HLRS

High-Performance
Computing Center
Stuttgart

Evaluating a Real-Time Lossy Array Compression Algorithm for a Lattice Boltzmann Solver

Darjan Krijan, Julian Vorspohl (AIA)

Me

Darjan Krijan M.Sc.

darjan.krijan@hlrs.de

- Mechanical Engineering background, focused on
  - Thermal Turbomachinery @ ITSM/Uni Stuttgart
  - Sealing Technology @ IMA/Uni Stuttgart
- At HLRS since 10/2019
  - In project SiVeGCS [2017-2032] funded by
    - BMFTR
    - MWK BW

- Optimizing node-level performance of HPC applications, notably
  - NS3Dneo IAG/Uni Stuttgart
  - m-AIA AIA/RWTH AAchen
- Started this PhD topic 04/2024

Overview

H L R S

- Introduction: Memory Bandwidth Problems
- State of Technology – Roofline Model
- State of Technology – Remedies
- New Approach – Proof of Concept & First Impressions
- Outlook

Introduction: Memory Bandwidth Problems

Moore’s Law

Moore’s Law

HLRS

Moore’s Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Our World in Data

Transistor count

Year in which the microchip was first introduced

Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

OurWorldinData.org – Research and data to make progress against the world’s largest problems.

Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

- Described by Gordon Moore, Intel founder (January 3, 1929 – March 24, 2023)
- Doubling complexity every ~2 years
  - ~relates to FLOP performance
- Does this apply to memory bandwidth as well?

FLOPS/Memory Bandwidth Progression

HLRS

- Memory Bandwidth falls behind by a factor of ~5.1x/decade since the 1990s
- Processors being starved out of data, limiting sustained performance e.g. ~~peak 1 ExaFLOPS~~ → real ~10s of PetaFLOPS
- Large fraction of HPC codes are memory bandwidth bound today due to this imbalance

Overview

H L R S

- Introduction: Memory Bandwidth Problems
- State of Technology – Roofline Model
- State of Technology – Remedies
- New Approach – Proof of Concept
- Outlook

State of Technology – Roofline Model

Memory Bandwidth vs. FLOPS

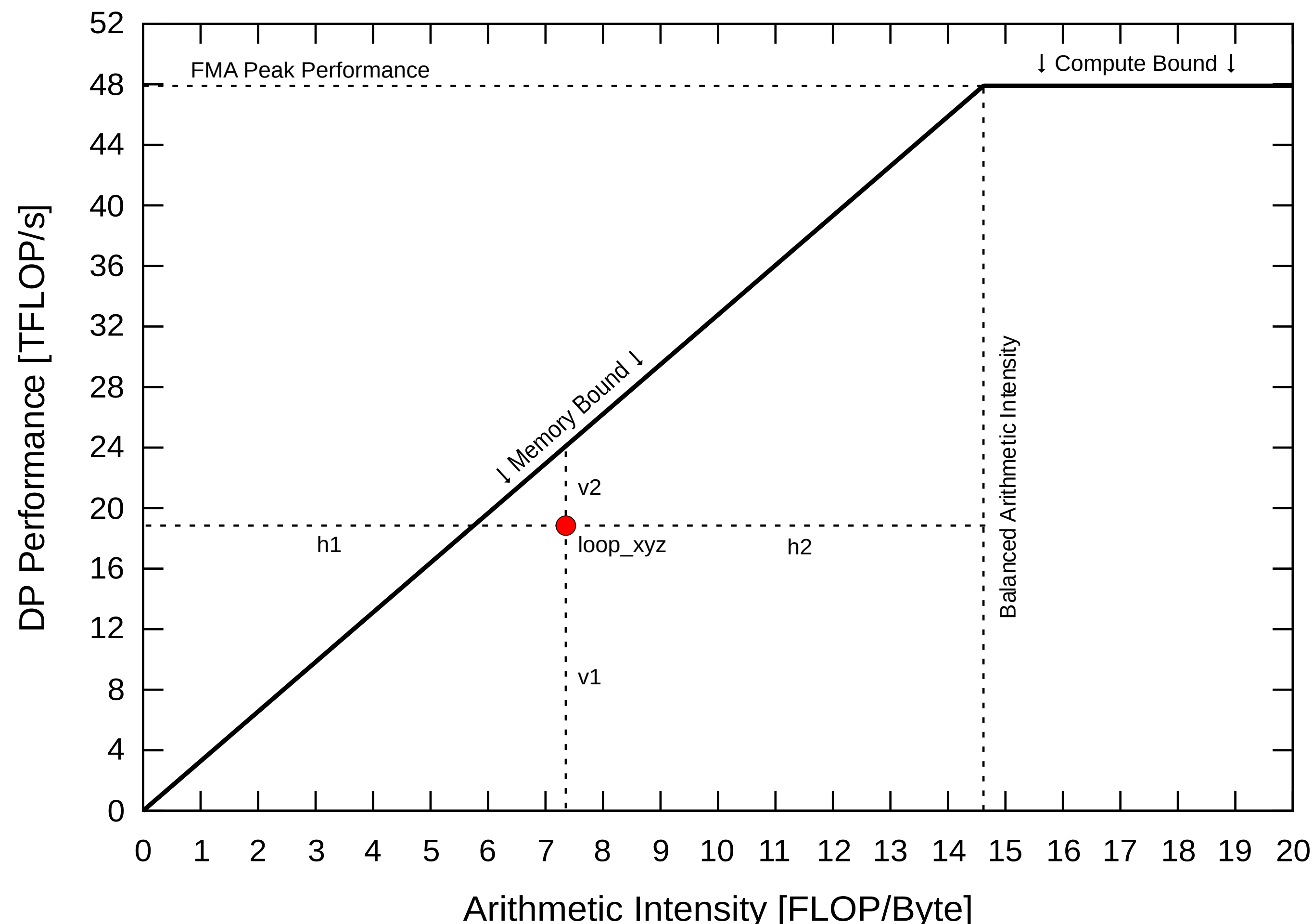
Roofline Model

H L R S

- Maximum achievable performance as a function of Arithmetic Intensity [FLOP/Byte]
- Arithmetic Intensity for e.g. a loop: #FLOPs / data_moved_from_to_memory
- Idealized model with peak performance calculated from FMA units working at 100%

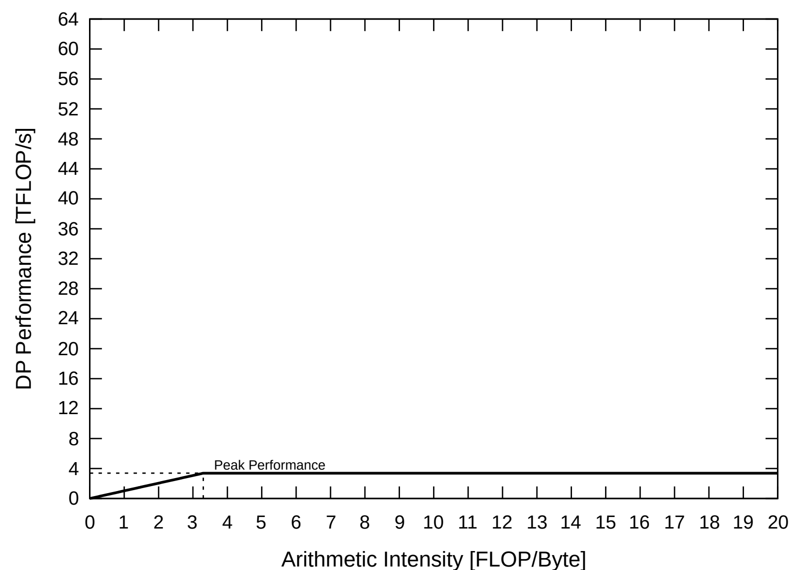
$\frac{h1}{h1+h2}$ = possible FPU usage [%]

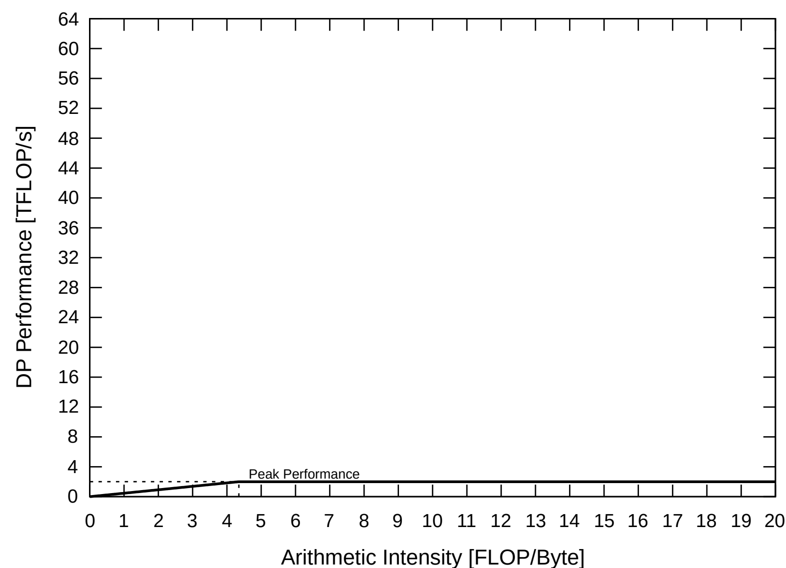
$\frac{v1}{v1+v2}$ = efficiency of possible FPU usage [%]

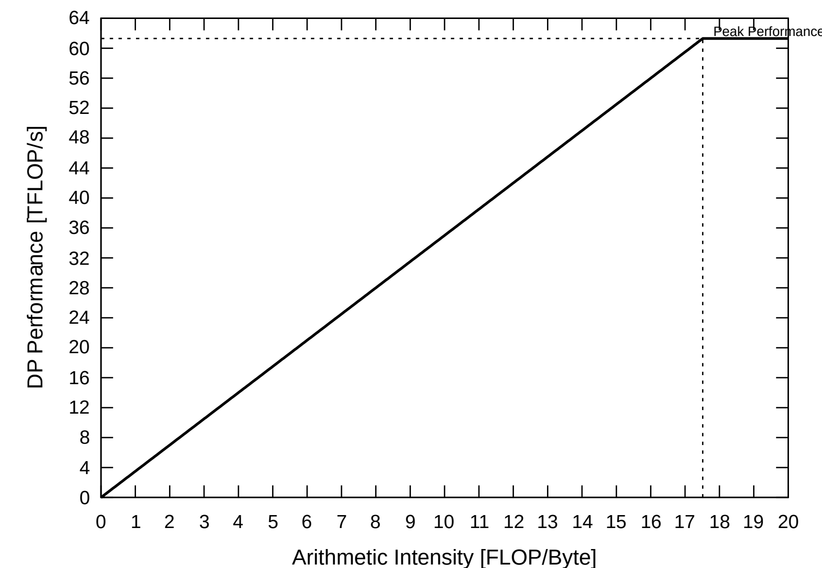


Roofline Model – Current Compute Architectures

HLRS








```
3.4   TFLOP/s
1.024 TB/s
```

```
~2.0   TFLOP/s
 0.460 TB/s
```

```
61.3 TFLOP/s
~3.5 TB/s
```

Overview

H L R S

- Introduction: Memory Bandwidth Problems
- State of Technology – Roofline Model
- State of Technology – Remedies
- New Approach – Proof of Concept
- Outlook

State of Technology – Remedies

Mixed Precision, Lossy Compression

State of Technology – Remedies

HLRS

- Mixed-Precision approaches
  - Storing as single-precision → convert to double for computation
    - Accuracy issues (~7-8 digits)
  - Instructions taking single-precision as input, but output to double precision
  - Fixed-point as storage
    - Scaling/offset (FMA → 2 FLOPs gone)
    - Rounding (+0.5/-0.5 → 1 FLOP gone, lots of instr. for branchless)
    - uint ↔ FP conversion (high latency)
- (lossy) memory compression, ZFP library from LLNL
  - Linkable library → function calls
  - Bandwidths of a couple ~100 MB/s, ~1 GB/s multi-threaded
    - New bottleneck…
  - Feasible for I/O
  - Basic concepts are used in new approach

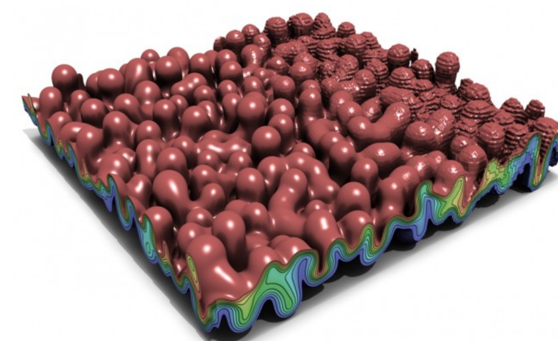


Illustration of zfp's ability to vary the compression ratio to any desired setting, from 10:1 on the left to 250:1 on the right, where the partitioning of the data set into small blocks is evident. The visualization shows the density field from a Rayleigh-Taylor instability simulation.

Overview

H L R S

- Introduction: Memory Bandwidth Problems
- State of Technology – Roofline Model
- State of Technology – Remedies
- New Approach – Proof of Concept
- Outlook

New Approach – Proof of Concept

Real-Time Lossy Array Compression

Real-Time Lossy Array Compression

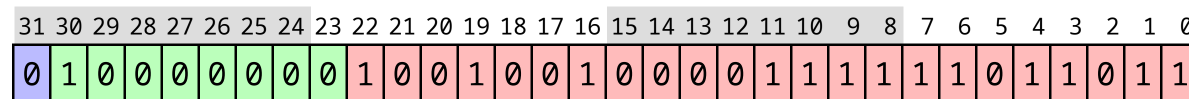
H L R S

- Computer simulations are simulating physical effects, thus have data arrays of physical properties
- Example physical properties for CFD simulatinos:
  - temperature, pressure, density, viscosity, enthalpy, entropy, etc.
- In a specific simulation, the upper and lower bounds of values of these arrays can be
  - determined by running a test case of the simulation
  - estimated from boundary conditions
- The **knowledge** of the array value bounds will be used at **compile-time** for a **real-time** compression algorithm
  - real-time: keep up with memory bandwidth and not take measurable time
  - designed from the bottom-up i.e. with overseeable countable instructions

Floating-Point Format – IEEE (32/64-bit)

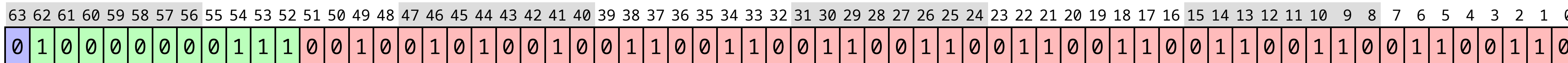
Components:

- Sign
- Exponent
- Significand



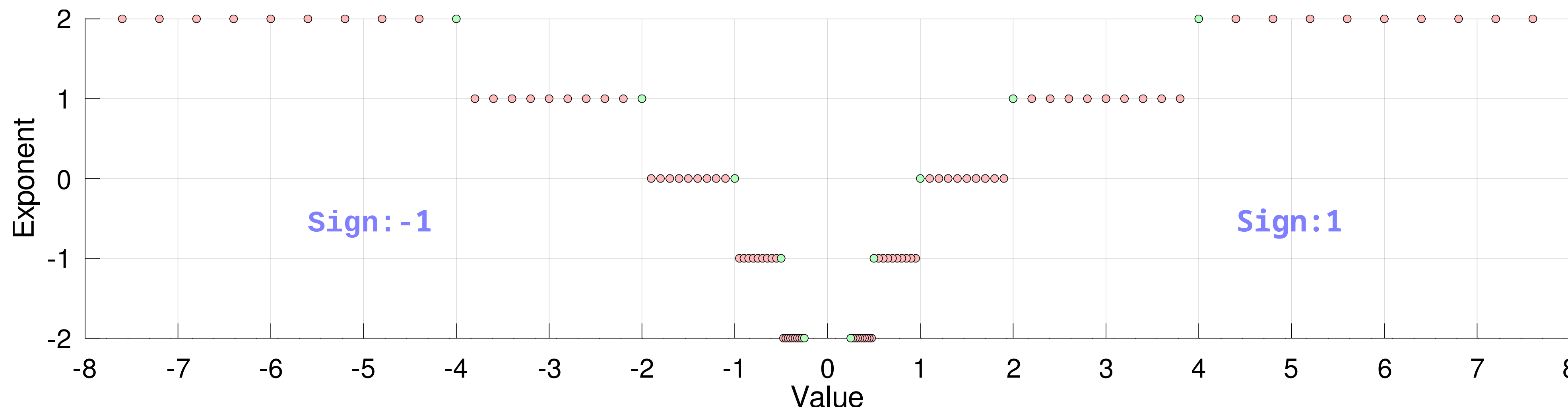
$1 \times 2^{1} \times 1.5707964 = 3.1415927$

Accuracy: ~7-8 digits **Range: ±1.17×10^-38 … ±3.4×10^38**



$1 \times 2^{8} \times 1.1451171875 = 293.15$

Accuracy: ~15-17 digits **Range: ±4.94×10^-324 … ±1.8×10^308**

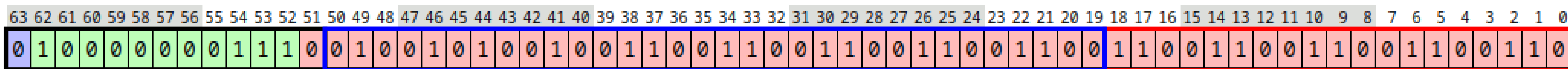


Floating-Point Format – Example Array

H L R S

Temperature array:

293.15K … 333.15K ≙ 20°C … 60°C

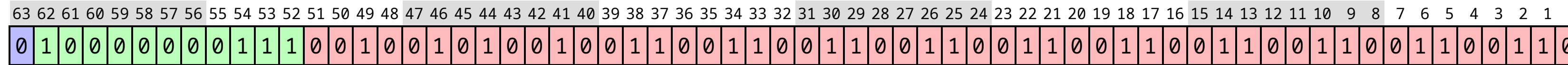


$1 \times 2^8 \times 1.1451171874999999 = 293.15$

constant **most significant 32 bits** **least significant bits**

Lossy Array Compression Algorithm (32-bit storage, speed mode)

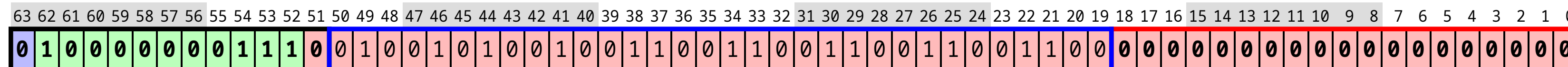
H L R | S



1 × 2^8 × 1.1451171874999999 = 293.15

Accuracy: ~15-17 digits

- e.g. a temperature value of `293.15K` ≙ `20°C`
- value range (incl. margin) for simulation:
  `293.15K … 333.15K`

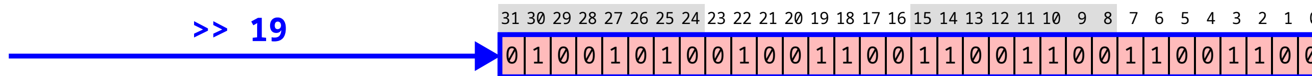


1 × 2^8 × 1.1451171874068677 = 293.14999997615814

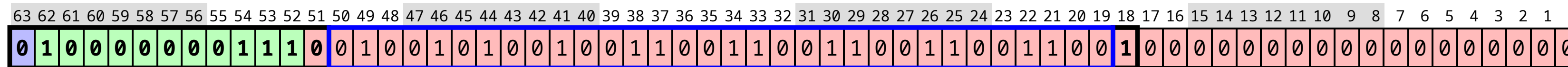
Accuracy: ~10 digits

Memorize:
- memorize **bits `51-63`** (base value **`256.0`**)
- least significant 19 bits essentially truncated

`>> 19`



Store value:
- swap type to **`uint64_t`**
- shift 19 bits right
- cast to **`uint32_t`**
- store **`uint32_t`** value to memory



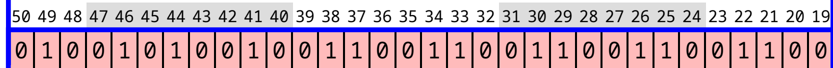
1 × 2^8 × 1.1451171874650754 = 293.1499999910593

"center" truncated bits

OR

OR

`<< 19`



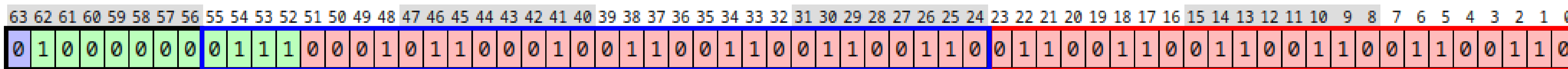
Reconstruct value:
- load **`uint32_t`** value from memory
- cast to **`uint64_t`**
- shift 19 bits left
- bitwise **`OR`** with base value
- swap type to **`double`**
- number crunching in **`double`** precision

Floating-Point Format – Example Array 2

H L R | S

Temperature array:

278.15K … 1823.15K ≙ 5°C … 1550°C

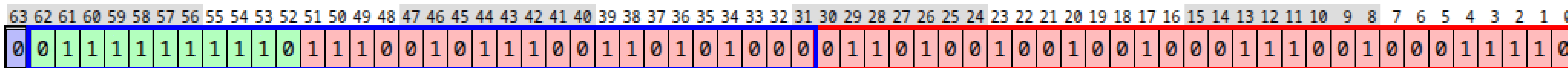


1 × 2^8 × 1.0865234374999999 = 278.15

constant **most significant 32 bits** **least significant bits**

Same temperature array normalized to 20°C (293.15K):

278.15K/293.15K … 1823.15K/293.15K ≙ ~0.949 … 6.22



1 × 2^{-1} × 1.8976633122980000 = 0.948831656149

Lossy Array Compression Algorithm (32-bit storage, precision mode)

H L R S

- value of 1717.82K normalized by 293.15K
- value range (incl. margin) for simulation:
  ~0.949 … 6.22

Memorize:
- memorize base value ~**0.949** as **uint64_t**

Store value:
- swap type to **uint64_t**
- calculate **integer difference** to base value
- shift **integer difference** 22 bits right
- cast to **uint32_t**
- **store uint32_t** value to memory

Reconstruct value:
- **load uint32_t** value from memory
- cast to **uint64_t**
- shift 22 bits left
- integer **ADD** to base value
- swap type to **double**
- number crunching in **double** precision

Proof of Concept
Real-Time Lossy Array Compression Code (RTLAC)

H L R S


```json
{
  "rtlac": [
    {
      "name": "T1",
      "type": "double",
      "bits":  32,
      "min":   293.15,
      "max":   333.15,
      "pref": "speed",
      "mode": "static"
    },{
      "name": "T2",
      "type": "double",
      "bits":  32,
      "min":   0.948,
      "max":   6.22,
      "pref": "precision",
      "mode": "static"
    }
  ]
}
```

rtlac.h:

```c
#include <stdint.h>

#ifndef RTLAC_DISABLE
#define iSFI static __attribute__((always_inline)) inline

typedef union {double f; uint64_t i;} _f64u;

// 293.15 … 333.15
iSFI uint32_t T1_c(double v)   {_f64u r = {v}; return (uint32_t)(r.i >> 19);;}
iSFI double   T1_d(uint32_t v) {_f64u r = {.i = 0x4070000000040000UL | (uint64_t)v << 19}; return r.f;}

// 0.948 … 6.22
iSFI uint32_t T2_c(double d)   {_f64u u = {d}; return (uint32_t)((u.i - 0x3FEE5604189374BCUL) >> 22);;}
iSFI double   T2_d(uint32_t c) {_f64u u; u.i = 0x3FEE5604189374BCUL + ((uint64_t)(c) << 22); return u.f;}
#else
[…] // Disable RTLAC Code
#endif
```

Fortran Port (Native Module + C Interface Module)

Native Fortran Module

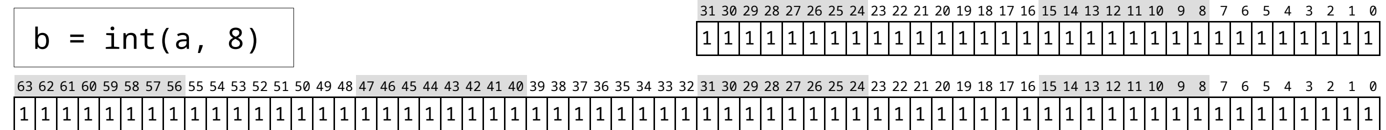
```fortran
module rtlac
  implicit none
  contains
#if !defined(RTLAC_DISABLE)
  pure integer(kind=4) function rho_c(v)
    implicit none
    real(kind=8), intent(in) :: v
    real(kind=8)             :: vr
    integer(kind=8)          :: vi
    equivalence(vr,vi)
    vr = v
    rho_c = int(ishft(vi, -17), 4)
  end function rho_c

  pure real(kind=8) function rho_d(v)
    implicit none
    integer(kind=4), intent(in)   :: v
    integer(kind=4), dimension(2) :: v2
    integer(kind=8)               :: vi
    real(kind=8)                  :: rr
    integer(kind=8)               :: ri
    equivalence(v2,vi)
    equivalence(rr,ri)
    vi     = 0
    v2(1)  = v
    ri     = ior(ishft(vi, 17), 4607745368753504256_8)
    rho_d = rr
  end function rho_d
#else
  […] ! Disable RTLAC Code
#endif
```

H L R I S

Technology for unsigned integers is just not there yet in Fortran:

```fortran
integer(kind=4) :: a
a = -1
```

```fortran
b = int(a, 8)
```



C Interface Module

```fortran
module rtlac_cif
  implicit none
  contains
#if !defined(RTLAC_DISABLE)
  integer(kind=4) function rho_c(v) bind(C, name = 'rho_c')
    use, intrinsic         :: iso_c_binding
    real(c_double), value :: v
  end function rho_c

  real(kind=8) function rho_d(v) bind(C, name = 'rho_d')
    use, intrinsic          :: iso_c_binding
    integer(kind=4), value :: v
  end function rho_d
#else
  […] ! Disable RTLAC Code
#endif
```

Before you ask (shortened due to limited time)

HLRS

- Yes, the compilers are able to SIMD vectorize this
- Yes, there is an implementation that adapts the range at runtime
  - e.g. temperature and pressure values increase as combustion happens in a simulation
  - e.g. every 100 time steps → check min/max values → add like ±10% margin → recompress with new parameters
- Yes, out-of-bounds checks are possible
  - Branchless for performance, e.g.:

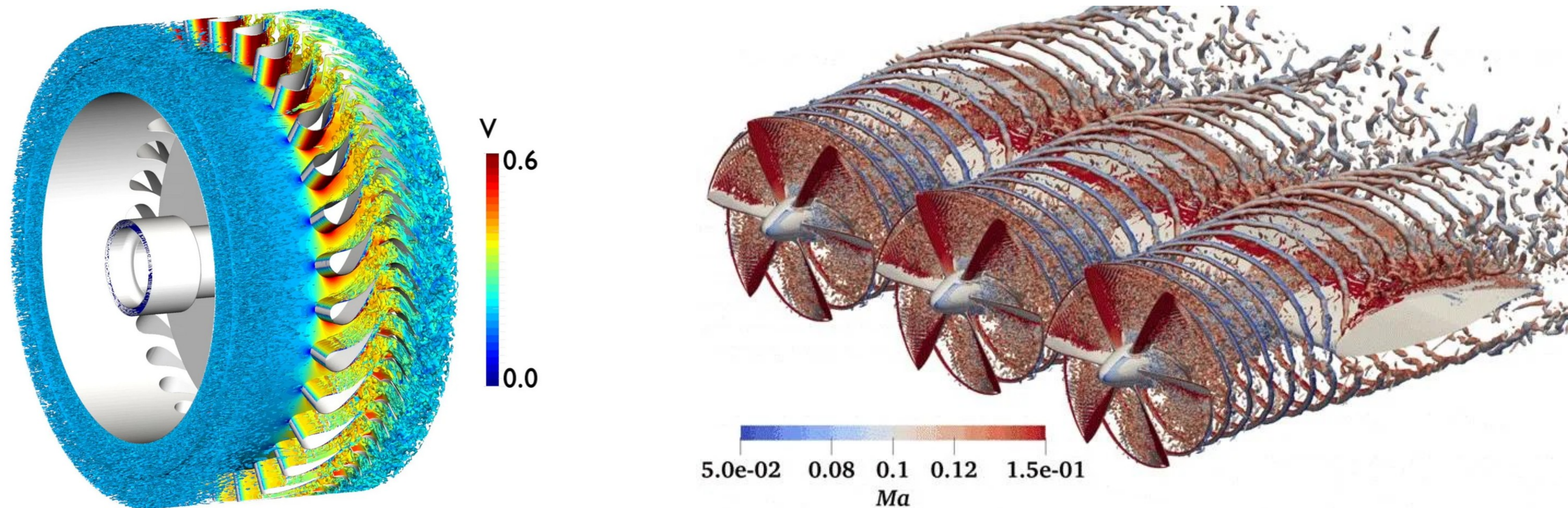
```
uint64_t oob = 0;
[…]
oob += (value < min_value); oob += (value > max_value);
[…]
if (oob != 0) {/* expand range and redo timestep or abort simulation */}
```

- Yes, 24-, 40-, 48-, 56- bit types can be properly stored
  - AoS → when other values can squeeze in the gap → goal: reduce struct size to cache line size
    - with bitmasks to leave adjacent data intact when storing and to mask away adjacent bits when reading
  - SoA → memcpy to/from local value

First Impressions
16-, 24- and 32-bit Lossy Compression in m-AIA LB Solver

H L R S

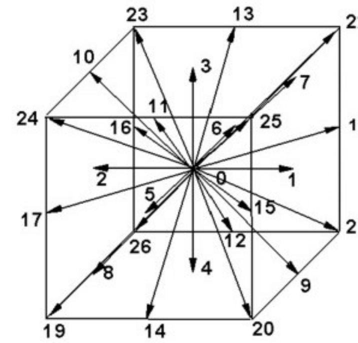
- m-AIA (multi-physics AIA) is a multi-physics partial differential equation (PDE) solver framework with a focus on problems related to computational fluid dynamics, computational aeroacoustics and structural mechanics.
- It is developed at the Institute of Aerodynamics (AIA) of the RWTH Aachen University in Germany.
- Projects using m-AIA use up a large fraction of the computing time at HLRS



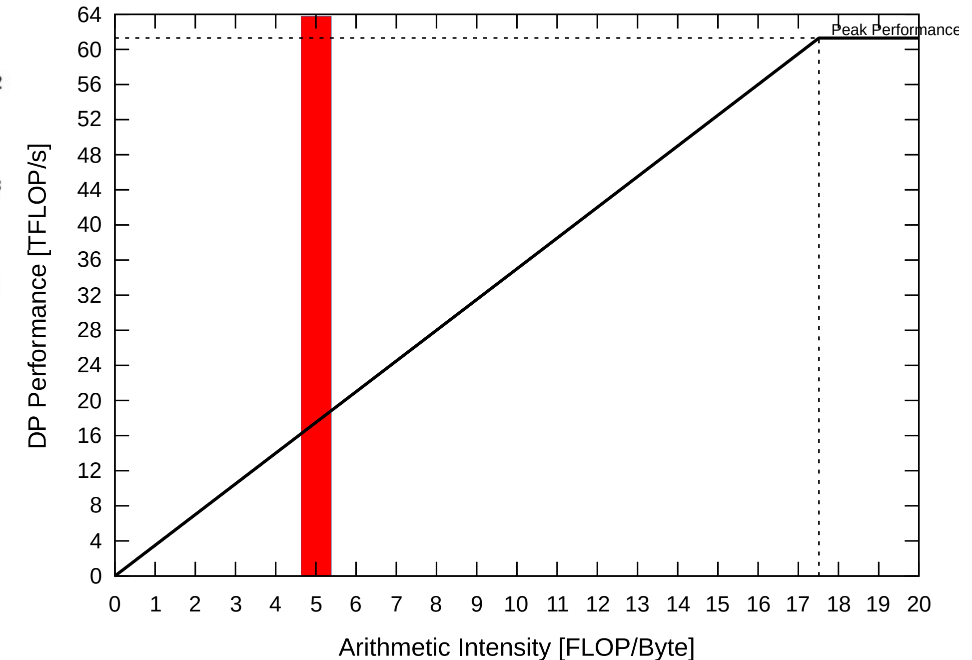
Images from: https://services.excellerat.eu/en/services/application-software/m-aia/

Impressions
16-, 24- and 32-bit Lossy Compression in m-AIA LB Solver

- We focus on the D3Q27 Lattice Boltzmann (LB) Solver with hotspots:
  - **Collision** step (~5 FLOP/Byte)
  - **Propagation** step (memory-intensive)



- With reducing memory, users would also benefit from
  - Being able to run larger models on given hardware
  - MPI transfers of halo cells transmit less data (faster)
  - Less MPI ranks (scaling)
  - Better cache re-use

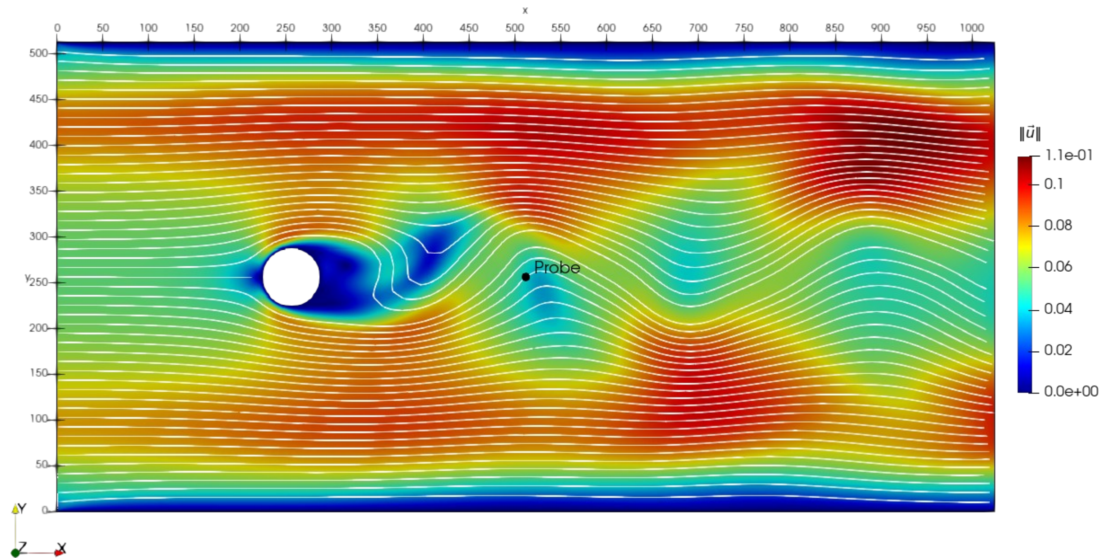


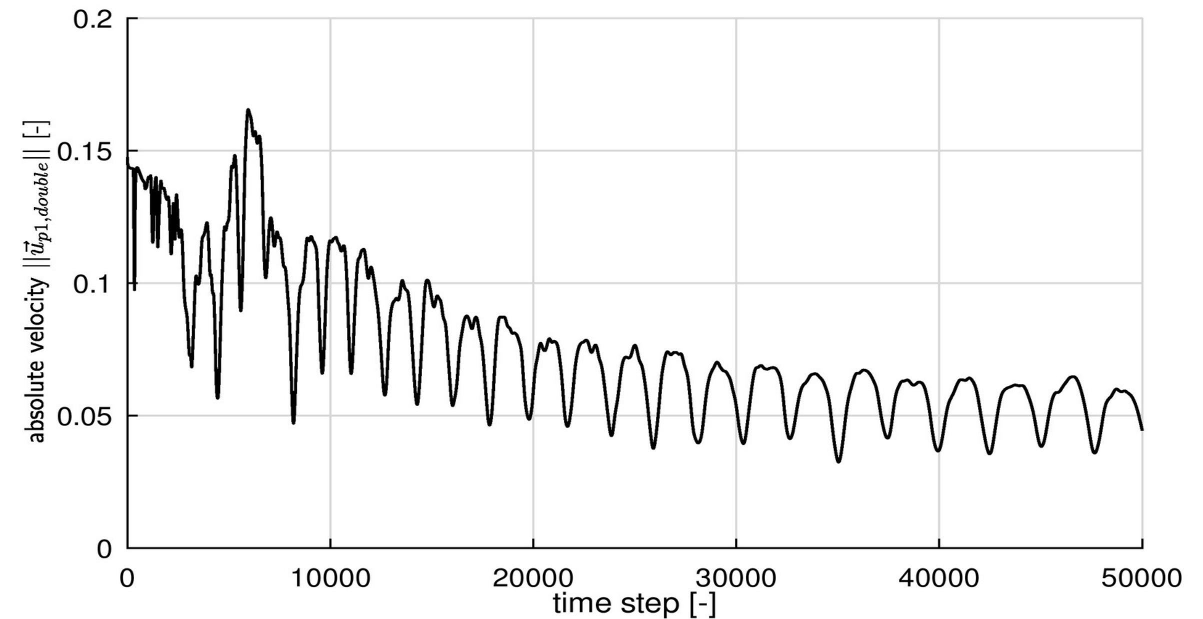
61.3 TFLOP/s
~3.5 TB/s

First Impressions
16-, 24- and 32-bit Lossy Compression in m-AIA LB Solver

HLRS

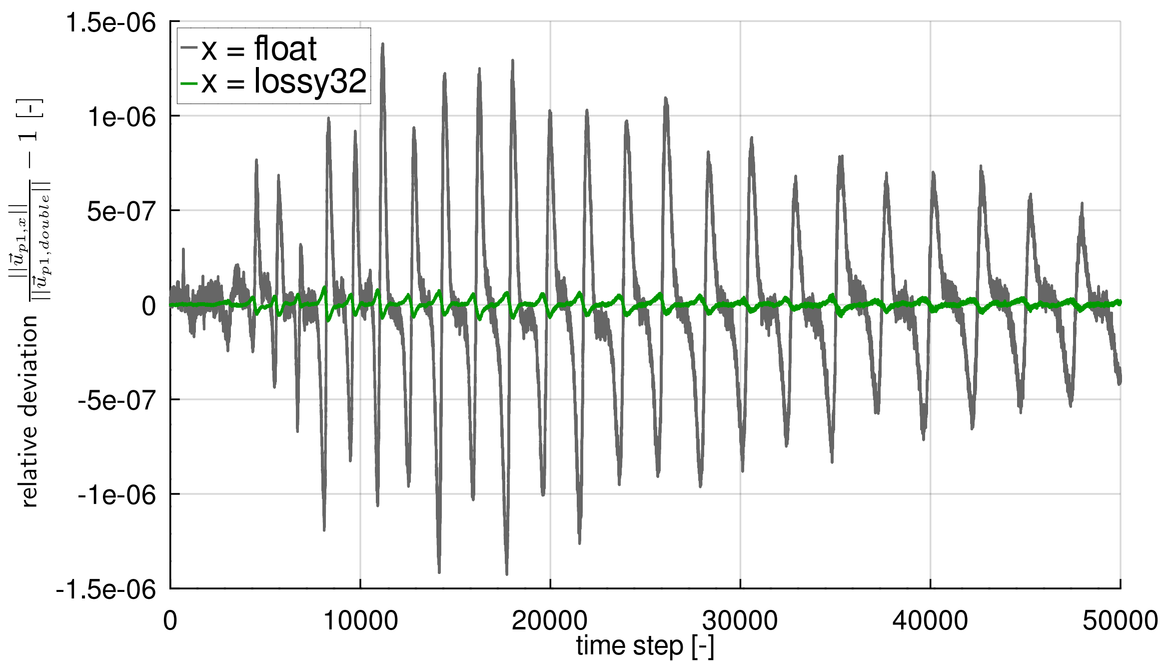
Karman Vortex 2D test case set up by **Julain Vorspohl (AIA)**:
- 160k cells, ~5GB allocated
- RTLAC algorithm was wrapped with C++ operator overloading

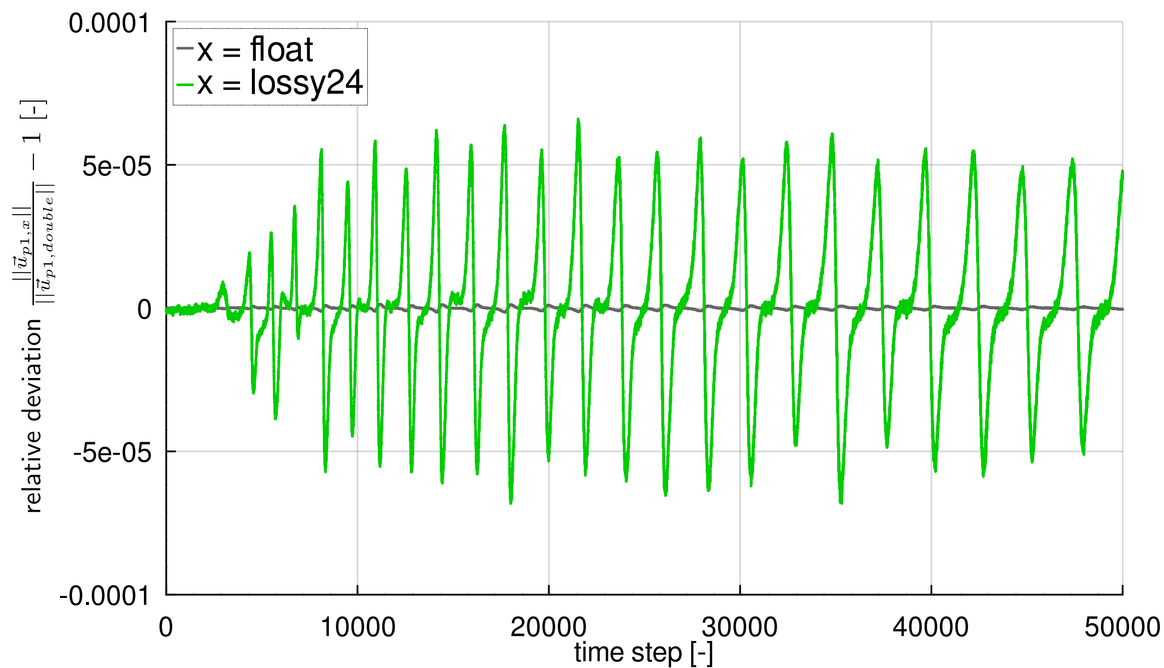


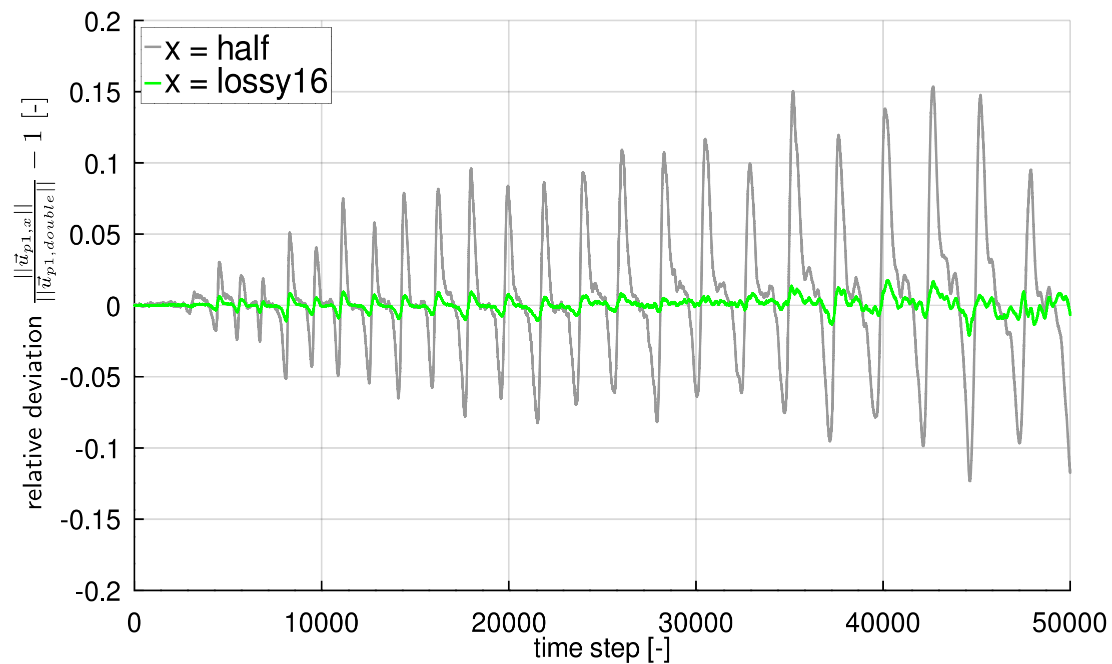


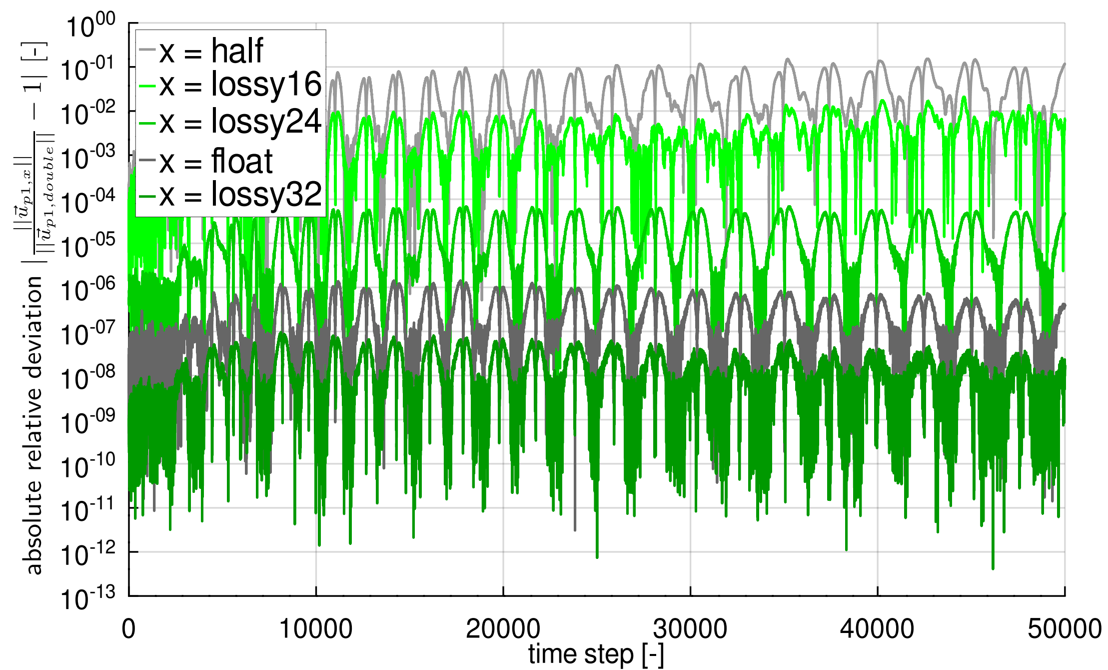
Density distribution functions' value range for this case:

```
-0.116 … 0.149 (measured)
-0.2   … 0.2   chosen as safe range
 0.55  … 0.95  by shifting by 0.75
```


x = float
x = lossy32
relative deviation $\frac{||\vec{u}_{p1,x}||}{||\vec{u}_{p1,double}||} - 1$ [-]
time step [-]

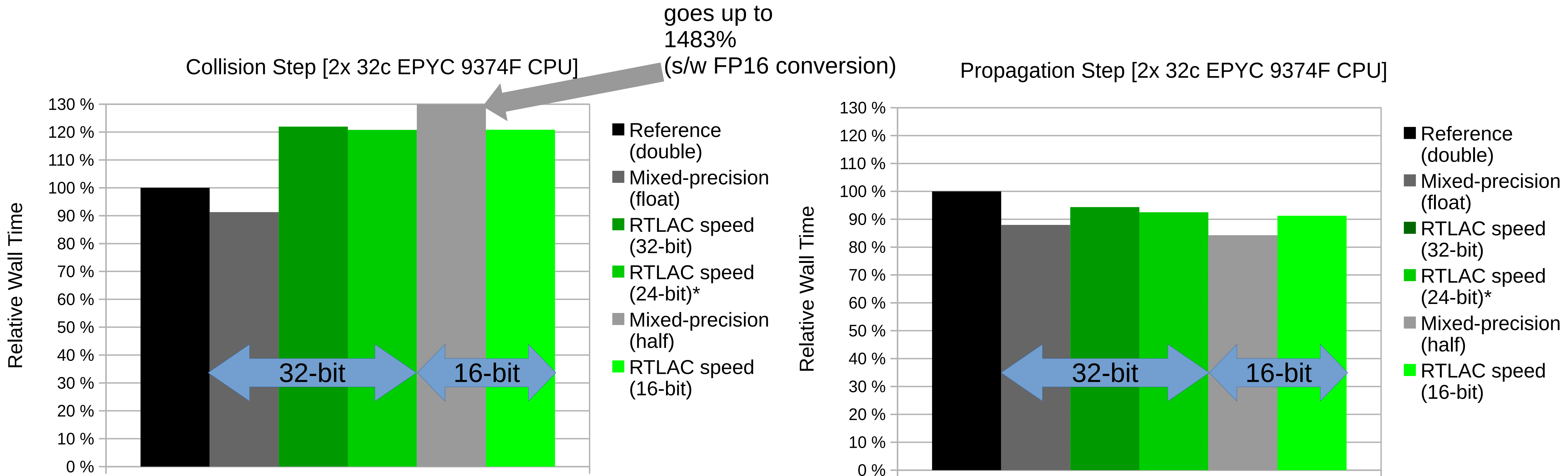

x = float
x = lossy24
relative deviation $\frac{||\vec{u}_{p1,x}||}{||\vec{u}_{p1,double}||} - 1$ [-]
time step [-]


x = half
x = lossy16
relative deviation $\frac{||\vec{u}_{p1,x}||}{||\vec{u}_{p1,double}||} - 1$ [-]
time step [-]


x = half
x = lossy16
x = lossy24
x = float
x = lossy32
absolute relative deviation $\left|\frac{||\vec{u}_{p1,x}||}{||\vec{u}_{p1,double}||} - 1\right|$ [-]
time step [-]

First Impressions
16-, 24- and 32-bit Lossy Compression in m-AIA LB Solver

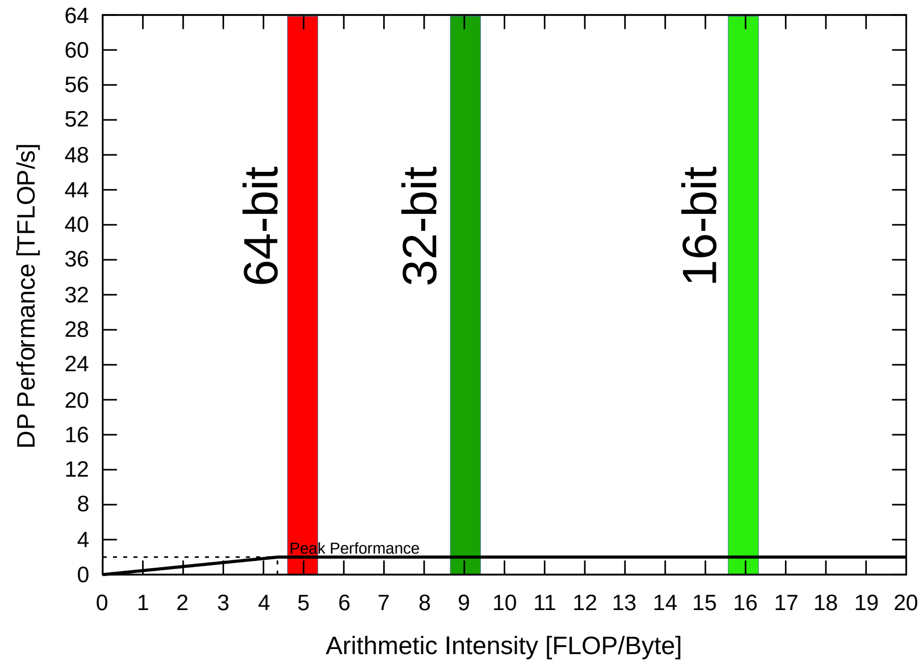
HLRS



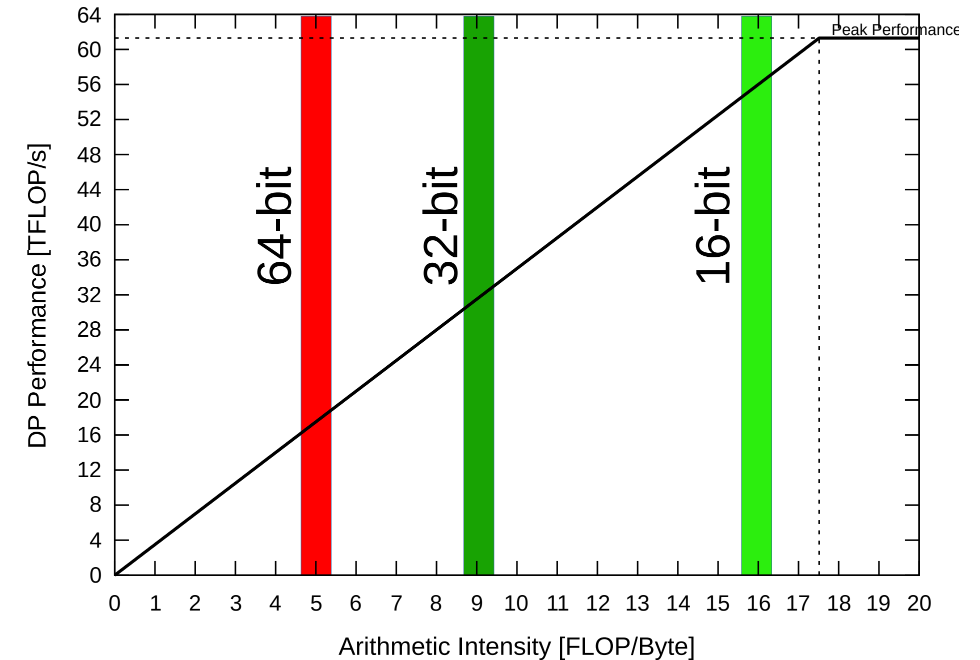
*) stored as 32-bit

First Impressions
16-, 24- and 32-bit Lossy Compression in m-AIA LB Solver
Collision Step Arithmetic Intensity

H L R S



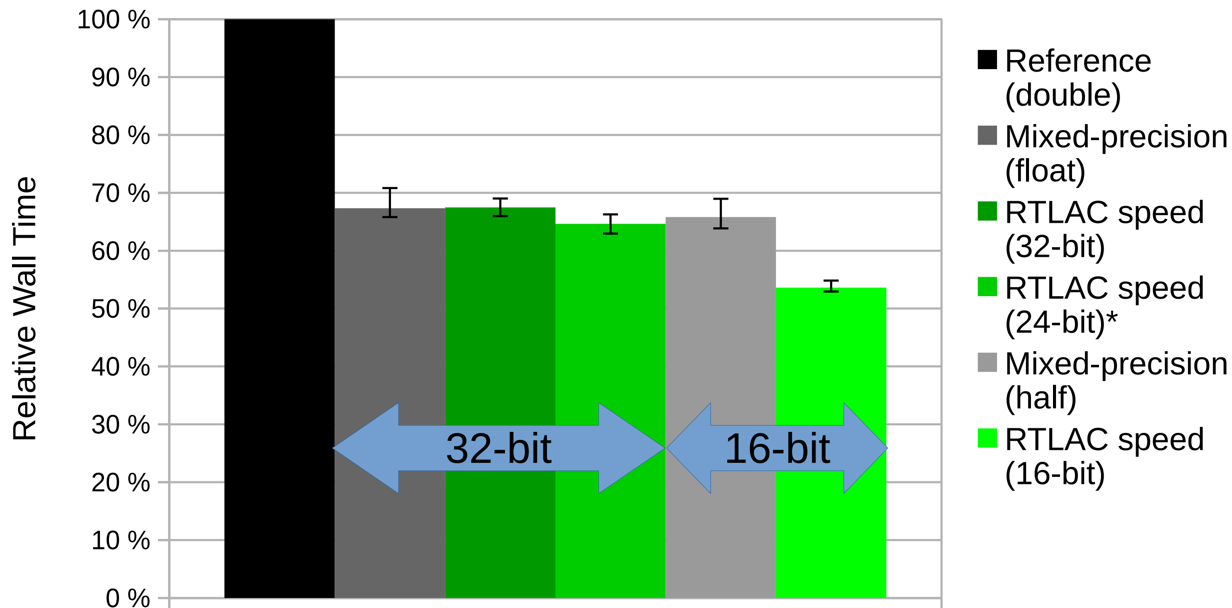
~2.0 TFLOP/s
0.460 TB/s

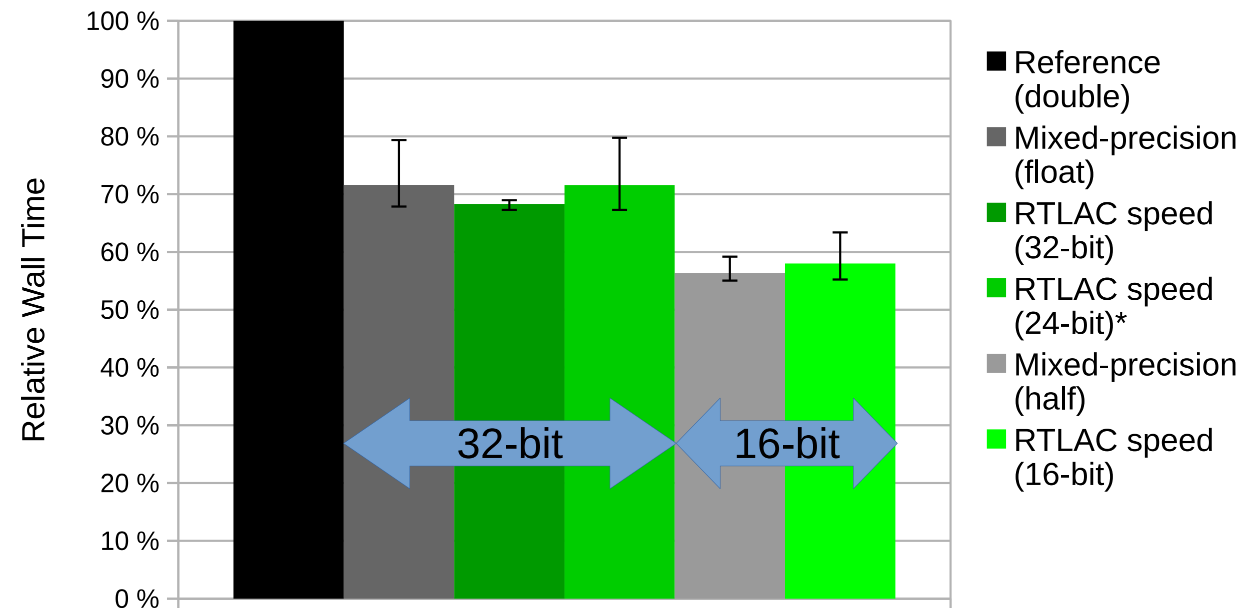


61.3 TFLOP/s
~3.5 TB/s

First Impressions
16-, 24- and 32-bit Lossy Compression in m-AIA LB Solver

H L R S





*) stored as 32-bit

Overview

H L R S

- Introduction: Memory Bandwidth Problems
- State of Technology – Roofline Model
- State of Technology – Remedies
- New Approach – Proof of Concept
- Outlook

Outlook

Outlook – Topics

HLRS

- Method looks very promising for some examples and a simulation code so far
- Further analyze feasibility with memory bandwidth bound simulation codes with actual numerics
  - m-AIA (formerly ZFS) C++ AIA RWTH Aachen
  - You?
- Bring the idea up conferences
  - So far presented at
    - 37th + 41th WSSP 2024 at HLRS
    - 30th PARS Workshop 2024 at TH Ingolstadt
    - SIAM PP 26 in Berlin
  - Save computing resources limited by memory bandwidth or memory capacity when applied in codes
- Write up and release the code
  - Code: 09/2026
  - Thesis: 12/2026

The End

H L R I S

Thank you for your attention.

Open for questions and feedback.