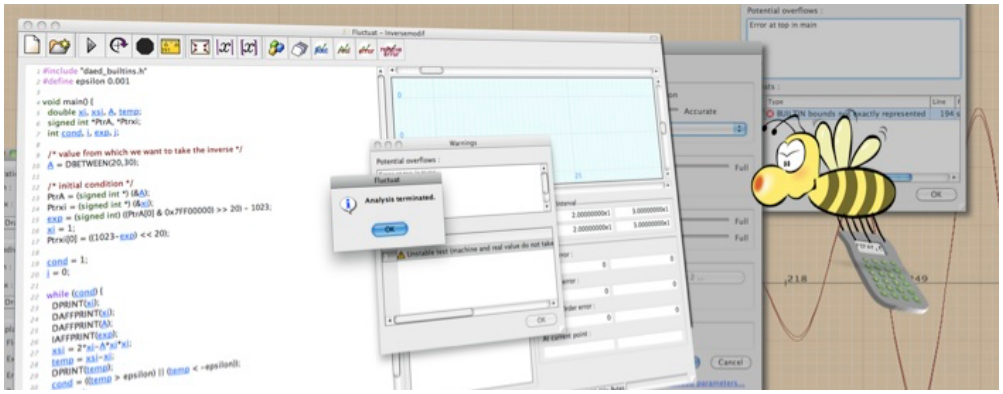
Floating-Point Analysis Tools

Session leader: Dr. Michael O. Lam, James Madison University

`tinyurl.com/fpanalysis`
`fpbench.org/community.html`

FLUCTUAT

- Abstract interpretation for C and Ada programs
- Interval arithmetic for guaranteed error bounds
- In development since 2001
- Targets safety-focused industrial applications
- Not open-source, but free for educational use



CRAFT and Precimonious

- Early precision auto-tuning projects
  - Both recently extended using shadow analysis
- **CRAFT** [ICS'13]
  - Mike Lam, JMU (prev. UMD)
  - Instruction-centric via binary analysis
- **Precimonious** [SC'13]
  - Cindy Rubio-Gonzáles, UC Davis (prev. UWisc-Madison and UC Berkeley)
  - Variable-centric using LLVM

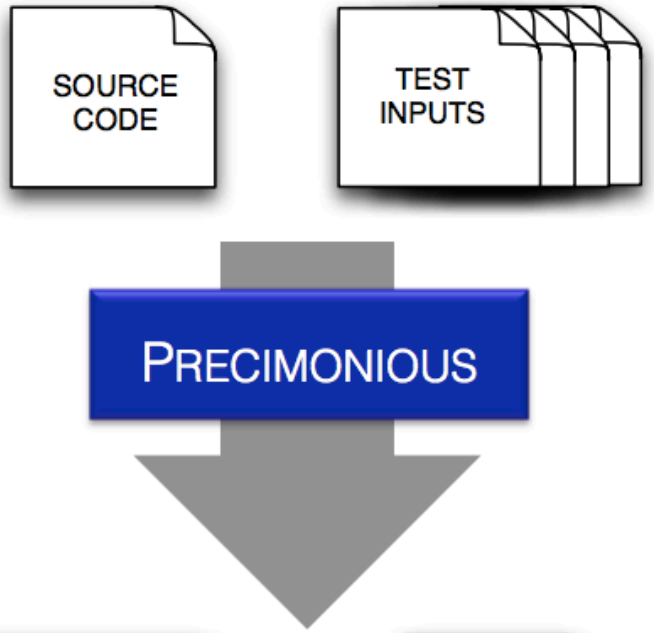
PRECIMONIOUS [SC'13]

"Parsimonious with Precision"

[Rubio et al. SC'13]

Dynamic Program Analysis for Floating-Point Precision Tuning

What error threshold to use?



If not available, random floating-point values

Proposed type assignment:
- ✓ Accurate enough answer
- ✓ Better performance

Modified program in executable format

Approach

- Based on the Delta-Debugging Search Algorithm [Zeller et. Al]
- Our definition of a change
  - Lowering floating-point point precision n the program
    - Example: double x → float x
- Our success criteria
  - Resulting program produces an “accurate enough” answer
  - Resulting program is faster than the original program
- Main idea:
  - Start by associating each variable with set of types
    - Example: x → {long double, double, float}
  - Refine set until it contains only one type
- Find a local minimum
  - Lowering the precision of one more variable violates success criteria

Experimental Results

	Original Type Configuration					Proposed Type Configuration					
	Program	L	D	F	Calls	L	D	F	Calls	# Config	mm:ss
GSL	bessel	0	18	0	0	0	18	0	0	130	37:11
GSL	gaussian	0	52	0	0	0	52	0	0	201	16:12
GSL	roots	0	19	0	0	0	0	19	0	3	1:03
GSL	polyroots	0	28	0	0	0	28	0	0	336	43:17
GSL	rootnewt	0	12	0	0	0	4	8	0	61	16:56
GSL	sum	0	31	0	0	0	9	22	0	325	28:14
GSL	fft	0	22	0	0	0	0	22	0	3	1:16
GSL	blas	0	17	0	0	0	0	17	0	3	1:06
NAS	EP	0	13	0	4	0	5	8	4	111	23:53
NAS	CG	0	32	0	3	0	2	30	3	44	0:57
	arclength	9	0	0	3	0	2	7	3	33	0:40
	simpsons	9	0	0	2	0	0	9	2	4	0:07

BLAME ANALYSIS [ICSE'16]

- PRECIMONIOUS: configurations lead to speedup, but requires running program numerous times during search
- BLAME ANALYSIS: successful at lowering precision, but does not guarantee speedup, single run of the program while performing shadow execution
- Largest impact when combining the analyses
  - Combined analysis time is 9x faster on average, and up to 38x in comparison with PRECIMONIOUS alone
  - Type configurations lead to speedup of up of up to 40%

CRAFT [ICS'13]

- **C**onfigurable **R**untime **A**nalysis for **F**loating-Point **T**uning
- Framework for x64 floating-point binary analysis tools
  - Cancellation detection
  - Dynamic range tracking
  - Mixed-precision auto-tuning
  - **Reduced-precision analysis**
  - Value histograms
    - (Andrew Shewmaker, previously LANL)
- Source: `github.com/crafthpc/craft`

CRAFT [IJHPCA'16]

```
APPLICATION: "sum2pi_x"    Prec=30
  MODULE: 0x400000 "sum2pi_x.c"    Prec=30          [6 instruction(s)]                                    [global:
    FUNC: 0x4005d0 "sum2pi_x"    Prec=30            [6 instruction(s)]                                      [global:
      BBLK: 0x400640    Prec=0
        INSN: 0x400643 "addsd xmm1, xmm1  [/g/g19/lam26/src/crafthpc...   Prec=0   [30000 execution(s)]
      BBLK: 0x40064b    Prec=28
        INSN: 0x400652 "divsd xmm7, xmm1  [/g/g19/lam26/src/crafthpc...   Prec=24   [2400 execution(s)]
        INSN: 0x400659 "addsd xmm2, xmm7  [/g/g19/lam26/src/crafthpc...   Prec=28   [2400 execution(s)]
      BBLK: 0x40065f    Prec=30
        INSN: 0x400662 "addsd xmm0, xmm2  [/g/g19/lam26/src/crafthpc...   Prec=30   [100 execution(s)]
```

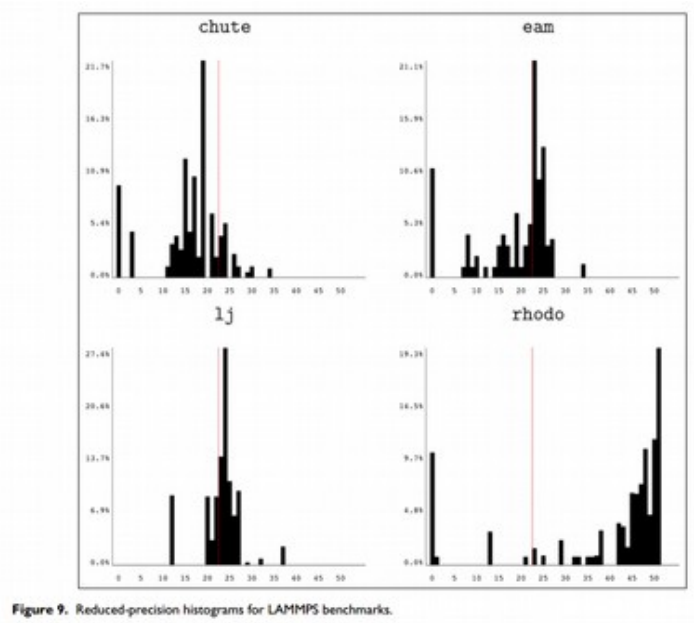


Figure 9. Reduced-precision histograms for LAMMPS benchmarks.

```
48        sum = 0.0;
49        for (i=0; i<OUTER; i++) {
50            acc = 0.0;
51            for (j=1; j<INNER; j++) {
52
53                /* calculate 2^j */
54                x = 1.0;
55                for (k=0; k<j; k++) {
56                    x *= 2.0;
57                }
58
59                /* accumulatively calculate pi */
60                y = (real)PI / x;
61                acc += y;
62
63                /*printf("  ACC%03d: %.16e\n", j, acc);*/
64            }
65            sum += acc;
66            /*printf("  SUM%03d: %.16e\n", i, sum);*/
67        }
```

SHVAL [ESPT'16]

- **SH**adow **V**alue **A**nalysis **L**ibrary (`github.com/lam2mo/shval`)
- Pintool for simulating alternative representations on compiled binaries
  - Native 32-bit IEEE float
  - Arbitrary precision (MPFR)
  - Unum 1.0 (library by G. Scott Lloyd, LLNL)
  - Posits (library by Isaac Yonemoto)

```
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	1.0000000000000005551...182	1.11e-16

TABLE I
ANALYSIS RESULTS ON SAMPLE PROGRAM

SHVAL [EMSOFT'17]

- Extended by Ramy Medhat (University of Waterloo) to aggregate and track error by instruction or memory location over time
  - Higher overhead, more information
- Apriltags case study
  - 1.7x speedup on average with only 4% error
  - 40% energy savings in embedded experiments

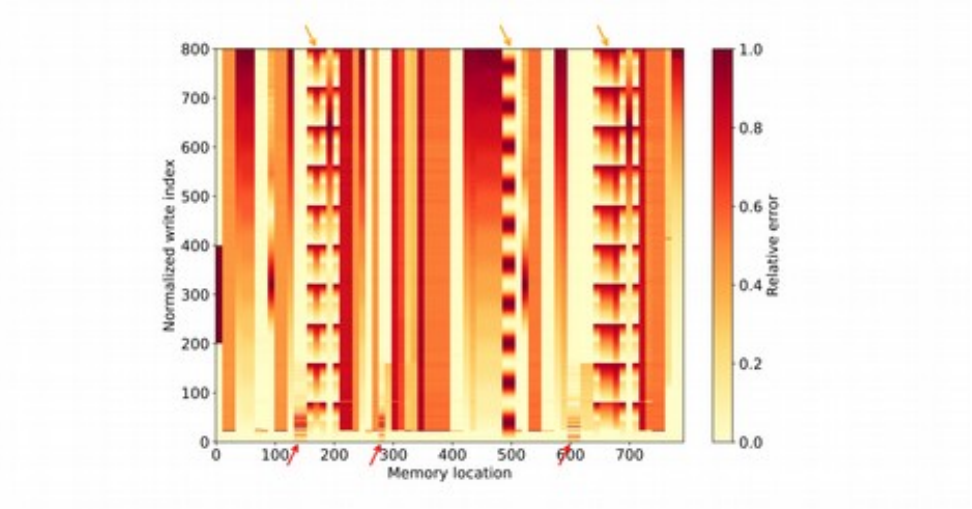


Fig. 1. Error trace per memory location. A darker pixel indicates higher error.

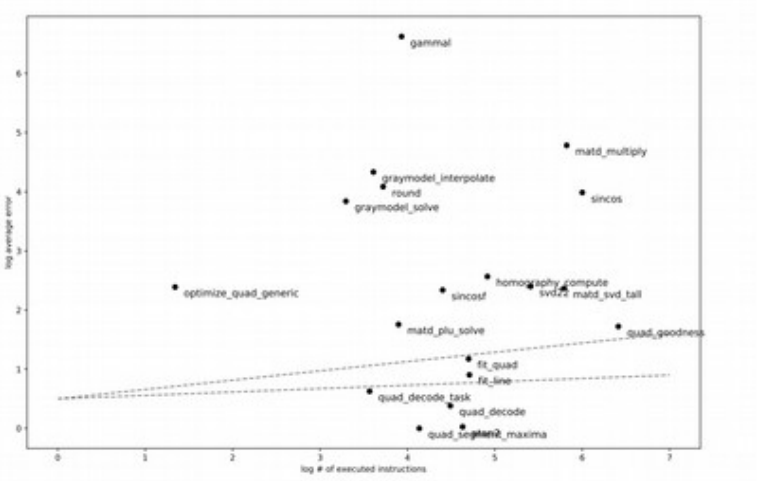
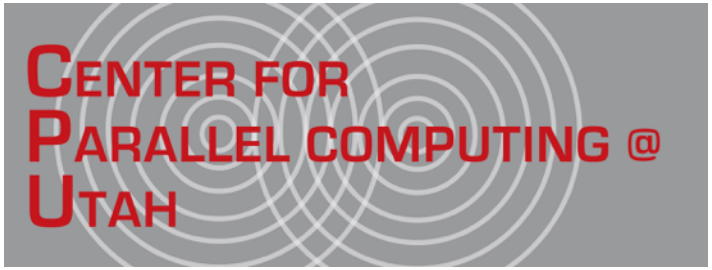


Fig. 8. Tradeoff between performance gain and error for every function in the Apriltags library.



UTAH FLOATING-POINT TOOLSET

Ganesh Gopalakrishnan
Zvonimir Rakamarić
and team
URL: soarlab.org
URL: www.cs.utah.edu/fv

Video at tinyurl.com/SC17-FP-BoF-Utah-Youtube
Slide deck at tinyurl.com/SC17-FP-BoF-Utah-FP-Tools

UTAH FLOATING-POINT TOOLSET

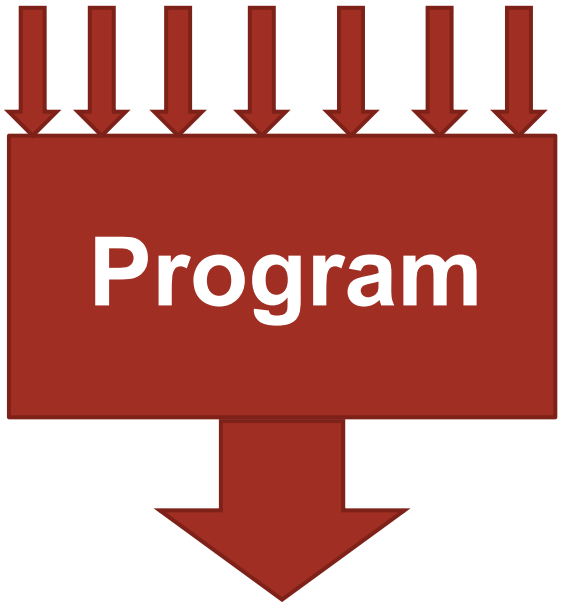
1. Verification of floating-point programs [**SMACK**]
2. Error analysis
   1. Dynamic [**S3FP**]
      - Best effort
      - Produces lower bound (under-approximation)
   2. Static [**FPTaylor**]
      - Rigorous
      - Produces upper bound (over-approximation)
        - Comes with rigorous global optimizer [**Gelpia**]
3. Rigorous mixed-precision tuning [**FPTuner**]
4. Compiler flag sensitivity [**FLiT**]

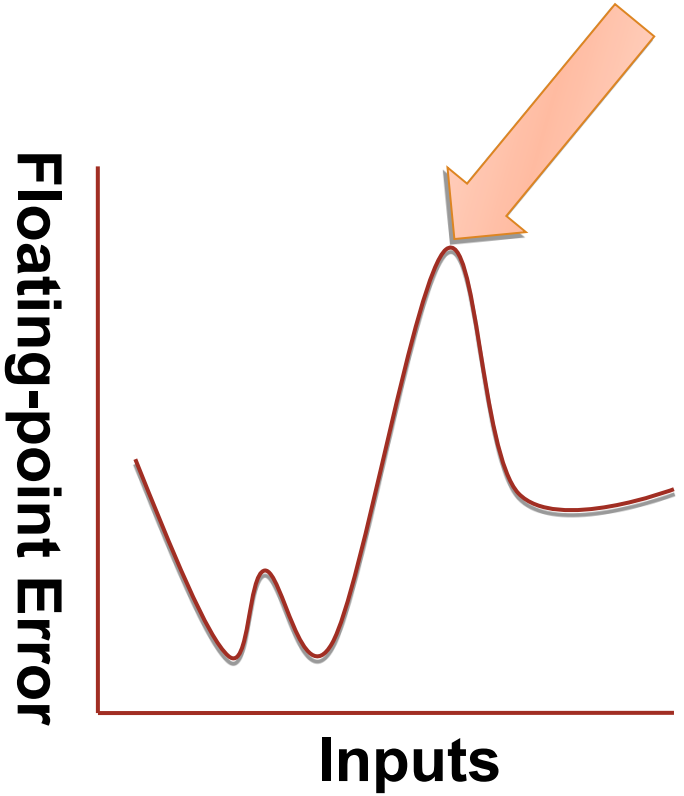
SMACK

- Automatic software verifier based on LLVM
- Extended with support for verification of properties that require precise reasoning about floating-points
- Leverage floating-point decision procedures implemented in Satisfiability Modulo Theories (SMT) solvers
  - Z3 SMT solver for now
- Part of official SMACK release
  - Enables verification of floating-point programs in C
  - Supports pointers, pointer arithmetic, dynamic memory allocation, structs, function pointers…

S3FP

- Finding program inputs that maximize floating-point error (black box)





S3FP

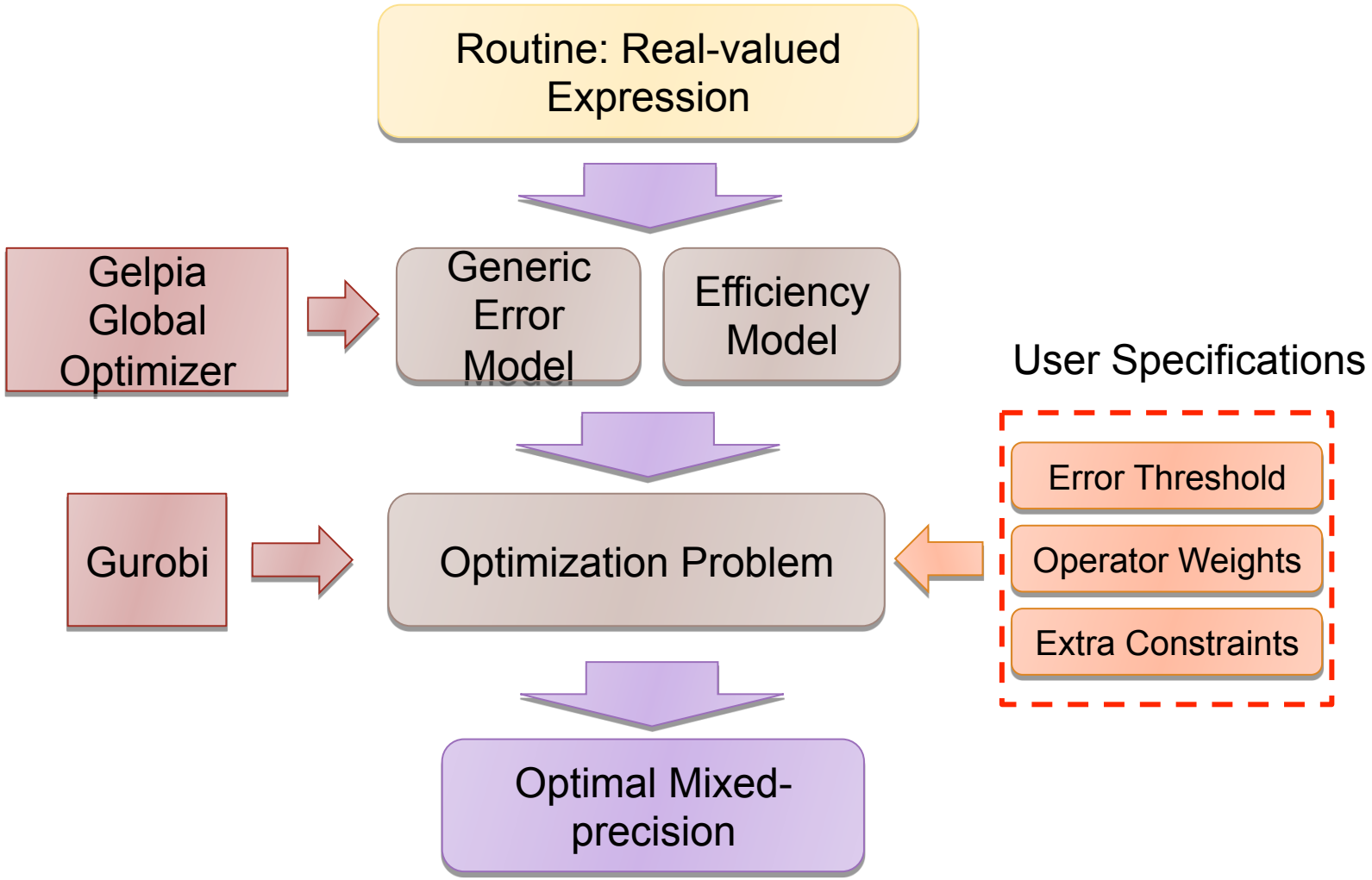
- Guided fuzzing overcomes some drawbacks of previous approaches
  - Improves scalability to real codes
  - Precisely handles diverse floating-point operations and conditionals
    - Shown to be able to handle divergent conditionals [LCPC 2015]

- Guided fuzzing can detect (much) higher floating-point errors than pure random testing

FPTaylor

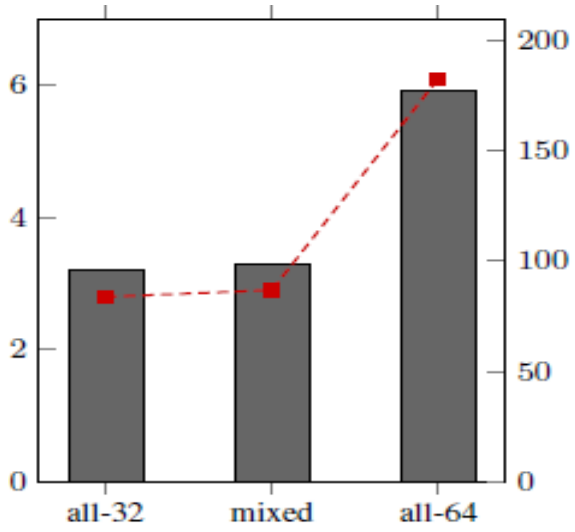
- Handles non-linear and transcendental functions
- Tight error upper bounds
- Rigorous
  - Over-approximation
  - Based on our own rigorous global optimizer
  - Emits a HOL-Lite proof certificate
    - Verification of the certificate guarantees estimate
- Tool called FPTaylor publicly available

FPTuner

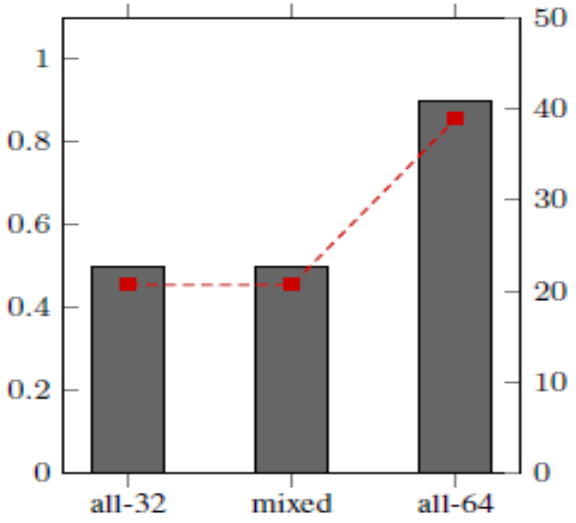

Routine: Real-valued Expression
Gelpia Global Optimizer
Generic Error Model
Efficiency Model
User Specifications
Error Threshold
Operator Weights
Extra Constraints
Gurobi
Optimization Problem
Optimal Mixed-precision

FPTuner

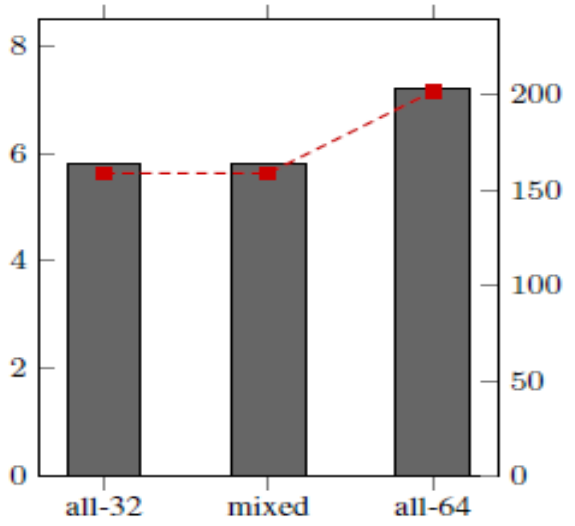
ENERGY CONSUMPTION BENEFITS



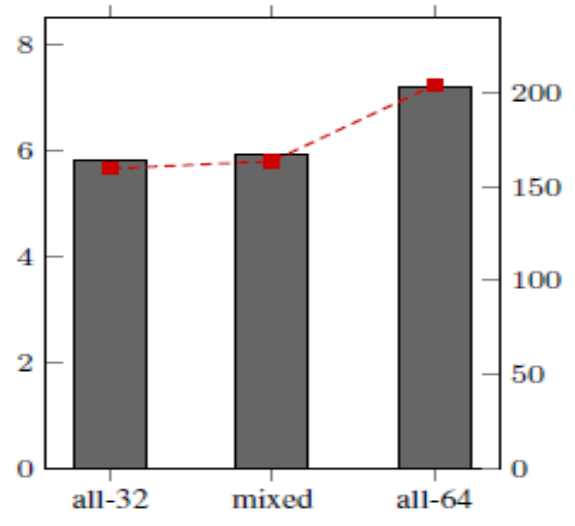
(a) sine



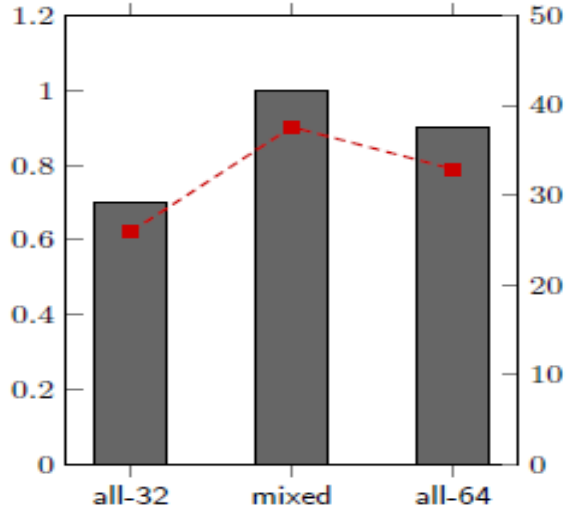
(b) sine



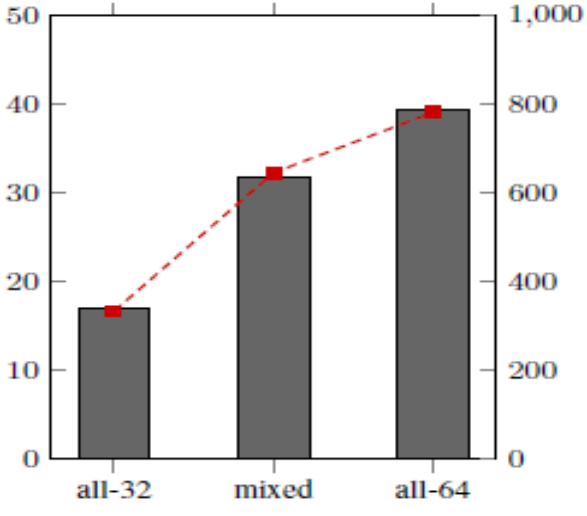
(c) maxBolt



(d) gaussian

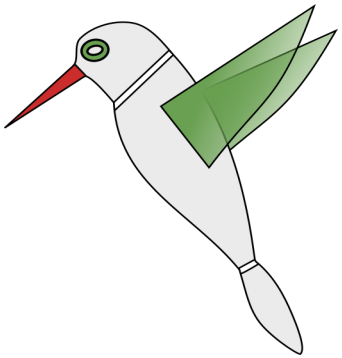


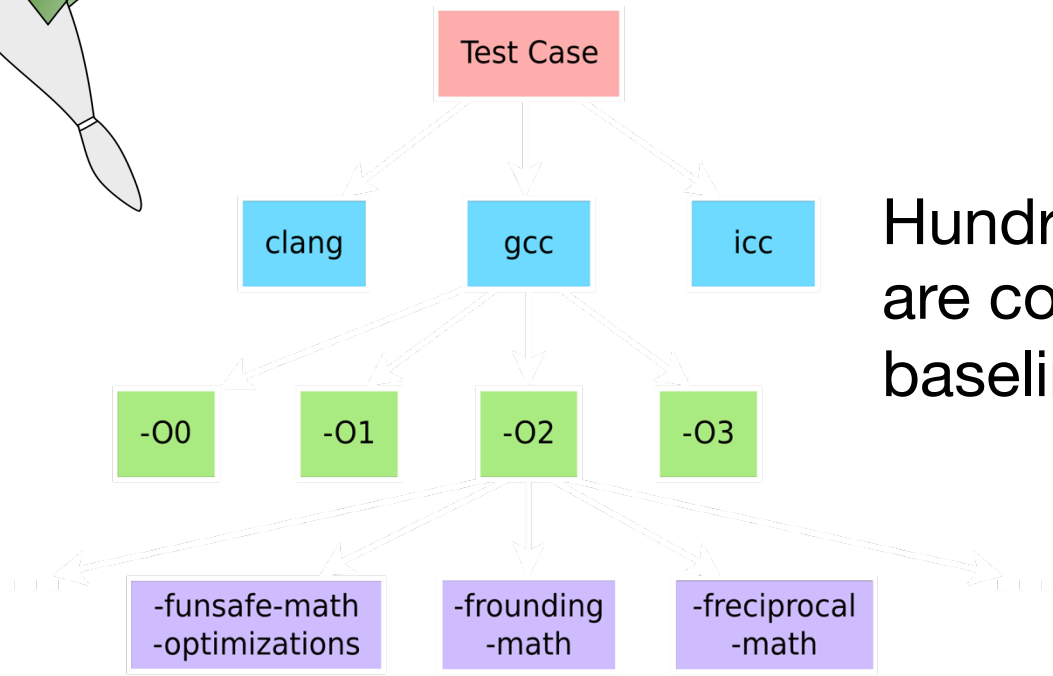
(e) jetEngine



(f) reduction

FLiT





FLiT is a reproducibility test framework in the PRUNERS toolset (pruners.gihub.io).

Hundreds of compilations are compared against a baseline compilation.

Daisy

a framework for accuracy analysis of numerical programs

optimisation
synthesis
...

Eva Darulova
eva@mpi-sws.org

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

Yet another tool: Daisy

a framework for the analysis and optimisation of numerical programs

Design Goals:

- modular —> easy to extend
- limited dependencies —> portable
- integration of previous techniques:
  - ➡ dataflow analysis (as in Rosa, Fluctuat)
  - ➡ optimisation-based analysis (as in Fluctuat, preliminary implementation)
  - ➡ interval subdivision for all techniques
- clarity of code above performance (written in Scala)

https://github.com/malyzajko/daisy

Yet another tool: Daisy

a framework for the analysis and optimisation of numerical programs

Supported Features:

- absolute roundoff error estimation
  - ➡ floating-point & fixed-point arithmetic
- mixed-precision
- transcendental functions
- rewriting optimisation
- dynamic error evaluation (with arbitrary precision)
- relative error estimation [1]
- formal certificates for Coq and HOL4 [2]
  - ➡ for absolute error analysis with intervals, more features to come
- soon to come: sound mixed-precision tuning [3]

Improving Accuracy with Herbie

z = sqrt(x+1) - sqrt(x)

z = 1/(sqrt(x+1) + sqrt(x))

Mechanisms

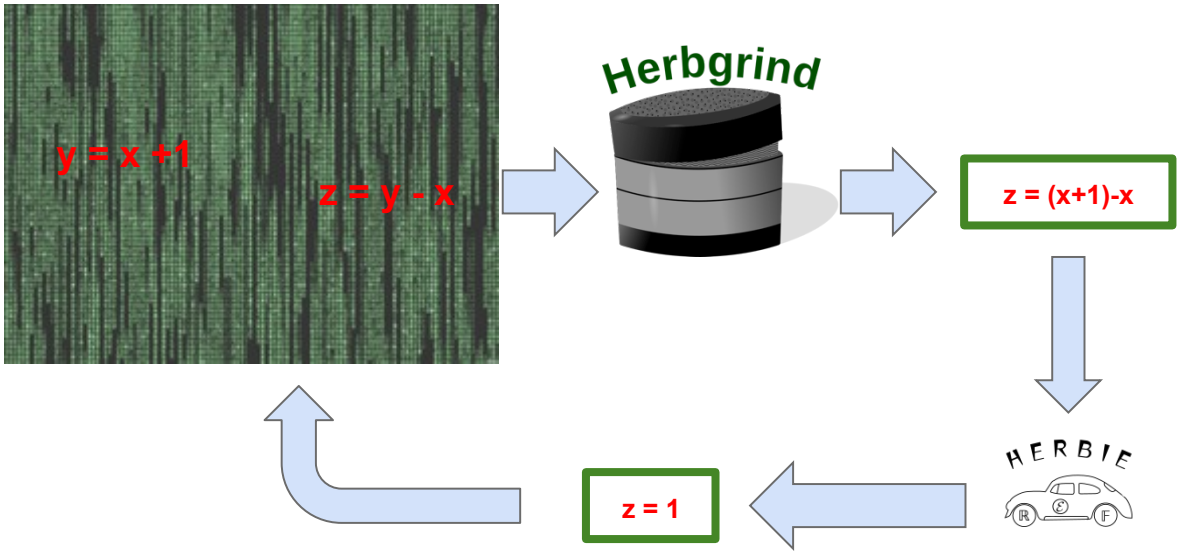
- Random sampling
  - Arbitrary-precision math
- Algebraic rewrites
  - Simplification
  - Series expansion
  - Infer branches

Evaluation

- Tried on 100s of exprs
  - Confirmed patches
- Multiple robust releases
  - Real-world users

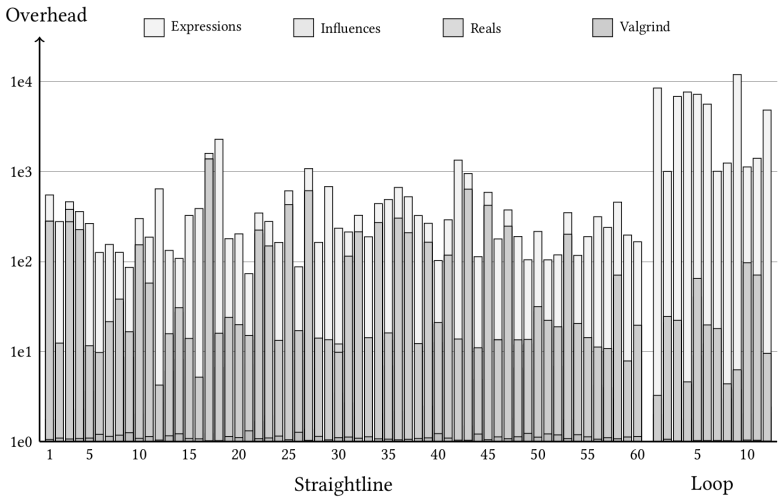
https://herbie.uwplse.org

Finding Root Causes with Herbgrind



Three Main Systems:

- Real Number Shadowing
- Symbolic Expression Inference
- Localization and Influence Tracking



- Found bugs in SPEC2006 Benchmark (Calculix)
- Runs on complex numerical software
  - Triangle
  - Polybench
  - GROMACS molecular dynamics simulation

https://herbgrind.ucsd.edu

FPBench: Community Standards for FP Tools

Compare FP tools: ~ 100 accuracy benchmarks

Benchmark sources		Features used		Domains	
Rosa	37	Arithmetic	111	Textbooks	28
Herbie	28	Temporaries	57	Mathematics	24
Salsa	25	Comparison	33	Controls	10
FPTaylor	21	Loops	28	Science	10
		Exponents	16	(unknown)	39
		Trigonometry	15		
		Conditionals	10		

Interop to compose tools : FPCore, FPImp (w/ C & Scala transl)

Accurate baseline $\mathbb{R}$: SMT-based (`titanic.uwplse.org`)

Growing community: Utah, JM, MPI, UW, etc.

fpbench.org

Floating-Point Tool Status

- Rigorous analyses that do not (yet) work at scale
  - FLUCTUAT, FPTuner, Rosa, Daisy, etc.
- Heuristic analyses that (partially) work at scale
  - CRAFT, Precimonious, Herbgrind, etc.
- Growing, diverse community of tool developers and users
  - Numerical analysis and scientific computing
  - High-performance computing
  - Programming languages and compilers
  - Systems tools and software engineering
  - Correctness and ESPT workshops at SC'17

Join us!

Thank you!

For more information:

`tinyurl.com/fpanalysis`
`fpbench.org/community.html`

Or contact me: `lam2mo@jmu.edu`