Rethinking Finite Precision

IPAM Science at Extreme Scales Workshop IV: New Architectures and Algorithms

CASC
Center for Applied Scientific Computing

30 November 2018

Jeffrey A. F. Hittinger
Director

LLNL-PRES-767245

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

This presentation describes the work of a diverse team of mathematicians and computer scientists

Numerical Analysis Team
Jeff Hittinger (PI)
Daniel Osei-Kuffuor
Geoff Sanders
Alyson Fox
David Beckingsale

Tool Development Team
Daniel Quinlan (Co-PI)
Markus Schordan
Scott Lloyd
Harshitha Menon
Tristan Vanderbruggen

Students
James Diffenderfer
James Herring
Hoang Duong
Pavol Klacansky
Garrett Folks

External Collaborators	
Valerio Pascucci	U. Utah
Alfredo Metere	ICSI, Berkeley
Mike Lam	JMU

Data Analysis Team
Peter Lindstrom (Co-PI)
Timo Bremer
Harsh Bhatia

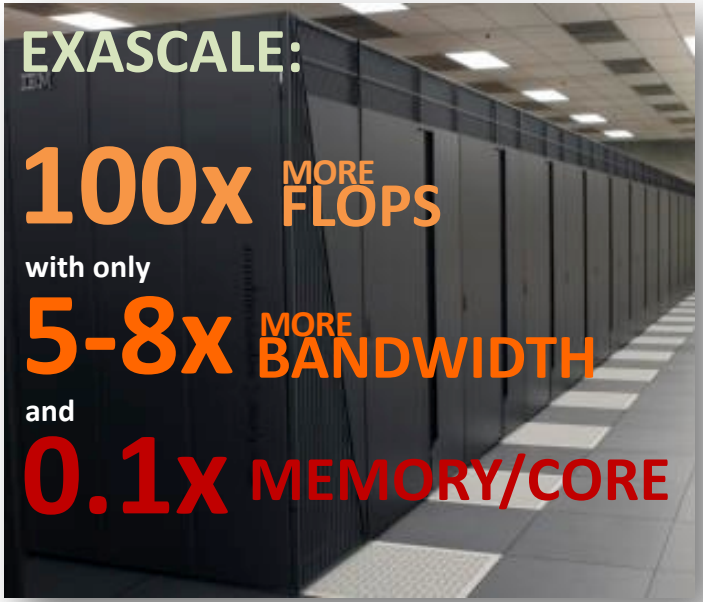
Funded by LLNL Laboratory Directed Research and Development as Project 17-SI-004: Variable Precision Computing

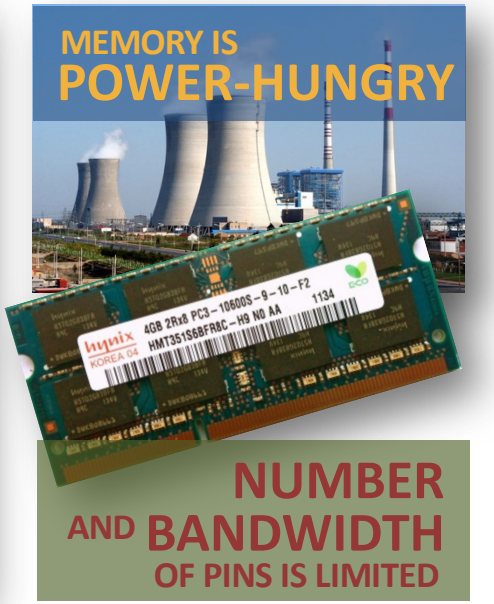
How to make an omelette poorly…

- Eggs come by the dozen

- Making an omelette, do you break all 12 but use only 3?
- This is how we use bits!

Data motion and memory capacity are becoming the limiting factors in high performance computing





We could solve bigger and/or more complex problems if we represented data more efficiently

Save the bits!

How can we represent data in a more efficient way?

We are developing a new paradigm:

Variable Precision Computing

- Increase scientific throughput up to 10x
- Reduce data storage needs by 50-99%
- Open up a new direction for algorithms

Others are pursuing mixed precision, reduced precision for specialized applications, and floating-point compression, but our approach is more comprehensive

Computing is experiencing disruptive changes – it's a good time to challenge the status quo

What do we do when we simulate on a computer?

Infinite Dimensional Vector Space

$$\frac{\partial u}{\partial t} = \mathcal{H}(u(x,t))$$

$$x \in \Omega \subset \mathbb{R}^D$$

$$t \in (0,T] \subset \mathbb{R}$$

$$u : \Omega \times (0,T] \to \mathbb{R}$$

$$\mathcal{H} : \mathbb{R} \to \mathbb{R}$$

$$u(x,t) = g(t) \quad \forall x \in \partial\Omega$$

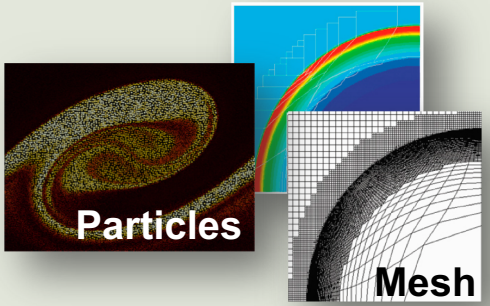
$$u(x,0) = u_0(x)$$

Finite Dimensional Vector Space

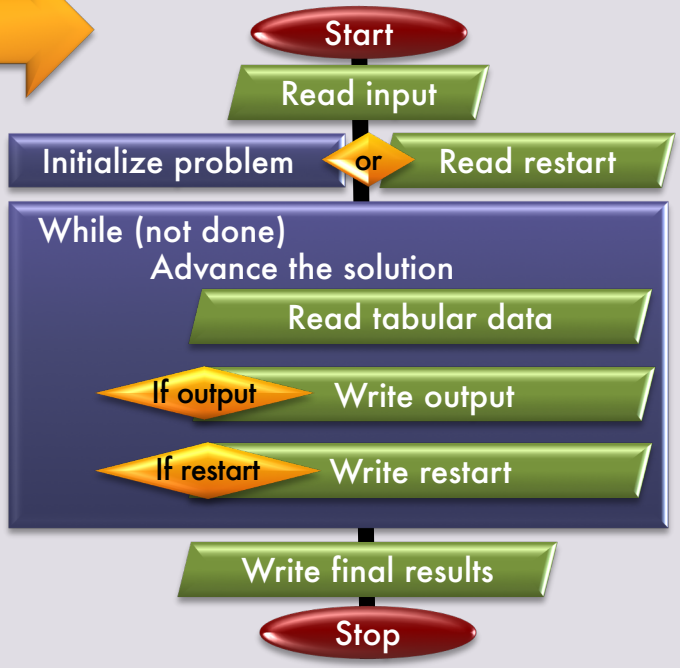
$$u^{n+1} = H(u^n)$$

$$u \in \mathbb{R}^N$$

$$H : \mathbb{R}^N \to \mathbb{R}^N$$

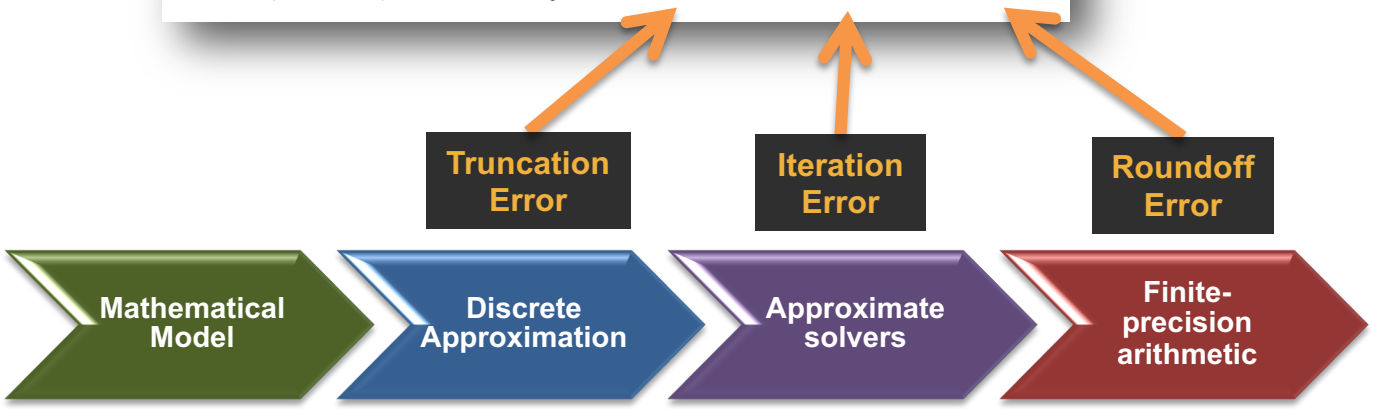


Simulation Code



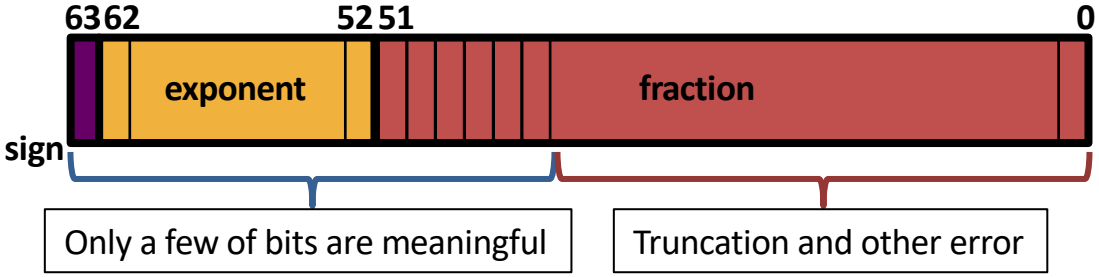
There are many approximations in simulations that produce corresponding approximation errors

$$u(x,t) = u_i^n + \epsilon_{\mathrm{T}} + \epsilon_{\mathrm{I}} + \epsilon_{\mathrm{R}}$$



Roundoff errors typically do not dominate

We use double precision floating-point by default, even when few significant digits are needed



- Many of the bits are error
- 11 bit exponent: 616 orders of magnitude
- This is wasteful!
  - Use more work, power, or time than necessary
  - Move around lots of meaningless bits

Do we need so much dynamic range?

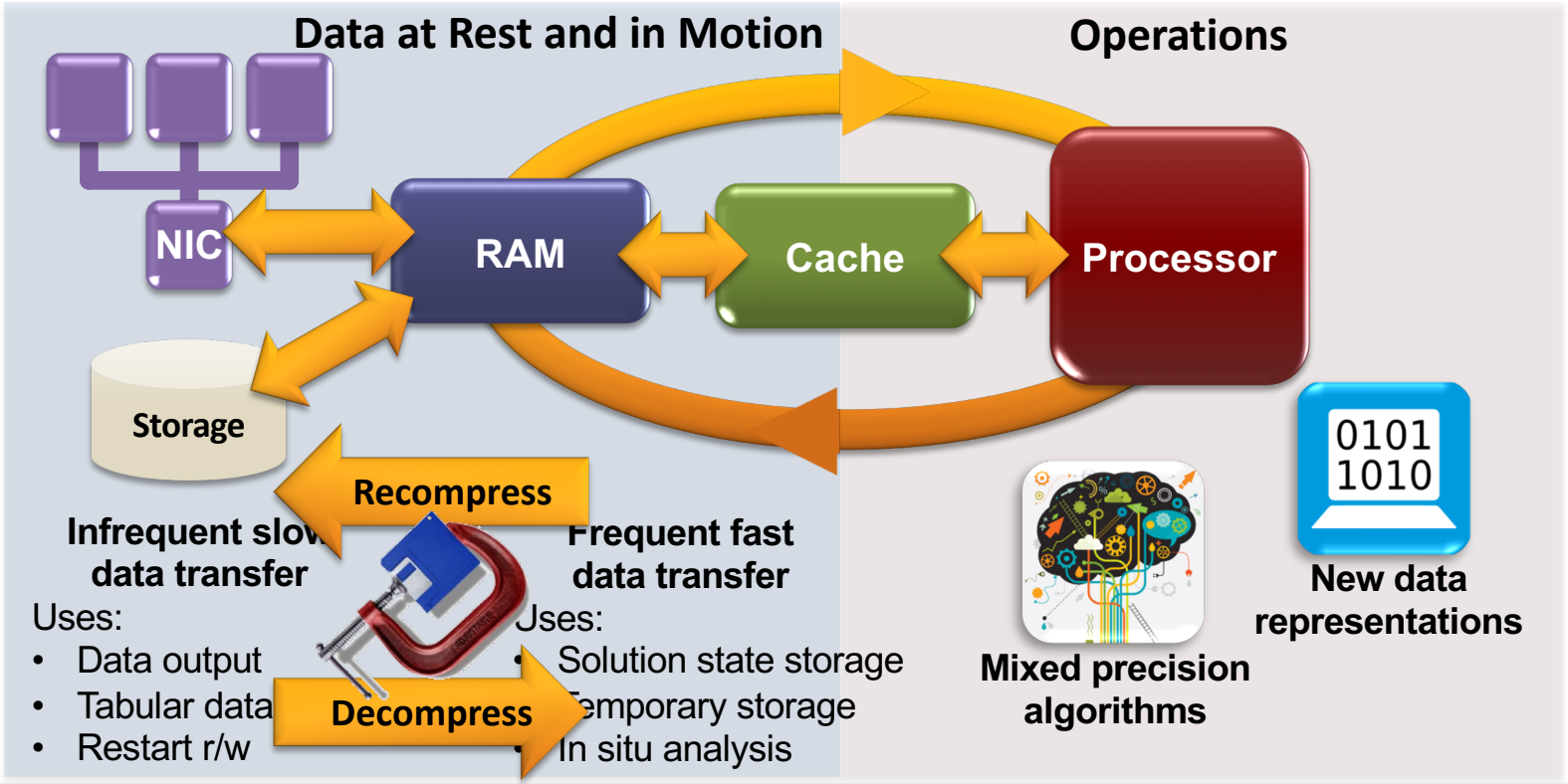
$$\frac{\text{Diameter of universe}}{\text{Planck length}} \sim 10^{61}$$

# of atoms in universe $\sim 10^{81}$

$$\frac{\text{Mass of universe}}{\text{Electron mass}} \sim 10^{83}$$

Eliminate the bottlenecks: use only as many bits as needed

Where in the computer can we modify precision?



We are building on and going beyond existing work on varying precision for computations

Single precision

- 40 yrs ago, memory was limited
- Expertise developed to use single precision
- Build on known techniques, e.g., iterative refinement
- **Requires great care**

Mixed precision

- Most current work (e.g., Buttari, Dongarra, Demmel, et al.)
- Static, task-based
- **One-off implementations**
- **OK for numerical libraries (MAGMA, XBLAS), but difficult for applications**

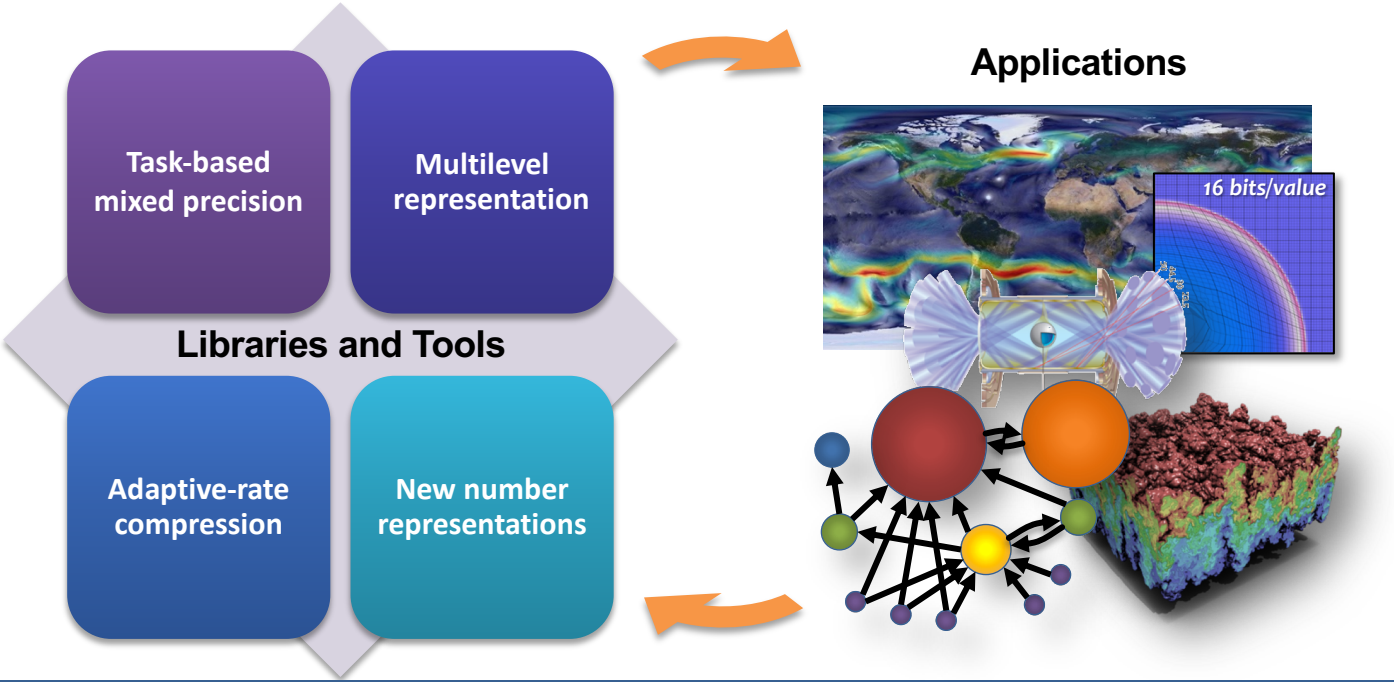
Arbitrary-precision arithmetic

- Focused on extending precision
- Computing irrationals
- May be able to leverage some of the ideas (GNU MPC)
- **Slow**

New number representations

- Do not accept the world as given
- What can be done beyond IEEE floating point?
- Unum, posit, Levenstein, Elias gamma, etc.
- **Difficult to change the inertia of computing as we know it**

We are investigating multiple techniques for varying precision to address the data motion and compute bottlenecks



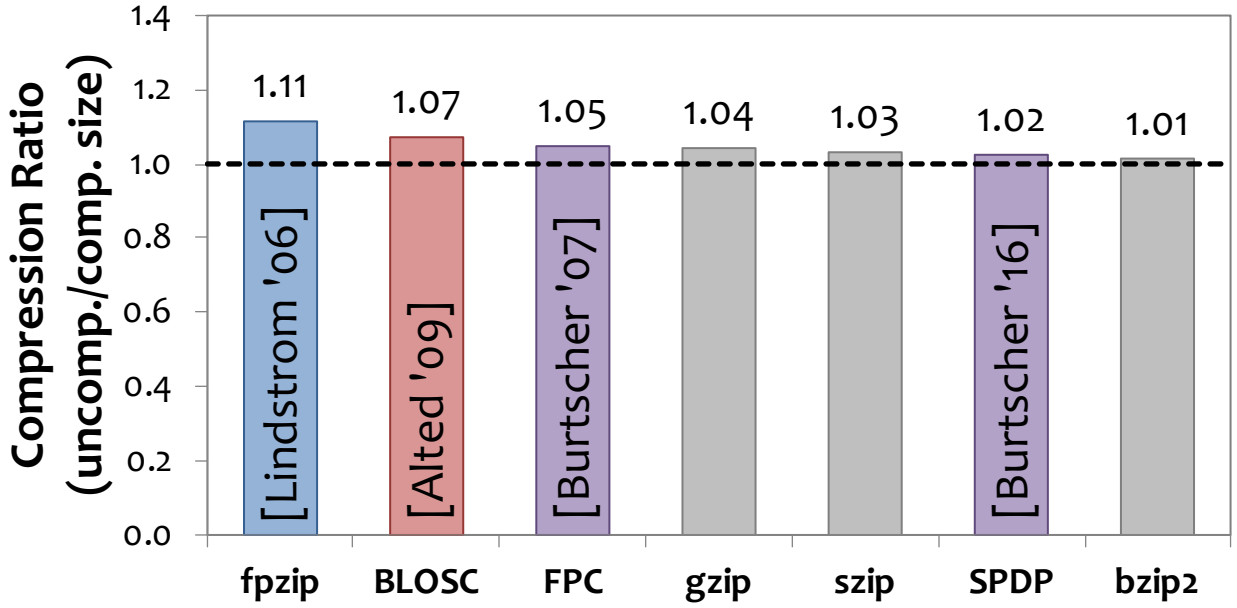
Requires close collaboration between computer scientists, applied mathematicians & domain experts

Chapter 1: Using compression in data storage and analysis

It is quality rather than quantity that matters

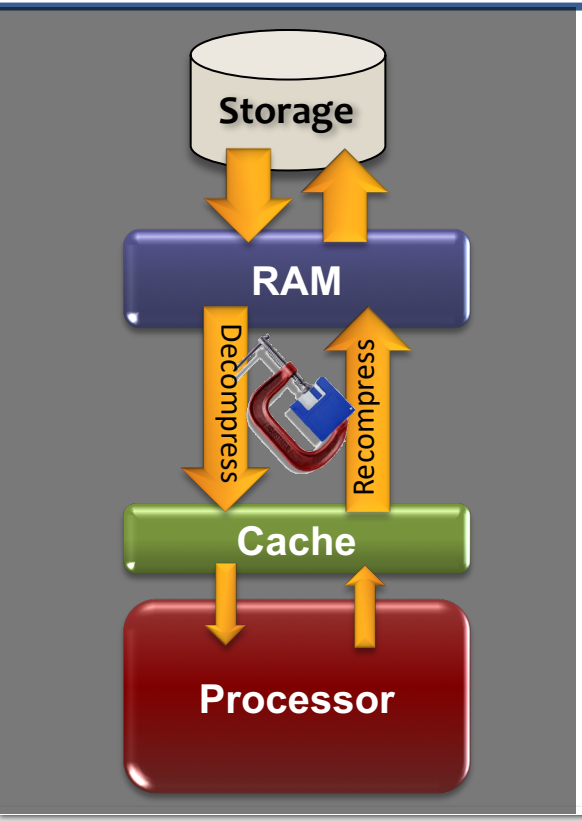
- Lucius Annaeus Seneca

Floating-point data is difficult to compress— lossless compression is often not sufficient



Lossy compression does far better, but is often met with skepticism

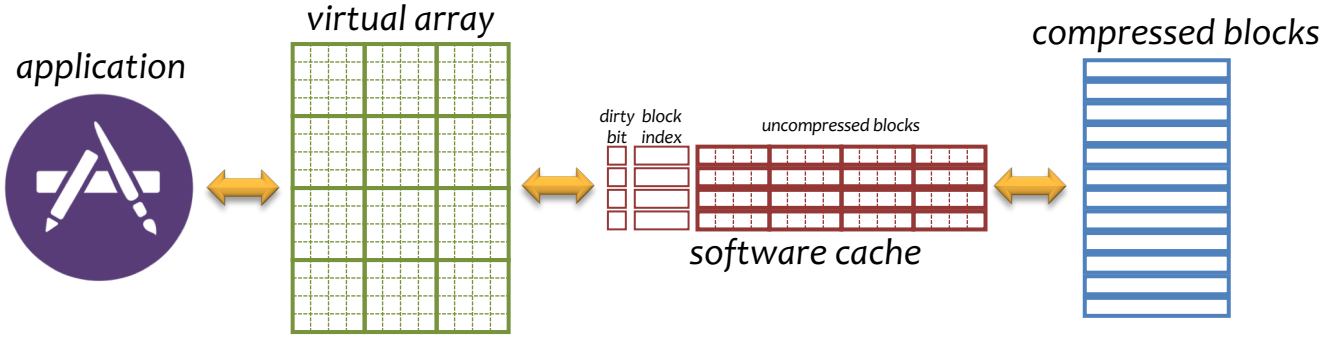
Lossy compression can address the I/O bottleneck



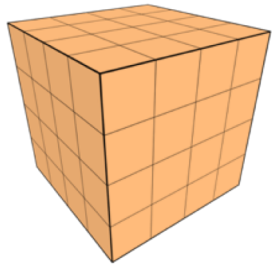
- **Develop algorithms & software supporting adaptive precision where errors do not amplify**
- **Approaches balance compression and adaptivity**
  - Locally adaptive ZFP: Adaptive Rate Compression (ARC)
  - Multi-resolution data format (IDX) with ARC
  - Data optimal algorithms for IDX+ARC
- **Applications include**
  - Data analysis
  - Visualization – demonstrated compression of 100x
  - Restart
  - Tabular data – demonstrated compression of 4-10x

ZFP: The first inline compressor for floating-point arrays

[Lindstrom, *IEEE Trans Vis. Comput. Graphics*, 2014]



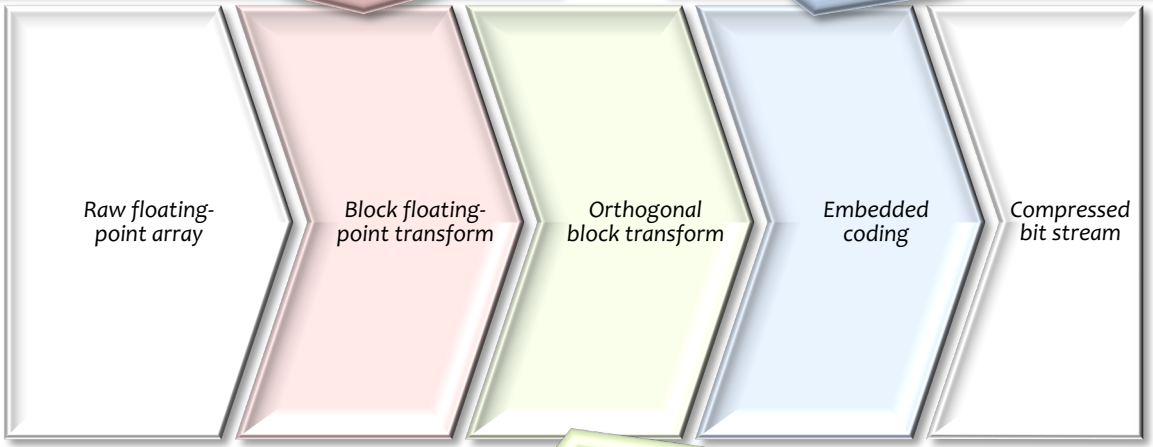
- **Inspired by ideas from h/w texture compression**
  - 1D, 2D, or 3D array divided into fixed-size 4×4×4 blocks
  - Fixed-size blocks ⇒ **random read/write access**
  - Blocks independently (de)compressed to user-specified number of bits or quality
  - (De)compression is done inline, on demand
  - Write-back cache of uncompressed blocks limits data loss
- **Compressed arrays via C++ operator overloading**
  - Can be dropped into existing code by changing type declarations
  - double a[n] ⇔ std::vector<double> a(n) ⇔ zfp::array<double> a(n, **precision**)



The ZFP compressor is comprised of three distinct components

- *Align values in a 4^d block to a common largest exponent*
- *Transmit exponent verbatim*

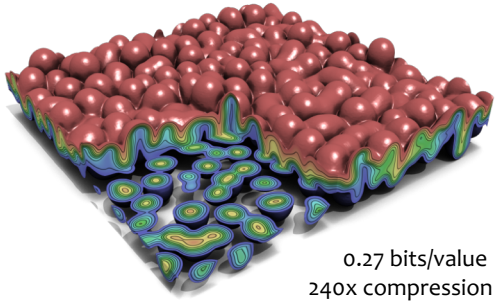
- *Encode one bit plane at a time from MSB using group testing*
- *Each bit increases quality—can truncate stream anywhere*



- *Lifted, separable transform using integer adds and shifts*
- *Similar to but faster and more effective than JPEG DCT*

ZFP is not YAC (yet another compressor)

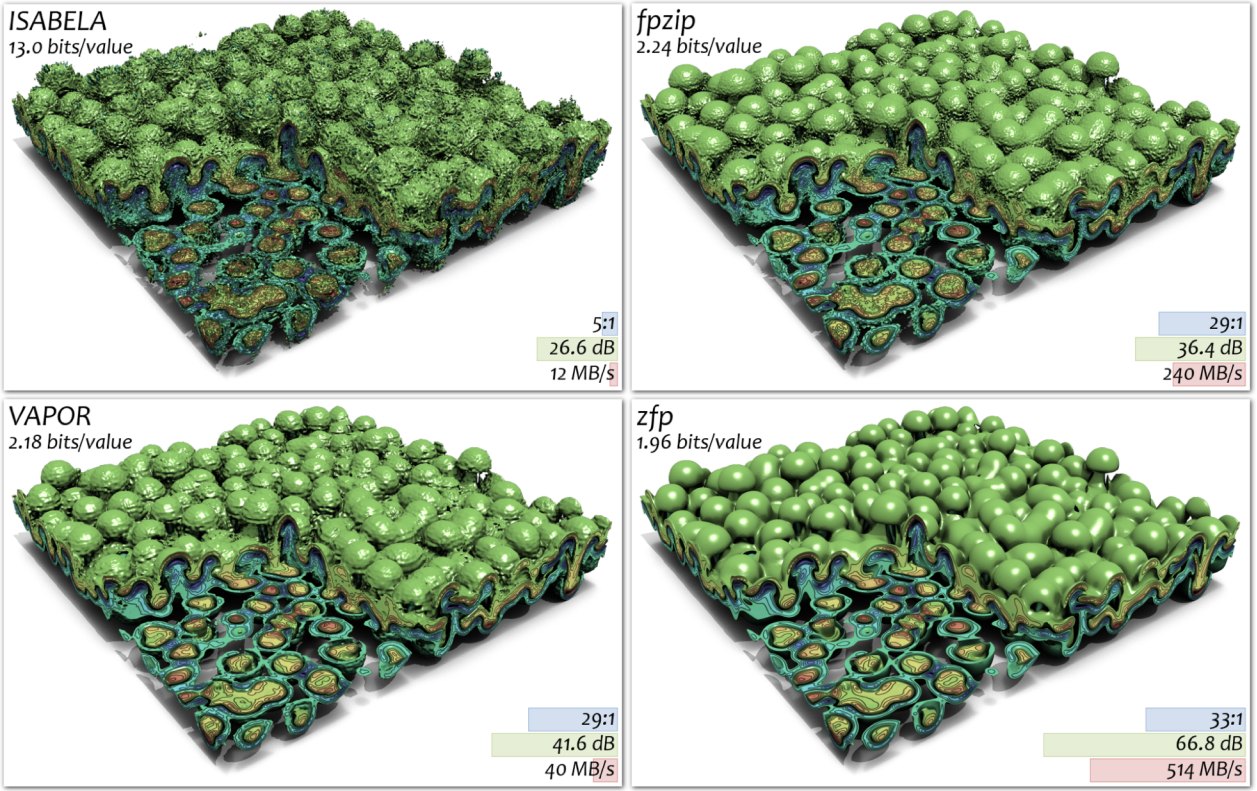
- **Three compression modes in single CODEC**
  - **Fixed rate**: Only inline (de)compressor with **read and write random access**
  - **Fixed precision**: Fixed no. mantissa bits ensures **relative error bound**
  - **Fixed accuracy**: User tolerance ensures **absolute error bound**
- **Very high quality across many science domains**
  - **>100x more accurate** than closest competitor
- **High, symmetric encoding & decoding speed**
  - Up to **2 GB/s/core**: **2-6x faster** than closest competitor
  - Simple algorithm amenable to **hardware implementation**



0.27 bits/value
240x compression

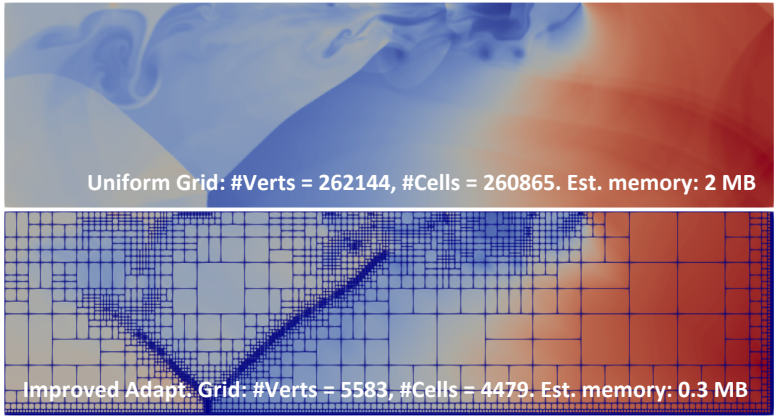
- **Small, independent blocks of compressed data**
  - Fine granularity provides **adaptive quality**, **culling**, **queries**, **domain decomp**, etc
  - Supports streaming with **tiny memory footprint**
  - **OpenMP**, **GPU parallelization** over (and within) blocks
  - **Resilience** to data corruption—flipped bit affects only single block

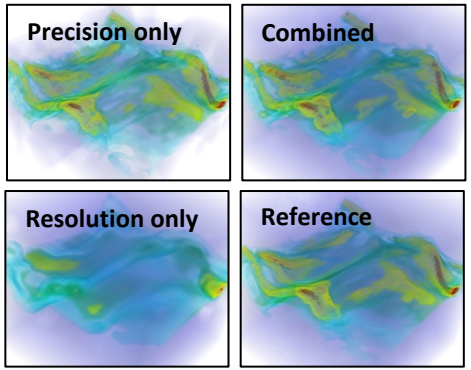
ZFP lossy compression shows no artifacts in derivative computations (velocity divergence)



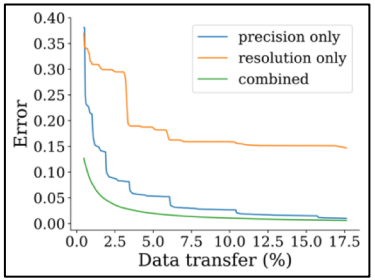
We have developed a flexible in-memory representation of variable resolution/precision data

- **Exact wavelet ordering gives more efficient representation**
- **Extensive experiments**
  - Simulations and images
  - Reduction in RMS Error by combining resolution and precision in data reduction
- **Identified data-independent bit orderings optimized for RMS Error, gradient, Laplacian, histogram, and isocontour**
- **Localization can bring further savings**





Magnetic reconnection: 0.1 bits per sample

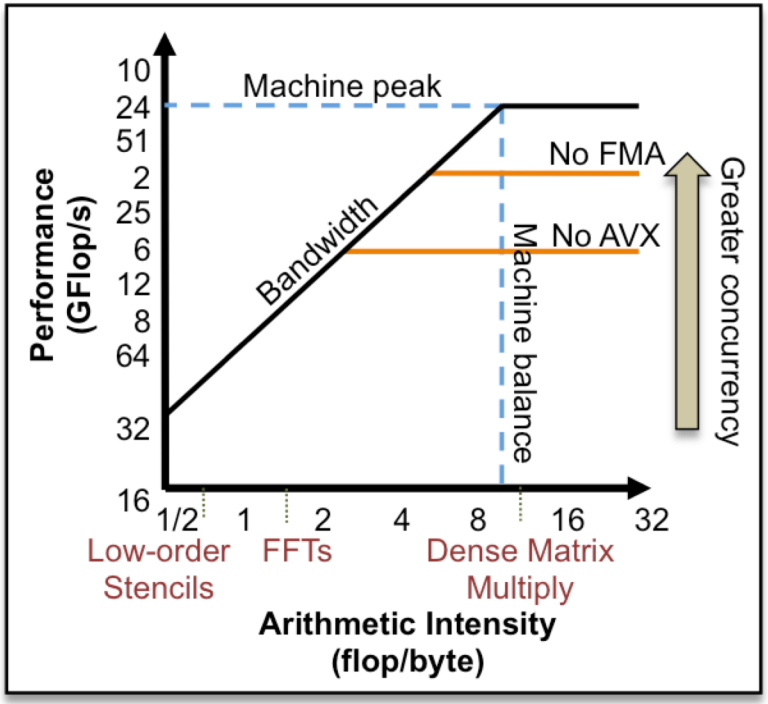


Part 2: Compression to battle memory bandwidth limits

The world as we have created it is a process of our thinking.
It cannot be changed without changing our thinking.

— **Albert Einstein**

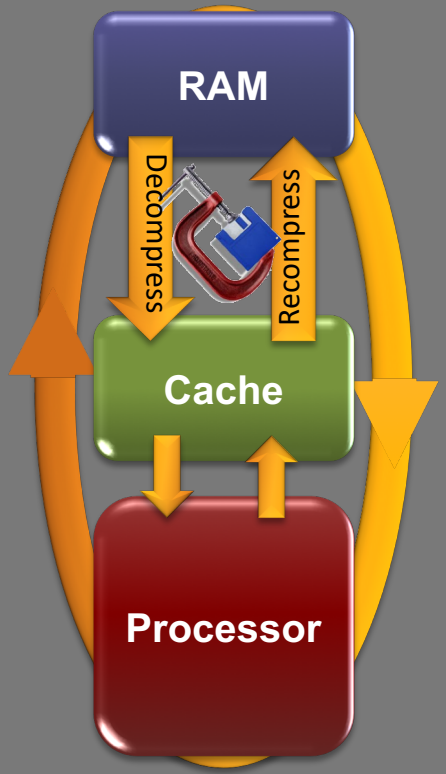
The roofline model help to explain visually some reasons why we get such a small percentage of peak performance



[Williams *et al.*, *Comm. ACM*, 2009]

- **Moving the balance point requires changes in hardware**
- **Another option: increase AI**
  - Increase the number of (useful) operations on transferred data
    - Higher-order discretizations
    - Loop fusion
  - Transfer less data
    - Communication avoiding or hiding algs
    - Subroutine fusion
    - ***COMPRESSION***

Can we make use of compression beyond I/O?

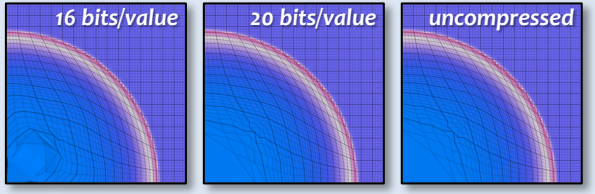


- **Address memory bandwidth limit while computing**
  - Store data in memory in compressed format
  - Decompress before computing
  - Recompress after computing
- **Ideally, handle compression/decompression in hardware**
- **How does this modify the simulation result?**
  - Compression errors can accumulate
  - Could effect accuracy and stability of algorithms

In lab codes, we have shown that 4x inline lossy compression reproduces results with little error

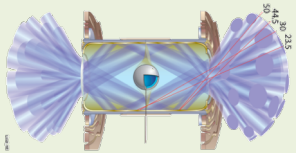
LULESH: Lagrangian shock hydrodynamics

- QoI: *radial shock position*
- 25 state variables compressed over 2,100 time steps
- At **4x compression**, relative **error < 0.06%**



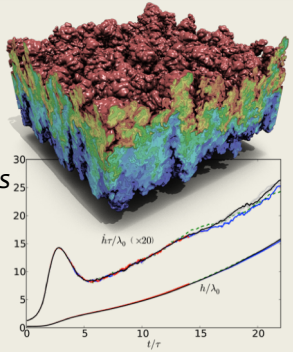
pf3D: Laser-plasma multi-physics

- QoI: *backscattered laser energy*
- At **4x compression**, relative **error < 0.1%**



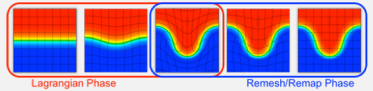
Miranda: High-order Eulerian hydrodynamics

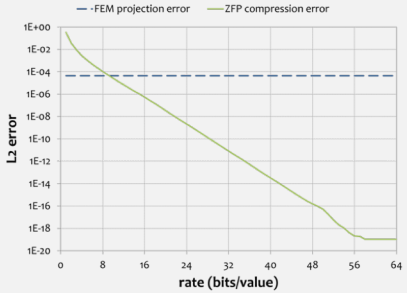
- QoI: *Rayleigh-Taylor mixing layer thickness*
- 10,000 time steps
- At **4x compression**, relative **error < 0.2%**



MFEM: Cubic finite elements

- QoI: *function approximation*
- **6x compression** with ZFP
  error < 0.7% relative to FEM error

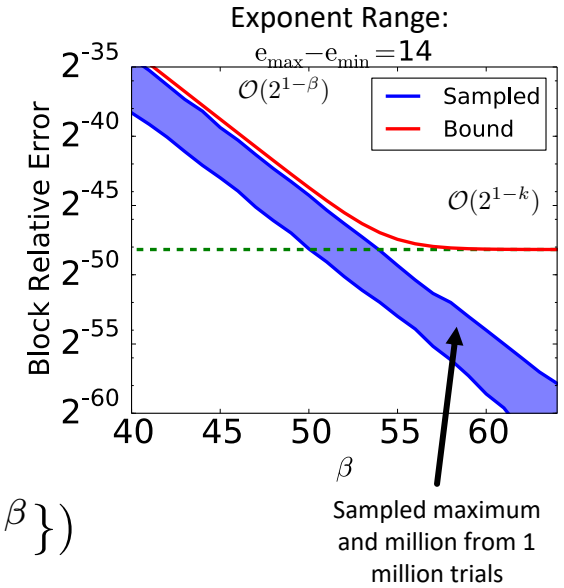




We have derived theoretical bounds for error caused by inline compression

$$\|D(C(\vec{\mathbf{x}})) - \vec{\mathbf{x}}\|_\infty \leq K_\beta \|\vec{\mathbf{x}}\|_\infty$$

- Decompression Operator
- Compression Operator
- Original Data



Error introduced through lossy compression and decompression is bounded in the max norm (pointwise)

$$K_\beta := \mathcal{O}(\max\{2^{1-k}, 2^{1-\beta}\})$$

Machine Precision

ZFP Fixed Precision (β: bit-plane index)

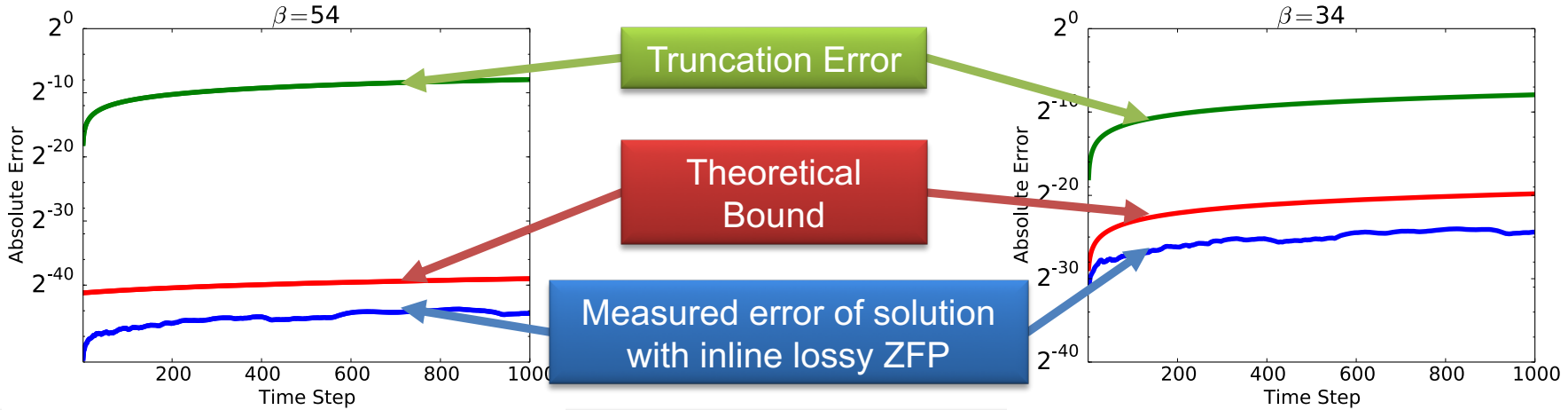
[Diffenderfer, Fox, H., Sanders, Lindstrom, SISC, (accepted); arXiv: 1805.00546]

Assuming standard properties, the ZFP error bound can be used to bound inline compression error for iterative methods

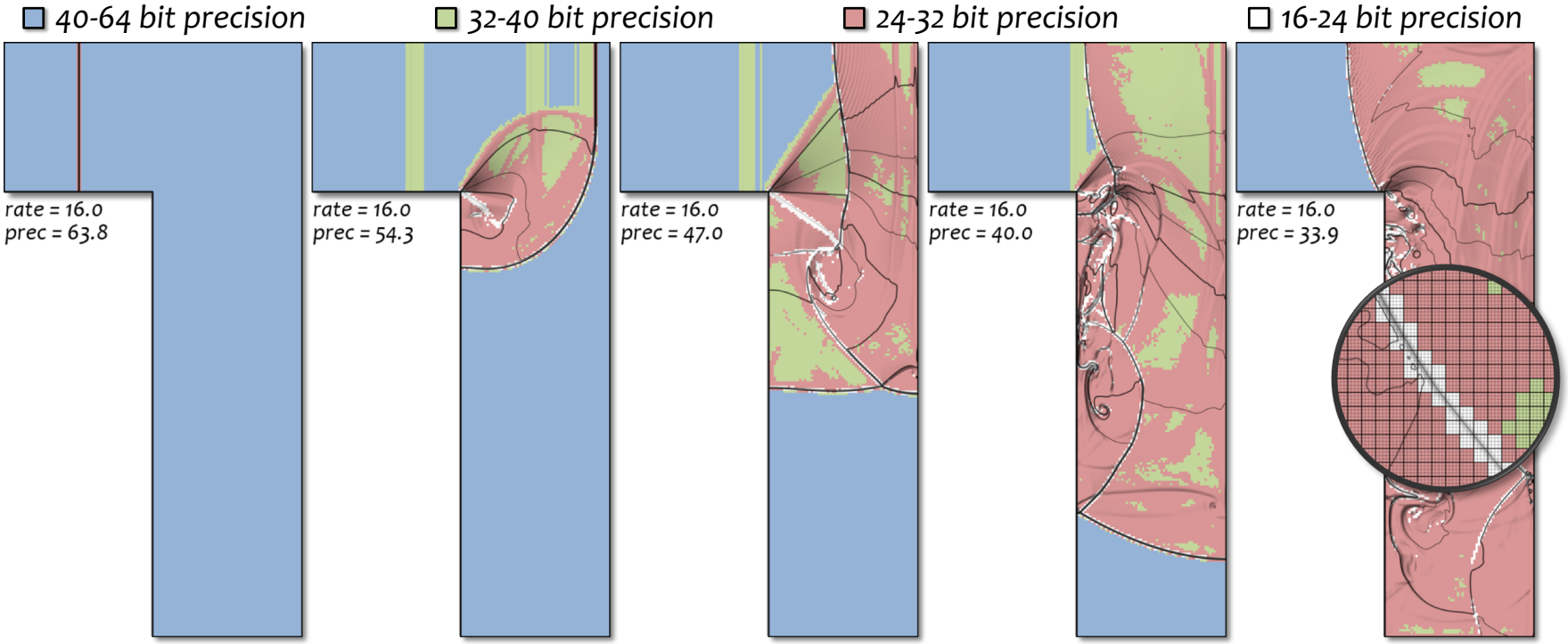
- **Consider bounded advancement operators** ($||Ak|| \leq M$)

$$\textbf{Theorem: } \| \underbrace{A(D(C(\vec{\mathbf{v}}^t)))}_{\text{(de)compression}} - A\vec{\mathbf{u}}^t \|_\infty \leq M \sum_{j=0}^{t} K_{\beta_j} \|\vec{\mathbf{v}}^j\|_\infty,$$

- **Example:** 1D Lax-Wendroff scheme with periodic boundary conditions ($M \leq 2$)

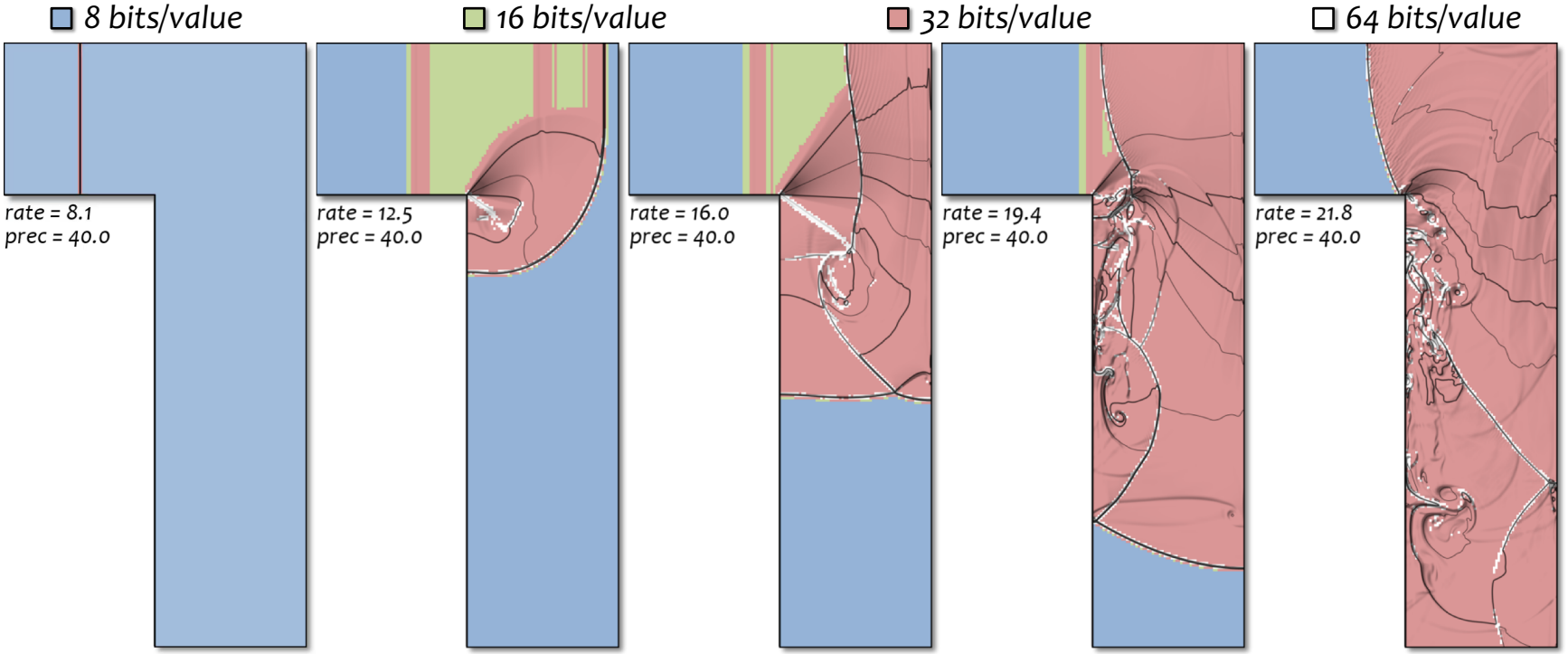


ZFP's fixed-rate arrays result in highest precision in regions of the domain that least need it



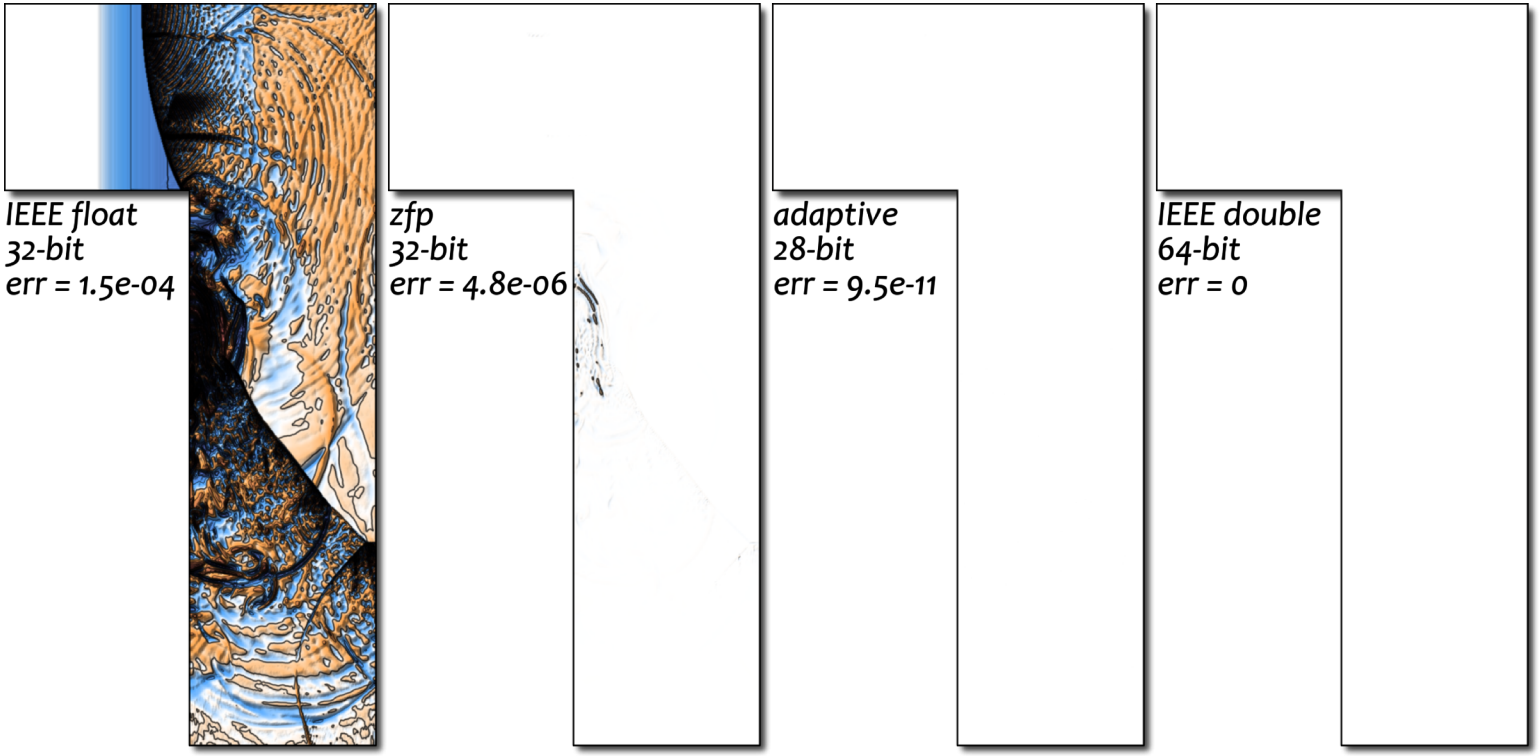
ZFP's conventional arrays use fixed rate (storage size) and variable precision

ARC: ZFP variable-rate arrays that adapt storage spatially and allocate bits to regions where they are most needed

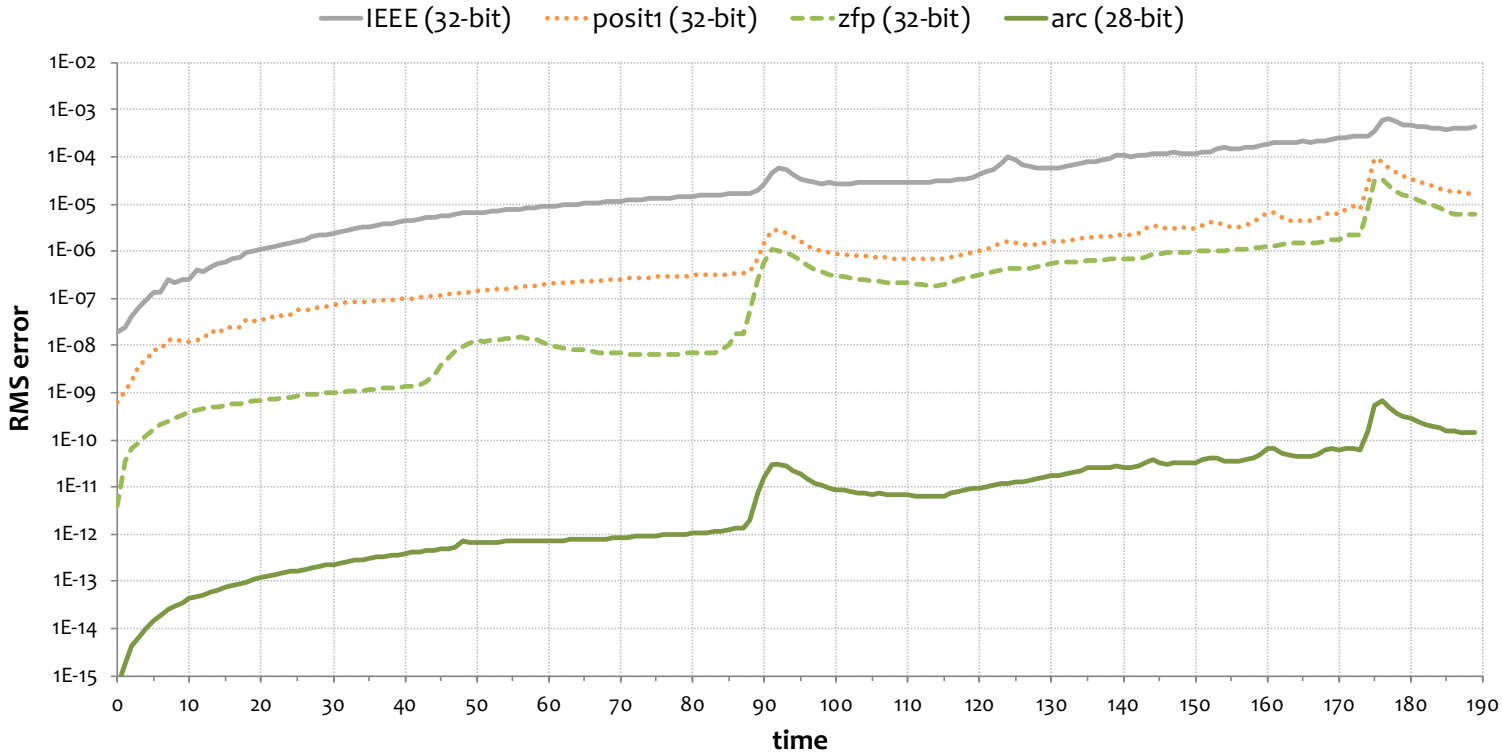


ZFP's new adaptive arrays (ARC) use variable rate (storage size) and fixed precision

ZFP adaptive arrays improve accuracy in PDE solution over IEEE by 6 orders of magnitude using less storage



ARC prototype improves accuracy in Euler2D PDE solution over IEEE by 6 orders of magnitude using less storage

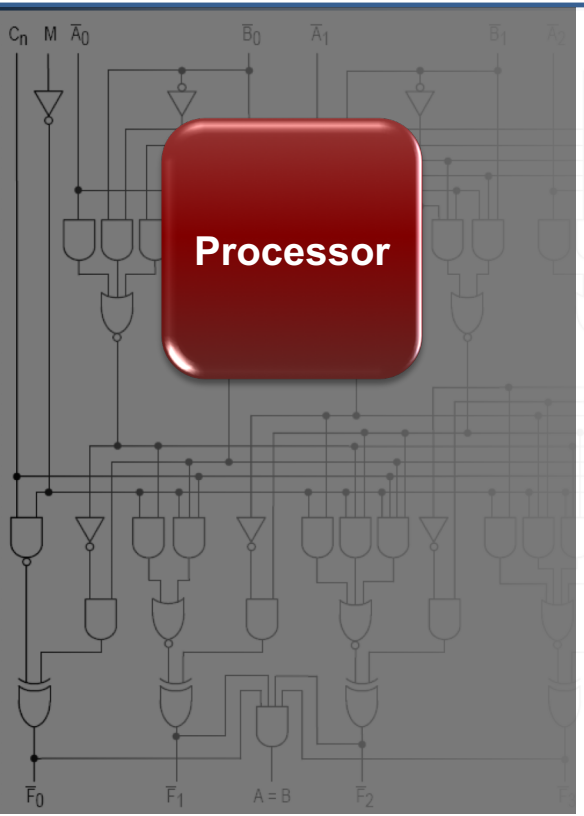


Part 3: Augmenting standard floating point

Without deviation from the norm, progress is not possible

- Frank Zappa

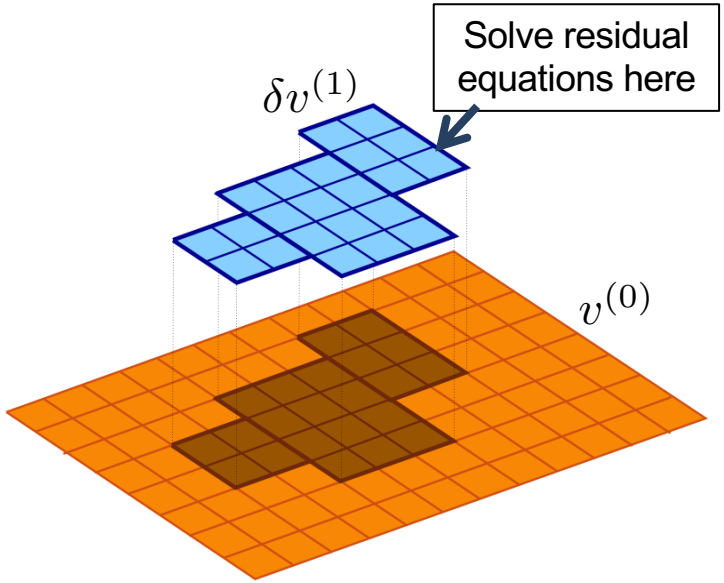
Can we also optimize the number of flops?



- **Goal is to accelerate calculations by computing only on meaningful bits**
  - Single precision can be more than 2x speed-up
  - Tensor core hardware on NVIDIA GPUs even greater for FP_{16}
- **Error transport and iterative refinement algorithms**
  - Can we make these adaptive?
  - Bootstrap a solution up to the required precision
- **Tasked-based mixed precision**
  - Optimal eigensolvers for quantum molecular dynamics
  - Very low precision eigensolvers for graph analysis
- **New floating point representations**
  - Better use of bits
  - Better mathematical properties

We are investigating the potential for an AMR-like dynamic, local mixed precision based on error transport

- **Dynamic Mixed Precision**
  - Hierarchical representation: sum of singles
  - Block-based refinement
  - Most calculations in single precision
  - Key issues
    - Refinement criteria
    - Propagation of round-off error
    - Cost/benefit
- **Use Nonlinear Error Transport (NET) techniques to understand error evolution**
- **We have shown that Error Transport gives the same answer as Iterative Refinement for linear systems**
  - NET useful for non-iterative problems
  - NET provides roundoff error as a field



Results for the explicit integration of the Porous Media Equation

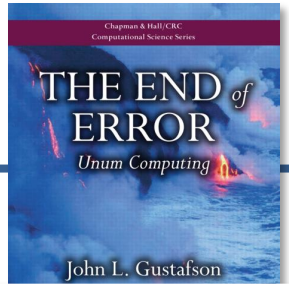
Problem:

- 1D PME, $m = 4$, zero source, $dx = 0.001$ in domain (0, 1), $0 < t < 1$
- $U(t,x) = x * t / (1+t)$

- FE_DP = double precision forward Euler
- FE_SP = single precision forward Euler
- Errors are compared to exact solution in the max norm

Solver	Error	Corrected Error
FE_DP	6.97663e-05	6.9605e-05
FE_SP	0.000695318	6.95884e-05
RK4_DP	6.9605e-05	6.9605e-05
RK4_SP	0.0025453	6.95586e-05

POSITS [Gustafson 2017] are a new float representation that improves on IEEE

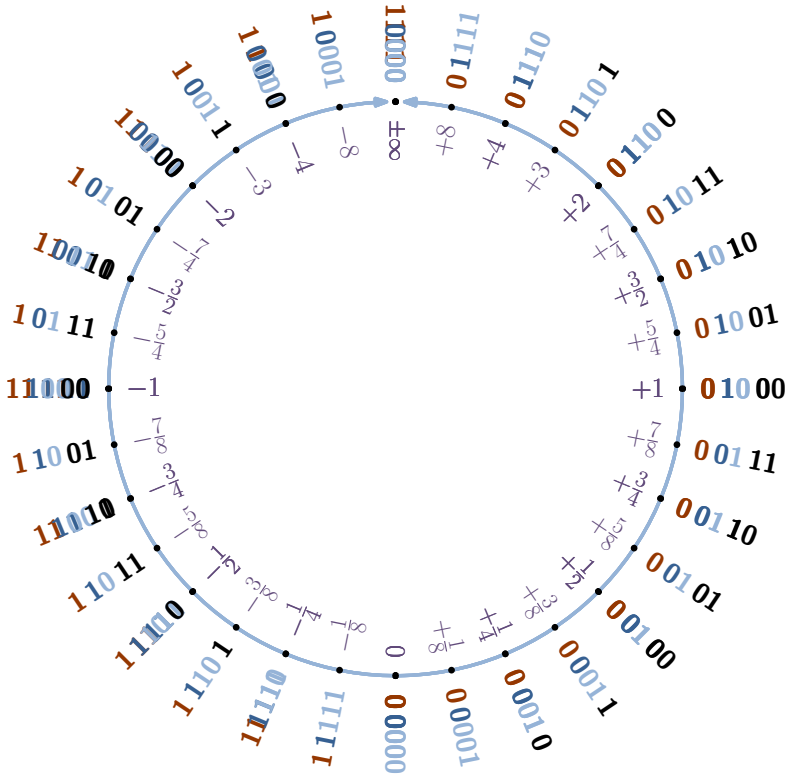


IEEE

	IEEE 754 floating point	Posits: UNUM version 3.0
Bit Length	{16, 32, 64, 128}	fixed length but arbitrary
Sign	sign-magnitude (−0 ≠ +0)	two's complement (−0 = +0)
Exponent	fixed length (biased binary)	variable length (Golomb-Rice)
Fraction Map	linear ($\varphi(x) = 1 + x$)	linear ($\varphi(x) = 1 + x$)
Infinities	{−∞, +∞}	±∞ (single point at infinity)
NaNs	many (9 quadrillion)	one
Underflow	gradual (subnormals)	gradual (natural)
Overflow	1 / FLT_TRUE_MIN = ∞ (oops!)	never (exception: 1 / 0 = ±∞)
Base	{2, 10}	$2^{2^m} \in \{2, 4, 16, 256, \ldots\}$

POSITS, Elias recursively refine intervals (0, ±1), (±1, ±∞)
Example: base-2 posits (aka. Elias γ)

- sign bit
- exponent sign
- exponent value
- fraction value



Beyond POSITS and Elias: NUMREP templated framework

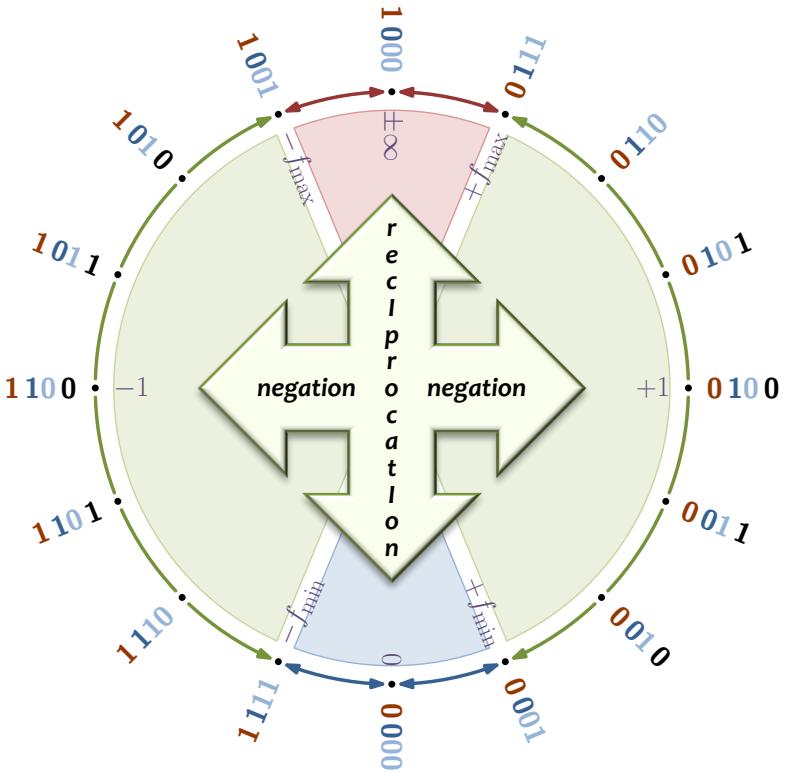
- NUMREP is a modular C++ framework for defining number systems
  - **Exponent coding scheme** (unary, binary, Golomb-Rice, gamma, omega, …)
  - **Fraction map** (linear, reciprocal, exponential, rational, quadratic, logarithmic, …)
    - *Conjugate* fraction maps for sub- & superunitaries enable reciprocal closure
  - Handling of **under- & overflow**
  - **Rounding** rules (to nearest, toward {−∞, 0, +∞})

- NUMREP unifies IEEE, POSITS, Elias, URR, LNS, …, under a single schema
  - Uses auxiliary **arithmetic type** to perform numerical computations (e.g. IEEE, MPFR)
  - Uses **operator overloading** to mimic intrinsic floating types
  - Supports 8, 16, 32, 64-bit types

[Lindstrom, Lloyd, H., CoNGA `18]

Many useful NUMREPS are given by simple interpolation and extrapolation rules (plus closure under negation)

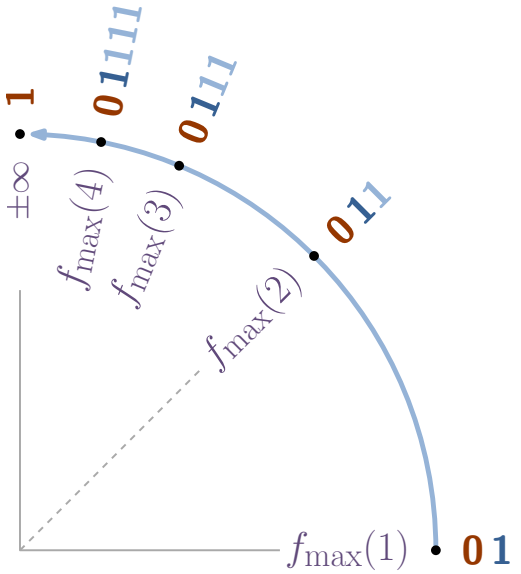
- extrapolation on (f_{max}, ∞)
- interpolation on (f_{min}, f_{max})
- reciprocation on (0, f_{min})



Extrapolation gives next f_{max} between previous f_{max} and ±∞ when adding one more bit of precision

- **Extrapolation rule**: $1 \le f_{max}(p) < f_{max}(p+1) < \infty$

type	$f_{max}(p+1)$	sequence
Unary	$1 + f_{max}(p)$	1, 2, 3, 4, 5, …
Elias γ	$2 \times f_{max}(p)$	1, 2, 4, 8, 16, …
POSITS (base b)	$b \times f_{max}(p)$	1, b, b^2, b^3, b^4, …
Elias δ	$f_{max}(p)^2$	1, 2, 4, 16, 256, …
Elias ω	$2^{f_{max}(p)}$	1, 2, 4, 16, 65536, …



- **Reciprocation rule**: $f_{min}(p) = 1 / f_{max}(p)$

Interpolation rule generates new number $g(a, b)$ between two finite positive numbers $a < b$

- For POSITS, Elias {γ, δ, ω}:

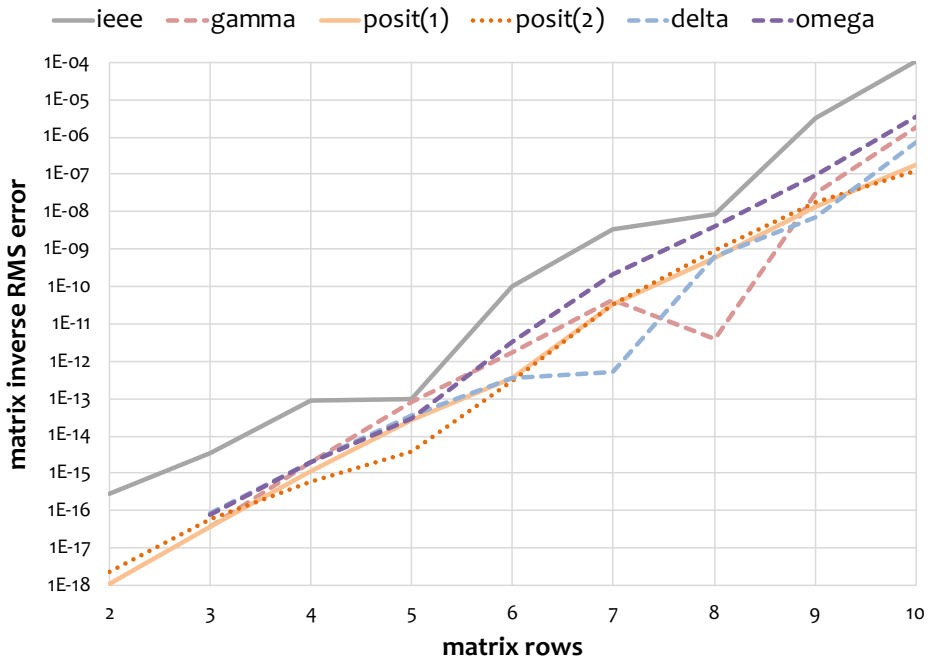
$$g(a,b) = \begin{cases} \dfrac{a+b}{2} & \text{if } b \le 2a \\ 2^{g(\lg a,\, \lg b)} & \text{otherwise} \end{cases}$$

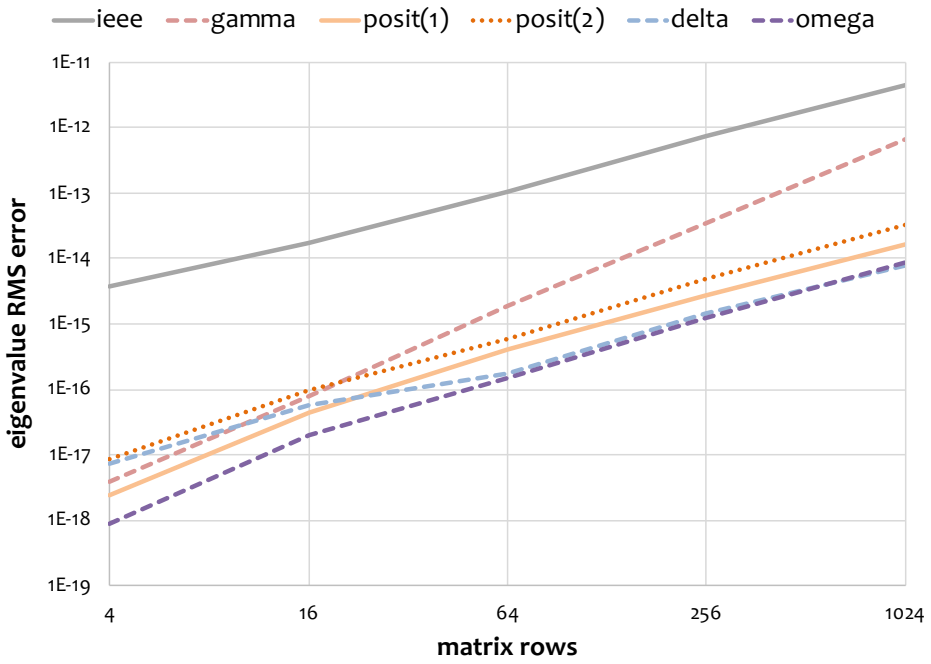
- Examples
  - $g(2, 4) = 3$ — arithmetic mean
  - $g(4, 16) = 2^{g(2,\,4)} = 2^3 = 8$ — geometric mean
  - $g(16, 65536) = 2^{g(4,\,16)} = 2^{2^{g(2,4)}} = 2^{2^3} = 2^8 = 256$ — "hypergeometric" mean (for ω only)

- For LNS:

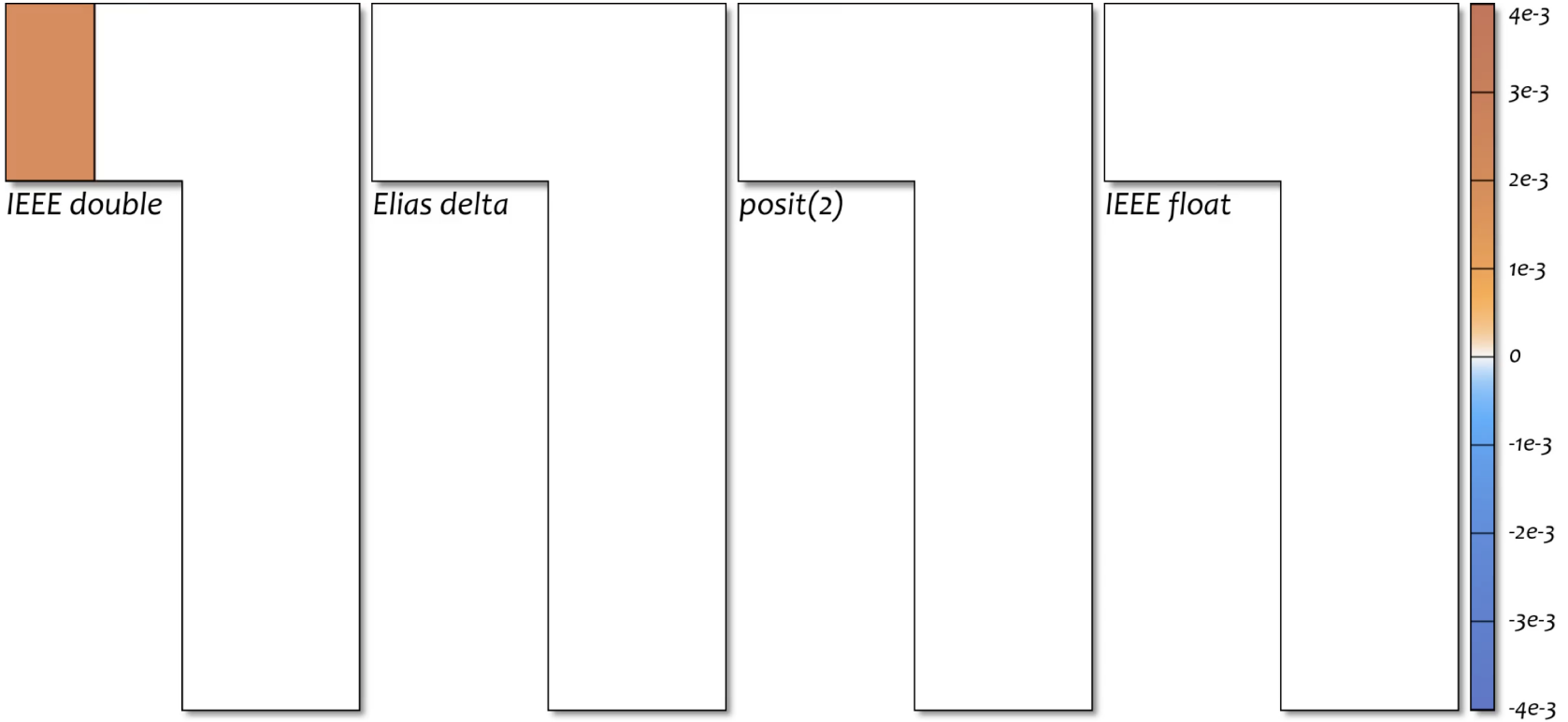
$$g(a,b) = \sqrt{ab}$$

NUMREP has led to new number formats based on Elias codes that improve accuracy over IEEE in linear algebra

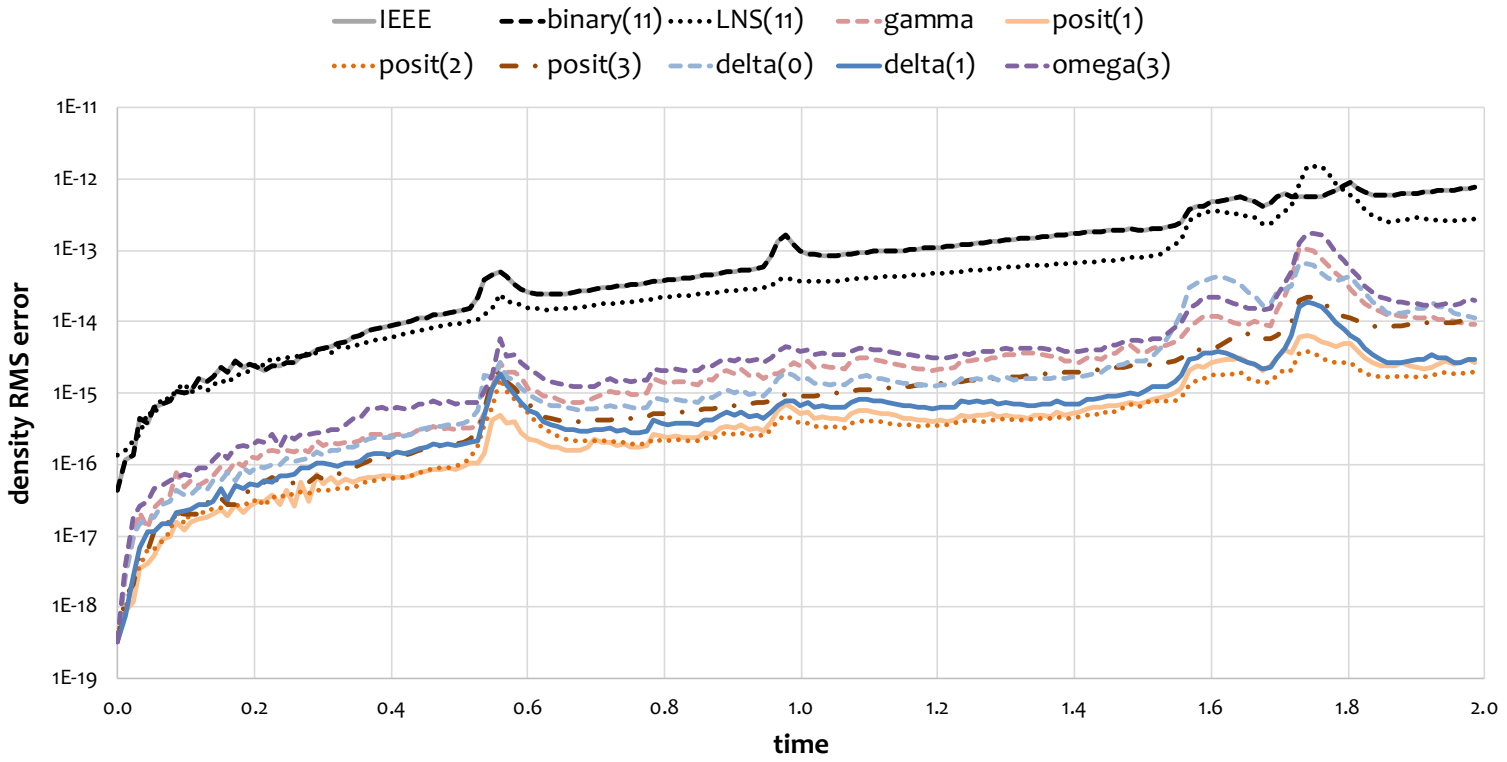




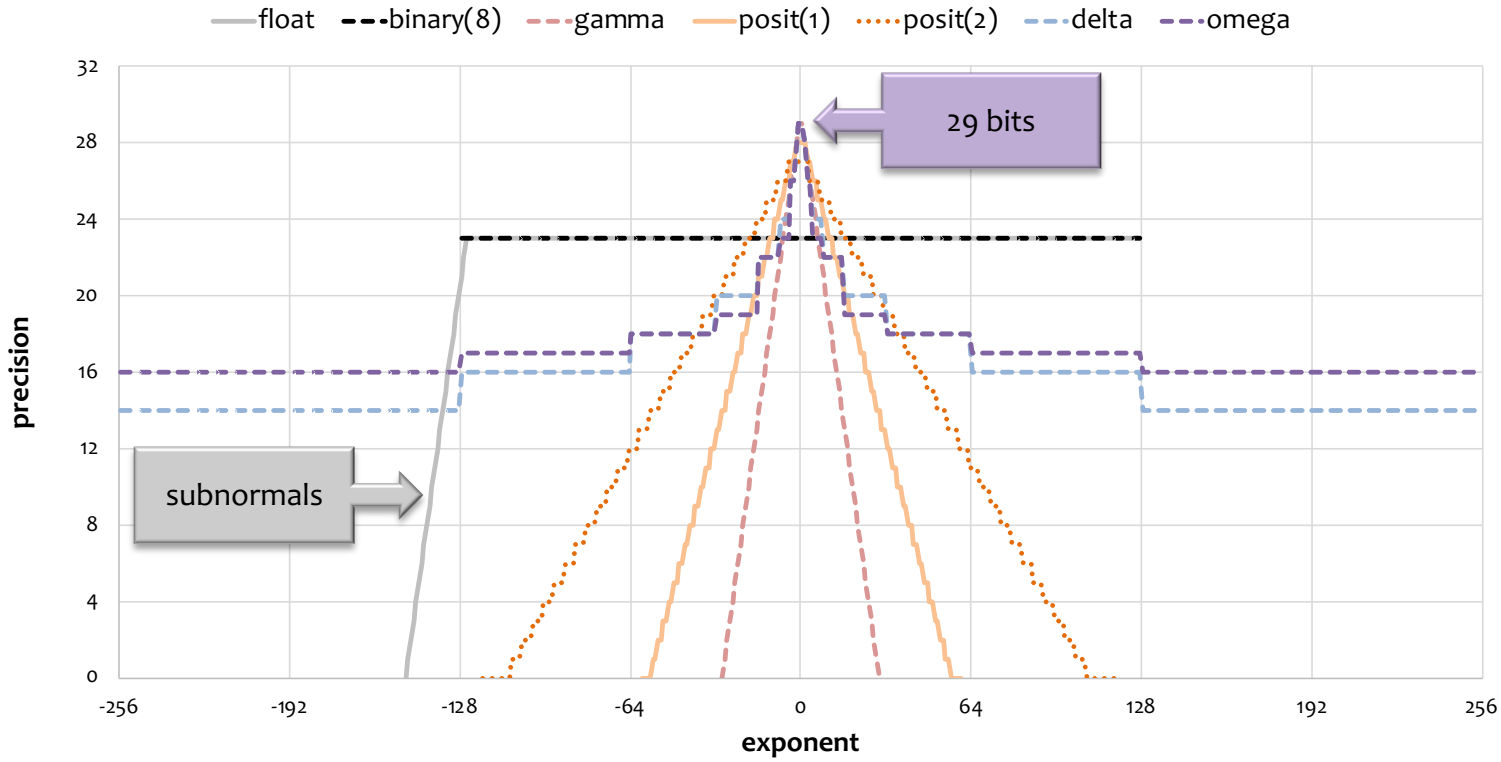
Euler2D shock propagation illustrates benefits of new types



IEEE consistently is the least accurate floating-point representation in numerical calculations (64-bit types)



Variable-length exponents lead to tapered precision: 6-9 more bits of precision than IEEE for numbers near one



Chapter 4: Aiding adoption

I have always wished for my computer to be as easy to use as my telephone;
my wish has come true because I can no longer figure out how to use my telephone

— Bjarne Stroustrup

For Variable Precision Computing to gain acceptance, we must be able to…

Developing techniques that are

- Often more accurate than IEEE
- Provide information about roundoff
- Mathematically justified

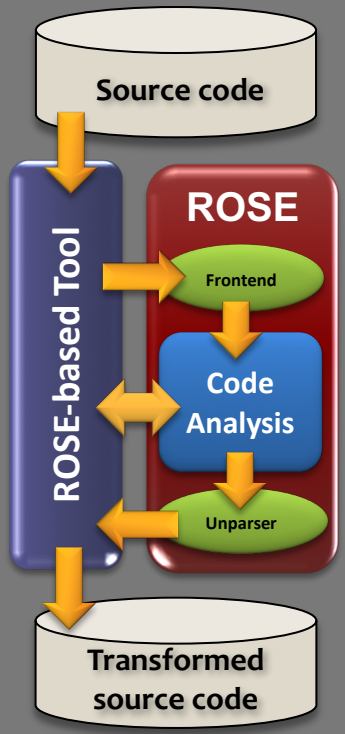
Such a paradigm shift could

- Increase scientific throughput up to 10x (*weeks to days*)
- Reduce data storage needs *by 50-99%*
- Increase the utilization of supercomputers

Need tools to support

- Developers
- Users

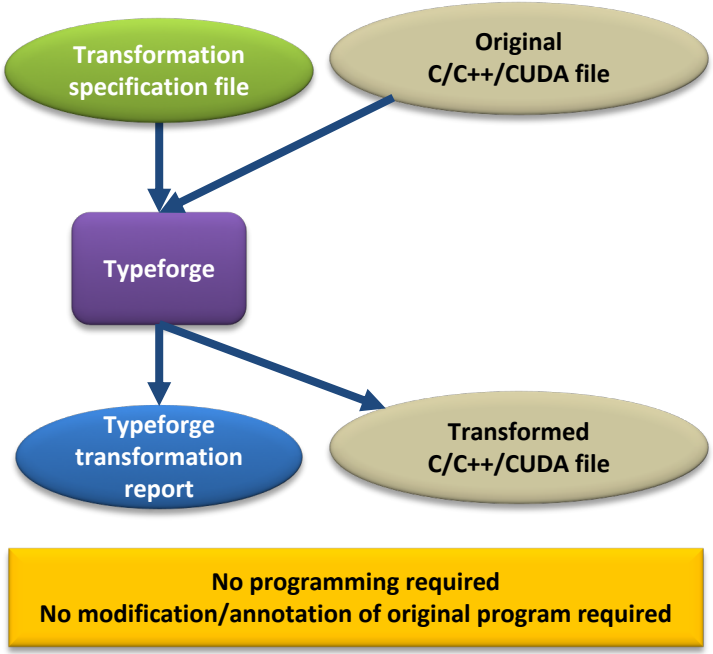
New technologies are not enough: Need tools that will help developers manage with complexity



- **Goal is to develop tools that will help users:**
  - Rapidly change type/implementations
  - Analyze code sections for precision sensitivity
  - Automate conversions
- **We are using the ROSE infrastructure to build new tools**
  - Software analysis and source-to-source transformation
  - Such tools will use software patches
    - Introduce generated transformations
    - Demonstrated on million-line C++ codes
- **We are making use of AD Tools for precision profiling**

Typeforge - Forging New Types into Existing Code

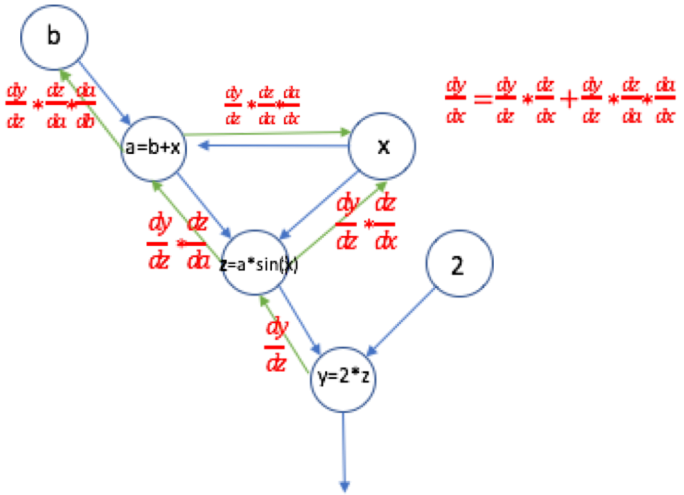
- ROSE tool for automatic replacement of types in programs
- Can replace floating-point types with lower-precision types or variable-precision types
- Supported languages: C/C++/CUDA
- Does not require programming
- Takes as input a specification file and the source file(s)
- Automatically minimizes casts between floating point types
- Example use cases:
  - ***Euler2D:*** Port implementation from C to C++ by replacing C types with C++ types and transform data structure access code
  - ***LULESH:*** Change certain variables from double to float to experiment with mixed precision (based on input from tool ADaPT)



We have developed an automated analysis technique to identify sensitivity to precision relative to a desired output accuracy

Reverse mode of algorithmic differentiation

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

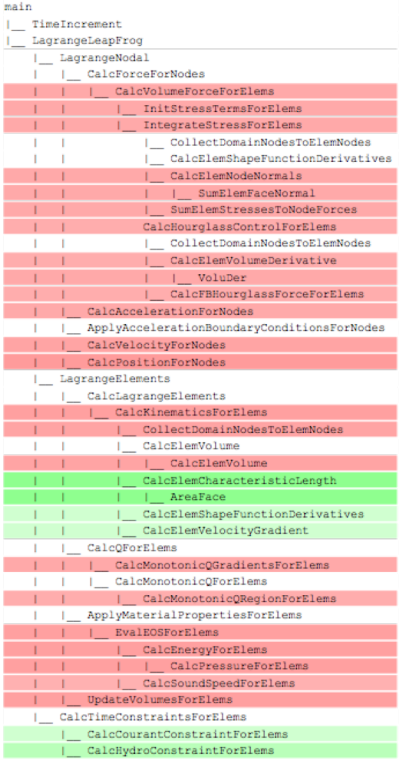


Reverse mode of AD is used to compute partial derivatives of all variables with respect to output in a single execution

ADAPT is a tool that implements this precision analysis

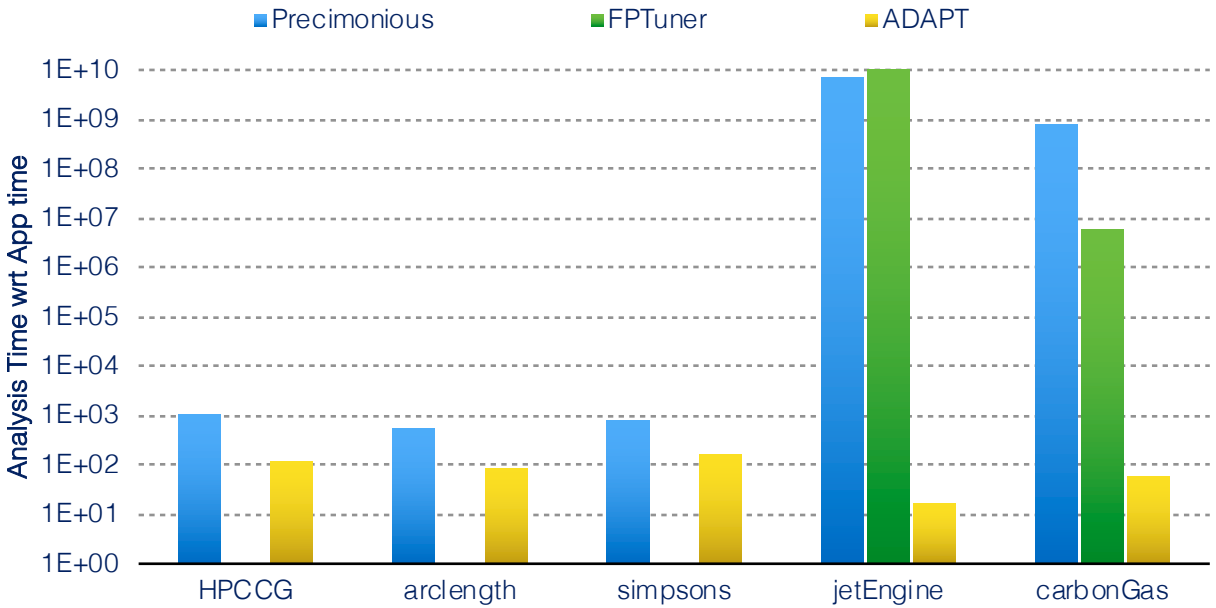
- **Estimate the error due to lowering the precision of every dynamic instance of a variable**
- **Aggregate the error over all dynamic instance of the variable**
- **Greedy approach**
  - Sort variables based on error contribution
  - Variables switched to lower precision - estimated error contribution within threshold

[Menon, Lam, Osei-Kuffuor, Schordan, Lloyd, Mohror, H., SC18]



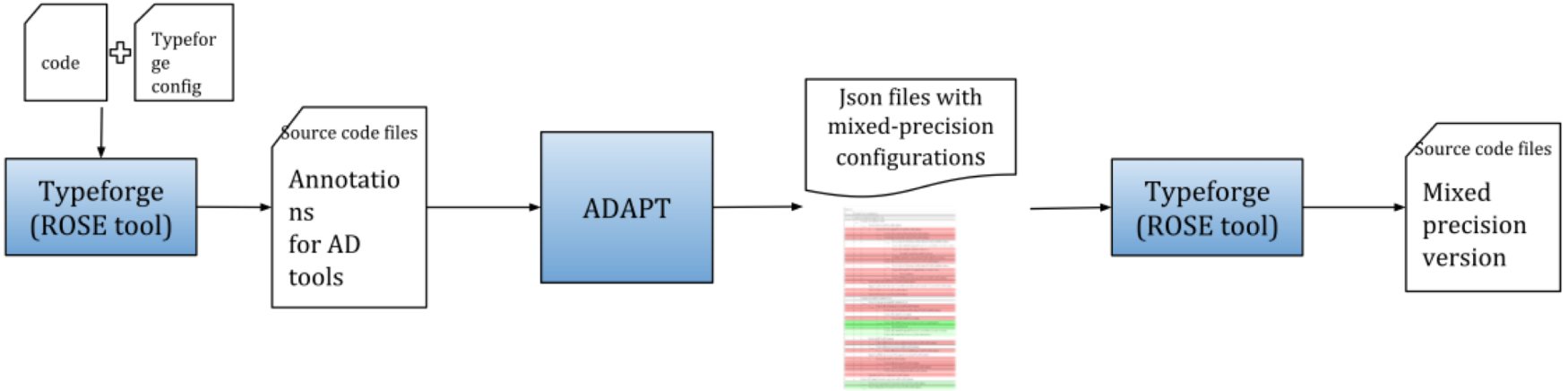
- **Used ADAPT on Lulesh to create mixed precision sensitivity profile**
- **Used profile as a guide to develop a mixed precision version for a CUDA implementation of Lulesh**
- **Achieved speedup of 1.2x**

ADAPT is significantly faster than search-based approaches



Analysis time wrt to the application time. ADAPT has the lowest analysis time

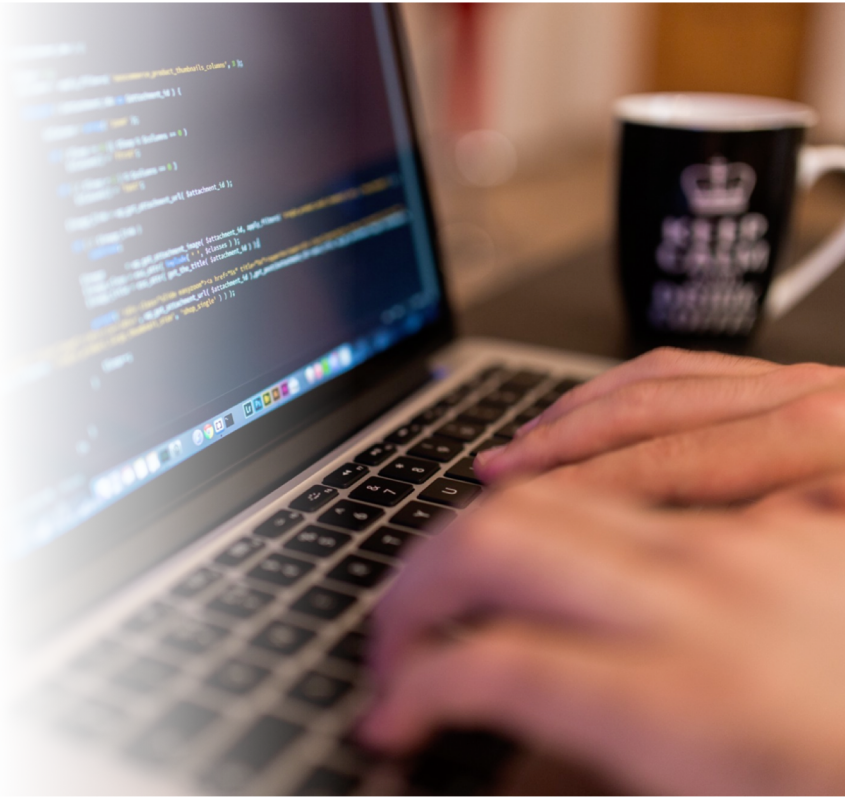
We have combined ADAPT and Typeforge for automated floating-point precision analysis and code transformation



Our research activities are producing software artifacts that we are releasing to community

- Typeforge
- ADAPT
- NumRep / FLEX
- Unum library
- C++ floating point statistics wrappers
- ZFP and Adaptive ZFP (ARC)

https://github.com/LLNL



CASC

Center for Applied
Scientific Computing

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.