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:

```bash
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](http://llvm.org/docs/GettingStarted.html)
- [https://openmp.llvm.org/SupportAndFAQ.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](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](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](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 `-ffooffload-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
LLVM/OpenMP - A Community Effort


“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)
  https://openmp.llvm.org/

- Weekly OpenMP in LLVM meetings (~25 people), GPU focused.
  https://docs.google.com/document/d/1Tz8WFN13n7yJ-SCE0Qjqf9LmjGUw0dWO9Ts1ss4YOdg

- LLVM/OpenMP office hours every week
  https://llvm.org/docs/GettingInvolved.html#office-hours

- LLVM Discourse OpenMP category
  https://discourse.llvm.org/c/runtimes/openmp/35

- LLVM OpenMP Optimization Slack + weekly meeting
  https://join.slack.com/t/openmp-opt/shared_invite/zt-1fe
  rf7gnb-BfGLg9g0brr8s9YpFZVspg

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

*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`

**Command line flags**:
- `-fopenmp-cuda-mode`
- `-fopenmp-assume-no-nested-parallelism`
- `-fopenmp-assume-no-thread-state`
- `-fopenmp-assume-teams-oversubscription`
- `-fopenmp-assume-threads-oversubscription`
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: --openmp-analysis=loop-vectorize. 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 -optimization-record-file flag and the -save-optimization-record flag

### OpenMP Remarks

<table>
<thead>
<tr>
<th>Diagnostics Number</th>
<th>Diagnostics Kind</th>
<th>Diagnostics Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP100</td>
<td>Analysis</td>
<td>Potentially unknown OpenMP target region caller.</td>
</tr>
<tr>
<td>OMP101</td>
<td>Analysis</td>
<td>Parallel region is used in unknown / unexpected ways. Will not attempt to rewrite the state machine.</td>
</tr>
<tr>
<td>OMP102</td>
<td>Analysis</td>
<td>Parallel region is not called from a unique kernel. Will not attempt to rewrite the state machine.</td>
</tr>
<tr>
<td>OMP110</td>
<td>Optimization</td>
<td>Moving globalized variable to the stack.</td>
</tr>
<tr>
<td>OMP111</td>
<td>Optimization</td>
<td>Replaced globalized variable with X bytes of shared memory.</td>
</tr>
<tr>
<td>OMP112</td>
<td>Missed</td>
<td>Found thread data sharing on the GPU. Expect degraded performance due to data globalization.</td>
</tr>
<tr>
<td>OMP113</td>
<td>Missed</td>
<td>Could not move globalized variable to the stack. Variable is potentially captured in call. Mark parameter as <strong>attribute</strong>((noescape)) to override.</td>
</tr>
<tr>
<td>OMP120</td>
<td>Optimization</td>
<td>Transformed generic mode kernel to SPMD-mode.</td>
</tr>
<tr>
<td>OMP121</td>
<td>Analysis</td>
<td>Value has potential side effects preventing SPMD-mode execution. Add <strong>attribute</strong>((assume(&quot;ompx_spmd_aware&quot;))) to the called function to override.</td>
</tr>
<tr>
<td>OMP122</td>
<td>Optimization</td>
<td>Removing unused state machine from generic mode kernel.</td>
</tr>
<tr>
<td>OMP123</td>
<td>Analysis</td>
<td>Generic-mode kernel is executed with a customized state machine that requires a fallback.</td>
</tr>
<tr>
<td>OMP133</td>
<td>Analysis</td>
<td>Call may contain unknown parallel regions. Use <strong>attribute</strong>((assume(&quot;omp_spmd_parallelism&quot;))) to override.</td>
</tr>
<tr>
<td>OMP140</td>
<td>Analysis</td>
<td>Could not internalize function. Some optimizations may not be possible.</td>
</tr>
<tr>
<td>OMP150</td>
<td>Optimization</td>
<td>Parallel region merged with parallel region at &lt;location&gt;.</td>
</tr>
<tr>
<td>OMP160</td>
<td>Optimization</td>
<td>Removing parallel region with no side-effects.</td>
</tr>
<tr>
<td>OMP170</td>
<td>Optimization</td>
<td>OpenMP runtime call &lt;call&gt; deduplicated.</td>
</tr>
<tr>
<td>OMP180</td>
<td>Optimization</td>
<td>Replacing OpenMP runtime call &lt;call&gt; with &lt;value&gt;.</td>
</tr>
</tbody>
</table>

**Example: LLVM Remarks**

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

```bash
$ clang++ -fopenmp -Rpass-analysis=loop-vectorize example.cpp -O2
remark: loop not vectorized: cannot prove it is safe to reorder floating-point operations
```

```c
#pragma omp simd
for (int i = 0; i < j; i++)
x_val -= x_ptr[i] * G_ptr[i]
```

```c
void work(void *
); void foo()
{
int local;
work(&local);
}
```

```bash
#pragma omp declare target(foo)
$ clang++ -fopenmp -fopenmp-targets=nvptx64 -Rpass-analysis=openmp-opt example.cpp -O2
remark: Could not move globalized variable to the stack. Variable is potentially captured in call. Mark parameter as __attribute__((noescape)) void *);
```

```c
void work(__attribute__((noescape)) void *
); void foo()
{
int local;
work(&local);
}
```

```bash
#pragma omp declare target(foo)
```
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
```
Multi-architecture Binaries

Can inspect the embedded device code with binary utils

```bash
$ 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
```
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
```
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
```
CUDA / HIP Interoperability

OpenMP interoperability with CUDA/ HIP

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

```c
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.

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

Credit to Jose Monsalve Diaz (ANL).
LLVM/OpenMP Target Profiling
Chrome Profiling Traces

LLVM 12 introduced

$\texttt{LIBOMPTARGET_PROFILE=\text{file.json}}$

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)
#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)

See: Remote OpenMP Offloading (ISC’22, best paper)
OpenMP as Intermediate Layer
LLVM/OpenMP as Target Independent Runtime Layer (WIP)

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

```c
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[...]6ddd 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}
```
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

Host RPC

Compiler orchestrated
“reverse offload” for
syscalls + libraries

Program

#omp parallel for →

#omp target teams distribute parallel for

run on

run on

run on

CPU

GPU
Brief Recap & Outlook
Enhanced GPU Debugging & Profiling
LLPP’21
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)
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

- Virtual and Remote GPU Offloading
  LLPP’21, ISC’22 (best paper)

- Offload JIT Specialization and Link-Time-Optimizations
  IWOMP’22 (submitted)

- 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

- Fully Portable and Inter-operable GPU Codes
  PACT’22 (accepted)
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)
- Record & Replay For (OpenMP) Target Regions (under development)
- Super-optimization / Speed-of-Light Kernels (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)

Smart Development and Runtime Environment (Planned)
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)
- **Smart Development and Runtime Environment** (Planned)
- out of time