Computer Physics Communications 180 (2009) 948–964

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

QCMPI: A parallel environment for quantum computing ☆

Frank Tabakin [a], Bruno Juliá-Díaz [b,*]

[a] Department of Physics and Astronomy, University of Pittsburgh, Pittsburgh, PA 15260, USA
[b] Departament de Estructura i Constituents de la Materia, Universitat de Barcelona, Diagonal 647, 08028 Barcelona, Spain

ARTICLE INFO

Article history:
Received 19 August 2008
Received in revised form 17 November 2008
Accepted 26 November 2008
Available online 30 November 2008

PACS:
03.67.Ac
03.67.Lx

Keywords:
Quantum algorithms
Parallel computing
Quantum simulation

ABSTRACT

QCMPI is a quantum computer (QC) simulation package written in Fortran 90 with parallel processing capabilities. It is an accessible research tool that permits rapid evaluation of quantum algorithms for a large number of qubits and for various "noise" scenarios. The prime motivation for developing QCMPI is to facilitate numerical examination of not only how QC algorithms work, but also to include noise, decoherence, and attenuation effects and to evaluate the efficacy of error correction schemes. The present work builds on an earlier Mathematica code QDENSITY, which is mainly a pedagogic tool. In that earlier work, although the density matrix formulation was featured, the description using state vectors was also provided. In QCMPI, the stress is on state vectors, in order to employ a large number of qubits. The parallel processing feature is implemented by using the Message-Passing Interface (MPI) protocol. A description of how to spread the wave function components over many processors is provided, along with how to efficiently describe the action of general one- and two-qubit operators on these state vectors. These operators include the standard Pauli, Hadamard, CNOT and CPHASE gates and also Quantum Fourier transformation. These operators make up the actions needed in QC. Codes for Grover's search and Shor's factoring algorithms are provided as examples. A major feature of this work is that concurrent versions of the algorithms can be evaluated with each version subject to alternate noise effects, which corresponds to the idea of solving a stochastic Schrödinger equation. The density matrix for the ensemble of such noise cases is constructed using parallel distribution methods to evaluate its eigenvalues and associated entropy. Potential applications of this powerful tool include studies of the stability and correction of QC processes using Hamiltonian based dynamics.

Program summary

Program title: QCMPI
Catalogue identifier: AECS_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AECS_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.: 4866
No. of bytes in distributed program, including test data, etc.: 42 114
Distribution format: tar.gz
Programming language: Fortran 90 and MPI
Computer: Any system that supports Fortran 90 and MPI
Operating system: developed and tested at the Pittsburgh Supercomputer Center, at the Barcelona Supercomputer (BSC/CNS) and on multi-processor Macs and PCs. For cases where distributed density matrix evaluation is invoked, the BLACS and SCALAPACK packages are needed.
Has the code been vectorized or parallelized?: Yes
Classification: 4.15
External routines: LAPACK, SCALAPACK, BLACS
Nature of problem: Analysis of quantum computation algorithms and the effects of noise.
Solution method: A Fortran 90/MPI package is provided that contains modular commands to create and analyze quantum circuits. Shor's factorization and Grover's search algorithms are explained in detail. Procedures for distributing state vector amplitudes over processors and for solving concurrent

☆ 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.
E-mail address: bjulia@gmail.com (B. Juliá-Díaz).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2008.11.021

(multiverse) cases with noise effects are implemented. Density matrix and entropy evaluations are provided in both single and parallel versions.
Running time: Test run takes less than 1 minute using 2 processors.

1. Introduction

Achieving a realistic Quantum Computer (QC) [1,2] requires the control, measurement, and stability of simple quantum systems called qubits. A qubit is any system with two accessible states which can form a quantum ensemble. That ensemble can be manipulated to store and process information. Since quantum states can exist as superpositions of many possibilities, and since an isolated quantum system propagates without loss of quantum phases, a QC provides the advantage of being a "massively parallel" device and having enhanced probability for solving difficult, otherwise intractable, problems. That enhancement is generated by constructive quantum interference. This ideal situation can be disrupted by external effects, which can cause the quantum system to loose its quantum interference capabilities—this is called decoherence and loss of entanglement. In addition, uncontrolled random pulses (noise[1]) could strike the QC during its controlled performance and thereby its operations or gates can be less than perfect.

To gauge the efficacy of a QC, even when influenced by such external environmental effects, and to evaluate the positive influence of error correction [3] steps, it is important to have large scale QC simulations. Such simulations can only represent a small part of the full "massively parallel" quantum ensemble dynamics, since a real QC goes way beyond the capabilities of any classical computer. Nevertheless, it seems natural to invoke the best, most parallel and largest memory computers we have available. Therefore, we embarked on developing a Fortran 90 parallel computer QC simulation, starting with the basic QC algorithms of quantum searching [4] and factorization [5]. Other authors have also attacked this problem to good effect [6–10]. Nevertheless, there is a need for a generally available, well-documented, and easy to use supercomputer version, to encourage others to contribute their own advances. In addition, we have developed a broader range of applications[2] and supercomputer techniques than previously available. An important feature of our work is that we invoke the algorithms on concurrent groups of processors, which are then subject to different noise. Then, the overall density matrix is constructed as an ensemble average over these noise groups. The density matrix can be stored on a grid of processors and its eigenvalues found using parallel codes, thereby avoiding the pitfalls of overly large matrix storage. Thus, we can evaluate the entropy, and indeed sub-entropies, for the dynamic evolution of a QC process in a simulation of a real world environment.

Our code is called QCMPI to indicate that it is a QC simulation based on the Message-Passing Interface (MPI) [11,12]. It is a Fortran 90 simulation of a Quantum Computer that is both flexible and an improvement over earlier such works [6–10]. The flexibility is generated by a modular approach to all of the initializations, operators, gates, and measurements, which then can be readily used to describe the basic QC Teleportation [13], Superdense coding [14], Grover's search [4] and Shor's factoring [5] algorithms. We also adopt a state vector,[3] rather than a density matrix [15], approach to facilitate representing a large number of qubits in a manner that allows for general treatments, such as handling the dynamics stipulated by realistic Hamiltonians. We include environmental effects by introducing random stochastic interactions in separate groups of processors, that we dub multiverses.

In Section 2, we introduce qubit state vectors along with various state vector notations. We stress that a wave function component description allows for changes induced by simple one-body operators such as local quantum gates and also one-body parts of Hamiltonians. Examples are provided in Section 3 of the affect of a general one-body operator on both two and more qubit systems. Expansion in a computational basis, using equivalent decimal and binary labels, is used to demonstrate the role of operators on the state vector amplitudes. It is shown how to distribute a wave function over numerous processors and how to handle the fact that a one-body operator acts on wave function amplitudes in an manner that not only modifies amplitudes stored on a given processor, but also affects amplitudes seated on other processors. Criteria for locating the processors involved in these classes of operators are derived. Understanding this combination of effects; namely, wave function distribution and the alteration of that distribution due to the action of a one-body operator, is central to all subsequent developments. It is handled by careful MPI invocations and serves as a model for the extension to multi-qubit operations.

In Section 4, the MPI manipulations described earlier for the one qubit case are generalized and then the layout for the two-qubit operator alterations of the quantum amplitudes are clarified. With that result in hand, the particular two-qubit gates **CNOT** and **CPHASE** are readily constructed, as are two-body Hamiltonians for dynamical applications. Generalization to three-qubit operators, in particular to the Toffolli gate, are obvious.

In Section 5.1, Grover's algorithm is discussed and it is shown how QCMPI allows one to simulate up to 30 qubits (depending on the number of processors and available memory) in a reasonable time.

Shor's algorithm is simulated using QCMPI as discussed in Section 5.2. Several standard codes that handle large-number modular and continued fraction manipulations are provided, but the heart of this case is the Quantum Fourier Transform (QFT) and an associated projective measurement. The QFT is generated by a chain of Hadamards and CNOT gates acting on a multi-qubit register. It is shown how to do a QFT with wave function components distributed over many processors. Here the benefit of using MPI is dramatic.

In Section 6, the procedures invoked to describe parallel universes, subject to stochastic noise, is explained for both the Grover and Shor algorithms. For brevity, similar application to teleportation and superdense coding are omitted here (although also implemented using QCMPI). Also in Section 6, the construction and evaluation of a density matrix is discussed in two ways. In one way, the full density matrix is stored on the master processor and its eigenvalues and the associated entropy is evaluated using a linear code subroutine. In the second, more general way, the density matrix is spread over many processors on a BLACS constructed processor grid and eigenvalues and entropy determined using the parallel library SCALAPACK [16]. The later version reduces the storage needs and enhances speed.

[1] There are various types of noise, such as thermal noise. We use the term noise in a generic sense, although specific noise models can be incorporated into QCMPI.
[2] Teleportation and superdense coding programs are also available, but were omitted for brevity.
[3] A state vector requires arrays of size 2^{n_q}, whereas a density matrix has a much larger size $2^{n_q} \times 2^{n_q}$. Here n_q denotes the number of qubits.

A brief description of the included routines is given in Section 7, and finally some conclusions and future developments are discussed in Section 8.

2. States

2.1. One-qubit states

The state of a quantum system is described by a wave function which in general depends on the space or momentum coordinates of the particles and on time. In Dirac's representation-independent notation, the state of a system is a vector in an abstract Hilbert space $|\Psi(t)\rangle$, which depends on time, but in that form one makes no choice between the coordinate or momentum space representation. The transformation between the space and momentum representation is contained in a transformation bracket. The two representations are thus related by Fourier transformation, which is the way Quantum Mechanics builds localized wave packets. In this manner, uncertainty principle limitations on our ability to measure coordinates and momenta simultaneously with arbitrary precision are embedded into Quantum Mechanics (QM). This fact leads to operators, commutators, expectation values and, in the special cases when a physical attribute can be precisely determined, eigenvalue equations with Hermitian operators. That is the content of many quantum texts. Our purpose is now to see how to define a state vector, to describe systems or ensembles of qubits as needed for quantum computing. Thus, the degrees of freedom associated with change in location are suppressed and the focus is on the two-state aspect.

Spin, which is the most basic example of two-valued quantum attribute, is missing from a spatial description. This subtle degree of freedom, whose existence is deduced, inter alia, by analysis of the Stern–Gerlach experiment, is an additional Hilbert space vector feature. For example, for a single spin 1/2 system the wave function including both space and spin aspects is:

$$\Psi(\vec{r}_1, t)|sm_s\rangle, \tag{1}$$

where $|sm_s\rangle$ denotes a state that is simultaneously an eigenstate of the particle's total spin operator $s^2 = s_x^2 + s_y^2 + s_z^2$, and of its spin component operator s_z. That is

$$s^2|sm_s\rangle = \hbar^2 s(s+1)|sm_s\rangle, \qquad s_z|sm_s\rangle = \hbar m_s|sm_s\rangle. \tag{2}$$

For a spin 1/2 system, we denote the spin up state as $|sm_s\rangle \to |\frac{1}{2}, \frac{1}{2}\rangle \equiv |0\rangle$, and the spin down state as $|sm_s\rangle \to |\frac{1}{2}, -\frac{1}{2}\rangle \equiv |1\rangle$.

A simpler, equivalent representation is as a two component amplitude

$$|0\rangle \to \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |1\rangle \to \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{3}$$

This matrix representation can be used to describe the two states of any quantum system and is not restricted to the spin attribute. In this matrix representation, the Pauli matrices $\vec{\sigma}$ are[4]:

$$\sigma_z \to \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \qquad \sigma_x \to \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \sigma_y \to \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}. \tag{4}$$

These are all Hermitian matrices $\sigma_i = \sigma_i^\dagger$. Along with the unit operator $\mathcal{I} \equiv \sigma_0$

$$\mathcal{I} \equiv \sigma_0 \to \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{5}$$

any operator acting on a qubit can be expressed as a combination of Pauli operators.

Operators on multi-qubit states can be expressed as linear combinations of the tensor product[5] of the Pauli operators. For example, a general operator Ω can be expressed as

$$\Omega = \sum_{i_1=0}^{3} \cdots \sum_{i_{n_q}=0}^{3} \beta_{i_1, i_2 \ldots i_{n_q}} [\sigma_{i_1} \otimes \sigma_{i_2} \cdots \otimes \sigma_{i_{n_q}}], \tag{6}$$

where $\beta_{i_1, i_2 \ldots i_{n_q}}$ is in general a set of complex numbers, but are real numbers for hermitian Ω.

A one qubit state is a superposition of the two states associated with the above 0 and 1 bits:

$$|\Psi\rangle = C_0|0\rangle + C_1|1\rangle, \tag{7}$$

where $C_0 \equiv \langle 0|\Psi\rangle$ and $C_1 \equiv \langle 1|\Psi\rangle$ are complex probability amplitudes for finding the particle with spin up or spin down, respectively. The normalization of the state $\langle\Psi|\Psi\rangle = 1$, yields $|C_0|^2 + |C_1|^2 = 1$. Note that the spatial aspects of the wave function are being suppressed; which corresponds to the particles being in a fixed location, such as at quantum dots.[6] A 2×1 matrix representation of this one-qubit state is thus:

$$|\Psi\rangle \to \begin{pmatrix} C_0 \\ C_1 \end{pmatrix}. \tag{8}$$

An essential point is that a QM system can exist in a superposition of these two bits; hence, the state is called a quantum-bit or "qubit". Although our discussion uses the notation of a system with spin 1/2, it should be noted again that the same discussion applies to any two distinct states that can be associated with $|0\rangle$ and $|1\rangle$.

[4] $\vec{s} = \frac{\hbar}{2}\vec{\sigma}$.

[5] A tensor product of two matrices $A \otimes B$ is defined by the rule: $\langle q_i, q_j|A \otimes B|q_s, q_t\rangle \equiv \langle q_i|A|q_s\rangle\langle q_j|B|q_t\rangle$, with obvious generalization to multi-qubit operators.

[6] When these separated systems interact, one might need to restore the spatial aspects of the full wave function.

2.2. Multi-qubit states

A quantum computer involves more than one qubit; therefore, we generalize the previous section to multi-qubit states.

For more than one qubit, a so-called computational basis of states is defined by a product space

$$|n\rangle_{n_q} \equiv |q_1\rangle \ldots |q_{n_q}\rangle \equiv |\mathbf{Q}\rangle, \tag{9}$$

where n_q denotes the total number of qubits in the system. We use the convention that the most significant qubit[7] is labeled as q_1 and the least significant qubit by q_{n_q}. Note we use q_i to indicate the quantum number of the ith qubit. The values assumed by any qubit are limited to either $q_i = 0$ or 1. The state label $\mathbf{Q}$ denotes the qubit array $\mathbf{Q} = (q_1, q_2, \ldots, q_{n_q})$, which is a binary number label for the state with equivalent decimal label n. This decimal multi-qubit state label is related to the equivalent binary label by

$$n \equiv q_1 \cdot 2^{n_q-1} + q_2 \cdot 2^{n_q-2} + \cdots + q_{n_q} \cdot 2^0 = \sum_{i=1}^{n_q} q_i \cdot 2^{n_q-i}. \tag{10}$$

Note that the ith qubit contributes a value of $q_i \cdot 2^{n_q-i}$ to the decimal number n. Later we will consider "partner states" $(|n_0\rangle,\ |n_1\rangle)$ associated with a given n, where a particular qubit i_s has a value of $q_{i_s} = 0$,

$$n_0 = n - q_{i_s} \cdot 2^{n_q-i_s}, \tag{11}$$

or a value of $q_{i_s} = 1$,

$$n_1 = n - (q_{i_s} - 1) \cdot 2^{n_q-i_s}. \tag{12}$$

These partner states are involved in the action of a single operator acting on qubit i_s, as described in the next section.

A general state with n_q qubits can be expanded in terms of the above computational basis states as follows

$$|\Psi\rangle_{n_q} = \sum_{\mathbf{Q}} C_{\mathbf{Q}} |\mathbf{Q}\rangle \equiv \sum_{n=0}^{2^{n_q}-1} C_n |n\rangle, \tag{13}$$

where the sum over $\mathbf{Q}$ is really a product of n_q summations of the form $\sum_{q_i=0,1}$. The above Hilbert space expression maps over to an array, or column vector, of length 2^{n_q}

$$|\Psi\rangle_{n_q} \equiv \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ \vdots \\ C_{2^{n_q}-1} \end{pmatrix} \quad \text{or with binary labels } \rightarrow \begin{pmatrix} C_{0\ldots00} \\ C_{0\ldots01} \\ \vdots \\ \vdots \\ C_{1\ldots11} \end{pmatrix}. \tag{14}$$

The expansion coefficients C_n (or $C_{\mathbf{Q}}$) are complex numbers with the physical meaning that $C_n = \langle n|\Psi\rangle_{n_q}$ is the probability amplitude for finding the system in the computational basis state $|n\rangle$, which corresponds to having the qubits pointing in the directions specified by the binary array $\mathbf{Q}$.

Switching between decimal n and equivalent binary $\mathbf{Q}$ labels is accomplished by the simple subroutines bintodec and dectobin in QCMPI. There we denote the binary number by an array $B(1)\cdots B(n_q)$. The routines are:

call bintodec(nq,B,D)	**call dectobin(nq,D,B)**

where **nq** is the number of qubits; **D** is a real decimal number and **B** is the equivalent binary array.

3. One-qubit operators

An important part of quantum computation is the act of rotating a qubit. The **NOT** and single qubit Hadamard $\mathcal{H}$ operators are of particular interest:

$$\mathbf{NOT} \equiv \sigma_x \rightarrow \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \mathcal{H} \equiv \frac{\sigma_x + \sigma_z}{\sqrt{2}} \rightarrow \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{15}$$

These have the following effect on the basis states $\mathbf{NOT}|0\rangle = |1\rangle$, $\mathbf{NOT}|1\rangle = |0\rangle$, and $\mathcal{H}|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$, $\mathcal{H}|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$.

General one-qubit operators can be constructed from the Pauli operators; we denote the general one-qubit operator acting on qubit s as Ω_s. Consider the action of such an operator on the multi-qubit state $|\Psi\rangle_{n_q}$:

$$\begin{aligned} \Omega_s|\Psi\rangle_{n_q} &= \sum_{\mathbf{Q}} C_{\mathbf{Q}} \Omega_s |\mathbf{Q}\rangle \\ &= \sum_{q_1=0,1} \cdots \sum_{q_s=0,1} \cdots \sum_{q_{n_q}=0,1} C_{\mathbf{Q}} |q_1\rangle \ldots \big(\Omega_s|q_s\rangle\big) \ldots |q_{n_q}\rangle. \end{aligned} \tag{16}$$

[7] An important aspect of relating the individual qubit state to a binary representation is that one can maintain the order of the qubits, since if a qubit hops over to another order the decimal number is altered.

Here Ω_s is assumed to act only on the qubit i_s of value q_s. The $\Omega_s|q_s\rangle$ term can be expressed as

$$\Omega_s|q_s\rangle = \sum_{q'_s=0,1} |q'_s\rangle\langle q'_s|\Omega_s|q_s\rangle, \tag{17}$$

using the closure property of the one qubit states. Thus Eq. (16) becomes

$$\begin{aligned}\Omega_s|\Psi\rangle_{n_q} &= \sum_{\mathbf{Q}} C_{\mathbf{Q}}\ \Omega_s|\mathbf{Q}\rangle \\ &= \sum_{q_1=0,1}\cdots\sum_{q_s=0,1}\cdots\sum_{q_{n_q}=0,1}\sum_{q'_s=0,1} C_{\mathbf{Q}}\langle q'_s|\Omega_s|q_s\rangle|q_1\rangle\ldots|q'_s\rangle\ldots|q_{n_q}\rangle.\end{aligned} \tag{18}$$

Now we can interchange the labels $q_s \leftrightarrow q'_s$, and use the label $\mathbf{Q}$ to obtain the algebraic result for the action of a one-qubit operator on a multi-qubit state

$$\Omega_s|\Psi\rangle_{n_q} = \sum_{\mathbf{Q}} \tilde{C}_{\mathbf{Q}}|\mathbf{Q}\rangle = \sum_{n=0}^{2^{n_q}-1} \tilde{C}_n|n\rangle, \tag{19}$$

where

$$\tilde{C}_{\mathbf{Q}} = \tilde{C}_n = \sum_{q'_s=0,1} \langle q_s|\Omega_s|q'_s\rangle C_{\mathrm{Q}'}, \tag{20}$$

where $\mathbf{Q} = (q_1, q_2, \ldots q_{n_q})$, and $\mathbf{Q}' = (q_1, \ldots q'_s \ldots q_{n_q})$. That is $\mathbf{Q}$ and $\mathbf{Q}'$ are equal except for the qubit acted upon by the one-body operator Ω_s.

A better way to state the above result is to consider Eq. (20) for the case that n has $q_s = 0$ and thus $n \to n_0$ and to write out the sum over q'_s to get

$$\tilde{C}_{n_0} = \langle 0|\Omega_s|0\rangle C_{n_0} + \langle 0|\Omega_s|1\rangle C_{n_1}, \tag{21}$$

where we introduced the partner to n_0 namely n_1. For the case that n has $q_s = 1$ and thus $n \to n_1$ Eq. (20), with expansion of the sum over q'_s yields

$$\tilde{C}_{n_1} = \langle 1|\Omega_s|0\rangle C_{n_0} + \langle 1|\Omega_s|1\rangle C_{n_1}, \tag{22}$$

or, written as a matrix equation, we have for each n_0, n_1 partner pair

$$\begin{pmatrix}\tilde{C}_{n_0}\\ \tilde{C}_{n_1}\end{pmatrix} = \begin{pmatrix}\langle 0|\Omega_s|0\rangle & \langle 0|\Omega_s|1\rangle\\ \langle 1|\Omega_s|0\rangle & \langle 1|\Omega_s|1\rangle\end{pmatrix}\begin{pmatrix}C_{n_0}\\ C_{n_1}\end{pmatrix}. \tag{23}$$

This is not an unexpected result. Later we will denote the matrix element $\langle 0|\Omega_s|0\rangle$ as Ω_{s00}, etc.

Eq. (23) above shows how a 2×2 one-qubit operator Ω_s acting on qubit i_s changes the state amplitude for each value of n_0. Here, n_0 denotes a decimal number for a computational basis state with qubit i_s having the q_s value zero and n_1 denotes its partner decimal number for a computational basis state with qubit i_s having the q_s value one. They are related by

$$n_1 = n_0 + 2^{n_q - i_s}. \tag{24}$$

At times, we shall call $2^{n_q-i_s}$ the "stride" of the i_s qubit; it is the step in n needed to get to a partner. There are $2^{n_q}/2$ values of n_0 and hence $2^{n_q}/2$ pairs n_0, n_1. Eq. (23) is applied to each of these pairs. In QCMPI that process is included in the subroutine **OneOpA**.

Note that we have replaced the full $2^{n_q}\times 2^{n_q}$ one qubit operator by a series of $2^{n_q}/2$ sparse matrices. Thus we do not have to store the full $2^{n_q}\times 2^{n_q}$ but simply provide a 2×2 matrix for repeated use. Each application of the 2×2 matrix involves distinct amplitude partners and therefore the set of 2×2 operations can occur simultaneously and hence in parallel.

In the next section, the procedure for distributing the state vector over several processors is described along with the changes induced by the action of a one-body operator. Later this procedure is generalized to multi-qubit operators, using the same concepts.

3.1. Distribution of the state

The state of the multi-qubit system is described at any given time by the array of coefficients $C_n(t)$ for $n = 0, \ldots 2^{n_q} - 1$, see Eq. (14). The action of a one-qubit gate, assumed to act instantaneously, is specified by the rules discussed in the previous section. Now we wish to distribute these state-vector coefficients stored in "standard order" with increasing n, over a number of processors N_P. For convenience, we assume that the number of processors invoked is a power of two, i.e. $N_P = 2^p$ and thus we can distribute the C_n coefficients uniformly over those processors with $N_x = 2^{n_q}/N_P = 2^{n_q-p}$ amplitudes on each processor. In the code we denote N_x as **NPART**. So, for example, we place

$$\begin{aligned}&C_0 \cdots C_{N_x-1} && \text{on processor myid} = 0;\\ &C_{N_x} \cdots C_{2N_x-1} && \text{on processor myid} = 1;\\ &\quad\vdots\\ &C_{(N_P-1)N_x} \cdots C_{N_P N_x-1} && \text{on processor myid} = N_P - 1.\end{aligned} \tag{25}$$

Where **myid** is the processor number, from 0 to $N_p - 1$. Note that $N_P \cdot N_x = 2^{n_q}$. This distribution of the state over the N_P processors places a demand of 2^{n_q-p} on the memory of each processor. For 64 processors $p = 6$ and the memory required is down by a factor of 64; and for 4096 processors $p = 12$ and the memory required is down by a factor of 4096, etc. As the number of processors available increases, so will the memory demands on each processor.

However, life is not that simple. A one-qubit operator for a given partner pair n_0, n_1 often involves a C_{n_0} that is seated on one processor and a C_{n_1} that is seated on another processor. We need to deal with that situation, while respecting the scheme for standard order distribution of the amplitude coefficients. The first question that arises is when are the pairs C_{n_0}, C_{n_1} seated on the same processor? We call that being "seated in the same section", in analogy to theater seating. That is, we dub being located on a particular processor as having the same section, with the location of a particular amplitude within that section being called its "seat". With that language, it is simple to state the condition for an amplitude pair C_{n_0}, C_{n_1} being on the same processor; namely, that the difference (we call this the "stride") $n_0 - n_1 = 2^{n_q-i_s}$ be less than the distance $2^{n_q}/N_P = 2^{n_q-p}$ or simply $i_s > p$. If this condition is not satisfied, the stride is large enough to jump out of the section and thus require inter-processor communications. This result holds true if the number of processors is of the form $N_P = 2^p = 1, 2, 4, 8, 16 \ldots$. One can prove this rule by induction.

This condition $i_s > p$, indicates that the larger i_s, that is for qubits that are the least significant contributors to the state label n, the associated pairs of amplitudes reside on the same processor. In contrast, the smaller i_s are the qubits for which the pair amplitudes are the furthest away in processor number. The stride ranges from a value of 1 for $i_s = n_q$ (least significant qubit) to $2^{n_q}/2$ for $i_s = 1$ (most significant qubit). Carrying out the 2×2 matrix multiplication Eq. (23) is simple for those pairs on the same processor, but suitable transfer to the requisite processors must be implemented before one can perform the requisite 2×2 matrix multiplication. To carry out that process requires a way to identify the processor (e.g., the section assignment) and the location within that processor (the seat) and to interchange the amplitudes. The latter task is carried out using the MPI protocol, as discussed later.

3.2. Pair section, seat and MPI

Distribution of the 2^{n_q} amplitudes $C_0 \ldots C_{2^{n_q}-1}$ over the N_P processors, places $N_x = 2^{n_q}/N_P = 2^{n_q-p}$ amplitudes on each processor. As the state label n ranges from 0 to $2^{n_q} - 1$ one jumps between different processors. The relationship between the n label and the processor on which the associated amplitudes sits is simply: section $= \mathrm{Int}(n/N_x)$, where Int() means the integer part and the seat (i.e. location within that processor) is $seat = \mathrm{Mod}(n, N_x)$ which denotes modular arithmetic of base N_x. In the code N_x is called **NPART** and section is identical with **myid**, the processor number.

With the ability to identify the processor/location or section/seat assignment associated with each index n, the remaining task is to transfer the requisite amplitudes to the "correct" location. That task is carried out by the Message Passing Interface (MPI) commands **MPI_SEND** and **MPI_RECV**. We need to coordinate the various processors and exchange data during a calculation. The main reason MPI was developed over the last several decades is to enable efficient communication between processors during a computation.

Why use MPI? The MPI protocol affords many advantages for developing parallel processing codes. The main advantages are that: (1) MPI provides a standard set of routines that are easy to use and (2) MPI is flexible and works on many platforms.[8] Thus MPI proved perfect for our need to develop a user-friendly, flexible realization of the action of multi-qubit operators on state vectors in a parallel computing environment.

3.3. Action of one-qubit operator

The following figures (Figs. 1–2) illustrate the role played by MPI in transferring distributed amplitudes to appropriate processor locations when the one-qubit operator acts. We use the case of $n_q = 3$ or $2^3 = 8$ components as a simple example.

The first case takes the partner labels $n = 1, 3$, which corresponds to the binary numbers (001) and (011). Here we use the binary labels for the components and consider the special case of a one-qubit operator acting on **qubit 2** and assuming two processors $p = 1$ (see Fig. 1). For that case, the two amplitudes affected by the one-qubit operator reside on the same processor, i.e., they have just different seats in the same section. Thus there is no need for MPI data transfer.

Now consider the partner labels $n = 0, 4$, which correspond to the binary numbers (000) and (100). Again we use the binary labels for the components, but now consider the special case of a one-qubit operator acting on **qubit 1** and again assuming two processors. For this case, the two amplitudes affected by the one-qubit operator do not reside on the same processor, i.e., they are in different sections. Thus there is now an essential need for MPI data transfer, which involves sending and receiving as illustrated in Fig. 2. This entails two sends and two receives.

Of course, one needs to continue this process for the other three amplitude pairs $n = 1, 5$, $n = 2, 6$, and $n = 3, 7$. In general, we have $2^{n_q}/2$ partner pairs. Those pairs require six more sends and six more receives. An important issue is then to see if the time gained by invoking more processors wins out over the time needed for all of these MPI transfers. Another important concept is one of "balance", which involves the extent to which the various processors perform equally in time and storage (ideally we assume they are all exactly equivalent and in balance).

It is important to understand the above illustrations, because for more qubits and more processors and for two- and three- qubit operators, the steps are simply generalizations of these basic cases. Careful manipulation of the amplitudes, allows for spreading the amplitudes over many processors and using MPI to do the requisite data transfers for all kinds of operators and gates.

For the one qubit case, the steps here are called by the command

```
call OneOpA(nq,is,Op,psi,NPART, COMM)
```

[8] We have run our codes on the Pittsburgh and Barcelona supercomputers, and also on arrays of imacs.

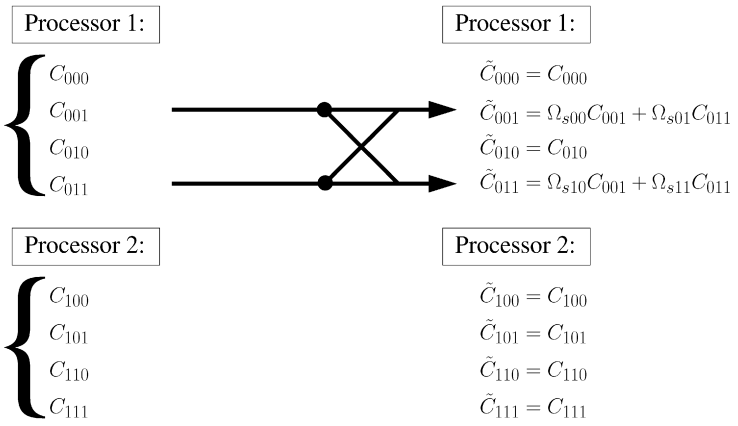

Fig. 1. A three qubit state vector is acted on by a one-qubit operator on qubit 2 ($i_s = 2$). The case illustrated is for the partners $n = 1, 3$, which correspond to the binary numbers (001) and (011). It is assumed that there are just two processors $N_P = 2^p$, with $p = 1$. Thus $i_s > p$ for this case and the two coupled amplitudes reside on the same processor and no MPI data transfer is invoked.

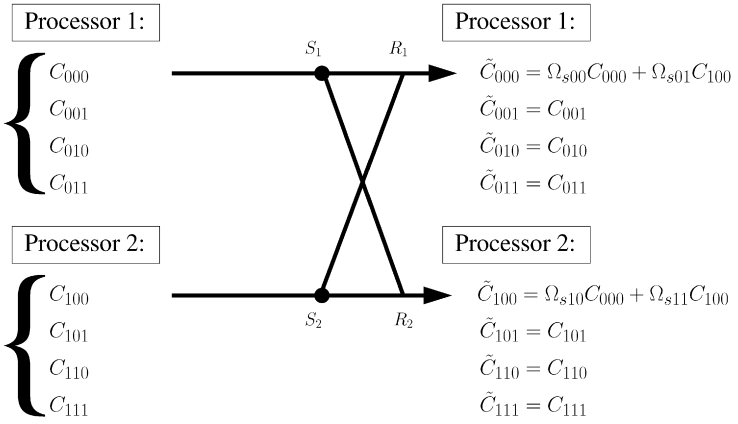

Fig. 2. A three qubit state vector is acted on by a one-qubit operator on qubit 1 ($i_s = 1$). The case illustrated is for the partners $n = 0, 4$, which corresponds to the binary numbers (000) and (100). It is assumed that there are just two processors $N_P = 2^p$, with $p = 1$. Thus the condition $i_s > p$ is not satisfied and indeed the two coupled amplitudes reside on different processors and MPI data transfer is invoked. We need to send (S_1) component C_{000} to processor one, and it is received at (R_1), and also send (S_2) component C_{100} to processor zero, and it is received at (R_2). Later we will specify the send and receive commands in the MPI language.

where nq denotes the number of qubits; "*is*" labels the qubit acted on by operator "Op", psi is an input wave function array of length **NPART** $= N_x$, which is returned as the modified state vector. The last entry **COMM** is included to allow for later extension to separate communication channels that we refer to as parallel universes or multiverses.

Let us emphasize that any operator, acting on one qubit is a special case of the one described here. Thus all rotations, all so-called local operations, including those generated by the one-body part of Hamiltonian evolution, are covered by the code **OneOpA**.

4. Multi-qubit operators

Let us return to the main issue of how to distribute the amplitudes over several processors and to cope with the action of operators on a quantum state. The case of a two-qubit operator is a generalization of the steps discussed for a one-qubit operator. Nevertheless, it is worthwhile to present those details, as a guide to those who plan to use and perhaps extend QCMPI.

We now consider a general two-qubit operator that we assume acts on qubits i_{s_1} and i_{s_2}, each of which ranges over the full $1, \ldots, n_q$ possible qubits. General two-qubit operators can be constructed from tensor products of two Pauli operators; we denote the general two-qubit operator as $\mathcal{V}$. Consider the action of such an operator on the multi-qubit state $|\Psi\rangle_{n_q}$:

$$\begin{aligned}\mathcal{V}|\Psi\rangle_{n_q} &= \sum_{\mathbf{Q}} C_{\mathbf{Q}}\mathcal{V}|\mathbf{Q}\rangle \\ &= \sum_{q_1=0}^{1}\cdots\sum_{q_{s1},q_{s2}=0}^{1}\cdots\sum_{q_{n_q}=0}^{1} C_{\mathbf{Q}}|q_1\rangle\ldots\big(\mathcal{V}|q_{s1}q_{s2}\rangle\big)\ldots|q_{n_q}\rangle. \end{aligned} \tag{26}$$

Here $\mathcal{V}$ is assumed to act only on the two i_{s1}, i_{s2} qubits. The $(\mathcal{V}|q_{s1}q_{s2}\rangle)$ term can be expressed as

$$\boldsymbol{\mathcal{V}}|q_{s1}q_{s2}\rangle = \sum_{q'_{s1},q'_{s2}=0}^{1} |q'_{s1}q'_{s2}\rangle\langle q'_{s1}q'_{s2}|\boldsymbol{\mathcal{V}}|q_{s1}q_{s2}\rangle \tag{27}$$

using the closure property of the two-qubit product states. Thus Eq. (27) becomes

$$\mathcal{V}|\Psi\rangle_{n_q} = \sum_{\mathbf{Q}} C_{\mathbf{Q}} \mathcal{V}|\mathbf{Q}\rangle = \sum_{q_1=0}^{1} \cdots \sum_{q_{s1}=0}^{1} \cdots \sum_{q_{s2}=0}^{1} \cdots \sum_{q_{nq}=0}^{1} \sum_{q'_{s1},q'_{s2}=0}^{1} C_{\mathbf{Q}} \langle q'_{s1} q'_{s2}|\boldsymbol{\mathcal{V}}|q_{s1} q_{s1}\rangle |q_1\rangle \ldots |q'_{s1} q'_{s2}\rangle \ldots |q_{n_q}\rangle. \tag{28}$$

Now we can interchange the labels $q_{s1} \leftrightarrow q'_{s1}, q_{s2} \leftrightarrow q'_{s2}$ and use the label $\mathbf{Q}$ to obtain the algebraic result for the action of a two-qubit operator on a multi-qubit state

$$\mathcal{V}|\Psi\rangle_{n_q} = \sum_{\mathbf{Q}} \tilde{C}_{\mathbf{Q}} |\mathbf{Q}\rangle = \sum_{n=0}^{2^{n_q}-1} \tilde{C}_n |n\rangle, \tag{29}$$

where

$$\tilde{C}_{\mathbf{Q}} = \tilde{C}_n = \sum_{q'_{s1},q'_{s2}=0}^{1} \langle q_{s1} q_{s2}|\Omega_s|q'_{s1} q'_{s2}\rangle \; C_{\mathbf{Q}'}, \tag{30}$$

where $\mathbf{Q} = (q_1, q_2, \ldots q_{n_q})$, and $\mathbf{Q}' = (q_1, \ldots q'_{s1} \ldots q'_{s2} \ldots q_{n_q})$. That is $\mathbf{Q}$ and $\mathbf{Q}'$ are equal except for the qubits acted upon by the two-body operator $\boldsymbol{\mathcal{V}}$.

A better way to state the above result is to consider Eq. (30) for the following four choices

$$\begin{aligned}
n_{00} &\rightarrow (q_1 \cdots q_{s1} = 0 \cdots q_{s2} = 0, \ldots q_{n_q}),\\
n_{01} &\rightarrow (q_1 \cdots q_{s1} = 0 \cdots q_{s2} = 1, \ldots q_{n_q}),\\
n_{10} &\rightarrow (q_1 \cdots q_{s1} = 1 \cdots q_{s2} = 0, \ldots q_{n_q}),\\
n_{11} &\rightarrow (q_1 \cdots q_{s1} = 1 \cdots q_{s2} = 1, \ldots q_{n_q}),
\end{aligned} \tag{31}$$

where the computational basis state label $n_{q_{s1},q_{s2}}$ denotes the four decimal numbers corresponding to $\mathbf{Q} = (q_1, \ldots q_{s1} \ldots q_{s2} \ldots q_{n_q})$.

Evaluating Eq. (30) for the four choices Eq. (31) and completing the sums over q'_{s1}, q'_{s2}, the effect of a general two-qubit operator on a multi-qubit state amplitudes is given by a 4×4 matrix

$$\begin{pmatrix} \tilde{C}_{n_{00}} \\ \tilde{C}_{n_{01}} \\ \tilde{C}_{n_{10}} \\ \tilde{C}_{n_{11}} \end{pmatrix} = \begin{pmatrix} \mathcal{V}_{00;00} & \mathcal{V}_{00;01} & \mathcal{V}_{00;10} & \mathcal{V}_{00;11} \\ \mathcal{V}_{01;00} & \mathcal{V}_{01;01} & \mathcal{V}_{01;10} & \mathcal{V}_{01;11} \\ \mathcal{V}_{10;00} & \mathcal{V}_{10;01} & \mathcal{V}_{10;10} & \mathcal{V}_{10;11} \\ \mathcal{V}_{11;00} & \mathcal{V}_{11;01} & \mathcal{V}_{11;10} & \mathcal{V}_{11;11} \end{pmatrix} \begin{pmatrix} C_{n_{00}} \\ C_{n_{01}} \\ C_{n_{10}} \\ C_{n_{11}} \end{pmatrix}, \tag{32}$$

where $\mathcal{V}_{ij;kl} \equiv \langle i, j|\mathcal{V}|k, l\rangle$. Eq. (32) above shows how a 4×4 two-qubit operator $\mathcal{V}$ acting on qubits i_{s1}, i_{s2} changes the state amplitude for each value of n_{00}. Here, n_{00} denotes a decimal number for a computational basis state with qubits i_{s1}, i_{s2} both having the values zero and its three partner decimal numbers for a computational basis state with qubits i_{s1}, i_{s2} having the values $(0, 1), (1, 0)$ and $(1, 1)$, respectively. The four partners $n_{00}, n_{01}, n_{10}, n_{11}$, or "amplitude quartet", coupled by the two-qubit operator are related by:

$$n_{01} = n_{00} + 2^{n_q - i_{s2}}, \qquad n_{10} = n_{00} + 2^{n_q - i_{s1}}, \qquad n_{11} = n_{00} + 2^{n_q - i_{s1}} + 2^{n_q - i_{s2}}, \tag{33}$$

where i_{s2}, i_{s2} label the qubits that are acted on by the two-qubit operator.

There are $2^{n_q}/4$ values of n_{00} and hence $2^{n_q}/4$ amplitude quartets $n_{00}, n_{01}, n_{10}, n_{11}$. Eq. (32) is applied to each of these quartets for a given pair of struck qubits. In QCMPI that process is included in the subroutine TwoOpA.

In this treatment, we are essentially replacing a large sparse matrix, by a $2^{n_q}/4$ set of 4×4 matrix actions, thereby saving the storage of that large matrix.

In the next section, the procedure for distributing the state vector over several processors is illustrated along with the changes induced by the action of a two-body operator.

4.1. Action of two-qubit operators

To visualize the distribution of the amplitudes over several processors and the role played by MPI in transferring the amplitudes to appropriate location, when the *two*-qubit operator acts, the following diagrams (Figs. 3, 4) lay out the scheme. We again use the case of $n_q = 3$ or $2^3 = 8$ components as a simple illustration.

The first case takes the amplitude quartet labels $n = 0, 1, 2, 3$ which corresponds to the binary numbers (000), (001), (010), and (011). Here we use the binary labels for the components and consider the special case of a two-qubit operator acting on **qubits 2 and 3.** We consider just two processors. In this case the four amplitudes affected by the two-qubit operator reside on the same processor, i.e., they have just different seats in the same section. Thus there is no need for MPI data transfer.

Now consider the amplitude quartet labels $n = 0, 2, 4, 6$, which corresponds to the binary numbers (000), (010), (100), and (110). Again we use the binary labels for the components, but now consider the special case of a two-qubit operator acting on **qubits 1 and 2.** We consider just two processors. For this case, the two amplitudes affected by the one-qubit operator do not reside on the same processor, i.e., they are in different sections. Thus there is now an essential need for MPI data transfer, which involves sending and receiving as illustrated in Fig. 4.

For the two qubit case, the steps here are called by the command

call TwoOpA(nq,is1,is2,Op,psi,NPART,COMM)

where **nq** denotes the number of qubits; **is1**, **is2** label the qubits acted on by operator **Op**, **psi** is an input wave function array of length **NPART** $= N_x$, which is returned as the modified state vector.

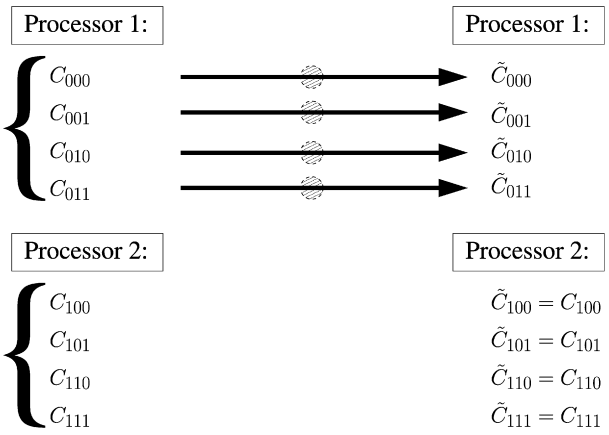

Fig. 3. A three qubit state vector is acted on by a two-qubit operator on qubits 2 and 3 ($i_{s1} = 2$, $i_{s2} = 3$). The case illustrated is for the amplitude quartet $n = 0, 1, 2, 3$, which corresponds to the binary numbers (000), (001), (010), and (011). It is assumed that there are just two processors $N_P = 2^p$, with $p = 1$. Thus $i_{s2} > i_{s1} > p$ for this case and the two coupled amplitudes reside on the same processor and no MPI data transfer is invoked. The dashed circles indicate that all four amplitudes contribute to forming the values of $\tilde{C}_{000}, \tilde{C}_{001}, \tilde{C}_{010}, \tilde{C}_{011}$ are given by Eq. (30).

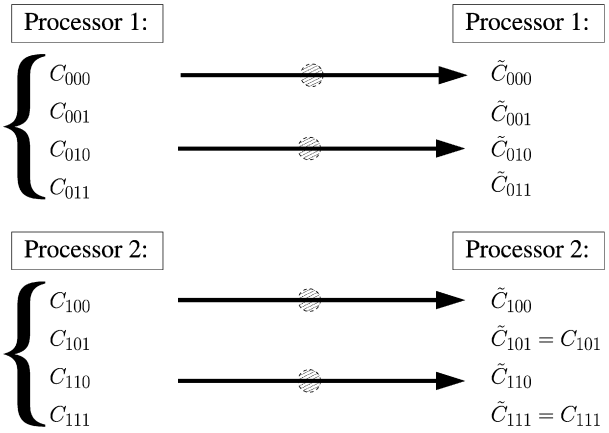

Fig. 4. A three qubit state vector is acted on by a two-qubit operator on qubits 1 and 2 ($i_{s1} = 1$, $i_{s2} = 2$). The case illustrated is for the amplitude quartet $n = 0, 2, 4, 6$, which corresponds to the binary numbers (000), (010), (110), and (110). It is assumed that there are just two processors $N_P = 2^p$, with $p = 1$. Thus $i_{s2} > p$, but we do **not** have $i_{s1} > p$, thus for this case amplitudes reside on different processors and MPI data transfer is invoked. The dashed circles indicate that all four amplitudes are to be sent/received from other locations.

4.2. CNOT and CPHASE

The two-qubit operators **CNOT** and **CPHASE** are oft-used special cases of the above two-qubit operator discussion. They are simpler than the general case because they are given by the sparse matrices

$$\mathcal{V} \to \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \qquad \mathcal{V} \to \text{CPHASE} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{34}$$

CNOT stores the rule $00 \to 00, 01 \to 01, 10 \to 11, 11 \to 10$, where qubit 1 is the control, and qubit 2 gets acted on by σ_1 only when qubit 1 has a value of one. In QCMPI, a subroutine **CNOT** codes this special two-qubit operator:

```
call CNOT(nq,is1,is2,psi,NPART,COMM)
```

CPHASE stores the rule $00 \to 00$, $01 \to 01$, $10 \to 10$, $11 \to -11$, where qubit 1 is the control, and qubit 2 gets acted on by σ_3 only when qubit 1 has a value of one. In QCMPI, a subroutine **CPHASEA** codes this special two-qubit operator:

```
call CPHASEA(nq,is1,is2,psi,NPART,COMM)
```

Another two-qubit operator which plays a key role in the quantum Fourier transformation, is the **CPHASEK**, given by a sparse matrix that depends on a positive integer k

$$\mathcal{V} \to \text{CPHASEK} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{(2\pi i)/2^k} \end{pmatrix}. \tag{35}$$

In QCMPI, a subroutine **CPHASEK** codes this special two-qubit operator:

call CPHASEK(nq,is1,is2,k, psi,NPART,COMM)

Note that there are no MPI commands required in this subroutine.

4.3. The full Hadamard–special handling

An important example of a multi-qubit operation is when Hadamards act on all of the qubits in a system—a step that is often used to initialize a QC. One way to do that is simply to repeat the prior discussion and use the subroutine for qubits $i_s = 1 \dots n_q$. That procedure is implemented in the subroutine **HALL**.

Hadamards acting on all qubits involves the operator

$$\mathcal{H}^{n_q} \equiv [\mathcal{H}_1 \otimes \mathcal{H}_2 \cdots \otimes \mathcal{H}_{n_q}]. \tag{36}$$

Another way to implement this operator is based on the realization that when acting on the column vector $(C_0 \dots C_{2^{n_q}-1})$ it forms an equal weighted combination with particular signs $s_{n.n'}$, whereby the effect of $\mathcal{H}^{n_q}$ is

$$C_n \to \frac{1}{2^{\frac{n_q}{2}}} \sum_{n'=0}^{2^{n_q}-1} s_{n,n'} C_{n'} \equiv \tilde{C}_n. \tag{37}$$

The task is to determine the signs $s_{n,n'}$. These signs are relatively simple to pin down. From the product structure of $\mathcal{H}^{n_q}$ it is simple to show that the signs are determined by the condition $s_{n,n'} = (-1)^{\delta}$, where δ denotes the number of times the binary bits for n, n' of unit value are equal, i.e. how many times $B(i) = B'(i) = 1$. This condition is carried out in the function SH:

FUNCTION SH(nq,n,np)

where nq is the number of qubits; $np = n'$. This procedure is implemented in the subroutine **HALL2**. The user should decide which version **HALL** or **HALL2** works best in their context.

Ironically, although a small subroutine, **HALL** or **HALL2** is used repeatedly in Grover's search and dominate the time expended in a large qubit quantum search. We shall refer to the operator $\mathcal{H}^{n_q}$ as **HALL** throughout this paper, recognizing that it can be implemented using either **CALL HALL** or by the special sign handling method **CALL HALL2**.

5. A sample of relevant quantum algorithms

QCMPI permits the simulation of any quantum circuit on a parallel computing environment. In this section we describe two well-known QC algorithms already included in the current package and which exemplify the use of QCMPI in practical applications.

5.1. Grover's searching algorithm

We now show how to apply the operators (gates), and the treatment of a multi-qubit state, to the first of several basic QC algorithms. These are standard procedures in QC and we examine them with QCMPI so that one can describe these algorithms dynamically using basic, realistic Hamiltonians and also subject these procedures to environmental effects. The case of superdense coding, [14] which is a way to enhance communication between Alice and Bob by means of shared entangled states, has also been developed in QCMPI.

Our first application presented here is Grover's search algorithm [4]. In this case, we start with a state of n_q qubits all pointing up $|000\dots0\rangle$, and act on it with **HALL**, see Eq. (36). Then, we need an all-knowing Oracle operator **O**racle to mark an item that is to be ferreted out by the algorithm. The Oracle step is very simple when we use amplitude coefficients $C(n)$; we simply find the processor (section) and location on that processor (seat) associated with the marked item n_x and reverse the sign of that amplitude $C(n_x) \to -C(n_x)$. All other amplitudes are unchanged.

$$\mathbf{O}\text{racle} \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{n_x} \\ \vdots \\ C_{2^{n_q}-1} \end{pmatrix} \to \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ -C_{n_x} \\ \vdots \\ C_{2^{n_q}-1} \end{pmatrix}. \tag{38}$$

This process can be extended to two or more marked items.

Grover's procedure, which entails using constructive interference to make the marked item's amplitude stand out from all others, involves acting repeatedly on the state **HALL**$|000\dots0\rangle$ with the "Grover operator"

$$\mathcal{G} \equiv \text{HALL} \cdot \mathcal{INV} \cdot \text{HALL} \cdot \mathbf{O}\text{racle}, \tag{39}$$

where $\mathcal{INV}$ is an operator

$$\mathcal{INV} = 2|000\cdots0\rangle\langle000\cdots0| - \mathbf{I}, \tag{40}$$

where **I** is an $2^{n_q} \times 2^{n_q}$ unit matrix. The operator $\mathcal{INV}$ is simply realized by changing the sign of all amplitudes except for the $n = 0$ one.

$$\mathcal{INV} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ \vdots \\ C_{2^{n_q}-1} \end{pmatrix} \to \begin{pmatrix} +C_0 \\ -C_1 \\ -C_2 \\ \vdots \\ -C_{2^{n_q}-1} \end{pmatrix}. \tag{41}$$

The combination **HALL** $\cdot\, \mathcal{INV} \cdot$ **HALL** is called an inversion about the mean and is an essential part of Grover's algorithm. The other essential part is to act on the initial state enough times n_t to produce an amplitude $\tilde{\mathbf{C}}(n_x)$ that stands out with high probability. We take the number $n_t = \frac{\pi}{4}\sqrt{2^{n_q}}$.

$$G^{n_t} \cdot \mathrm{HALL}|000\cdots 0\rangle \to \begin{pmatrix} \tilde{C}_0 \\ \tilde{C}_1 \\ \vdots \\ \tilde{\mathbf{C}}_{n_x} \\ \vdots \\ \tilde{C}_{2^{n_q}-1} \end{pmatrix}. \tag{42}$$

In all of these simple states, it is the **HALL** operator and its repeated application in G^{n_t} that involves the most time expenditure.

The QCMPI Grover search for a large number of qubits is included as Guniv.f90. Instructions for running the code and a guide to steps invoked are incorporated directly as comments in the listing. Generalization to include noise using a multi-universe approach is discussed later.

5.2. Shor's factoring algorithm

Shor's algorithm [5] is a QC method for factoring a large number. The basic idea is to prepare a state that, when subjected to a Quantum Fourier Transform (QFT), permits one to search for the period of a function that reveals the requisite factors with high probability. It uses quantum enhancement to go way beyond classical factoring procedures and yields the factors with high probability for very large non-prime numbers, after relatively few tries. To simulate this algorithm, where we are restricted to numbers much smaller than possible in a future QC, there are several steps implemented in the QCMPI code subshor.f90. A pedagogic analysis of the reasons for each step of Shor's algorithm is presented in Ref. [19].

Step 1. Choose the number, M, and set the register sizes

Choose the number, M, to be factorable number: $15, 21, 33, 35, 39, 55, 77\ldots$ and determine the size of two requisite registers.

Preparatory tests on the input number M are made so that the code continues only if the input number is not a power of 2 or a prime and is thus a suitable candidate for factoring. These are classical procedures performed by standard codes. Then, based on having an acceptable input number, an initial state of two distinct registers are prepared, with register one having n_1 and register two having n_2 qubits. The first register should [5,19] have enough qubits to store in base 2 all numbers in the range M^2 to $2M^2$, i.e. $M^2 \geqslant 2^{n_1} < 2M^2$. Therefore the choice for n_1 is set as

$$n_1 = \mathrm{Ceiling}\big(\log_2(Q)\big), \qquad Q = 2^{\mathrm{Ceiling}(\log_2(M^2))}, \tag{43}$$

where Ceiling(x) gives the smallest integer greater than or equal to x.[9]

The number of qubits in register two is set by

$$n_2 = \mathrm{Ceiling}\big(\log_2(M)\big) \tag{44}$$

so that there are enough qubits to store in base 2 all numbers up to and including the value of the input number M.[10]

Here we invoke the minimum number of qubits for both registers. A larger n_1 lengthens the computation, albeit providing higher probability of success.

Step 2. Load the first register

Load the first register with all the integers less than or equal to $2^{n_1} - 1$.

This is achieved by acting with a Hadamard on all qubits in register one, that is use **HALL** on a basis state of n_1 qubits so that register one is set to the state

$$|\Psi_1\rangle = \mathrm{HALL}|00\ldots 0\rangle_{n_1} = \frac{1}{2^{\frac{n_1}{2}}} \sum_{n=0}^{2^{n_1}-1} |n\rangle_{n_1}. \tag{45}$$

Thus each of the 2^{n_1} amplitudes appear with equal weight, which is the quantum massive parallel processing feature.

Next we attend to setting register two.

[9] For example, if $M = 21$, and $M^2 = 441$, then $n_1 = 9$ and $2^9 = 512$, and register one includes the value 441. If n_1 were taken as 8, then register one would not include the value 441, since $2^8 = 256$. If n_1 were taken as 10, then register one would exceed the value $2M^2 = 882$, since $2^{10} = 1024$.

[10] For example, if $M = 21$, then $n_2 = 5$ and $2^5 = 32 > M$, and register two includes the value 21. If n_2 were taken as 4, then register two would not include the value 21, since $2^4 = 16$.

Step 3. Load the second register

Select an integer (xguess) coprime to M and load the function $f(j) \equiv \text{xguess}^j \pmod M$ into the second register for a fixed choice of xguess and all possible j values: $0 \leqslant j \leqslant 2^{n_1} - 1$.

Note that in QCMPI, we simply compute $|f(n)\rangle$ and tack it on to the $|n\rangle_{n_1}$ state. As discussed by Shor [5], one must actually do this crucial step by a quantum process for modular exponentiation.

In Shor's algorithm, xguess is a random choice for a number that is coprime to M, where coprime means that M and xguess have no common factor other than 1. Euler's phi function of M is used to determine the range of integers between 1 and M that are coprime to M; a number in this interval a is selected and then tested using $\text{GCD}[a, M] \equiv 1$ to assure that it is coprime to M. If it passes that test we set the value for $a \to \text{xguess}$.

The reason for defining the above function (aka the Shor Oracle) is that this function $f(j)$ has a characteristic period for each value of M and xguess.

$$\big|f(n+r)\big\rangle = \big|f(n)\big\rangle, \qquad f(n) = \text{xguess}^n \pmod M. \tag{46}$$

Finding the period r is a key goal.

The full state composed of registers 1 and 2 ($n_q = n_1 + n_2$) is built in the following way:

$$|\Psi\rangle_{n_q} = \frac{1}{2^{n_1/2}} \sum_{n=0}^{2^{n_1}-1} |n\rangle_{n_1} \big|f(n)\big\rangle_{n_2}. \tag{47}$$

Step 4. Measure register 2

We could measure the second register next or postpone that act to coincide with step 6 below, because step 5 involves register 1 only. It is helpful to think of the action of measuring register 2 now to motivate the need for step 6. For each possible value of $n_1 \to k$, one asks if register 2 is in state ${}_{n_2}\langle k|$ by projecting the full state

$${}_{n_2}\langle k|\Psi\rangle_{n_q} = \frac{1}{2^{n_1/2}} \sum_{n=0}^{2^{n_1}-1} |n\rangle_{n_1} \big\langle k\big|f(n)\big\rangle_{n_2} \to \frac{1}{D^{1/2}} \sum_{j=0}^{D-1} |n_k + jr\rangle_{n_1}, \tag{48}$$

where at the last stage the state is normalized after projection–the usual Born rule for a projective measurement. Note that for every choice of k, D terms of the first register appear in superposition where D is $\approx 2^{n_1}/r$. The integer D is ascertained[11] by the period r for a fixed xguess by

$$\big|f\big(n_k + (D-1)r\big)\big\rangle = \big|f\big(n_k + (D-2)r\big)\big\rangle = \cdots = \big|f(n_k + 1r)\big\rangle = \big|f(n_k)\big\rangle = k. \tag{49}$$

The next step involves acting on register 1 to search for the period r by enhancing the quantum interference using a quantum Fourier transform.

Step 5. Quantum Fourier Transform register 1

Register one, which is now in the state

$$|\Phi\rangle_{n_1} = \frac{1}{\sqrt{D}} \sum_{j=0}^{D-1} |n_k + jr\rangle_{n_1}, \tag{50}$$

is next acted upon by a quantum Fourier transformation operator QFT (see Appendix A for a discussion of the QFT operator) which changes the above state

$$\begin{aligned} \text{QFT}|\Phi\rangle_{n_1} &= \frac{1}{\sqrt{2^{n_1} D}} \sum_{j=0}^{D-1} \sum_{n=0}^{2^{n_1}-1} e^{2\pi i n(n_k + jr)/2^{n_1}} |n\rangle_{n_1} \\ &= \frac{1}{\sqrt{2^{n_1} D}} \sum_{n=0}^{2^{n_1}-1} e^{2\pi i n n_k/2^{n_1}} \sum_{j=0}^{D-1} \big(e^{2\pi i n r/2^{n_1}}\big)^j |n\rangle_{n_1}. \end{aligned} \tag{51}$$

The QFT is a unitary operator which switches to a basis in which the superposition is isolated into the above exponential amplitude. The sum on j can be performed[12] and thus the result is

$$\text{QFT}|\Phi\rangle_{n_1} = \frac{1}{\sqrt{2^{n_1} D}} \sum_{n=0}^{2^{n_1}-1} e^{2\pi i n n_k/2^{n_1}} e^{\pi i n(D-1)/2^{n_1}} \frac{\sin(D\pi n \frac{r}{2^{n_1}})}{\sin(\pi n \frac{r}{2^{n_1}})} |n\rangle_{n_1}. \tag{52}$$

The probability for finding the final state with register 1 in state $\langle n|$ and register 2 in state $\langle k|$ is therefore

$$p(n,k) = \frac{1}{2^{n_1} D} \left[\frac{\sin(D\pi n \frac{r}{2^{n_1}})}{\sin(\pi n \frac{r}{2^{n_1}})}\right]^2, \tag{53}$$

[11] The value of D is constrained by the conditions $0 \leqslant n_k + (D-1)r \leqslant 2^{n_1} - 1$ and $0 \leqslant n_k \leqslant r - 1$. Hence, the integer D is constrained by $D \leqslant \frac{2^{n_1}}{r} + \frac{r-1-n_k}{r}$.

[12] The following summation rule is used here $\sum_{j=0}^{D-1} X^j = \frac{X^D-1}{X-1}$. One can also display this as 2-D vector additions of equal length phases, as in Fresnel zone plate interference.

where the dependence on k has dropped out (except for a possible dependence of D on k-see earlier footnote). Note that for the special case that $nr/2^{n_1}$ is an integer, the above result reduces to $p(n,k) \to \frac{D}{2^{n_1}} \approx \frac{1}{r}$, which is understood by returning to Eq. (51) and using $\sum_{j=0}^{D-1}(1)^j = D$.

How can one extract the period r from making such a measurement on registers 1 and 2?

Step 6. Measure register 1 and determine period and factors

The measurement of registers 1 and 2 has a probability given by Eq. (53). At select values of $n \to \bar{n}$, the probability $p(n,k)$ has local maxima. Consider the associated fraction $\frac{\bar{n}}{2^{n_1}}$, which is extracted from a determination of those local maxima. At these maxima

$$p_{\max} = \frac{1}{2^{n_1} D}\left[\frac{\sin(D\pi r\frac{\bar{n}}{2^{n_1}})}{\sin(\pi r\frac{\bar{n}}{2^{n_1}})}\right]^2. \tag{54}$$

In the arguments of the sin functions in Eq. (54), D is an integer, so the maximum probability occurs in the vicinity of an integer value for $r\frac{\bar{n}}{2^{n_1}}$. We therefore seek an approximate value of the ratio $\frac{\bar{n}}{2^{n_1}} \approx \text{integer}/r$, for an even r. That ratio is found by expressing $\frac{\bar{n}}{2^{n_1}}$ as a continued fraction and determining its first convergent of the form integer$/r$, for an even r.[13]

That determines the value of the period r, which we require to be even so that we can use the final step[14]

$$f_1 = \text{GCD}\big[\text{xguess}^{r/2}+1, M\big]; \qquad f_2 = \text{GCD}\big[\text{xguess}^{r/2}-1, M\big] \tag{55}$$

to determine the factors f_1, f_2 of M. The above process simplifies if the ratio $\frac{2^{n_1}}{r} = D$ is already an integer.

In QCMPI the local probability maxima and the associated factors are all stipulated. In an actual measurement, one of those results would be found with that probability.

6. Parallel universe and noise

A real quantum computer will involve the manipulation of qubits using external fields and interactions with single qubits and between qubits. Clearly, each physical realization has its set of Hamiltonians that describe that system and these QC manipulations. The circuit description of QC involves gates, which in turn should be described by the action of Hamiltonians on qubits. For example, the simple one-qubit Hadamard gate can be realized by rotating the qubit's spin axis from the $\hat{z}$ to the $\hat{x}$ axis by means of a $-\vec{\mu}\cdot\vec{B}$ interaction acting for the proper time. More complicated gates involve clever design of one and two-qubit interactions. In the future, we hope that QCMPI will provide a tool for describing all requisite gates based on Hamiltonian evolution. Dynamical evolution involves one- (H_1) and two- (H_2) body Hamiltonians $|\Psi(t+\delta t)\rangle = [1 - \frac{i}{\hbar}(H_1+H_2)\delta t]\ |\Psi(t)\rangle$. Their action over a small time interval δt can be calculated by repeated application of the **OneOpA** and **TwoOpA** codes provided in QCMPI. Such applications are the subject for future studies.

The major obstacle to the implementation of such gates required for the success of QC algorithms is the strong possibility that random intrusions, such as noise, will decohere the quantum system and remove the essential feature of quantum interference. That issue behoves us to simulate the affect of noise by considering many replications of the QC algorithm, which ideally are identical, and then subject each of them to random single and double one-qubit as well as single two-qubit errors. For that task MPI is ideally suited and therefore, as a major part of this paper, we have implemented that "Parallel Universe" approach, for which we include herein the Grover and Shor algorithms. Other cases (teleportation and superdense coding) have also been implemented. Subsequent numerical studies of the efficacy of error correction protocols can be implemented using the framework provided by the parallel universe feature of QCMPI.

The next feature of this "Parallel Universe" approach is that all of the state vector amplitudes can be gathered together and used to construct an ensemble average in the form of a density matrix. This process corresponds to solving a set of stochastic Schrödinger equations [20] and using those solutions to produce a density matrix. Let us now examine the steps needed to construct a density matrix.

6.1. Density matrix

There are advantages to using a density matrix to describe QC dynamics. The density matrix describes an ensemble average of quantum systems, with its evolution determined not only by the system's Hamiltonian but also by environmental terms using either Kraus operator [17] or Lindblad [18] differential equation forms In addition, the description of entanglement and of mixed states is handled nicely and concepts like entropy and Fidelity can be evaluated more readily. To form a density matrix in QCMPI and to determine the entropy, affords a good example of how to extend QCMPI to such ensemble averages.

For a definite state vector, the pure state density matrix[15] is simply

$$\rho = |\Psi\rangle\langle\Psi| = \sum_{n=0}^{2^{n_q}-1}\sum_{n'=0}^{2^{n_q}-1} C^*(n')C(n)|n\rangle\langle n'|. \tag{56}$$

This large $(2^{n_q}\times 2^{n_q})$ matrix can be distributed over $N_P = 2^p$ processors by placing $2^{n_q-p}\times 2^{n_q-p}$ matrices on each processor.[16] Matrix multiplication, traces and eigenvalue determination can then be implemented using MPI procedures, supplemented by BLACS processor grid and parallel linear algebra SCALAPACK programs [16]. Once the eigenvalues of ρ are calculated the entropy can be determined. But

[13] Gerjuoy [19] showed that the maximum probability is not less than $\frac{4}{\pi^2} \approx 0.4$, but more likely to be $\geqslant \frac{8}{r\pi^2} \approx 0.81$.

[14] Note that the periodic function xguess^r Mod$[M]$ = xguess^0 Mod$[M] = 1$. For even period r this yields $(\text{xguess}^{r/2})^2 - 1 \equiv 0$ Mod$[M] = (\text{xguess}^{r/2}-1)(\text{xguess}^{r/2}+1)$. As long as xguess$^{r/2}$ is not one, at least one of (xguess$^{r/2}$)$^2 \pm 1$ must have a common factor with M, and therefore finding GCD[(xguess$^{r/2}$)$^2 \pm 1$] yields the factors of M.

[15] The density matrix is Hermitian, has unit trace, and is positive definite. In general $\rho^2 \leqslant \rho$, with the equal sign applied for pure states.

[16] To facilitate the parallel treatment of the density matrix, we take p as even.

for a pure state, we know that $\rho^2 \equiv \rho$, and since the trace of ρ is one, the eigenvalues for a pure state are 1 and $2^{n_q} - 1$ zeros. Thus the entropy is zero, as it should be for a well-defined, non-chaotic, albeit probabilistic state.

How do we go beyond a pure state density matrix within the QCMPI setup? There are several options, but one overall goal. The overall goal is to build a state $|\Psi_\alpha\rangle$ repeatedly as labeled by α, with an associated probability $\mathcal{P}_\alpha$ with $\sum_\alpha \ \mathcal{P}_\alpha = 1$. For each case, the state $|\Psi_\alpha\rangle$ could be generated in a different way. One option is to get a set of amplitudes $C_\alpha(n)$ randomly, with each random set assigned a probability $\mathcal{P}_\alpha$. Another way is to select a few qubits and subject them to random one and two body interactions and possible stochastic pulses (noise), again assigning each case a probability $\mathcal{P}_\alpha$. The associated mixed state density matrix would then be

$$\rho = \sum_\alpha \mathcal{P}_\alpha |\Psi_\alpha\rangle\langle\Psi_\alpha| = \sum_{n=0}^{2^{n_q}-1} \sum_{n'=0}^{2^{n_q}-1} \sum_\alpha \mathcal{P}_\alpha \ C^*_\alpha(n') C_\alpha(n) |n\rangle\langle n'|. \tag{57}$$

The above result can be expressed as[17]

$$\langle n|\rho|n'\rangle = \sum_\alpha \mathcal{P}_\alpha \ C_\alpha(n) C^*_\alpha(n'). \tag{58}$$

This is perhaps not the most general density matrix, but one can trace out some of the ancilla qubits and/or subject the density matrix to additional entangling operations using $\rho' = U\rho U^\dagger$ or even apply the non-unitary Lindblad [18] process to generate an enhanced range of density matrices. These procedures, which we outline here, are included in this version of QCMPI to facilitate studies of decoherence and environmental effects. A major advantage of QCMPI is that the invocation of parallel universes (aka multiverses) to describe the influence of noise on a QC does not involve much increase in computation time compared to a single pure run, especially since the only communication between groups is that used is to construct the density matrix. This scheme provides an efficient use of multi-processor computers.

6.2. Parallel universe implementation

The above steps are implemented in QCMPI by first splitting the overall number of processors N_P (nprocU) into many groups N_G, each group is referred to as a "multiverse". For convenience, we take both N_P and N_G to be powers of 2. Within each multiverse, there are $N_P/N_G \equiv N_g$ (nprocM) processors that are used to perform a distinct QC algorithm. The MPI command **MPI_COMM_SPLIT** is used to produce these separate groups. Each group is specified by its group rank (rankM), which ranges from zero to NGROUPS-1, where NGROUPS denoted the total number of multiverses.[18]

The method used to store and evaluate the density matrix is controlled by an integer **lentropy**. For the choice **lentropy** $= 0$, there is no evaluation of the density matrix. For the choice **lentropy** $= 1$, the full density matrix is constructed on the master processor and its eigenvalues determined by a LAPACK code. That procedure should be used when storage space for ρ is ample. For **lentropy** $= 2$ the density matrix is not stored on one processor, but is distributed on a BLACS generated processor grid and the parallel eigenvalue code **PCHEEVX** from the SCALAPACK package is invoked to evaluate $\rho's$ eigenvalues. To carry out this last task the number of processors, groups and qubits have to be carefully monitored for consistency with the codes conventions, as indicated directly in the listings.

6.3. Noise scenarios

A simple example of a "noise scenario" has been included[19] in QCMPI to show how the role of noise can be examined. The motivation here is to first introduce noise and later to evaluate various error correction schemes.

The division of a large number of processors into groups was made so that only the first group (rankM=0) functions without noise. All of the other groups perform the algorithm with noise. The noise is introduced separately for each group (or multiverse) where the users can design their own scenarios. We have input noise using a one-qubit unitary operator (subroutine D2) that we take as a 2×2 Wigner rotation function $\mathcal{D}^{1/2}(\alpha, \beta, \gamma)$, where α, β, γ are three Euler angles.[20] This can be specialized to either small deviations or, within a phase, to one of the Pauli operators. One can introduce one qubit noise, acting on a random qubit, typically once within each processor multiverse, but two or more one-qubit noise intrusions can be invoked at various stages of the algorithm, by suitable placement of the subroutine "Noise".

In addition, a two-qubit unitary operator (subroutine D4) that we take as a 4×4 Wigner function $\mathcal{D}^{3/2}(\alpha, \beta, \gamma)$, can also be specialized to either small deviations or, within a phase, to one of the Pauli operator products $\sigma_i \otimes \sigma_j$. This allows one to introduce a single error that acts on two qubits once, in contrast to two one-qubit errors.

The one-qubit operator is assumed to act on a random selected qubit (qhit) and at selected, variable stages of the algorithm (eloc). Extension to two-qubit noise is obvious. Of course the associated universes which allow two one-qubit or single two-qubit errors should carry lower weight.

By using unitary operators in each universe the overall density matrix still maintains unit trace, but of course the trace of ρ^2 will be decreased by noise. The probability of success will also decrease.

Thus, QCMPI provides a framework for introducing errors and, along with Hamiltonian-driven gates, provides an important tool for dynamical studies of QC with noise and in the future with error correction.

---

[17] An abbreviated version is $\rho = \sum_\alpha \mathcal{P}_\alpha |\alpha\rangle\langle\alpha|$, with $C_\alpha(n) = \langle n|\alpha\rangle$.

[18] There are spawning features of MPI-2 that might be invoked to carry out this process more efficiently, but at this stage we found MPI-1 sufficient for our needs.

[19] See subroutine Noise called in subgrover.f90 and subshor.f90.

[20] We take random α, β, and set $\gamma = 0$ for simplicity.

Table 1
Performance of a number of sample runs all realized using Guniv.f, subgrover.f90 and qcmpisubs.f90.

NP	nq	2^{nq}	Gflop/sec	Gbytes	Wallclock (sec)	% communication
4	10	1024	1.99618	1.4043	1.25	26.63
4	12	4096	3.31855	9.96069	48.98	5.65
16	10	1024	1.90923	5.54911	1.07	62.27
16	12	4096	10.4583	38.7235	11.29	39.14
64	10	1024	0.65133	22.1393	3.21	31.19
64	12	4096	15.2444	153.814	7.54	56.32

In all cases the lentropy=2 option is chosen and thus the entropy is computed making use of the SCALAPACK routines and two multiverses are considered. The Gflop/sec and Gbytes refer to the total amount used by all processors. The information presented has been obtained using IPM [22].

7. Fortran and MPI codes

Sample QCMPI codes are provided which incorporate directions as to how to run the code. From these examples, the user should be able to see the benefit of being able to handle problems with a considerable number of qubits, organized into parallel universes, in reasonable time. Some improvements could be invoked to accelerate QCMPI, for example, by collecting messages and sending them as a group (collective communications). The issue here is the standard fight between sharing the work load over the available processors (balance) and minimizing the cost of sending messages. However, the major benefit of dividing a large number of processors into multiverses and subjecting each one to separate noise scenarios, is in itself justification and reason to use a multi-processor supercomputer.

The list of files contained in QCMPI are:

- qcmpisubs.f90, contains all QCMPI subroutines,
- Guniv.f90, builds multiverse environment for Grover's search,
- subgrover.f90, Grover's search routine,
- Suniv.f90, builds multiverse setup for Shor's factoring algorithm,
- subshor.f90, Shor's factoring routine,
- makefile, sample of compiling options for several supercomputing facilities,
- *.job, sample job submitting scripts,
- README, instructions.

7.1. Performance

As an indication of the performance of QCMPI, we have run a number of sample cases using the multiverse Grover codes included in the package. In Table 1, we show the global memory requirements together with the wallclock time and the percentage of the latter used in MPI operations.

8. Conclusions and future developments

In conclusion, the Fortran 90 code QCMPI provides a modular approach to quantum algorithms that provides accessible implementation of quantum computation algorithms. All of the gates needed for the circuit model are provided, as well as the quantum Fourier transformation procedure. Extension to three-qubit operators and to the one-way model of computation are straightforward, as is the extension to the qubit case. Such extensions will be provided in the future by the authors and hopefully also by interested users.

The main features of QCMPI are the distribution of state-vector amplitudes over processors, to allow for increased number of qubits and the use of MPI to carry out the requisite communication needed when one- and two-body operators (gates) act on states. This task is carried out in a manner that allows ready extension to Hamiltonian driven QC dynamics.

In addition. QCMPI provides a multi-universe setup, which replicates the QC algorithm over many groups, at little cost in computation time. That procedure provides a major advantage of QCMPI, not generally available in the literature, to provide a framework for studying the role of noise on the efficacy of QC. That is, we believe, the major task in this subject.

The methods demonstrated here for the distribution and evaluation of a large density matrix can be generalized to the case of large unitary matrices to represent gates.

There is much to do with this tool such as studies of: Hamiltonian driven QC dynamics using realistic Hamiltonians, along with environmental effects, influence of random pulses, and efficacy of error correction protocols.

Acknowledgements

We gratefully acknowledge the help and participation of Prof. C.W. Finley at an early stage of this work. This project was supported in part by the U.S. National Science Foundation and in part under Grants PHY070002P and PHY070018N from the Pittsburgh Supercomputing Center, which is supported by several federal agencies, the Commonwealth of Pennsylvania and private industry. Circuit graphs were prepared using codes from Ref. [21] which we appreciate. Thanks to PSC staff members Dr. Roberto Gomez and Rick Costa. We also thank the OpenMPI and SCALAPACK groups, especially Julie Langou and Jeff Squyres.

Appendix A. The quantum Fourier transform circuit and QCMPI

The quantum Fourier transform is performed using the circuit in Fig. A.1. A ladder of Hadamards and two-qubit control CPHASEK gates (Eq. (35)) are used to produce the QFT.

The above steps are carried out in QCMPI with the following code, which includes a series of pair swaps to reorder the qubits labels.

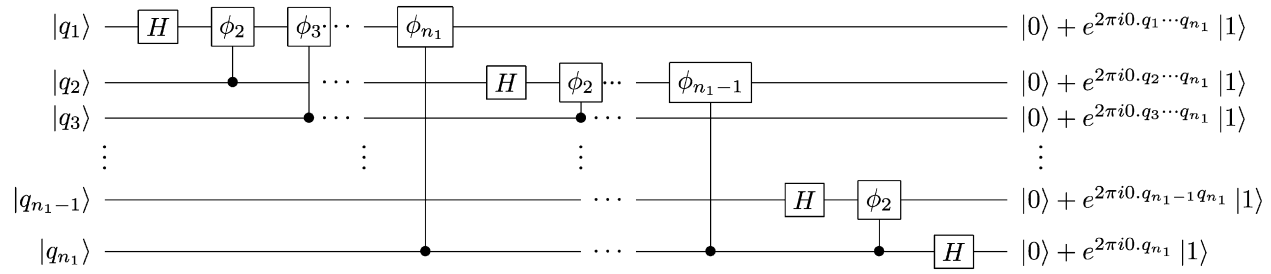

Fig. A.1. The quantum Fourier transform circuit. Here $\phi_k \equiv e^{2\pi i/2^k}$ and the register one has n_1 qubits. The binary number $q_1 q_2 \ldots q_{n_1}$ corresponds to a decimal number n which ranges as $0 \leqslant n \leqslant 2^{n_1} - 1$. The fraction binary notation is used where $0.q_a \ldots q_b \equiv \frac{q_a}{2} + \frac{q_{a+1}}{2^2} + \cdots + \frac{q_b}{2^{b-a+1}}$. To restore the qubits to standard order with the most significant bit to the left (top of figure), an addition reversal of the qubit order must be applied. An overall normalization of $1/2^{n_1/2}$ is understood.

```
Do ic =1,n1-1
call OneOpA(nq,ic,had,psi,NPART,COMM)
Do k=ic+1,n1
call CPHASEK(nq,k,ic,k+1-ic,psi,NPART,COMM)
enddo
enddo
! Final Hadamard
call OneOpA(nq,n1,had,psi,NPART,COMM)

! Reverse order using pair swaps
Do i=1,n1/2
call SWAP(nq,i,n1+1-i,Psi,NPART,COMM)
enddo
```

Here **had** denotes the Hadamard, **psi** is the input and then the output state vector at each stage, and **NPART** denotes the part of **psi** on the current processor. The qubits are restored to standard order by a set of pair swaps. This also demonstrates how to use the **CPHASEK, OneOP** (for a Hadamard case), and the **SWAP** subroutine.

To understand how the ladder of Hadamard and control phase gates yields a quantum Fourier transform, note that the final state shown in the code,

$$|q_1 q_2 \ldots q_{n_1}\rangle \to \frac{1}{\sqrt{2^{n_1}}}\big(|0\rangle + e^{2\pi i 0.q_1 \ldots q_{n_1}}|1\rangle\big) \otimes \big(|0\rangle + e^{2\pi i 0.q_2 \cdots q_{n_1}}|1\rangle\big) \otimes \cdots \big(|0\rangle + e^{2\pi i 0.q_{n_1-1}q_{n_1}}|1\rangle\big) \otimes \big(|0\rangle + e^{2\pi i 0.q_{n_1}}|1\rangle\big), \tag{A.1}$$

which becomes

$$|q_1 q_2 \cdots q_{n_1}\rangle \to \frac{1}{\sqrt{2^{n_1}}}\big(|0\rangle + e^{2\pi i 0.q_{n_1}}|1\rangle\big) \otimes \big(|0\rangle + e^{2\pi i 0.q_{n_1-1}q_{n_1}}|1\rangle\big) \otimes \cdots \big(|0\rangle + e^{2\pi i 0.q_2 \cdots q_{n_1}}|1\rangle\big) \otimes \big(|0\rangle + e^{2\pi i 0.q_1 \cdots q_{n_1}}|1\rangle\big), \tag{A.2}$$

after the final qubits are reordered by a series of pair swap operations. The last result can be written as:

$$\mathrm{QFT}|q_1 q_2 \ldots q_{n_1}\rangle = \frac{1}{\sqrt{2^{n_1}}} \sum_{Q'} e^{2\pi i [q'_1 0.q_{n_1} + q'_2 0.q_{n_1-1}q_{n_1} \cdots + q'_{n_1-1} 0.q_2 \ldots q_{n_1} + q'_{n_1} 0.q_1 \ldots q_{n_1}]} |Q'\rangle, \tag{A.3}$$

where Q' denotes the binary number $q'_1 q'_2 \ldots q'_{n_1}$, corresponding to the decimal number n'. The above is equivalent to

$$\mathrm{QFT}|n\rangle = \frac{1}{\sqrt{N}} \sum_{n'=0}^{N-1} e^{2\pi i n n'/N} |n'\rangle, \tag{A.4}$$

where $N = 2^{n_1}$. A simple product nn'/N appears in the exponent because $[q'_1 0.q_{n_1} + q'_2 0.q_{n_1-1}q_{n_1} \cdots + q'_{n_1-1} 0.q_2 \ldots q_{n_1} + q'_{n_1} 0.q_1 \ldots q_{n_1}] \to nn'/N$, which can be shown by noting that $e^{2\pi i 2^s} \equiv 1$ for all integers $s \geqslant 0$. Therefore, in the product $nn'/N = (q_1 2^{n_1-1} + q_2 2^{n_1-2} \ldots q_{n_1} 2^0) \times (q'_1 2^{n_1-1} + q'_2 2^{n_1-2} \ldots q'_{n_1} 2^0)$, we can drop all cross terms that yield a 2^s which suffices to prove the equivalence. Hence, one sees that the **QFT** is a unitary transformation from basis $|n\rangle$ to $|n'\rangle$ of the form $\langle n|\mathrm{QFT}|n'\rangle = \frac{1}{\sqrt{N}}(e^{2\pi i/N})^{nn'}$.

Appendix B. The MPI codes

The **bintodec**, **dectobin**, **OneOpA**, **TwoOpA**, **EulerPhi**, **splitn**, **ProjA**, **Randx**, **QFT**, **CF**, **SWAP**, **CPHASEK**, **HALL**, **HALL2**, **Entropy**, **EntropyP** codes are best understood by examination of the explicit directions within the code and also by the useage in the sample algorithms.

B.1. Sample algorithm codes

Teleportation and Superdense coding codes are also available. In this paper, we present the Grover and Shor cases. The Grover code is called Guniv.f90 and initiates the process by selecting a marked item that is to be searched for, with that item (labelled as IR) distributed to all the processors. There are $N_P = 2^p$ processors that are split into $N_G = 2^g$ groups (called multiverses) each multiverse then consists of $N_x = N_P/N_G = 2^{p-g}$ members, where both N_P and N_G are assumed to be powers of 2. Independent searches are carried out in each multiverse and at the end (this could be done at any preferred stage) the state-vector amplitudes for each group are used to form a group's density matrix. An overall ensemble average of all the group's density matrices are then computed and either the $2^{n_q} \times 2^{n_q}$ array is located on the master processor of the first group using subroutine **Entropy** or is distributed over a BLACS grid using subroutine **EntropyP**.

The first group (rankM=0) is free of noise, whereas all the other groups (rankM>0) are subject to various random disturbances with assigned probabilities. This is where particular noise models could be invoked by the user. This structure is also used for the Shor case.

In the Shor case (Suniv.f90, subshor.f90), there is an initial setup process to pick and test the number to be factored that is broadcast to all N_P processors, with again a split into N_G groups (multiverses) and separate searches done on the N_g members of each universe. Again group one is free of noise, whereas noise is introduced on all other groups, with a subsequent build up of the full density matrix using either the **ientropy=1** or **ientropy=2** options. Others cases and extensions all follow this same general pattern.

One can also examine the particular eigenvalues of the full density matrix, at selected stages, and also obtain fidelities and subtraces if desired.

References

[1] M.A. Nielsen, I.I. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000.
[2] J. Preskill's Caltech remarkable lectures are available at: http://www.theory.caltech.edu/people/preskill/ph229/.
[3] I. Kremsky, M.-H. Hsieh, T.A. Brun, Classical enhancement of quantum error-correcting codes, Phys. Rev. C 78 (2008) 012341, arXiv: 0802.2414.
[4] L.K. Grover, Phys. Rev. Lett. 79 (1997) 325–328.
[5] P.W. Shor, SIAM J. Comput. 26 (5) (1997) 1484.
[6] G. Patz, A Parallel Environment for simulating quantum computation, PhD thesis, MIT, 2003.
[7] K. Obenland, A. Despain, A Parallel Quantum Computer Simulator, quant-ph/9804039 (Presented at High Performance Computing 1998).
[8] J. Niwa, K. Matsumoto, H. Imai, General-purpose parallel simulator for quantum computing, Phys. Rev. A 66 (2002) 062317.
[9] K. De Raedta, K. Michielsenb, H. De Raedtb, B. Trieuc, G. Arnoldc, M. Richterc, Th. Lippertc, H. Watanabed, N. Ito, Massively parallel quantum computer simulator, Comput. Phys. Comm. 176 (2) (15 January 2007) 121–136.
[10] I. Glendinning, B. Ömer, Parallelization of the QC-Lib quantum computer simulator library, in: R. Wyrzykowski, et al. (Eds.), Parallel Processing and Applied Mathematics: Proceedings 5th International Conference, PPAM 2003 Czestochowa, Poland, in: Lecture Notes in Computer Science, vol. 3019, Springer, 2004, pp. 461–468. See also: Parallelization of the general single qubit gate and CNOT for the QC-lib quantum computer simulator library. Technical Report TR 2003-01, Institute for Software Science, University of Vienna, June 2003.
[11] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Comput. 22 (6) (1996) 789.
[12] For OpenMPI see: http://www.open-mpi.org/.
[13] C.H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, W.K. Wootters, Phys. Rev. Lett. 70 (1993) 1895–1899.
[14] C.H. Bennett, S.J. Wiesner, Phys. Rev. Lett. 69 (1992) 2881.
[15] B. Juliá-Díaz, J.M. Burdis, F. Tabakin, QDENSITY A Mathematica Quantum Computer simulation, Comput. Phys. Comm. 174 (2006) 914.
[16] Scalapack and Blacs packages, http://www.netlib.org/scalapack/scalapack_home.html.
[17] K. Kraus, States, Effects and Operations: Fundamental Notions of Quantum Theory, Springer-Verlag, 1983.
[18] G. Lindblad, Commun. Math. Phys. 4 (1976) 119.
[19] E. Gerjuoy, Am. J. Phys. 73 (6) (2005) 521.
[20] T.A. Brun, Decoherence and quantum trajectories, in: H.-T. Elze (Ed.), Decoherence and Entropy in Complex Systems, Springer, Berlin, 2004;
S.L. Adler, T.A. Brun, Generalized stochastic Schrödinger equations for state vector collapse, J. Phys. A 34 (2001) 4797–4809.
[21] Circuit graphs have been drawn using I. Chuang "qasm2circ", http://www.media.mit.edu/quanta/qasm2circ/, and S. Flammia and B. Eastin, Qcircuit, http://info.phys.unm.edu/Qcircuit/.
[22] Integrated Performance Monitoring, IPM, http://ipm-hpc.sourceforge.net/.