システムＬＳＩ設計技術　108－22
(2003. 1. 29)

FPGAを用いた高速量子計算エミュレータ

小野内雅文† 斉藤 康祐† 藤島 実† 鳳 紘一郎†,††

† 東京大学大学院新領域創成科学研究科基盤情報学専攻 〒277-8561 千葉県柏市柏の葉5-1-5 基盤棟7A8
†† 戦略的基礎研究推進事業
E-mail: †info@sf.t.u-tokyo.ac.jp

あらまし 問題の規模に応じて指数関数的に処理時間を要した整数問題（NP問題）を、量子力学の重ねあわせを利用した量子コンピュータは高速に解けるため、近年注目を集めている。しかし、量子現象を直接用いる量子コンピュータでは、ハードウェアを用いる周囲の環境の影響を受け、実用的な規模のアーキテクチャを構築することが困難である。そこで、量子コンピュータと同等の演算能力を持ち、実用的なアルゴリズムを実行可能なアーキテクチャを実現すべく、FPGA内部で大規模な並列演算を行う量子計算エミュレータを試作した。その結果、NP問題である充足可能性問題(SAT問題)を解くために必要な時間は、従来のコンピュータと比較し数百分の1となった。これにより、集積回路の並列性を利用し、量子コンピュータの計算能力に匹敵するプロセッサの可能性を示すことができた。
キーワード FPGA, 専用プロセッサ, 量子コンピュータ, 充足可能問題

Masafumi ONOUCHI†, Kosuke SAITO†, Minoru FUJISHIMA†, and Koichiro HOH†,††

† Department of Frontier Informatics, School of Frontier Sciences, The University of Tokyo 5-1-5-7A8 Kashiwanoha, Kashiwa, Chiba, 277-8561, JAPAN
†† CREST
E-mail: †info@sf.t.u-tokyo.ac.jp

Abstract Recently, quantum computers have attracted attention because it quickly solves the integer problem, which requires exponential processing time according to the problem scale (NP problem), utilizing quantum superposition. However, it is difficult to build the quantum computer with a practical scale directly using the quantum phenomenon since it is influenced of the circumference. Thus we have fabricated the quantum-computing emulator which performs large-scale parallel operation on an FPGA with computing capability equivalent to a quantum computer in order to solve practical quantum algorithms. The required time to solve the satisfiability problem (SAT problem), which is one of NP problems, is reduced down to 1/200 compared with the conventional computer. As a result, the possibility of the processor equal to the calculation capability of a quantum computer was shown using the parallelism of an integrated circuit.
Key words FPGA, Dedicated Processor, Quantum Computing and Satisfiability Problem

1. はじめに

現在の汎用プロセッサは、問題の規模に応じ処理時間が指数的に増大するため、多くの中から特定の解を見つけ出す検索問題が苦手である。たとえば、ある数を素数の積に分解する素因数分解が挙げられるが、解くことの困難さがRSA暗号の根拠にもなっている。

これに対して量子力学の重ね合わせを用いた量子コンピュータの有効性が示されている。量子コンピュータで演算の単位として用いられる量子ビットは、0と1の状態がある確率で同時に存在する。1量子ビットの場合は

$$|\psi\rangle = c_0|0\rangle + c_1|1\rangle \quad (1)$$

と表される。ここで$0\rangle$または$|1\rangle$を基底と呼び、その係数の絶対値の2乗がその値になる確率に相当する。この量子ビットを並べることで、表現できる基底の数は指数的に増大する。n量子ビットの例では、2^n状態が一度に表現され、それぞれの係数に応じていずれかの値になる。このように量子ビットの値を決める操作を観測操作という。

量子コンピュータを用いたアルゴリズムとしては既に2つが考案されており、1994年にP. W. Shorが発見した素因数分解を多項式時間で行うアルゴリズム[1]と、1996年にL. K.

図 1　3 量子ビットの例。$|000\rangle$ から $|111\rangle$ まで 8 つの状態が一度に表現され、観測という操作によりいずれかの値になる。そのときの確率は係数の絶対値の 2 乗に比例する。

Grover が発見したランダムなデータベースにおけるサーチのアルゴリズム [2] がある。これらのアルゴリズムの発見により、物理系を用いて量子コンピュータを実現する研究が盛んに行われ、2001 年には 7 量子ビットの量子コンピュータが実現された [3]。しかし、物理現象を直接用いる量子コンピュータでは、周囲の環境から影響を受けるために計算規模の拡大が非常に困難である。この現象はデコヒーレンスと呼ばれ、計算規模が大きくなる程その影響は大きくなることが知られている [4]。

我々は多数のデバイスを並列に動作させることで、量子コンピュータと同等の演算能力を持ち、汎用プロセッサが苦手な問題を高速に解く専用プロセッサを試作した [5]〜[8]。また、量子コンピュータにおいて重要なデコヒーレンスの影響を見積もることが出来る。本論文では以上 2 点に関して報告する。

2. 量子演算と量子アルゴリズム

量子コンピュータ上で実行される量子アルゴリズムは幾つかの量子操作からなる。ここではその流れを 3 段階に分けて説明する。

2.1 初 期 化

最初、全ての量子ビットは 0 になっているとする。つまり、基底 $|000\cdots0\rangle$ の確率のみが 1 で他は全て 0 という状態を考える。

量子ビットに対して物理的な操作を加えると、特定の基底同士をペアにして確率値の遷移が一斉に起こる。このとき、物理的な操作を量子操作と呼び、ペアとなる基底を指定する役割を果たすビットを標的ビットと呼ぶ。3 量子ビット系で標的ビットを 2 ビット目にした場合は 2 ビット目がそれぞれ異なる基底をペアにして 4 並列で確率値の行列演算が行われる。

$$\begin{pmatrix} c'_{000} \\ c'_{010} \end{pmatrix} = U \begin{pmatrix} c_{000} \\ c_{010} \end{pmatrix} \quad \begin{pmatrix} c'_{001} \\ c'_{011} \end{pmatrix} = U \begin{pmatrix} c_{001} \\ c_{011} \end{pmatrix}$$
$$\begin{pmatrix} c'_{100} \\ c'_{110} \end{pmatrix} = U \begin{pmatrix} c_{100} \\ c_{110} \end{pmatrix} \quad \begin{pmatrix} c'_{101} \\ c'_{111} \end{pmatrix} = U \begin{pmatrix} c_{101} \\ c_{111} \end{pmatrix} \tag{2}$$

一般に n 量子ビットの系において量子操作を行うと、上式の積和演算が 2^{n-1} 並列で行われる。この超並列性こそが量子計算の高速性を生み出しているといえる。

第 1 段階において用いる WH 操作は確率を等分配したり再び集めたりする操作で、

$$U_{WH} : \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3}$$

と表される。WH 操作を m 回かければ、2^m 個の基底に確率を $1/\sqrt{2^m}$ ずつ等分配できる。

2.2 確率値の交換

第 2 段階では等分配した確率値を基底間で交換する。この操作は一般の計算機における加算や減算にあたり、NOT 操作によって実現される。NOT 操作における行列は

$$U_{NOT} : \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{4}$$

と表される。

しかし、NOT 操作だけでは加算や減算が構成できないので、演算の及ぶ範囲を限定できる制御演算が必要となる。制御演算においては、制御ビットで指定されたビットが 1 の基底間のみで確率値の行列演算が起こる。3 量子ビットの例において、標的ビットを 2 ビット目とし、制御ビットを 1 ビット目とすれば

$$\begin{pmatrix} c'_{100} \\ c'_{110} \end{pmatrix} = U \begin{pmatrix} c_{100} \\ c_{110} \end{pmatrix} \quad \begin{pmatrix} c'_{101} \\ c'_{111} \end{pmatrix} = U \begin{pmatrix} c_{101} \\ c_{111} \end{pmatrix} \tag{5}$$

なる確率値の行列演算だけが起こる。ここで、ペアとなる基底のうち標的ビットが 0 である基底を 0 側の基底、標的ビットが 1 である基底を 1 側の基底と呼ぶことにする。

第 2 段階では NOT 操作あるいは制御 NOT 操作しか行われないので、確率の移動が終わった後も確率の値は 0 か $1/\sqrt{2^m}$ のいずれかである。

2.3 解 の 抽 出

確率の移動が行われた状態で観測を行っても、確率を持っている基底がランダムに観測されてしまうので、第 3 段階として、所望の解が得られるように特定の基底に確率を集中させる必要がある。具体的な方法としては、量子フーリエ変換を用いて周期を抽出する方法 [1] と、特定の条件を満たす基底に確率を集中させる方法 [2] とがある。

これらの操作は NOT 操作や WH 操作以外を用いて行われ、操作後の確率値は複素数になる。

2.4 量 子 回 路

量子操作の手順を図で表現したものが量子回路である。図 2 で示すように、量子回路の横線が各量子ビットに対応し、各量子操作は横線の上に置かれたブロックで示される。また、制御ビットは横線上の黒丸で示され、制御をかける量子操作と縦線で結ばれる。量子回路全体は音楽の五線譜のように左から右へと進行し、左からの入力が様々な量子操作を受けて右から出力されるイメージとなる。

3. ハードウェアによる実現

3.1 基底の表現

従来のハードウェアではレジスタに 0 と 1 を同時に保持出来ないので、基底の数だけメモリを用意し各基底の確率値を保持する。n 量子ビットに対して 2^n 個の基底が存在するが、メモリのアドレスを基底と対応させる。ここで、確率値の精度を p ビットとすると必要なメモリは $2^n \times p$ ビットとなる。しかし、この方法でハードウェアを実装するには 2 つの問題が生じる。1 つ目は必要なメモリの容量が多くなること。ハードウェアに

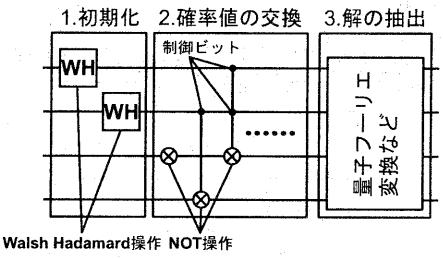

図 2　量子アルゴリズムの量子回路。量子ビットは横線、量子操作はブロック、制御ビットは黒丸で表現される。回路全体は左から右へと進行する。

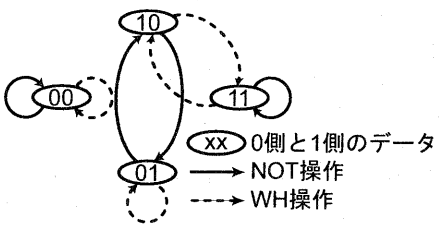

図 3　状態遷移図。NOT 操作と WH 操作を行うと、ペアとなる基底間のデータが図のように遷移する。

は限りがあるので、確率値 1 つに割くビット数を減らせばより多くの確率値を保持出来、大規模な問題に対応出来る。2 つ目は確率値 1 つに割くビット数を増やすと積和演算を行う回路が大きくなり、並列数が小さくなってしまうことである。そこで、演算回路を多数設けて並列動作をさせるために、個々の演算回路を出来る限り小さくする必要がある。以上のことから、確率値 1 つに割くビット数を減らすことは、演算規模と演算速度の両方においてメリットがあるといえる。

量子アルゴリズムのエミュレートを考えた場合、2.2 節までは係数を別にすれば 0 と 1 しか出てこないので、確率は 1 ビットで表現可能である。本エミュレータでは確率値の 0 をデータの 0 に、確率値の $1/\sqrt{2^m}$ をデータの 1 に対応させている。ただし、確率の精度が 1 ビットでは 2.3 節で述べた解の抽出が行えないので、その代替となる操作を考える必要がある。この操作に関しては 3.3 節で述べる。

3.2　量子操作の実現

確率の精度を 1 ビットで考えた場合の、各量子操作について説明する。WH 操作はペアとなる基底同士で確率を分配したり、再び集めたりする操作であった。そこで、ペアとなるアドレスのデータの 0 側だけが 1 であれば両方のデータに 1 を書き込み、両方が 1 であれば 0 側の基底に 0 を書き込む。また、データが両方とも 0 であるか、1 側のみ 1 であれば何のデータ操作も行わない。また、NOT 操作はペアとなる基底間で確率の交換を行う操作であるから、ペアとなるアドレス間のデータをスワップして書き戻せばよい。以上のようなデータ操作を行う演算回路を LU(Logic Unit) と呼ぶことにする。

3.3　解の抽出

量子演算においては確率の値が 0 でない基底を、その周期を見るなどして間接的に得ているが、本エミュレータではデータが 1 のメモリのアドレスを検索することで周期を用いずに解を直接得ることが出来る。本エミュレータではこのアドレス検索をもって観測操作に代えるものとする。ハードウェアの動作としては、特定のアドレスを持つメモリ間でデータの論理和を取ることで、バイナリサーチを行う。

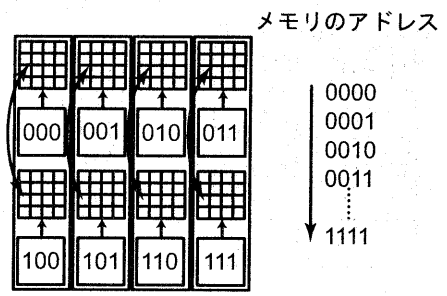

図 4　データ操作。PE の並列数を 2^3、各 PE に付随するメモリを 2^4 ビットとし、全体で 7 量子ビットの系を構成している。1 ビット目が標的ビットの場合は、最上位ビットが異なる PE 同士がデータを通信し演算を行う。このときメモリのアドレスは $000\cdots0$ から 1 つずつ増えていく。

3.4　量子操作の並列化

量子コンピュータと同等の並列性を実現しようとすると、n 量子ビットに対して 2^n ビットのメモリと 2^n 個の LU が必要となる。この場合、LU がデータの操作を全て並列に行うことはできるが、限られたハードウェアでは配置できる LU の個数も限られてしまうので、扱える問題の規模が限られてしまう。メモリ 1 ビットあたりの面積は LU1 個の面積よりはるかに小さいため、本エミュレータでは 1 つの LU に対し複数ビットのメモリを持たせる構造にした。データをシリアルに処理する LU をパラレルに動作させることで、大規模な問題を高速に処理することが可能となる。プロセッサのメモリの容量を 2^n ビット、LU の並列数を 2^m とすると、1 つの LU は 2^{n-m} ビットのメモリのデータ操作を担当することになる。ここで、2^{n-m} ビットのメモリを持った LU を 1 つの単位として、PE(Processing Element) と呼ぶことにする。つまり、メモリにおける n ビットのアドレス空間のうち上位 m ビットを PE のアドレスに対応させ、下位 $n-m$ ビットを PE に付随するメモリのアドレスに対応させる。

このハードウェアは標的ビットの位置により動作が異なるので、以下で順に説明する。標的ビットが上位 m ビットになった場合、ペアとなるメモリのデータはアドレスが 1 ビットずつ異なる PE にある。従って、ペアとなる PE 同士がお互いにデータをシリアルに通信し合い演算を行う。このとき演算の対象となるメモリのアドレスは全ての PE において同じであり、$000\cdots0$ から 1 つずつ増えている。

標的ビットが下位 $n-m$ ビットになった場合、ペアとなるメモリのデータは同じ PE 内にある。このときもメモリのアドレスは全ての PE で同じになるが、ペアとなるアドレス同士をシリアルに呼び出し演算を行うので、メモリに割り当てられた

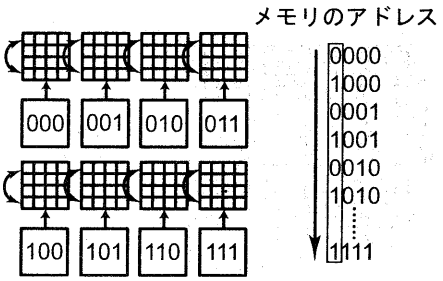

図 5　データ操作。図 4 と同じ 7 量子ビットの系で 4 ビット目が標的ビットの場合は、メモリのアドレスにおいて 1 ビット目が異なるアドレス同士で演算を行う。メモリのアドレスは 1 ビット目がカウンタの最下位ビットになり、ペアとなるアドレスが交互に呼び出される。

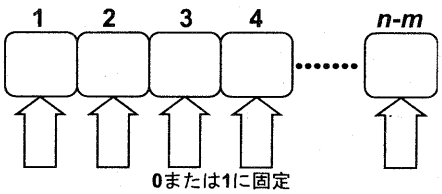

図 6　アドレス生成ユニット。任意の標的ビットと制御ビットに対応するため、各ビットにはそのビットをカウンタの最下位ビットに指定する機能と、0 または 1 に固定する機能がある。

$n-m$ ビットのアドレスのうち、任意のビットをカウンタの最下位ビットにするアドレス生成が必要となる。

以上 2 つの操作に加えて制御演算を行う場合、制御ビットが上位 m ビットにあるときはそのビットが 1 となるアドレスを持つ PE のみを動作させる。そして、制御ビットが下位 $n-m$ ビットにあるときは、全ての PE がそのビットが 1 となるアドレスを持つメモリのデータに対してのみ演算を行う。また、制御ビットが上位 m ビットと下位 $n-m$ ビットの両方に存在するときは、両方の動作が複合して行われる。

これらの動作を実現するため、特定のビットを固定する機能と、任意のビットをカウンタの最下位ビットに指定できる $n-m$ ビットのカウンタをアドレス生成に用いた。

また、命令セットは下の表のようになっている、

表 1　命令セット

命令語	操作の内容
init	アドレスが 0 のメモリのみに 1 を書き込む。
wh	指定したアドレスのペアでデータの分配と収集を行う。
not	指定したアドレスのペアでデータの交換を行う。
inq0	指定したアドレス領域の中で特定のビットが 0 のアドレス間でデータの論理和を取る。
inq1	指定したアドレス領域の中で特定のビットが 1 のアドレス間でデータの論理和を取る。
set0	指定したアドレスのペアのどちらかに 1 があれば 0 側に移す。
set1	指定したアドレスのペアのどちらかに 1 があれば 1 側に移す。

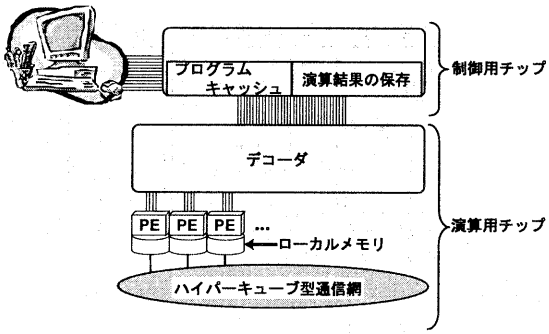

図 7　システム全体図。外部から制御を行う PC、プログラムと演算結果を保持する制御チップ、演算を行う演算チップから構成される。

3.5　システム構成

試作したシステムでは、機能を制御用チップと演算用チップの 2 枚に分け、システムを制御するための PC を加えた 3 つの部分から構成されている。PC は制御用チップと繋がっており、量子回路を記述したプログラムのダウンロードや演算結果の表示を行う。制御用チップは PC とのインターフェースの役割を果たし、PC から送られたプログラムを演算用チップへ送り出したり、演算終了時には演算結果を PC へと引き渡したりする。演算用チップは送られてくるプログラムを元に実際の量子計算を行う。

3.6　プロセッサアーキテクチャ

本試作で用いた FPGA は回路素子として使える領域とメモリとして使える領域の割合が決まっているので、64k ビットのメモリに対し PE を 1024 並列で配置する構造にした。PE は、データの操作や検索を行うための LU（Logic Unit）とパイプライン化のためのレジスタから構成され、64bit のメモリが付随している。

3.7　FPGA への実装結果

本試作では、Verilog-HDL でハードウェアを記述し Synplicity 社の Amplify で論理合成を行い Altera 社の QuartusII で配置配線を行った。

表 2　プロセッサの実装結果

動作周波数		40MHz
1 命令当たりに要するクロック数		9〜72 クロック
Chip0	ゲート数	6 万ゲート
	メモリ	192kbit
	命令数	4096 命令
Chip1	ゲート数	110 万ゲート
	メモリ	64kbit
	並列数	1024

各 PE が操作する基底のデータ数に応じて、演算に要するクロック数は変化する。

4.　高速量子計算エミュレータの応用

実装したエミュレータの効果を実証するため、充足可能性問題と量子エラーシミュレーションの 2 つに対して適用した。

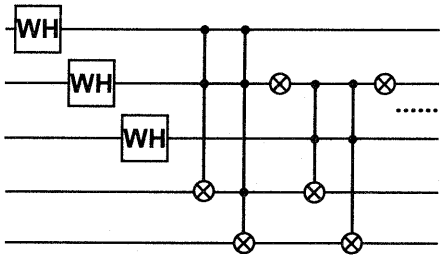

図 8　カウンタを用いた量子回路。上の 3 量子ビットは変数値の組み合わせ、下の 2 量子ビットはカウンタに相当する。全変数の組み合わせの中から各節を満たさない状態のカウンタを 1 つずつ増やし、最後にカウンタが 0 の基底を検索することで解を得る。

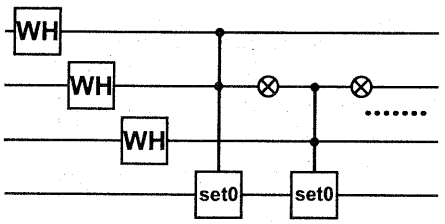

図 9　非量子操作を用いた量子回路。上の 3 量子ビットは変数値の組み合わせを表現する。制御演算と set0 命令により、各節を満たさない状態を最下位の量子ビットが 1 の領域から 0 の領域へと移動する。最後に最下位ビットが 1 の領域のデータに 1 があるかを検索する。

4.1　充足可能性問題

充足可能性問題とは積和標準形論理式の値を 1 にする 2 値変数の組み合わせがあるかどうかを判定する問題である [9]。

例として 2 値変数を x_i とし、真を 1、偽を 0 に対応させ和積標準形を

$$(x_1 + x_2) \cdot (\overline{x_2} + x_3) \cdot (\overline{x_3} + x_1) \cdot (\overline{x_2} + \overline{x_3}) = 1 \qquad (6)$$

とする。このとき論理式を満たす変数 x_1, x_2, x_3 の組は

$$(x_1, x_2, x_3) = (1,0,0) \text{ または } (1,0,1) \qquad (7)$$

の 2 組である。変数に対し論理和を取った、$(x_1 + x_2)$ などを節と呼ぶ。この充足可能性問題に対する量子回路は図 8 のようになる。

この量子回路では上側の量子ビットが変数値の全組み合わせに対応し、それぞれの組が節を満たさない場合は下側の量子ビットで構成するカウントを 1 つずつ増やす仕組みになっている。最後にカウンタが 0 のままになっている基底を検索することで、充足可能性問題の解を得ることが出来る。

しかし、カウンタを用いた量子回路では量子ビットの多くをカウンタに割かねばならず、大規模な問題を解くことが出来ない。そこで、カウンタを用いずに非量子操作で充足可能性問題を解く手法を考案した。この手法では論理式全体の否定を取った同値な式に対し非量子操作を用いて解く。

$$(\overline{x_1} \cdot \overline{x_2}) + (x_2 \cdot \overline{x_3}) + (x_3 \cdot \overline{x_1}) + (x_2 \cdot x_3) = 0 \qquad (8)$$

まず、メモリ全体を半分の領域に分け、半分を変数値の組み合わせが論理式を満たしている領域、残りの半分を変数値の組み合わせが論理式を満たしていない場合に対応させる。これは量子計算で言うと、ある量子ビットが 0 の状態と 1 の状態にそれぞれの領域を割り当てることに対応する。ここでは、最下位の量子ビットが 0 のとき論理式を満たしていないとし、1 のとき論理式を満たしているとする。また、それ以外の量子ビットに対応する基底を変数値の組み合わせに対応させる。まず、変数値の全ての組み合わせに論理式を満たす可能性があるので、最下位の量子ビットが 1 の領域全てに 1 を書き込む。そして、各節毎にその節を満たさない変数値の組み合わせに対応する基底のデータを最下位の量子ビットが 1 の領域から 0 の領域へと移動する。この不可逆な操作は set0 命令により可能である。全ての節に対してこの操作を行った後、最下位の量子ビットが 1 の領域でデータが 1 の基底を検索すれば解を得ることが出来る。

4.2　ビットエラーシミュレータ

実際の物理系によって構成される量子コンピュータでは、周囲の環境から受ける熱雑音などにより、意図しない量子操作が行われてしまう可能性がある。この現象は量子エラーと呼ばれ、それにより量子情報が失われることをデコヒーレンスと呼ぶ。

デコヒーレンスの原因としては 3 つ知られており、それぞれビットエラー・位相エラー・ビット&位相エラーと呼ばれる。実際の量子系で起こる量子エラーの影響を見積もるため、本プロセッサを用いてモンテ・カルロシミュレーションを行った。量子エラーとしてはビットエラーを NOT 操作により実現している。

4.3　エミュレータの応用結果

プロセッサとソフトウェアで同じ充足可能性問題を同じ手法で解かせ速度を比較した。充足可能性問題としては 15 変数 182 節の問題を解かせている。ソフトウェアのプログラミングは Visual C++で行い、ハードウェアと同様の操作を経て解を得ている。結果を表 3 に示す。

表 3　プロセッサの実装結果

	PC シミュレーション	専用プロセッサ
システム構成	Intel Xeon 2.2GHz	2^{16} 状態 1024 並列 40MHz
1 問当たりの時間	41.7msec	0.152msec

表 3 より、ソフトウェアよりもハードウェアの方が 275 倍高速であることが分かる。

また、5 変数 10 節の充足可能性問題に一定の確率で NOT 操作を挿入したモンテカルロシミュレーションの結果を図 10 に示す。

5.　結　　論

従来の計算機が苦手な問題に対して有効性が示された量子コ

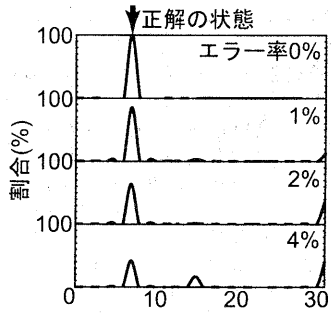

図 10　デコヒーレンスによる正解率の減衰。充足可能性問題において、NOT 操作の挿入率が高くなると正解を得る確率が減衰する。

ンピュータを、高速にエミュレートするアーキテクチャを提案した。提案したハードウェアを用いて、汎用性のある充足可能性問題を高速に解くアルゴリズムを示した。処理速度は汎用プロセッサである Pentium4 2.2GHz と比較して 275 倍となることを実測により示した。本プロセッサで扱える問題の規模は小さいものの、ソフトウェアを組み合わせることで大規模な問題に対応できるようになれば、専用プロセッサの利点を活かすことがことができるであろう。

文　献

[1] P. W. Shor "Algorithms for Quantum Computation :Discrete Log and Factoring," *Proc. of the 35th Annual IEEE Symposium on Foundations of Computer Sience*, pp.124-134, 1994.

[2] L. K. Grover "A Fast Quantum Mechanical Alogorithm for Database Search," *Proc. of the 28th ACM Symposium on Theory of Computing*, pp.212-219, 1996.

[3] Lieven M. K. Vandersypen, Matthias Steffen, Gregory Breyta, Costanino S. Yannoni, Mark H. Sherwood and Isaac L. Chung "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance" Nature pp.883-887 20/27 December 2001.

[4] D. P. DiVincenzo and Peter W. Shor "Fault-Tolerant Error Correction with Efficient Quantum Codes," Phys. Rev. Lett., 77 pp.3260-3263 1996.

[5] 鈴木 康文, 斉藤 康祐, 藤島 実, 鳳 紘一郎 "16-Qubit ディジタル量子回路プロセッサ," 2002 応物全大, 分冊 1, pp.213, Mar. 2002.

[6] K. Saito, Y. Suzuki, M. Fujishima and K. Hoh "High-Speed Emulation of the Quantum Computing Based on Logic Operations," SSDM, pp376-377, Nagoya, Japan, Sept. 2002.

[7] 斉藤 康祐, 鈴木 康文, 藤島 実, 鳳 紘一郎 "16-Qubit 論理量子プロセッサ," 応物, 分冊 1 pp.167, Sept. 2002.

[8] 小野内 雅文, 藤島 実, 鳳 紘一郎 "専用プロセッサを用いた量子コンピュータのエラーシミュレーション," 応物, 分冊 1, pp.168, Sept. 2002.

[9] S. Cook, "The complexity of theorem proving procedures," *Proc. 3rd Annual ACM Symposium on the Theory of Computation*, pp.151-158, 1971.