13th International Workshop on OpenCL and SYCL

IWOCL 2025

Scaling up FluidX3D CFD beyond 100 Billion cells on a single computer - a story about the cross-compatibility of OpenCL

Dr. Moritz Lehmann, Intel Corporation

April 7-11, 2025 | Heidelberg, Germany | iwocl.org

KHRONOS GROUP

ProjectPhysX/FluidX3D: The fast

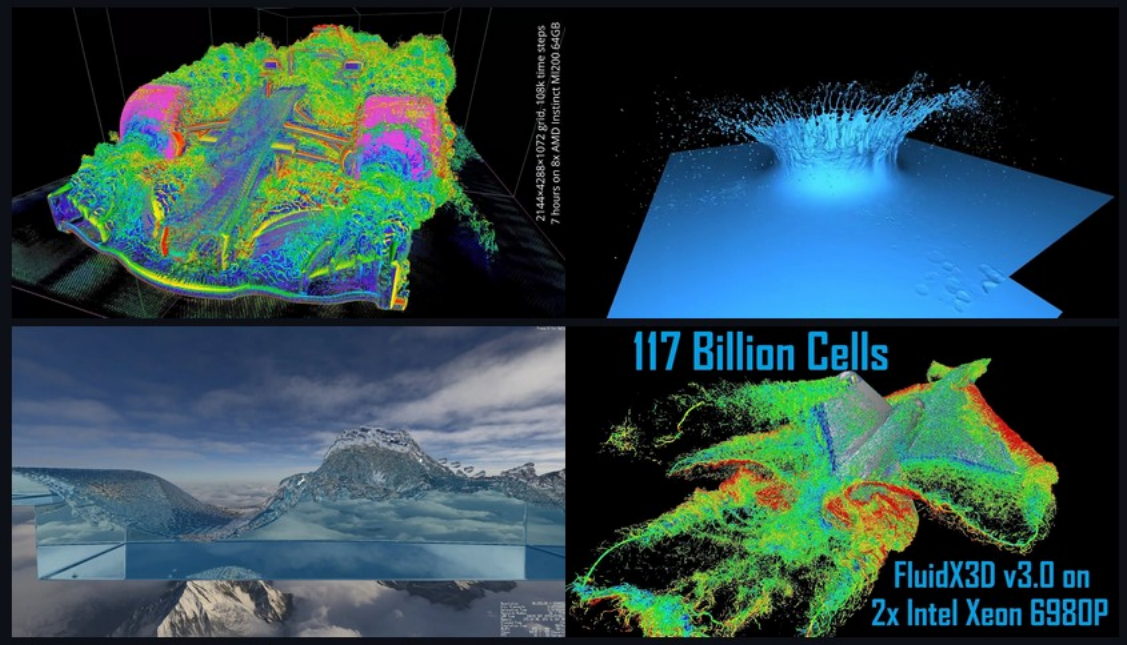
https://github.com/ProjectPhysX/FluidX3D

LICENSE.md	Workaround for Intel CPU compiler bug that causes renderin...	last month
README.md	Updated H100 SXM5 benchmarks in Readme	last week
make.sh	Enabled g++ compiler optimizations on Linux for faster start...	11 months ago
makefile	Fixed that voxelization did not always produce binary identic...	11 months ago

README License

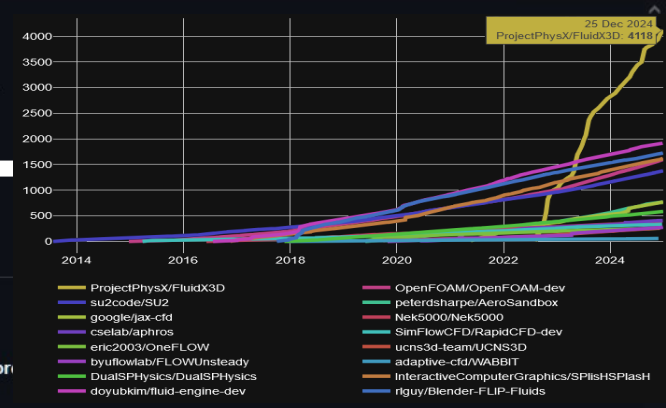
FluidX3D

The fastest and most memory efficient lattice Boltzmann CFD software, running on all GPUs and CPUs via OpenCL. Free for non-commercial use.



(click on images to show videos on YouTube)

Readme
View license
Cite this repository
Activity
4.3k stars
61 watching
366 forks



Releases 28

FluidX3D v3.2 (fast for
last week

+ 27 releases

Languages

C++ 69.8% C 29.4% Other 0.8%

OpenCL™

Rankings on github.com/topics/
- #1 in cfd
- #6 in hpc
- #9 in opencl

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Part 1: OpenCL™

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

OpenCL™ - Big Advantages of Open Standard

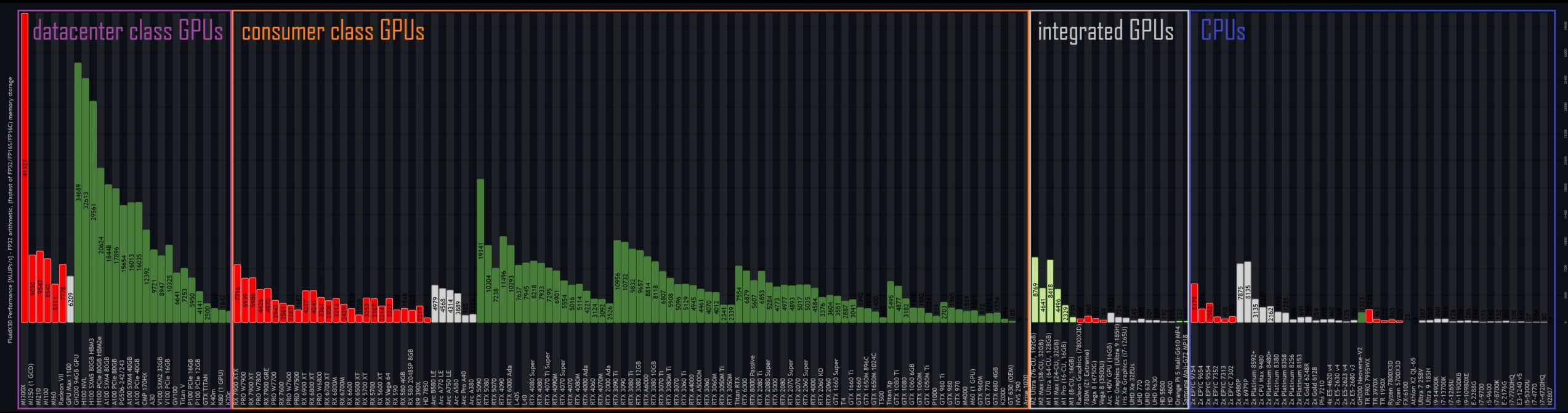
- out-of-the-box compatibility with
  - all GPUs from all vendors (AMD, Intel, Nvidia, ARM, Apple, Glenfly, …)
    - gaming
    - workstation
    - datacenter
    - integrated
  - all modern CPUs (AMD, Intel) and Intel Xeon Phi
  - FPGAs, Accelerators
- same performance as CUDA
- optimize for one GPU, optimize for all, no porting required!

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

FluidX3D performance across different Hardware



Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Part 2: Lattice Boltzmann Method

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

The lattice Boltzmann method

1. Streaming $$f_i^{\text{temp}}(\vec{x},t) = f_i^{\text{A}}(\vec{x}-\vec{e}_i,\,t)$$ flags

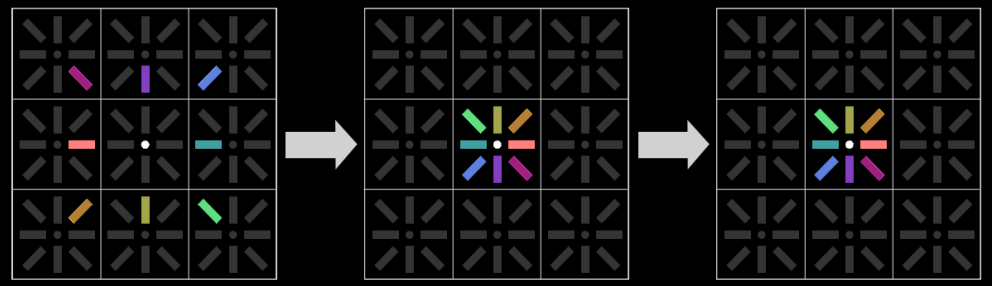
2. Collision (SRT)

$$\rho(\vec{x},t) = \sum_i f_i^{\text{temp}}(\vec{x},t)$$

$$\vec{u}(\vec{x},t) = \frac{1}{\rho(\vec{x},t)} \sum_i \vec{c}_i\, f_i^{\text{temp}}(\vec{x},t)$$

$$f_i^{\text{eq}}(\vec{x},t) = f_i^{\text{eq}}(\rho(\vec{x},t),\,\vec{u}(\vec{x},t)) = w_i\,\rho\cdot\left(\frac{(\vec{u}\circ\vec{c}_i)^2}{2\,c^4}+\frac{\vec{u}\circ\vec{c}_i}{c^2}+1-\frac{\vec{u}\circ\vec{u}}{2\,c^2}\right)$$

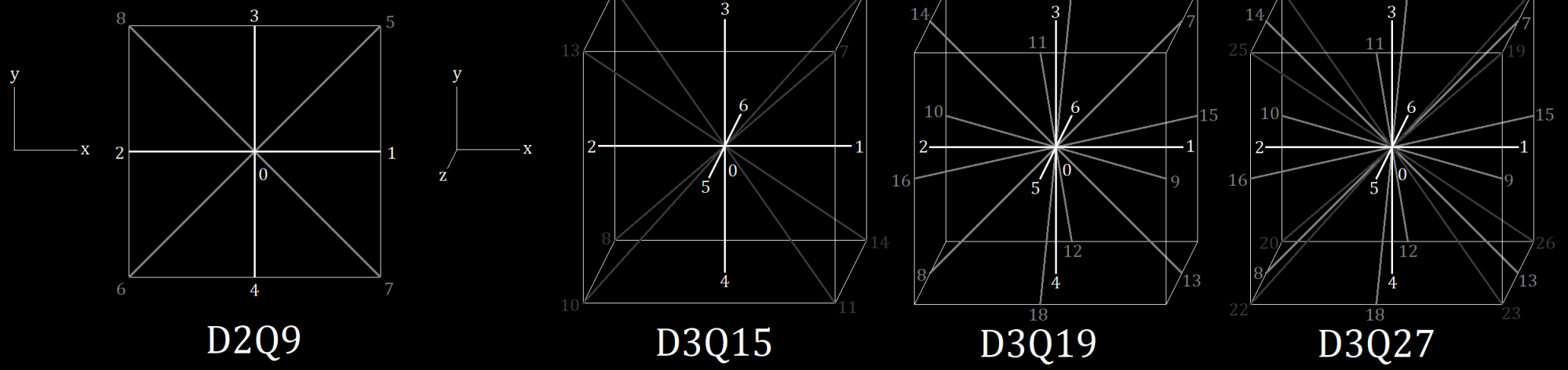
$$f_i^{\text{B}}(\vec{x},\,t+\Delta t) = \left(1-\frac{\Delta t}{\tau}\right) f_i^{\text{temp}}(\vec{x},t) + \frac{\Delta t}{\tau}\, f_i^{\text{eq}}(\vec{x},t)$$



read DDFs from global memory

temporary DDFs for collision in registers

write DDFs to global memory



D2Q9 D3Q15 D3Q19 D3Q27

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel UNIVERSITÄT BAYREUTH

Part 3: Counting the Bytes

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

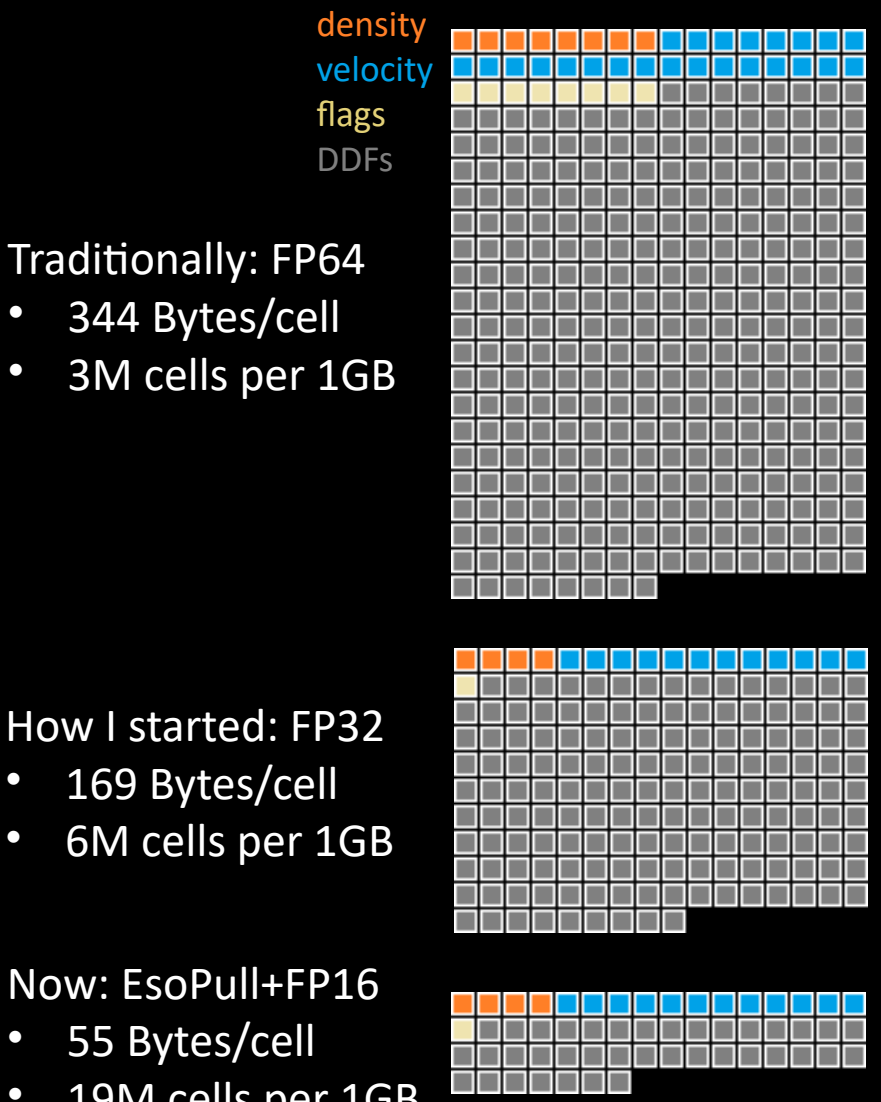
UNIVERSITÄT BAYREUTH

Getting the Memory Problem under Control

LBM‘s Dilemma: Memory Demand

- LBM is super fast on GPUs
- LBM has high memory demand
- GPUs don‘t have a lot of memory

1 LBM grid cell in Bytes



Traditionally: FP64

- 344 Bytes/cell
- 3M cells per 1GB

How I started: FP32

- 169 Bytes/cell
- 6M cells per 1GB

Now: EsoPull+FP16

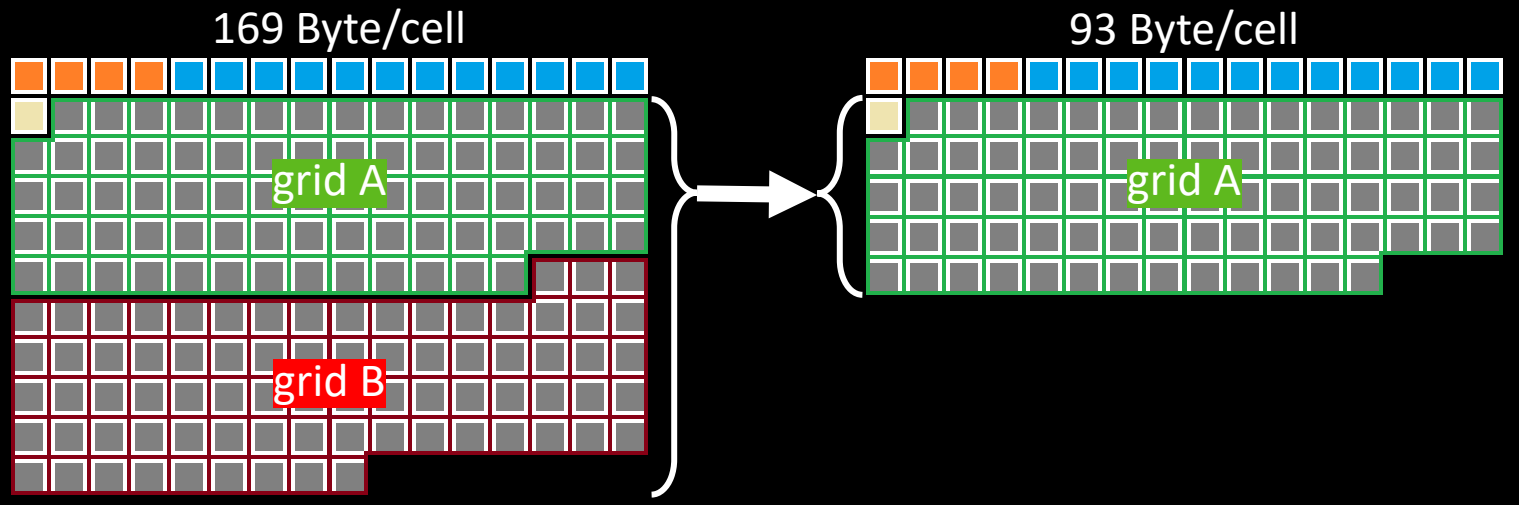
- 55 Bytes/cell
- 19M cells per 1GB

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

a) Esoteric-Pull In-Place Streaming



Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

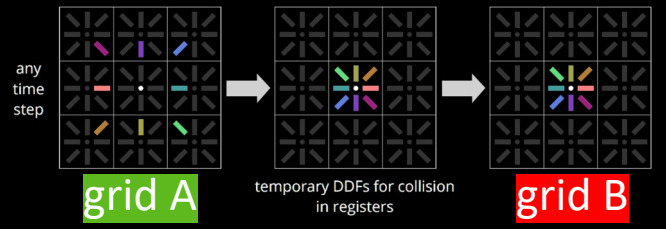
intel

UNIVERSITÄT BAYREUTH

LBM Streaming Schemes

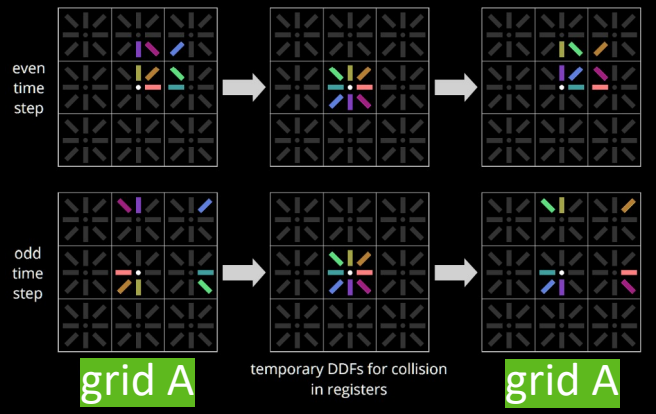
One-Step-Pull

- 2x memory demand: needs 2 copies A&B of the DDFs



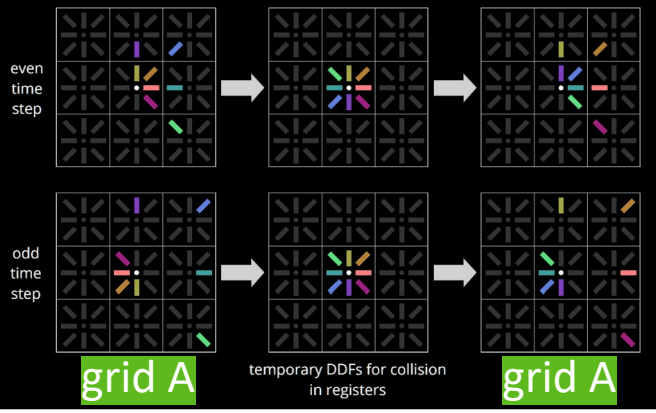
Esoteric-Twist (Geier & Schönherr, 2017)

- 1x memory demand
- streaming directions ≠ neighbor directions



Esoteric-Pull (Lehmann, 2022)

- 1x memory demand
- streaming directions = neighbor directions
- optimal pattern for coalesced memory access

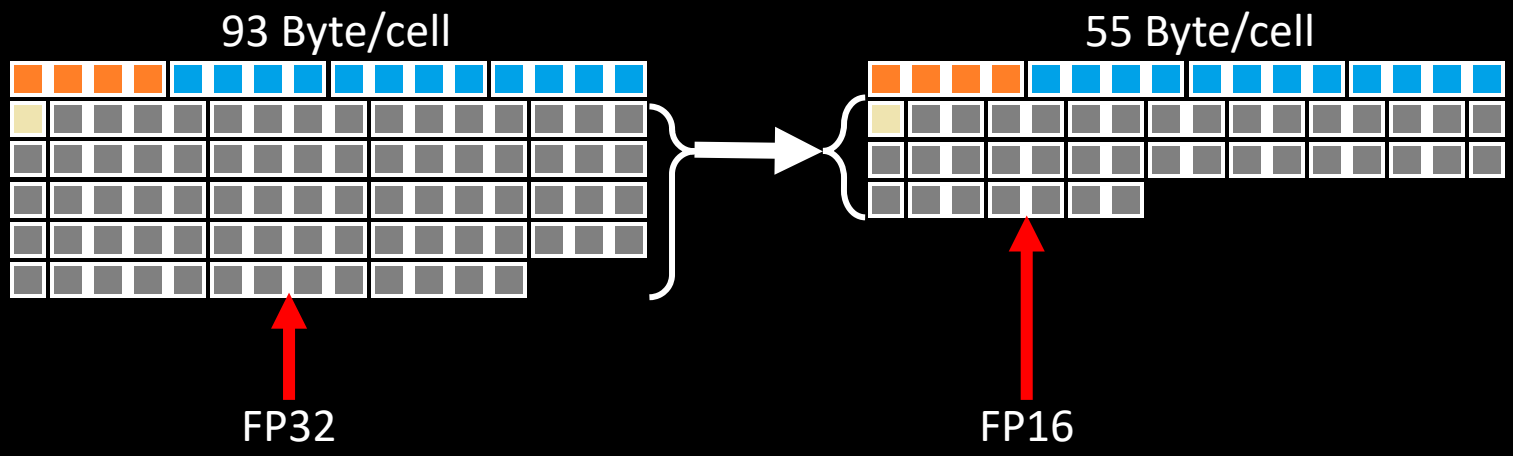


Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

b) FP16 Memory Compression



Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Decouple arithmetic precision and memory precision

1. Streaming

$$f_i^{\text{temp}}(\vec{x},t) = f_i^{\text{A}}(\vec{x}-\vec{e}_i,\,t)$$

2. Collision (SRT)

$$\rho(\vec{x},t) = \left(\sum_i f_i^{\text{temp}}(\vec{x},t)\right) + 1$$

$$\vec{u}(\vec{x},t) = \frac{1}{\rho(\vec{x},t)} \sum_i \vec{c}_i\, f_i^{\text{temp}}(\vec{x},t)$$

$$f_i^{\text{eq-shifted}}(\vec{x},t) = f_i^{\text{eq-shifted}}(\rho(\vec{x},t),\,\vec{u}(\vec{x},t)) = w_i\,\rho \cdot \left(\frac{(\vec{u}\circ\vec{c}_i)^2}{2\,c^4} - \frac{\vec{u}\circ\vec{u}}{2\,c^2} + \frac{\vec{u}\circ\vec{c}_i}{c^2}\right) + w_i\,(\rho-1)$$

$$f_i^{\text{B}}(\vec{x},\,t+\Delta t) = \left(1-\frac{\Delta t}{\tau}\right) f_i^{\text{temp}}(\vec{x},t) + \frac{\Delta t}{\tau}\, f_i^{\text{eq}}(\vec{x},t)$$

FP32 arithmetic

conversion

FP32 or FP16 storage

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

DDF-shifting (Skordos, 1993)

naive implementation: DDFs f_i

1. Streaming

$$f_i^{\text{temp}}(\vec{x},t) = f_i^{\text{A}}(\vec{x}-\vec{e}_i,\,t)$$

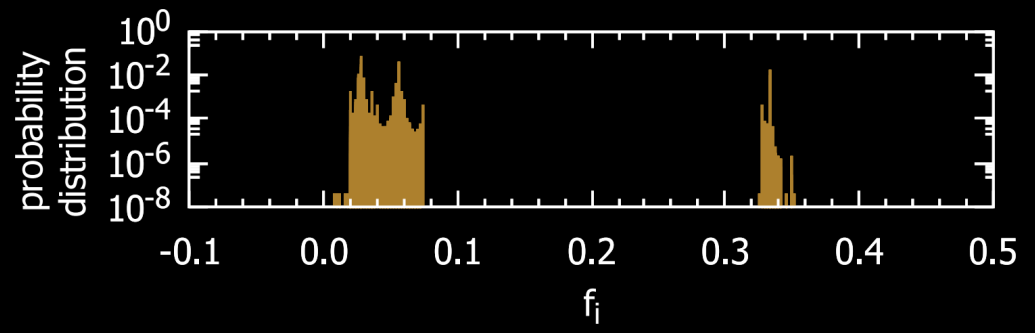
2. Collision (SRT)

$$\rho(\vec{x},t) = \sum_i f_i^{\text{temp}}(\vec{x},t)$$

$$\vec{u}(\vec{x},t) = \frac{1}{\rho(\vec{x},t)} \sum_i \vec{c}_i\, f_i^{\text{temp}}(\vec{x},t)$$

$$f_i^{\text{eq}}(\vec{x},t) = f_i^{\text{eq}}(\rho(\vec{x},t),\,\vec{u}(\vec{x},t)) = w_i\,\rho \cdot \left(\frac{(\vec{u}\circ\vec{c}_i)^2}{2\,c^4} + \frac{\vec{u}\circ\vec{c}_i}{c^2} + 1 - \frac{\vec{u}\circ\vec{u}}{2\,c^2} \right)$$

$$f_i^{\text{B}}(\vec{x},\,t+\Delta t) = \left(1-\frac{\Delta t}{\tau}\right) f_i^{\text{temp}}(\vec{x},t) + \frac{\Delta t}{\tau}\, f_i^{\text{eq}}(\vec{x},t)$$



Skordos: shift DDFs $f_i^{\text{shifted}} = f_i - w_i$

1. Streaming

$$f_i^{\text{temp}}(\vec{x},t) = f_i^{\text{A}}(\vec{x}-\vec{e}_i,\,t)$$

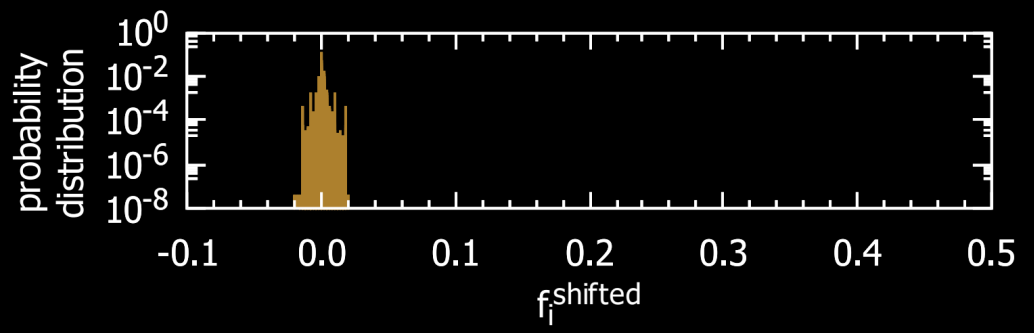
2. Collision (SRT)

$$\rho(\vec{x},t) = \left(\sum_i f_i^{\text{temp}}(\vec{x},t) \right) + 1$$

$$\vec{u}(\vec{x},t) = \frac{1}{\rho(\vec{x},t)} \sum_i \vec{c}_i\, f_i^{\text{temp}}(\vec{x},t)$$

$$f_i^{\text{eq-shifted}}(\vec{x},t) = f_i^{\text{eq-shifted}}(\rho(\vec{x},t),\,\vec{u}(\vec{x},t)) = w_i\,\rho \cdot \left(\frac{(\vec{u}\circ\vec{c}_i)^2}{2\,c^4} - \frac{\vec{u}\circ\vec{u}}{2\,c^2} + \frac{\vec{u}\circ\vec{c}_i}{c^2} \right) + w_i\,(\rho-1)$$

$$f_i^{\text{B}}(\vec{x},\,t+\Delta t) = \left(1-\frac{\Delta t}{\tau}\right) f_i^{\text{temp}}(\vec{x},t) + \frac{\Delta t}{\tau}\, f_i^{\text{eq}}(\vec{x},t)$$



Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Floating-Point / Posit

0	01111111111	0000000000000000000000000000.. ..00000000000000000000000000000000

IEEE-754 FP64 (1 | 11 | 52)

0	01111111	00000000000000000000000

IEEE-754 FP32 (1 | 8 | 23)

0	01111	0000000000

IEEE-754 FP16 (1 | 5 | 10)

IEEE FP16 × 2^{-15}

0	11110	0000000000

FP16S (1 | 5 | 10)

entirely new

0	1111	00000000000

custom FP16C (1 | 4 | 11)

0	111111110	000000

posit $P16_0S$ (1 | n_r+1 | 14-n_r)

0	11110	1	000000000

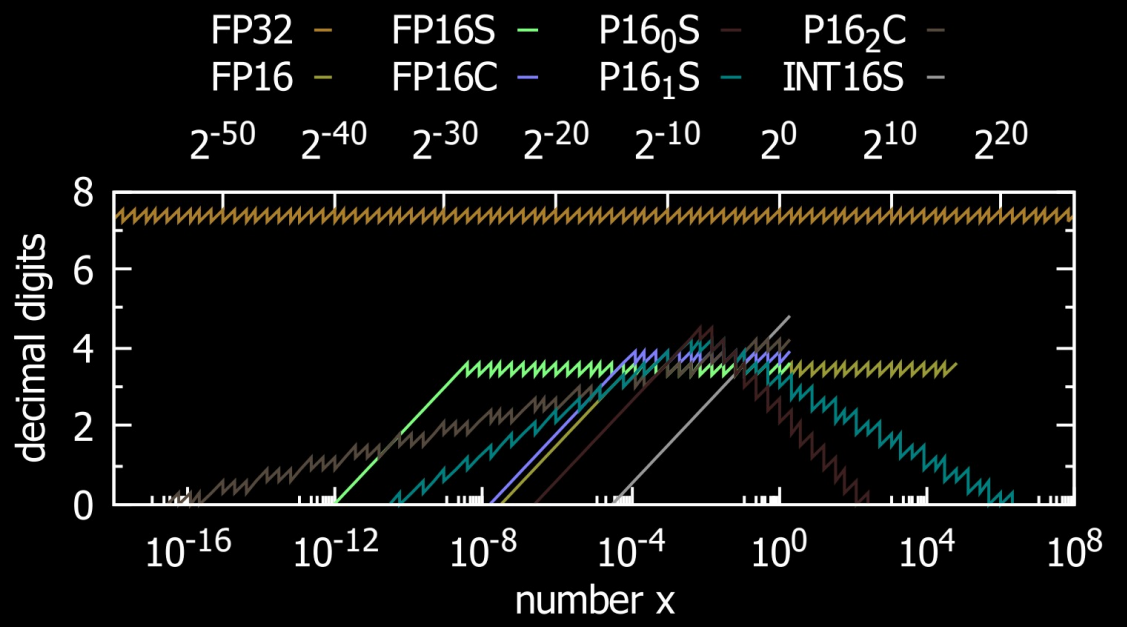
posit $P16_1S$ (1 | n_r+1 | 1 | 13-n_r)

0	1	11	000000000000

custom asymmetric posit $P16_2C$ (1 | n_r | 2 | 13-n_r)

0	1000000000000000

INT16S (1 | 15)



FP16S: more range towards tiny numbers

FP16C: halved truncation error

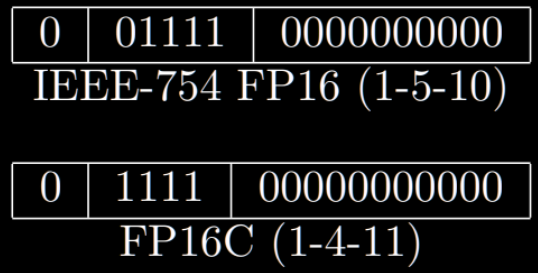
Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Custom FP16C format

- steal 1 bit from exponent to make mantissa larger
- tune exponent bias to have range ±2



- 3.6 digits instead of 3.3
- halved truncation error
  → superior accuracy & range towards small numbers

- conversion algorithms
  - 51 PTX instructions
  - correct rounding
  - both normals & denormals
  - any GPU/CPU can run it

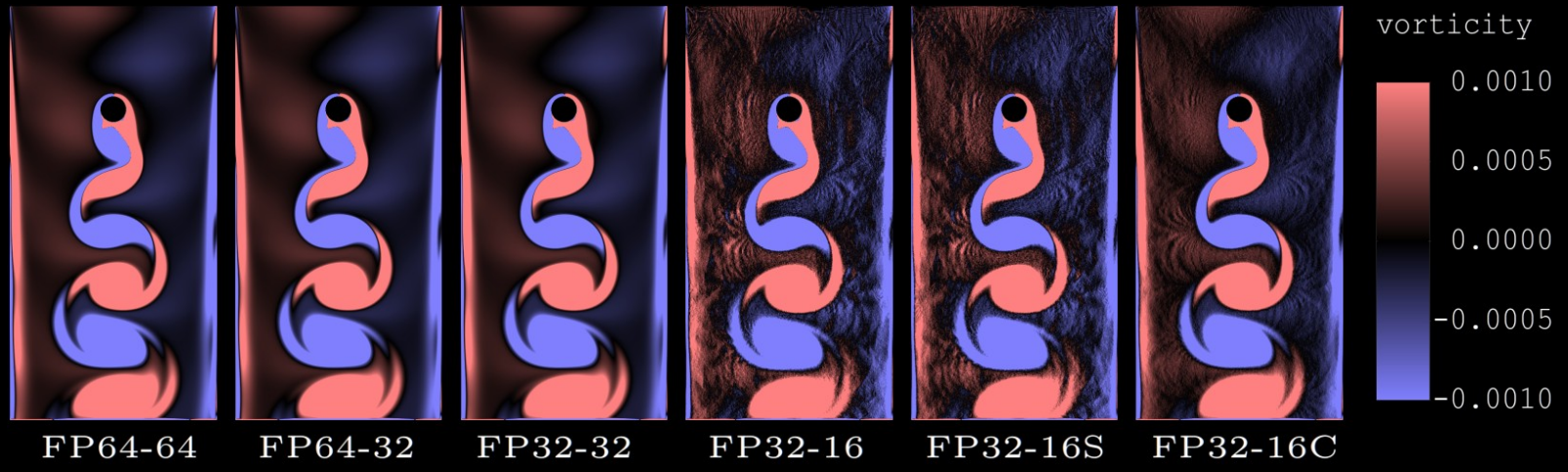
```
// custom 16-bit floating-point format, 1-4-11, exp-15, +-1.99951168, +-6.10351562E-5, +-2.98023224E-8, 3.612 digits
float half_to_float_custom(const ushort x) {
    const uint e = (x&0x7800)>>11; // exponent
    const uint m = (x&0x07FF)<<12; // mantissa
    const uint v = as_uint((float)m)>>23; // evil log2 bit hack to count leading zeros in denormalized format
    return as_float((x&0x8000)<<16 | (e!=0)*((e+112)<<23|m) | ((e==0)&(m!=0))*((v-37)<<23|((m<<(150-v))&0x007FF000))); // sign|normalized|denormalized
}
ushort float_to_half_custom(const float x) {
    const uint b = as_uint(x)+0x00000800; // round-to-nearest-even: add last bit after truncated mantissa
    const uint e = (b&0x7F800000)>>23; // exponent
    const uint m = b&0x007FFFFF; // mantissa; in line below: 0x7FF800=0x800000-0x000800 = decimal indicator flag - initial rounding
    return (b&0x80000000)>>16 | (e>112)*((((e-112)<<11)&0x7800)|m>>12) | ((e<113)&(e>100))*((((0x7FF800+m)>>(124-e))+1)>>1); // sign|normalized|denormalized
}
```

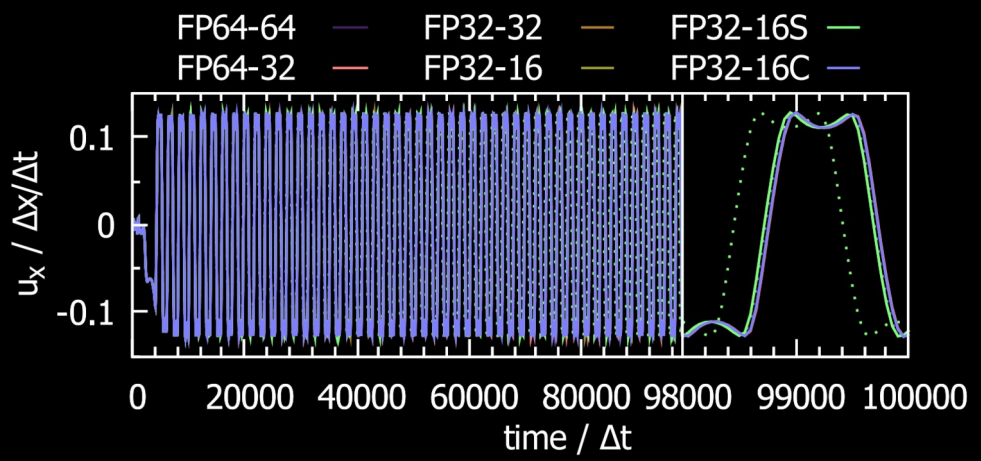
Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Test Setup 3/6: Karman Vortex Street





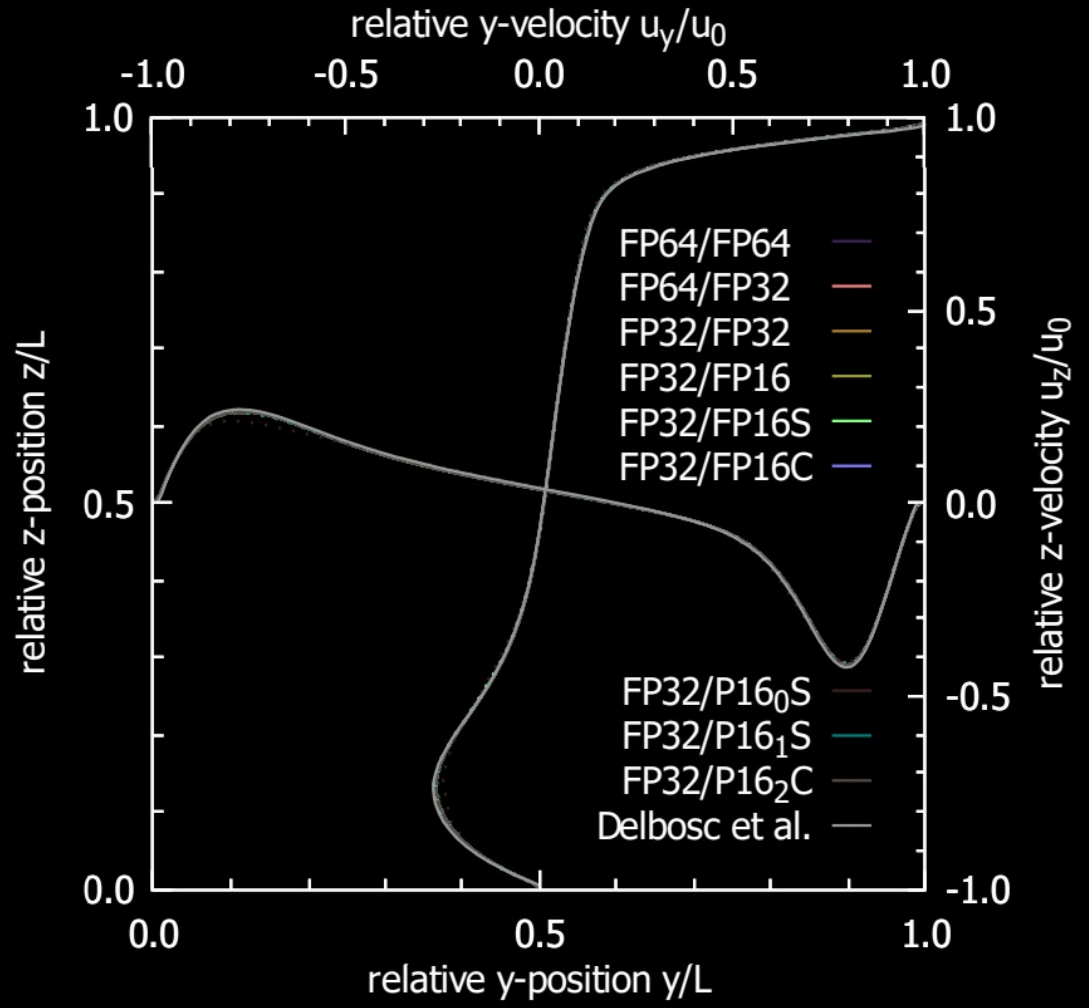
x-velocity at box center over time

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Test Setup 4/6: Lid-driven Cavity @ Re=1000



Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

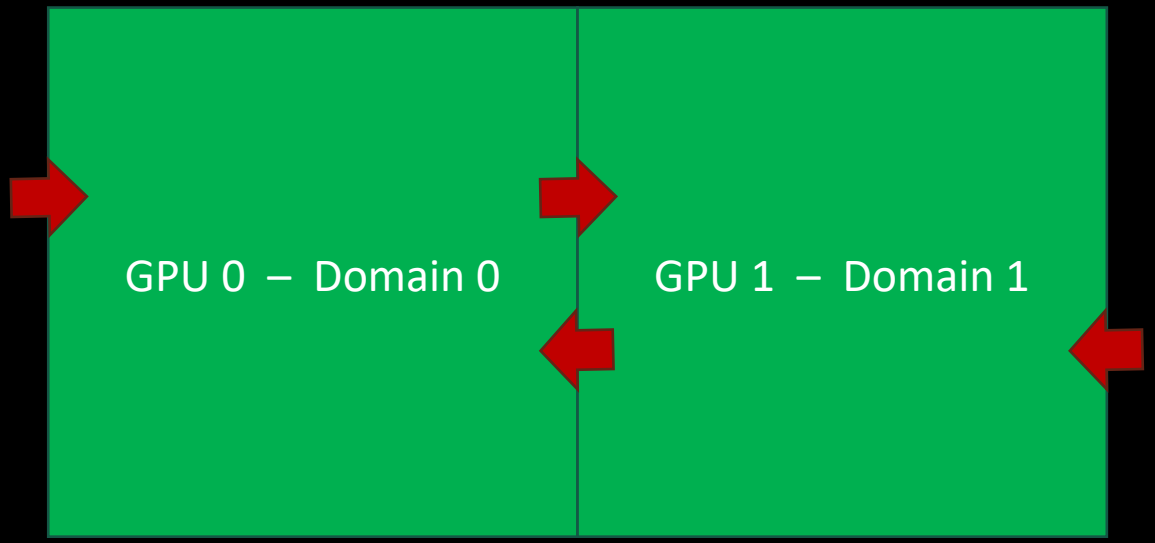
Part 3: Scaling Up: Multi-GPU

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Multi-GPU for LBM – Domain Decomposition

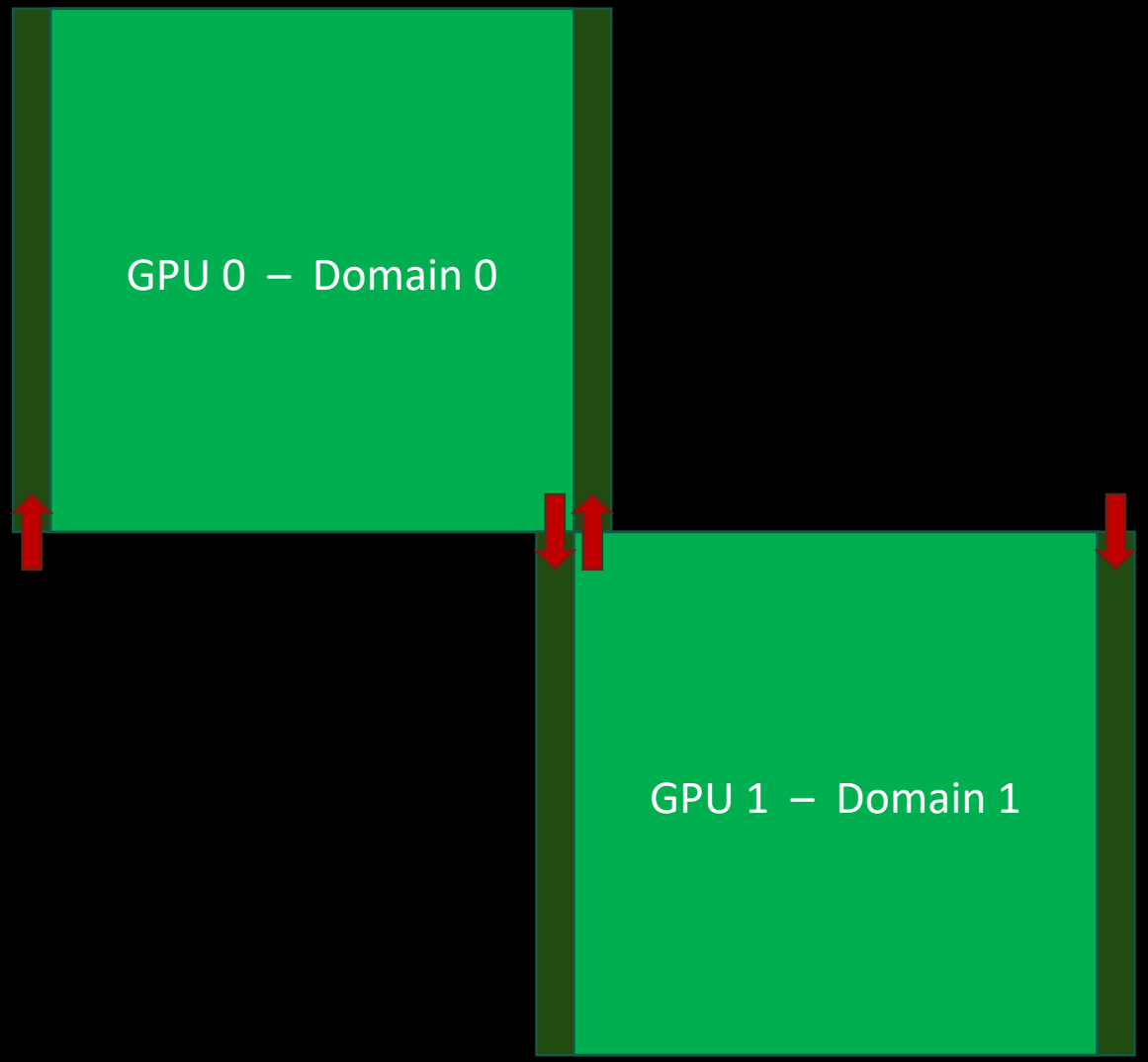


- split box into equal cuboid domains
- each GPU stores only its own domain in VRAM, doesn't know other domains
- at boundaries, they communicate (attention: periodic boundary conditions!)
- only outward-pointing DDFs need to be communicated (use lookup-tables for LBM indices)

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel UNIVERSITÄT BAYREUTH

Multi-GPU for LBM – Domain Decomposition



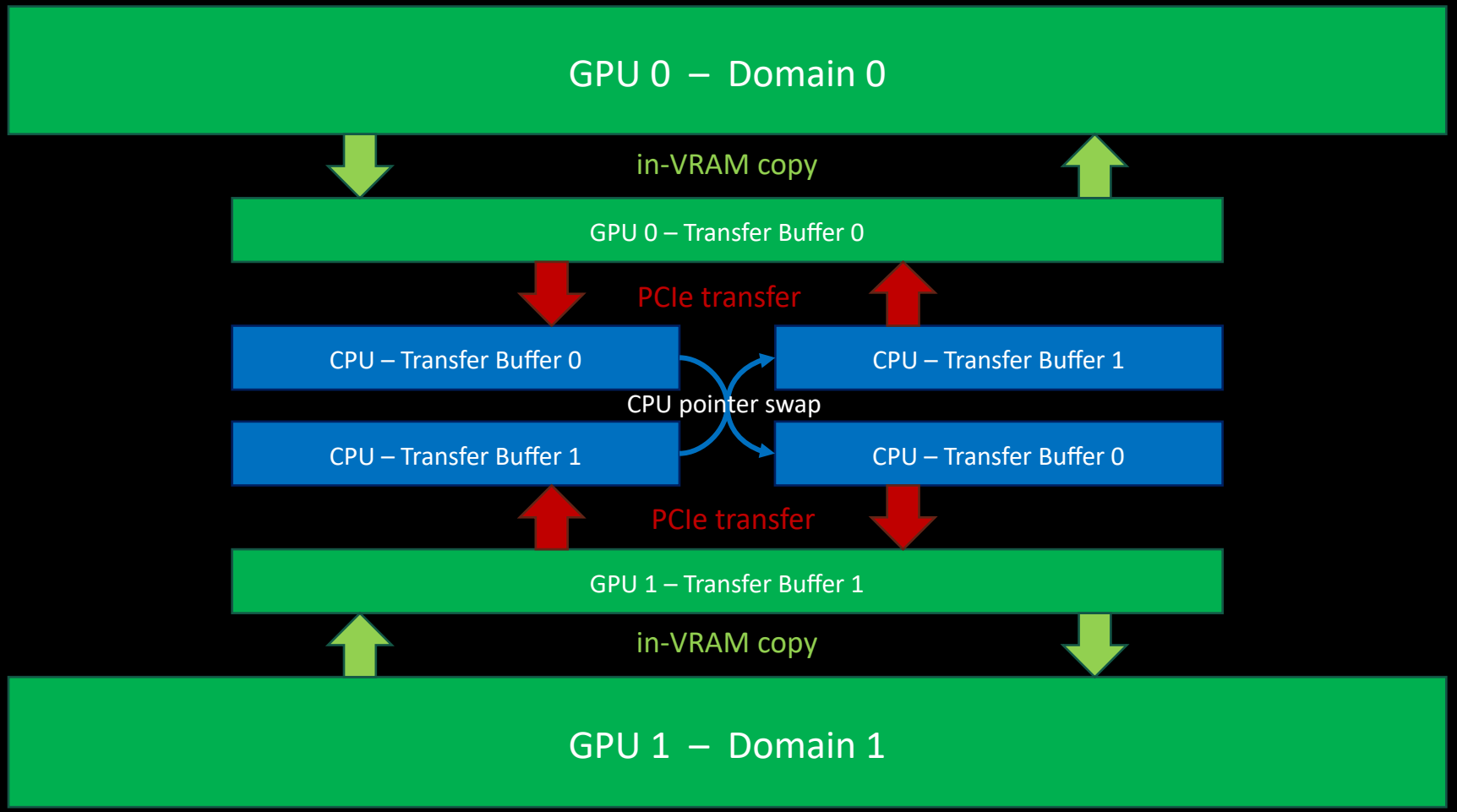
- 1 halo layer left and 1 halo layer right is needed for communication
- in 3D: 6 halo layers

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Single-node multi-GPU with OpenCL- simplified



Pros

- cross-vendor compatible with all hardware
- single GPU transfer buffers

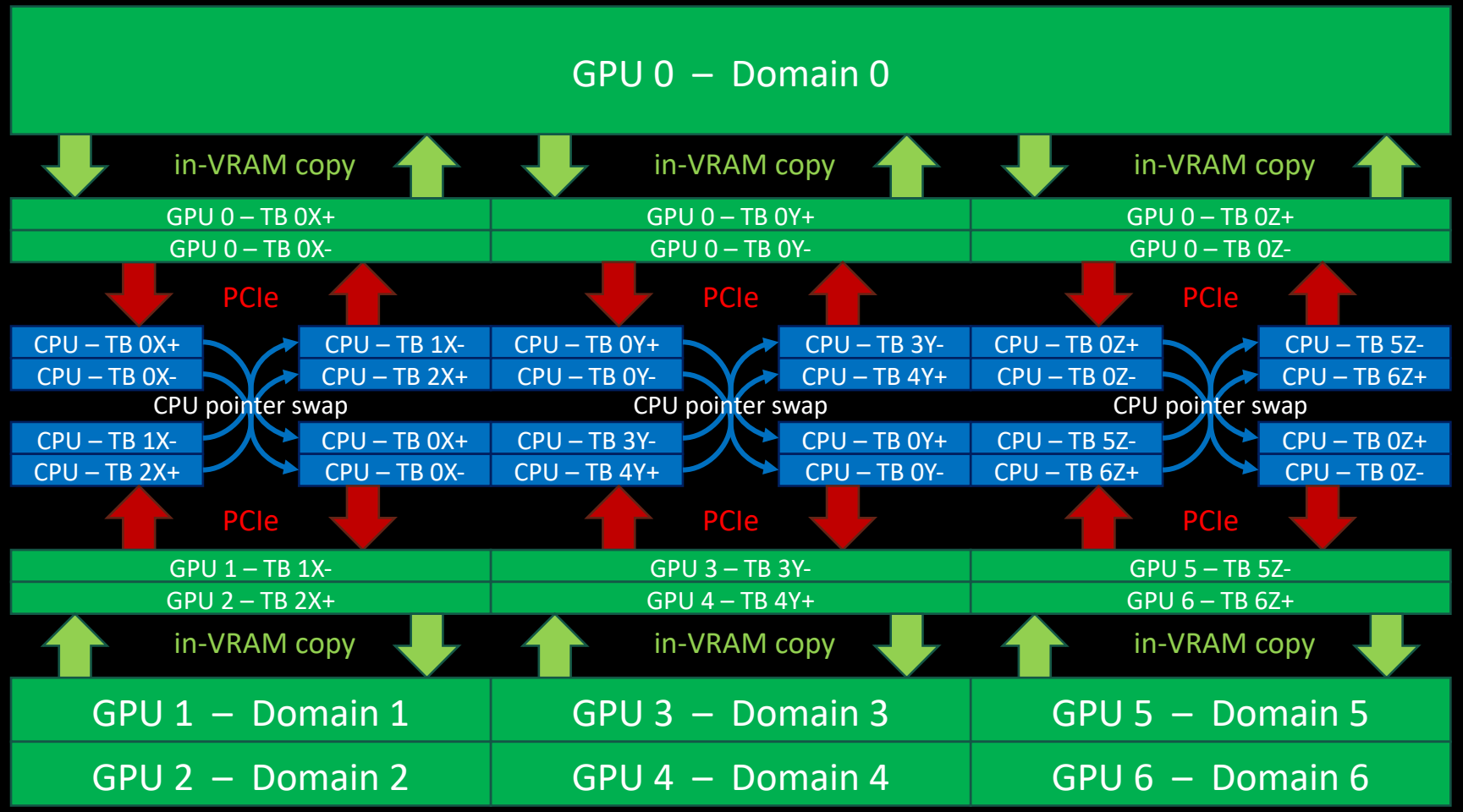
Cons

- single node only
- no GPU<->GPU P2P PCIe copy for transfer buffers

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel UNIVERSITÄT BAYREUTH

Single-node multi-GPU with OpenCL – 3D grid



- in 3D, GPU 0 may communicate with 6 other GPUs
- edge/corner cells are correctly handled via sequential x/y/z-transfer into halos

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

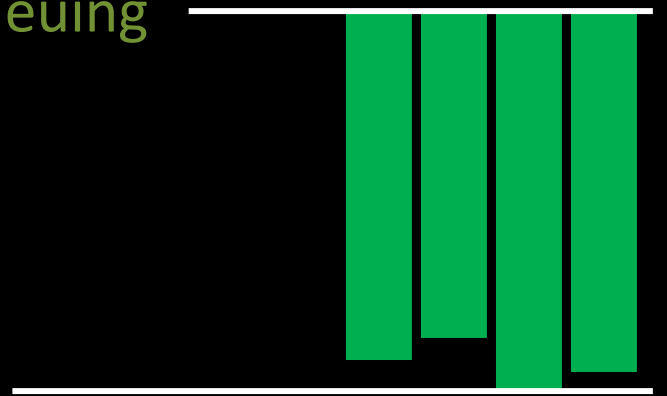
intel

UNIVERSITÄT BAYREUTH

Leverage non-blocking enqueuing for multi-GPU

```
for(int i=0; i<N; i++) {
    devices[i].enqueue_kernel(...); // non-blocking enqueuing
}


for(int i=0; i<N; i++) {
    devices[i].finish_queue(); // synchronization barrier
}
```



→ no OpenMP / std::threads / concurrency::parallel_for

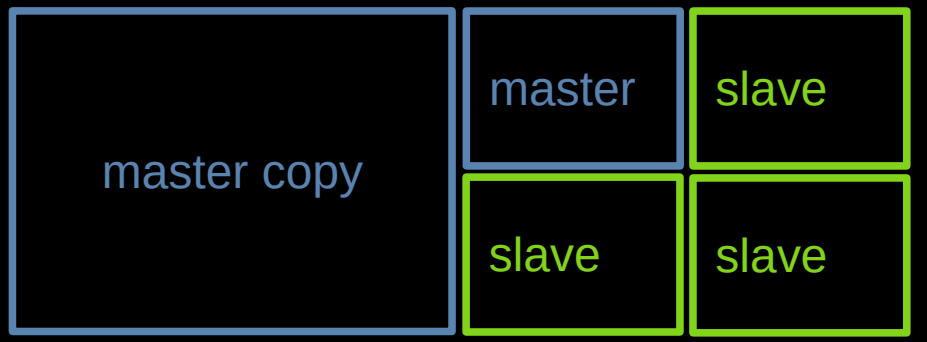
Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel UNIVERSITÄT BAYREUTH

Benefit of single-node: no master copy needed

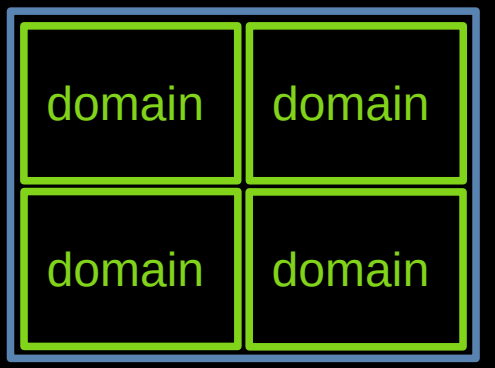
multi-node MPI model:

- 1 master, many slave domains
- master domain contains master copy of all slave domains



single-node model:

- all domains are the same
- master copy is assembled on-the-fly by stitching together domain memory arrays



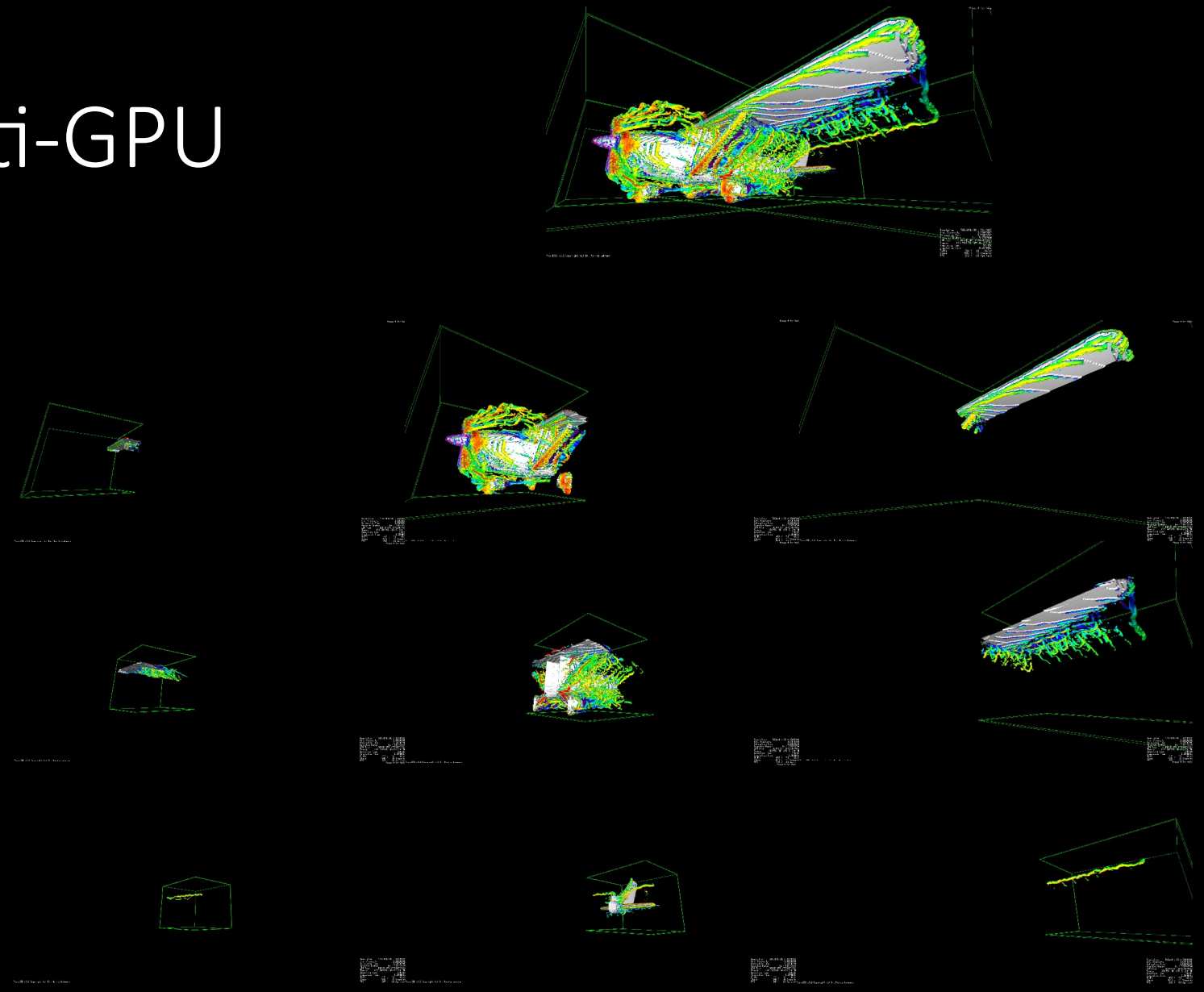
Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel UNIVERSITÄT BAYREUTH

Graphics with multi-GPU

- each GPU renders only its own domain, with 3D domain offset

  → image + zbuffer
- copy images + zbuffers from all GPUs to CPU
- CPU overlays all images pixel-by-pixel based on z-values
- CPU sends overlay image to screen

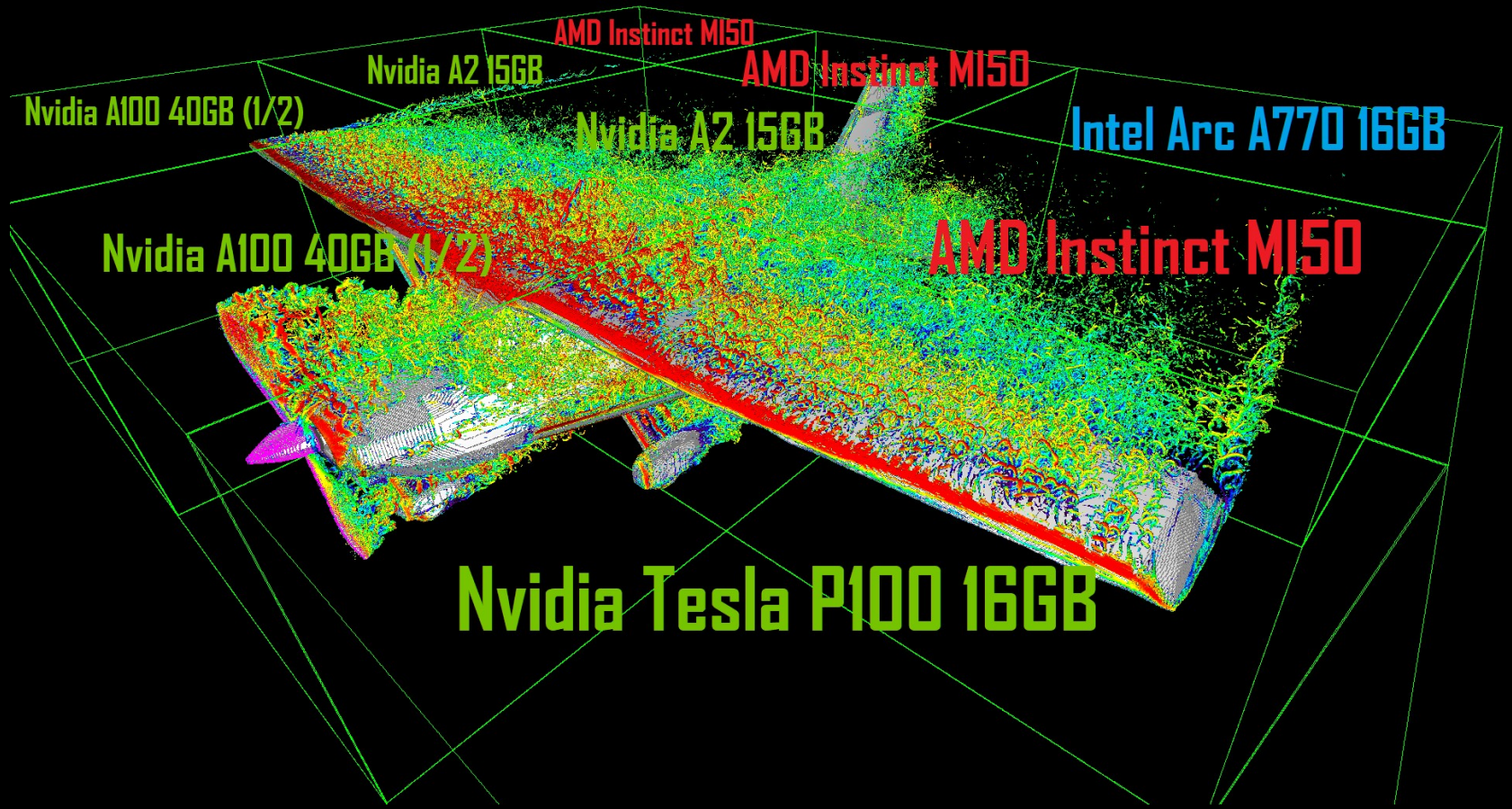


Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Cross-vendor multi-GPU with OpenCL - run Nvidia+AMD+Intel GPUs in “SLI”!



Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Part 4: Scaling Up: Big CPU

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Multi-TB memory CPU support

- GPU VRAM capacity is very limited (192GB)
- multi-GPU servers: max 2TB combined VRAM
- CPU servers: max 12TB RAM
- exciting: 6TB RAM (MRDIMMs) @ 1.7TB/s on dual-socket Xeon 6900P

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Multi-TB memory CPU support

Problem 1: 32-bit uint linear index overflows for $>2^{32}$ grid cells (~225 GB)

- always using 64-bit ulong causes slowdown on GPUs
- solution: switchable macro data type uxx for uint/ulong
  - OpenCL C is compiled at runtime
  - if a setup uses $>2^{32}$ grid cells, switch from 32-bit to 64-bit indexing before compiling OpenCL kernels

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

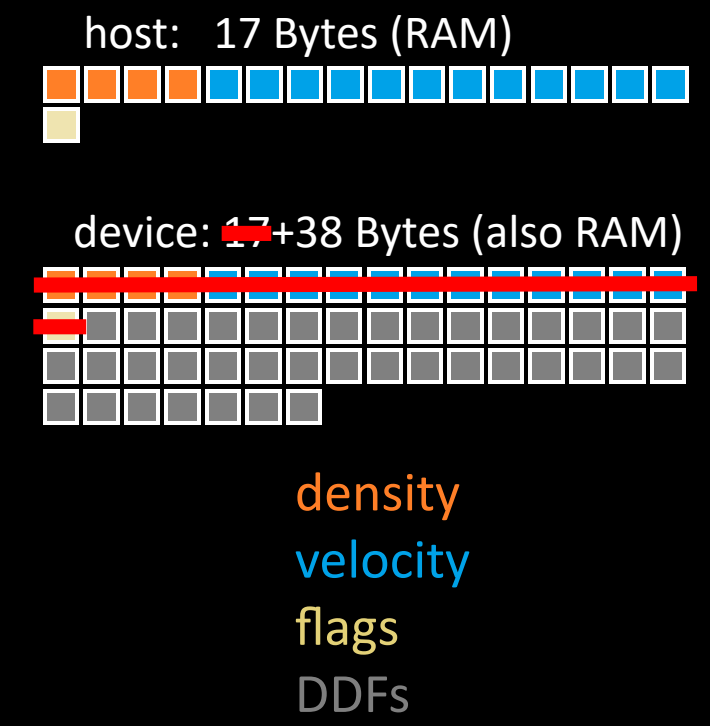
intel

UNIVERSITÄT BAYREUTH

Multi-TB memory CPU support

Problem 2: some host+device buffers are duplicate in RAM

- solution: use zero-copy buffers on CPUs/iGPUs
- requirements:
  - CL_MEM_USE_HOST_PTR flag for cl::Buffer
  - align host pointer to 4096 Bytes (cache line boundary)
  - pad host pointer allocation so cl::Buffer size is multiple of 64 Bytes



Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH

Multi-TB memory CPU support

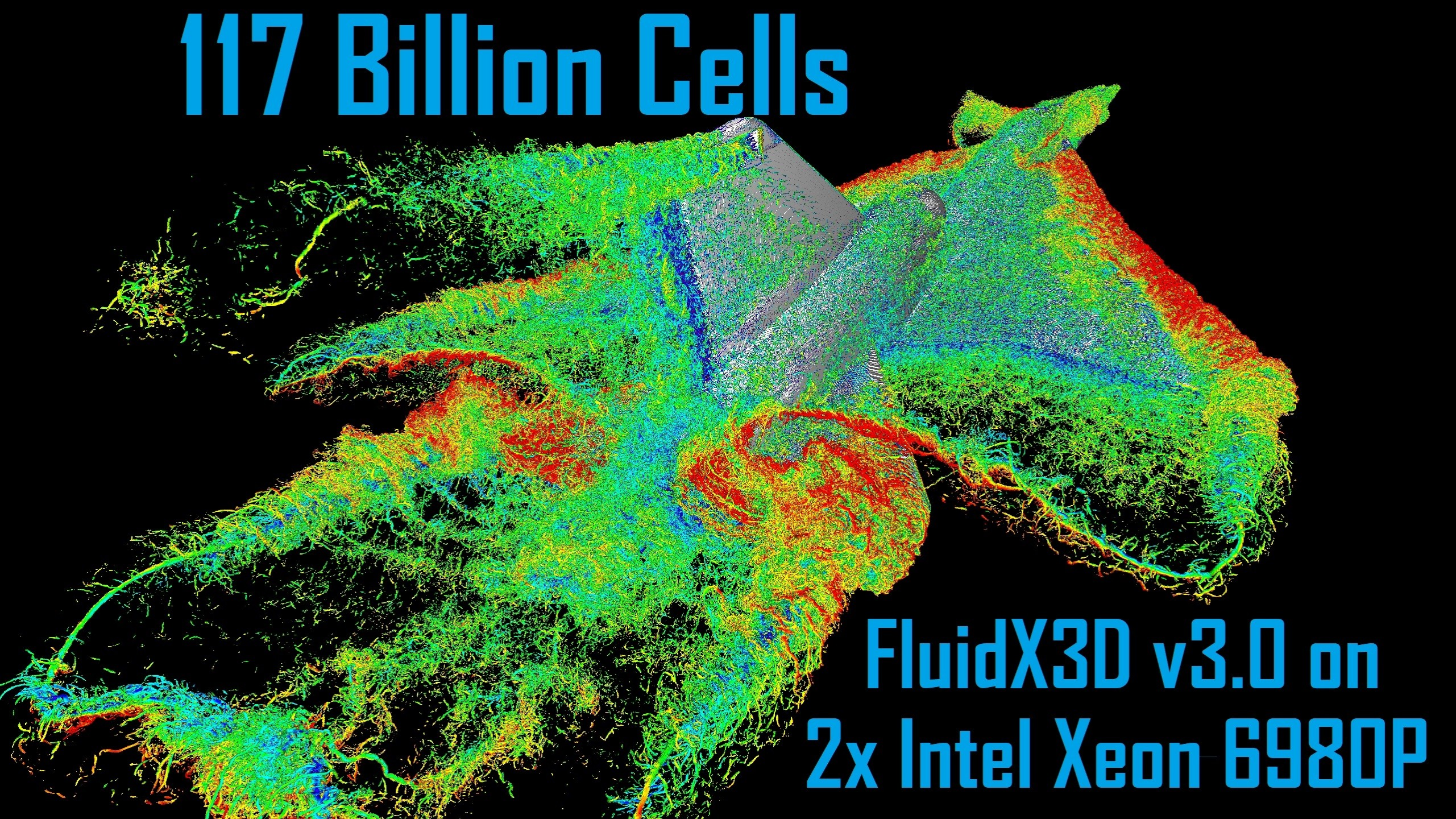
Problem 3: danger zone 

- noone before me has ever dispatched a single OpenCL kernel with >100 Billion threads
- AVX2/AVX512 vectorization was broken for 64-bit indexing
  - found the right colleague at Intel
  - they fixed it

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel UNIVERSITÄT BAYREUTH

117 Billion Cells



FluidX3D v3.0 on 2x Intel Xeon 6980P

You have the power to change the GPGPU compute landscape

- OpenCL standard is not set in stone; have suggestions?
  → Khronos Group OpenCL Advisory Panel
- vendor OpenCL implementations still have some bugs
  → report them, vendors are eager to fix bugs!
- GPGPU compute landscape is made by people
  → you have the power to change it to the better!
  → choose OpenCL/SYCL over proprietary CUDA/HIP

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel UNIVERSITÄT BAYREUTH

Questions

Dr. Moritz Lehmann | dr.moritz.lehmann@gmail.com
linkedin.com/in/dr-moritz-lehmann | github.com/ProjectPhysX

intel

UNIVERSITÄT BAYREUTH