John von Neumann Institute for Computing



High Performance Simulation of Ideal Quantum Computers

Guido Arnold, Marcus Richter, and Thomas Lippert

published in

NIC Symposium 2006 ,
G. Münster, D. Wolf, M. Kremer (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **32**, ISBN 3-00-017351-X, pp. 349-356, 2006.

© 2006 by John von Neumann Institute for Computing
Permission to make digital or hard copies of portions of this work for personal or classroom use is granted provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise requires prior specific permission by the publisher mentioned above.

http://www.fz-juelich.de/nic-series/volume32

High Performance Simulation of Ideal Quantum Computers

Guido Arnold[1], **Marcus Richter**[1], **and Thomas Lippert**[1,2]

[1] Central Institute for Applied Mathematics, John von Neumann Institute for Computing
Research Centre Jülich, 52425 Jülich, Germany
E-mail: {g.arnold, m.richter, th.lippert}@fz-juelich.de

[2] Department C, Bergische Universität Wuppertal, 52425 Wuppertal, Germany

1 Relevance of Quantum Computing

Quantum processing of information has become a rapidly evolving field of research in physics, mathematics, computer science, and engineering[1] and has led to substantial progress in quantum computation, quantum communication and control of quantum systems. Quantum computers have become of great interest primarily due to their potential of solving certain computationally hard problems such as factoring integers[2] and searching databases faster than a conventional computer[3]. Candidate technologies for realizing quantum computers include trapped ions, atoms in QED cavities, Josephson junctions, nuclear or electronic spins, quantum dots, and molecular magnets. Grover's quantum search[3] and Shor's quantum prime factorization algorithm[2] have been successfully implemented on systems of up to 7 qubits using liquid NMR techniques[4], experimentally demonstrating the viability of the concept of quantum computation. Recently the first quantum byte has been realized using linear ion traps[5].

In spite of this impressive development, a demonstration that quantum computation can solve a non-trivial problem is still lacking. To be of practical use, quantum computers will need error correction, which requires at least several tens of qubits and the ability to perform hundreds of gate operations. This imposes a number of strict requirements[6], and narrows down the list of candidate physical systems. Simulating numbers of qubits in this range is important to numerically test the scalability of error correction codes and fault tolerant quantum computing schemes and their robustness to errors typically encountered in realistic quantum computer architectures.

2 The Need for Simulation

A physically realizable quantum computer is a complex many-body quantum system. In order to exercise control over many qubits and to suppress the rate at which errors are introduced during a quantum computation, it is in principle necessary to understand the full time evolution of the whole quantum system. Sources of errors are the loss of coherence (decoherence) due to unwanted interaction with the environment[7] and systematic errors due to imperfections of the operational pulse sequences.

In *first principle* simulations the time dependent behavior can be derived from the Hamiltonian of the physical model chosen to describe a specific hardware realization. Pulses are modeled as time-dependent external fields acting on the relevant degrees of

freedom. The coupling of the environment is taken into account by including interactions with other degrees of freedom, also represented by pseudo-spins.

This kind of simulations is needed to analyze decoherence resulting from interactions with the environment. Depending on the assumptions that where made in deriving the microscopic Hamiltonian and/or the manner in which the effect of the coupling to the environment is taken into account, the calculation of the real-time quantum dynamics of the quantum computer readily requires the simulation of systems of many (20-40) qubits over extended periods of time. To perform such very demanding computations, highly optimized simulation code that runs on different high-end computer systems has to be developed.

3 Simulation of Ideal Quantum Computers

In a first step towards realistic quantum computer simulations we implement so called *ideal simulations*, where each gate is modeled by a quantum operation that acts instantaneously on the internal state of the quantum computer, neglecting both implementation imperfections and interactions with the environment. The drawback is that the state of the quantum computer is known only after the application of each gate, but this is sufficient for most algorithmic purposes.

In contrast to a classical bit the state of an elementary storage unit of a quantum computer, the quantum bit or qubit, is described by a two-dimensional vector of Euclidean length one. Denoting two orthogonal basis vectors of the two-dimensional vector space by $|0\rangle$ and $|1\rangle$, the state $|\psi\rangle$ of a **single qubit** can be written as a linear superposition of the basis states $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle_1 = a_0|0\rangle + a_1|1\rangle \; = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}, \tag{1}$$

where a_0 and a_1 are complex numbers such that $|a_0|^2+|a_1|^2 = 1$. Useful computations require more than one qubit. The state of a quantum computer with N qubits can be represented in the 2^N- dimensional Hilbert space as

$$\begin{aligned} |\psi\rangle_N &= a_{0\ldots00}|0\ldots00\rangle + a_{0\ldots01}|0\ldots01\rangle + \ldots + a_{1\ldots10}|1\ldots10\rangle + a_{1\ldots11}|1\ldots11\rangle, \\ &= a_0|0\rangle + a_1|1\rangle + \ldots + a_{2^N-1}|2^N-1\rangle \\ &= (a_0, a_1, \ \ldots \ , a_{2^N-1})^T . \end{aligned} \tag{2}$$

According to the rules of quantum mechanics any evolution in time means changing the system state unitarily. Each operation on a quantum computer can be described by a 2^Nx 2^N dimensional unitary transformation $U = e^{-iHt}$ acting on the state vector $|\psi'\rangle = U|\psi\rangle$, with the hermitian matrix H being the Hamiltonian of the quantum computer model. In this paper we will not describe any details of quantum computer hardware modeled by appropriate Hamiltonians. It is sufficient to know that an ideal quantum computer can be modeled by simple spin models such as the Ising model associating the two single-spin states *up*$= |\uparrow\rangle$ and *down*$=|\downarrow\rangle$ with the single-qubit basis states $|0\rangle$ and $|1\rangle$[8].

As the unitary transformation U may change all amplitudes simultaneously, a quantum computer is a massively parallel machine. In order to simulate an arbitrary unitary operation on a conventional computer the resulting matrix-vector multiplication requires in the worst case $\mathcal{O}(2^{2N})$ complex valued arithmetic operations.

As in the case of programming a conventional computer, it is extremely difficult to write down explicitly that single one-step operation that transforms the input state into a desired output state. Usually a quantum algorithm consists of a sequence of many elementary gates. These elementary gates are represented by very sparse unitary matrices. The resultant matrix-vector multiplication can be implemented very efficiently and requires typically $\mathcal{O}(2^N)$ arithmetic operations per elementary gate. A small set of elementary **one-qubit gates** (such as the Hadamard gate and the Phase shift gate) and a nontrivial **two-qubit gate** (such as the controlled NOT gate) are sufficient (but not unique) to construct a *universal* quantum computer[9]. In the framework of ideal quantum operations any one-(two-) qubit operation can be decomposed into a sequence of 2x2 (4x4) matrix operations each acting on an orthogonal subspace of the 2^N dimensional Hilbert space.

In the following we describe an efficient parallel simulation of ideal quantum computers on a high-end computer system that allows simulating up to 37 qubits requiring 3 TB of memory and a considerable compute power.

4 Quantum Operations

We will discuss in detail the implementation of a typical one-qubit operation, the Hadamard gate. This gate is often used to prepare the state of uniform superposition. The Hadamard operation on a single-qubit state is defined by

$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Let us consider a quantum computer consisting of three qubits and its state vector $|\psi\rangle = (a_{000}, a_{001}, a_{010}, a_{011}, a_{100}, a_{101}, a_{110}, a_{111})^T$. Instead of computing the 8x8 matrix appropriate to a Hadamard operation H_q acting on qubit q we can compute $H_q|\psi\rangle$ as given by the scheme in Fig. 1.

From this simple example we can learn some characteristics of any one-qubit operation on a N-qubit quantum computer influencing qubit $q = 0, \ldots, N-1$ by acting on the 2^N-dimensional state vector $|\psi\rangle$:

i) H_q can be decomposed into 2^{N-1} applications of H involving a pair of state vector components (k, l) with relative stride $|l - k| = 2^q$ each.

ii) the (2x2) matrices H operate on orthogonal subspaces of the 2^N- dim Hilbert space. Hence they commute and computations can be done in parallel.

From i) we can derive that with the exception of H_0 all Hadamard gates operate purely on even *or* odd state vector elements. This claim also holds for any other quantum operation that does not involve qubit 0. Thus we will split the state vector $|\psi\rangle$ given by Eq.(2)

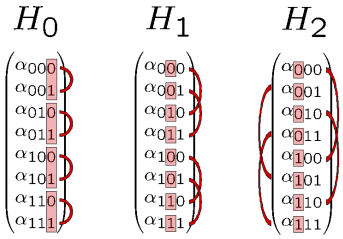


Figure 1. Decomposing a Hadamard transformation acting on qubit q on a three qubit computer $H_q|\psi\rangle$ into four parallel applications of the single-qubit Hadamard gate H. H_0 for example splits into $\binom{a_{ij0}}{a_{ij1}} \mapsto H\binom{a_{ij0}}{a_{ij1}}$ with $i, j \in \{0, 1\}$.

into an even part $|\psi_e\rangle$ and its odd counterpart $|\psi_o\rangle$. For $0 \leq k \leq 2^{N-1} - 1$ we define:

$$|\psi_e(k)\rangle = |\psi(2k)\rangle \quad \text{and} \quad |\psi_o(k)\rangle = |\psi(2k+1)\rangle. \tag{3}$$

This state vector splitting saves half the effort to determine all pairs of indices (k, l) involved in the corresponding one-qubit operation. For H_q with $q > 0$ consecutive pairs (k, l) are mapped to identical pairs $(k', l')_e = (k', l')_o$ with stride $|l' - k'| = 2^{q-1}$. Only H_0 shows an even and odd mixing but trivial pattern that is implemented differently.

5 Parallelization and Computational Resources

More important than gaining half of the integer arithmetics, needed for state vector referencing, the splitting decreases communication overhead in the parallelized simulation code due to sending and receiving *non-stridden* sections of the state vectors $|\psi_e\rangle$ and $|\psi_o\rangle$. This leads to a gain of up to 30% of the wall clock time for one-qubit operations (depending on the system size N and the qubit number q the gate operates on).

The problem of simulating quantum computers is clearly memory bounded. Due to the exponentially increasing amount of memory needed we developed and implemented a large scale simulation on the Juelich SMP supercomputer IBM p690 providing enough memory to handle a 37 qubit system. Simple storage of the state vector in case of a 37 qubit system requires a memory of 2 TB. An efficient implementation of quantum operations on that state vector even requires 3 TB of memory.

The Juelich IBM p690 is a *cluster of 32 compute nodes each containing 32 Power4+ processors* (64bit) and 112 GB memory per node leading to 3.5 TB overall memory available to user access. A quantum computer with up to $N = 32$ qubits reserving at max. $2^{36} = 64$ GB memory to store the complex valued state vector in double precision can be simulated on one node using 32 processors. In the following table we describe the typical simulation requirements depending on the system size N. The last row indicates the overall memory requirements to efficiently simulate quantum operations.

#qubits N	32	33	34	35	36	*37*
#procs	32	64	128	256	512	*1024*
#nodes	1	2	4	8	16	*32*
memory (state vector)	64 GB	128 GB	256 GB	512 GB	1 TB	*2 TB*
memory (operation)	96 GB	192 GB	384 GB	768 GB	1.5 TB	*3 TB*

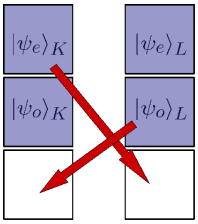


Figure 2. Communication pattern for a one-qubit operation on qubit $q > N - p$. The N qubit state vector is partitioned into 2^p tasks. The computational effort is equally distributed to 2^{p-1} disjoint pairs of tasks (K, L). Task K (L) operates on all odd (even) state vector elements $|\psi_o\rangle_{K+L}$ ($|\psi_e\rangle_{K+L}$).

Partitioning the 2^N- dimensional state vector into $P = 2^p$ *tasks* allows to store the state vector of an N-qubit quantum computer on 2^{N-32} compute nodes equivalent to 2^{N-27} processors with the obvious limitation

$$N - 32 \leq p \leq N - 27. \tag{4}$$

MPI-based exchange of the local $|\psi_e\rangle$ and $|\psi_o\rangle$ allows computation of one-qubit operations on "nonlocal" qubits $q \geq N - p$. In that case the relevant state vector components (k, l), the single-qubit gate operates on, are separated as wide as (or wider than) the number of states per task: $|l-k| = 2^q \geq 2^{N-p}$. As shown in Fig.2 task K (containing all components k) sends its local $|\psi_e\rangle_K$ to task L and receives the local part $|\psi_o\rangle_L$ from task L.

After task K has computed locally the operation $H|\psi_o\rangle_{K+L}$ on all $(k, l)_o$ and task L has computed the even part $H|\psi_e\rangle_{K+L}$ on all $(k, l)_e$ respectively, both send back their results: K sends $H|\psi_o\rangle_L$ to L and receives $H|\psi_e\rangle_K$ from L. After that K contains the updated vectors $|\psi'_{e/o}\rangle_K = H|\psi_{e/o}\rangle_K$. L respectively stores $|\psi'_{e/o}\rangle_L = H|\psi_{e/o}\rangle_L$ in place. Thus the operation requires an intermediate buffer of 2^{N-p-1} elements (half the size of the local state vector). Remember that any one-qubit operation on qubit 0 is local.

- Setting p to the maximum given by Eq.(4) *(finest graining)* means distributing the state vector on all processors of the nodes involved. This is equivalent to a pure MPI parallelization ansatz. For data exchange within a node the MPI-library is mapped to fast shared memory access.

- Choosing the minimal number of tasks given by Eq.(4) *(coarsest graining)* leads to one task per node which means that 32 processors are available to that task in parallel. To do this the core routine (a double loop) is done in parallel by $T = 2^t = 32$ OpenMP threads using shared memory access to the whole node memory of 64 GB reserved for the "local" state vector.

- Any other choice of $p + t = N - 27$ with $t \geq 0$ leads to an *a priori* reasonable *hybrid* parallelization strategy in the sense that all processors of the involved nodes are used for computation.

Our detailed investigation on systems of sizes $N = 32, 33, 34, 35, 36$ shows that different OpenMP parallelization strategies using more than 4 threads per MPI-task fail to reach the efficiency of the pure MPI-parallelization. Since we cannot provide a large enough additional buffer, task K for example is forced to operate "in place" on the state vector parts

$H|\psi_o\rangle_{K+L}$ having write access to a *global=shared* vector. In that case synchronization of different OpenMP threads becomes necessary (within a task) and slows down computation.

Finest graining in the pure MPI-ansatz benefits from simple coding of one qubit operations on nonlocal qubits $q \geq N - p$ with maximal p according to Eq.(4). Since the stride $|l - k|$ is as large as (or larger than) the size of the local state vector stored by each task these gates operate *consecutively on all* components. The compiler can build two streams prefetching the entire local state vector.

Since the memory access dominates the time needed to perform a quantum operation, it is crucial a) to minimize consecutive access to widely separated memory entries and b) to use a simple access pattern allowing for efficient (compiler driven) prefetching. We investigate different ways to code the core routine, that determines the state vector components k, l and performs the computation on local qubits $q < N - p$.

version 1

```
do i=imin,imax,i1
   do k=i,i+i1n-1
      l=k+i1n
      ...
   enddo k
enddo i
```

version 2

```
do i=imin,imax,2
     i2=iand(i,i1n)
     k=i-i2+i2/i1n
     l=k+i1n
     ...
enddo i
```

Recoding of the core routine shifting from version 1 (nested loop) to version 2 (single loop) makes OpenMP loop parallelization simpler but hampers streaming because of additional "jumps" in the sequence of index k. To comprehend this we respectively present a part of a typical sequence of consecutive pairs of state vector elements (k, l) to be read from memory.

version 1	k	**0**	1	2	3	4	5	6	7	**16**	17	18	19	20	21	22	23	**32**
	l	**8**	9	10	11	12	13	14	15	**24**	25	26	27	28	29	30	31	**40**
version 2	k	**0**	2	4	6	**1**	3	5	7	**16**	18	20	22	**17**	19	21	23	**32**
	l	**8**	10	12	14	**9**	11	13	15	**24**	26	28	30	**25**	27	29	31	**40**

To halve the number of co-resident streams to be prefetched from memory we additionally split the computation of $|\psi_e\rangle$ and $|\psi_o\rangle$ into two sequential loops of version 1. This speeds up computation considerably. We measure that our fastest nested OpenMP parallelization of version 1 is about 25% faster than the dense coded version 2. This gain applies to the usage of 1,2,4 and 8 OpenMP threads.

Analyzing the time needed to compute H_q depending on the qubit q the operation acts on (see Fig.3), we can identify three regions according to different speeds of memory access. In case of $T = 1$ we obtain fast computation for $q < N - p = 27$, because all state vector components involved are located within processor memory. Higher timings for $N - p \leq q < N - p + 5 = 32$ arise from intra-node communication. The communication between processors is mapped to shared memory access, which is slower than access to the memory associated to a single processor, but faster than MPI based inter-node communication for $q \geq 32$. Timings for parallelizations using 16 (32) threads per MPI process are not given in Fig.3, since computation gets slower by a factor of about 3 (6) compared to pure MPI. This is due to the machine architecture: each node is built up by 4 multichip modules each containing 8 processors. The memory access within a multichip module is

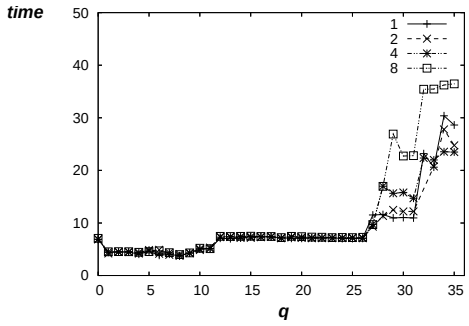


Figure 3. Timings for a Hadamard operation on qubit q at system size N=36 depending on the number of threads $T = 1, 2, 4, 8$ per MPI-task using $T * 2^p = 512$ processors.

faster than getting data from memory associated to another module.

Keeping the local size of the state vector fixed at 2 GB per processor we compare the time needed to compute Hadamard operations on all qubits for different choices for the number of tasks and threads. Fig.4 shows the average timings $t_{av}(N)$ for a complete Hadamard transformation $N^{-1}\prod_{q=0}^{N-1} H_q$ on different system sizes N depending on the number of MPI tasks $P = 2^p$ and different numbers of OpenMP threads $T = 2^t$ respecting $t + p = N - 27$. Multiple measurements indicate a statistical timing error of max 5%. On this error level we identify the usage of one or two OpenMP-threads per MPI-task as optimal. Using 4 threads gives a slightly worse timing.

Taking the best average timing results normalized to $\min(t_{av}(32))$ from the l.h.s of Fig.4 for each system size N we observe the weak (=local size fixed) scaling behavior plotted on the r.h.s of Fig.4. Compared to the optimal (weak) scaling of a constant $\min(t_{av}(N))/\min(t_{av}(32)) = 1$ we still have an efficiency of 70% simulating a 37 qubit-system. The efficiency loss at larger systems is due to the linearly increasing fraction of operations on nonlocal qubits $q \geq 32$ using internode MPI-communication.

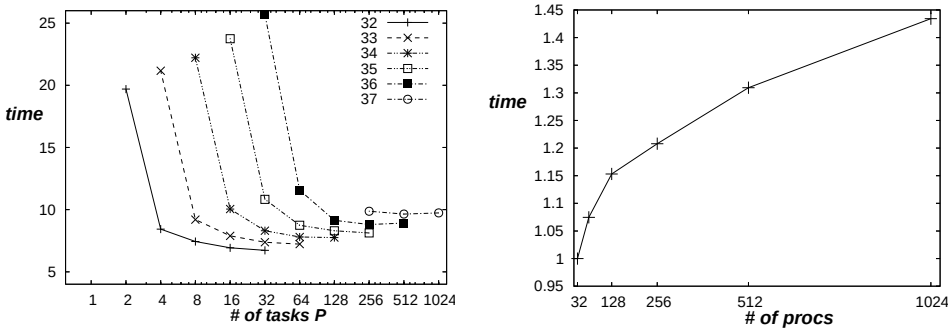


Figure 4. Left: average timings $t_{av}(N)$ for a Hadamard operation on different system sizes N depending on the number of MPI tasks $P = 2^p$ using $T = 2^t$ OpenMP threads with $t + p = N - 27$. Right: scaling of the minimal average timings for the system sizes $N = 32, 33, 34, 35, 36, 37$ using 32,64,128,256,512,1024 processors respectively.

A universal quantum computer also needs two-qubit operations such as the CNOT gate to incorporate qubit interaction. We illustrate the action of the $CNOT_{CT}$ gate on a two qubit state that flips the target qubit T if the control qubit C is set to $|1\rangle$

$$CNOT_{10} \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \end{pmatrix} \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix}. \tag{5}$$

Even though the CNOT-gate depending additionally on a control qubit its enfolded compute pattern is very similar to the one given in Fig.1. The stride of the state vector components involved in the operation is given by the target qubit $|l - k| = 2^T$. Without presenting further details our simulator includes load balanced implementations of the controlled NOT and the controlled phase shift operations as fundamental two-qubit gates.

6 Results

An efficient parallelization technique was applied to the memory bounded problem of simulating ideal quantum computers, based on hybrid usage of MPI and inner node OpenMP communication using 1,2 and 4 threads. A compact state vector referencing reduces significantly cache misses produced by irregular access to widely separated parts of the memory. An algorithm built up from elementary one- and two-qubit gates scales very well on the IBM p690 up to the max. available memory of 3 TB using 1024 processors (keeping the local state vector size fixed at 2 GB memory per processor).

7 Acknowledgments

Our presented investigations of simulating ideal quantum computers on high-end classical computer systems are part of an ongoing collaboration with H. De Raedt, K. Michielsen and K. De Raedt from the University of Groningen. We thank them for the initial incentive and a series of very instructive discussions.

References

1. ”Quantum Computation and Quantum Information”, M.A. Nielsen and I.L. Chuang, Cambridge University Press, Cambridge, 2004.
2. P. W. Shor, SIAM J. Computing 26, 1484 (1997).
3. L. K. Grover, Phys. Rev. Lett. 79, 4709 (1997).
4. L.M.K. Vandersypen, M. Steffen, G. Breyta, C.S. Yannoni, M.H. Sherwood, and I.L. Chuang, Nature 414, 883 (2001).
5. H. Haeffner et al., Nature 438, 643-646 (01 Dec 2005).
6. D. P. Di Vincenzo, http://arxiv.org/abs/quant-ph/0002077.
7. W. H. Zurek, Phys. Rev. D 24, 1516 (1981), Phys. Rev. D 26, 1862 (1982); E. Joos and H. D. Zeh, Z. Phys. B 59, 223 (1985).
8. S. Lloyd, Science **261**, 1569 (1993).
9. A. Barenco et al., Phys. Rev. A 52, 3457 (1995).