JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. xx. No xx (200x), pp. xx-xx

Parallel implementation of a quantum computing simulator

Marek Sawerwain, Jakub Pilecki
University of Zielona Góra
Institute of Control and Computation Engineering
Podgórna 50, 65-246 Zielona Góra
e-mail: M.Sawerwain@issi.uz.zgora.pl
e-mail: J.Pilecki@weit.uz.zgora.pl

Abstract. *In this paper the specialized software called quantum computation simulator is presented. Basic properties of quantum computation simulations are discussed. The algorithm of parallel implementation of a vector state transformation is presented also. Some examples of the presented software application are shown.*

1. Introduction

The Quantum Computation is a new trend in the computer science. It is hard to estimate the exact date of birth of this research area, but the paper [1] can be seen as a first example of work where the quantum computation model is discussed. Since then interest in the quantum computation began to grow rapidly. Notably in 1985 David Deutsch used a quantum algorithm working exponentially faster than any other known classical algorithm, to solve a problem of checking whether a given function is constant or balanced. This rapid development of the quantum computation theory exceeded technological capabilities of the modern science. So far experiments are restricted to a very small quantum register. Simulation of the quantum computation [5, 6, 7, 8] is therefore one of main aspects of the quantum physics. This paper presents a simulator of the quantum computation model (QCM). Our system is capable of simulation utilizing distinctly different approaches such as CHP model, computation model based on pure states and mixed states.

Content of presented paper is as follows: section 2 briefly discusses computational cost of the quantum computation model simulation. A lemma formalizing the Feynman's remark [1] is included. In section 3 main Quantum Computation Simulator (QCS) features are presented. The main problem of this paper is discussed in section 4, where a parallel algorithm for simulation of the quantum computation model with pure states is introduced. We present a short analysis of computational cost for some cases where our parallel algorithm achieved very good results (the time of computation is divided by the number of available nodes).

2. Simulations of the quantum computation model

In the paper [1] R.P.Feynman has argued that classical computers will never be able to perform simulations of full behaviour of a quantum system in a polynomial time. The original citation from Feynman's paper [1] is as follows:

Can physics be simulated by a universal computer? [...] the physical world is quantum mechanical, and therefore the proper problem is the simulation of quantum physics [...] the full description of quantum mechanics for a large system with R particles [...] has too many variables, it cannot be simulated with a normal computer with a number of elements proportional to R [... but it can be simulated with] quantum computer elements. [...] Can a quantum system be probabilistically simulated by a classical (probabilistic, I'd assume) universal computer? [...] If you take the computer to be the classical kind I've described so far [...] the answer is certainly, No!

Feynman's remark can be formalized by formulation of the following lemma:

Lemma 1: Let us assume that there exists an algebra, which includes an effective representation (with compression) of a quantum register with entanglement between n qudits. Symbol $\hat{\otimes}$ denotes special operators, which permits composition of several qudits and preserve the entanglement between them:

$$|\Psi\rangle = |\Psi_1\rangle \hat{\otimes} |\Psi_2\rangle \hat{\otimes} |\Psi_3\rangle \hat{\otimes} \ldots \hat{\otimes} |\Psi_n\rangle \qquad (1)$$

Nevertheless, the compressed vector still contains exponential amount of data describing the state of the quantum register. In the worst case, the process of measurement needs $O(n) = 2^n$ classical operations, if compression property of the vector state was lost.

Proof: The sketch of the proof is following: let T represent a classical Turing machine. On the tape the state vector containing d-level qudits (even with compression) is written. Its representation is contained in n fields of the tape. Without compression, the tape has length equal to d^n complex values. We have a register, which is not an equal superposition of states. The register can collapse to one of d^n base states. It is necessary to look through all states to find the maximum probability of amplitude what requires $O(n) = 2^n$ comparison operations.

Thus, quantum computation simulators are only widely available tool known today, suitable for quantum algorithms testing. Unfortunately, amount of classical computation required, restricts simulations to length of the register not exceeding 20-26 qubits on a typical PC hardware. Physical experiments are very expensive and difficult to perform, even in a typical university laboratory, therefore simulators of a quantum computation are very important, especially for educational and research purposes.

Another problem not discussed here is formulation of a theorem more general than lemma 1, where it can be proved, that simulation of the general quantum computation model has an exponential computational cost and the problem of simulation belongs to the exponentially-complete class. Considering the Feynman'a remark and lemma 1 the following hypothesis was formulated: the problem of simulation of QCM on a classical machine belongs to the exponential class of complexity:

$$EXP(n) = TIME(2^{n^k}) \tag{2}$$

On other hand taking into consideration a more general definition of the quantum information with qudits, the problem of simulation stills belongs the exponential class and not fall into to the elementary class of complexity:

$$TIME(2^{2^{\cdot^{\cdot^{\cdot^{n}}}}}) \tag{3}$$

3. The quantum computation simulator

The quantum computation simulator (QCS) [3, 4] is being implemented at the University of Zielona Góra since 2005 year. Several open source tools were used:

- Gnu Compiler Collection v4.x
- Python script language v2.4.x
- SWIG 1.3.29 wrapper generator
- LAPACK and BLAS linear algebra library
- MPI library for parallel implementation

The core library of QCS system is written in a pure ANSI C, programming language permitting creation of ports to many other software platforms. The main port of QCS is prepared in the Python language but port to Java exists also. For the MPI version, a simple language similar to the classical assembler qasm (quantum assembler) was prepared. This language makes writing scripts executing in a parallel environment much easier. Generally, QCS is working mainly in Windows and Linux operating systems in 32 and 64 bits environment. Figure 1 depicts a simple script for Python language producing one of Bell states.

```
import qcs
def makePsiPlus(r):
   r.SetKet("0000")
   r.HadN(0)
   r.CNot(0,1)
```

```
r=qcs.QubitReg(4)
r.Reset()
makePsiPlus(r)
r.Pr()
del r
```

Fig. 1. Elementary script in the Python language which produce one of the maximally entangled states (aka Bell states)

Implemented features of QCS system permits to approach a simulation of a quantum system in a variety of ways:

•PQC -- pseudo quantum circuits which consist of two gates only: not and cnot gates,

•CHP -- quantum circuits composed with following gates: cnot, hadamard, phase change and single qubit measurement,

•QCM -- standard quantum circuit model,

•density matrices and several standard matrices tools like fidelity, eigenvector and eigenvalue functions and Schmidt decomposition.

We plan to improve the future version of our simulator with a one-way quantum computation model based on teleportation and measurement operations. Another interesting feature could be an addition of some symbolic tools permitting for the simulation of QCM in a symbolic way, similar to the software for Mathematica [7, 8] but without usage of the specialised software.

4. The parallel implementation of QCS system

The basic engine designed for the one-processor architecture is presented in [3]. The designed parallel engine is effective, but not in sense of Nick class. In paper [2] authors presents another algorithm for a massive parallel simulations of a pure state quantum computational model. The approach presented in [4] however, sets some important limits to the gate applying process. The number of control qubits c and target qubits t must preserve the relation: $c < t < n$ where n is total number qubits in the register. The parallel version of QCS has no such limits. The QCS allow for any order of steering and target lines.

For example the following table presents the result for a small script executed on a one-processor machine:

register size	10	20	21	22	23	24	25
time in sec.	0.0s	1.5s	3.1s	6.5s	13.5s	28.2s	58.7s

Obtained results confirm the observation that addition of one new qubit to the quantum register doubles the time of a script execution. Comparison of our software with other quantum simulators like QCL [5] has revealed our algorithm to be more efficient. In the case of 23-qbit quantum register, QCS used only 70MB memory while QCL as much as 512MB. The simulation of two Hadamard gate application used to create a superposition of the whole register took about 18secs with QCS and about 100sec with the software developed by Bernard Omer.

Algorithm of the vector state transformation for one qubit gate has a very simple description depicted in Figure 2. Two typical "for" loops and some invariants introduced in Proposition 2 (presented later) are used. These invariants are used to find an index of elements, which must be changed by the values taken from the gate matrix u. The computational cost of execution of code from Figure 2 depends on size of the quantum register and is given by the following relation:

$$T(n) = \sum_{ip=0}^{2^{n-1}} \sum_{i=0}^{2^{n-t}} A = (1 + 2^{n-1} + 2^{n-t} + 2^{2n-t-1})A = O(2^n), \text{ and } A = (t_c + t_{fnc}) \qquad (4)$$

The symbol A denotes the computational requirements of the instruction in the second loop inner code, t_c represents the cost of executing of trivial operations implementing *r1* and *r2* variables, and t_{fnc} is the execution cost of *oper_on_rows* function.

```
m = pow( 2, n - 1 ); step = pow( 2, n - t ) - 1;
p = pow( 2, t - 1 ); vstep = pow( 2, n ) / p;
irow = 0;

for( ip = 0 ; ip < p ; ip++ ) {
    for( i = 0 ; i < step + 1 ; i++ ) {
        r1=irow+i;
        r2=irow+i+step+ 1;
        oper_on_rows(q_reg, r1, r2, u);
    }
    irow = irow + vstep;
}
```

Fig. 2. Pseudo-code describes an algorithm of the vector transformation for pure states, where the matrix operation is denoted by small letter *u*

Figure 3 depicts the basic parallel model of our state transformation algorithm. The register is divided into several equal parts. The total amount of nodes denoted by *n* means, that there exists the one master node managing *n-1* slave nodes. For the best results the number of nodes is given by the following relation:

$$N_{DP} = \lceil \log_M 2^L \rceil \qquad (5)$$

where L denotes the number of qubits in the quantum register and M is the number of qubits processed within one slave node. Unfortunately, the total parallel time is exponential. It is given by the following proposition:

Proposition 1

The computational cost of simulation of QCM model on a parallel machine is given by:

$$T_{DP}(L) = \left\lceil \frac{2^L}{N_{DP}} \right\rceil \tag{6}$$

Proof

Direct consequence of tensor product of the vector state, vector representing the quantum register. This means that algorithm is not optimal for parallel computations and do not belong to the Nick parallel complexity class.

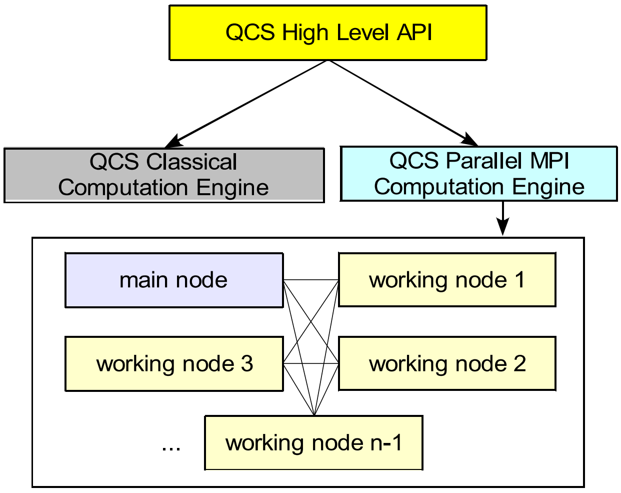

Fig. 3. Standard parallel architecture in QCS software

4.1 Parallel algorithm on the tensor structure of quantum register

The procedure applying gates to the quantum register covers some important information useful for developed of the more effective vector state transformation algorithm. Of course, as section 2 has shown, probability of obtaining an effective algorithm of QCM simulation is rather small. Still, presented algorithm achieved very good results, especially compared with other [5, 6] QCM simulators.

It is known that for one qubit gate, the operational matrix U that transforms the vector state is described by a tensor of unitary matrices. Let u represent some unitary matrix for one qubit gate (the matrix size is 2×2). Then, the operation of applying of one qubit operation to the second qubit of the three-qubit register is denoted as

$$U = I \otimes u \otimes I \tag{7}$$

The symbol I denotes the identity matrix and U represents the large matrix transforming the quantum register. Detailed analysis of the state vector tensor product makes possible finding vector invariant values. These values permit for the more effective process of applying of a matrix to the state vector. The main advantage is omitting creation of the large *U* matrix. Only a small *u* matrix and number of qubit are required. For example, the three-qubit tensor product is calculated in the following way:

$$|q_0q_1q_2\rangle = \begin{bmatrix}\alpha_0\\ \beta_0\end{bmatrix} \otimes \begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix} \otimes \begin{bmatrix}\alpha_2\\ \beta_2\end{bmatrix} = \begin{bmatrix}\alpha_0\begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix}\\ \beta_0\begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix}\end{bmatrix} \otimes \begin{bmatrix}\alpha_2\\ \beta_2\end{bmatrix} =$$

$$= \begin{bmatrix}\alpha_0\alpha_1\\ \alpha_0\beta_1\\ \beta_0\alpha_1\\ \beta_0\beta_1\end{bmatrix} \otimes \begin{bmatrix}\alpha_2\\ \beta_2\end{bmatrix} = \begin{bmatrix}\alpha_0\alpha_1\alpha_2\\ \alpha_0\alpha_1\beta_2\\ \alpha_0\beta_1\alpha_2\\ \alpha_0\beta_1\beta_2\\ \beta_0\alpha_1\alpha_2\\ \beta_0\alpha_1\beta_2\\ \beta_0\beta_1\alpha_2\\ \beta_0\beta_1\beta_2\end{bmatrix} \tag{8}$$

The range of following qubits can be observed. In the final state, amplitudes of all qubits always appear four times, but with a different step depending on the position of the influenced qubit. For the operation matrix denoted as $U = I \otimes u \otimes I$, we have the following matrix:

$$\begin{pmatrix}\alpha & . & \beta & . & . & . & . & .\\ . & \alpha & . & \beta & . & . & . & .\\ \gamma & . & \delta & . & . & . & . & .\\ . & \gamma & . & \delta & . & . & . & .\\ . & . & . & . & \alpha & . & \beta & .\\ . & . & . & . & . & \alpha & . & \beta\\ . & . & . & . & \gamma & . & \delta & .\\ . & . & . & . & . & \gamma & . & \delta\end{pmatrix} \tag{9}$$

Similarly to the tensor of vectors, some invariant values can be found. For example, upper elements of *u* matrix are separated by one field. Generally, we can define several important invariants:

Proposition 2 Let *n* denote the size of the quantum register and *t* be a position of the target qubit. Then,

- number of "small matrices" is given by $m = 2^{n-1}$
- step between element of small matrices $step = 2^{n-t} - 1$
- block counts $p = 2^t - 1$
- distance in the vector state $vstep = 2^n / p$

Proof

These equations are based on the tensor product of qubits describing the quantum register. Correctness of equations can be proved by induction.

These invariants permit for an easier addressing process in the space of U matrix and the state vector. Then, creation of the full U matrix is not necessary. Two "smart" for-like loops (similar to the code form Figure 2) can search through the whole register and change its state without need for an additional copy of the vector's state.

Controlled gates can be treated in a similar way. In such case though, the situation is much better, because controlled gates are generating only small changes in the vector's state. An amount of small gates is given by the equation:

$$m = 2^{n-q}, \tag{10}$$

where q denotes the number of controlled qubits.

The observation of the tensor product of the vector state and synthesis of the operation matrix gives us the answer for whether the parallel processing will be more effective than computation on a one processor architecture. In many cases the total parallel computation time will be approaching the ideal case given by equation:

$$T_{DP}(L) = \left\lceil \frac{2^L}{N_{DP}} \right\rceil \tag{11}$$

Even though, the total amount of operational steps in nodes is still exponential.

4.2 Examples of gate applying

Observation of the way U operation given by the tensor product: $U = I \otimes u \otimes I$ influences the vector state in equation (12), can answer the question whether computation on the state vector can be divided into two nodes and executed independently, without swapping any additional information.

$$\begin{pmatrix} \alpha & \cdot & \beta & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \alpha & \cdot & \beta & \cdot & \cdot & \cdot & \cdot \\ \gamma & \cdot & \delta & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \gamma & \cdot & \delta & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \alpha & \cdot & \beta & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \alpha & \cdot & \beta \\ \cdot & \cdot & \cdot & \cdot & \gamma & \cdot & \delta & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \gamma & \cdot & \delta \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \end{pmatrix} = \begin{pmatrix} \alpha\alpha_0 + \beta\alpha_2 \\ \alpha\alpha_1 + \beta\alpha_3 \\ \gamma\alpha_0 + \delta\alpha_2 \\ \gamma\alpha_1 + \delta\alpha_3 \\ \alpha\alpha_4 + \beta\alpha_6 \\ \alpha\alpha_5 + \beta\alpha_7 \\ \gamma\alpha_4 + \delta\alpha_6 \\ \gamma\alpha_5 + \delta\alpha_7 \end{pmatrix} \tag{12}$$

This situation results in the smallest computational cost equal to the cost described by equation (6). Unfortunately, there exist many cases where additional information must be swapped. For example the operation $U = u \otimes I \otimes I$:

$$\begin{pmatrix} \alpha & \cdot & \cdot & \cdot & \beta & \cdot & \cdot & \cdot \\ \cdot & \alpha & \cdot & \cdot & \cdot & \beta & \cdot & \cdot \\ \cdot & \cdot & \alpha & \cdot & \cdot & \cdot & \beta & \cdot \\ \cdot & \cdot & \cdot & \alpha & \cdot & \cdot & \cdot & \beta \\ \gamma & \cdot & \cdot & \cdot & \delta & \cdot & \cdot & \cdot \\ \cdot & \gamma & \cdot & \cdot & \cdot & \delta & \cdot & \cdot \\ \cdot & \cdot & \gamma & \cdot & \cdot & \cdot & \delta & \cdot \\ \cdot & \cdot & \cdot & \gamma & \cdot & \cdot & \cdot & \delta \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \end{pmatrix} = \begin{pmatrix} \alpha\alpha_0 + \beta\alpha_4 \\ \alpha\alpha_1 + \beta\alpha_5 \\ \alpha\alpha_0 + \beta\alpha_6 \\ \alpha\alpha_1 + \beta\alpha_7 \\ \gamma\alpha_0 + \delta\alpha_4 \\ \gamma\alpha_1 + \delta\alpha_5 \\ \gamma\alpha_2 + \delta\alpha_6 \\ \gamma\alpha_3 + \delta\alpha_7 \end{pmatrix} \tag{13}$$

The information concerning amplitudes from the first and second node must be swapped, otherwise the simulation of U operation cannot be calculated properly.

Much better results were obtained for multiqubit gates. The CNot gate has shown, that there exists gates that do not change the state of a whole register but only it's selected parts. For example, the typical application of CNot gate where qubit zero is the control qubit and the second qubit represent the target qubit, can be encoded in the following way:

$$\begin{pmatrix} 1 & . & . & . & . & . & . & . \\ . & 1 & . & . & . & . & . & . \\ . & . & 1 & . & . & . & . & . \\ . & . & . & 1 & . & . & . & . \\ . & . & . & . & \alpha & . & \beta & . \\ . & . & . & . & . & \alpha & . & \beta \\ . & . & . & . & \gamma & . & \delta & . \\ . & . & . & . & . & \gamma & . & \delta \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha\alpha_4 + \beta\alpha_6 \\ \alpha\alpha_5 + \beta\alpha_7 \\ \gamma\alpha_4 + \delta\alpha_6 \\ \gamma\alpha_5 + \delta\alpha_7 \end{pmatrix} \quad (14)$$

Amplitudes in the first node (first four amplitudes) are not changed while amplitudes in the second part of the register are modified. If we divide the register into two equal parts, the sending of additional information between them is not necessary. As such applying CNot gate can be, in some cases, calculated independently at nodes, the ideal time complexity is given by

$$T(L) = \frac{2^{L-q_c}}{N} \quad (15)$$

The symbol q_c denotes the number of control lines and the big letter "L" denotes total size of the quantum register.

4.3 Efficient tests of QCS system

The results of simulation of IQFT are presented in Table 1. In this test the four-processor system based on an Intel Itanium IA-64 processor architecture was used (with a total available memory amount equal to 4GB RAM). The obtained results for the of one qubit and two qubit gates execution confirms the obvious expectation, that addition of new nodes lowers an amount of time required to perform a computation. On other hand results confirms also the fact of the good implementation of parallel algorithms of vector processing.

Table 1. Timing of gates in IQFT simulations for IA-64 Itanium 2 system

	1-proc	3-proc	5-proc
1q gate	0.3s	0.2s	0.1s
2q gate	0.7s	1.5s/0.4s	1.4s/0.4s

Results obtained for two qubit gates are worse due to the transfer of additional data between nodes. It's necessary to mention, that the cross-nodes information transfer travels through the shared memory subsystem.

The more realistic results were obtained for a cluster of nine PC computers (1GB Ram, Pentium 4 2.4 Ghz) connected by 100Mbits Ethernet network. Those results are obviously worse. The communication took more time, as did the process of preparation of data for the cross-node transfer.

Table 2. Results for three, five and nine nodes in cluster build from the typical PC hardware, 1GB Ram and Intel Pentium 4 Processor 2.4Ghz

	3	5	9
25q a	1m 17s	44 s	24s
25q b	1m 14s	37 s	20s
26q	--	--	43s
27q	--	--	1m 15s
28q	--	--	2m 29s

The nine nodes cluster offers enough operational memory to make calculation with the quantum register containing 28 qubits. The test “25q a” requires swapping of additional information between nodes, therefore it generated worse results. The test “25q b” and many others don’t swap any amount of data. Every node process data independently.

5. Conclusions

Presented article describes an implementation of a parallel algorithm of the vector state transformation working for pure states. It must be observed, that obtained algorithm is not effective in the sense of the Nick class. Still, the presented software QCS system is a tool more effective in a real simulation of the quantum computation model than other comparable simulators like [5]. It is especially true in regard of the scripts execution time.

The obtained results have shown additional ways of obtaining better performance of a parallel algorithm implementation. For example, communication can be improved by usage of special compression techniques. Applying the compression results in another interesting feature: the transfer between nodes can be optimised by compression of fragments of the vector state. Even a simplest compression can be useful. Example of that is the run-length encoding kind of compression.

In a future we intend to implement the teleportation-measuring computation model. Nowadays there exist no systems able to perform simulations of this part of the quantum computation model.

References

[1] Feynman R.P., *Simulating physics with computers*, International Journal of Theoretical Physics 21:6/7. pp. 467-488 (1982).

[2] de Raedt K., Michielsen K., de Raedt H., Trieu B., Arnold G., Richter M., Lippert Th., Watanabe H., Ito N., *Massive Parallel Quantum Computer Simulator*, submitted to Computer Physics Communications.

[3] Sawerwain M., *Quantum Computing Simulator (in polish)*, CMS '05 : V konferencja. Kraków, Polska, 2005-Kraków: Oprogramowanie Naukowo-Techniczne, 2005 - Vol. 2: Regular sessions, pp. 185--190.

[4] Sawerwain M., *Parallel implementation of quantum computing Simulator (in polish)* : teoria, projekty, wdrożenia, aplikacje : XIV konferencja. Łódź, Polska, 2006 .- Łódź : "Piątek Trzynastego Wydaw.", 2006, pp. 241--244.

[5] Omer B., *Structured Quantum Programming*, Ph.D. Thesis, TU Vienna, Vienna, 2003.

[6] Gawron P., Miszczak J.A., *Numerical simulations of mixed states quantum computation*, www.arxiv.org/quant-ph/0406211.

[7] Touchette H., Dumai P., *QuCalc - The quantum computation package for Mathematica*, http://crypto.cs.mcgill.ca/QuCalc/, 2000.

[8] Juliá-Díaz B., Burdis J.M., and Tabakin F. *QDENSITY - A Mathematica Quantum Computer simulation,* Computer Physics Communications. Vol. 174, Issue 11 , 1 June 2006, pp. 914—934.