Optimization: Assembly Optimization

CPSC 501: Advanced Programming Techniques
Fall 2020

Jonathan Hudson, Ph.D
Instructor
Department of Computer Science
University of Calgary

Wednesday, August 5, 2020

UNIVERSITY OF CALGARY

Code Tuning - Assembly

- Assembly language techniques
  - Are specific to a CPU architecture
    - Thus are not generally portable
  - Goal is to minimize the number of clock cycles it takes to execute an algorithm
    - That is, code the algorithm using the fewest number of instructions possible
    - A **clever** programmer can usually beat the best optimizing compiler
      - We're not always as clever as we think

Code Tuning – Assembly - Quantify

- We can quantify execution time precisely, since each instruction takes a defined number of clock cycles to complete
  - A fixed number on a RISC CPU
    - E.g. 4 cycles per instruction on SPARC V8
  - A variable number on a CISC CPU
    - E.g. Intel Core 2
      - add: 1 cycle   mul: 5   div: 40
    - Some assemblers produce output files showing this cycle count

Instructions

Code Tuning – Assembly - Instructions

- Eliminate instructions where possible
  - Sparc example:
    - We save register window and create new
    - Restore after
    - Uses input registers (function inputs)

```
cube:     save      %sp, -96, %sp
          smul      %i0, %i0, %l0
          smul      %i0, %l0, %i0
          restore
          ret
          nop
```

Code Tuning – Assembly - Instructions

- Eliminate instructions where possible
  - Sparc example:
    - We save register window and create new
    - Restore after
    - Uses input registers (function inputs)
  - Eliminate 2 instructions by converting into a leaf subroutine:
    - We won't call others (leaf)
    - Can only use output registers

    Note: this also prevents the triggering of window overflow/underflow, which is expensive

```
cube:    save     %sp, -96, %sp
         smul     %i0, %i0, %l0
         smul     %i0, %l0, %i0
         restore
         ret
         nop
```

```
cube:    smul     %o0, %o0, %o1
         smul     %o0, %o1, %o0
         retl
         nop
```

Pipeline

Code Tuning – Assembly - Pipeline

- Reorder instructions to keep the pipeline full or to avoid pipeline stalls

```
cube:    smul    %o0, %o0, %o1
         smul    %o0, %o1, %o0
         retl
         nop
```


UNIVERSITY OF CALGARY

Code Tuning – Assembly - Pipeline

- Reorder instructions to keep the pipeline full or to avoid pipeline stalls

```
cube:     smul     %o0, %o0, %o1
          smul     %o0, %o1, %o0
          retl
          nop
```

- E.g. Above code can be changed to:

```
cube:     smul     %o0, %o0, %o1
          retl
          smul     %o0, %o1, %o0     ! filled the delay slot
```

Code Tuning – Assembly - Pipeline

- Reorder instructions to keep the pipeline full or to avoid pipeline stalls

```
cube:    smul    %o0, %o0, %o1
         smul    %o0, %o1, %o0
         retl
         nop
```

  - E.g. Above code can be changed to:

```
cube:    smul    %o0, %o0, %o1
         retl
         smul    %o0, %o1, %o0     ! filled the delay slot
```

  - Eliminates 1 instruction
  - retl has to go through CPU 4 cycle (fetch, execute, memory, write) so we can slide in delay slot so cube is done by time retl gives reaches using it

Inline

Code Tuning – Assembly - Inline

- Use macros to inline subroutines
  - Avoids call/return overhead
  - E.g. Calling code before optimization:

```
. . .
mov       5, %o0
call      cube
nop
. . .                              ! 6 instructions executed
```

  - A macro such as:

```
define(cube,  `smul      $1, $1, %g1
               smul      $1, %g1, $1')
```

SIMD

Code Tuning – Assembly – SIMD

- Use SIMD instructions to move data while calculating
  - **S**ingle **i**nstruction, **m**ultiple **d**ata
  - Motorola DSP56001 example:

```
. . .
MPY     X0, Y1, A
MOVE    X:(R0)+, X0
MOVE    Y:(R4)+, Y0
MAC     X0, Y0, A                              ; 4 cycles
. . .
```

Multiply w/o Accumulate (MPY)
Multiple and Accumulate (MAC)
Move data (MOVE)

Code Tuning – Assembly – Inline (cont'd)

- In extreme cases, one might inline every subroutine!
  - Usually results in a much bigger executable (i.e. more RAM is used)
    - We are trading memory for speed
- Note that some compilers allow one to inline assembly code into C or C++ code
  - sdcc example:

```
unsigned char counter;
. . .
counter = 0;
__asm
        inc     _counter
__endasm;
. . .
```


UNIVERSITY OF CALGARY

Code Tuning – Assembly – Inline (cont'd)

- Can be used in calling code:

```
. . .
mov       5, %o0
cube(%o0)
. . .
```

- gets expanded to

```
. . .
mov       5, %o0
smul      %o0, %o0, %g1
smul      %o0, %g1, %o0
. . .                          ! 3 instructions executed
```

- Eliminates 3 more instructions

Code Tuning – Assembly – SIMD

- Use SIMD instructions to move data while calculating
  - **S**ingle **i**nstruction, **m**ultiple **d**ata
  - Motorola DSP56001 example:

```
. . .
MPY       X0, Y1, A
MOVE      X:(R0)+, X0
MOVE      Y:(R4)+, Y0
MAC       X0, Y0, A                                ; 4 cycles
. . .
```

  - Can be improved to:

```
. . .
MPY       X0, Y1, A    X:(R0)+, X0    Y:(R4)+, Y0
MAC       X0, Y0, A                                ; 2 cycles
```

Code Tuning – Assembly – SIMD (cont'd)

- There are libraries available that use SIMD instructions on vectors of data (and may exploit the parallelism of multi-core CPUs)
  - Intel Vector Math Library (VML)
    - Is a C/C++ API for Windows, Linux, OS X
    - Part of the Intel Math Kernel Library (MKL)
  - Accelerate framework
    - Is a C API for OS X

Onward to …
Java optimization.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~hudsonj/

UNIVERSITY OF CALGARY