Software Engineering

CS 217

A contest problem

COS 333 PROGRAMMING CONTEST #1: QUICKROOT

From: Dr. Guy Jacobson
Due: 12:01 A.M. 13 February 1995

Your boss at Yoyodyne laboratories has discovered that a critical slowdown in their mission software is due to a routine that calculates integer cube roots:

```
double cbrt (double);
int quickroot (int i)     {
    return (int) cbrt ((double) i);
}
```

You must rewrite this routine so that it is faster. Much faster. Your boss insists that you give him the following by 13 February:

1. A single file "quickroot.c" that implements the function quickroot.
2. A short (1-2 page) description of how your function works.

Software engineering

- Lots of important things to learn about software engineering
- You've already learned some important ones:
  - modularity, abstract data types, ...
- There are many others
- I'm not in the mood to tell you about them.

- Instead...

A contest problem (cont'd)

Furthermore, he insists that:

a. Your function return, for any non-negative integer $0 \le n < 2^{31}$, the greatest integer not greater than the cube root of n, just like the old slow quickroot().

b. Your function must be in a single .c file of ≤ 5000 characters.

c. Your function have no other externally-visible side effects, except perhaps for allocating memory.

Other than that, all he cares about is speed. Raw, blinding speed. He says that he's going to test your program on arizona, compiling with gcc and using `time(1)` to measure user time, by linking with the following driver program:

How to Cheat

CS 217

A contest problem (cont'd)

```
#include <stdio.h>
main (int ac, char *av[])  {
      int i, j;
      srandom (atoi (av[1]));
      for (i = 0; i < 10000000; i++)
          j = quickroot (random ());
  }
```

He won't tell you what number he's going to use as a random seed to srandom,

You are in competition with all the other engineers here at Yoyodyne. Your grade will depend primarily on the speed of your function as measured above (and, of course, its correctness), ranked against to the speed of all the other entries.. The fastest entries get special recognition as well.

No excuses, and good luck.

How to solve it?

- A quick hack:

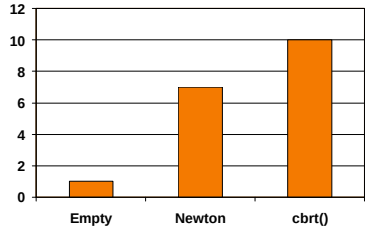
```
int quickroot (int i)     {
        return 0;
}
```

- Blindingly fast when used with driver.c:

```
#include <stdio.h>
main (int ac, char *av[])  {
      int i, j;
      srandom (atoi (av[1]));
      for (i = 0; i < 10000000; i++)
          j = quickroot (random ());
}
```

- Unfortunately, violates rule (a), that **`quickroot`** must be correct in any context, not just this driver.
- However, quickroot need not be *fast* in all contexts...

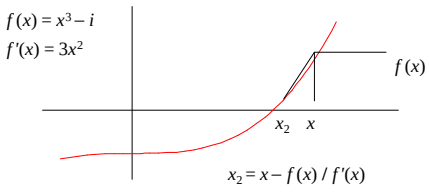
Results:



Reduce time from 10 seconds down to 7 seconds.

Question: cbrt( ) uses Newton's method too; why the improvement?

Newton's method



$x_2 = x - (x^3\text{-}i) / (3x^2) = {}^1/_3(2x + i/x^2)$

```
for (n=0;n<7;n++) {
    x = (1/3.0)*(x+x+i/(x*x));
 }
```

Winning at all costs

- Let's use programming-languages theory
  - A *continuation* is a theoretical representation of "the rest of the execution of the program"
  - Use continuation transform to write quickroot as, quickroot(i, k) = k(√i)
  - Invent all-new, special purpose equality-test operator for continuations: ;=)
    k1 ;=) k2 means, continuation k1 has same structure as k2

quickroot(i,k) =
if k ;=) driver_main
then $k_0(0)$
else k(√i)

Claim: driver_main(√i) = $k_0(0)$

Picking a good start point

```
  if (i > 1<<15)
    if (i > 1<<24)
      if (i > 1<<27)
          x = 1<<9;
      else x = 1<<8;
    else if (i > 1<<18)
      if (i > 1<<21)
          x = 1<<7;
      else x = 1<<6;
    else x = 1<<5;
  else if (i > 1<<9)
    if (i > 1<<12)
        x = 1<<4;
    else x = 1<<3;
  else x = 1<<2;

for (n=0;n<7;n++) {
    x = (1/3.0)*(x+x+i/(x*x));
  }
```

My quickroot.c

```
#include <stdio.h>
mainX (int ac, char *av[])  {
      int i, j;
      srandom (atoi (av[1]));
      for (i = 0; i < 10000000; i++)
          j = quickroot (random ());
   }
endMain(){}

double cbrt (double);
extern main();
unsigned mycaller[] ={0x81c3e008,0x9010001f};
```

My quickroot.c

```
int quickroot(int i) {
 static x=0;
 if (x) return (int) cbrt ((double) i);
 x=1;

   here is the special hack . . .
}
```

Step 2: Is the caller ;=) driver-main?

```
#include <stdio.h>
mainX (int ac, char *av[])  {
      int i, j;
      srandom (atoi (av[1]));
      for (i = 0; i < 10000000; i++)
          j = quickroot (random ());
   }
endMain(){}


if (caller <= (unsigned)main ||
    caller >= (unsigned)main +
              (unsigned)endMain-(unsigned)mainX )
   return quickroot(i);
```

If return address doesn't point within a certain number of bytes from the beginning of the main( ) function, then we're not being called from the test driver. In that case, slowly and carefully compute the actual cube root.

The special hack in quickroot.c

```
unsigned *p, *q, caller;
 union {unsigned *z; unsigned (*f)();} u;
 u.z=mycaller; caller = u.f();
 if (caller <= (unsigned)main ||
     caller >= (unsigned)main+(unsigned)endMain-(unsigned)mainX)
   return quickroot(i);
 for(p=(unsigned*)mainX, q=(unsigned*)main;
       p<(unsigned*)endMain; p++,q++) {
    unsigned px = *p, qx = *q;
    if ((px&0xf0000000) == 0x40000000 &&
        (qx&0xf0000000) == 0x40000000)
        {px += ((unsigned) p)>>2; qx += ((unsigned) q)>>2;}
    if (px != qx) return quickroot(i);
  }
 exit(1);
```

Step 2, continued

```
#include <stdio.h>
mainX (int ac, char *av[])  {
      int i, j;
      srandom (atoi (av[1]));
      for (i = 0; i < 10000000; i++)
          j = quickroot (random ());
   }
endMain(){}


for(p=(unsigned*)mainX, q=(unsigned*)main;
      p<(unsigned*)endMain; p++,q++) {
    unsigned px = *p, qx = *q;
    if (px != qx) return quickroot(i);
  }
```

If any of the instructions in the caller don't match the instructions of mainX, then give up: slowly and carefully compute cube root.

Step 1: find out who called you

```
unsigned mycaller[] ={0x81c3e008,  0x9010001f};
                        retl ;     mov %i7,%o0



unsigned *p, *q, caller;
union {unsigned *z; unsigned (*f)();} u;
u.z=mycaller;
caller = u.f();
```

Now `caller` *is the return address of quickroot, i.e. points somewhere into the middle of main()*

Disassembly of mainX

```
#include <stdio.h>
mainX (int ac, char *av[])  {
      int i, j;
      srandom (atoi (av[1]));
      for (i = 0; i < 10000000; i++)
          j = quickroot (random ());
   }
endMain(){}
```

```
save  %sp, -112, %sp
call  0x209cc <atoi>
ld  [ %i1 + 4 ], %o0
call  0x209d8 <srandom>
nop
sethi  %hi(0x989400), %o1
or  %o1, 0x27f, %o1
add  %o1, 1, %i1
call  0x209e4 <random>
nop
call  0x107ac <quickroot>
nop
addcc  %i1, -1, %i1
bne  0x10780 <mainX+32>
nop
call  0x2099c <exit>
clr  %o0        ! 0x0
```

main mainX

```
0x9de3bf90    0x9de3bf90   save  %sp, -112, %sp
0x400040ab    0x4000409a   call  0x209cc <atoi>
0xd0066004    0xd0066004   ld  [ %i1 + 4 ], %o0
0x400040ac    0x4000409b   call  0x209d8 <srandom>
0x01000000    0x01000000   nop
0x13002625    0x13002625   sethi  %hi(0x989400), %o1
0x9212627f    0x9212627f   or  %o1, 0x27f, %o1
0xb2026001    0xb2026001   add  %o1, 1, %i1
0x400040aa    0x40004099   call  0x209e4 <random>
0x01000000    0x01000000   nop
0x4000001a    0x40000009   call  0x107ac <quickroot>
0x01000000    0x01000000   nop
0xb2867fff    0xb2867fff   addcc  %i1, -1, %i1
0x12bffffb    0x12bffffb   bne  0x10780 <mainX+32>
0x01000000    0x01000000   nop
0x40004091    0x40004080   call  0x2099c <exit>
0x90102000    0x90102000   clr  %o0        ! 0x0
```

Results

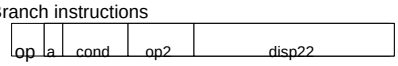
- Correct in all contexts!
  - In any test that actually measures whether it computes cube roots correctly, quickroot() just calls cbrt()
- Very fast in the contest-driver context!
  - Just tests whether called from the contest driver, and if so,...

```
#include <stdio.h>
main (int ac, char *av[])  {
      int i, j;
      srandom (atoi (av[1]));
      for (i = 0; i < 10000000; i++)
          j = quickroot (random ());
 }
```

  - calls exit() at the very first call to quickroot; doesn't execute the loop 10000000 times

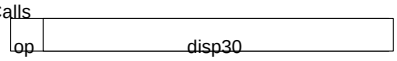
Control Transfer

- Branch instructions



nPC = PC + signextend(disp22) << 2

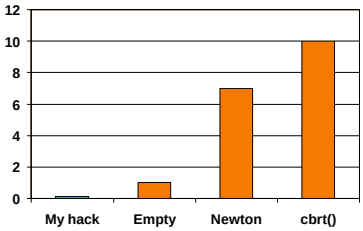
- Calls



nPC = PC + signextend(disp30) << 2

  - position-independent code does not depend on where it's loaded; uses PC-relative addressing

Results:



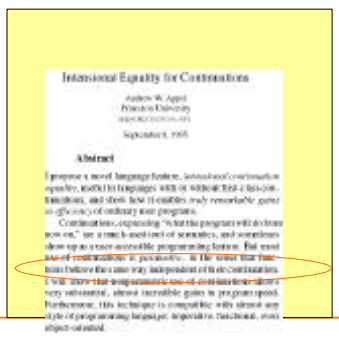
Reduce time from 10 seconds down to 0.0 seconds (measured to the nearest tenth of a second)

This is even faster than the driver running by itself!

My quickroot.c

```
unsigned *p, *q, caller;
 union {unsigned *z; unsigned (*f)();} u;
 u.z=mycaller; caller = u.f();
 if (caller <= (unsigned)main ||
     caller >= (unsigned)main+(unsigned)endMain-(unsigned)mainX)
   return quickroot(i);
 for(p=(unsigned*)mainX, q=(unsigned*)main;
       p<(unsigned*)endMain; p++,q++) {
    unsigned px = *p, qx = *q;
    if ((px&0xf0000000) == 0x40000000 &&
        (qx&0xf0000000) == 0x40000000)
        {px += ((unsigned) p)>>2; qx += ((unsigned) q)>>2;}
    if (px != qx) return quickroot(i);
 }
 exit(1);
```

Publish or perish!


Intensional Equality for Continuations

Andrew W. Appel
Princeton University

September, 1995

Abstract

Spoof or serious?

From: Andrew W. Appel

To: Simon Peyton Jones, Editor, Journal of Functional Programming

Dear Simon: I enclose a short paper for consideration for publication in J. Functional Programming. It's not exactly a research article...

From: Simon Peyton Jones

To: Andrew W. Appel

Dear Andrew:

. . . I don't know what to make of it. (Spoof or serious? If it were dated April 1st I'd know.) Apart from anything else, it patently doesn't work in general (you'd have to compare the stacks too). And it's far from clear that it has applications beyond fooling inadequate test programs.

Warning

- When you have your fun and games, avoid coming too close to academic fraud.
  - (This applies to professors just as much as students)
- It's always possible to tune your program to the particular benchmark test; excessive tuning constitutes fraud.

Revised title

Intensional Equality $=_I$ for Continuations

Andrew W. Appel
Princeton University

September 1, 1995

Abstract

Try again

From: Andrew W. Appel

To: Richard Wexelblat, Editor, SIGPLAN Notices

Dear Dr. Wexelblat: I hereby submit the enclosed short paper, "Intensional Equality ;=) for Continuations", for publication in ACM SIGPLAN Notices.

From: Richard L. Wexelblat

To: Andrew W. Appel

Dear Andrew:

. . . will apper in February (or possibly March) . . . . Having read it carefully three times, I'm not sure but that it ought to appear in the April first issue,... but that would be unfair to so obviously dedicated a person as yourself.