

# NDQJava语言处理系统量子汇编及解释程序\*

焦阳, 吴楠, 宋方敏

(南京大学计算机软件新技术国家重点实验室, 南京, 210093)

**摘要:** 量子计算被认为是有可能超越经典计算的一种新型计算模型, 目前国内外已有大量相关实验和理论工作围绕量子计算及量子计算机而展开。南京大学计算机系量子计算和量子信息研究小组在量子计算领域中的量子程序设计语言方向上开展了一些工作, 设计了一种量子程序设计语言 NDQJava 并在经典计算机上对其处理系统加以模拟实现。为了更好地描述量子计算的逻辑流程并和物理模型相对应, 在 NDQJava 处理系统的设计和实现过程中定义了量子汇编及机器语言。量子汇编语言作为高级量子程序设计语言和量子器件之间的接口语言, 其设计必须考虑完备、简明、易用三方面的因素。为了模拟实现 NDQJava 处理系统亦定义了量子机器语言并通过软件解释执行。文中在定义了量子汇编及机器语言后着重描述了 NDQJava 处理系统中汇编及解释程序的设计思想与实现方法, 对其中的若干关键问题给出了相应图表及源程序片段加以阐明。

**关键词:** 量子计算, 程序设计语言, 汇编程序, 解释程序

**中图法分类号:** TP301 **文献标识码:** A

---

\* 本文得到江苏省自然科学基金 No.BK2007138 资助

# Quantum assembler and interpreter of NDQJava processing system \*

Jiao Yang, Wu Nan, Song Fang-Min

(State Key Laboratory for Novel Software Technology,  
Nanjing University, Nanjing, 210093, China)

**Abstract:** Quantum computing is considered possible for being more powerful than classic computing model. So far there are a lot of works have been done in the fields related to quantum computing. The QCI (Quantum Computing and Quantum Information) group of Department of Computer Science and Technology in Nanjing University are working on the design and implementation of quantum programming language. At present a new kind of quantum programming language named NDQJava have been designed and implemented in a simulation system based on classic computer. In this process quantum assembly and machine language are defined. As the interface between high level programming languages and quantum devices, the design of quantum assembly language has to take completeness, simplicity and user-friendly into consideration. In order to simulate the quantum computing process via software, a quantum machine language is also defined and interpreted by the interpreter running on classic computers. This article defines the quantum assembly language and corresponding machine language used by NDQJava processing system at first. Furthermore, it focuses on describing the design and implementation of its assembler and interpreter. Finally, it tries to explain some key issues in detail with diagrams and source program pieces.

**Key words:** quantum computation, programming language, assembler, interpreter

---

\* Supported by Natural Science Foundation of Jiangsu Province of China under Grant No.BK2007138

# 1 量子程序设计语言NDQJava及其处理系统

**1.1 NDQJava 语言** NDQJava 语言<sup>[1]</sup>是一种基于 Java 的新型量子程序设计语言。为了描述量子计算，该语言引入了量子数据类型 `qtype`。它是一种包含经典语言成分与量子语言成分的命令式混成语言。

**1.2 NDQJava 处理系统** NDQJava 处理系统<sup>[2]</sup>是一个先编译后解释的系统，其中尽可能将 NDQJava 源程序的经典部分与量子部分分别处理，经典部分交由 Java 编译程序进行编译并由 Java 解释程序解释执行；量子部分则编译成量子汇编语言码，通过函数调用交由量子汇编及解释程序汇编为量子机器语言后在经典计算机上模拟解释执行。

## 2 量子汇编及机器语言

虽然目前通用量子计算机尚未问世<sup>[3-5]</sup>，但是为了设计并在经典计算机上模拟实现 NDQJava 处理系统，我们针对设想的量子计算机<sup>[6]</sup>设计了相应的量子汇编语言以及量子机器语言。NDQJava 处理系统将源程序的量子部分编译为一列量子汇编指令，执行时通过量子汇编程序汇编为一列相应量子机器指令后由量子解释程序在经典计算机上模拟解释执行。

**2.1 量子基本指令集** 基本指令集的选取须遵从完备、简明、易用三原则。

- 完备指基本指令集可完成所有量子操作；
- 简明指只选取使用频率较高的操作作为基本指令；
- 易用指所选取之指令集易于书写量子算法程序。

从以上三原则出发，并且参考国外相关工作成果<sup>[7-8]</sup>选定了：

表 1 量子操作集

Table1 Quantum Operation Set

指令名	指令说明
Nop	空操作
H	阿达玛变换
I	等同变换
X	X 变换
Z	Z 变换
S	S 变换
T	真条件置位
F	假条件置位
Swap	对换
Cnot	受控非变换
RotX	绕 X 轴条件旋转变换

RotY	绕 Y 轴条件旋转变换
RotZ	绕 Z 轴条件旋转变换
Phase	条件相位变换
Measure	测量操作

作为量子汇编及机器语言所采用的基本指令集。出于完备性<sup>[9]</sup>的考虑其中必须包含受控非变换、绕 X、Y、Z 轴的旋转变换以及测量操作，理论上可由受控非变换及旋转变换构作出任意酉变换操作。空操作的设置出于书写两地址指令的需要。而引入阿达玛变换、等同变换、X 变换、Z 变换、S 变换、对换、真条件置位、假条件置位以及条件相位变换，则是考虑到简明性、易用性等方面，这些为数不多的操作在现有量子算法<sup>[10-12]</sup>中出现概率颇高，提供相应指令便于使程序人员能写出更为紧凑的程序。

**2.2 量子机器语言指令格式** 在量子指令集的所有指令中，一部分为一地址指令，另一部分为两地址指令，所以量子汇编及机器语言采用变长指令。为了便于系统处理量子汇编及机器语言指令，规定两地址指令的长度为一地址指令的 2 倍。量子机器语言格式规定如下：

每条一地址机器指令以二进制书写长 16 位，其中前 4 位为操作码，中间 4 位为量子存储器段编号，最后 8 位用于标示所操作之量子位。

表 2 量子机器指令集

Table2 Quantum machine code set

指令名	指令格式
Nop	'0000' 'XXXX' 'XXXXXXXXXX'
H	'0001' 'XXXX' 'XXXXXXXXXX'
I	'0010' 'XXXX' 'XXXXXXXXXX'
X	'0011' 'XXXX' 'XXXXXXXXXX'
Z	'0100' 'XXXX' 'XXXXXXXXXX'
S	'0101' 'XXXX' 'XXXXXXXXXX'
T	'1011' 'XXXX' 'XXXXXXXXXX'
F	'1100' 'XXXX' 'XXXXXXXXXX'
Phase	'1101' 'XXXX' 'XXXXXXXXXX'

Swap、Cnot、RotX、RotY、RotZ、Measure 为两地址指令，其长度为一地址指令的 2 倍，其中后 16 位旨在提供第二操作数。

表 2 (续) 量子机器指令集

Table2 (cont.) Quantum machine code set

指令名	指令格式
Swap	'0110' 'XXXX' 'XXXXXXXXXX' '0000' 'XXXX' 'XXXXXXXXXX'
Cnot	'0111' 'XXXX' 'XXXXXXXXXX' '0000' 'XXXX' 'XXXXXXXXXX'
RotX	'1000' 'XXXX' 'XXXXXXXXXX' '0000' 'XXXX' 'XXXXXXXXXX'
RotY	'1001' 'XXXX' 'XXXXXXXXXX' '0000' 'XXXX' 'XXXXXXXXXX'
RotZ	'1010' 'XXXX' 'XXXXXXXXXX' '0000' 'XXXX' 'XXXXXXXXXX'

**2.3 量子汇编语言指令格式** 与量子机器语言指令格式对应,量子汇编语言指令也分为一地址指令和两地址指令。指令格式规定如下:

每条一地址汇编指令使用 4 个字符,其中第一个字符为操作码,第二、三、四个字符为操作数,第二个字符为 'M' 表示量子存储单元,第三、四个字符为 '00' - '99' 用以标示所要进行操作的量子位。

表 3 量子汇编指令集

Table3 Quantum assembly instruction set

指令名	指令格式
Nop	'N' 'MXX'
H	'H' 'MXX'
I	'I' 'MXX'
X	'X' 'MXX'
Z	'Z' 'MXX'
S	'S' 'MXX'
T	'T' 'MXX'
F	'F' 'MXX'
Phase	'A' 'MXX'

Swap、Cnot、RotX、RotY、RotZ、Measure 为两地址指令,其格式为:

表.3 (续) 量子汇编指令集

Table.3 (Cont.) Quantum assembly instruction set

指令名	指令格式
Swap	'W' 'MXX' 'N' 'MXX'
Cnot	'C' 'MXX' 'N' 'MXX'
RotX	'O' 'MXX' 'N' 'MXX'
RotY	'P' 'MXX' 'N' 'MXX'
RotZ	'Q' 'MXX' 'N' 'MXX'
Measure	'M' 'MXX' 'N' 'MXX'

### 3 设计思想

**3.1 接口部分** 量子汇编及解释程序是 NDQJava 处理系统的组成部分,其功能为处理 Java 处理系统所不能处理的量子部分。故量子汇编及解释程序需要与 NDQJava 处理系统的其余部分交互,交互工作通过相应接口完成。接口设计的任务在于明确相应的功能、输入及输出。

(1.) 功能:接收量子汇编指令,由量子汇编程序汇编成相应机器指令,再由解释程序对机器指令解释执行。若是酉变换操作,则改变各个量子态的概率幅,若是测量操作,回送测量结果。

(2.) 输入:量子汇编指令,包括非测量类指令和测量类指令。测量类指令考虑到现实需要,又

分为全测量指令和单位测量指令两种。

(3.) 输出：当输入指令为量子测量操作时，回送测量结果，否则无回送值。

对非测量类指令，只需接收量子汇编指令并且汇编为机器指令后交由解释程序解释执行。对测量类指令，除了将其汇编并解释执行外，还须将测量结果回送。因此设计两种接口函数，即无回送值的、针对非测量类指令的接口函数，以及有回送值的、针对测量类指令的接口函数。

**3.2 汇编程序** 接收量子汇编指令后，由量子汇编程序将汇编指令汇编成相应机器指令。这一过程的工作是，首先操作码转换，其次操作数转换，最后将汇编得到的机器指令交给解释程序解释执行。之所以采取先汇编再解释的处理策略而没有直接对量子机器指令执行主要是基于以下考虑：首先，目前尚无实际可用的量子计算机，故只能采取解释执行的方式通过经典计算机模拟相应动作。然而，一旦量子计算机被制造出来并投入使用，只需要修改相应汇编程序，将 NDQJava 处理系统定义的量子汇编指令汇编为量子计算机所定义并使用的机器指令即可在量子计算机上运行。所以，虽然目前看来这一过程相对繁琐，但使得 NDQJava 处理系统具有较高易移植性，便于今后将 NDQJava 处理系统移植到真正的量子计算机上。

根据前文 NDQJava 处理系统所定义的量子汇编语言和量子机器语言，将汇编语言的操作码一一对应到机器语言的操作码，同时将量子汇编指令中操作数部分转换为对应量子机器指令的操作数。在完成以上工作之后，量子汇编指令已经被汇编为量子机器指令，接下来只需将其转交量子解释程序解释执行即可。

汇编程序流程如图 1 所示。

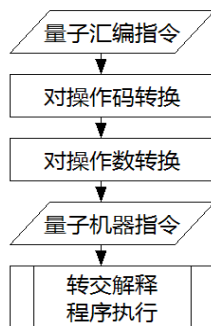


图 1 量子汇编指令处理流程图

Fig.1 Quantum assembly code processing flow

**3.3 解释程序** 解释程序接收量子机器指令后对之进行解释并且模拟执行相应动作。 对非测量类指令的模拟，核心在于量子态的变化。若一个酉变换对应的酉矩阵为  $A$ ，变化前各量子态的概率幅为列向量  $S$ ，变化后对应量子态概率幅为列向量  $S'$ ，即

$$S' = A \times S$$

对不同酉变换操作须选取相应酉矩阵，而变换过程以矩阵乘法加以模拟 例如单量子位处于基态  $|0\rangle$  时可用列向量  $(1, 0)^T$  表示，Hadamard 变换的酉矩阵表示为

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

则对此单量子位做  $H$  变换可由下式描述：

$$\begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

结果表明，经过变换后该量子位由基态变为叠加态。

对测量类指令的模拟，重点在于随机性的体现，即量子系统的状态以一定概率随机塌缩至某一基态。

全测量后，整个量子系统完全塌缩至一个基态，此时该基态的概率幅为 1 即 100%，记  $P_i$  为第  $i$  个量子态测量前所对应的概率幅， $P_i'$  为测量后的概率幅，其公式如下：

$$P_i' = \begin{cases} 0 & , \text{测量结果未塌缩至 } |i\rangle \\ 1 & , \text{测量结果塌缩至 } |i\rangle \end{cases}$$

单位测量后，只有被测量位与测量结果相同的量子态的概率幅得以保留：

$$P_i' = \begin{cases} 0 & , \text{被测量位与测量结果不同} \\ P_i & , \text{被测量位与测量结果相同} \end{cases}$$

为了满足所有量子态概率幅的平方和为 1，即归一化条件，还需要做归一化操作，其公式如下：

$$P_i'' = P_i' / \sum P_i'$$

解释程序流程如图 2 所示：

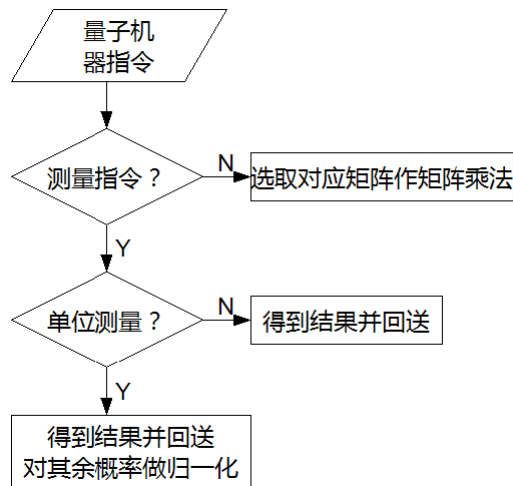


图 2 量子解释程序执行流程

Fig.2 Quantum interpreter processing flow

## 4 实现方法

**4.1 实现平台** 目前 基于 x86 指令集的 PC 是使用最为普遍的计算平台，因此，NDQJava 处理系统也在此平台上进行开发。量子汇编及解释程序的接口部分（即 API 函数首部）使用 Java 语言书写，开发工具为 Java 2 Standard Edition 1.5.0 08，集成开发环境为 eclipse3.2。与接口部分对应的函数体（即 API 函数的函数体）可以通过多种途径实现，本文将介绍采用 Java 开发的版本。

接口函数

为能与处理系统的其他部分顺利接口，汇编和解释程序部分提供了以下四个用 Java 语言书写的

API 函数

- `public static void command(String cmd);`
- `public static void command(double cmd);`
- `public static boolean measure(int reg, int bit);`
- `public static int measure(int reg);`

函数 `public static void command(String cmd)`的功能是接收以字符序列形式传入的量子汇编指令，并转交汇编程序进行后续处理。

函数 `public static void command(double cmd)`的功能是接收旋转/相位操作所需要的弧度参数。虽然设计之初在旋转/相位弧度的传输上采用了传递地址的方案，但 Java 中无法直接访问变量在内存空间

中的具体地址，故通过传递地址的方法无法实现弧度的传入。因此为传输旋转弧度提供一个 API 函数，函数名仍为 `command`，通过函数一名多用传递一个双精度型的变量，从而完成旋转/相位弧度的传入工作。

函数 `public static boolean measure(int reg, int bit)`的功能是对指定量子寄存器上的量子位进行测量并且将结果以布尔值回送，若测量结果为 1，回送 `true`；反之，回送 `false`。

函数 `public static int measure(int reg)`的功能是对指定量子寄存器上的全部量子位进行测量并且将结果以整数值回送。

**4.2 汇编程序** 汇编程序的功能是将通过接口程序接收的量子汇编指令汇编为相应的量子机器指令。在具体实现时，根据量子机器语言格式定义了以下数据结构作为量子机器指令的结构：

```
class MachineCode {
    public int op; public char reg;
    public char bitHigh;
    public char bitLow;
    public MachineCode(String cmd) {...}
    public MachineCode(int tmp_op, char tmp_reg, char tmp_bitHigh, char tmp_bitLow) {...}
    public void dump() {...}
    public MachineCode cloneMe() {...}
}
```

其中 `op` 为 4 位操作码，`reg` 为 4 位地址寄存器编号，`bitLow` 和 `bitHigh` 各 4 位共 8 位用于标示所操作之量子位在汇编具体指令时需要对指令的类型加以分析，若为一地址指令，则完成汇编动作后可直接交给解释程序解释执行，若为两地址指令，须等待两个操作数均汇编后转交解释程序。其实现代码框架如下：

```
public static void assemble(String asmcode) {
    if (4 == asmcode.length()) {
        MachineCode machinecode = new MachineCode(asmcode);
        switch (asmcode.charAt(0)) {
            case 'N': machinecode.op = 0x0; //若前一条指令为一地址指令则出错
                if (last_op == 'N' || last_op == 'H' || last_op == 'I' || last_op == 'X' || last_op == 'Z' || last_op == 'S' || last_op == 'T' ||
                    last_op == 'F' || last_op == 'A' || last_op == 'O' || last_op == 'P' || last_op == 'Q') {
                    errorreport("Invalid ASM instruction!");
                }
            else {
                //将两操作数一起转交解释程序
                dispatch(last_machinecode, machinecode);
            }
            break;
            case 'H': machinecode.op = 0x1;
                last_op = 'H'; dispatch(machinecode);
        }
    }
}
```

```

        break;
        ...
        case 'W': machinecode.op = 0x6;
            last_op = 'W';
        break;
        ...
    }
    last_machinecode = machinecode.cloneMe();
} else {
    errorreport("Invalid ASM instruction!");
}
}

```

**4.3 解释程序** 量子解释程序的主要工作在于分析和解释执行量子机器指令，通过经典计算机提供的计算能力对量子计算过程加以模拟。如前所述，为了模拟  $n$  个量子位，需要提供  $2n$  个经典位，所以在程序中通过定义：

```
static Complex[] state = new Complex[256];
```

实现对 8 位量子位的所有量子态的记录和模拟工作，例如：`state[0]`存放基态 $|00000000\rangle$ 所对应的概率幅，而 `state[1]`存放基态 $|00000001\rangle$ 所对应的概率幅，其他情况可依此类推。

**4.4.1 非测量类指令的解释** 正如我们在设计时所考虑的情况，指令中分为非测量类与测量类两类，其中非测量类指令用于改变量子位的状态，而这种改变是通过酉变换（酉矩阵）实现的。相应的数学形式则是酉矩阵。为了能在程序中表达酉矩阵，提供了如下数据结构定义：

```

class Umatrix {
    public Complex x1;
    public Complex x2;
    public Complex y1;
    public Complex y2;
    public Umatrix() {...}
    public Umatrix(Complex tmp_x1, Complex tmp_x2) {...}
    public Umatrix(Complex tmp_x1, Complex tmp_x2, Complex tmp_y1, Complex tmp_y2) {...}
    public Umatrix multiply(Umatrix tmp) {...}
}

```

用以描述  $2 \times 2$  的酉矩阵，对应一位酉变换，一位量子位的酉变换在程序中具体实现为一个二元行向量和对应酉矩阵相乘，这一乘法的实现代码如下：

```

public Umatrix multiply(Umatrix tmp) {
    Complex tmp_x1x1 = this.x1.cloneMe().multiply(tmp.x1);
    Complex tmp_x2y1 = this.x2.cloneMe().multiply(tmp.y1);
    Complex tmp_x1x2 = this.x1.cloneMe().multiply(tmp.x2);
    Complex tmp_x2y2 = this.x2.cloneMe().multiply(tmp.y2);
    this.x1 = tmp_x1x1.add(tmp_x2y1);
    this.x2 = tmp_x1x2.add(tmp_x2y2); return this;
}

```

**4.4.2 测量类指令的解释** 和非测量类指令相比，测量指令的特点在于测量结果的随机性。虽然经典世界中很难找到真随机事件，但是量子测量所带来的塌缩被认为是一种真随机事件，这也就意味着在测量发生之前，结果一定未知且不可预测。为了能在经典计算机上模拟这一过程，首先要通过计算机生成一个随机数：

```
double rand_num = Math.random();
```

在此基础上对需要测量的量子位所对应的概率幅加以选择，并且以某一基态所反映之经典值作为测量结果回送 全测量指令实现相对简单

```
public static int measure(int reg) {
    ...
    for (i = 0; i < 256; i++) {
        sum += Math.pow(state[i].abs(), 2);
    }
    for (i = 0; i < 256; i++) {
        rand_num -= Math.pow(state[i].abs(), 2) / sum;
        if (rand_num < 0) {
            //对所有量子态做初始化操作
            init_states();
            //回送测量结果
            return i;
        }
    }
}
```

单位测量由于测量后只有一位量子位发生塌缩，而其他量子位仍然有效，故须加入归一化操作，其代码如下：

```
public static boolean measure(int reg, int bit) {
    ...
    if ((rand_num - poss_0) > 0) {
        measure_result = true;
        //归一化操作
        for (i = 0; i < 128; i++) {
            state[bit_no_array[bit][i]] = new Complex(0);
        }
        for (i = 0; i < 256; i++) {
            state[i].div(Math.sqrt(poss_1));
        }
    } else {...}
    return measure_result;
}
```

## 5 示例

以 Deutsch 算法<sup>[11]</sup>为例，其 NDQJava 程序如下：

```

public class Deutsch
{
    public static void main(String [] args) {
        Deutsch d = new Deutsch();
        int i = d.deutsch_1();
        System.out.println("the result1 of Deutsch is: "+i);
        i = d.deutsch_2();
        System.out.println("the result2 of Deutsch is: "+i);
    }
    public int deutsch_1() {
        int i=0;
        begin
        qtype qs;
        qs := init 2::0;
        qs := qs u_i(1) u_x(0);
        qs := qs u_h(1) u_h(0); //输入 函数f为平衡函数
        qs := qs u_cnot(1,0);
        qs := qs u_h(1) u_i(0);
        i := qs q_measure(1);
        end
        return i;
    }
    public int deutsch_2() {
        int i=0;
        begin
        qtype qs;
        qs := init 2::0;
        qs := qs u_i(1) u_x(0);
        qs := qs u_h(1) u_h(0); //输入 函数f为常函数
        qs := qs u_i(0) u_i(1);
        qs := qs u_h(1) u_i(0);
        i := qs q_measure(1);
        end
        return i;
    }
}

```

经过 NDQJava 处理系统处理后得到量子部分汇编指令序列如下：

```

XM00 HM01 HM00 CM01 NM00 HM01
MM00 XM02 HM03 HM02 HM03 MM02

```

由量子汇编程序汇编为如下量子机器指令并解释执行：

```

0011 0000 0000 0001      0001 0000 0000 0010      0001 0000 0000 0001      0111 0000 0000 0010
0000 0000 0000 0001      0001 0000 0000 0010      1110 0000 0000 0001      0011 0000 0000 0100
0001 0000 0000 1000      0001 0000 0000 0100      0001 0000 0000 1000      1110 0000 0000 0100

```

最终程序输出结果为：

```

the result1 of Deutsch is: 1
the result2 of Deutsch is: 0

```

与预期结果相符

## 6 结语

作为 NDQJava 处理系统的组成部分之一，量子汇编和解释程序承担了全部量子指令的汇编和解释工作。目前，通过模拟实现的量子汇编和解释程序还只能提供极为有限的量子位。尽管如此，NDQJava 及其处理系统已经可以用作已有量子算法的验证。

今后，量子汇编及解释程序部分的主要改进方向是：（1.）提高可以模拟的位数，改进现有汇编及解释程序所采用之数据结构和算法；（2.）尝试和现有量子设备接口，将量子汇编指令汇编为实际可用的量子设备上的机器指令，争取早日能在量子设备上实现 NDQJava 处理系统。

**致谢** 感谢徐家福教授、钱士钧教授、戴静安、刘吉、钱辰、徐明君、吴庆曦、董青、朱晓瑞和颜仙乐对本项工作的帮助。

### References

- [1] Xu Jia-Fu, Song Fang-Min, Qian Shi-Jun, et al. Quantum Programming Language NDQJava. *Journal of Software*, 2008, 19(1): 1~8. (徐家福, 宋方敏, 钱士钧, 等. 量子程序设计语言 NDQJava. *软件学报*, 2008, 19(1): 1~8).
- [2] Song Fang-Min, Qian Shi-Jun, Dai Jai-An, et al. Quantum Programming Language NDJava Processing System. *Journal of Software*, 2008, 19(1): 9~16. (宋方敏, 钱士钧, 戴静安, 等. 量子程序设计语言 NDQJava 处理系统 *软件学报*, 2008, 19(1): 9~16).
- [3] M. Tegmark, J. A. Wheeler. 100 Years of Quantum Mysteries. *Scientific American*, 2001, 284(3): 68~75.
- [4] A. Zeilinger. The Quantum Centennial. *Nature*, 2000, 408(2): 639~641.
- [5] D. Kleppner, R. Jackiw. One Hundred Years of Quantum Physics. *Science*, 2000, 289(3): 893~898.
- [6] R. Feynman. *Feynman Lectures on Computation*. Massachusetts: Addison-Wesley, 1996, 121~134.
- [7] P.Selinger. Towards a Quantum Programming Language. *Mathematical Structure in Computer Science*, 2004, 14(4): 527~586.
- [8] P. Zuliani. Compiling quantum programs. *Acta Informatica*, 2005, 41(7): 435~473.
- [9] Zhang Yong-De. *Quantum Mechanics*. Beijing: Science Press, 2002, 19~29 (张永德. *量子力学*. 北京: 科学出版社, 2002, 19~29)
- [10] P.Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 1997, 26: 1484~1509.
- [11] D.Deutsch, R.Jozsa. Rapid solution of problems by quantum computation. Sir Michael Berry FRS. *Proceedings of Royal Society A. London: The Royal Society*, 1992: 553~558.
- [12] Lov K. Grover. A fast quantum mechanical algorithm for database search. Gary L. Miller. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. New York: ACM Press, 1996: 212~219.