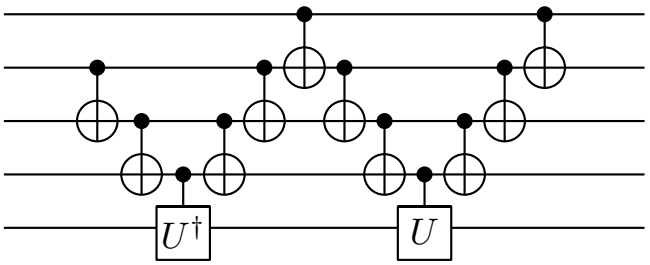
Theoretical Science Group

理論科学グループ



部報 299 号
— 駒場祭決起コンパ号 —

目　次

一般記事

量子計算言語 (QCL) 入門

koba2abc

Introduction

注意:この記事は駒場祭のときに発行する部誌に載せようと思っている記事の序文です.
この記事ではたった一つのアルゴリズムの紹介しかしません.
(また, おまけが本編です.)

量子コンピュータの有名な性質として,「素因数分解を素早くできる」というものがあげられますが、そのメカニズムを解説したいと思います.
(注意:この記事は単なる序文にすぎないので内容はほとんどありません. 次回の駒場祭号に詳しい記事を載せていただこうと考えています.)

準備

分数の近似

以下の問題を考えます.

ある正の整数 M と有理数 q が存在します. この q は $0 \leq q < 1$ と,

$$\exists r, s \in \mathbb{Z}, 0 \leq r < s \leq M \wedge q = \frac{r}{s}$$

を満たします. このとき,r, s を知るためには q をどのくらいの精度で知ればよいでしょうか?
ここで, **精度 ε である数 q を知る**とは,

$$n\varepsilon \leq q < (n+1)\varepsilon$$

を満たす整数 n を知る, という意味です.(n が整数に限られることに注意してください)

これに対して,$\varepsilon = 1/(M(M-1))$ の精度で q を知れば十分なことが証明できます.
略証:

二つの候補 $r_1/s_1, r_2/s_2$ について, **それらが異なるとすれば**その差は

$$\left|\frac{r_1}{s_1} - \frac{r_2}{s_2}\right| = \left|\frac{r_1 s_2 - r_2 s_1}{s_1 s_2}\right| \geq \frac{1}{\mathrm{lcm}(s_1, s_2)} \geq \frac{1}{M(M-1)} = \varepsilon$$

を満たすので,

$$n\varepsilon \leq q < (n+1)\varepsilon$$

が成り立つとすれば, 二つの候補がともに

$$n\varepsilon \leq \frac{r}{s} < (n+1)\varepsilon$$

を満たすことはありません.

□

さて, ここで興味のあるのは $M = 2^m \quad (m \in \mathbb{Z})$ が成り立つときであり, その場合は

$$\varepsilon = \frac{1}{M(M-1)} \geq \frac{1}{M^2} = 2^{-2m}$$

であるため,q の精度は小数点以下 $2m$ 桁で十分です.

次に考えることは,$q = r/s$ を満たす r, s を知ることはどのくらいの時間がかかるのか, ということです. これは量子コンピュータにおける素因数分解 ($m = \log_2 M$ の多項式時間) に使われるので,$m = \log_2 M$ の多項式時間以内に終了する必要があります.
この条件を満たすアルゴリズムは存在し, 以下のようになります.

q を, 問題となる有理数とする.
$s_{\max}$ を,s の最大値 (上の場合では M) とおく.

1. y を実数とし, その値を q とする.
2. a_0 を整数とする
3. a_1 を整数とし, その値を 1 とする.
4. a_2 を整数とする
5. 以下 6. から 12. までのの動作を繰り返す.
6. $z := y - \mathrm{floor}(y)$
7. $z < 0.5/{s_{\max}}^2$ ならば a_1 を返して終了する
8. $y \leftarrow 1/z$
9. $a_2 \leftarrow \mathrm{floor}(y)a_1 + a_0$
10. $a_2 \geq s_{\max}$ ならば a_1 を返して終了する
11. $a_0 \leftarrow a_1$
12. $a_1 \leftarrow a_2$
13. 5. に戻る

(疑似コードが見づらくてすみません. 次には QCL におけるソースコードも載せようと思います.)

このアルゴリズムは s, r のうち r の方のみを返します.(s は $s = qr$ として計算できるので返す必要がない)
このアルゴリズムは連分数と深いかかわりがあります. 詳しくは 2011 年度東大入試数学第 2 問などを参照. 紙面の都合上このアルゴリズムが必ず停止することの証明は次回に回すか, あるいは省略します.

量子アルゴリズム

すいませんが次回に回します. 興味のある方は
`http://tph.tuwien.ac.at/~oemer/doc/quprog/node28.html`
などを調べてみてください.

おまけ

void main(void)

C 言語では書いてはいけないコードとしてよく知られている

```
void main(void){/* ... */}
```

ですが, なぜいけないのかをアセンブリ言語の観点から検証してみようと思います.

基礎知識の確認

そもそも `int main(void)` ないし `int main(int,char**)` 関数は何を戻り値として返すのかというと, 正常終了/異常終了という status を返すのでした (0 が正常, 非 0 が異常)
しかし `void` 型では戻り値を返すことができないので, プログラムの呼び出し元が本来の終了コードを知る方法はなくなります.(`void exit(int)` などでプログラムを終了している場合には問題ない).
このとき終了コードには何が返されるのでしょうか. それを調べるために C/C++で書かれたコードをアセンブリ言語に変換し, 調べてみました.(アセンブリ言語の読み方については適当なサイト等を参照, ここでは

- `mov A,B` がレジスタ A に値 B を代入する命令である
- `xor A,B` がレジスタ A の値と B を xor し, その結果を A に代入する命令である
- `ret` がサブルーチンを終了し, 呼び出し元に戻る命令である
- サブルーチンの戻り値はレジスタ `eax` に格納される

ということを理解していればよいです.)
なお C 言語における関数はアセンブリ言語においてはサブルーチンというものになっています.

MinGW(gcc) の場合

参考ソース (”tmp3.c”/”tmp3.cpp”)

```
#include <stdio.h>

void main(void)
{
    printf("%dss",2);
}
```

これに対して (狭義の) コンパイル (アセンブリ言語への変換) をしてみました.
g++(C++コンパイラ) では

```
C:\Users\2adc\tmp>g++ tmp3.cpp -S -o tmp3-cpp.S
tmp3.cpp:3:15: エラー: '::main' must return 'int'
```

となってコンパイルに失敗しましたが,gcc(C コンパイラ) ではコンパイルに成功しました.
そこで、二通りのコンパイルを試してみてその差異を比較しました.(なおこの場合, アセンブリの文法は AT&T 方式になる)
(i)

```
gcc tmp3.c -S -o tmp3-c.S
```

(何のオプションもつけずに)
tmp3.c のコンパイル結果である tmp3-c.S

```
    .file    "tmp3.c"
    .def    ___main;    .scl    2;    .type    32;    .endef
    .section .rdata,"dr"
LC0:
    .ascii "%dss\0"
    .text
    .globl    _main
    .def    _main;    .scl    2;    .type    32;    .endef
_main:
LFB6:
    .cfi_startproc
    pushl    %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
```

```
    .cfi_def_cfa_register 5
    andl    $-16, %esp
    subl    $16, %esp
    call    ___main
    movl    $2, 4(%esp)
    movl    $LC0, (%esp)
    call    _printf
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE6:
    .def    _printf;    .scl    2;    .type    32;    .endef
```

(ii)

```
gcc tmp3.c -S -o tmp3-c-O2.S -O2
```

(-O2 により最適化を図る)

tmp3-c-O2.S

```
    .file    "tmp3.c"
    .def    ___main;    .scl    2;    .type    32;    .endef
    .section .rdata,"dr"
LC0:
    .ascii "%dss\0"
    .section    .text.startup,"x"
    .p2align 2,,3
    .globl    _main
    .def    _main;    .scl    2;    .type    32;    .endef
_main:
LFB7:
    .cfi_startproc
    pushl    %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
```

```
    andl    $-16, %esp
    subl    $16, %esp
    call    ___main
    movl    $2, 4(%esp)
    movl    $LC0, (%esp)
    call    _printf
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE7:
    .def    _printf;    .scl    2;    .type    32;    .endef
```

これにより, どちらの場合でも差はないことがわかりました.
(おまけとして,exe ファイル (PE フォーマット) に変換した後

```
objdump tmp3-c-O2.exe -d
```

によって逆アセンブルしたコードを紹介します.

```
00401b70 <_main>:
  401b70:    55                      push   %ebp
  401b71:    89 e5                   mov    %esp,%ebp
  401b73:    83 e4 f0                and    $0xfffffff0,%esp
  401b76:    83 ec 10                sub    $0x10,%esp
  401b79:    e8 2a fd ff ff          call   4018a8 <___main>
  401b7e:    c7 44 24 04 02 00 00    movl   $0x2,0x4(%esp)
  401b85:    00
  401b86:    c7 04 24 64 30 40 00    movl   $0x403064,(%esp)
  401b8d:    e8 4e ff ff ff          call   401ae0 <_printf>
  401b92:    c9                      leave
  401b93:    c3                      ret
```

)

以上から, どちらの場合にも eax に 0 を代入する処理が行われていないことがわかります. したがって, この様なプログラムを書いた場合,gcc ではその動作 (終了コード) は不定となります.

Visual C++ 2010 Express の場合

今度は少しだけ異なるコードをコンパイルしました (ほとんど変わりませんが).
test00-1.cpp

```
// test00-1.cpp : コンソール アプリケーションのエントリ ポイントを定義します。
//

#include "stdafx.h"


void main(void)
{
    printf("dd%d",2);
}
```

これを Debug/Release 両方のモードでコンパイルしました. その結果, それぞれのモードで C としてコンパイルした結果と, C++としてコンパイルした結果は (変換後のアセンブリ言語が) 一致しました. 以下に変換後のアセンブリ**と, 元のソースを並べたもの**をを示します. (なお, このアセンブリ言語の文法は Intel 方式である. また, 高級言語の一行に対応するアセンブリコードが高級言語のすぐ下に書かれていることも注意)
(i)Debug モード (C/C++)

```
--- c:\users\2adc\documents\favorites\vc2010\test00\test00-1\test00-1.cpp ----
// test00-1.cpp : コンソール アプリケーションのエントリ ポイントを定義します。
//

#include "stdafx.h"


void main(void)
{
00E113B0 55                   push        ebp
00E113B1 8B EC                mov         ebp,esp
00E113B3 81 EC C0 00 00 00    sub         esp,0C0h
00E113B9 53                   push        ebx
00E113BA 56                   push        esi
00E113BB 57                   push        edi
```

```
00E113BC 8D BD 40 FF FF FF    lea         edi,[ebp-0C0h]
00E113C2 B9 30 00 00 00       mov         ecx,30h
00E113C7 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
00E113CC F3 AB                rep stos    dword ptr es:[edi]
    printf("dd%d",2);
00E113CE 8B F4                mov         esi,esp
00E113D0 6A 02                push        2
00E113D2 68 3C 57 E1 00       push        offset string "dd%d" (0E1573Ch)
00E113D7 FF 15 D4 82 E1 00    call        dword ptr [__imp__printf (0E182D4h)]
00E113DD 83 C4 08             add         esp,8
00E113E0 3B F4                cmp         esi,esp
00E113E2 E8 4A FD FF FF       call        @ILT+300(__RTC_CheckEsp) (0E11131h)
}
00E113E7 33 C0                xor         eax,eax
00E113E9 5F                   pop         edi
00E113EA 5E                   pop         esi
00E113EB 5B                   pop         ebx
00E113EC 81 C4 C0 00 00 00    add         esp,0C0h
00E113F2 3B EC                cmp         ebp,esp
00E113F4 E8 38 FD FF FF       call        @ILT+300(__RTC_CheckEsp) (0E11131h)
00E113F9 8B E5                mov         esp,ebp
00E113FB 5D                   pop         ebp
00E113FC C3                   ret
```

Release モード (C/C++)

```
--- c:\users\2adc\documents\favorites\vc2010\test00\test00-1\test00-1.cpp ----
// test00-1.cpp : コンソール アプリケーションのエントリ ポイントを定義します。
//

#include "stdafx.h"


void main(void)
{
    printf("dd%d",2);
00901000 6A 02                push        2
00901002 68 F4 20 90 00       push        offset string "dd%d" (9020F4h)
```

```
00901007 FF 15 A0 20 90 00    call        dword ptr [__imp__printf (9020A0h)]
0090100D 83 C4 08             add         esp,8
}
00901010 33 C0                xor         eax,eax
00901012 C3                   ret
```

両方のコードに

```
xor eax,eax
```

というコードが認められます. これは C 風に書くと

```
eax^=eax;
```

となるので, 結局は `eax` を 0 にするコードということになります. (なぜこのようなコードになっているのかというと, 直接 `mov eax,0` で代入しようとすると機械語が 5 バイト (`b8 00 00 00 00`) になるので, 最適化が行われたためです.)

編集後記

· $\int_{\mathbb{R}} \exp -\frac{r^2}{\omega^2} \exp -\frac{(2d-r)^2}{\omega^2} dr$

· $\sim \int_{\mathbb{R}} \exp -\frac{2(r-d)^2}{\omega^2} \exp -\frac{2d^2}{\omega^2} dr$

· $\sim \int_{\mathbb{R}} \exp -\frac{2d^2}{\omega^2} dr$

＾ 1 年もすると後記のネタなんかなくなりますよ＞次期編集

理論科学グループ　部報 第 299 号

2012 年 11 月 9 日　発行

発行者　河内谷耀一

編集者　久良尚任

発行所　理論科学グループ

〒 153–0041　東京都目黒区駒場 3–8–1

東京大学教養学部内学生会館 305

Telephone: 03–5454–4343

理論科学グループ部報 第 299 号
— 駒場祭決起コンパ号 —
2012 年 11 月 9 日

THEORETICAL SCIENCE GROUP