An Update on the Progress towards Distributed OpenMP

Atmn Patel, Northwestern University
September 30th, 2022

Acknowledgements

We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

Part of this research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

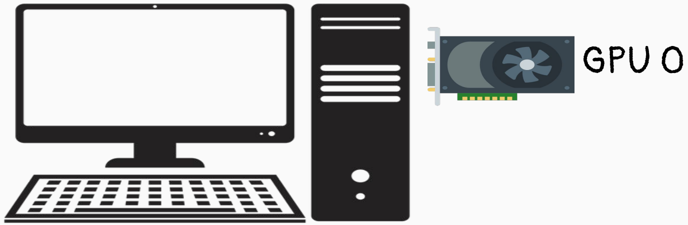
Part of this research was supported by the Lawrence Livermore National Security, LLC ("LLNS") via MPO No. B642066.

Credits

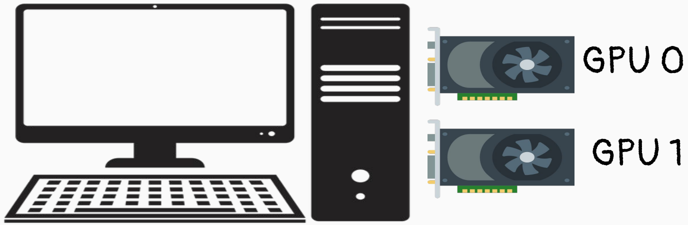
This research direction was initiated under the supervision of Johannes Doerfert at Argonne National Lab.

Since the initial work, the work has been taken over by the Exasca||ab at Stony Brook University.

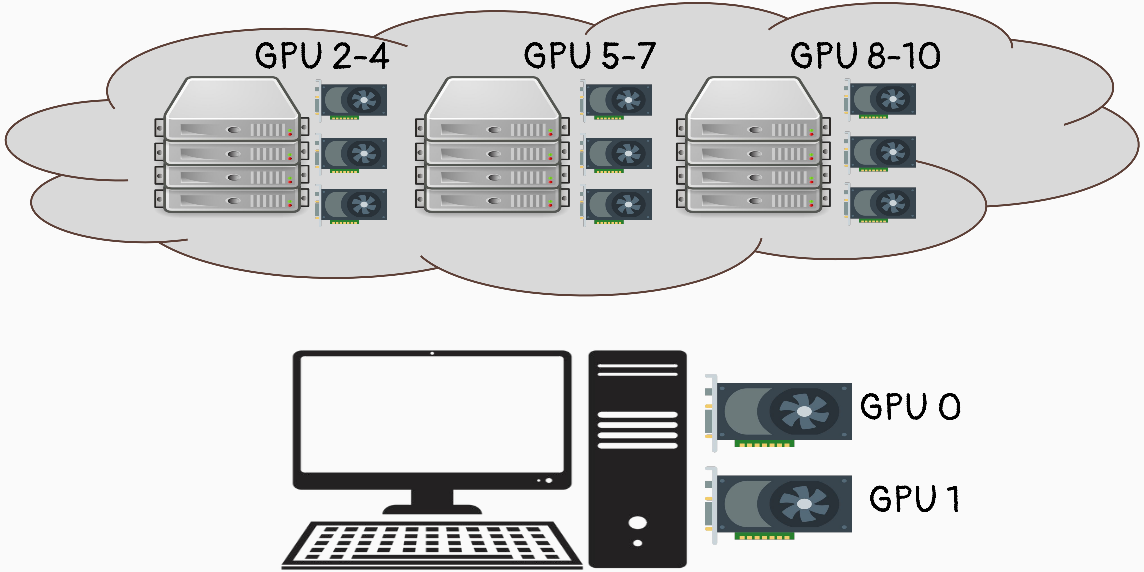
OpenMP Offload



Multi-GPU OpenMP Offload



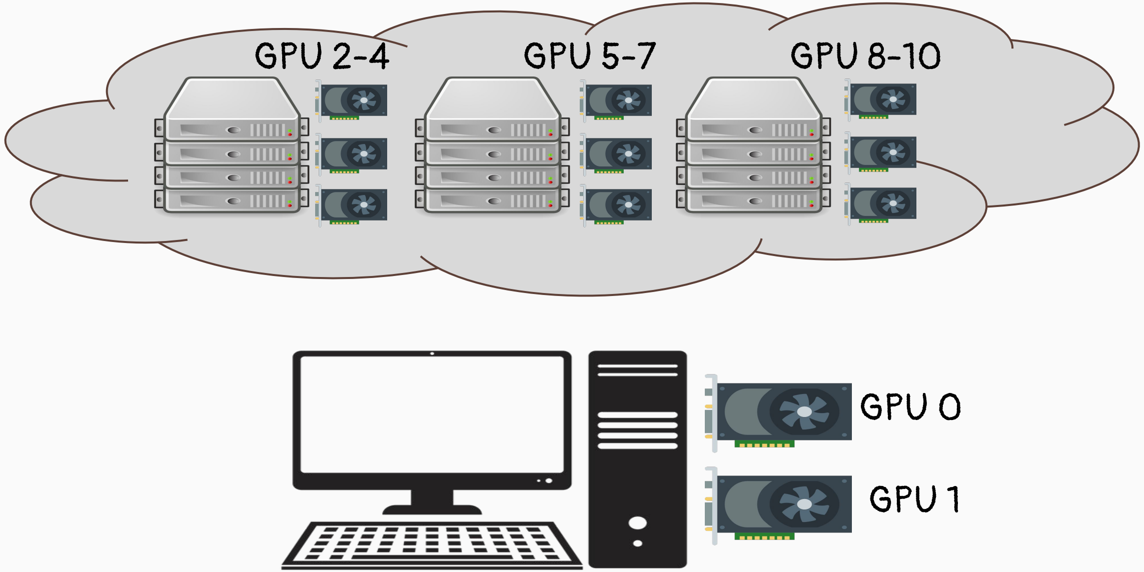
Remote Multi-GPU OpenMP Offload



Remote Multi-GPU OpenMP Offload

distributed environment ⟶

- non-unified memory +
- non-unified address space



Remote Multi-GPU OpenMP Offload

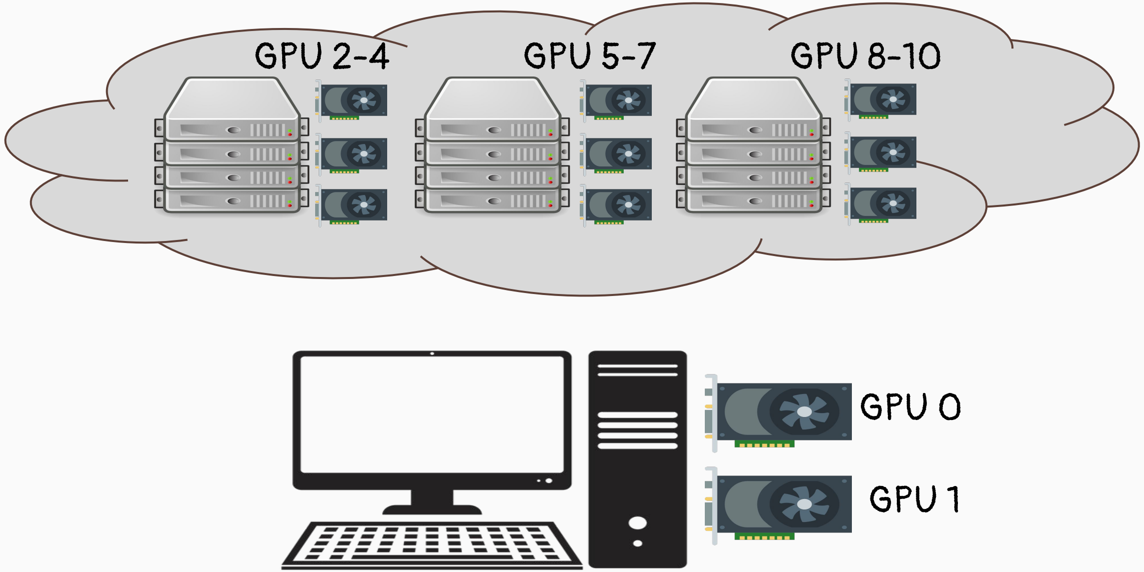
distributed environment ⟶

non-unified memory +

non-unified address space

Benefits:

+ no compiler changes necessary *



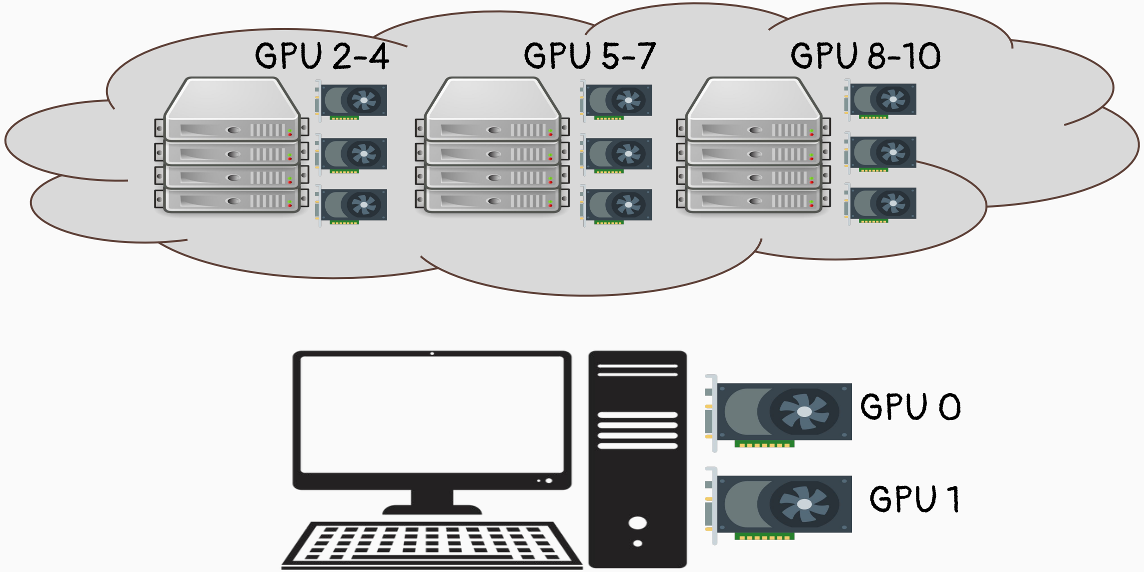
Remote Multi-GPU OpenMP Offload

distributed environment ⟶

- non-unified memory +
- non-unified address space

Benefits:

+ no compiler changes necessary *
+ no user code changes necessary



Remote Multi-GPU OpenMP Offload

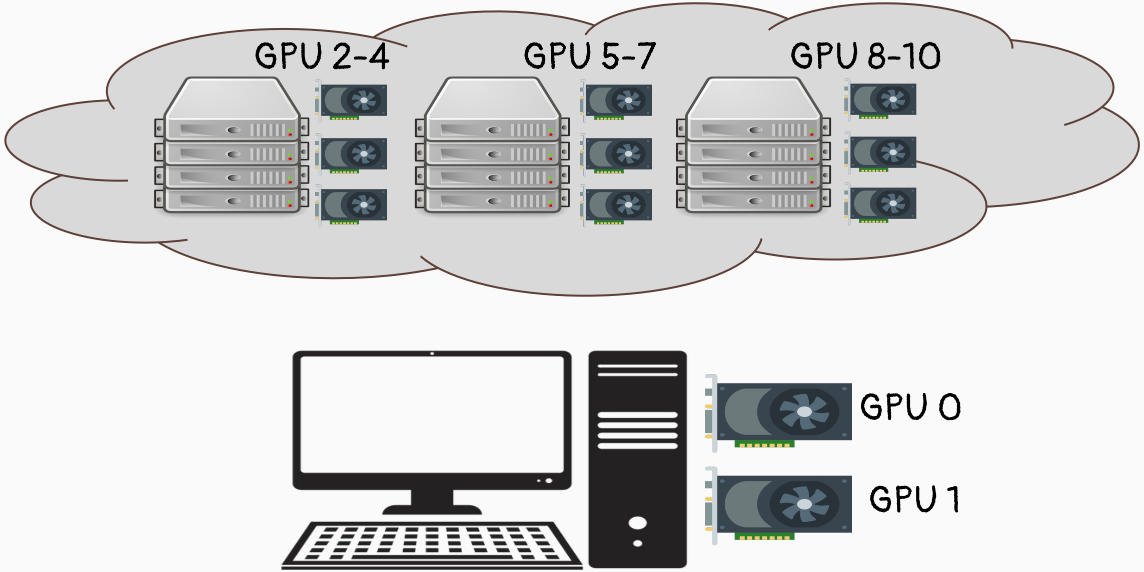
distributed environment ⟶

non-unified memory +

non-unified address space

Benefits:

+ no compiler changes necessary *
+ no user code changes necessary
+ composable (CPU, GPU, JIT, ...)



Remote Multi-GPU OpenMP Offload

distributed environment ⟶

non-unified memory +

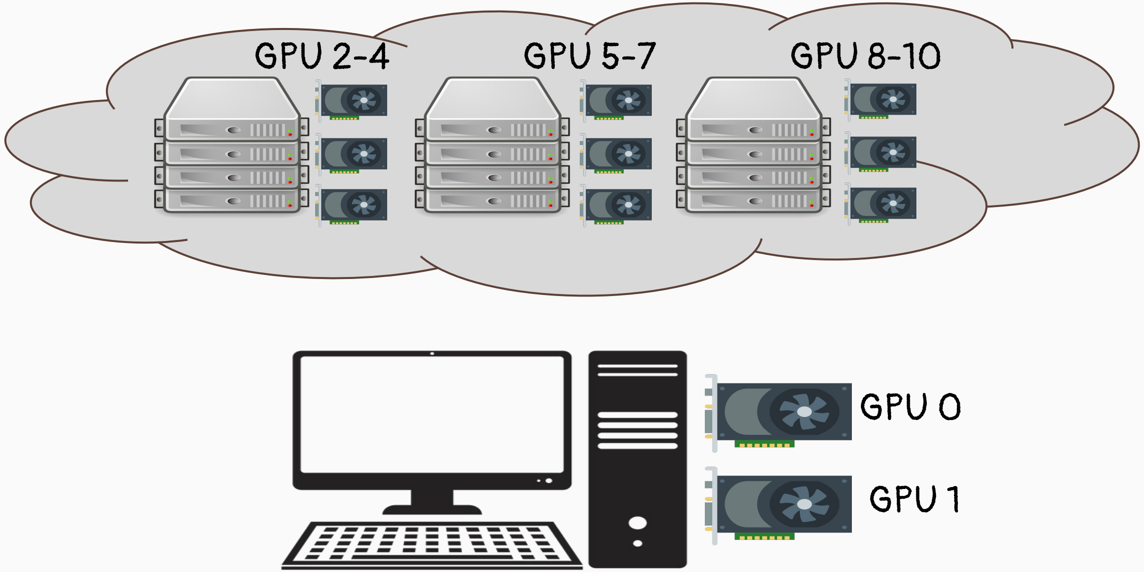
non-unified address space

Benefits:

- + no compiler changes necessary *
- + no user code changes necessary
- + composable (CPU, GPU, JIT, …)

Drawbacks:

- - limited to the “host-centric” model

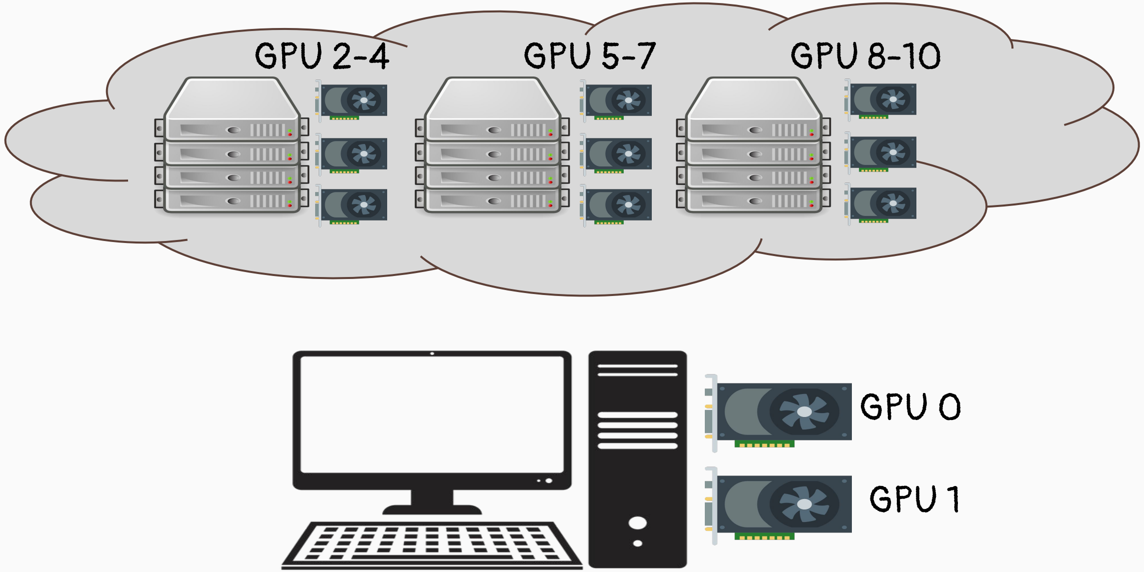


Remote Multi-GPU OpenMP Offload

distributed environment ⟶

non-unified memory +

non-unified address space



Benefits:

+ no compiler changes necessary *
+ no user code changes necessary
+ composable (CPU, GPU, JIT, ...)

Drawbacks:

- limited to the “host-centric” model
- opaque topology

Remote Multi-GPU OpenMP Offload

distributed environment ⟶

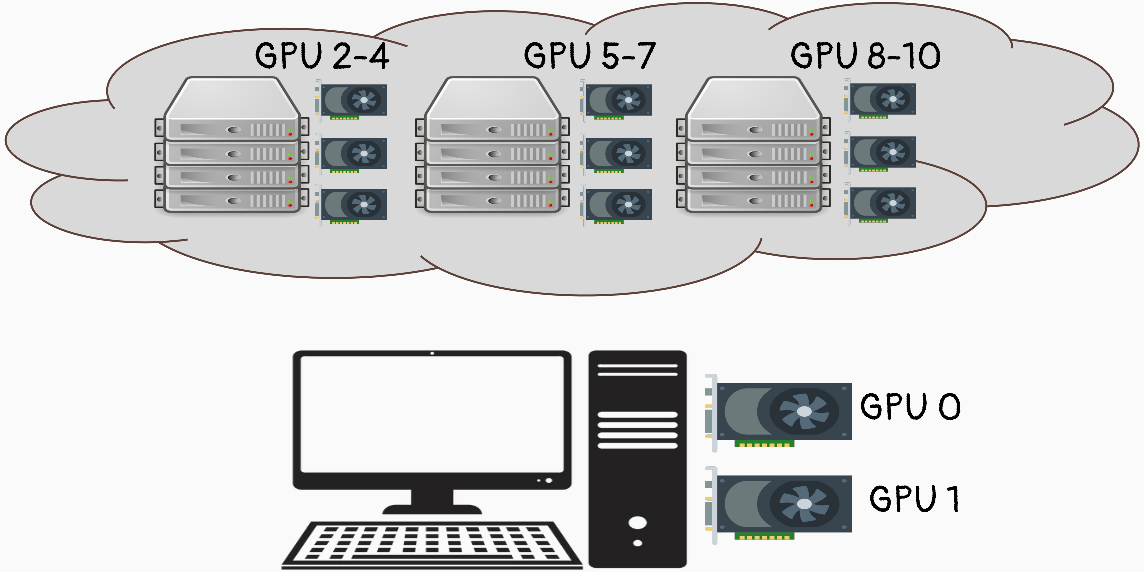
- non-unified memory +
- non-unified address space

Benefits:

- + no compiler changes necessary *
- + no user code changes necessary
- + composable (CPU, GPU, JIT, …)

Drawbacks:

- - limited to the “host-centric” model
- - opaque topology



***Remote OpenMP Offloading* offers distributed compute resource usage through a *single, coherent parallel programming model*.**

Implementation

Slide originally presented at LLVM-Dev Meeting 2020 by Johannes Doerfert https://youtu.be/M0DrhQbjrro

OpenMP in LLVM

https://openmp.llvm.org/docs

Slide originally presented at LLVM-Dev Meeting 2020 by Johannes Doerfert https://youtu.be/M0DrhQbjrro

OpenMP in LLVM

https://openmp.llvm.org/docs

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP runtimes

libomp.so
(classic, host)

Slide originally presented at LLVM-Dev Meeting 2020 by Johannes Doerfert https://youtu.be/M0DrhQbjrro

OpenMP in LLVM

https://openmp.llvm.org/docs

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

Slide originally presented at LLVM-Dev Meeting 2020 by Johannes Doerfert https://youtu.be/M0DrhQbjrro

OpenMP in LLVM

https://openmp.llvm.org/docs

Flang

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP-IR-Builder

frontend independant OpenMP LLVM-IR generation

favor simple and expressive LLVM-IR

reusable for non-OpenMP parallelism

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

Slide originally presented at LLVM-Dev Meeting 2020 by Johannes Doerfert https://youtu.be/M0DrhQbjrro

OpenMP in LLVM

https://openmp.llvm.org/docs

Flang / Clang	OpenMP-IR-Builder	OpenMP-Opt	OpenMP runtimes
OpenMP Parser	frontend independant OpenMP LLVM-IR generation	interprocedural optimization pass	libomp.so (classic, host)
OpenMP Sema	favor simple and expressive LLVM-IR	contains host & device optimizations	libomptarget + plugins (offloading, host)
OpenMP CodeGen	reusable for non-OpenMP parallelism	run with -O1 and higher	libomptarget-nvptx (offloading, device)

Slide originally presented at LLVM-Dev Meeting 2020 by Johannes Doerfert https://youtu.be/M0DrhQbjrro

OpenMP in LLVM

https://openmp.llvm.org/docs

Flang

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP-IR-Builder

frontend independant OpenMP LLVM-IR generation

favor simple and expressive LLVM-IR

reusable for non-OpenMP parallelism

OpenMP-Opt

interprocedural optimization pass

contains host & device optimizations

run with `-O1` and higher

OpenMP runtimes

libomp.so (classic, host)

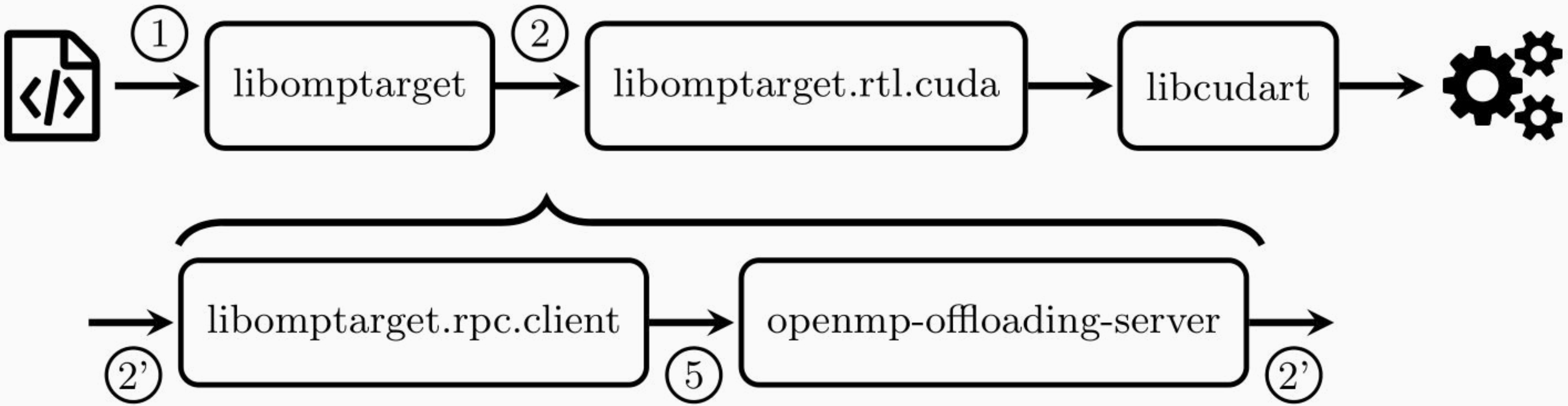
libomptarget + **plugins** (offloading, host)

libomptarget-nvptx (offloading, device)

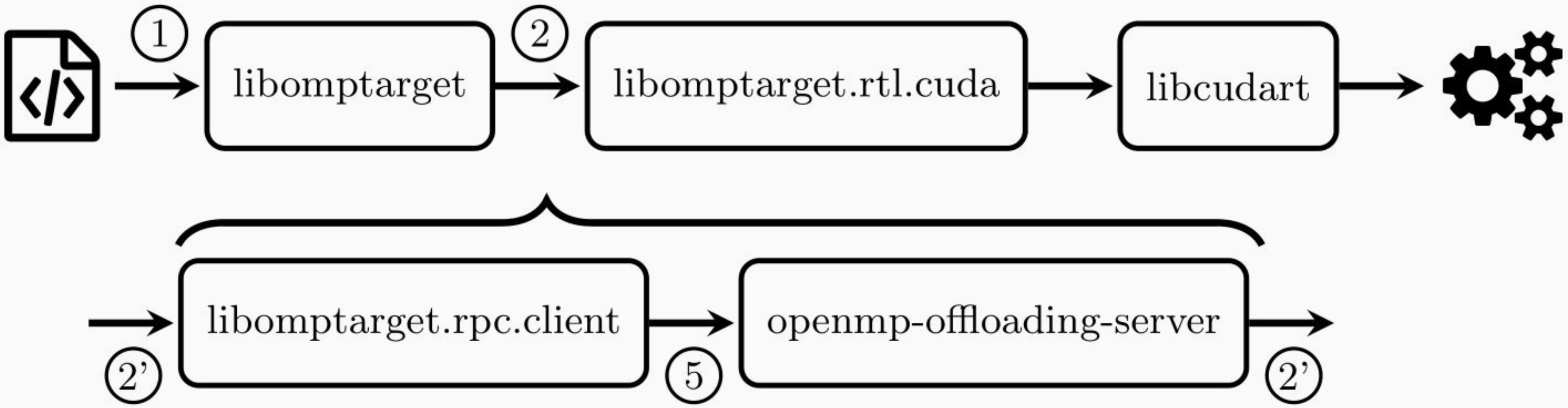
Slide originally presented at LLVM-Dev Meeting 2020 by Johannes Doerfert https://youtu.be/M0DrhQbjrro

Remote OpenMP Offloading (Plugin)

Remote OpenMP Offloading (Plugin)



Remote OpenMP Offloading (Plugin)



* interface (2') is (2) with two optional API functions exposed by the *remote* plugin.

Networking Backends

gRPC (may be deprecated very soon)
(google's Remote-Procedure-Call)

- \+ many out-of-the-box features: thread pools, concurrency, compression, …
- ● optimized for small messages (< 2 MB)
- \- tied to (google's) protobuf
- \- general purpose & little customization e.g., for compression, specialized networks and access kinds

UCX
(Unified Communication X)

- \+ highly configurable (RMA, AMO, Tag Matching, Active Message, Stream, …)
- \+ network layer aware (IP over InfiniBand)
- \- Using MPICH directly has better performance for large messages for now

Implementation Notes

It has been only tested on NVIDIA GPUs, but it should extend to any accelerator targeted by LLVM.

It is known to work from x86 and ARM to remote GPUs, SmartNIC CPU and GPUs, etc.

The upstream has been broken for a while, but many performance updates + fixes are in-flight from Exasca||ab.

Stony Brook has been working on a more efficient implementation, where they:

- Use CUDA-Aware Communication
- Improved NUMA Awareness through some fun techniques
- Presented at IWOMP this week

Evaluation

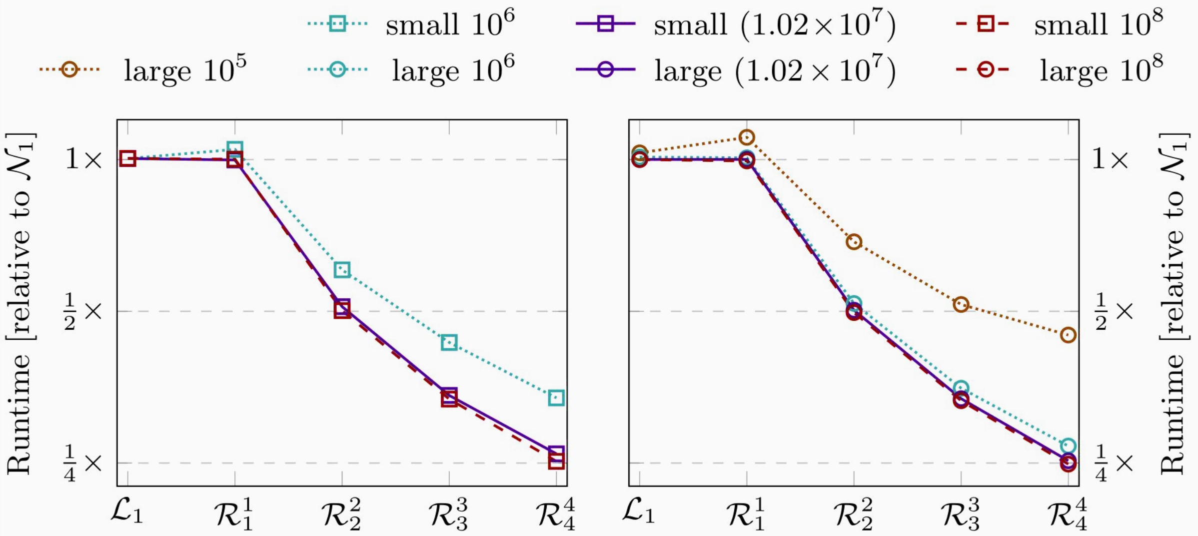
RSBench/XSBench

- Monte Carlo simulation codes
- particle transport in reactors
- available for single-GPU OpenMP offload
- extended to multi-GPU OpenMP offload (easy to map)
- weak scaling in the Google cloud (4 nodes, 1 NVIDIA T4 GPU each)
- strong scaling on ThetaGPU (15 nodes, 8 NVIDIA A100 GPUs each)

RSBench/XSBench

- Monte Carlo simulation codes
- particle transport in reactors
- available for single-GPU OpenMP offload
- extended to multi-GPU OpenMP offload (easy to map)
- weak scaling in the Google cloud (4 nodes, 1 NVIDIA T4 GPU each)
- strong scaling on ThetaGPU (15 nodes, 8 NVIDIA A100 GPUs each)

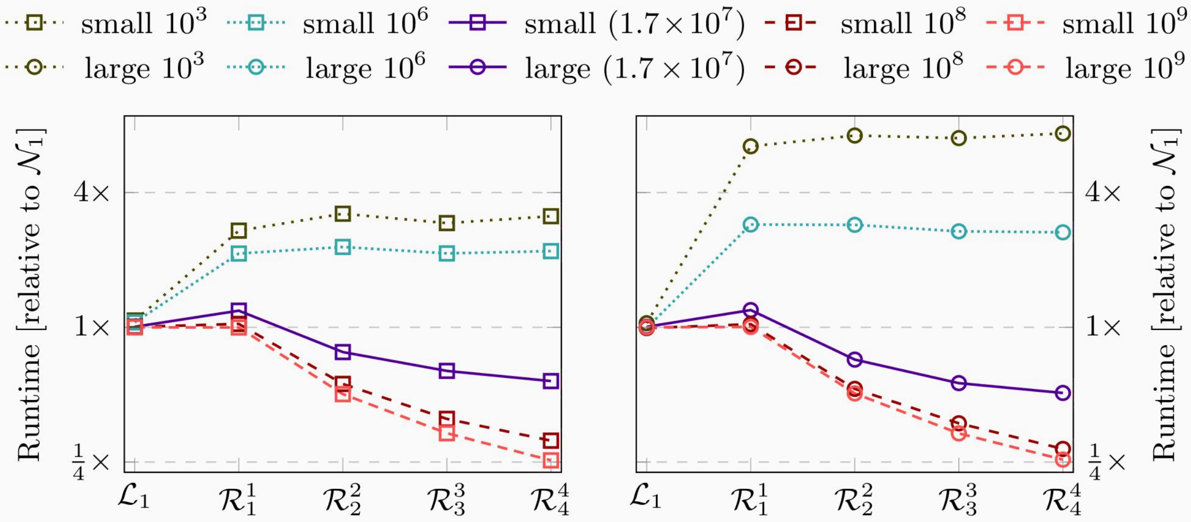
RSBench on Google Cloud



RSBench/XSBench

- Monte Carlo simulation codes
- particle transport in reactors
- available for single-GPU OpenMP offload
- extended to multi-GPU OpenMP offload (easy to map)
- weak scaling in the Google cloud (4 nodes, 1 NVIDIA T4 GPU each)
- strong scaling on ThetaGPU (15 nodes, 8 NVIDIA A100 GPUs each)

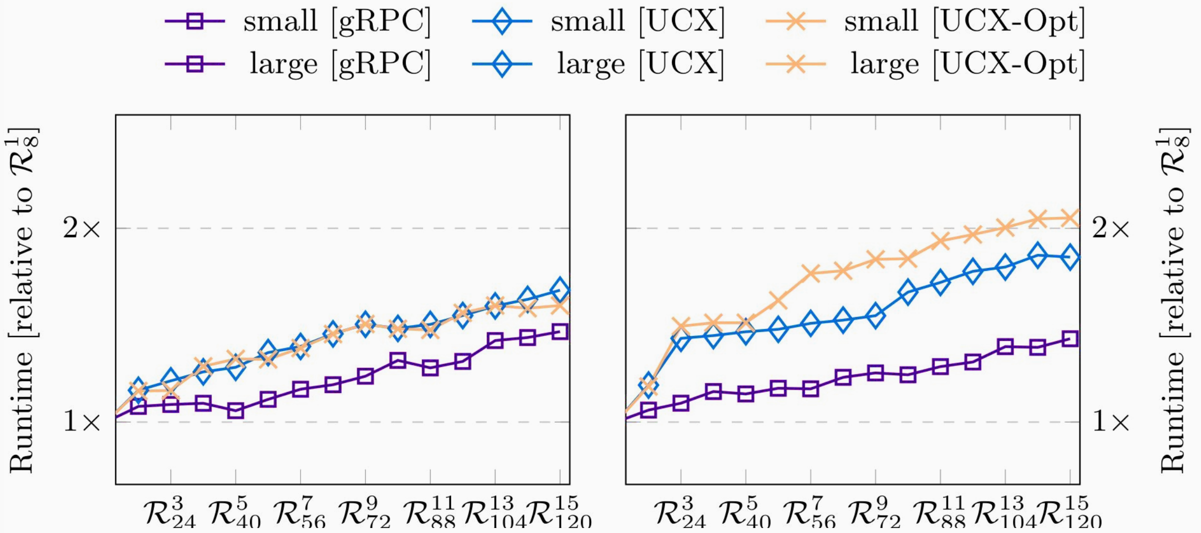
XSBench on Google Cloud



RSBench/XSBench

- Monte Carlo simulation codes
- particle transport in reactors
- available for single-GPU OpenMP offload
- extended to multi-GPU OpenMP offload (easy to map)
- weak scaling in the Google cloud (4 nodes, 1 NVIDIA T4 GPU each)
- strong scaling on ThetaGPU (15 nodes, 8 NVIDIA A100 GPUs each)

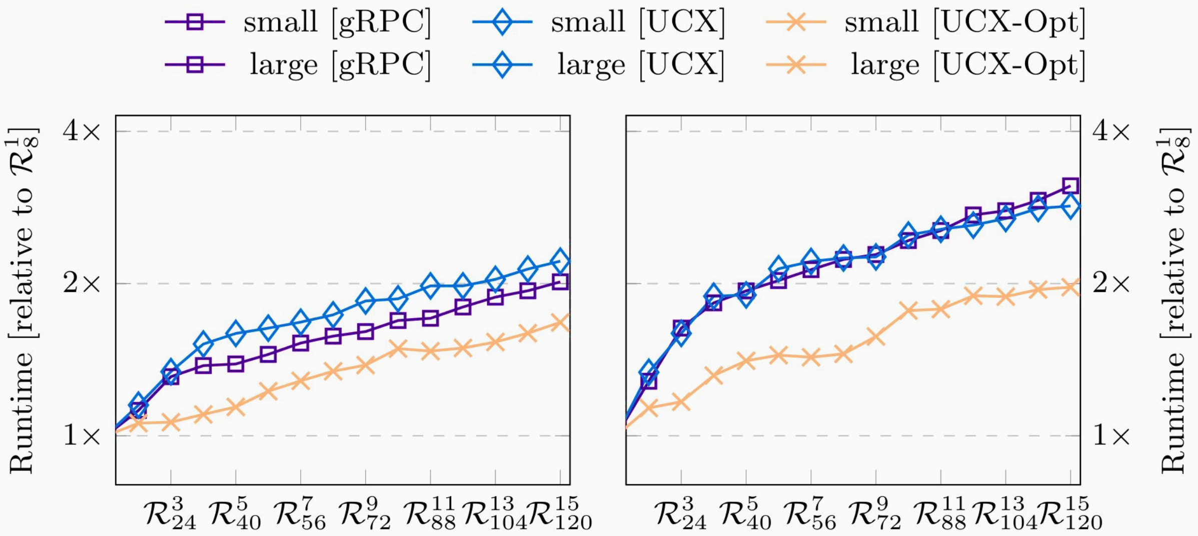
RSBench on ThetaGPU



RSBench/XSBench

- Monte Carlo simulation codes
- particle transport in reactors
- available for single-GPU OpenMP offload
- extended to multi-GPU OpenMP offload (easy to map)
- weak scaling in the Google cloud (4 nodes, 1 NVIDIA T4 GPU each)
- strong scaling on ThetaGPU (15 nodes, 8 NVIDIA A100 GPUs each)

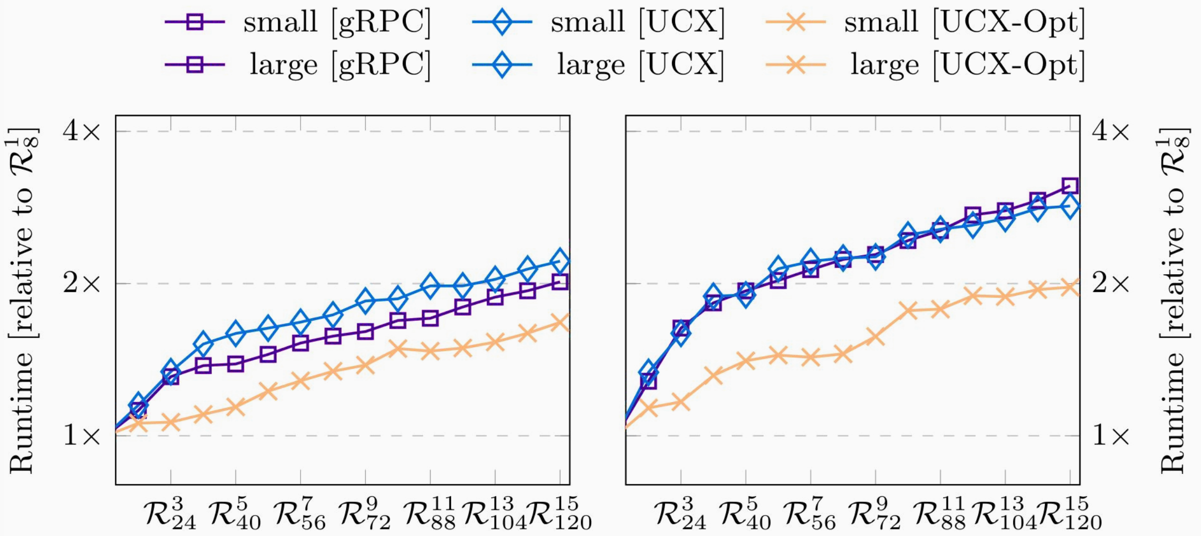
XSBench on ThetaGPU



RSBench/XSBench

- Monte Carlo simulation codes
- particle transport in reactors
- available for single-GPU OpenMP offload
- extended to multi-GPU OpenMP offload (easy to map)
- weak scaling in the Google cloud (4 nodes, 1 NVIDIA T4 GPU each)
- strong scaling on ThetaGPU (15 nodes, 8 NVIDIA A100 GPUs each)

XSBench on ThetaGPU



The compute to memory transfer ratio determines the effectiveness of *OpenMP Remote Offloading*.

Future Work

Multi-Device Features

```
void array(float *A, int N) {
 int numD = omp_get_num_devices();

 for (int d = 0; d < numD; ++d) {
  int chunkBegin = …, chunkSize = …, chunkEnd = …;
  #pragma omp target teams distribute           \
              parallel for default(firstprivate) \
              map(tofrom:A[chunkBegin:chunkSize])\
              device(d)
  for (int i = chunkBegin; i < chunkEnd; ++i)
    A[i] = A[i] * 2;
 }
}
```

Multi-Device Features

```
void array(float *A, int N) {
 int numD = omp_get_num_devices();
 #pragma omp parallel for
 for (int d = 0; d < numD; ++d) {
  int chunkBegin = …, chunkSize = …, chunkEnd = …;
  #pragma omp target teams distribute           \
              parallel for default(firstprivate) \
              map(tofrom:A[chunkBegin:chunkSize])\
              device(d)
  for (int i = chunkBegin; i < chunkEnd; ++i)
    A[i] = A[i] * 2;
 }
}
```

```
void array(float *A, int N) {
 int numD = omp_get_num_devices();
 #pragma omp parallel for
 for (int d = 0; d < numD; ++d) {
  int chunkBegin = …, chunkSize = …, chunkEnd = …;
  #pragma omp target teams distribute            \
              parallel for default(firstprivate) \
              map(tofrom:A[chunkBegin:chunkSize])\
              device(d)
  for (int i = chunkBegin; i < chunkEnd; ++i)
    A[i] = A[i] * 2;
 }
}
```

Multi-Device Features

- missing bulk launch

```c
void array(float *A, int N) {
 int numD = omp_get_num_devices();
 #pragma omp parallel for
 for (int d = 0; d < numD; ++d) {
  int chunkBegin = …, chunkSize = …, chunkEnd = …;
  #pragma omp target teams distribute            \
              parallel for default(firstprivate) \
              map(tofrom:A[chunkBegin:chunkSize])\
              device(d)
  for (int i = chunkBegin; i < chunkEnd; ++i)
    A[i] = A[i] * 2;
 }
}
```

Multi-Device Features

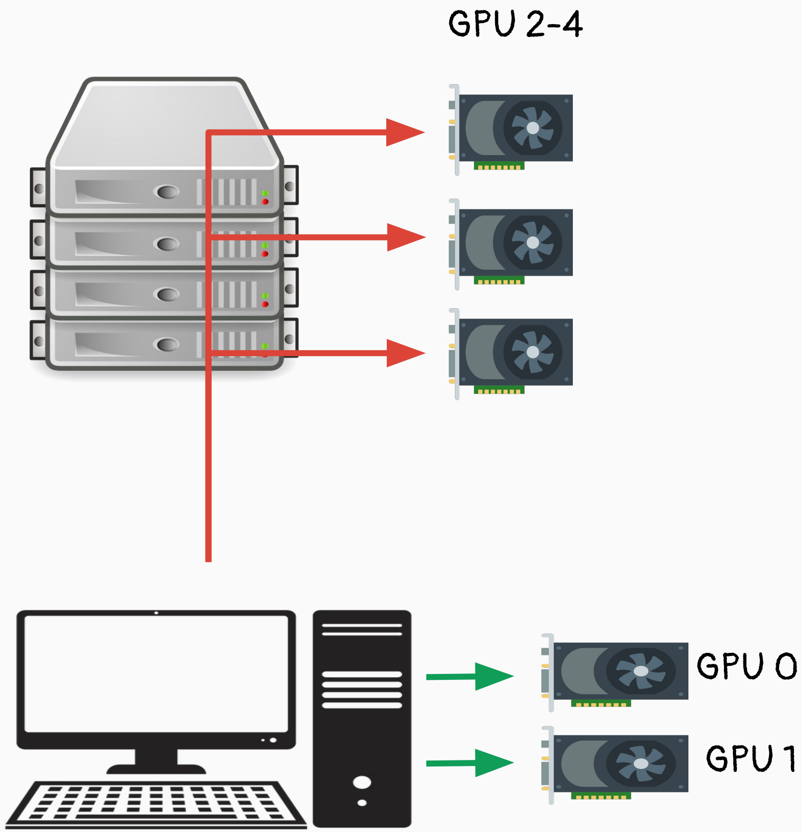
- missing bulk launch
- missing auto chunking

OpenMP Extension Sketch

```
void array(float *A, int N) {
 int numD = omp_get_num_devices();



  #pragma omp target teams distribute           \
              parallel for default(firstprivate) \
              map(tofrom,chunked:A[:N])          \
              devices(0:numD)
  for (int i = 0; i < N; ++i)
    A[i] = A[i] * 2;

}
```

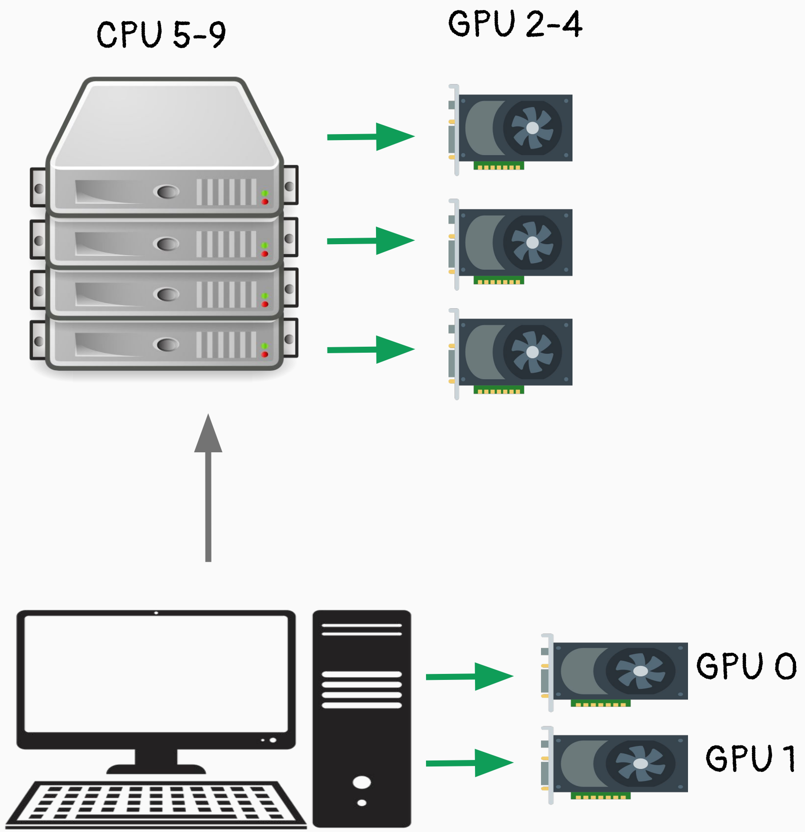
Multi-Device Features

- missing bulk launch
- missing auto chunking



Multi-Device Features

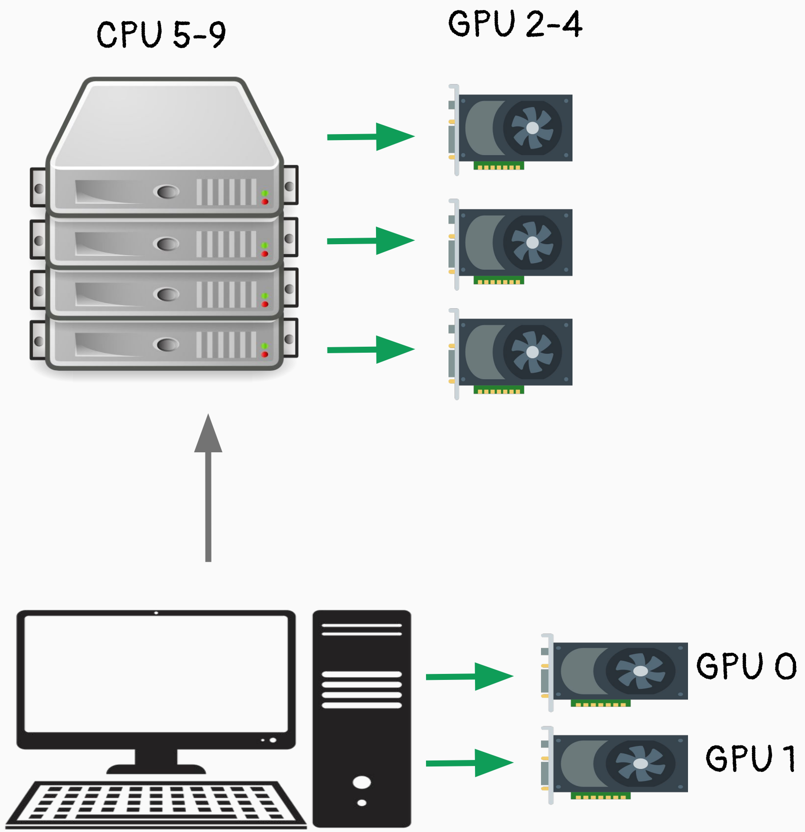
- missing bulk launch
- missing auto chunking



Multi-Device Features

- missing bulk launch
- missing auto chunking

\+ `omp_target_memcpy[_async,_rect](...,`
`dst_device_num, src_device_num)`

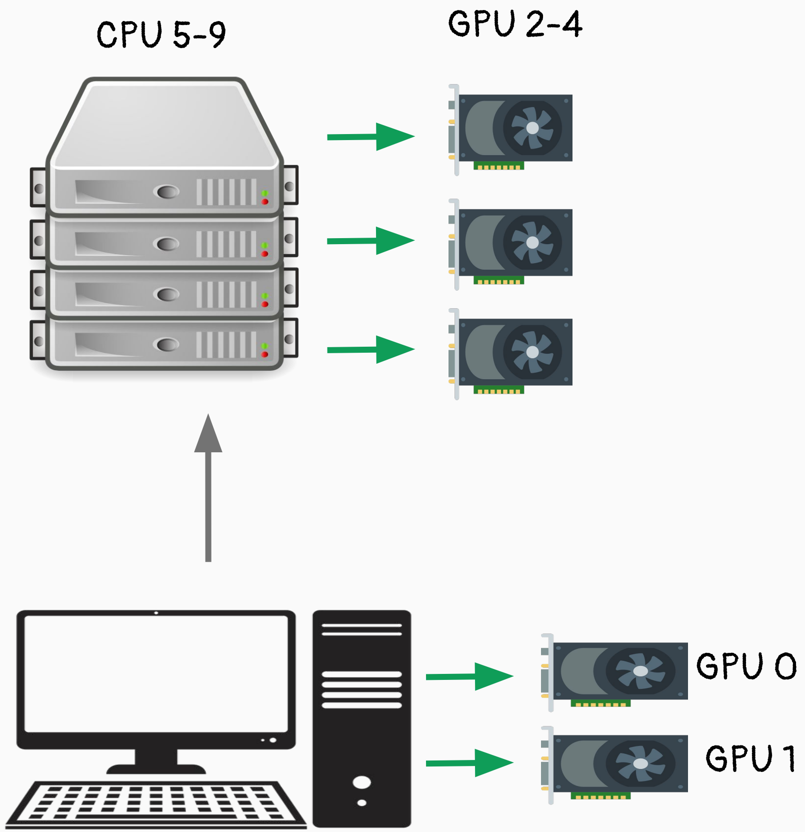


Multi-Device Features

- missing bulk launch
- missing auto chunking

\+ `omp_target_memcpy[_async,_rect](...,`
`dst_device_num, src_device_num)`

- missing device / topology information

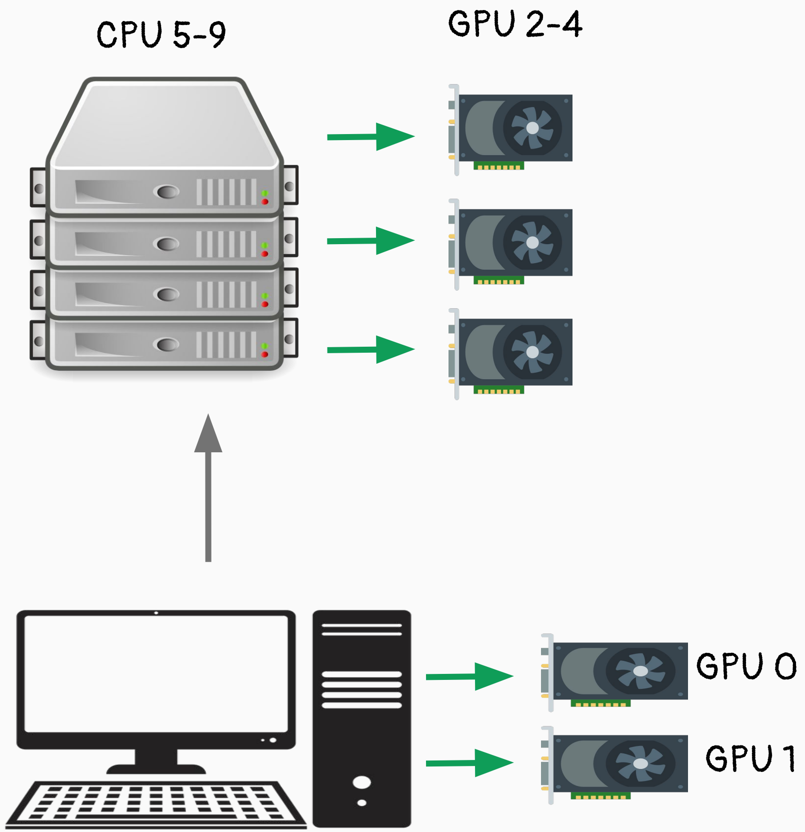


Multi-Device Features

- missing bulk launch
- missing auto chunking

\+
```
omp_target_memcpy[_async,_rect](...,
     dst_device_num, src_device_num)
```

- missing device / topology information
- missing hierarchical / nested offloading



Multi-Device Features

- missing bulk launch
- missing auto chunking

\+ `omp_target_memcpy[_async,_rect](...,`
`dst_device_num, src_device_num)`

- missing device / topology information
- missing hierarchical / nested offloading
- native collective communication

Questions?