
We enhance excellent research for a better world.
NRIS
Norwegian research infrastructure services.

Scientific programming for GPUs

Jørgen Nordmoen - GPU Team lead

Agenda for today

- Introduction to GPUs
  - Terminology
  - Main differences between CPU and GPU
- GPU programming with OpenMP offloading
  - Introduction to OpenMP offloading syntax
    - Example: Vector addition
  - Importance of data handling with GPUs
    - Example: Jacobi iteration
    - Simple profiling with Nvidia NSight Systems
- A broader perspective on GPU programming
  - Alternative technologies
  - Portability concerns
  - Future HPC systems
- Questions

- 10 minutes
- 40 minutes
- 10 minutes
- 15 minutes

Introduction to GPUs

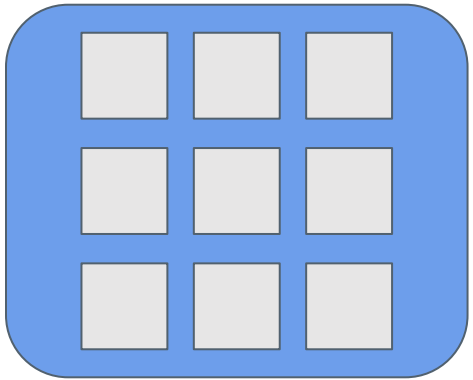
What is a GPU?

- **G**raphics **P**rocessing **U**nit
  - Used to manipulate pixels on screen
  - Image processing
  - Accelerator card
- Found in most computing devices
  - Mobile phones, laptops and workstations
- We will focus on GPGPU
  - **G**eneral **P**urpose computing on **G**raphics **P**rocessing **U**nits
  - Using GPUs to perform scientific calculations

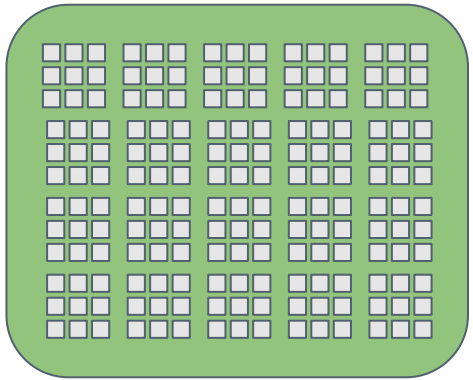
Nvidia A100

Inner workings of a GPU

- What makes a GPU special
  - Massive SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata) architecture
    - In reality **S**ingle **P**rogram **M**ultiple **T**hreads (SPMT) + SIMD
  - >> 1000 cores vs <= 128 cores on CPU
  - Large memory bandwidth

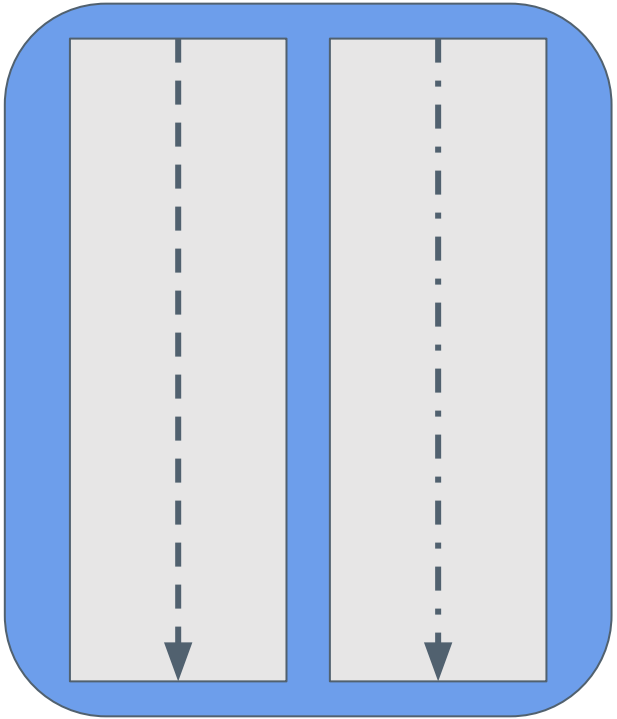


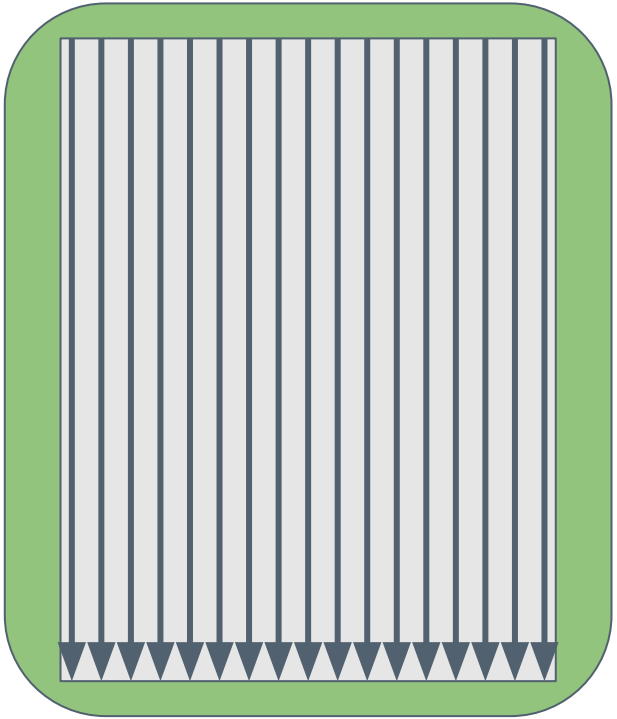
CPU



GPU

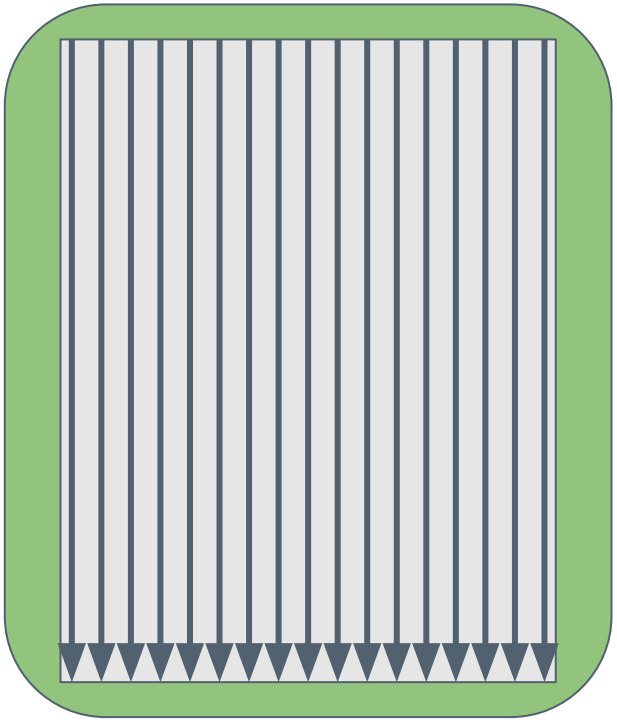
Inner workings of a GPU





Inner workings of a GPU

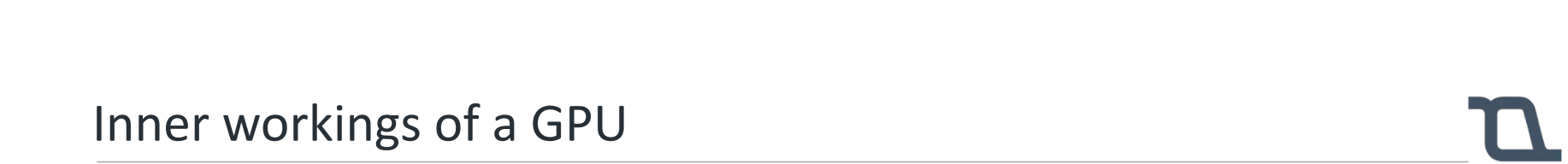
- Minimum 32 / 64 threads* (Nvidia / AMD)
- Lock-step execution
  - SIMD-like
- Limited atomic memory between thread groups
  - Limited synchronization
  - Optimize for SIMD



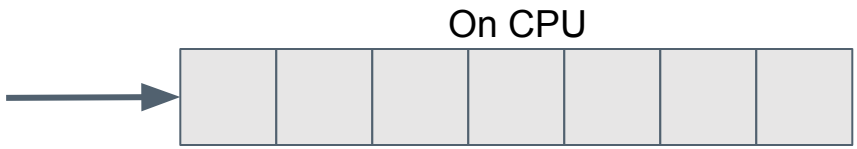
GPU

* Intel has 8/16/32 thread support

Inner workings of a GPU

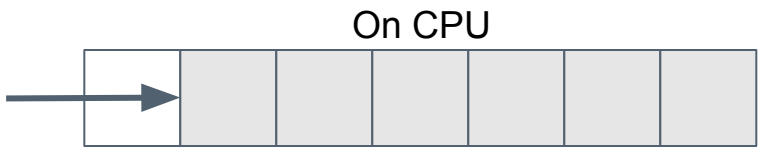


```
float data[100];
for (int i = 0; i < 100; i++) {
        data[i] = i * i + 10;
}
```



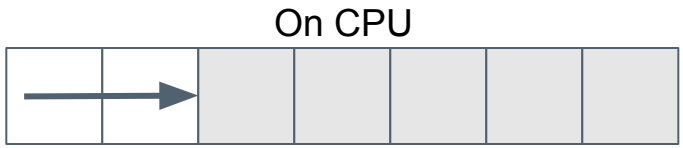
Inner workings of a GPU

```
float data[100];
for (int i = 0; i < 100; i++) {
        data[i] = i * i + 10;
}
```



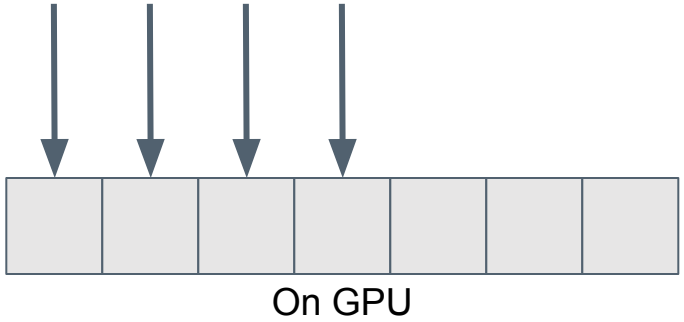
Inner workings of a GPU

```
float data[100];
for (int i = 0; i < 100; i++) {
        data[i] = i * i + 10;
}
```



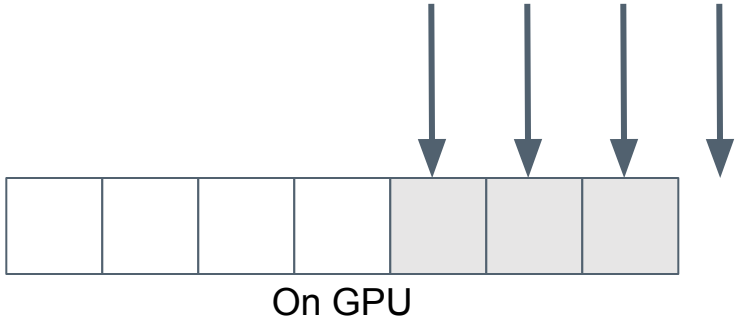
Inner workings of a GPU

```
float data[100];
#pragma acc kernels
for (int i = 0; i < 100; i++) {
        data[i] = i * i + 10;
}
```



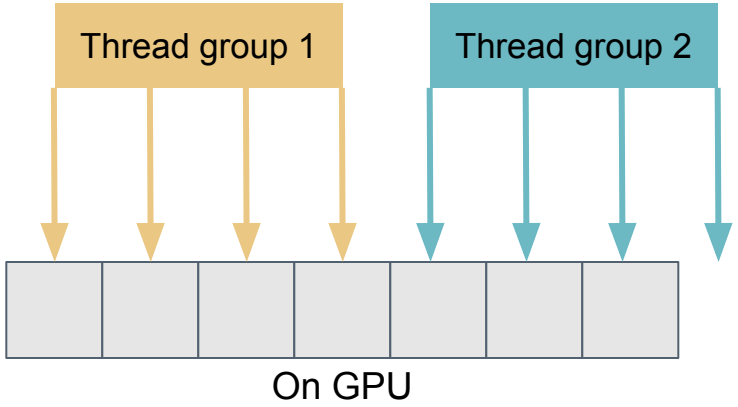
Inner workings of a GPU

```
float data[100];
#pragma acc kernels
for (int i = 0; i < 100; i++) {
        data[i] = i * i + 10;
}
```



Inner workings of a GPU

```
float data[100];
#pragma acc kernels
for (int i = 0; i < 100; i++) {
        data[i] = i * i + 10;
}
```



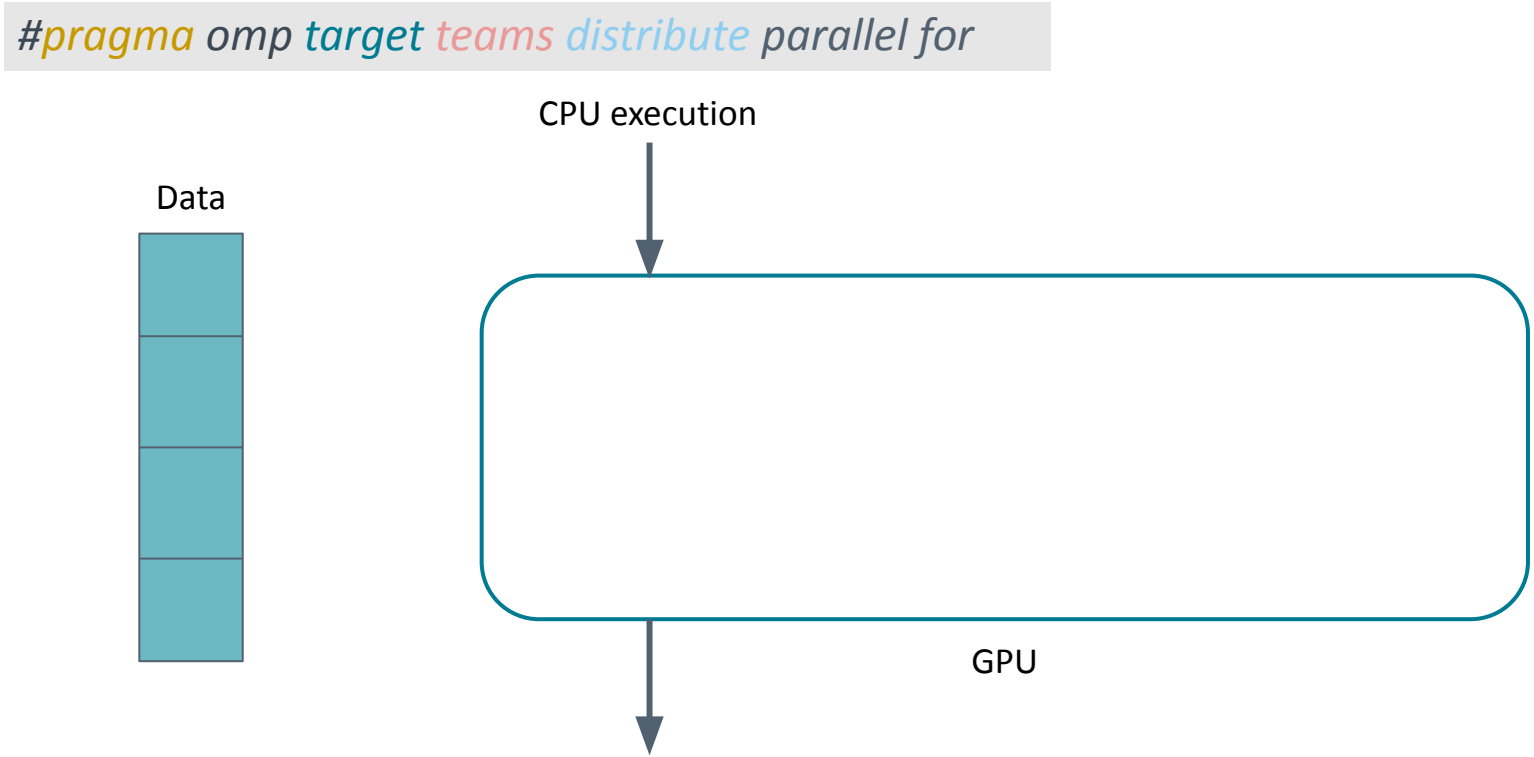
GPU programming with OpenMP offloading

Vector addition with OpenMP offloading

- Vector addition i C
- Vector addition with OpenMP shared memory parallelization
- Vector addition with OpenMP offloading

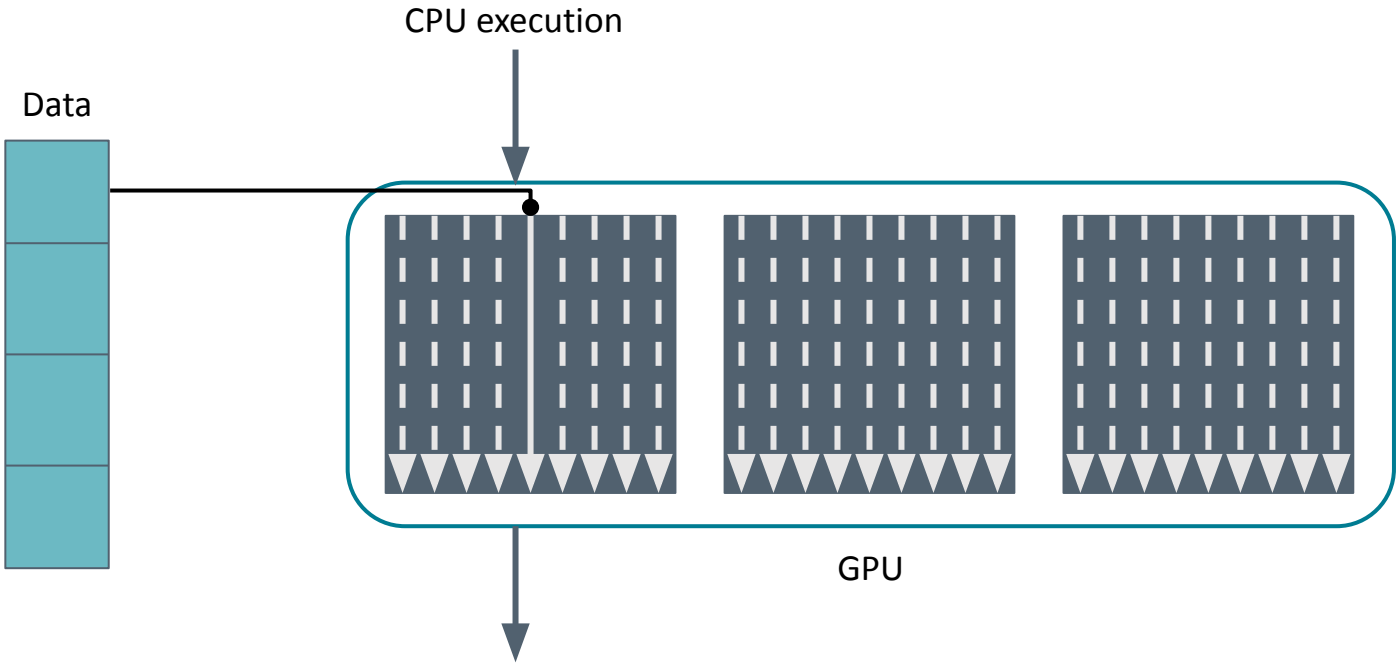
- Vector addition in CUDA

Vector addition with OpenMP offloading



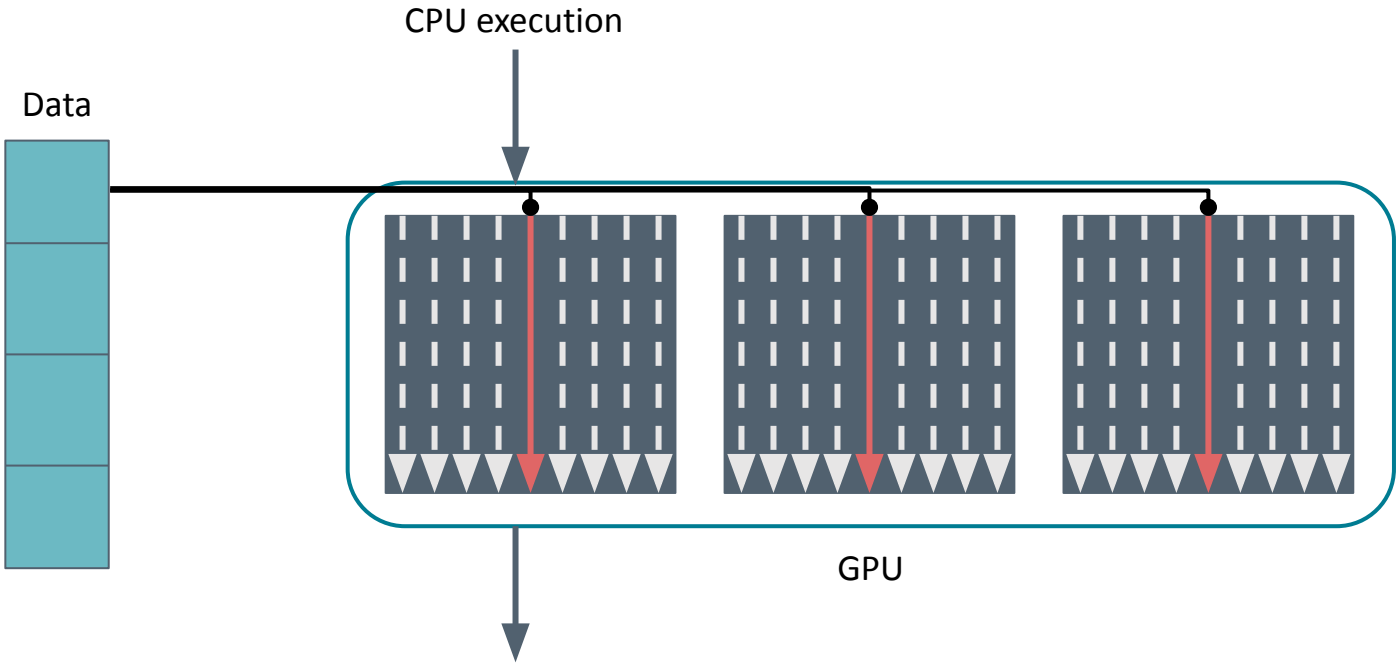
Vector addition with OpenMP offloading

```
#pragma omp target
```



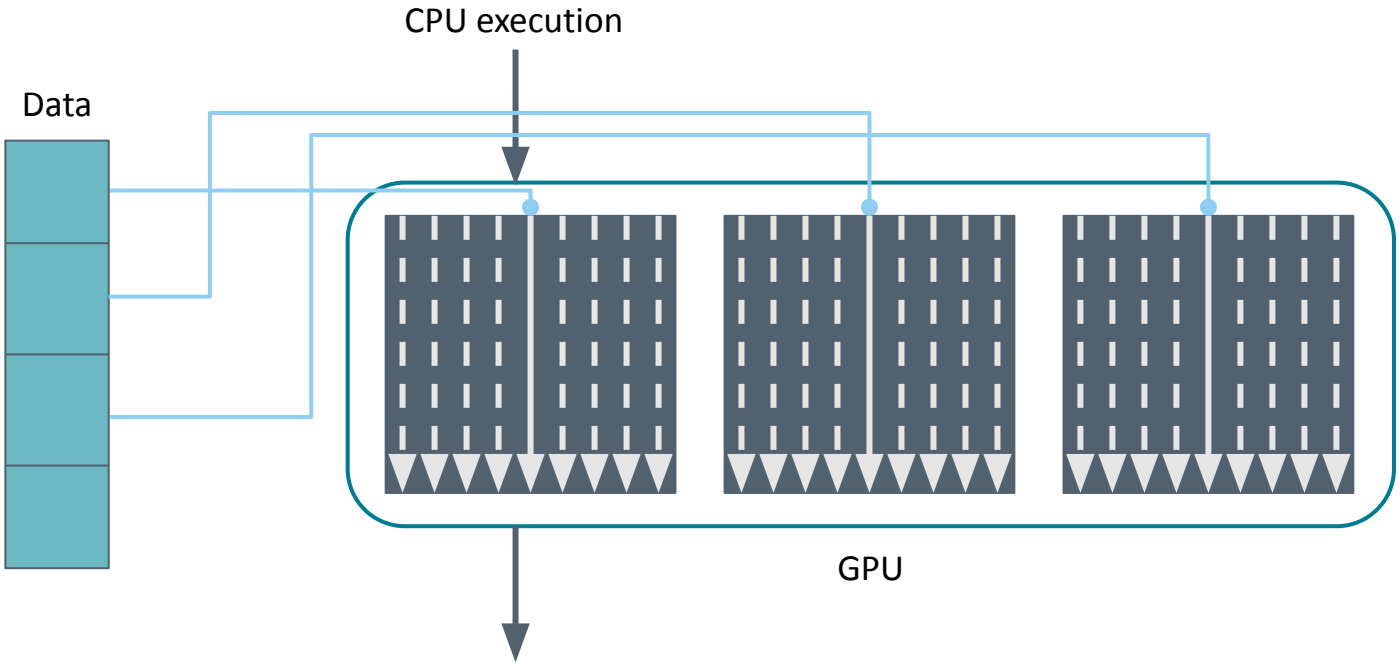
Vector addition with OpenMP offloading

```
#pragma omp target teams
```

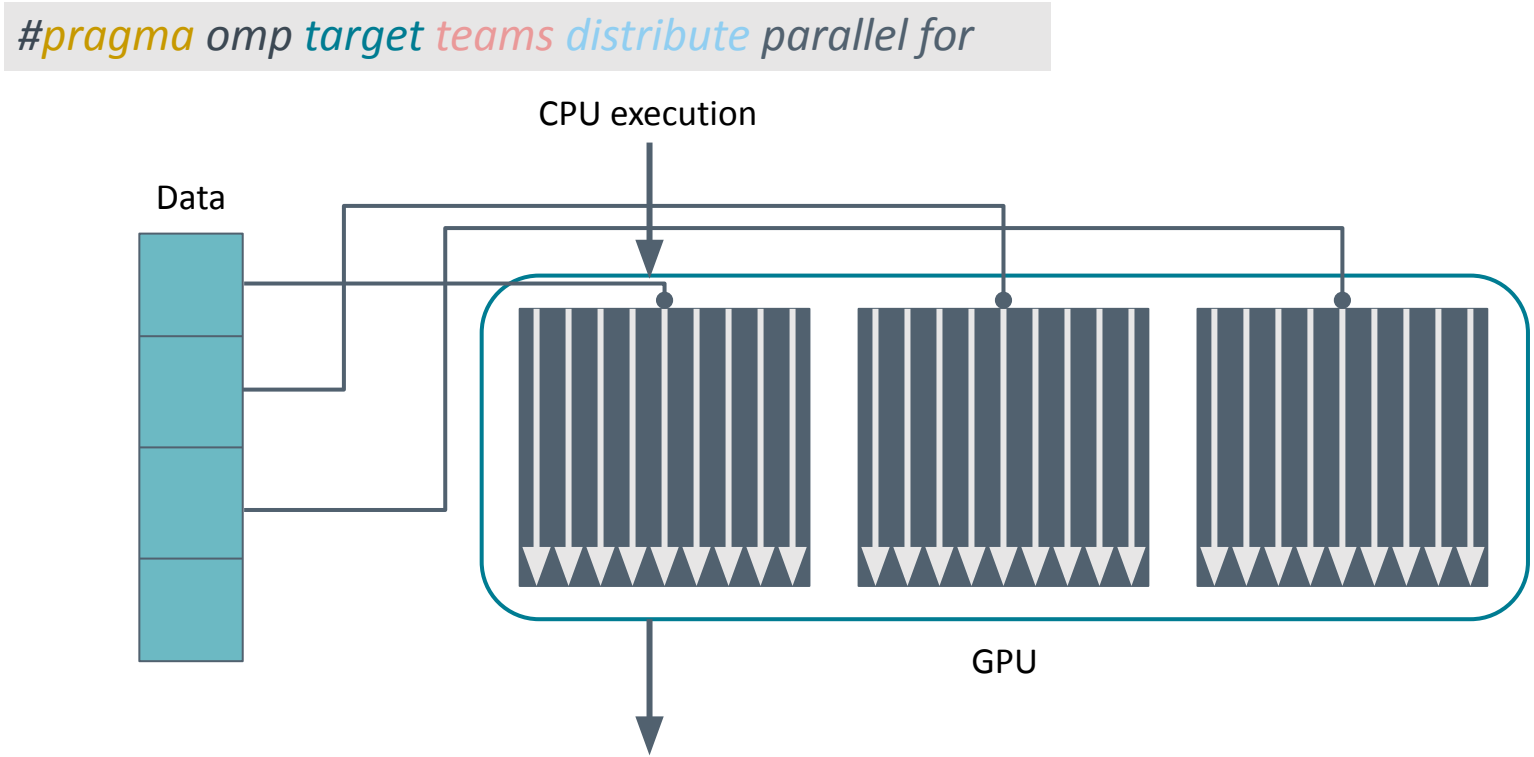


Vector addition with OpenMP offloading

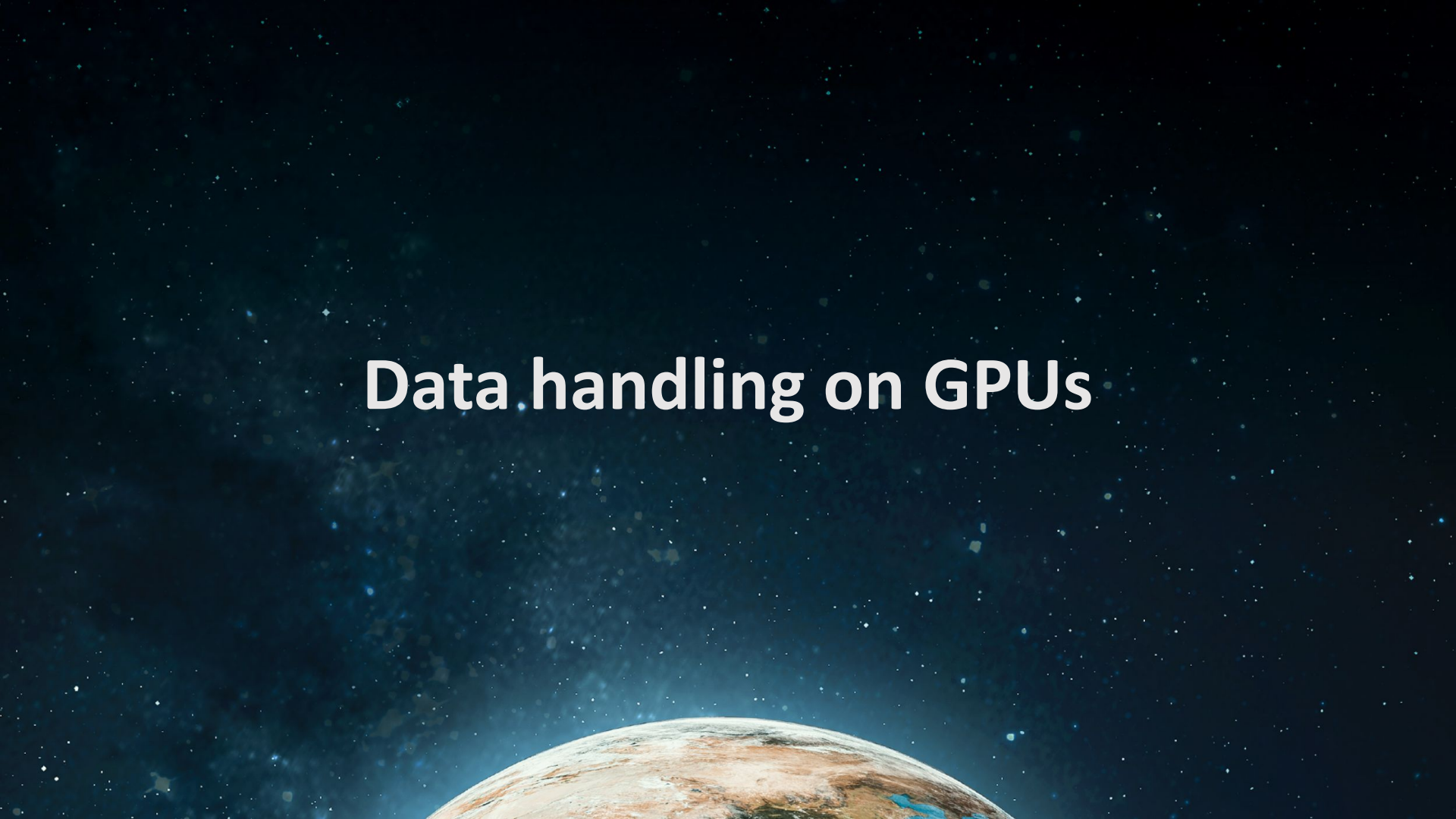
```
#pragma omp target teams distribute
```



Vector addition with OpenMP offloading



Data handling on GPUs



Data handling with OpenMP offloading

```
while (error > MAX_ERROR && iterations < MAX_ITER)
        error = 0.;
        for (int i = 1; i < NUM_ELEMENTS - 1; i++)
                for (int j = 1; j < NUM_ELEMENTS - 1; j++)
                        arr_new[i][j] = 0.25 * (array[i][j + 1] + array[i][j - 1] +
                                                array[i - 1][j] + array[i + 1][j]);
                        error = fmaxf (error, fabsf (arr_new[i][j] - array[i][j]));
        for (int i = 1; i < NUM_ELEMENTS - 1; i++)
                for (int j = 1; j < NUM_ELEMENTS - 1; j++)
                        array[i][j] = arr_new[i][j];
        iterations += 1;
```

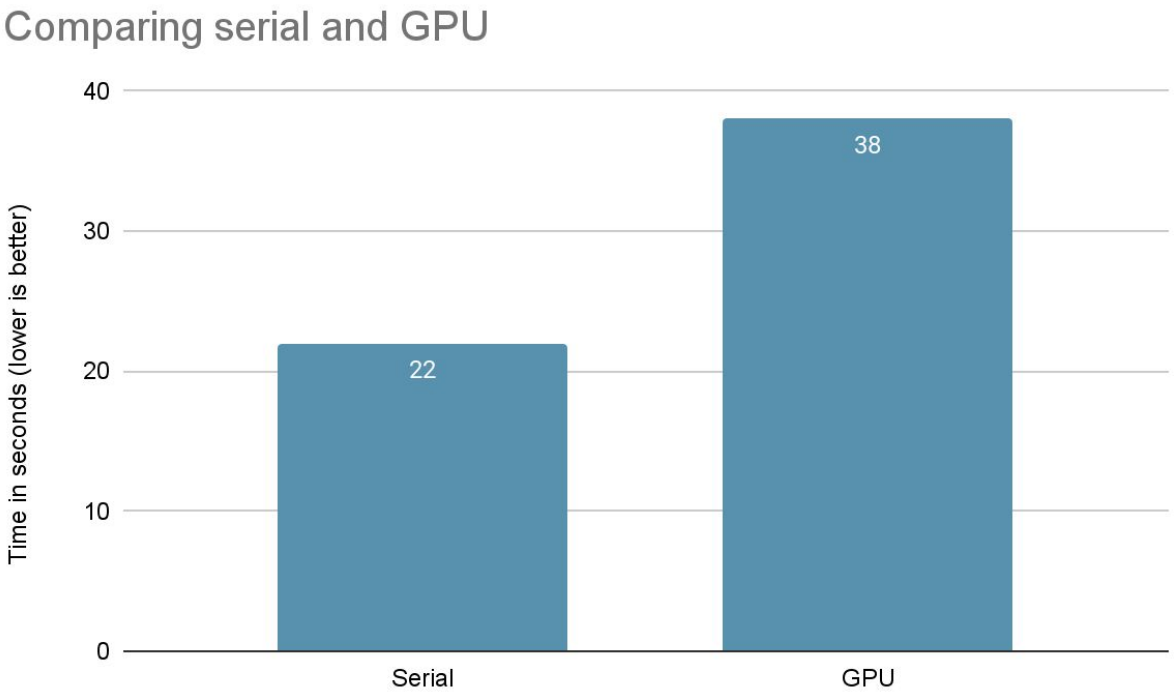
Warning: Not complete example

Data handling with OpenMP offloading

```
while (error > MAX_ERROR && iterations < MAX_ITER)

    error = 0.;
    #pragma omp target teams distribute parallel for collapse(2) reduction(max: error)
    for (int i = 1; i < NUM_ELEMENTS - 1; i++)
        for (int j = 1; j < NUM_ELEMENTS - 1; j++)
            arr_new[i][j] = 0.25 * (array[i][j + 1] + array[i][j - 1] +
                                    array[i - 1][j] + array[i + 1][j]);
            error = fmaxf (error, fabsf (arr_new[i][j] - array[i][j]));
    #pragma omp target teams distribute parallel for collapse(2)
    for (int i = 1; i < NUM_ELEMENTS - 1; i++)
        for (int j = 1; j < NUM_ELEMENTS - 1; j++)
            array[i][j] = arr_new[i][j];
    iterations += 1;
```

Warning: Not complete example

Data handling with OpenMP offloading

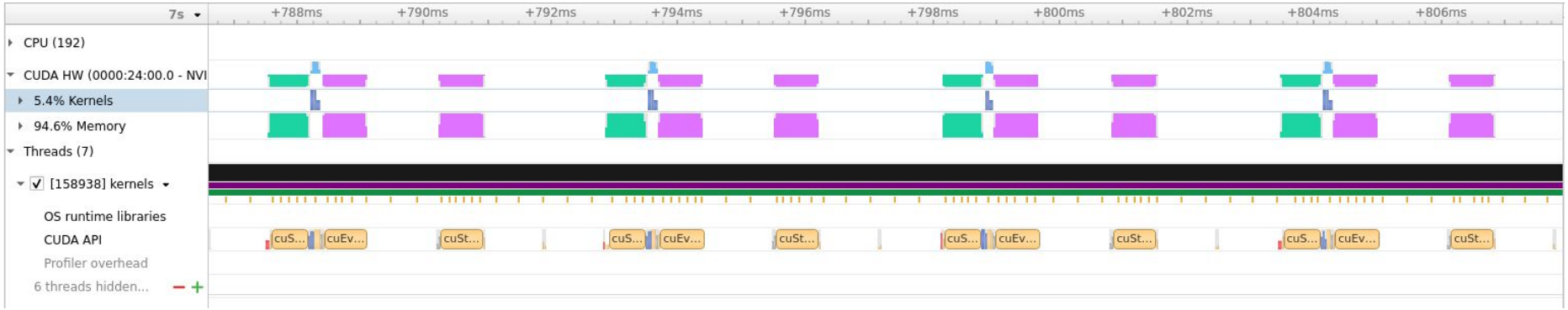


Data handling with OpenMP offloading

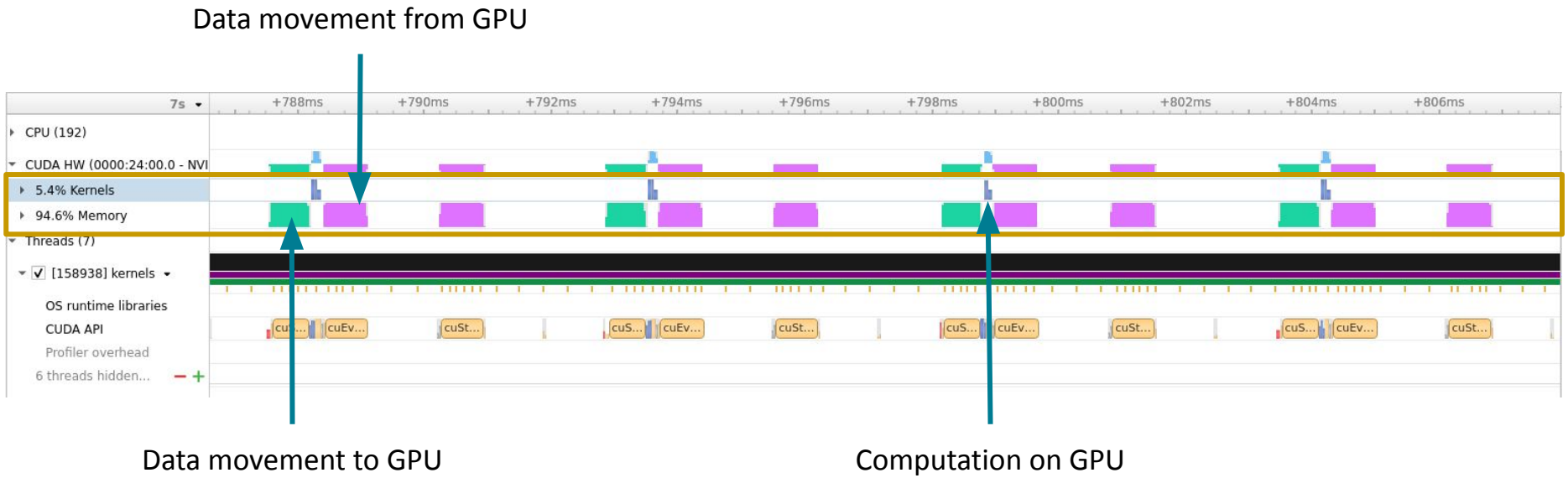
```
while (error > MAX_ERROR && iterations < MAX_ITER)

    error = 0.;
    #pragma omp target teams distribute parallel for collapse(2) reduction(max: error)
    for (int i = 1; i < NUM_ELEMENTS - 1; i++)
        for (int j = 1; j < NUM_ELEMENTS - 1; j++)
            arr_new[i][j] = 0.25 * (array[i][j + 1] + array[i][j - 1] +
                                    array[i - 1][j] + array[i + 1][j]);
            error = fmaxf (error, fabsf (arr_new[i][j] - array[i][j]));
    #pragma omp target teams distribute parallel for collapse(2)
    for (int i = 1; i < NUM_ELEMENTS - 1; i++)
        for (int j = 1; j < NUM_ELEMENTS - 1; j++)
            array[i][j] = arr_new[i][j];
    iterations += 1;
```

Warning: Not complete example

Data handling with OpenMP offloading



Data handling with OpenMP offloading

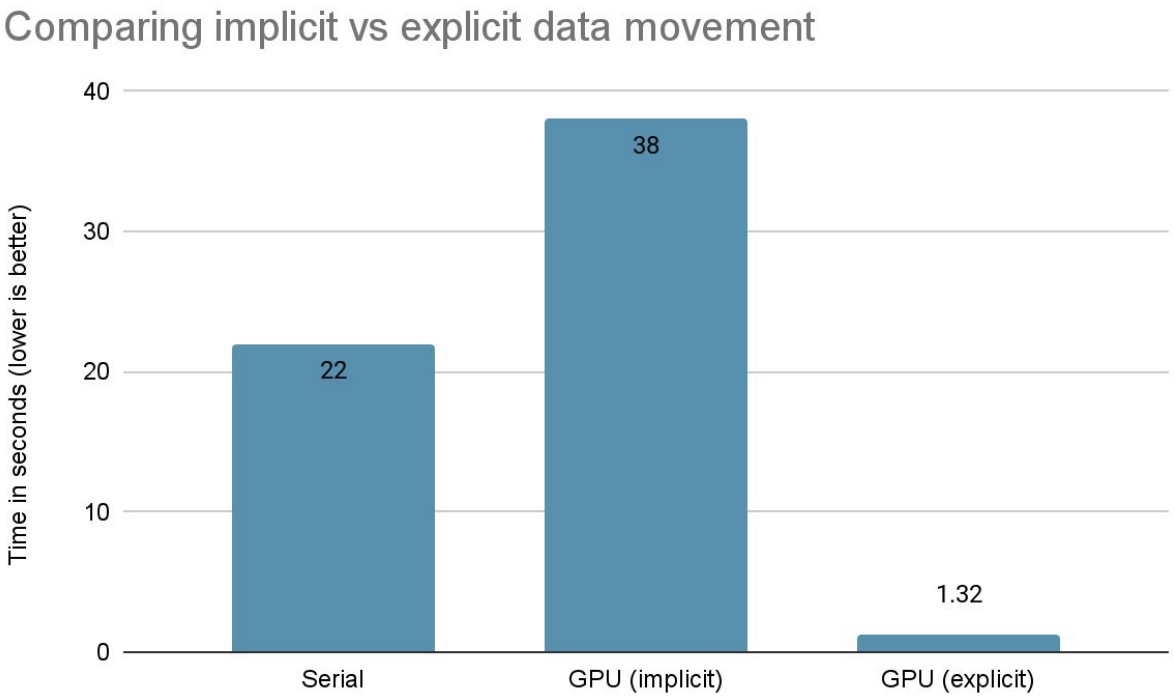


Data handling with OpenMP offloading

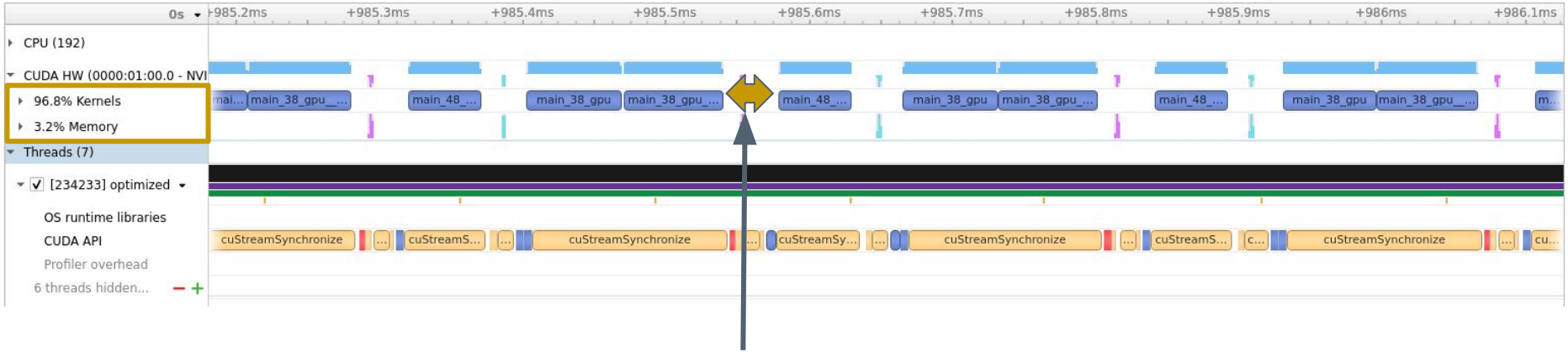
```
#pragma omp target data map(alloc:arr_new) map(tofrom:array)
while (error > MAX_ERROR && iterations < MAX_ITER)

    error = 0.;
    #pragma omp target teams distribute parallel for collapse(2) reduction(max: error)

    for (int i = 1; i < NUM_ELEMENTS - 1; i++)

        for (int j = 1; j < NUM_ELEMENTS - 1; j++)

            arr_new[i][j] = 0.25 * (array[i][j + 1] + array[i][j - 1] +
                                    array[i - 1][j] + array[i + 1][j]);

            error = fmaxf (error, fabsf (arr_new[i][j] - array[i][j]));
    #pragma omp target teams distribute parallel for collapse(2)

    for (int i = 1; i < NUM_ELEMENTS - 1; i++)

        for (int j = 1; j < NUM_ELEMENTS - 1; j++)

            array[i][j] = arr_new[i][j];

    iterations += 1;
```

Warning: Not complete example

Data handling with OpenMP offloading



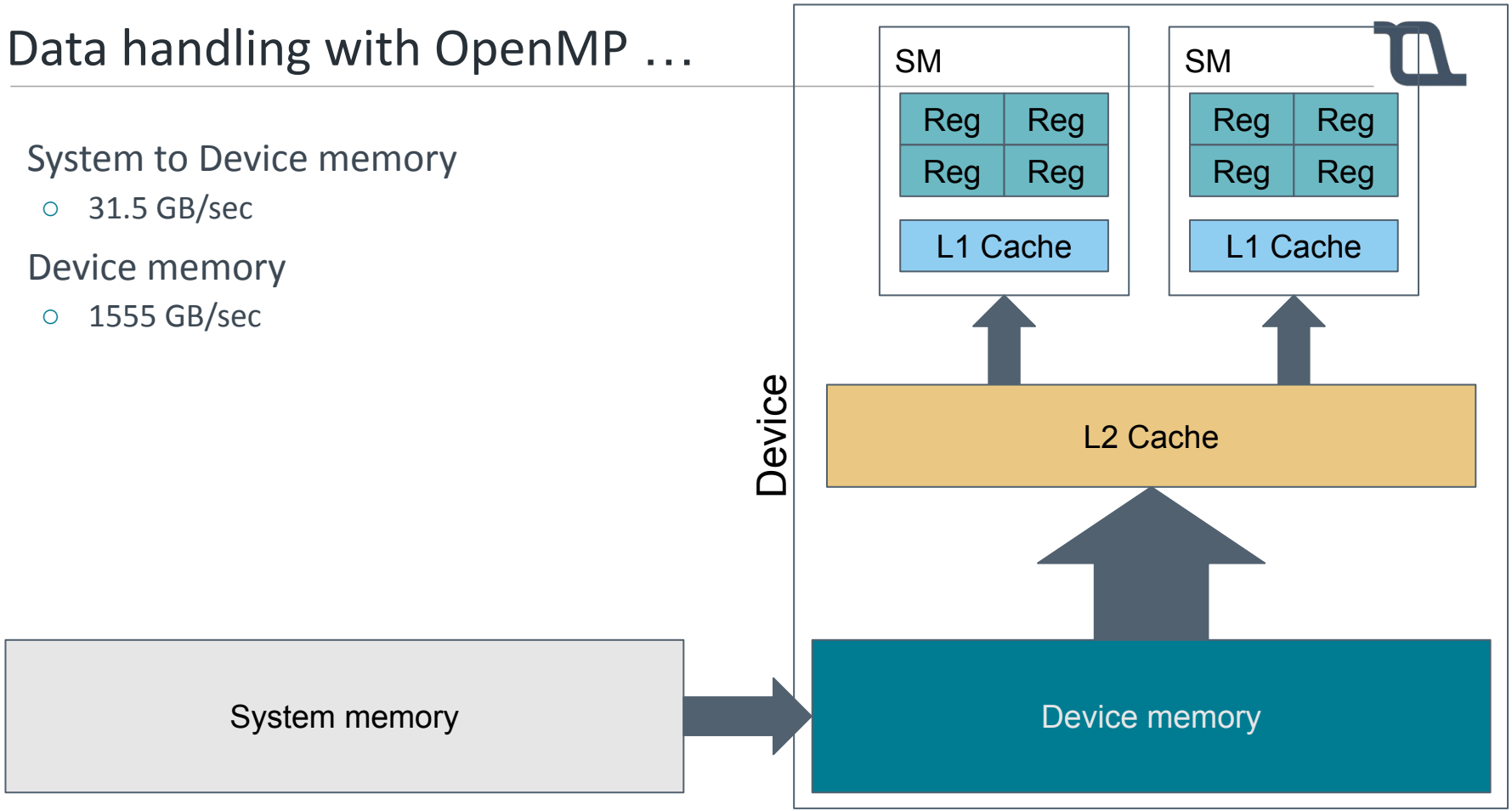
Data handling with OpenMP offloading



35 *micro*seconds!

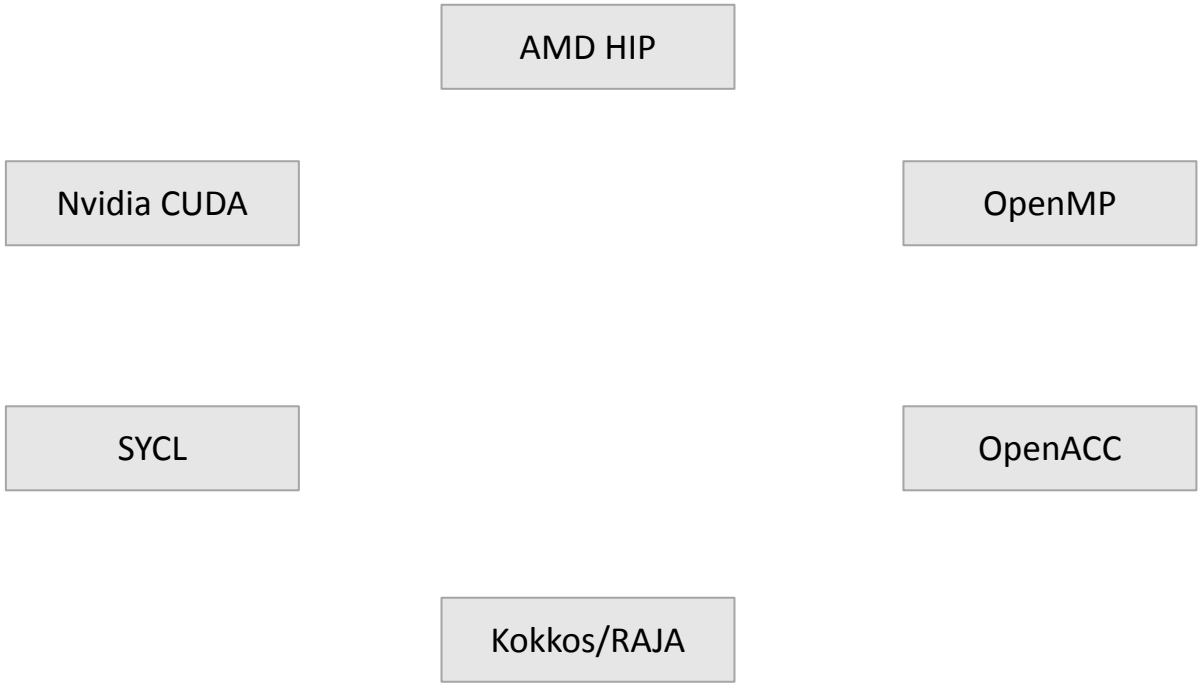
Data handling with OpenMP …

- System to Device memory
  - 31.5 GB/sec
- Device memory
  - 1555 GB/sec



A broader perspective on GPU programming

Software technologies



Software technologies

C/C++

C/C++ & Fortran

AMD HIP

Nvidia CUDA

OpenMP

SYCL

OpenACC

Kokkos/RAJA

Software technologies

Compiler based

Directive based

Library based

AMD HIP

Nvidia CUDA

OpenMP

SYCL*

OpenACC

Kokkos/RAJA

Software technologies

Vendor specific

Vendor neutral

AMD HIP*

Nvidia CUDA

OpenMP

SYCL

OpenACC*

Kokkos/RAJA

Software technologies

Least complex

Medium

Most complex

OpenMP

OpenACC

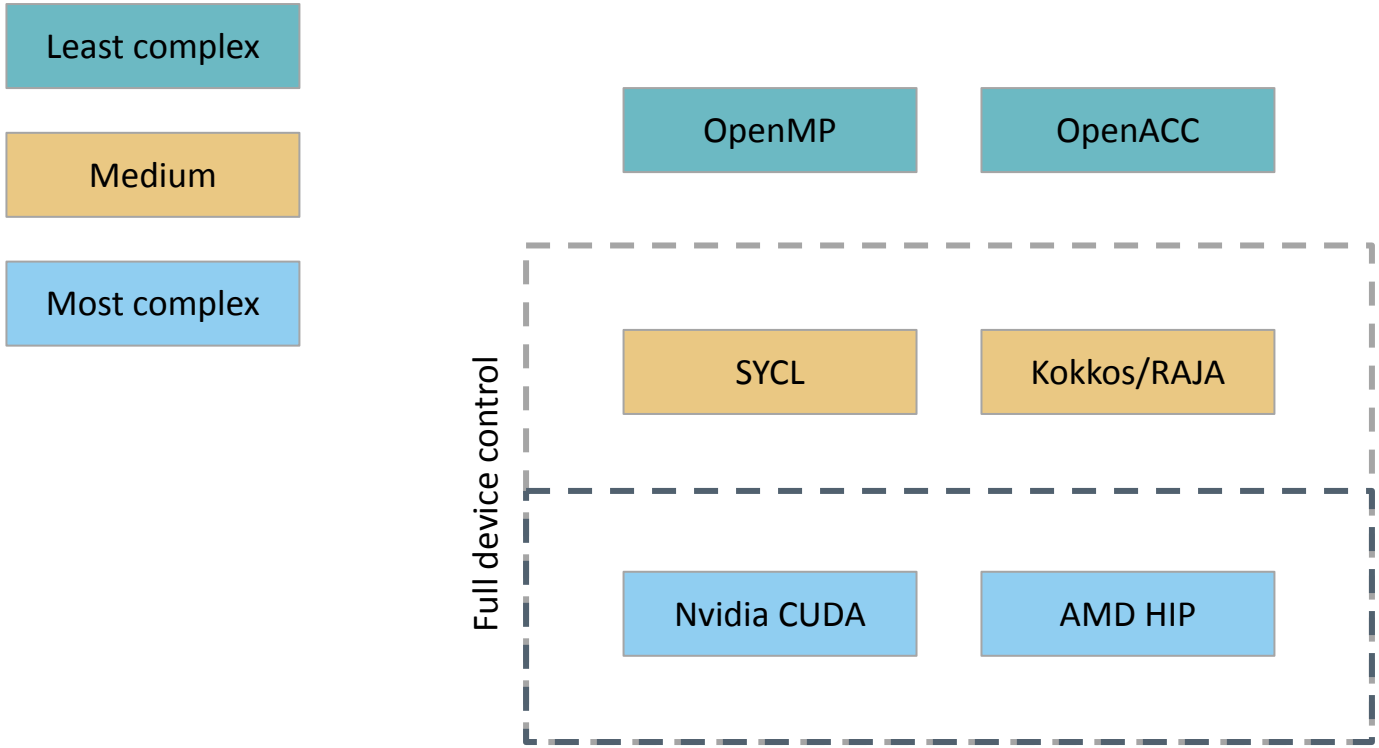
SYCL

Kokkos/RAJA

Nvidia CUDA

AMD HIP

Software technologies



Software technologies

- Preference should be based on available hardware

- Vendor neutrality
- Type of code base
  - Who will develop the code
  - Takes time to transition/onboard

- Always use an optimized library!
  - Reduce development burden
  - Share knowledge
  - Help the community

Software technologies

- Preference should be based on available hardware

- Vendor neutrality
- Type of code base
  - Who will develop the code
  - Takes time to transition/onboard

- Always use an optimized library!
  - Reduce development burden
  - Share knowledge
  - Help the community

OpenMP

SYCL

Future HPC systems

- LUMI
  - Based on AMD GPUs
  - 2560 nodes
    - 4 x AMD MI250X
    - 14080 stream processors per GPU
    - 128 GB memory per GPU
    - Each GPU is directly attached to interconnect
  - Fastest supercomputer in Europe
    - 550 petaflops theoretical performance
  - Norway is a member of the LUMI consortium

Future HPC systems

- Software portability will become an even larger concern for scientific compute
  - AMD, Intel and Nvidia will soon™ offer
    - Fully integrated CPU + GPU systems
    - AMD CPU + AMD GPU / Intel CPU + Intel GPU / Nvidia CPU + Nvidia GPU
  - Different HPC systems will have different GPU architecture
  - Potentially also different CPU architecture (!)
    - Nvidia is using ARM64 as its CPU architecture
- Portable solutions do exist!
  - SYCL and OpenMP
  - Libraries with similar interfaces
    - BLAS, FFT, Rand

Questions?