BoF: Improving Numerical Computation with Practical Tools and Novel Computer Arithmetic

Dr. John L. Gustafson
National University of Singapore and A*STAR Computational Resource Center

Dr. Michael O. Lam
James Madison University and Lawrence Livermore National Laboratory

Live Q&A: `tinyurl.com/sc17qna`

SC17 Denver, CO | hpc connects

NGA Next Generation Arithmetic — Unum & Posit

James Madison University

Floating-Point Analysis Tools

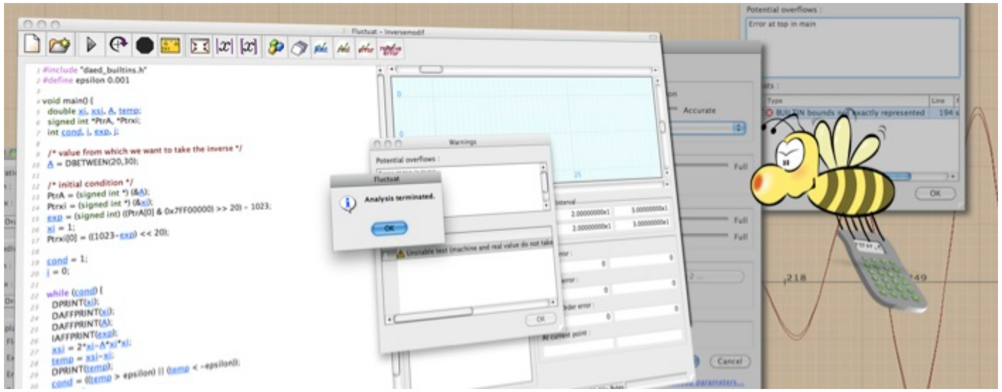
Session leader: Dr. Michael O. Lam, James Madison University

`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

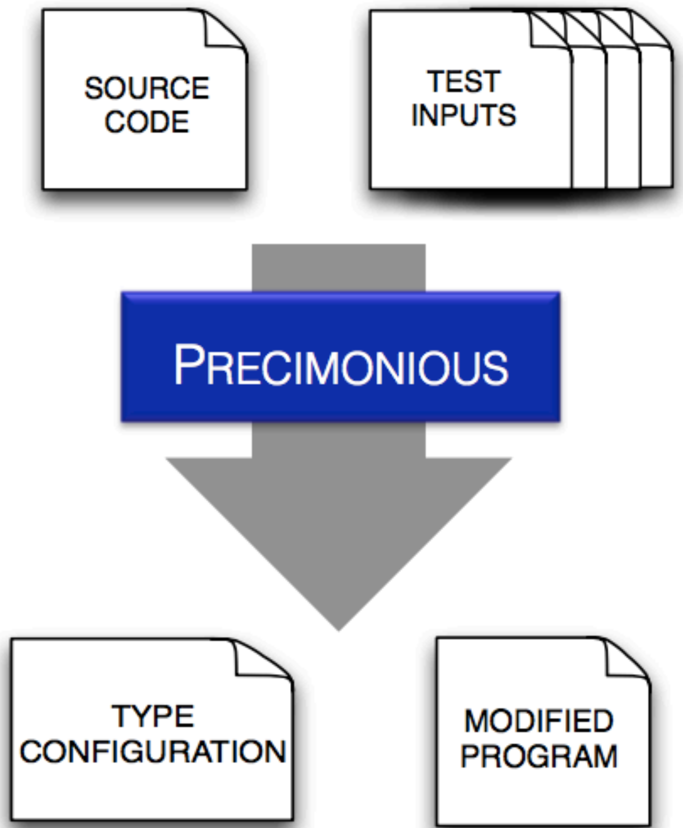
PRECIMONIOUS [SC'13]

"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

	Program	L	D	F	Calls
GSL	bessel	0	18	0	0
GSL	gaussian	0	52	0	0
GSL	roots	0	19	0	0
GSL	polyroots	0	28	0	0
GSL	rootnewt	0	12	0	0
GSL	sum	0	31	0	0
GSL	fft	0	22	0	0
GSL	blas	0	17	0	0
NAS	EP	0	13	0	4
NAS	CG	0	32	0	3
	arclength	9	0	0	3
	simpsons	9	0	0	2

Proposed Type Configuration

L	D	F	Calls	# Config	mm:ss
0	18	0	0	130	37:11
0	52	0	0	201	16:12
0	0	19	0	3	1:03
0	28	0	0	336	43:17
0	4	8	0	61	16:56
0	9	22	0	325	28:14
0	0	22	0	3	1:16
0	0	17	0	3	1:06
0	5	8	4	111	23:53
0	2	30	3	44	0:57
0	2	7	3	33	0:40
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)]
```

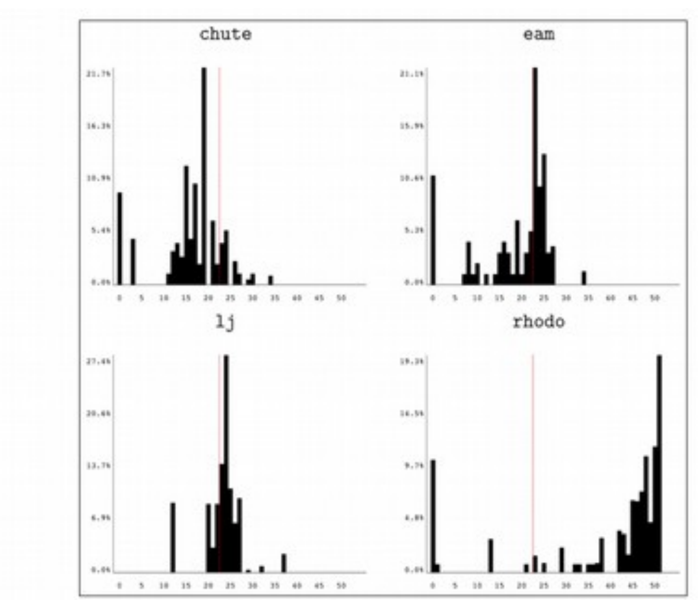


Figure 9. Reduced-precision histograms for LAMMPS benchmarks.

```
    sum = 0.0;
    for (i=0; i<OUTER; i++) {
        acc = 0.0;
        for (j=1; j<INNER; j++) {

            /* calculate 2^j */
            x = 1.0;
            for (k=0; k<j; k++) {
                x *= 2.0;
            }

            /* accumulatively calculate pi */
            y = (real)PI / x;
            acc += y;

            /*printf("  ACC%03d: %.16e\n", j, acc);*/
        }
        sum += acc;
        /*printf("  SUM%03d: %.16e\n", i, sum);*/
    }
```

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	1.00000000000000005551...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

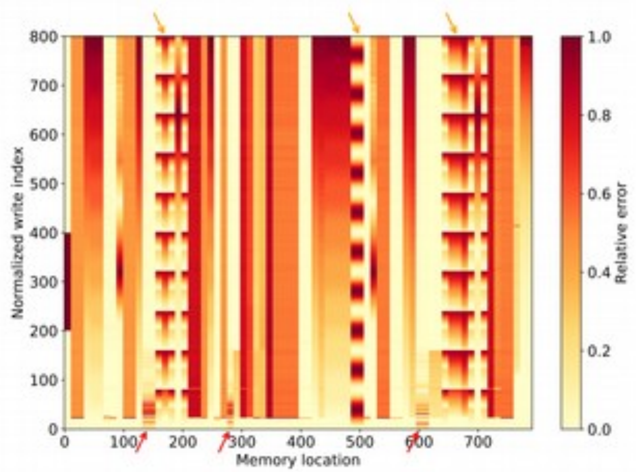


Fig. 1. Error trace per memory location. A darker pixel indicates higher error.

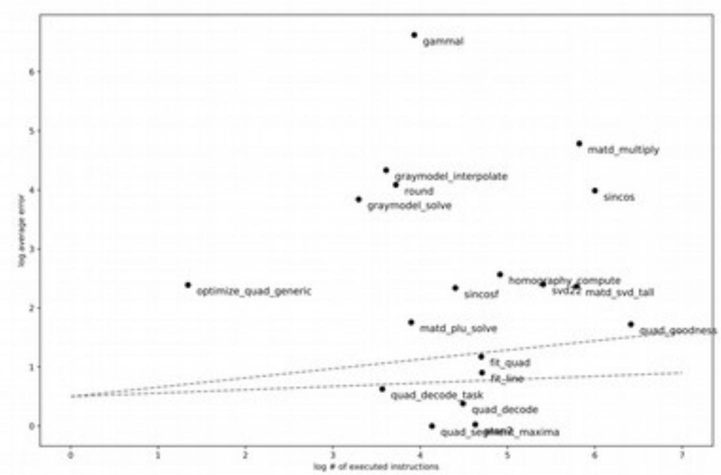


Fig. 8. Tradeoff between performance gain and error for every function in the Apriltags library.

CENTER FOR PARALLEL COMPUTING @ UTAH

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

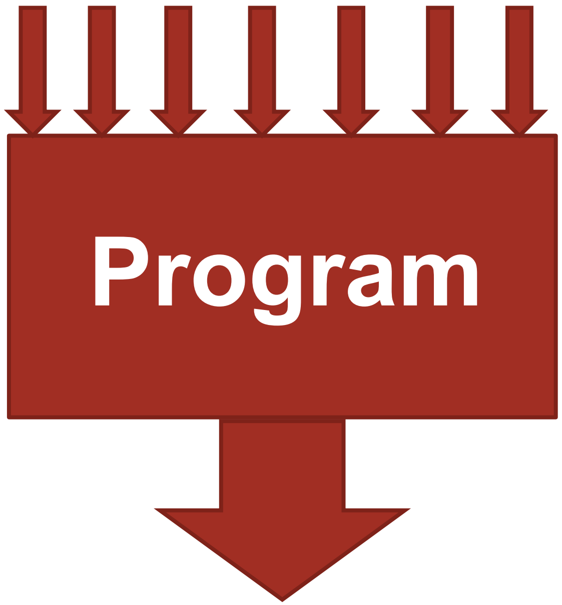
1. Verification of floating-point programs [**SMACK**]
2. Error analysis
   1. Dynamic [**S3FP**]
      - Best effort
      - Produces lower bound (under-approximation)
   2. Static [**FPTaylor**]
      - Rigorous
      - Produces upper bound (over-approximation)
        - Comes with rigorous global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

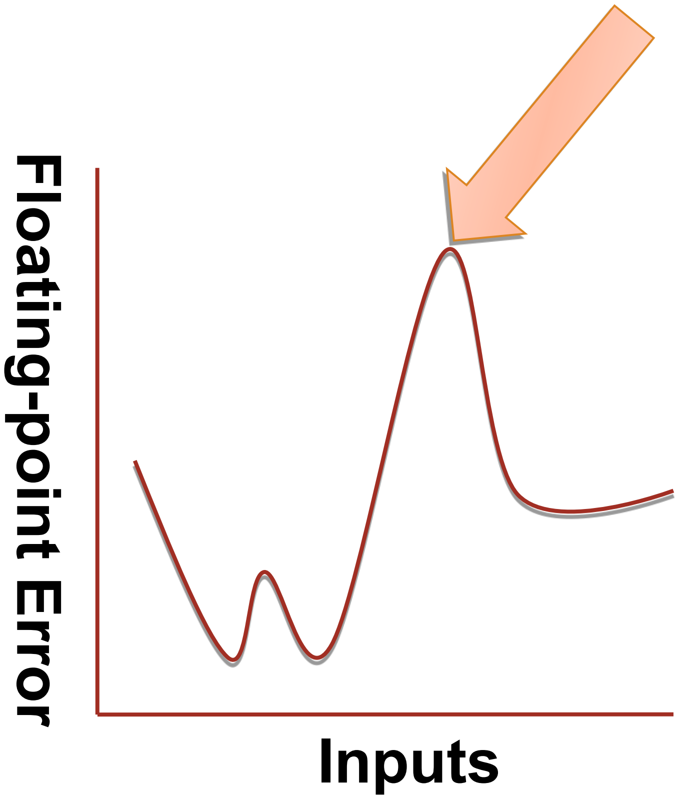
SMACK

- 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

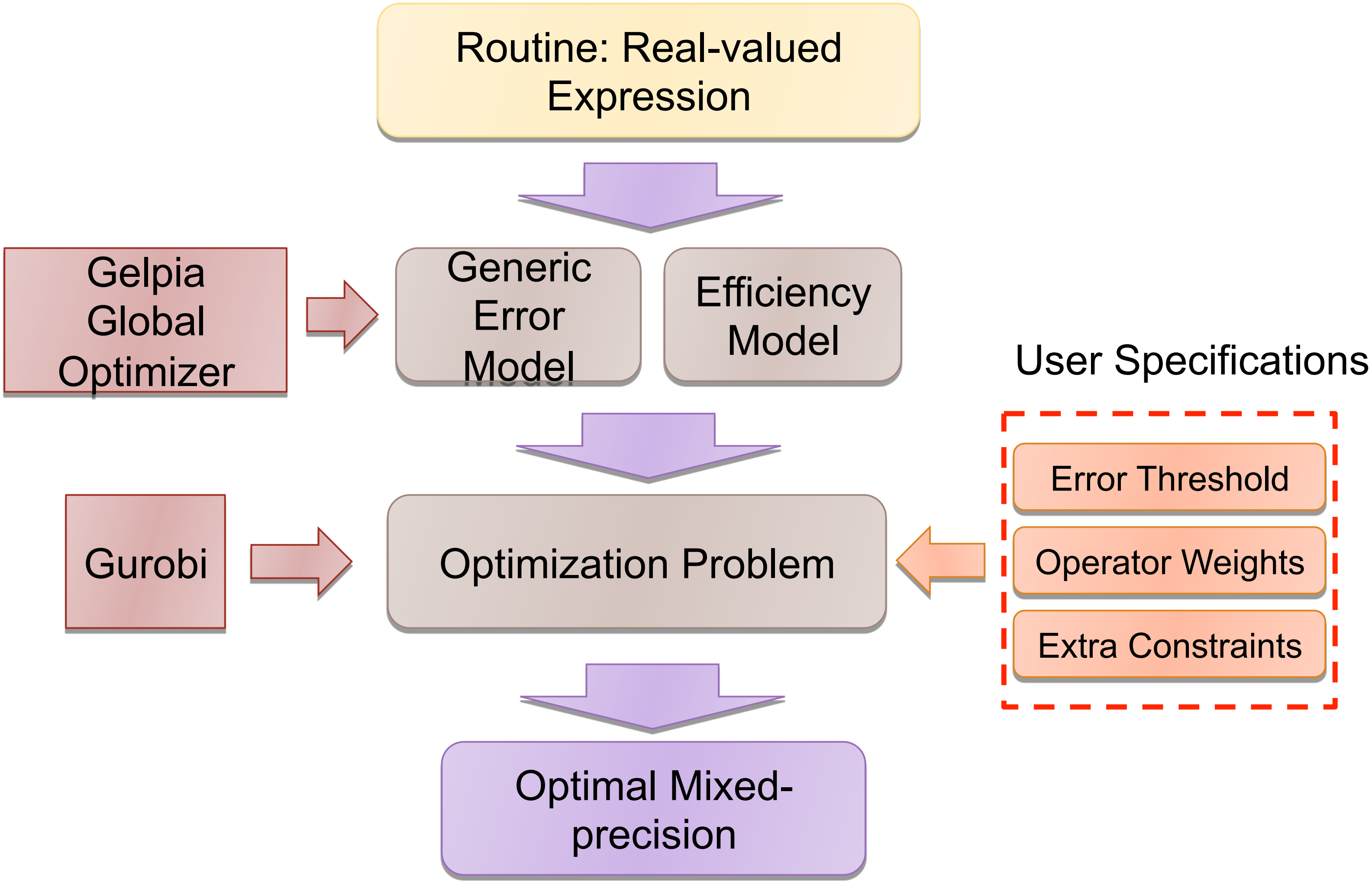
- Guided fuzzing overcomes some drawbacks of previous approaches
  - Improves scalability to real codes
  - Precisely handles diverse floating-point operations and conditionals
    - Shown to be able to handle divergent conditionals [LCPC 2015]

- 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

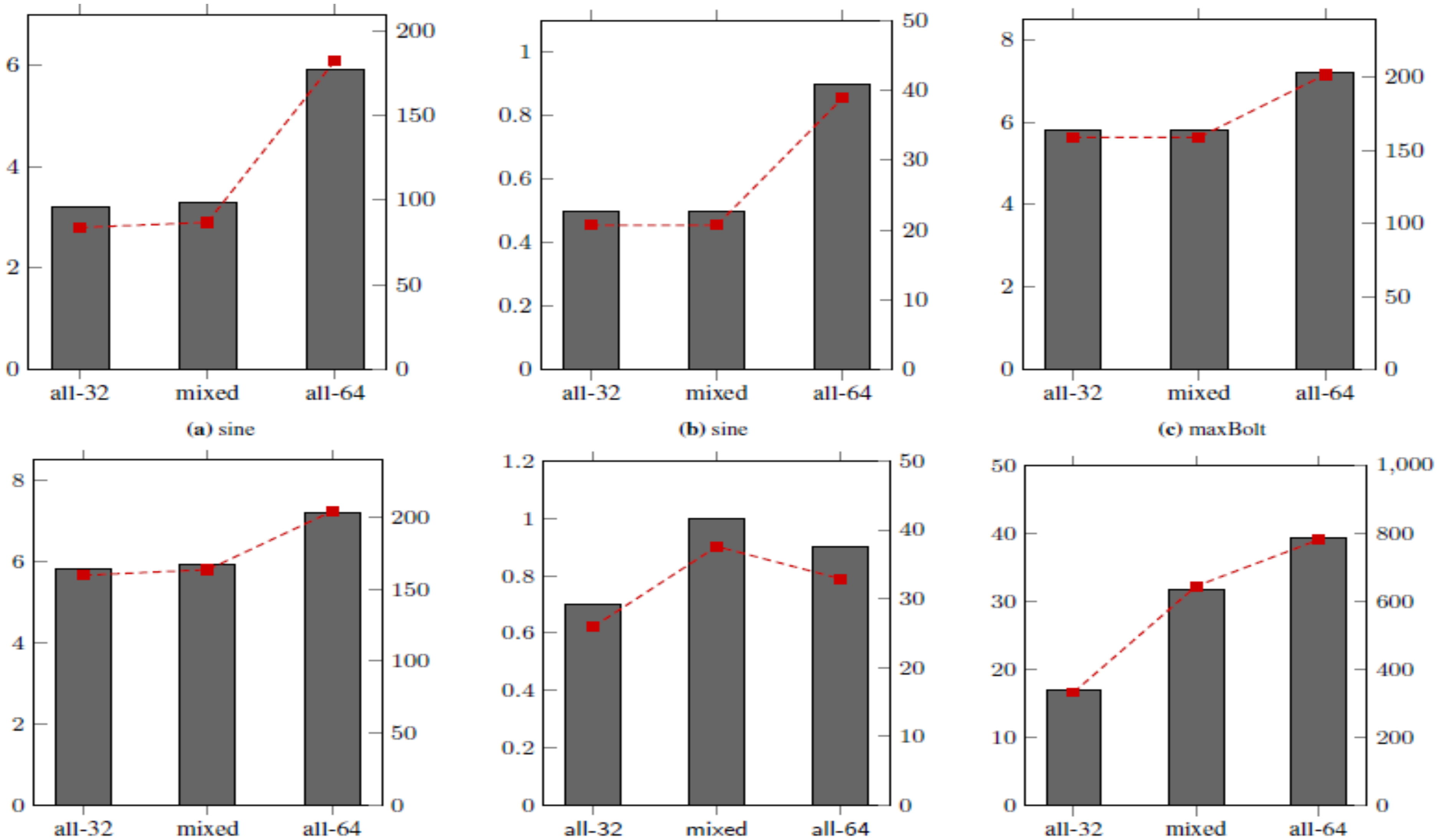
FPTuner



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

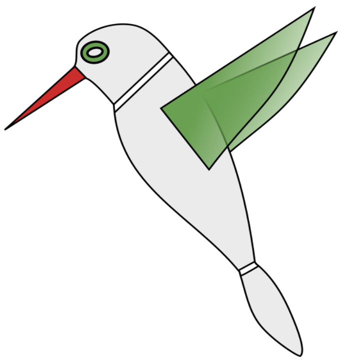
ENERGY CONSUMPTION BENEFITS



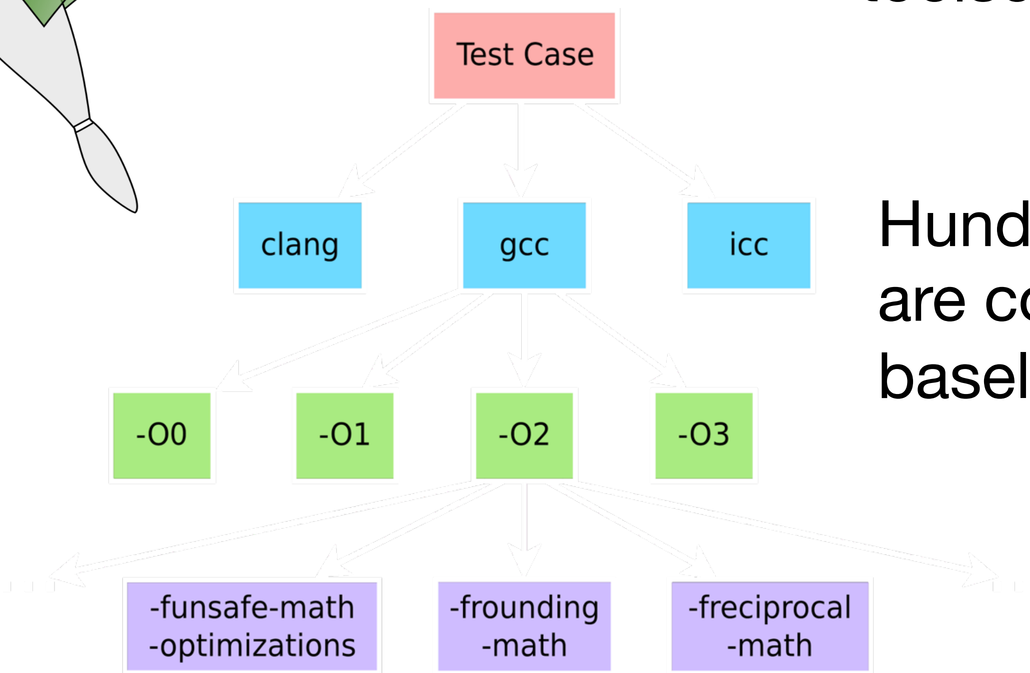
(a) sine (b) sine (c) maxBolt

(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

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

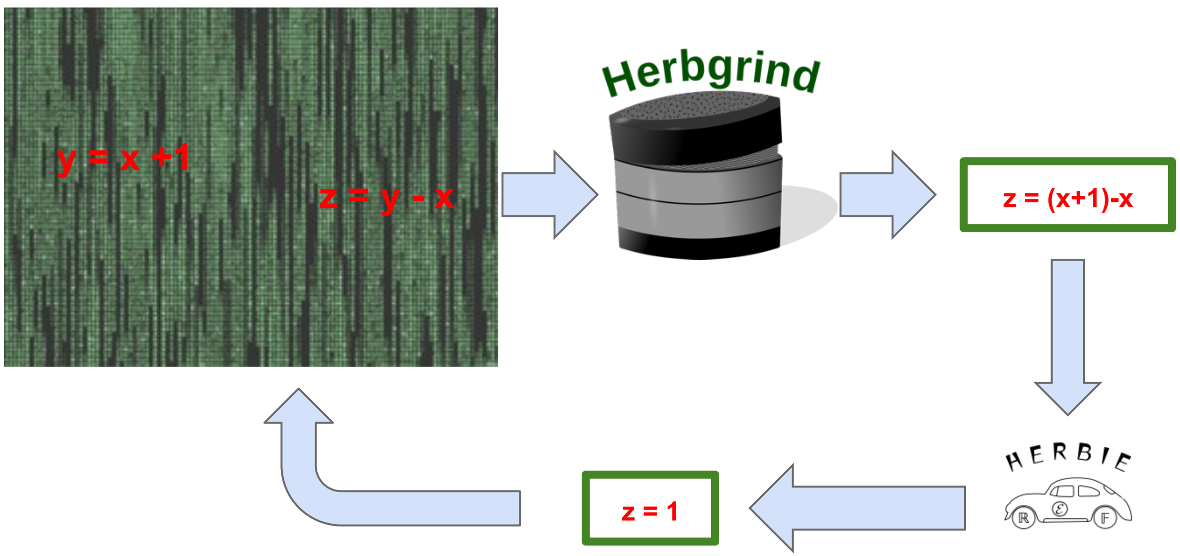
Confirmed patches

Multiple robust releases

Real-world users

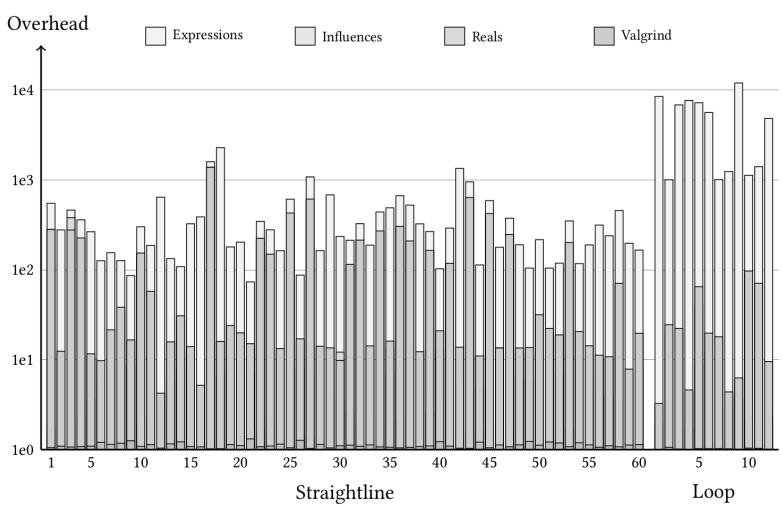
https://herbie.uwplse.org

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	
Rosa	37
Herbie	28
Salsa	25
FPTaylor	21

Features used	
Arithmetic	111
Temporaries	57
Comparison	33
Loops	28
Exponents	16
Trigonometry	15
Conditionals	10

Domains	
Textbooks	28
Mathematics	24
Controls	10
Science	10
(unknown)	39

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**

Next-Generation Arithmetic

Session leader: Dr. John L. Gustafson, National University of Singapore

posithub.org

Unum & Posit

Birds-of-a-Feather Session Part II: The Need for New Arithmetic

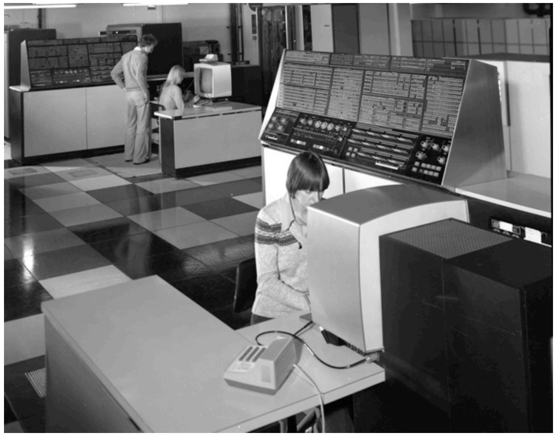
- **Why now**?
  - The AI revolution is making everyone discover that floats have better alternatives
  - Reaching exascale requires overcoming power and bandwidth limitations; floats are efficient at *neither*
  - Transistors are millions of times cheaper now than when the IEEE 754 Standard was set
  - A new format has been invented that *uniformly* improves on the IEEE 754 Standard, so it’s no longer a *tradeoff* argument.

Birds-of-a-Feather Session Part II: The Need for New Arithmetic

- A **new** standard lets us fix ossified mistakes. IEEE 754 cannot *evolve* without *breaking.*
  - Gradual overflow: **rejected**
  - Exact dot product: **rejected**
  - Repeatable answer requirement: **rejected**
  - Correctly rounded elementary functions: **rejected**
  - Hidden scratch work that makes some answers inexplicably different: **encouraged**

Problems with Existing Arithmetic

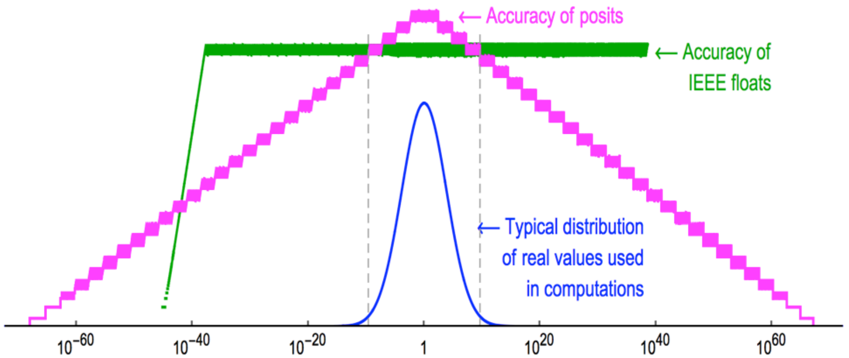
IEEE Standard Floats are a **bandwidth-inefficient, 1980s-era** design.

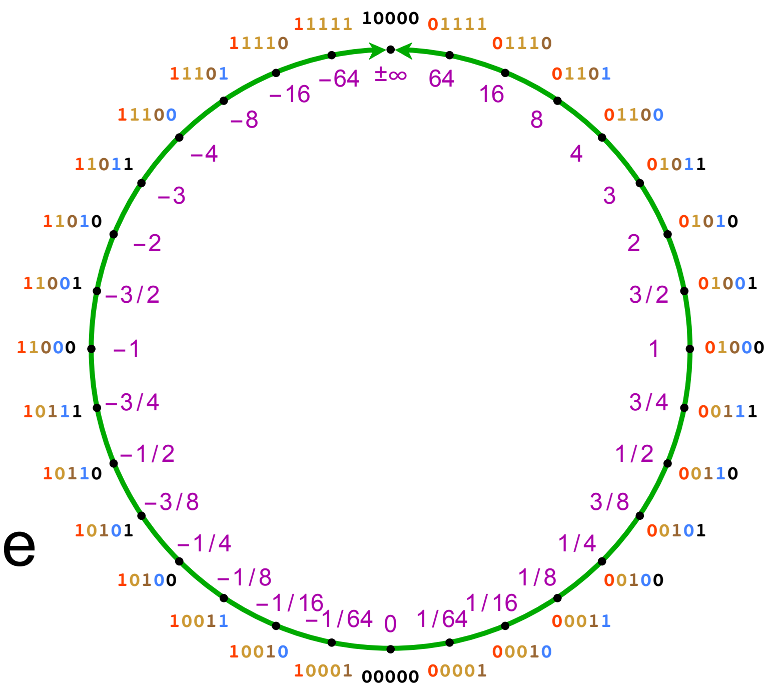


- No guarantee of **repeatable** or **portable** behavior
- Insufficient 32-bit accuracy forces wasteful use of 64-bit data
- Fails to obey laws of *algebra* (associative, distributive laws)
- Fails to obey laws of *logic* ($X = X$, $a = b$ means $f(a) = f(b)$)
- Poor handling of overflow, underflow, Not-a-Number results
- Dynamic ranges are ill-suited to actual application needs
- Rounding errors are invisible, hazardous, costly to debug
- Computations change when parallelized

Posit arithmetic (invented Dec 2016)

- Better accuracy with fewer bits
- Absolute **repeatability** and **portability** across systems
- “Valids” allow automatic control of rounding errors
- Clean, mathematical design
- 32-bit can often replace 64-bit
- Reduces *energy*, *power*, *bandwidth*, *storage*, and *programming costs*
- Like parallel computing, the change will be a lot of work, but **worth it.**





Posit Research

High-performance error-free Linear Algebra

Theodore Omtzigt, Cerlane Leong, Anantha Kinnal,
Vijay Holimath, John Gustafson

Problem

Float addition and multiplication are not associative

Non-reproducible results when going parallel

- Multi-thread, multi-core, many-core

Dangerous for embedded intelligent systems

- Error analysis is very difficult
- Failure analysis is next to impossible

Solution

Posit Number System

- Tapered floating point format
- Use bits where you need them
- Quires and fused dot-products

Error-free Linear Algebra

Solve a system of equations $Ax = b$

Gauss-Jordan yields a Reduced Row Echelon Form

With posits, the RREF can be the Identity Matrix

Other matrix transformations work the same

LU, LDM^T, LDL^T, QR, SVD, etc.

Fused operators

- Depend on a super accumulator, called quire
- For cache-friendly block schedules
  - Multiple quires need to be active
  - Organize as a new register file and ISA src/tgt
  - Increases context switch state significantly

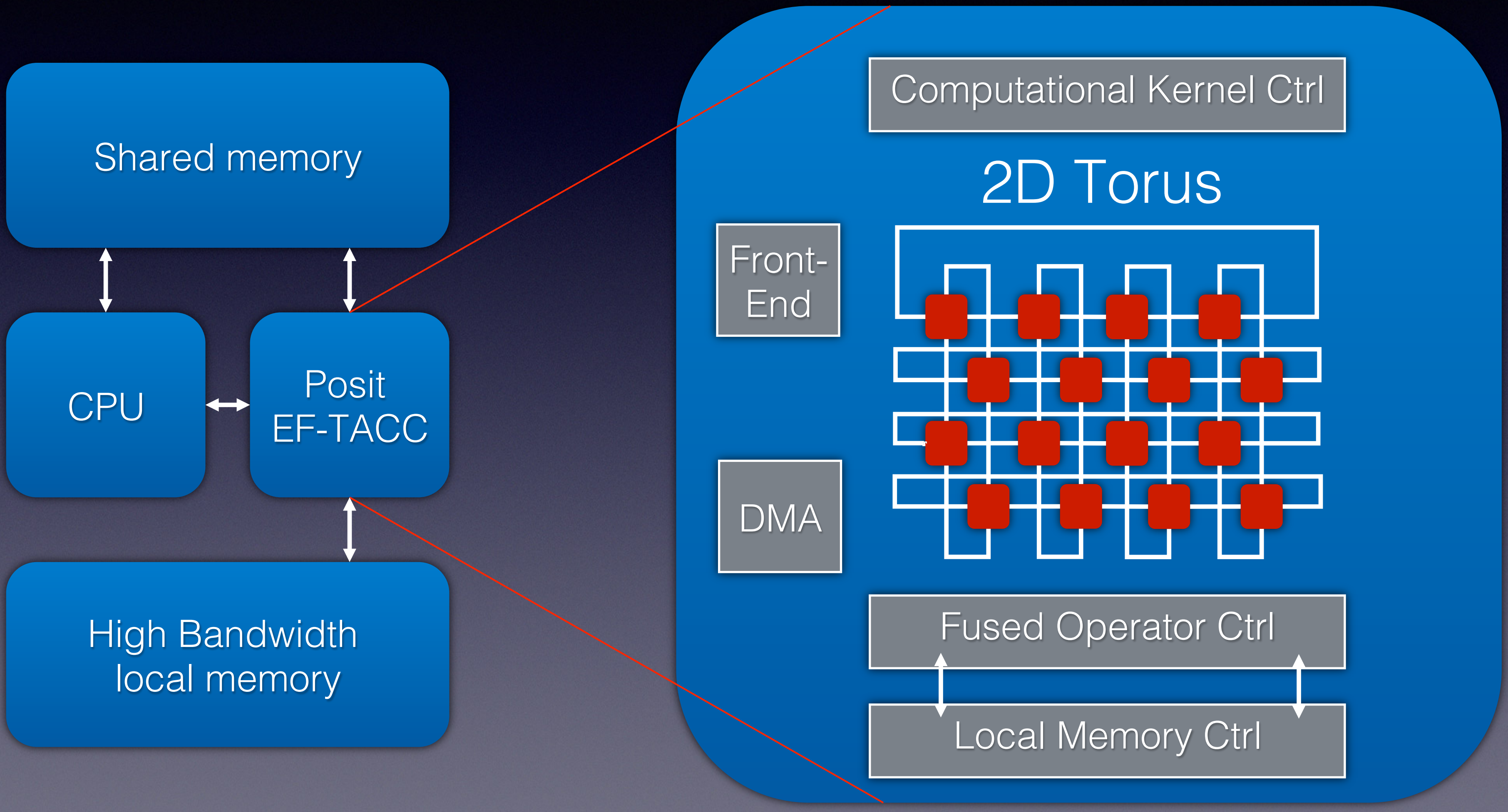
Quires impose a significant impact on General Purpose CPU uArchitecture

HW Accelerators

- Organize computation around fused operator pipelines
- Execution schedule takes quire into account
- Quire name space integrated into pipeline
- Context switch overhead is removed

HW Accelerators can deliver error-free linear algebra with more flexibility

Posit Error-Free Tensor Accelerator



Posit SDK: ready to go

Application
(C/C++/Python)

Error-Free Arithmetic Libraries
(Linear Algebra, DSP, AI)

PAL (Posit Arithmetic Library)	HAL (Hardware Abstraction Library)
General Purpose Hardware (x86, ARM, GPUs)	Posit Hardware (Posit Arith Unit, Accelerators)
Software emulation path	Hardware acceleration path

Universal Coding of the Reals: Alternatives to IEEE Floating Point - An Example Application

Improving Numerical Computation with Practical Tools and Novel Computer Arithmetic

SC17 Birds of a Feather Flash Talk

14 November 2017

Jeff Hittinger, Peter Lindstrom, and Scott Lloyd
Center for Applied Scientific Computing

LLNL-PRES-741490
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 17-SI-004

Lawrence Livermore National Laboratory

We consider an application solving the nonlinear hyperbolic Euler equations in 2D

- Shock wave passing through initially quiescent L-shaped chamber
- Ideal gas Euler equations

$$\partial_t u + \nabla \cdot F(u) = 0$$

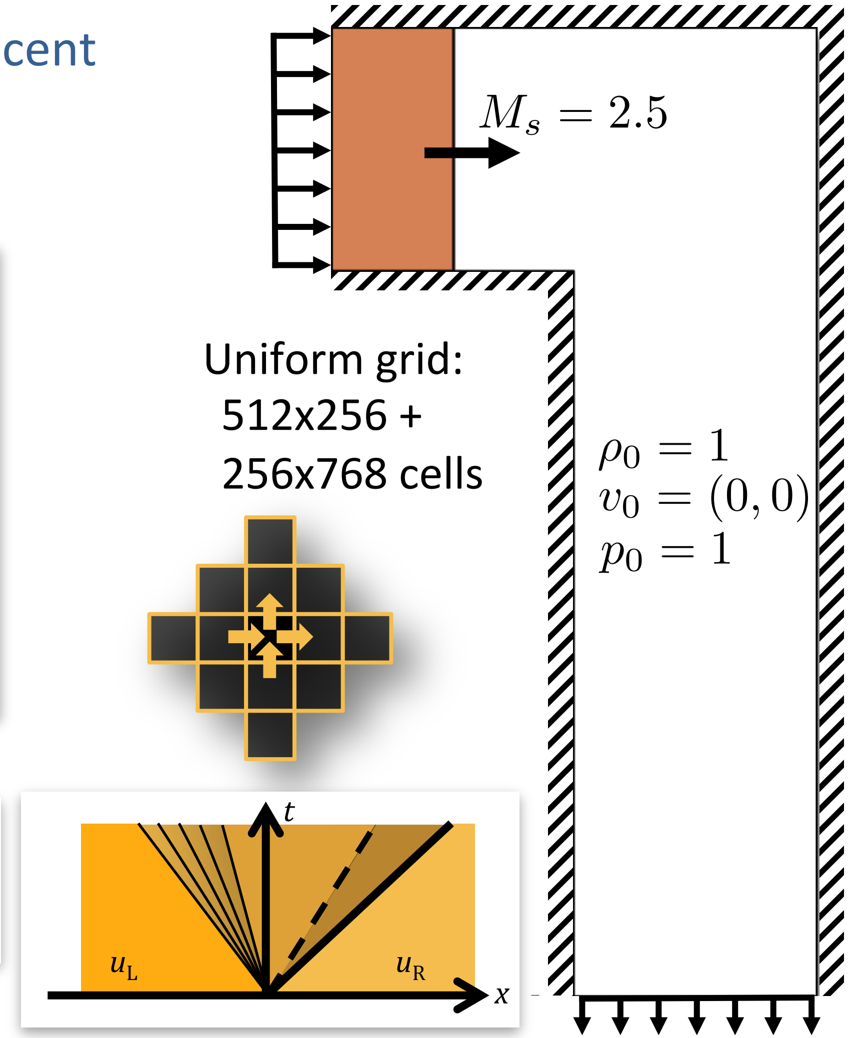
$$u = \begin{pmatrix} \rho \\ \rho v \\ \rho E \end{pmatrix} \qquad F(u) = \begin{pmatrix} \rho v \\ \rho v \otimes v + p \\ \rho v H \end{pmatrix}$$

$$\rho E = \frac{p}{\gamma - 1} + \frac{1}{2}|v|^2 \qquad \rho H = \rho E + p$$

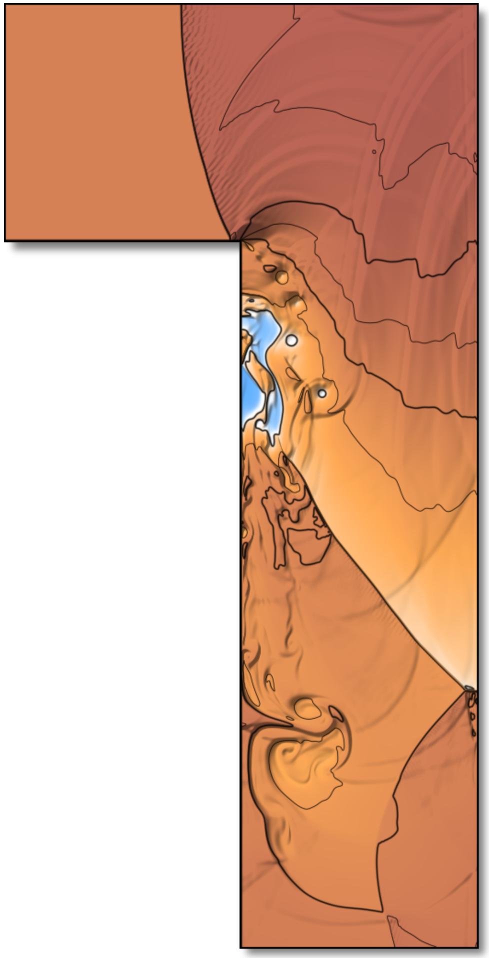
- Explicit finite volume discretization

$$u_{\mathbf{i}}^{n} = u_{\mathbf{i}}^{n} - \frac{\Delta t}{\Delta x} \sum_{d=1}^{2} \left[F^{d}_{\mathbf{i}+\frac{1}{2}\mathbf{e}^d} - F^{d}_{\mathbf{i}-\frac{1}{2}\mathbf{e}^d} \right]$$

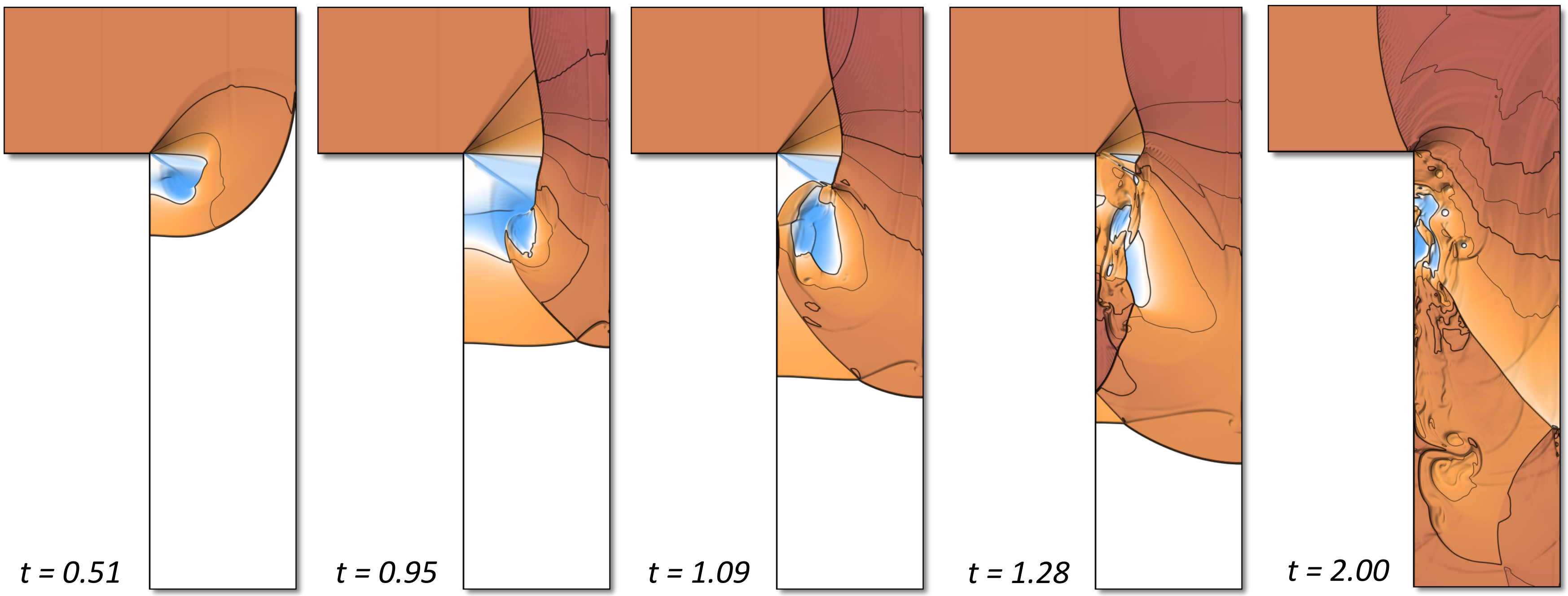
- High-resolution Godunov solver



The solution generates complicated wave interactions

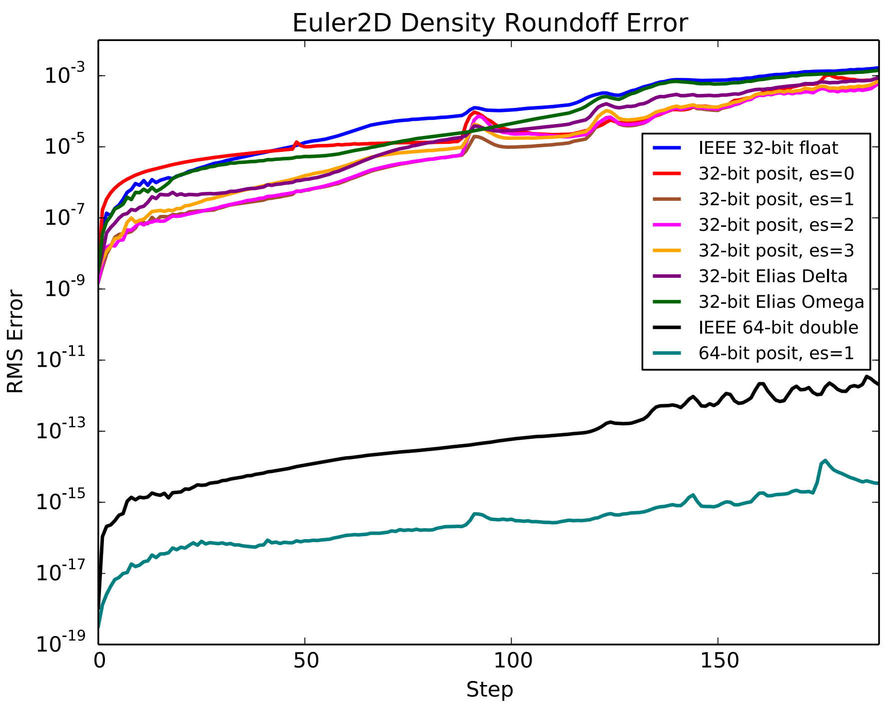


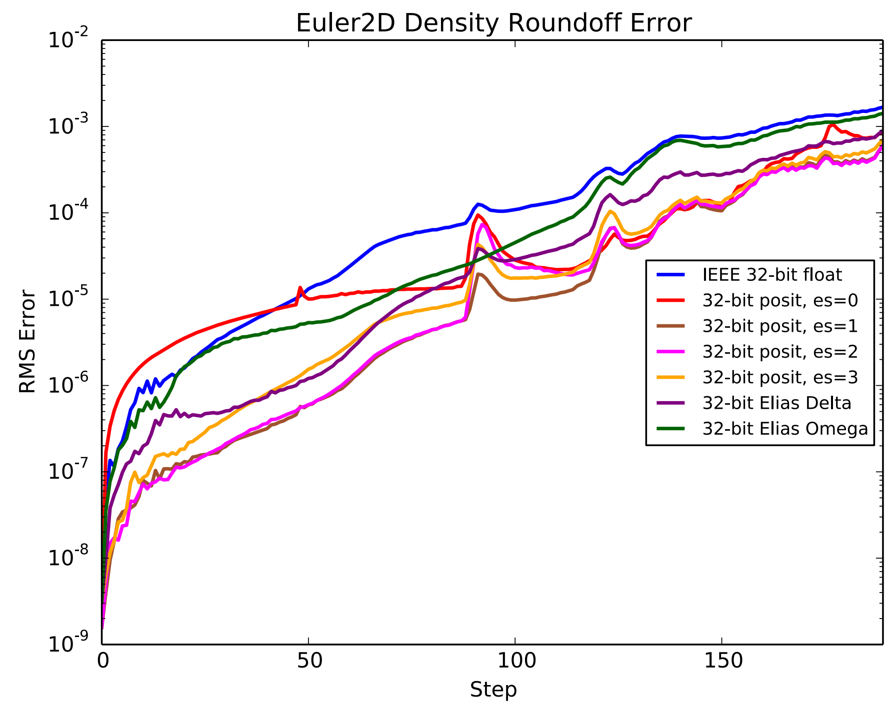
It is useful to understand the solution evolution in order to understand the precision results



Shock reflects off of far wall	Reflected shock hits vortex	Shock reflects off of near wall	Second reflection hits vortex	Multiple wave-wave and wave-vortex interactions
Steps 48-89	*Steps 90-102*	*Steps 103-120*	*Steps 121-135*	*Steps 136-189*

We have tested numerous types, including posits, using the Euler2D code





- Features in the results correlate to features in the solution data
- The general trend is that you can do better than IEEE float at 32-bits
- It appears 32-bit posit, es=1 performs the best of the 32-bit types tested

Universal Representations of the Reals: Alternatives to IEEE Floating Point

Peter Lindstrom

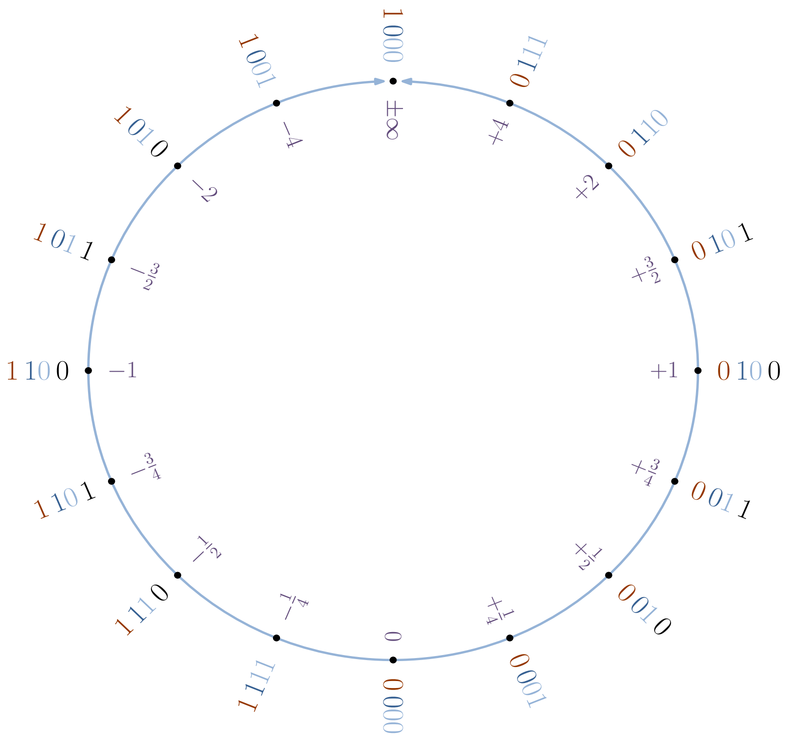
Nov 10, 2017

LLNL-PRES-731413
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

We are developing a modular framework for number representations that encompasses IEEE, posits, and others

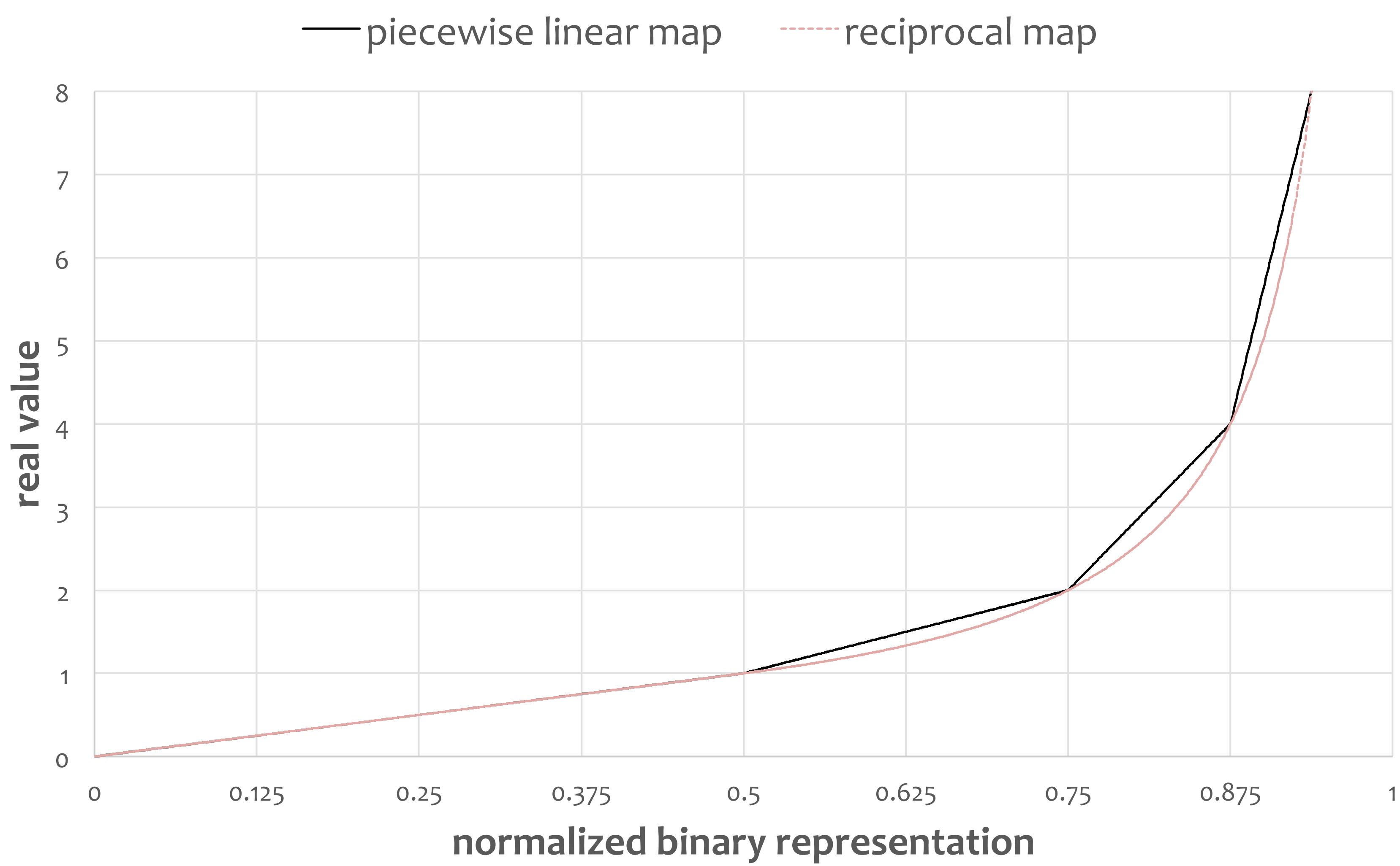
- Common representation: $y = (-1)^s\, 2^e\, \varphi(f)$
  - e is the integer exponent
  - $0 \le f < 1$ is the fraction
  - $\varphi : [0, 1) \longrightarrow [1, 2)$ is a monotonic map
- IEEE
  - Fixed-length encoding of e
  - Normalized numbers: $\varphi(f) = 1 + f$
  - Subnormal numbers: $\varphi(f) = 0 + f$
- Posits
  - Golomb-Rice variable-length encoding of e
  - $\varphi(f) = 1 + f$



We are developing a modular framework for number representations that encompasses IEEE, posits, and others

- Other exponent encoding schemes
  - From universal integer codes: Elias gamma, delta, omega, …
- Other fraction maps have nice properties
  - Linear reciprocal: $\varphi(f) = 2 / (2 - f)$ when $|y| < 1$ ensures reciprocal closure
  - Exponential: 2^f gives smooth mapping on entire domain
  - Rational: Self-reciprocal, cheaper than exponential
- Several rounding, under- and overflow strategies
- Can mix and match via modular design

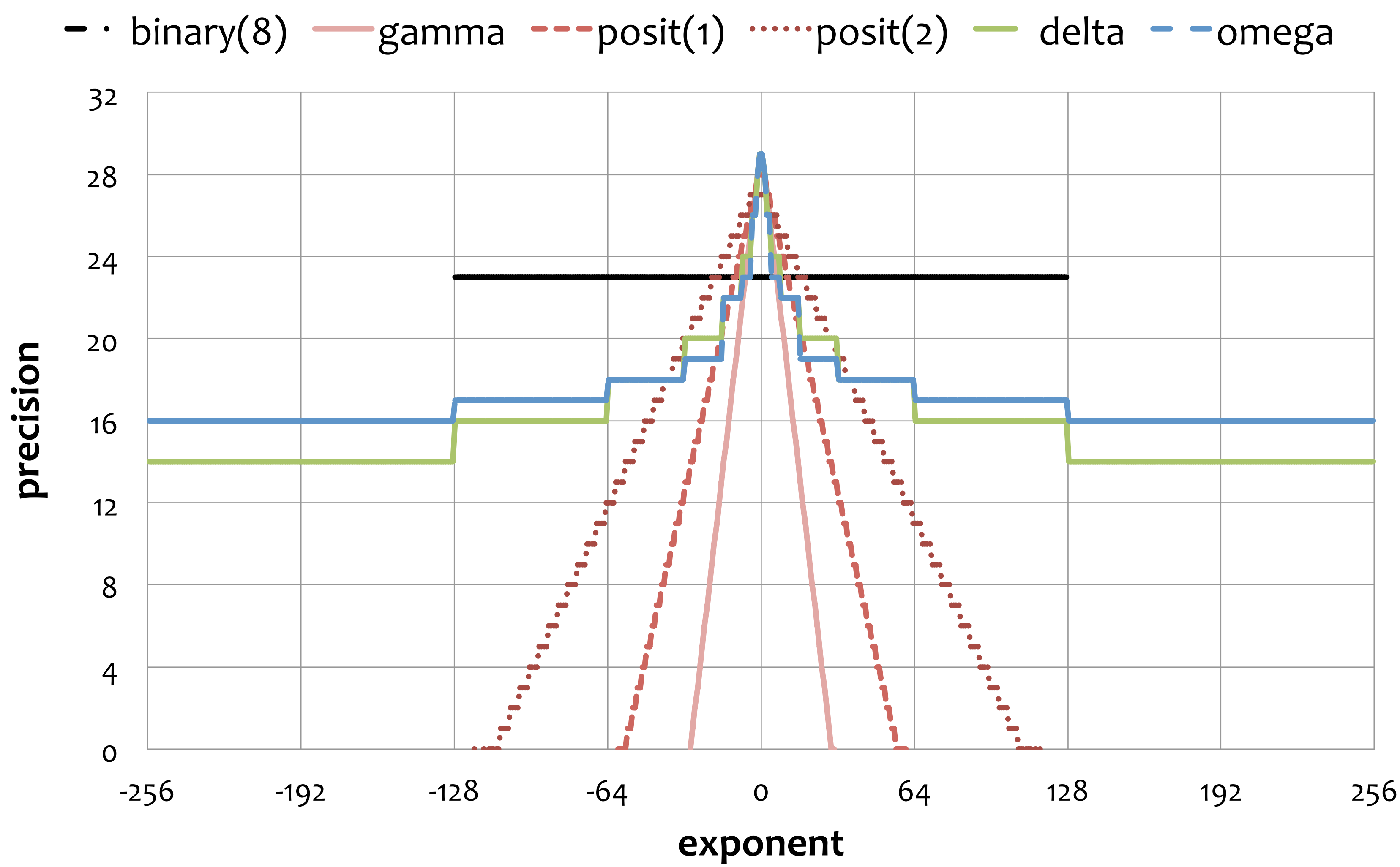
Via reciprocal map, decode function for posits (es = 0) is C^∞ except at y = 1, where it is C^1



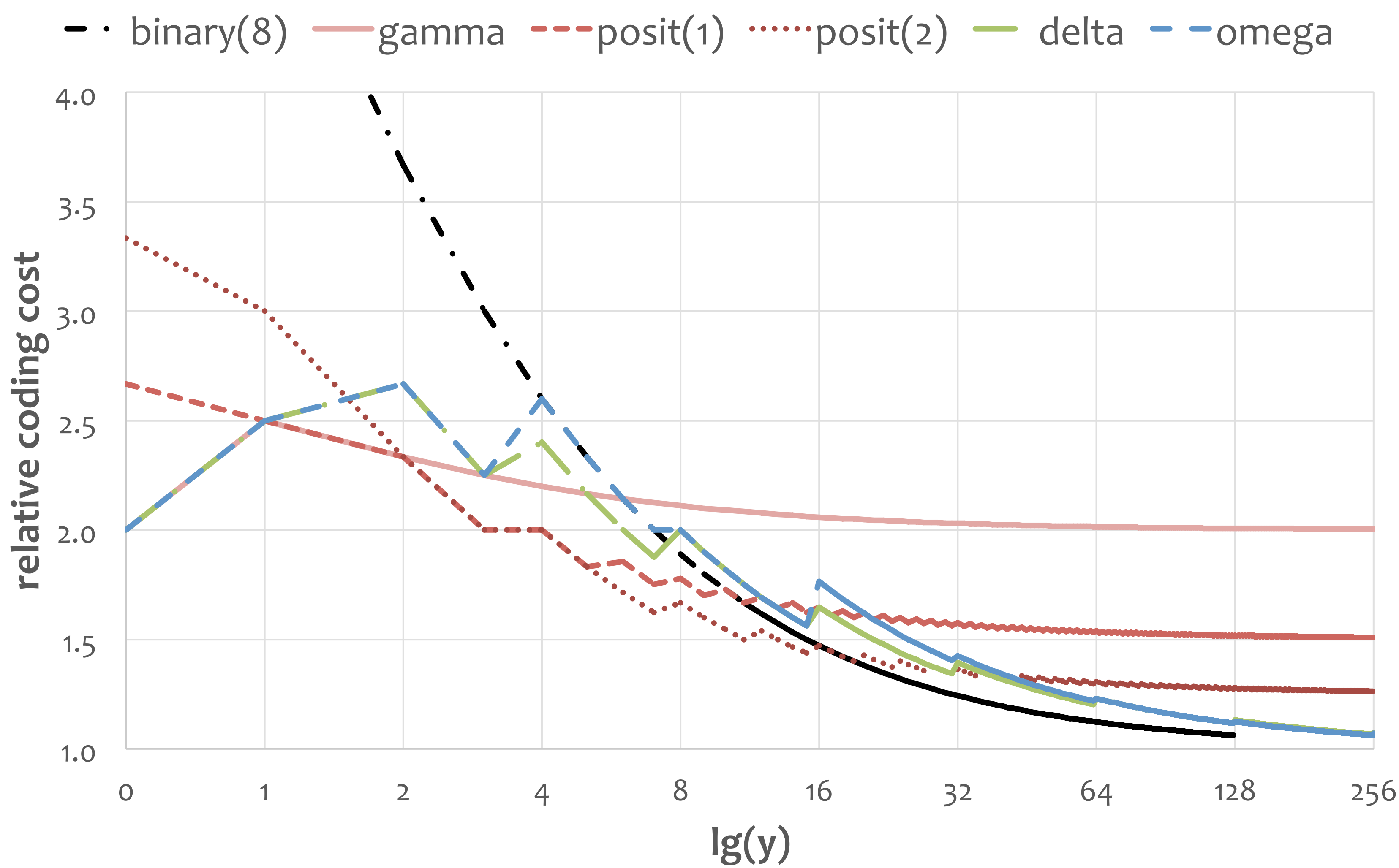
Properties of 16-bit types

type name	exponent	fraction map	dynamic range (bits)	additive closure	additive error	multiplic. closure	multiplic. error
IEEE half	5-bit binary	linear	~40	8.79%	1.08e-4	0.37%	3.16e-2
binary	5-bit binary	linear	~32	8.74%	1.84e-3	0.34%	3.89e-1
exponential	5-bit binary	exponential	~32	0.01%	1.80e-3	**46.24%**	3.88e-1
Elias omega2	Elias omega	linear	~32	40.35%	1.60e-4	0.23%	1.27e-2
Elias omega3	Elias omega	linear	**130,048**	47.30%	?	1.02%	?
Elias delta	Elias gamma	linear	16,384	41.88%	?	0.24%	?
posit0 (aka. gamma)	unary	linear	28	**46.88%**	**9.36e-5**	0.16%	**7.05e-4**
posit1	Rice 1	linear	56	24.22%	1.17e-4	0.28%	1.29e-3
posit2	Rice 2	linear	112	12.31%	1.30e-4	0.48%	2.31e-3
posit3	Rice 3	linear	224	7.70%	1.32e-4	0.84%	4.12e-3
zfp	block float	linear	2,108	N/A (data dependent)			

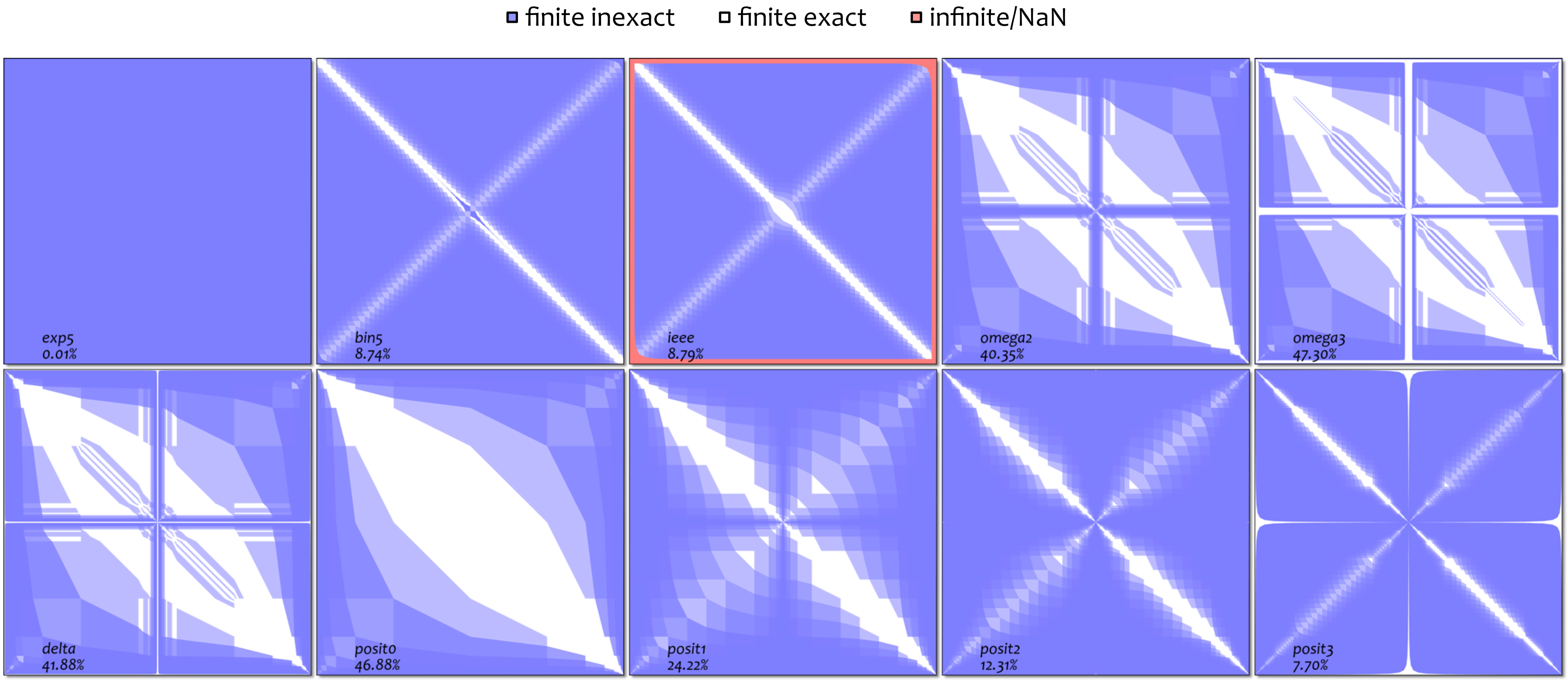
Non-IEEE types exhibit tapered precision



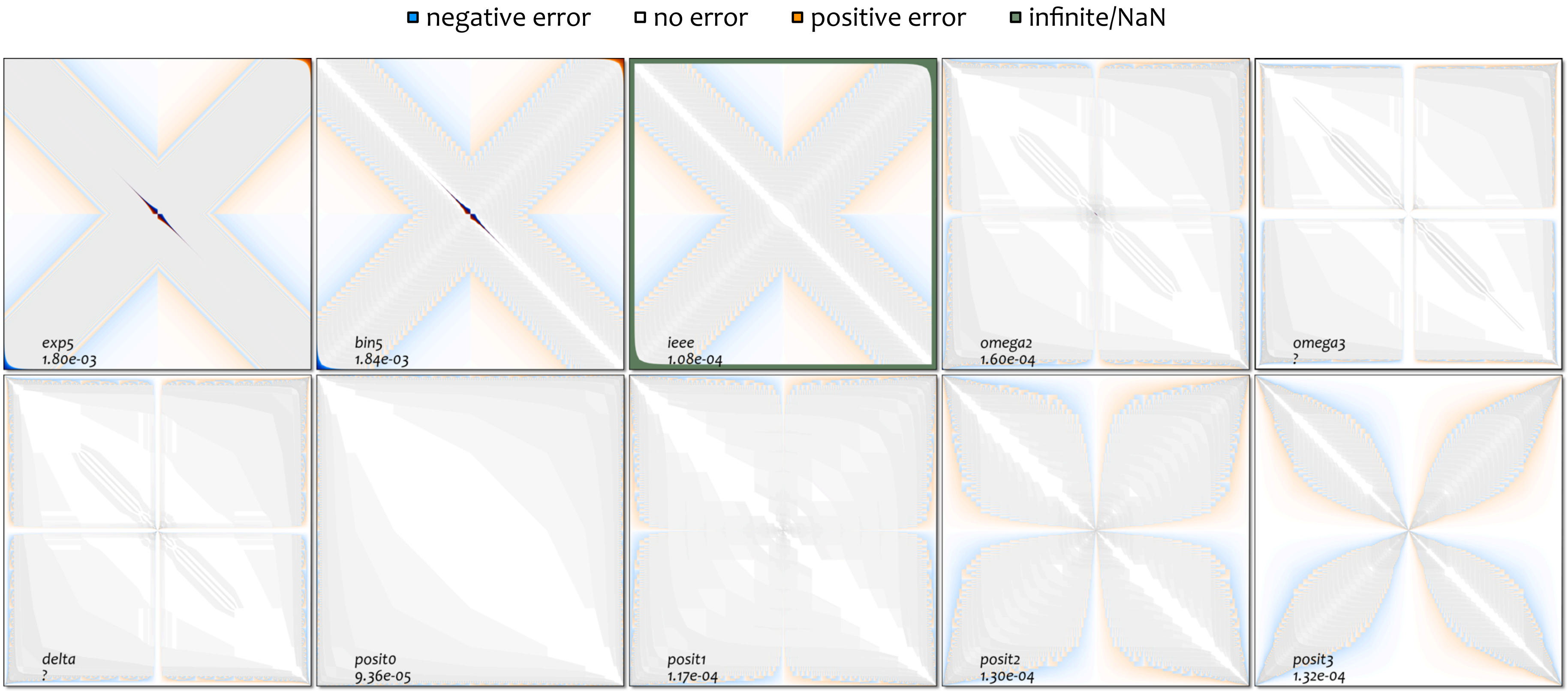
Elias delta and omega are asymptotically optimal



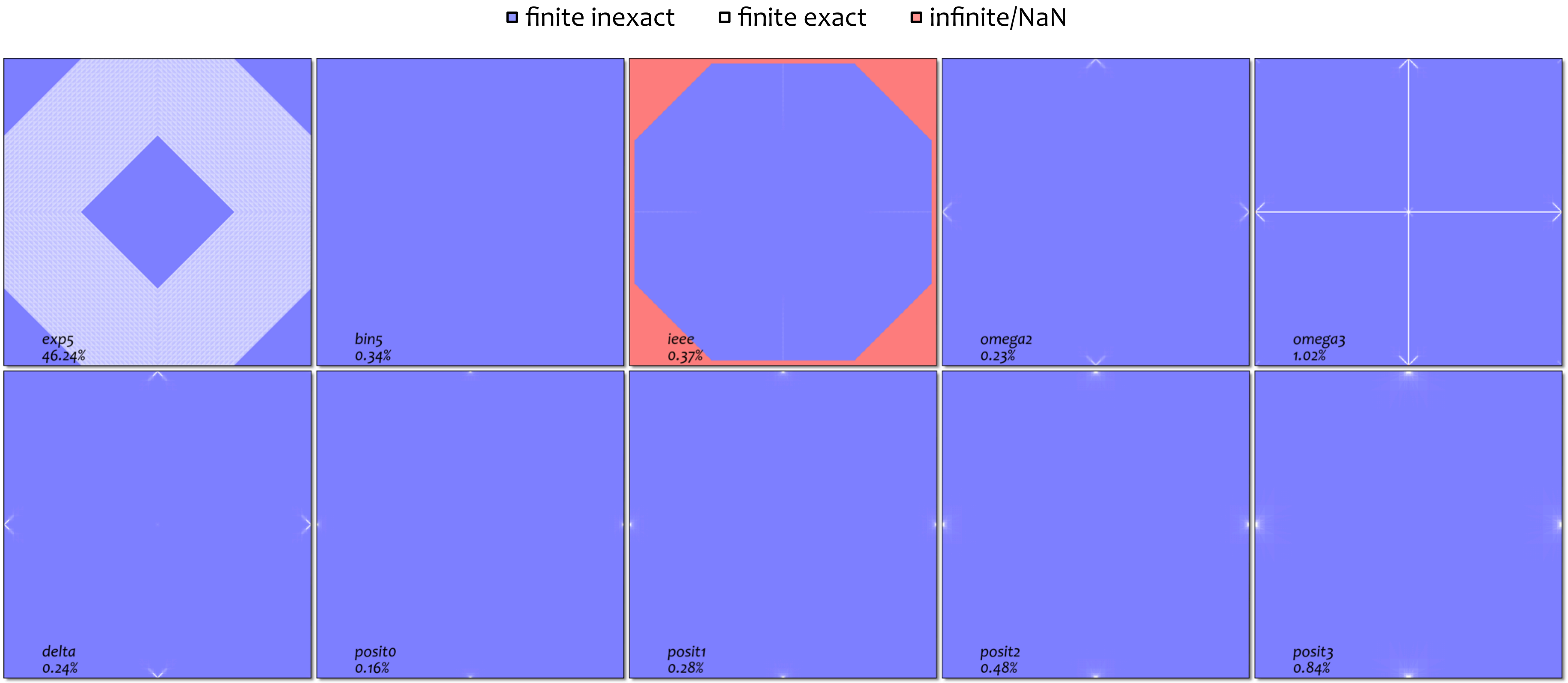
Closure for 16-bit addition—representable sums



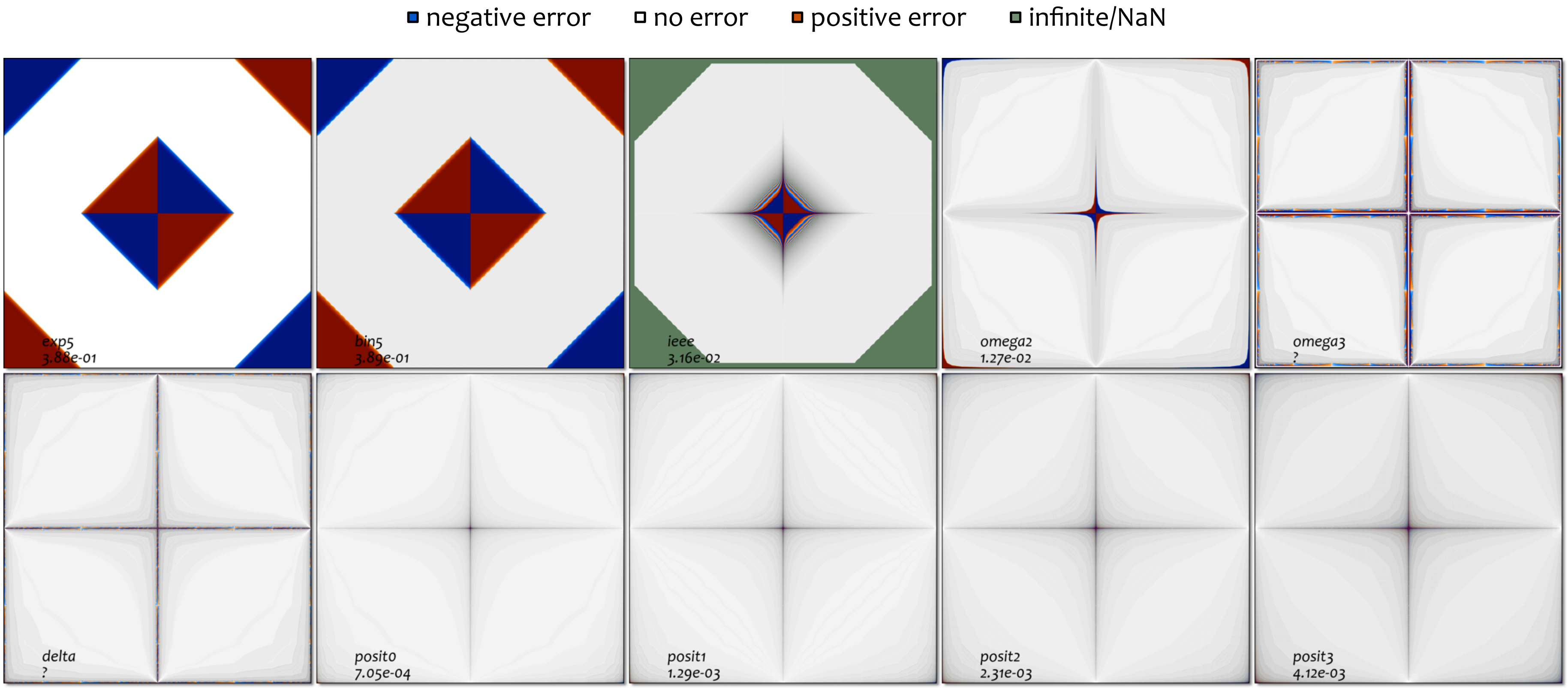
Mean relative error for 16-bit addition



Closure for 16-bit multiplication—representable products



Mean relative error for 16-bit multiplication



Open Discussion

Live Q&A: `tinyurl.com/sc17qna`

Feedback survey: `tinyurl.com/sc17bof`

For more information:

`tinyurl.com/fpanalysis`

`posithub.org`

Don't forget to fill out the conference survey as well:

`submissions.supercomputing.org/eval.html`

Unum & Posit