Computer Physics Communications 182 (2011) 1083–1092

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Diagonalisation of quantum observables on regular lattices and general graphs ☆

Niall Moran [a,*,1], Graham Kells [a], Jiri Vala [a,b]

[a] Department of Mathematical Physics, National University of Ireland, Maynooth, Ireland
[b] Dublin Institute for Advanced Studies, School of Theoretical Physics, 10 Burlington Rd, Dublin, Ireland

ARTICLE INFO

Article history:
Received 23 June 2010
Accepted 29 December 2010
Available online 11 January 2011

Keywords:
Exact diagonalisation
Quantum observable
Distributed memory
Lanczos
Lattice symmetries

ABSTRACT

In the study of quantum mechanical systems, exact diagonalisation (ED) methods play an extremely important role. We have developed an ED code named DoQO (Diagonalisation of Quantum Observables). This code is capable of constructing and diagonalising the observables for spin $\frac{1}{2}$ and spinless fermionic particles with many body interactions on arbitrary graphs using massively parallel distributed memory machines. At the same time, the code can exploit physical symmetries to reduce the size of the relevant basis set and provide useful physical information about each eigenstate. DoQO has been employed successfully to directly diagonalise systems with basis sets containing a billion elements. By exploiting symmetries it has been possible to perform calculations on systems with 36 spin $\frac{1}{2}$ particles. Here we present essential background details, the structure and usage of DoQO, and a study of the performance characteristics of DoQO on different machines.

Program summary

Program title: DoQO
Catalogue identifier: AEII_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEII_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 81 845
No. of bytes in distributed program, including test data, etc.: 495 379
Distribution format: tar.gz
Programming language: C++ (dependencies require Fortran)
Computer: Standard workstations and distributed memory machines
Operating system: Any operating system with C++, Fortran, MPI, PETSc and SLEPc (code developed and tested on OS X and Linux)
Has the code been vectorised or parallelised?: Yes code uses MPI for interprocess communication. One to thousands of processors may be used
RAM: Depends on problem size. Ranges from MBs to TBs
Classification: 7.8
External routines: PETSc, SLEPc, LAPACK, BLAS, MPI, BOOST, tinyxml
Nature of problem: To calculate the low lying eigenvalues and eigenstates of quantum observables for spin $\frac{1}{2}$ and spinless fermionic systems on arbitrary graphs efficiently in parallel.
Solution method: Large scale linear scaling iterative exact diagonalisation methods are used on distributed memory machines. Physical symmetries are exploited to extend the size of systems which can be treated and to provide important additional information about the eigenstates.
Restrictions: The size of the systems that DoQO can handle is restricted by the amount of available memory.
Unusual features: The main feature that makes DoQO stand out from other diagonalisation codes is its ability to exploit physical symmetries efficiently using parallel computer architectures without the use of model specific optimisations. The ability to treat systems with arbitrarily complex interactions is also unique.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author. Tel.: +353 1 7083548.
E-mail address: niall.moran@gmail.com (N. Moran).
[1] Room 1.1, Department of Mathematical Physics, National University of Ireland, Maynooth, Maynooth, Kildare, Ireland.

0010-4655/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2010.12.051

Running time: The running time ranges from seconds to hours depending on the problem size and computational resources used.

1. Motivation and background

Solving a quantum mechanical system means finding the eigenvalues and eigenstates of a relevant set of quantum operators. In many instances, finding the extremal part of the spectrum is not only the most tractable numerically but also the most relevant to the physics of the system being studied. This is particularly relevant to strongly correlated many body quantum problems which lie at the heart of condensed matter and statistical physics research today.

Here we present a code for Diagonalisation of Quantum Observables (DoQO). It is capable of constructing and diagonalising observables for spin $\frac{1}{2}$ and spinless fermionic particles on both regular lattices and general graphs. It produces numerically exact data about the low energy part of the operator spectrum and provides access to the full eigenfunctions, making it possible to calculate useful physical quantities. Exact diagonalisation techniques, like those implemented in DoQO, are essential for getting important physical insights and also for understanding the limits of approximative techniques which in principle allow larger systems to be studied. DoQO can exploit physical symmetries which reduce the relevant basis set size and the related memory requirements, providing additional information about each eigenstate. In addition, DoQO has been designed to work in parallel to take advantage of modern High Performance Computing (HPC) resources and has been benchmarked on various HPC platforms. The memory, load and indexing issues are taken care of automatically using a number of custom built and generalisable algorithms.

In the rest of this section we introduce essential concepts relevant to quantum observables and symmetries and then discuss the associated implementation issues. In Section 2 we provide an overview of the software together with usage information. In Section 3 the central components of the software are explained in more detail, with a demonstration of how DoQO is used given in Section 4. In Section 5 the scaling and performance of the code for a benchmark system on a number of different HPC platforms are investigated. Finally in Section 6 we explore possible ways to extend the code. These possible extensions are related to handling different particle types and exploiting additional symmetries.

1.1. Quantum observables

The main objective of DoQO is to provide exact low energy spectral data, specifically ground state and low lying eigenstates and the relevant eigenvalues, of quantum observables. This data can be used to formulate and verify relevant analytical models and to benchmark approximative techniques. By benchmarking these techniques we can quantify their reliability for a given system or physical context.

Quantum observables are self-adjoint operators which represent observable physical quantities. Their eigenvalues are real numbers and together with the related eigenstates correspond to the possible measurement outcomes. A prominent example of an observable is the Hamiltonian. This observable represents the total energy of the system and generates quantum dynamics through the Schrödinger equation. Its eigenvalues are the energy levels and the eigenstates are the corresponding stationary states. In the context of condensed matter and statistical physics we are generally interested in the ground state and/or low lying part of the energy spectrum.

DoQO is designed to work with systems of spin $\frac{1}{2}$ or spinless fermionic particles on regular lattices or general graphs. It is possible to describe interactions involving arbitrary numbers of particles. Each spin $\frac{1}{2}$ particle is a two-level quantum system whose states are vectors in a two-dimensional Hilbert space $\mathcal{H}^2$. Likewise each site of a spinless fermionic system can be either empty or occupied. The two level nature of these particles makes DoQO relevant to quantum information processing systems.

The Hilbert space of the system of n spin $\frac{1}{2}$ particles is the n-fold tensor product $\bigotimes_1^n \mathcal{H}^2$, so its dimension is an exponential of the number of spin $\frac{1}{2}$ particles. DoQO tackles the system size limitation, which derives from the exponential scaling of the basis set, by several means. First, symmetries can be used to reduce the basis set without compromising the quantitative accuracy of the computed data. Second, DoQO exploits the sparsity of the matrix representation to save memory. Lastly DoQO constructs the observables such that the diagonalisation procedure can be performed efficiently in parallel on massively parallel distributed memory architectures.

1.2. Employing symmetries

The use of symmetries reduces the computational resources required to diagonalise an observable while simultaneously providing additional physical information about each eigenstate. By working in the eigenbasis of a symmetry operator (SO) and reordering the basis elements appropriately, the matrix representation of the observable becomes block diagonal. Each block corresponds to a particular eigenspace of the SO and can be diagonalised separately. The states resulting from the diagonalisation of a given block are labelled by a quantum number that is the eigenvalue of the SO associated with that block.

DoQO is capable of exploiting the symmetries which conserve parity, filling and momentum. Where multiple symmetries are compatible they can be exploited simultaneously. Here we define these symmetries more precisely and show how DoQO determines and indexes the relevant eigenvectors of each SO.

1.2.1. Parity

The SO that conserves parity is defined as $\prod_i \sigma_i^z$ for spin $\frac{1}{2}$ systems and $\prod_i (2n_i - 1)$ for spinless fermionic systems. Here the products are over the subset of sites for which parity is conserved and n_i is the spinless fermionic number operator. These operators are diagonal in the standard basis that DoQO uses. For spin $\frac{1}{2}$ systems the standard basis refers to the basis in which σ^z operators are diagonal and for fermionic systems the occupancy number basis. As a result of the SO's being diagonal in the standard basis no change of basis elements is required to exploit this symmetry. DoQO can also exploit situations where parity is conserved over subsets of spins or sites. This results in a further reduction of the basis set size for each block and thus the computational resources required. For example, if there are M subsets of sites over which parity is conserved then there are 2^M blocks each with dimension 2^{N-M} where N is the total number of sites.

1.2.2. Filling

For spin $\frac{1}{2}$ systems the SO's that conserves filling are defined as $\frac{1}{2}\sum_i(\sigma_i^z + 1)$ (for spin $\frac{1}{2}$) and $\sum_i n_i$ (for spinless fermions) with i running over the subset of sites for which filling is conserved. Like the SO's that conserve parity these operators are diagonal in the standard bases. If filling is conserved over a subset of sites this implies that parity is also conserved over that subset. In this respect conservation of filling is stronger than conservation of parity. If filling is conserved over M subsets of sites with each subset containing s_i sites there are $\prod_i^M (s_i + 1)$ blocks. The dimension of the block with filling f_i in each subset is $\prod_i \binom{s_i}{f_i}$.

1.2.3. Momentum

DoQO is capable of exploiting the translational symmetry, which conserves the momentum in each direction. The SO's in this case are the translation operators $T = e^{i\hat{x}\hat{k}}$ and the quantum numbers e^{ik}, where k is the momentum and $0 \leqslant k \leqslant 2\pi$. As these operators are not diagonal in the standard basis a change of basis elements is required to exploit these symmetries. In DoQO the method of representatives is used to work in the eigenbasis of the SO's, and calculate the matrix elements of each block. This method is discussed in [1,2].

From each set of configurations which can be related via translation a representative configuration $|\psi_r\rangle$ is selected. This is conventionally chosen to be the configuration with the smallest numerical label in the set. An eigenstate of the translation operators with a given momentum can be determined from each representative configuration. These eigenstates have the form of a weighted sum of states obtained by translating the representative configuration. The weight of each state is a function of the momentum and of the number of translations needed to cycle back to the representative configuration. For a square lattice with $L \times L$ sites and translation operators T_1 and T_2, an eigenstate with momentum k_i in the direction of T_i is given by

$$|\psi_{k_1,k_2,r}\rangle = \frac{1}{\mathcal{N}} \sum_{x_1=0}^{L-1} \sum_{x_2=0}^{L-1} e^{i(x_1 k_1 + x_2 k_2)} |\psi_r^{x_1,x_2}\rangle$$

where $\mathcal{N}$ is a normalisation factor ensuring $\langle\psi_{k_1,k_2,r}|\psi_{k_1,k_2,r}\rangle = 1$ and $|\psi_r^{x_1,x_2}\rangle = T_1^{x_1} T_2^{x_2} |\psi_r\rangle$. This is in essence the discrete Fourier transform.

DoQO exploits the fact that the matrix elements for a given block can be calculated by working with the representative configurations alone. This can be clearly seen by expanding the expression for each matrix element in a given block $\langle\psi_{k_1,k_2,r}|H|\psi_{k_1,k_2,r'}\rangle$.

1.2.4. Labelling

In order to implement the exploitation of symmetries, the basis elements of each block must also be labelled appropriately. To do this DoQO uses perfect mapping functions for the symmetries which conserve parity and filling and uses a sorted distributed array to label the representative configurations when exploiting the translational symmetries. The implementation details of the so-called perfect mapping functions used are explained in Section 3.1.1 and the use of distributed sorted arrays is discussed in Sections 3.1.2 and 3.1.3.

2. Overview of the software structure and usage

DoQO is written in C++ and makes extensive use of the PETSc [3–5] and SLEPc [6] libraries which are required for DoQO to work. The XML file format is used for some of the input files due to its extensibility. An open source library named TinyXML [7] is used to parse the XML files and is included with the DoQO code.

2.1. Compiling DoQO

A makefile has been included in the DoQO source directory which can be used to build DoQO. For this to work the PETSc and SLEPc libraries must first be built and the relevant environment variables set. Users are advised to consult the README file included with the code for detailed instructions about compiling PETSc, SLEPc and DoQO.

2.2. Basic usage

DoQO can be run as a single process or as multiple intercommunicating processes. To run a single process of DoQO a command as in Code block 1 is used from the directory containing the DoQO executable. The input filename is passed to DoQO via the 'input' switch:

Code block 1 Command used to launch a single process of DoQO.

```
./doqo -input sample_input.xml
```

To run multiple intercommunicating processes of DoQO the MPI launcher application is used. This is named 'mpiexec' or 'mpirun' depending on the MPI implementation in use. An example of the command used to launch 16 DoQO processes is given in Code block 2:

Code block 2 Command used to launch DoQO with sixteen processes using mpiexec.

```
mpiexec -np 16 ./doqo -input sample_input.xml
```

The input file which is passed to the DoQO executable is an XML file which contains all the parameters that control how DoQO runs. An example of a simple input file is given in Code block 3:

Code block 3 Sample input file for DoQO containing required parameters. These are the model file which describes the observable, the task list file which specifies the coefficient values for each task and the output prefix which is the prefix used for the output files.

```
<?xml version="1.0" encoding="UTF-8"?>
<SIMULATION>
    <PARAMETERS>
        <MODEL_FILE>ising_chain_L_8.ham</MODEL_FILE>
        <TASK_LIST>ising_chain_tasks</TASK_LIST>
        <OUTPUT_PREFIX>ising_chain_L_8</OUTPUT_PREFIX>
    </PARAMETERS>
</SIMULATION>
```

The file specified by the MODEL_FILE parameter describes the observable that is being diagonalised. An example of such a file for a system of spin $\frac{1}{2}$ particles can be seen in Code block 4:

Code block 4 Input specification for Hamiltonian of Ising chain in a transverse magnetic on an 8 site ring.

```
SITES 8
PARAMETERS
J
h
TERMS
1 X,2 X * J
2 X,3 X * J
3 X,4 X * J
4 X,5 X * J
5 X,6 X * J
6 X,7 X * J
7 X,8 X * J
8 X,1 X * J
1 Z * h
2 Z * h
3 Z * h
4 Z * h
5 Z * h
6 Z * h
7 Z * h
8 Z * h
```

This example describes the Hamiltonian for the transverse field Ising model on a ring with eight spins. That is

$$H = J\sum_{i=1}^{8}\sigma_i^x\sigma_{i+1}^x + h\sum_{i=1}^{8}\sigma_i^z$$

with $\sigma_{8+1}^x = \sigma_1^x$. The format of this file is:

- The first line specifies the number of sites over which the observable is defined.
- The second line contains the label ‘PARAMETERS’.
- The following lines up to the one containing the label ‘TERMS’ contain the parameter names of the coefficients used in the definition of the observable.
- Next is a line containing the label ’TERMS’.
- The rest of the lines specify the terms that make up the observable. The format for these lines is:
  - A comma separated list followed by an asterisk where each entry is composed of a site index and an X, Y or Z which signify a σ^x, σ^y or σ^z operator acting on the site specified by the site index.
  - A comma separated list of the parameters that are multiplied together to get the coefficient for each term.

There is an implicit identity for each site not explicitly mentioned. In this way ‘* J’ would describe the identity operator over all sites multiplied by the coefficient J. For spinless fermionic systems a similar format is used except instead of using X, Y and Z to represent σ^x, σ^y and σ^z we use C and A to represent the creation and annihilation operators $c^\dagger$ and c. Refer to Appendix B for further details on constructing operators.

The file specified by the TASK_LIST parameter contains a list of the parameter values to be used for each task. This file has one line per task, where each line consists of a comma separated list of parameters. These parameters are used to calculate the coefficients for each term of the observable. Any parameters which are not specified are set to zero. By specifying appropriate parameter values in the task file, scans of the parameter space can be performed with a single call to DoQO. An example of a task file with seven tasks is given in Code block 5:

Code block 5 Example of task file specifying seven tasks for the Ising chain described in Code block 4.

```
J = -1.0
J = -1.0, h = 0.5
J = -1.0, h = 1.0
J = -1.0, h = 1.5
J = -1.0, h = 2.0
J = -1.0, h = 2.5
J = -1.0, h = 3.0
```

DoQO produces XML output files. A general output file is created which lists the specific output files that contain the results of each diagonalisation. The name of the general output file is composed of the output prefix as specified in the input file with the suffix ‘.output.xml’ appended. Details about the code version and environment are also provided in this file. In addition for each task and symmetry block a separate output file is created containing the results of that diagonalisation. The information written to these output files includes the eigenvalues converged, the error estimates as well as additional information. This includes the basis set size, the time taken and the number of iterations of the solver method used.

2.3. *Exploiting symmetries*

To enable the use of symmetries in DoQO an element describing the symmetries to be exploited is added to the XML input file. The element is enclosed within tags labelled ‘SYMMETRIES’. For each symmetry that one wishes to exploit a child element describing that symmetry is added inside the initial element. These child elements are enclosed between tags labelled ‘PARITY’, ‘FILLING’ and ‘MOMENTUM’ corresponding to the symmetries that conserve parity, filling and momentum respectively. Code block 6 contains an extract of an input that has a symmetries element with a child element specifying a symmetry that conserves parity.

Additional details for each symmetry are given in a metadata file specified by the ‘file’ attribute for each symmetry element. For the symmetries that conserve parity and filling this metadata file specifies the subsets of sites on which the parity or filling is conserved. For translational symmetries this file specifies the lattice geometry and translation vectors. The format for these files is explained in more detail later in this section.

DoQO will by default diagonalise an observable in each eigenspace of the supplied symmetry operators one after another. An optional child element can be added within each symmetry element that allows one to restrict the calculation to specific eigenspaces. This element is enclosed by tags labelled ‘RELEVANT_SECTORS’ and has an attribute called ‘number’ that specifies how many of the eigenspaces are to be used. Each eigenspace to be used is then specified by a child element of this element labelled by ‘SECTOR’ that encloses an integer that uniquely labels an eigenspace of the symmetry operator. The details of how each eigenspace is labelled for each of the symmetries that DoQO exploits is described later in this section. Code block 6 shows an example of an input file that specifies that DoQO exploit the conservation of parity and that it is restricted to the eigenspaces of the parity operators labelled one and three:

Code block 6 Input file specifying that symmetry conserving parity should be used.

```
<SIMULATION>
    <PARAMETERS>
       ...
    <SYMMETRIES>
        <PARITY file="subsets.txt">
            <RELEVANT_SECTORS number="2">
                <SECTOR>1</SECTOR>
                <SECTOR>3</SECTOR>
            </RELEVANT_SECTORS>
        </PARITY>
    </SYMMETRIES>
    </PARAMETERS>
</SIMULATION>
```

For the symmetry that conserves parity the metadata file identifies the subsets of sites for which parity is conserved. This file contains one line for each subset of sites. Each line consists of a bit string with the number of bits matching the number of sites in the system. Each site corresponds to a bit in this string with the site indices increasing from right to left. For sites included in the subset, there is a ‘1’ in the position that corresponds to that site and a ‘0’ otherwise. In Code block 7 an example of such a file for a system with eight sites is shown:

Code block 7 File specifying two subsets of sites, 1 to 4 and 5 to 8.

```
00001111
11110000
```

Here two subsets of sites are defined, one which spans sites with indices 1 to 4 and one that spans sites with indices 5 to 8. The SO’s that conserve parity over these subsets of sites for spin $\frac{1}{2}$ systems are $P_0 = \prod_{i=1}^{4}\sigma_i^z$ and $P_1 = \prod_{i=5}^{8}\sigma_i^z$. The four eigenspaces with eigenvalues $(p_0, p_1) = (1,1), (1,-1), (-1,1), (-1,-1)$ are labelled with integers from zero to three respectively. If, as specified in Code block 6, only eigenspaces labelled one and three were used, this would select the eigenspaces $(p_0, p_1) = (1, -1), (-1, -1)$.

When using the symmetry that conserves filling the metadata file specifies the subsets of sites over which filling is conserved. The format of this file is the same as that used for the symmetry that conserves parity. In this case the SO's are $C_i = \frac{1}{2}\sum_{j\in N_i}(\sigma_j^z + 1)$ where N_i is the set of sites in the ith subset. There are $\prod_i(s_i + 1)$ eigenspaces where s_i is the number of sites in the ith subset. These eigenspaces correspond to the possible fillings that each subset can have. If f_i are the fillings for each subset then each eigenspace is labelled by $\sum_i(f_i \prod_{j=0}^{j<i}(s_j + 1))$.

The metadata file supplied when describing symmetries that conserve momentum provide details of the lattice geometry as well as the translation vectors for which the system is invariant:

Code block 8 Example of the metadata file that would be supplied to exploit translational invariance for a four by four square lattice. The LATTICE_VECTOR vectors specify the translation vectors in terms of lattice sites. The NORM_VECTOR vectors correspond to the same vectors except normalised to one and the LATTICE_DIMENSIONS specifies the size of the lattice in each direction.

```
LATTICE_VECTOR1 = 1,0
LATTICE_VECTOR2 = 0,1
NORM_VECTOR1 = 1,0
NORM_VECTOR2 = 0,1
LATTICE_DIMENSIONS = 4,4
```

The information contained in this file consists of two translation vectors in lattice units in each direction, the normalised version of these vectors and the dimensions of the lattice in each direction. Code block 8 shows an example of what this file looks like for a four by four square lattice with periodic boundary conditions. The number of eigenspaces of the translation operators is then $n_0 n_1$ where n_0 and n_1 are the numbers of translations possible in each direction. The corresponding eigenvalues of the translation operators for these eigenspaces are $e^{\frac{2\pi i x_0}{n_0}}$ and $e^{\frac{2\pi i x_1}{n_1}}$ where x_0 and x_1 are integers with $0 \leqslant x_0 < n_0$ and $0 \leqslant x_1 < n_1$. DoQO uses numerical labels given by $x_0 + x_1 n_0$ to refer to each of these eigenspaces.

2.4. Nearest neighbour exclusion

DoQO has the capability of working in a basis in which all configurations obey a nearest neighbour exclusion condition. This condition only allows configurations in which no two neighbouring sites are occupied. The supersymmetric lattice models discussed in [8] enforce this condition. Being able to work in this restricted basis significantly reduces the computational resource requirements for treating these systems. To enable this feature in DoQO a parameter named 'NN_EXCLUSION' is added to the input file. Code block 9 shows an example of how this parameter is used:

Code block 9 Parameter enabling nearest neighbour exclusion using adjacency file file.adj.

```
<SIMULATION>
    <PARAMETERS>
         ....
        <NN_EXCLUSION adjacency_file="file.adj">true
        </NN_EXCLUSION>
         ....
    </PARAMETERS>
</SIMULATION>
```

An XML file is also provided which provides information about which sites are adjacent. An example of such a file for a four site ring is shown in Code block 10:

Code block 10 File providing adjacency information for a four site ring.

```
<?xml version="1.0" encoding="UTF-8"?>
<EDGES number="4">
<EDGE from="1" to="2" ></EDGE>
<EDGE from="2" to="3" ></EDGE>
<EDGE from="3" to="4" ></EDGE>
<EDGE from="4" to="1" ></EDGE>
</EDGES>
```

2.5. Additional parameters

DoQO supports additional parameters that can be specified in the input file. A complete list of the possible parameters is as follows (required parameters are marked with a *).

MODEL_FILE*: File describing the observable. Example of spin $\frac{1}{2}$ operator in Code block 4.
TASK_LIST*: The file containing the values of parameters for each task that is to be run. Example in Code block 5.
OUTPUT_PREFIX*: Prefix for output files.
MODEL_TYPE: The type of observable that is being used. Currently supported options are SPIN_HALF and FERMIONIC. The default is SPIN_HALF.
VERBOSITY: Specifies the level of verbosity to use. For minimal output while the code is running choose a small value and for more detailed output choose a higher value for the verbosity. The range is from zero to fifteen with the default set to one.
EIGENVALUES: Number of eigenvalues to retrieve for each diagonalisation. Default is two.
NN_EXCLUSION: Specifies that neighbouring sites cannot be occupied. Adjacency information required see Section 2.4 for more details.
SAVE_STATES: Save states to disk for further analysis. Format can be set to 'ascii' to output vectors in human readable ASCII format.
BENCHMARK: Set initial vector for solver to all ones to get constant number of iterations for benchmarking purposes.
SOLVER_TYPE: Sets the type of solver to use. Any SLEPc eigenproblem solver type can be used. To select a particular solver to use one specifies the label for that solver. These are 'arnoldi' for Arnoldi, 'lanczos' for Lanczos, 'krylovschur' for Krylov–Schur and 'arpack' for Arpack. The default is to use Krylov–Schur.
MAX_ITERATIONS: The maximum number of iterations of the solver method to perform before giving up. Default is 500.
SAVE_MATRIX: Saves each matrix to a file on disk.
USE_DISK: Use shared disk in basis list construction when using symmetries. Details at end of Section 3.1.2.
USE_BST: Use binary sorting tree instead of linked list in exchange of basis indices. For further details see the last paragraph of Section 3.1.3.
SOLVER_TOLERANCE: Tolerance for converged eigenvalues. Default is $1.0e^{-13}$.
DEGENERACY_TOLERANCE: Tolerance for degeneracy analysis. Default is $1.0e^{-10}$.
PHASE_TOLERANCE: Tolerance for comparing phases when exploiting translational invariance. Default is $1.0e^{-10}$.

As well as the parameters in the input file, it is possible to pass command line arguments to change the behaviour of PETSc and SLEPc. A complete list can be found in the PETSc and SLEPc documentation. Some useful arguments are:

–eps_monitor: Option to print detail of convergence after iteration.
–eps_monitor_draw: Option to show plot monitoring convergence (requires X11 graphical environment).

–eps_plot_eigs: Option to plot approximations of converged eigenvalues (requires X11 graphical environment).

3. Description of the individual software components

In this section we provide technical details of some components of the DoQO code. It is not essential to know these details to use DoQO. However they provide a deeper understanding of how DoQO works and shed light on possible performance issues. They are also helpful when attempting to extend the functionality of DoQO.

3.1. Parallelism

DoQO can take advantage of modern massively parallel distributed memory machines. On these machines multiple DoQO processes are run simultaneously on interconnected processing nodes. The key data structures, including the matrix representation of the observable as well as the vectors used during diagonalisation are partitioned among all the processes.

DoQO makes use of MPI as well as the PETSc [3–5] and SLEPc [6] libraries. The MPI (Message Passing Interface) manages groups of processes as well as allowing communication between the individual processes. PETSc (Portable, Extensible Toolkit for Scientific computation) is built on top of MPI and provides distributed vectors and matrix data structures and provides functionality for performing basic operations on these efficiently in parallel. One such operation of particular relevance to iterative diagonalisation methods that is provided by PETSc is the sparse matrix vector multiplication routine. SLEPc (Scalable Library for Eigenvalue Problem computations) is a library that leverages the functionality of PETSc and implements a variety of iterative eigensolver methods making use of the underlying PETSc data structures and operations. MPI, PETSc and SLEPc have the added advantage of allowing DoQO to be very portable.

3.1.1. Perfect mapping functions

Perfect mapping functions are used to index subsets of basis elements such that the indices are consecutive integers. DoQO employs such functions when exploiting the symmetries which conserve filling and parity. Here we demonstrate how such functions work in the case of symmetries that conserve filling. It can easily be extended for symmetries which conserve parity.

To index all the possible configurations of c particles on n sites a bijective function L is used which maps the set of bit strings with length n, containing c ones and $(n-c)$ zeros, to the set of natural numbers less than $\binom{n}{c}$. For example if we have four sites with filling two the function operates as:

$$L(0011) = 0$$

$$L(0101) = 1$$

$$L(0110) = 2$$

$$L(1001) = 3$$

$$L(1010) = 4$$

$$L(1100) = 6$$

Given a bit string b of length n, with filling c and bits labelled from left to right $b_i, 1 \leqslant i \leqslant n$, we can use the index

$$L(b) = \sum_{i=1}^{n} b_i \binom{n-i}{c - (\sum_{j=1}^{i-1} b_j)}$$

for this bit string, where we have used the recursive relation

$$\binom{n}{c} = \binom{n-1}{c} + \binom{n-1}{c-1}, \quad \forall c > 0$$

The inverse function which given an index returns the corresponding bit string can also be easily implemented. For n sites and filling c the algorithm is:

(1) set $i = 1$ and set l to the value of the index,
(2) if $l < \binom{n-i}{c}$ then $b_i = 0$,
(3) If $l \geqslant \binom{n-i}{c}$ then $b_i = 1$, $l \to l - \binom{n-i}{c}$ and $c \to c - 1$,
(4) if $i < n$ increment i and repeat from step two.

These functions can also be used for parity symmetries. In this case we map the set of bit strings with length n containing either an even or odd number of '1's to the set of natural numbers less than $\sum_{c \in P} \binom{n}{c}$ where P is the set of natural numbers less than n with the desired parity.

3.1.2. Construction of the sorted distributed basis array

When exploiting translational invariance and/or using a model with a nearest neighbour exclusion condition DoQO uses a distributed sorted array to index the basis elements. The valid basis elements (VBE) are stored in this array and indexed by their position within the array. In the context of exploiting translational invariance the VBEs are the representative configurations discussed in Section 1.2. For models with a nearest neighbour exclusion condition the VBEs are those configurations in which no two adjacent sites are occupied. Two methods for populating these arrays in parallel have been implemented in DoQO. Here we discuss the difficulties involved in populating this array in parallel and how each method works.

In the absence of a deterministic method for finding and indexing the VBEs DoQO iterates over the full list of possible basis elements (PBEs) to find the set of all VBEs. Iterating over this list of PBEs in parallel and efficiently populating a sorted distributed array with the VBEs is a non-trivial task. This is because the VBEs are not in general uniformly distributed throughout this list.

The first and default method can be broken into three steps:

(1) Partitioning the list of PBEs equally among all processes, then iterating over each of these partitions on each process and counting the number of VBEs in each partition. These individual counts are then communicated to all processes using a collective MPI gather operation.
(2) These counts and linear interpolation are then used to repartition the list of PBEs so that the VBEs are more evenly distributed among the partitions. Each process then iterates over the new partitions and counts the VBEs in each partition. The counts from the first iteration are used where possible to make this process more efficient. The counts are again communicated among all the processes.
(3) The counts of VBEs from the different processes are summed to determine the total number of VBEs. From this the size of the portion of the final sorted distributed array on each process is calculated and the space for this array is allocated. Using the counts from each process it is also possible to determine the indices in the final sorted distributed array of the first and last VBEs in each partition. Each process then uses this information and starts iterating from the appropriate position in the list of PBEs to populate its local portion of the final sorted distributed array.

Fig. 1 illustrates how this process works using four processes with possible full basis set of 256 elements of which 76 are valid basis elements.

The second method makes use of a shared filesystem to create the sorted distributed array. This method performs better in situations where the VBEs are very sparsely distributed over the set of

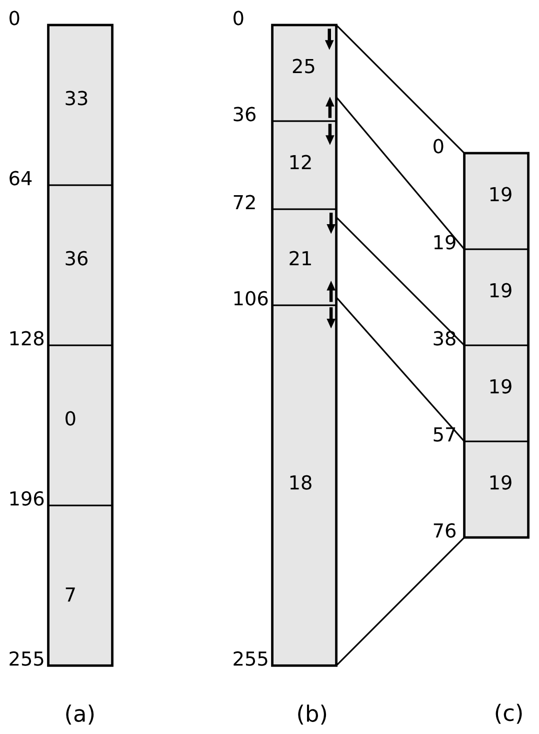

Fig. 1. Figures showing the construction of the sorted distributed array of basis elements using four processes with 76 elements out of a possible 256 elements. (a) Shows the initial partitioning of possible basis elements amongst the four processes with the indices along the left and the number of elements found within each partition shown within the partition itself. (b) Shows the repartitioned possible basis set and as can be seen the basis elements are more evenly distributed among the partitions. The arrows indicate the places in the possible basis set from which the iterations begin to populate the portions of the final distributed array. At the partition points the correct global indices in the array are known so it is possible to begin from these points on each process simultaneously. (c) Shows the sorted distributed array as it is stored across the memory of the four processes. The lines to (b) show to which range of the possible basis the valid basis elements belong.

PBEs. This is the case for models with a nearest neighbour exclusion condition. Here the set of PBEs is partitioned as in the first method. Each process then iterates over these partitions but this time writes each VBE it finds to a file on the shared filesystem. Once this has completed each process communicates how many elements it found within its own partition to the other processes. Each process then allocates space for its portion of the array and populates this by reading from the appropriate positions in the files on the shared filesystem. To enable this method in DoQO the USE_DISK parameter is set to true in the input file as mentioned in Section 2. This method requires that a shared filesystem is available and its performance will depend on the performance of this filesystem.

3.1.3. *Constructing the matrix using the sorted distributed array*

In DoQO the matrix representation of a quantum observable is stored in a distributed sparse matrix data structure provided by PETSc. Each process has ownership over a range of rows and stores all the non-zero elements for those rows. When using a sorted distributed array to index the basis elements, the row and column indices of a matrix element $\langle\psi_a|O|\psi_b\rangle$ correspond to the global indices of the basis elements $|\psi_a\rangle$ and $|\psi_b\rangle$ in this array respectively. On each process the indices of the basis elements for each local row are easily obtained from the local portion of the sorted distributed array. However to determine the column indices it is necessary to communicate with the process that has that basis element in its portion of the sorted distributed array. The fact that the distributed array is sorted makes it possible to identify the process on which a particular basis element is stored.

In DoQO the interprocess communication required to exchange indices is done in an organised fashion using only peer to peer communication. In this way synchronisation issues from collective communication operations are avoided and the method is scalable. Each process works out all the basis elements it will need indices for. It then requests the indices for these from the relevant processes. Once these have been retrieved each process can populate its own portion of the matrix. The amount of additional memory required by each process to store these basis elements and indices is in the worst case equal to the amount required to store the non-zero elements of the final sparse matrix. This memory is freed up once the matrix has been created and is available during the diagonalisation.

A data structure is required on each process to accumulate the basis elements corresponding to the columns for which indices are required. This data structure stores the basis elements so they are sorted to avoid duplicate entries and to facilitate the communication. Two different data structures can be used for this purpose in DoQO. The default is the doubly linked list implementation from the C++ standard template library and the alternative is the AVL tree implementation from the BOOST intrusive package. The AVL tree is more efficient and should be used where possible. However not all C++ compilers support the BOOST intrusive package so in these cases the doubly linked list can be used.

When using the doubly linked list it takes $O(n^2)$ to insert n items into a sorted list. This results from the fact that it takes $O(n)$ to find the correct position in the list to insert an item. The maximum length of list possible is the size of the partition of basis elements stored on each process. This is the basis size divided by the number of processes N/M. For basis sets of $O(10^9)$ even using $O(10^3)$ processes the individual lists on each process will be $O(10^6)$. The number of operations required to fill each of these lists is on the order of trillions or $O(10^{12})$. This very quickly becomes the main performance bottleneck.

An AVL tree is a self-balancing binary search tree which takes $O(n\log(n))$ to insert n items. This is a significant improvement over the $O(n^2)$ it takes to insert n items into the doubly linked list. This feature is enabled by setting the USE_BST parameter to true in the DoQO input file. DoQO must be compiled with BOOST for this to work. See README for further details in relation to compiling DoQO with BOOST.

4. DoQO in operation

To date DoQO has been used to treat a number of spin $\frac{1}{2}$ and spinless fermionic systems. Calculations performed using DoQO were used to verify DMRG results for models related to valence bond states in [9]. DoQO was also used to verify the analytical treatment of the finite size effects in the Kitaev honeycomb lattice model in [10]. In addition we have been able to use DoQO to diagonalise the 36 spin system and verify analytical expectations regarding the topological ground state degeneracy. Work is currently being undertaken to use DoQO to investigate the spinless fermionic supersymmetric lattice models first proposed in [8]. For these models the nearest neighbour exclusion condition along with the fact that filling is conserved means that it is possible for DoQO to treat systems with up to approximately fifty sites.

4.1. *Thin torus limit of the Kitaev honeycomb lattice model*

Here we demonstrate the importance of exact techniques for validating the results of approximative and analytical approaches. Using the thin torus limit of the Kitaev honeycomb lattice model

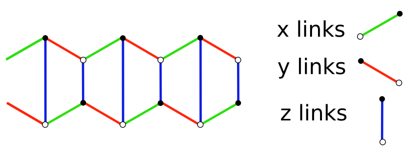

Fig. 2. Thin torus lattice limit with 12 spins. Here the lattice has been compactified along the z-link direction, the opposite horizontal sides are identified.

we compare the results from DoQO against the results from DMRG and the analytical solution of the model [11].

The Kitaev honeycomb lattice model [12] is a spin $\frac{1}{2}$ model consisting of two body interactions on a honeycomb lattice. This model exhibits both abelian and non-abelian topological phases. The Hamiltonian is given as:

$$H = -J_x \sum_{x\,\text{links}} \sigma_i^x \sigma_j^x - J_y \sum_{y\,\text{links}} \sigma_i^y \sigma_j^y - J_z \sum_{z\,\text{links}} \sigma_i^z \sigma_j^z$$

The thin torus limit of this model results from reducing the size of the lattice in the direction of the z links until we end up with a 1D ladder system. Fig. 2 shows a thin torus ladder with twelve spins. In the thermodynamic limit there are phase transitions at $J_x = J_y + J_z$ and $J_y = J_x + J_z$. Here we focus on the point where $J_x = \frac{1}{2}$, $J_y = J_z = \frac{1}{4}$ there is indications that the model is critical. Calculations of the excitation gap for various lattice sizes were calculated to see how it scales as the system size is increased.

Using DoQO it was possible to calculate the energies of the ground state and first excited states for systems with up to 32 spins. The conservation of parity along each row was exploited to reduce the basis size by a factor of four. For each of the eigenstates returned the expectation value of the sum of the plaquette operators was calculated. From this the number of vortices in each state is determined. Here the plaquettes operators are defined as $W_p = \sigma_1^x \sigma_2^y \sigma_3^z \sigma_4^x \sigma_5^y \sigma_6^z$ around each plaquette, the spins are labelled in the anti-clockwise direction starting at the lowermost spin of each plaquette.

The DMRG tool from ALPS [13] was used to calculate the energies of the ground state and first excited states for systems with up to 72 spins as well as the entanglement entropy of each state. For these calculations 30 sweeps were performed and up to 400 states were kept from each sweep. DMRG [14,15] is an approximative method which attempts to truncate the Hilbert space to the most physically relevant subspace. It has been very successful but is limited to 1D and results must be checked against exact techniques to gauge both qualitative and quantitative accuracy. With DMRG the entanglement entropy for each state is easily obtained as a result of the calculation of the density matrix at each step.

The exact solution used in this case is based on fermionization [11]. Using this solution energies for the ground state and first excited states in the zero vortex sector for systems with up to 200 sites were calculated.

The results from each method are shown in the plot in Fig. 3. As expected the results from DoQO and those from the analytical solution match exactly. The DMRG results show good agreement up to five decimal places for all points except for the 28 spin system. In this case DMRG converges to the energy of the ground state of the four vortex sector and thus gives a qualitatively incorrect result. The exact diagonalisation results from DoQO show that there is a level crossing in the vortex sectors close to this point that explains to the DMRG convergence problem.

5. Performance

A performance study of DoQO was undertaken to determine the most relevant factors influencing the performance of DoQO. A benchmark system was chosen and results were obtained for a variety of lattice sizes and numbers of processors on different machines.

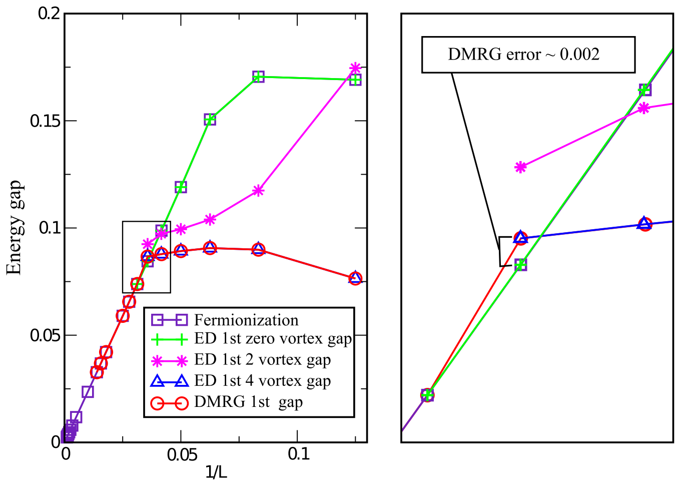

Fig. 3. Gap scaling with $\frac{1}{L}$ (where L is the chain length) for the thin torus limit of the Kitaev honeycomb lattice model. For the 28 spin ladder the energies of the first and second excited states are very close together. In this case the DMRG method fails to converge past the second excited state to the true first excited state. The numerical error is approximately 0.002 and is highlighted in the inset. For these DMRG calculations 30 sweeps were performed and 400 states were kept at each step.

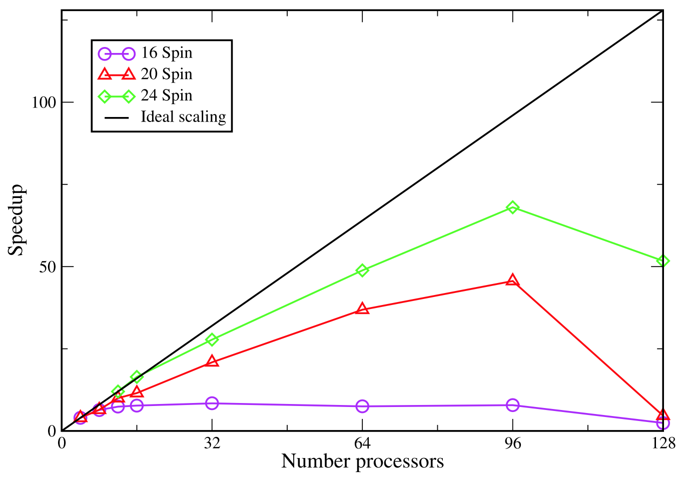

Fig. 4. Scaling behaviour of benchmark on Opteron basis gigabit Ethernet cluster using up to 128 cores.

The benchmark system chosen was the Kitaev honeycomb lattice model [12] with parameters $J_x = -0.1$, $J_y = -0.45$, $J_z = -0.45$. Two eigenvalues were calculated using the Krylov–Schur algorithm for systems of 16, 20, 24 and 28 spins and to keep the number of iterations constant for each system size the initial vector was set to all ones (set BENCHMARK parameter to true in the input file).

The machines used were an Opteron based gigabit Ethernet cluster, an IBM Blue Gene/P and a Xeon based SGI ICE cluster with ConnectX Infiniband interconnect. Plots of the strong scaling are shown in Figs. 4, 5 and 6. The speedup in each case is calculated as $s = \frac{m t_m}{t_n}$ where n is the number of processors for which the scaling is being calculated, m is the minimum number of processors on which the given system can be treated on and t_i is the time taken on i processors.

On the Opteron based gigabit Ethernet cluster (Fig. 4) a speedup is observed for the 20 and 24 spin systems up to 96 processors. This machine consists of nodes with two AMD Opteron 250

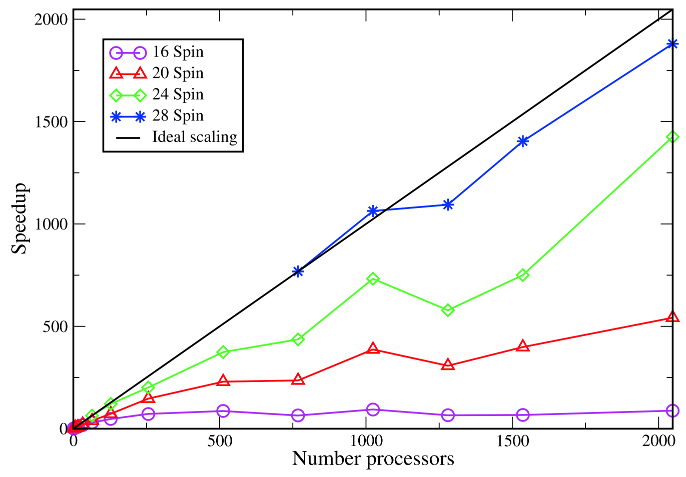

Fig. 5. Scaling behaviour of benchmark on Blue Gene/P using up to 2048 cores.

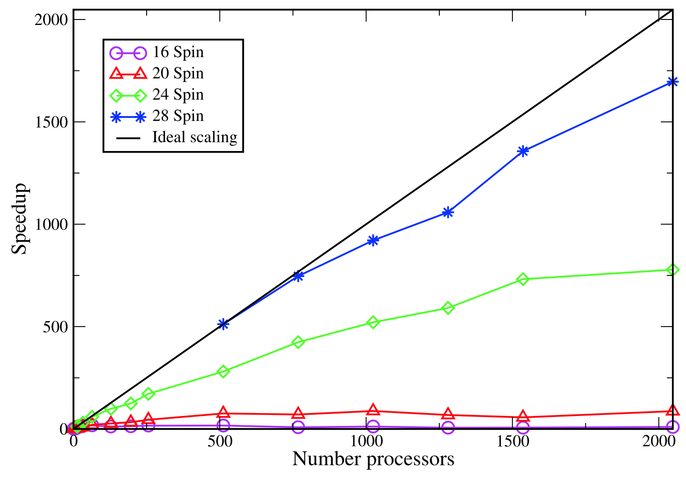

Fig. 6. Scaling behaviour of benchmark on SGI ICE system using up to 2048 cores.

2.4 GHz processors connected by a core routed gigabyte Ethernet switch.

On the Blue Gene/P system (Fig. 5) impressive scaling is observed all the way up to 2048 cores for 24 and 28 spin systems. The Blue Gene/P is made up nodes consisting of four PowerPC 450 cores running at 850 MHz and 2 GB of memory. These are connected with multiple networks including a high speed low latency 3D toroidal network used for peer to peer communication. Peaks can be seen in the plot at 512 and 1024 cores and troughs inbetween. It is thought that these are due to the fact that there are 512 cores on each mid-plane and thus a more optimal mapping of the cores occurs when using multiples of 512 cores [16].

On the SGI ICE machine (Fig. 6) good scaling is observed for the 28 spin system up to 2048 cores. For the 24 spin system the speedup drops off towards 2048 cores. The nodes of this machine each consist of two quad core Intel Xeon E5462 processors running at 2.8 GHz. These are connected via a ConnectX Infiniband interconnect.

5.1. Performance conclusions

From the performance study undertaken we conclude that the principle factors that influence the performance of DoQO are the performance of the communications network and the available memory bandwidth.

The matrix vector multiplication operation which is the central operation in DoQO involves significant interprocess communication. The gigabit Ethernet interconnect has substantially higher latency and lower bandwidth than the interconnects found on the Blue Gene/P and the SGI ICE machines. This explains the poor scaling performance past 64 processors which was observed. The multiple special purpose networks used on the Blue Gene/P for interprocess communication accounts for the superior scaling behaviour observed.

With sparse matrix vector multiplication the memory bandwidth is of central importance to ensure that the processing cores are fully utilised. Even though each core on the SGI ICE is significantly faster than those found on the Blue Gene/P the time taken for the calculations on the same numbers of cores is comparable. This perhaps indicates that the Xeon cores are not being fully utilised due to insufficient memory bandwidth. Evidence of this is observed by running DoQO on two and four cores of a Xeon quad core processor. There is almost no perceptible gain in speed while using four cores over two cores.

6. Conclusions

DoQO is a versatile tool which can be used for a broad range of interesting models on a large range of platforms. Without exploiting any symmetries DoQO can typically treat systems with up to around twenty particles on standard workstations, 24 particles on commodity distributed memory machines with relatively small numbers of processes and 30 and possibly more particles on large capability machines with high performance interconnects and large memory capacities. With symmetries and restrictions on the basis DoQO is capable of treating still larger systems.

Matrix free methods allow diagonalisation of quantum observables without requiring that the matrix elements be stored in memory. This results in reduced memory requirements and an ability to perform calculations for larger systems. These methods were investigated in the context of DoQO, however problems were encountered while attempting to achieve acceptable scalability whilst maintaining the desired generality to deal with arbitrary spin $\frac{1}{2}$ and spinless fermionic systems. As a result the current version of DoQO does not use matrix free methods but this may change in future versions.

DoQO currently addresses systems of spin $\frac{1}{2}$ and spinless fermionic particles on arbitrary lattices and graphs. The methods used are not specific to these particle types and in future DoQO may be extended to be more general. Currently DoQO is capable of exploiting symmetries which conserve parity, filling and momentum. Future versions may extend this capability to include other common symmetries.

Acknowledgements

This work has been supported by Science Foundation Ireland through the President of Ireland Young Researcher Award 05/YI2/I680. The authors wish to acknowledge the SFI/HEA Irish Centre for High-End Computing (ICHEC) for the provision of computational facilities and support, within the National Service Class A “Grand Challenge” project nmphy002a “Topological Phases in Quantum Lattice Systems”. We would also like to thank the Kavli Institute of Theoretical Physics and Aspen Center of Physics for their hospitality where the preliminary part of this work was carried out. In addition we would like to acknowledge Simon Trebst and Andreas Läuchli for their useful comments during the development of the code.

Appendix A. Matrix memory requirements

Here we discuss how to work out an upper bound on the amount of memory required to store the non-zero elements of a matrix operator. Each term of an operator can contribute at most one non-zero entry per row.

Matrices are stored using Compressed Sparse Row (CSR) format. To store the matrix each non-zero entry requires an integer for the column index, a double for storing the value (two doubles in the case of complex values) and for each row an integer is required to point to the starting position in the values and columns arrays for that row. For systems with thirty two particles and above, eight byte integers are required to accommodate the column and row indices but for smaller systems four byte integers are sufficient.

If n is the number of spins in the system and t is the number of terms then the upper bound on the amount of memory required to store all the non-zero elements using double and complex arithmetic and four and eight byte integers are listed below (where D indicates double arithmetic and C complex arithmetic and the four and eight specify the number of bytes used for storing integers in each case):

$$M_{\mathrm{D4}} = 2^n(12t+4)$$
$$M_{\mathrm{C8}} = 2^n(16t+8)$$
$$M_{\mathrm{D4}} = 2^n(20t+4)$$
$$M_{\mathrm{C8}} = 2^n(24t+8)$$

As well as requiring memory to store the non-zero elements, additional memory is needed to store meta data. This data enables efficient matrix vector multiplication operations in parallel. The amount of memory required for this depends on the number of processes and the structure of the matrix. Typically it is on the order of an additional 20%–30%.

Appendix B. Operator construction

The matrices representing the operators are generally extremely sparse. These matrices are stored in sparse matrix format where only the non zero elements and their indices are stored. This data is distributed uniformly across the available processes.

B.1. Spin $\frac{1}{2}$ systems

This section provides explicit demonstrations of how quantum operators are built up for the benefit of readers that are not familiar with these concepts. The matrix for a spin $\frac{1}{2}$ operator for a finite quantum system is made up of a sum of terms. Each term can be a single site term or an interaction term acting on multiple sites. Terms are written in terms of the Pauli matrices and the identity matrix.

The term $\sigma_1^x\sigma_2^y$ is an interaction term between the spin at position one and the spin at position two. The matrix corresponding to this term results from taking the tensor product of the matrices for the individual operators.

$$\sigma_1^x\sigma_2^y := \sigma_1^x \otimes \sigma_2^y = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix}$$

When constructing the matrix for a term implicit identity matrices are used for the sites not mentioned. For a three spin system the term $\sigma_1^x\sigma_2^y$ becomes $\sigma_1^x \otimes \sigma_2^y \otimes I_3$.

The column index and value of the non-zero entry corresponding to a given term on a given row can be calculated easily. The process is as follows:

(1) Set the column index to the given row index. Set the non-zero value to one.
(2) For each σ^x and σ^y operator in the term flip the appropriate bit in the binary representation of the column index. The bit to flip is the one in the position corresponding to the site index of the σ^x or σ^y operator in question. A bitwise exclusive or operator can be used to do this efficiently.
(3) For each σ^z operator in the term check if there is a 1 in the binary representation of the row index in the position on which the σ^z acts. If there is multiply the value by minus one.
(4) For each σ^y operator in the term check if there is a 1 in the binary representation of the column index in the position on which the σ^y acts. If there is multiply the value by $-i$ and if not then multiply the value by i.

Using this process one can determine the values and column indices of all the non-zero values for a given row. Each process then loops over its local chunk of rows and sets the non-zero values for those rows without any communication with the other processes.

References

[1] N. Laflorencie, D. Poilblanc, Simulations of pure and doped low-dimensional spin-1/2 gapped systems, Lect. Notes Phys. 645 (2004) 227–252.
[2] A. Läuchli, Introduction to exact diagonalization, Workshop in numerical approaches to quantum many-body systems, 2009, https://www.ipam.ucla.edu/publications/qs2009/qs2009_8379.pdf.
[3] S. Balay, K. Buschelman, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, H. Zhang, PETSc, http://www.mcs.anl.gov/petsc.
[4] S. Balay, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, H. Zhang, PETSc Users Manual.
[5] S. Balay, W.D. Gropp, L.C. McInnes, B.F. Smith, Efficient management of parallelism in object oriented numerical software libraries, 1997.
[6] V. Hernandez, J.E. Roman, V. Vidal, SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems, ACM Trans. Math. Softwate 31 (3) (2005) 351–362.
[7] L. Thomason, TinyXML website, http://www.grinninglizard.com/tinyxml/.
[8] P. Fendley, K. Schoutens, J. de Boer, Lattice models with $N=2$ supersymmetry, Phys. Rev. Lett. 90 (12) (2003) 1–4.
[9] E. Rico, R. Hübener, S. Montangero, N. Moran, B. Pirvu, J. Vala, H.J. Briegel, Valence bond states: link models, Ann. Phys. 324 (2009) 1875–1896.
[10] G. Kells, N. Moran, J. Vala, Finite size effects in the Kitaev honeycomb lattice model on a torus, J. Stat. Mech.: Theory Exp. (03) (2009) P03006, 13 pp.
[11] G. Kells, J.K. Slingerland, J. Vala, A description of Kitaev's honeycomb model with toric code stabilizers, Phys. Rev. B 80 (2009) 125415.
[12] A. Kitaev, Anyons in an exactly solved model and beyond, Ann. Phys. 321 (2006) 2–111.
[13] F. Albuquerque, The ALPS project release 1.3: open source software for strongly correlated systems, J. Magn. Magn. Mater. 310 (2007) 1187.
[14] S.R. White, Density matrix formulation for quantum renormalization groups, Phys. Rev. Lett. 69 (19) (1992) 2863–2866.
[15] U. Schollwöck, The density-matrix renormalization group, Rev. Mod. Phys. 77 (1) (2005) 259.
[16] G. Lakner, B. Knudson, C. Sosa, Blue Gene/P Application Development, IBM Redbooks, 2008, http://www.redbooks.ibm.com/.