General-purpose parallel simulator for quantum computing

Jumpei Niwa,[1,*] Keiji Matsumoto,[2] and Hiroshi Imai[1,2]
[1]*Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan*
[2]*ERATO, Project Quantum Computation and Information, JST Daini Hongo White Building 201, 5-28-3, Hongo, Bunkyo-ku, Tokyo, Japan*
(Received 11 May 2002; published 30 December 2002)

With current technologies, it seems to be very difficult to implement quantum computers with many qubits. It is therefore of importance to simulate quantum algorithms and circuits on the existing computers. However, for a large-size problem, the simulation often requires more computational power than is available from sequential processing. Therefore, simulation methods for parallel processors are required. We have developed a general-purpose simulator for quantum algorithms/circuits on the parallel computer (Sun Enterprise4500). It can simulate algorithms/circuits with up to *30 qubits*. In order to test efficiency of our proposed methods, we have simulated Shor's factorization algorithm and Grover's database search, and we have analyzed robustness of the corresponding quantum circuits in the presence of both decoherence and operational errors. The corresponding results, statistics, and analyses are presented in this paper.

DOI: 10.1103/PhysRevA.66.062317 PACS number(s): 03.67.Lx, 03.65.Yz

I. INTRODUCTION

Modern computers, not only PCs, but also super computers, are based on the semiconductor switches. As their sizes have gotten smaller, their processing speed has gotten faster and their processing data has gotten larger. However, it has become apparent that if development of computing technology is to follow Moore's law, at some stage, probably within the next 10–20 years, the number of atoms in the structures will become so small that the underlying physical phenomena will not follow the laws of classical physics, but the laws of quantum physics. Therefore, it will be necessary to take quantum effects into account when designing new, more powerful computers.

A quantum computer [1,2] could solve efficiently some important algorithmic problems using superposition and interference principles of quantum mechanics. When an atom or a particle is used as a quantum bit, then quantum mechanics says that it can be prepared as a coherent superposition of two basis states $|0\rangle$ and $|1\rangle$. Strings of qubits can be *entangled* and encode in some sense a vast amount of information. Quantum computers can support entirely new kinds of computation, with qualitatively new algorithms based on quantum principles. Shor [3] proposed polynomial quantum algorithms for integer factoring and discrete logarithm computation. Grover [4] suggested a quantum search algorithm that achieves quadratic speedup with respect to classical ones.

Having such theoretical results, a natural question is whether we could ever build quantum computers. One of the obstacles is *decoherence*—an interaction of quantum systems with their environment that destroys fragile superposition in the quantum systems in which computations are performing. Decoherence is caused because the environment is ''measuring'' the state of a quantum system by interacting with it. Second one is *inaccuracy* (operational errors). Quantum computers are actually analog: that is, they have a continuum of states. For example, one of the most common quantum gates ''rotates a quantum bit by an angle θ.'' When applying this gate, there is always some inaccuracy in the rotation.

With current technologies, it seems to be very difficult to implement quantum computers with many qubits. It is therefore of importance to simulate quantum algorithms and circuits on the existing computers. The purpose of the simulation is:

(1) To investigate quantum algorithms behavior. Of course, we have already known that Grover's search algorithm is optimal. That is, no quantum algorithm can search N items using fewer than $\Omega(\sqrt{N})$ accesses to the search oracle. However, we do not know whether Shor's factorization algorithm is optimal. Therefore, it is important to check by simulations how effective the improved Shor's factorization algorithm is.

(2) To analyze performance and robustness of quantum circuits in the presence of decoherence and operational errors. Not to mention, quantum error-correcting codes are effective in fighting decoherence and operational errors. It is one of our goals to establish useful quantum error-correcting codes. As the first step, this paper aims to check effects of these errors in practice.

Simulations often require more computational power than is usually available on sequential computers. Therefore, we have developed a simulation method for parallel computers. That is, we have developed a general-purpose simulator for quantum algorithms and circuits on a parallel computer: symmetric multiprocessor (shown in Fig. 1).

II. BASIC DESIGN

A. Registers

The simulation is for circuit model of quantum computation. A set of n qubits is called a *register* of size n. The general state of the n-qubit register is

*Electronic address: niwa@is.s.u-tokyo.ac.jp

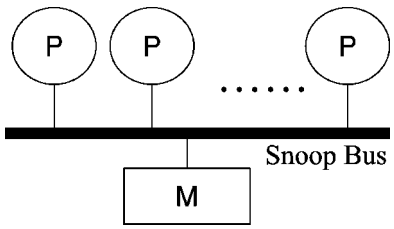

FIG. 1. SMP (symmetric multiprocessors).

$$|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \quad \text{where} \quad \alpha_i \in \mathcal{C}, \quad \sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1.$$

That is, the state of an n-qubit register is represented by a unit-length complex vector in $\mathcal{H}_{2^n}$. In a classical computer, to store a complex number $\alpha = x + iy$, one needs to store a pair of real numbers (x,y). In our implementation each real number will be represented by a *double precision word*. The double precision word is 16 bytes (64 bits) on most of the computers. 2^{n+4} bytes memory are therefore required to deal with the state of an n-qubit register in a classical computer.

B. Evolution

The time evolution of an n-qubit register is determined by a unitary operator of $\mathcal{H}_{2^n}$. The size of the matrix is $2^n \times 2^n$. In general, it requires $2^n \times 2^n$ space and $2^n(2^{n+1}-1)$ arithmetic operations to perform classically such an evolution step.

However, we mostly use operators that have simple structures when we design quantum circuits. That is, an evolution step is performed by applying a unitary operator (2×2) to a single qubit (a single qubit gate) or by applying the controlled unitary operator such as a controlled-NOT gate. It requires only 2×2 space and 3×2^n arithmetic operations to simulate such an evolution step as explained below.

1. Single-qubit gates

Suppose that the MSB (most significant qubit) is 0th qubit. When a unitary matrix $\boldsymbol{U} = \left(\begin{smallmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{smallmatrix}\right)$ is applied to the ith qubit (Fig. 2), the overall unitary operation applied to the n-qubit register state has the form $X = (\otimes_{k=0}^{i-1} I) \otimes U \otimes (\otimes_{k=i+1}^{n-1} I)$. $2^n \times 2^n$ matrix X is the sparse regular matrix shown in Fig. 3.

That is, all the S_i are the same. We therefore do not have to generate X explicitly. We have to only store the 2×2 matrix U. Since there are only two nonzero elements for each

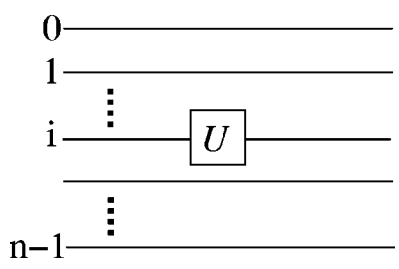

FIG. 2. Single-qubit gate.

$$X = \underbrace{\begin{pmatrix} S_0 & & & 0 \\ & S_1 & & \\ & & \ddots & \\ & & & S_{2^i-2} \\ 0 & & & & S_{2^i-1} \end{pmatrix}}_{2^n} \quad \text{where } S_k = \underbrace{\begin{pmatrix} u_{11} & 0 & u_{12} & 0 \\ & \cdots & & \cdots \\ 0 & u_{11} & 0 & u_{12} \\ u_{21} & 0 & u_{22} & 0 \\ & \cdots & & \cdots \\ 0 & u_{21} & 0 & u_{22} \end{pmatrix}}_{2^{n-i}}$$

$$(0 \le k < 2^i)$$

FIG. 3. Total unitary matrix.

row in X, the evolution step (i.e., multiplication of a matrix and a vector) is simulated in $3 \cdot 2^n$ arithmetical operations.

Parallelization. Of course, the evolution step $(X|\phi\rangle)$ can be executed in parallel. Let 2^P be the number of processors available in the simulating system. The evolution step is decomposed into a sequence of submatrix-subvector multiplication M_k $(0 \le k < 2^i)$. M_k is defined as $S_k \phi_k$, that is, as the multiplication of a submatrix S_k $(2^{n-i} \times 2^{n-i})$ and a subvector ϕ_k whose length is 2^{n-i} (shown in Fig. 4). Note that there are no data dependencies between M_k and M_l $(k \ne l)$. Therefore, M_k and M_l can be executed in parallel. We assign a sequence of computation:

$$\underbrace{M_{p2^{i-P}}, M_{p2^{i-P}+1}, \ldots, M_{(p+1)2^{i-P}-1}}_{2^{i-P}}$$

to a processor p $(0 \le p < 2^P)$. That is, the processor p computes 2^{i-P} submatrix-subvector multiplications, and the rests of multiplications are performed in other processors in parallel. After each processor has finished computations that were assigned to it, it executes a synchronization primitive, such as the barrier, to make its modifications to the vector (ϕ), that is, the state of the register visible to other processors.

When the number of submatrices is smaller than the number of processors (i.e., $2^i < 2^P$), it is inefficient to assign the computation $M_k (= S_k \phi_k, 0 \le k < 2^i)$ to one processor as described above. It can cause a load imbalance in the simulation system. In this case, we should decompose the computation M_k itself to improve parallel efficiency. Each submatrix S_k is then divided into 2^{P+1} chunks of rows. Each chunk of rows, R_j $(0 \le j < 2^{P+1})$, contains the adjoining $2^{n-i-(P+1)}$ rows of S_k. The multiplications using the chunk of rows R_j and R_{2^P+j} are assigned to a processor j as described in Fig. 5. This decomposition is applied to all M_k computations $(0 \le k < 2^i)$.

Note that the computation using jth row of the submatrix must be always paired with that using $(j + 2^{n-i-1})$th row when we use an "in-place" algorithm (i.e., The results of $X|\phi\rangle$ are stored in $|\phi\rangle$).

That is, multiplications using the chunk of rows R_j and R_{2^P+j} are assigned to the same processor j. This is because there are dependencies across processors. Consider for example the following case.

$$X|\phi\rangle = \begin{pmatrix} S_0 & & & 0 \\ & S_1 & & \\ & & \cdots & \\ & & & S_{2^i-2} \\ 0 & & & S_{2^i-1} \end{pmatrix} \begin{pmatrix} \phi_0 \\ \phi_1 \\ \cdots \\ \phi_{2^i-2} \\ \phi_{2^i-1} \end{pmatrix} \begin{matrix} \} (processor\ 0) \\ \\ \cdots \\ \\ \} (processor\ 2^P) \end{matrix}$$

$$\text{where } \phi_k = \begin{pmatrix} \alpha_{k2^{n-i}} \\ \alpha_{k2^{n-i}+1} \\ \cdots \\ \alpha_{(k+1)2^{n-i}-2} \\ \alpha_{(k+1)2^{n-i}-1} \end{pmatrix} \quad (0 \le k < 2^i)$$

FIG. 4. Computation decomposition in the general case.

$$\begin{bmatrix} xu_{11}+yu_{12} \\ \cdots \\ \cdots \\ xu_{21}+yu_{22} \\ \cdots \\ \cdots \end{bmatrix} = \begin{bmatrix} u_{11} & & \mathbf{0} & u_{12} & & \mathbf{0} \\ & \cdots & & & \cdots & \\ \mathbf{0} & & u_{11} & \mathbf{0} & & u_{12} \\ u_{21} & & \mathbf{0} & u_{22} & & \mathbf{0} \\ & \cdots & & & \cdots & \\ \mathbf{0} & & u_{21} & \mathbf{0} & & u_{12} \end{bmatrix} \times \begin{bmatrix} x \\ \cdots \\ \cdots \\ y \\ \cdots \\ \cdots \end{bmatrix}.$$

If the first element is computed and the result $(xu_{11}+yu_{12})$ is stored before the fourth element is computed, then the result of the fourth element computation is not $xu_{21}+yu_{22}$, but $(xu_{11}+yu_{12})u_{21}+yu_{22}$. However, this is wrong. To avoid this situation, all the processors have to execute barrier operations before storing the results of computations. However, a barrier operation per store operation can cause heavy overheads.

Therefore, the first-element computation and fourth-element computation should be assigned to the same processor. Then, the data dependencies are not cross processor but in processor.

First, the processor computes $xu_{11}+yu_{12}$ and stores the result in a temporary variable t_1 on the local storage area (i.e., stack). Second, the processor itself computes the result $xu_{21}+yu_{22}$ and stores it in the fourth element. Third, the processor stores the contents of the temporary variable t_1 in the first element. In this way, we can avoid the above wrong situation without performing synchronization primitives. If

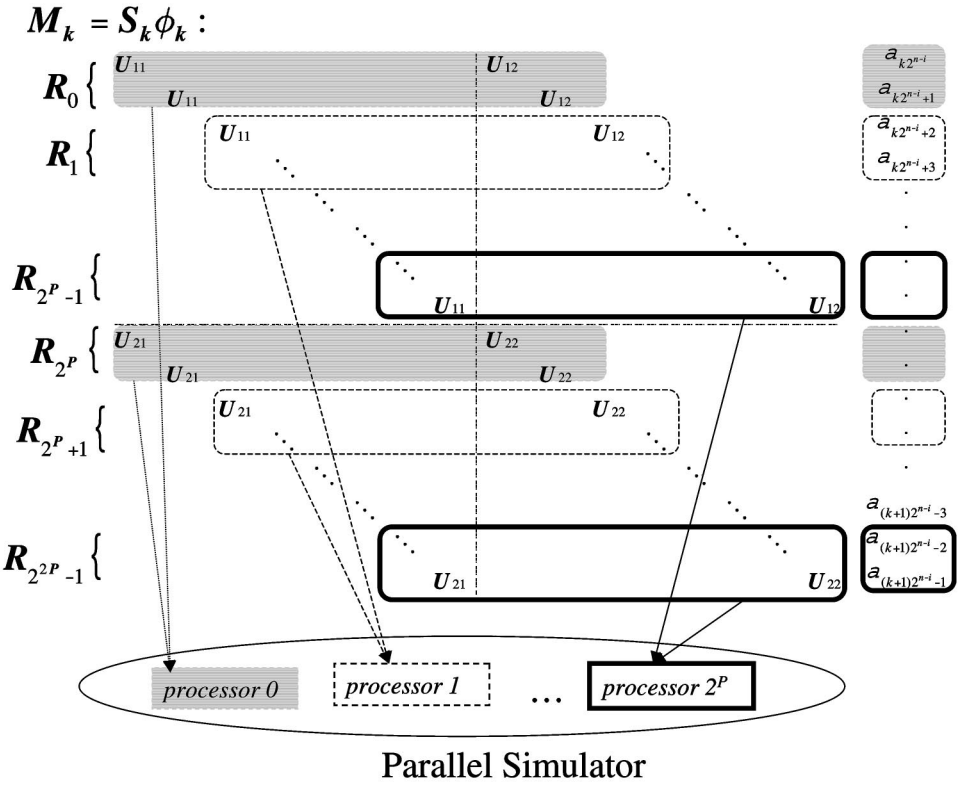

FIG. 5. Computation decomposition in the large subblock case.

there are no overheads for parallel execution, the time complexity is thus reduced to $O(2^{n-P})$, where 2^P is the number of processors available in the system.

2. Controlled qubit gates

Suppose that a unitary matrix $\boldsymbol{U}=\left(\begin{smallmatrix}u_{11} & u_{12}\\ u_{21} & u_{22}\end{smallmatrix}\right)$ is applied to the ith qubit if and only if the cth bit (controlled bit) is 1 (Fig. 7).

Let CTX be the overall unitary matrix $(2^n\times 2^n)$. First, we consider the matrix X mentioned in Sec. II B 1 as if there were no controlled bits. Then, for each j $(0\leqslant j<2^n-1)$, the jth row of CTX ($CTX[j]$) is defined as follows:

$$CTX[j]=\begin{cases} X[j] & \text{the } c\text{th bit in } j \text{ is } 1,\\ I[j] & \text{the } c\text{th bit in } j \text{ is } 0,\end{cases}$$

where $I[j]$ is jth row of the unit matrix I and $X[j]$ is jth row of the matrix X. In this case, we also do not have to generate CTX or X explicitly. We have only to store the 2×2 matrix U. It is easy to extend this method to deal with the case of many controlled bits. The evolution step is executed in parallel as described in Sec. II B 1. Therefore, the simulation time is $O(2^{n-P})$ if there are no overheads for parallel execution (2^P is the number of processors available in the simulation system.)

f-controlled U gate is also simulated when f is a boolean function. It is similar to the controlled U gate. But in this case, the U gate is applied to the target bit iff $f(c)=1$ (the cth bit is the controlled bit). This is used in the simulation of Grover's search algorithm [4].

3. Measurement gates

The measurement step for an n-qubit register state $|\phi\rangle=\Sigma_{j=0}^{2^n-1}\alpha_j|j\rangle$ is simulated in $O(2^n)$ time as follows.

(1) A random number r, $0\leqslant r<1$, is generated

(2) An integer i, $0\leqslant i\leqslant 2^n-1$, is determined such that

$$\sum_{j=0}^{i-1}|\alpha_j|^2\leqslant r<\sum_{j=0}^{i}|\alpha_j|^2.$$

We assume that the measurement is done with respect to the standard basis $\{|i\rangle\}$.

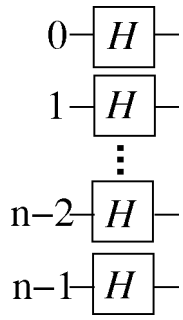

FIG. 6. H_n circuit.

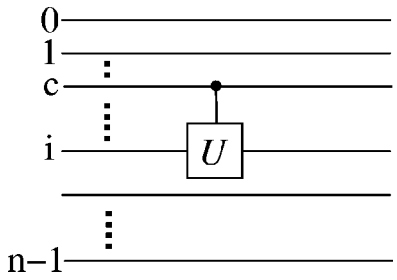

FIG. 7. Controlled qubit gate.

C. Basic circuits

1. Hadamard transform

The Hadamard transform H_n is defined as follows:

$$H_n|x\rangle=\frac{1}{\sqrt{2^n}}\sum_{y\in 0,1^n}(-1)^{x\cdot y}|y\rangle,\quad \text{for } x\in\{0,1\}^n.$$

H_n is implemented by the circuit in Fig. 6, where $\boldsymbol{H}=(1/\sqrt{2})\left(\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right)$. Note that simulation requires $O(n2^{n-P})$ time, if there are no overheads for parallel execution (2^P is the number of processors available in the simulation system).

2. Quantum Fourier transform

Quantum Fourier transform (QFT) is a unitary operation that essentially performs discrete Fourier transform (DFT) on quantum register states. QFT maps a quantum state $|\phi\rangle=\Sigma_{x=0}^{2^n-1}\alpha_x|x\rangle$ to the state $\Sigma_{x=0}^{2^n-1}\beta_x|x\rangle$,

$$\beta_x=\frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}\omega^{xy}\alpha_y,\quad \omega=e^{2\pi i/2^n}.$$

The circuit implementing the QFT is shown in Fig. 8. $\boldsymbol{H}$ is the Hadamard gate, and $\boldsymbol{R_d}$ is the phase-shift gate denoted as $\left(\begin{smallmatrix}1 & 0\\ 0 & e^{i\pi/2^d}\end{smallmatrix}\right)$.

For general n, this circuit is of size $O(n^2)$ [20]. Therefore, the evolution step is simulated in $O(n^2 2^{n-P})$ time if there are no overheads for parallel execution (There are 2^P processors available in the system.) Of course, we can reduce the circuit size to $O(n\log(n/\epsilon))$ [5,6], if we settle to the implementation with a fixed accuracy (ϵ), because the controlled phase-shift gates acting on distantly separated qubits contribute only exponentially small phases. In this case, the evolution step is simulated in $O(n\log(n/\epsilon)2^{n-P})$ steps if there are no overheads for parallel execution.

If we regard QFT transform as a *black box operator*, we do not have to use this quantum circuit in the simulator to

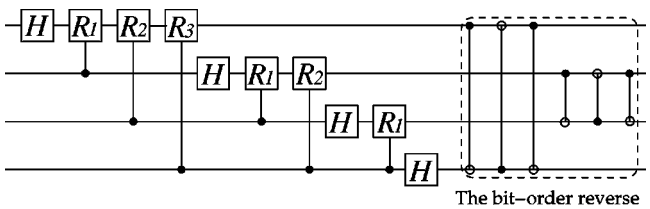

FIG. 8. The QFT_{2^n} circuit $(n=4)$.

perform QFT transformation. We can use fast Fourier transform (FFT) in the simulator instead of the QFT circuit if we suppose that QFT circuit has no error. FFT algorithm requires only $O(n2^{n-P})$ steps if there are no overheads for parallel execution. Of course, FFT gives the exact solution. We use the eight-radix in-place FFT algorithm.

3. Arithmetical circuits

Arithmetical circuits are important for quantum computing [7]. In the Shor's factoring algorithm [3], arithmetical circuits are needed to compute modular exponentiation. Therefore, according to Ref. [8], we have implemented the modular exponentiation circuit by using constant adders, constant modular adders, and constant multipliers. $x^a \pmod N$ can be computed using the decomposition,

$$x^a \pmod N = \prod_{i=0}^{l-1} [(x^{2^i})^{a_i} \pmod N],$$

where

$$a = \sum_{i=0}^{l-1} a_i 2^i [= a_{l-1} a_{l-2}, \ldots, a_0 (\text{binary representation})].$$

Thus, modular exponentiation is just a chain of multiplications where each factor is either 1 $(a_i=0)$ or x^{2^i} $(a_i=1)$. Therefore, the circuit is constructed by the pairwise controlled constant multipliers [21].

Let N be an n-bit number, and a a $2n$-bit number (that is, l is equal to $2n$ in the above equation.) in the Shor's factoring algorithm because a is as large as N^2. $n+1$ qubits are required as the work space for the controlled multiplier and $n+4$ for the controlled adders. The total number of required qubits becomes $5n+6$.

The circuit is constructed with the $O(l)$ [that is, $O(n)$] pairwise controlled constant multipliers. A controlled constant multiplier consists of $O(n)$ controlled constant modular adders. A controlled constant modular adder consists of five controlled constant adders. A controlled constant adder consists of $O(n)$ XOR (controlled-NOT) gates. Thus, one modular exponentiation circuit requires $O(n^3)$ gate. Details are described in Ref. [8]. It is simulated in $O(n^3 2^{n-P})$ steps if there are no overheads for parallel execution (2^P is the number of processors available in the simulation system).

III. ERROR MODEL

A. Decoherence

We consider a quantum depolarizing channel as the decoherence error model. In this channel, each qubit is left unchanged with probability $1-p$. Otherwise, each of the following possibilities has probability $p/3$: the qubit is negated, or its phase if changed, or it is both negated and its phase is changed.

B. Operational errors

In general, all single-qubit gates can be generated from *rotations*,

$$\boldsymbol{U_R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix},$$

and *phase shifts*,

$$\boldsymbol{U_{P1}}(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$$

and

$$\boldsymbol{U_{P2}}(\phi) = \begin{pmatrix} e^{i\phi} & 0 \\ 0 & 1 \end{pmatrix}.$$

For example,

$$H = \boldsymbol{U_R}(\pi/4)\boldsymbol{U_{P1}}(\pi),$$

and

$$G_{\mathrm{NOT}} = \boldsymbol{U_R}(\pi/2)\boldsymbol{U_{P1}}(\pi).$$

The simulator represents inaccuracies by adding small deviations to the angles of rotation θ and ϕ. Each error angle is drawn from the Gaussian distribution with the standard deviation (σ).

IV. PRELIMINARY EXPERIMENTS

We describe the simulation environment and some experiments about basic quantum circuits.

A. Simulation environment

We have designed a simulator for the parallel computer: Sun Enterprise 4500 (E4500). E4500 has eight UltraSPARC-II processors (400 MHz) with 1 MB E-cache and 10 GB memory. The system clock is 100 MHz. OS is SOLARIS 2.8 (64bit OS). The simulator is written in the C language and the compiler we use is Forte Compiler 6.0. The compiler option "-xO5 -fast -xtarget=ultra2 -xarch=v9" is used. We use the solaris thread library for multiprocessor execution. Under this environment, if we use an in-place algorithm, *30-qubit quantum register states can be simulated*.

B. Quantum Fourier transform

Table I shows QFT execution time by the simulator that uses QFT circuit and (classical) FFT algorithm. Numerical error value ranges from 10^{-15} to 10^{-14}. Recall that 2^P is the number of processors available in the simulation system. FFT algorithm requires $O(n2^{n-P})$ steps, and QFT circuit requires $O(n^2 2^{n-P})$ steps for n-qubit quantum register, if there are no overheads for parallel execution. The execution time is increased exponentially with respect to n. Table I shows that the execution time of FFT is about 20–30 times as fast as that of QFT circuit. Both execution times are de-

TABLE I. QFT execution time (sec).

Qubits (n)	Algorithm	Number of processors 1	2	4	8
20	Circuit	26.08	7.25	5.01	5.33
	FFT	1.21	0.92	0.72	0.53
22	Circuit	124.78	66.96	38.03	23.40
	FFT	5.01	3.71	2.79	1.83
24	Circuit	643.02	331.98	183.01	137.7
	FFT	20.00	12.61	8.40	5.84
26	Circuit	2745.56	1469.73	799.57	526.82
	FFT	113.29	73.08	48.39	32.84
28	Circuit	12597.8	6738.13	3661.51	2338.19
	FFT	567.19	319.16	205.98	142.01
29	Circuit	31089.6	16790.6	9189.68	5811.49
	FFT	1232.16	697.68	423.00	286.29

creased when the number of processors is increased. The speedup ratios on 8-processor execution are about 4–5. The reason why the speedup ratios on 8-processor execution are not 8 is that the parallel execution has some overheads that single-processor execution does not have. The parallel execution overheads are operating system overheads (multithreads creation, synchronization, and so on), load imbalance, memory-bus saturation, memory-bank conflict, false sharing, and so on. For small-size problems, the ratio of overheads to the computation for parallel execution is relatively large and speedup ratios on multiprocessor execution may be less than four. The decoherence and operational errors experiment for QFT is described in Sec. V.

C. Hadamard transform

Table II shows Hadamard transform (HT) execution time. HT circuit requires $O(n2^{n-P})$ steps for n-qubit quantum register. The speedup ratio on 8-processor execution becomes about 5.

Effect of errors

We have investigated the decrease of the $|0\rangle\langle 0|$ term in the density matrix for the 20-qubit register.

Decoherence errors. We have analyzed decoherence in the HT circuit on the depolarizing channel. Of course, the simulation deals with pure states. Therefore, the experiments were repeated 10 000 times and we use the average values.

TABLE II. HT execution time (sec).

Qubits (n)	Number of processors 1	2	4	8
20	2.38	1.18	0.76	0.40
22	10.85	5.73	3.20	1.35
24	46.94	24.96	13.40	9.58
26	205.81	109.97	58.83	38.71
28	887.40	467.71	253.82	167.31
29	2027.9	1081.1	592.08	395.81

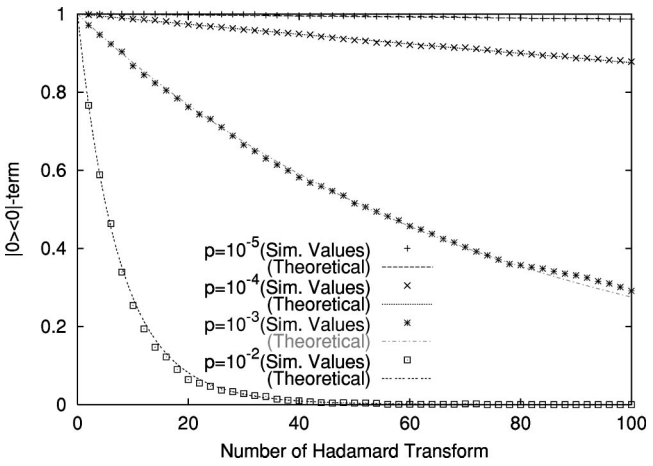

FIG. 9. Decrease of the $|0\rangle\langle 0|$ term in the density matrix (20 qubits).

Each experiment uses different initial random seed. The initial state of the quantum register is $|00\cdots 0\rangle = |0\rangle$. HT circuit is applied to the quantum register over and over. The x axis in Fig. 9 shows the even iteration number. If there are no errors (i.e., the error probability is 0) and the number of iterations is even, the state remains to be $|0\rangle$ and $|0\rangle\langle 0|$ term in the density matrix remains 1. Figure 9 shows how decoherence errors degrade for the $|0\rangle\langle 0|$ term. The noise degrades the $|0\rangle\langle 0|$ term significantly if the error probability is greater than 10^{-3}. When the error probability is 10^{-2}, the $|0\rangle\langle 0|$ term is decreased in exponential order in proportion to the number of iterations.

In this easy case, we can compute $|0\rangle\langle 0|$ term in the density matrix also theoretically. First, consider the 1-qubit case. Let p be the error probability and ρ_k be the density matrix after the HT circuit is applied to the quantum register k times. The density matrix ρ_{k+1} is then calculated as follows:

$$\rho_{k+1} = (1-p)H\rho_k H^* + \frac{p}{3}\sigma_x H\rho_k H^*\sigma_x^* + \frac{p}{3}\sigma_y H\rho_k H^*\sigma_y^* + \frac{p}{3}\sigma_z H\rho_k H^*\sigma_z^*.$$

When the initial state of the quantum register is $|0\rangle$ and k is even, ρ_k is calculated as follows:

$$\rho_k = \frac{1}{2}\begin{pmatrix} 1+\left(1-\frac{4}{3}p\right)^k & 0 \\ 0 & 1-\left(1-\frac{4}{3}p\right)^k \end{pmatrix}.$$

In the n-qubit case, we can calculate the density matrix similarly if the initial state of the quantum register is $|0,\ldots,0\rangle$ and k is even. $|0\rangle\langle 0|$ term of ρ_k is

$$\left(\frac{1+\left(1-\frac{4}{3}p\right)^k}{2}\right)^n.$$

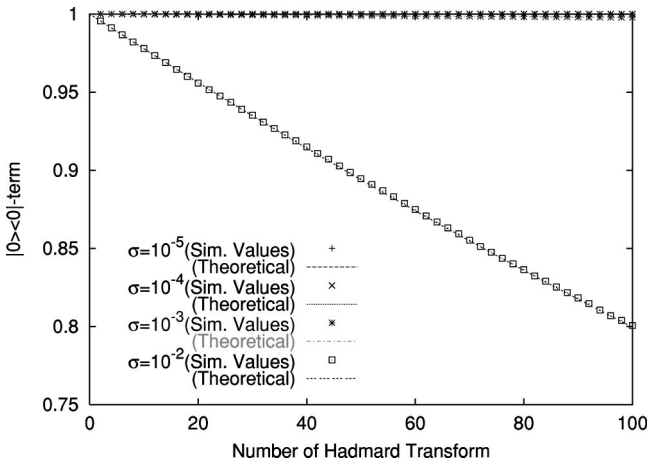

FIG. 10. Decrease of the $|0\rangle\langle 0|$ term in the density matrix (20 qubits).

Figure 9 also shows this theoretical value of $|0\rangle\langle 0|$ term in the density matrix if $p=10^{-5}\sim 10^{-2}$ and $n=20$. We can see that the simulations and the theoretically computations yield almost the same result.

Operational errors. The simulator implements ''inaccuracies'' by adding small deviations to two angles of rotations. Since $H=\boldsymbol{U_R}(\pi/4)\boldsymbol{U_{P1}}(\pi)$, we add small deviations x and y to $(\pi/4)$ and π, respectively. That is, we use $H(x,y)=\boldsymbol{U_R}((\pi/4)+x)\boldsymbol{U_{P1}}(\pi+y)$ as H gate in this experiment. x and y are drawn from the Gaussian distribution with the standard deviation σ. As mentioned above, the experiments were executed 10 000 times and we use the average value. Each experiment uses different initial random seed. Figure 10 shows how operational errors degrade the $|0\rangle\langle 0|$ term when $\sigma=10^{-5}\sim 10^{-2}$ and $n=20$. The $|0\rangle\langle 0|$ term is not affected by the operational error if σ is less than 10^{-2}.

In this case, we can also compute theoretically the $|0\rangle\langle 0|$ term in the density matrix. First, consider the 1-qubit case. Let ρ_k be the density matrix after HT circuit is applied to the quantum register k times. The density matrix ρ_{k+1} can be calculated as follows:

$$\rho_{k+1}=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}H(x,y)\rho_k H(x,y)^* p(x)p(y)dxdy,$$

where $p(z)=(1/\sqrt{2\pi}\sigma)e^{-z^2/2\sigma^2}$. If the initial state of the quantum register is $|00\cdots 0\rangle=|0\rangle$, ρ_k can be expressed as follows:

$$\rho_k=\frac{1}{2}\begin{pmatrix}1+e^{-(\sigma^2/4)9k} & 0\\ 0 & 1-e^{-(\sigma^2/4)9k}\end{pmatrix}.$$

As for the general n-qubit case, we can calculate the density matrix similarly if the initial state of the quantum register is $|00\cdots 0\rangle$, and k is even. $|0\rangle\langle 0|$ term of ρ_k is then

$$\left(\frac{1+e^{-(\sigma^2/4)9k}}{2}\right)^n.$$

TABLE III. Combined effects for HT.

	Operational (σ)			
Decoherence (p)	0	10^{-5}	10^{-4}	10^{-3}
0	1.0000	1.0000	0.9999	0.9977
10^{-5}	0.9870	0.9870	0.9849	0.9797
10^{-4}	0.9010	0.9010	0.8909	0.8780
10^{-3}	0.2910	0.2790	0.2779	0.2664

Figure 10 also shows this theoretical value of $|0\rangle\langle 0|$ term in the density matrix if the standard deviation $\sigma=10^{-5}\sim 10^{-2}$ and $n=20$. It follows from the theoretical computation that $|0\rangle\langle 0|$ term decreases exponentially with respect to the number of iterations k.

Operational and decoherence errors. Each element of Table III represents the $|0\rangle\langle 0|$ term of the density matrix after HT is applied to the state $|0\rangle$ of a 20-qubit register 10 000 times.

The combined effect of two factors may be worse than in case of each factor alone. That is to say, the effect seems to be the product of each factor. Table III shows this situation.

V. EXPERIMENTS

A. Shor's factorization algorithm

We investigate behavior of Shor's factorization algorithm. The point is (1) how effective the improved algorithm [9] is, (2) effects of decoherence errors and operational errors.

First, we review the algorithm briefly.

Input. An l bit odd number n that has at least two distinct prime factors.

Output. A nontrivial factor of n:

(1) Choose an arbitrary $x\in\{1,2,\ldots,n-1\}$.

(2) (Classical step) Compute $d=\gcd(x,n)$ (greatest common divisor of x and n) using Euclid's algorithm. If $d>1$, output d and stop.

(3) (Quantum step) Try to find the order of x:

(a) Initialize an l-qubit register and a $2l$-qubit register to state $|0\rangle|0\rangle$.

(b) Apply HT to the second register.

(c) Perform modular exponentiation operation, that is, $|0\rangle|a\rangle\rightarrow|x^a(\mathrm{mod}\,n)\rangle|a\rangle$.

(d) Measure the first register and apply the QFT to the second register and measure it. Let y be the result.

(4) (Classical step) Find relatively prime integers k and r $(0<k<r<n)$, such that $|(y/2^{2l})-(k/r)|\leqslant 1/2^{(2l+1)}$ by using the continued fraction algorithm. If $x^r\not\equiv 1(\mathrm{mod}\,n)$, or if r is odd, or if $x^{r/2}\equiv\pm 1(\mathrm{mod}\,n)$, output ''failure'' and stop.

(5) (Classical step) Compute $d_\pm=\gcd(n,x^{r/2}\pm 1)$ using Euclid's algorithm. Output numbers $d_\pm$ and stop.

TABLE IV. Execution time in Shor's factorization algorithm, when $n=15$ and $x=11$. (All quantum operations are executed on the circuit.)

Modular exponentiation	QFT
18184 (sec)	0.64270 (sec)

TABLE V. Number of needed iterations for Shor's factoring algorithm.

	Number of iterations		
	Theoretical	Simulation	
n		Original	Improved
21311(=211×101)	15.79	6.690	1.760
21733(=211×103)	15.85	8.990	2.356
22999(=211×109)	16.00	6.360	1.730
22523(=223×101)	15.88	5.480	1.770
22927(=227×101)	15.91	3.790	1.470
22969(=223×103)	15.94	8.050	2.070
23129(=229×101)	15.92	7.133	1.636

When the simulator performs all the step-3 operations (not only QFT but also modular exponentiation) on the quantum circuit, $5l+6$ qubits are totally required, as described in Sec. II C 3. Therefore, the simulator can only deal with 4-bit integer n $(5l+6<=30\rightarrow l\leqslant 4)$. The 4-bit integer that satisfies the input property is only 15. We have tried to factor 15 on the simulator. Beyond our expectation, modular exponentiation is computationally much heavier than QFT.

Modular exponentiation requires $O(l^3 2^{l-P})$ steps and QFT on the circuit requires $O(l^2 2^{l-P})$ steps, when there are 2^P processors available in the simulation system and there are no overheads for parallel execution. Of course, in the classical computer, modular exponentiation consists of basic operations such as addition, multiplication, and division. However, these basic operations are not so heavy if the classical computer is used, because it has the dedicated nonreversible circuit [the so-called (ALU) arithmetic logic unit]. This situation suggests that a brand-new fast quantum algorithm for arithmetic operations is required to factor larger numbers. 15 is not enough to investigate the behavior of Shor's factoring algorithm. In order to factor much larger number in a reasonable time, the simulator performs the step 3(c) and the step 3(d) classically. That is, the modular exponentiation are computed classically and QFT is computed by FFT algorithm in the simulator (see Table IV). In this case, the simulator does not need to generate the first register. Therefore, the simulator can factor about 14–15-bit integers (for example, 23089).

The factoring algorithm succeeds with the probability greater than

TABLE VI. Detailed effect of improved algorithm.

	Ratio of success/failure			
n	1 (Neighbor)	2 (GCD)	3 (SF)	4 (LCM)
21311	27/9	52/19	12/4	3/4
23129	27/9	52/19	12/4	3/4
22999	37/6	47/79	13/8	2/58
22969	41/8	22/82	31/22	1/28
22927	25/3	35/49	18/2	1/28
22523	37/6	45/76	18/22	7/54

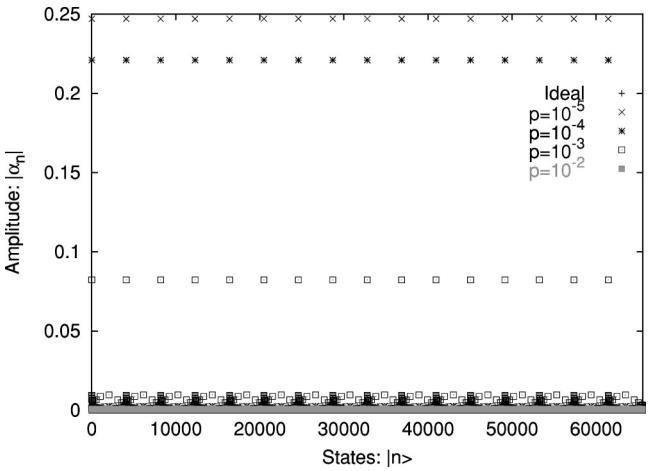

FIG. 11. Amplitude amplification by QFT in the presence of decoherence error (top) and the required number of iterations (bottom) (16 qubits).

$$\mathrm{Prob}_{\mathrm{succ}}(n)=p_{step2}+(1-p_{step2})p_{step3\sim 4}$$
$$=\left(1-\frac{\phi(n)}{n-1}\right)+\frac{\phi(n)}{n-1}\left(\frac{1}{2}\frac{4}{\pi^2}\frac{e^{-\gamma}}{\log\log n}\right),$$

where p_{step2} denotes the probability that the step (2) succeeds and $p_{step3\sim 4}$ denotes the probability that step (3) and the step (4) succeed and γ is the Euler constant, and $\phi(n)$ is the Euler number of n. If the above algorithm is repeated $O(1/\mathrm{Prob}_{\mathrm{succ}}(n))$ times, the success probability can be as close to 1 as desired.

We choose an $n=pq$ where p and q are prime numbers. These kinds of integers are chosen in an RSA cryptosystem (developed by Ronald Rivest, Adi Shamir, and Leonald Adleman) because it is believed that it is hard to factor such integers easily. Recall that $\phi(n)=(p-1)(q-1)$ for such integers. We have experimented with several RSA-type 14 $\sim$15-bit integers.

The simulator repeats the above algorithm until a nontrivial factor of n is found, and records the number of itera-

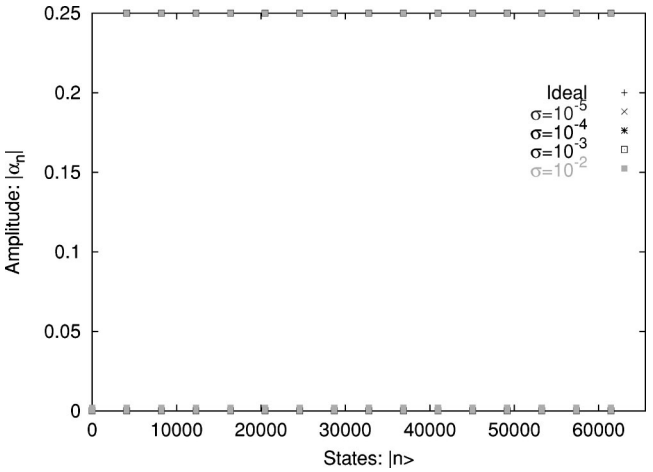

FIG. 12. Amplitude amplification by QFT in the presence of operational error (top) and the required number of iterations (bottom) (16 qubits).

TABLE VII. Combined effects for QFT (16 bit).

	Operational (σ)				
Decoherence (p)	0	10^{-5}	10^{-4}	10^{-3}	10^{-2}
0	1.0000	0.9999	0.9999	0.9999	0.9998
10^{-5}	0.9880	0.9840	0.9860	0.9880	0.9848
10^{-4}	0.8837	0.8897	0.8827	0.8801	0.8980
10^{-3}	0.3287	0.3399	0.3332	0.3209	0.3363
10^{-2}	0.0027	0.0015	0.0019	0.0017	0.0031

tions. The experiment was executed 100 times, and we calculate the average of these recorded iterations. We have compared the simulation values with the theoretical number of needed iterations [i.e., $1/\mathrm{prob}_{\mathrm{succ}}(n)$]. The results are shown in the Table V. Theoretical values (''theoretical'' column) are about only $2\sim4$ times as large as simulation values (''Original'' column). Although much more simulations are required, the theoretical values seem to be fairly good.

As suggested in Ref. [9], the algorithm was optimized to perform less quantum computation and more (classical) postprocessing.

(1) Neighbor y check. No relatively prime integers k and r are found by using the continued fraction algorithm, then it is wise to try $y\pm1$, $y\pm2$.

(2) gcd check. Even if $x^r\not\equiv1 \pmod n$, try to compute $d_\pm=\gcd(n,x^{r/2}\pm1)$.

(3) Small factor check. If $x^r\not\equiv1(\mathrm{mod}\,n)$, it is wise to try $2r$, $3r$, ... This is because if $(y/2^{2l})\approx(k/r)$, where k and r have a common factor, this factor is likely to be small. Therefore, the observed value of $(y/2^{2l})$ is rounded off to (k'/r') in the lowest terms.

(4) lcm (least common muliplier) check. If two candidates for r, r_1, and r_2, have been found, it is wise to test $\mathrm{lcm}(r_1,r_2)$ as a candidate for r.

We have tested how much the algorithm is improved by these modifications. The results are also shown in Table V (''Improved'' column). The number of iterations is reduced to about $1/5\sim2/5$. The detailed effect of the improved algorithm is described in Table VI. Each element of Table VI represents the ratio s/f, where s means the number of success iterations and f is the number of failure iterations. For example, for $n=23129$, the first optimization, ''neighbor check'' is performed for $27+9=36$ iterations and a candidate of the order is found successfully in 27 iterations. It seems that the second optimization ''gcd check'' works well for all n that we have experimented with. From this result, we can see that even if $x^r\not\equiv1(\mathrm{mod}\,n)$, $d_\pm=\gcd(n,x^{r/2}\pm1)$ often becomes a factor of n. That is, even if the candidate r is not equal to $ord(x)$ (an order of x), r may be a divisor of $ord(x)$. That is, $\mathbf{N}\ni\exists a>1$, $a\cdot r=ord(x)$. In this case, the following equation holds when r is even.

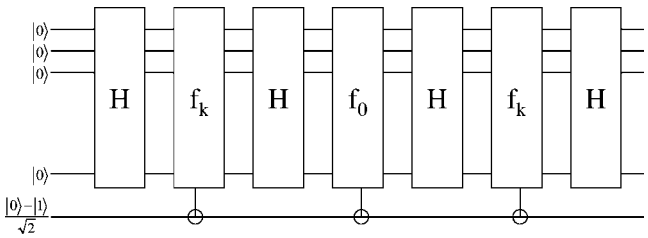

FIG. 13. The circuit of Grover's algorithms.

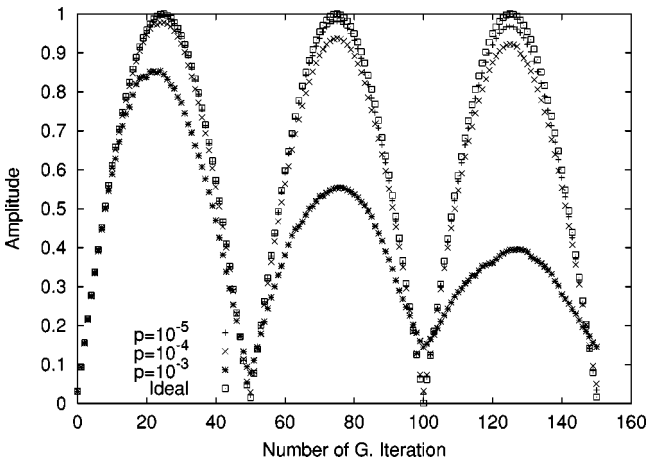

FIG. 14. Decrease of the amplitude of the correct element in the presence of decoherence errors (10 qubits).

$$0(\mathrm{mod}\,n)\equiv x^{ord(x)}-1\equiv(x^r-1)(x^{(a-1)r}+x^{(a-2)r}+\cdots+1)$$
$$\equiv(x^{r/2}-1)(x^{r/2}+1)(x^{(a-1)r}+x^{(a-2)r}+\cdots1).$$

Thus, there is the possibility that n and $x^{r/2}\pm1$ have a common nontrivial factor.

B. Effect of errors

We have analyzed decoherence and operational errors in the QFT circuit.

Decoherence errors. We assume that each qubit is left intact with probability $1-p$ and it is affected by each of the error operators $\sigma_x,\sigma_y,\sigma_z$ with the same probability $(p/3)$, each time the register is applied by the controlled rotation

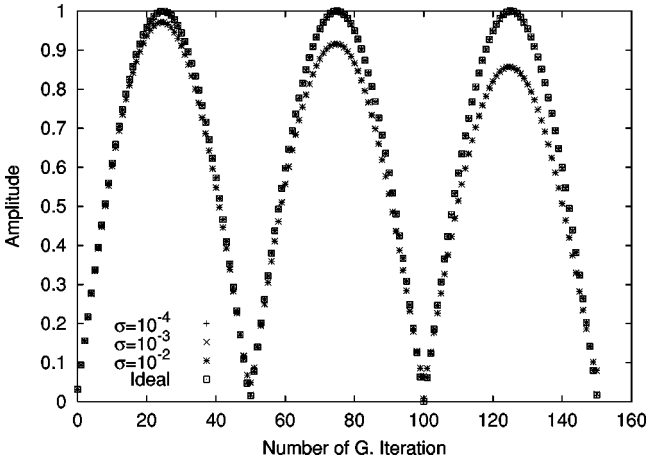

FIG. 15. Decrease of the amplitude of the correct element in the presence of operational errors (10 qubits).

gate $\boldsymbol{R_d}$. Figure 11 shows the amplitude amplification phase by the QFT circuit on the depolarizing channel in Shor's factorization algorithm [step 3 (d)], when $n=187$ and $x=23$. The y axis in Fig. 11 shows the amplitude. The experiment was executed 1000 times and we use the average. If the error probability is greater than 10^{-3}, it is hard to use QFT circuit for period estimation.

Operational errors. In the simulator, ''inaccuracies'' are implemented by adding small deviations to angles of rotations of $\boldsymbol{R_d}$. We consider $H=\boldsymbol{U_R}(\pi/4)\boldsymbol{U_{P1}}(\pi)$, and $G_{\mathrm{NOT}}=\boldsymbol{U_R}(\pi/2)\boldsymbol{U_{P1}}(\pi)$. The simulator also represents inaccuracies by adding small deviations to these angles of rotations. The error is drawn from the Gaussian distribution with the standard deviation (σ). As mentioned above, the experiment was executed 1000 times and we use the average value. Figure 12 shows the amplitude amplification phase by QFT in Shor's factorization algorithm [step 3(d)], when $n=187$ and $x=23$. It seems that the period extraction by using QFT is not affected by the operational error.

Operational and decoherence errors. We investigate also the combined effect of operational and decoherence errors. Table VII shows the results. Each element of the table represents the *fidelity*. The fidelity is defined as the inner product of the correct state and the simulated state with errors.

The combined effect of two factors may be worse than each factor alone. That is to say, the effect seems to be the product of each factor. However, when the decoherence rate is relatively higher, the small-deviation operational error can improve the results contrary to our expectations.

C. Grover's search algorithm

Suppose that a function $f_k:\{0,1\}^n\rightarrow\{0,1\}$ is an oracle function such that $f_k(x)=\delta_{xk}$. The G iteration is defined as $-H_nV_{f_0}H_nV_{f_k}$. The sign-changing operator V_f is implemented by using the *f*-controlled NOT gate and one ancillary bit. Figure 13 shows the circuit of Grover's algorithm.

Effect of errors

We have analyzed the impacts of decoherence and operational errors in the circuit for Grover's algorithm. We assume again that the depolarizing channel is used. We consider $H=\boldsymbol{U_R}(\pi/4)\boldsymbol{U_{P1}}(\pi)$, and $G_{\mathrm{NOT}}=\boldsymbol{U_R}(\pi/2)\boldsymbol{U_{P1}}(\pi)$. The simulator also represents inaccuracies by adding small deviations to these angles of rotations. Each error angle is drawn again from the Gaussian distribution with the standard deviation σ.

Figures 14 and 15 show the impacts of errors for a 10-qubit register. The experiments were repeated 1000 times and we use the average values. If there are no errors, by plotting the amplitude of the correct element (that is, k) we get a sine curve. However, the amplitudes are decreased as G iterations are repeated in the presence of errors. Figure 14 shows the impacts of decoherence error. We can see that the decoherence error affects the period of the sine curve. Figure 15 shows the impacts of operational errors. It seems that the operational error does not affect the period of the sine curve.

VI. RELATED WORKS

There are many quantum simulators for quantum circuit model of computation [10–13]. QDD [11] aims to use binary decision diagram in order to represent the states of quantum register. QCL [12] and OPENQUBIT [13] both use complex number representation of quantum states like our simulator. In addition, QCL tries to establish a high-level, architecture-independent programming language. Obenland's simulator [10,14] is based on an actual physical experimental realization and it uses parallel processing like our simulator. Although it runs on the distributed-memory multicomputers, our simulator runs on the shared-memory multicomputers. Therefore, in our simulator, there is no need to distribute and collect states of the quantum register. In addition, our simulator uses more efficient evolution algorithms and adopts (classical) FFT algorithms for fast simulation of the large-size problems. Our simulator does not depend on any actual physical experimental realizations. It is not easy to say which realizations are best at the moment. In other words, our simulator is more general purpose.

Berman *et al.* simulated Shor's factorization algorithm (four-qubit case) using the Ising-spin quantum computer. The nonresonant effects are analyzed in detail [15]. They also presented the results of simulations of controlled-NOT gate between remote qubits, and the creation of long-distance entanglement in a one-dimensional nuclear-spin quantum computer with many qubits (up to 1000) [16]. However, this simulation did not take into account of all the parameters. In Ref. [17], they developed a consistent dynamical perturbation theory that takes into account of all the parameters and numerical simulations are used to prove their theory.

Long *et al.* investigated the effects of gate imperfections (operational errors) in Grover's search and Shor's factorization by performing numerical simulations [18,19]. But they do not consider decoherence errors. Our simulator deals not only with operational errors but also with decoherence errors.

VII. CONCLUSION

We have developed a parallel simulator for quantum computing on a parallel computer (Sun, Enterprise4500). Up to *30 qubits* can be handled. We have performed Shor's factorization and Grover's database search by using the simulator. Our results show that the improved Shor's factorization algorithm is really effective. We analyzed robustness of the corresponding quantum circuits in the presence of decoherence and operational errors. If the decoherence rate is greater than 10^{-3}, it seems to be hard to use both quantum algorithms in practice.

For future work, we will investigate the correlation between decoherence and operational errors. That is, why small-deviation operational errors can improve the results when the decoherence rate is relatively higher. Furthermore, we will investigate how effective quantum error-correcting codes are.

[1] D. Deutsch, Proc. R. Soc. London, Ser. A **400**, 97 (1985).
[2] D. Deutsch, Proc. R. Soc. London, Ser. A **425**, 73 (1989).
[3] P.W. Shor, in *Proceedings of IEEE Symposium on Foundations of Computer Science, November 20–22*, New Mexico, edited by Shafi Goldwasser (IEEE, New York, 1994), pp. 124–134.
[4] L.K. Grover, in *Proceedings of ACM Symposium on Theory of Computing, May 22–27, Pennsylvania* (ACM, New York, 1996), pp. 212–219.
[5] R. Cleve and J. Watrous, in *Proceedings of IEEE Symposium on Foundations of Computer Science, November 12–14, California* (IEEE, New York, 2000), pp. 526–536.
[6] A. Barenco, A. Ekert, K. Suominen, and P. Torma, Phys. Rev. A **54**, 139 (1996).
[7] V. Vedral, A. Barenco, and A.K. Ekert, Phys. Rev. A **54**, 147 (1996).
[8] C. Miquel, J.P. Paz, and R. Perazzo, e-print quant-ph/9601021.
[9] P.W. Shor, SIAM J. Comput. **26**, 1484 (1997).
[10] K. Obenland and A. Despain, e-print quant-ph/9804039.
[11] QDD ver.0.2, http://home.plutonium.net/~ dagreve/qdd.html
[12] B. Ömer, Master's thesis, Institute of Information Systems Technical University of Vienna, 2000.
[13] OpenQubit 0.2.0, http://www.ennui.net/~quantum
[14] K.M. Obenland, Ph. D. thesis, University of Southern California, 1998.
[15] G.P. Berman, G.D. Doolen, G.V. López, and V.I. Tsifrinovich, Phys. Rev. A **61**, 042307 (2000).
[16] G.P. Berman, G.D. Doolen, G.V. López, and V.I. Tsifrinovich, Phys. Rev. A **61**, 062305 (2000).
[17] G.P. Berman, G.D. Doolen, D.I. Kamenev, and V.I. Tsifrinovich, Phys. Rev. A **65**, 012321 (2002).
[18] G.L. Long, Y.S. Li, W.L. Zhang, and C.C. Tu, Phys. Rev. A **61**, 042305 (2000).
[19] H. Guo, G.L. Long, and Y.S. Li, Chin. Chem. Soc. **48**, 449 (2001).
[20] There is a new quantum circuit that computes QFT (modulo 2^n) that has the size $O(n(\log n)^2 \ln\ln n)$ [5].
[21] Of course, we must classically compute the numbers $x^{2^i} (\mathrm{mod}\, N)$.