



OpenMP[®]

UNVEILING THE POWER OF HETEROGENEOUS COMPUTING

A Brief Dive into Host and Target Tasks with OpenMP LLVM

LLVM CGO24

Rafael A. Herrera Guaitero

*PhD Candidate
University of Delaware*

Rodrigo Ceccato

*PhD Student
UNICAMP*

Rémy Neveu

*PhD Student
UNICAMP*

Dr. Jose M. Monsalve Diaz

*Postdoctoral Appointee
Argonne National Lab*

OUTLINE

INTRODUCTION

EXECUTION MODEL

EXAMPLE 1

REFERENCE MATERIAL

COMPILATION PIPELINE

THE LIBOMPTARGET LIBRARY

EXAMPLE 2

GETTING STARTED

HOW TO RUN, COMPILE AND DEBUG?

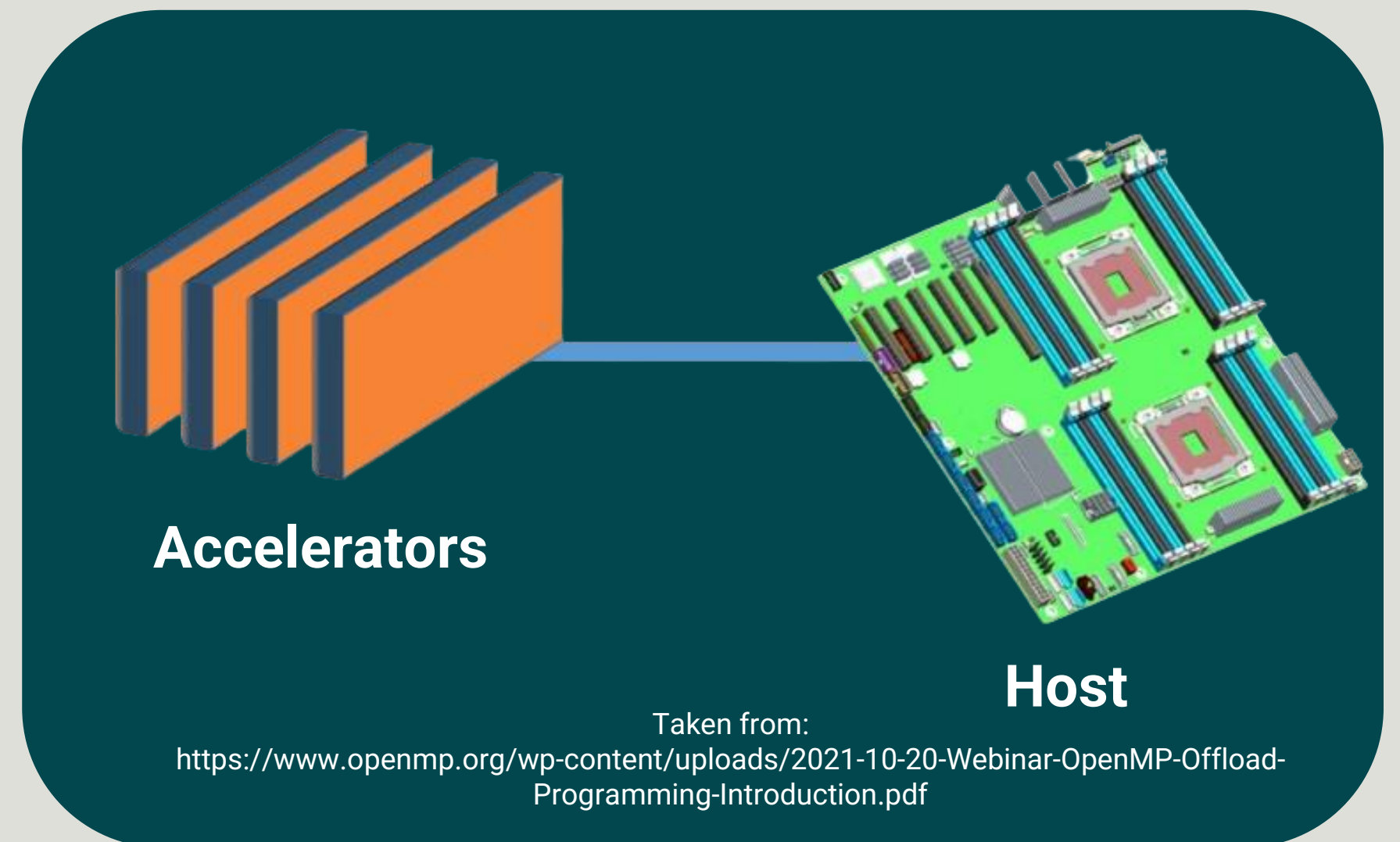
HOW TO GET INVOLVED?

THANK YOU

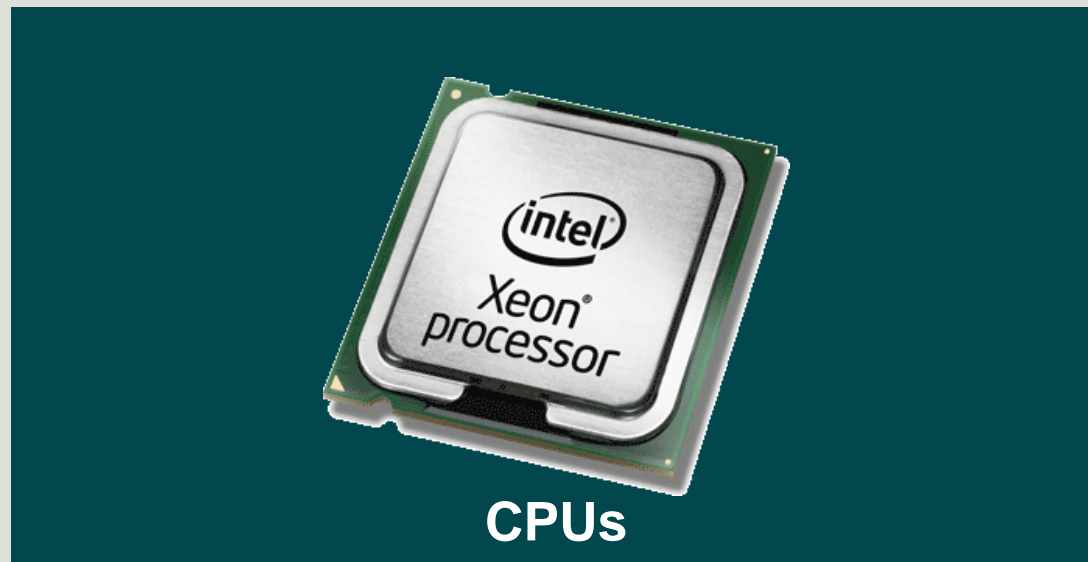
QUESTIONS

OpenMP Offloading

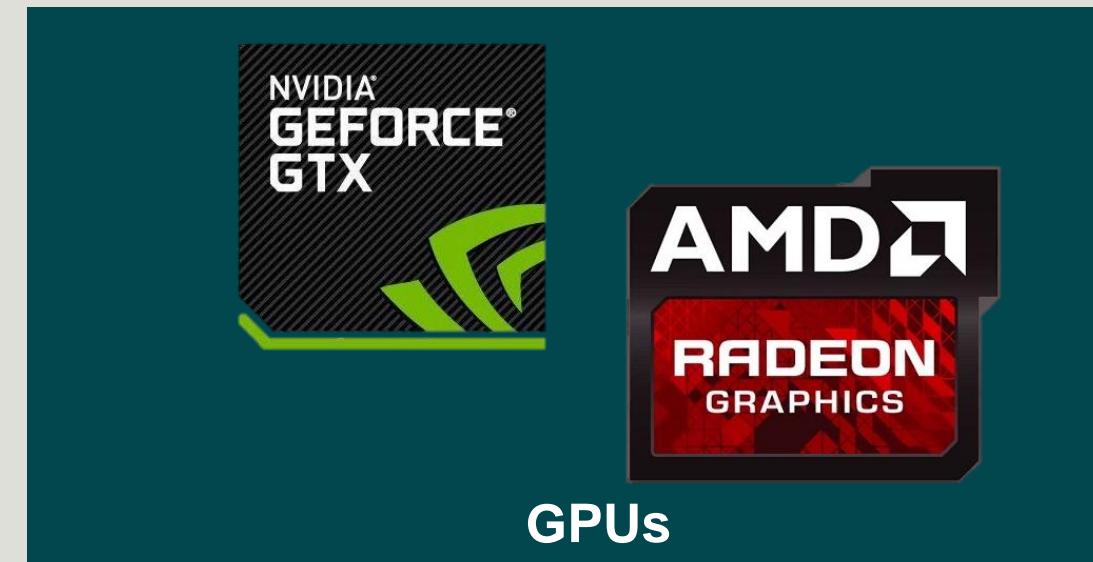
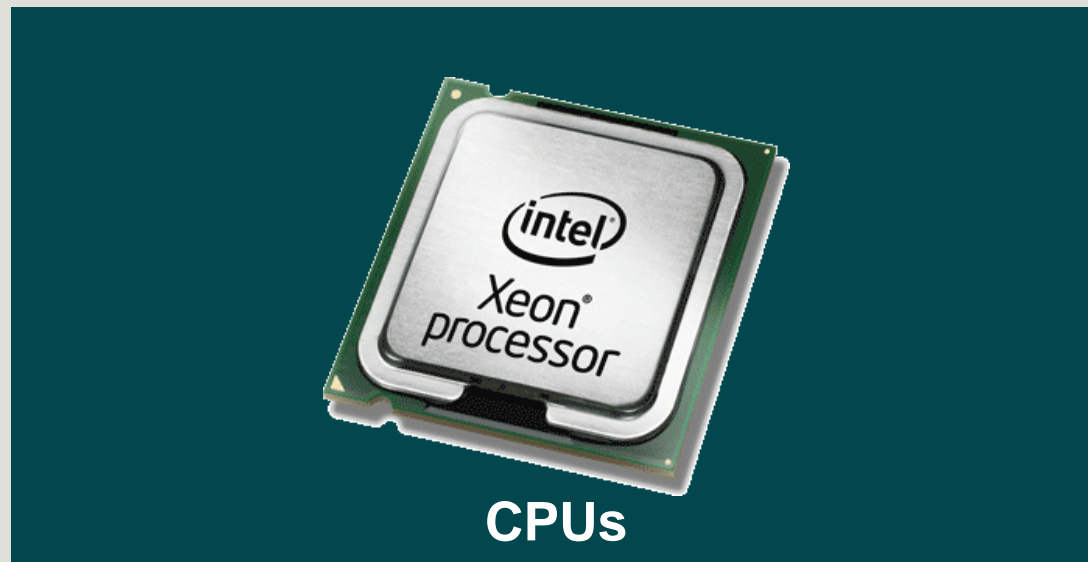
- Since specification 4.0, OpenMP allows execution on different coprocessors/accelerators
- The *'target'* construct maps variables to a device data environment and execute the construct on that device.



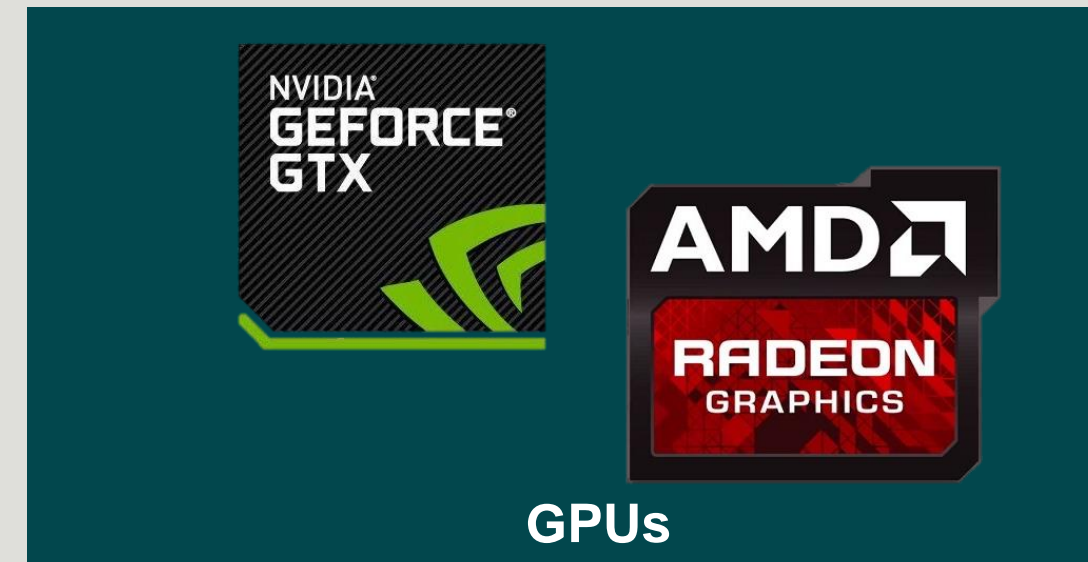
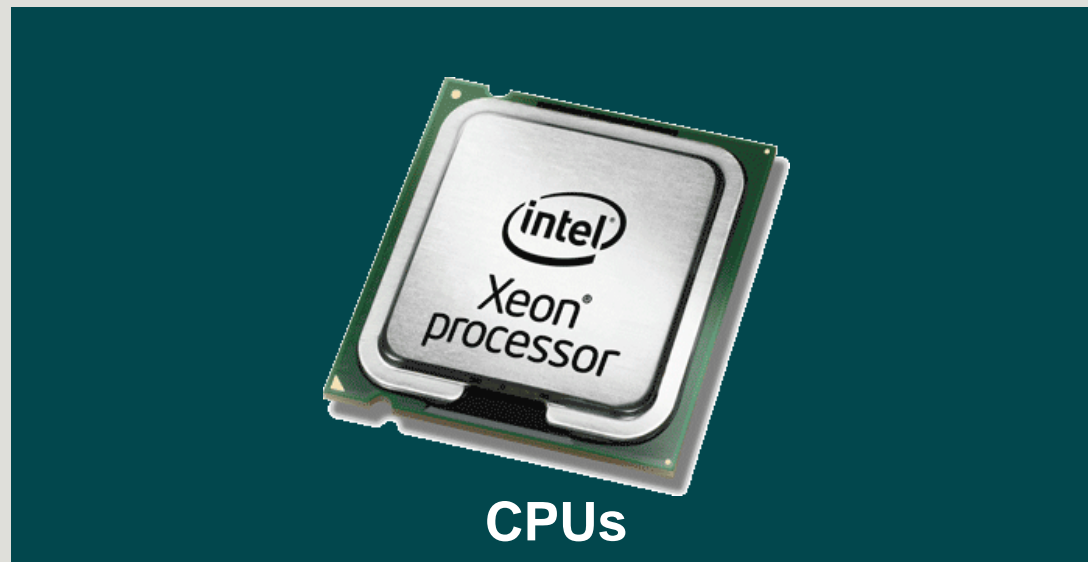
OpenMP and Heterogeneous Computing



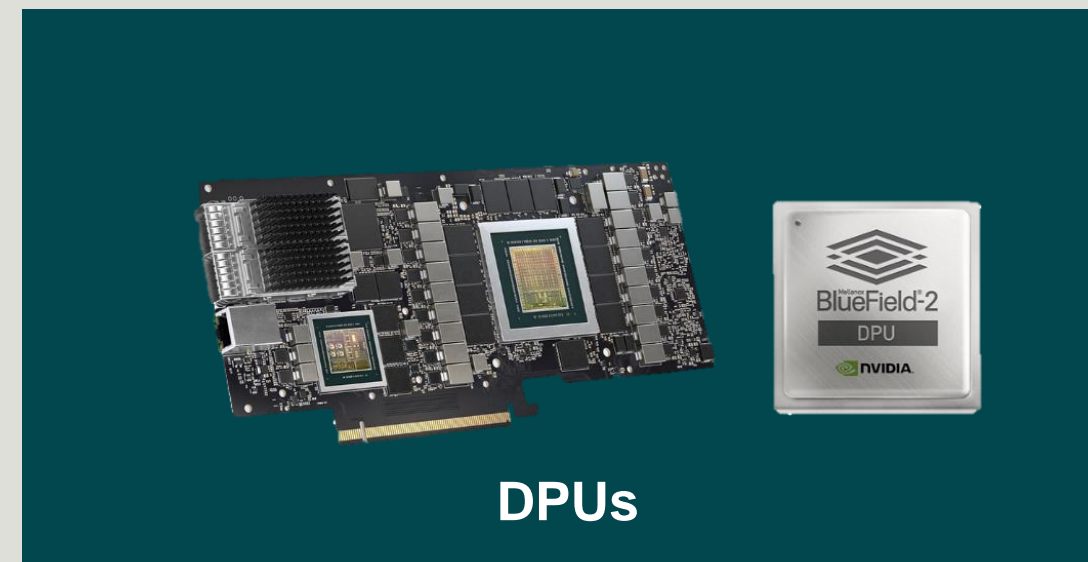
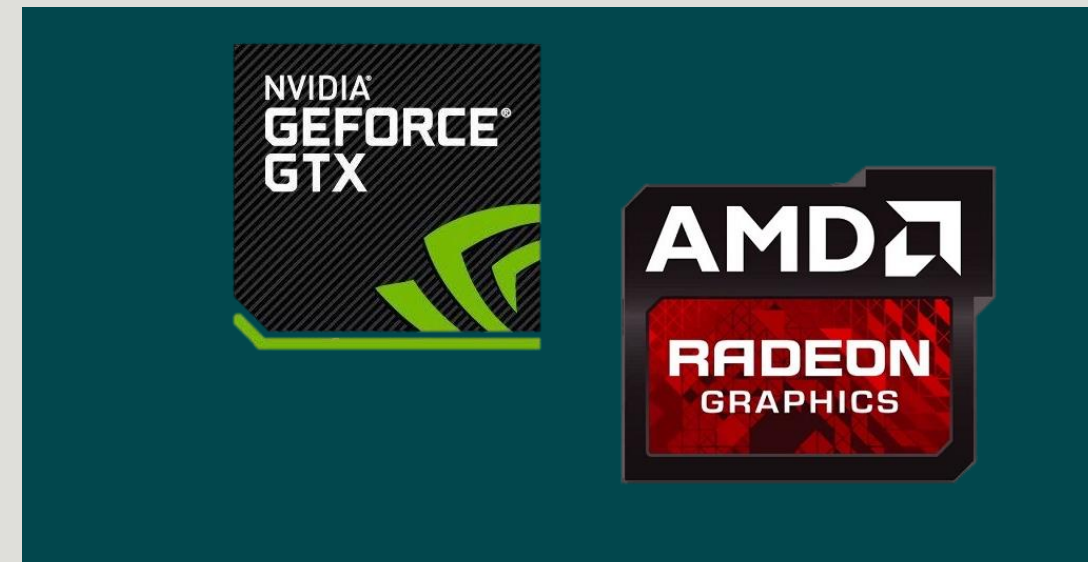
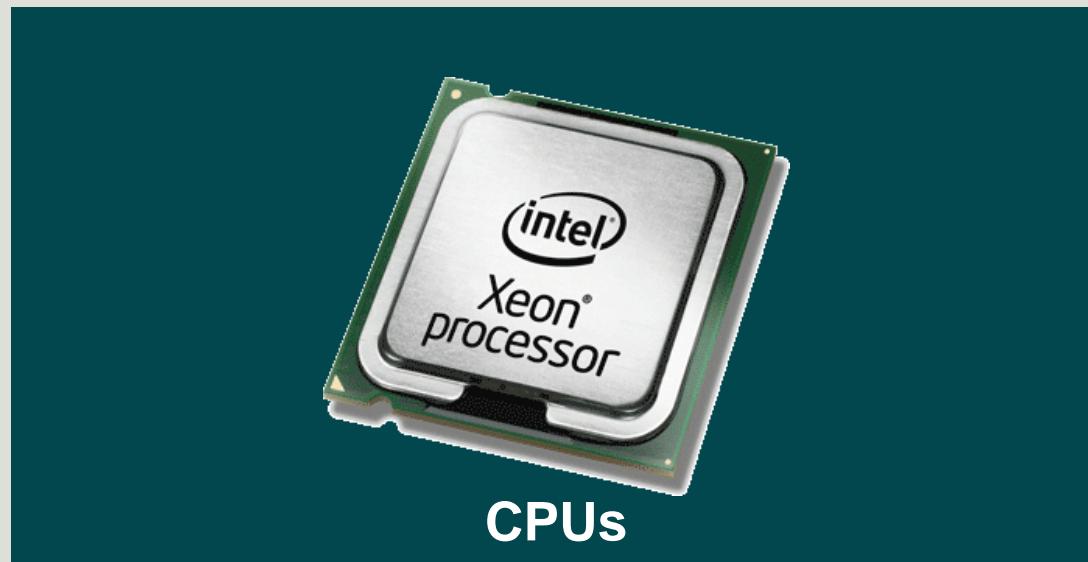
OpenMP and Heterogeneous Computing



OpenMP and Heterogeneous Computing



OpenMP and Heterogeneous Computing



OpenMP allows to:

Allocate device and host variables and control data movements

Control parallelism (e.g SIMD)

Set the order in which the host and device tasks are executed

Define synchronization points

Execution Model

1

the host creates the data environments on the device(s)

The execution on the device
is **host-centric**

Execution Model

1

the host creates the data environments on the device(s)

2

the host maps data to the device data environment

The execution on the device
is **host-centric**

Execution Model

1

the host creates the data environments on the device(s)

2

the host maps data to the device data environment

3

the host offloads OpenMP target regions to the target device to be executed

The execution on the device is **host-centric**

Execution Model

1

the host creates the data environments on the device(s)

2

the host maps data to the device data environment

3

the host offloads OpenMP target regions to the target device to be executed

4

the host transfers data from the device to the host

The execution on the device is **host-centric**

Execution Model

1

the host creates the data environments on the device(s)

2

the host maps data to the device data environment

3

the host offloads OpenMP target regions to the target device to be executed

4

the host transfers data from the device to the host

5

the host destroys the data environment on the device

The execution on the device is **host-centric**

Example: saxpy

```

void saxpy() {
    float a, x[SZ], y[SZ];
    double t = 0.0;
    double tb, te;
    tb = omp_get_wtime();
    #pragma omp target "map(tofrom:y[0:SZ])"
    for (int i = 0; i < SZ; i++) {
        y[i] = a * x[i] + y[i];
    }
    te = omp_get_wtime();
    t = te - tb;
    printf("Time of kernel: %lf\n", t);
}

```

The compiler identifies variables that are used in the target region.

All accessed arrays are copied from host to device and back

host

a
x[0:SZ]
y[0:SZ]

target

Synchronous Device Task

Presence check: only transfer if not yet allocated on the device.

host


x[0:SZ]
y[0:SZ]

Copying x back is not necessary: it was not changed.

- clang/LLVM: clang -fopenmp -fopenmp-targets=<target triple>
- GNU: gcc -fopenmp
- AMD ROCm: clang -fopenmp -offload-arch=gfx908
- NVIDIA: nvcc -mp=gpu -gpu=cc80
- Intel: icx -fiopenmp -fopenmp-targets=spir64
- IBM XL: xlc -qsmp -qoffload -qtgtarch=sm_70

Slide from Dr. Michael Klemm!


For more information about OpenMP Offloading



Intro to GPU Programming
with the OpenMP API


Dr.-Ing. Michael Klemm

Chief Executive Officer
OpenMP Architecture Review Board
Principal Member of Technical Staff
HPC Center of Excellence
AMD



OpenMP Webinar - Youtube


For more information about OpenMP Offloading



Intro to GPU Programming
with the OpenMP API

Dr.-Ing. Michael Klemm

Chief Executive Officer
OpenMP Architecture Review Board
Principal Member of Technical Staff
HPC Center of Excellence
AMD



OpenMP Webinar - Youtube




Jose Monsalve
(Argonne National Laboratory)





Parallel Programming with OpenMP 4.5 **Workshop**
Colombian 5 HPC Summer School 2022: Space Exploration

Cybercolombia 2022 - Youtube

OpenMP Offloading: Synchronous and Asynchronous Execution



 Stony Brook University  Argonne NATIONAL LABORATORY


Asynchronous OpenMP Offloading on NVIDIA GPUs

Shilei Tian¹, Johannes Doerfert², Barbara Chapman¹

¹Stony Brook University
²Argonne National Laboratory

CGO20 – LLVM Performance Workshop

OpenMP Offloading: Synchronous and Asynchronous Execution



Asynchronous OpenMP Offloading on NVIDIA GPUs

Shilei Tian¹, Johannes Doerfert², Barbara Chapman¹

¹Stony Brook University
²Argonne National Laboratory

CGO20 – LLVM Performance Workshop

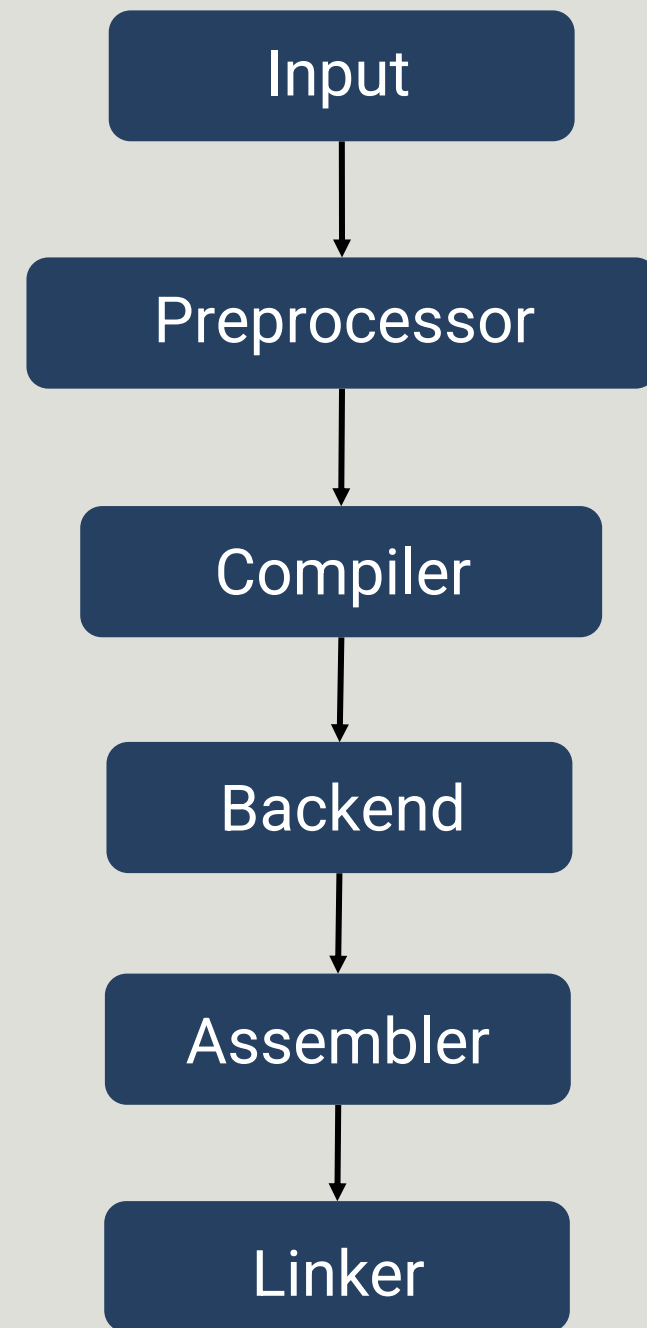
Automatic Asynchronous Execution of Synchronously Offloaded OpenMP Target Regions

Session: [LLVM-HPC2022: The Eighth Workshop on the LLVM Compiler Infrastructure in HPC](#)

Description: Use of heterogeneous architectures has steadily increased during the past decade. However, non-homogeneous systems present a challenge to the programming model as the execution models between CPU and accelerator might differ considerably. OpenMP, since version 4.0, has been trying to bridge this gap by allowing to offload a code block to a target device. Among the additions to the OpenMP offloading API since, the most notably probably is asynchronous execution between device and host. By default, offloaded regions are executed synchronously, thus the host thread blocks until their completion. The `nowait` clause allows work to overlap between the host and target device. However, `nowait` must be manually added by the user, along with the tasks data dependencies and appropriate synchronization to avoid race conditions, increasing the program complexity and developer burden.

SC22 – LLVM Workshop

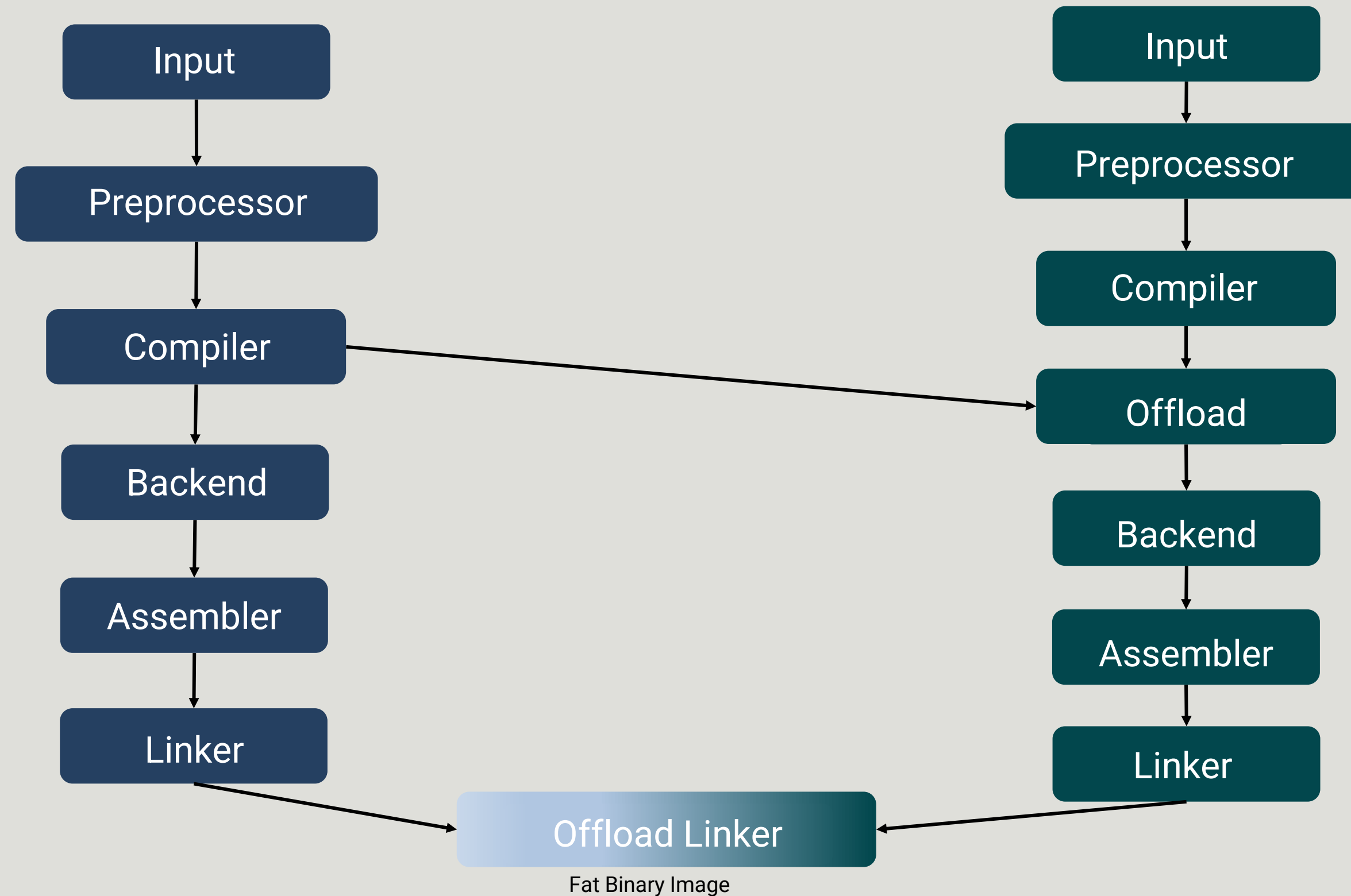
Compilation Pipeline for OpenMP Offloading



A MUST READ: Offloading Support for OpenMP in Clang and LLVM

Slide originally from Jose Monsalve and Johannes Doerfter

Compilation Pipeline for OpenMP Offloading



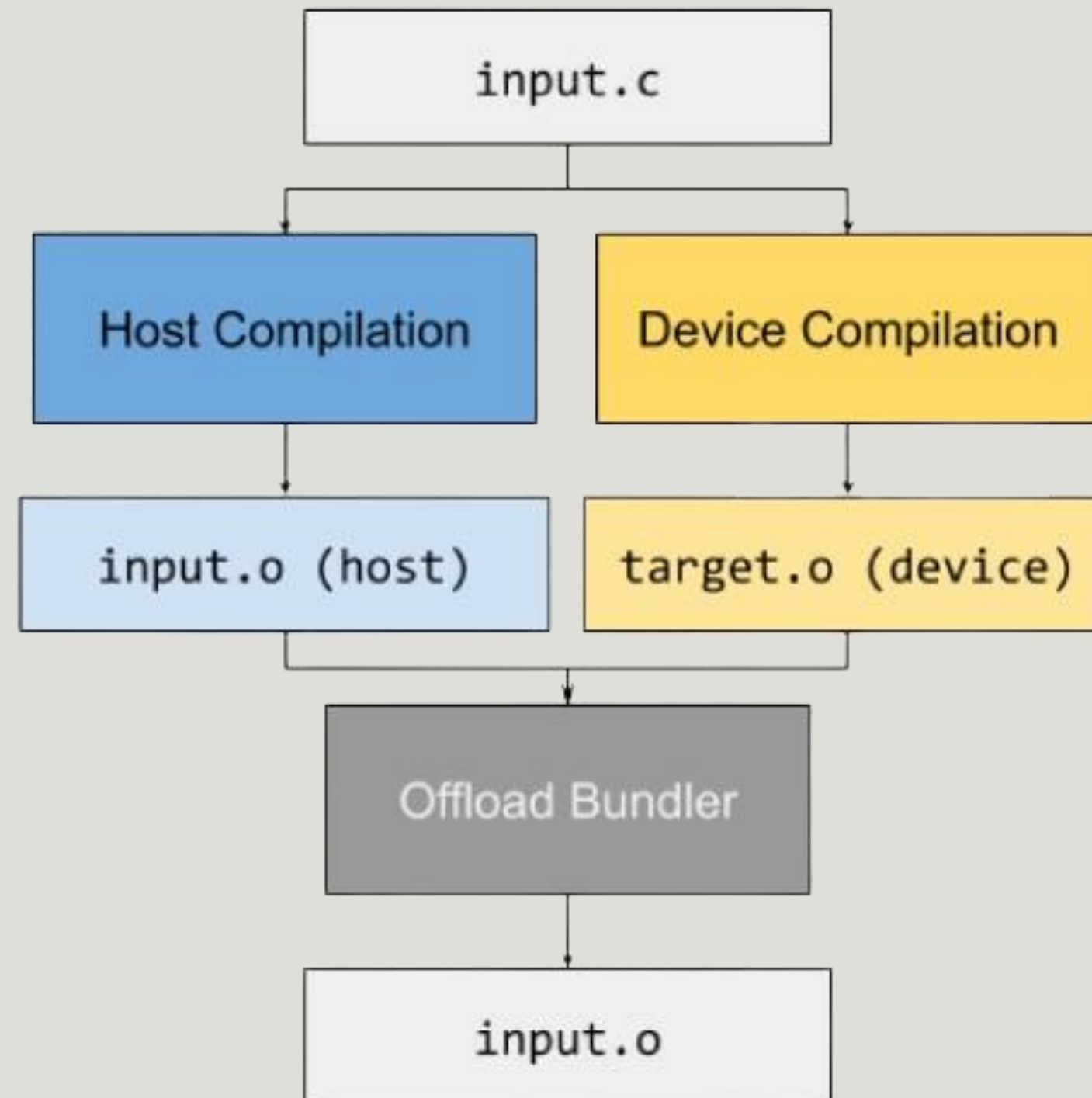
A MUST READ: Offloading Support for OpenMP in Clang and LLVM

Slide originally from Jose Monsalve and Johannes Doerfter

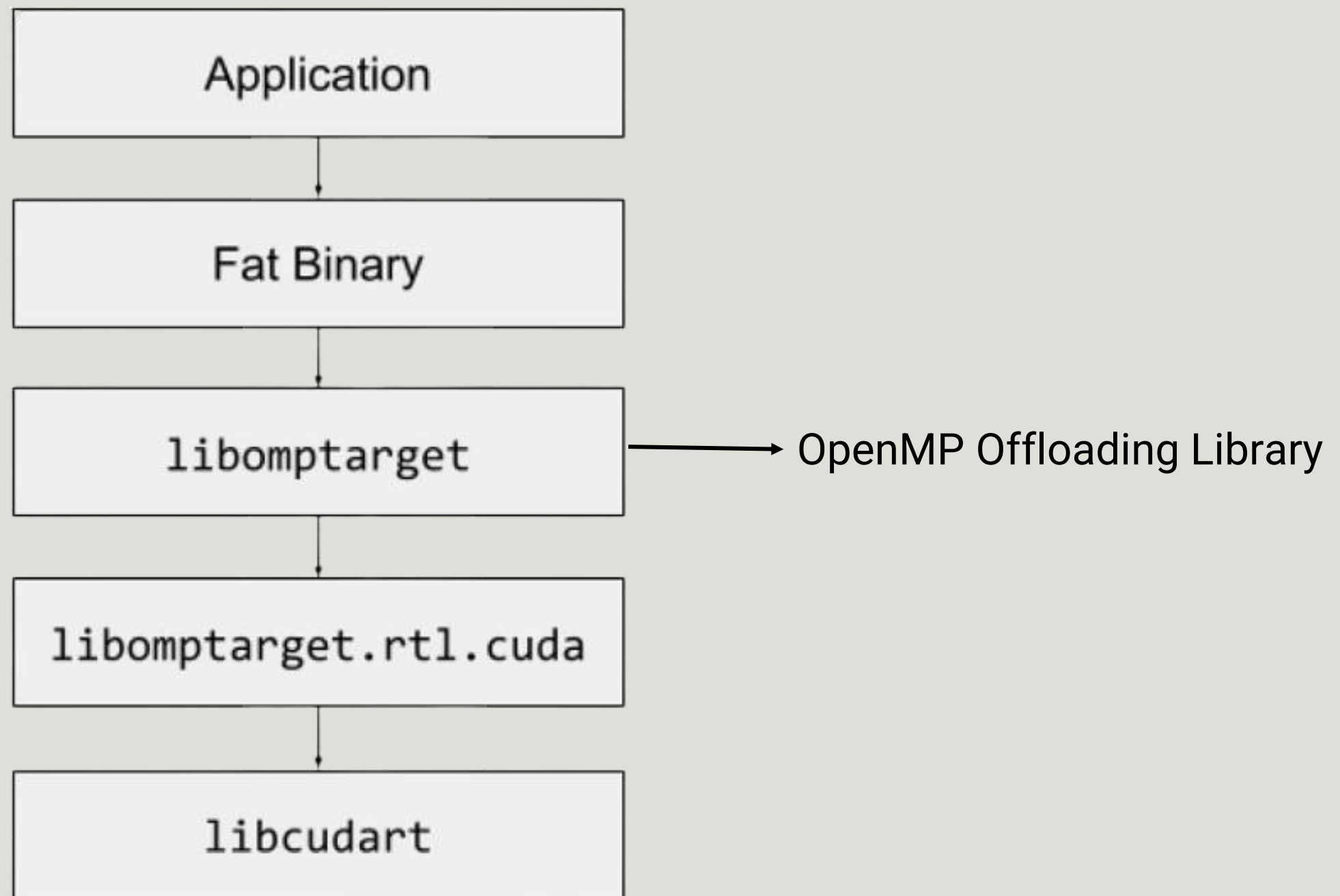
host-OpenMP

device-OpenMP

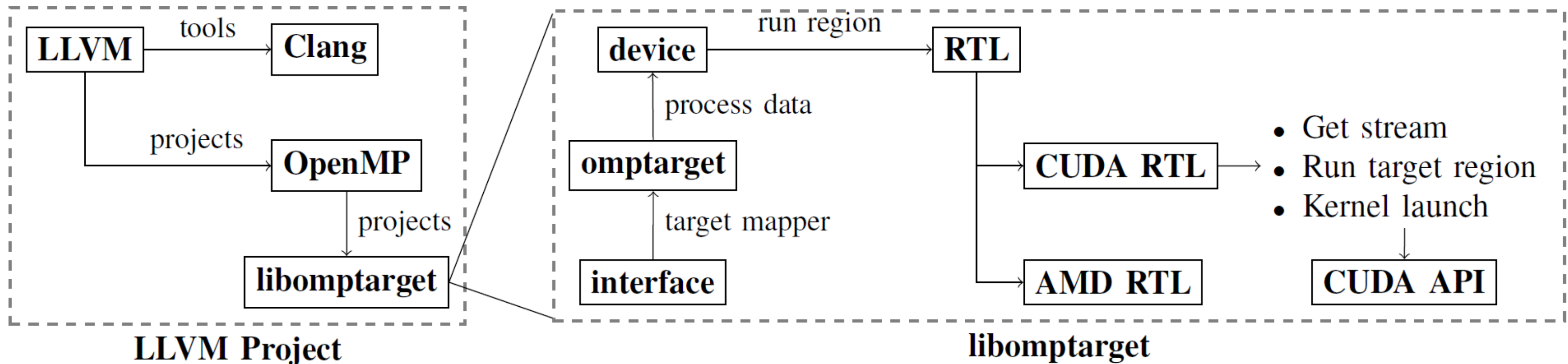
Compilation Pipeline for OpenMP Offloading



Example for the nvptx target.




libomptarget library






For more information...



[RFC] Introducing `llvm-project/offload`
LLVM Project

 **jdoerfert** 3  Oct 2023






LLVM has a [growing community around, and interest in, offloading](#) ⁶¹. People target other CPUs on the system, GPUs, AI accelerators, FPGAs, remote threads or machines, and probably more things. The base principles are always the same, yet we have only loosely connected approaches to tackle this. Everyone has to write, maintain, and distribute their own flavor of an *offload runtime* and the connected components. This is certainly not the situation we want as a community.

[RFC] Introducing `llvm-project/offload` - Discourse

RESEARCH-ARTICLE   

Breaking the Vendor Lock: Performance Portable Programming through OpenMP as Target Independent Runtime Layer

Authors:  Johannes Doerfert,  Marc Jasper,  Joseph Huber,  Khaled Abdelaal,  Giorgis Georgakoudis,  Thomas Scogland,  Konstantinos Parasyris [Authors Info & Claims](#)

Research paper

Unifying LLVM/OpenMP offload across GPU vendors and leveraging async operations on AMD accelerators

Kevin Sala Penades
Research Aide Technical @ ANL,
PhD Student + Junior Research Engineer @ BSC

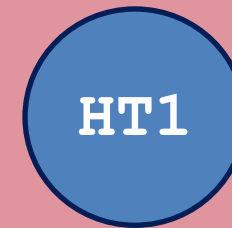
Kevin's Presentation - Youtube

Host and Device Tasks

```
1  #include <stdio.h>
2
3  void host_task1(int x);
4  int host_task2(int x, int y);
5  void device_task(int y);
6
7  void test(int x, int y, int output) {
8      #pragma omp parallel
9      {
10         #pragma omp single
11         {
12             #pragma omp task depend(out: x)
13             host_task1(x);
14         }
15         #pragma omp target map(from:y) nowait depend(inout: y)
16         device_task(y);
17
18         #pragma omp task depend(in: x, y)
19         output = host_task2(x, y);
20
21         #pragma omp taskwait
22         printf("The results is: %d", output);
23     }
24 }
25 }
```

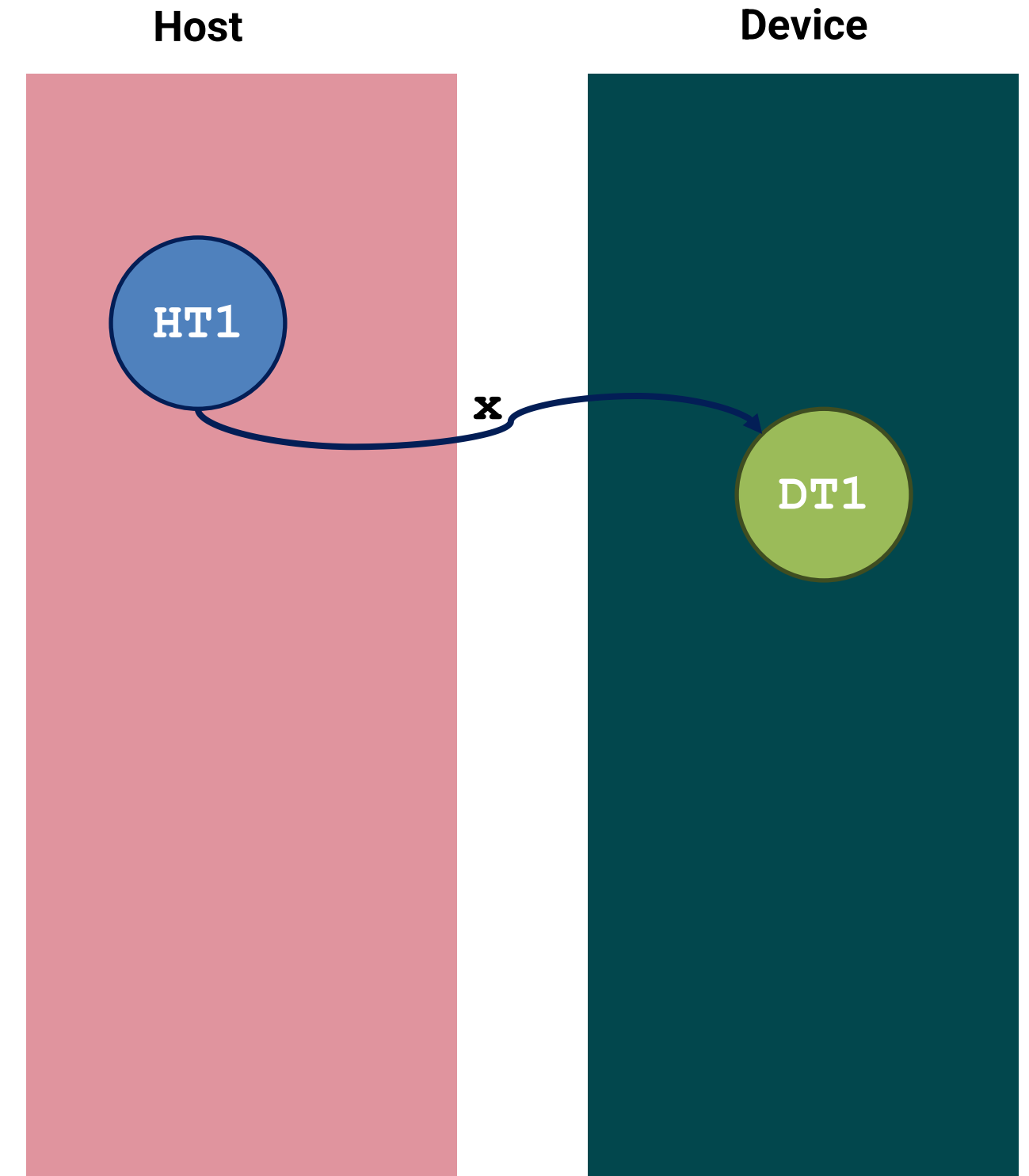
HT1

Host



Host and Device Tasks

```
1  #include <stdio.h>
2
3  void host_task1(int x);
4  int host_task2(int x, int y);
5  void device_task(int y);
6
7  void test(int x, int y, int output) {
8      #pragma omp parallel
9      {
10         #pragma omp single
11         {
12             #pragma omp task depend(out: x)           HT1
13             host_task1(x);
14         }
15         #pragma omp target map(from:y) nowait depend(inout: y) DT1
16         device_task(y);
17     }
18     #pragma omp task depend(in: x, y)
19     output = host_task2(x, y);
20
21     #pragma omp taskwait
22     printf("The results is: %d", output);
23 }
24 }
25 }
```



The target region with the 'nowait' clause is embedded in a Host task

Host and Device Tasks

```
98 %19 = getelementptr inbounds [1 x ptr], ptr %.offload_baseptrs, i32 0, i32 0
99 store ptr %1, ptr %19, align 8
100 %20 = getelementptr inbounds [1 x ptr], ptr %.offload_ptrs, i32 0, i32 0
101 store ptr %1, ptr %20, align 8
102 %21 = getelementptr inbounds [1 x ptr], ptr %.offload_mappers, i64 0, i64 0
103 store ptr null, ptr %21, align 8
104 %22 = getelementptr inbounds [1 x ptr], ptr %.offload_baseptrs, i32 0, i32 0
105 %23 = getelementptr inbounds [1 x ptr], ptr %.offload_ptrs, i32 0, i32 0
106 %24 = getelementptr inbounds %struct.anon.0, ptr %agg.captured1, i32 0, i32 0
107 store ptr %1, ptr %24, align 8
108 %25 = call ptr @__kmpc_omp_target_task_alloc(ptr @1, i32 %4, i32 1, i64 64, i64 8, ptr @.omp_task_entry..2, i64 -
109 %26 = getelementptr inbounds %struct.kmp_task_t_with_privates.1, ptr %25, i32 0, i32 0
110 %27 = getelementptr inbounds %struct.kmp_task_t, ptr %26, i32 0, i32 0
111 %28 = load ptr, ptr %27, align 8
112 call void @llvm.memcpy.p0.p0.i64(ptr align 8 %28, ptr align 8 %agg.captured1, i64 8, i1 false)
113 %29 = getelementptr inbounds %struct.kmp_task_t_with_privates.1, ptr %25, i32 0, i32 1
114 %30 = getelementptr inbounds %struct..kmp_privates.t, ptr %29, i32 0, i32 0
115 call void @llvm.memcpy.p0.p0.i64(ptr align 8 %30, ptr align 8 %22, i64 8, i1 false)
116 %31 = getelementptr inbounds %struct..kmp_privates.t, ptr %29, i32 0, i32 1
117 call void @llvm.memcpy.p0.p0.i64(ptr align 8 %31, ptr align 8 %23, i64 8, i1 false)
118 %32 = getelementptr inbounds %struct..kmp_privates.t, ptr %29, i32 0, i32 2
119 call void @llvm.memcpy.p0.p0.i64(ptr align 8 %32, ptr align 8 @.offload_sizes, i64 8, i1 false)
120 %33 = getelementptr inbounds [1 x %struct.kmp_depend_info], ptr %.dep.arr.addr2, i64 0, i64 0
121 %34 = ptrtoint ptr %1 to i64
122 %35 = getelementptr %struct.kmp_depend_info, ptr %33, i64 0
123 %36 = getelementptr inbounds %struct.kmp_depend_info, ptr %35, i32 0, i32 0
124 store i64 %34, ptr %36, align 8
125 %37 = getelementptr inbounds %struct.kmp_depend_info, ptr %35, i32 0, i32 1
126 store i64 4, ptr %37, align 8
127 %38 = getelementptr inbounds %struct.kmp_depend_info, ptr %35, i32 0, i32 2
128 store i8 3, ptr %38, align 8
129 store i64 1, ptr %dep_counter.addr3, align 8
130 %39 = call i32 @__kmpc_omp_task_with_deps(ptr @1, i32 %4, ptr %25, i32 1, ptr %33, i32 0, ptr null)
```

A new task with dependencies
is allocated

Host LLVM IR

```
98 %19 = getelementptr inbounds [1 x ptr], ptr %offload_baseptrs, i32 0, i32 0
99 store ptr %1, ptr %19, align 8
100 %20 = getelementptr inbounds [1 x ptr], ptr %offload_ptrs, i32 0, i32 0
101 store ptr %1, ptr %20, align 8
102 %21 = getelementptr inbounds [1 x ptr], ptr %offload_mappers, i64 0, i64 0
103 store ptr null, ptr %21, align 8
104 %22 = getelementptr inbounds [1 x ptr], ptr %offload_baseptrs, i32 0, i32 0
105 %23 = getelementptr inbounds [1 x ptr], ptr %offload_ptrs, i32 0, i32 0
106 %24 = getelementptr inbounds %struct.anon.0, ptr %agg.captured1, i32 0, i32 0
107 store ptr %1, ptr %24, align 8
108 %25 = call ptr @__kmpc_omp_target_task_alloc(ptr @1, i32 %4, i32 1, i64 64, i64 8, ptr @.omp_task_entry..2, i64 -
109 %26 = getelementptr inbounds %struct.kmp_task_t_with_privates.1, ptr %25, i32 0, i32 0
110 %27 = getelementptr inbounds %struct.kmp_task_t, ptr %26, i32 0, i32 0
111 %28 = load ptr, ptr %27, align 8
112 call void @llvm.memcpy.p0.p0.i64(ptr align 8 %28, ptr align 8 %agg.captured1, i64 8, i1 false)
113 %29 = getelementptr inbounds %struct.kmp_task_t_with_privates.1, ptr %25, i32 0, i32 1
114 %30 = getelementptr inbounds %struct.kmp_privates.t, ptr %29, i32 0, i32 0
115 call void @llvm.memcpy.p0.p0.i64(ptr align 8 %30, ptr align 8 %22, i64 8, i1 false)
116 %31 = getelementptr inbounds %struct.kmp_privates.t, ptr %29, i32 0, i32 1
117 call void @llvm.memcpy.p0.p0.i64(ptr align 8 %31, ptr align 8 %23, i64 8, i1 false)
118 %32 = getelementptr inbounds %struct.kmp_privates.t, ptr %29, i32 0, i32 2
119 call void @llvm.memcpy.p0.p0.i64(ptr align 8 %32, ptr align 8 @.offload_sizes, i64 8, i1 false)
120 %33 = getelementptr inbounds [1 x %struct.kmp_depend_info], ptr %dep.arr.addr2, i64 0, i64 0
121 %34 = ptrtoint ptr %1 to i64
122 %35 = getelementptr %struct.kmp_depend_info, ptr %33, i64 0
123 %36 = getelementptr inbounds %struct.kmp_depend_info, ptr %35, i32 0, i32 0
124 store i64 %34, ptr %36, align 8
125 %37 = getelementptr inbounds %struct.kmp_depend_info, ptr %35, i32 0, i32 1
126 store i64 4, ptr %37, align 8
127 %38 = getelementptr inbounds %struct.kmp_depend_info, ptr %35, i32 0, i32 2
128 store i8 3, ptr %38, align 8
129 store i64 1, ptr %dep_counter.addr3, align 8
130 %39 = call i32 @__kmpc_omp_task_with_deps(ptr @1, i32 %4, ptr %25, i32 1, ptr %33, i32 0, ptr null)
```

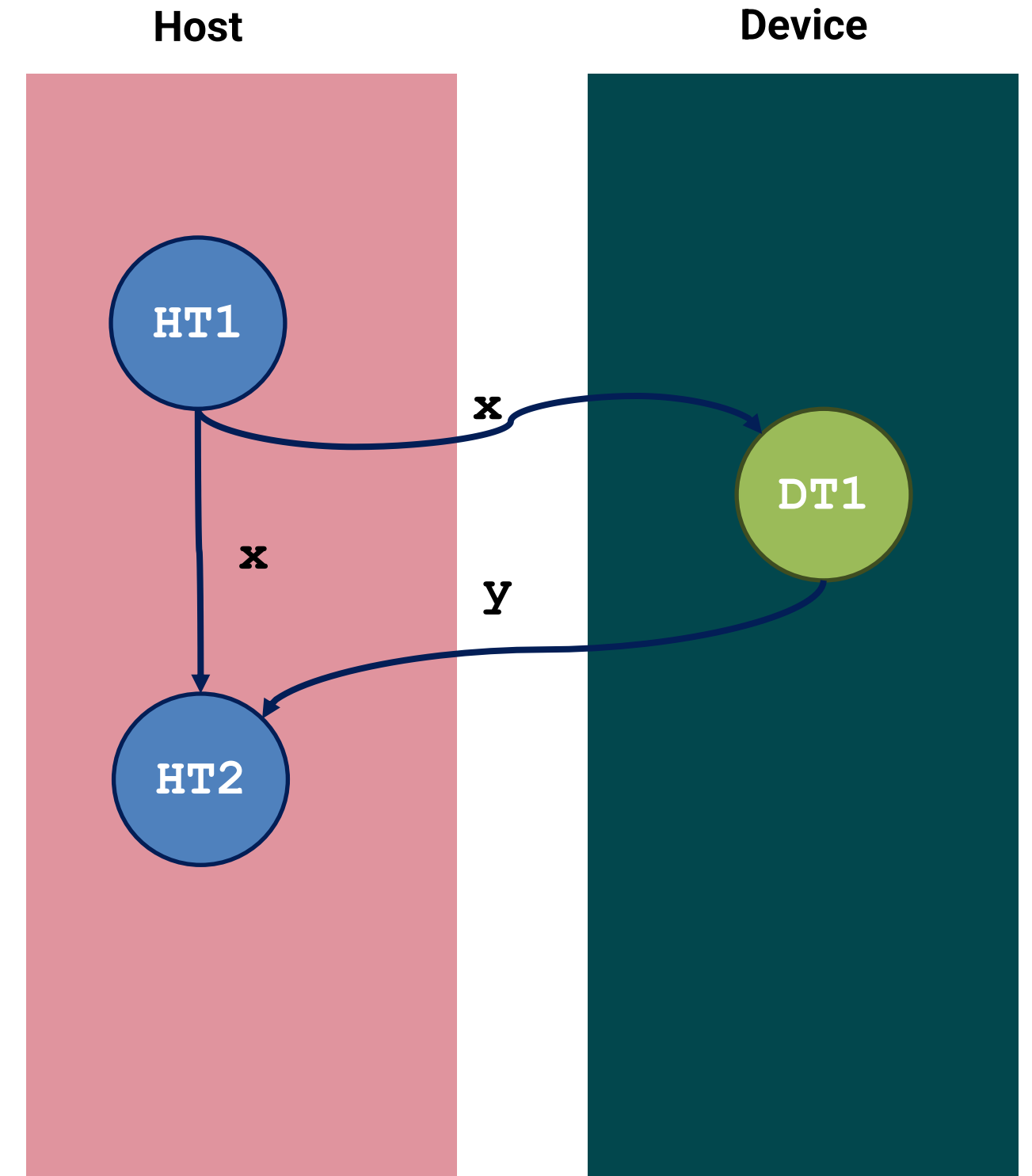
A new task with dependencies is allocated

Device Task

```
store [3 x i32] [i32 -1, i32 0, i32 0], ptr %25, align 4, !noalias !30
%26 = getelementptr inbounds %struct.__tgt_kernel_arguments, ptr %kernel_args.i, i32 0, i32 11
store [3 x i32] zeroinitializer, ptr %26, align 4, !noalias !30
%27 = getelementptr inbounds %struct.__tgt_kernel_arguments, ptr %kernel_args.i, i32 0, i32 12
store i32 0, ptr %27, align 4, !noalias !30
%28 = call i32 @__tgt_target_kernel(ptr @1, i64 -1, i32 -1, i32 0, ptr @.__omp_offloading_10302_bd3a9_test_l15.region_id, ptr %kernel_args.i)
%29 = icmp ne i32 %28, 0
br i1 %29, label %omp_offload.failed.i, label %omp_outlined..1.exit
```

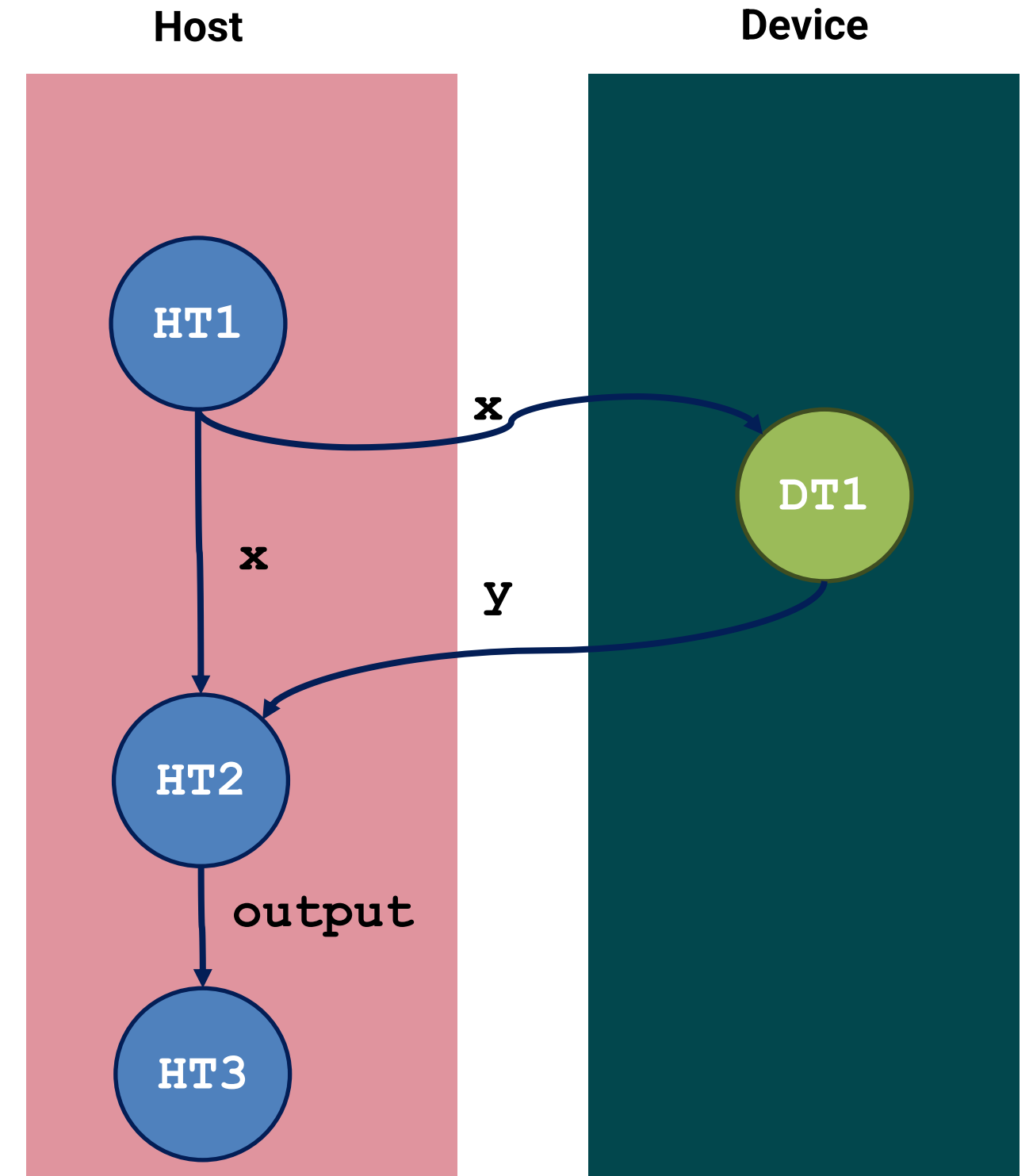
Host and Device Tasks

```
1  #include <stdio.h>
2
3  void host_task1(int x);
4  int host_task2(int x, int y);
5  void device_task(int y);
6
7  void test(int x, int y, int output) {
8      #pragma omp parallel
9      {
10         #pragma omp single
11         {
12             #pragma omp task depend(out: x)           HT1
13             host_task1(x);
14
15             #pragma omp target map(from:y) nowait depend(inout: y) DT1
16             device_task(y);
17
18             #pragma omp task depend(in: x, y)         HT2
19             output = host_task2(x, y);
20
21             #pragma omp taskwait
22             printf("The results is: %d", output);
23         }
24     }
25 }
```



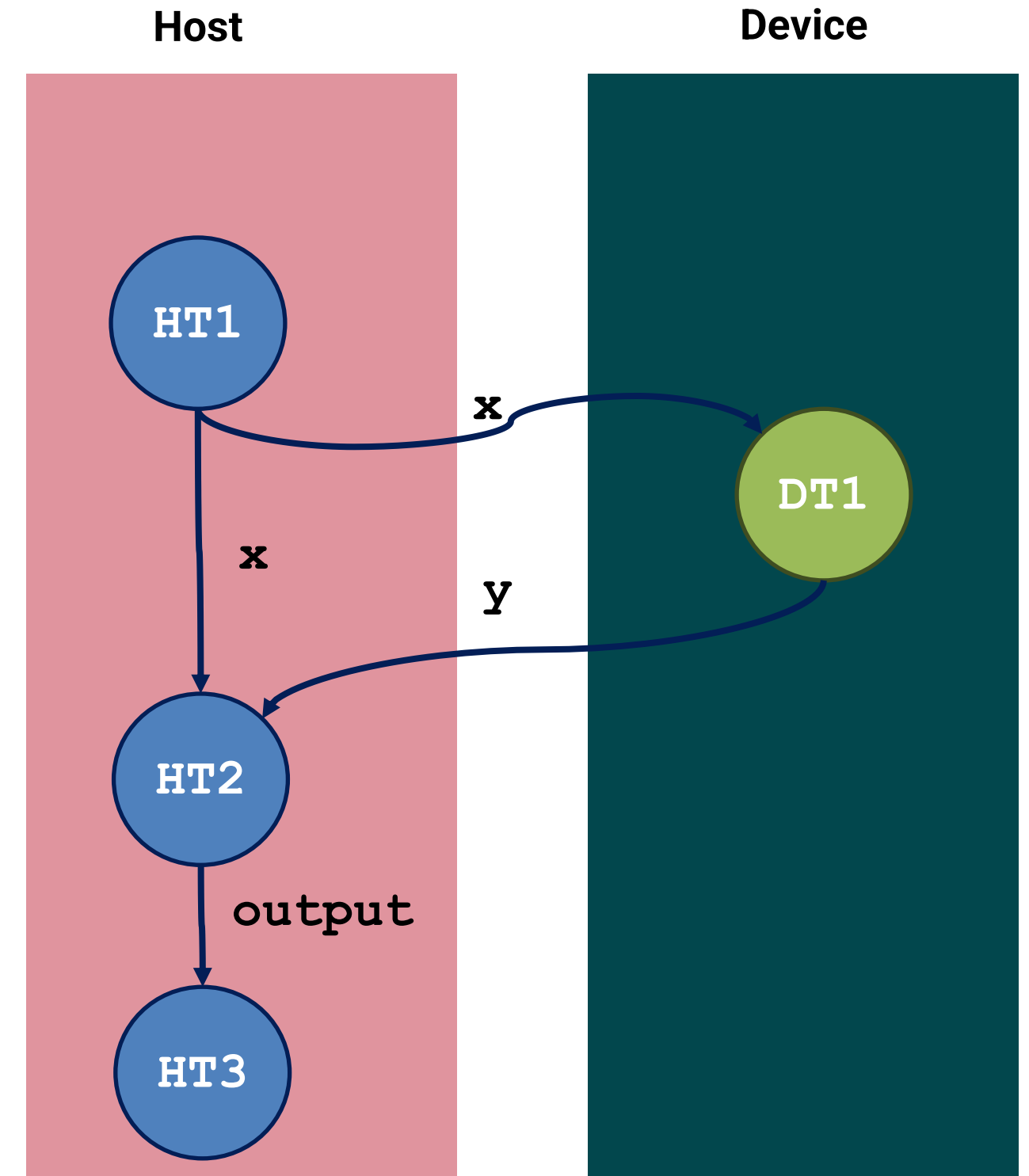
Host and Device Tasks

```
1  #include <stdio.h>
2
3  void host_task1(int x);
4  int host_task2(int x, int y);
5  void device_task(int y);
6
7  void test(int x, int y, int output) {
8      #pragma omp parallel
9      {
10         #pragma omp single
11         {
12             #pragma omp task depend(out: x)           HT1
13             host_task1(x);
14
15             #pragma omp target map(from:y) nowait depend(inout: y) DT1
16             device_task(y);
17
18             #pragma omp task depend(in: x, y)         HT2
19             output = host_task2(x, y);
20
21             #pragma omp taskwait
22             printf("The results is: %d", output);     HT3
23         }
24     }
25 }
```






Host and Device Tasks

```
1  #include <stdio.h>
2
3  void host_task1(int x);
4  int host_task2(int x, int y);
5  void device_task(int y);
6
7  void test(int x, int y, int output) {
8      #pragma omp parallel
9      {
10         #pragma omp single
11         {
12             #pragma omp task depend(out: x)           HT1
13             host_task1(x);
14
15             #pragma omp target map(from:y) nowait depend(inout: y) DT1
16             device_task(y);
17
18             #pragma omp task depend(in: x, y)         HT2
19             output = host_task2(x, y);
20
21             #pragma omp taskwait
22             printf("The results is: %d", output);    HT3
23         }
24     }
25 }
```



For more information...




Taskgraph: A Low Contention OpenMP Tasking Framework

Chenle Yu , Sara Royuela , Eduardo Quiñones 

Research Paper - BSC

For more information...

Taskgraph: A Low Contention OpenMP Tasking Framework

Chenle Yu , Sara Royuela , Eduardo Quiñones 

Research Paper - BSC




Quick explanation of the LLVM's OpenMP Task runtime and the new record and replay feature - Rémy Pierre Gwenaël Neveu

This talk offers a quick overview of the current implementation of OpenMP tasking. The objective of this talk is to explain the current state of the implementation and provide guidelines for other developers on how to get started and improve LLVM's OpenMP task support. We first describe the overall compilation pipeline of programs that include OpenMP tasks. We emphasize source code location and the role of each part of the compilation process. Following, we provide a deeper overview of the clang front-end. Next, we provide an overview of the runtime system interface and implementation. Finally, we provide hints on how to test new functionalities and debug the runtime.

Student Talk – LLVM Dev Meeting 23

For more information...

Taskgraph: A Low Contention OpenMP Tasking Framework

Chenle Yu , Sara Royuela , Eduardo Quiñones 

Research Paper - BSC

Quick explanation of the LLVM's OpenMP Task runtime and the new record and replay feature - Rémy Pierre Gwenaël Neveu

This talk offers a quick overview of the current implementation of OpenMP tasking. The objective of this talk is to explain the current state of the implementation and provide guidelines for other developers on how to get started and improve LLVM's OpenMP task support. We first describe the overall compilation pipeline of programs that include OpenMP tasks. We emphasize source code location and the role of each part of the compilation process. Following, we provide a deeper overview of the clang front-end. Next, we provide an overview of the runtime system interface and implementation. Finally, we provide hints on how to test new functionalities and debug the runtime.

TDG discovery and compile-time optimizations of OpenMP Tasks - Rafael Andres Herrera Guaitero

In this session, the talk will focus on enhancing LLVM's ability to optimize OpenMP task code by proposing an approach for encoding the partial Task Dependence Graph (TDG) during compilation using LLVM-IR, attributes, and metadata. The goal is to enable the compiler to perform traditional code analysis, aiding in the TDG's construction, and optimizations that improve program execution. The significance of efficient and optimal code in the context of OpenMP tasking will be highlighted, along with the challenges and potential opportunities. To address these challenges effectively, a simple yet novel abstraction for OpenMP tasking analysis and optimizations will be presented. The discussion will cover the outcomes and current status of the implementation, demonstrating the feasibility and benefits of adopting this approach, aiming to achieve greater efficiency and performance in tasking programming models.

Student Talk – LLVM Dev Meeting 23

Student Talk – LLVM Dev Meeting 23

How can I get started?

Documentation:

- <https://llvm.org/docs/GettingStarted.html>
- <https://openmp.llvm.org/>



How to build it? - I

Install dependencies:

```
sudo apt-get install libelf-dev  
sudo apt-get install -y libffi-dev
```

Clone LLVM:

```
git clone https://github.com/llvm/llvm-project  
cd llvm-project  
mkdir build  
cd build
```

How to build it? - II

```
cmake ../llvm
-DCMAKE_BUILD_TYPE=Release
-DLLVM_ENABLE_PROJECTS="clang;clang-tools-extra;compiler-rt;lld"
-DLLVM_ENABLE_RUNTIMES="openmp"
-DLLVM_ENABLE_ASSERTIONS=ON
-DLIBOMPTARGET_ENABLE_DEBUG=ON
-DLIBOMPTARGET_DEVICE_ARCHITECTURES=all
-DLLVM_USE_LINKER=lld
-DCLANG_DEFAULT_LINKER=lld
-DLLVM_OPTIMIZED_TABLEGEN=ON
-DBUILD_SHARED_LIBS=ON
-DLLVM_CCACHE_BUILD=ON -DLLVM_APPEND_VC_REV=OFF -G Ninja && ninja
```

How to compile it?

```
#include <complex>

using complex = std::complex<double>;

void zaxpy(complex *X, complex *Y, complex D, std::size_t N) {
#pragma omp target teams distribute parallel for
    for (std::size_t i = 0; i < N; ++i)
        Y[i] = D * X[i] + Y[i];
}

int main() {
    const std::size_t N = 1024;
    complex X[N], Y[N], D;
#pragma omp target data map(to:X[0 : N]) map(tofrom:Y[0 : N])
    zaxpy(X, Y, D, N);
}
```

Example from OpenMP Documentation

How to compile it?

```
#include <complex>
```

```
using complex = std::complex<double>;
```

```
void zaxpy(complex *X, complex *Y, complex D, std::size_t N) {  
#pragma omp target teams distribute parallel for  
    for (std::size_t i = 0; i < N; ++i)  
        Y[i] = D * X[i] + Y[i];  
}
```

```
int main() {  
    const std::size_t N = 1024;  
    complex X[N], Y[N], D;  
#pragma omp target data map(to:X[0 : N]) map(tofrom:Y[0 : N])  
    zaxpy(X, Y, D, N);  
}
```

```
clang++ -fopenmp -fopenmp-targets=nvptx64 zaxpy.cpp
```

Target architecture

Example from OpenMP Documentation

How to optimize it?

```
#include <complex>

using complex = std::complex<double>;

void zaxpy(complex *X, complex *Y, complex D, std::size_t N) {
#pragma omp target teams distribute parallel for
    for (std::size_t i = 0; i < N; ++i)
        Y[i] = D * X[i] + Y[i];
}

int main() {
    const std::size_t N = 1024;
    complex X[N], Y[N], D;
#pragma omp target data map(to:X[0 : N]) map(tofrom:Y[0 : N])
    zaxpy(X, Y, D, N);
}
```

Example from OpenMP Documentation

How to optimize it?

```
#include <complex>
```

```
clang++ -fopenmp -fopenmp-targets=nvptx64 -O3 zaxpy.cpp
```

```
using complex = std::complex<double>;
```

Level 3 optimizations

```
void zaxpy(complex *X, complex *Y, complex D, std::size_t N) {  
#pragma omp target teams distribute parallel for  
    for (std::size_t i = 0; i < N; ++i)  
        Y[i] = D * X[i] + Y[i];  
}
```

```
int main() {  
    const std::size_t N = 1024;  
    complex X[N], Y[N], D;  
#pragma omp target data map(to:X[0 : N]) map(tofrom:Y[0 : N])  
    zaxpy(X, Y, D, N);  
}
```

Example from OpenMP Documentation

How to optimize it?

```
#include <complex>
```

```
using complex = std::complex<double>;
```

```
void zaxpy(complex *X, complex *Y, complex D, std::size_t N) {  
#pragma omp target teams distribute parallel for  
    for (std::size_t i = 0; i < N; ++i)  
        Y[i] = D * X[i] + Y[i];  
}
```

```
int main() {  
    const std::size_t N = 1024;  
    complex X[N], Y[N], D;  
#pragma omp target data map(to:X[0 : N]) map(tofrom:Y[0 : N])  
    zaxpy(X, Y, D, N);  
}
```

```
clang++ -fopenmp -fopenmp-targets=nvptx64 -O3 zaxpy.cpp
```

Level 3 optimizations

```
-Rpass=openmp-opt -Rpass-missed=openmp-opt -Rpass-analysis=openmp-opt
```

OpenMP Optimization remarks

Example from OpenMP Documentation

How to run it?

`./zaxpy`

Slurm Job manager -> `srun zaxpy`

```
#!/bin/bash
# Job name:
#SBATCH --job-name=test
#
# Account:
#SBATCH --account=account_name
#
# Partition:
#SBATCH --partition=partition_name
#
# Request one node:
#SBATCH --nodes=1
#
# Specify one task:
#SBATCH --ntasks-per-node=1
#
# Number of processors for single task needed for use case (example):
#SBATCH --cpus-per-task=4
#
# Wall clock limit:
#SBATCH --time=00:00:30
#
## Command(s) to run (example):
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./a.out
```

How to debug it?

`LIBOMPTARGET_INFO=1` ./zaxpy

Documentation

<https://openmp.llvm.org/design/Runtimes.html#debugging>

```
Info: Entering OpenMP data region at zaxpy.cpp:14:1 with 2 arguments:
Info: to(X[0:N])[16384]
Info: tofrom(Y[0:N])[16384]
Info: Creating new map entry with HstPtrBegin=0x00007fff0d259a40,
      TgtPtrBegin=0x00007fdb5800000, Size=16384, RefCount=1, Name=X[0:N]
Info: Copying data from host to device, HstPtr=0x00007fff0d259a40,
      TgtPtr=0x00007fdb5800000, Size=16384, Name=X[0:N]
Info: Creating new map entry with HstPtrBegin=0x00007fff0d255a40,
      TgtPtrBegin=0x00007fdb5804000, Size=16384, RefCount=1, Name=Y[0:N]
Info: Copying data from host to device, HstPtr=0x00007fff0d255a40,
      TgtPtr=0x00007fdb5804000, Size=16384, Name=Y[0:N]
Info: OpenMP Host-Device pointer mappings after block at zaxpy.cpp:14:1:
Info: Host Ptr      Target Ptr      Size (B) RefCount Declaration
Info: 0x00007fff0d255a40 0x00007fdb5804000 16384    1          Y[0:N] at zaxpy.cpp:13:17
Info: 0x00007fff0d259a40 0x00007fdb5800000 16384    1          X[0:N] at zaxpy.cpp:13:11
Info: Entering OpenMP kernel at zaxpy.cpp:6:1 with 4 arguments:
Info: firstprivate(N)[8] (implicit)
Info: use_address(Y)[0] (implicit)
Info: tofrom(D)[16] (implicit)
Info: use_address(X)[0] (implicit)
Info: Mapping exists (implicit) with HstPtrBegin=0x00007fff0d255a40,
      TgtPtrBegin=0x00007fdb5804000, Size=0, RefCount=2 (incremented), Name=Y
Info: Creating new map entry with HstPtrBegin=0x00007fff0d2559f0,
      TgtPtrBegin=0x00007fdb5808000, Size=16, RefCount=1, Name=D
Info: Copying data from host to device, HstPtr=0x00007fff0d2559f0,
      TgtPtr=0x00007fdb5808000, Size=16, Name=D
Info: Mapping exists (implicit) with HstPtrBegin=0x00007fff0d259a40,
      TgtPtrBegin=0x00007fdb5800000, Size=0, RefCount=2 (incremented), Name=X
Info: Mapping exists with HstPtrBegin=0x00007fff0d255a40,
      TgtPtrBegin=0x00007fdb5804000, Size=0, RefCount=2 (update suppressed)
Info: Mapping exists with HstPtrBegin=0x00007fff0d2559f0,
      TgtPtrBegin=0x00007fdb5808000, Size=16, RefCount=1 (update suppressed)
Info: Mapping exists with HstPtrBegin=0x00007fff0d259a40,
      TgtPtrBegin=0x00007fdb5800000, Size=0, RefCount=2 (update suppressed)
Info: Launching kernel __omp_offloading_10305_c08c86_Z5zaxpyPst7complexIdES1_S0_m_l6
```

How to get involved?

- Slack Channel: <https://tinyurl.com/ptw5b8vm>
- Projects: <https://tinyurl.com/ycyje4xc>



- **LLVM/OpenMP Community** – Johannes, Shilei, Joseph and Others.
- **LLVM Foundation** – Tanya Lattner

THANKS



THANK YOU

Any questions?

Rafael Andres Herrera Guaitero

rafaelhg@udel.edu

Phd Candidate

University of Delaware