Optimization of Floating-point Precision using Binary Modification

Michael Lam
University of Maryland, College Park

Jeff Hollingsworth, Advisor
Background

- Floating-point represents real numbers as $(\pm \text{sgnf} \times 2^{\text{exp}})$
  - Sign bit
  - Exponent
  - Significand ("mantissa" or "fraction")

Floating-point numbers have finite precision
  - Single-precision: 24 bits (~7 decimal digits)
  - Double-precision: 53 bits (~16 decimal digits)
Example

1/10 ➞ 0.1

0x3DCCCCCD = 00111101 11001100 11001100 11001101

Single-precision

0x3FB999999999999A = 00111111 10111001 10011001 10011001 10011001 10011001 10011010

Double-precision
Motivation

• Finite precision causes round-off error
  o Compromises “ill-conditioned” calculations
  o Hard to detect and diagnose

• Increasingly important as HPC scales
  o Double-precision data movement is a bottleneck
  o Streaming processors are faster in single-precision (~2x)
  o Need to balance speed (singles) and accuracy (doubles)
Previous Work

• Traditional error analysis  (Wilkinson 1964)
  o Forwards vs. backwards
  o Requires extensive numerical analysis expertise

• Interval/affine arithmetic (Goubault 2001)
  o Conservative static error bounds are largely unhelpful
Previous Work

- Manual mixed-precision (Dongarra 2008)
  - Requires numerical expertise

1. \( LU \leftarrow PA \)
2. solve \( Ly = Pb \)
3. solve \( Ux_0 = y \)
4. for \( k = 1, 2, \ldots \) do
   5. \( r_k \leftarrow b - Ax_{k-1} \)
   6. solve \( Ly = Pr_k \)
   7. solve \( Uz_k = y \)
   8. \( x_k \leftarrow x_{k-1} + z_k \)
   9. check for convergence
10. end for

- Fallback: ad-hoc experiments
  - Tedious, time-consuming, and error-prone
Our Goal

Develop automated analysis techniques to inform developers about floating-point behavior and make recommendations regarding the precision level that each part of a computer program must use in order to maintain overall accuracy.
Framework

CRAFT: Configurable Runtime Analysis for Floating-point Tuning

• Static binary instrumentation
  o Controlled by configuration settings
  o Replace floating-point instructions with new code
  o Re-write a modified binary

• Dynamic analysis
  o Run modified program on representative data set
  o Produces results and recommendations
Advantages

• Automated
  o Minimize developer effort
  o Ensure consistency and correctness

• Binary-level
  o Include shared libraries without source code
  o Include compiler optimizations

• Runtime
  o Dataset and communication sensitivity
Implementation

- Current approach: in-place replacement
  - Narrowed focus: doubles ➔ singles
  - In-place downcast conversion
  - Flag in the high bits to indicate replacement

![Diagram of downcast conversion between double and single precision, indicating non-signalling NaN representation.]

1  movsd 0x601e38(%rax, %rbx, 8) ➞ %xmm0

2  mulsd -0x78(%rsp) ➞ %xmm0

3  addsd -0x4f02(%rip) ➞ %xmm0

4  movsd %xmm0 ➞ 0x601e38(%rax, %rbx, 8)
Example


1. movsd 0x601e38(%rax, %rbx, 8) → %xmm0
   check/replace -0x78(%rsp) and %xmm0

2. mulss -0x78(%rsp) → %xmm0
   check/replace -0x4f02(%rip) and %xmm0

3. addss -0x20dd43(%rip) → %xmm0

4. movsd %xmm0 → 0x601e38(%rax, %rbx, 8)
Implementation

- original instruction in block
- block splits
- double ➔ single conversion
- initialization
- cleanup
00400000    addsd xmm0, xmm1
00606000  nop
00606001  mov qword ptr [rsp-0xb8], rax
00606009  mov rax, 0x0
00606013  lahf
00606014  seto al
00606017  mov qword ptr [rsp-0xc0], rax
0060601f  mov qword ptr [rsp-0xd0], rax
00606027  mov qword ptr [rsp-0xe0], rcx
00606037  mov qword ptr [rsp-0xe0], rcx
0060603e  cmp rbx, rax
00606041  mov qword ptr [rax]
00606054  mov rax, 0x605000
0060605c  movlpd xmm1, qword ptr [rax]
00606061  mov rax, 0x0
00606069  mov rax, 0xffffffff
00606070  mov qword ptr [rsp-0xe8], xmm0
00606078  and qword ptr [rsp-0xe8], rcx
00606080  or qword ptr [rsp-0xe8], rax
0060608b  movlpd xmm0, qword ptr [rsp-0xe8]
00606091  cvtsd2ss xmm1, xmm1
00606099  mov rax, 0xffffffff
006060a7  and rax, rbx
006060a9  mov rax, 0x7f4dead00000000
006060ab  movq rax, xmm0
006060b0  addss xmm0, xmm1
006060b8  mov qword ptr [rsp-0xb8], rax
006060bc  mov qword ptr [rsp-0xd0], rax
006060c3  cvtsd2ss xmm0, xmm0
006060c7  mov qword ptr [rsp-0xe8], xmm1
006060c9  mov rcx, 0xffffffff
006060d0  and rax, rcx
006060da  mov rax, qword ptr [rsp-0xb8]
Configuration
Configuration Search

• Helper script
  o Generates and tests a variety of configurations
  o Keeps a “work queue” of untested configurations
  o Brute-force attempt to find maximal replacement

• Algorithm:
  o Initially, build individual configurations for each module and add them to the work queue
  o Retrieve the next available configuration and test it
    o If it passes, add it to the final configuration
    o If it fails, build individual configurations for any child members (functions, basic blocks, instructions) and add them to the queue
  o Build and test the final configuration
Demo
Observations

• **Representation-dependent operations**
  - Random number generators
  - “Libm” functions (log, exp, sin, cos, etc.)

• **Accumulators**
  - Multiple operations concluded with an addition
  - Requires double-precision only for the final addition

• **Minimal dataset-based variation**
  - Larger variation due to optimization level
Future Work

Automated configuration tuning

System Inputs

Threshold = 0.10

Original program & error threshold
(all double-precision)

Candidate replacement configurations
(shaded = single-precision)

Candidate errors
(compared to original)

Chosen configuration
(under threshold w/ max replacement)
Conclusion

Automated instrumentation techniques can be used to implement **mixed-precision configurations** for floating-point code, and there is much opportunity in the future for **automated precision recommendations**.
Acknowledgements

Jeff Hollingsworth, University of Maryland (Advisor)

Drew Bernat, University of Wisconsin

Bronis de Supinski, LLNL
Barry Rountree, LLNL
Greg Lee, LLNL

Matt Legendre, LLNL
Dong Ahn, LLNL

http://sourceforge.net/projects/crafthpc/