量子計算回路の設計と計算シミュレーターの開発

The Development of Quantum Circuit Designing and Simulating Environment

渡辺宙志　鈴木将　山崎淳之介
Hiroshi WATANABE　Masaru SUZUKI　Junnosuke YAMAZAKI

1) 東京大学工学系研究科物理工学専攻 (〒113-8656 東京都文京区本郷 7-3-1 E-mail: kaityo@acolyte.t.u-tokyo.ac.jp)

ABSTRACT. We develpped a total environment of Quantum Computing Simulation. It supports designing, simulating and analyzing quantum circuits. The system contains a full GUI circuit designer and a multi-platform simulating system. It allows the researcher to study the quamtum computing more easily and efficiently.

1　背景

一般的に、開発に専門知識が必要とされるソフトウェアは購入すると高価であることが多い (物理分野における解析では数十万円〜数百万円程度)。したがって購入すると研究費を圧迫するか、研究者自身がプログラムの開発に多くの時間を割くことになる。このような状態を防ぐために、数多くのライブラリが開発されているが、ＧＵＩで手軽に試すことができるソフトウェアは数少ない。

量子計算機の設計ソフトウェア、シミュレーターも例外ではなく、CUI のライブラリは散見されるが、GUI で手軽に試すことのできるソフトウェアは、特にフリーウェアとしては見受けられない。本プロジェクトではでは、このような現状を改善するため、GUI で手軽に設計できる量子計算回路デザイナーと、設計した回路のシミュレーターを開発し、フリーウェアとして公開することを目的とする。

2　目的

本プロジェクトは、量子計算機研究を総合的にサポートする環境の作成を目的とする。

具体的には、GUI を用いた簡単かつ直感的な操作により、量子計算回路の設計、シミュレーション実行、実行結果の解析までをＰＣ上で行えるソフトウェア QCAD を開発した。QCAD は、単に計算を行うのみでなく、ステップ実行やデバッグなどの機能を有しており、難しい概念を含む量子計算を理解するのに有効である。QCAD で開発した量子計算回路は、小規模なものであれば PC 上でシミュレートできるが、大規模なものは莫大なメモリを必要とするため、ワークステーションや超並列計算機でシミュレートする必要がある。そこで、設計した回路を中間コードで表現し、中間コードをコンパイル、実行することでワークステーションや超並列計算機で実行可能とした。さらに QCAD は回路図の出力機能や結果の解析機能を備えており、量子計算回路設計から結果解析、論文執筆までをトータルにサポートする環境を実現する。図 1 に QCAD の概念図を示す。

3　開発成果

本システムは、機能別にモジュール化した以下のようなソフトウェアで構成される。

（1） qcad(回路設計、データ解析ソフトウェア)

Microsoft Windows 上で動作する GUI ソフトウェアで回路の設計、保存、印刷、出力形式を指定したエキスポートが可能であり、設計した回路はそのままシミュレーションすることができる。他の計算機 (ワークステーションや超並列計算機) で計算を行う場合は回路に対応する中間コードを出力する。その場合も実行結果の解析は qcad 上で行う。

（2） qcrun(中間コードのインタプリタ)

中間コードをインタプリタ実行するソフトウェアで、メモリを多く積んだワークステーションでの動作を想定している。中規模な回路のシミュレーションやステップ実行など、デバッグを目的とする。

（3） qcc(中間コードコンパイラ)

中間コードを C 言語に変換した後にコンパイルし、独立した実行可能形式を出力するコンパイラ。自動並列化機能を持ち、並列計算機用の実行ファイルが出力可能である。並列化には 標準規格の MPI を用いているため、MPI をサポートしている並列計算システムならどこでも並列計算が可能である。

インタプリタとコンパイラは Ansi C++で書いてあるため、make すれば基本的にはどこでも動作する。実際に Windows, Linux, Macintosh での動作を確認した。さらに、国際的な標準ツールを目指し、英語によるドキュメント類を用意し、世界への普及を目指している。

4　量子計算回路

（1） 量子計算機とは

量子計算機とは、量子的な重ね合わせ状態を利用した計算機のことである。膨大な状態空間を用いた高い並列性が特徴で、古典計算機では実現できないような計算ができると期待されている。特に Shor が量子計算機において整数の素因数分解を劇的に早く計算できるアルゴリズムを発表してから [1]、量子アルゴリズムの研究が盛んに行われるようになった [2]。

アルゴリズム研究のみならず、Shor のアルゴリズムを実際に量子計算機で計算した例が報告されたり、数ビット程度ながら室温動作する量子計算機も実現されており [3]、

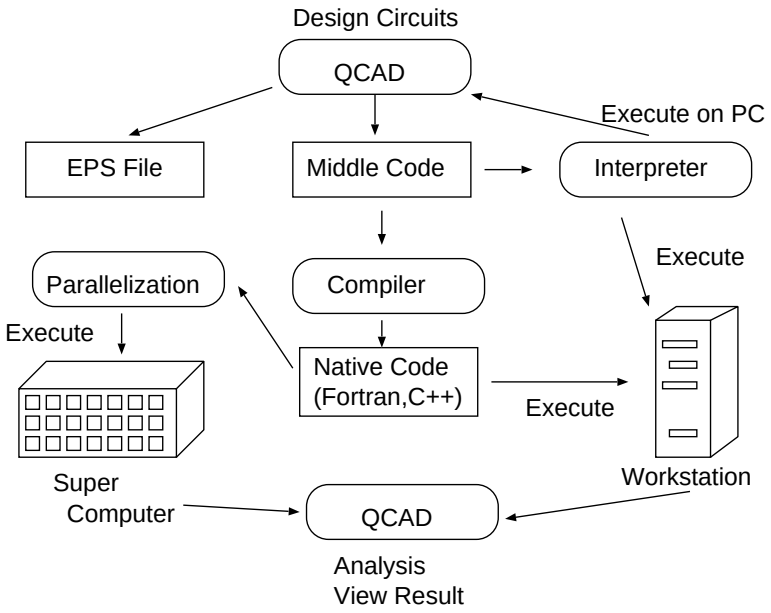


図 1: QCAD の概念図。Windows 上で動作する QCAD で量子回路を設計し、作成した回路は論文用に EPS ファイル、実行用に中間コードで出力される。中間コードはインタプリタで読み込んで実行することができ、結果はそのまま PC で確認できる。インタプリタは独立して動作できるため、ワークステーションなどで、より高速に実行させることもできる。また、中間コードをコンパイルして Fortran や C 言語で出力して実行できる。さらに MPI などを用いた並列化を行い、スーパーコンピューターで実行し、結果を QCAD で解析する。

次世代の計算機としての期待が高まっている。

しかし、数十ビット程度の量子計算機の実装は困難であり、現在大きな回路を実際に計算することはできない。そこで、量子計算機を通常の計算機 (以後古典計算機と呼ぶ) でシミュレートし、実際の回路の動作を調べることで新たな量子アルゴリズムの開発が進められている。

（2） 記法について

量子計算について説明する前に、よく使う記法についてまとめておく。量子計算機におけるビットは、古典計算機と区別して量子ビット、もしくは qubit と呼ばれる。量子ビットはブラケットベクトルであらわし、オフなら $|0\rangle$、オンなら $|1\rangle$ と表記する。量子ビットの集まりは量子レジスタ、もしくは単にレジスタと呼ぶ。複数の量子ビットの状態はそれぞれの直積で表せるため、$|q_n \otimes \cdots \otimes q_1 \otimes q_0\rangle$ と表記するが、単に $|q_n \cdots q_1 q_0\rangle$ と略記する。この時、慣習により、番号が若いビットを右に表記する。

（3） 量子計算機の実際

古典計算機と違い、量子計算機はひとつのゲートが全ビットの情報に影響する。一番簡単なゲート、論理否定 (NOT) ゲートを考えよう。今、量子ビットが 1 ビットだけだとすると、とりうる状態は $|0\rangle$ と $|1\rangle$ の 2 種類だけである。したがって、すべての状態はこの状態の線形結合 $\alpha|0\rangle + \beta|1\rangle$ でかける (α、β は一般に複素数)。この状態に論理否定を演算すると、$|0\rangle$ であった状態は $|1\rangle$ に、$|1\rangle$ であった状態は $|0\rangle$ になるため、結果は $\beta|0\rangle + \alpha|1\rangle$ となる。すなわち 2 つの状態の重みを取り替えれば良い。これを行列形式で書くと、

$$\begin{pmatrix} \alpha' \\ \beta' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

となる。次に、量子ビットが 2 ビットである場合を考えよう。とりうる状態は $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ の 4 つとなる。1 ビット目に論理否定を演算すると、$|00\rangle$ は $|01\rangle$ に、$|10\rangle$ は $|11\rangle$ となる。それぞれの重みを $\alpha, \beta, \gamma, \delta$ として行列形式で書くと、

$$\begin{pmatrix} \alpha' \\ \beta' \\ \gamma' \\ \delta' \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

と書くことができる。古典計算機では回路が実行されるたびに状態が遷移していくのに対し、量子計算機では取りうるすべての状態の重みが遷移していく。一般に、n 量子ビットのゲートは $2^n \times 2^n$ の行列であらわすことができる。したがって、量子計算機の一度の演算は、古典計算機で最大 2^{2n} 個の複素数演算に相当する。

5 計算エンジンの実装

（1） 量子ビットの表現

まず、n ビットの量子計算機には、2^n 個の状態がありうる。それぞれに 0 から $2^n - 1$ まで通し番号をつけることにしよう。各ビットの状態を 0 と 1 で表現すると、たとえば $|010100\rangle$ と状態を決めることができる。この状態の中の数字を右から 2 進数表記だと思って数字に直し、それをこの状態の状態番号と定義する。このように定義しておくと、後で量子計算の演算を実装しやすくなる。

C++言語で実装するには、実数部分と虚数部分それぞれ 2^n 個ずつ、2^{n+1} 個の実数の配列を用意しなければならない。それぞれ `R[i]`、`I[i]` としよう。たとえば、`R[3]` は、状態番号 3 の状態 ($|0\cdots011\rangle$) の重みの実数部分をあらわす。

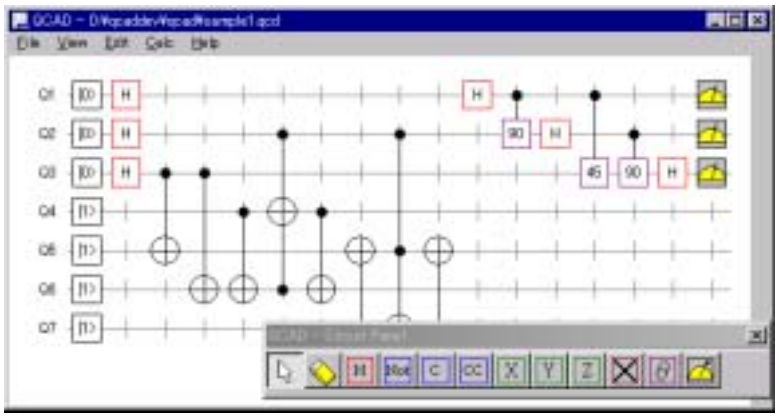


図 2: QCAD のスクリーンショット。Windows 95/98/ME/2000/NT/XP で動作し、マウスで簡単に操作できる。回路をパレットから選んで置きたいところをクリックしていけば回路図を作成することができる。アンドゥなどもできる。データの出力形式は独自形式のほか、EPS、BMP をサポートしている。開発した回路はその場で動作確認することができる。大規模な回路については、それに対応する中間コードを作成し、インタプリタ実行やコンパイルによる実行を可能とする。

実数の表現として double 型を使うと、ひとつの実数に 8Byte 必要なため、全体で、2^{n+4}Byte のメモリが必要となる。たとえば 32 ビットの量子計算機をシミュレートするには、64GB のメモリが必要となる。

（2） 1ビットゲートの例

では、実際に量子回路の計算例を見てみよう。前述したようにすべての量子回路は $2^n \times 2^n$ の行列で表現できる。しかし、その行列は疎行列であるため、そのまま計算すると無駄が多い。そこで、計算に関係する状態のみを取り出して計算することになる。

まず、p ビット目に論理否定を演算することを考えよう。そのためには、既に説明したように p ビット目が 0 である状態と、1 である状態の重みをすべて取り替えればよい。すなわち、$|q_{n-1}\cdots q_{p+1}0q_{p-1}\cdots q_0\rangle$ と $|q_{n-1}\cdots q_{p+1}1q_{p-1}\cdots q_0\rangle$ の実数の重みと虚数の重みを取り替える。ただし、p ビット以外の残りのビット $q_i (i \neq p)$ のとりうる状態すべてにたいして取替えを行わなくてはならない。したがって、2^{n-1} 個の状態について取替えを行う。

まず、`int i = 0 ;i++; (1<< (N-1))` であるようなループを作る。それぞれの i について、p ビット目に 0 を挿入すると、我々の量子ビットの定義から p ビット目が 0 であるような状態、2^{n-1} 個をすべて尽くす事ができる。p ビット目が 1 であるような状態も同様である。したがって、実際のコードは以下のようにかける。

```
//N = Number of qubits
unsigned int state = 1<< (N-1);
for(unsigned int i=0;i<state;i++)
 {
  unsigned int ix0 = insert0(i,p);
  unsigned int ix1 = insert1(i,p);
  //Swap weights of two states
  swap(R[ix0], R[ix1]);
  swap(I[ix0], I[ix1]);
 }
```

ただし、swap は二つの引数の値を入れ替える関数、insert0(i,p) は、i の p ビット目に 0 を挿入する関数であり、実態は次のような関数である。

```
unsigned int
insert0(unsigned int i0,
        unsigned int BitNum)
{
unsigned int msk = (1<<BitNum) -1;
return ((~msk & i0) << 1) | (msk & i0);
}
```

insert1 も同様に定義される。

（2） 2ビットゲートの例

さらに、2 ビットゲートをの例を見てみよう。2 ビットゲートで一番簡単な回路は Controlled Not 回路である。Controlled Not とは p_1 ビット目がオンの時のみ p_2 ビットに対して論理否定を取る回路のことである。つまり p_1 ビットがオンである状態すべてについて、p_2 ビットがオンである状態とオフである状態の重みを取り替えれば良い。

```
//N = Number of qubits
unsigned int state = 1<< (N-2);
int q1,q2;

if(p1<p2)
 {
  q1 = p1;
  q2 = p2;
 }else
 {
  q1 = p2;
  q2 = p1;
 }

for(unsigned int i=0;i<state;i++)
 {
   unsigned int ix0 = insert1(i,q1);
   ix0 = insert0(ix0,q2);
   unsigned int ix1 = insert1(i,q1);
   ix1 = insert1(ix1,q2);
  //Swap weights of two states
  swap(R[ix0], R[ix1]);
  swap(I[ix0], I[ix1]);
 }
```

ビットの挿入により以後のビット列がずれるために挿入

機能名	中間コードの書式例	機能説明
Initialize		
Initialize	INIT(N)	q[0] から q[N-1] までの N 個の量子ビットを生成し、すべて $\|0\rangle$ で初期化する。
Object		
qubit	q[i]	i 番目の量子ビットをあらわす
Operator		
Hadamard	H(q[i])	i 番目のビットを Walsh-Hadamard 変換する。
Not	NOT(q[i])	i 番目のビットの論理否定を取る。
Controlled Not	CNOT(q[i1],q[i2])	$i2$ 番目のビットがオンなら $i1$ 番目のビットの否定を取る。
Toffoli	CCNOT(q[i1],q[i2],q[i3])	$i2$、$i3$ 番のビットが両方ともオンなら $i1$ 番目のビットの否定を取る。
Controlled Phase	CROT(q[i1],q[i2],w)	$i2$ 番のビットがオンなら、$i1$ のビットの位相を w だけまわす。
Others		
Comment	# QCAD MIDCODE Ver 1.0	#記号から行の最後まではすべて無視される。

表 1: 中間コードの仕様。

順序に気を配る必要があるが、Toffoli ゲート (Controlled Controlled Not) 等の 3 ビットゲートも同様にして実装できる[*1]。以上のように、量子計算回路は簡単に実装でき、$2^n \times 2^n$ の行列を作る必要も無い。さらにほとんどの量子ゲートが、演算される状態番号を二つ選んだあとは 2×2 の複素行列で表すことができることに注意したい。これにより、後に述べる自動並列化が容易になる。

6 中間コード

（1） 中間コードとは

中間コードとは、設計された回路を一定の書式に基づいてテキスト形式で出力された、回路の計算方法を示すテキストのことである。設計された回路図と計算エンジンとをつなぐ役割を持つ。コードの独立性、移植性を高めるため、CAD 情報は一度中間コードに落とされる。回路のシミュレーションは、この中間コードを読み込んで、インタプリタ形式かコンパイル形式で実行することになる。このような仕様とすることで、Windows 上で開発した回路をどんなプラットフォームでもシミュレートができるマルチプラットフォーム環境を実現する。

（2） 中間コードの仕様

中間コードは、コメント部分、コード部分、付加情報部分の三つのパートから構成される。

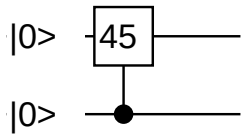
コメント部分 #から始まる行で、これは解析時に無視される。

コード部分 最初にビット数を宣言する。その後はゲートの名前、扱う量子ビットの情報、演算に必要な情報を関数型で宣言する。関数の引数は、操作を受けるビット、制御ビット 1、制御ビット 2、$\cdots$、必要な情報、という形とする。

付加情報部分 デバッグのためのブレークポイントや行番号情報などを含む[*2]。

角度などの定数引数と区別するため、量子ビットは、q[i] という形であらわす。この場合は回路において上から i 番目の量子ビットに対しての演算であることを意味し、実際の波動関数や状態とは関係が無いことに注意してほしい。

たとえば、以下の回路は、次のような中間コードに対応する。




```
CROT(q[0],q[1],45)
```

これは 1 番目の量子ビットがオンなら、0 番目の量子ビットの位相を 45 度まわす、という意味である。

中間コードの全仕様は表 1 の通りである。

（3） 中間コードの例

以下に、図 3 の回路に対応する中間コードを載せる。図 3 の回路は、Nature に掲載された、$15 = 5 \times 3$ の素因数分解を実行する量子回路である [3]。

```
# file name: "D:\qcad\sample1.mcd"
# QCAD MIDCODE Ver 1.0
INIT(7)
NOT(q[6])
H(q[0])
H(q[1])
H(q[2])
CNOT(q[4],q[2])
CNOT(q[5],q[2])
CNOT(q[5],q[3])
CCNOT(q[3],q[1],q[5])
CNOT(q[5],q[3])
CNOT(q[4],q[6])
CCNOT(q[6],q[1],q[4])
CNOT(q[4],q[6])
H(q[0])        #Start Inverse QFT
CROT(q[1],q[0],90)
H(q[1])
CROT(q[2],q[0],45)
CROT(q[2],q[1],90)
H(q[2])        #End of Inverse QFT
```

[*1] ここでは理解の便のためにわかりやすいコードを紹介したが、実際にはもっと早い手法が考案されている。しかし、そのような手法では関係するビットが多くなるとソースが複雑になる。

[*2] 現在仕様は未定

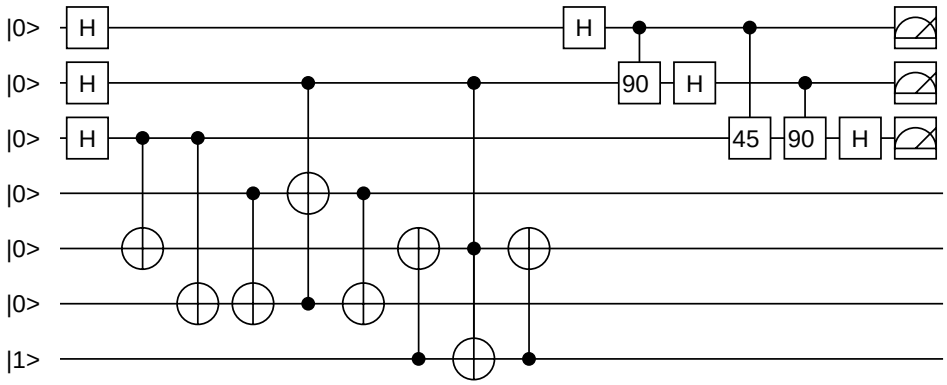


図 3: 実際の回路の例 (EPS 形式)：EPS 形式で出力できるため、論文などに貼り込みやすい。また、EPS 形式はベクターデータであるために、拡大縮小が自由である。

回路図の一番上のビットを中間コードでは 0 番と定義している。

7 自動並列化

（1） 量子計算回路の計算量

量子計算回路は、通常の古典的な回路では実現できない高速な計算が可能となる[*3]。逆に、量子計算回路を古典計算回路でシミュレートしようとすると、膨大な計算量とメモリが必要となる。

量子計算回路では、ひとつのゲートを計算するのにほぼ 2^n 回の計算が必要になるが、実際にどれだけの計算量になるか見積もってみよう。400Mflops(一秒間に 4 億回の浮動小数点演算が可能) のマシンで 32 ビットの 1 ビット回路を計算することを考える。1 ビット回路の計算には 2^{32} 個の浮動小数点の四則演算が最低 4 回程度必要なため[*4]、ほぼ 80 秒程度で計算できる。数千段の回路を考えたとしても数日で計算できる量であり、非現実的な計算量ではない。

次に、必要なメモリを考える。すでに述べたように、32 ビットの全状態を記憶するためには、2^{32} 個の複素数が必要となる。実数を 8Byte(C 言語なら double) で表現すると、$2^{32} \times 8 \times 2 \sim 64 \times 10^{30}$Byte、すなわち 64GB(ギガバイト) のメモリが必要となる[*5]。計算量は家庭用の PC でも十分計算可能であるが、64GB のメモリは非現実的である。もちろんビット数が増えれば計算量も指数関数的に増えるが、まず計算量よりもメモリ容量がボトルネックとなることがわかる。したがって、実際の計算では複数台の計算機で分散処理を行い、メモリの問題を解決する必要がある。そのためには並列化プログラミングが必要となるが、量子回路を書き換えるたびに並列化処理を施すのは現実的ではない。以上から、量子計算シミュレーターには自動並列化システムが必須であることがわかる。

（2） 自動並列化システム

本システムでは、自動並列化を以下のような仕様で実現している。

これらを実現するため、状態タグによる実行時動的並列化システムを考案した。これは、いわば郵便番号のようなシステムで、状態番号の一部をタグとし、どのマシンに属すべきかを決定するシステムである。タグは、状態番号をビット表記した場合、上位ビットを用いる。

図 4 に概念図を示す。8qubit の状態番号は、$|00000001\rangle$ のようにあらわすことができる。8qubit の場合、状態数は $2^8 = 256$ 個であるが、これを 4 つの計算機で分散処理することを考えよう。この場合、上位 2 ビットをタグとして扱う。マシンにそれぞれマシン番号 0,1,2,3 をつけて対応するマシンがそのメモリを保持することとする。たとえば $|\underline{00}000001\rangle$ は 0 番のマシンが $|\underline{01}101011\rangle$ なら 1 番のマシンがデータを保持する。下線を引いた部分がタグであり、4 並列なら 2bit、8 並列なら 3 ビットをタグとする[*6]このように、状態番号が与えられたときは、上位ビットを見ればどのマシンにデータが存在するかがわかる。

逆に、各マシンはそれぞれ 64 個のデータを保持していることになる。これを二進数表記すると、たとえば 5 番目のデータは $|000101\rangle$ と 6 ビットで表現できる。いま、マシン番号が 2 番だとすると、上位ビットに 10 を加え、$|\underline{10}000101\rangle$ が実際の状態番号となる。

以上の仕様を MPI を用いて実装した。実行時にタグを判別する動的タグシステムを採用したため、実行時ノードスケーラブルとなった。実行時ノードスケーラブルとは、コンパイルされた実行可能ファイルが、再コンパイルの必要なく並列数を変更できる機能のことで、マシンの数を変えてもそのまま並列実行できるために便利である。

8 計算結果の解析

量子計算回路の結果は膨大なデータとなるため、そのままの解析は難しい。すでに述べたように 32qubit の情報は 64GB のデータとなる。そのため、可視化の方法を工夫しなくてはならない。QCAD は現在 3 種類の表示方法をサポートしている。まず、そのままのデータを表示する方法である。この方法では全状態数の位相と、実数部、虚数部、絶対値の表示などを表示する。さらに、絶対値の大きさによるソートができる。

[*3] これは、古典回路で動作するアルゴリズムが量子回路ではより高速に動作するという意味ではない。高速な計算を実現するためには、量子計算回路のみで動作する量子アルゴリズムを考案しなくてはならない。

[*4] Controlled Not の場合。三角関数や指数関数が必要な場合はさらに 10 倍近くの計算量が必要となる。

[*5] 最後の 2 倍は、ひとつの複素数を表現するのに実数部と虚数部の二つの実数を必要とすることによる。

[*6] したがってこの方法では 2 の冪乗個のマシンの数で並列化することを想定しているが、これは並列計算機では標準的な仕様なので問題は無い。

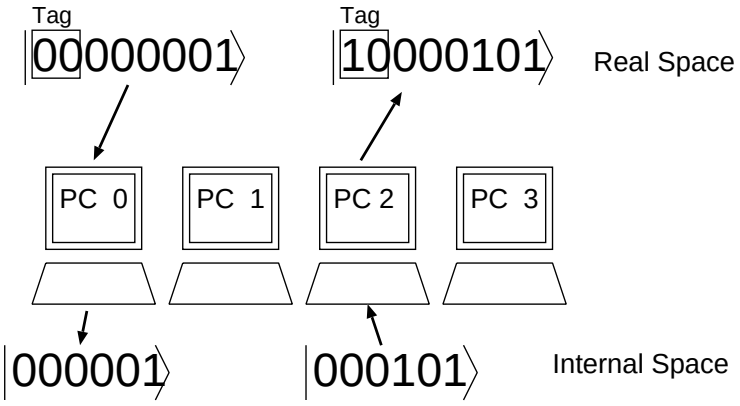


図 4: 自動並列化の概念図：8qubit の情報を 4 台のマシンで分散保持する場合の例を示す。4 台の場合はタグは上位 2bit となる。状態 $|00000001\rangle$ の情報は、0 番のマシンが保持しており、内部では 6bit の状態番号 000001 として管理している。逆に 2 番のマシンが内部で 000101 として保持している情報は、実際には $|10000101\rangle$ の状態に対応する。下線部はタグとして使われていることをしめす。このようにして、256 個の状態を 64 個ずつにわけて管理することができる。

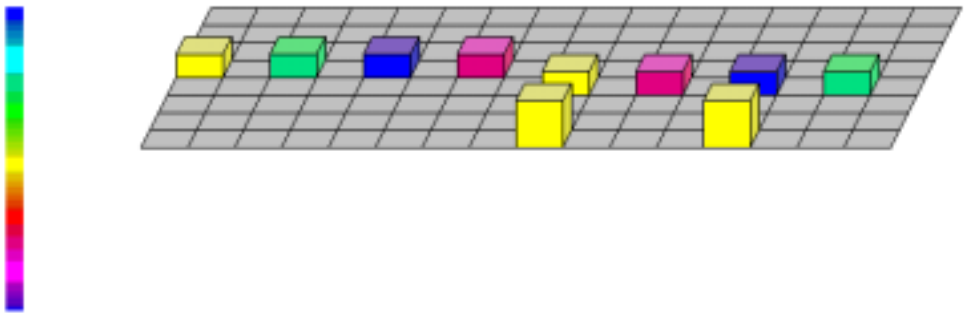


図 5: 計算結果の例：QCAD の結果表示法の一つ、3D View。色が位相、高さが絶対値をあらわす。注目すべき状態番号をすぐに見つけるのに便利な表示法である。

他に、絶対値を高さ、位相を色相であらわす 3D View と、絶対値を彩度、位相を色相であらわす 2D View をサポートしている。このうち、3D View の例を図 5 に示す。

9 これからの展望

QCAD により、量子計算機研究にかかる手間が大幅に改善され、新しい量子アルゴリズムの開発などが活発化されることが期待される。しかし、本システムはまだ開発中であり、いくつかの問題点もある。まず、回路設計において数千段の回路を開発する際にはマクロ機能が必要であろう。自動並列化においては、この方法でメモリの問題は解消されるが、通信コストの軽減が図られていないため、並列化していない場合に比べ計算時間が大幅に増えてしまう。そのためスケジューリングや、データをまとめて送信するなどの高速化が必要となる。結果表示について、2D View や 3D View はそれぞれ、全体の状態を把握するのに便利であるが、実用的に使用するためには SQL ライクな文法による検索、詳細な条件を指定できるソート等、さらに洗練されたインターフェースが必要であろう。これらはこれからの課題として、順次対処していく予定である。

10 開発成果の公開

開発されたシステムはウェブサイト http://acolyte.t.u-tokyo.ac.jp/~kaityo/qcad/ からダウンロードすることができる。英語によるオンラインドキュメントもあわせて閲覧できる。ソースは現在[*7]整理中であるが、コメントなどを整備したあとに公開する予定である。

11 参加企業及び機関

三菱マテリアル株式会社
（プロジェクト管理組織）

12 謝辞

本プロジェクトの機会を与えてくださり、かつ様々な相談に載ってくださった竹内郁雄プロジェクトマネージャに感謝いたします。

13 参考文献

参考文献

[1] P. W. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring,” FOCS, pp. 124–134 (1994), SIAM Journal on Computing, Vol. 26, No.5, 1484(1997).

[2] L. K. Grover, “A fast Quantum Mechanical Algorithm for Database Search”,ACM Symposium on Theory of Computing(1996).

[*7] 2003 年 4 月現在。

[3] L. M. K. Vandersypen *et. al.* , "Experimental realization of Shor's quantum factoring algorithm using nuclear mangetic resonance", Nature, Vol. 414, 883(2001).