Master in High Performance Computing

Large-scale Exact Diagonalization of spin models with non-Abelian symmetries for the study of Many-Body Localization.

Supervisor(s):

Antonello Scardicchio,

Ivan Girotto

Candidate:

Rajat Kumar Panda

4th edition

2017–2018

Abstract

In this thesis, we developed a massively parallel exact-diagonalization application to study effects of the non-Abelian symmetries namely the $SU(2)$ symmetries on Many-body localization (MBL) in disordered 1D Heisenberg spin chains using the well known parallel libraries PETSc and SLEPc. Using the shift-invert spectral transformation technique, we study the eigenstates in the middle of the spectrum. We are able to look at the level statistics of the spin chain of sizes up to $L = 26$. Also, the code can calculate the Renyi entropy of the highly-excited eigenstates.

I. INTRODUCTION

In the presence of strong quenched disorder, the isolated many-body system may fail to thermalize and this is termed as many-body localization(MBL) [1–3]. In recent times this phenomenon has been of keen interest in the physics community to both theoretical [4–21] and experimental [22**?** –29] groups, especially with the experimental groups [2, 30, 31] being able to investigate these systems with high accuracy and precision[22].

From the study of the effect of $\mathbb{Z}_2$ symmetries [32–34] on MBL clearly indicates the that symmetries present in the system play a vital role in MBL phase. We are going to explore a disordered system with continuous non-Abelian symmetries, namely the $SU(2)$ through the exact diagonalization method.

In this system, we encounter many different components of quantum many-body physics, like strong disorder, out of equilibrium physics, interacting multi-particle system and highly excited eigenstates which make it really difficult to treat the problem analytically. Hence numerical simulations are extremely helpful to understand and explore the problems in MBL [20, 21].

But numerical simulations come with their own limitations, from the exponentially increasing Hilbert space to the eigenstate at infinite-temperature limit residing deep inside the eigenspectrum of the Hamiltonian which presents us with a formidable challenge. Another big limiting factor is the solver for the exact diagonalization. Because we are looking at the middle of the spectrum through the shift-invert method and we can not use any iterative solver like Lanczos algorithm as they fail to converge, only useful to find the eigenvalues at the boundaries of the the eigenspectrum[20]. In section III G, we elaborate more about the implementation of the diagonalization part. The study of disorder system requires us to average over many realization or instances and it's better to have more realization. Hence, the use of high-performance computing with massively parallel algorithms seems necessary to deal with these kinds of problems.

II. THE MODEL

The model we are interested to study is the disordered Heisenberg spin ($s = 1/2$) chain which has a continuous non-Abelian symmetry, namely the $SU(2)$ symmetry is described by the Hamiltonian

$$H = \sum_{i=1}^{L} J_i \mathbf{s}_i \cdot \mathbf{s}_{i+1} \tag{1}$$

where $s_i = (s_i^x, s_i^y, s_i^z)$ are the Pauli operators and the couplings J_i are drawn randomly from the probability distribution $P(J)$ given by

$$P(|J|) \propto |J|^{\alpha-1} \tag{2}$$

$P(J)$ is sampled by the function *Physics* :: *pow_law* taking α as the input, as shown in Fig.1 for $\alpha = 0.3$. The distribution takes different shapes for different values of α but the magnitude of the couplings always stays bounded by 1. Also, the value can be seen as a measure for the strength of the disorder, a smaller α would correspond to a higher disorder in the sense that the median ratio between two neighboring couplings goes as $2^{1/\alpha}$. The distribution coincides with the uniform box distribution between $[-1.0, 1.0]$ at $\alpha = 1.0$.

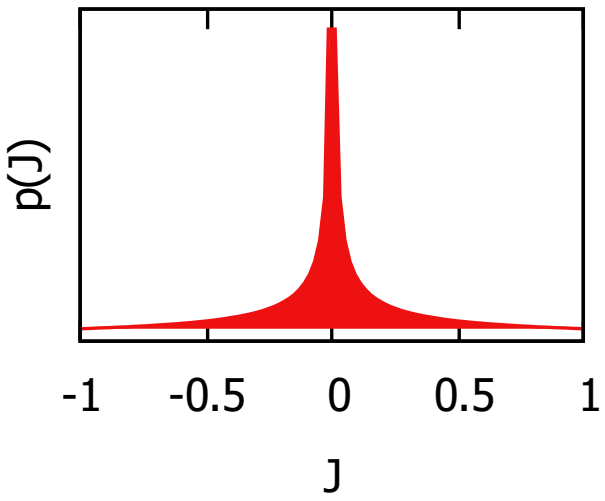

FIG. 1: The graph shows the distribution of $P(J)$ with $\alpha = 0.3$.

Starting with a Hamiltonian H for a L particle system with a SU(2)-symmetry, the many-body states arrange themselves into multiplets labeled by the total spin S and its z-projection M [35]. The states for this Hamiltonian can be uniquely represented as $|\psi\rangle = |S, M, f\rangle$. S

and M are considered as good quantum numbers here as $[S^2, H] = [S_z, H] = 0$ but aren't enough to span whole invariant subspace. We require the quantum number f to span the leftover degeneracy of the H-invariant subspace. Using the composition property

$$s_1 \otimes s_2 = |s_1 - s_2| \oplus |s_1 - s_2| + 1 \oplus \cdots \oplus |s_1 + s_2| \tag{3}$$

we can represent these SU(2)-covariant states in the form of a "fusion tree". These fusions are carried out by the help of the Clebsch-Gordan coefficients. In the Fig.2, the spins s(1,0) and s(1,1) fuse to give the spin s(0,0). The state is given by

$$\begin{aligned} |S, M, f\rangle = \sum_{m(1,0)+m(1,1)=M} & CG((s(1,0), m(1,0)) \oplus (s(1,1), m(1,1)) \to (S, M)) \\ & |s(1,0), m(1,0)\rangle |s(1,1), m(1,1)\rangle \end{aligned} \tag{4}$$

where $|s(1,0), m(1,0)\rangle$ and $|s(1,1), m(1,1)\rangle$ represent the multiplets at (1,0) and (1,1). By fixing the fusion pattern amounts to fixing the tree topology which decides the order of fusion. Labeling the topology with the appropriate spin value aggregates to the enumeration of the states. In our code we exploit the full symmetry of the problem which makes it tougher to implement than fixing the M quantum number only but it helps us in restricting the basis to a smaller dimension.

III. IMPLEMENTATION

A. Data Representation and Defination

In regard of our previous considerations, a state $|S, M, f\rangle$, will be interpreted as a synonym of a tree. It was convenient for us to implement where the tree topology is fixed. We selected perfectly balanced trees because it allows to build the trees recursively. A perfectly balanced means every leaf is equidistant from the root where leaves are the nodes without any children node in lower level. This forces us to have trees that have leaf numbers only the powers of 2. For system size non-power of 2, the tree is padded with ghost zero spins. As zeros are the trivial representations of the algebra they don't change any physics. A tree is represented as an array or standard vector of integers of length $L-2$ where L is the length of the spin chain. We do not store the root which represented by the total spin of the system and same for all the trees, neither the last layer as it always spin ($s = 1/2$) which reduces the representation by half. Here in Fig.2, an example of the tree structure is given. The number of layers in the tree will be called height and denoted as K. The tree structure branching below from each node will called the subtree of that node. A tree will be called short if the height of the tree is 2, and trivial if it is 1.

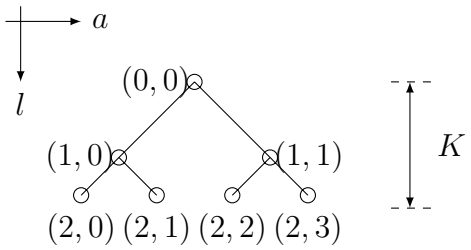

FIG. 2: An example of a tree.

Now to read the tree correctly, we will adopt a coordinate system of $s(l, a)$, with l being the layer and a being the column in the layer to encode the spin data at each node of the tree. The root of a tree will always be denoted as $s(0, 0)$ as given in Fig.2. A tree is valid if the spin data satisfies the selection rule

$$|s(l+1, 2a) - s(l+1, 2a+1)| \leq s(l, a) \leq s(l+1, 2a) + s(l+1, 2a+1) \tag{5}$$

for all the spins in every layer of the tree. The fusion rule in above equation determines the allowed values of the spins at each node. By filling the nodes of the tree with all the allowed spin values, the tree basis is built. The dimension of the tree basis, that is to say, the number of valid trees that can be generated for the system size for L and a fixed value

S of the total spin $s(0,0)$ and a fixed value M of the total spin's z-projection, is given by

$$\dim(L,S) \equiv \frac{2(2S+1)}{(L+2S+2)}\binom{L}{L/2-S} \sim \frac{2^L}{L^{3/2}} \tag{6}$$

This number in Eq.6 goes to the Catalan Number for $S=0$

$$\dim(L,0) = \frac{1}{\frac{L}{2}+1}\binom{L}{L/2} = C_{L/2} \tag{7}$$

We consider SU(2)-symmetric operators of the form

$$\langle i,j\rangle \equiv \mathbf{s}_i \cdot \mathbf{s}_j \tag{8}$$

where i and j represent the ith and jth spin in the spin chain and without loss of generality we can assume that the $i<j$ and if $i=j$ then the operator acts trivially on the states like in equation below

$$\mathbf{s}_i^2|state\rangle = \frac{3}{4}|state\rangle \tag{9}$$

From [35] we know how to compute the brackets of the operator $\langle 0, L-1\rangle$ on perfectly-balanced binary tree states of length L. This formula can be easily generalized to any i and j, which allows us to construct the $\langle i,j\rangle$ operators in the tree basis element-by-element.

B. External Libraries Used

The implementation of the code depends on the external libraries which are PETSc-3.9.3 [36–38], SLEPc-3.9.2 [39, 40] and MUMPS [41, 42]. PETSc is used for the parallel sparse matrix and vector representation and manipulation, matrix-vector operation. SLEPc provides the interface for the large sparse eigenvalue problems on parallel computers. The full-form of MUMPS, which is MUltifrontal Massively Parallel sparse direct Solver, tells us what is the use of the library. MUMPS can solve through both Cholesky and LU-factorization on both parallel and sequential sparse matrices. During the installation of the PETSc and compilation of the code, it was taken care of if proper flags for optimization was used or not. We have developed the code using the c++ standard 2011. The code was compiled with the -std=c++11 flag, and the -O3 optimization flag. Also, the cluster provided modules were used for the installation. For the linear algebra operations the Intel Math Kernel Library (MKL) 2017 was used interfacing with PETSc.

C. Work-flow

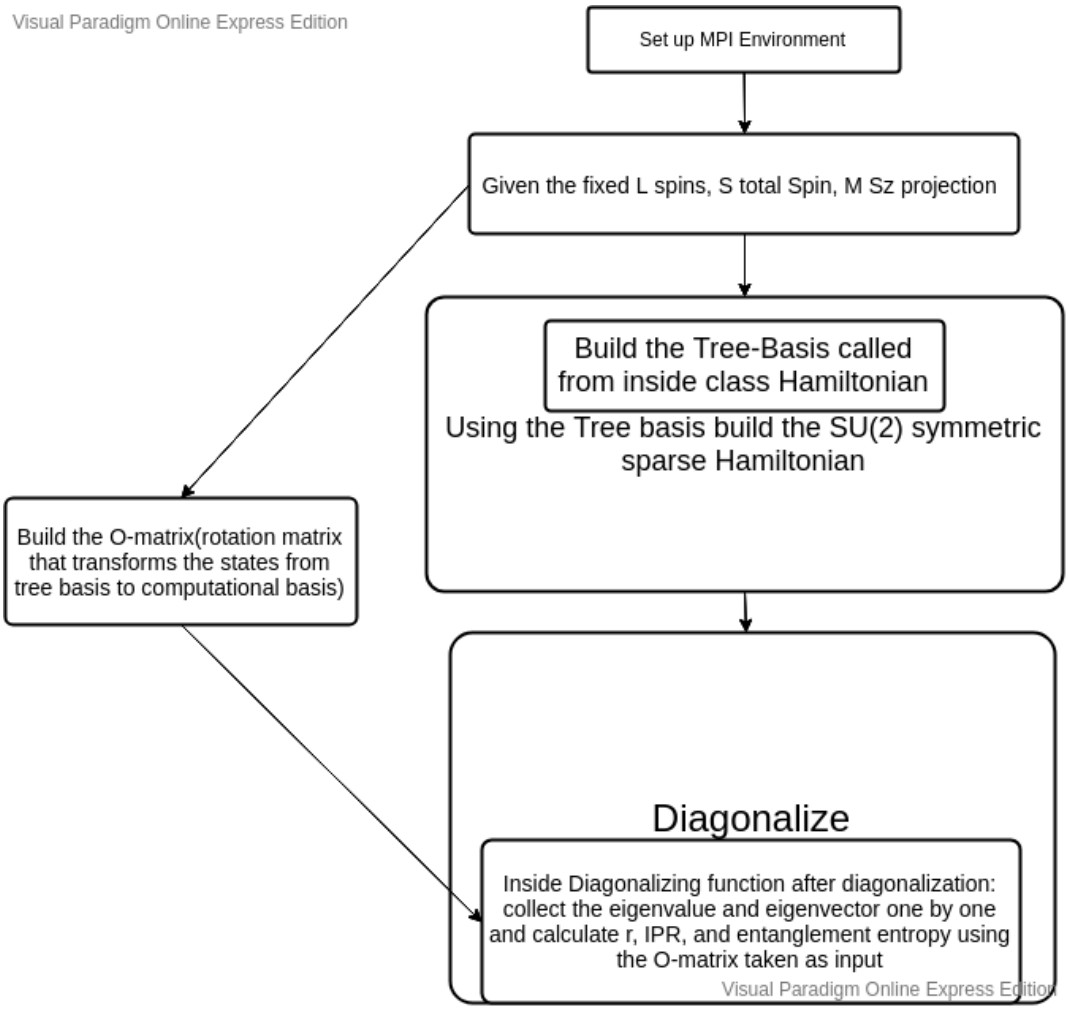

FIG. 3: The graph above shows the workflow of the code.[43]

D. Setting Up the MPI-environment

In the code, we begin with the *SetEnv* class, takes *argc* and *argv* as the input variable which calls the *SlepcInitialize*() setting up the MPI(Message Passing Interface) environment with the collective call on $PETSC\_COMM\_WORLD$ for the parallelization. *SlepcInitialize*() initializes the SLEPc library and calls the *PetscInitialize*() if it hasn't been called before.

E. Build Tree Basis

The next step is to generate the tree-basis for the construction of the Hamiltonian operator from the Eq.1. Given the total spin, magnetization and the length of the one-dimensional quantum system, class *Basis* creates the tree-basis through the function *Basis* :: *build_basis* which works as a wrapper for 3 helper functions namely *loop_function*(..), *for_loop_recurse*(..) and, *allowed_combo*(..). The function *allowed_combo*(..) generates the *std* :: *vector* that has been filled with all the allowed pairs of children node for the lower level. The allowed children are decided by selection rule in Eq.5.

In Eq.5 $s(l, a)$ is the parent node at the lth level and ath column. There are two repeating parts in the filling of a single tree which is carried out separately and recursively through. In function *Basis* :: *loop_function*(..), it both creates the allowed children nodes dynamically through *Basis* :: *allowed_combo*(..) functions and fills in the tree. But the $Basis :: for\_loop\_recurse(..)$ function only fills the tree at different levels. For the system length of non-power of 2, the *allowed_combo*(..) allocates zeros at the end for the padding in the tree state.

The functions *Basis* :: *loop_function*(..) and *Basis* :: *for_loop_recurse*(..) both take a reference to an integer as an input called *counter* which keeps ticking with each tree added to the basis vector. In the code, the key to parallelization is decided on building the part of the basis on each process. There is no way that the basis generation could be parallelized, other than each process start generating the trees for the basis but saves only part of it which belongs to the process which is indicated through the input parameters start and end. The generation of the basis takes a very small amount of time as shown in Fig.4.

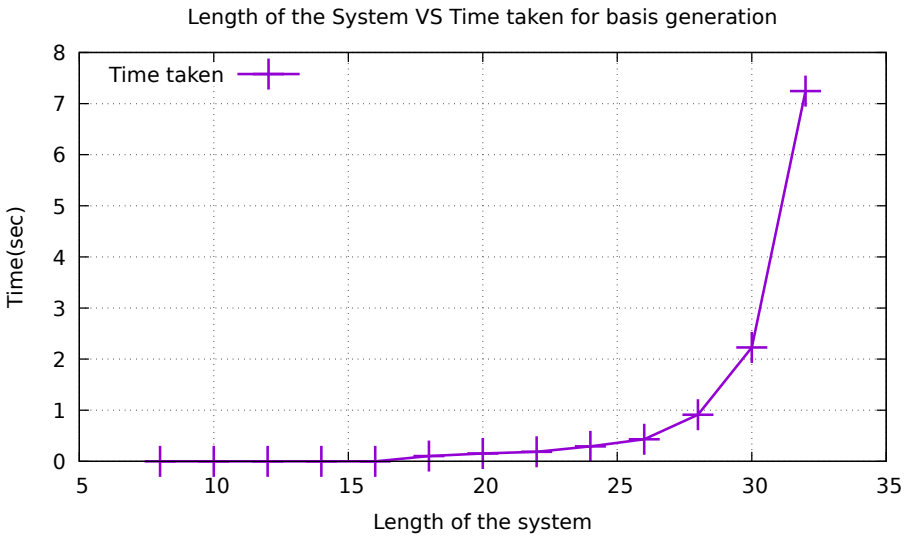

FIG. 4: The graph shows the time taken for the code to generate the basis of different system sizes.

F. Build Hamiltonian

The construction of the Hamiltonian operator of the system is dependent on the basis generation, it is also distributed over the processes as the tree-basis. For the Hamiltonian elements, each tree needs to be bracketed with all the other trees through $\langle bra|\langle i,j\rangle|ket\rangle$ for all i's. But as our matrix is sparse most of the operation spits out zeros. To save ourselves from the extra computation the function $Basis :: connected\_trees(..)$ returns the list of all the trees that may give a nonzero matrix element for the operator $\langle i,j\rangle$ for a given tree-state and the brackets are performed only on the connected tree list. But to fill in the right element in the Hamiltonian the tree-state need to be indexed, so that given two connected tree states the matrix element it belongs to can be identified and stored at the right place. The trees generated in the basis are sorted from smallest to largest when represented as a pair of long long integers and specialized binary search function is used to find the right index for the given tree. As the basis creation is distributed among the processes, the uniques keys for the trees need to communicated to all the processes. This is done by $MPI\_Allgatherv$, which stores the keys of the basis as the trees in the basis across the processes. This is the only communication that happens during the construction of the matrix. After this, each process allocates, computes, and holds the rows under it's ownership-range of the matrix, which is

done locally.

In the code, PETSc's Mat object used for the Hamiltonian matrix of the type *MATMPIAIJ*, a sparse parallel matrix in AIJ format. The preallocation is a must and also recommended PETSc [36, 38] which could be as high as 50 times faster than the non-preallocated version. Indeed we do observe that the preallocated version is an order faster. After preallocation, the code loops over the tree-states on the local basis for computing the Hamiltonian element and sets the value at the appropriate position if it is non-zero.

The binary search has the complexity of the order $O(\log N)$ and takes the advantage of the fact that the generated basis is sorted and easier to convert into a lookup table. This does act as an overhead but saves almost 95 percent of time overall with respect to calculating each tree-state over all the tree states one by one finding all the non-zeros. In the older implementation we had the immature way of calculating the matrix elements which used to be the most expensive part of the code taking more than 90 percent of the total computational time as given in Fig.5.

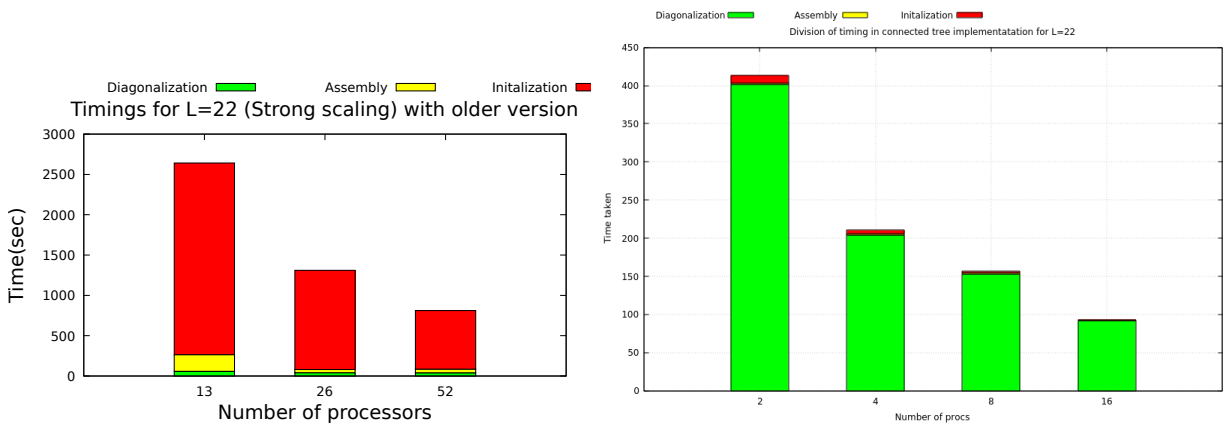

FIG. 5: This is timings of the older version with calculating each Hamiltonian elements and discarding the zeros for $L = 22$ and the new version of the implementation with connected trees. The scaling achieved by increasing the number of processes in the second version solely depends on the efficiency in parallelization of the linear solver used, in our case MUMPS.

G. Diagonalization

After building the Hamiltonian matrix, the next step is to find the eigenvalues and eigenvectors at the middle of the spectrum. This is achieved by applying the shift-invert technique

from the spectral transformation of SLEPc library. This method enables us to find the interior eigenvalues in a particular neighborhood[20, 39, 40], in our case at the middle of the spectrum. For any given $\sigma \in [E_{max}, E_{min}]$, instead of solving $Ax = \lambda x$ the solver converts the problem to

$$(A - I\sigma)^{-1}x = \theta x \tag{10}$$

Where θs are the transformed eigenvalues with the largest magnitude that are close to the target value σ which correspond the original eigenvalue λ and θ by

$$\theta = \frac{1}{\lambda - \sigma} \tag{11}$$

Once the θ values are found they are transformed back to the original λ values through $\lambda = \sigma + 1/\theta$. At the middle of the eigenspectrum, the eigenvalues are very close to each other, as seen in the graph 6.

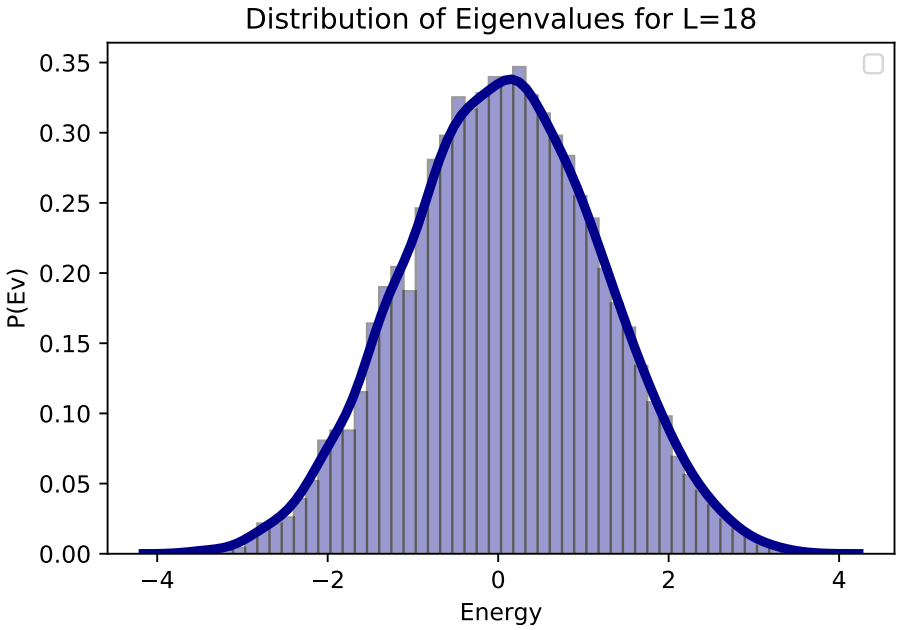

FIG. 6: Showing the eigenvalue distribution of system size $L = 18$. We can observe that at the middle of the spectrum eigenvalues are densely populated and decays towards the ends.

In our case, after the transformation, the biggest bottleneck is to find the inverse of the shifted Hamiltonian. One could argue that at this point the linear equation can be solved through iterative solvers but unfortunately, the iterative methods do not converge. The shifted Hamiltonian extremely ill-conditioned for large system sizes as the condition

number scales $\kappa \propto O(L\exp(L))$ [20]. At this point, the only viable option available for to us is the rather stable, full or partial Gaussian elimination. The shifted Hamiltonian is LU-decomposed into lower and upper diagonal matrices after reordering with a permutation matrix[20].

In the present context, we have a parallel sparse matrix. We can use the linear solvers provided by PETSc for this task. But the direct solver provided by PETSc is not yet implemented in parallel, though PETSc provides the interface to use many other external massively parallel sparse linear solver. In our case, we tried some of the solvers for LU decomposition, like MUMPS [41, 42], SUPERLU_DIST [44] and STRUMPACK [45]. In our experience the MUMPS turned out to be the faster solver though it took around $5 \sim 10$ percent more memory than STRUMPACK. The Cholesky decomposition by MUMPS required more memory than its LU-decomposition implementation. In this project here onwards, the solver used for the diagonalization, is always MUMPS.

MUMPS is a package for solving systems of linear equations through LU decomposition for the sparse matrix *aij* type and through the Cholesky method for symmetric block sparse matrices *sbaij* type of the form $Ax = b$. The A sparse matrix can be either asymmetric, symmetric positive definite, or general symmetric, on distributed memory computers. MUMPS is built upon the libraries like MPI for message passing and BLAS, LAPACK, BLACS, and ScaLAPACK like cluster linear algebra libraries for the matrix vector operation during the diagonalization and requires them to be installed during the installation[41, 42]. It is strongly recommended to install PORD, SCOTCH or Metis for the ordering algorithms. MUMPS distributes the task during factorization but a the master-slave approach is adopted during the analysis phase and to collect solution. MUMPS support both real and complex calculations for single and double precision.

After all the optimization, diagonalization remains the biggest time consumer. But the biggest limiting factor comes with the memory consumption during the diagonalization which blows up with the dimension of the Hamiltonian reaches the order of TeraBytes for $L = 26$ shown in Fig.7. During the diagonalization, the distribution of the memory was another big problem. MUMPS handles the distribution of memory well but the difference between the maximum and the average still could be as high as $\sim$10-15 percent. That increased the chances of the node to go out of memory and crash. It was taken care by finding the right combination cores per node used making sure there is a padding of memory which can

accommodate these fluctuations.

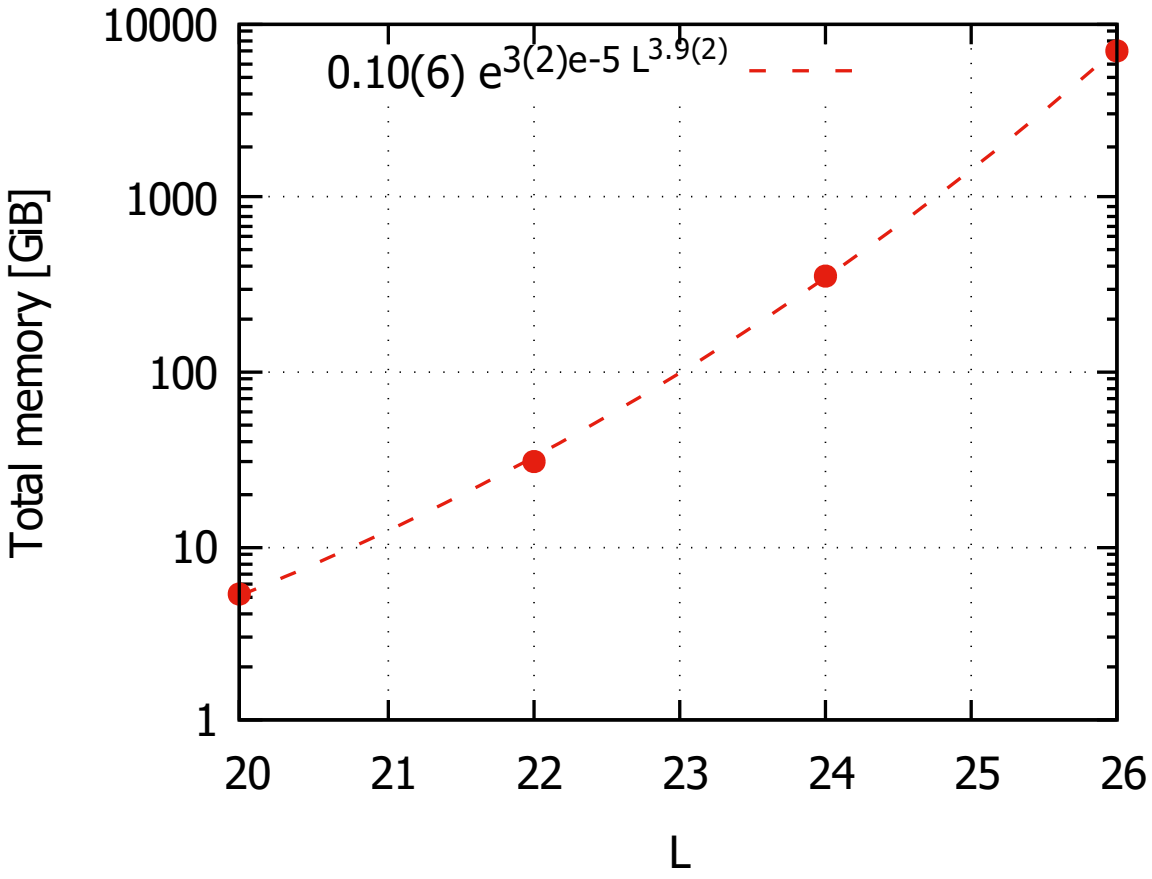

FIG. 7: The maximum amount of memory utilize by the program during the entire run is plotted with varying system size. The memory occupation peaks during the LU-factorization by the linear solver MUMPS.

During this work, we tried to find a possible iterative solver with an appropriate preconditioner. There were a few which successfully converged for smaller systems but failed for large system sizes. The bcgs or BiCGSTAB(biconjugate gradient stabilized) implementation provided by PETSc managed to converge for system sizes up to a few thousand. But the number of iteration increased with the dimension of the Hamiltonian and were of the same order of the Hamiltonian dimension which made it impossible to use as a solver. The number of iterations also varied with the change in the number of processes. We tried different preconditioner for this solver but none of them were useful.

H. Physical Quantities

1. *Inverse Participation Ratio*

MBL phases can be recognized and studied in two ways either eigenstate properties or dynamical properties. In dynamical properties, one could study the non-thermal relaxation of local observables. But in our case, we are going to focus on eigenstate properties like inverse-participation ratio(IPR), level statistics. After diagonalization of the Hamiltonian, the eigenvectors of all the 50 states at the middle of the spectrum are collected. Given a state $|\Psi\rangle = \sum_a \Psi_a |a\rangle$ where $|a\rangle$ is a state and $\{|a\rangle\}\_a$ is a basis of the Hilbert space, the IPR is defined as

$$\text{IPR} = \sum_a |\Psi|^4 \tag{12}$$

In fully ergodic systems one expects the infinite-temperature eigenstates to be "random", so they should mix with all the basis states equally, this implies $IPR \sim 1/d_L$ where d_L is the dimension of the Hilbert space. That's because if IPR decays more slowly, this excludes full ergodicity as the eigenstate mixes only with a certain sector of the Hilbert space.

2. *Level Statistics*

Another quantity that is a good signal of the presence of localization is the level spacings[6] which is the difference between the adjacent eigenvalues δ_n.

$$r \equiv \frac{\min(\delta_n, \delta_{n+1})}{\max(\delta_n, \delta_{n+1})}, \tag{13}$$

with δ_n and δ_{n+1} being consecutive level spacings in the spectrum. This value is averaged eigenstates and realizations of disorder which gives us the r-parameter. The distribution of level statistics in ergodic phase is Gaussian-orthogonal ensemble (GOE) and the average value of r converges to $[r] \cong 0.53$ for large system size but for the localized phase of system the level statistics turns out to be Poisson and $[r] \cong 0.39$[6].

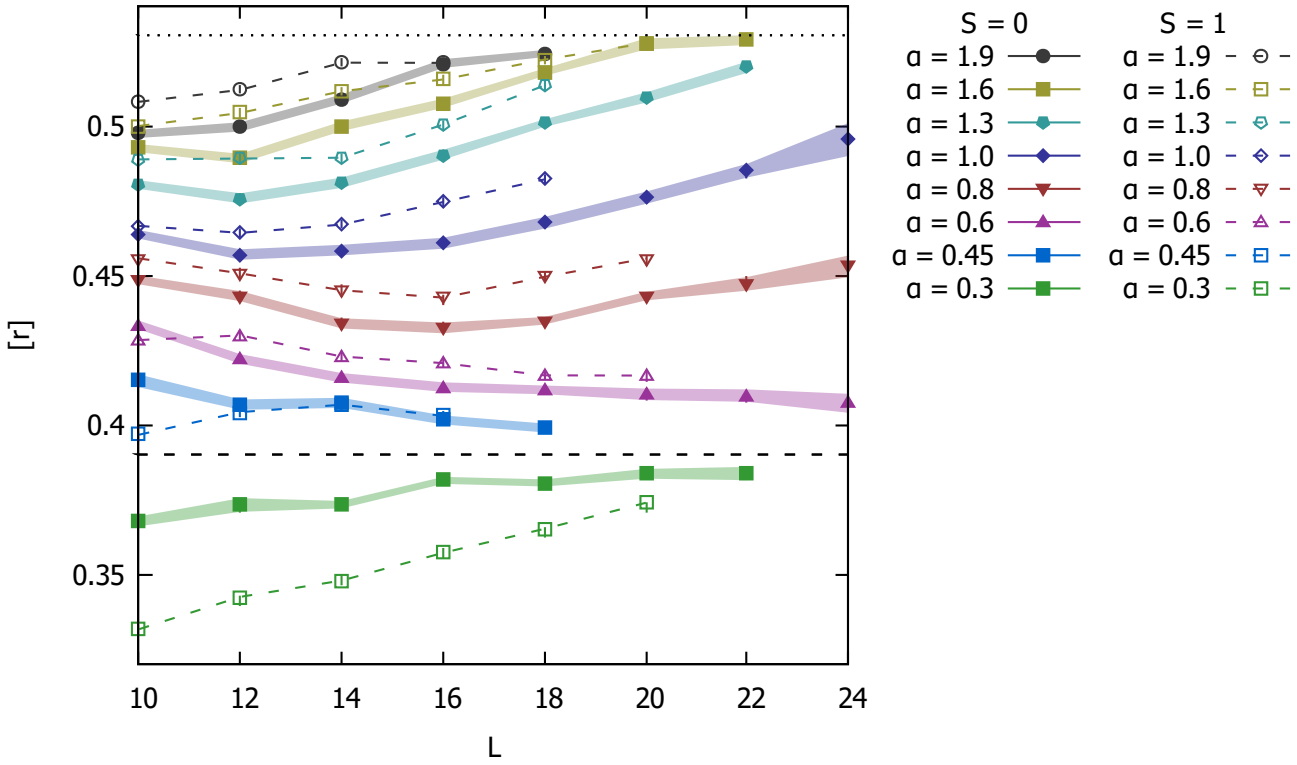

FIG. 8: The average of r-parameter as a function of L is shown for different values of α over 100s(most have 1000) of disorder realization for both S=0 and 1. We calculate the average of r for each realization of 50 eigenstates at the middle of the spectrum then average them over realizations. The dashed lines at $r = 0.53$ and $r = 0.39$ represent respectively the GOE and Poisson values[6]. The flattened bands show the standard error for the $S = 0$ values.

In Fig.8, the r-parameter for α value 1.6 at $L = 20$ reaching the ergodic limit. It seems with the current trend for $\alpha = 1.3$ will reach the ergodic limit at $L = 24$. It is evident that the systems with high α vales are inching towards the ergodic limit. In the mid range of α with the current trend the system would attain ergodicity. We predict that all the system will move towards the ergodic limit for larger system sizes.

3. Entanglement Entropy

To calculate the reduced density matrix in the computational basis we need to transform the eigenstates in the tree-basis to the computational basis which requires the appropriate basis-change matrix which we will refer to as the "O-matrix". The O-matrix is a sparse rectangular matrix whose elements are calculated by the function *Physics* :: *overlapping*(..) by calculating the $\langle s_i|tree\rangle$ using the function *ClebschGordan*(..), which calculates the Clebsch–Gordan coefficients where $\langle s_i|$ is a canonical basis state. These states are encoded with $\{0, 1\}$ instead of $\{-0.5, 0.5\}$. The O-matrix is larger than the Hamiltonian as the dimension of the O-matrix is the size of the tree basis time 2^L. Because there is no known way of computing only the non-zero values, the code calculates all the elements and discard the zeros which makes it the most expensive part of the whole code. The building of the O-matrix only depends on (L,S,M), just done once. To avoid calculating the rotation matrix each time, we store the matrix once it is built through the function *MyMatDump*() and read it once at the beginning of the run and keep in the memory during the entire run through all the disorder realizations.

After the rotation matrix is built and the eigenstate is obtained from the diagonalization, we perform the matrix-vector multiplication which results in a vector of size 2^L. This vector is distributed over all the processes. Using $MPI\_Allgatherv$ the distributed vector is collected locally on each process and then reshaped to create the distributed block diagonal square matrix of size $2^{L/2} \times 2^{L/2}$. The multiplication of the transpose of the matrix with itself gives us the reduced density matrix. The reduced density matrix is fully diagonalized using the Krylov-Schur subspaces method. The diagonalization of ρ_A(also can be written as $\rho_{L/2}$ in our case), the reduced density matrix is much easier, as the size of these matrices are very small. The entanglement entropy is given by

$$\mathcal{S}(\rho_A) = -\operatorname{Tr}[\rho_A \log \rho_A] = -\operatorname{Tr}[\rho_B \log \rho_B] = \mathcal{S}(\rho_B) \tag{14}$$

where ρ_A is the reduced density matrix. The calculation of the entropy is given by

$$S(\rho_A) = -\sum_k \lambda_k \log_2 \lambda_k \tag{15}$$

where λ_ks are the eigenvalues of the spectrum of ρ_A.

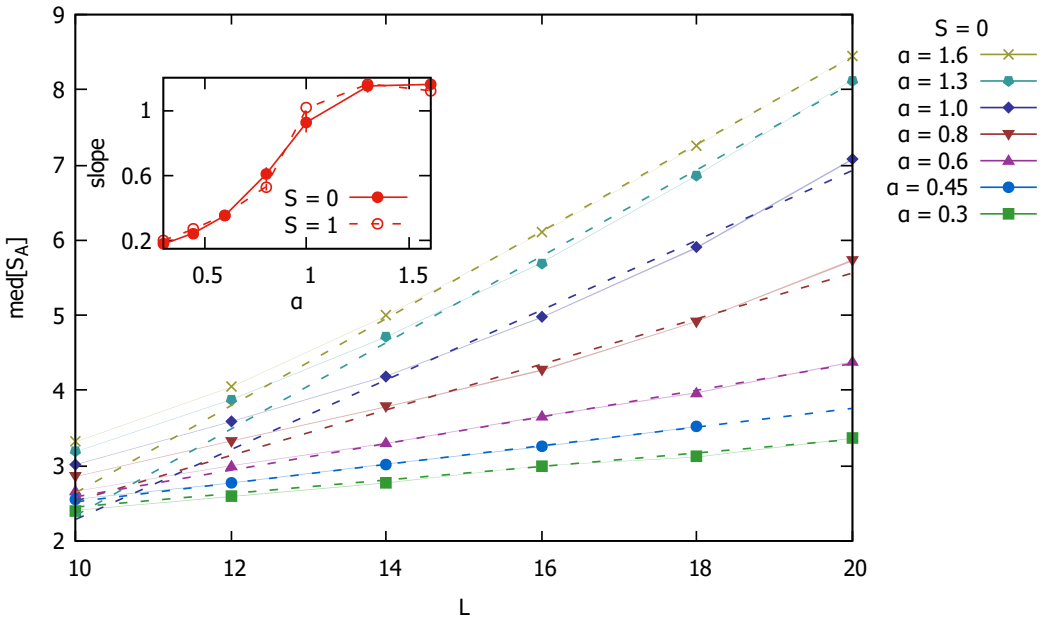

FIG. 9: For different α values the median S_A for different system size has been plotted with 1000 disordered realization for each combination having fixed $S = 0$ value.The dash-lines show the linear-fits. In the inset, it shows the slopes obtained from the linear-fits.

In the Fig.9, for higher α values the slope is one or higher(within numerical error), indicates that the systems have fully thermalized. The reduced density matrices are maximally mixed which results in the maximum entropy.

IV. PERFORMANCE ANALYSIS

A. Computational Tools

1. *System Architecture- Knights Landing(KNL) and Skylake(SKL)*

We have run our code on two different partitions namely Marconi A2 running Knights Landing(KNL) and Maconi A3 running on Skylake(SKL) at the supercomputing center CINECA Marconi which holds the $19th$ position in the list of Top500 supercomputer according to the Nov. 2018 edition [46]. The detailed comparison of the two architectures is given in the table I. The external libraries were compiled separately for the different architectures to accommodate all the features of the systems.

	Marconi A2 (KNL)	**Marconi A3 (SKL)**
Node Configuration	68-core Intel Xeon Phi 7250 CPU (Knights Landing) at 1.40 GHz	2*24-core Intel Xeon 8160 CPU Skylake @ 2.10GHz
Cores	68 cores/node; 244,800 cores total	48 core/node,72.576 + 38.016 cores in total
Instruction Set Extensions	AVX-512	AVX-512
RAM	16 GB/node of MCDRAM and 96 GB/node of DDR4	192 GB DDR4 RAM
Max Memory Bandwidth	115.2 GB/s	119.21 GiB/s

TABLE I: The table contains the details of the specifications of KNL and SKL architecture[47]. Both of the clusters run on a network of Intel OmniPath (100Gb/s) high-performance network.[48]

After running on both the partition the SKL seems to be $3.5 - 4.5$ times faster then KNL as expected but the strong scaling is higher in KNL than SKL. For our code, as it is highly memory intensive the SKL architecture is better suited because of its high memory per process ratio. While using KNL, to fit our problem in the provided memory we had to use a lot of nodes but not able to use more than half of the cores as there is an increment in the memory required with the number of core used per node. So the consumption of

total computational hours in KNL becomes many folds that of SKL. This fact matters a lot when we are planning to run as many disorder realization as possible provided with limited computational hours.

Number of cores	Timetaken on KNL (sec)	TIme taken on SKL (sec)
4	837.437542	204.260246
8	537.101202	153.41781
16	375.398813	91.673767
24	383.196548	78.197508
32	325.546695	77.147023
48	279.050299	81.87931

TABLE II: This is a one to one comparison of KNL and SKL performance for $L = 22$.

B. Scaling

1. Strong Scaling

In this section, we will analyze the performances and timings that was obtained during the benchmarking of the code. In our context process always represent a single processing unit. Strong scaling is defined as the variation of the time taken to complete the task with the number of parallel processes while keeping the problem size constant. In Fig.10, we report the intra-node strong scaling where the problem size fits inside a single node and in Fig.13, the inter-node strong scaling which involves communication through Intel OmniPath network between the node. The code spends most of its time diagonalizing the matrix. The scaling times are mere reflections of scaling of the linear solver MUMPS. The timing of the strong scaling is presented in the Fig.10. We can see in both the cases, the parallelization starts saturating and we don't gain any speed up even if we increase the number of processes further. Rather, we notice that the program takes more time than before for a higher number of processes, at this point the communication overtakes the computational gain we got through parallelization. In SKL the communication starts dominating earlier than KNL. The strong scalability plot the reflects the same observations11. But one thing to notices

that when SKL strong scalability is already saturated and on a downward trend, but the KNL scalability is still growing.

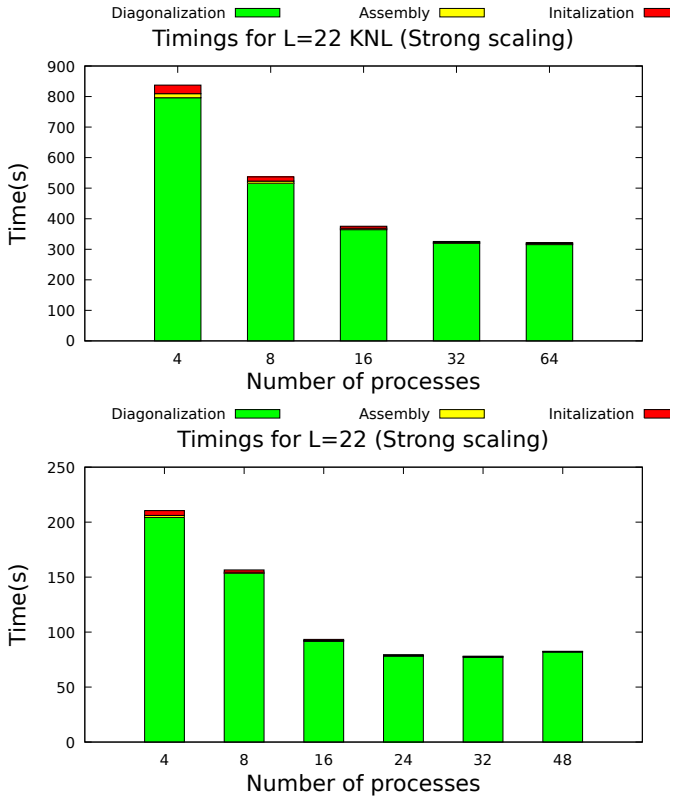

FIG. 10: Timing of strong scaling benchmarking of KNL and SKL architecture for $L = 22$ with varying processing units.

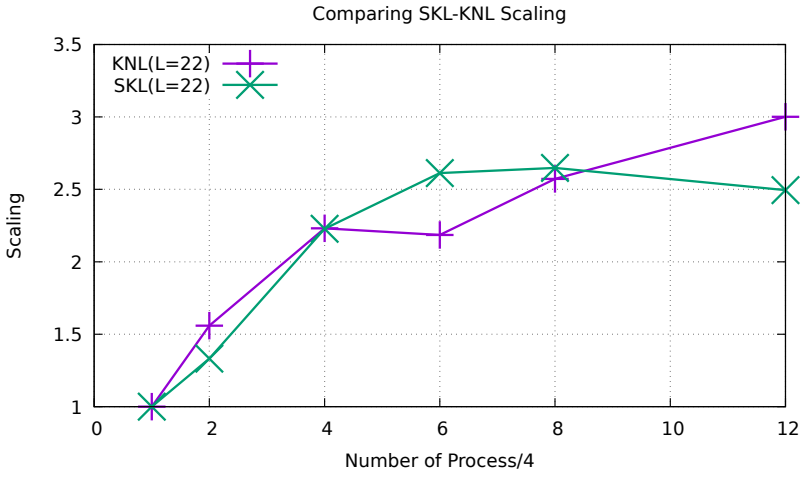

FIG. 11: Strong scalability comparison of the KNL and SKL architecture. In x-axis for the number of processes are divided as we take 4 processes as a single unit in this context.

There is one more thing to notice about the parallelization is that the there is a big overhead in setting up the parallelization of MUMPS[41, 42]. Also, the efficiency of the parallelization depends on how big is the task. With a higher workload, MUMPS shows better parallelization and it takes more processes to reach saturation. In Fig.12 we can observe that

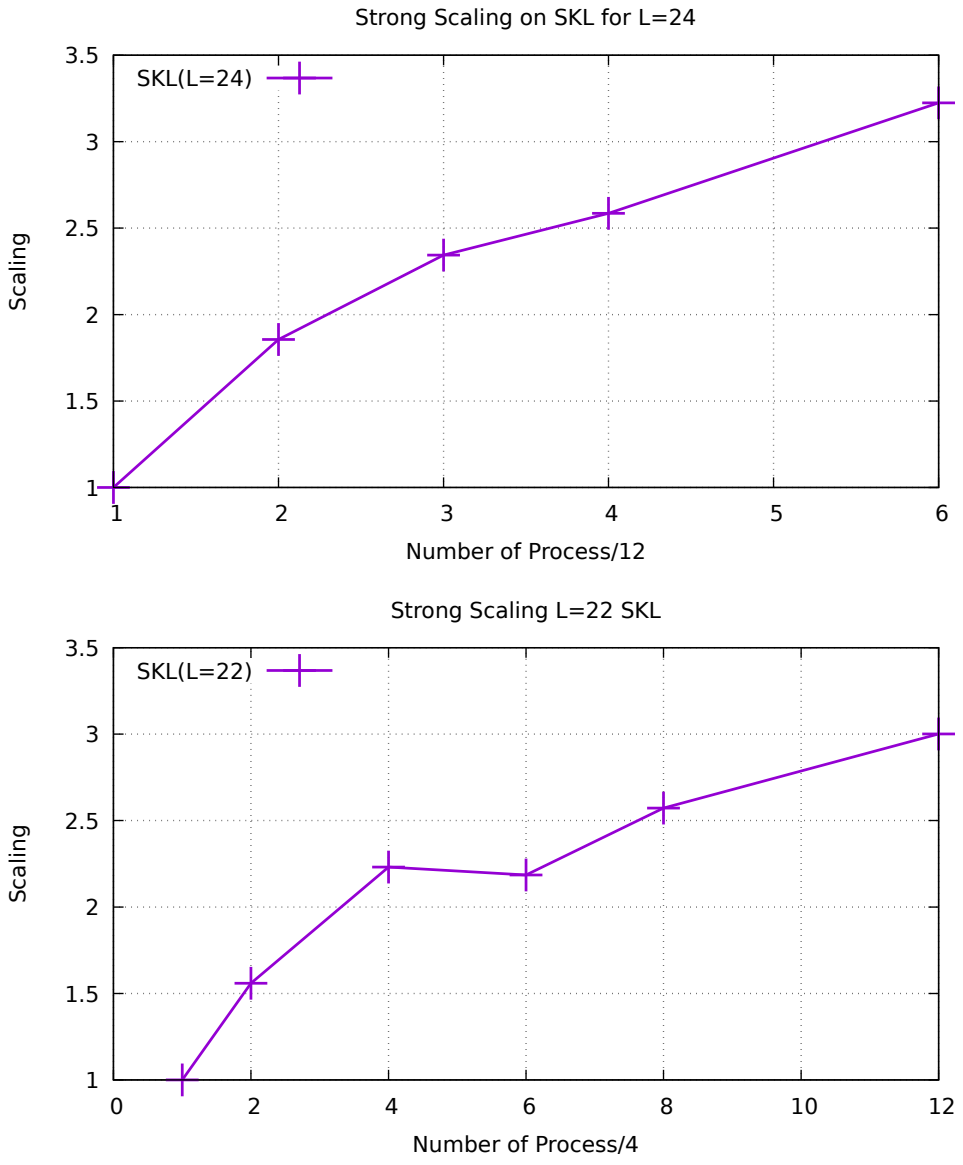

FIG. 12: In this figure, the scalability for $L = 24$ can be seen to be much better than $L = 22$ on SKL, because of the higher workload for $L = 24$ the saturation in scalability occurs much later.

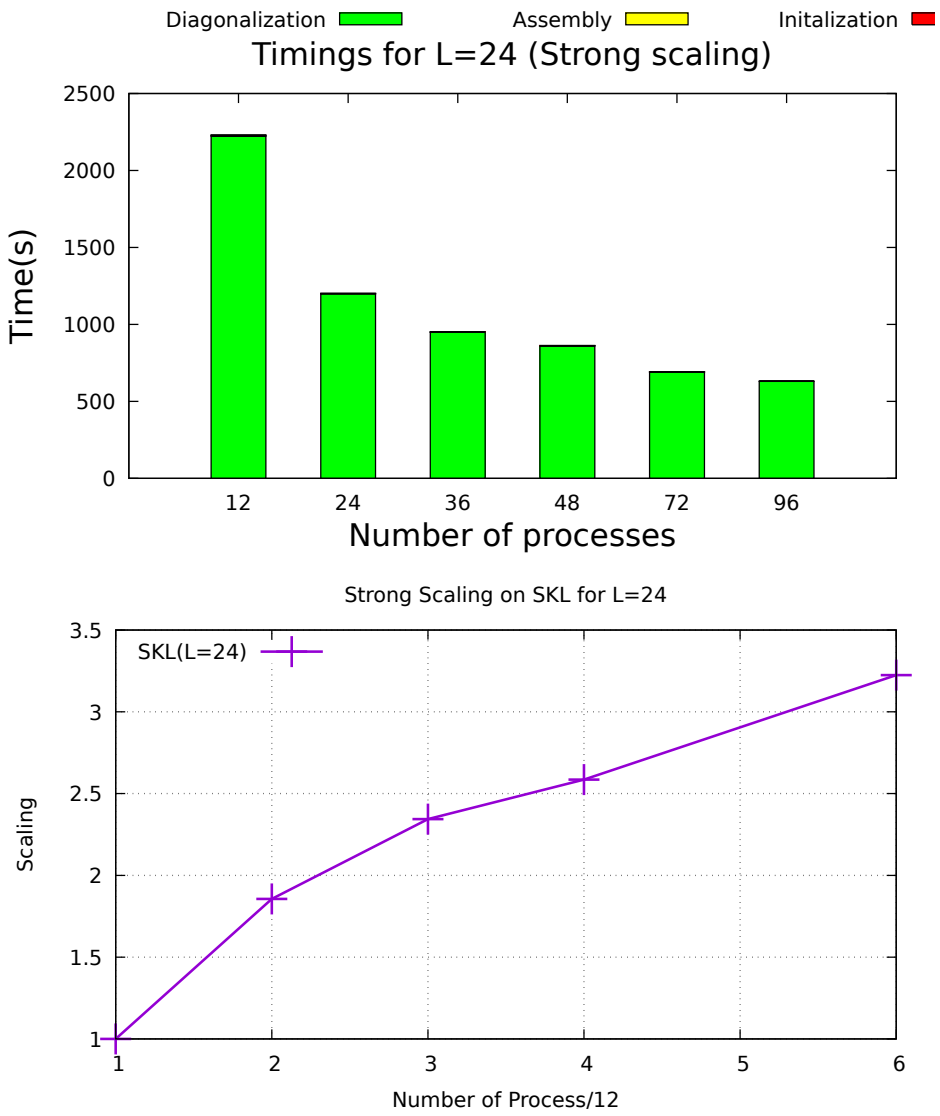

FIG. 13: Inter-node strong scaling on skylake architecture. In the presented graph the represents the timings when the processes are scattered in 3 different nodes for each of the run.

In all the scaling and performance analysis above, the O-matrix was always read from a file. Below we report the scaling of the code including the building of O-matrix. The building of O-matrix takes more than 95 percent of the total runtime.

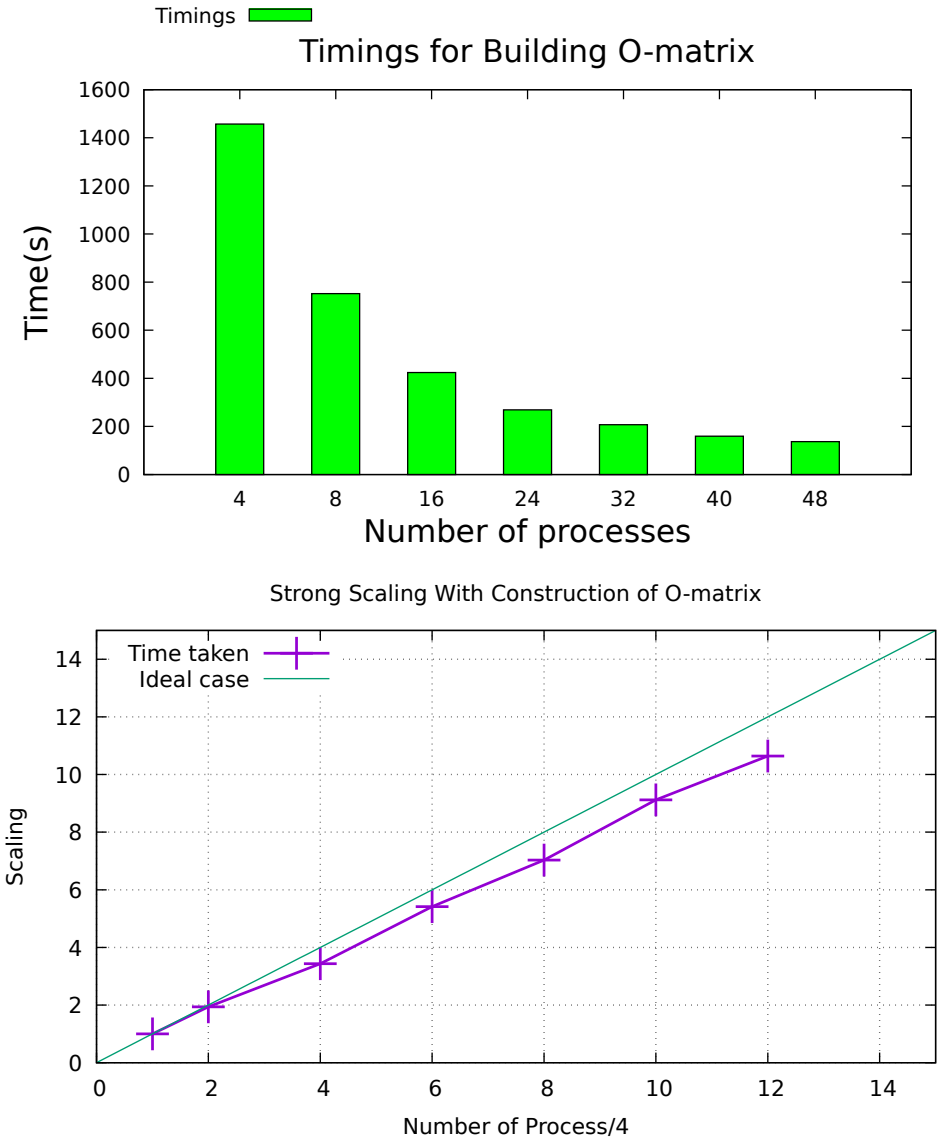

FIG. 14: We notice close to perfect scaling of the code. But this was expected, most of the runtime is devoted in building of the O-matrix which is completely distribute in both memory and computation.

2. Weak Scaling

Weak scaling can be defined as the variation in time taken to complete the task with the number of processes while keeping workload per process constant. We report weak-scaling benchmark of our code on SKL architecture15.

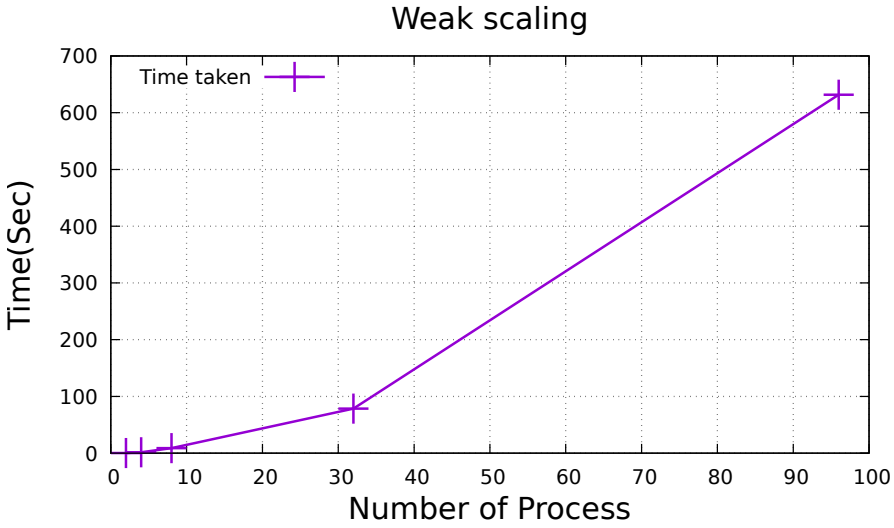

FIG. 15: The weak scaling of the code. The workload of here is scaling with the system size starting from $L = 16$ upto 24 with all the even value for L. So, we observe growth in time.

V. CONCLUSION

In this thesis, we have studied the effect of SU (2) symmetry on MBL through a massively parallel exact diagonalization code. To implement to code, we have used renowned parallel libraries PETSc and SLEPc which handle sparse matrices and linear algebra libraries very efficiently. We use the MPI for the parallelization. We build the sparse Hamiltonian with SU(2) symmetry through the fusion tree basis in a distributed memory manner. We implement the disordered Heisenberg spin ($s = 1/2$) chain.

After building the Hamiltonian, we perform the diagonalization through shift-invert targeting the middle of the spectrum asking for 50 or more eigenstates. Then we calculate the physical quantities like IPR, r and entanglement entropy of mid-spectrum eigenvalues obtained. All these quantities are instrumental in studying the presence of MBL in the system. Using our code are able to look at larger systems which give us a better understanding of the problem. We were able to look at the largest physical spin chain ($L = 26$) for this kind of problems[20].

We benchmarked the code on Skylake(SKL) and Knights landing(KNL) and measured the performance and scalability of the implementation. The code exhibits good scalability. We recognize many limitations in the implementation such as the high memory requirement during the diagonalization through LU-decomposition, not having an iterative solver, fitting the problem size in available memory, communication dominating the computation, varying scalability for different workload and overhead for setting up the parallelization by the solver.

Through this thesis, we present a fully equipped framework for the study of spin models with non-Abelian symmetries(SU(2)) for the evaluation of Many-Body Localization.

VI. ACKNOWLEDGEMENTS

I wish to convey my gratitude and admiration to my supervisor Prof. Antonello Scardicchio whose support and guidance has been invaluable to me. I would like to thank co-advisor of this project Ivan Girotto for his encouragement and patience. I will be thankful to Tommaso Parolini for the numerous discussions on the project and problem. I will be forever grateful to my parents, my sisters and my dear friends.

[1] M. A. Cazalilla and M. Rigol. Focus on dynamics and thermalization in isolated quantum many-body systems. *New J. Phys.*, 12:055006, 2010.

[2] Immanuel Bloch, Jean Dalibard, and Wilhelm Zwerger. Many-body physics with ultracold gases. *Rev. Mod. Phys.*, 80:885–964, Jul 2008.

[3] Rahul Nandkishore and David A Huse. Many-body localization and thermalization in quantum statistical mechanics. *Annu. Rev. Condens. Matter Phys.*, 6(1):15–38, 2015.

[4] D.M. Basko, I.L. Aleiner, and B.L. Altshuler. Metal–insulator transition in a weakly interacting many-electron system with localized single-particle states. *Annals of Physics*, 321(5):1126 – 1205, 2006.

[5] I. V. Gornyi, A. D. Mirlin, and D. G. Polyakov. Interacting electrons in disordered wires: Anderson localization and low-t transport. *Phys. Rev. Lett.*, 95:206603, Nov 2005.

[6] Vadim Oganesyan and David A. Huse. Localization of interacting fermions at high temperature. *Phys. Rev. B*, 75:155111, Apr 2007.

[7] M. Znidaric, T. Prosen, and P. Prelovsek. Many-body localization in the heisenberg xxz magnet in a random field. *Phys. Rev. B*, 77:064426, Feb 2008.

[8] Arijeet Pal and David A. Huse. Many-body localization phase transition. *Phys. Rev. B*, 82:174411, Nov 2010.

[9] Ronen Vosk and Ehud Altman. Many-body localization in one dimension as a dynamical renormalization group fixed point. *Phys. Rev. Lett.*, 110:067204, Feb 2013.

[10] Maksym Serbyn, Z. Papi ć, and Dmitry A. Abanin. Universal slow growth of entanglement in interacting strongly disordered systems. *Phys. Rev. Lett.*, 110:260601, Jun 2013.

[11] Maksym Serbyn, Z. Papi ć, and Dmitry A. Abanin. Local conservation laws and the structure of the many-body localized states. *Phys. Rev. Lett.*, 111:127201, Sep 2013.

[12] David A. Huse, Rahul Nandkishore, and Vadim Oganesyan. Phenomenology of fully many-body-localized systems. *Phys. Rev. B*, 90:174202, Nov 2014.

[13] Jens H. Bardarson, Frank Pollmann, and Joel E. Moore. Unbounded growth of entanglement in models of many-body localization. *Phys. Rev. Lett.*, 109:017202, Jul 2012.

[14] David J. Luitz, Nicolas Laflorencie, and Fabien Alet. Many-body localization edge in the random-field heisenberg chain. *Phys. Rev. B*, 91:081103, Feb 2015.

[15] Kartiek Agarwal, Sarang Gopalakrishnan, Michael Knap, Markus Müller, and Eugene Demler. Anomalous diffusion and griffiths effects near the many-body localization transition. *Phys. Rev. Lett.*, 114:160401, Apr 2015.

[16] Pedro Ponte, Z. Papić, François Huveneers, and Dmitry A. Abanin. Many-body localization in periodically driven systems. *Phys. Rev. Lett.*, 114:140401, Apr 2015.

[17] Achilleas Lazarides, Arnab Das, and Roderich Moessner. Fate of many-body localization under periodic driving. *Phys. Rev. Lett.*, 115:030402, Jul 2015.

[18] Dmitry A. Abanin, Wojciech De Roeck, and Franois Huveneers. Theory of many-body localization in periodically driven systems. *Annals of Physics*, 372:1 – 11, 2016.

[19] Vedika Khemani, Achilleas Lazarides, Roderich Moessner, and S. L. Sondhi. Phase structure of driven quantum systems. *Phys. Rev. Lett.*, 116:250401, Jun 2016.

[20] Francesca Pietracaprina, Nicolas Mac, David J. Luitz, and Fabien Alet. Shift-invert diagonalization of large many-body localizing spin chains. *SciPost Phys.*, 5:45, 2018.

[21] Marlon Brenes, Vipin Kerala Varma, Antonello Scardicchio, and Ivan Girotto. Massively parallel implementation and approaches to simulate quantum dynamics using krylov subspace techniques. *CoRR*, abs/1704.02770, 2017.

[22] Michael Schreiber, Sean S. Hodgman, Pranjal Bordia, Henrik P. Lüschen, Mark H. Fischer, Ronen Vosk, Ehud Altman, Ulrich Schneider, and Immanuel Bloch. Observation of many-body localization of interacting fermions in a quasirandom optical lattice. *Science*, 349(6250):842–845, 2015.

[23] Pranjal Bordia, Henrik P. Lüschen, Sean S. Hodgman, Michael Schreiber, Immanuel Bloch, and Ulrich Schneider. Coupling identical one-dimensional many-body localized systems. *Phys. Rev. Lett.*, 116:140401, Apr 2016.

[24] J. Smith, A. Lee, P. Richerme, B. Neyenhuis, P. W. Hess, P. Hauke, M. Heyl, D. A. Huse, and C. Monroe. Many-body localization in a quantum simulator with programmable random disorder. *Nat. Phys.*, 12(10):907–911, 10 2016.

[25] Kai Xu, Jin-Jun Chen, Yu Zeng, Yu-Ran Zhang, Chao Song, Wuxin Liu, Qiujiang Guo, Pengfei Zhang, Da Xu, Hui Deng, Keqiang Huang, H. Wang, Xiaobo Zhu, Dongning Zheng, and Heng Fan. Emulating many-body localization with a superconducting quantum processor. *Phys. Rev. Lett.*, 120:050507, Feb 2018.

[26] M. Ovadia, D. Kalok, I. Tamir, S. Mitra, B. Sacépé, and D. Shahar. Evidence for a finite-temperature insulator. *Scientific Reports*, 5:13503, 08 2015.

[27] Soonwon Choi, Joonhee Choi, Renate Landig, Georg Kucsko, Hengyun Zhou, Junichi Isoya, Fedor Jelezko, Shinobu Onoda, Hitoshi Sumiya, Vedika Khemani, Curt von Keyserlingk, Norman Y. Yao, Eugene Demler, and Mikhail D. Lukin. Observation of discrete time-crystalline order in a disordered dipolar many-body system. *Nature*, 543:221–225, 03 2017.

[28] A. Rubio-Abadal, J.-y. Choi, J. Zeiher, S. Hollerith, J. Rui, I. Bloch, and C. Gross. Probing many-body localization in the presence of a quantum bath. *ArXiv e-prints*, April 2018.

[29] A. Lukin, M. Rispoli, R. Schittko, M. E. Tai, A. M. Kaufman, S. Choi, V. Khemani, J. Léonard, and M. Greiner. Probing entanglement in a many-body-localized system. *ArXiv e-prints*, May 2018.

[30] Lincoln D Carr, David DeMille, Roman V Krems, and Jun Ye. Cold and ultracold molecules: science, technology and applications. *New Journal of Physics*, 11(5):055049, 2009.

[31] R. Blatt and C. F. Roos. Quantum simulations with trapped ions. *Nature Physics*, 8:277 EP –, 04 2012.

[32] David Pekker, Gil Refael, Ehud Altman, Eugene Demler, and Vadim Oganesyan. Hilbert-glass transition: New universality of temperature-tuned many-body dynamical quantum criticality. *Phys. Rev. X*, 4:011052, Mar 2014.

[33] Jonas A. Kjäll, Jens H. Bardarson, and Frank Pollmann. Many-body localization in a disordered quantum ising chain. *Phys. Rev. Lett.*, 113:107204, Sep 2014.

[34] David A. Huse, Rahul Nandkishore, Vadim Oganesyan, Arijeet Pal, and S. L. Sondhi. Localization-protected quantum order. *Phys. Rev. B*, 88:014206, Jul 2013.

[35] Ivan V. Protopopov, Wen Wei Ho, and Dmitry A. Abanin. Effect of su(2) symmetry on many-body localization and thermalization. *Phys. Rev. B*, 96:041122, Jul 2017.

[36] Shrirang Abhyankar, Jed Brown, Emil M Constantinescu, Debojyoti Ghosh, Barry F Smith, and Hong Zhang. Petsc/ts: A modern scalable ode/dae solver library. *arXiv preprint arXiv:1806.01437*, 2018.

[37] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong

Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.10, Argonne National Laboratory, 2018.

[38] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2018.

[39] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[40] J. E. Roman, C. Campos, E. Romero, and A. Tomas. SLEPc users manual. Technical Report DSIC-II/24/02 - Revision 3.10, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, 2018.

[41] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[42] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[43] VisualParadigm online. VisualParadigm.

[44] Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325, September 2005.

[45] Strumpack – structured matrices package, version 00, 12 2014.

[46] top500. The list of top 500 supercomputers.

[47] Wiki Chip. Wiki Chip.

[48] 2018 Elda Rossi, last modified by Silvia Giuliani on Nov 28. MARCONI UserGuide.