NVIDIA HPC STANDARD LANGUAGE PARALLELISM, C++

MATT STACK

HPC PROGRAMMING IN ISO C++

```
std::sort(std::execution::par, c.begin(), c.end());

std::for_each(std::execution::par, c.begin(), c.end(), func);
```

- Introduced in C++17
- Parallel and vector concurrency via execution policies

_unseq, std::execution::seq

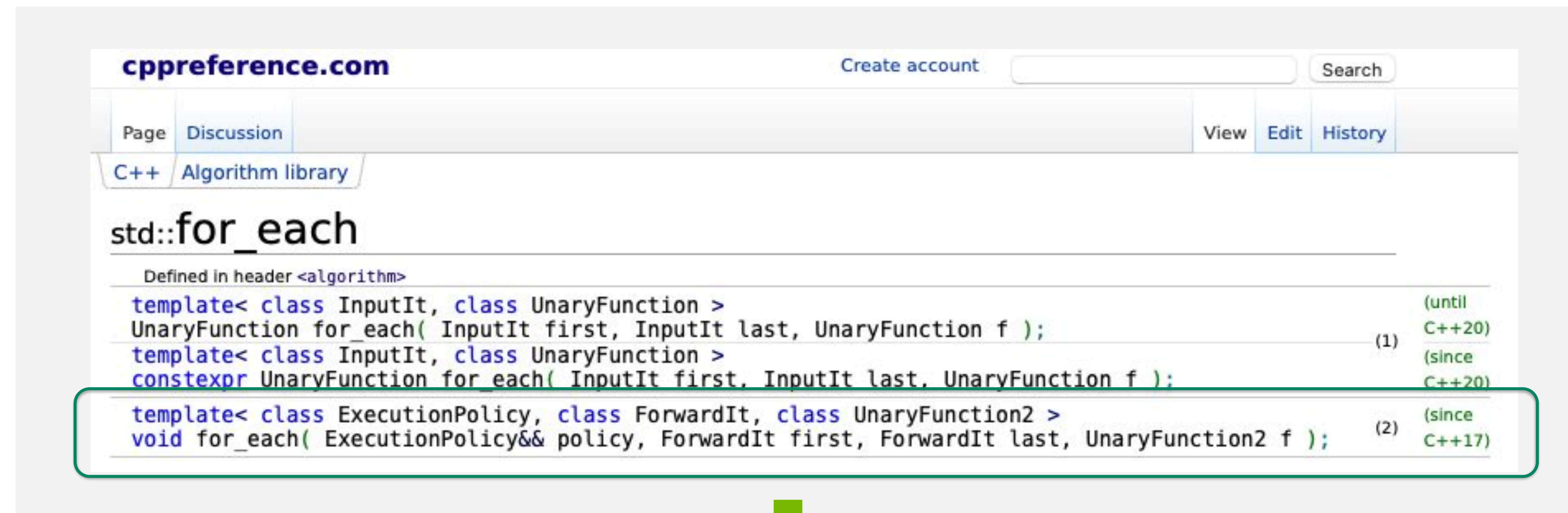
er when calling algorithms

acceleration of

```
Aside: Cuda Unified Memory
float *x, *y;

 // Allocate Unified Memory -- accessible from
CPU or GPU
 cudaMallocManaged(&x, N*sizeof(float));
```

USING C++ STDPAR




```
#include <algorithm> // std::for_each and other functions
#include <execution> // seq, par, par_unseq, un_seq
. . .
std::vector<double> vec = . . .
std::for_each(std::execution::par, vec.begin(), vec.end(), [=](auto i){
        . . . // doing work for each element in the vector
});
```


NVIDIA

C++ PARALLEL ALGORITHMS

- When using the parallel execution policy, make sure there are no data races or deadlocks

- StdPar execution on GPU leverages CUDA Unified Memory
  - data needs to reside in heap memory
    - std::vector works
    - std::array does not

- Unlike CUDA C++, functions do not need the __device__ annotation

- Execution on GPU requires random access iterators

- To compile using StdPar, use the –stdpar flag
  - nvc++ -stdpar ./file.cpp
    - -stdpar currently has two options, -stdpar=gpu (which is the default when not given an option) for parallel execution on GPU, and –stdpar=multicore for parallel execution on CPU

C++ PARALLEL ALGORITHMS

Problem: There is a std::vector I want to sort

```
std::vector<int> vec1;
```

Solution: Using standard algorithm std::sort

```
std::sort(vec1.begin(), vec1.end());
```

Potential Performance Improvement: Using parallel execution and –stdpar to offload work and data to GPU

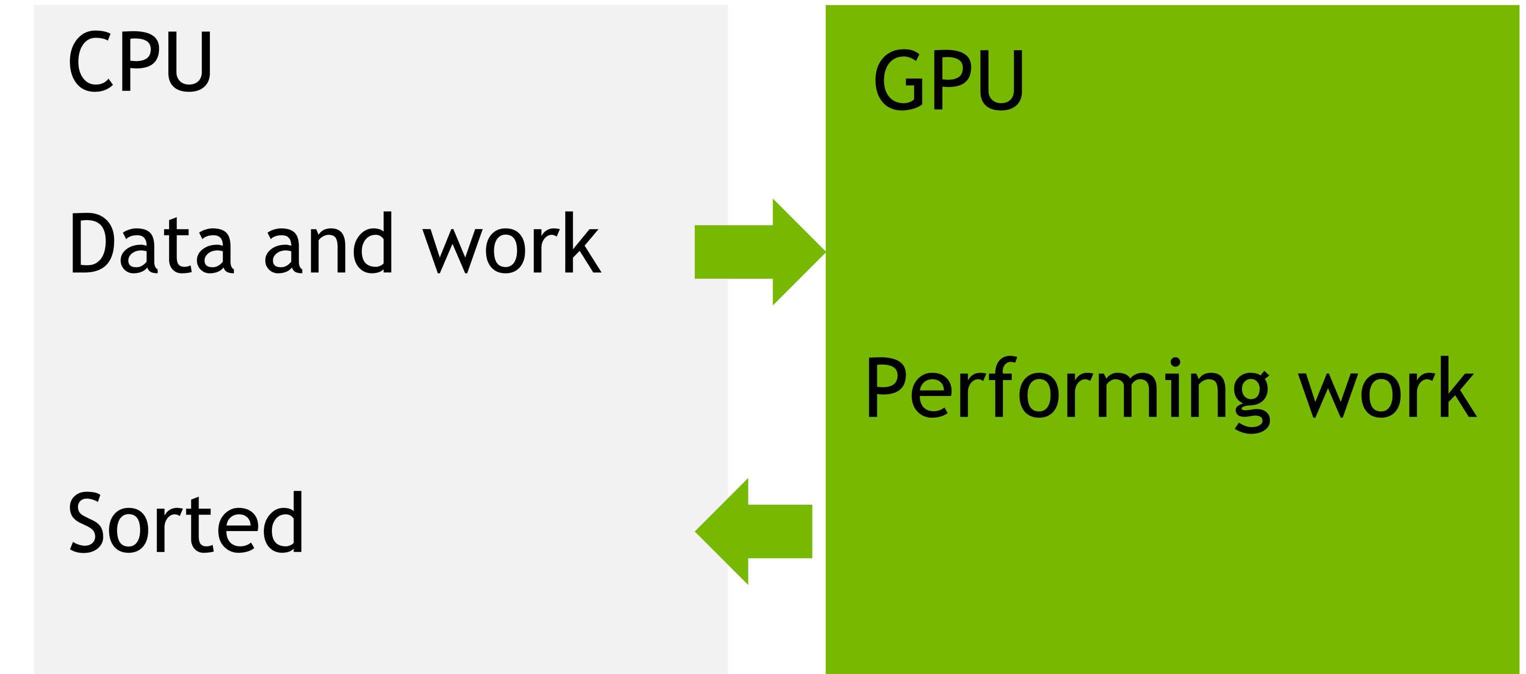
```
std::sort(std::execution::par, vec1.begin(), vec1.end());
```

-during compile-

```
nvc++ -stdpar=gpu ./main.cpp
```

{1, 9, 2, 8, 3, 7, 4, 6, 5, 0}

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}



NVIDIA.

C++ PARALLEL ALGORITHMS

std::Sort

```
std::sort(std::execution::par,
        employees.begin(), employees.end(),
        CompareByLastName());
```

std::TransformReduce

```
int ave_age =
  std::transform_reduce(std::execution::par_unseq,
                        employees.begin(), employees.end(),
                        0, std::plus<int>(),
                        [](const Employee& emp){
                            return emp.age();
                        })
  / employees.size();
```

STLBM

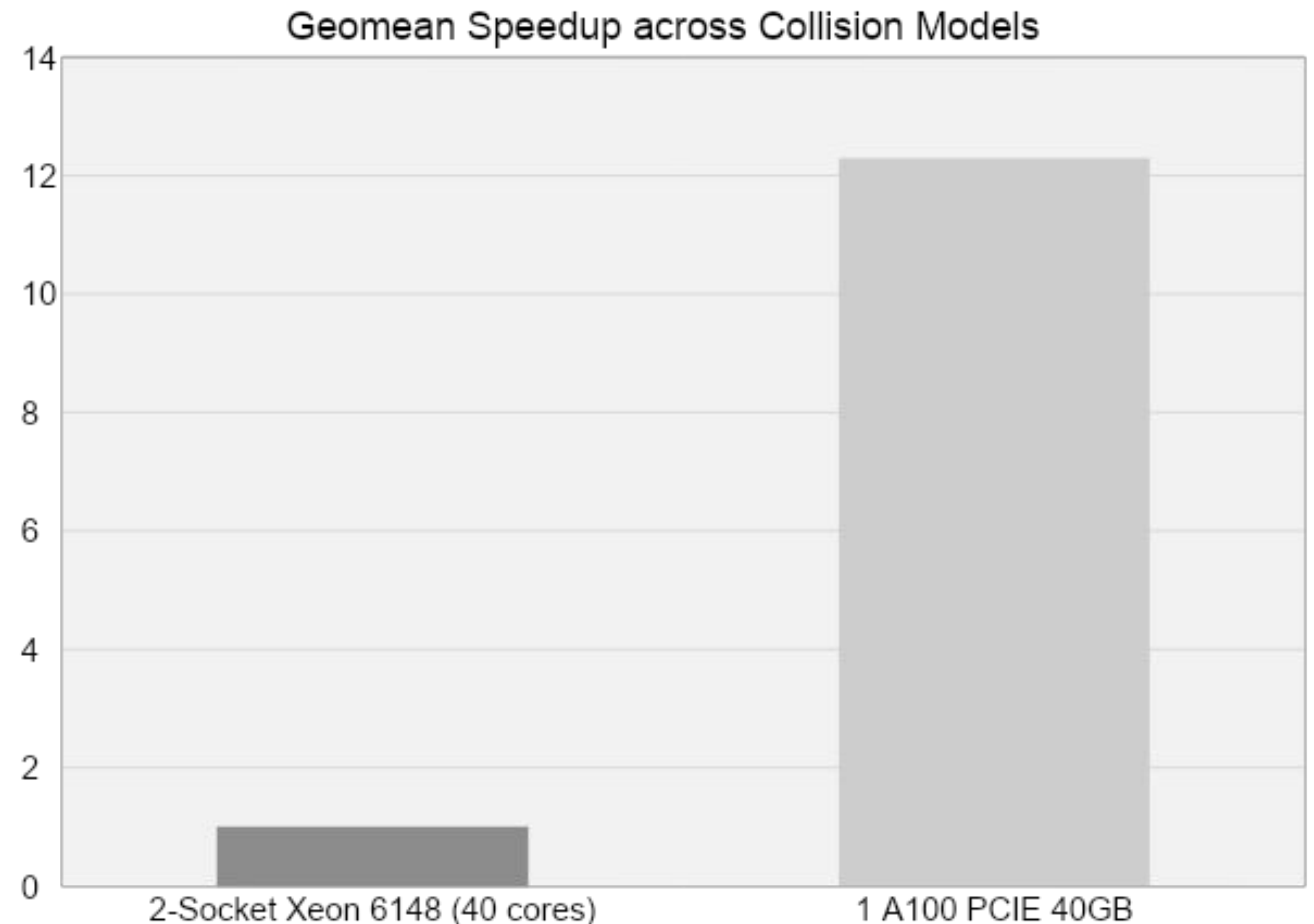
Many-core Lattice Boltzmann with C++ Parallel Algorithms

- Framework for parallel lattice-Boltzmann simulations on multiple platforms, including many-core CPUs and GPUs
- Implemented with C++17 standard (Parallel Algorithms) to achieve parallel efficiency
- No language extensions, external libraries, vendor-specific code annotations, or pre-compilation steps

*"We have with delight discovered the NVIDIA "stdpar" implementation of C++ Parallel Algorithms. … We believe that the result produces state-of-the-art performance, is highly didactical, and introduces **a paradigm shift in cross-platform CPU/GPU programming** in the community.“*

-- Professor Jonas Latt, University of Geneva

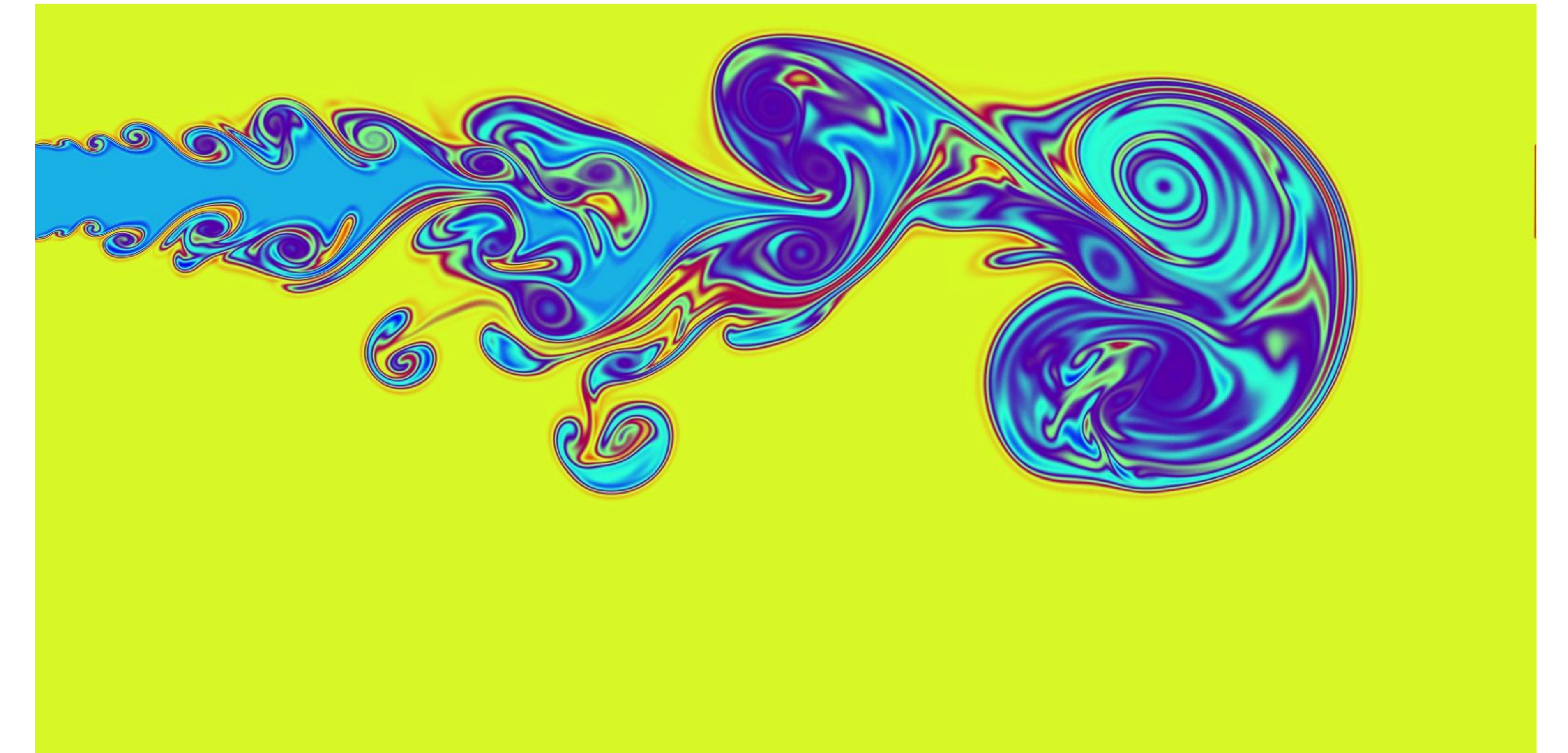
https://gitlab.com/unigehpfs/stlbm
https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s32076/
GTC Fall Session A31329



NVIDIA

Example- Miniweather

https://github.com/mrnorman/miniWeather/



```
!Compute fluxes in the x-direction for each cell
 do concurrent (k=1:nz, i=1:nx+1) local(d3_vals,vals,stencil,ll,s,r,u,t,p,w)
   !Use fourth-order interpolation from four cell averages to compute the value at the interface in question
   do ll = 1 , NUM_VARS
     do s = 1 , sten_size
       stencil(s) = state(i-hs-1+s,k,ll)
     enddo
     !Fourth-order-accurate interpolation of the state
     vals(ll) = -stencil(1)/12 + 7*stencil(2)/12 + 7*stencil(3)/12 - stencil(4)/12
     !First-order-accurate interpolation of the third spatial derivative of the state (for artificial viscosity)
     d3_vals(ll) = -stencil(1) + 3*stencil(2) - 3*stencil(3) + stencil(4)
   enddo
```

NVIDIA

Tips and Tricks

Using -Minfo for compile time info

```
nvc++ -stdpar=gpu -Minfo=stdpar --std=c++20 test.cpp
main:
    13, stdpar: Generating NVIDIA GPU code
        13, std::for_each with std::execution::par_unseq policy parallelized on GPU
```

Use std::Views::iota in C++20 for easy iterator

```
auto v = std::views::iota(0, 9);

  std::for_each(std::execution::par_unseq, v.begin(), v.end(),
      [=](int i){
        printf("%d, ", threadIdx.x);
        printf("%d, ", blockIdx.x);
      });
```

More Reading and NVFORTRAN

NVFOTRAN

https://www.youtube.com/watch?v=KhZvrF_w1ak

In depth discussion on Unified memory

https://vimeo.com/431616420

NVIDIA