Research Paper

International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Enhancing scalability of a matrix-free eigensolver for studying many-body localization

The International Journal of High
Performance Computing Applications
2022, Vol. 36(3) 307–319
© The Author(s) 2022
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420211060365
journals.sagepub.com/home/hpc
SAGE

Roel Van Beeumen[1], Khaled Z. Ibrahim[1], Gregory D. Kahanamoku–Meyer[2], Norman Y. Yao[2] and Chao Yang[1]

Abstract

We propose several techniques to enhance the parallel scalability of a matrix-free eigensolver designed for studying many-body localization (MBL) of quantum spin chain models with nearest-neighbor interactions and on-site disorder. This type of problem is computationally challenging because the dimension of the associated Hamiltonian matrix grows exponentially with respect to the number of spins L, and we need to average over different realizations of the random disorder to obtain relevant statistical behavior. For each disorder realization, we need to compute eigenvalues from different regions of the spectrum and their corresponding eigenvectors. In previous work, the interior eigenstates for a single eigenvalue problem are computed via the shift-and-invert Lanczos algorithm. Due to the extremely high memory footprint of the LU factorizations, this technique is not well suited for large L's. For example, we need thousands of compute nodes on modern high performance computing infrastructures to go beyond $L = 24$. The matrix-free approach does not suffer from this memory bottleneck, however, its scalability is limited by a computation and communication load imbalance. To reduce this imbalance and to significantly enhance the scalability of the matrix-free eigensolver, we reorder the matrix and leverage the consistent space runtime, CSPACER. We also show its efficiency in managing irregular communication patterns at scale compared to optimized MPI non-blocking two-sided and one-sided RMA implementation variants. This effort enables us to study MBL for spin chains with a larger number of spins. The efficiency and effectiveness of the proposed algorithm is demonstrated by computing eigenstates on a massively parallel many-core high performance computer.

Keywords

Quantum many-body problem, many-body localization, eigenvalue problem, matrix-free eigensolver, locally optimal block preconditioned conjugate gradient method, preconditioner, scalability, graph partitioning, METIS, communication optimization, CSPACER

1. Introduction

A fundamental assumption in the traditional theory of statistical mechanics is that an isolated system will in general reach an equilibrium state, or *thermalize*. As early as the mid-20th century, Anderson demonstrated that a single particle moving in a highly disordered landscape can violate this assumption (Anderson, 1958). While surprising that result does not readily extend to many-particle systems that exhibit strong interactions between the constituent particles. The question of whether a similar effect could manifest in a strongly interacting many-body system remained open for decades. This elusive phenomenon has been termed "many-body localization" (MBL).

Recently, advances in both high performance computing and experimental control of individual quantum particles have begun to yield insight into MBL. Both experimental (Bordia et al., 2017; Choi et al., 2016; Kohlert et al., 2019; Lukin et al., 2019; Schreiber et al., 2015; Smith et al., 2016) and numerical (Bauer and Nayak, 2013; Cuevas et al., 2012; Johri et al., 2015; Luitz et al., 2015; Pal and Huse, 2010) results have shown evidence of localization in small strongly interacting multiparticle systems of 10–20 spins. Unfortunately, extrapolating results from these small system

[1] Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[2] Department of Physics, University of California at Berkeley, Berkeley, CA, USA

Corresponding author:
Roel Van Beeumen, Computational Research Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA.
Email: rvanbeeumen@lbl.gov

sizes to the infinitely large thermodynamic limit has proven difficult. This lack of clarity has inspired a vigorous debate in the community about precisely what can be learned from small-size results. For example, it has been proposed that certain features do not actually exist at infinite system size (De Roeck et al., 2016), and even that MBL itself is only a finite-size effect (Abanin et al., 2021; Šuntajs et al., 2020)!

The primary goal of most studies is to identify and characterize a *localization transition*. In the thermodynamic limit, as the strength of the system's disorder increases, theory predicts a sharp, sudden change from a thermal to a localized state. Unfortunately, in the small systems available for study, that sharp transition turns into a smooth *crossover*, leading to the confusion about what constitutes the transition itself. Numerical evidence suggests that the transition sharpens rapidly as system size increases, so accessing as large systems as possible is imperative for investigating MBL.

In pursuit of that goal, Luitz et al. used large-scale numerical linear algebra to show a localization transition for system sizes up to $L = 22$ (Luitz et al., 2015), and in a following paper extracted useful data up to $L = 24$ (Pietracaprina et al., 2018). In order to compute interior eigenstates for the MBL problem, the shift-and-invert Lanczos algorithm was used in combination with sparse direct solvers for solving the linear systems. One of the major disadvantages of this technique is that constructing the LU factorizations becomes extremely memory demanding, due to the so called fill in, for large number of spins L. Figure 1 shows that the memory footprint of the LU factorization computed via STRUMPACK (Ghysels et al., 2017) grows rapidly as function of L; see also (Pietracaprina et al., 2018). Hence, thousands of nodes on modern high performance computing infrastructures are needed to go beyond $L = 24$.

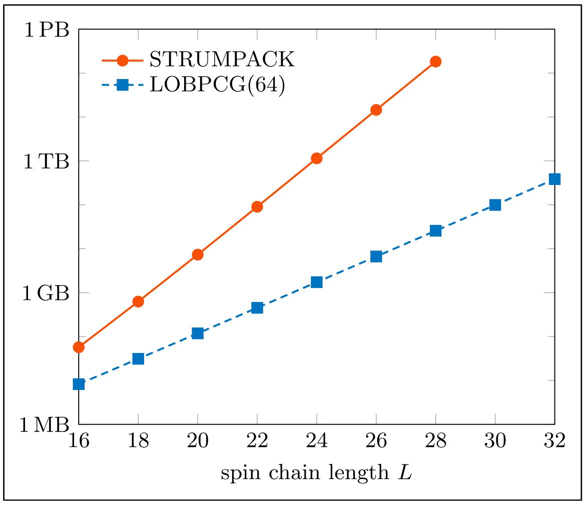

Figure 1. Total memory footprint as a function of the spin chain length L for LU factorizations, computed via STRUMPACK, and the matrix-free LOBPCG algorithm [18], with block size 64.

To overcome the memory bottleneck that the shift-and-invert Lanczos algorithm faces, we recently proposed in (Van Beeumen et al., 2020) using the matrix-free locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm (Knyazev, 2001). As shown in Figure 1, this approach reduces the memory footprint by several orders of magnitude, for example, from 15 TB to only 7 GB for $L = 26$, and enables simulating spin chains on even a single node, up to $L = 24$. In contrast to the shift-and-invert Lanczos algorithm, where the dominant computational cost is the construction of the LU factorization, the dominant computational cost of the LOBPCG algorithm is the (block) matrix-vector (MATVEC) product. As illustrated in (Van Beeumen et al., 2020), the scalability of this MATVEC is limited at high concurrency which is due to a computation and communication imbalance. In the current paper, we present different strategies to overcome this imbalance and to significantly enhance the scalability of the matrix-free eigensolver.

The paper is organized as follows. We first review the Heisenberg spin model and MBL metrics in Section 2. The multiple levels of concurrency and the matrix-free LOBPCG eigensolver are discussed in Section 3. Next, we present the balancing of computation and communication within the MATVEC in Section 4 and the optimization of the communication performance in Section 5. Then in Section 6, we illustrate the different proposed strategies for improving the computation and communication imbalance of the matrix-free LOBPCG eigensolver for $L = 24$ and $L = 26$ problems. Finally, the main conclusions are formulated in Section 7.

2. Problem formulation

In this section, we briefly review the properties of the spin chain model that most frequently is studied by numerical simulations of MBL.

2.1. Heisenberg spin model

We consider the nearest-neighbor interacting Heisenberg spin model with random on-site fields

$$H = \sum_{\langle i,j \rangle} \vec{S}_i \cdot \vec{S}_j + \sum_i h_i S_i^z \qquad (1)$$

where the angle brackets denote nearest-neighbor i and j, h_i is sampled from a uniform distribution $[-w, w]$ with $w \in \mathbb{R}_0^+$, and

$$\vec{S}_i \cdot \vec{S}_j = S_i^x S_j^x + S_i^y S_j^y + S_i^z S_j^z$$

where $S_i^\alpha = 1/2\sigma_i^\alpha$, with σ_i^α the Pauli matrices operating on lattice site i and $\alpha \in \{x, y, z\}$. The parameter w is called the *disorder strength*, and is responsible for inducing the MBL transition. The values h_i are sampled randomly each time the Hamiltonian is instantiated, and the relevant physics lies in the statistical behavior of the set of all such Hamiltonians. The individual Hamiltonians H with independently sampled h_i are called *disorder realizations*.

Note that in (1) each term of each sum has an implied tensor product with the identity on all the sites not explicitly written. Consequently, the Hamiltonian for L spins is a symmetric matrix of dimension $N = 2^L$ and exhibits the following tensor product structure

$$\begin{aligned} H = &\sum_{i=1}^{L-1} I \otimes \cdots \otimes I \otimes H_{i,i+1} \otimes I \otimes \cdots \otimes I \\ &+ \sum_{i=1}^{L} I \otimes \cdots \otimes I \otimes h_i S_i^z \otimes I \otimes \cdots \otimes I \end{aligned} \qquad (2)$$

where $H_{i,i+1} = S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z$ is a 4-by-4 real matrix and I is the 2-by-2 identity matrix. Remark that by definition, all matrices $H_{i,i+1}$ are the same and independent of the site i. For our experiments, we use open boundary conditions, meaning that the nearest-neighbor terms do not wrap around at the end of the spin chain. Open boundary conditions can be considered to yield a larger *effective* system size because of the reduced connectivity.

The state of each spin is described by a vector in $\mathbb{C}^2$, and the configuration of the entire L-spin system can be described by a vector on the tensor product space $(\mathbb{C}^2)^{\otimes L}$. In this specific case, however, the Hamiltonian's matrix elements happen to all be real, so we do not include an imaginary part in any of our computations. Furthermore, our Hamiltonian commutes with the total magnetization in the z direction, $S^z = \sum_{i=1}^{L} S_i^z$. Thus, it can be block-diagonalized in sectors characterized by $S^z \in [-L/2, -L/2+1, \ldots, L/2-1, L/2]$. The vector space corresponding to each sector has dimension $n = \binom{L}{S^z + L/2}$ such that the largest sector's dimension is $n = L!/[(L/2)!(L/2)!]$, and this corresponds to the actual dimension of the matrices on which we operate, see Figure 1. While these subspaces are smaller than the full space, their size still grows exponentially with the number of spins L. Thus, the problem becomes difficult rapidly as L increases. Furthermore, the density of eigenvalues in the middle of the spectrum increases exponentially with L. Thus, the tolerance used to solve for these internal eigenvalues must be made tighter rapidly as L increases.

2.2. Many-body localization

With the problem's matrix clearly defined, we now review ways for quantifying localization from the eigenvalues and eigenvectors. There are multiple quantities that can be used for identifying localization.

One of the commonly used quantities is the *adjacent gap ratio* (Cuevas et al., 2012; Johri et al., 2015; Oganesyan and Huse, 2007; Šuntajs et al., 2020). This approach is based on the statistical distribution of the eigenvalues of different disorder realizations, hence, only eigenvalues need to be computed. Random matrix theory informs us that the statistical distribution of eigenvalues will differ between localizing and thermalizing Hamiltonians (Oganesyan and Huse, 2007). In particular, we expect eigenvalues of a thermal Hamiltonian to *repel* each other, that is, hybridization of eigenvectors prevents them from generally coming too close to one another. The eigenvalues of a localized Hamiltonian should not display this behavior: we expect them to be Poisson distributed. Therefore, we can measure localization by comparing the relative size of gaps between the eigenvalues. Thermal Hamiltonians will generally have more consistently sized gaps due to level repulsion. However, this technique suffers from large statistical noise and thus requires many samples to be usable.

Another quantity for measuring localization is the *eigenstate entanglement entropy* (Van Beeumen et al., 2020) which is based on the eigenvectors of the Hamiltonians. In a thermal system, we expect quantum entanglement to be widespread, while in a localized system, the entanglement is not expected to be extensive. This idea can be quantified by choosing a *cut* which divides the spin chain into two pieces, and measuring the entanglement across it. Not only the value of the entropy changes during the localization transition: the statistics change as well. When compared across disorder realizations, the thermal entanglement entropy has small variance. During the transition, however, the entanglement entropy depends strongly on the specific disorder realization and thus the statistic will have a large variance. Empirically, examining the *variance* of the entanglement entropy is one of the best ways to identify the localization transition and requires fewer samples than the adjacent gap ratio approach.

3. Massively parallel simulation

In order to maximally reduce the finite-size effects on the determination of the MBL transition point, we need to study spin models with as many spins as possible. Consequently, this problem is computationally demanding and requires lots of resources.

3.1. Multiple levels of concurrency

The MBL study allows for at least four levels of concurrency. The first level corresponds to the need of averaging over (many) different and independently sampled *disorder realizations* in order to obtain relevant statistical behavior.

Since the *disorder strength* is responsible for inducing the MBL transition, we also have to vary the disorder strength, giving rise to the second level of concurrency. The third level corresponds to the *eigenvalue chunks*, that is, for each (large) eigenvalue problem, originating from one disorder realization and a particular disorder strength, we have to compute eigenvalues from different regions of the spectrum and their corresponding eigenvectors.

All previous levels of concurrency are completely independent and can be implemented in a massively parallel fashion by making use of iterative eigensolvers. In this paper, we therefore only focus on the fourth level of parallelism taking place within these eigensolvers. Although most iterative eigensolvers follow a rather sequential procedure, each of the different steps within one iteration can be implemented in parallel.

3.2. Matrix-free LOBPCG eigensolver

The Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) algorithm (Duersch et al., 2018; Knyazev, 2001) is a widely used eigensolver for computing the smallest or largest eigenvalues and corresponding eigenvectors of large-scale symmetric matrices. Key features of the LOBPCG algorithm are: (i) It is matrix-free, that is, the solver does not require storing the coefficient matrix explicitly and it access the matrix by only evaluating matrix-vector products; (ii) It is a block method, which allows for efficient matrix-matrix operations on modern computing architectures; (iii) It can take advantage of preconditioning, in contrast to, for example, the Lanczos algorithm.

The standard LOBPCG algorithm allows for computing either the lower or upper part of the spectrum. In order to compute interior eigenvalues and their corresponding eigenvectors, we make use of the so called *spectral fold* transformation (Tomov et al., 2005; Wang and Zunger, 1994)

$$(H - \sigma I)^2$$

where $\sigma \in \mathbb{R}$ is the shift around which we want to compute eigenvalues. This spectral transformation maps all eigenvalues to the positive real axis and the ones closest to the shift σ to the lower edge close to 0. Hence, after applying this transformation, we can also use the LOBPCG eigensolver for computing interior eigenvalues. Because the transformed eigenvalue problem

$$(H - \sigma I)^2 x = \lambda x$$

is symmetric positive definite, we use a diagonal (Jacobi) preconditioned conjugate gradient (PCG) method as preconditioner for the LOBPCG eigensolver. For more details on the matrix-free LOBPCG eigensolver used in the particular case of studying MBL, we refer to (Van Beeumen et al., 2020).

In contrast to the shift-and-invert Lanczos algorithm, where the dominant computational cost is the construction of the LU factorization, the dominant computational cost of the LOBPCG and PCG algorithms is the (block) MATVEC. In the remainder of the paper, we therefore will mainly focus on enhancing the scalability of the MATVEC.

4. Balancing computation and communication

In this section we have a closer look at the MBL (block) matrix-vector product and focus on how to enhance its scalability by reducing the computation and communication imbalance.

4.1. Matrix-free Matrix-vector product

As a starting point, we take the hybrid MPI–OpenMP MATVEC introduced in (Van Beeumen et al., 2020). This matrix-free MATVEC uses one MPI rank per node and OpenMP for on-node parallelism. The block of vectors to be multiplied by the Hamiltonian matrix is partitioned by rows and distributed among different MPI ranks (and nodes). Within each MPI rank, a local sparse MATVEC is performed. A subset of the rows in the local vector block needs to be sent to other MPI ranks to be multiplied and accumulated on the target MPI ranks. Each MPI rank also receives vector block contributions from other MPI ranks to be combined with the local product. It has been illustrated in (Van Beeumen et al., 2020) that the parallel MATVEC implementation using non-blocking MPI communication, in combination with overlapping communication and local computation, results in the best performance.

Figure 2 shows the average wall clock time as a function of the rank for 100 samples of the $L = 26$ non-blocking MPI-OpenMP MATVEC. The different amounts of time spend in the MPI barrier shows clearly the high computation and communication imbalance of this MATVEC. Note that the local computation time can be fully overlapped with the communication and that the computation time shown only corresponds to the *remote* computation time which can only start once the incoming data has arrived. Figure 2 also illustrates that this MATVEC is communication dominated. The difference between computation and communication time further increases for higher concurrency. Therefore, in the remainder of this paper, we will focus on different strategies for reducing and optimizing the communication time.

4.2. Graph partitioning

The row partition of the vectors among different MPI ranks corresponds to a partition of the adjacency graph induced by

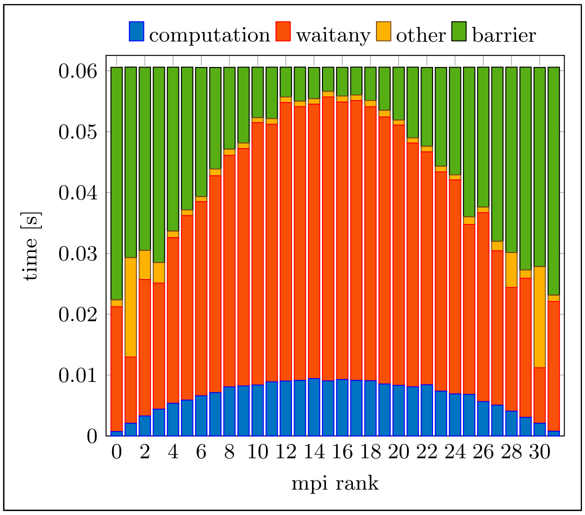

Figure 2. Average wall clock time as a function of the MPI rank for the different components of the L = 26 non-blocking MPI–OpenMP MATVEC with block size 64.

the many-body Hamiltonian, with vertices mapped rows or columns, and edges mapped to nonzero elements of the Hamiltonian. Within each partition, vertices that are not connected with vertices in other partitions by edges are strictly local. The corresponding rows do not need to be sent to other MPI ranks. The vertices that are connected to vertices in other partitions are called *shared rows*. They need to be sent to other MPI ranks in a parallel MATVEC implementation.

The state ordering for the hybrid MPI–OpenMP MATVEC in (Van Beeumen et al., 2020) leads to a simple communication pattern where, up to ~50 ranks, only communication with neighboring ranks in a linear topology is required. The ordering of the states also allows for efficiently computing the off-diagonal element indices on the fly for applying the matrix-free MATVEC. However, as shown in Figure 2, the communication is largely imbalanced and far from optimal. In particular, the graph for the Hamiltonian is quite nonuniform, and vertices in the *middle* are much more densely interconnected than vertices on the *edge*.

One of the properties of the Heisenberg spin model with random on-site fields is that the sparsity pattern of (2) does not change for different disorder realizations, neither does the corresponding matrix graph. Therefore, we can apply graph partition techniques in order to reduce the communication volume and better balance the communication time among the different MPI ranks. A comparison of the communication volume, as a function of the total number of MPI ranks, between the hybrid MPI–OpenMP state ordering used in (Van Beeumen et al., 2020) and the METIS k-way graph partitioning (Karypis & Kumar, 1998) reordering of the states is shown in Table 1. For the METIS graph partitioning, we used the objective function for total communication volume minimization (Karypis, 2013).

Table 1. Communication volume as a function of the total number of MPI ranks for the L = 26 MPI–OpenMP MATVEC with the state ordering in [18] and the METIS reordering.

Ranks	4	8	16	32
(a) Total communication volume				
[18]	1097 358	2552 781	4788 256	9620 799
METIS	1406 736	2350 508	4082 179	5893 137
Gain	−28%	8%	15%	39%
(b) Maximum communication volume per rank				
[18]	376 007	408 962	416 484	419 402
METIS	481 086	389 561	348 559	277 669
Gain	−28%	5%	16%	34%

Table 1 shows that for the L = 24 MATVEC beyond four MPI ranks both the total communication volume as well as the maximum communication volume per rank can significantly be reduced by using the METIS reordering. Note that this reordering will have an effect on both the computation and communication within the MATVEC. First, although the communication volume is reduced, the METIS reordering of the states results in a more complicated communication pattern compared to the original MPI–OpenMP one in (Van Beeumen et al., 2020). Hence, each rank needs, in general, to communicate with more than two other ranks. Second, the METIS reordering of the states is also less structured and therefore the remote MATVEC computation will require more complicated lookup tables. However, since the MATVEC is communication dominant and we maximally overlap computation with communication, the extra overhead from a slightly slower computational portion will be marginal.

5. Communication performance optimization

The load-balancing of computation using METIS affects the communication pattern—specifically, the number of messages per rank from being constant to being a function of the rank count.

Figure 3 shows the distribution of the neighbor count a rank needs to communicate the vector with, as we increase the job size. With METIS partitioning, depending on the rank position within a job, a different number of neighbors are involved in the vector exchange. With each of these neighbors, a rank needs to communicate a portion of their assigned vector. As shown in Figure 3, for the L = 26 problem, the number of neighbors increases super-linearly with respect to log(r), where r is the rank count.

Figure 4 shows the number of rows, of the partitioned block vector, each rank exchanges with their neighbors as a

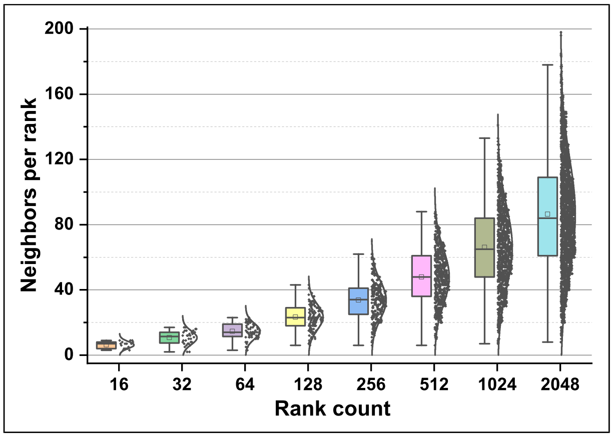

Figure 3. The distribution of neighbor count for vector data exchange as we scale the job size due to load-balancing the L = 26 MATVEC. Not only the number of communication messages increases, but also the variability increases significantly while scaling.

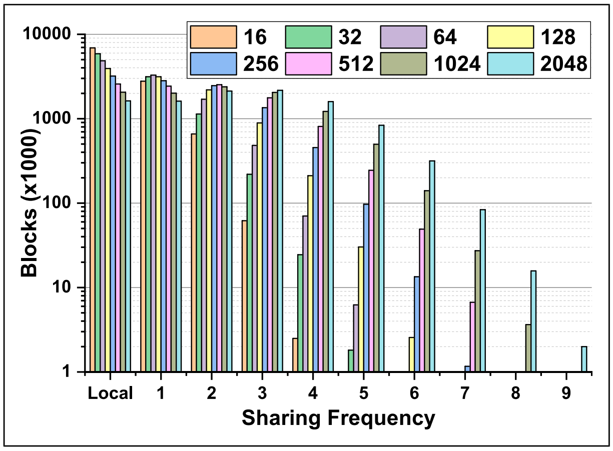

Figure 4. The amount of data to be communicate (or "shared") among different ranks in the L = 26 MATVEC communication phase as we scale the computation. The sharing level increases as the number of ranks increases, making the scaling communication bound.

function of the *sharing level*, that is, the number of neighbors a row block is sent to. From this figure, we see that the number of row blocks not involved in data exchange decreases as we strong scale the computation, hence, making it necessary to communicate an increasing fraction of rows. Moreover, the sharing level increase as well, making it necessary to exchange the same row with multiple ranks. Such sharing makes the volume of communicated data to decrease sub-linearly with the rank count. Overall, the total volume of communicated data increases as we scale the job, roughly proportional to the $\sqrt{r}$.

Fortunately, the sparsity pattern that influences the communication pattern remains unchanged across iterations. As such, one could classify this communication pattern as a static irregular one, and we can construct all needed information about the communication pattern before the communication starts. Although the algorithm relies on other communication primitives, such as allreduce, they do not significantly contribute to the execution time.

The discussed attributes show the challenge in strong-scaling the computation, which could be summarized by the need for processing an increasingly large number of small messages.

5.1. Structuring communication using MPI

MPI provides multiple techniques to implement a communication pattern. While not claiming that we exhausted all possible methods, we explored a few widely used techniques that are likely to serve our communication pattern.

The most natural way to implement our matrix-vector communication pattern is to use non-blocking point-to-point transfers. We overlapped local computation with some of the communication cost by assigning a thread to progress communication in the background while performing local computations.

In addition to using point-to-point communication primitives, we explored the use of MPI-3 non-blocking all-to-all collective primitives (Hilbrich et al., 2016), and MPI-3 RMA (Hoefler et al., 2015). The collective-based implementation matches more or less the non-blocking point-to-point and is omitted for brevity. We found the MPI one-sided implementation efficient at small scale, but the scaling behavior is inefficient in our experience on the Cray XC40 system.

5.2. Structuring communication using CSPACER

In this study, we explored the use of CSPACER (Ibrahim, 2020, 2021), Consistent SPACE Runtime, which provides a low overhead communication abstraction for irregular communication patterns. The runtime extends the support of the consistency space abstraction (Ibrahim, 2019) to Cray systems. The runtime could interoperate with the MPI runtime, allowing for incremental integration and tackling communication hotspots while retaining the bulk of MPI's communication code.

The space consistency abstraction (Ibrahim, 2019) is a generalization of full-empty synchronization for distributed computing, where each memory region is associated with a counter that determines its consistency. A memory space becomes consistent, that is, ready for consumption, when the counter matches a specific consistency tag. To construct a consistent state, a space typically receives one or multiple transfers from one or more producers. The runtime provides APIs to facilitate checking the consistency of a space for consumers, but it does not provide the functionality of tracking the completions for individual data transfers. The runtime relies on symmetric space allocations across a team of ranks, and supports communication primitives such as one-sided put, various collectives that are implemented as patterns of multiple underlying primitives. Due to its simple design, the CSPACER runtime enjoys a low injection overhead, in the range of 0.4 µs on KNL architectures, and provides good scalability, especially for irregular communication patterns.

The space consistency adopts a memory-centric approach while orchestrating communication across ranks, rather than relying on transfer-centric strategies. It supports multiple mechanisms for issuing transfer operations that help achieving a consistent state, including concurrent threaded injection from an OpenMP parallel region, pipelined injection and progress, etc. The runtime implements these operations without significant injection overhead or serialization between concurrent threads. Moreover, successfully injected transfers progress in the background without requiring the runtime polling for progress, or assigning the progress to a thread. While transfer injection is non-blocking by default, a transfer injection could be blocked until resources are available. This back-pressure mechanism provides some throttling mechanism to avoid congesting the interconnect.

By not providing a mechanism of tracking completion of individual transfers or ordering constraints between transfers, the runtime could handle a large number of transfers with minimal overhead. We structured the communication such that a single space per rank receives the contribution of all producer ranks. Therefore, involving more ranks in the communication does not result in an increase of the overhead of checking the data readiness for consumption. The advantage of such approach manifests at scale.

6. Numerical experiments

All numerical experiments were performed on the NERSC super computer called Cori, a Cray XC40 system powered by Intel Xeon Phi “Knights Landing” (KNL) compute nodes @1.4 GHz, 68 cores and each with four hyperthreads, 96 GB DDR4 RAM, 16 GB MCDRAM. The Cray XC40 nodes are connected using a Dragonfly Aries interconnect.

Throughout the numerical experiments we use a fixed block size of 64 for the MATVECs and the LOBPCG eigensolver. We also use one MPI rank per node and OpenMP for on-node parallelism.

6.1. MPI–OpenMP MATVEC

In a first experiment, we compare the different implementations of the MATVEC. Figure 5 shows the strong-scaling results for the $L = 24$ and $L = 26$ MATVECs. In this figure, the dashed lines correspond to the state ordering of (Van Beeumen et al., 2020) with non-blocking MPI communication and the dotted and solid lines correspond to the proposed METIS state reordering with non-blocking MPI and CSPACER communication, respectively.

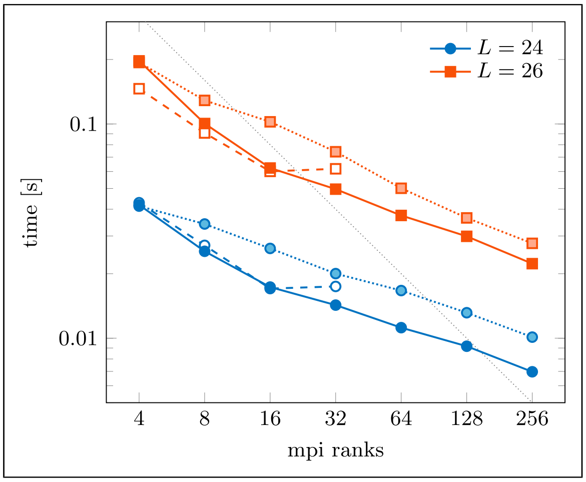

Figure 5. Strong scaling of 1 MATVEC with block size 64. The dashed lines correspond to the state ordering used in [18], the dotted lines to the METIS ordering, and the solid lines to the METIS ordering + CSPACER.

Due to the large computation and communication imbalance, as shown in Figure 2, the dashed lines in Figure 5 show that the scalability of the MATVEC implementation using the original ordering of the states stops at 16 ranks. On the other hand, this figure clearly shows that using the proposed METIS reordering of the states yields the MATVEC to continue scaling for higher concurrency. Although the METIS reordering reduces both the total communication volume and the maximum communication volume per rank, just changing the ordering of the states itself does not lead to an improvement of the wall clock time for low concurrency. This is because the METIS reordering leads to a more complicated communication pattern, which could explain the higher overall wall clock time of the dotted lines in Figure 5. As shown by the solid lines, the extra cost originating from a more complicated communication pattern can be mitigated with an optimized runtime, for instance using CSPACER.

The quality of the load-balancing of the matrix partitions influences the scaling behavior because the slowest rank dictates the overall performance. The graph partitioning is an NP-hard problem and grows in complexity with the problem size and the number of partitions (rank count). Later results show that the work distribution, despite being significantly improved, is not perfectly balanced. A such, we conjecture that an improved load-balancing will result in a better scaling behavior for the studied problem.

6.2. Matrix-free LOBPCG eigensolver

As discussed in Section 3.1, the MBL problem exhibits multiple levels of concurrency and requires computing eigenvalues/eigenvectors from different spectral regions. Since computing eigenvalues in the middle of the spectrum are the hardest, we focus on computing eigenvalues around the shift $\sigma = 0$.

In all remaining experiments, we use the matrix-free LOBPCG eigensolver with the METIS reordering for the MATVEC. The PCG preconditioner has been implemented as single-vector PCG running in parallel for each residual and with a synchronization on the MATVEC, so that we can always make use of the efficient (block) MATVEC. The allreduce operations in the LOBPCG and PCG solver use MPI, in contrast to the MATVEC for which we compare different communication strategies. Because most of the MATVECs take place in the preconditioner, we also perform all PCG iterations in single precision and only the LOBPCG iterations in double precision. This mixed precision approach for the MBL problem turns out to have no effect on the overall eigenvalue accuracy or the total number of LOBPCG iterations, however, it significantly reduces the wall clock time.

The $L = 24$ strong-scaling behavior of the different communication strategies within the MATVEC are presented in Figure 6. Since the number of LOBPCG iterations required to reach convergence depends on the disorder strength, the region of the spectrum from which eigenvalues are computed, and the number of iterations in the preconditioner, we only report the timings for 1 $L = 24$ LOBPCG iteration with 5000 PCG iterations per LOBPCG iteration. Note that the number of outer LOBPCG iterations and the number of inner PCG iterations are related, that is, the number of outer iterations (using double precision MATVECs) can be reduced when the number of inner iterations (using single precision MATVECs) is increased. From Figure 6, we notice that the CSPACER-variant outperforms the non-blocking MPI-variant and that the difference grows for increasing concurrency. On the other hand, one-sided remote memory access (rma) communication only performs well at low concurrency and is even in that case not competitive with the CSPACER-variant. Therefore, due to the poor scalability of the rma MPI-variant, we will not further consider this type of communication.

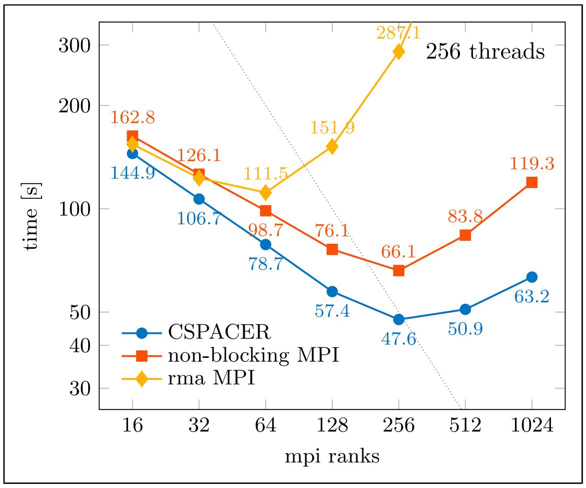

Figure 6. Strong scaling of 1 $L = 24$ LOBPCG iteration with 5000 PCG iterations in single precision.

The upper part of Figure 7 presents the strong-scaling behavior for the $L = 24$ problem, while using different thread counts per node. We notice that the need for full thread concurrency diminishes as we scale, to the extent we start seeing performance degradation at high node concurrency. This behavior could be attributed to an increased overhead for managing thread pools, for example, barrier synchronization for the amount of work assigned to the threads. We notice that the CSPACER-variant suffers less from performance degradation compared to the MPI-variant. We attribute that to the former using threading more efficiently in injecting and progressing multiple transfer lanes in the interconnect. The corresponding speedup factors for the CSPACER-variant compared to MPI-variant are presented in Table 2. We notice that in almost all situations the CSPACER-variant results in a significant speedup and, as also shown in Figures 6 and 7, the speedup factors increase, in general, for increasing concurrency.

CSPACER has a constant, $O(1)$, complexity in checking the consistency of the memory space, regardless of the number of ranks involved in constructing this space. Such a consistency check is necessary before the data consumer is able to process incoming data in all-to-all exchange. In contrast, MPI's localized all-to-all has a linear, $O(r)$, asymptotic complexity in processing incoming transfers, with r the number of MPI ranks. The difference manifests at scale, especially for strong-scaling experiments, where the runtime overhead plays a significant factor in performance.

CSPACER's use of threaded injection at the source is not associated with overhead because the runtime determines the lock-free concurrency level (Ibrahim, 2021) (injection lanes) that the application should use.

The lower part of Figure 7 shows the decomposition of the execution time for the best performing threading configuration for both the MPI- and CSPACER-variants. As shown, the MATVEC, blue box, exhibits a significant fraction of the execution time, especially at low concurrency. As we strong scale the computation, the allreduce operations, red box, start contributing significantly to the execution time. Because the number of elements in the reduction remains constant, we expect the overhead of the reduction to increase with the number of nodes. Instead,

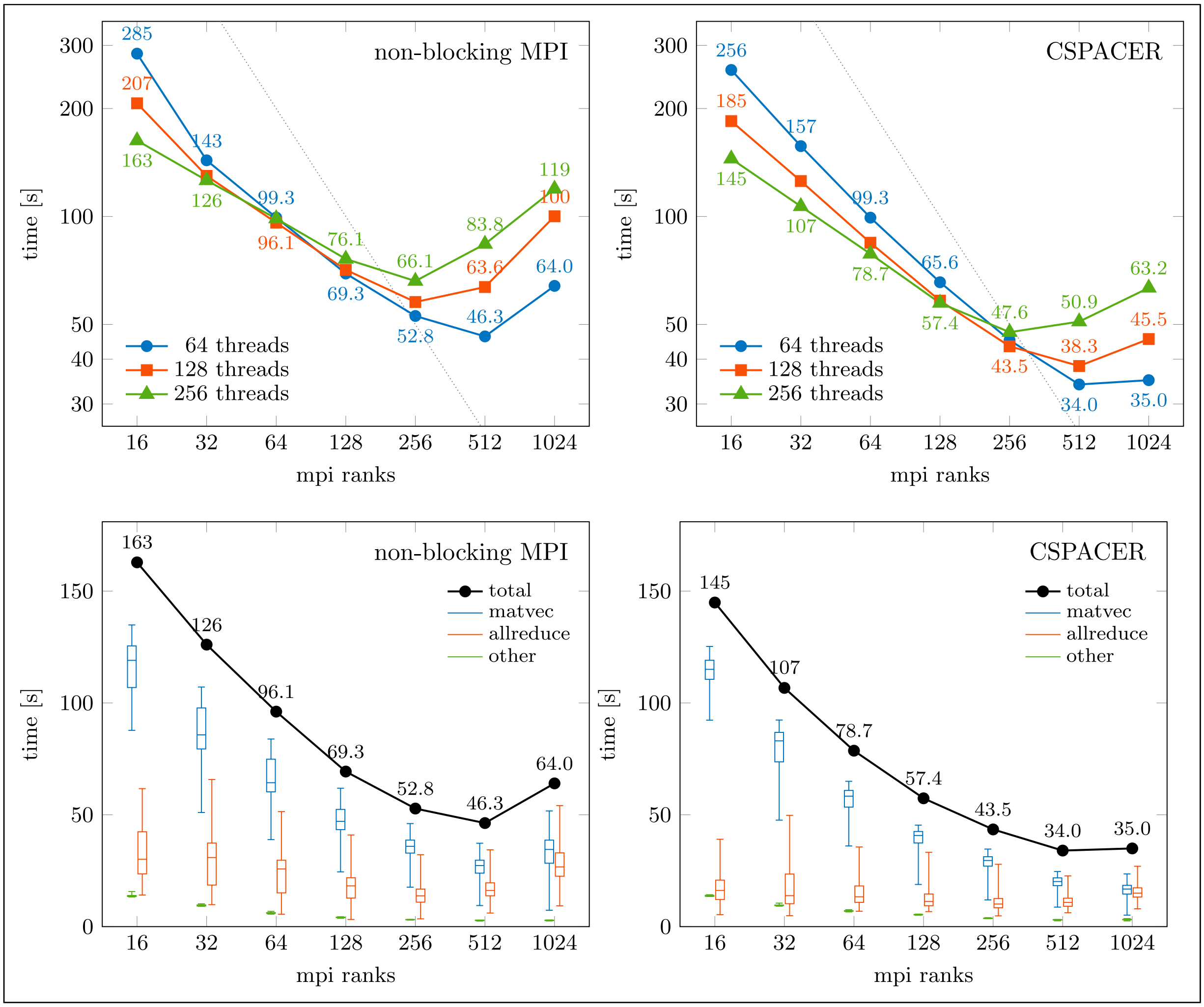

Figure 7. Strong scaling of 1 L = 24 LOBPCG iteration with 5000 PCG iterations in single precision.

Table 2. L = 24 speedups for the CSPACER-variant compared to non-blocking MPI communication.

Ranks	256 threads (%)	128 threads (%)	64 threads (%)
16	11.0	10.8	9.9
32	15.3	3.3	−9.5
64	20.3	12.1	0.0
128	24.6	18.0	5.4
256	28.0	24.7	14.4
512	39.2	39.8	26.6
1024	47.0	54.5	45.4

we noticed a strong correlation between the variability of the MATVEC and the allreduce phases. Given that these two phases are executed consecutively, we conducted an experiment with an extra barrier between them and found that the barrier captured most of the variability. We omitted this extra barrier synchronization due to its unnecessary overheads in the presented results.

In general, we note that there are multiple sources of performance variability across nodes in our code. The first is due to the computational load imbalance originating from the imperfect partitioning; The second is due to the system noise through the shared interconnect; The third is due to the communication runtime. The MATVEC overlap of computation with communication makes it difficult to isolate the impact of communication from computation imbalance. Having two runtime implementations allow for classifying sources of variability better. For instance, the lower variability of the CSPACER-variant compared to the MPI-variant shed some light on the minimum variability due to the MPI runtime. In our experiments, we found the interquartile range for the MATVEC variability for MPI compared to CSPACER to be 1.5× at low concurrency and reaching 2.7× at 1024 nodes. We also consider the

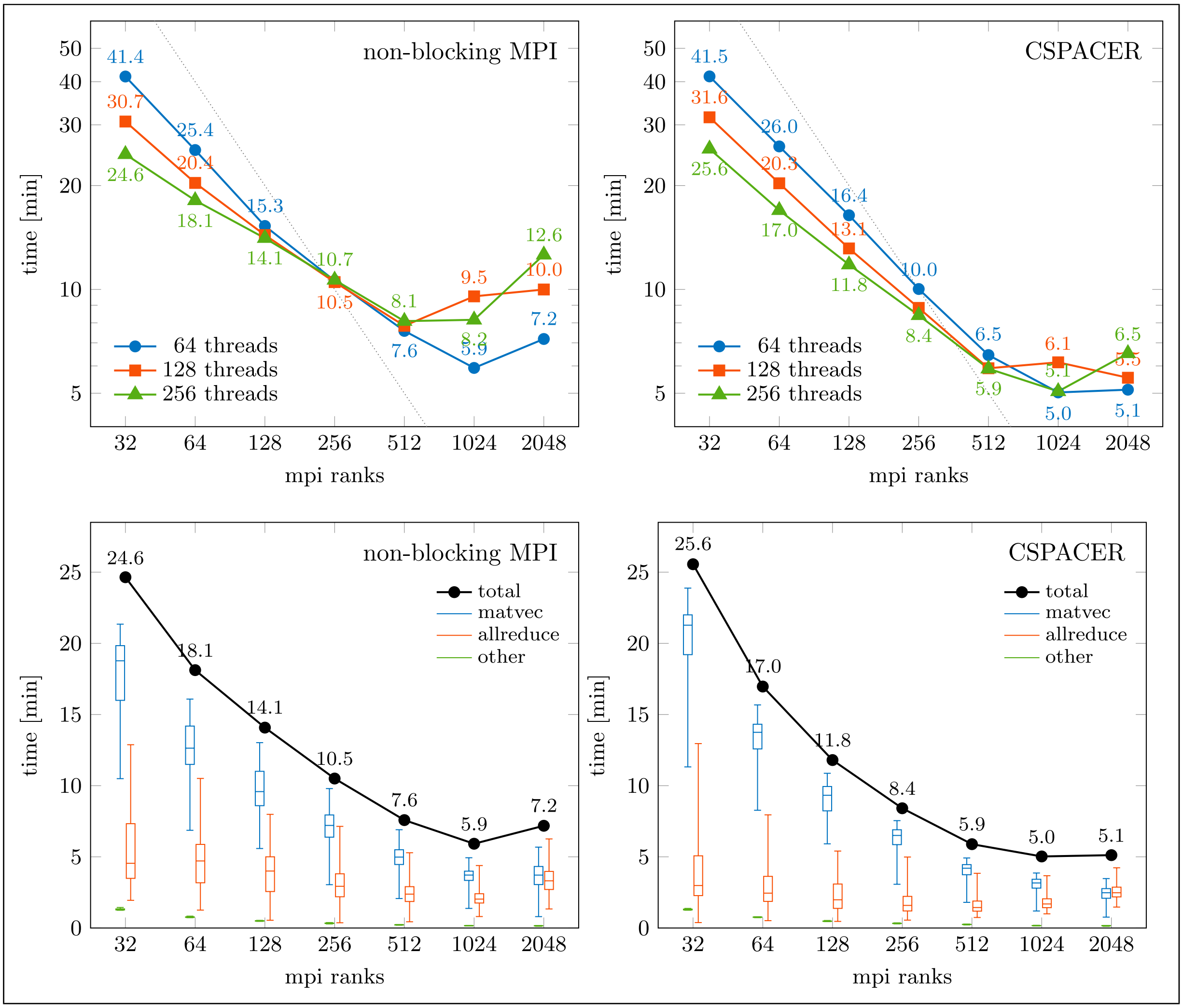

Figure 8. Strong scaling of 1 $L = 26$ LOBPCG iteration with 20,000 PCG iterations in single precision.

Table 3. $L = 26$ speedups for the CSPACER-variant compared to non-blocking MPI communication.

Ranks	256 threads (%)	128 threads (%)	64 threads (%)
32	−3.7	−2.9	−0.1
64	6.4	0.2	−2.4
128	16.2	8.5	−7.4
256	21.0	15.8	5.4
512	27.2	24.7	14.9
1024	38.0	35.6	15.1
2048	48.2	44.5	28.7

variability of the CSPACER-variant as an upper limit on the computation load imbalance.

Figure 8 shows the time decomposition and scaling behavior with various thread concurrency for the $L = 26$ problem. The corresponding speedup factors are given in Table 3. While for $L = 24$, we noticed performance advantage for the CSPACER-variant across all concurrency levels, for $L = 26$, the performance advantage starts at 128 nodes. This behavior is somewhat expected because the $L = 26$ problem is associated with a larger volume of data movement, making the performance less dependent on the runtime efficiency. In general, we notice similar trends for the two cases regarding the need to switch to lower thread concurrency as we scale and the correlation of the MATVEC and allreduce variabilities.

We notice that the optimal threading is problem dependent and is likely to change with the underlying systems. For $L = 24$, the advantage for reducing the thread concurrency manifests at 512 nodes, for $L = 26$, we need to change the thread-level at 1024 nodes. Currently, we rely on empirical measurement to identify the best configuration. Leveraging the iterative nature of the algorithm, we could dedicate few iterations for finding the optimal configuration. Ideally, we need to do such change automatically. We also notice that the optimal thread choice is dependent on the communication runtime. For both presented problem configurations, MPI tends to require switching to lower thread concurrency at lower node count compared to the CSPACER-variant. The implementation of the latter leverages threads to accelerate the communication progress, which is more advantageous when the number of neighboring ranks increase.

Finally, we compare the overall wall clock time for the $L = 26$ problems reported in [18, Table 3]. Using the combination of (i) graph partitioning for reducing the communication volume; (ii) runtime optimization via CSPACER; (iii) performing the MATVECs in the preconditioner only in single precision, we have been able to significantly increase the scalability of the matrix-free LOBPCG eigensolver to 1024 nodes. All these techniques together resulted in a speedup factor of 10× so that the overall wall clock time for computing eigenvalues/eigenvectors in the middle of the spectrum (30 LOBPCG iterations with 20,000 PCG iterations as preconditioner) got reduced from more than 1 day to only 2.5 h.

7. Conclusions

We have presented several strategies to significantly reduce the computation and communication imbalance within the matrix-free LOBPCG eigensolver for computing many eigenvalues and corresponding eigenvectors of large spin Hamiltonians. Using graph partitioning for reordering the states, both the total communication volume and the maximum communication volume per rank reduces and enhances the scalability of the matrix-free eigensolver. Although we have only used the METIS partitioning software in this work, it is possible to use other partitioning algorithms such as multi-level spectral partitioning methods (Barnard and Simon, 1994; Zhuzhunashvili and Knyazev, 2017), which can potentially further improve the scalability of the computations. By combining graph partition-based matrix reordering with communication performance optimization by using CSPACER, Consistent SPACE Runtime, we have been able to scale the LOBPCG eigensolver up to 512 and 1024 nodes for $L = 24$ and 26 spins, respectively. The numerical experiments have illustrated that the proposed techniques of graph partitioning, runtime optimization, and using mixed precision arithmetic, reduce the overall wall clock time for the $L = 26$ problem, reported in (Van Beeumen et al., 2020), by a factor of 10. Because the MBL study requires solving eigenvalue problems for many instances of Hamiltonians with random disorder terms, and computing eigenvalues from different regions of the spectrum, the overall computation can scale to hundreds of thousands of computational cores.

Acknowledgements

The authors are grateful to Pieter Ghysels for providing the memory footprints of the LU factorizations generated by STRUMPACK.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work is partially supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program and Center for Novel Pathways to Quantum Coherence in materials, an Energy Frontier Research Center funded by the US Department of Energy, Director, Office of Science, Office of Basic Energy Sciences under Contract No. DE-AC0205CH11231. GDKM was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program. This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

ORCID iD

Roel Van Beeumen https://orcid.org/0000-0003-2276-1153

References

Abanin DA, Bardarson JH, De Tomasi G, et al. (2021) Distinguishing localization from chaos: challenges in finite-size systems. *Annals of Physics* 427: 168415. DOI: 10.1016/j.aop.2021.168415.

Anderson PW (1958) Absence of diffusion in certain random lattices. *Physical Review* 109(5): 1492–1505. DOI: 10.1103/PhysRev.109.1492.

Barnard ST and Simon HD (1994) Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience* 6(2): 101–117. DOI: 10.1002/cpe.4330060203.

Bauer B and Nayak C (2013) Area laws in a many-body localized state and its implications for topological order. *Journal of Statistical Mechanics: Theory and Experiment* 2013(9): P09005. DOI: 10.1088/1742-5468/2013/09/p09005.

Bordia P, Lüschen H, Scherg S, et al. (2017) Probing slow relaxation and many-body localization in two-dimensional quasiperiodic systems. *Physical Review X* 7(4): 41047. DOI: 10.1103/PhysRevX.7.041047.

Choi J-y, Hild S, Zeiher J, et al. (2016) Exploring the many-body localization transition in two dimensions. *Science* 352(6293): 1547–1552. DOI: 10.1126/science.aaf8834.

Cuevas E, Feigel'man M, Ioffe L, et al. (2012) Level statistics of disordered spin-1/2 systems and materials with localized Cooper pairs. *Nature Communications* 3(1): 1128. DOI: 10.1038/ncomms2115.

De Roeck W, Huveneers F, Müller M, et al. (2016) Absence of many-body mobility edges. *Physical Review B* 93(1): 14203. DOI: 10.1103/PhysRevB.93.014203.

Duersch JA, Shao M, Yang C, et al. (2018) A robust and efficient implementation of LOBPCG. *SIAM Journal on Scientific Computing* 40(5): C655–C676. DOI: 10.1137/17M1129830.

Ghysels P, Li XS, Gorman C, et al. (2017) A robust parallel preconditioner for indefinite systems using hierarchical matrices and randomized sampling. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), Orlando, FL, 29 May–2 June 2017, pp. 897–906. USA: IEEE. DOI: 10.1109/IPDPS.2017.21.

Hilbrich T, Weber M, Protze J, et al. (2016) Runtime correctness analysis of MPI-3 nonblocking collectives. In: Proceedings of the 23rd European MPI Users' Group Meeting, pp. 188–197. DOI: 10.1145/2966884.2966906. EuroMPI.

Hoefler T, Dinan J, Thakur R, et al. (2015) Remote memory access programming in MPI-3. *ACM Transactions on Parallel Computing* 2(2): 1–26. DOI: 10.1145/2780584.

Ibrahim KZ (2019) Optimizing breadth-first search at scale using hardware-accelerated space consistency. In: IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC), Hyderabad, India, 17–20 December 2019, pp. 23–33. India: IEEE

Ibrahim KZ (2020) *CSPACER: Consistent SPACE Runtime*. URL https://bitbucket.org/kibrahim/cspacer_xc/.

Ibrahim KZ (2021) CSPACER: A reduced API set runtime for the space consistency model. In: The International Conference on High Performance Computing in Asia-Pacific Region, pp. 58–68. DOI: 10.1145/3432261.3432272. HPC Asia.

Johri S, Nandkishore R and Bhatt RN (2015) Many-body localization in imperfectly isolated quantum systems. *Physical Review Letters* 114(11): 117401. DOI: 10.1103/PhysRevLett.114.117401.

Karypis G and Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 20(1): 359–392. DOI: 10.1137/S1064827595287997.

Karypis G (2013) *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*. Version 5.1.0. Technical report. University of Minnesota.

Knyazev AV (2001) Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing* 23(2): 517–541. DOI: 10.1137/S1064827500366124.

Kohlert T, Scherg S, Li X, et al. (2019) Observation of many-body localization in a one-dimensional system with a single-particle mobility edge. *Physical Review Letters* 122(17): 170403. DOI: 10.1103/PhysRevLett.122.170403.

Luitz DJ, Laflorencie N and Alet F (2015) Many-body localization edge in the random-field Heisenberg chain. *Physical Review B* 91(8): 81103. DOI: 10.1103/PhysRevB.91.081103.

Lukin A, Rispoli M, Schittko R, et al. (2019) Probing entanglement in a many-body-localized system. *Science* 364(6437): 256–260. DOI: 10.1126/science.aau0818.

Oganesyan V and Huse DA (2007) Localization of interacting fermions at high temperature. *Physical Review B* 75(15): 155111. DOI: 10.1103/PhysRevB.75.155111.

Pal A and Huse DA (2010) Many-body localization phase transition. *Physical Review B* 82(17): 174411. DOI: 10.1103/PhysRevB.82.174411.

Pietracaprina F, Macé N, Luitz DJ, et al. (2018) Shift-invert diagonalization of large many-body localizing spin chains. *SciPost Physics* 5(5): 45. DOI: 10.21468/SciPostPhys.5.5.045.

Schreiber M, Hodgman SS, Bordia P, et al. (2015) Observation of many-body localization of interacting fermions in a quasirandom optical lattice. *Science* 349(6250): 842–845. DOI: 10.1126/science.aaa7432.

Smith J, Lee A, Richerme P, et al. (2016) Many-body localization in a quantum simulator with programmable random disorder. *Nature Physics* 12: 907–911. DOI: 10.1038/nphys3783.

Šuntajs J, Bonča J, Prosen T, et al. (2020) Quantum chaos challenges many-body localization. *Physical Review E* 102(6): 62144. DOI: 10.1103/PhysRevE.102.062144.1905.06345.

Tomov S, Langou J, Canning A, et al. (2005) Comparison of nonlinear conjugate-gradient methods for computing the electronic properties of nanostructure architectures. *Computational Science – ICCS* 3: 317–325. DOI: 10.1007/11428862_44.

Van Beeumen R, Kahanamoku-Meyer GD, Yao NY, et al. (2020) A scalable matrix-free iterative eigensolver for studying many-body localization. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, pp. 179–187. DOI: 10.1145/3368474.3368497. HPCAsia.

Wang LW and Zunger A (1994) Solving Schrödinger's equation around a desired energy: application to silicon quantum dots. *The Journal of Chemical Physics* 100(3): 2394–2397. DOI: 10.1063/1.466486.

Zhuzhunashvili D and Knyazev A (2017) Preconditioned spectral clustering for stochastic block partition streaming graph challenge (Preliminary version at arXiv.). In: 2017 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, 12–14 September 2017, pp. 1–6. DOI: 10.1109/HPEC.2017.8091045. USA. IEEE

Author biographies

Roel Van Beeumen is a Research Scientist in the Applied Mathematics and Computational Research Division at

Lawrence Berkeley National Laboratory. He received his PhD in Computer Science from KU Leuven, University of Leuven in 2015. His research interests include numerical linear algebra, high performance computing software development, and quantum computing. He is a member of SIAM.

Khaled Z. Ibrahim is a Computer Scientist at Lawrence Berkeley National Laboratory. He received his PhD in Computer Engineering from North Carolina State University in 2003. His research interests include programming models for high performance computing, runtime system designs, performance analysis and modeling, and scalable power-efficient architectures.

Gregory D. Kahanamoku-Meyer is a fifth-year PhD student in the Physics department of the University of California at Berkeley. His interests include both highly parallel software engineering for the solution of complex quantum many-body dynamics, and development of novel quantum algorithms that leverage cryptography to achieve provable advantage for quantum computers.

Norman Y. Yao is an Associate Professor of Physics at the University of California, Berkeley and a faculty scientist in the Materials Sciences Division at Lawrence Berkeley National Laboratory. He received his PhD in Physics from Harvard University in 2014. His research lies at the interface of atomic, molecular, and optical physics, condensed matter physics, and quantum information science. Current interests include the development of numerical methods for simulating quantum dynamics in non-equilibrium systems.

Chao Yang is a Senior Scientist in the Applied Mathematics and Computational Research Division at Lawrence Berkeley National Laboratory. He received his PhD from Rice University in 1998. He was a Householder fellow at the Oak Ridge National Laboratory from 1999 to 2000. He joined LBNL in 2000. His research interests include numerical linear algebra with applications in electronic structure calculations and quantum many-body problems, inverse problems, and high performance computing. He is a member of SIAM.