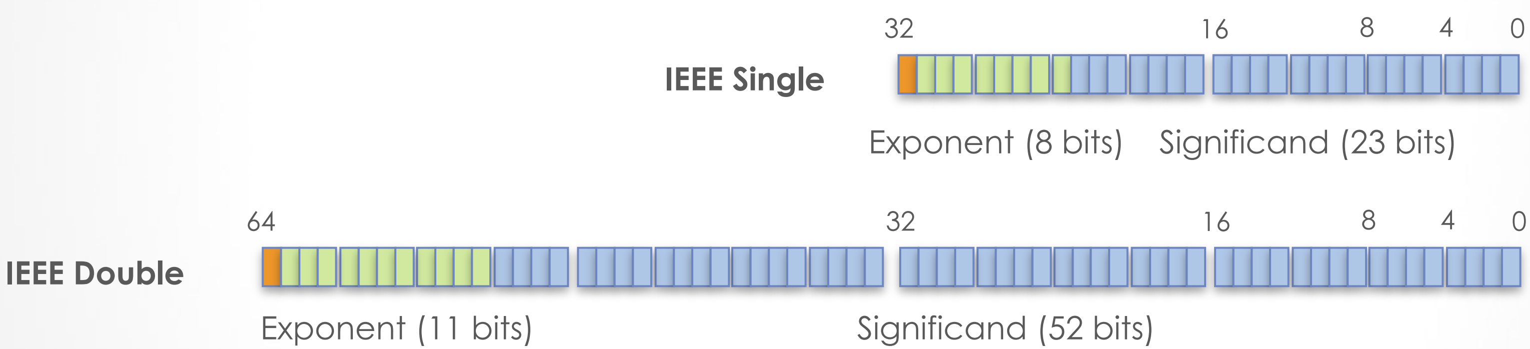
Optimization of Floating-point Precision using Binary Modification

Michael Lam
University of Maryland, College Park

Jeff Hollingsworth, Advisor

Background

- Floating-point represents real numbers as $(\pm\ \mathrm{sgnf} \times 2^{\mathrm{exp}})$
  - 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 10011001 10011010

Double-precision

Images courtesy of BinaryConvert.com

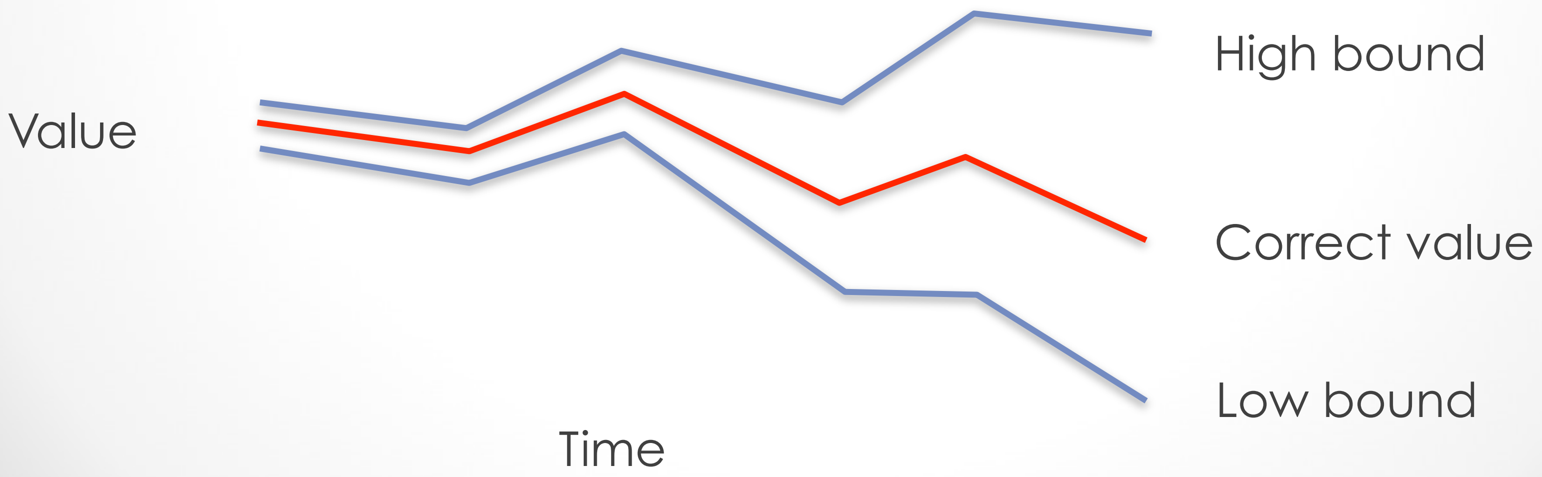
Motivation

- Finite precision causes round-off error
  - Compromises "ill-conditioned" calculations
  - Hard to detect and diagnose
- Increasingly important as HPC scales
  - Double-precision data movement is a bottleneck
  - Streaming processors are faster in single-precision (~2x)
  - Need to balance speed (singles) and accuracy (doubles)

Previous Work

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

- Interval/affine arithmetic (Goubault 2001)
  - Conservative static error bounds are largely unhelpful



Previous Work

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

```
LU ← PA
solve Ly = Pb
solve Ux_0 = y
for k = 1, 2, ... do
        r_k ← b – Ax_{k-1}
        solve Ly = Pr_k
        solve Uz_k = y
        x_k ← x_{k-1} + z_k
        check for convergence
end for
```

Mixed-precision linear solver algorithm

Red text indicates steps performed in double-precision (all other steps are single-precision)

- 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

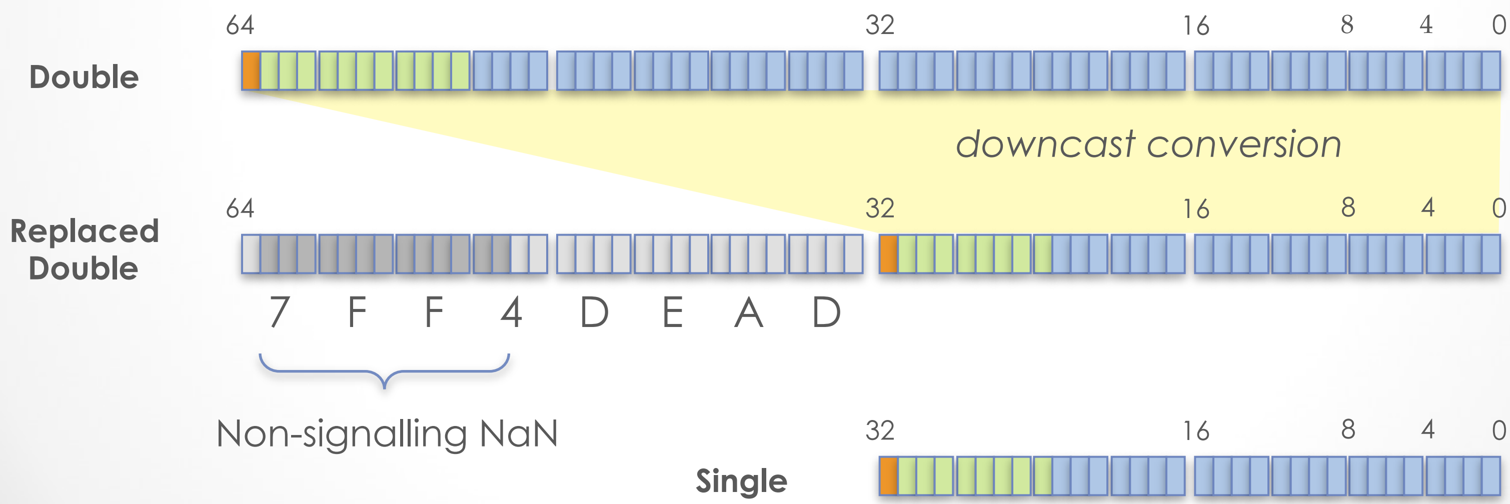
- Static binary instrumentation
  - Controlled by configuration settings
  - Replace floating-point instructions with new code
  - Re-write a modified binary
- Dynamic analysis
  - Run modified program on representative data set
  - Produces results and recommendations

Advantages

- Automated
  - Minimize developer effort
  - Ensure consistency and correctness
- Binary-level
  - Include shared libraries without source code
  - Include compiler optimizations
- Runtime
  - 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



Example

```
gvec[i,j] = gvec[i,j] * lvec[3] + gvar

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

2   mulsd -0x78(%rsp) → %xmm0

3   addsd -0x4f02(%rip) → %xmm0

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

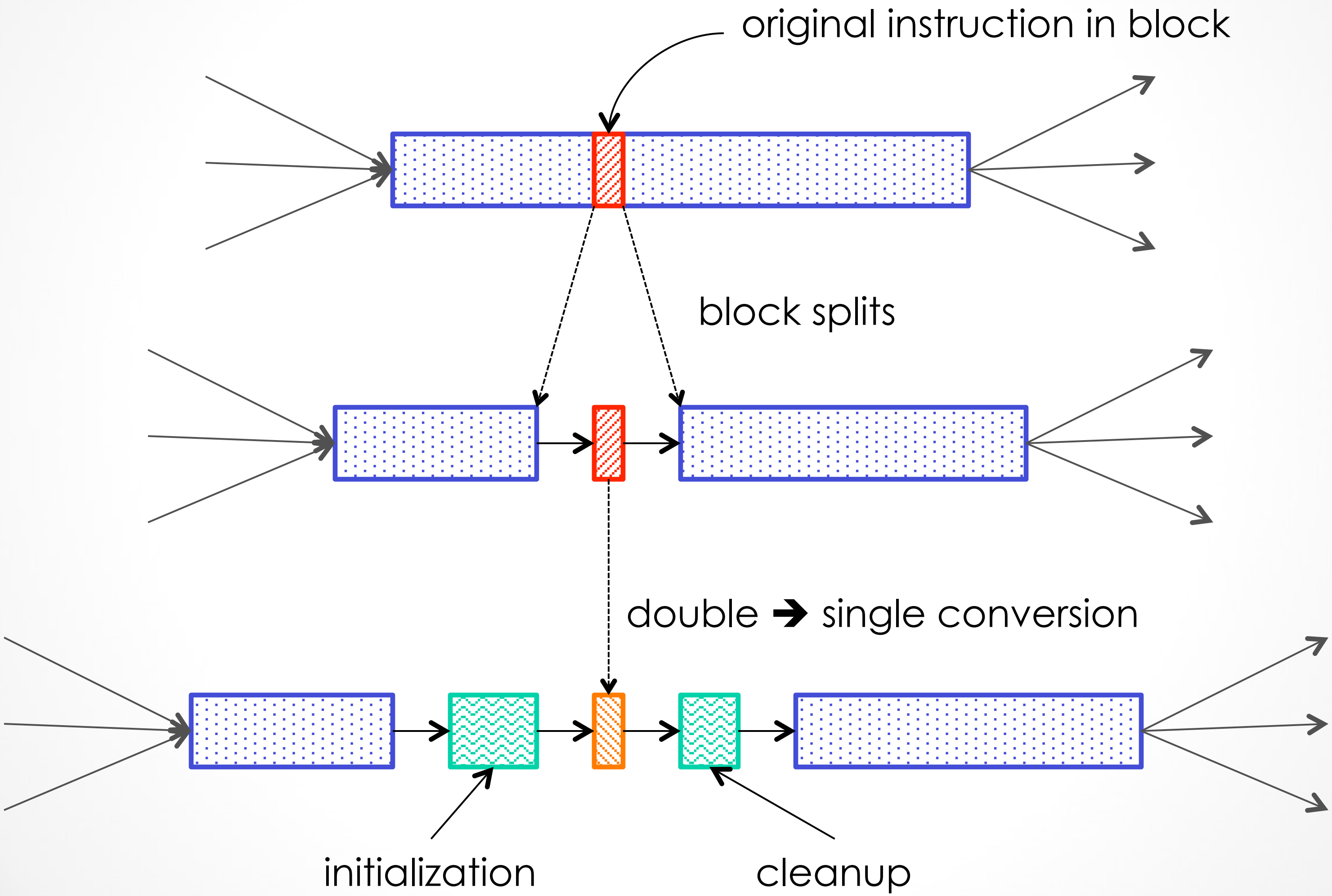
Example

```
gvec[i,j] = gvec[i,j] * lvec[3] + gvar

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



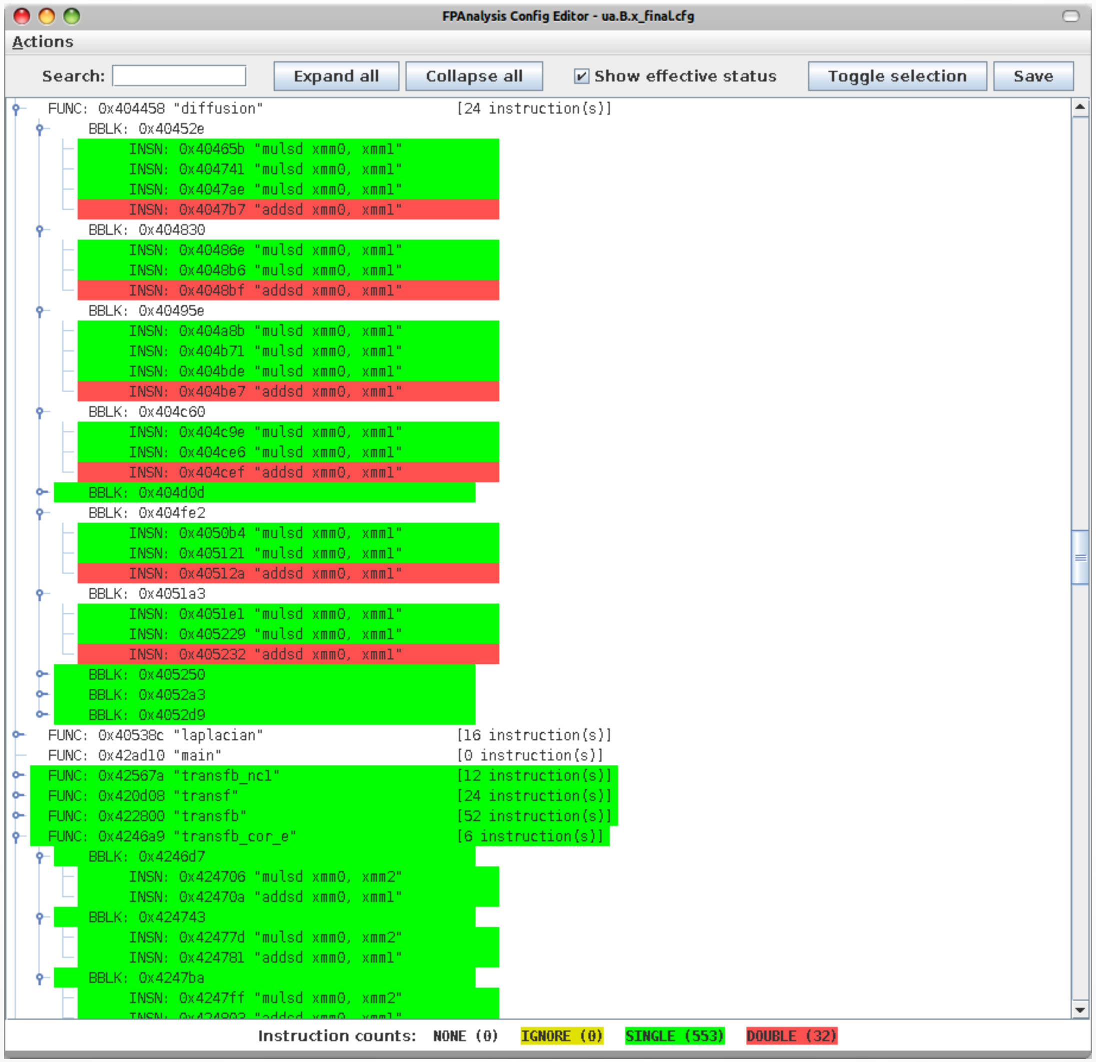
```
00400000        addsd xmm0, xmm1
```

```
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-0xd8], rbx
0060602f    mov qword ptr [rsp-0xe0], rcx
00606037    movq rax, xmm0
0060603c    mov rbx, 0x7fffffff00000000
00606046    and rax, rbx
00606049    mov rbx, 0x7ff4dead00000000
00606053    cmp rbx, rax
00606056    jz 0x7fff1d923f6c
006060a2    movq rax, xmm1
006060a7    mov rbx, 0x7fffffff00000000
006060b1    and rax, rbx
006060b4    mov rbx, 0x7ff4dead00000000
006060be    cmp rbx, rax
006060c1    jz 0x7fff1d923f6c
0060605c    cvtsd2ss xmm0, xmm0
00606060    mov eax, 0x7ff4dead
00606065    mov rcx, 0xffffffff
0060606f    and rax, rcx
00606072    rol rax, 0x20
00606076    movlpd qword ptr [rsp-0xe8], xmm0
0060607f    mov rcx, 0xffffffff
00606089    and qword ptr [rsp-0xe8], rcx
00606091    or qword ptr [rsp-0xe8], rax
00606099    movlpd xmm0, qword ptr [rsp-0xe8]
006060c7    cvtsd2ss xmm1, xmm1
006060cb    mov eax, 0x7ff4dead
006060d0    mov rcx, 0xffffffff
006060da    and rax, rcx
006060dd    rol rax, 0x20
006060e1    movlpd qword ptr [rsp-0xe8], xmm1
006060ea    mov rcx, 0xffffffff
006060f4    and qword ptr [rsp-0xe8], rcx
006060fc    or qword ptr [rsp-0xe8], rax
00606104    movlpd xmm1, qword ptr [rsp-0xe8]
0060610d    mov rax, 0x605000
00606117    mov rbx, qword ptr [rax]
0060611a    inc qword ptr [rbx+0xf8]
00606121    mov rcx, qword ptr [rsp-0xe0]
00606129    mov rbx, qword ptr [rsp-0xd8]
00606131    mov rax, qword ptr [rsp-0xd0]
00606139    mov rax, qword ptr [rsp-0xb8]

00606141    addss xmm0, xmm1

00606145    mov qword ptr [rsp-0xb8], rax
0060614d    mov qword ptr [rsp-0xd0], rax
00606155    mov qword ptr [rsp-0xd8], rcx
0060615d    mov eax, 0x7ff4dead
00606162    mov rcx, 0xffffffff
0060616c    and rax, rcx
0060616f    rol rax, 0x20
00606173    movlpd qword ptr [rsp-0xe0], xmm0
0060617c    mov rcx, 0xffffffff
00606186    and qword ptr [rsp-0xe0], rcx
0060618e    or qword ptr [rsp-0xe0], rax
00606196    movlpd xmm0, qword ptr [rsp-0xe0]
0060619f    mov rcx, qword ptr [rsp-0xd8]
006061a7    mov rax, qword ptr [rsp-0xd0]
006061af    mov rax, qword ptr [rsp-0xc0]
006061b7    add al, 0x7f
006061ba    sahf
006061bb    mov rax, qword ptr [rsp-0xb8]
006061c3    nop
```

Configuration



Configuration Search

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

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

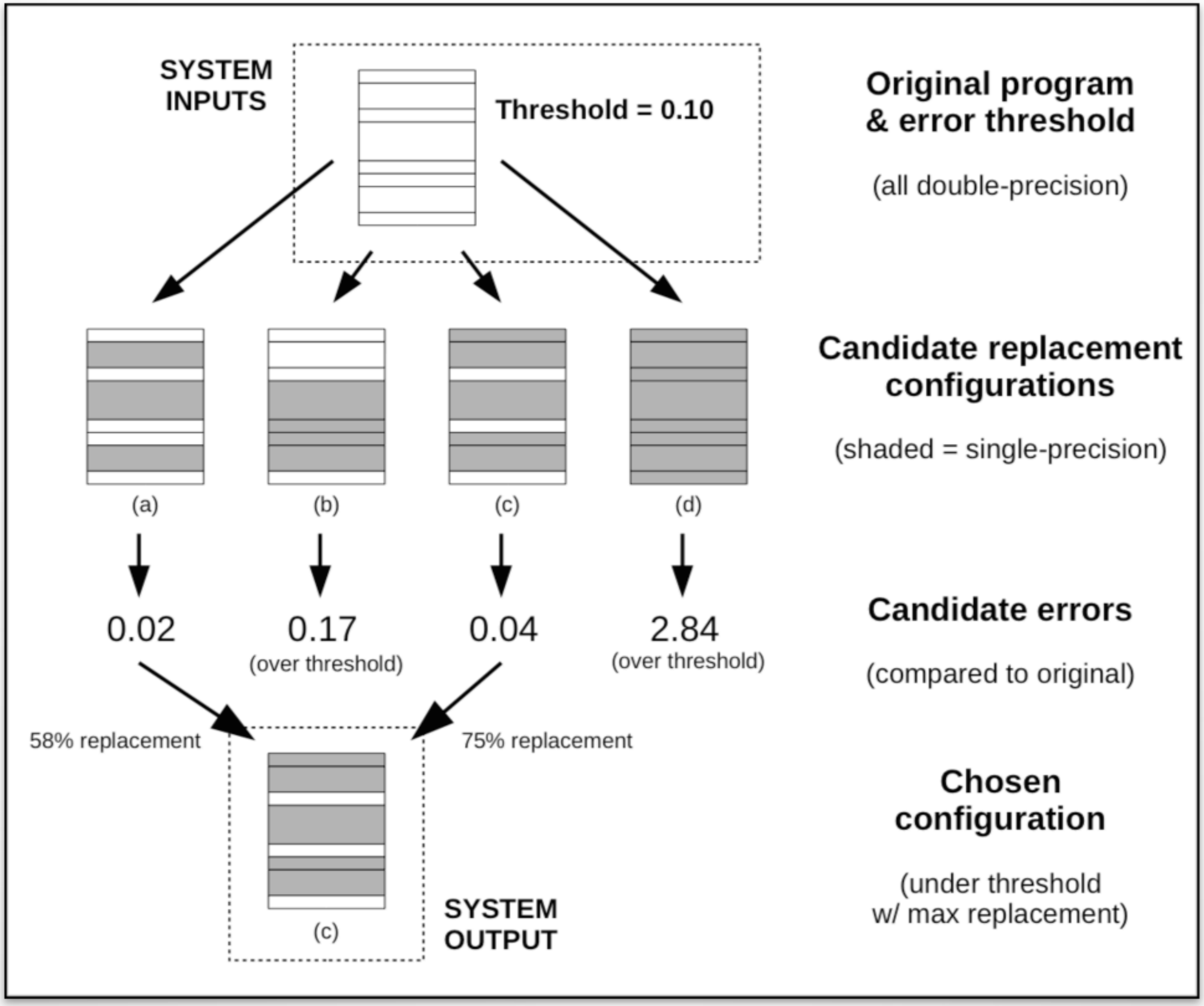
Demo

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



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

Matt Legendre, LLNL

Greg Lee, LLNL

Dong Ahn, LLNL

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