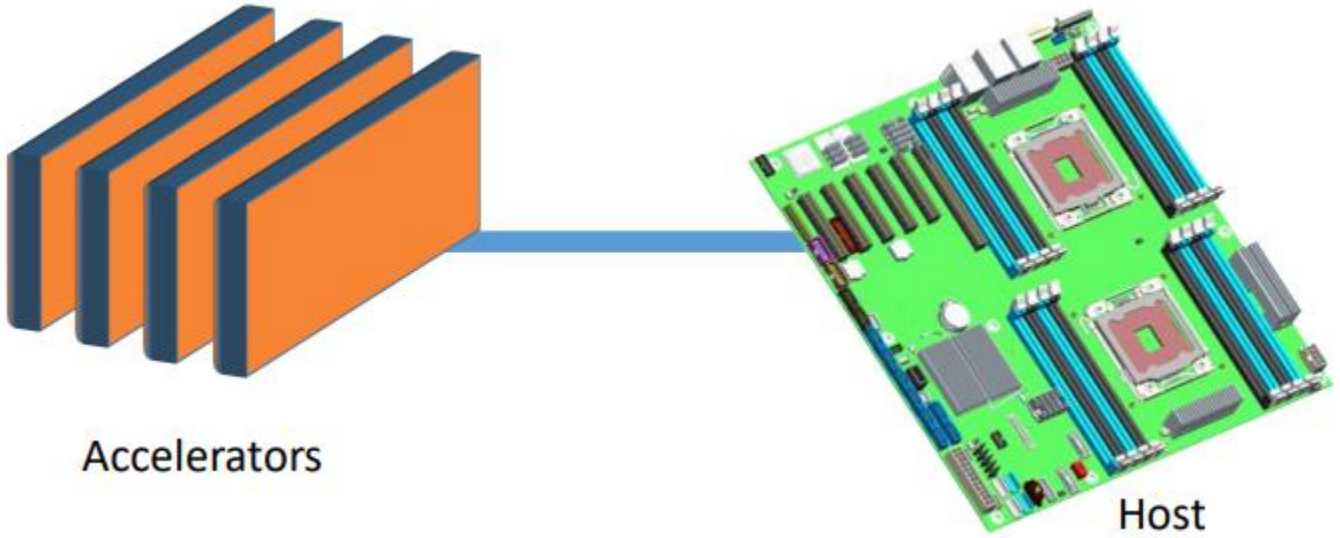
OPENMP OFFLOADING

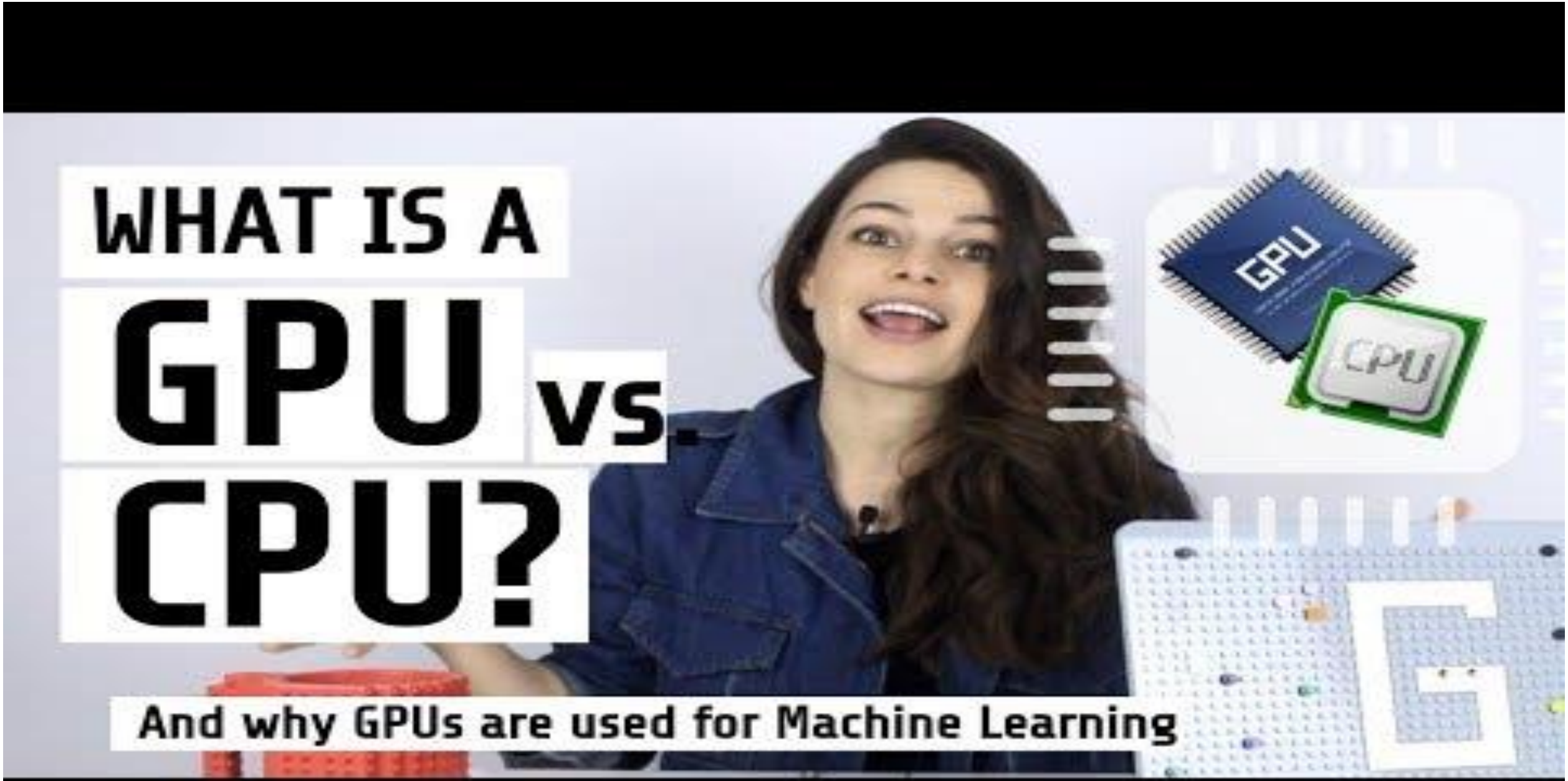
GPU Work

What is offloading ?

- Offloading is the process of transferring a portion of a computation from a central processor (such as a CPU) to a specialized accelerator (such as a coprocessor , FPGA or GPU) or to a remote machine.
- It can be use to improve the performance of a computation

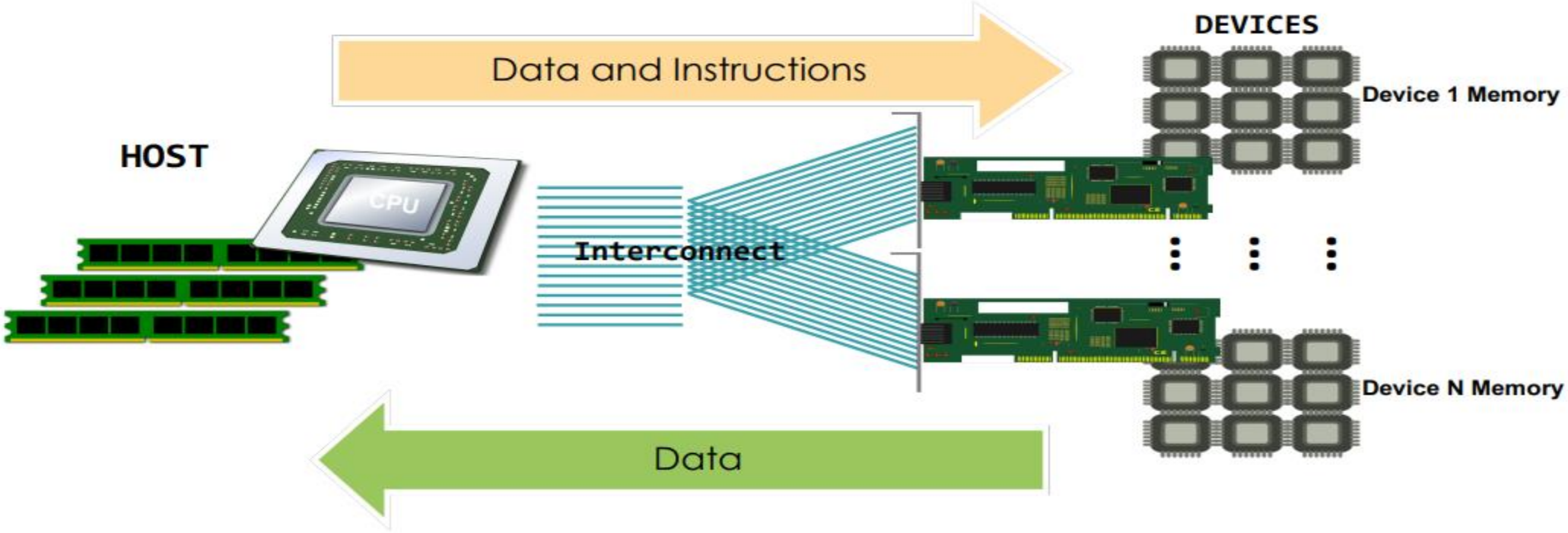


What is the Difference Between GPU and CPU?


WHAT IS A
GPU vs.
CPU?
GPU
CPU
And why GPUs are used for Machine Learning

Introduction: OpenMP Offload

OpenMP offload constructs are a set of directives for C++ and Fortran that were introduced in OpenMP 4.0 and further enhanced in later versions



OpenMP Offload: Steps

- Identification of compute kernels
  - CPU initiates kernel for execution on the device
- Expressing parallelism within the kernel
- Manage data transfer between CPU and Device
  - relevant data needs to be moved from host to device memory
  - kernel executes using device memory
  - relevant data needs to be moved from device to main memory

Step 1: Identification of Kernels to Offload

- Look for compute intensive code and that can benefit from parallel Execution
  - Use performance analysis tools to find bottlenecks
- Track independent work units with well defined data accesses
- Keep an eye on platform specs
  - GPU memory is a precious resource

How to Offload ?

C/C++	Description
#pragma omp target [clause[[,] clause] ...] newline structured-block	The target construct offloads the enclosed code to the accelerator.

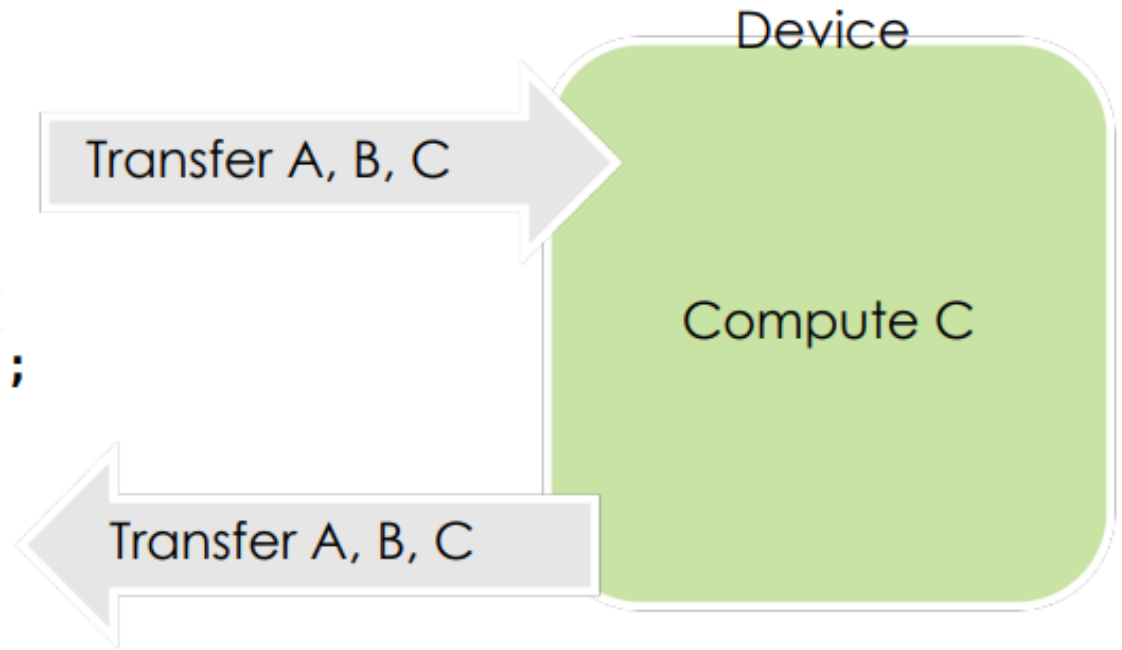
– A device data environment is created for the structured block
  – The code region is mapped to the device and executed.

Target Directive

- Clauses allowed on the target directive:
  - if([target :] scalar-expression)
  - device([device-modifier :] integer-expression)
  - thread_limit(integer-expression)
  - private(list)
  - firstprivate(list)
  - in_reduction(reduction-identifier : list)
  - map([[map-type-modifier[,] [map-type-modifier[,] ...]] map-type:] locator-list)
  - is_device_ptr(list)
  - has_device_addr(list)
  - defaultmap(implicit-behavior[:variable-category])
  - nowait
  - depend([depend-modifier,] dependence-type : locator-list)
  - allocate([allocator :] list)
  - uses_allocators(allocator[(allocator-traits-array)] [,allocator[(allocator-traits-array)] ...])

Example using omp target

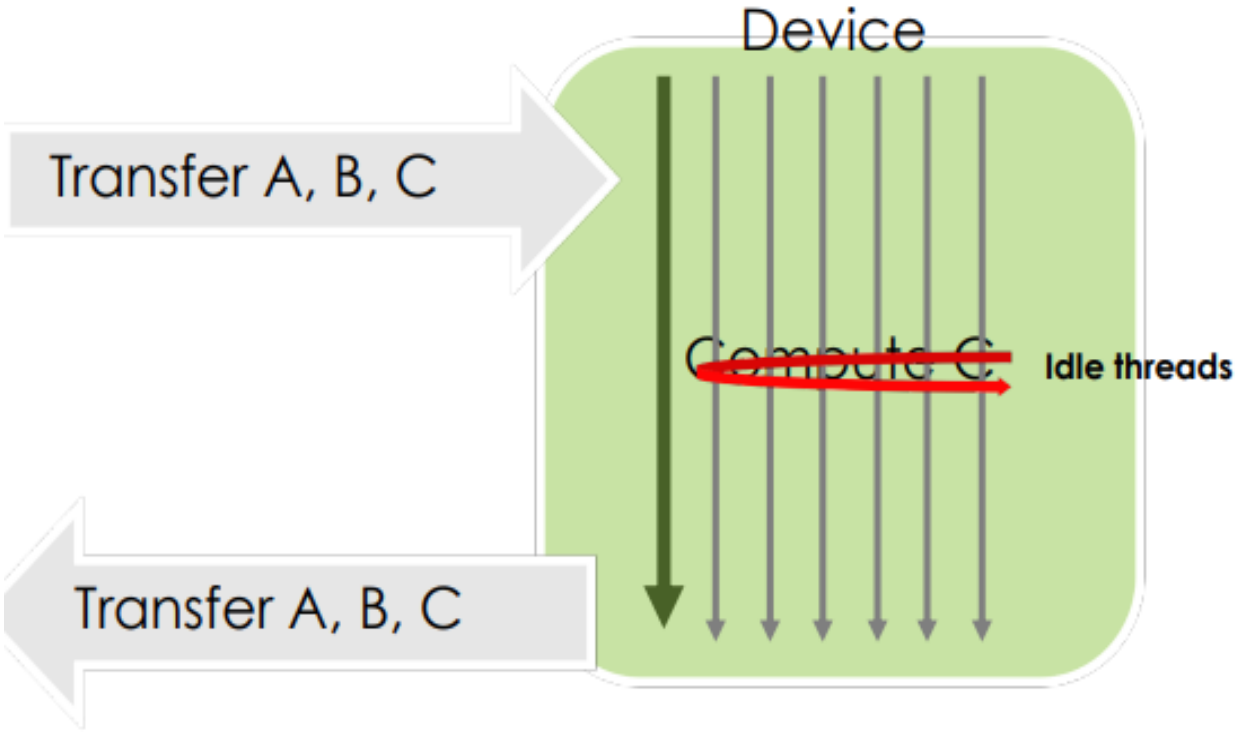
```
int A[N][N], B[N][N], C[N][N];
/* initialize arrays */
#pragma omp target
{
      for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
      C[i][j] = A[i][j] + B[i][j];
}
}
} // end target
```



The target construct is a task generating construct

Example using omp target(Cont.)

```
int A[N][N], B[N][N], C[N][N];
/*
initialize arrays
*/
#pragma omp target
{
for (int i = 0; i < N; ++i) {
for (int j = 0; j < N; ++j) {
C[i][j] = A[i][j] + B[i][j];
}
}
} // end target
```



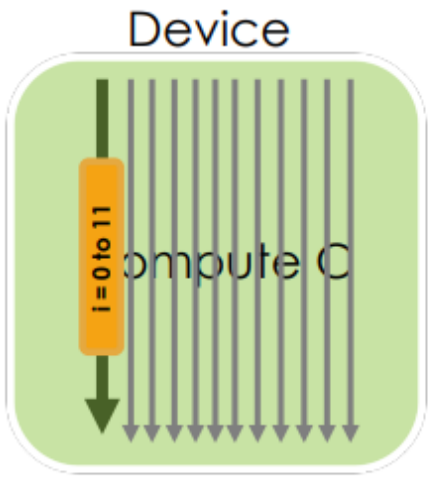
Device Execution Directives

C/C++	Description
#pragma omp target [clause[[,] clause] ...] new-line structured-block	The target construct offloads the enclosed code to the accelerator.
#pragma omp target teams [clause[[,] clause] ...] new-line structured-block	The target construct offloads the enclosed code to the accelerator.The teams construct creates a league of teams. The initial thread of each team executes the code region.
#pragma omp target teams distribute [clause[[,] clause] ...] new-line loop-nest	The target construct offloads the enclosed code to the accelerator. A league of thread teams is created, and loop iterations are distributed and executed by the initial teams.
#pragma omp target teams distribute parallel for [clause[[,] clause] ...] new-line loop-nest	The target construct offloads the enclosed code to the accelerator. A league of thread teams are created, and loop iterations are distributed and executed in parallel by all threads of theteams.

Device Execution Directives(Cont.)

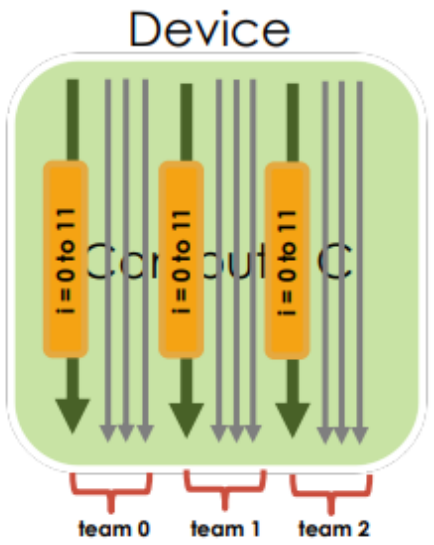
target

```
#pragma omp target
for (int i = 0; i < 12; ++i)
{
    C[i] = A[i] + B[i];
}
```



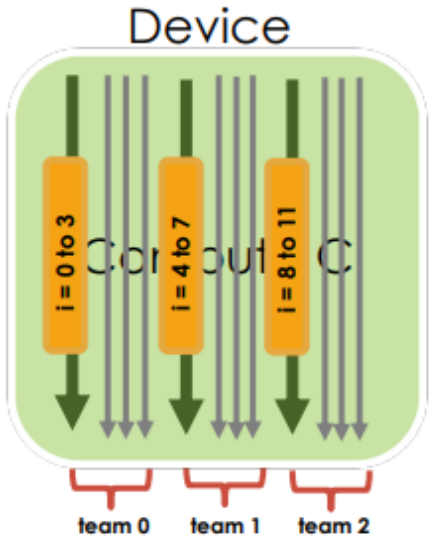
target teams

```
#pragma omp target teams
num_teams(3)
for (int i = 0; i < 12; ++i)
{
    C[i] = A[i] + B[i];
}
```



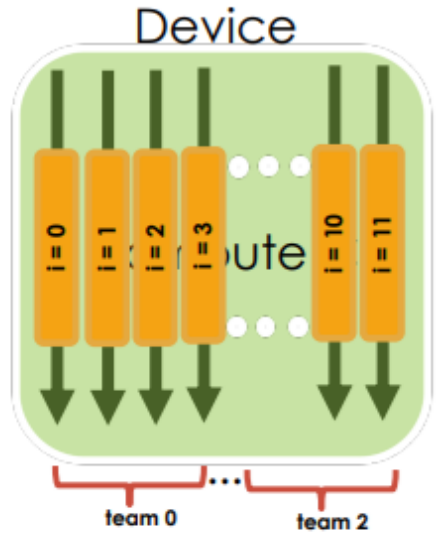
target teams distribute

```
#pragma omp target teams
distribute num_teams(3)
for (int i = 0; i < 12; ++i)
{
    C[i] = A[i] + B[i];
}
```



target teams distribute parallel

```
#pragma omp target teams
distribute parallel for
num_teams(3)
for (int i = 0; i < 12; ++i)
{
    C[i] = A[i] + B[i];
}
```




Demo

Thank You

Contact:
Name:Parikshit Ardhapurkar
Email: aparikshit@cdaca.in
Mod: 8605045054