Making Every Bit Count: Variable Precision?

Jeff Hittinger
Group Leader, Scientific Computing Group, CASC

3 Feb 2017

LLNL-PRES-722917
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

How to make an omlette inefficiently…

- Eggs come by the dozen

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

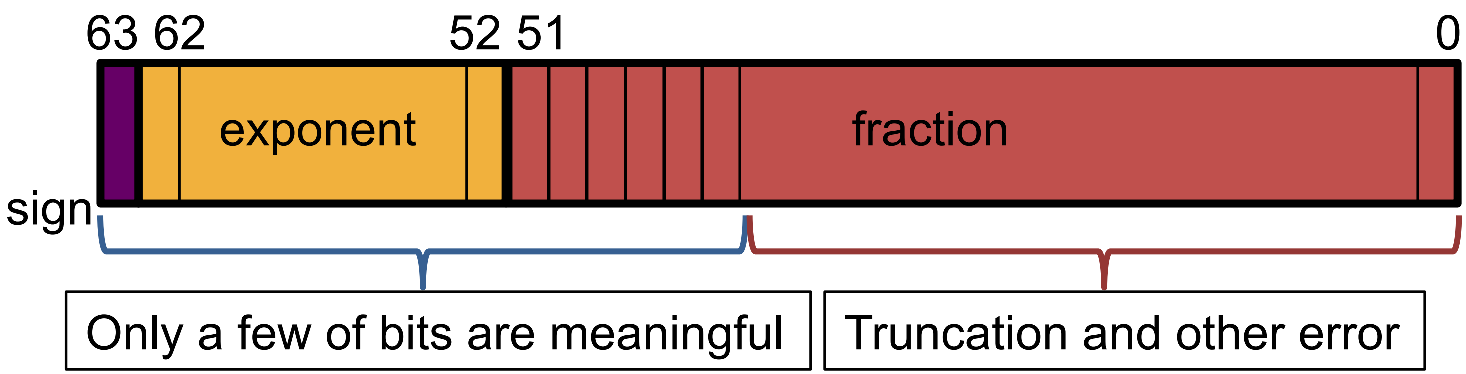
Exascale: like making a *billion billion* omlettes per second!

The cost of data motion is a critical issue on future computing architectures

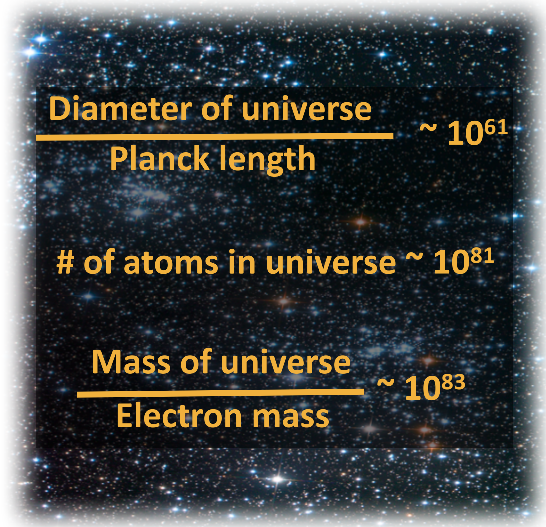
- On-node flops are increasing at least an order of magnitude faster than bandwidth
- Memory per core is decreasing
- Bandwidth-limited algorithms will not access the full potential of exascale

We could get a 10x or greater improvement in data motion and storage if we were more efficient representing data!

We use double precision floating-point by default (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
  - Get less performance



Eliminate the bottlenecks: use only as many bits as needed

We are developing tools and methods that we hope will enable adoption of variable precision

Lower precision

- Faster calculations
- Move/store less data
- Challenges
  - Harder to make robust
  - Harder to maintain
  - Insufficient at times

Variable precision

- Faster calculations
- Move/store less data
- Challenges
  - Harder to make robust
  - Harder to maintain
  - Adapts as needed

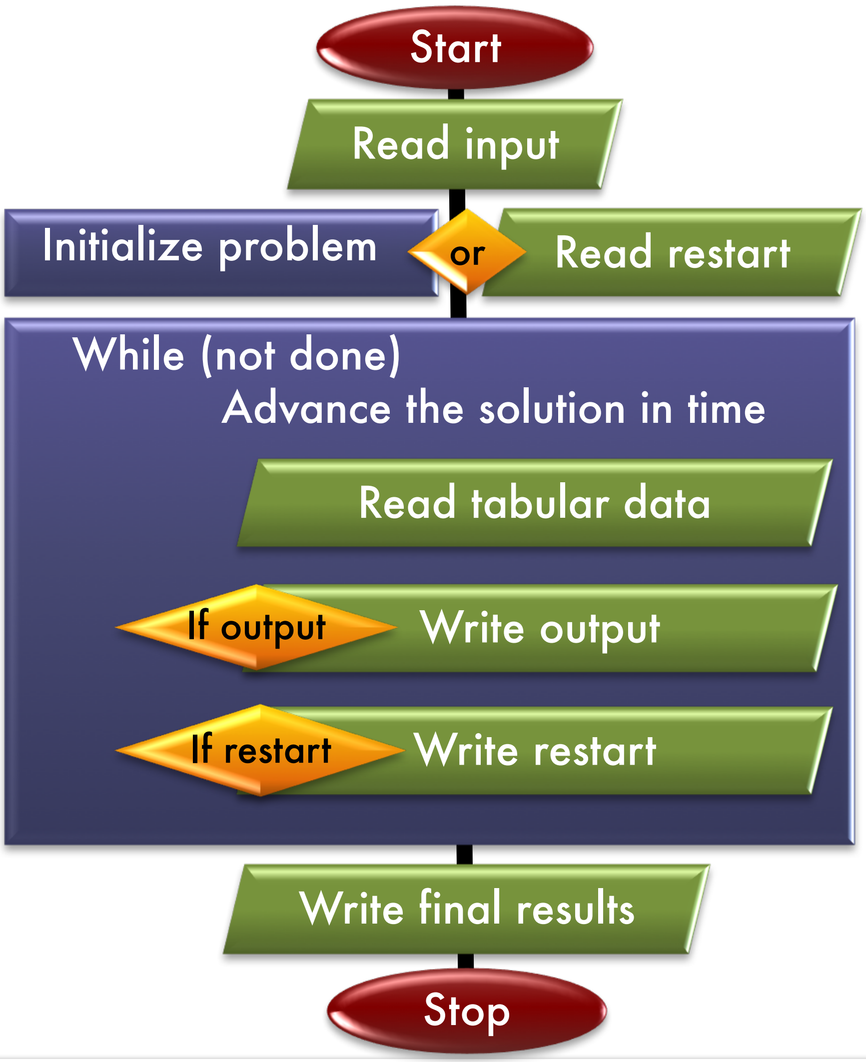
Automate

- When memory was scarce, we were clever in our use of single precision
- Until now, the cost/benefit has been too high

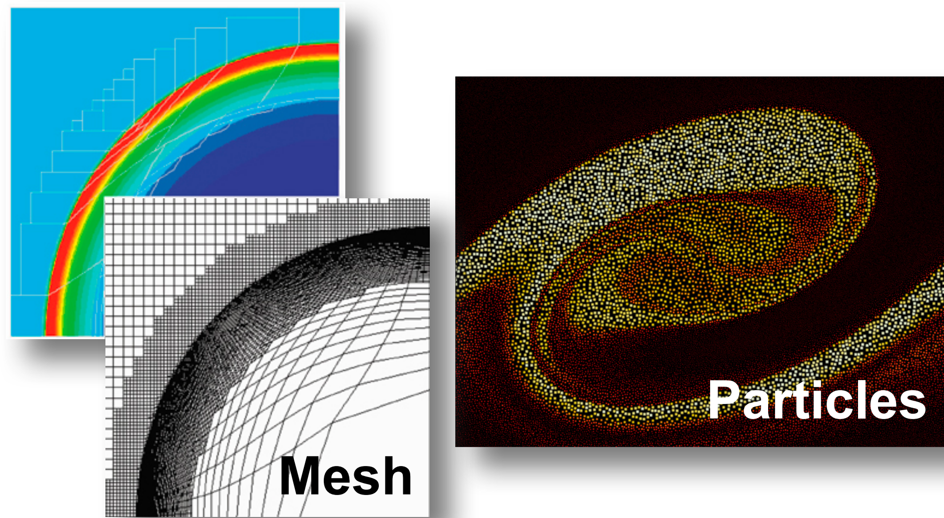
- Locally adapt the precision to the needs of the application
- Develop tools to make the cost/benefit favorable

We already adapt mesh size, order, models – *why not precision?*

What do we do when we “simulate”?



Discrete Representation

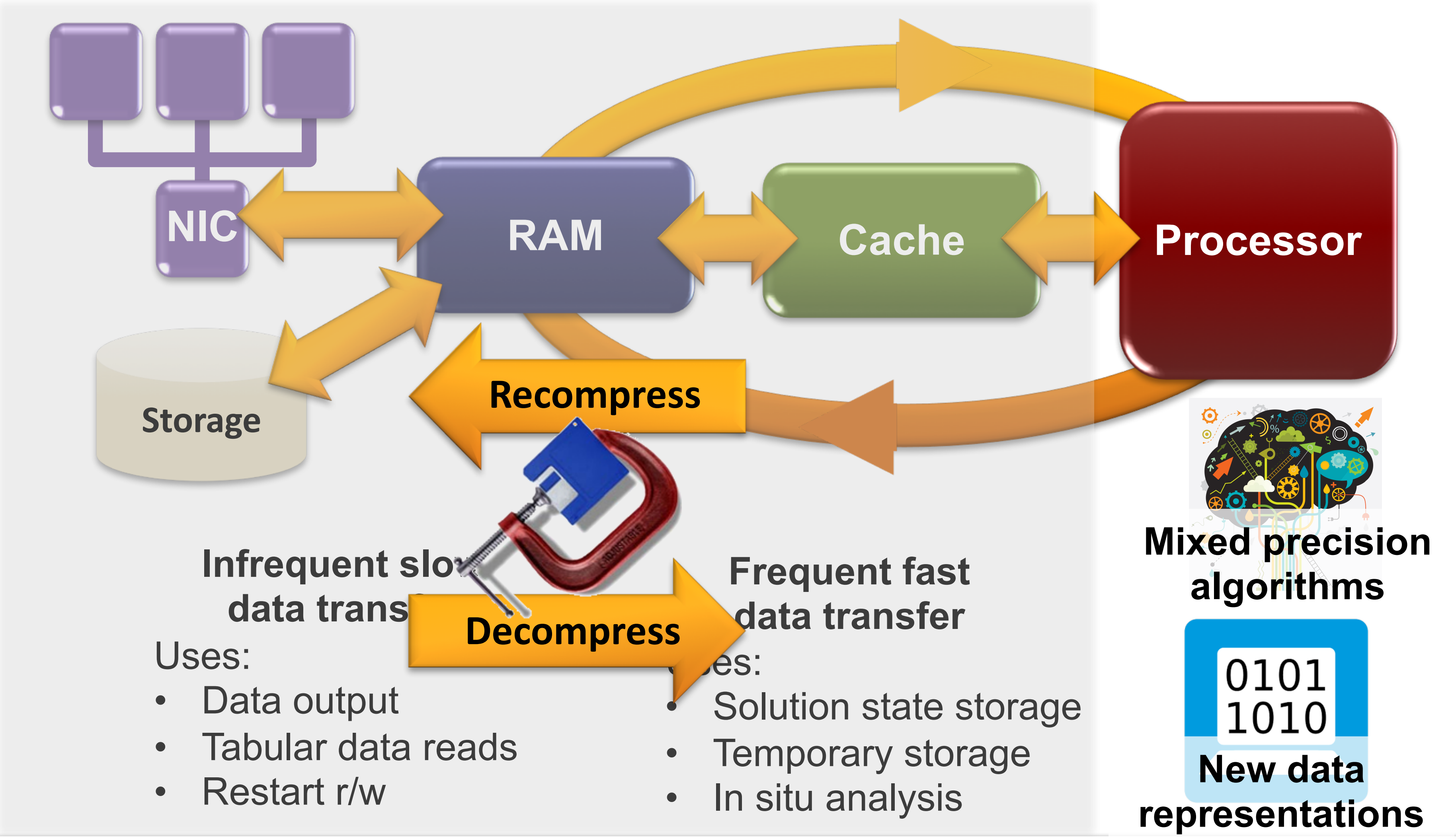


$$u^{n+1} = H(u^n)$$

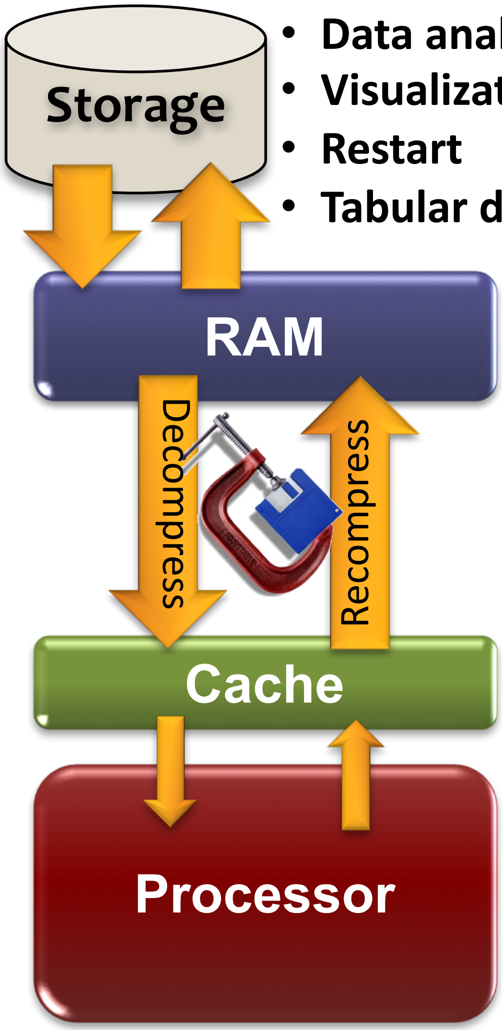
$$u \in \mathbb{R}^N$$

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

Where can we address precision issues?



Lossy compression can address the I/O bottleneck



- Data analysis
- Visualization
- Restart
- Tabular data

GOAL

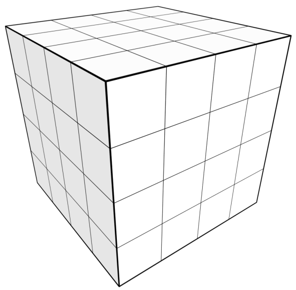
Develop algorithms & software supporting *adaptive precision* where *errors do not amplify*

APPROACH

- Adaptive Rate Compression (ARC)
  - Rate of compression differs between data components (by time, space, and/or variable)
- Multi-resolution data format (IDX) with ARC
- *Data optimal algorithms* for IDX+ARC
  - Compute desired solution to necessary precision with the minimal number of bits

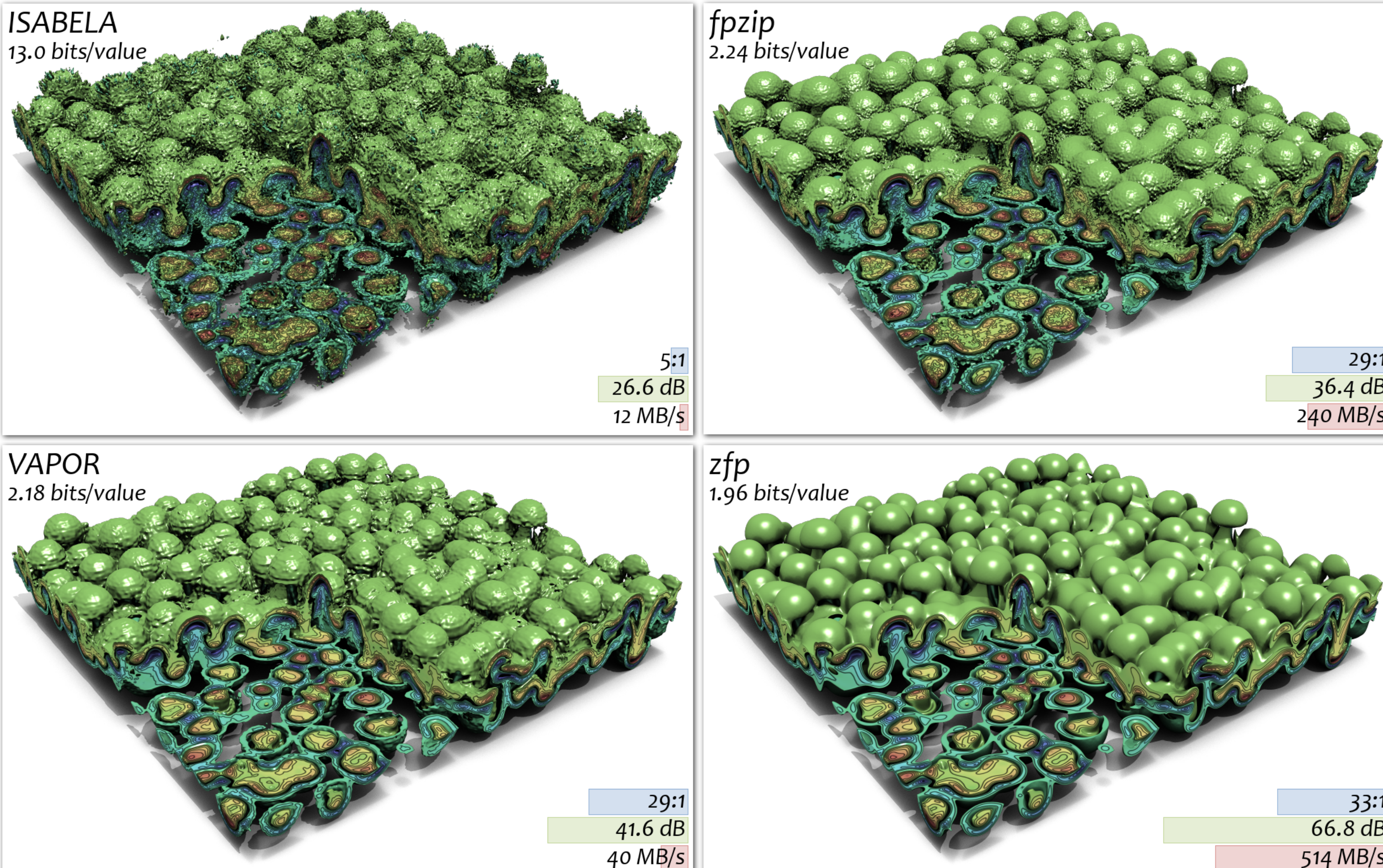
We have developed ZFP: the first inline compressor for floating-point arrays

- Inspired by ideas from h/w texture compression
  - 1D, 2D, or 3D array divided into fixed-size $4\times4\times4$ blocks
  - Each block is independently (de)compressed
    - e.g., to a user-specified number of bits or quality
  - Fixed-size blocks ⇒ **random read/write access**
  - (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**)

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

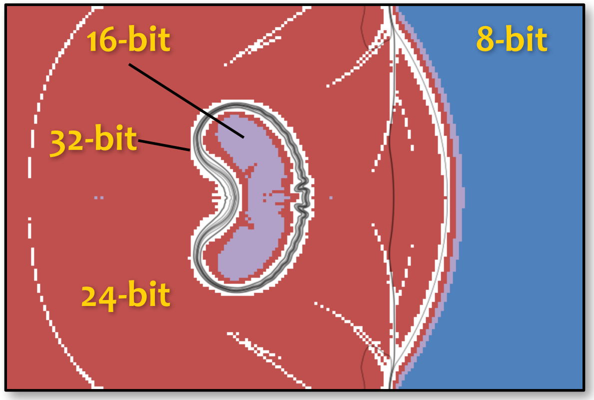


We will base our Adaptive Rate Compression (ARC) method on LLNL's ZFP compression algorithm

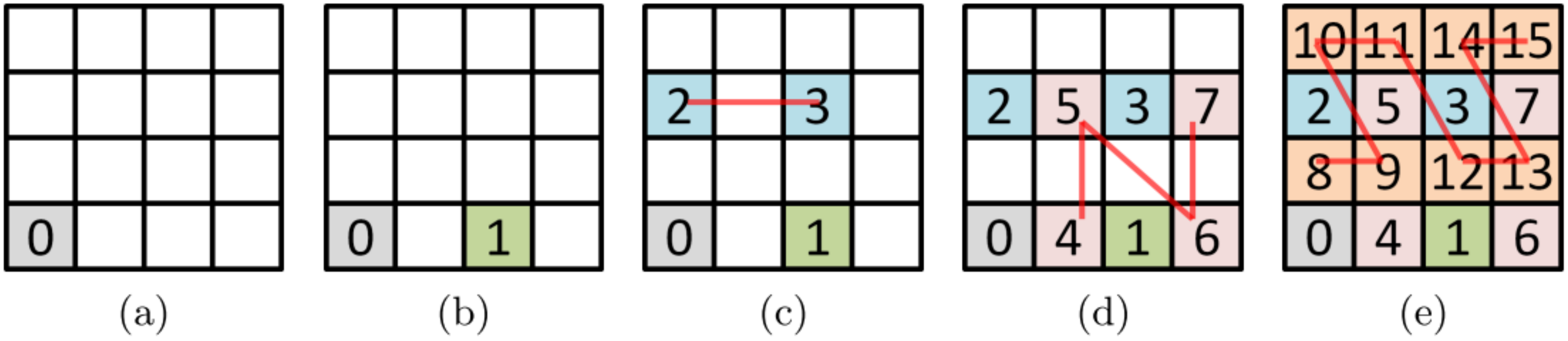
- **ZFP CODEC supports fixed-size storage or minimum quality**
  - **Fixed quality**: User tolerance ensures **relative or absolute error bound**
  - **Fixed rate**: Inline compressor supports **read and write random access**
    - Compressed array primitive with user-specified footprint

- **Very high quality and speed**
  - **100x more accurate** than closest competitor
  - **~40 bits of accuracy** for **16 bits of storage**
  - Up to **2 GB/s/core: 2-6x faster** than competition
  - Algorithm amenable to **h/w implementation**

- **Small, independent compressed blocks**
  - Enable adaptive precision, data parallelism
  - ***Potential for using different compression rate on each block***

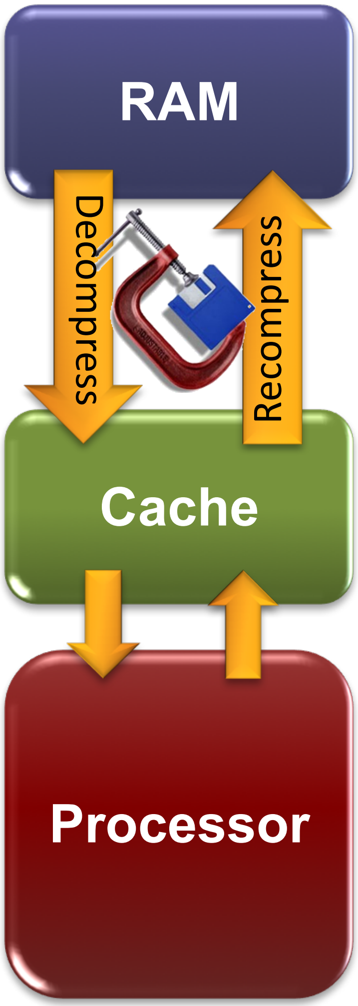


IDX is a hierarchical indexing scheme that supports progressive reads of sub-sampled data



- **Stores data belonging to the same "resolution" together**
  - Uses Z-like space filling curves to preserve spatial locality
  - No redundant data
- **Supports fast reads of low-resolution data (avoids disk seeks)**
- **Supports progressive data streaming**

Addressing the memory bandwidth limit while computing

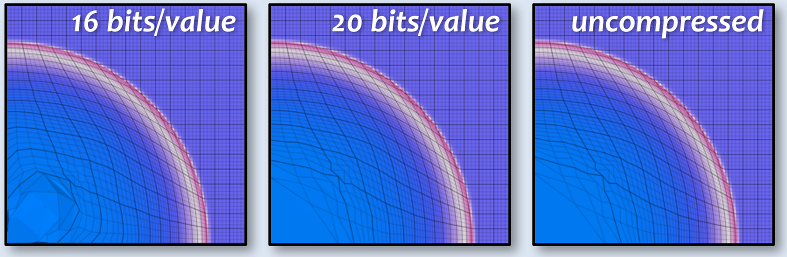


- **Store data in memory in compressed format**
- **Decompress before computing**
- **Recompress after computing**
- ***Ideally, the compression/decompression would be handled in hardware***

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

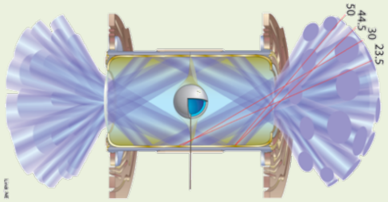
LULESH: Lagrangian shock hydrodynamics

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



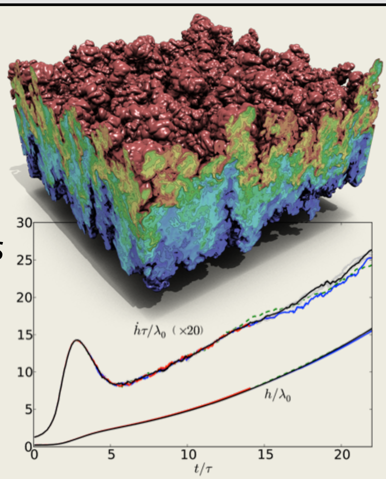
pf3D: Laser-plasma multi-physics

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



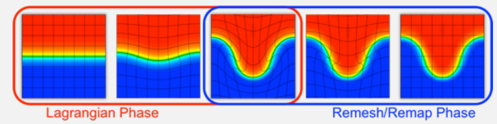
Miranda: High-order Eulerian hydrodynamics

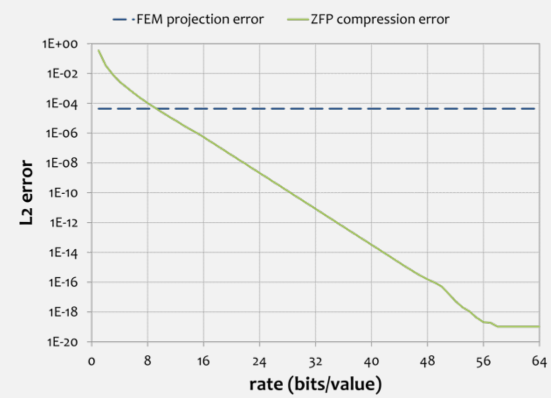
- QoI: *Rayleigh-Taylor mixing layer thickness*
- 10,000 time steps
- At **4x compression**, relative **error < 0.2%**



MFEM: Cubic finite elements

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





Inline compression introduces interesting mathematical questions

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

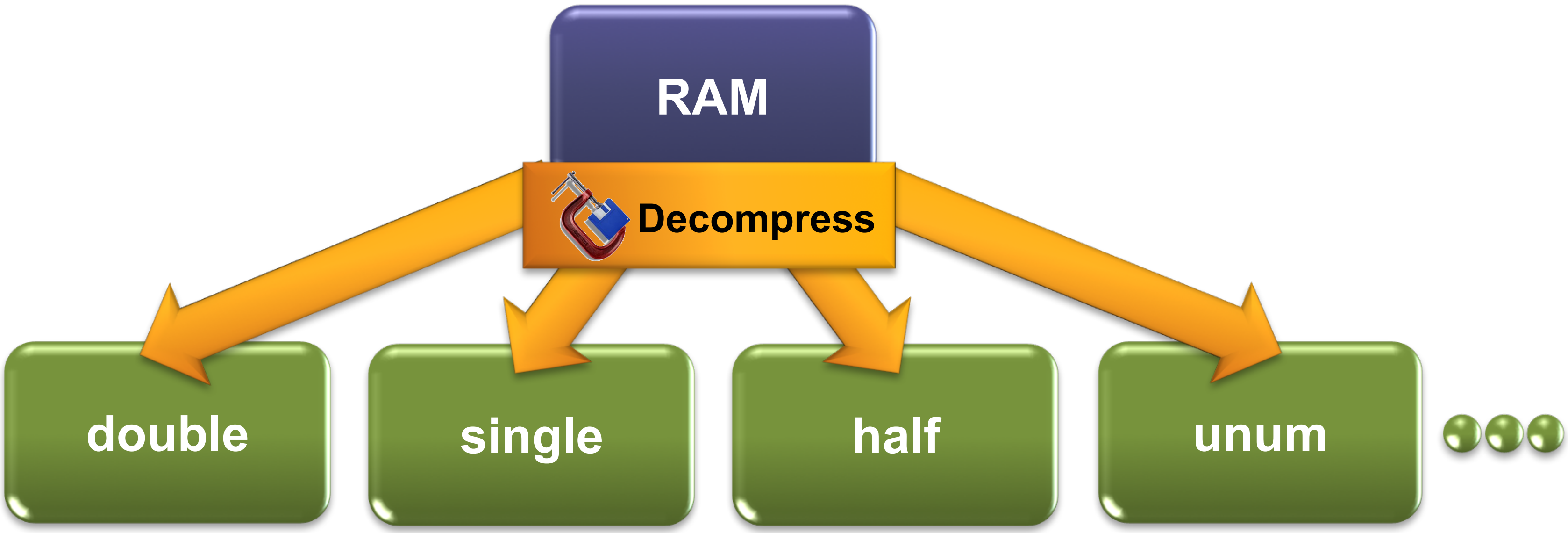
Lossy compression — **Decompression**

$$H' \equiv C \circ H \circ D$$

- What are the properties of the composite operator?
- Accuracy?
- Stability?

Data transfer is one thing, but can we also increase computational throughput?

- Compressed data transfer still wastes operations on unneeded bits
- Better performance in single precision (more than factor of 2X)
- Into what does one decompress?
- How does one compute in lower precision without loss of accuracy?

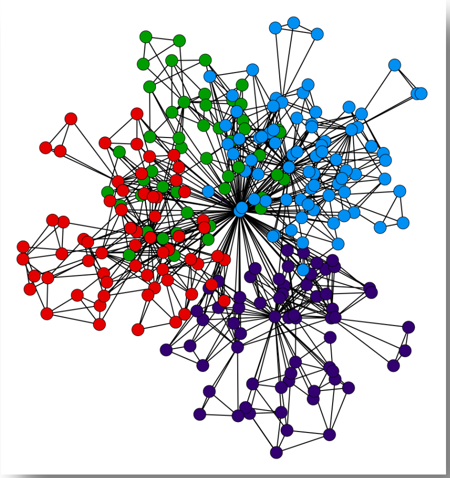


We are building on and going beyond existing work on varying precision

Single precision	Mixed precision	Arbitrary-precision	New formats
• 30 yrs ago: memory was limited • Expertise developed to use single precision	• Most current work (e.g., Buttari, Li, Demmel, Dongarra) • Static • Task-based • Great for libraries; more difficult for applications	• Focused on extending precision • Computing irrationals • Too slow for simulation • Can leverage some ideas	• Unums • Elias Gamma • Levenstein • Exponential • Tangent • Disruptive – changes fundamentals of floating point computing

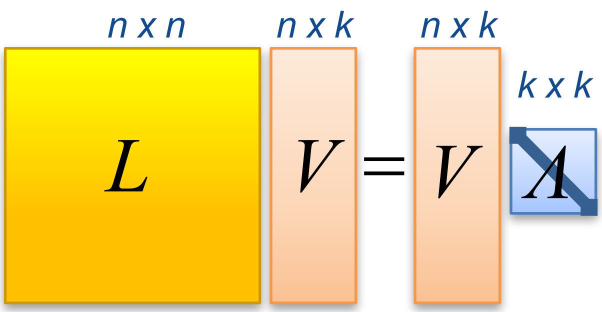
Graph Analysis can be done via Linear Algebra

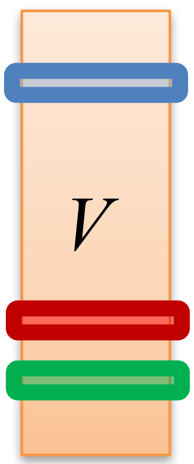
- Consider analyzing the connectivity structure of a social network
- Such relational datasets are modelled with graphs
- Several graph analysis tasks are useful tools for analysis:
  - **Ranking:** Who are the most important members?
  - **Clustering:** What members are more internally connected?
  - **Classification:** Which members are similar to a given set?
- Linear algebra kernels are useful to accomplish these goals
  - Linear solvers, Eigensolvers, SVD, NMF, Tensor factorization, etc
- Low-fidelity approximations are often as useful as high-fidelity
- We are investigating the efficacy of low-precision approximations



Spectral embedding of vertices can be used by a variety of analysis algorithms

Let $L \in \mathbb{R}^{n \times n}$ be the **combinatorial Laplacian**



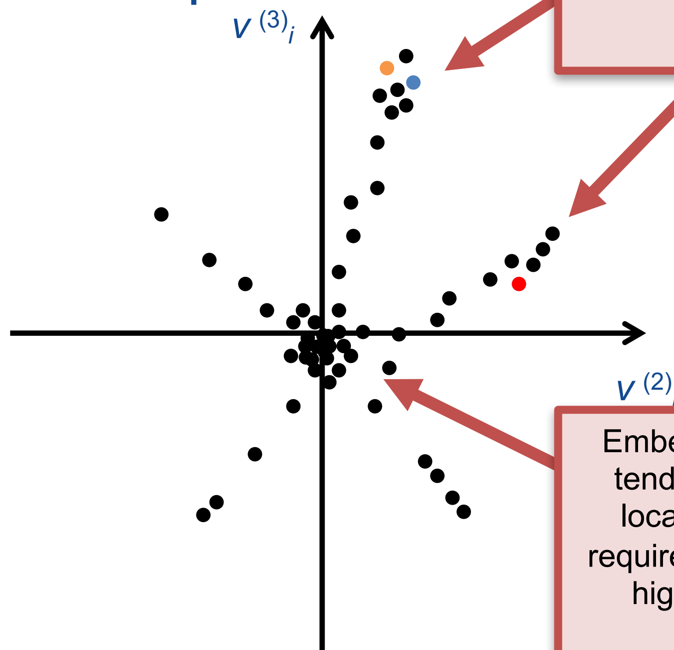


$V^t \mathbf{e}_i$

Spectral coordinate of vertex *i* is *i*-th row of *V*, a *k*-dimensional vector.

Eigenvectors provide a mapping from the vertices into R^k.

Spectral Coordinates



The partitioning algorithm uses the locations in this embedding to make decisions.

Several clusters are easily separable, even in low-fidelity approximations.

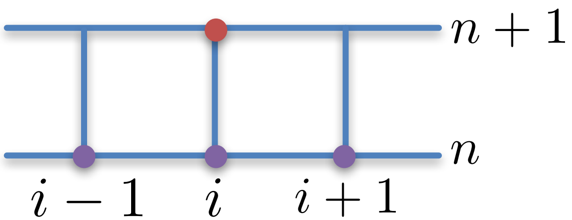
Embeddings of real-world graphs tend to have many centrally co-located points and recursion is required to further partition, even in high-fidelity, double precision approximations

Relevant for spatial clustering algorithms and ML algorithms for classification tasks

We are also investigating other mixed precision techniques

- Error transport
  - Developed as a posteriori error estimator for truncation error
  - Form of defect correction
  - Can also be used to estimate roundoff error

$$u_t = u_{xx} \qquad u_i^n = v_i^n + \epsilon_i^n$$



$$\left[\frac{v_i^{n+1} - v_i^n}{\Delta t} - \frac{v_{i+1}^n - 2v_i^n + v_{i-1}^n}{\Delta x^2}\right]_{\text{single}} = 0$$

Primal

$$\left[\frac{\epsilon_i^{n+1} - \epsilon_i^n}{\Delta t} - \frac{\epsilon_{i+1}^n - 2\epsilon_i^n + \epsilon_{i-1}^n}{\Delta x^2}\right]_{\text{single}} = -\operatorname{Res}\left(v^n, v^{n+1}\right)$$

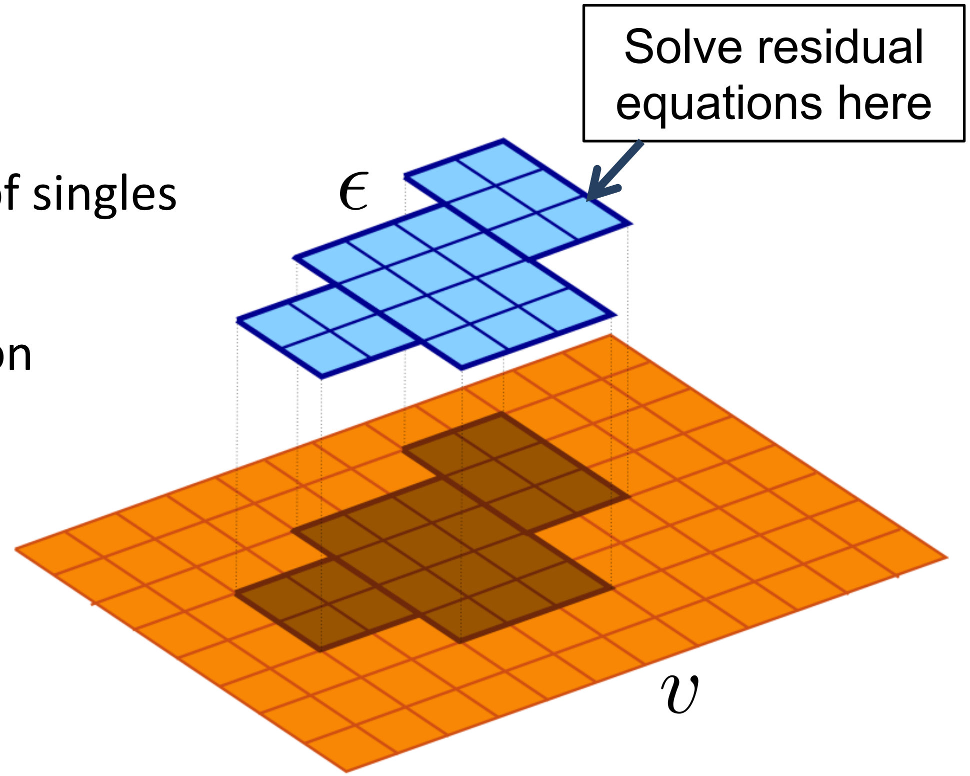
Error

$$\operatorname{Res}\left(v^n, v^{n+1}\right) = \left[\frac{v_i^{n+1} - v_i^n}{\Delta t} - \frac{v_{i+1}^n - 2v_i^n + v_{i-1}^n}{\Delta x^2}\right]_{\text{double}}$$

But this is *more* expensive!

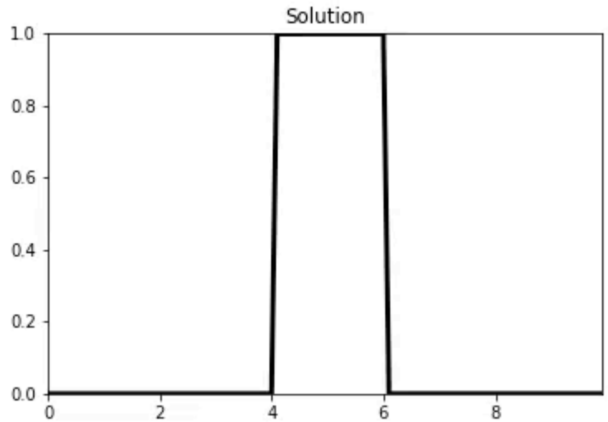
We are investigating the potential for an AMR-like dynamic, local mixed precision

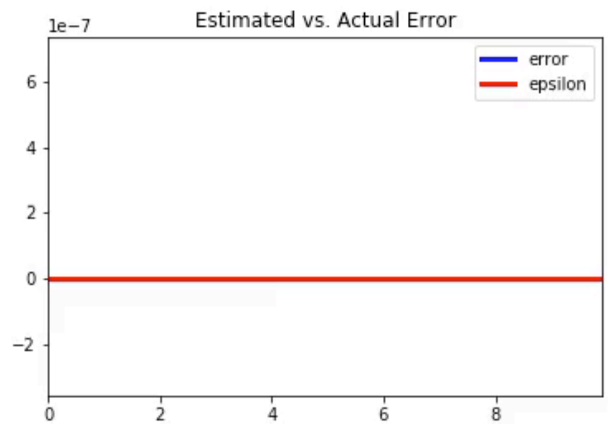
- **Dynamic Mixed Precision**
  - Hierarchical representation: sum of singles
  - Block-based refinement
  - Most calculations in single precision
  - Key issues
    - Refinement criteria
    - Propagation of round-off error
    - Cost/benefit
- **Could be done recursively**

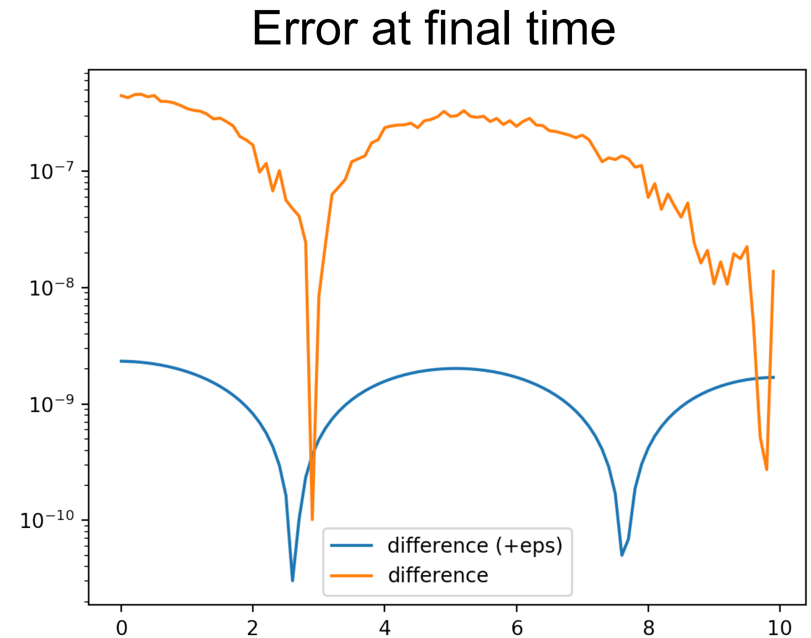


$$\underbrace{u}_{\texttt{double}} = \underbrace{v}_{\texttt{single}} + \underbrace{\epsilon^{(0)}}_{\texttt{single}} + \underbrace{\epsilon^{(1)}}_{\texttt{single}} + \cdots$$

We have promising (and puzzling) preliminary results







- **Correction gains factor of 100**
- **Why so little?**

Why limit ourselves to existing types?

Piecewise linear systems

- IEEE (**half**, **float**): $y = (-1)^s\, 2^e\, (1 + f)$
  - $e + bias$ written in binary using m bits
  - e bits of f encode int, rest encode frac
- Elias **gam**ma: for $y \geq 1$
  - $|e|$ written in unary using $e + 1$ bits
  - Use sign, reciprocal bit when $y < 1$
- **Lev**enstein (modified for Z_+)
  - Encode e recursively: $e = 2^{e'}\,(1 + f')$
  - As with IEEE/gamma, append fraction

Corresponding "smooth" systems

- **Exp**onential: $y = (-1)^s\, (2^b)^t = (-1)^s\, 2^{b\,t}$
  - 2^b = IEEE bias, e.g. b = 128 for floats
  - $t = 2|x| - 1$
- **Tan**gent: $\tan(\pi/2\, x)$
  - $\tan(\pi/2\,(1 - x)) = \cot(\pi/2\, x)$
- **Sup**erexponential: $\lg f(x) = f(2\,x - 1)$
  - log, exp, mul, div trivial if we have addition, subtraction

And a few more…

- **Bin**ary exponent
  - Identical to IEEE, but uses reciprocal bit instead of negative exponent when $0 < |y| < 1$

- Gustafson's **type-ii** unums (w/o u-bit)
  - Uses sign bit, reciprocal bit
  - Starts with seed set of "Kindergarten numbers" {0.1, 0.2, …, 1, 2, …, 10}
  - Expands via reciprocal and multiplicative closure (multiply, divide by 10)
  - Reduces wobbling precision, but still not very smooth
  - NOTE: requires lookup table, binary search to encode/decode

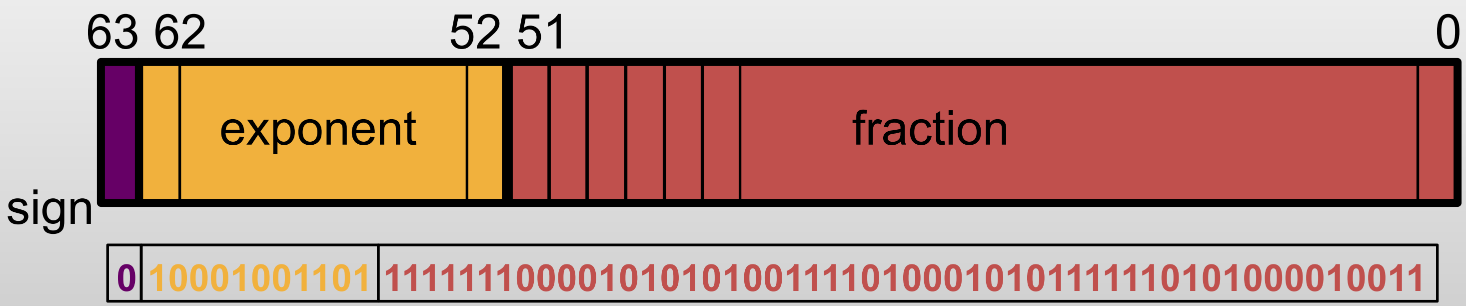
- Gustafson's **type-iii** unums (w/o u-bit)
  - Identical to Elias gamma (when *useed* = 2), but without reciprocal bit (piecewise linear everywhere)

- "**Hyp**erbolic" numbers [hyp]
  - $f(x) = x / (1 - |x|) = \operatorname{sign}(x) \exp(2 \operatorname{arctanh}(2 |x| - 1))$
  - Smooth map
  - Cheap conversion

- **zfp**: fixed-rate compressed arrays
  - Fixed-length compressed bit string amortized over blocks of 4 x 4 values

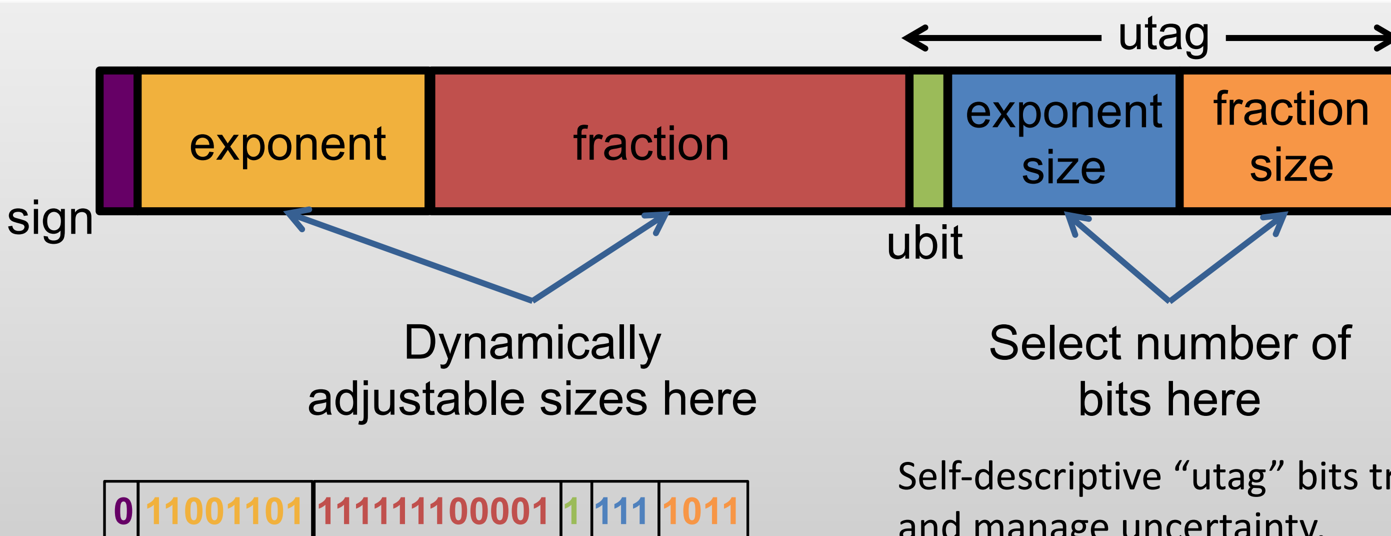
What is a Unum? Let's represent 6.022×10^{23}

IEEE 754 double



64 bits

Unum

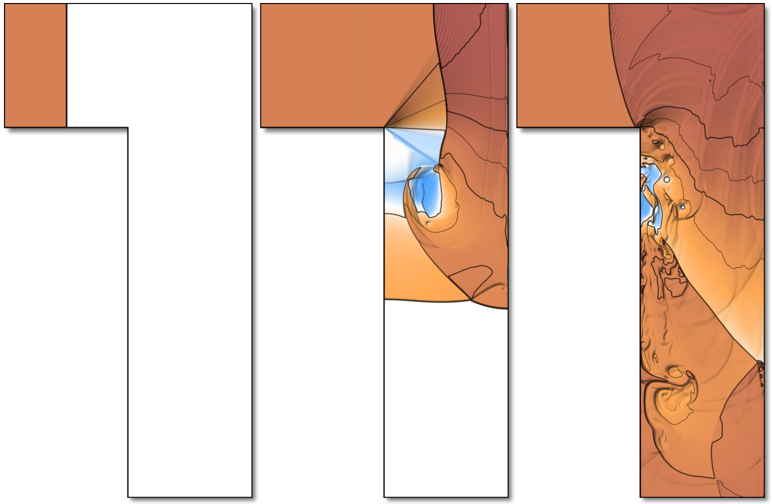


29 bits

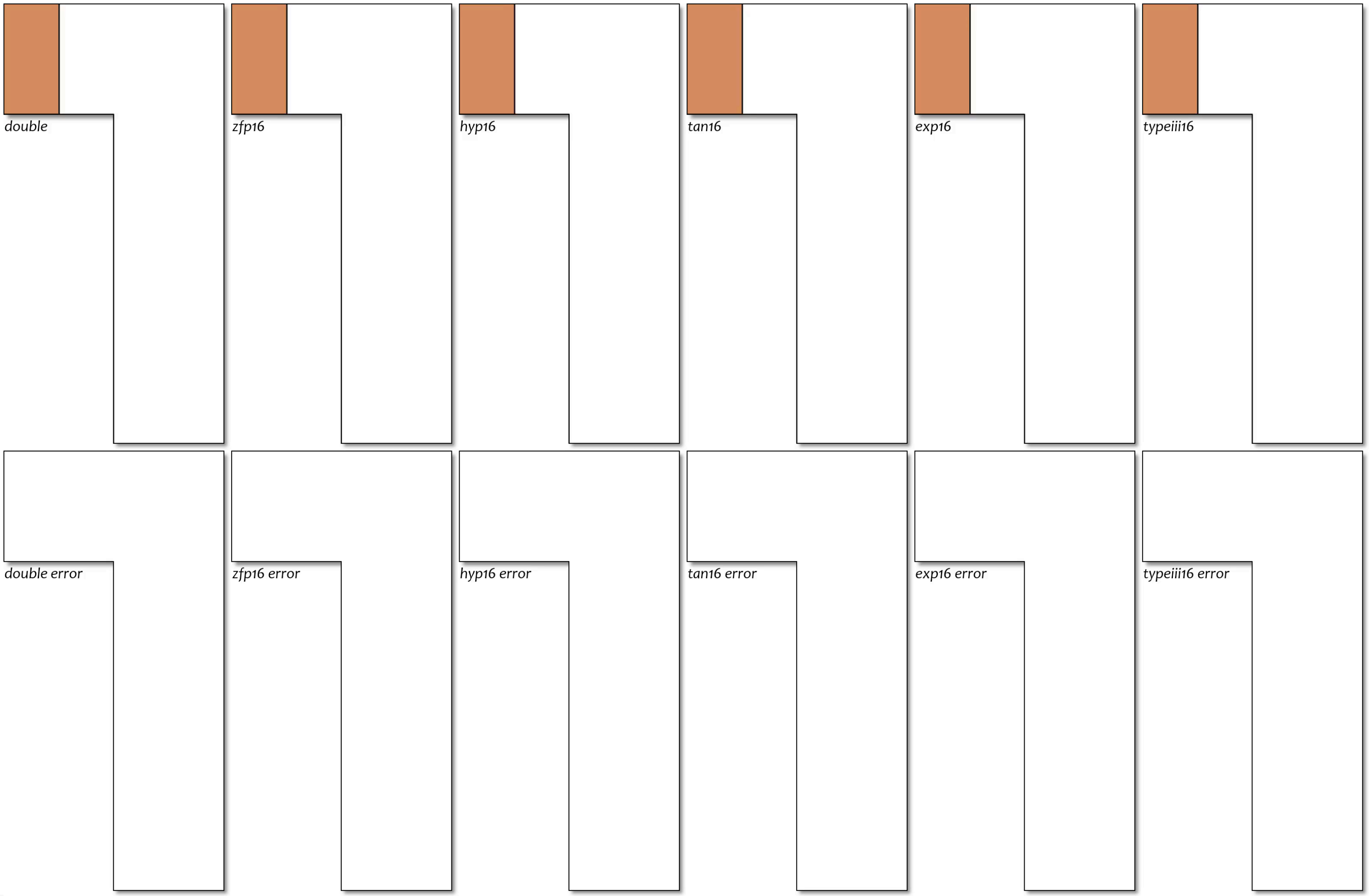
Self-descriptive "utag" bits track and manage uncertainty, exponent size, and fraction size

We can consider an accuracy evaluation using the nonlinear hyperbolic Euler PDEs

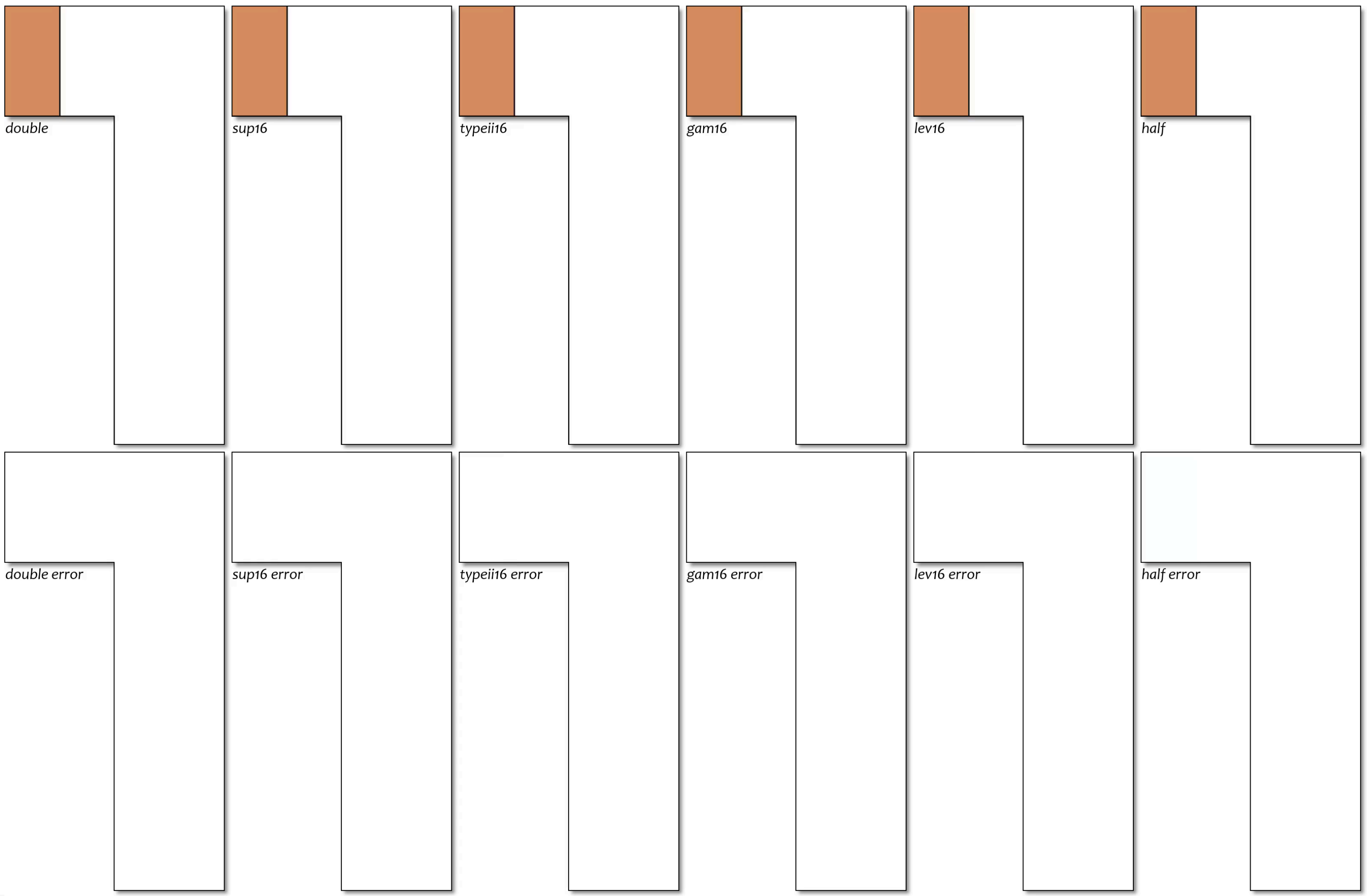
- Shock passing through L-shaped chamber
- Uniform grid: 512x256 + 256x768 cells
- All arithmetic done in IEEE double precision
- All data stored as 16- or 32-bit precision
  - 640 kB or 1.25 MB per array
  - vs. 2.5 MB per array using IEEE double
- zfp run with and without a cache
  - zfp16: 16 bits/value, single-block “cache”
  - zfp16c: 16 bits/value, 64 kB cache per array



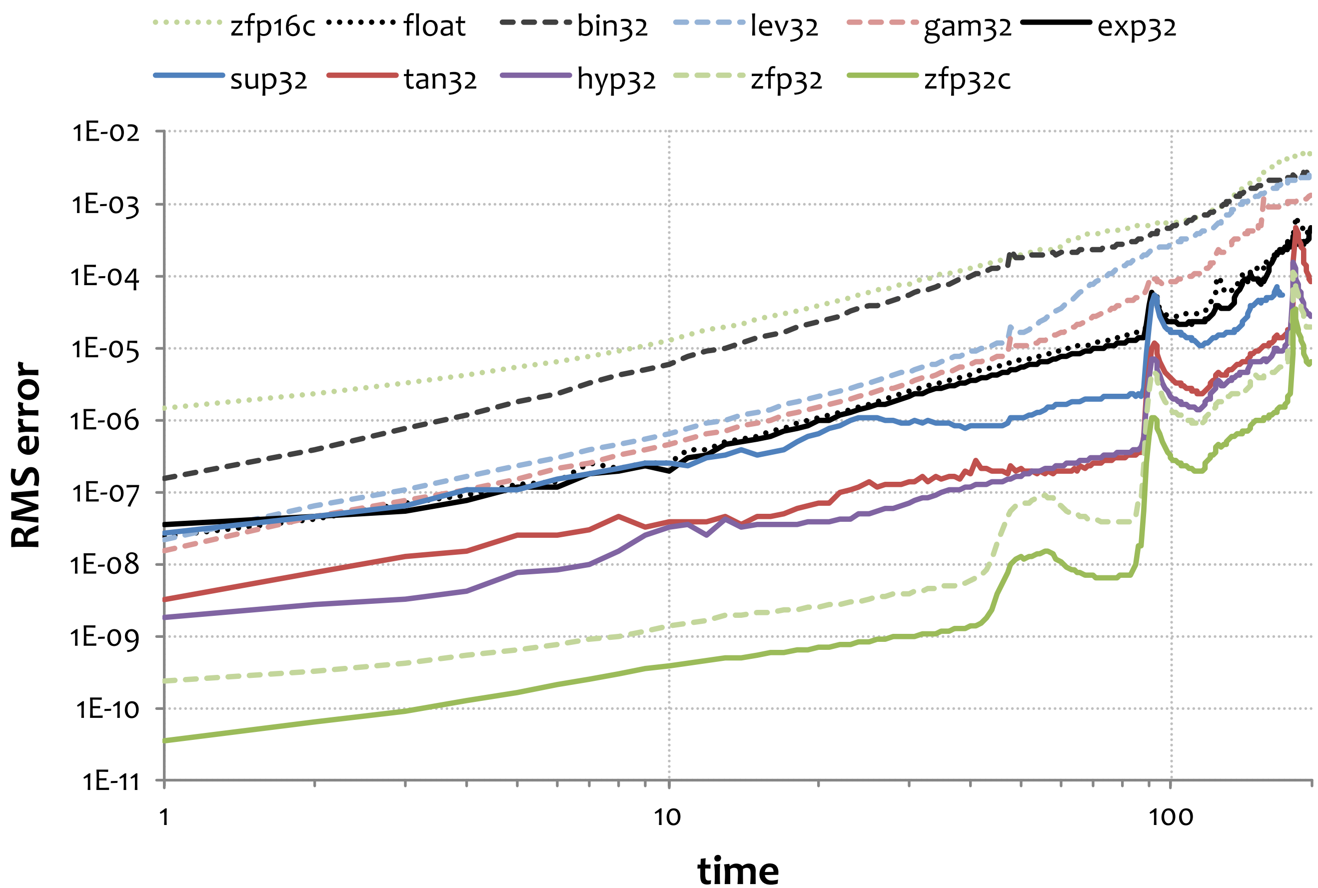
Some representations are promising



Some representations are abysmal

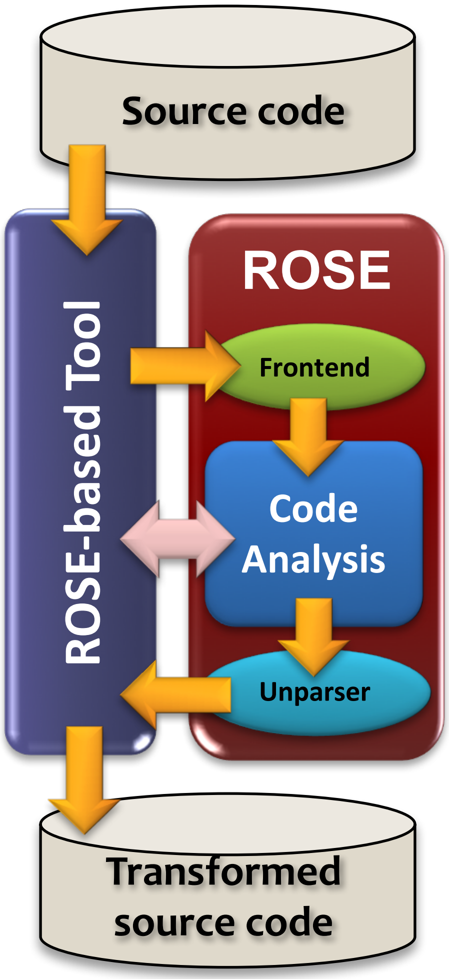


RMS error vs. time 32-bit precision



- IEEE is consistently among the worst numerical representations!
- zfp is 1-3 orders more accurate than uncompressed representations

Important products of our project are tools that will help developers deal with complexity



- **Goal is to develop tools that will help users:**
  - Rapidly change type/implementations
  - Analyze code sections for precision sensitivity
  - Automate conversions
- **We will use the ROSE infrastructure to build new tools**
  - Software analysis and source-to-source transformation
- **Variable precision tools will use software patches**
  - Introduce generated transformations
  - Demonstrated on million-line C++ ASC apps for OpenMP optimizations

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

But such a paradigm shift could

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

Credits

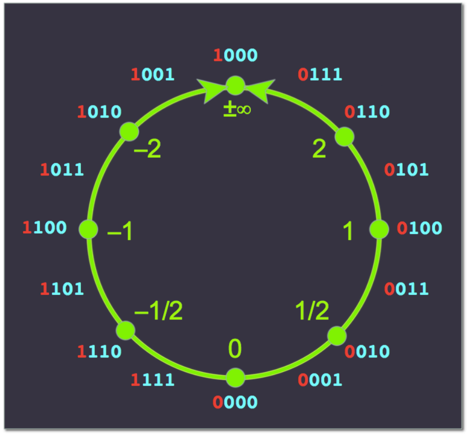
Daniel Osei-Kuffuor	Mixed precision, solvers, MD applications
David Beckingsale	AMR, performance analysis
Geoff. Sanders	Solvers, complex networks apps
Peter Lindstrom (Co-PI)	Data compression, multi-resolution methods
Timo Bremer	Multi-resolution methods
Daniel Quinlan (Co-PI)	Compiler tools
Markus Schordan	Program analysis
Scott Lloyd	Unums, reconfigurable computing

This work was funded by LLNL Laboratory Directed Research and Development as Project 17-SI-004: *Variable Precision Computing*

Lawrence Livermore National Laboratory

Axioms for a closed number system

- Consider the monotonic mapping $f : (-1, +1) \to R$
  - x in $(-1, +1)$ is the binary representation of the real value $y = f(x)$
  - In practice, we uniformly sample the interval $(-1, +1)$ at 2^p points
- Monotonicity, closure under negation & reciprocation impose these constraints
  - $-f(x) = f(-x)$ (two's complement avoids negative zero)
  - $1/f(x) = f(1-x) \quad 0 < x < 1$
- Hence
  - $f(0) = 0$
  - $f(\pm 1/2) = \pm 1$
  - $f(+1) = f(-1) = \pm \infty$ (the point at infinity)
- We are free to map $f : (1/2, 1) \to (1, \infty)$ as we like
  - $(-1, 1/2)$ map given by negation, reciprocation



We will investigate multiple techniques for varying precision to address the bottlenecks

Task-based mixed precision

Multilevel representation

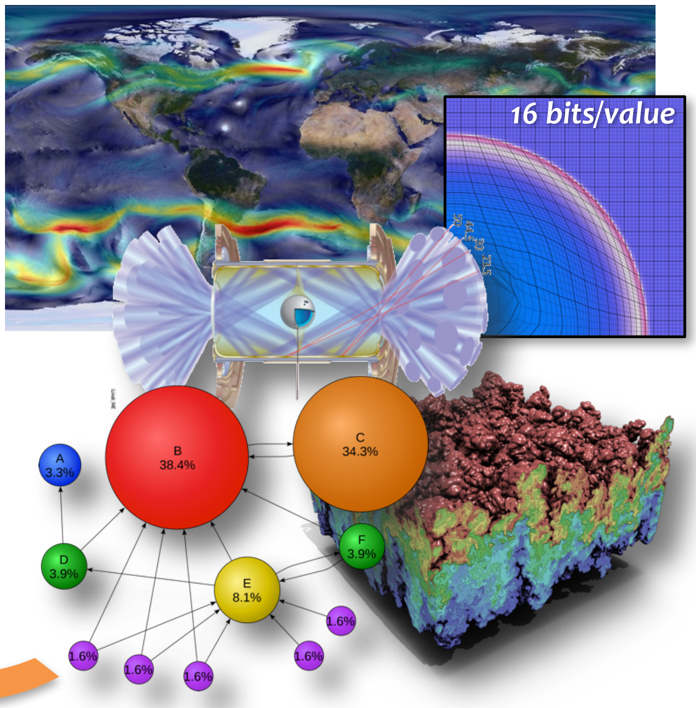
Libraries and Tools

Adaptive-rate compression

Unum computing

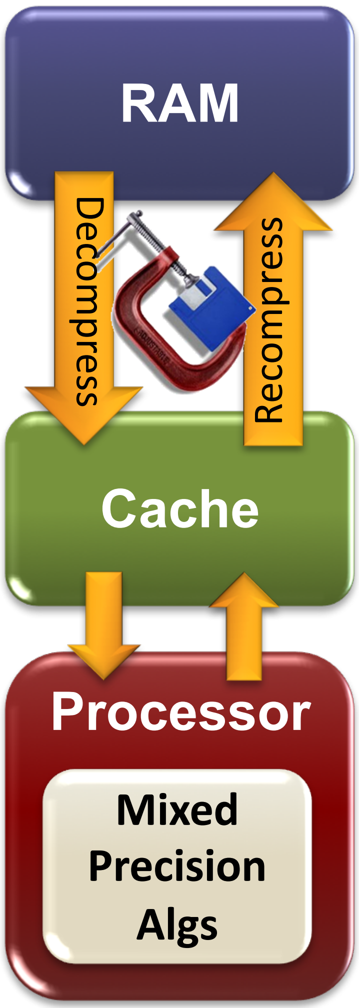
ARM / Commodity GPU cluster

Applications



Success requires close collaboration with applications

Thrust 2: We will develop Variable Precision Algorithms for dynamic data



GOAL

Using *standard data types*, develop algorithms and software to support *adaptive precision* on data where *errors can amplify*

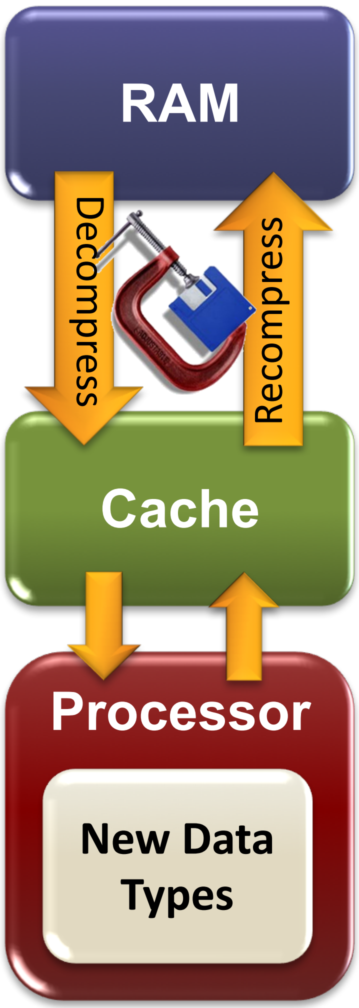
APPROACH

- Extend static mixed precision algorithms
- Develop dynamic mixed precision through layered representation (like AMR)
- Apply Adaptive Rate Compression (ARC) inline

ISSUES

- Precision refinement criteria
- Stability of new algorithms
- Behavior (propagation) of roundoff errors
- Non-contiguous data layouts

Thrust 3: We will investigate new data representations for variable precision computing



GOAL

Using *new data types*, develop algorithms and software to support *adaptive precision* on data where *errors can amplify*

APPROACH

- Universal numbers (unums)
- ZFP as a new number format
- New floating-point compression algorithm suitable for unums

ISSUES

- Utility of unums in numerical algorithms
- Stability/accuracy/convergence of new algorithms
- Ability to transform operations w/ compression
- Prospects for hardware implementations

We will continue the work of a feasibility study to evaluate the potential benefits of unums

- Better answers with fewer bits
- No rounding errors
- Less memory usage without loss of information
- Bit-identical results across systems
- New algorithms leveraging variable precision

Unum FS Status

- ✓ **Developed first C unum implementation**
  - Built on GNU multi-precision library
  - Includes C++ API
- ✓ **Unum implementation demonstrated in parts of LULESH**
- ✓ **Developed compiler support to identify and transform types within application code**
- ○ **Ongoing work:**
  - Additional operators added as needed
  - Trasnslate more of LULESH to unums
  - Extend compiler work