Using OpenMP on ThetaGPU

2021 Virtual ALCF Computational Performance Workshop

Ye Luo and Colleen Bertoni
Argonne Leadership Computing Facility

www.anl.gov

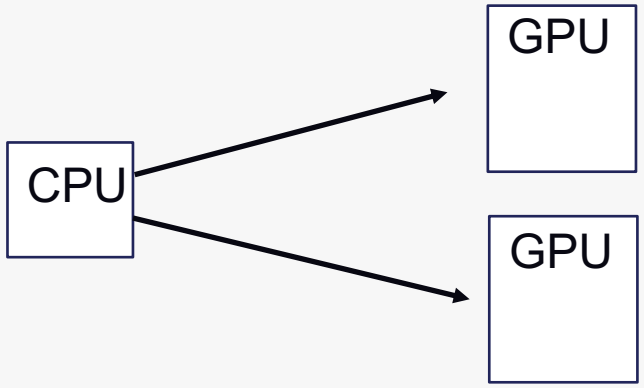
Motivation

Maximum speed at which you can compute is bound by

(clock rate of cores) * (number of cores) * (number of operations each core can do per cycle)

Amount of parallelism you can exploit

Motivation

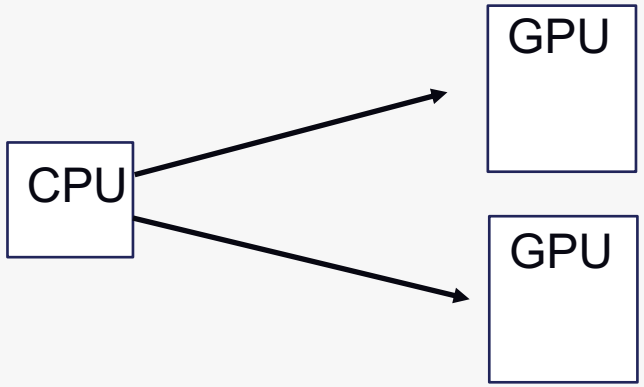


Maximum speed at which you can compute is bound by

(clock rate of cores) * (number of cores) * (number of operations each core can do per cycle)

Amount of parallelism you can exploit

Motivation



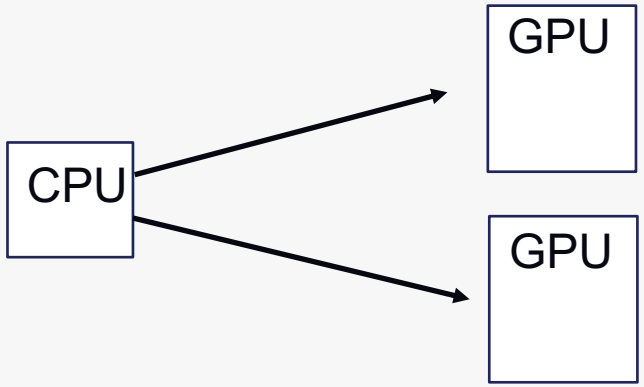
Maximum speed at which you can compute is bound by

(clock rate of cores) * (number of cores) * (number of operations each core can do per cycle)

Amount of parallelism you can exploit

For the GPUs on a ThetaGPU node:
78 Tflops ~= (1.41 *10^9 cycles/second)* (108 SMs/ GPU) * (8 GPUs) * (32 FPU inst/ (SM*cycle))* 2 (FMA factor)
For the CPUs on a ThetaGPU node:
2.3 Tflops ~= (2.25 *10^9 cycles/second)* (64 cores) * (2 CPUs) * 4 (SIMD width) * 2 (FMA factor)

Motivation



Maximum speed at which you can compute is bound by

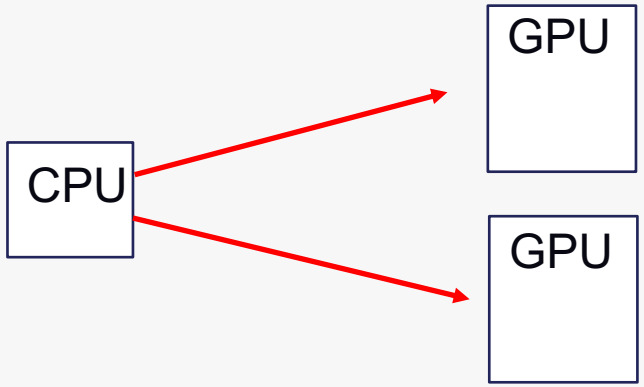
(clock rate of cores) * (number of cores) * (number of operations each core can do per cycle)

1. Far more computational power on GPUs than CPUs

Amount of parallelism you can exploit

For the GPUs on a ThetaGPU node:
78 Tflops ~= (1.41 *10^9 cycles/second)* (108 SMs/ GPU) * (8 GPUs) * (32 FPU inst/ (SM*cycle))* 2 (FMA factor)
For the CPUs on a ThetaGPU node:
2.3 Tflops ~= (2.25 *10^9 cycles/second)* (64 cores) * (2 CPUs) * 4 (SIMD width) * 2 (FMA factor)

Motivation



Maximum speed at which you can compute is bound by

(clock rate of cores) * (number of cores) * (number of operations each core can do per cycle)

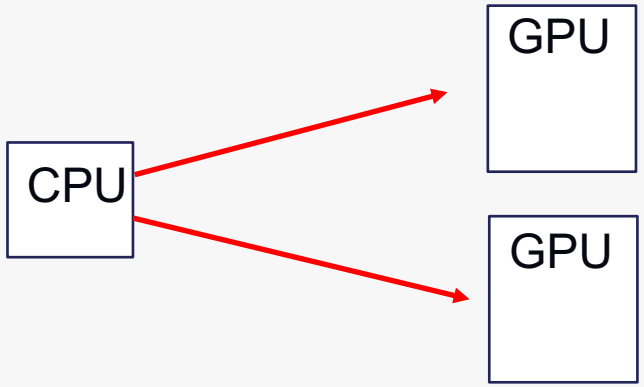
1. Far more computational power on GPUs than CPUs

Amount of parallelism you can exploit

2. Be careful of data transfer bottlenecks

For the GPUs on a ThetaGPU node:
78 Tflops ~= (1.41 *10^9 cycles/second)* (108 SMs/ GPU) * (8 GPUs) * (32 FPU inst/ (SM*cycle))* 2 (FMA factor)
For the CPUs on a ThetaGPU node:
2.3 Tflops ~= (2.25 *10^9 cycles/second)* (64 cores) * (2 CPUs) * 4 (SIMD width) * 2 (FMA factor)

Agenda: how OpenMP can help you…



Maximum speed at which you can compute is bound by

(clock rate of cores) * (number of cores) * (number of operations each core can do per cycle)

1. Effectively use the computational power on GPUs

Amount of parallelism you can exploit

2. Avoid data transfer bottlenecks

For the GPUs on a ThetaGPU node:
78 Tflops ~= (1.41 *10^9 cycles/second)* (108 SMs/ GPU) * (8 GPUs) * (32 FPU inst/ (SM*cycle))* 2 (FMA factor)
For the CPUs on a ThetaGPU node:
2.3 Tflops ~= (2.25 *10^9 cycles/second)* (64 cores) * (2 CPUs) * 4 (SIMD width) * 2 (FMA factor)

Later today: Hands-on!

- Using OpenMP (~20 min)
- Demo of OpenMP (~20 min)
  - OpenMP 101 and basics on ThetaGPU
- Hands-on Exercises (~20 min)

```
$ git clone https://github.com/argonne-lcf/CompPerfWorkshop-2021.git
$ cd CompPerfWorkshop-2021
$ cd 01_openmp
```

Brief OpenMP offload overview and upcoming features

High-level OpenMP Programming Model Overview

- Why OpenMP?
  - Open standard for parallel programming with support across vendors
  - OpenMP runs on CPU threads, GPUs, SIMD units
  - C/C++ and Fortran
  - Supported by Intel, Cray, GNU, LLVM compilers and others
  - Specification and examples http://www.openmp.org
  - OpenMP offload will be supported on Aurora, Frontier, Perlmutter
- Four Important high-level features to express parallelism
  - Fork and join thread parallelism
  - SIMD parallelism (added in 4.0)
  - Device Offload parallelism (added in 4.0)
  - Tasking parallelism (added in 3.0)

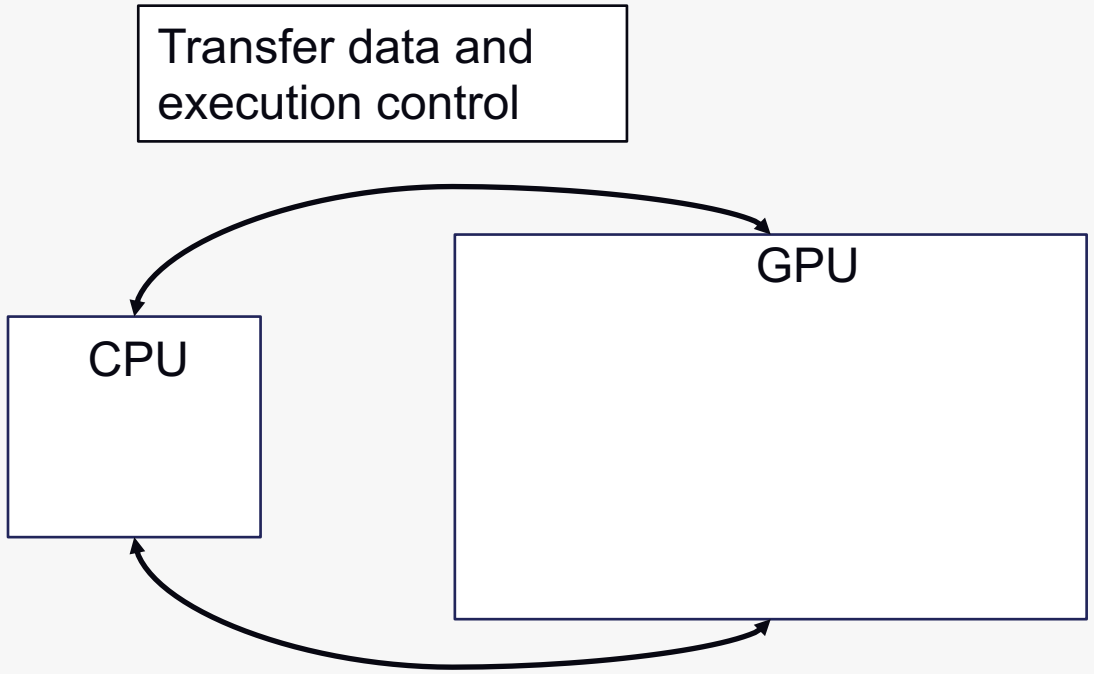
https://www.alcf.anl.gov/support-center/theta/openmp-theta

OpenMP Offload Introduction

The goal is to introduce important basic topics for OpenMP offloading

We will cover three basic offloading topics:

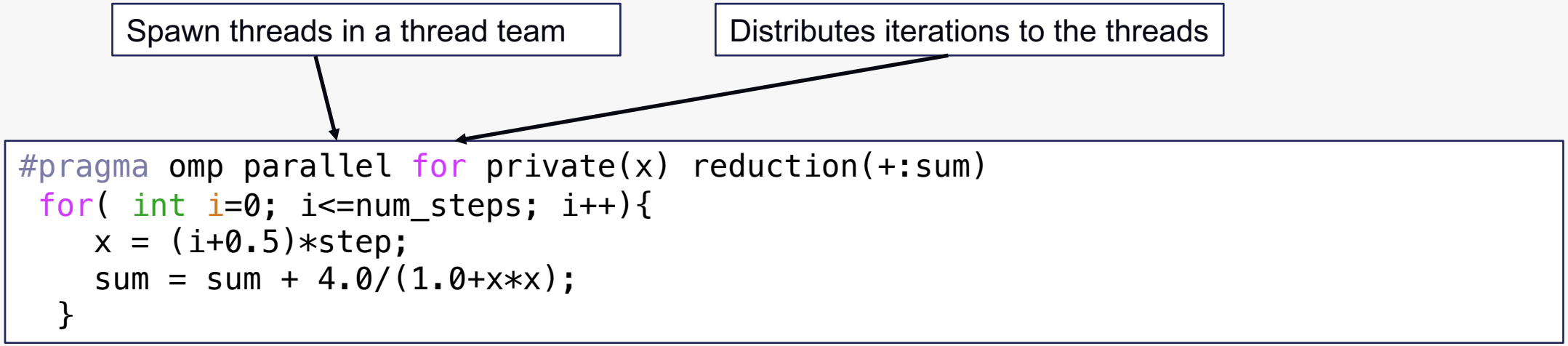
1. Offloading code to the device and getting device info
2. Expressing parallelism
3. Mapping data



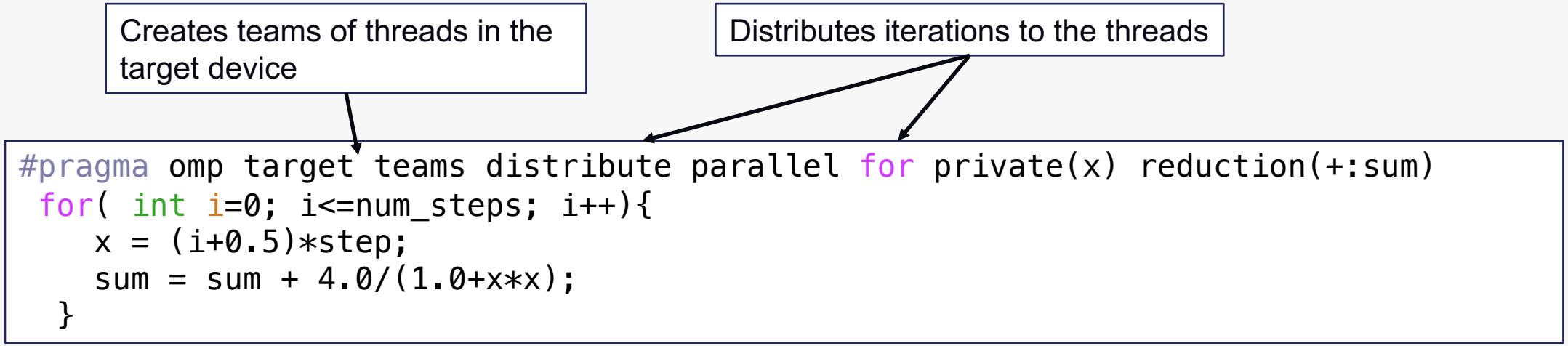
Compiler support for offloading

• GCC	• NVIDIA
• LLVM	• Intel
• IBM XL	• AMD
• Cray	

CPU OpenMP parallelism

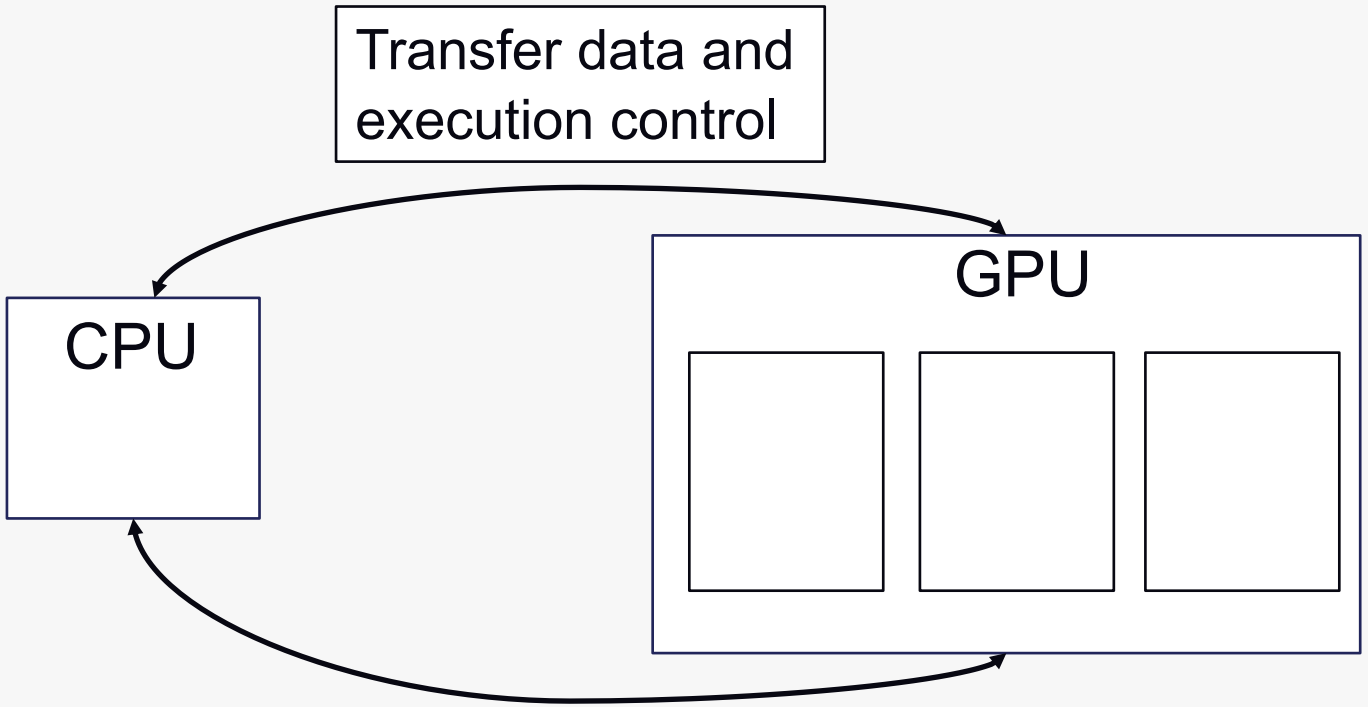


GPU OpenMP parallelism



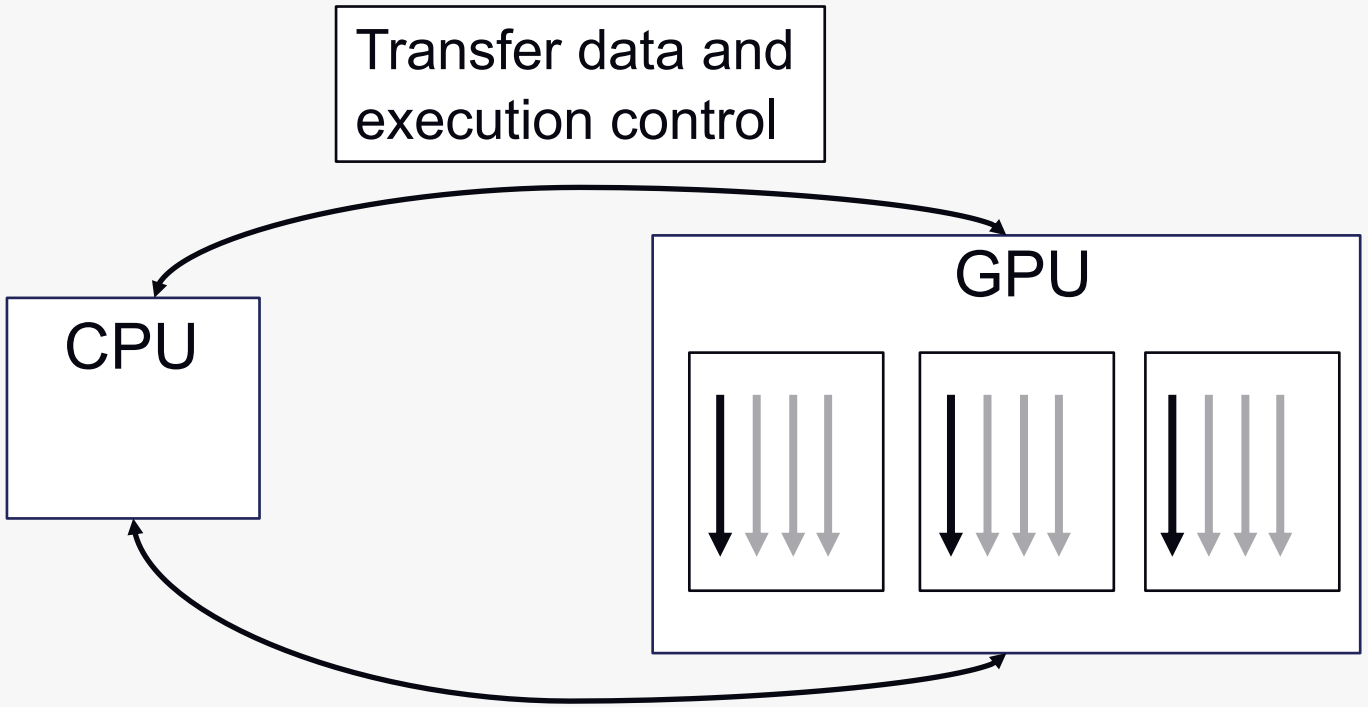
OpenMP Offload Introduction

- **Target construct**: offloads code and data to the device and runs in serial on the device



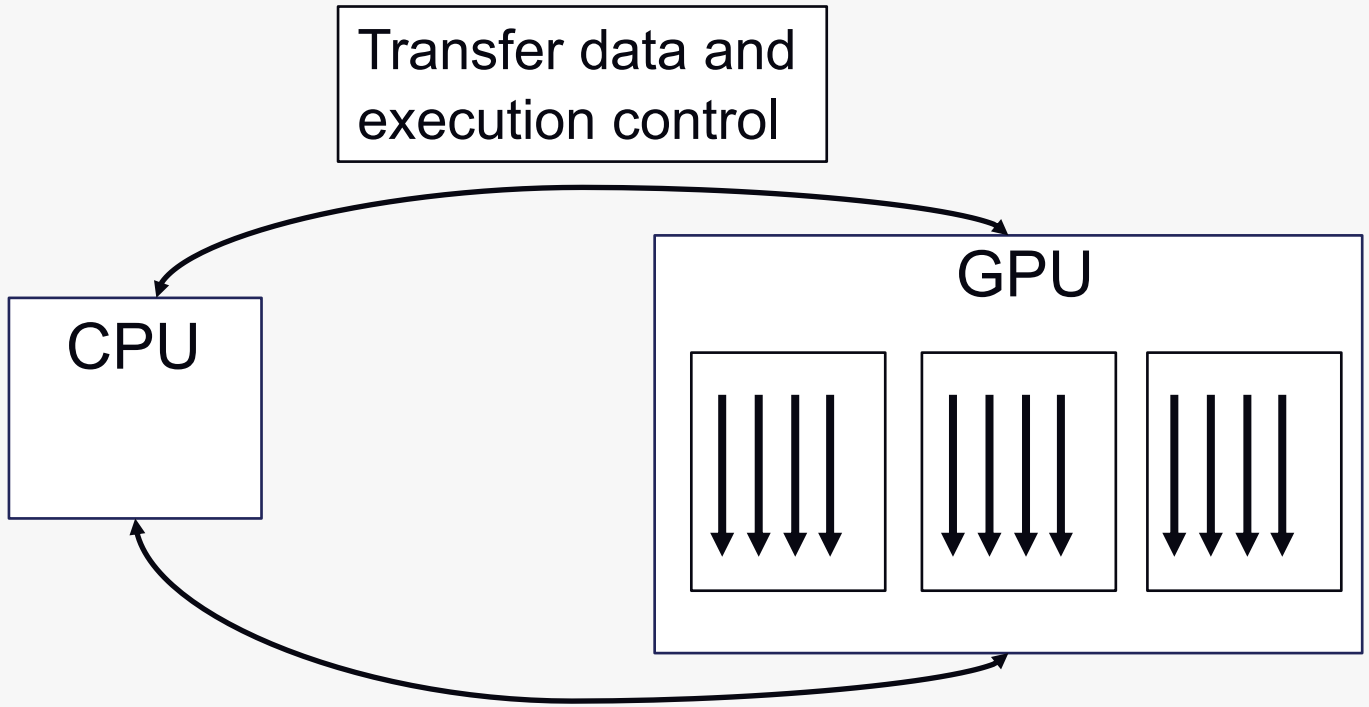
OpenMP Offload Introduction

- **Target construct**: offloads code and data to the device and runs in serial on the device
- **Teams construct**: creates a league of teams, each with one thread, which run concurrently on SMs (Nvidia terminology)



OpenMP Offload Introduction

- **Target construct**: offloads code and data to the device and runs in serial on the device
- **Teams construct**: creates a league of teams, each with one thread, which run concurrently on SMs (Nvidia terminology)
- **Parallel construct**: creates multiple threads in the teams, each which can run concurrently



GPU OpenMP parallelism

Distributes iterations to the threads, where each thread uses SIMD parallelism

```
...

#pragma omp target teams distribute parallel for simd map(v1[0:N],p[0:N])
 for (i=0; i<N; i++)
 {
    p[i] = v1[i];
 }

...
```

Controlling data transfer

Creates teams of threads in the target device

New features in OpenMP 5.0/5.1

- Unified Shared Memory Support (no need to explicitly map data)
- Loop construct (simpler expression of parallelism)
- Declare variant (portable wrappers for variants of a function)
- Metadirective
- Host teams
- Implicit declare target
- Declare mapper
- Collapse on non-rectangular loops and additional loop conditions
- More…

OpenMP and exploiting parallelism

Find the concurrency in you app

Within a kernel running on a computing device

- Coarse level concurrency (>10~100)
  - OpenMP teams(GPU SMs), OpenMP threads(CPU cores)
  - Minimize (~zero) synchronization
  - Emphasize on weak scaling

- Fine level concurrency (100~1000)
  - OpenMP threads within teams (GPU threads)
  - OpenMP simd within threads (CPU vector unit)
  - Emphasize on data locality.

- If two levels are fused (>10^4~10^5)

Find the concurrency in you app

Beyond a compute kernel. Handling the control flow by the host

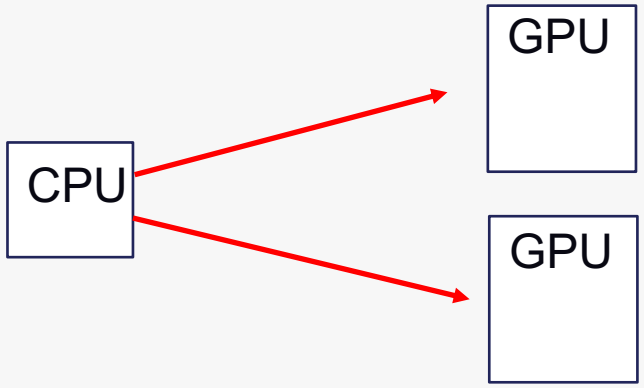
- Exploit task parallelism (very coarse level)
  - Enqueue target tasks (OpenMP target nowait)
  - Having a few CPU threads to offload more rapidly (OpenMP threads + target)

- Using leverage more GPUs (extremely coarse level)
  - Decompose the computation into multiple subtasks
    - Distribute them using MPI to multiple GPU
    - Distribute them by leveraging OpenMP multi device support.

Application developers are responsible to find as much concurrency as possible
OpenMP compiler/runtimes enable corresponding parallelism

OpenMP and Data transfer

Agenda: how OpenMP can help you…



Maximum speed at which you can compute is bound by

(clock rate of cores) * (number of cores) * (number of operations each core can do per cycle)

1. Effectively use the computational power on GPUs

It's roughly an order of magnitude slower to access memory over PCIe than accessing memory on the device

2. Avoid data transfer bottlenecks

OpenMP and data transfer

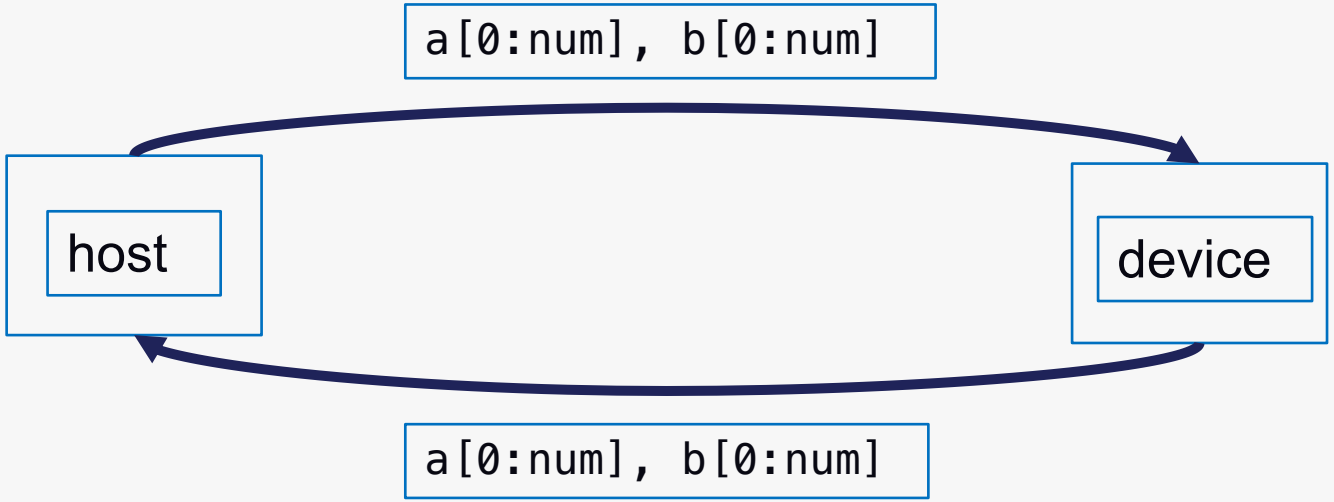
```
...


#pragma omp target teams distribute parallel for map(tofrom:a[0:num], b[0:num])
     for (size_t j=0; j<num; j++) {
        a[j] = a[j]+scalar*b[j];

      }
...


...
```

Maps a and b to and from the device



OpenMP and data transfer

```
...


#pragma omp target teams distribute parallel for map(tofrom:a[0:num], b[0:num])
     for (size_t j=0; j<num; j++) {
        a[j] = a[j]+scalar*b[j];

      }
...
#pragma omp target teams distribute parallel for map(tofrom:a[0:num], c[0:num])
     for (size_t j=0; j<num; j++) {
        c[j] = c[j]+scalar*a[j];

      }
...
```

Maps a and b to and from the device

Maps a and c to and from the device

OpenMP and data transfer

```
...
#pragma omp target enter data map(to:a[0:num],b[0:num],c[0:num])

#pragma omp target teams distribute parallel for map(tofrom:a[0:num], b[0:num])
     for (size_t j=0; j<num; j++) {
        a[j] = a[j]+scalar*b[j];

      }
...
#pragma omp target teams distribute parallel for map(tofrom:a[0:num], b[0:num])
     for (size_t j=0; j<num; j++) {
        c[j] = c[j]+scalar*a[j];

      }
...
#pragma omp target exit data map(from:c[0:num])
```

Only maps a,b,c to device once and c back once

OpenMP and multiple GPUs

Two ways of handling multiple on-node GPUs

- Using MPI, one GPU per MPI rank
  - Pros: No difference intra-node vs inter-node, locality imposed by MPI.
  - Cons: cross-rank communication is non-trivial if performance is critical
    - IPC, GPU-aware communication

- Using OpenMP device clause
  - Pros: all the GPUs are within one process, no OS barrier
  - Cons:
    - explicit device management. Both compute and memory spaces.
    - Multi threading/tasking required to keep all the devices busy
    - CPU-GPU affinity matters on multi-socket nodes

OpenMP offload device control

- Device information routines
  - omp_get_default_device/omp_set_default_device
  - omp_get_num_devices/omp_get_device_num
- Device memory routines
  - omp_target_alloc/omp_target_free
  - omp_target_memcpy/omp_target_memcpy_async
- Device clause on target construct
  - #pragma omp target enter/exit data map(…) device(deviceID)
  - #pragma omp target teams distribute map(…) device(deviceID)

Later today: Hands-on!

- Using OpenMP (~20 min)
- Demo of OpenMP (~20 min)
  - OpenMP 101 and basics on ThetaGPU
- Hands-on Exercises (~20 min)

$ git clone https://github.com/argonne-lcf/CompPerfWorkshop-2021.git
$ cd CompPerfWorkshop-2021
$ cd 01_openmp

References and Resources

1. “Using OpenMP Effectively on Theta”
   – https://www.alcf.anl.gov/files/Using%20OpenMP%20Effectively%20on%20Theta.pdf
2. **Using OpenMP – The Next Step** by van der Pas, Stotzer and Terboven, MIT Press, 2017


Thank You!