Computer Systems

CS107

Cynthia Lee

Today's Topics

- Code optimization!

Optimization reality check

Don't let it be your Waterloo.

Optimization Reality Check

- Optimization is really exciting
- …but it's easy to be overzealous or misguided about it.

- "We should forget about small inefficiencies, say about 97% of the time: premature optimization is the root of all evil." – Donald Knuth

- "More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason–including blind stupidity." – W.A. Wulf

- "Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you have proven that's where the bottleneck is." – Rob Pike

Stanford University
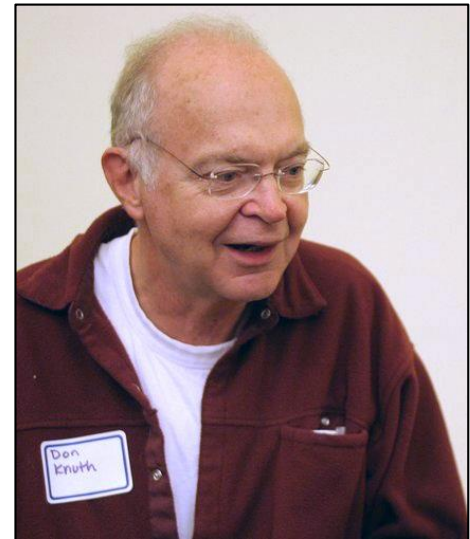
Optimization Reality Check

- Most of what <u>you</u> need to do with optimization can be summarized in 3 easy steps:

- Step 1:
  - › If doing something seldom and only on small inputs, do whatever is simplest to code, understand, and debug

- Step 2:
  - › If doing things thing a lot, or on big inputs, make the algorithm's Big-O cost reasonable

- Step 3:
  - › **Let gcc do its magic from there**

gcc optimization levels

- Today, we'll be comparing two levels of optimization in the gcc compiler:
  - › `gcc –O0  //mostly just literal translation of C`
  - › `gcc –O2  //enable nearly all reasonable optimizations`
  - › (we use –Og, like –O0 but with less needless use of the stack)

- There are other custom and more aggressive levels of optimization, e.g.:
  - › `-O3     //more aggressive than O2, trade size for speed`
  - › `-Os     //optimize for size`
  - › `-Ofast  //disregard standards compliance (!!)`

- Exhaustive list of gcc optimization-related flags:
  - › https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Example: gcc performance optimization

- Just a standard matrix multiply, triply-nested for loop:

```
static void mmm(double a[][DIM], double b[][DIM],
                double c[][DIM], int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                c[i][j] += a[i][k]*b[k][j];
}
```

Measuring performance: Example code

› ./mult   // -O0 (no optimization)

  matrix multiply 25^2: cycles   0.44M

  matrix multiply 50^2: cycles   3.13M

  matrix multiply 100^2: cycles 24.80M

› ./mult_opt   // -O2 (with optimization)

  matrix multiply 25^2: cycles   0.11M (opt)

  matrix multiply 50^2: cycles   0.47M (opt)

  matrix multiply 100^2: cycles   3.67M (opt)

When I said, "Let gcc do its magic from there," this is what I meant by magic!

AWESOME! WOW!

Kinds of optimization

Some main categories of optimization that are undertaken by the compiler

Kinds of optimization

- Constant folding
- Common subexpression elimination
- Dead code
- Strength reduction
- Code motion
- Tail recursion
- Loop unrolling

Kinds of optimization

- **Constant folding**
  - › Precalculate constants at compile time where possible:
    - `int volume = WIDTH * HEIGHT * DEPTH;`
    - `double radius = sqrt(area) / 3.14;`
- Common subexpression elimination
- Dead code
- Strength reduction
- Code motion
- Tail recursion
- Loop unrolling

- **Discussion question:**
  - › **What is a consequence of this for you as a programmer?**
    What should you do (or do differently) now that you know about compilers doing <u>constant folding</u> for you?

Stanford University

Constant folding

Before:

```
000000000040098d <CF>:
  40098d:      55                      push   %rbp
  40098e:      48 89 e5                mov    %rsp,%rbp
  400991:      41 54                   push   %r12
  400993:      53                      push   %rbx
  400994:      48 83 c4 80             add    $0xffffffffffffff80,%rsp
  400998:      89 bd 7c ff ff ff       mov    %edi,-0x84(%rbp)
  40099e:      c7 45 ec 07 01 00 00    movl   $0x107,-0x14(%rbp)
  4009a5:      8b 45 ec                mov    -0x14(%rbp),%eax
  4009a8:      6b c0 55                imul   $0x55,%eax,%eax
  4009ab:      89 45 e8                mov    %eax,-0x18(%rbp)
  4009ae:      48 b8 00 00 00 00 00    movabs $0x4000000000000000,%rax
  4009b5:      00 00 40
  4009b8:      48 89 85 70 ff ff ff    mov    %rax,-0x90(%rbp)
  4009bf:      f2 0f 10 85 70 ff ff    movsd  -0x90(%rbp),%xmm0
  4009c6:      ff
  4009c7:      e8 b4 fe ff ff          callq  400880 <sqrt@plt>
  4009cc:      f2 0f 2c c0             cvttsd2si %xmm0,%eax
  4009d0:      89 45 e4                mov    %eax,-0x1c(%rbp)
  4009d3:      8b 45 ec                mov    -0x14(%rbp),%eax
  4009d6:      0f af 85 7c ff ff ff    imul   -0x84(%rbp),%eax
  4009dd:      89 c3                   mov    %eax,%ebx
  4009df:      b8 15 00 00 00          mov    $0x15,%eax
  4009e4:      99                      cltd
  4009e5:      f7 7d e4                idivl  -0x1c(%rbp)
  4009e8:      89 c2                   mov    %eax,%edx
  4009ea:      8b 45 ec                mov    -0x14(%rbp),%eax
  4009ed:      01 d0                   add    %edx,%eax
  4009ef:      4c 63 e0                movslq %eax,%r12
  4009f2:      bf b8 1b 40 00          mov    $0x401bb8,%edi
  4009f7:      e8 d4 fd ff ff          callq  4007d0 <strlen@plt>
```

After:

```
0000000000400d80 <CF>:
  400d80:      69 c7 07 01 00 00       imul   $0x107,%edi,%eax
  400d86:      05 61 6d 00 00          add    $0x6d61,%eax
  400d8b:      c3                      retq
```

** Notice it also got rid of the stack frame handling overhead

Kinds of optimization

- Constant folding
- **Common subexpression elimination**
  - › Prevent recalculation of the same thing many times by doing it once and saving result

    ```
    int a = (param2 + 0x107);
    int b = param1 * (param2 + 0x107) + a;
    return a * (param2 + 0x107) + b * (param2 + 0x107);
    ```
- Dead code
- Strength reduction
- Code motion
- Tail recursion
- Loop unrolling

Stanford University

CSE

Before:

```
0000000000400a1c <CSE>:
  400a1c:       55                  push    %rbp
  400a1d:       48 89 e5            mov     %rsp,%rbp
  400a20:       89 7d ec            mov     %edi,-0x14(%rbp)
  400a23:       89 75 e8            mov     %esi,-0x18(%rbp)
  400a26:       8b 45 e8            mov     -0x18(%rbp),%eax
  400a29:       05 07 01 00 00      add     $0x107,%eax
  400a2e:       89 45 fc            mov     %eax,-0x4(%rbp)
  400a31:       8b 45 e8            mov     -0x18(%rbp),%eax
  400a34:       05 07 01 00 00      add     $0x107,%eax
  400a39:       0f af 45 ec         imul    -0x14(%rbp),%eax
  400a3d:       89 c2               mov     %eax,%edx
  400a3f:       8b 45 fc            mov     -0x4(%rbp),%eax
  400a42:       01 d0               add     %edx,%eax
  400a44:       89 45 f8            mov     %eax,-0x8(%rbp)
  400a47:       8b 45 f8            mov     -0x8(%rbp),%eax
  400a4a:       8b 55 fc            mov     -0x4(%rbp),%edx
  400a4d:       01 c2               add     %eax,%edx
  400a4f:       8b 45 e8            mov     -0x18(%rbp),%eax
  400a52:       05 07 01 00 00      add     $0x107,%eax
  400a57:       0f af c2            imul    %edx,%eax
  400a5a:       5d                  pop     %rbp
  400a5b:       c3                  retq
```

After:

```
0000000000400d90 <CSE>:
  400d90:       81 c6 07 01 00 00   add     $0x107,%esi
  400d96:       0f af fe            imul    %esi,%edi
  400d99:       8d 04 77            lea     (%rdi,%rsi,2),%eax
  400d9c:       0f af c6            imul    %esi,%eax
  400d9f:       c3                  retq
```

Kinds of optimization

- Constant folding
- Common subexpression elimination
- **Dead code**
  - › Remove code that doesn't serve a purpose:
    ```
    while (false) {
        j = solve(j);
        printf("This loop can't happen!\n");
        i++;
    }
    ```
- Strength reduction
- Code motion
- Tail recursion
- Loop unrolling

Stanford University

DC

Before:

```
0000000000400a5c <DC>:
  400a5c:       55                      push    %rbp
  400a5d:       48 89 e5                mov     %rsp,%rbp
  400a60:       48 83 ec 20             sub     $0x20,%rsp
  400a64:       89 7d ec                mov     %edi,-0x14(%rbp)
  400a67:       89 75 e8                mov     %esi,-0x18(%rbp)
  400a6a:       8b 45 ec                mov     -0x14(%rbp),%eax
  400a6d:       3b 45 e8                cmp     -0x18(%rbp),%eax
  400a70:       7d 17                   jge     400a89 <DC+0x2d>
  400a72:       8b 45 ec                mov     -0x14(%rbp),%eax
  400a75:       3b 45 e8                cmp     -0x18(%rbp),%eax
  400a78:       7e 0f                   jle     400a89 <DC+0x2d>
  400a7a:       bf c0 1b 40 00          mov     $0x401bc0,%edi
  400a7f:       b8 00 00 00 00          mov     $0x0,%eax
  400a84:       e8 57 fd ff ff          callq   4007e0 <printf@plt>
  400a89:       c7 45 fc 00 00 00 00    movl    $0x0,-0x4(%rbp)
  400a90:       eb 04                   jmp     400a96 <DC+0x3a>
  400a92:       83 45 fc 01             addl    $0x1,-0x4(%rbp)
  400a96:       81 7d fc e7 03 00 00    cmpl    $0x3e7,-0x4(%rbp)
  400a9d:       7e f3                   jle     400a92 <DC+0x36>
  400a9f:       8b 45 ec                mov     -0x14(%rbp),%eax
  400aa2:       3b 45 e8                cmp     -0x18(%rbp),%eax
  400aa5:       75 06                   jne     400aad <DC+0x51>
  400aa7:       83 45 ec 01             addl    $0x1,-0x14(%rbp)
  400aab:       eb 04                   jmp     400ab1 <DC+0x55>
  400aad:       83 45 ec 01             addl    $0x1,-0x14(%rbp)
  400ab1:       83 7d ec 00             cmpl    $0x0,-0x14(%rbp)
  400ab5:       75 07                   jne     400abe <DC+0x62>
  400ab7:       b8 00 00 00 00          mov     $0x0,%eax
  400abc:       eb 03                   jmp     400ac1 <DC+0x65>
  400abe:       8b 45 ec                mov     -0x14(%rbp),%eax
  400ac1:       c9                      leaveq
```

After:

```
0000000000400da0 <DC>:
  400da0:       8d 47 01                lea     0x1(%rdi),%eax
  400da3:       c3                      retq
  400da4:       66 66 66 2e 0f 1f 84    data32 data32
```

Kinds of optimization

- Constant folding
- Common subexpression elimination
- Dead code
- **Strength reduction**
  - › Change divide to multiply, multiply to add or shift, and mod to and
  - › Avoids using instructions that cost many cycles (multiply and divide)
    ```
    int doge_years = human_years * 7;
    ```
- Code motion
- Tail recursion
- Loop unrolling

Stanford University

Kinds of optimization

- Constant folding
- Common subexpression elimination
- Dead code
- Strength reduction
- **Code motion**
  - › Move code out of a loop if possible

```
for (int i = 0; i < n; i++) {
    sum += arr[i] + foo * (bar + 3);
}
```

- Tail recursion
- Loop unrolling

Kinds of optimization

- Constant folding
- Common subexpression elimination
- Dead code
- Strength reduction
- Code motion
- **Tail recursion**
  - › Compiler notices some simple recursion patterns that could be more efficiently implemented using iteration (i.e. a loop) to avoid function call and return overhead

```
long factorial(int n) {
    if (n<=1) return 1;
    else return n * factorial(n-1);
}
```

- Loop unrolling

Stanford University

Kinds of optimization

- Constant folding
- Common subexpression elimination
- Dead code
- Strength reduction
- Code motion
- Tail recursion
- **Loop unrolling**
  - › Do n loop iterations' worth of work per actual loop iteration, so we save ourselves from doing the loop overhead (test and jump) every time, and instead incur overhead only every n-th time

```
for (int i=0; i<=n-4; i+=4) {
    sum += arr[i];
    sum += arr[i+1];
    sum += arr[i+2];
    sum += arr[i+3];
} // after the loop handle any leftovers
```

Stanford University

Measuring performance

Lab preview

Two techniques for measuring performance—learn more in lab

- Wall clock time
  - › `gettimeofday() // #include <sys/time.h>`

- Cycle counting
  - › RTC: real-time clock counts elapsed cycles of the CPU
  - › Available on some hardware
  - › Will reveal some cases where an instruction takes more than one cycle (and a few interesting cases where more than one instruction was able to execute per clock cycle)
    - Recall that multiply instruction is more expensive (takes more cycles) than add
  - › See `/afs/ir/class/cs107/samples/lect15/fcyc.h`

Stanford University