# LLVM/OpenMP Status

## Tips & Tricks for Application Developers

Johannes Doerfert <doerfert1@llnl.gov>

# Learn More

https://youtu.be/R9PUdx1ya1o

Full 3h tutorial

lots of details

for IWOMP participants till the end of the year

# Building LLVM

Single command often suffices to configure:

```
cmake …/llvm-project/llvm -DLLVM_ENABLE_PROJECTS='clang' -DLLVM_ENABLE_RUNTIMES=’openmp’
make -j
```

Useful options include:

```
CMAKE_BUILD_TYPE={Release,Asserts,…}
LLVM_ENABLE_ASSERTIONS={ON,OFF}
LLVM_CCACHE_BUILD={ON,OFF}
-G Ninja
```

May need debug build to debug certain compiler-based issues,
**release + assert** is often used as trade off.

Various resources available online! Start here:

http://llvm.org/docs/GettingStarted.html
https://openmp.llvm.org/SupportAndFAQ.html

# Using LLVM (cheat sheet)

- Use a fast linker (`lld`), `ccache`, and `ninja`
- Consider LTO, either thin or full
- Use tooling (`clang-format`, `clang-tidy`, `clang-modernize`, …)
- Use `-O3/Ofast -march=native` as default
- Online documentation is not great but often not bad either
- Debug with sanitizers enabled
- A release + asserts build is best for every-day use

# Using LLVM/OpenMP Offload (cheat sheet)

- Use a recent (e.g., nightly) compiler version.
- Enable compilation remarks https://openmp.llvm.org/remarks/OptimizationRemarks.html
- Use `LIBOMPTARGET_INFO(=16)` to learn about the GPU execution https://openmp.llvm.org/design/Runtimes.html#libomptarget-info
- Use `LIBOMPTARGET_PROFILE` for built in profiling support.
- Use `LIBOMPTARGET_DEBUG` (and `-fopenmp-target-debug`) for runtime assertions and other opt-in debug features https://openmp.llvm.org/design/Runtimes.html#debugging
- Consider assumptions for better performance:
  `LIBOMPTARGET_MAP_FORCE_ATOMIC=false, -fopenmp-assume-no-thread-state, ...`
- Use device-side LTO `-foffload-lto`

# Ask the Community

Many ways to interact:

- Discourse (forum/mailing list)
- Discord (persistent chat)
- IRC (non-persistent chat)
- Online Sync-Ups:
  - AA, MLIR, ML, OpenMP, RISC-V, …
- Office Hours ***NEW***
  - “AMA” with an “expert”
- Meetups (soon again!)

# Latest LLVM Release

LLVM 15 has been released and contains various new offloading features, incl.

- A new compiler driver for offloading (OpenMP, CUDA, HIP)
- Multi-Architecture binaries
- Link Time Optimization
- Static Library Support
- OpenMP and CUDA / HIP interoperability
- Extra flags improving offloading performance

Slide originally by *Joseph Huber*

# LLVM/OpenMP - A Community Effort

Weekly Meeting: https://bit.ly/2Zqt49v

## “Academia”

- Shilei Tian (SBU)
- Giorgis Georgakoudis (LLNL)
- Michael Kruse (ANL)
- Joachim Protze (RWTH A.)
- Joel Denny (ORNL)
- Atmn Patel (Northwestern)
- Konstantinos Parasyris (LLNL)
- Marc Jasper (LLNL)
- Many, many, more

## Industry

- Joseph Huber (AMD)
- Alexey Bataev (Intel)
- Jon Chesterfield (AMD)
- George Rokos (Intel)
- Pushpinder Singh (AMD)
- Kiran Chandramohan (ARM)
- Chi Chun Chen (HPE/Cray)
- Andrey Churbanov (Intel)
- Carlo Bertolli (AMD)
- Valentin Clement (NVIDIA)
- Many, many, more

## Power Users

- Ye Luo (ANL)
- Christopher Daley (NERSC)
- John Tramm (ANL)
- Rahul Gayatri (NERSC)
- Itaru Kitayama (RIKEN)
- Wael Elwasif (ORNL)
- Tom Scogland (LLNL)
- More that I have forgotten

# Getting Involved in LLVM/OpenMP

- LLVM/OpenMP webpage (incl. FAQ)

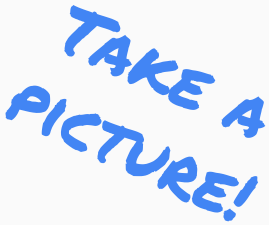
**Table I: Performance improvements from different optimizations on OpenMC for the HM-large benchmark problem. All optimizations are added to the ones above them in the table.**

| | Optimization Description | Inactive Batch Performance [particles/sec] | Additional Speedup |
|---|---|---|---|
| A | CMake unity build | 602 | - |
| B | -fopenmp-cuda-mode flag usage | 7,714 | 12.8 |
| C | LLVM update() clause optimization | 58,529 | 7.6 |
| D | OpenMC particle object size reduction | 89,732 | 1.5 |
| E | XS lookup kernel optimizations: inlining + reference removal | 117,067 | 1.3 |
| F | Continuous particle refill | 129,345 | 1.1 |
| G | XS lookup queue sort by energy | 164,114 | 1.3 |
| H | Removal of microscopic XS cache | 336,636 | 2.1 |
| I | Increasing number of particles in-flight to 8 million | 349,237 | 1.1 |

Also, feel free to reach out at any time!

# Papers & Presentations
(selection)

Papers:




- Automatic Asynchronous Execution of Synchronously Offloaded OpenMP Target Regions (LLVM-HPC'22)
- Direct GPU Compilation and Execution for Host Applications with OpenMP Parallelism (LLVM-HPC'22)
- Breaking the Vendor Lock --- Performance Portable Programming Through OpenMP as Target Independent Runtime Layer (PACT'22)
- Just-in-Time Compilation and Link Time Optimization for OpenMP Target Offloading (IWOMP'22)
- Efficient Execution of OpenMP on GPUs (CGO'22)
- Co-Designing an OpenMP GPU Runtime Optimizations for Near-Zero Overhead Execution (IPDPS'22)
- Remote OpenMP Offloading (ISC'22, **best paper**)
- Toward Portable GPU Acceleration of the OpenMC Monte Carlo Particle Transport Code (PHYSOR'22)
- A Virtual GPU as Developer-Friendly OpenMP Offload Target (LLPP'21)
- Advancing OpenMP Offload Debugging Capabilities in LLVM (LLPP'21)
- Experience Report: Writing A Portable GPU Runtime with OpenMP 5.1 (IWOMP'21)
- Compiler Optimizations For Parallel Programs (LCPC'18)

Presentations:

A Compiler's View of OpenMP https://www.openmp.org/events/webinar-a-compilers-view-of-the-openmp-api/

# Improved OpenMP Offload Error Diagnostic

```
$ clang++ -fopenmp -fopenmp-targets=nvptx64 -O3 -gline-tables-only sum.cpp -o sum
$ ./sum
CUDA error: an illegal memory access was encountered
Libomptarget error: Copying data from device failed.
Libomptarget error: Call to targetDataEnd failed, abort target.
Libomptarget error: Failed to process data after launching the kernel.
Libomptarget error: Consult https://openmp.llvm.org/design/Runtimes.html for debugging options.
sum.cpp:5:1: Libomptarget error 1: failure of target construct while offloading is mandatory
```

See: *Advancing OpenMP Offload Debugging Capabilities in LLVM* (LLPP'21)

# Improved OpenMP GPU Runtime Information

```
$ clang++ -fopenmp -fopenmp-targets=nvptx64 -O3 -gline-tables-only sum.cpp -o sum
$ env LIBOMPTARGET_INFO=$((0x1 | 0x10 | 0x20)) ./sum
Entering OpenMP kernel at sum.cpp:5:1 with 3 arguments:
      firstprivate(N)[8] (implicit)
      tofrom(sum)[8] (implicit)
      to(A[:N])[8192]
Copying data from host to device, Size=8, Name=sum
Copying data from host to device, Size=8192, Name=A[:N]
Launching kernel __omp_offloading_fd02_60a38a2f__Z3sumPdm_l5 with 1 blocks and 128 threads in SPMD mode
```

See: *Advancing OpenMP Offload Debugging Capabilities in LLVM* (LLPP'21)

# Improved OpenMP GPU Runtime Checks

```
$ clang++ -fopenmp -fopenmp-targets=nvptx64 -fopenmp-target-debug=0x5 sum.cpp -o sum
$ env LIBOMPTARGET_DEVICE_RTL_DEBUG=0x5 ./sum
Shared memory stack full, fallback to dynamic allocation of global memory will negatively impact performance.
nullptr returned by malloc!
CUDA error: an illegal memory access was encountered
```

See: Co-Designing an OpenMP GPU Runtime and Optimizations (IPDPS'21)

# Remarks & Assumptions - Interactive Optimization

OpenMP-Opt emits remarks:

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

to report success and failure,

and utilizes assumptions:

- `#pragma omp assumes ...`
- `__attribute__((assume("...")))`
- command line flags

to enhance static analysis.

*New* environment assumptions:
`LIBOMPTARGET_MAP_FORCE_ATOMIC=false`

OpenMP 5.1 spec assumptions:

```
omp_no_openmp
omp_no_parallelism
omp_no_openmp_routines
```

LLVM assumption extensions:

```
ompx_spmd_amenable
ompx_aligned_barrier
ompx_no_sync
```

```
-fopenmp-cuda-mode
-fopenmp-assume-no-nested-parallelism
-fopenmp-assume-no-thread-state
-fopenmp-assume-teams-oversubscription
-fopenmp-assume-threads-oversubscription
```

# Example: LLVM Remarks

## OpenMP Optimization Remarks

The OpenMP-Aware optimization pass is able to generate compiler remarks for performed and missed optimisations. To emit them, pass these options to the Clang invocation: `-Rpass=openmp-opt -Rpass-analysis=openmp-opt -Rpass-missed=openmp-opt`. For more information and features of the remark system, consult the clang documentation:

- Clang options to emit optimization reports
- Clang diagnostic and remark flags
- The -foptimization-record-file flag and the -fsave-optimization-record flag

### OpenMP Remarks

| Diagnostics Number | Diagnostics Kind | Diagnostics Description |
|---|---|---|
| OMP100 | Analysis | Potentially unknown OpenMP target region caller. |
| OMP101 | Analysis | Parallel region is used in unknown / unexpected ways. Will not attempt to rewrite the state machine. |
| OMP102 | Analysis | Parallel region is not called from a unique kernel. Will not attempt to rewrite the state machine. |
| OMP110 | Optimization | Moving globalized variable to the stack. |
| OMP111 | Optimization | Replaced globalized variable with X bytes of shared memory. |
| OMP112 | Missed | Found thread data sharing on the GPU. Expect degraded performance due to data globalization. |
| OMP113 | Missed | Could not move globalized variable to the stack. Variable is potentially captured in call. Mark parameter as *__attribute__((noescape))* to override. |
| OMP120 | Optimization | Transformed generic-mode kernel to SPMD-mode. |
| OMP121 | Analysis | Value has potential side effects preventing SPMD-mode execution. Add *__attribute__((assume("ompx_spmd_amenable")))* to the called function to override. |
| OMP130 | Optimization | Removing unused state machine from generic-mode kernel. |
| OMP131 | Optimization | Rewriting generic-mode kernel with a customized state machine. |
| OMP132 | Analysis | Generic-mode kernel is executed with a customized state machine that requires a fallback. |
| OMP133 | Analysis | Call may contain unknown parallel regions. Use *__attribute__((assume("omp_no_parallelism")))* to override. |
| OMP140 | Analysis | Could not internalize function. Some optimizations may not be possible. |
| OMP150 | Optimization | Parallel region merged with parallel region at <location>. |
| OMP160 | Optimization | Removing parallel region with no side-effects. |
| OMP170 | Optimization | OpenMP runtime call <call> deduplicated. |
| OMP180 | Optimization | Replacing OpenMP runtime call <call> with <value>. |
| OMP190 | Optimization | Redundant barrier eliminated. (device only) |

```
for (int i = 0; i < j; i++)
  x_val -= x_ptr[i] * G_ptr[i]
```

```
$ clang++ -fopenmp -Rpass-ana
remark: loop not vectorized:
```

```
void work(void *);
void foo() {
  int local;
  work(&local);
}
#pragma omp declare target(foo)
```

```
$ clang++ -fopenmp -fopenmp-t
remark: Could not move globali
parameter as __attribute__((n
```

```
p simd
< j; i++)
tr[i] * G_ptr[i]
```

```
perations
```

```
ttribute__((noescape)) void *);
;
 declare target(foo)
```

```
op -O2
 captured in call. Mark
```

# Multi-architecture Binaries

- LLVM now supports compiling for many architectures
  - Allows the same binary to run on several machines
- Without --fopenmp-targets we will try to infer the triples

```
$ clang app.c -fopenmp -fopenmp-targets=nvptx64,amdgcn -c \
     -Xopenmp-target=nvptx64 --offload-arch=sm_80         \
     -Xopenmp-target=amdgcn --offload-arch=gfx90a
$ clang app.c -fopenmp --offload-arch=sm_80 --offload-arch=gfx90a -c
$ llvm-readelf -S app.o
Section Headers:
[Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
[11] .llvm.offloading LLVM_OFFLOADING 0000000000002058 002058 0024c0 00 E   0  0   8
[12] omp_offloading_entries PROGBITS  0000000000005048 004048 000020 00 A   0  0   8
```

Slide originally by *Joseph Huber*

# Multi-architecture Binaries

Can inspect the embedded device code with binary utils

```
$ clang app.c -fopenmp --offload-arch=sm_80 --offload-arch=gfx90a -o app
$ llvm-objdump --offloading ./app
OFFLOADING IMAGE [0]:
kind elf
arch gfx90a
triple amdgcn-amd-amdhsa
producer openmp

OFFLOADING IMAGE [1]:
kind elf
arch sm_80
triple nvptx64-nvidia-cuda
producer openmp
```

Slide originally by *Joseph Huber*

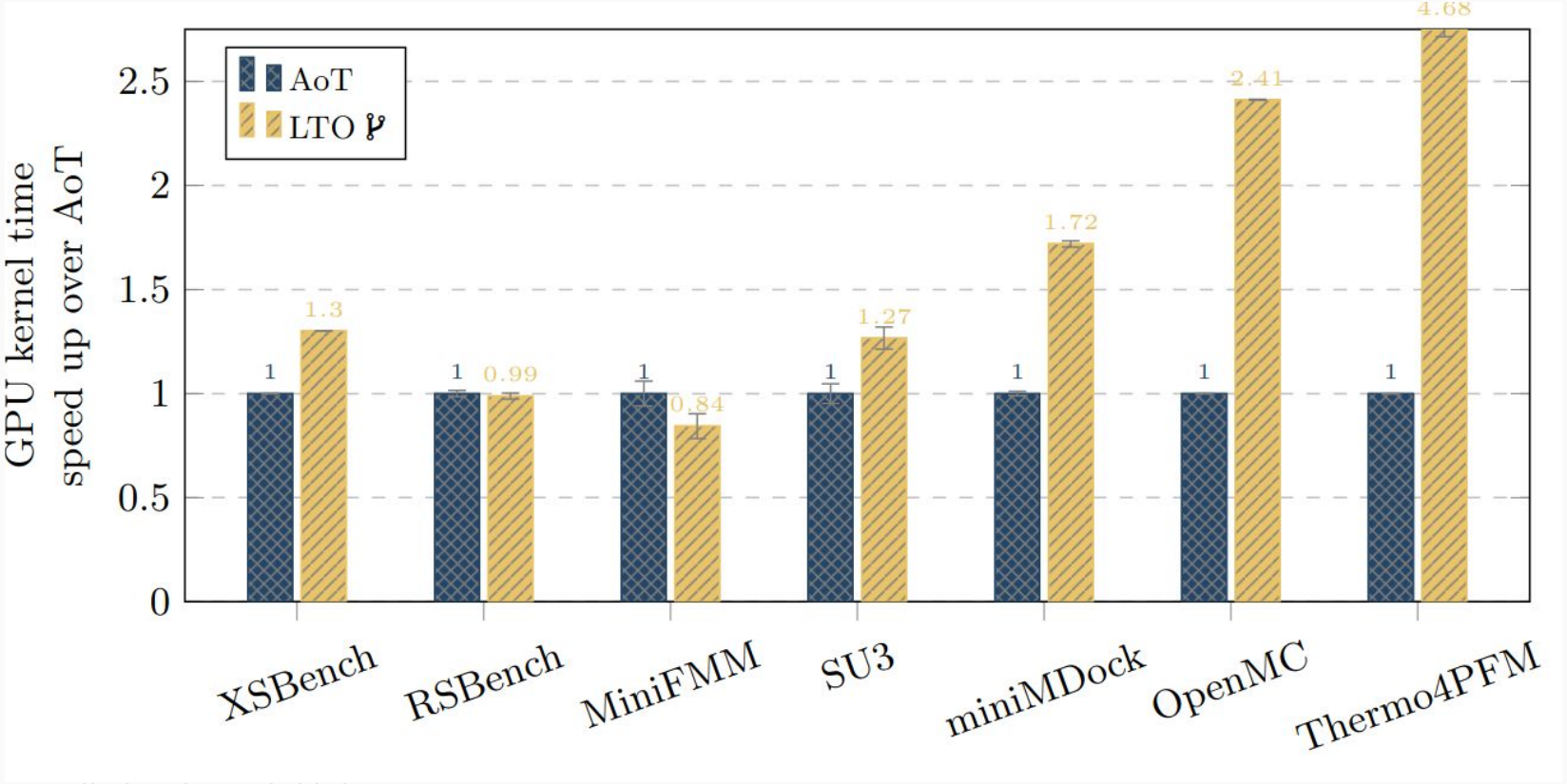
# Link Time Optimization (LTO)

- Compilers normally optimize a single translation unit (TU) at a time
  - LTO allows the compiler to optimize the whole program
- LLVM now supports LTO for the device
- Currently needs to be specified for both

```
$ clang app.c -fopenmp -fopenmp-targets=nvptx64 -foffload-lto -O3 -c
$ clang app.o -fopenmp -fopenmp-targets=nvptx64 -foffload-lto -O3
```

Slide originally by *Joseph Huber*

# LTO Performance Improvement (A100 Nvidia GPU)




Slide originally by *Joseph Huber*

# Static Library Support

- LLVM now completely supports static libraries
  - Any method of creating static libraries should work now
- The linker only imports used symbols from static libraries
  - Somewhat inherit this behaviour for multi-architecture binaries

```
$ clang foo.c -fopenmp --offload-arch=sm_70 --offload-arch=sm_80 --offload-arch=gfx908 -c
$ llvm-ar rcs libfoo.a foo.o
$ clang app.c -fopenmp --offload-arch=sm_70 -lfoo -o app
$ llvm-objdump --offloading
OFFLOADING IMAGE [0]:
kind elf
arch sm_70
triple nvptx64-nvidia-cuda
producer openmp
```

Slide originally by *Joseph Huber*

# Static Library Support

Can use this to create generic libraries, with LTO -> zero runtime overhead

```
#pragma omp begin declare target device_type(nohost)

#pragma omp begin declare variant match(...)
void foo() {...}
#pragma omp end declare variant

#pragma omp end declare target
```

```
$ clang device.c -c -fopenmp --offload-arch=sm_52,sm_70,sm_80,gfx908,gfx90a,gfx90c -O3 \
    -foffload-lto -fvisibility=hidden -fopenmp-cuda-mode
$ llvm-ar rcs libdevice.a device.o
$ clang app.c -fopenmp --offload-arch=sm_80 -foffload-lto -ldevice
```

Slide originally by *Joseph Huber*

# CUDA / HIP Interoperability

- The new driver can compile both CUDA and HIP
  - Requires explicitly using the new Driver
- LLVM now supports CUDA compilation in RDC-mode
  - Previously required external build systems

```
$ clang++ cuda.cu util.cu -fgpu-rdc --offload-arch=sm_70 --offload-new-driver -c
$ clang++ cuda.o util.o --offload-link -lcudart -o app
$ ./a.out
```

Slide originally by *Joseph Huber*

# CUDA / HIP Interoperability

OpenMP interoperability with CUDA/HIP

- Caveat: Global state is not yet shared; would require having state registered by OpenMP *or* CUDA

```
void openmp() { printf ("Hello from OpenMP\n"); }
#pragma omp declare target device_type(nohost) to(openmp)

__device__ cuda() { printf ("Hello from CUDA\n"); }
```

```
$ clang++ cuda.cu -fgpu-rdc --offload-arch=sm_70 --offload-new-driver -c
$ clang++ openmp.cpp -fopenmp --offload-arch=sm_70 -c
$ clang++ cuda.o openmp.o -fopenmp -fopenmp-targets=nvptx64 -lcudart
./a.out
Hello from OpenMP
Hello from CUDA
```

Slide originally by *Joseph Huber*

# Device Only Compilation

- Device only compilation to output the device code
  - Caveat: Can only output a single architecture currently
- Mainly useful for inspecting output

```
$ clang app.c -fopenmp --offload-arch=sm_70 -S -emit-llvm --offload-device-only -o -
< LLVM IR >
```

Slide originally by *Joseph Huber*

# Mandatory Offloading

- OpenMP offloading supports host-fallback by default
- This requires emitting each device function on the host
- Can be disabled using a command line flag
  - Makes interoperability with CUDA easier.

```
$ clang app.c -fopenmp --offload-arch=sm_70 -fopenmp-offload-mandatory
```

Slide originally by *Joseph Huber*

# Passing Arguments to the Device Linker

- The linker wrapper links many devices in a single invocation
- Extra arguments can be forwarded to the device linker if needed

```
$ clang app.c -fopenmp --offload-arch=sm_70 -Xoffload-linker -g
$ clang app.c -fopenmp --offload-arch=sm_70 -Xoffload-linker-nvptx64-nvidia-cuda -g
```

Slide originally by *Joseph Huber*

# LLVM/OpenMP Device Info

# llvm-omp-device-info

A command line utility that, by using `libomptarget`, and the device plugins, list devices information as seen from the OpenMP Runtime.

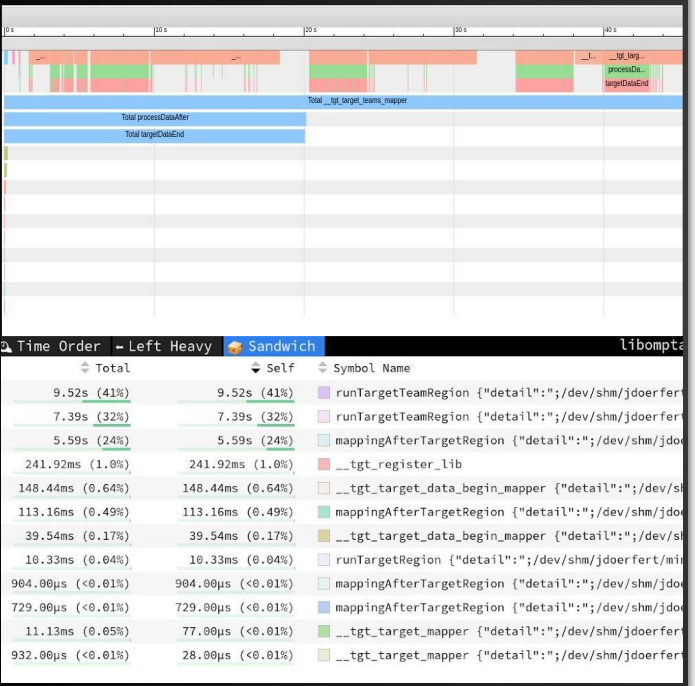
```
➜  ./llvm-omp-device-info
Device (0):
    print_device_info not implemented
...

Device (4):
    CUDA Driver Version:                11040
    CUDA Device Number:                 0
    Device Name:                        NVIDIA GeForce RTX 2080
    Global Memory Size:                 4294967295 bytes
    Number of Multiprocessors:          46
    Concurrent Copy and Execution:      Yes
    Total Constant Memory:              65536 bytes
    Max Shared Memory per Block:        49152 bytes
    Registers per Block:                65536
    Warp Size:                          32 Threads
    Maximum Threads per Block:          1024
    Maximum Block Dimensions:           1024, 1024, 64
    Maximum Grid Dimensions:            2147483647 x 65535 x 65535
    Maximum Memory Pitch:               2147483647 bytes
    Texture Alignment:                  512 bytes
    Clock Rate:                         1710000 kHz
    Execution Timeout:                  No
    Integrated Device:                  No
    Can Map Host Memory:                Yes
    Compute Mode:                       DEFAULT
    Concurrent Kernels:                 Yes
    ECC Enabled:                        No
    Memory Clock Rate:                  7000000 kHz
    Memory Bus Width:                   256 bits
    L2 Cache Size:                      4194304 bytes
    Max Threads Per SMP:                1024
    Async Engines:                      Yes (3)
    Unified Addressing:                 Yes
    Managed Memory:                     Yes
    Concurrent Managed Memory:          Yes
    Preemption Supported:               Yes
    Cooperative Launch:                 Yes
    Multi-Device Boars:                 No
    Compute Capabilities:               75
```

Credit to *Jose Monsalve Diaz* (ANL).

# LLVM/OpenMP Target Profiling

# Chrome Profiling Traces




*LLVM 12 introduced*

*LIBOMPTARGET_PROFILE=file.json*

*to portably track target interaction.*

*Chrome tracing format, source line information, …*

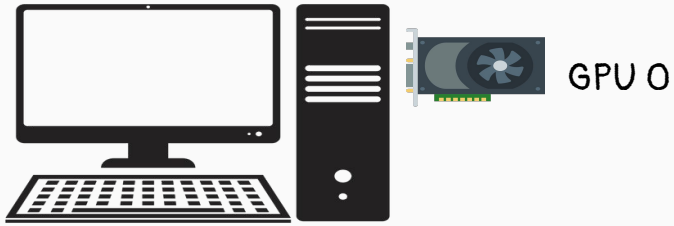
*https://openmp.llvm.org/docs/design/Runtimes.html#libomptarget-profile*

*LLVM 16 will allow kernel profiling including user-defined regions!*

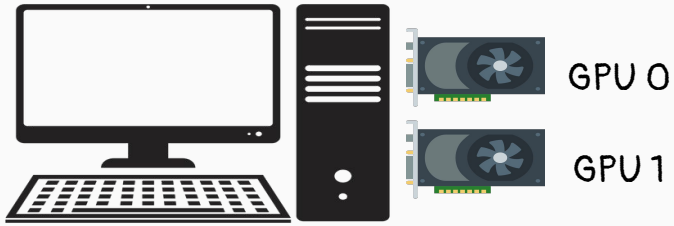
Credit to Giorgis Georgakoudis (LLNL).

# Remote OpenMP Offloading

# Remote OpenMP Offloading (Plugin)



# Remote OpenMP Offloading (Plugin)

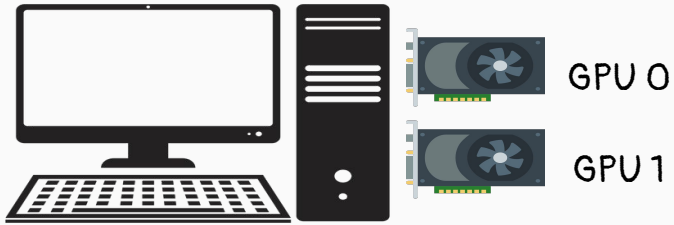


# Remote OpenMP Offloading (Plugin)

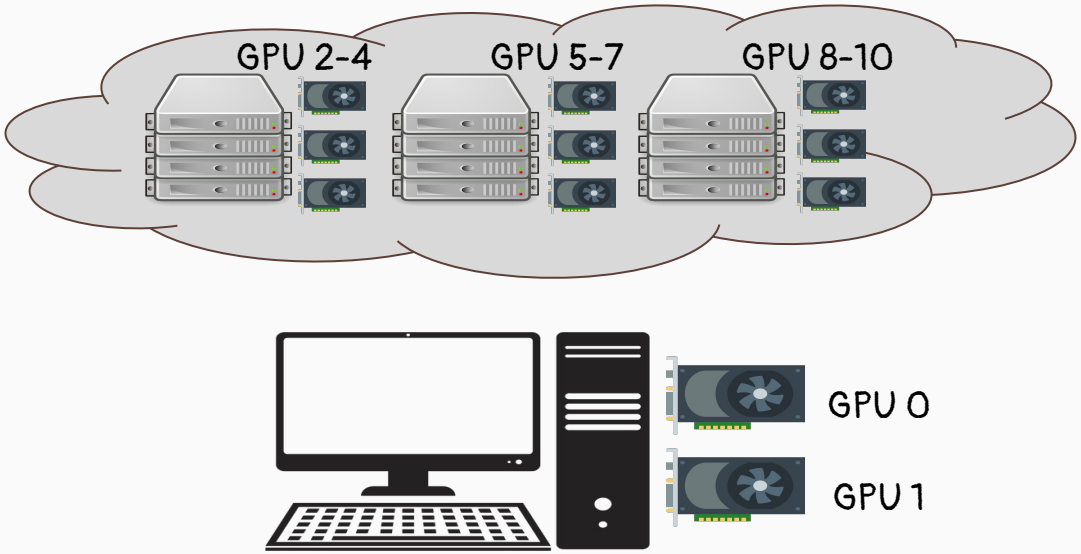
```
#pragma omp parallel for num_threads(num_devices)
for (auto K = 0; K < num_devices; K++) {
    #pragma omp target ... device(K)
    for (auto i = 0; i < lookups_per_device; i++) {

        ...

    }
}
```

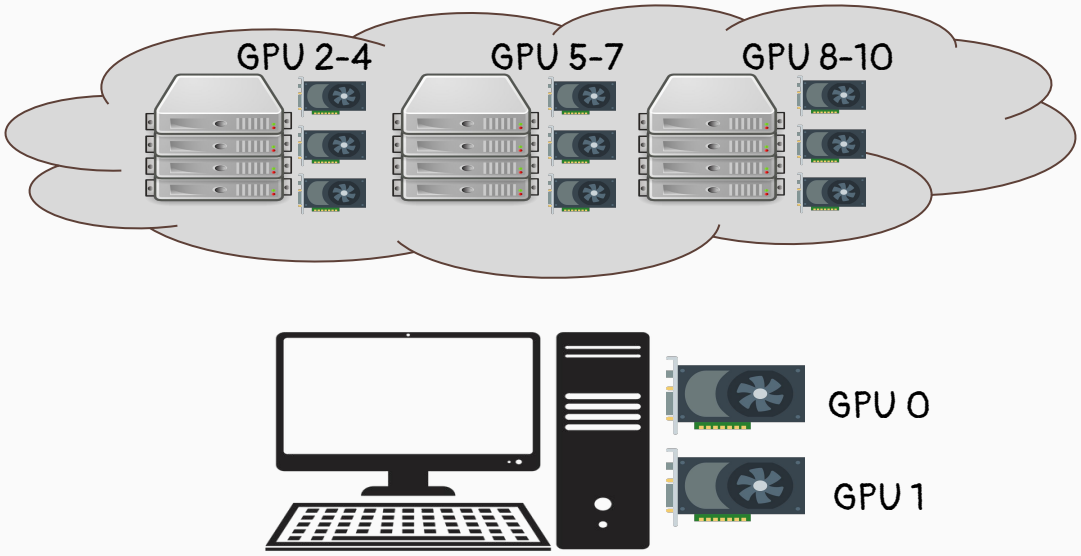


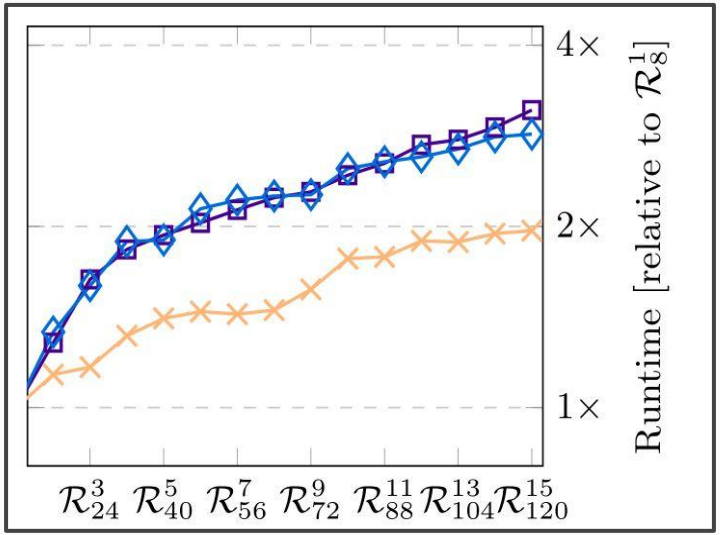
# Remote OpenMP Offloading (Plugin)




See: *Remote OpenMP Offloading* (ISC'22, **best paper**)

# Remote OpenMP Offloading (Plugin)




XSBench - scaling up to
15x8 A100 GPUs compared to 1x8

See: *Remote OpenMP Offloading* (ISC'22, **best paper**)

# OpenMP as Intermediate Layer

# LLVM/OpenMP as Target Independent Runtime Layer (WIP)

```
Thread 17 "su3" hit Breakpoint 1, k_mat_nn (a=0x[...]8b10, b=0x[...]
      cb20, c=0x[...]cc50, total_sites=256) at ./mat_nn_cuda.hpp:22
22 int myThread = blockDim.x * blockIdx.x + threadIdx.x;
(gdb) bt
#0 k_mat_nn (a=0x[...]8b10, b=0x[...]cb20, c=0x[...]cc50,
      total_sites=256) at ./mat_nn_cuda.hpp:22
#1 0x[...]d6dd in ?? () from /usr/lib64/libffi.so.7
#2 0x[...]9a69 in VGPUTy::VGPUTy()::{lambda()#2}::operator()() ()
   from [...]/lib/libomptarget.rtl.vgpu.so
(gdb) print myThread
$2 = 15
(gdb) next
25 if (mySite < total_sites) {
(gdb) cont
Continuing.

Thread 17 "su3" hit Breakpoint 2, k_mat_nn (a=<opt out>, b=<opt out>,
       c=0x[...]cc50, total_sites=256) at ./mat_nn_cuda.hpp:32
32 CMULSUM(a[mySite].link[j].e[k][m], b[j].e[m][l], cc);
(gdb) next
36 c[mySite].link[j].e[k][l] = cc;
(gdb) print cc
$3 = {real = 1, imag = 0}
```

Host GDB running the SU3 bench CUDA code via the OpenMP layer on the virtual GPU.

# LLVM/OpenMP as Target Independent Runtime Layer (WIP)

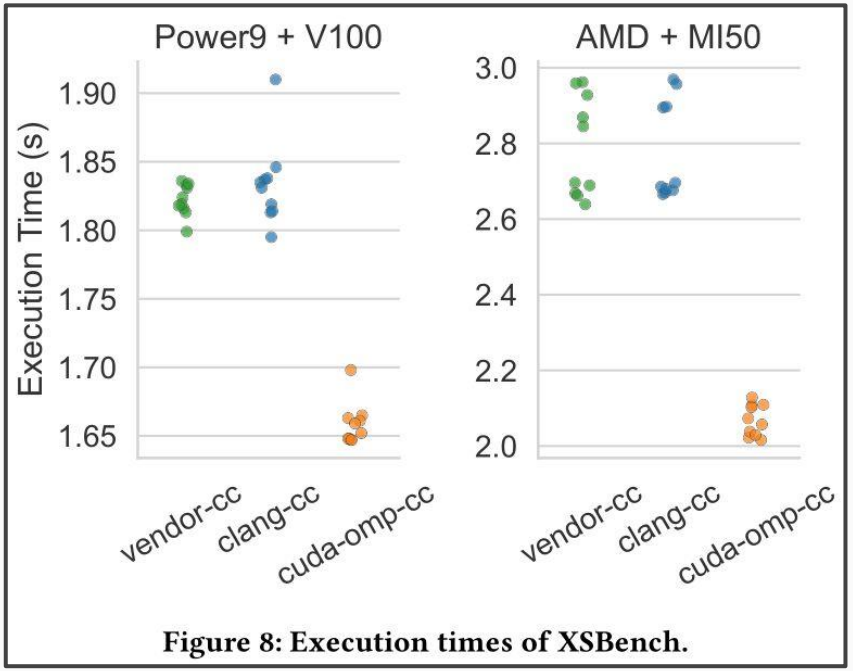



**Figure 8: Execution times of XSBench.**

Breaking the Vendor Lock — Performance Portable Programming Through OpenMP as Target Independent Runtime Layer (PACT'22, accepted)

# LLVM/OpenMP as Target Independent Runtime Layer (WIP)

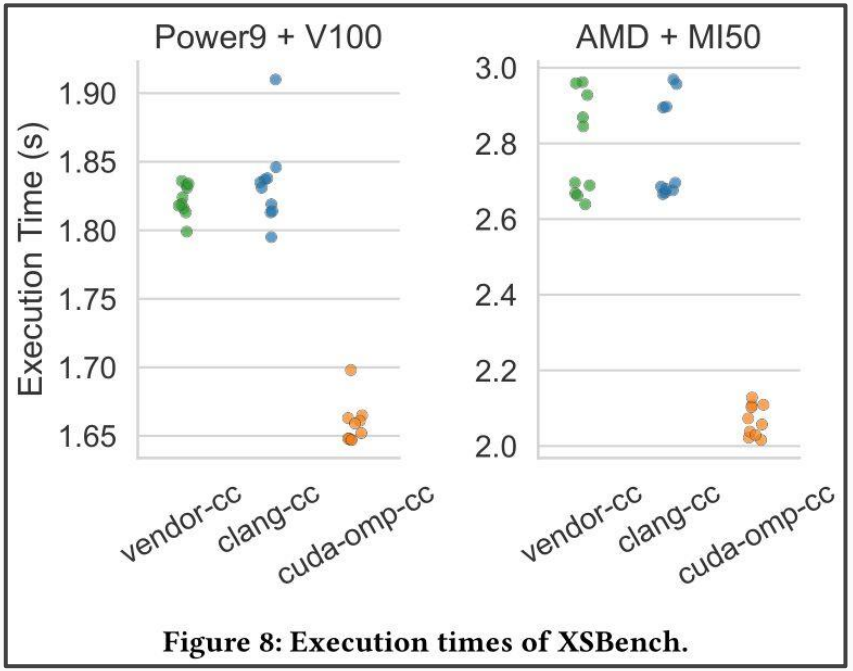



**Figure 8: Execution times of XSBench.**

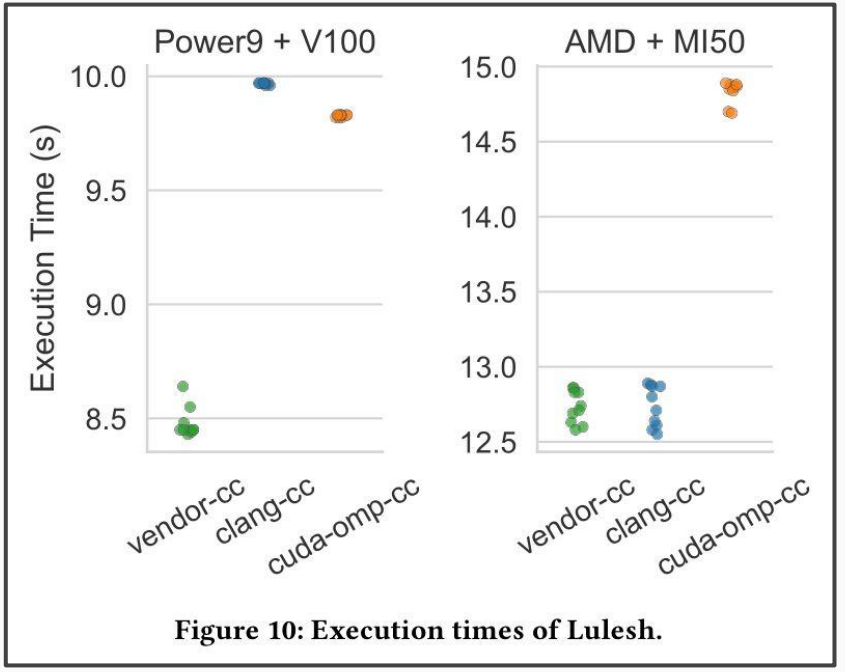
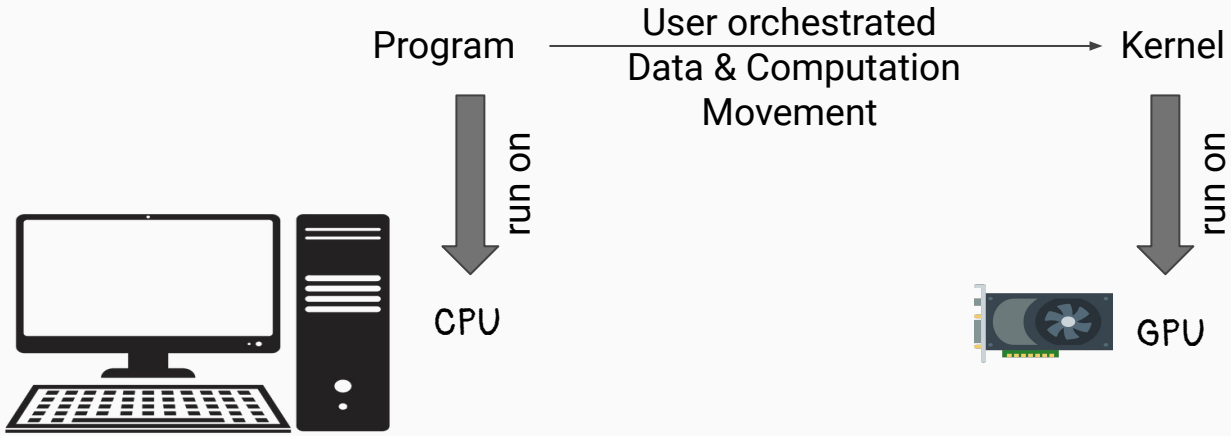



**Figure 10: Execution times of Lulesh.**

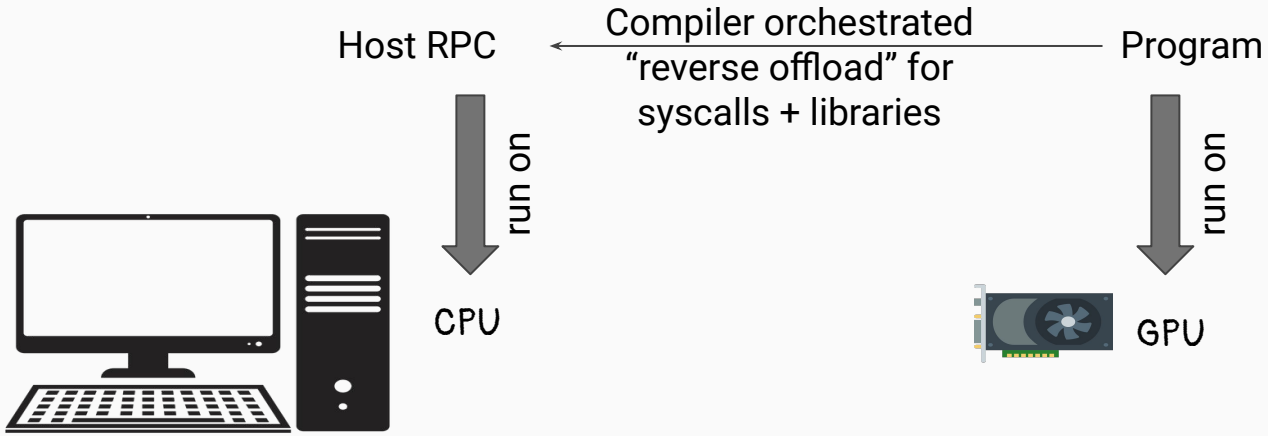
Breaking the Vendor Lock — Performance Portable Programming Through OpenMP as Target Independent Runtime Layer (PACT'22, accepted)

# Direct GPU Compilation

# Direct GPU offloading



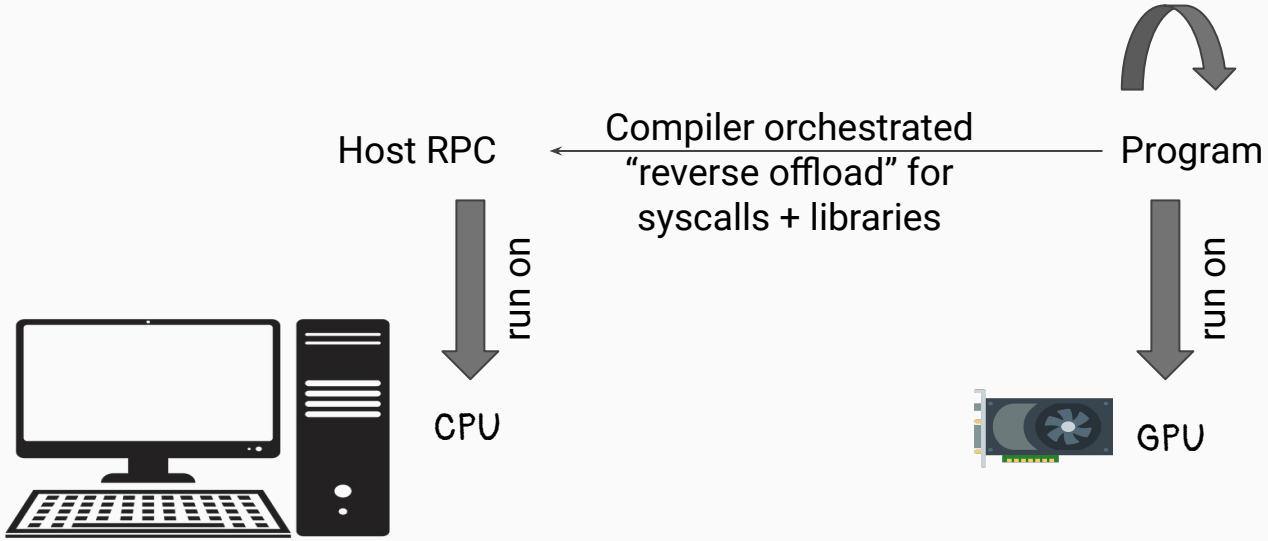
# Direct GPU offloading




Direct GPU Compilation and Execution for Host Applications with OpenMP Parallelism (LLVM-HPC’22, accepted)

# Direct GPU offloading

```
#omp parallel for ⟶
#omp target teams distribute parallel for
```



# Brief Recap & Outlook

# Brief Recap

# Brief Recap

**Enhanced GPU Debugging & Profiling**

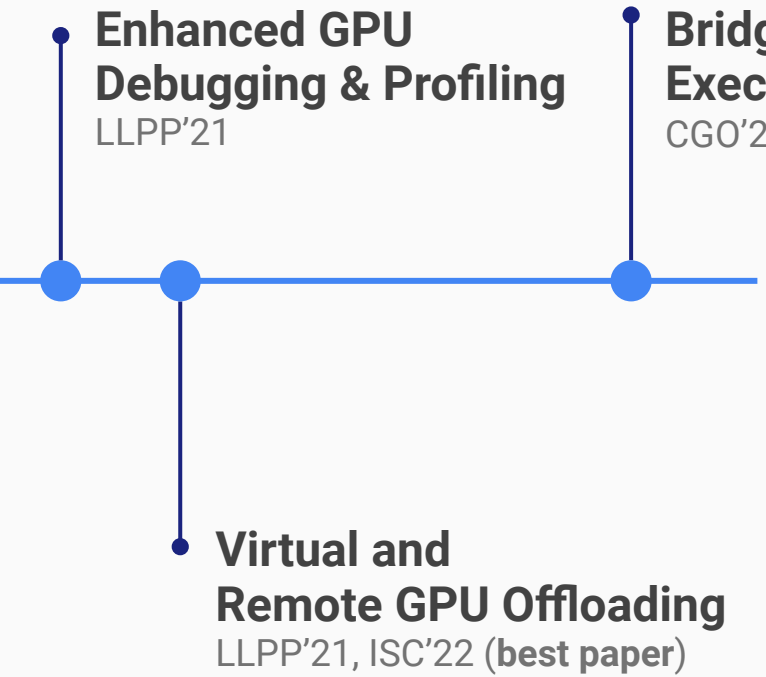
LLPP'21

# Brief Recap

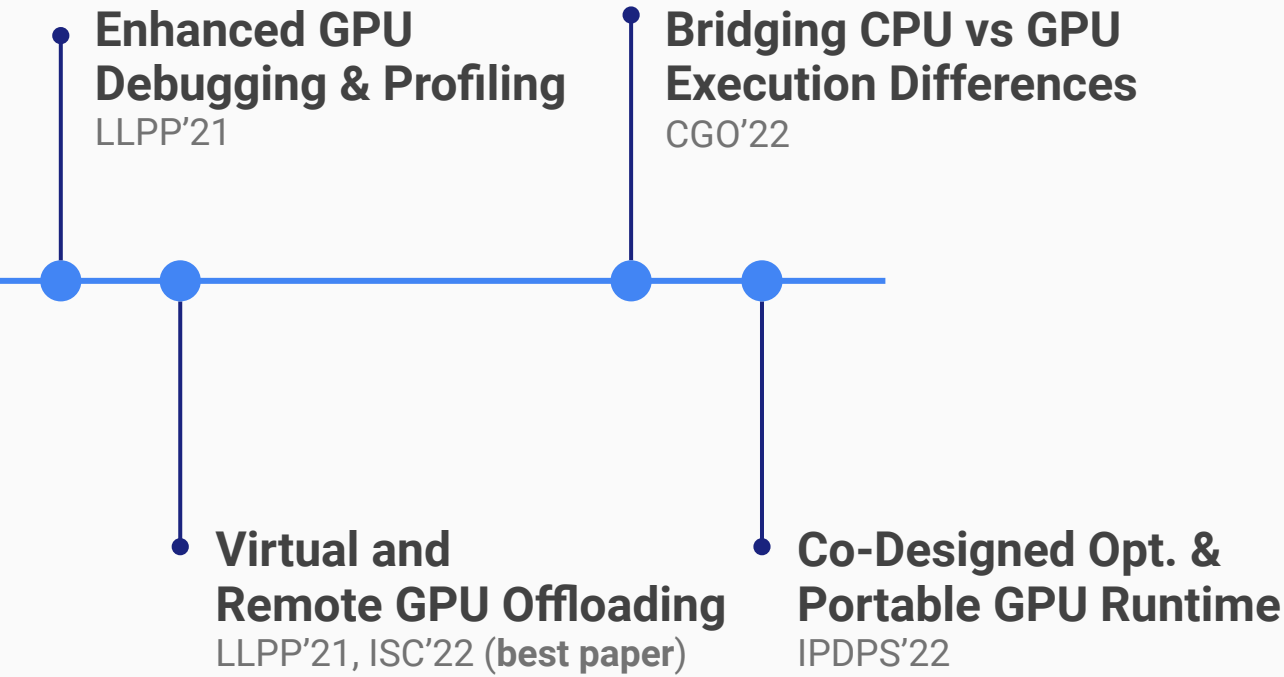
**Enhanced GPU Debugging & Profiling**
LLPP'21

**Virtual and Remote GPU Offloading**
LLPP'21, ISC'22 (**best paper**)

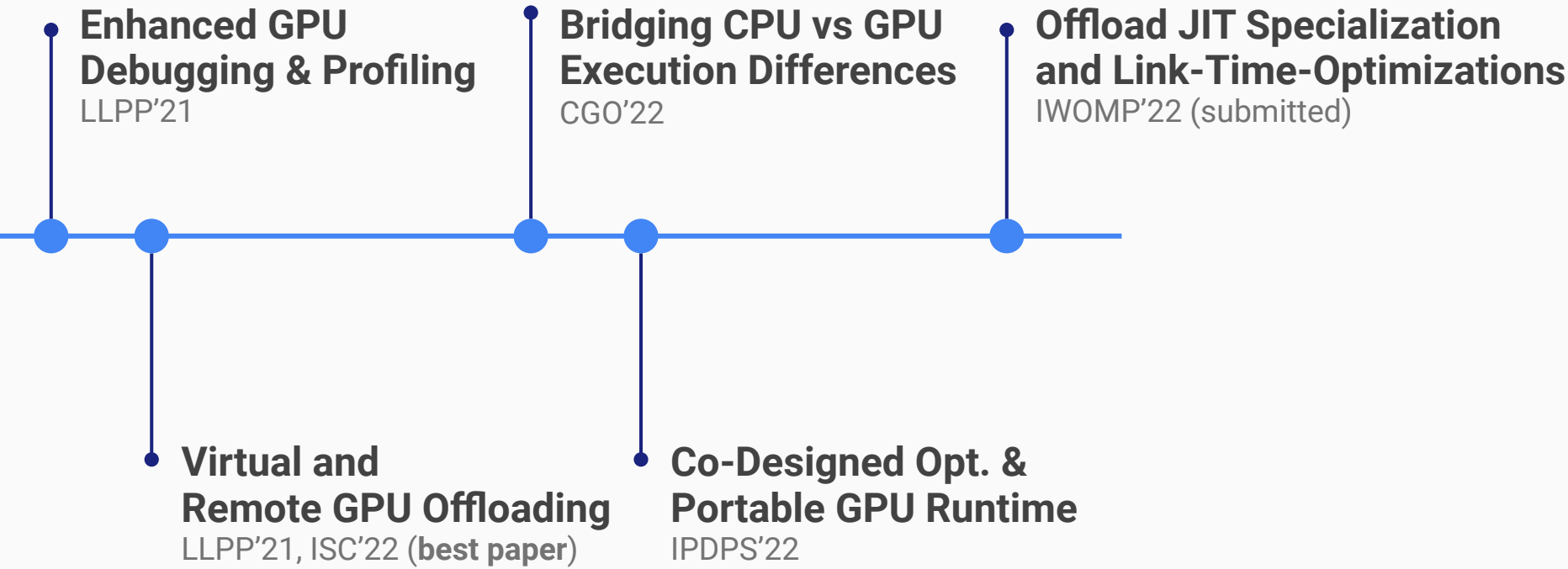
# Brief Recap


Enhanced GPU Debugging & Profiling
LLPP'21
Bridging CPU vs GPU Execution Differences
CGO'22
Virtual and Remote GPU Offloading
LLPP'21, ISC'22 (best paper)

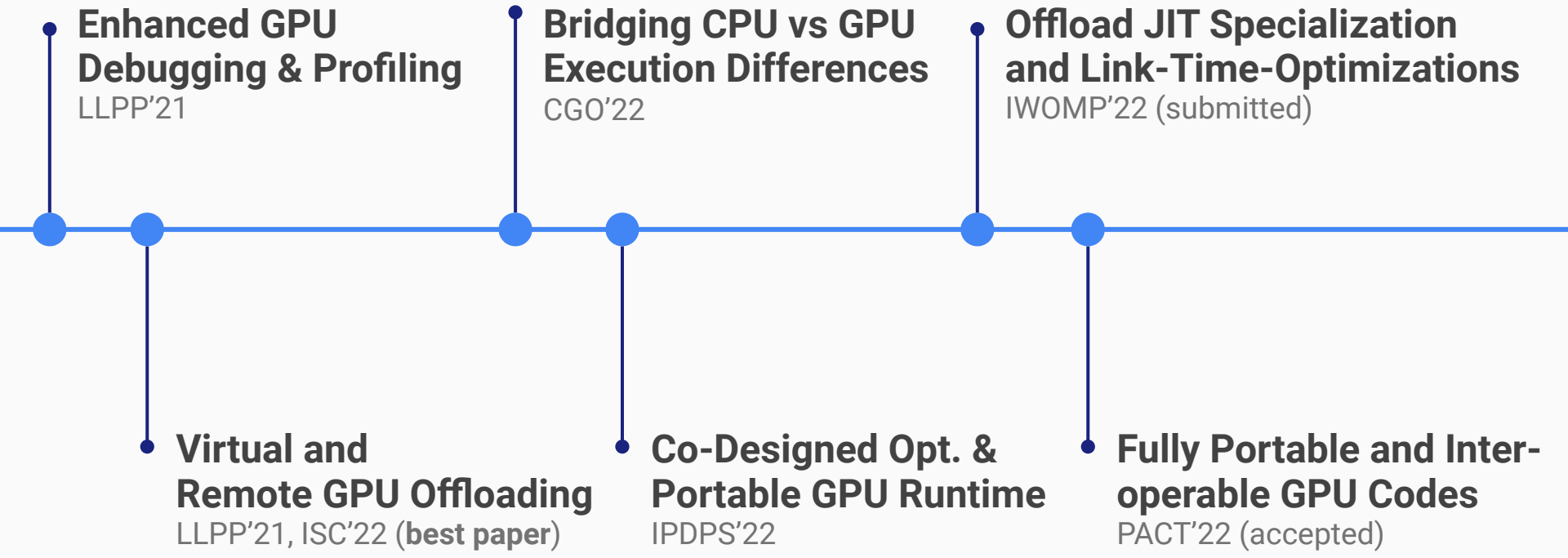
# Brief Recap


Enhanced GPU Debugging & Profiling
LLPP'21
Bridging CPU vs GPU Execution Differences
CGO'22
Virtual and Remote GPU Offloading
LLPP'21, ISC'22 (best paper)
Co-Designed Opt. & Portable GPU Runtime
IPDPS'22

# Brief Recap


Enhanced GPU Debugging & Profiling
LLPP'21
Bridging CPU vs GPU Execution Differences
CGO'22
Offload JIT Specialization and Link-Time-Optimizations
IWOMP'22 (submitted)
Virtual and Remote GPU Offloading
LLPP'21, ISC'22 (best paper)
Co-Designed Opt. & Portable GPU Runtime
IPDPS'22

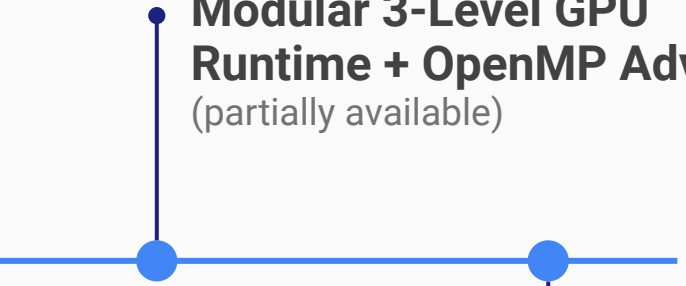
# Brief Recap


Enhanced GPU Debugging & Profiling
LLPP'21
Virtual and Remote GPU Offloading
LLPP'21, ISC'22 (best paper)
Bridging CPU vs GPU Execution Differences
CGO'22
Co-Designed Opt. & Portable GPU Runtime
IPDPS'22
Offload JIT Specialization and Link-Time-Optimizations
IWOMP'22 (submitted)
Fully Portable and Inter-operable GPU Codes
PACT'22 (accepted)

# Brief Outlook

**Modular 3-Level GPU Runtime + OpenMP Advisor**
(partially available)

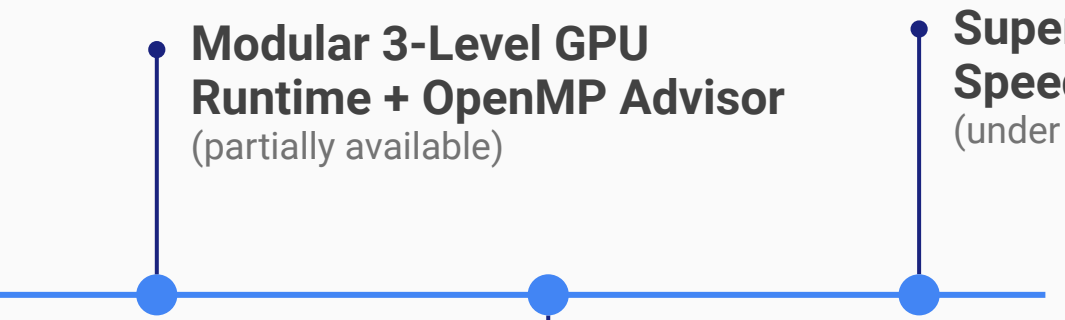
# Brief Outlook



**Modular 3-Level GPU Runtime + OpenMP Advisor**
(partially available)

**Record & Replay For (OpenMP) Target Regions**
(under development)

# Brief Outlook

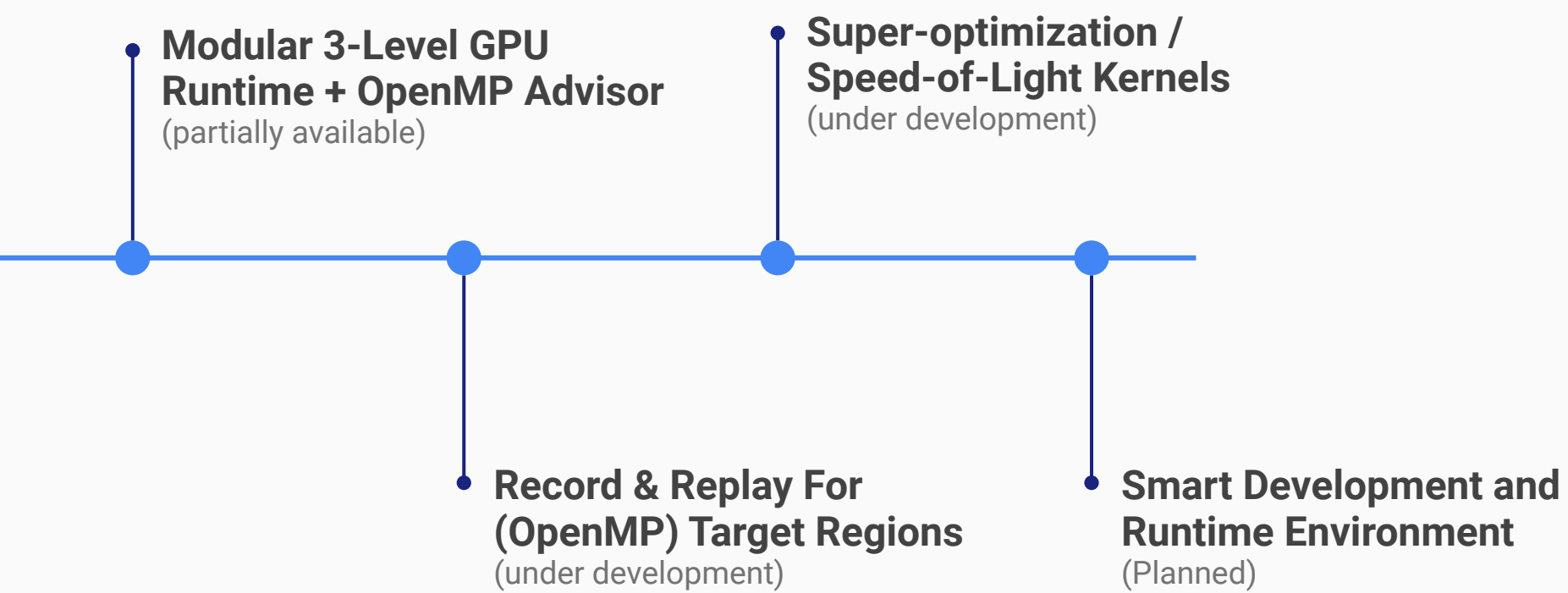


**Modular 3-Level GPU Runtime + OpenMP Advisor**
(partially available)

**Super-optimization / Speed-of-Light Kernels**
(under development)

**Record & Replay For (OpenMP) Target Regions**
(under development)

# Brief Outlook



# Brief Outlook