

ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ ИМ. БУДКЕРА

А.Г. Грозин

Квантовый компьютер для чайников

Препринт ИЯФ 2004-40

НОВОСИБИРСК

2004

# Квантовый компьютер для чайников

*А.Г. Грозин*

Институт ядерной физики им. Будкера  
Новосибирск 630090

## АННОТАЦИЯ

Введение в квантовые компьютеры, квантовую криптографию и квантовую телепортацию для тех, кто никогда об этих вещах не слышал, но знает квантовую механику. Ряд простых примеров подробно рассмотрен с использованием эмулятора квантового компьютера QCL.

---

## 1. Квантовый компьютер

Я не буду обсуждать попытки технической реализации квантового компьютера. Много групп во всём мире этим занимаются, используя разнообразные подходы. Достигнут некоторый прогресс, и, несомненно, значительно большее продвижение произойдёт в близком будущем. Но до работающего квантового компьютера, способного делать что-то полезное, пока далеко. Я не специалист по конкретным методам реализации. Вместо этого, я расскажу о некоторых простых вещах, которые можно будет делать с квантовым компьютером, когда технические трудности его создания будут преодолены.

Литература на эту тему весьма обширна. Я не пытался составить подробный список литературы; приведены только ссылки на некоторые работы, посвященные тем конкретным вопросам, которые будут обсуждаться. Более обширные списки литературы можно найти в учебниках и подробных обзорах.

### 1.1. Устройство и система команд

Главная часть квантового компьютера – память, состоящая из квантовых бит. Будем считать, что квантовый бит – это спин  $\frac{1}{2}$ ; спин вверх означает  $|0\rangle$ , вниз –  $|1\rangle$ . Конечно, годится и любая другая система с двумя базисными состояниями. Память из  $n$  бит может быть в состоянии  $|0000000\rangle$  или  $|0110001\rangle$  или ... или в суперпозиции таких базисных состояний. Состояние памяти – вектор в  $2^n$ -мерном пространстве.

Как известно, всякий эксперимент в квантовой механике состоит из 3 этапов: приготовление начального состояния; эволюция состояния (описываемая уравнением Шрёдингера); измерение какой-нибудь наблюдаемой в конечном состоянии. Цикл вычислений на квантовом компьютере состоит из тех же 3 этапов:

- память приводится в начальное состояние  $|00000000\rangle$ ;
- производится некоторая последовательность унитарных преобразований, действующих на отдельные биты или пары битов;
- измеряется один или несколько битов.

Унитарные преобразования (команды квантового компьютера) выполняются под управлением программы в обычном классическом компьютере. Это показано на рис. 1, который позаимствован из [1].

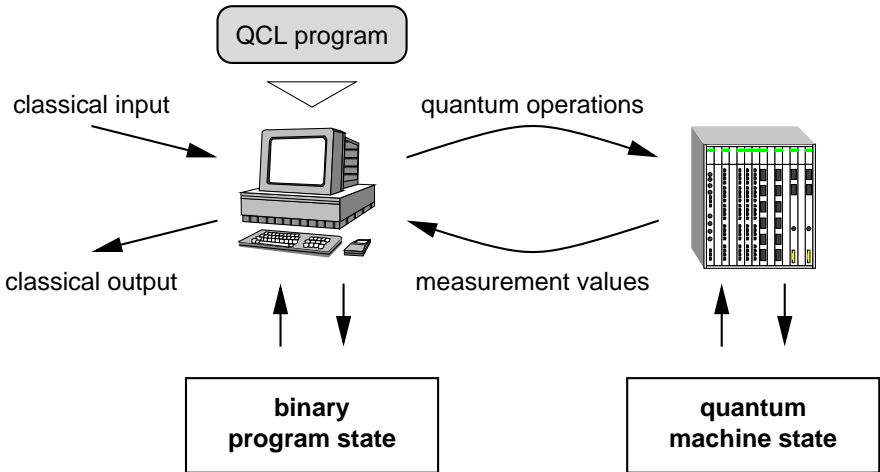


Рис. 1: Квантовый компьютер

Законы квантовой механики ограничивают то, какие операции мы можем производить с памятью квантового компьютера во время вычислений. Всякое унитарное преобразование обратимо

$$\hat{U}^{-1} = \hat{U}^\dagger.$$

Поэтому во время работы квантового компьютера невозможны необратимые операции. Нельзя, например, установить бит в 0, затерев его предыдущее значение.

Нельзя также скопировать бит в другой бит, даже если второй бит находится перед этой операцией в определённом состоянии, скажем,  $|0\rangle$ . Допустим, существует оператор  $\hat{U}$  такой, что для всех  $|\psi\rangle$

$$\hat{U}|\psi\rangle \otimes |0\rangle = |\psi\rangle \otimes |\psi\rangle.$$

Тогда

$$\hat{U}|0\rangle \otimes |0\rangle = |0\rangle \otimes |0\rangle, \quad \hat{U}|1\rangle \otimes |0\rangle = |1\rangle \otimes |1\rangle.$$

Для  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ , по линейности,

$$\hat{U}|\psi\rangle \otimes |0\rangle = \alpha|0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle \neq |\psi\rangle \otimes |\psi\rangle,$$

то есть такого оператора не существует.

Если некоторое вычисление  $\hat{U}$  с начальным состоянием памяти  $|i\rangle$  даёт результат  $\hat{U}|i\rangle$ , то при начальном состоянии памяти  $\sum_i \psi_i |i\rangle$  оно даст результат  $\sum_i \psi_i \hat{U}|i\rangle$ . То есть все  $2^n$  вычислений  $\hat{U}|i\rangle$  как бы производятся параллельно. Это позволяет надеяться, что некоторые задачи можно решать на квантовом компьютере гораздо быстрее, чем на классическом. Но при измерении мы узнаём лишь один ответ, причём мы не можем контролировать, какой именно. Хорошо спроектированный квантовый алгоритм должен выдавать желаемый ответ с вероятностью 1, или хотя бы с вероятностью порядка 1 (тогда можно повторить вычисление несколько раз). Если вероятность получения желаемого ответа экспоненциально мала, то от такого алгоритма нет никакой пользы.

Лучше один раз увидеть, чем сто раз услышать. Поскольку у нас нет работающего квантового компьютера, мы будем использовать вместо него эмулятор qcl (quantum computing language) [1], работающий на обычном классическом компьютере. Это свободная программа, каждый может скачать её и поэкспериментировать с ней. Ниже приводится сеанс работы с qcl, демонстрирующий команды квантового компьютера – операции с квантовыми битами и их парами.<sup>1</sup>

<sup>1</sup>Этот сеанс работы производился из GNU TeXmacs [2] – свободного wysiwyg текст-процессора, обеспечивающего прекрасное качество не только текста, но и формул. Я занимался его русификацией, а также написал интерфейс TeXmacs с рядом систем компьютерной алгебры [3] и с qcl (и при этом внёс ряд мелких улучшений в qcl). Здесь использовалась версия qcl-0.6.1; команда его вызова из TeXmacs была заменена на

```
qcl --texmacs --dump-format=b --auto-dump=32
```

QCL Quantum Computation Language (32 qubits, seed 1081693763)

```
[0/32] 1 |0>
```

Квантовый компьютер с 32 битами памяти запущен. Назовём один бит  $x$ :

```
qc1> qureg x[1]
```

Поворот вокруг оси  $x$ :

$$\hat{R}_x(\vartheta) = \begin{pmatrix} \cos \frac{\vartheta}{2} & -i \sin \frac{\vartheta}{2} \\ -i \sin \frac{\vartheta}{2} & \cos \frac{\vartheta}{2} \end{pmatrix}.$$

```
qc1> RotX(pi/2, x)
```

```
[1/32] 0.70711 |0> - 0.70711i |1>
```

Поворот вокруг оси  $y$ :

$$\hat{R}_y(\vartheta) = \begin{pmatrix} \cos \frac{\vartheta}{2} & -\sin \frac{\vartheta}{2} \\ \sin \frac{\vartheta}{2} & \cos \frac{\vartheta}{2} \end{pmatrix}.$$

```
qc1> RotY(pi/2, x)
```

```
[1/32] (0.5 + 0.5i) |0> + (0.5 - 0.5i) |1>
```

Поворот вокруг оси  $z$ :

$$\hat{R}_z(\vartheta) = \begin{pmatrix} e^{-i\vartheta/2} & 0 \\ 0 & e^{i\vartheta/2} \end{pmatrix}.$$

```
qc1> RotZ(pi, x)
```

```
[1/32] (0.5 - 0.5i) |0> + (0.5 + 0.5i) |1>
```

Оператор Not:

$$\hat{N} = i\hat{R}_x(\pi) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \hat{N}|0\rangle = |1\rangle, \quad \hat{N}|1\rangle = |0\rangle.$$

```
qc1> Not(x)
```

```
[1/32] (0.5 + 0.5i) |0> + (0.5 - 0.5i) |1>
```

Приведём память в исходное состояние:

```
qc1> reset
```

```
[1/32] 1 |0>
```

Преобразование Адамара – поворот на  $\pi$  вокруг биссектрисы  $x$  и  $z$ :

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

Когда оно применяется к состоянию  $|0\rangle$ , получается

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} :$$

```
qc1> Mix(x)
[1/32] 0.70711 |0> + 0.70711 |1>
```

Его квадрат равен 1:

```
qc1> Mix(x)
[1/32] 1 |0>
```

Когда оно применяется к состоянию  $|1\rangle$ , получается

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} :$$

```
qc1> Not(x)
[1/32] 1 |1>
```

```
qc1> Mix(x)
[1/32] 0.70711 |0> - 0.70711 |1>
```

Опишем ещё 1 бит и назовём его  $y$ :

```
qc1> qureg y[1]
```

Повернём его вокруг оси  $x$ :

```
qc1> RotX(pi/2,y)
[2/32] 0.5 |0,0> - 0.5 |1,0> - 0.5i |0,1> + 0.5i |1,1>
```

Если каждый бит живет своей жизнью, это неинтересно: тогда мы имеем  $n$  независимых 1-битовых компьютеров вместо одного  $n$ -битового. Нужен оператор, действующий на пару битов. Обычно в качестве него выбирают простейший – Controlled Not. Если первый (управляющий) бит равен 0, то со вторым (управляемым) битом ничего не делается; если управляющий бит равен 1, то к управляемому биту применяется Not:

$$\begin{aligned} \hat{C}|00\rangle &= |00\rangle, & \hat{C}|01\rangle &= |01\rangle, \\ \hat{C}|10\rangle &= |11\rangle, & \hat{C}|11\rangle &= |10\rangle. \end{aligned}$$

Легко видеть, что это унитарная матрица  $4 \times 4$ .

```
qc1> x->y
[2/32] 0.5 |0,0> + 0.5i |1,0> - 0.5i |0,1> - 0.5 |1,1>
```

Теперь выключим квантовый компьютер:

qc1> exit

Разумеется, когда у нас будет настоящий квантовый компьютер, мы не сможем видеть состояние памяти после каждой операции. Это возможно только для эмулятора. Мы сможем что-то узнать только после измерения, да и то лишь вероятностным образом.

## 1.2. Пример квантового алгоритма

В обычном классическом программировании функции одного параметра реализуют так:

- параметр помещается во входной регистр;
- команды, образующие тело функции, производят над ним некие манипуляции, и помещают результат в выходной регистр, затирая его прежнее состояние.

В квантовом программировании последняя операция невозможна, так как она необратима. Вместо этого, биты результата прибавляются к битам выходного регистра по модулю 2. Иными словами, над ними производится операция Хог (исключающее или). Эта операция, очевидно, обратима: достаточно применить её второй раз, и память вернётся в исходное состояние.

Рассмотрим функции, отображающие 1 бит в 1 бит:

$$f : \{0, 1\} \rightarrow \{0, 1\}.$$

Каждой из них сопоставляется унитарный оператор

$$\hat{U}_f |x, y\rangle = |x, y \oplus f(x)\rangle.$$

Таких функций 4:

- 2 константы:  $f(x) = 0$  и  $f(x) = 1$ ;
- 2 “уравновешенные” функции:  $f(x) = x$  и  $f(x) = 1 - x$  (они называются уравновешенными, потому что принимают значения 0 и 1 в равном числе точек).

Допустим, кто-нибудь загадал функцию, а мы хотим угадать, к которому из этих двух классов она принадлежит. Для этого существует



квантовый алгоритм [4], который совершенно не похож на всё, к чему мы привыкли. Как  $\hat{U}_f$  действует на состояние  $|0\rangle \otimes |-\rangle$ , где

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}?$$

Мы получим

$$\begin{aligned}\hat{U}_f|0, 0\rangle &= |0, f(0)\rangle, & \hat{U}_f|0, 1\rangle &= |0, 1 - f(0)\rangle, \\ \hat{U}_f|0\rangle \otimes |-\rangle &= \frac{|0, f(0)\rangle - |0, 1 - f(0)\rangle}{\sqrt{2}} = (-1)^{f(0)}|0\rangle \otimes |-\rangle.\end{aligned}$$

Точно так же

$$\hat{U}_f|1\rangle \otimes |-\rangle = (-1)^{f(1)}|1\rangle \otimes |-\rangle.$$

Теперь возьмём начальное состояние  $|+\rangle \otimes |-\rangle$ :

$$\begin{aligned}\hat{U}_f|+\rangle \otimes |-\rangle &= \hat{U}_f \frac{|0\rangle \otimes |-\rangle + |1\rangle \otimes |-\rangle}{\sqrt{2}} \\ &= \frac{(-1)^{f(0)}|0\rangle \otimes |-\rangle + (-1)^{f(1)}|1\rangle \otimes |-\rangle}{\sqrt{2}}.\end{aligned}$$

Если  $f(x)$  – это константа, то  $(-1)^{f(1)} = (-1)^{f(0)}$ , и получается  $(-1)^{f(0)}|+\rangle \otimes |-\rangle$ . Если  $f(x)$  – уравновешенная функция, то  $(-1)^{f(1)} = -(-1)^{f(0)}$ , и получается  $(-1)^{f(0)}|-\rangle \otimes |-\rangle$ . Применим  $\hat{M}$  к первому биту. Для константы получится  $|0\rangle$ , а для уравновешенной функции  $|1\rangle$ .

QCL Quantum Computation Language (32 qubits, seed 1081606808)  
[0/32] 1 |0>

Необходимые описания:

```
qcl> qureg x[1]; qureg y[1]; int r;
```

Это оператор  $\hat{U}_f$ :

- $n = 0$  – для  $f(x) = 0$  (этот оператор ничего не делает);
- $n = 1$  – для  $f(x) = 1$ ;
- $n = 2$  – для  $f(x) = x$ ;
- $n = 3$  – для  $f(x) = 1 - x$ .

```

qc1> procedure U(int n, qreg x, qreg y)
  { if n==1 { Not(y); }      /* f(x)=1 */
    else { if n==2 { x->y; } /* f(x)=x */
      else { if n==3 { Not(x); x->y; Not(x); }}}
                                /* f(x)=1-x */
  }

```

Приготовим состояние  $|-\rangle$  бита  $y$ :

```

qc1> Not(y)
[2/32] 1 |0, 1>

```

```

qc1> Mix(y)
[2/32] 0.70711 |0, 0> - 0.70711 |0, 1>

```

Теперь приготовим состояние  $|+\rangle \otimes |-\rangle$ :

```

qc1> Mix(x)
[2/32] 0.5 |0, 0> + 0.5 |1, 0> - 0.5 |0, 1> - 0.5 |1, 1>

```

Теперь применим оператор  $\hat{U}_f$ , соответствующий функции  $f(x) = 1$ :

```

qc1> U(1, x, y)
[2/32] -0.5 |0, 0> - 0.5 |1, 0> + 0.5 |0, 1> + 0.5 |1, 1>

```

Применим преобразование Адамара к биту  $x$ :

```

qc1> Mix(x)
[2/32] -0.70711 |0, 0> + 0.70711 |0, 1>

```

Мы видим, что бит  $x$  находится в состоянии  $|0\rangle$ , как и должно быть в случае постоянной функции. Произведём измерение бита  $x$ , запишем результат в переменную  $r$ , и напечатаем её:

```

qc1> measure x, r
[2/32] -0.70711 |0, 0> + 0.70711 |0, 1>

qc1> print r
0

```

Приведём память в исходное состояние, и повторим всё для другой функции  $f(x)$ :

```

qc1> reset
[2/32] 1 |0, 0>

```

```

qc1> Not(y)
[2/32] 1 |0, 1>

```

```

qc1> Mix(y)

```

```
[2/32] 0.70711 |0,0> - 0.70711 |0,1>
```

```
qc1> Mix(x)
```

```
[2/32] 0.5 |0,0> + 0.5 |1,0> - 0.5 |0,1> - 0.5 |1,1>
```

```
qc1> U(2,x,y)
```

```
[2/32] 0.5 |0,0> - 0.5 |1,0> - 0.5 |0,1> + 0.5 |1,1>
```

```
qc1> Mix(x)
```

```
[2/32] 0.70711 |1,0> - 0.70711 |1,1>
```

```
qc1> measure x,r
```

```
[2/32] 0.70711 |1,0> - 0.70711 |1,1>
```

Теперь бит  $x$  равен 1, как и должно быть для уравновешенной функции  $f(x) = x$ :

```
qc1> print r
```

```
1
```

Можно написать процедуру, автоматизирующую весь алгоритм. Параметр  $n$  определяет, которую функцию  $f(x)$  использовать.

```
qc1> procedure Deutsch(int n)
```

```
{ reset;
```

```
  Not(y); Mix(y); Mix(x); /* |+> * |-> */
```

```
  U(n,x,y);
```

```
  Mix(x);
```

```
  measure x,r; print r;
```

```
}
```

```
qc1> Deutsch(0)
```

```
0
```

```
[2/32] 0.70711 |0,0> - 0.70711 |0,1>
```

```
qc1> Deutsch(1)
```

```
0
```

```
[2/32] -0.70711 |0,0> + 0.70711 |0,1>
```

```
qc1> Deutsch(2)
```

```
1
```

```
[2/32] 0.70711 |1,0> - 0.70711 |1,1>
```

```
qc1> Deutsch(3)
```

```
1
```

```
[2/32] -0.70711 |1,0> + 0.70711 |1,1>
```

qc1> exit

Мы видим, что алгоритм действительно выдаёт 0 для первых двух функций  $f(x)$  (константы 0 и 1) и 1 для остальных (уравновешенные функции  $x$  и  $1 - x$ ).

### 1.3. Пример, в котором классический алгоритм имеет экспоненциальную сложность, а квантовый – линейную

Теперь мы рассмотрим пример задачи, для решения которой на классическом компьютере требуется экспоненциально большое время, а на квантовом – линейное [4]. Сама по себе эта задача очень искусственная, и интереса не представляет. Есть и более интересные задачи, для которых известны квантовые алгоритмы, более эффективные, чем классические (хотя таких задач и не очень много). Но этот алгоритм легче всего понять.

Рассмотрим обобщение задачи из предыдущего параграфа. А именно, рассмотрим функции  $f(x)$ , отображающие  $n$ -битовые целые числа  $x$  в 1 бит:

$$f : \{0, 1\}^n \rightarrow \{0, 1\}.$$

Кто-то загадал функцию  $f$ . Разрешено загадывать только функции, принадлежащие одному из 2 классов:

- Константы (0 или 1)
- “Уравновешенные” функции, принимающие значение 0 в половине точек  $x$  и 1 в другой половине

Мы хотим угадать, к которому из этих двух классов принадлежит функция  $f$ .

В классическом случае, мы вычисляем  $f(0), f(1), \dots$ . В худшем случае, все время будет получаться одно и то же, скажем, 0. После  $2^{n-1}$  вычислений функции мы всё ещё не будем знать, константа это или уравновешенная функция.

Теперь пусть у нас есть квантовая функция

$$\hat{U}_f |x, y\rangle = |x, y \oplus f(x)\rangle.$$

Применим её к  $|x\rangle \otimes |-\rangle$ :

$$\hat{U}_f |x\rangle \otimes |-\rangle = (-1)^{f(x)} |x\rangle \otimes |-\rangle.$$

Теперь применим её к  $|+\rangle \otimes |-\rangle$ , где

$$|+\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle, \quad N = 2^n.$$

Получится

$$\hat{U}_f |+\rangle \otimes |-\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} (-1)^{f(x)} |x\rangle \otimes |-\rangle.$$

Чтобы привести  $|y\rangle$  в состояние  $|-\rangle$ , применяем  $\text{Not}(y)$ ;  $\text{Mix}(y)$ ; . Чтобы привести  $|x\rangle$  в состояние  $|+\rangle$ , применяем  $\text{Mix}(x)$ ; , то есть  $\text{Mix}(x[0])$ ;  $\text{Mix}(x[1])$ ; ... Если наша функция – константа, то

$$\hat{U}_f |+\rangle \otimes |-\rangle = (-1)^{f(0)} |+\rangle \otimes |-\rangle,$$

то есть после применения  $\hat{U}_f$  регистр  $x$  будет в состоянии  $|+\rangle$ . Применим опять  $\text{Mix}(x)$ ; . Поскольку  $\hat{M}^2 = 1$ , получится  $|00\dots 0\rangle$ . Если наша функция уравновешенная, то в сумме половина членов будет со знаком  $+$ , а половина – со знаком  $-$ . Поэтому состояние регистра  $x$  будет ортогонально к  $|+\rangle$ . После применения  $\text{Mix}(x)$ ; получится состояние, ортогональное к  $|00\dots 0\rangle$ , то есть суперпозиция любых состояний типа  $|10\dots 1\rangle$ , кроме состояния  $|00\dots 0\rangle$ . Все вычисления  $f(x)$  происходят одновременно, когда начальное состояние – суперпозиция разных  $|x\rangle$ .

QCL Quantum Computation Language (32 qubits, seed 1081610812)  
[0/32] 1 |0>

Пусть регистр  $x$  состоит из 3 бит.  
qcl> qureg x[3]; qureg y[1]; int r;

Оператор  $\hat{U}_f$  для 3 функций  $f(x)$ :

- $n = 0 - f(x) = 0$  ( $\hat{U}_f = 1$ );
- $n = 1 - f(x) = x_0$
- $n = 2 - f(x) = x_0 \oplus x_1 \oplus x_2$

Первая из них – константа, остальные – уравновешенные функции.

```
qcl> procedure U(int n, qureg x, qureg y)
  { if n==1 { x[0]->y; } /* f(x)=x[0] */
    else { if n==2 { x[0]->y; x[1]->y; x[2]->y; }}
          /* f(x)=x[0] xor x[1] xor x[2] */
```

Приготовим состояние  $|+\rangle \otimes |-\rangle$ :

```
qc1> Not(y)
[4/32] 1 |000, 1>
```

```
qc1> Mix(y)
[4/32] 0.70711 |000, 0> - 0.70711 |000, 1>
```

```
qc1> Mix(x)
[4/32] 0.25 |000, 0> + 0.25 |001, 0> + 0.25 |010, 0> + 0.25 |011, 0> +
0.25 |100, 0> + 0.25 |101, 0> + 0.25 |110, 0> + 0.25 |111, 0> - 0.25 |000, 1> -
0.25 |001, 1> - 0.25 |010, 1> - 0.25 |011, 1> - 0.25 |100, 1> - 0.25 |101, 1> -
0.25 |110, 1> - 0.25 |111, 1>
```

Применим  $\hat{U}_f$  для  $f(x) = 0$ :

```
qc1> U(0, x, y); dump
STATE: 4 / 32 qubits allocated, 28 / 32 qubits free
0.25 |0000> + 0.25 |0001> + 0.25 |0010> + 0.25 |0011> + 0.25 |0100> +
0.25 |0101> + 0.25 |0110> + 0.25 |0111> - 0.25 |1000> - 0.25 |1001> -
0.25 |1010> - 0.25 |1011> - 0.25 |1100> - 0.25 |1101> - 0.25 |1110> -
0.25 |1111>
```

Применим преобразование Адамара к регистру  $x$ :

```
qc1> Mix(x)
[4/32] 0.70711 |000, 0> - 0.70711 |000, 1>
```

Произведём измерение регистра  $x$  и напечатаем результат:

```
qc1> measure x, r
[4/32] 0.70711 |000, 0> - 0.70711 |000, 1>
```

```
qc1> print r
0
```

Получился 0, как и должно быть для функции  $f(x)$ , являющейся константой. Теперь очистим память и повторим всё для  $f(x) = x_0$ :

```
qc1> reset
[4/32] 1 |000, 0>
```

```
qc1> Not(y); Mix(y); Mix(x)
[4/32] 0.25 |000, 0> + 0.25 |001, 0> + 0.25 |010, 0> + 0.25 |011, 0> +
0.25 |100, 0> + 0.25 |101, 0> + 0.25 |110, 0> + 0.25 |111, 0> - 0.25 |000, 1> -
0.25 |001, 1> - 0.25 |010, 1> - 0.25 |011, 1> - 0.25 |100, 1> - 0.25 |101, 1> -
0.25 |110, 1> - 0.25 |111, 1>
```

```
qc1> U(1, x, y)
```

```
[4/32] 0.25 |000,0> - 0.25 |001,0> + 0.25 |010,0> - 0.25 |011,0> +
0.25 |100,0> - 0.25 |101,0> + 0.25 |110,0> - 0.25 |111,0> - 0.25 |000,1> +
0.25 |001,1> - 0.25 |010,1> + 0.25 |011,1> - 0.25 |100,1> + 0.25 |101,1> -
0.25 |110,1> + 0.25 |111,1>
```

```
qc1> Mix(x)
```

```
[4/32] 0.70711 |001,0> - 0.70711 |001,1>
```

```
qc1> measure x,r; print r
```

```
1
```

```
[4/32] 0.70711 |001,0> - 0.70711 |001,1>
```

Результат ненулевой, как и должно быть для уравновешенной функции. Повторим ещё раз для  $f(x) = x_0 \oplus x_1 \oplus x_2$ :

```
qc1> reset
```

```
[4/32] 1 |000,0>
```

```
qc1> Not(y); Mix(y); Mix(x)
```

```
[4/32] 0.25 |000,0> + 0.25 |001,0> + 0.25 |010,0> + 0.25 |011,0> +
0.25 |100,0> + 0.25 |101,0> + 0.25 |110,0> + 0.25 |111,0> - 0.25 |000,1> -
0.25 |001,1> - 0.25 |010,1> - 0.25 |011,1> - 0.25 |100,1> - 0.25 |101,1> -
0.25 |110,1> - 0.25 |111,1>
```

```
qc1> U(2,x,y)
```

```
[4/32] 0.25 |000,0> - 0.25 |001,0> - 0.25 |010,0> + 0.25 |011,0> -
0.25 |100,0> + 0.25 |101,0> + 0.25 |110,0> - 0.25 |111,0> - 0.25 |000,1> +
0.25 |001,1> + 0.25 |010,1> - 0.25 |011,1> + 0.25 |100,1> - 0.25 |101,1> -
0.25 |110,1> + 0.25 |111,1>
```

```
qc1> Mix(x)
```

```
[4/32] 0.70711 |111,0> - 0.70711 |111,1>
```

```
qc1> measure x,r; print r
```

```
7
```

```
[4/32] 0.70711 |111,0> - 0.70711 |111,1>
```

Результат ненулевой, как и должно быть.

```
qc1> exit
```

Какова сложность этого алгоритма? `Mix(x)`; означает `Mix(x[0])`; `Mix(x[1])`; ... Мы делаем `Mix(x)`; дважды, то есть  $2n$  элементарных операций. Сложность линейная вместо экспоненциальной!

Все известные классические алгоритмы факторизации больших целых чисел ( $n$ -битовых) имеют сложность, растущую быстрее любой степени

*n*. В 1994 году Питер Шор [5] предложил квантовый алгоритм, имеющий полиномиальную сложность. Многие популярные программы (ssh, rgr и т.д.) используют криптосистему с публичными ключами RSA [6]. Публичный ключ содержит большое целое число; если факторизовать его, можно легко найти приватный ключ. Безопасность интернета держится на том, что известные алгоритмы не позволяют факторизовать такие числа за обозримое время. Если появится устройство, позволяющее факторизовать такие числа достаточно быстро, то тот, у кого оно есть, сможет читать зашифрованные письма, подсматривать пароли и т.д.

Эмулятор QCL делает вычисления с разными компонентами состояний последовательно, а не одновременно, как настоящий квантовый компьютер. Поэтому его скорость экспоненциально убывает с ростом числа битов. Естественно, эмулятор не годится для того, чтобы получать выгоду от быстрых квантовых алгоритмов; для этого нужен настоящий квантовый компьютер.

## 2. Квантовая криптография

Раньше в литературе по криптографии говорили что-нибудь вроде: “А посылает сообщение В”. В последнее время вместо этого принято говорить: “Алиса посылает сообщение Бобу”. В трёхсторонних обменах мнениями участвует также Сесиль (девичья фамилия С). Есть также зловредный персонаж Ева, которая занимается подслушиванием (eavesdropping).

Алиса и Боб хотят передавать друг другу сообщения таким образом, что, если их перехватит Ева, то она не сможет их расшифровать. Для этого существует надёжный метод – использование одноразовых блокнотов. Допустим, что у Алисы и Боба есть одинаковые блокноты, на страничках которых написана случайная последовательность 0 и 1. Больше ни у кого во всём мире этой последовательности нет. Сообщения – это файлы, т.е. последовательности битов. Алиса прибавляет по модулю 2 (т.е. делает XOR) к битам своего файла биты из блокнота. И использованные странички она вырывает и сжигает. Зашифрованный файл она посылает Бобу по email. Боб прибавляет по модулю 2 (т.е. делает XOR) к полученным битам биты из своего блокнота. И использованные странички он вырывает и сжигает. После этого он читает расшифрованный файл. Если email перехватила Ева, она никак не сможет расшифровать сообщение (предполагается, что одна и та же последовательность никогда не используется дважды).



Если Алиса работает резидентом в тылу врага, то передать ей из рук в руки новый блокнот взамен закончившегося может быть затруднительно. Посылать эту случайную последовательность битов по классическому кабелю любой природы опасно – Ева может скопировать их, и сможет расшифровывать сообщения.

Простой метод, основанный на квантовой механике, предложен в [7]. Пусть кто-то создаёт EPR-состояния пар частиц со спином  $\frac{1}{2}$  с суммарным спином 0. Одну частицу посылают Алисе, а другую Бобу. Алиса кидает монетку и решает, измерять ей  $z$ -компоненту спина или  $x$ -компоненту; Боб делает то же самое. Они вывешивают на своих home-page на WWW (где все могут видеть) последовательность орлов и решек, и выбрасывают бесполезные результаты измерений, в которых они измерили разные компоненты (примерно 50%). Остальные результаты дают случайную последовательность 0 и 1, одинаковую у Алисы и Боба.

Теперь допустим, что Ева перехватывает частицы, посылаемые Алисе. Она не знает, какую компоненту спина ей нужно мерить, и в 50% случаев меряет не ту. В этих случаях Алиса и Боб получают не EPR-пару, а нескоррелированную пару спинов. То есть примерно 25% результатов Алисы и Боба отличаются. Алиса может послать Бобу открытым текстом некоторую часть своих 0 и 1 (эту часть они использовать не будут). Если Боб обнаружит 25% расхождений со своими, то они будут знать, что их подслушивали. Если же нет, то в это время их не подслушивали, и есть надежда, что и в остальное время тоже.

```
QCL Quantum Computation Language (32 qubits, seed 1081653831)
[0/32] 1 |0>
```

Пара квантовых бит:  
`qc1> qureg x[2]`

Процедура, создающая EPR-состояние с суммарным спином 0:

```
qc1> operator EPR(qureg x)
{ Not(x[0]); Mix(x[0]); x[0]->x[1]; Not(x[0]); }
```

Проверим:

```
qc1> EPR(x)
[2/32] 0.70711 |01> - 0.70711 |10>
```

Эта процедура описывает генерацию и измерение  $n$  EPR-пар. Если у Алисы выпал орёл, она записывает это ( $m_A=1$ ) и поворачивает свой спин на  $\pi/2$  вокруг оси  $x$ . Результат измерения она записывает в  $r_A$ . Аналогично,

у Боба тип измерения записывается в mB, а результат в rB. Если mA и mB совпадают, результаты принимаются.

```

qcl> procedure Crypto(int n, boolean Eve)
  { int mA; int rA; int mB; int rB; int i;
    for i=1 to n
      { reset; EPR(x);
        /* Eve */
        if Eve { measure x[0]; }
        /* Alice */
        if random()>0.5 { mA=1; RotX(pi/2,x[0]); }
        else { mA=0; }
        measure x[0],rA;
        /* Bob */
        if random()>0.5 { mB=1; RotX(pi/2,x[1]); }
        else { mB=0; }
        measure x[1],rB;
        rB=1-rB;
        /* Comparison */
        if mA==mB { print rA,rB; }
      }
  }
}

```

Если нет подслушивания, результаты должны совпадать:

```

qcl> Crypto(100,false)
00  11  11  11  00  11  11  11
00  11  00  11  11  00  00  00
11  11  00  11  11  00  11  00
00  11  00  00  11  11  11  00
11  11  11  00  00  11  11  11
11  00  00  00  11  00  11
[2/32] 1 |01>

```

Если параметр Eve есть true, то Ева измеряет z-компоненту спина частиц, посылаемых Алисе. В этом случае примерно в 25% результатов обнаружатся расхождения:

```

qcl> Crypto(100,true)
00  00  10  00  00  10  01  11
00  11  11  10  00  11  01  11
01  01  10  00  11  01  11  00
11  01  00  00  00  00  01  11

```

```

1 1      1 1      1 1      1 1      1 1      0 0      0 0      0 0
0 0      1 1
[2/32] -i |11>
qc1> exit

```

Похожая идея – деньги, защищённые от подделки принципами квантовой механики [8]. Выпускающий их банк наносит на каждую банкноту номер, а также встраивает несколько десятков спинов  $\frac{1}{2}$ . Каждый из них имеет определённую проекцию на  $z$  или  $x$ . В базе данных банка для каждого номера банкноты хранится информация о том, какие спины имеют определённую  $\hat{s}_z$ , какие –  $\hat{s}_x$ , и чему именно они равны. (В исходной формулировке речь шла про фотоны, запертые между параллельными зеркалами и имеющие определённую линейную либо циркулярную поляризацию; это, конечно, несущественно.) Служащий банка может проверить подлинность банкноты, измерив нужные проекции спинов. Фальшивомонетчик не знает, какую проекцию измерять для какого спина, и в половине случаев будет измерять не ту. Поэтому он не только не сделает хорошей копии (которая выдержала бы проверку), но и испортит оригинал.

### 3. Квантовая телепортация

Пусть у Алисы есть частица со спином  $\frac{1}{2}$ , находящаяся в некотором состоянии

$$|x\rangle = \alpha|0\rangle + \beta|1\rangle.$$

Кто-то готовит пару частиц со спином  $\frac{1}{2}$  в EPR-состоянии

$$|y\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}},$$

и посылает первую из них Алисе, а вторую Бобу. Тогда можно привести частицу Боба в точно то же состояние  $\alpha|0\rangle + \beta|1\rangle$  (при этом бит Алисы уже не будет в этом состоянии) [9].

Итак, начальное состояние

$$\begin{aligned}
& (\alpha|0\rangle + \beta|1\rangle) \otimes \frac{|00\rangle + |11\rangle}{\sqrt{2}} \\
&= \frac{1}{\sqrt{2}} [\alpha (|000\rangle + |011\rangle) + \beta (|100\rangle + |111\rangle)].
\end{aligned}$$

Алиса применяет к своей паре бит  $\hat{C}$ :

$$\frac{1}{\sqrt{2}} [\alpha (|000\rangle + |011\rangle) + \beta (|110\rangle + |101\rangle)] ,$$

затем к первому биту  $\hat{M}$ :

$$\begin{aligned} & \frac{\alpha}{2} (|000\rangle + |100\rangle + |011\rangle + |111\rangle) \\ & + \frac{\beta}{2} (|010\rangle - |110\rangle + |001\rangle - |101\rangle) \\ & = \frac{1}{2} (|00\rangle \otimes |x\rangle + |01\rangle \otimes \sigma_x |x\rangle + |10\rangle \otimes \sigma_z |x\rangle - i |11\rangle \otimes \sigma_y |x\rangle) , \end{aligned}$$

где

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

– матрицы Паули (они унитарны). Алиса измеряет 2 своих бита, и получает классическое двухбитовое число (0, 1, 2 или 3 с равными вероятностями). Она посылает это число Бобу по классическому каналу (скажем, по email). Боб действует на свой бит оператором 1,  $\sigma_x$ ,  $\sigma_z$  или  $-i\sigma_y$ , соответственно.

```
QCL Quantum Computation Language (32 qubits, seed 1087133710)
[0/32] 1 |0>
```

```
x – бит Алисы; y – пара бит для EPR-состояния
qc1> qureg x[1]; qureg y[2]; int r
```

Процедура, создающая EPR-состояние:

```
qc1> operator EPR(qureg y)
    { Mix(y[0]); y[0]->y[1]; }
```

Проверим:

```
qc1> EPR(y)
[3/32] 0.70711 |0,00> + 0.70711 |0,11>
```

```
qc1> reset
[3/32] 1 |0,00>
```

Приведём бит Алисы в случайное состояние:

```
qc1> randomize(x)
[3/32] (-0.35679 - 0.77939i) |0,00> + (0.50955 + 0.074842i) |1,00>
```

Приготовим EPR-состояние пары  $y$ :

```
qc1> EPR(y)
[3/32] (-0.25229 - 0.55111i) |0,00> + (0.36031 + 0.052921i) |1,00> +
(-0.25229 - 0.55111i) |0,11> + (0.36031 + 0.052921i) |1,11>
```

Алиса применяет CNot к своим 2 битам:

```
qc1> x->y[1]
[3/32] (-0.25229 - 0.55111i) |0,00> + (0.36031 + 0.052921i) |1,01> +
(0.36031 + 0.052921i) |1,10> + (-0.25229 - 0.55111i) |0,11>
```

Алиса применяет преобразование Адамара к биту  $x$ :

```
qc1> Mix(x)
[3/32] (-0.1784 - 0.3897i) |0,00> + (-0.1784 - 0.3897i) |1,00> +
(0.25478 + 0.037421i) |0,01> + (-0.25478 - 0.037421i) |1,01> +
(0.25478 + 0.037421i) |0,10> + (-0.25478 - 0.037421i) |1,10> +
(-0.1784 - 0.3897i) |0,11> + (-0.1784 - 0.3897i) |1,11>
```

Алиса измеряет свои 2 бита:

```
qc1> measure y[1]&x,r; print r
1
[3/32] (0.50955 + 0.074842i) |0,10> + (-0.35679 - 0.77939i) |0,11>
```

Боб действует на свой бит  $y[0]$  в соответствии с полученной информацией:

```
qc1> if r==1 { X(y[0]); }
      else { if r==2 { Z(y[0]); }
              else { if r==3 { RotY(-pi,y[0]); }}}
[3/32] (-0.35679 - 0.77939i) |0,10> + (0.50955 + 0.074842i) |0,11>
```

Теперь бит  $y[0]$ , находящийся у Боба, находится в состоянии  $|x\rangle$ .

Процедура Tele повторяет это  $n$  раз. Она печатает состояние перед телепортацией и после неё.

```
qc1> procedure Tele(int n)
  { int i;
    for i=1 to n
      { reset;
        randomize(x); dump;
        EPR(y);
        /* Alice */
        x->y[1]; Mix(x);
```

```

    measure y[1]&x,r; print r;
    /* Bob */
    if r==1 { X(y[0]); }
    else { if r==2 { Z(y[0]); }
    else { if r==3 { RotY(-pi,y[0]); }}}
    dump;
  }
}

```

qc1> Tele(10)

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(0.25304 + 0.34095i) |000\rangle + (-0.41323 + 0.80558i) |001\rangle$   
 3

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(0.25304 + 0.34095i) |101\rangle + (-0.41323 + 0.80558i) |111\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(0.25496 - 0.77909i) |000\rangle + (0.38057 + 0.42799i) |001\rangle$   
 0

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(0.25496 - 0.77909i) |000\rangle + (0.38057 + 0.42799i) |010\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(0.95379 + 0.28287i) |000\rangle + (0.06279 - 0.079523i) |001\rangle$   
 2

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(0.95379 + 0.28287i) |001\rangle + (0.06279 - 0.079523i) |011\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(-0.74984 + 0.16696i) |000\rangle + (0.27132 + 0.57987i) |001\rangle$   
 2

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(-0.74984 + 0.16696i) |001\rangle + (0.27132 + 0.57987i) |011\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(0.19238 - 0.22205i) |000\rangle + (0.61016 + 0.73579i) |001\rangle$   
 2

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(0.19238 - 0.22205i) |001\rangle + (0.61016 + 0.73579i) |011\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
 $(-0.074875 - 0.17213i) |000\rangle + (-0.98212 - 0.014345i) |001\rangle$

3

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $-0.074875 - 0.17213i$ )  $|101\rangle + (-0.98212 - 0.014345i) |111\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $-0.13448 - 0.52058i$ )  $|000\rangle + (-0.68431 + 0.49257i) |001\rangle$

2

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $-0.13448 - 0.52058i$ )  $|001\rangle + (-0.68431 + 0.49257i) |011\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $-0.042565 - 0.40042i$ )  $|000\rangle + (-0.27303 - 0.87367i) |001\rangle$

3

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $-0.042565 - 0.40042i$ )  $|101\rangle + (-0.27303 - 0.87367i) |111\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $0.0019455 + 0.0098884i$ )  $|000\rangle + (0.90142 + 0.43282i) |001\rangle$

1

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $0.0019455 + 0.0098884i$ )  $|100\rangle + (0.90142 + 0.43282i) |110\rangle$

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $0.77744 - 0.49842i$ )  $|000\rangle + (-0.11673 - 0.36542i) |001\rangle$

2

STATE: 3 / 32 qubits allocated, 29 / 32 qubits free  
( $0.77744 - 0.49842i$ )  $|001\rangle + (-0.11673 - 0.36542i) |011\rangle$

[3/32] ( $0.77744 - 0.49842i$ )  $|1, 00\rangle + (-0.11673 - 0.36542i) |1, 01\rangle$   
qc1> exit

На первый взгляд это кажется странным. Состояние спина у Алисы описывается двумя комплексными числами  $\alpha$  и  $\beta$ , в которых, вообще говоря, бесконечно много значащих цифр (десятичных или двоичных). Алиса посылает Бобу всего 2 бита информации, и это состояние передаётся спину у Боба. На самом деле Алиса не знала состояния спина (т.е.  $\alpha$  и  $\beta$ ), так что никакой передачи информации здесь нет (если бы она знала, она могла бы просто послать Бобу email с инструкцией, как это состояние приготовить).

Если представить себе возможность телепортации квантового состояния более сложных систем, чем спин  $\frac{1}{2}$ , например, человека, то это бы выглядело так. В пункте А человек заходит в телепортационную кабину. Результаты измерений передаются в пункт В по email. Там из телепорта-

ционной кабины выходит точно такой же человек. В первой кабине вместо человека остаётся груда мусора (потому что состояние первой системы не сохраняется).

## Список литературы

- [1] B. Ömer, *Structured Quantum Programming*, Ph. D. Thesis, Technical University of Vienna (2003); <http://tph.tuwien.ac.at/~oemer/>
- [2] J. van der Hoeven, *GNU T<sub>E</sub>Xmacs*, <http://www.texmacs.org/>
- [3] A.G. Grozin, *T<sub>E</sub>Xmacs interfaces to Maxima, MuPAD and REDUCE*, Proc. 5 Int. workshop on *Computer algebra and its applications to physics*, Dubna, JINR E5,11-2001-279; cs.SC/0107036
- [4] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computer*, Proc. Roy. Soc. A **439** (1992) 553
- [5] P.W. Shor, *Algorithms for quantum computation: Discrete logarithms and factoring*, Proc. 35 Annual Symposium on *Foundations of Computer Science*, IEEE Press (1994);  
*Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Computing **26** (1997) 1484
- [6] R. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Comm. ACM **21** (2) (1978) 120
- [7] C.H. Bennett, G. Brassard, and N.M. Mermin, *Quantum cryptography without Bell's theorem*, Phys. Rev. Lett. **68** (1992) 557
- [8] S. Wiesner, *Conjugate coding*, написано в 1970, опубликовано в SIGACT News **15** (1) (1983) 78
- [9] C.H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W.K. Wootters, *Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels*, Phys. Rev. Lett. **70** (1993) 1895