Software Tools for Mixed-Precision Program Analysis

Mike Lam

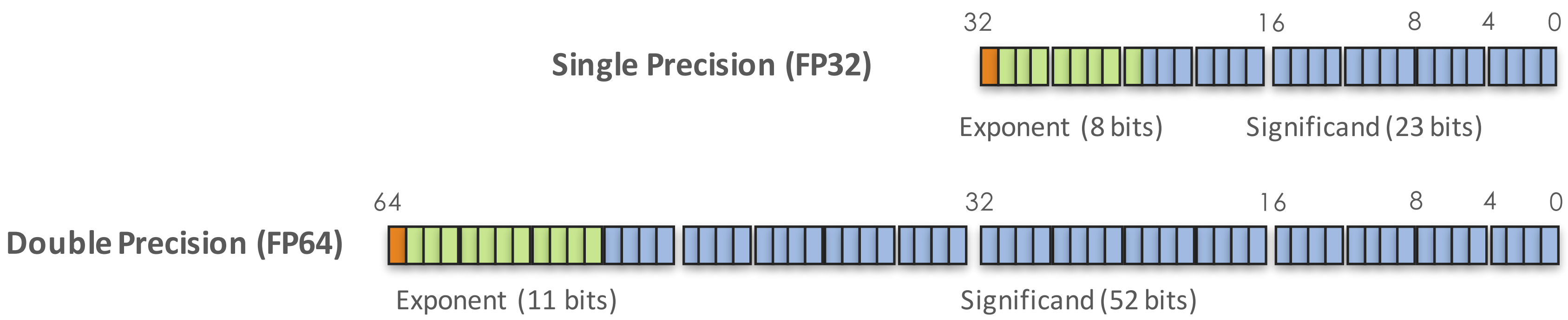
James Madison University
Lawrence Livermore National Lab

About Me

- Ph.D in CS from University of Maryland ('07-'14)
  - Topic: Automated floating-point program analysis (w/ Jeff Hollingsworth)
  - Intern @ Lawrence Livermore National Lab (LLNL) in Summer ’11
- Assistant professor at James Madison University since '14
  - Teaching: computer organization, parallel & distributed systems, compilers, and programming languages
  - Research: high-performance analysis research group (w/ Dee Weikle)
- Faculty scholar @ LLNL since Summer '16
  - Energy-efficient computing project (w/ Barry Rountree)
  - Variable precision computing project (w/ Jeff Hittinger et al.)

Context

- IEEE floating-point arithmetic
  - Ubiquitous in scientific computing
  - More bits => higher accuracy (usually)
  - Fewer bits => higher performance (usually)



Motivation

- Vector single precision 2X+ faster
  - Possibly better if memory pressure is alleviated
  - Newest GPUs use mixed precision for tensor ops

Operation	FP32	Packed FP32	FP64
Add	6	6	6
Subtract	6	6	6
Multiply	6	6	6
Divide	27	32	42
Square root	28	38	43

Instruction latencies for Intel Knights Landing

	Tesla V100 PCIe	Tesla V100 SXM2	
GPU Architecture	NVIDIA Volta		
NVIDIA Tensor Cores	640		
NVIDIA CUDA® Cores	5,120		
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS	FP64
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS	FP32
Tensor Performance	112 TFLOPS	125 TFLOPS	Mixed FP16 / FP32

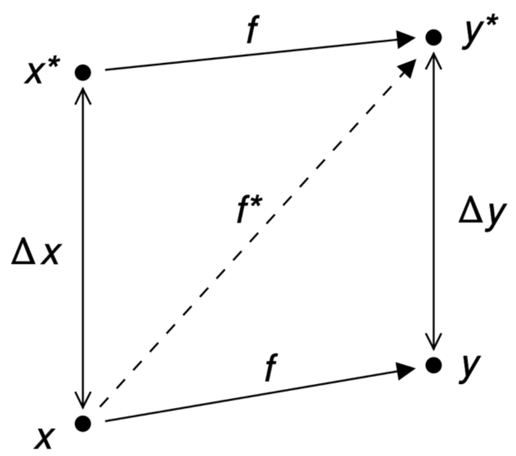
Credit: https://agner.org/optimize/ and NVIDIA Tesla V100 Datasheet

Questions

- How many bits do you *need*?
- Where does reduced precision *help*?

Prior Approaches

- Rigorous: forwards/backwards error analysis
  - Requires numerical analysis expertise
- Pragmatic: "guess-and-check"
  - Requires manual code conversion effort




```
//double x[N], y[N];
float x[N], y[N];
double alpha;
```

Credit: Wikimedia Commons

Research Question

- What can we learn about floating-point behavior with **automated** analysis?
  - Specifically: can we build *mixed-precision* versions of a program automatically?
- Caveat: few (or no) formal guarantees
  - Rely on user-provided representative run (and sometimes a verification routine)

```
double sum = 0.0;

void sum2pi_x()
{
  double tmp;
  double acc;
  int i, j;

  [...]
```

→

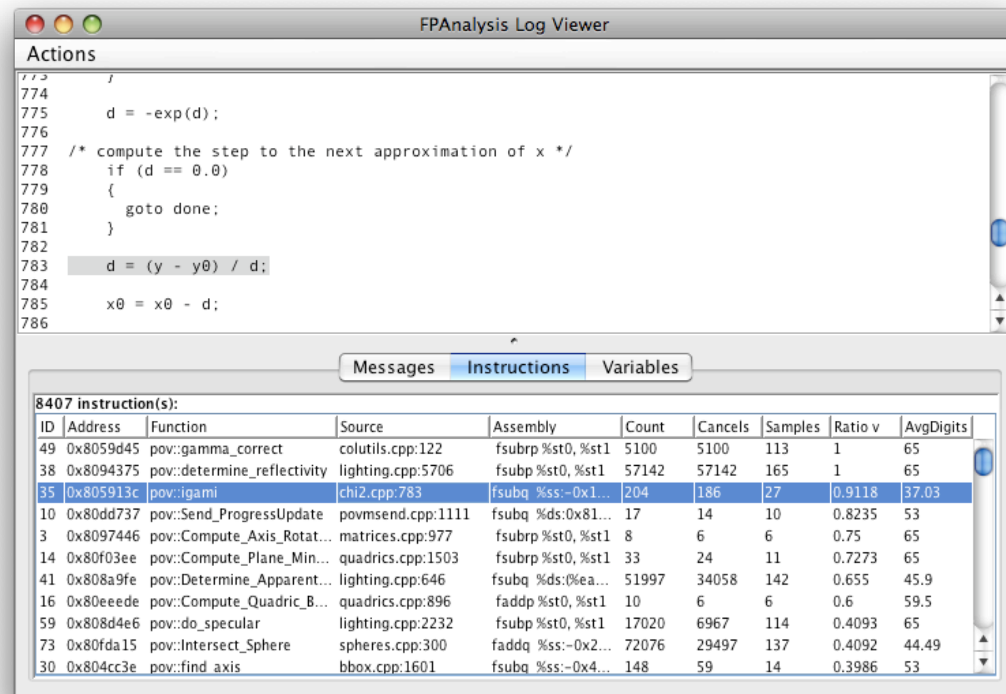
```
double sum = 0.0;

void sum2pi_x()
{
  float tmp;
  float acc;
  int i;
  int j;

  [...]
```

FPAnalysis / CRAFT (2011)

- Dynamic binary analysis via Dyninst
- Cancellation detection
- Range (exponent) tracking

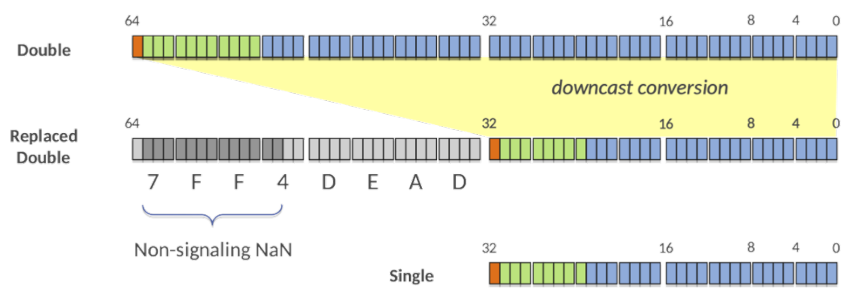
```
  3.682236
- 3.682234
  0.000002
```

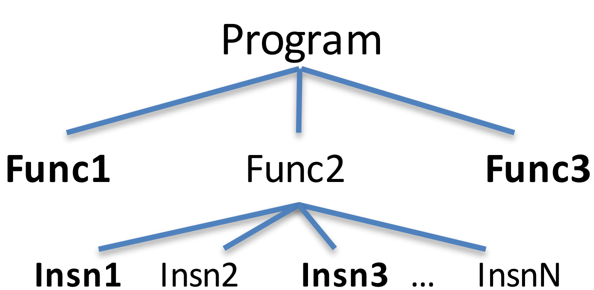
(6 digits cancelled)

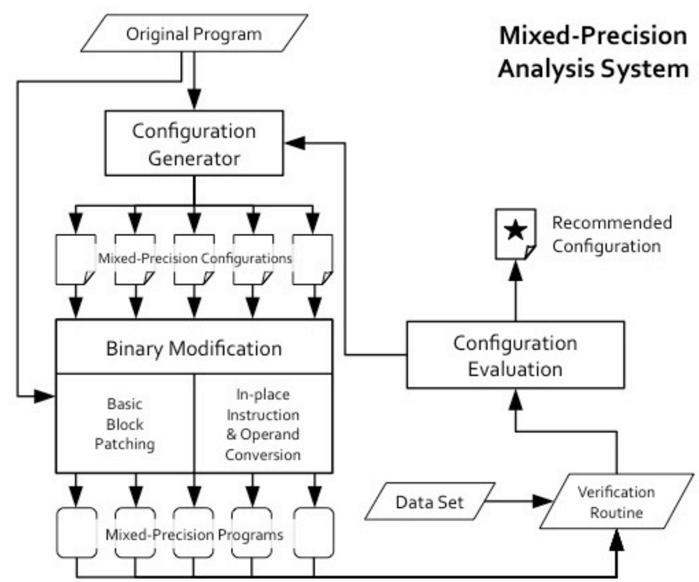


CRAFT (2013)

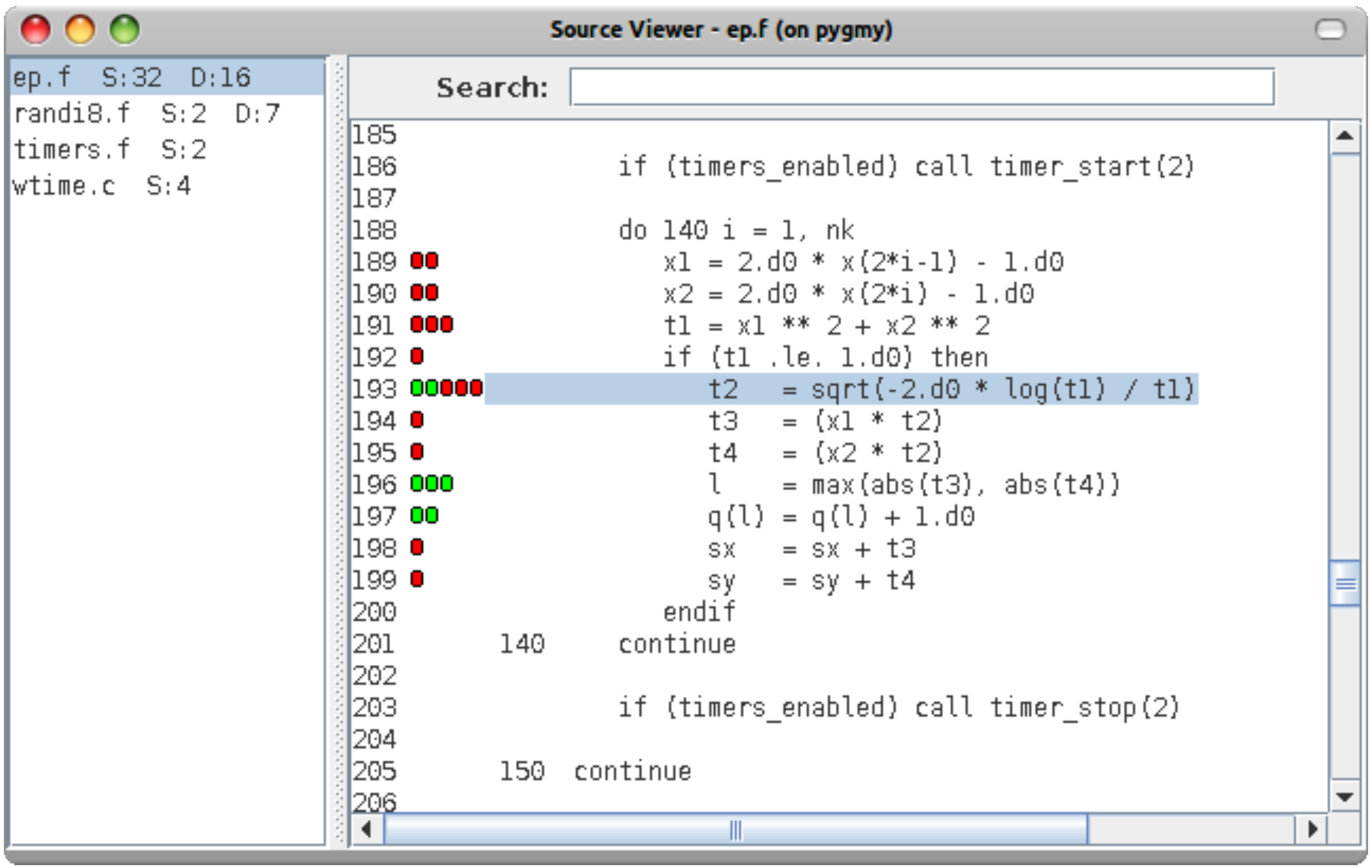
- Dynamic binary analysis via Dyninst
- Instruction-level replacement of doubles w/ floats
- Hierarchical search for valid replacements







CRAFT (2013)


Source Viewer - ep.f (on pygmy)
ep.f S:32 D:16
randi8.f S:2 D:7
timers.f S:2
wtime.c S:4
Search:
if (timers_enabled) call timer_start(2)
do 140 i = 1, nk
x1 = 2.d0 * x(2*i-1) - 1.d0
x2 = 2.d0 * x(2*i) - 1.d0
t1 = x1 ** 2 + x2 ** 2
if (t1 .le. 1.d0) then
t2 = sqrt(-2.d0 * log(t1) / t1)
t3 = (x1 * t2)
t4 = (x2 * t2)
l = max(abs(t3), abs(t4))
q(l) = q(l) + 1.d0
sx = sx + t3
sy = sy + t4
endif
140 continue
if (timers_enabled) call timer_stop(2)
150 continue

CRAFT (2013)

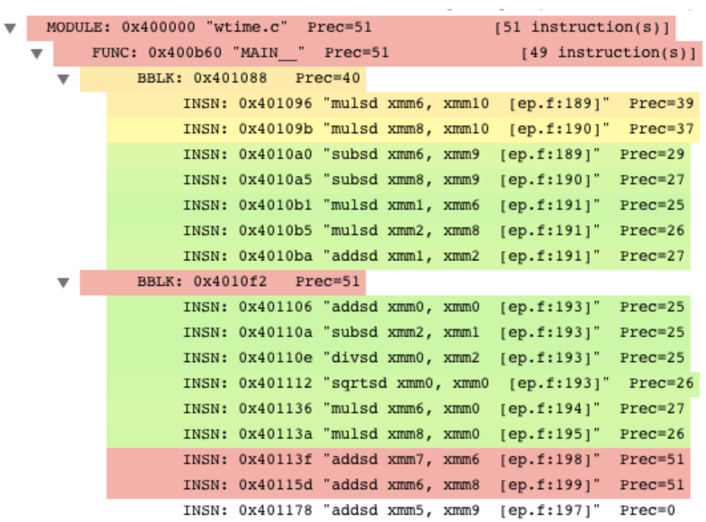
NAS Benchmark (name.CLASS)	Candidate Instructions	Configurations Tested	% Dynamic Replaced
bt.A	6,262	4,000	78.6
cg.A	956	255	5.6
ep.A	423	114	45.5
ft.A	426	74	0.2
lu.A	6,014	3,057	57.4
mg.A	1,393	437	36.6
sp.A	4,507	4,920	30.5

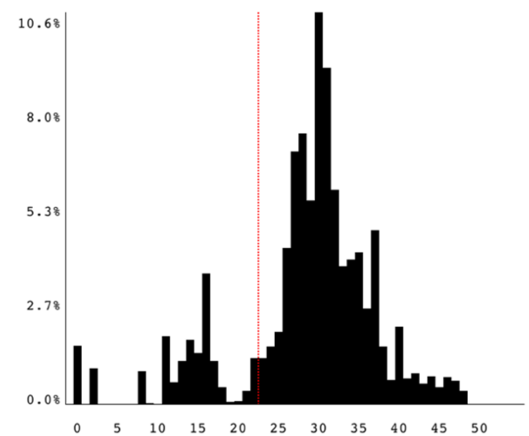
Issues

- High overhead
  - Must check and (possibly) convert operands before each instruction
- Lengthy search process
  - Search space is exponential wrt. instruction count
- Coarse-grained analysis
  - Binary decision: single or double

CRAFT (2016)

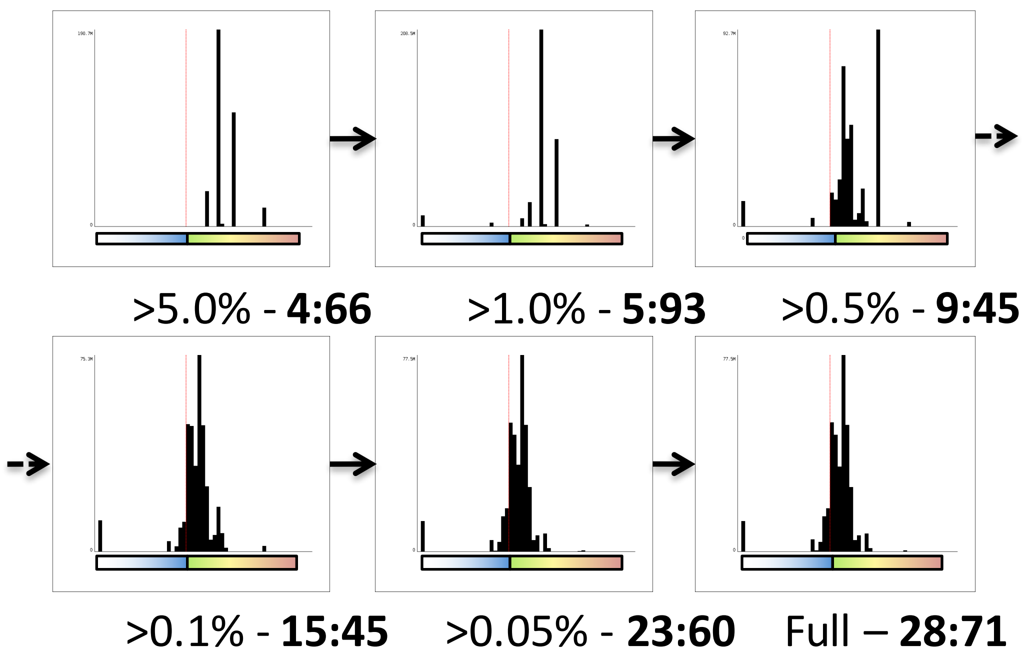
- Reduced-precision analysis
  - Simulate conservatively via bit-mask truncation
  - Report min output precision for each instruction
  - Finer-grained analysis and lower overhead





CRAFT (2016)

- Scalability via heuristic search
  - Focus on most-executed instructions
  - Analysis time vs. benefit tradeoff



Issue

- Only considers precision reduction
  - No higher precision or arbitrary-precision
  - No alternative representations
  - No dynamic tracking of error

SHVAL (2016)

- Generic floating-point shadow value analysis
  - Maintain “shadow” value for every memory location
  - Execute shadow operations for all computation
  - Shadow type is parameterized (native, MPFR, Unum, Posit, etc.)
  - Pintool: less overhead than similar frameworks like Valgrind

```
double sum = 0.0;
for (int i = 0; i < 10; i++) {
    sum += 0.1;
}
printf("%25.20f\n", sum);
```

Fig. 3. Sample C program

```
Original machine code:                              Inserted shadow code:

  pxor    xmm0, xmm0         (set to 0.0)             xmm[0] = convert(0.0)
  mov     eax, 10
  movsd   xmm1, 0x400628     (load 0.1)               xmm[1] = convert(*(0x400628))
loop:
  sub     eax, 1
  addsd   xmm0, xmm1         (increment)              xmm[0] += xmm[1]
  jne     loop
  movsd   0x8(rsp), xmm0     (store sum)              mem[rsp+0x8] = xmm[0]
```

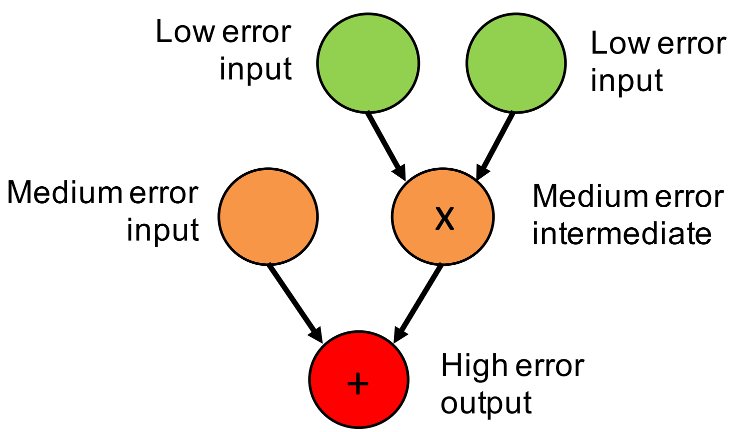
Fig. 4. Compiled assembly of program from Figure 3

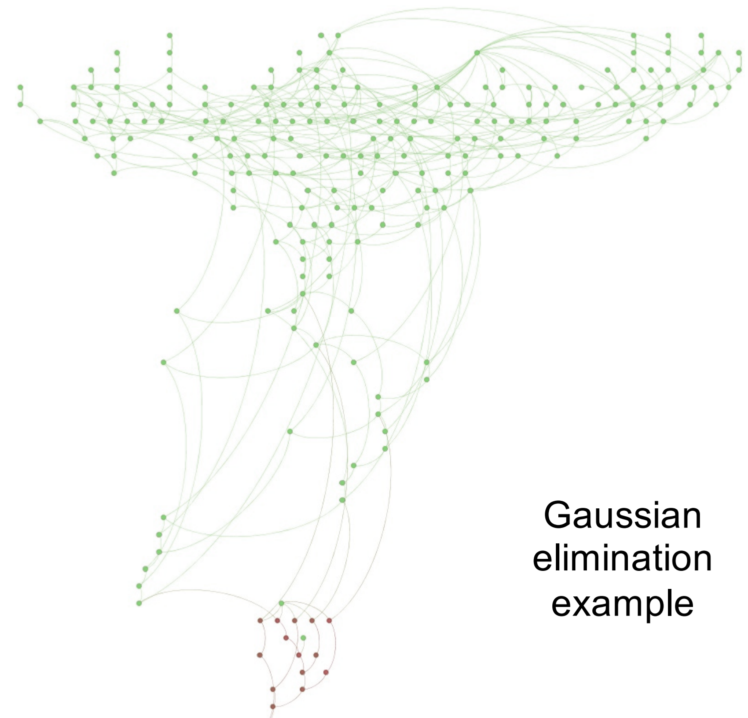
Shadow Value Type	Exp Size	Frac Size	Final Shadow Value	Relative Error
32-bit (native single)	8	23	1.000000	1.19e-07
64-bit (native double)	11	52	1.000000000000000	0
128-bit GNU MPFR	15	112	1.00000000000000005551e+00	1.11e-16
Unum (3,2)	8	4	(0.9375, 1.1875)	0.059
Unum (3,4)	8	16	(0.9999847412109375, 1.0000457763671875)	1.53e-05
Unum (4,6)	16	64	1.00000000000000005551...182	1.11e-16

TABLE I
ANALYSIS RESULTS ON SAMPLE PROGRAM

SHVAL (ongoing)

- Single precision shadow values
  - Trace execution and build data flow graph
  - Color nodes by error w.r.t. original double precision values
  - Highlights high-error regions
  - Inherent scaling issues





Gaussian elimination example

Issue

- No source-level mixed precision
  - Difficult to translate instruction-level analysis results to source-level transformations
  - Some users might be satisfied with opaque compiler-based optimization, but most HPC users want to know what changed!

CRAFT (2013)

- Memory-based replacement analysis
  - Leave computation intact but round outputs
  - Aggregate instructions that modify same variable
  - Found several valid variable-level replacements

NAS Benchmark (name.CLASS)	Candidate Operands	Configurations Tested	% Executions Replaced
bt.A	2,342	300	97.0
cg.A	287	68	71.3
ep.A	236	59	37.9
ft.A	466	108	46.2
lu.A	1,742	104	99.9
mg.A	597	153	83.4
sp.A	1,525	1,094	88.9

SHVAL (2017)

- Single-vs-double shadow value analysis
  - Aggregate error by instruction or memory location over time
- Computer vision case study (Apriltags)
  - 1.7x speedup on average with only 4% error
  - 40% energy savings in embedded experiments

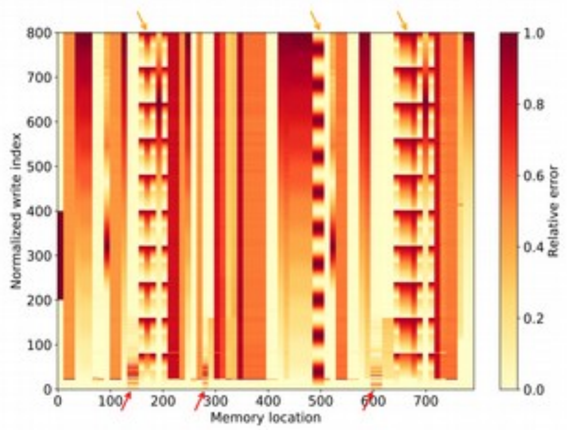


Fig. 1. Error trace per memory location. A darker pixel indicates higher error.

Credit: Ramy Medhat (ramy.medhat@uwaterloo.ca)

Issues

- Each instruction or variable is tested in isolation
  - Union of valid replacements is often invalid
- Cannot ensure speedup
  - Instrumentation overhead
  - Added casts to convert data between regions
  - Lack of vectorization and data packing

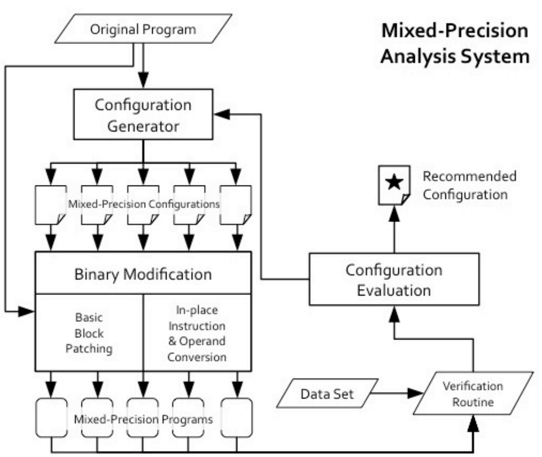
CRAFT (ongoing)

- Variable-centric mixed precision analysis
  - Use TypeForge (an AST-level type conversion tool) for source-to-source mixed precision
- Search for best speedup
  - Run full compiler backend w/ optimizations
  - Report fastest configuration that passes verification

```
double sum = 0.0;

void sum2pi_x()
{
  double tmp;
  double acc;
  int i, j;

  [...]
```

→

```
double sum = 0.0;

void sum2pi_x()
{
  float tmp;
  float acc;
  int i;
  int j;

  [...]
```



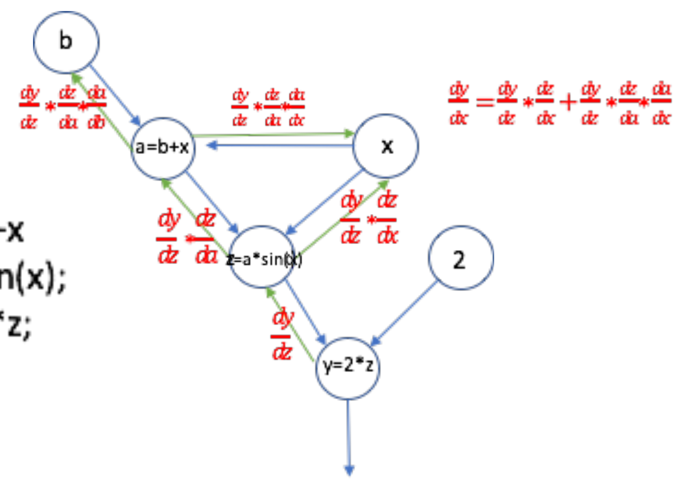
Related Work

- CRAFT/SHVAL, Precimonious [Rubio'13], GPUMixer [Laguna'19], etc.
  - Very **practical**
  - Widely-used tool frameworks (Dyninst, Pin, LLVM)
  - Few (or no) formal guarantees
  - Tested on HPC benchmarks on Linux/x86
- Daisy [Darulova'18], FPTuner [Chiang'17], etc.
  - Very **rigorous**
  - Custom input formats
  - Provable error bounds for given input range
  - Impractical for HPC benchmarks

ADAPT (2018)

- Automatic backwards error analysis
  - Obtain gradients via reverse-mode algorithmic differentiation (CoDiPack or TAPENADE)
  - Calculate error contribution of intermediate results
  - Aggregate by program variable
  - Greedy algorithm builds mixed-precision allocation

```
a = b+x
z = a*sin(x);
y = 2*z;
```



Credit: Harshitha Menon (gopalakrishn1@llnl.gov)

ADAPT (2018)

Original C Code

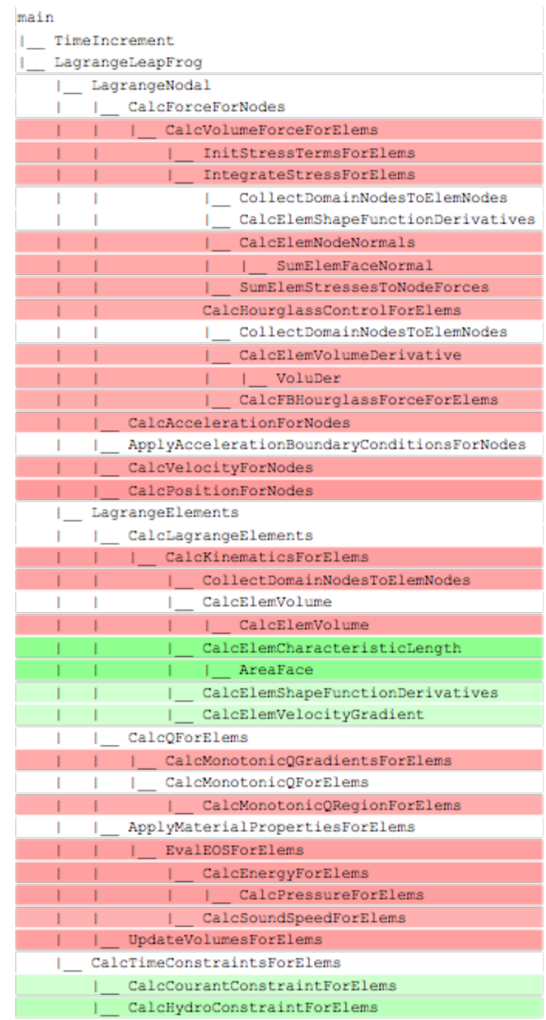
```
#include <iostream>



double sum = 0.0;
double inc = 0.1;

double do_sum() {
    int i;
    for (i = 0; i < 1000; i++) {
        sum += inc;
    }
    return sum;
}

int main() {


    do_sum();
    cout << sum << endl;





    return 0;
}
```

AD Instrumented Code

```
#include <iostream>
#include <adapt.h>                     – AD Libraries
#include <adapt-impl.cpp>

AD_real sum = 0.0;
AD_real inc = 0.1;                     – Type Changes

AD_real do_sum() {
    int i;
    for (i = 0; i < 1000; i++) {
        sum += inc;
    }
    return sum;
}

int main() {
    AD_begin();                          – Initialization
    AD_independent(inc, "inc");
    do_sum();
    cout << AD_value(sum) << endl;

    AD_dependent(sum, "sum", 8);
    AD_report();                         – Output
    return 0;
}
```

ADAPT (2018)

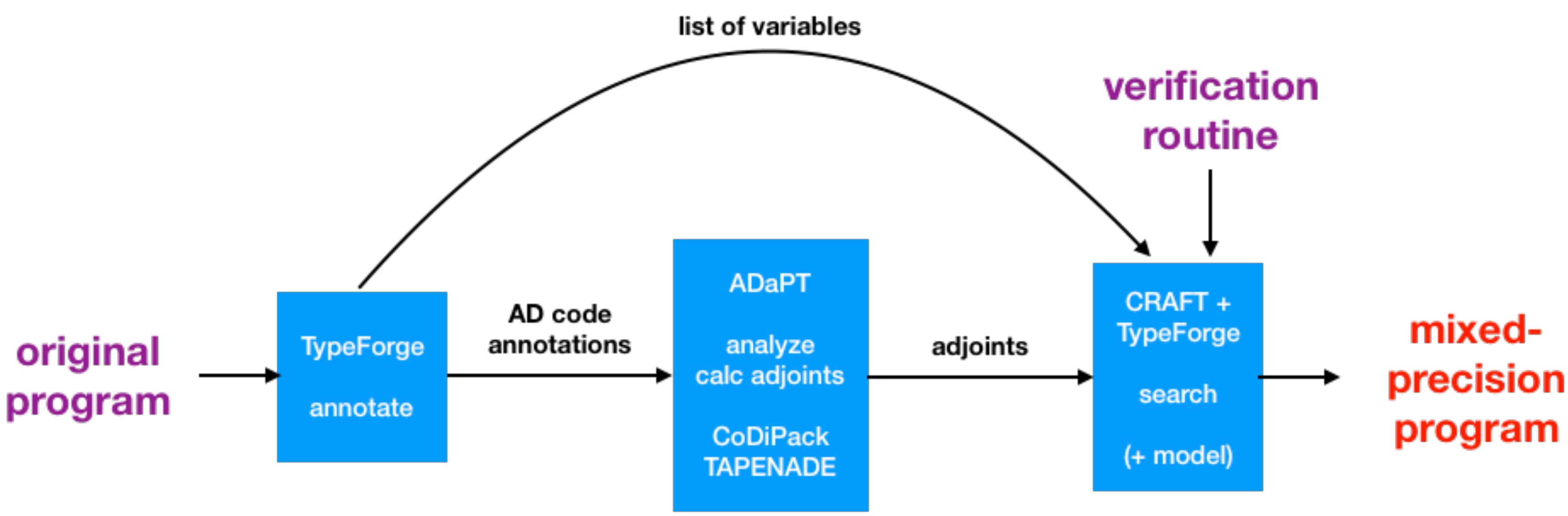
- Used ADAPT on LULESH benchmark to help develop a mixed-precision CUDA version
- Achieved speedup of 20% within original error threshold on NVIDIA GK110 GPU



Credit: Harshitha Menon (gopalakrishn1@llnl.gov)

FloatSmith (ongoing)

- Mixed-precision search via CRAFT
- Source-to-source translation via TypeForge
- Optionally, use TypeForge-automated ADAPT analysis to narrow search and provide more rigorous guarantees



FloatSmith (ongoing)

- Guided mode (Q&A)
- Batch mode (command-line parameters)
- Dockerfile provided
- Can offload configuration testing to a cluster

```
floatsmith -B --run "./demo"
```

```
double p = 1.00000003;
double l = 0.00000003;
double o;

int main() {
  o = p + l;
  // should print 1.00000006
  printf("%.8f\n", (double)o);
  return 0;
}
```

→

```
double p = 1.00000003;
float l = 0.00000003;
double o;

int main() {
  o = p + l;
  // should print 1.00000006
  printf("%.8f\n", (double)o);
  return 0;
}
```

FPHPC (ongoing)

- Benchmark suite aimed at facilitating scale-up for mixed-precision analysis tools
  - “Middle ground” between real-valued expressions and full applications
  - Currently looking for good case studies

Future Work

- (Better) OpenMP/MPI support
- (Better) GPU and FPGA support
- Model-based performance prediction
- Dynamic runtime precision tuning
- Ensemble floating-point analysis

Summary

- Automated mixed precision is possible
  - Practicality vs. rigor tradeoff
- Multiple active projects
  - Various goals and approaches
  - All target HPC applications
- Many avenues for future research

Papers

- **CRAFT**
  - **2016**: Michael O. Lam and Jeffrey K. Hollingsworth. “Fine-Grained Floating-Point Precision Analysis.” Int. J. High Perform. Comput. Appl. 32, 2 (March 2018), 231-245.
  - **2013**: Michael O. Lam, Jeffrey K. Hollingsworth, Bronis R. de Supinski, and Matthew P. Legendre. “Automatically Adapting Programs for Mixed-Precision Floating-Point Computation.” In Proceedings of the International Conference on Supercomputing (ICS '13). ACM, New York, NY, USA, 369-378.
  - **2011**: Michael O. Lam, Jeffrey K. Hollingsworth, and G. W. Stewart. “Dynamic Floating-Point Cancellation Detection.” Parallel Comput. 39, 3 (March 2013), 146-155.

- **SHVAL**
  - **2017**: Ramy Medhat, Michael O. Lam, Barry L. Rountree, Borzoo Bonakdarpour, and Sebastian Fischmeister. “Managing the Performance/Error Tradeoff of Floating-point Intensive Applications.” ACM Trans. Embed. Comput. Syst. 16, 5s, Article 184 (October 2017), 19 pages.
  - **2016**: Michael O. Lam and Barry L. Rountree. “Floating-Point Shadow Value Analysis.” In Proceedings of the 5th Workshop on Extreme-Scale Programming Tools (ESPT '16). IEEE Press, Piscataway, NJ, USA, 18-25.

- **ADAPT**
  - **2018**: Harshitha Menon, Michael O. Lam, Daniel Osei-Kuffuor, Markus Schordan, Scott Lloyd, Kathryn Mohror, and Jeffrey Hittinger. “ADAPT: Algorithmic Differentiation Applied to Floating-Point Precision Tuning.” In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18). IEEE Press, Piscataway, NJ, USA, Article 48.

Acknowledgements

Jeff Hollingsworth
Bronis de Supinski
Barry Rountree
Jeff Hittinger
Matthew Legendre
Scott Lloyd
Harshitha Menon
Markus Schordan
Tristan Vanderbruggen
Ramy Medhat
Nathan Pinnow
Shelby Funk
Dee Weikle
Garrett Folks
Logan Moody
Nkeng Atabong

U.S. Department of Energy

DE-CFC02-01ER25489, DE-FG02-01ER25510, DE-FC02-06ER25763, and DE-AC52-07NA27344

Lawrence Livermore National Laboratory

LDRD project 17-SI-004

James Madison University

various provost awards, college grants, and department student funding

Lawrence Livermore National Laboratory

JMU Department of Computer Science

Thank you!

github.com/crafthpc

github.com/llnl/adapt-fp

tinyurl.com/fpanalysis

Contact me:

lam2mo@jmu.edu

Lawrence Livermore National Laboratory

JMU Department of Computer Science