平成 17 年 10 月 5 日

量子計算シミュレータ
～ QCAD 計画　～

渡辺宙志 [1,2*] , 鈴木将 [2], 山崎淳之介 [2]

[1] 名古屋大学情報科学研究科複雑系科学専攻
[2] 東京大学工学系研究科物理工学専攻

概 要

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行えるソフトウェア QCAD を開発した。QCAD で設計された回路は、小規模な回路であればそのまま実行することができ、さらに、自動並列化されたコードを出力し、並列計算機で実行することで大規模な回路のシミュレーションをサポートする。回路図の出力、印刷機能等も備えており、QCAD により量子計算研究がより効率的に行われることができると期待できる。本稿では、量子計算機シミュレーションの基礎から、QCAD 計画の概要、自動並列化についてまで述べる。

目 次

*E-mail:hwatanabe@is.nagoya-u.ac.jp

1 はじめに

本資料は量子計算シミュレータについて解説し、QCAD 計画を説明するために渡辺が作成したものである。わかりやすさを優先したため、所々に正確さを欠く表現があることに注意して欲しい。構成として、まず量子計算の実際に触れ、なぜシミュレータが必要となるか、シミュレータで何が問題になるかを解説した。その後、量子計算シミュレータの諸問題を解決すべく立ち上げたプロジェクト、QCAD 計画の詳細を説明している。全体的に誤解を招きそうな個所には注釈を入れるよう心がけたが、明らかな誤りなどを見つけた場合には渡辺まで連絡していただければ修正する。

2 量子計算回路

2.1 量子計算とは

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, 4]、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。

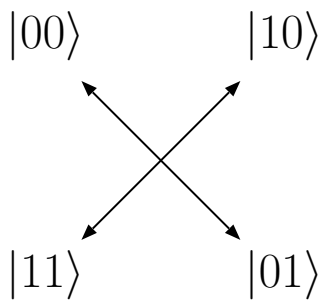
2.2 記法について

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。インデックスは 0 から始めることにし、0 ビット目、と言った場合、レジスタの一番右のビットのことを表すことと約束する。

2.3 古典計算と量子計算の違い

量子計算機と区別するため、通常の計算機を古典計算機と呼ぼう。簡単のため、2bit のレジスタを考える。このレジスタは二つの bit のオン ($|1\rangle$) とオフ ($|0\rangle$) により、0($|00\rangle$) から 3($|11\rangle$) までの 4 つの状態を取ることが出きる。これは古典、量子双方で同じである。

古典計算機では、レジスタの値はいつでも 4 つの状態のどれかであり、一度の演算で、その状態から別の状態へ遷移する。例えば状態が 0($|00\rangle$) なら 0 ビット目に論理否定をかけると 1($|01\rangle$) へかわる。同様に状態 2 は状態 3 になる。これを行列で表すと図 1 のようになる。



$$\begin{pmatrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{pmatrix}$$

図 1: 0 ビット目に論理否定を演算した場合のレジスタの状態遷移図。右に対応する遷移行列を示す。

状態 0 から状態 3 の重みをそれぞれ $\alpha_n (n = 0, 1, 2, 3)$ としよう。古典計算機では、状態は必ずどれか一つであるので、α_n の取りうる値は 0 か 1、さらに α_n は同時に二個以上が 1 になることはできない。この重みを並べたベクトルを状態ベクトルと呼ぼう。

古典レジスタの状態ベクトルは $(1,0,0,0)$, $(0,1,0,0)$, $(0,0,1,0)$, $(0,0,0,1)$ の 4 つの状態しか取ることができない。例えば $(1,0,0,0)$ は状態 0($|00\rangle$) に、$(0,1,0,0)$ は状態 1($|01\rangle$) に対応する。

さて、同様な計算は量子レジスタではどのように表されるだろうか。量子計算の最大の特徴は、状態の重ね合わせが許されることにある。これは先ほどの状態の重み α_n が 0 と 1 以外の値を取ることができるということに対応する。例えば状態ベクトルとして $(1/\sqrt{2}, 1/\sqrt{2}, 0, 0)$ という状態は、状態 0 と状態 1 が半分ずつ混ざったような状態となる。いま、量子レジスタが重み $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ で表されているとしよう。この状態に先ほどと同様に 0 ビット目に論理否定を演算すると、演算前に

$|00\rangle$ であった状態は $|01\rangle$ に、$|01\rangle$ は $|00\rangle$ になるので ($|10\rangle, |11\rangle$ も同様)、重み α_0 と α_1、α_2 と α_3、をそれぞれ交換することになる。これを行列で表すと、

$$\begin{pmatrix} \alpha'_0 \\ \alpha'_1 \\ \alpha'_2 \\ \alpha'_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} \tag{1}$$

となる。ただし α'_n は演算後の重みである。式 (1) を見てわかるとおり、古典計算機の場合と同じ行列である。違うのは入力となる初期状態に重なりが許されるか許されないか、ということだけである。この違いがどのような違いを生むだろうか。

まず、2bit のレジスタで表現できる情報量を考える。古典レジスタでは 4 つの状態しか表すことができない。しかし量子レジスタでは、状態を表すのに 4 つの複素数の組 $\alpha_n (n = 0, 1, 2, 3)$ を必要とするので、逆にこの 4 つの複素数で表すことができる情報を持つことができる。この違いは大きい。かたや離散 1 次元の情報空間しかないのに、量子系では 4 次元の複素数による情報空間を持っていることになる (正確には規格化条件があるので 4 次元複素空間の球面上に制限されるが、次数が一つ下がるだけなので古典計算機と比べて桁違いに大きな情報空間を持っていることにはかわらない)。従って、同じ 2 ビットのレジスタを用意しても、古典計算機に比べ量子計算機は圧倒的に大きな表現力を持っていることになる。一般に、$n$qubit のゲートは $2^n \times 2^n$ の複素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} 回の複素数演算、すなわち 2^{2n+1} 回の実数演算に相当する。この違いが量子計算機が古典計算機より早く計算できる理由となる。

2.4 これまでの量子計算研究

これまでの量子計算研究を振り返ってみよう。量子計算機の元になったアイディアは発熱しない計算機は可能か、すなわち「可逆な計算機は可能か」という問題から生まれた。発熱はエントロピーが増える時、すなわちメモリの消去が必要な時に起きる。したがって計算過程では発熱しないようにすることが可能である。1973 年に Bennett は可逆なゲートのみから構成される計算機の理論を構築し、この計算機では原理的に発熱しないことを示した [5]。次いで 1980 年、Benioff が量子論の原理に基づいた可逆計算機を示した。1982 年には、Feynman が量子的な現象を古典計算機でシミュレートすると指数関数的な時間 (もしくは手間) が必要となるが、量子計算機で計算すればそうはならない、ということを論じた [7]。1985 年に Deutsch が量子 Turing 機械を示し [8]、それに基づいて 1993 年に Bernstein と Vazirani が量子 Turing 機械でどのように古典 Turing 機械を実現するか示した [9]。したがって、原理的に計算可能なことは全て量子計算機で計算可能であることが証明された。(もちろん量子計算機の方が早く計算できるかどうかは別問題である)。1994 年に Shor が多項式時間で因数分解できる量子アルゴリズムを示した [1]。この論文により量子計算研究が注目を浴びるようになったと言ってよいだろう。1996 年には Grover が $O(N^{1/2})$ の探索アルゴリズムを発表した [2]。量子暗号など、他にも量子計算機でなくては実現できないアルゴリズムも数多くある提案されているが、古典計算機でできる計算で、古典計算機よりも飛躍的に早く計算できるアルゴリズム、という意味ではこの二つが代表的である。

量子計算機の実機を作ろうという研究も盛んである。これまでに、NMR(核磁気共鳴) を用いる方法、量子ドットを用いる方法、光の閉じ込めにより量子状態を作る方法などが提案されている。このうち、CeP 結晶や Si 結晶を用いた結晶量子コンピュータでは、10qubit 以上の量子計算の室温動

作に成功している [3]。最近では 2001 年に Vandersypen らが、液体の分子のスピンを利用し、NMR による操作で Shor のアルゴリズムを実現している [4]。

3 量子計算シミュレーション

3.1 シミュレーションの実際

前節までで量子計算の概要と実際の量子計算機について説明したが、まだ量子実用的なアルゴリズムは少なく、また、容易に回路設計できる量子計算機も無い。特に量子計算機はノイズによる重ね合わせ状態が壊れる (デコヒーレンス) のが問題であり、あまり大きな回路を試すことができない。そこで、量子計算機を古典計算機でシミュレートすることで、新しい量子アルゴリズムの研究が進められている。

量子計算を古典計算機でシミュレーションする原理は非常に簡単である。n ビットのレジスタには 2^n 個の状態が存在するが、それぞれの状態の重みは一つの複素数、すなわち二つの実数で表現できる。したがって、2^n 次元のベクトルを用意し、回路上のゲートに対応した $2^n \times 2^n$ の行列を掛け算していけばよい。さらに任意の演算を 1bit ゲートと Controlled Not の組み合わせだけで表現できることが証明されているため、一度の計算にかかわる状態は二つだけであるので、原理的には 2×2 の複素行列を次々と演算していくことで量子計算を表現することができる。このことをもう少し詳しく見てみよう。

2.3 節で述べたように、2qubit のレジスタに対する 1bit の否定演算は 4×4 の行列で表された。同様に $n$qubit のレジスタに対する 1bit の否定演算は、$2^n \times 2^n$ の行列で表される。しかし、その計算内容は、0 ビット目への否定演算なら、$|q_{n-1}q_{n-2}\cdots q_1 1\rangle$ と $|q_{n-1}q_{n-2}\cdots q_1 0\rangle$ の重みを交換しているだけであり、行列で表現すれば $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ を演算していることになる。結局、全ての $q_i (i = 1, 2, \cdots, n-1)$ について 2 行 2 列の複素行列を計算すればよいので、この計算を 2^{n-1} 回繰り返せば良い。以上から、量子計算シミュレーションでは、一つの量子ゲートの計算は 2 行 2 列の複素行列を 2^{n-1} 回繰り返すことに対応する。

3.2 量子計算の計算量

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が、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。

既に述べたように、量子計算回路のゲートひとつを古典計算機で計算するのには、ほぼ 2^n 回の計算が必要になるが、これが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 32qubit レジスタに作用するゲート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 Controlled Not の場合で 4 回程度必要なためほぼ 80 秒程度で計算できる。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が、それらを含めて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よう。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 8 バイト (C 言語なら double) で表現すると、$2^{32} \times 8 \times 2 \sim 64 \times 10^{30}$ バイト、すなわち 64GB(ギガバイト) のメモリが必要となる。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。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。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。もちろん搭載メモリがどんどん増えていけば、64GB の計算を家庭用 PC で行えるようになる時も来るだろう。しかし、順調な伸びを見せる CPU の計算性能に対し、メモリ搭載量は最近伸び悩みを見せている気配がある。いずれにせよ、ビット数が増えれば計算速度よりも必要メモリがボトルネックとなる状況は変わらないであろう。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。

4 実際のプログラミング

4.1 量子レジスタの表現

まず、$n$qubit 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 進数表記として解釈し、その数字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、後で量子計算の実装が容易になる。

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。

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。

4.2 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 ビット目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)) i++;` 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。

4.3 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);

    //p1,p2 ... Control Bits

    if(p1>p2){
        swap(p1,p2);
      }
    for(unsigned int i=0;i<state;i++){
       unsigned int ix0 = insert1(i,p1);
       ix0 = insert0(ix0,p2);
       unsigned int ix1 = insert1(i,p1);
       ix1 = insert1(ix1,p2);
      //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。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。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。

5 自動並列化

5.1 並列計算の必要性

3.2 節で述べたように、量子シミュレーションには莫大なメモリが必要となり、多くの場合そこがボトルネックとなる。そこで、多数のマシンで分散処理する必要がある。たとえば 32qubit 回路のシミュレーションには最低で 64GB のメモリが必要となるが、64 台の PC クラスタで計算を行えば (当たり前だが) 一台あたり 1GB のメモリですむ。最近では数台〜数十台の大きさの PC クラスタは一般的であるし、超並列型のスーパーコンピュータを使うこともできるだろう。このような分散メモリ型並列計算機を利用して量子計算のシミュレーションを行うことを考える。

5.2 並列計算の実装

並列計算にはいくつかの手法があるが、ここでは MPI を用いた実装を行う。MPI(Message Passing Interface) とは、分散メモリ型の並列計算機で 2 つ以上のプロセス間でのデータをやりとりするために用いるメッセージ通信操作の仕様標準である。MPI とは仕様であり、その仕様を実装したライブラリ MPICH が自由に利用できる。MPI の仕様は現在 MPI2 まで制定されており、MPICH はまだその全てを実装しているわけではないが、いずれ完全にカバーすると思われる。MPI を用いたプログラムは、mpirun というコマンドで起動する。

実行時、並列計算を行うマシンには一意な番号が割り当てられる。それは rank と呼ばれ、その番号により動作を変えることで並列計算を実現する。たとえば、rank0 のマシンから rank1 のマシンへデータを送ったり、その逆も行うことができる。このように MPI は基本的に一対一通信を行うことで並列計算を行う規格である[1]。

5.3 状態タグシステム

筆者らは量子計算シミュレーションの並列化を容易に行うため、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 2 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。したがってこの方法では 2 のべき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。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

[1] 全プロセスからデータを集めるなど、一対多数の命令も存在する。余談だが、こういった命令も実際には一対一通信の組み合わせで実装されていることが多く、一対一通信より通信速度が早いとは限らない。

え、$|\underline{10}0000101\rangle$ が実際の状態番号となる。

マシンの管理タグを状態タグの上位ビットとすることで、この並列化コードは実行時ノードスケーラブルとなる。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実行できるために便利である。具体的には、2^n 台のマシンで Nqubit の量子計算を行うとしたら、上位 n ビットを状態タグとし、それぞれのマシンで 2^{N-n+4} バイトのメモリを確保すればよい。

5.4 並列化の実際のコーディング

状態タグを用いて p ビット目に論理否定を演算するコードについて考えよう。並列化をしない場合と同様に、全てのマシンにおいて `int i = 0 ;i < (1<< (N-1));i++` 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 を挿入して得られた `ix0` と 1 を挿入して得られた `ix1` の値を全ての `i` について入れ替えなくてはならない。この時、状態 `ix0` と `ix1` のデータがどのマシンに属すか調べ、それによって通信を行い適切に計算を実行する必要がある。

`ix0` と `ix1` のタグと計算マシンの rank(固有な ID 番号) によって、次の 3 通りの可能性がありうる。以下、「自分」というのは計算を行っているマシンのうちの一つであり、「タグが自分の rank と一致する」とは、たとえば `ix0` の上位 2 ビットが自分に割り当てられた rank と同じであることを意味する。

- **タグが両方とも自分の rank と一致** この場合、 計算に必要なデータは両方とも自分が持っているので、通信を行う必要は無い。並列計算でない場合と同様に計算し、結果を保存すればよい。
- **タグが両方とも自分の rank と異なる** 計算に必要なデータは自分には無いので、何もしなくて良い。
- **タグが片方だけ自分の rank と一致** 計算に必要なデータの片方は自分が持っており、もう片方は別のマシンが保持しているため、通信をする必要がある。タグを調べ、もう片方のマシンにデータを要求し、自分のデータも相手に送る。そうして得られたデータについて演算し、結果を自分に関係するところだけ保存する。

以上の手続きを全ての状態番号について行えば計算は完了する。論理否定より複雑な回路でも、基本は変わらない。すなわち、計算に必要なデータがどこにあるか調べ、必要があれば通信を行い、計算結果を保存すればよい。

5.5 並列化の問題点

前述の方法でメモリの問題は解消されるが、計算速度が非常に遅くなるという問題点が生じる。まず、通信を頻繁に行うため、データ要求から通信が開始されるまでの時間、遅延時間 (latency) の影響を強く受ける。遅延時間は一般に CPU の数百〜数千サイクルほどであり、そのあいだ計算機は遊んでしまうことになる。

これを防ぐため、スケジューリングを行って効率的な通信をする、データをまとめて送ることで通信回数を減らす (すなわち遅延時間を減らす) などの対策をする必要がある。

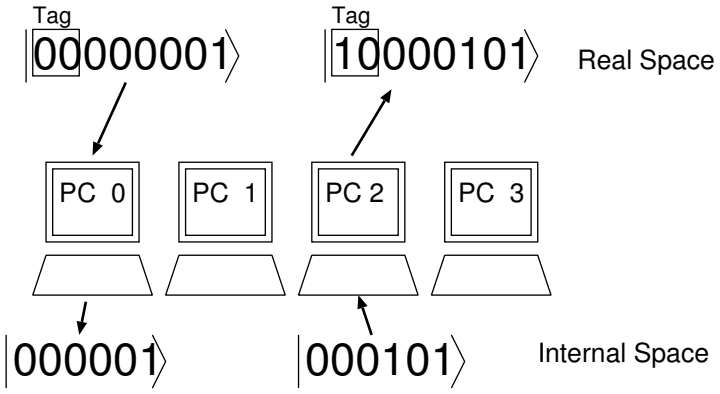


図 2: 自動並列化の概念図：8qubit の情報を 4 台のマシンで分散保持する場合の例を示す。4 台の場合はタグは上位 2bit となる。状態 $|\underline{00}0000001\rangle$ の情報は、0 番のマシンが保持しており、内部では 6bit の状態番号 000001 として管理している。逆に 2 番のマシンが内部で 000101 として保持している情報は、実際には $|\underline{10}0000101\rangle$ の状態に対応する。下線部はタグとして使われていることをしめす。このようにして、256 個の状態を 64 個ずつにわけて管理することができる。

---

6 量子計算統合環境 QCAD

6.1 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ソフトウェアでは、簡単なものでも数十万円〜数百万円以上のすることが多い)。従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見受けられない。本プロジェクトでは、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。

6.2 プロジェクトの概要

本プロジェクトでは、量子計算機研究を総合的にサポートすることを目的として、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行える統合環境 QCAD を作成した。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 上でシミュレートすることも可能であり、大規模な回路は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することができる。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 3 に QCAD の概念図を示す。

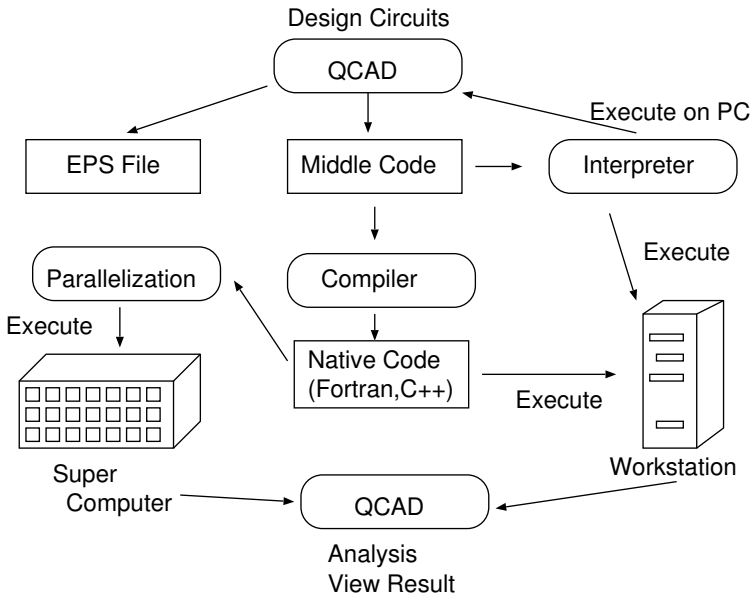


図 3: QCAD の概念図。Windows 上で動作する QCAD で量子回路を設計し、作成した回路は論文用に EPS ファイル、実行用に中間コードで出力される。中間コードはインタプリタで読み込んで実行することができ、結果はそのまま PC で確認できる。インタプリタは独立して動作できるため、ワークステーションなどで、より高速に実行させることもできる。また、中間コードをコンパイルして Fortran や C 言語で出力して実行できる。さらに MPI などを用いた並列化を行い、スーパーコンピューターで実行し、結果を QCAD で解析する。

6.3 システム構成

本システムは、回路設計ソフトウェア、インタプリタ、コンパイラの三つのソフトウェアで構成されている。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。それぞれのソフトウェアの概要は以下の通りである。

6.3.1 qcad(回路設計、データ解析ソフトウェア)

Windows 上で動作する GUI ソフトウェアで回路の設計、保存、印刷、出力形式を選んだ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。

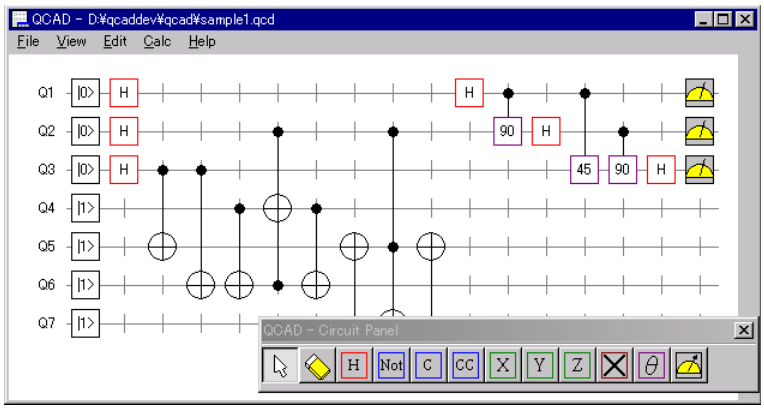


図 4: QCAD のスクリーンショット。Windows 95/98/ME/2000/NT/XP で動作し、マウスで簡単に操作できる。回路をパレットから選んで置きたいところをクリックしていけば回路図を作成することができる。アンドゥなどもできる。データの出力形式は独自形式のほか、EPS、BMP をサポートしている。開発した回路はその場で動作確認することができる。大規模な回路については、それに対応する中間コードを作成し、インタプリタ実行やコンパイルによる実行を可能とする。

6.3.2 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。

6.3.3 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

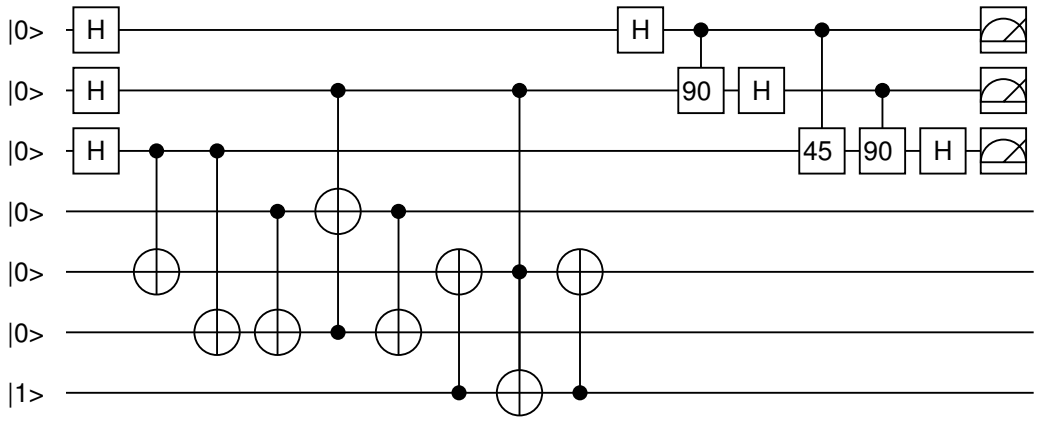


図 5: 実際の回路の例 (EPS 形式)：EPS 形式で出力できるため、論文などに貼り込みやすい。また、EPS 形式はベクターデータであるために、拡大縮小が自由である。

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。

6.4 中間コード

6.4.1 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表現するテキストのことであり、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、回路図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。このような仕様にすることで、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。

6.4.2 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。

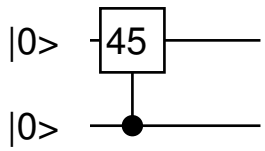
コメント部分 #から始まる行で、これは解析時に無視される。

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、…、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。

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。

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。

6.4.3 中間コードの例

以下に、図 5 の回路に対応する中間コードを載せる。図 5 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 [4]。

```
# 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
```

ただし、中間コードでは、回路図の一番上のビットを 0 番と定義している。

6.5 結果解析

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。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 6 に示す。

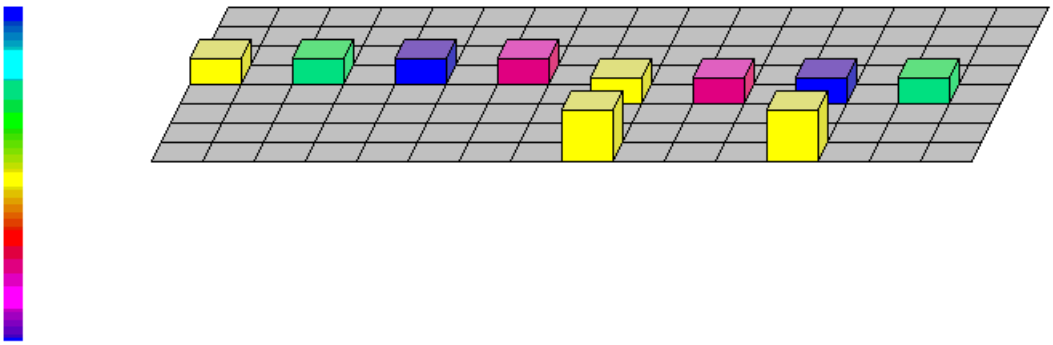


図 6: 計算結果の例：QCAD の結果表示法の一つ、3D View。色が位相、高さが絶対値をあらわす。注目すべき状態番号をすぐに見つけるのに便利な表示法である。

6.6 開発成果の公開

開発されたシステムはウェブサイト[2]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。

7 おわりに

本稿では量子計算機シミュレーションの実装方法から、QCAD プロジェクトの概要までを述べた。我々が開発した 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を期待する。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。

QCAD プロジェクトは、情報処理振興事業協会の公募事業、平成１４年度未踏ソフトウェア創造事業「未踏ユース」に採択されたものであり、ＰＭ（電気通信大学　竹内郁雄教授)、プロジェクト管理組織（三菱マテリアル株式会社）と我々開発者の体制で実施した。開発その他について様々なアドバイスを下さった竹内教授および管理組織の池田様に感謝いたします。

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).

[3] F. Yamaguchi and Y. Yamamoto, ”Crystal Lattice Quantum Computer,” Applied Physics A **68**, 1 (1999); T. D. Ladd *et. al.* , ”Decoherence in Crystal Lattice Quantum Computation,” Applied Physics A **71**, 27 (2000).

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

[5] C. H. Bennett, “Logical Reversibility of Computation”, IBM J. Res. Develop. **17**, (1973) 525-532.

[6] P. Benioff, “The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines”, J. Stat. Phys. **22**, (1980) 563.

[7] R. P. Feynman, “Simulating Physics with Computers”, Int. J. of Theor. Phys., **21** (1982) 467–488.

[8] D. Deutsch, ”Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer,” Proc. R. Soc. Lond. A, **400**, (1985) 97.

[9] E. Bernstein and U. Vazirani, “ Quantum complexitytheory ”in Proc. 25th ACM Symp. on Theory of Computating, (1993) 11.

[10] A. Barenco et al. , “Elementary gates for quantum computation”, Phys. Rev. A 52 (1995) 3457-3467.

[2] http://www.phys.cs.is.nagoya-u.ac.jp/~watanabe/qcad/