Asynchronous OpenMP Offloading on NVIDIA GPUs

Shilei Tian[1], Johannes Doerfert[2], Barbara Chapman[1]

[1]Stony Brook University

[2]Argonne National Laboratory

Why Do We Need It

- NVIDIA GPU is getting more and more powerful so that it can perform multiple CUDA operations simultaneously.
- Fermi architecture can simultaneously support:
  - Up to **16** CUDA kernels on GPU
  - 2 `cudaMemcpyAsync`s
  - Computation on the CPU

Data source: https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf
Image source: https://www.techpowerup.com/243163/nvidia-waves-goodbye-to-their-fermi-graphics-cards#g243163-4

How to Do Asynchronous Offloading in OpenMP

```
void foo() {
#pragma omp target teams nowait
  { //kernel 1 }

#pragma omp target teams nowait
  { // kernel 2 }

  // Some CPU workloads
  // ...
  // Maybe some synchronization
}
```

How to Do Asynchronous Offloading in OpenMP

```
void foo() {
#pragma omp target teams nowait
  { //kernel 1 }

#pragma omp target teams nowait
  { // kernel 2 }

  // Some CPU workloads
  // ...
  // Maybe some synchronization
}
```

NOT Working!

Image source: https://www.danclarke.com/it-doesnt-work

How to Do Asynchronous Offloading in OpenMP – Another Way

```
void kernel1() {
#pragma omp target teams
  { // kernel 1}
}

void kernel2() {
#pragma omp target teams
  { // kernel 2 }
}

void foo() {
  pthread_create(tid1, ..., kernel1, ...);
  pthread_create(tid2, ..., kernel2, ...);

  // Some CPU workloads
  // ...
  // Maybe some synchronization here
}
```

How to Do Asynchronous Offloading in OpenMP – Another Way

```
void kernel1() {
#pragma omp target teams
  { // kernel 1}
}

void kernel2() {
#pragma omp target teams
  { // kernel 2 }
}

void foo() {
  pthread_create(tid1, ..., kernel1, ...);
  pthread_create(tid2, ..., kernel2, ...);

  // Some CPU workloads
  // ...
  // Maybe some synchronization here
}
```

Still NOT Working!

CUDA Stream

- A sequence of operations that execute *in issue-order* on the GPU

https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf

Default Stream

- Stream used when no stream is specified
- Completely synchronous w.r.t. host and device
  - As if `cudaDeviceSynchronize()` inserted before and after every CUDA operation
- Exceptions – asynchronous w.r.t. host
  - Kernel launches in the default stream
  - `cudaMemcpy*Async`
  - …

https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf

Default Stream

- Stream used when no stream is specified
- Completely synchronous w.r.t. host and device
  - As if `cudaDeviceSynchronize()` inserted before and after every CUDA operation
- Exceptions – asynchronous w.r.t. host
  - Kernel launches in the default stream
  - `cudaMemcpy*Async`
  - …

We're using it…

https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf

Concurrency

- CUDA operations in *different streams* may run concurrently

https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf

Multiple Streams!

http://users.wfu.edu/choss/CUDA/docs/Lecture%2013.pdf
https://steamcommunity.com/sharedfiles/filedetails/?id=798035133

Our Contribution

- Implement the support for multiple streams in LLVM OpenMP (merged)
- Make `nowait` clause work (WIP)

Our Contribution Cont.

```
void foo() {
#pragma omp target teams nowait
  { //kernel 1 }

#pragma omp target teams nowait
  { // kernel 2 }

  // Some CPU workloads
  // ...
  // Maybe some synchronization
}
```

https://upload.wikimedia.org/wikipedia/commons/8/88/Yellow_Happy.jpg

Multiple Streams in OpenMP

- Runtime library initializes 256 streams for each CUDA device. CUDA runtime will cap the number of “real” streams.
- For each CUDA operation, like memory copy and kernel launch, the runtime library selects a stream in a *round-robin* manner.

Micro Benchmark

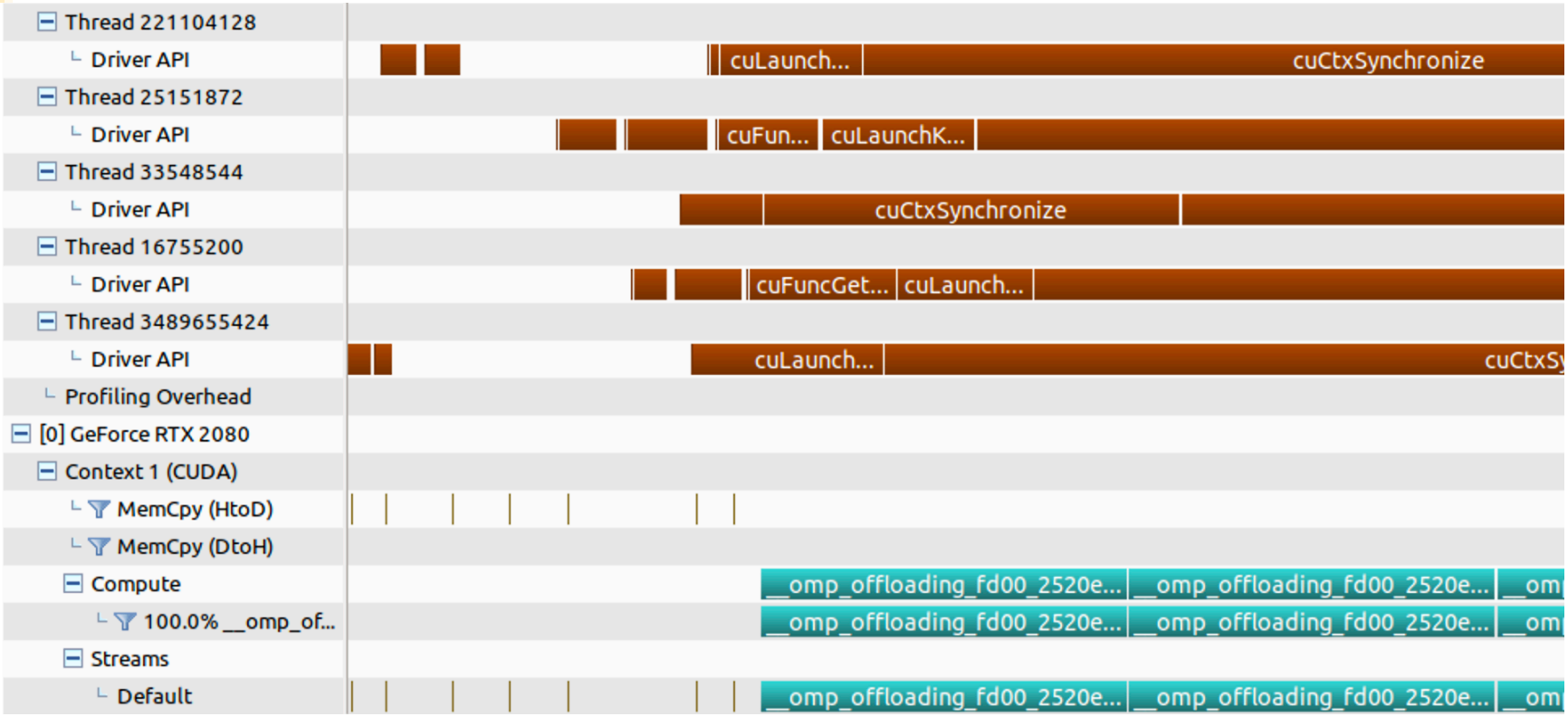
```
clock_t run(const size_t kernel_size) {
  int data[NUM][kernel_size]; // NUM=8

  const clock_t start = clock();

#pragma omp parallel for
  for (int i = 0; i < NUM; ++i) {
#pragma omp target teams distribute parallel for map(to:
data[i][0:kernel_size])
    for (size_t j = 0; j < kernel_size; ++j) {
      for (size_t k = 0; k < kernel_size; ++k) {
        data[i][j] += k;
      }
    }
  }

  return (clock() - start) * 1000 / CLOCKS_PER_SEC;
}
```

Hardware Configuration

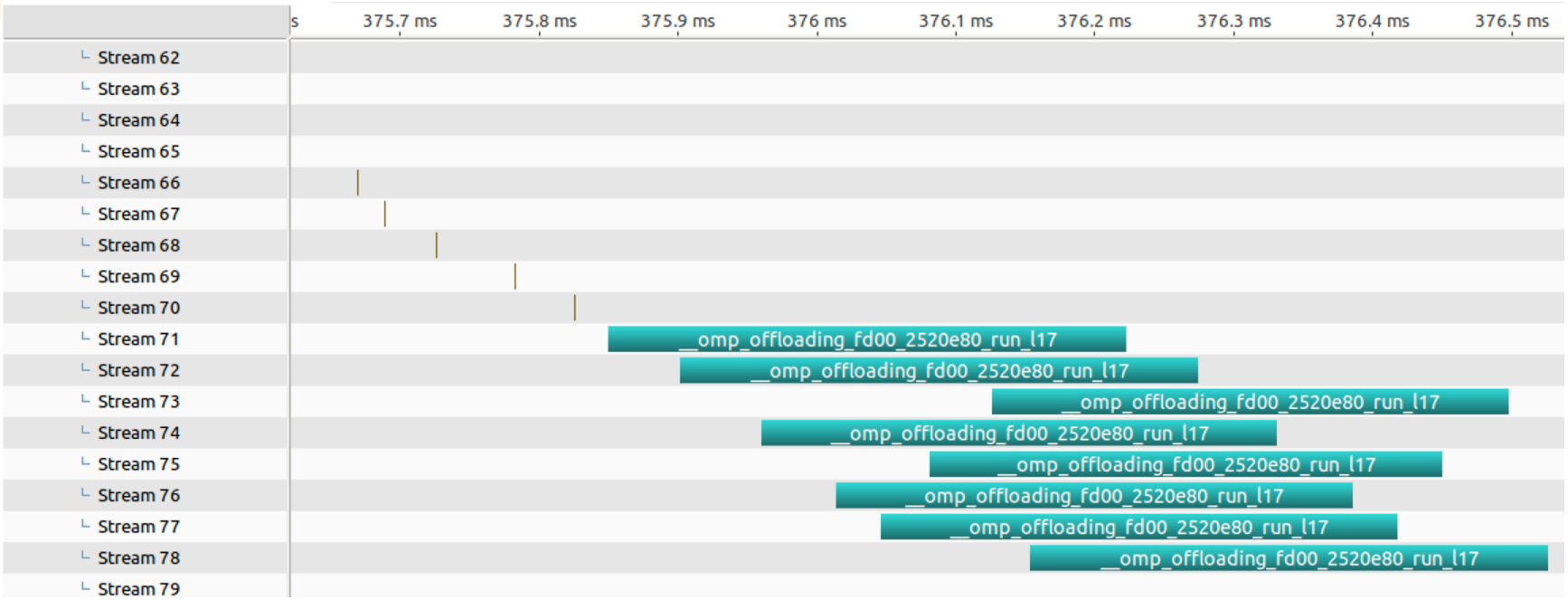
- CPU: Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz x 2
- GPU: NVIDIA GeForce RTX 2080 8GB

- OS: Ubuntu 18.04.01
- LLVM Code Base: 3ff4e2ee

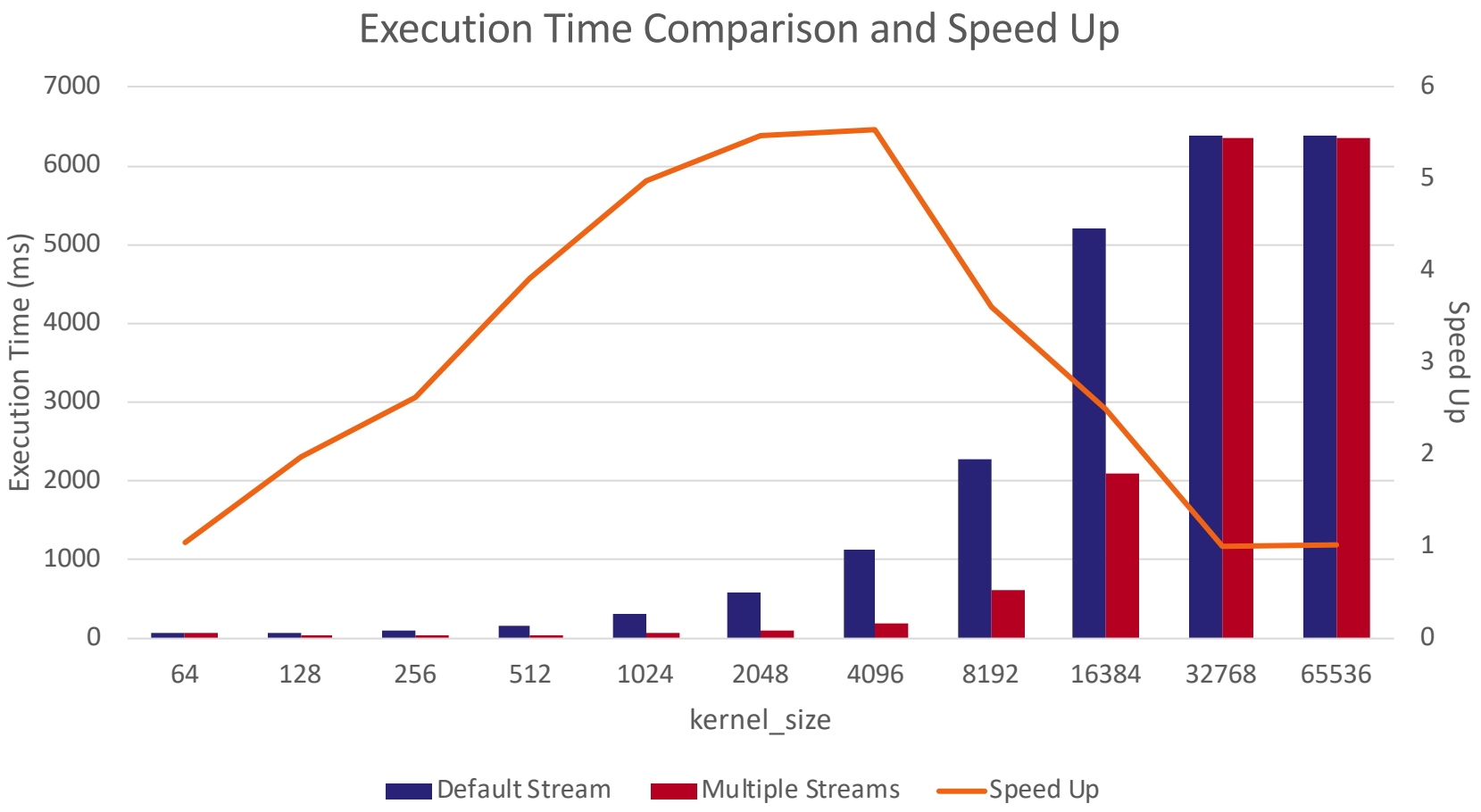
NVVP Result


Thread 221104128
Driver API
cuLaunch...
cuCtxSynchronize
Thread 25151872
Driver API
cuFun...
cuLaunchK...
Thread 33548544
Driver API
cuCtxSynchronize
Thread 16755200
Driver API
cuFuncGet...
cuLaunch...
Thread 3489655424
Driver API
cuLaunch...
Profiling Overhead
[0] GeForce RTX 2080
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
__omp_offloading_fd00_2520e...
100.0% __omp_of...
Streams
Default

NVVP Result Cont.



Performance Result



`nowait` Clause

- A `nowait` target is transformed into an OpenMP task. Dependencies will be handled by task.

`nowait` Clause

- A `nowait` target is transformed into an OpenMP task. Dependencies will be handled by task.

```
#pragma omp target depend(D) nowait
{ ... }
```

`nowait` Clause

- A `nowait` target is transformed into an OpenMP task. Dependencies will be handled by task.

```
#pragma omp target depend(D) nowait
{ ... }
```

```
#pragma omp task depend(D)
#pragma omp target
{ ... }
```

`nowait` Clause

- A `nowait` target is transformed into an OpenMP task. Dependencies will be handled by task.

```
#pragma omp target depend(D) nowait
{ ... }
```

```
#pragma omp task depend(D)
#pragma omp target
{ ... }
```

Is that it?

`nowait` Clause

- A `nowait` target is transformed into an OpenMP task. Dependencies will be handled by task.

```
#pragma omp target depend(D) nowait
{ ... }
```

```
#pragma omp task depend(D)
#pragma omp target
{ ... }
```

Is that it?

NO!

OpenMP Task

- A task region binds to the innermost enclosing parallel region.

OpenMP Task

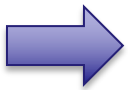
- A task region binds to the innermost enclosing parallel region.

```
#pragma omp task
{ ... }
```

OpenMP Task

- A task region binds to the innermost enclosing parallel region.

```
#pragma omp task
{ ... }
```

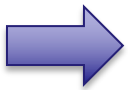


```
~~#pragma omp task~~
{ ... }
```

OpenMP Task

- A task region binds to the innermost enclosing parallel region.

```
#pragma omp task
{ ... }
```



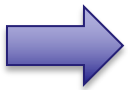
```
#pragma omp task
{ ... }
```

```
#pragma omp task
{ ... }
#pragma omp task
{ ... }
```

OpenMP Task

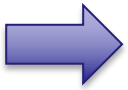
- A task region binds to the innermost enclosing parallel region.

```
#pragma omp task
{ ... }
```



```
~~#pragma omp task~~
{ ... }
```

```
#pragma omp task
{ ... }
#pragma omp task
{ ... }
```

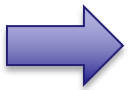


```
~~#pragma omp task~~
{ ... }
~~#pragma omp task~~
{ ... }
```

OpenMP Task

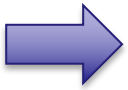
- A task region binds to the innermost enclosing parallel region.

```
#pragma omp task
{ ... }
```



```
~~#pragma omp task~~
{ ... }
```

```
#pragma omp task
{ ... }
#pragma omp task
{ ... }
```



```
~~#pragma omp task~~
{ ... }
~~#pragma omp task~~
{ ... }
```

Serialized

OpenMP Task

- A task region binds to the innermost enclosing parallel region.

```
#pragma omp target teams nowait
{ ... }
#pragma omp target teams nowait
{ ... }
```

OpenMP Task

- A task region binds to the innermost enclosing parallel region.

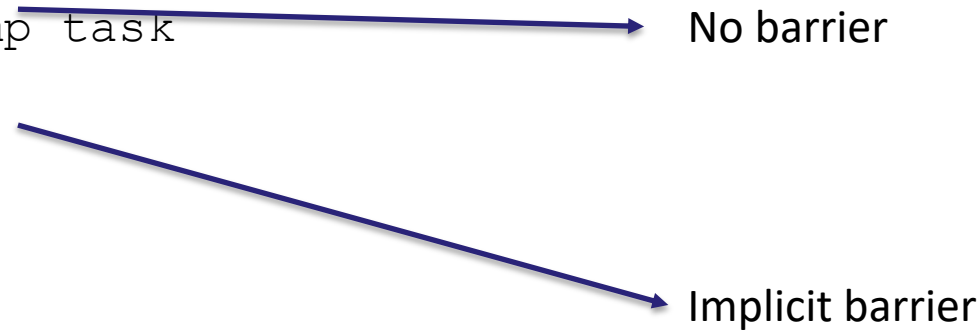
```
#pragma omp target teams nowait
{ ... }
#pragma omp target teams nowait
{ ... }
```

Serialized

OpenMP Task Cont.

- The right way to write asynchronous tasks is:

```
#pragma omp parallel master
{
#pragma omp task
   { ... }
#pragma omp task
   { ... }
}
```



It turns out…

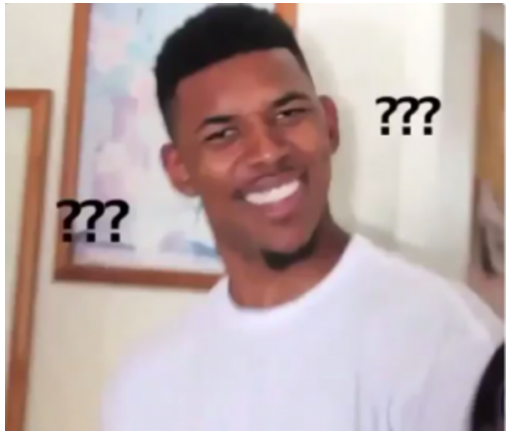
- We need to write in the following way if we want asynchronous offloading:

```
#pragma omp parallel master
{
#pragma omp target nowait
  { ... }
#pragma omp target nowait
  { ... }
}
```

It turns out…

- We need to write in the following way if we want asynchronous offloading:

```
#pragma omp parallel master
{
#pragma omp target nowait
  { ... }
#pragma omp target nowait
  { ... }
}
```



Unshackled Task

- An unshackled task is a task that is not bound to any parallel region.

Unshackled Task

- An unshackled task is a task that is not bound to any parallel region.

```
#pragma omp target depend(D1) nowait
{ // kernel 1 }
#pragma omp target depend(D2) nowait
{ // kernel 2 }
```

Unshackled Task

- An unshackled task is a task that is not bound to any parallel region.

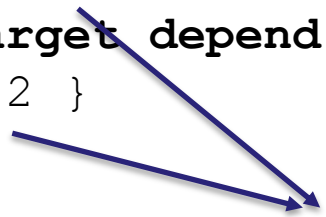
```
#pragma omp target depend(D1) nowait
{ // kernel 1 }
#pragma omp target depend(D2) nowait
{ // kernel 2 }
```

```
#pragma omp task depend(D1) unshackled
#pragma omp target
{ // kernel 1 }
#pragma omp task depend(D2) unshackled
#pragma omp target
{ // kernel 2 }
```

Unshackled Task Cont.

- An unshackled task is a task that is not bound to any parallel region.

```
#pragma omp parallel master
{
#pragma omp target depend(D1) nowait
  { // kernel 1 }
#pragma omp target depend(D2) nowait
  { // kernel 2 }
}
```



No implicit barrier!

Unshackled Task Cont.

- An unshackled task is a task that is not bound to any parallel region.

```
#pragma omp parallel master
{
#pragma omp target depend(D1) nowait
  { // kernel 1 }
#pragma omp target depend(D2) nowait
  { // kernel 2 }
}
```

No implicit barrier!

Current Status

- Code transformation to unshackled task (done)
- Runtime support for unshackled task (WIP)
- Will submit for code review afterwards

Future Work

- Optimize the selection of streams taking dependencies into account
  - For example, kernel A depends on kernel B. B and A can be scheduled into a same stream so that no need of any host side dependency process. It also works for data mapping.
- …

Stony Brook University

Argonne NATIONAL LABORATORY

Thank You!