despite a very high checkpoint frequency.

The benchmark presented in Fig. 17 demonstrates the overhead caused by checkpoint/restart as well as by the communication recovery after process failures for the Lanczos application. The first two bars, namely 'No CP Intel MPI' and 'No CP ULFM-MPI' show the runtime between non-fault-tolerant (Intel-MPI) vs. a fault-tolerant MPI implementation (ULFM-MPI), and creates a baseline for ULFM-MPI implementation without any failures. The next two group of bars show the application runtime with 0-,1-, and 2-failures with checkpoints taken on PFS- and node-level. The failures are triggered at the mid-point of two successive checkpoints from within the application to have a deterministic re-computation time, where each failure simulates a complete node-crash (2 simultaneous process failures) and recovery is performed in a non-shrinking fashion on spare nodes. The largest contribution to the overhead is caused by the re-computation part, whereas the communication repair overhead takes an average of $\approx$ 2.6 sec. only.

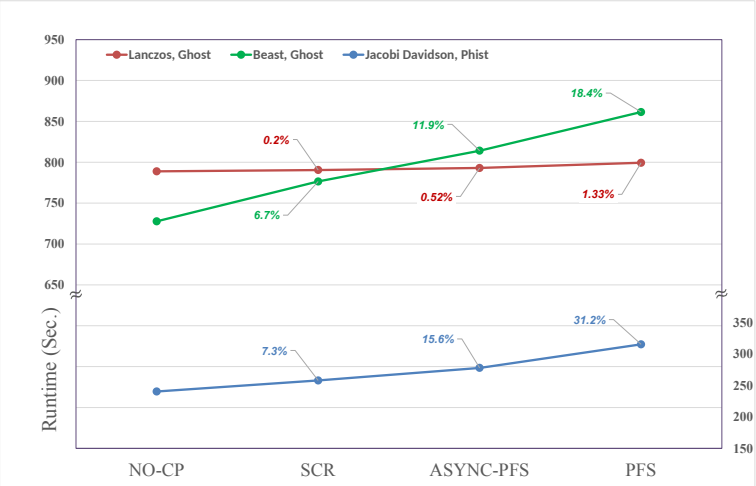


Fig. 16: CRAFT checkpointing overhead comparison for the Lanczos, Jacobi-Davidson, and BEAST eigenvalue solvers using three checkpointing methods of CRAFT, namely, node-level checkpointing with SCR, asynchronous PFS, and synchronous PFS checkpoints. The overhead for each checkpoint case is shown as a percentage. (number of nodes=128, number of processes=256, Intel MPI).

Besides ESSEX, CRAFT has been utilized in [22] to create a process-level fault tolerant FEM code based on the shrinking recovery style. Moreover, CRAFT has been recently integrated in the EXASTEEL [21] project.

5.5 ScaMaC

Sparse matrices are central objects in the ESSEX project because of its focus on large-scale numerical linear algebra problems. A sparse matrix, whether derived from the Hamiltonian of a quantum mechanical system, from the Laplacian in a partial differential equation, or simply given as an abstract entity with unknown properties, defines a problem to be solved. The solution may then consist of a set of eigenvalues and eigenvectors computed with the BEAST or Jacobi-Davidson algorithms or, more moderately, of an estimate of some matrix norm or the spectral radius.

Testing and benchmarking of linear algebra algorithms, but also of computational kernels such as spMVM, requires matrices of different type and different size. Standard collections such as the Matrix Market [59] or Florida Sparse Matrix Collection [17] cover a wide range of examples, but mainly provide matrices of fixed moderate size. As algorithms and implementations improve, such matrices become readily too small and limited to serve as realistic test and benchmark cases.

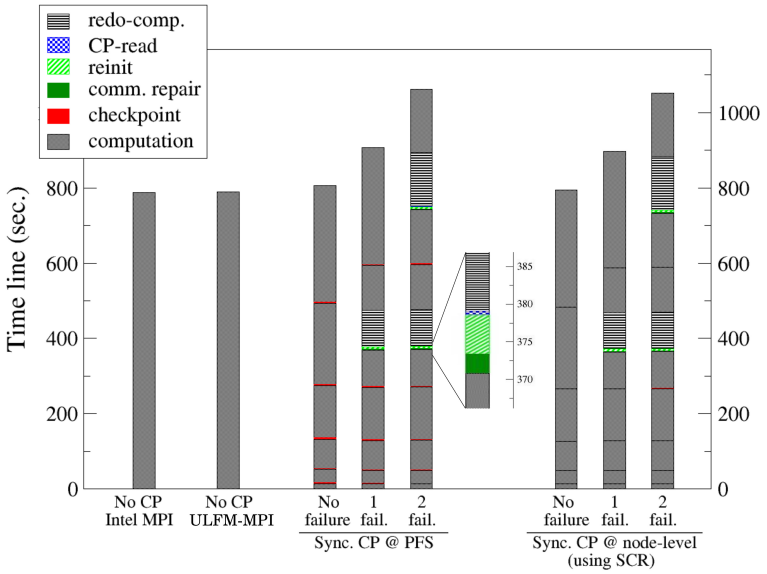


Fig. 17: Lanczos application with various checkpoint/restart and process failure recovery scenarios using 128 nodes (256 processes) on the RRZE Emmy cluster. On average the communication recovery time is 2.6 seconds (ULFM-MPI v1.1).

We therefore decided in the ESSEX project to establish a collection of *scalable* matrices — the ScaMaC. Every matrix in ScaMaC is parameterized by individual parameters that allow the user to scale up the matrix dimension and to modify other, for example spectral, properties of the matrix. ScaMaC includes simple test and benchmark matrices but also 'real-world' matrices from research studies and applications. A major goal of ScaMaC is to provide a flexible yet generic interface for matrix generation, together with the necessary infrastructure to allow for immediate access to the collection irrespective of the concrete usage case.

The ScaMaC approach to matrix generation is straightforward and simple: Matrices are generated row-by-row (or column-by-column). The entire complexity of the actual generation technique, which depends on the specific matrix example, is encapsulated in a `ScamacGenerator` type and hidden from the user. ScaMaC provides routines to create and destroy such a matrix generator, to query matrix parameters prior to the actual matrix generation, and to obtain each row of the matrix. The ScaMaC interface is entirely generic and identical for all matrices in the collection.

A minimal code example is given in Figure 18. In this example, the matrix and its parameters are set by parsing an argument string of the form `"MatrixName, parameter=...,..."` in line 3, before all rows are generated in the loop in lines 12–17. As this examples shows, parallelization of matrix generation is not part of the ScaMaC, but lies within the responsibility of the calling program. All ScaMaC routines are thread-safe and can be embedded directly into MPI processes and OpenMP

```
// step 1: obtain a generator - per process
ScamacGenerator * my_gen;
err = scamac_parse_argstr("Hubbard,n_sites=20", &my_gen, &errstr);
err = scamac_generator_finalize(my_gen);
    ................
// step 2: allocate workspace - per thread
ScamacWorkspace * my_ws;
err = scamac_workspace_alloc(my_gen, &my_ws);
    ................
// step 3: generate the matrix row by row
ScamacIdx nrow = scamac_generator_query_nrow(my_gen);
for (idx=0; idx<nrow; idx++) { // parallelize loop with OpenMP, MPI, ...
    // obtain the column indices and values of one row
    err = scamac_generate_row(my_gen, my_ws, idx, SCAMAC_DEFAULT, &nz, cind, val);
    // store or process the row
        ................
}
// step 4: clean up
err = scamac_workspace_free(my_ws);     // in each thread
err = scamac_generator_destroy(my_gen); // in each process
// step 5: use matrix
    ................
```

Fig. 18: Code example for row-by-row matrix generation with the generic ScaMaC generators.

threads. This approach guarantees full flexibility for the user and is easily integrated into existing parallel matrix frameworks such as PETSc or Trilinos. Both BEAST and PHIST provide direct access to the ScaMaC, therefore freeing the user from any additional considerations when using ESSEX software.

ScaMaC is written in plain C. Auto-generated code is included already in the release, such that requirements at compile time are minimal. Interoperability with other programming languages is straightforward, e.g., by using the ISO C bindings of the FORTRAN 2003 standard. Runtime requirements are equally minimal. Matrix generation has negligible memory overhead, requiring only a few KiB workspace to store lookup tables and similar information.

The key feature of ScaMaC is scalability, since the matrix rows (or columns) can be generated independently and in arbitrary order. For example at Oakforest-PACS (see Sec. 4.2.2), a `Hubbard` matrix (see below) with dimension $\geq 9 \times 10^9$ and $\geq 1.5 \times 10^{11}$ non-zeros is generated in less than a minute, using 2^{10} MPI processes each of which generates an average of 1.5×10^5 rows per second. As explained, the task of efficiently storing or using the matrix is left to the calling program.

ScaMaC is accompanied by a small toolkit for exploration of the collection. The toolkit addresses some basic tasks such as querying matrix information or plotting the sparsity pattern, but is not intended to compete with production-level code or full-fledged solver libraries, as the ESSEX project provides with the BEAST, GHOST, and PHIST libraries.

At the moment[16], the matrix generators included in ScaMaC strongly reflect our personal research interests in quantum physics, but the ScaMaC framework is entirely flexible and allows for easy inclusion of new types of matrices, provided that they can be generated in a scalable way. The next update (scheduled for spring 2020) will extend ScaMaC primarily with matrix generators for standard partial differential equations, including stencil matrices and finite element discretizations for advection-diffusion and elasticity problems, wave propagation, and the Schrödinger equation. Additional examples that are well suited for scalable generation are regular and irregular graph Laplacians, which have gained renewed interest in the context of machine learning [16, 60].

To obtain an idea of the 'real-world' application matrices already contained in ScaMaC, consider two examples: The celebrated Hubbard model of condensed matter physics (`Hubbard`) [34] and a theoretical model for excitons in the cuprous oxide from our own research in this field (`Exciton`) [4]. These matrices appear as Hamiltonians in the Schrödinger equation, and thus are either symmetric real (`Hubbard`) or Hermitian complex (`Exciton`). The respective application requires a moderate number (typically, $10-1000$) of extremal or interior eigenpairs, which is less than 0.1% of the spectrum. Other ScaMaC generators provide general (non-symmetric or non-Hermitian) matrices, with a variety of sparsity patterns, spectral properties, etc. All generators depend on a number of application-specific parameters[17], which are partly listed in Table 2 for the `Hubbard` and `Exciton` generator.

For the `Hubbard` example, two parameters determine the matrix dimension and sparsity pattern: n_fermions gives the number electrons with a spin-up or spin-down orientation, n_sites the number of orbitals occupied by the electrons. In terms of these parameters, the matrix dimension is $D = \binom{\text{n\_sites}}{\text{n\_fermions}}^2$. This dependency results in the rapid growth of D shown in Table 3. In the physically very interesting case of half-filling (n_fermions = n_sites/2 = n) we have asymptotically $D \simeq 2^n/\sqrt{(\pi/2)n}$, that is, exponential growth of D.

The `Exciton` example has the more moderate dependence $D = 3(2\mathrm{L}+1)^3$ (see Table 3). Here, the parameter L is a geometric cutoff that limits the maximal distance between the electron and hole that constitute the exciton. This example has a number of other parameters that are adapted literally from [4]. These parameters enter into the matrix entries, and thus affect the matrix spectrum and, finally, the algorithmic hardness of computing the eigenvalues of interest that determine the physical properties of the exciton.

Both `Hubbard` and `Exciton` are examples of difficult matrices, albeit for different reasons. For `Hubbard`, one unresolved challenge is to compute multiple interior eigenvalues for large n_fermions, n_sites, which becomes extremely difficult because of the rapid growth of the matrix dimension (specialized techniques for the Hubbard model such as the density-matrix renormalization group [72] cannot compute interior eigenvalues). Due to the irregular sparsity pattern of the `Hubbard` ma-

[16] In version 0.8.2, ScaMaC contains 15 different matrix generators with a total of 95 parameters.

[17] For a full list of generators and parameters, consult the ScaMaC documentation included with the code, or at `https://alvbit.bitbucket.io/scamac_docs/_matrices_page.html`.

Table 2: Parameters of the `Hubbard` and `Exciton` matrix generator in the ScaMaC.

`Hubbard`		
matrix type: symmetric real		
int	n_sites	number of sites
int	n_fermions	number of fermions
double	t	hopping strength
double	U	Hubbard interaction
	⋮	
double	ranpot	random potential
rngseed	seed	random seed

`Exciton`		
matrix type: Hermitian complex		
int	L	cube length
double	so	spin orbit
double	ex	exchange
double	mlh	mass light hole
double	mhh	mass heavy hole
double	me	mass electron
double	eps	dielectric constant
	⋮	

trices (see Figure 19 below), already the communication overhead of spMVM poses a serious obstacle to scalability and parallel efficiency. For `Exciton`, which are essentially stencil-like matrices of moderate size, the challenge is to compute some hundred eigenvalues out of a strongly clustered spectrum. Here, it is the poor convergence of iterative eigenvalue solvers for nearly degenerate eigenvalues that renders this problem hard. Thanks to the algorithmic advances in the ESSEX project, we now have reached a position that allows for future progress on these problems.

ScaMaC comes with several convenient features. For example, the `Hubbard` matrix includes the parameter `ranpot` to switch on a random potential. Random numbers in ScaMaC are entirely reproducible, and independent of the number of threads or processes that call the ScaMaC routines, or of the order in which the matrix rows are generated. An identical random seed gives the same matrix under all circumstances. In particular, individual matrix rows can be reconstructed at any time, which simplifies a fault-tolerant program design (see Section 5.3). Another feature is the possibility to effortlessly generate the (conjugate) transpose of non-symmetric (non-Hermitian) matrices, which is considerably easier than constructing the transpose of a (distributed) sparse matrix after generation.

Table 3: Matrix dimension D for the `Hubbard` and `Exciton` example, as a function of the respective parameter n_sites (and default value n_fermions $= 5$) or L.

`Hubbard`	
n_sites	D
10	63 504
15	9 018 009
20	240 374 016
25	2 822 796 900
30	20 307 960 036
40	432 974 528 064

`Exciton`	
L	D
10	27 783
20	206 763
50	3 090 903
100	24 361 803
150	81 812 703
200	193 443 603

6 Application Results

6.1 Eigensolvers in quantum physics: Graphene, topological insulators, and beyond

Because of the linearity of the Schrödinger equation, quantum physics is a paradigm for numerical linear algebra applications. Historically, some application cases, such as the computation of the ground state (i.e. of the eigenvector to the minimal eigenvalue), have received so much attention that only gradual progress remains possible nowadays. In the ESSEX project we instead address two major cases where novel algorithmic improvements and systematic utilization of large-scale computing resources through state-of-the-art implementations still result in substantial qualitative progress. These two cases are the computation of (i) extreme eigenvalues with high degeneracy, which is addressed with a block Jacobi-Davidson algorithm, (ii) multiple interior eigenvalues, which is addressed by various filter diagonalization techniques. Application case (i) has been documented in [66], including the example of spin chain matrices (`SpinChainXXZ` in the ScaMaC). For application case (ii) the primary quantum physics example are graphene [13] and topological insulators [31] (`Graphene` and `TopIns` in the ScaMaC). For these examples, eigenvalues towards the center of the spectrum, near the Fermi energy of the material, are those of interest. This situation is similar to applications in quantum chemistry and density functional theory, but in our case the matrices represent a full (disordered or structured) two or three-dimensional domain, and are usually larger than those considered elsewhere [58].

Starting with the paper [63] on Chebyshev filter diagonalization (ChebFD) and culminating in the BEAST software package (see Section 3.2), the computation of interior eigenvalues of large-to-huge graphene and topological insulator matrices has been successfully demonstrated with ESSEX algorithms, using polynomial filters derived from Chebyshev polynomials. Already with the simple ChebFD algorithm we could compute $N_T \simeq 100$ eigenvectors from the center of the spectrum of a matrix with dimension $D \simeq 10^9$ (i.e. an effective problem size $N_T \times D \simeq 10^{11}$), in order to understand the electronic properties of a structured topological insulator (see Figure 13 in [63]). With improved filter coefficients and a more sophisticated implementation, the polynomial filters in the (P-) BEAST package deal with such problems at reduced computational cost (see Section 3.2). Such large-scale computations heavily rely on the optimized spMMVM and TSMM kernels of the GHOST library (see Section 5).

To appreciate the numerical progress reflected in these numbers one should note the different scaling of the numerical effort N_{MVM} (measured in terms of the dominant operation of spMVM) for the computation of extreme and interior eigenvalues (cf. the discussion in [63]). In an idealized situation with equidistant eigenvalues, we have roughly $N_{\mathrm{MVM}} \sim D^{1/2}$ for extreme but $N_{\mathrm{MVM}} \sim D$ for interior eigenvalues. For the $D \simeq 10^9$ example, we have to compensate for a factor 10^4–10^5 to enable computation of interior instead of extreme eigenvalues.

Algorithm and software development in ESSEX has been to a large degree application-driven. Now, at the end of the ESSEX project, where the algorithms for our main application cases have become available, we follow two ways to go beyond the initial quantum physics applications. First, entirely new applications can now be addressed with ESSEX software, extending our efforts to non-linear and non-Hermitian problems (see Section 6.2). Second, relevant applications such as the `Hubbard` and `Exciton` examples (see Section 5.5) still fit into the two major application areas already addressed in ESSEX, but further increase the computational complexity. For `Exciton`, the strongly clustered spectrum with many nearly-degenerate eigenvalues leads to a numerical effort $N_{\mathrm{MVM}} \gg D^{1/2}$ already for extreme eigenvalues. For `Hubbard`, the huge matrix dimension D is a serious obstacle for the computation of interior eigenvalues.

The `Hubbard` matrices also hint at an application-specific issue of general interest that we encountered but could not solve within ESSEX. Specifically, it is the complicated sparsity pattern of many of our quantum physics matrices (see Figure 19) that adversely affects the parallel efficiency of distributed spMVM, and thus of our entire software solutions. Node-level performance engineering is here easily overcompensated by communication overhead. Unfortunately, the communication overhead is not reduced by standard matrix reordering strategies [62, 73, 87]. This problem can be partially alleviated by overlapping communication with computation, as in the spMMVM (see Section 2), but a full solution to restore parallel efficiency is not yet available. Clearly, our different application scenarios still provide enough incentive to think about future numerical, algorithmic, and computational developments beyond the ESSEX project.

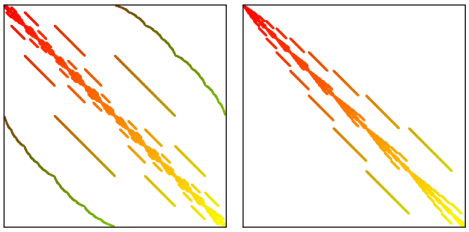


Fig. 19 Sparsity pattern of the Hubbard (`Hubbard,n_sites=40,n_fermions=20`) and spin chain (`SpinChainXXZ, n_sites=32,n_up=8`) example.

6.2 New applications in nonlinear dynamical systems

The block Jacobi-Davidson QR eigensolver in PHIST is capable of solving non-symmetric and generalized eigenvalue problems of the form

$$\mathbf{A}\mathrm{x} = \lambda \mathbf{B}\mathrm{x}, \tag{1}$$

where **B** should be symmetric and positive definite. In [78], we exploited several unique features of this implementation to study the linear stability of a three-dimensional reaction-diffusion equation: the Jacobian is non-symmetric, the preconditioner was implemented in Epetra (which can be used directly as a backend for PHIST), and the high degree of symmetry in the model yields eigenvalues with high geometric multiplicity (up to 24). We therefore use a relatively large block size of 8 for these computations to achieve convergence to the desired 20-50 eigenpairs $(\lambda_i, \mathrm{x}_i)$, with the real part of λ_i near 0. In a recent Ph.D. thesis [77], the solver was also used for studying the linear stability of incompressible flow problems. Here **B** is in fact only semi-definite, and the preconditioner has to make sure that the solution stays in the 'divergence-free space', in which the velocity field satisfies $\nabla \cdot \mathrm{u} = 0$ and **B** induces a norm.

Another ongoing effort concerning dynamical systems is the use of PHIST to parallelize the dynamical systems analysis tool PyNCT, which has as its main application the study of superconductors [85]. We have taken first steps to use PHIST as backend for the Python-based algorithms in PyNCT. Furthermore, it is possible to solve the eigenvalue problems arising in PyNCT directly by the BJDQR method in PHIST. Our goal here is the scalable parallel and fully automatic computation of bifurcation diagrams using PyNCT and any backend supported by PHIST.

The Statistical Learning Lab led by Dr. Marina Meila at the University of Washington started to use the PHIST eigensolver to compute spectral gaps for Laplacian matrices obtained from conformation trajectories in molecular dynamics simulations, and other scientific data [16, 60]. These are symmetric positive definite matrices whose dimensions equal the number of simulation steps, typically of the order of $n = 10^6$. When the data intrinsic dimension d is fixed, and much smaller than n (in our examples $d < 10$), the Laplacian is a sparse matrix. The sparsity pattern is not regular, and it is data dependent, as it reflects the neighborhood relationships in the data. Hence, in densely sampled regions rows will have many more non-zeros than in the sparsely populated regions of the data. In a manifold embedding algorithm, the eigengaps identify the optimal number of coordinates in which to embed the data. Furthermore, for data sizes $n \gg 10^6$, PHIST is used to compute the diffusion map embedding itself for the higher frequency coordinates for which existing methods are prohibitively slow.

7 International Collaborations

The internationalisation effort in the second phase of SPPEXA has fostered the ESSEX-II activities in several directions. First and foremost it amplified the scientific expertise in the project. Soon it became clear that complementing knowledge and developments could be leveraged across the partners. A specific benefit of the collaboration between German and Japanese partners is their very different background in terms of HPC infrastructures.Through close personal collaboration within the project all partners could easily access and use latest supercomputers on

either side (see 4.2 and note that the BEAST framework has also been ported to the K-computer). Together with the joint collaboration on scientific problems and software development a steady exchange evolved with many personal research visits which also opened up collaboration with partners not involved directly in ESSEX-II.

The results described in Section 3.1 on preconditioners are a direct result of the collaboration of ESSEX-II with ppOpen-HPC[18] project led by Univ. of Tokyo. On the other hand, the CRAFT library developed at Univ. of Erlangen is utilized in an FEM code of Univ. of Tokyo and is part of a follow on JHPCN project with the German partner involved as associated partner.

Collaboration between Japanese and German working groups made possible the expansion of the BEAST framework for projection based eigensolvers to include Sakurai–Sugiura methods. Various numerical and theoretical issues associated with the implementation of the solver within an iterative framework were resolved, and new ideas explored during research visits. Results based on this collaboration have so far been presented in multiple conferences and a paper in preparation [35].

The linear systems arising from numerical quadrature in the BEAST-C and BEAST-M framework were used in the testing and development of an Block Cholesky-based ILU preconditioner. The integration of an interface to this solver into BEAST has begun. Examining strategies and expectations for solving these extreme ill-conditioned problems was a point of intense discussion and collaboration between working groups. One results was the development of RACE (see 3.4). Beyond the discussion between several Japanese and German ESSEX-II partners also a strong collaboration with the Swiss partner (O. Schenk) of EXASTEEL-II evolved, who is an expert on direct solvers and graph partitioning. In this context also a collaboration with T. Iwashita (Hokkaido Univ., Japan) started in terms of hardware efficient coloring.

Throughout the project, the variety of large matrices continuously added to the ScaMaC library allowed for testing with a variety of realistically challenging problems of both real and complex types in all ESSEX-II working groups.

Acknowledgements The project is funded by the German DFG priority programme 1648 (Software for Exascale Computing) under the ESSEX-I/II (Equipping Sparse Solvers for Exascale) projects. We are grateful for computer time granted on the LRZ SuperMUC and SuperMUC-NG, the CSCS Piz Daint, and the OakForest PACS systems.

References

1. Alappat, C.L., Hager, G., Schenk, O., Thies, J., Basermann, A., Bishop, A.R., Fehske, H., Wellein, G.: A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication. CoRR **abs/1907.06487** (2019). URL `http://arxiv.org/abs/1907.06487`. Submitted.
2. Alvermann, A., Basermann, A., Bungartz, H.J., Carbogno, C., Ernst, D., Fehske, H., Futamura, Y., Galgon, M., Hager, G., Huber, S., Huckle, T., Ida, A., Imakura, A., Kawai, M., Köcher, S.,

[18] `http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/`

Kreutzer, M., Kus, P., Lang, B., Lederer, H., Manin, V., Marek, A., Nakajima, K., Nemec, L., Reuter, K., Rippl, M., Röhrig-Zöllner, M., Sakurai, T., Scheffler, M., Scheurer, C., Shahzad, F., Simoes Brambila, D., Thies, J., Wellein, G.: Benefits from using mixed precision computations in the ELPA-AEO and ESSEX-II eigensolver projects. Japan Journal of Industrial and Applied Mathematics **36**, 699–717 (2019). DOI 10.1007/s13160-019-00360-8. URL https://doi.org/10.1007/s13160-019-00360-8

3. Alvermann, A., Basermann, A., Fehske, H., Galgon, M., Hager, G., Kreutzer, M., Krämer, L., Lang, B., Pieper, A., Röhrig-Zöllner, M., Shahzad, F., Thies, J., Wellein, G.: ESSEX: Equipping sparse solvers for exascale. In: L. Lopes, et al. (eds.) Euro-Par 2014: Parallel Processing Workshops, *LNCS*, vol. 8806, pp. 577–588. Springer (2014)
4. Alvermann, A., Fehske, H.: Exciton mass and exciton spectrum in the cuprous oxide. J. Phys. B **51**(4), 044001 (2018). DOI 10.1088/1361-6455/aaa060. URL http://stacks.iop.org/0953-4075/51/i=4/a=044001
5. Anzt, H., Tomov, S., Dongarra, J.: Accelerating the LOBPCG method on GPUs using a blocked Sparse Matrix Vector Product. University of Tennessee Innovative Computing Laboratory Technical Report UT-EECS-14-731 (2014). URL http://www.eecs.utk.edu/resources/library/589
6. Anzt, H., Tomov, S., Dongarra, J.: Implementing a sparse matrix vector product for the SELL-C/SELL-C-σ formats on NVIDIA GPUs. University of Tennessee Innovative Computing Laboratory Technical Report UT-EECS-14-727 (2014). URL http://www.eecs.utk.edu/resources/library/585
7. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for nonlinear eigenvalue problems using contour integrals. JSIAM Letters **1**, 52–55 (2009). DOI 10.14495/jsiaml.1.52
8. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for polynomial eigenvalue problems using contour integral. Japan Journal of Industrial and Applied Mathematics **27**(1), 73–90 (2010). DOI 10.1007/s13160-010-0005-x. URL https://doi.org/10.1007/s13160-010-0005-x
9. Balay, S., Abhyankar, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W.D., Karpeyev, D., Kaushik, D., Knepley, M.G., May, D.A., McInnes, L.C., Mills, R.T., Munson, T., Rupp, K., Sanan, P., Smith, B.F., Zampini, S., Zhang, H., Zhang, H.: PETSc Web page. https://www.mcs.anl.gov/petsc (2019). URL https://www.mcs.anl.gov/petsc
10. Barel, M.V., Kravanja, P.: Nonlinear eigenvalue problems and contour integrals. Journal of Computational and Applied Mathematics **292**, 526–540 (2016). DOI https://doi.org/10.1016/j.cam.2015.07.012. URL http://www.sciencedirect.com/science/article/pii/S037704271500374X
11. Bartlett, R., Demeshko, I., Gamblin, T., Hammond, G., Heroux, M., Johnson, J., Klinvex, A., Li, X., McInnes, L., Moulton, J.D., Osei-Kuffuor, D., Sarich, J., Smith, B., Willenbring, J., Yang, U.M.: xSDK foundations: Toward an extreme-scale scientific software development kit. Supercomput. Front. Innov.: Int. J. **4**(1), 69–82 (2017). DOI 10.14529/jsfi170104. URL https://doi.org/10.14529/jsfi170104
12. Beyn, W.J.: An integral method for solving nonlinear eigenvalue problems. Linear Algebra and its Applications **436**(10), 3839–3863 (2012). DOI https://doi.org/10.1016/j.laa.2011.03.030. URL http://www.sciencedirect.com/science/article/pii/S0024379511002540. Special Issue dedicated to Heinrich Voss's 65th birthday
13. Castro Neto, A.H., Guinea, F., Peres, N.M.R., Novoselov, K.S., Geim, A.K.: The electronic properties of graphene. Rev. Mod. Phys. **81**, 109–162 (2009). DOI 10.1103/RevModPhys.81.109. URL https://link.aps.org/doi/10.1103/RevModPhys.81.109
14. Chen, H., Imakura, A., Sakurai, T.: Improving backward stability of Sakurai-Sugiura method with balancing technique in polynomial eigenvalue problem. Applications of Mathematics **62**(4), 357–375 (2017). DOI 10.21136/AM.2017.0016-17. URL https://doi.org/10.21136/AM.2017.0016-17

15. Chen, H., Maeda, Y., Imakura, A., Sakurai, T., Tisseur, F.: Improving the numerical stability of the Sakurai-Sugiura method for quadratic eigenvalue problems. JSIAM Letters **9**, 17–20 (2017). DOI 10.14495/jsiaml.9.17
16. Chen, Y.C., Meilă, M.: Selecting the independent coordinates of manifolds with large aspect ratios. arXiv e-prints arXiv:1907.01651 (2019)
17. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2011). DOI 10.1145/2049662.2049663. URL http://doi.acm.org/10.1145/2049662.2049663
18. Druskin, V., Knizhnerman, L.: Two polynomial methods to compute functions of symmetric matrices. U.S.S.R. Comput. Maths. Math. Phys. **29**(6), 112–121 (1989)
19. Edwards, H.C., Trott, C.R., Sunderland, D.: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. Journal of Parallel and Distributed Computing **74**(12), 3202 – 3216 (2014). DOI https://doi.org/10.1016/j.jpdc.2014.07.003. URL http://www.sciencedirect.com/science/article/pii/S0743731514001257. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing
20. Ernst, D., Hager, G., Thies, J., Wellein, G.: Performance engineering for a tall & skinny matrix multiplication kernel on GPUs. CoRR **abs/1905.03136** (2019). URL http://arxiv.org/abs/1905.03136. Accepted for publication at PPAM'19, the 13h International Conference on Parallel Processing and Applied Mathematics, Bialystok, Poland, September 8-11, 2019
21. EXASTEEL project website: www.numerik.uni-koeln.de/14426.html
22. Fukasawa, T., Shahzad, F., Nakajima, K., Wellein, G.: pFEM-CRAFT: A Library for Application-Level Fault-Resilience Based on the CRAFT Framework. In: Poster at the 2018 SIAM Conference on Parallel Processing for Scientific Computing (SIAM PP18). Tokyo, Japan (2018)
23. Galgon, M., Krämer, L., Lang, B.: Improving projection-based eigensolvers via adaptive techniques. Numerical Linear Algebra with Applications **25**(1), e2124 (2018). DOI 10.1002/nla.2124. URL http://dx.doi.org/10.1002/nla.2124
24. Galgon, M., Krämer, L., Lang, B., Alvermann, A., Fehske, H., Pieper, A.: Improving robustness of the FEAST algorithm and solving eigenvalue problems from graphene nanoribbons. PAMM **14**(1), 821–822 (2014)
25. Galgon, M., Krämer, L., Lang, B., Alvermann, A., Fehske, H., Pieper, A., Hager, G., Kreutzer, M., Shahzad, F., Wellein, G., Basermann, A., Röhrig-Zöllner, M., Thies, J.: Improved coefficients for polynomial filtering in ESSEX. In: T. Sakurai, S.L. Zhang, T. Imamura, Y. Yamamoto, Y. Kuramashi, T. Hoshi (eds.) Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing, pp. 63–79. Springer International Publishing, Cham (2017)
26. Galgon, M., Krämer, L., Thies, J., Basermann, A., Lang, B.: On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues. Parallel Computing **49**, 153–163 (2015)
27. Gamblin, T., LeGendre, M.P., Collette, M.R., Lee, G.L., Moody, A., de Supinski, B.R., Futral, W.S.: The Spack package manager: Bringing order to HPC software chaos (2015). LLNL-CONF-669890
28. Giorgi, P., Vialla, B.: Generating optimized sparse matrix vector product over finite fields. In: Proceedings of ICMS 2014: Fourth International Congress on Mathematical Software, Seoul, Korea., vol. 8592, pp. 685–690. Springer LNCS (2014). URL http://www.lirmm.fr/~giorgi/icms2014-giovia.pdf
29. Gordon, D., Gordon, R.: CGMN revisited: Robust and efficient solution of stiff linear systems derived from elliptic partial differential equations. ACM Trans. Math. Softw. **35**(3), 18:1–18:27 (2008). DOI 10.1145/1391989.1391991. URL http://doi.acm.org/10.1145/1391989.1391991
30. Guettel, S., Polizzi, E., Tang, P., Viaud, G.: Zolotarev quadrature rules and load balancing for the FEAST eigensolver. SIAM Journal on Scientific Computing **37**(4), A2100–A2122 (2015). DOI 10.1137/140980090. URL https://doi.org/10.1137/140980090
31. Hasan, M.Z., Kane, C.L.: Colloquium: Topological insulators. Rev. Mod. Phys. **82**, 3045–3067 (2010). DOI 10.1103/RevModPhys.82.3045. URL https://link.aps.org/doi/10.1103/RevModPhys.82.3045

32. Hasegawa, T., Imakura, A., Sakurai, T.: Recovering from accuracy deterioration in the contour integral-based eigensolver. JSIAM Letters **8**, 1–4 (2016). DOI 10.14495/jsiaml.8.1
33. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An overview of the Trilinos project. ACM Trans. Math. Softw. **31**(3), 397–423 (2005). DOI http://doi.acm.org/10.1145/1089014.1089021
34. Hubbard, J., Flowers, B.H.: Electron correlations in narrow energy bands. Proc. Roy. Soc. London, Ser. A **276**(1365), 238–257 (1963). DOI 10.1098/rspa.1963.0204. URL `https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1963.0204`
35. Huber, S., Futamura, Y., Galgon, M., Imakura, A., Lang, B., Sakurai, T.: Flexible subspace iteration with moments for an effective contour-integration based eigensolver (2019). In preparation
36. Ikegami, T., Sakurai, T.: Contour integral eigensolver for non-hermitian systems: A Rayleigh-Ritz-type approach. Taiwanese Journal of Mathematics **14**(3A), 825–837 (2010). URL `http://www.jstor.org/stable/43834819`
37. Ikegami, T., Sakurai, T., Nagashima, U.: A filter diagonalization for generalized eigenvalue problems based on the Sakurai–Sugiura projection method. Journal of Computational and Applied Mathematics **233**(8), 1927–1936 (2010). DOI https://doi.org/10.1016/j.cam.2009.09.029. URL `http://www.sciencedirect.com/science/article/pii/S0377042709006529`
38. Imakura, A., Du, L., Sakurai, T.: A block Arnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems. Applied Mathematics Letters **32**, 22–27 (2014). DOI https://doi.org/10.1016/j.aml.2014.02.007. URL `http://www.sciencedirect.com/science/article/pii/S0893965914000421`
39. Imakura, A., Du, L., Sakurai, T.: Error bounds of Rayleigh–Ritz type contour integral-based eigensolver for solving generalized eigenvalue problems. Numerical Algorithms **71**(1), 103–120 (2016). DOI 10.1007/s11075-015-9987-4. URL `https://doi.org/10.1007/s11075-015-9987-4`
40. Imakura, A., Du, L., Sakurai, T.: Relationships among contour integral-based methods for solving generalized eigenvalue problems. Japan Journal of Industrial and Applied Mathematics **33**(3), 721–750 (2016). DOI 10.1007/s13160-016-0224-x. URL `https://doi.org/10.1007/s13160-016-0224-x`
41. Imakura, A., Futamura, Y., Sakurai, T.: An error resilience strategy of a complex moment-based eigensolver. In: T. Sakurai, S.L. Zhang, T. Imamura, Y. Yamamoto, Y. Kuramashi, T. Hoshi (eds.) Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing, pp. 1–18. Springer International Publishing, Cham (2017)
42. Imakura, A., Sakurai, T.: Block Krylov-type complex moment-based eigensolvers for solving generalized eigenvalue problems. Numerical Algorithms **75**(2), 413–433 (2017). DOI 10.1007/s11075-016-0241-5. URL `https://doi.org/10.1007/s11075-016-0241-5`
43. Imakura, A., Sakurai, T.: Block SS–CAA: A complex moment-based parallel nonlinear eigensolver using the block communication-avoiding Arnoldi procedure. Parallel Computing **74**, 34–48 (2018). DOI https://doi.org/10.1016/j.parco.2017.11.007. URL `http://www.sciencedirect.com/science/article/pii/S0167819117301886`. Parallel Matrix Algorithms and Applications (PMAA'16)
44. Iwashita, T., Nakashima, H., Takahashi, Y.: Algebraic block multi-color ordering method for parallel multi-threaded sparse triangular solver in iccg method. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS '12, pp. 474–483. IEEE Computer Society, Washington, DC, USA (2012). DOI 10.1109/IPDPS.2012.51
45. K. Sato et al.: Design and modeling of a non-blocking checkpointing system. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 19:1–19:10. IEEE Computer Society Press, Los Alamitos, CA, USA (2012)
46. Kawai, M., Ida, A., Nakajima, K.: Hierarchical parallelization of multi-coloring algorithms for block IC preconditioners. In: 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE

3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 138–145. IEEE (2017)
47. Kawai, M., Ida, A., Nakajima, K.: Modified IC preconditioner of CG method for ill-conditioned problems. Tech. Rep. Vol. 2017-HPC-158, No.9, IPSJ SIG (2017). In Japanese
48. Kawai, M., Ida, A., Nakajima, K.: Higher precision for block ILU preconditioner. In: CoSaS2018. FAU (2018)
49. Kawai, M., Iwashita, T., Nakashima, H., Marques, O.: Parallel smoother based on block red-black ordering for multigrid Poisson solver. In: High Performance Computing for Computational Science – VECPAR 2012, pp. 292–299 (2013)
50. Krämer, L.: Integration based solvers for standard and generalized Hermitian eigenvalue problems. Ph.D. thesis, Bergische Universität Wuppertal (2014). URL `http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:468-20140701-112141-6`
51. Krämer, L., Di Napoli, E., Galgon, M., Lang, B., Bientinesi, P.: Dissecting the FEAST algorithm for generalized eigenproblems. J. Comput. Appl. Math. **244**, 1–9 (2013)
52. Kreutzer, M.: Performance engineering for exascale-enabled sparse linear algebra building blocks. Ph.D. thesis, FAU Erlangen-Nürnberg, Technische Fakultät, Erlangen (2018). DOI 10.25593/978-3-96147-104-1
53. Kreutzer, M., Ernst, D., Bishop, A.R., Fehske, H., Hager, G., Nakajima, K., Wellein, G.: Chebyshev filter diagonalization on modern manycore processors and GPGPUs. In: R. Yokota, M. Weiland, D. Keyes, C. Trinitis (eds.) High Performance Computing, pp. 329–349. Springer International Publishing, Cham (2018)
54. Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Bishop, A.: A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. SIAM Journal on Scientific Computing **36**(5), C401–C423 (2014). DOI 10.1137/130930352. URL `https://doi.org/10.1137/130930352`
55. Kreutzer, M., Pieper, A., Hager, G., Wellein, G., Alvermann, A., Fehske, H.: Performance engineering of the Kernel Polynomal Method on large-scale CPU-GPU systems. In: 2015 IEEE International Parallel and Distributed Processing Symposium, pp. 417–426 (2015). DOI 10.1109/IPDPS.2015.76
56. Kreutzer, M., Thies, J., Pieper, A., Alvermann, A., Galgon, M., Röhrig-Zöllner, M., Shahzad, F., Basermann, A., Bishop, A.R., Fehske, H., Hager, G., Lang, B., Wellein, G.: Performance engineering and energy efficiency of building blocks for large, sparse eigenvalue computations on heterogeneous supercomputers. In: H.J. Bungartz, P. Neumann, W.E. Nagel (eds.) Software for Exascale Computing — SPPEXA 2013–2015, pp. 317–338. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-40528-5_14
57. Kreutzer, M., Thies, J., Röhrig-Zöllner, M., Pieper, A., Shahzad, F., Galgon, M., Basermann, A., Fehske, H., Hager, G., Wellein, G.: GHOST: Building blocks for high performance sparse linear algebra on heterogeneous systems. International Journal of Parallel Programming **45**(5), 1046–1072 (2017). DOI 10.1007/s10766-016-0464-z. URL `https://doi.org/10.1007/s10766-016-0464-z`
58. Li, R., Xi, Y., Erlandson, L., Saad, Y.: The Eigenvalues Slicing Library (EVSL): Algorithms, implementation, and software. Preprint, available at `http://www-users.cs.umn.edu/~saad/software/EVSL/index.html`
59. Matrix Market. `https://math.nist.gov/MatrixMarket/`. Accessed: 26 July 2019
60. Meila, M., Koelle, S., Zhang, H.: A regression approach for explaining manifold embedding coordinates. arXiv e-prints arXiv:1811.11891 (2018)
61. Müthing, S., Ribbrock, D., Göddeke, D.: Integrating multi-threading and accelerators into DUNE-ISTL. In: A. Abdulle, S. Deparis, D. Kressner, F. Nobile, M. Picasso, A. Quarteroni (eds.) Numerical Mathematics and Advanced Applications – ENUMATH 2013, *Lecture Notes in Computational Science and Engineering*, vol. 103, pp. 601–609. Springer (2014). DOI 10.1007/978-3-319-10705-9_59
62. ParMETIS - parallel graph partitioning and fill-reducing matrix ordering. URL `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`

63. Pieper, A., Kreutzer, M., Alvermann, A., Galgon, M., Fehske, H., Hager, G., Lang, B., Wellein, G.: High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations. J. Comput. Phys. **325**, 226–243 (2016). URL http://dx.doi.org/10.1016/j.jcp.2016.08.027
64. Polizzi, E.: Density-matrix-based algorithm for solving eigenvalue problems. Physical Review B **79**(11), 115112 (2009)
65. Polizzi, E.: Density-matrix-based algorithm for solving eigenvalue problems. Phys. Rev. B **79**, 115112 (2009). DOI 10.1103/PhysRevB.79.115112. URL https://link.aps.org/doi/10.1103/PhysRevB.79.115112
66. Röhrig-Zöllner, M., Thies, J., Kreutzer, M., Alvermann, A., Pieper, A., Basermann, A., Hager, G., Wellein, G., Fehske, H.: Increasing the performance of the Jacobi-Davidson method by blocking. SIAM J. Sci. Comp. **37**(6), 206–239 (2015). DOI 10.1137/140976017. URL http://dx.doi.org/10.1137/140976017
67. Sakurai, T., Asakura, J., Tadano, H., Ikegami, T.: Error analysis for a matrix pencil of Hankel matrices with perturbed complex moments. JSIAM Letters **1**, 76–79 (2009). DOI 10.14495/jsiaml.1.76
68. Sakurai, T., Futamura, Y., Imakura, A., Imamura, T.: Scalable Eigen-Analysis Engine for Large-Scale Eigenvalue Problems, pp. 37–57. Springer Singapore, Singapore (2019). DOI 10.1007/978-981-13-1924-2_3. URL https://doi.org/10.1007/978-981-13-1924-2_3
69. Sakurai, T., Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration. Journal of computational and applied mathematics **159**(1), 119–128 (2003)
70. Sakurai, T., Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration. Journal of Computational and Applied Mathematics **159**(1), 119–128 (2003). DOI https://doi.org/10.1016/S0377-0427(03)00565-X. URL http://www.sciencedirect.com/science/article/pii/S037704270300565X. 6th Japan-China Joint Seminar on Numerical Mathematics; In Search for the Frontier of Computational and Applied Mathematics toward the 21st Century
71. Schenk, O., Gärtner, K., Fichtner, W.: Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors. BIT Numerical Mathematics **40**(1), 158–176 (2000). DOI 10.1023/A:1022326604210. URL https://doi.org/10.1023/A:1022326604210
72. Schollwöck, U.: The density-matrix renormalization group. Rev. Mod. Phys. **77**, 259–315 (2005)
73. SCOTCH: Static mapping, graph, mesh and hypergraph partitioning, and parallel and sequential sparse matrix ordering package. URL http://www.labri.fr/perso/pelegrin/scotch/
74. Shahzad, F.: Checkpoint/Restart and Automatic Fault Tolerance(CRAFT) library. https://bitbucket.org/essex/craft. Accessed: 2017-07-27
75. Shahzad, F., Thies, J., Kreutzer, M., Zeiser, T., Hager, G., Wellein, G.: CRAFT: A library for easier Application-Level Checkpoint/Restart and Automatic Fault Tolerance. IEEE Transactions on Parallel and Distributed Systems **30**(3), 501–514 (2019). DOI 10.1109/TPDS.2018.2866794
76. Shahzad, F., Thies, J., Kreutzer, M., Zeiser, T., Hager, G., Wellein, G.: CRAFT: A library for easier application-level checkpoint/restart and automatic fault tolerance. IEEE Transactions on Parallel & Distributed Systems **30**(03), 501–514 (2019)
77. Song, W.: Matrix-based techniques for (flow-)transition studies. Ph.D. thesis, University of Groningen (2019). URL https://elib.dlr.de/125176/
78. Song, W., Wubs, F.W., Thies, J., Baars, S.: Numerical bifurcation analysis of a 3D Turing-type reaction-diffusion model. Communications in Nonlinear Science and Numerical Simulation (accepted) (2018)
79. Tang, P.T.P., Polizzi, E.: FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection. SIAM Journal on Matrix Analysis and Applications **35**(2), 354–390 (2014). DOI 10.1137/13090866X. URL https://doi.org/10.1137/13090866X

80. Thies, J., Galgon, M., Shahzad, F., Alvermann, A., Kreutzer, M., Pieper, A., Röhrig-Zöllner, M., Basermann, A., Fehske, H., Hager, G., Lang, B., Wellein, G.: Towards an exascale enabled sparse solver repository (2016). URL https://elib.dlr.de/100211/
81. Thies, J., Röhrig-Zöllner, M., Overmars, N., Basermann, A., Ernst, D., Hager, G., Wellein, G.: PHIST: a pipelined, hybrid-parallel iterative solver toolkit. ACM Transactions on Mathematical Software (2019). URL https://elib.dlr.de/123323/. Submitted.
82. ViennaCL - the Vienna computing library. URL http://viennacl.sourceforge.net/doc/index.html
83. Van der Vorst, H.A.: Iterative Krylov methods for large linear systems, vol. 13. Cambridge University Press (2003)
84. Weiße, A., Wellein, G., Alvermann, A., Fehske, H.: The kernel polynomial method. Rev. Mod. Phys. **78**, 275–306 (2006). DOI 10.1103/RevModPhys.78.275. URL https://link.aps.org/doi/10.1103/RevModPhys.78.275
85. Wouters, M., Vanroose, W.: Automatic exploration techniques for the numerical bifurcation study of the ginzburg-landau equation (2019). Submitted to SIAM J. Dynamical Systems
86. Yokota, S., Sakurai, T.: A projection method for nonlinear eigenvalue problems using contour integrals. JSIAM Letters **5**, 41–44 (2013). DOI 10.14495/jsiaml.5.41
87. Zoltan: Parallel partitioning, load balancing and data-management services. URL http://www.cs.sandia.gov/zoltan/Zoltan.html