OpenMP Roadmap for Accelerators Across DOE Pre-Exascale/Exascale Machines

2022 ECP Community BOF Days

Approved for public release

JaeHyuk Kwack (ANL)  Bronis de Supinski (LLNL)  Wael Elwasif (ORNL)
Colleen Bertoni (ANL)  Tom Scogland (LLNL)  Barbara Chapman (HPE)
Kalyan Kumaran (ANL)  Stephen Olivier (SNL)  Jeff Larkin (NVIDIA)
Chris Daley (LBNL)    Thomas Applencourt (ANL)  Tim Costa (NVIDIA)
Reuben Budiardja (ORNL)  Michael Kruse (ANL)  Xinmin Tian (Intel)
Johannes Doerfert (ANL)  Vivek Kale (BNL)  Saiyedul Islam (AMD)
Ye Luo (ANL)  Catherine Moore (Siemens)  Deepak Eachempati (HPE)
              Tobias Burnus (Siemens)  Jeff Hammond (NVIDIA)

Wednesday, May 11, 2022
11:00 AM – 12:30 PM ET

Joe Zerr (LANL)
Carlo Bertolli (AMD)
Ron Lieberman (AMD)
Greg Rodgers (AMD)
Jeff Sandoval (HPE)
SPEAKERS

- JaeHyuk Kwack (ANL) - Introduction and moderator of vendors talks
- Kalyan Kumaran (ANL) - Moderator of panel discussion
- Johannes Doerfert (ANL) - Representative of LLVM and panelist
- Carlo Bertolli (AMD) - Representative of AMD and panelist
- Tobias Burnus (GNU, Siemens) - Representative of GNU and panelist
- Deepak Eachempati (HPE) - Representative of HPE and panelist
- Xinmin Tian (Intel) - Representative of Intel and panelist
- Jeff Hammond (NVIDIA) - Representative of NVIDIA and panelist
CONTRIBUTORS

- Colleen Bertoni (ANL)
- Chris Daley (LBL)
- Reuben Budiardja (ORNL)
- Joe Zerr (LANL)
- Bronis De Supinski (LLNL)
- Tom Scogland (LLNL)
- Stephen Olivier (SNL)
- Vivek Kale (BNL)
- Thomas Applencourt (ANL)
- Ye Luo (ANL)
- Michael Kruse (ANL)
- Wael Elwasif (ORNL)
- Catherine Moore (GNU, Siemens)
- Saiyed Islam (AMD)
- Ron Lieberman (AMD)
- Greg Rodgers (AMD)
- Jeff Sandoval (HPE)
- Barbara Chapman (HPE)
- Jeff Larkin (NVIDIA)
- Tim Costa (NVIDIA)
MOTIVATION FOR THIS BOF

- The current HPC environment is diverse and complex
  - Variety of hardware and multiple vendors providing their own programming interfaces and runtimes
- Critical for application developers to consider portable (and even better performance portable) solutions which can target different platforms across vendors
  - OpenMP is an open standard supported by nearly every vendor, and a promising solution
- Goals
  - Present vendors’ OpenMP roadmap for DoE pre-exascale/exascale systems
  - Discuss performance and evaluation, interoperability, feature support and implementation details, and community support
  - Give advice to application developers about what works well in implementations (both now and in the future)
MULTIPLE COMPILERS WILL SUPPORT A COMMON SET OF OPENMP DIRECTIVES ON GPUS (NON-EXHAUSTIVE LIST) (1/2)

As of 5/11/2022

✓ : yes
✓ (✓): yes with caveats
✘ : no

<table>
<thead>
<tr>
<th>Levels of parallelism</th>
<th>LLVM/Clang</th>
<th>AMD</th>
<th>HPE/Cray</th>
<th>Intel</th>
<th>NVIDIA</th>
<th>GNU (GCC 12)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 now, 3 under development</td>
<td>2 (teams, parallel)</td>
<td>2 (teams, parallel or simd)</td>
<td>3 (teams, parallel, simd)</td>
<td>2 (teams, parallel)</td>
<td>3 (teams, parallel, simd)</td>
</tr>
<tr>
<td>OpenMP directive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>declare target</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>map</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>target data</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>target enter/exit data</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>target update</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>teams</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>distribute</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>parallel</td>
<td>✓</td>
<td>✓</td>
<td>✓ (may be inactive)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>for/do</td>
<td>✓</td>
<td>✓</td>
<td>✓ (may be inactive)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>reduction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>simd</td>
<td>✓ (GPU under development)</td>
<td>✓ (on host)</td>
<td>✓</td>
<td>✓</td>
<td>✓ (ignored)</td>
<td>✓</td>
</tr>
<tr>
<td>atomic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ (OMP 5.1 ext)</td>
</tr>
<tr>
<td>critical</td>
<td>✓ (✓)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sections</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>master</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>single</td>
<td>✓ (✓)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>barrier</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>loop directive</td>
<td>eventually</td>
<td>✓ (recognize syntax)</td>
<td>✓ (Fortran only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>collapse of a perfectly nested loop</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>collapse of an imperfectly nested loop</td>
<td>✓</td>
<td>✓ (C/C++)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>collapse of a non-rectangular nested loop</td>
<td>✓</td>
<td>✓ (C/C++)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>loop transformation with tile</td>
<td>✓</td>
<td>✓</td>
<td>✓ (C/C++ only)</td>
<td>✓</td>
<td>✓</td>
<td>✓ (GCC13/OG12)</td>
</tr>
<tr>
<td>loop transformation with unroll</td>
<td>✓</td>
<td>✓</td>
<td>✓ (C/C++ only)</td>
<td>✓</td>
<td>✓</td>
<td>✓ (GCC13/OG12)</td>
</tr>
<tr>
<td>array reduction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ (GCC13/OG12)</td>
</tr>
<tr>
<td>scan</td>
<td>eventually</td>
<td>✓ (recognize syntax)</td>
<td>✓</td>
<td>✓</td>
<td>✓ (WIP)</td>
<td>✓</td>
</tr>
</tbody>
</table>

C/C++: ✓
/ F90: X (F90: GCC13/OG11)
X (GCC13/OG12)
X (GCC13/OG12)

/ F90 array sections: X
### MULTIPLE COMPILERS WILL SUPPORT A COMMON SET OF OPENMP DIRECTIVES ON GPUS (NON-EXHAUSTIVE LIST) (2/2)

<table>
<thead>
<tr>
<th>Requirement</th>
<th>LLVM/Clang</th>
<th>AMD</th>
<th>HPE/Cray</th>
<th>Intel</th>
<th>NVIDIA</th>
<th>GNU (GCC 12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires unified_shared_memory</td>
<td>✓</td>
<td>✓</td>
<td>✓ (some platforms)</td>
<td>✓</td>
<td>X (unnecessary)</td>
<td>X (WIP for nvptx)</td>
</tr>
<tr>
<td>requires dynamic_allocators</td>
<td>✓</td>
<td>✓</td>
<td>✓ (C/C++ only)</td>
<td>✓</td>
<td>X</td>
<td>✓ (GCC13/OG12)</td>
</tr>
<tr>
<td>declare reduction</td>
<td>✓</td>
<td>✓</td>
<td>✓ (C/C++ only)</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>declare mapper</td>
<td>✓</td>
<td>✓</td>
<td>✓ (C/C++ only)</td>
<td>✓</td>
<td>X</td>
<td>X (GCC13/OG12)</td>
</tr>
<tr>
<td>metadirective</td>
<td>✓</td>
<td>✓</td>
<td>✓ (limited, OMP 5.0 only)</td>
<td>X (WIP)</td>
<td>partial</td>
<td>X (GCC13/OG11)</td>
</tr>
<tr>
<td>declare variant</td>
<td>✓</td>
<td>✓</td>
<td>✓ (limited, OMP 5.0 only)</td>
<td>✓</td>
<td>partial</td>
<td>✓</td>
</tr>
<tr>
<td>&quot;target nowait&quot; supporting asynchronous execution</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ (sync w/ in_reduction)</td>
</tr>
<tr>
<td>&quot;target depend&quot; supporting fine-grained dependencies</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>&quot;target device&quot; supporting multiple non-host devices per process</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>use_device_addr</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ (+ has …)</td>
</tr>
<tr>
<td>detachable tasks: &quot;detach&quot; clause and &quot;omp_fulfill_event&quot; runtime routine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Memory management APIs

<table>
<thead>
<tr>
<th>API</th>
<th>LLVM/Clang</th>
<th>AMD</th>
<th>HPE/Cray</th>
<th>Intel</th>
<th>NVIDIA</th>
<th>GNU (GCC 12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocate directive for allocating variables in managed memory via allocator</td>
<td>✓</td>
<td>✓</td>
<td>✓ (extension)</td>
<td>✓</td>
<td>X</td>
<td>X (GCC13/OG11)</td>
</tr>
<tr>
<td>allocate clause for allocating privatized variables in managed memory via allocator</td>
<td>✓</td>
<td>✓</td>
<td>✓ (extension)</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>APIs for allocating/freeing memory via allocator</td>
<td>✓</td>
<td>✓</td>
<td>✓ (limited support on device with predefined allocators)</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>APIs for defining new allocators with custom traits (e.g. pinned memory)</td>
<td>✓</td>
<td>✓ (only pinned)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Interop objects/directive and APIs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C++ attribute syntax</td>
<td>eventually</td>
<td>✓</td>
<td>✓</td>
<td>✓ (WIP)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Orphaned parallel regions (any limitations? e.g. serialized)</td>
<td>No limitations</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓, parallel but slow</td>
<td>✓</td>
</tr>
<tr>
<td>Creating C++ objects containing virtual functions inside target regions (GPU)</td>
<td>✓</td>
<td>✓</td>
<td>✓ (WIP)</td>
<td>✓</td>
<td>X</td>
<td>✓ (if vtable+ methods emit.)</td>
</tr>
<tr>
<td>Mapping C++ objects containing virtual functions from host to the GPU</td>
<td>eventually</td>
<td>✓</td>
<td>✓</td>
<td>✓ (WIP)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>printf/print support in a target region (GPU)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Call CUDA/SYCL/HIP kernels in an OpenMP target region</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>CUDA works, but it depends on the details (✓) (may work)</td>
</tr>
</tbody>
</table>

As of 5/11/2022

- ✓: yes
- ✓ (✓): yes with caveats
- X: no
OPENMP RESOURCES

OpenMP website
– https://www.openmp.org

OpenMP Validation and Verification
– https://crpl.cis.udel.edu/ompvvso1ve/

OpenMP YouTube Channel
– https://www.youtube.com/user/OpenMPAR/1B/

OpenMP Users Monthly Teleconferences
– https://www.openmp.org/events/ecp-sollve-openmp-monthly-teleconference/

At 2022 ECP Annual Meeting:
– Early Experience of Application Developers with OpenMP Offloading
– Wed. May 4, 2022, 4:00 PM - 6:00 PM (ET)
– Recording available at ECP Annual Meeting Page
## SCHEDULE AT THIS BOF

<table>
<thead>
<tr>
<th>Topics</th>
<th>Minutes</th>
<th>Presenter or Moderator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
<td>JaeHyuk Kwack/ Colleen Bertoni</td>
</tr>
<tr>
<td><strong>Roadmap Presentations</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLVM</td>
<td>7</td>
<td>Johannes Doerfert</td>
</tr>
<tr>
<td>AMD</td>
<td>7</td>
<td>Carlo Bertolli</td>
</tr>
<tr>
<td>GNU</td>
<td>7</td>
<td>Tobias Burnus</td>
</tr>
<tr>
<td>HPE</td>
<td>7</td>
<td>Deepak Eachempati</td>
</tr>
<tr>
<td>Intel</td>
<td>7</td>
<td>Xinmin Tian</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>7</td>
<td>Jeff Hammond</td>
</tr>
<tr>
<td><strong>Panel discussion</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Preselected questions</td>
<td>45</td>
<td>Kalyan Kumaran and other panelists</td>
</tr>
<tr>
<td>- Questions/comments from audience (alternating)</td>
<td>45</td>
<td>Kalyan Kumaran and other panelists</td>
</tr>
<tr>
<td><strong>Total time</strong></td>
<td>90</td>
<td></td>
</tr>
</tbody>
</table>
ROADMAP PRESENTATIONS
LLVM/OpenMP in HPC
A Brief Overview
Building LLVM + OpenMP offloading

Single command often suffices to configure:
```
cmake /src/llvm-project/llvm -DLLVM_ENABLE_PROJECTS='clang;lld' -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
```

Various resources available online! Start here:

https://llvm.org/docs/GettingStarted.html
https://openmp.llvm.org/SupportAndFAQ.html
LLVM/OpenMP Features

- Device-side LTO for OpenMP offload (and CUDA)
- OpenMP offloading to a remote process (or to remote GPUs)
- Host debugging on the OpenMP virtual GPU
- Mix CUDA device code and OpenMP offload code
- JIT compilation (and specialization) for OpenMP offload kernels
- Extraction of OpenMP kernels and isolated replay, tuning, etc. [WIP]
- Portable wrapper for common libraries (Thrust, BLAS, ...) [WIP]
OpenMP-Aware Optimizations
Automatic SPMDzation + shared memory usage (LLVM 13+)

```
#pragma omp target teams
{
    double team_local_memory[M];
    team_main_thread_only();
    #pragma omp parallel
every_thread(team_local_memory);
}
```

SPMDzation - “CUDA”-like execution mode

```
#pragma omp target teams
#pragma omp parallel
{
    double team_local_memory[M];
    #pragma omp allocate(team_local_memory)
    allocator(omp_cgroup_mem_alloc)
    #pragma omp masked
team_main_thread_only();
    #pragma omp barrier
every_thread(team_local_memory);
}
```

Shared memory usage for scratchpads

Automatic guarding and synchronization
OpenMP-Optimization Remarks & Assumptions

1) OpenMP-Opt emits remarks (above)
2) The web provides explanations (right)
3) Users add OpenMP assumptions, e.g.,
   #pragma omp assume ext_spmd_amenable

Visit openmp.llvm.org for more!

https://openmp.llvm.org/remarks/OptimizationRemarks.html
OpenMP offload Recommendations

- 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` and `-fopenmp-assume-no-thread-state`
- Use device-side LTO `-foffload-lto`
OffloadArch Library & *offload-arch* Tool

- Tool (and LLVM™ library) to query capabilities of the target runtime
  - Like, (arch name: gfx90a, or features like shared memory ECC turned on/off)

- Capabilities
  - Pre-decided characteristics of the target which require a dedicated image in a fat binary.

- libomptarget uses LLVM library interface to query the target system and extract a compatible image, if any.

- Works with multi-GPU systems as well

- Query a binary for list of image requirements

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Print the help message.</td>
</tr>
<tr>
<td>a</td>
<td>Print values for all devices. Don't stop at first device found.</td>
</tr>
<tr>
<td>m</td>
<td>Print device code name (often found in pci.ids file).</td>
</tr>
<tr>
<td>n</td>
<td>Print numeric pci-id.</td>
</tr>
<tr>
<td>t</td>
<td>Print clang offload triple to use for the offload arch.</td>
</tr>
<tr>
<td>v</td>
<td>Verbose = -a -m -n -t For all devices, print codename, numeric value and triple</td>
</tr>
<tr>
<td>f</td>
<td>Print offload requirements including offload-arch for each compiled offload image built into an application binary file.</td>
</tr>
<tr>
<td>c</td>
<td>Print offload capabilities of the underlying system. This option is used by the language runtime to select an image when multiple images are available. A capability must exist for each requirement of the selected image.</td>
</tr>
</tbody>
</table>
Multi-architecture Compilation

- Possible target configs:
  1. gfx906 and gfx906
  2. gfx908:xnack- and gfx908:xnack+
  3. (gfx906 and gfx908) or (sm_70 and sm_85)
  4. gfx906 and sm_70

- Build a common binary which can run on one GPU at a time for any of the above configuration

- Build once, run anywhere!

- Generate a multi-image binary such that:
  - Each image is tagged and compiled for a specific target
    - create a ToolChain for each target in clang driver
  - Tags should be sufficient to uniquely describe its target
    - define “Requirements” of image
  - Images are packed in a (fat) binary
    - use clang-offload-wrapper
  - Load the right image from the binary at the runtime, using mechanisms:
    - to identify characteristics of the current target (H/W + S/W configuration)
      - use OffloadArch library to identify “Capabilities” of current target
    - to test compatibility of current target with each image in the binary
      - modify libomptarget

```
clang -O2 -fopenmp-fopenmp-targets=amdgcn-amd-amdhsa,amdgcn-amd-amdhsa /
-Xopenmp-target=amdgcn-amd-amdhsa -march=gfx906:xnack- /
-Xopenmp-target=amdgcn-amd-amdhsa -march=gfx908:xnack+ /
helloworld.c -o helloworld
```
Unified Shared Memory

• Modes:
  • Default Mode
  • USM Mode (maps are optional)

• Default mode → USM Mode (always portable)
• USM Mode → Default Mode (not necessarily)

• ROCm™ AMDGPU Implementation USM Mode → maps give better performance
  • Maps → Coarse grain memory
  • Coarse grain faster than fine grain

• Programs written for default mode will give best USM mode performance

• Maps are the way to incrementally improve performance of critical/hotspot kernels in USM mode
Unified Shared Memory on ROCm™ AMDGPU

```c
#pragma omp requires unified_shared_memory
int main() {
    double *a = new double[n];
    double *b = new double[n];

    #pragma omp target teams distribute parallel for map(tofrom: a[:n]) map(to: b[:n])
    for(int i = 0; i < n; i++)
        a[i] += b[i];
}
```

- If maps are used, pages used by a and b switch to coarse grain
- Still, no device memory allocation, nor memory copies

```sh
clang -fopenmp -fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa
    -march=gfx90a helloworld.c -o helloworld

HSA_XNACK=1 ./helloworld
```
Disclaimer and Attribution

DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.’ AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

©2022 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Radeon is the registered trademark of Advance Micro Devices, Inc. LLVM is a trademark of LLVM Foundation. The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.
GNU Compiler Collection (GCC) & OpenMP

- Widely used & supported open-source software
  - contributing is simple & welcoming community
  - paid/unpaid contributors
  - Linux distros also pack offloading support (via optional packages)
- C17 (most of C2x), C++20 (most of C++23)
  Fortran: 2008 + coarray/interop TS (mostly), initial F2018
- OpenMP/OpenACC support in C, C++, Fortran
  - Full OpenMP 4.5, much of 5.0, some of 5.1
  - OpenACC 2.6
- Offloading to nvptx (Nvidia) + AMD GCN (Radeon)
- Annual major releases around late spring (~ end of April)
  - GCC 12: Released on May 6, 2022
  - GCC 11: Released April 2021, last 11.3 (April 2022)
    → Linux distros use git branch directly, mainline also quite stable
    → Also avail: OG12 (= devel/omp/gcc-12) SIEMENS’ public branch

OpenMP Now Supported & Implementation Status

GCC 11
- Non-rect loop nests, allocator routines, declare variant ext. (C/C++)
- Fortran: full OpenMP 4.5, order(concurrent), device_type, memorder-clauses for flush, lastprivate with conditional modifier, atomic construct and reduction clause 5.0 ext.
- GCN:gfx908 (MI100) support

GCC 12
- OpenMP 5.1: C++ 11 attributes, masked/scope/error/nothing, atomic extensions, memory-allocation routines, strictly structured blocks
- OpenMP 5.0: affinity clause. Fortran: declare variant, depobj, mutixinoutset, iterator, defaultmap 5.0 ext., loop
- GCN: Debugging (ROCGDB), wavefronts per compute unit restrictions lifted, wavefront-workgroup tunings
- NVPTX: Updates related to sm_xx target and PTX ISA

Mainline (GCC 13): Several OpenMP patches already pending

Supported Releases

GCC 12.1 (changes)
Status: 2022-04-28 (frozen for release).
Serious regressions. All regressions.

GCC 11.3 (changes)
Status: 2022-04-21 (regression fixes & docs only).
Serious regressions. All regressions.

GCC 10.3 (changes)
Status: 2021-04-08 (regression fixes & docs only).
Serious regressions. All regressions.

GCC 9.4 (changes)
Status: 2021-06-01 (regression fixes & docs only).
Serious regressions. All regressions.

Development: GCC 13.0 (release criteria, changes)
Status: 2022-04-28 (general development).
Serious regressions. All regressions.

2 OpenMP Implementation Status

- OpenMP 4.5: Feature completion status to 4.5 specification
- OpenMP 5.0: Feature completion status to 5.0 specification
- OpenMP 5.1: Feature completion status to 5.1 specification

GCC → 12 Changes → OpenMP or
https://gcc.gnu.org/onlinedocs/libgomp/
Following OpenMP Spec, Appendix B
Enabling offloading

- `fopenmp` – automatically enables offloading for `omp` target regions
- `fopenmp-simd` – only SIMD, no parallelization/lib dependency
- `foffload=[disable|default|nvptx-none,amdgcn-amdhsa,...]`
  Disable offloading, use default (all avail), or only specified types (list)

Argument passing to offload compiler

- `foffload-options=-lm` `-foffload-options=nvptx-none=-latomic`

GCC <12: Use `-foffload= instead (undocumented, has corner case)

Optimization

- `-O0` (default), `-O1/-O2/-O3`, `-Og`, `-Ofast` (`-ffast-math`)
- `-mveclibabi=[svml,acml,mass]` vector math libs by Intel/AMD/IBM

Diagnostic

- `-fopt-info=-` (`-fopt-info-loops`, `-fopt-info-omp`, `-fopt-info-vec-missed`, …):
  Checking/debugging optimizations
Offload Targets

Nvidia GPUs (nvptx)

• GCC generates nvptx (generic code)
• JIT compiled by CUDA run-time library at startup (→ CUDA_CACHE docu)
• -march=sm_xx (GCC 12) / -misa=sm_xx (alias + GCC < 12)
  
  - sm_30, sm_35, (GCC 12:) sm_53, sm_70, sm_75, sm_80

• -march-map=sm_xx: (GCC 12) maps sm_xx to a supported sm_xx (↑)

• https://github.com/MentorEmbedded/fortran-cuda-interfaces – cublas, cublas_v2, cublasxt, openacc_cublas, cufft

AMD GCN

• GCN generates code for: fiji (GCN3, gfx803), gfx900/gfx906 (GCN5, VEGA 10/20), gfx908 (MI100)
  
  • Example: -fopenmp-options=-march=gfx908
  • Offload debugging with GCC 12 and ROCGDB: https://linuxplumbersconf.org/event/11/contributions/997/
Acknowledgement

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725

Disclaimer

© Siemens 2022

Subject to changes and errors. The information given in this document only contains general descriptions and/or performance features which may not always specifically reflect those described, or which may undergo modification in the course of further development of the products. The requested performance features are binding only when they are expressly agreed upon in the concluded contract.

All product designations may be trademarks or other rights of Siemens AG, its affiliated companies or other companies whose use by third parties for their own purposes could violate the rights of the respective owner.
HPE CRAY COMPILING ENVIRONMENT (CCE)

- Fortran compiler
  - Proprietary front end and optimizer; HPE-modified LLVM backend
  - Fortran 2018 support (including coarray teams)

- C and C++ compiler
  - HPE-modified closed-source build of Clang+LLVM compiler
  - C11 and C++17 support
  - UPC support

- OpenMP Offloading support for NVIDIA/AMD GPUs
  - OpenMP 4.5 and partial 5.0/5.1
  - some differences between Fortran and C/C++ compilers in support
  - Other models available: OpenACC 2.0 (Fortran only), HIP (C++, AMD GPUs only)
CCE OPENMP SUPPORT

• Uses proprietary OpenMP runtime libraries

• Supports cross-language and cross-vendor OpenMP interoperability
  • CCE’s libcraymp behaves as drop-in replacement for Clang’s libomp and GNU’s libgomp
  • GNU OpenMP interface support is limited to OpenMP 3.1 constructs – update planned for future release

• Implements HPE-optimized code generation for OpenMP offload regions

• OpenMP 5.0 and 5.1 – in progress, implemented over several CCE releases
  • See release notes and intro_openmp man page for full list of supported features
  • OpenMP 5.0 is near complete as of CCE 13.0 (Nov 2021)
  • OpenMP 5.1/5.2 support in progress for 2022-2023
# CCE OPENMP 5.0 STATUS

## CCE 10.0 (May 2020)
- OMP_TARGET_OFFLOAD
- reverse offload
- implicit declare target
- omp_get_device_num
- OMP_DISPLAY_AFFINITY
- OMP_AFFINITY_FORMAT
- set/get affinity display
- display/capture affinity
- requires
- unified_address
- unified_shared_memory
- atomic_default_mem_order
- dynamic_allocators
- reverse_offload
- combined master constructs
- acq/rel memory ordering (Fortran)
- deprecated nested-var
- taskwait depend
- simd nontemporal (Fortran)
- lvalue map/motion list items
- allow != in canonical loop
- close modifier (C/C++)
- extend defaultmap (C/C++)

## CCE 11.0 (Nov 2020)
- noncontig update
- map Fortran DVs
- host teams
- use_device_addr
- nested declare target
- allocator routines
- OMP_ALLOCATOR
- allocate directive
- allocate clause
- order(concurrent)
- atomic hints
- default nonmonotonic
- imperfect loop collapse
- pause resources
- atoms in simd
- simd in simd
- detachable tasks
- omp_control_tool
- OMPT
- declare variant (Fortran)
- loop construct
- metadirectives (Fortran)
- pointer attach
- array shaping
- acq/rel memory ordering (C/C++)
- device_type (C/C++)
- non-rectangular loop collapse (C/C++)

## CCE 12.0 (Jun 2021)
- device_type (Fortran)
- affinity clause
- conditional lastprivate (C/C++)
- simd if (C/C++)
- iterator in depend (C/C++)
- depobj for depend (C/C++)
- task reduction (C/C++)
- task modifier (C/C++)
- simd nontemporal (C/C++)
- scan (C/C++)
- lvalue list items for depend
- mutexinoutset (C/C++)
- taskloop cancellation (C/C++)

## CCE 13.0 (Nov 2021)
- declare variant (C/C++)
- metadirectives (C/C++)
- mapper (C/C++)
- extend defaultmap (Fortran)
- loop construct
- extend defaultmap (Fortran)
- close modifier (Fortran)
- mutexinoutset (Fortran)

## CCE 14.0 (May 2022)
- task reduction (Fortran)
- task modifier (Fortran)
- target task reduction (Fortran)
- simd if (Fortran)

## Future CCE Release
- loop construct (C/C++)
- mapper (Fortran)
- iterator in depend (Fortran)
- non-rectangular loop collapse (Fortran)
- depobj for depend (Fortran)
- uses_allocators
- concurrent maps
- taskloop cancellation (Fortran)
- scan (Fortran)
- target task reduction (C/C++)

Refer to CCE release notes or intro_openmp man page for current implementation status
## OPENMP CONSTRUCT MAPPING TO GPU

<table>
<thead>
<tr>
<th>NVIDIA</th>
<th>AMD</th>
<th>CCE Fortran OpenACC</th>
<th>CCE Fortran OpenMP</th>
<th>CCE C/C++ OpenMP</th>
<th>Clang C/C++ OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threadblock</td>
<td>Work group</td>
<td>acc gang</td>
<td>omp teams</td>
<td>omp teams</td>
<td>omp teams</td>
</tr>
<tr>
<td>Warp</td>
<td>Wavefront</td>
<td>acc worker</td>
<td>omp simd</td>
<td>omp parallel or</td>
<td>omp parallel</td>
</tr>
<tr>
<td>Thread</td>
<td>Work item</td>
<td>acc vector</td>
<td>omp simd</td>
<td>omp simd</td>
<td></td>
</tr>
</tbody>
</table>

- **Current best practice:**
  - Use **teams** to express GPU threadblock/work group parallelism
  - Use **parallel for simd** to express GPU thread/work item parallelism

- **Future direction:**
  - Improve CCE support for **parallel** and **simd** in accelerator regions
  - Upstream Clang is expanding support for **simd** in accelerator regions

*Long-term goal: let users express parallelism with any construct they think makes sense, and CCE will map to available hardware parallelism*
ASYNC OFFLOAD CAPABILITIES

- OpenMP offload **nowait** constructs map to independent GPU streams
  - **depend** clauses are handled with necessary stream synchronization

- Task “detach” support introduced in CCE 11.0 (Nov 2020)

- Cross-device dependences are not yet optimized well (overly conservative synchronization)

- Multi-threaded use of GPU is optimized as of CCE 13.0 (Nov 2021) – relaxed locking strategy
THANK YOU

Deepak Eachempati
deepek.eachempati@hpe.com
Intel® Compilers Update

Xinmin Tian
Intel Corporation
ECP OpenMP Community RoadMap BoF’2022
Notices & Disclaimers

DISTRIBUTION STATEMENT: None Required

Disclosure Notice: This presentation is bound by Non-Disclosure Agreements between Intel Corporation and the Department of Energy, and Argonne National Lab, and is therefore for Internal Use Only and not for distribution outside these organizations or publication outside the above referenced Subcontracts.

Intel Corp Proprietary Information: This document contains trade secrets and/or proprietary information of Intel Corporation and Intel Federal LLC (“Intel”) and is exempt from disclosure under the Freedom of Information Act. The information contained herein shall not be duplicated, used or disclosed outside the U.S. Department of Energy, UChicago Argonne LLC except as permitted by the contract previously referenced. The data subject to this restriction are contained in all sheets of this document.

USG Disclaimer: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Export Control: This document contains information that is subject to export control under the Export Administration Regulations. However the contents remain within the applicable ECCN’s provided in the most recent Multi Party for Intel Restricted Secret Information that is applicable to the CORAL Aurora Program.

Intel Disclaimer: Intel makes available this document and the information contained herein in furtherance of the CORAL Aurora Program. None of the information contained herein is, or should be construed, as advice. While Intel makes every effort to present accurate and reliable information, Intel does not guarantee the accuracy, completeness, efficacy, or timeliness of such information. Use of such information is voluntary, and reliance on it should only be undertaken after an independent review by qualified experts. Access to this document is with the understanding that Intel is not engaged in rendering advice or other professional services. Information in this document may be changed or updated without notice by Intel.

This document contains copyright information, the terms of which must be observed and followed. Reference herein to any specific commercial product, process or service does not constitute or imply endorsement, recommendation, or favoring by Intel or the US Government. Intel makes no representations whatsoever about this document or the information contained herein. IN NO EVENT SHALL INTEL BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES FOR ANY USE OF THIS DOCUMENT, INCLUDING, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, OR OTHERWISE, EVEN IF INTEL IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright © 2020, Intel Corporation. All rights reserved – unpublished work.
Agenda

- OpenMP Standards Support in Intel® Compilers
- Unified Shared Memory (USM) allocators
- OpenMP and SYCL/DPC++ Composability
- Async Offloading
- OpenMP SIMD
- Fortran (IFX) Status Update
OpenMP Standards Support in Intel compilers

- OpenMP 4.0/4.5 offloading will not be supported in ICC and IFORT for GPUs and will not be conformant to OpenMP 5.0/5.1.

- OpenMP 5.0/5.1/5.2 features are planned to be implemented in ICX and IFX by continuously leveraging Clang/LLVM community work.

<table>
<thead>
<tr>
<th>Intel Compiler</th>
<th>Driver</th>
<th>Target*</th>
<th>OpenMP Support</th>
<th>OpenMP Offload Support</th>
<th>Included in oneAPI Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® C++ Compiler Classic (ICC)</td>
<td>icc</td>
<td>CPU</td>
<td>Yes</td>
<td>No</td>
<td>HPC, IoT</td>
</tr>
<tr>
<td>Intel® oneAPI DPC++/C++ Compiler (ICX)</td>
<td>dpcpp</td>
<td>CPU, GPU, FPGA</td>
<td>Yes</td>
<td>Yes</td>
<td>Base</td>
</tr>
<tr>
<td></td>
<td>icx</td>
<td>CPU, GPU</td>
<td>Yes</td>
<td>Yes</td>
<td>Base</td>
</tr>
<tr>
<td>Intel® Fortran Compiler Classic (IFORT)</td>
<td>ifort</td>
<td>CPU</td>
<td>Yes</td>
<td>No</td>
<td>HPC</td>
</tr>
<tr>
<td>Intel® Fortran Compiler (Beta) (IFX)</td>
<td>ifx</td>
<td>CPU, GPU</td>
<td>Yes</td>
<td>Yes</td>
<td>HPC</td>
</tr>
</tbody>
</table>
Use OpenMP Memory Allocator for USM

program reduction_example
  use omp_lib
  integer :: n = 32768
  integer :: m = 2048

  integer :: i, j
  double precision :: val = 0.0
  double precision :: val_ver = 0.0
  double precision, allocatable :: a_h(:), b_h(:), c_h(:)
  real*8 a_x(32768), b_x(32768), c_x(32768)

  !$omp allocate allocator(omp_target_shared_mem_alloc)
  allocate(a_h(n))

  !$omp allocate allocator(omp_target_shared_mem_alloc)
  allocate(b_h(n))

  !$omp allocate allocator(omp_target_shared_mem_alloc)
  allocate(c_h(n))

  do i = 1, n
    a_h(i) = dble(i);
    b_h(i) = 0.2;
    c_h(i) = 0.3;
    a_x(i) = dble(i);
    b_x(i) = 0.2;
    c_x(i) = 0.3;
  end do

  ! Reduction on val is done in C implementation below
  call red_02(a_h, b_h, c_h, n, m, val)

val_ver = 0.0

!$omp target data map(tofrom: val_ver) map(to: a_x, b_x, c_x)
!$omp target teams distribute parallel do reduction(+: val_ver)
  & collapse(2)
    do i = 1, n
      do j = 1, m
        val_ver = val_ver + a_x(i) * b_x(i) * c_x(i);
      end do
    end do
  end do
!$omp end target teams distribute parallel do
!$omp end target data

if(abs(int(val*1.0d+15) - int(val_ver*1.0d+15)) .lt. 1.0) then
  write(*,*) "Congratulations!! Correct Results"
  write(*,*) "val[", val, "]
  & val,"; val_ver[", val_ver, "]"
else
  write(*,*) "Incorrect Result", "val[",
  & val,"; val_ver[", val_ver, "]"
endif

deallocate(a_h)
deallocate(b_h)
deallocate(c_h)
end program
OpenMP and SYCL/DPC++ Composability

- Several codes might need a smooth transition to/from OMP offload and DPC++
- Question coming from many customers
- A very simple test just to understand how compilation and execution works
Offloading 2 Different Kernels

- Simple main.cpp
- We are creating 2 OMP tasks each one sending a kernel
- The first kernel is OMP
- *The second kernel is DPC++*

```c
#pragma omp parallel sections shared(size)
{
    //OMP target section
    #pragma omp section
    {
        run_omp(Aomp, Bomp, Comp, size);
    }
    //DPCPP section
    #pragma omp section
    {
        run_dpcpp(Adpcpp, Bdpcpp, Cdpcpp, size);
    }
}
```
Asynchronous Offloading

```c
#include <stdio.h>
#include <omp.h>

int main() {
  int ret = 0;
  #pragma omp target map(ret) nowait
  {
    for (int i = 0; i < 1000; i++)
      for (int j = 0; j < 1000; j++)
        ret--;
    if (ret <= 0)
      ret = 1;
    printf("Device ret = %d\n", ret);
  }
  printf("Before explicit offload sync: ret = %d\n", ret);
  #pragma omp taskwait
  printf("After explicit offload sync: ret = %d\n", ret);
  return 0;
}
xitian@scsel-cfl-12:$ icpx -fiopenmp -fopenmp-targets=spir64 target_nowait.cpp -o run.x
xitian@scsel-cfl-12:$ ./run.x

Before explicit offload sync: ret = 0
Device ret = 1
After explicit offload sync: ret = 1
```

Added compiler support of enabling free agent helper thread running concurrently with the initial thread

Leveraged community free agent helper thread support
OpenMP SIMD for GPUs

```c
#pragma omp target enter data map( alloc:a[0:TOTAL_SIZE] )
#pragma omp target enter data map( alloc:b[0:TOTAL_SIZE] )
#pragma omp target enter data map( alloc:c[0:TOTAL_SIZE] )
#pragma omp target update to(a[0:TOTAL_SIZE])
#pragma omp target update to(b[0:TOTAL_SIZE])

const int no_max_rep = 400;
double time = omp_get_wtime();
for ( int irep = 0; irep < no_max_rep; ++irep ) {
    #pragma omp target teams distribute parallel for
    for ( int isimd = 0; isimd < TOTAL_SIZE; isimd += SIMD_SIZE<<2 ) {
        #pragma omp simd simdlen(32)
        for ( int ilane = 0; ilane < SIMD_SIZE<<2; ++ilane ) {
            const int index = isimd + ilane;
            c[index] = a[index] + b[index];
        }
    }
}

time = omp_get_wtime() - time;
time = time/no_max_rep;
... ... ...
#pragma omp target exit data map( release:a[0:TOTAL_SIZE] )
#pragma omp target exit data map( release:b[0:TOTAL_SIZE] )
#pragma omp target exit data map( release:c[0:TOTAL_SIZE] )
```
Fortran (IFX) Compiler Status Update

- F2003 complete (PDT’s now implemented)
- F2008 complete except coarrays (F2008 in Q3, F2018 in Q4)
- F2018 development (IEEE compares, DIM opt arg in intrinsics)
- Fortran extension VAX structs/unions implemented
- Complete IFX OpenMP DECLARE MAPPER and TILE
- Continue coarrays work for F08 feature complete
- Fortran quality and hardening, continuous perf improvements
- Auto-offload of Fortran DO CONCURRENT
- Fortran development: F18 C-interop, DLLIMPORT/EXPORT, /Qinit, /check:bounds
Call to Action & Resources

Call to Action – Get the Intel oneAPI Base, HPC & IoT Toolkit today!

• Current Customers - Log into Intel Registration Center - registrationcenter.intel.com

Resources

• oneAPI Initiative – oneAPI.com
OpenMP in NVIDIA’s HPC Compilers

Jeff Hammond and Jeff Larkin

9 May 2022
std::transform(par, x, x+n, y, y, [=](float x, float y) { return y + a*x; });

do concurrent (i = 1:n)
y(i) = y(i) + a*x(i)
enddo

import legate.numpy as np ...
def saxpy(a, x, y):
y[:] += a*x

#pragma acc data copy(x,y) {
...
#pragma acc parallel loop
for (i=0; i<n; i++) {
  y[i] += a * x[i];
}
...}
#pragma acc omp target data map(x,y) {
...
#pragma acc omp target teams loop
for (i=0; i<n; i++) {
  y[i] += a * x[i];
}
...}
#pragma omp target data map(x,y)
void saxpy(int n, float a, float *x, float *y) {
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
... cudaMemcpy(d_x, x, ...);
cudaMemcpy(d_y, y, ...);
saxpy<<<(N+255)/256,256>>>(...);
cudaMemcpy(y, d_y, ...);
}
NVIDIA HPC COMPILER

Using OpenMP

• OpenMP
  - -mp → Enable OpenMP targeting Multicore
  - -mp=gpu → Enable OpenMP targeting GPU and Multicore

• GPU Options
  - -gpu=ccXX → Set GPU target, specialize for one generation, or many

• Compiler Diagnostics
  - -Minfo=mp → Compiler diagnostics for OpenMP

• Environment variable for NOTIFY
  - export NVCOMPILER_ACC_NOTIFY = 1|2|3
OPENMP MODEL
OpenMP Execution Mapping to NVIDIA GPUs and Multicore

`omp target` → Starts Offload

`omp teams` → [GPU] CUDA Thread Blocks in grid
       → [CPU] `num_teams(1)`

`omp parallel` → [GPU] CUDA threads within thread block
       → [CPU] CPU threads

`omp simd` → [GPU] `simdlen(1)` i.e. ignored
       → [CPU] Hint for vector instructions
## WHY THE SUBSET?

### SCALABILITY-CHALLENGED OPENMP FEATURES

<table>
<thead>
<tr>
<th>Master</th>
<th>Single</th>
<th>Critical</th>
<th>Ordered</th>
<th>Sections</th>
<th>Barrier</th>
<th>SIMD (SAFELEN)</th>
<th>Task</th>
<th>Taskloop</th>
<th>Taskgroup</th>
<th>Depend</th>
<th>Taskwait</th>
<th>Cancel</th>
<th>Procbind</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Directives

- `omp_init_lock()`
- `omp_init_lock_with_hint()`
- `omp_set_lock()`
- `omp_test_lock()`
- `omp_unset_lock()`
- `omp_destroy_lock()`
- `omp_init_nest_lock()`
- `omp_init_nest_lock_with_hint()`
- `omp_set_nest_lock()`
- `omp_test_nest_lock()`
- `omp_unset_nest_lock()`
- `omp_destroy_nest_lock()`

### Locks

- `OMP_SCHEDULE`
- `OMP_NUM_THREADS`
- `OMP_DYNAMIC`
- `OMP_PROC_BIND`
- `OMP_PLACES`
- `OMP_NESTED`
- `OMP_WAIT_POLICY`
- `OMP_MAX_ACTIVE_LEVELS`
- `OMP_THREAD_LIMIT`
- `OMP_CANCELLATION`
- `OMP_DISPLAY_ENV`
- `OMP_MAX_TASK_PRIORITY`

---

START OFFLOADING ‘OMP LOOP’
Three Ways

1. `omp target teams loop`
   - Recommended way
   - You can use num_teams and thread_limit clauses

2. `omp target loop`
   - Fully automatic
   - You cannot use num_teams / thread_limit

3. `omp target parallel loop`
   - Uses only threads, and doesn’t use teams
   - Might be useful for light kernels
CASE STUDY: MATRIX TRANSPOSE
OpenMP prescriptive parallelism

```fortran
!$omp target teams distribute parallel do simd collapse(2)
do j=1,order
  do i=1,order
    B(i,j) = B(i,j) + A(j,i) ! Contiguous RW of B
  enddo
enddo

!$omp target teams distribute parallel do simd collapse(2)
do j=1,order
  do i=1,order
    B(j,i) = B(j,i) + A(i,j) ! Contiguous R of A
  enddo
enddo
```

51% peak
12% peak
CASE STUDY: MATRIX TRANSPOSE

OpenMP descriptive parallelism

```c
!$omp target teams loop collapse(2)
do j=1,order
  do i=1,order
    B(i,j) = B(i,j) + A(j,i) ! Contiguous RW of B
  enddo
enddo

!$omp target teams loop collapse(2)
do j=1,order
  do i=1,order
    B(j,i) = B(j,i) + A(i,j) ! Contiguous R of A
  enddo
enddo
```

“teams loop” = more performance, less typing

57% peak

13% peak
CASE STUDY: MATRIX TRANSPOSE

Descriptive parallelism plus tiling

```c
!$omp target teams loop collapse(2)
do jt=1,order,tile_size
do it=1,order,tile_size
  !$omp loop collapse(2)
do j=jt,min(order,jt+32-1)
do i=it,min(order,it+32-1)
  B(i,j) = B(i,j) + A(j,i) ! Contiguous RW of B
endo
endo
endo
endo
endo

!$acc parallel loop tile(32,32)
do j=1,order
do i=1,order
  B(i,j) = B(i,j) + A(j,i) ! Contiguous RW of B
endo
endo
```

72% peak

76% peak
CASE STUDY: AXPY
Memory management options

allocate(X,Y,Z)

<table>
<thead>
<tr>
<th>MAP_ALLOC</th>
<th>MANAGED</th>
<th>allocate</th>
<th>data in</th>
<th>init</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.000015</td>
<td>2.367367</td>
<td>0.014560</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.348643</td>
<td>0.017112</td>
<td>3.049976</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.361456</td>
<td>0.018193</td>
<td>3.055903</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.000013</td>
<td>0.388539</td>
<td>0.020914</td>
</tr>
</tbody>
</table>

#if MAP_ALLOC
!$omp target data map(alloc:X,Y,Z)
#else
!$omp target data map(tofrom: Z) &
!$omp&      map(to: X,Y)
#endif

! init
do i=1,length
   X(i) = i-1
   Y(i) = i-1
   Z(i) = 0
enddo
BEST PRACTICES FOR OPENMP ON GPUS

Use the `teams` and `distribute` directive to expose all available parallelism.

Use the `loop` directive when the mapping to hardware isn’t obvious.

Aggressively `collapse` loops to increase available parallelism.

Use the `target data` directive and `map` clauses to reduce data movement between CPU and GPU.

...or just skip the `target data` directive and use managed memory.

Use OpenMP tasks to go asynchronous and better utilize the whole system.

Use host fallback (`if` clause) to generate host and device code.

Use accelerated libraries whenever possible.

*Less is more with the NVIDIA compiler. Being pedantic can reduce performance.*
PANEL DISCUSSION

Moderator: Kalyan Kumaran (ANL)
Panelists:

• Johannes Doerfert (LLVM, ANL)
• Carlo Bertolli (AMD)
• Tobias Burnus (GNU, Siemens)
• Deepak Eachempati (HPE)
• Xinmin Tian (Intel)
• Jeff Hammond (NVIDIA)
ACKNOWLEDGEMENT FOR ECP-FUNDED RESEARCH

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.
THANKS!
BACK-UP SLIDES
( FEATURE SUPPORT TABLE IN 2021)
MULTIPLE COMPilers WILL SUPPORT A COMMON SET OF OPENMP DIRECTIVES ON GPUs (NON-EXHAUSTIVE LIST) (1/2)

<table>
<thead>
<tr>
<th>Levels of parallelism</th>
<th>LLVM/Clang</th>
<th>AMD</th>
<th>HPE/Cray</th>
<th>IBM</th>
<th>Intel</th>
<th>NVIDIA</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 (teams + parallel), eventually SIMD</td>
<td>2 (teams, parallel)</td>
<td>2 (teams, parallel or simd)</td>
<td>2 (teams, parallel)</td>
<td>3 (teams, parallel, simd)</td>
<td>2 (teams, parallel)</td>
<td>3 (teams, parallel, simd)</td>
</tr>
</tbody>
</table>

**OpenMP directive**

<table>
<thead>
<tr>
<th>OpenMP directive</th>
<th>LLVM/Clang</th>
<th>AMD</th>
<th>HPE/Cray</th>
<th>IBM</th>
<th>Intel</th>
<th>NVIDIA</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>declare target</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>map</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>target data</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>target enter/exit data</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>target update</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>teams</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>distribute</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>parallel</td>
<td>✓</td>
<td>✓</td>
<td>✓ (may be inactive)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>for/do</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>reduction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>simd</td>
<td>✓ (used for optimization, not for mapping)</td>
<td>✓ (on host)</td>
<td>✓</td>
<td>✓ (ignored)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>atomic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>critical</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>sections</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>master</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>single</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>barrier</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>declare variant</td>
<td>✓</td>
<td>✓</td>
<td>✓ (C/C++ partial)</td>
<td>X</td>
<td>✓</td>
<td>✓ (planned)</td>
<td>X (OG11)</td>
</tr>
</tbody>
</table>

As of 4/15/2021
## MULTIPLE COMPILERS WILL SUPPORT A COMMON SET OF OPENMP DIRECTIVES ON GPUS (NON-EXHAUSTIVE LIST) (2/2)

<table>
<thead>
<tr>
<th>Feature</th>
<th>LLVM/Clang</th>
<th>AMD</th>
<th>HPE/Cray</th>
<th>IBM</th>
<th>Intel</th>
<th>NVIDIA</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop directive</td>
<td>eventually</td>
<td>X</td>
<td>✓ (Fortran only)</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X (OG12)</td>
</tr>
<tr>
<td>collapse of a perfectly nested loop</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>collapse of an imperfectly nested loop</td>
<td>✓</td>
<td>✓ (c/c++)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X (OG12)</td>
</tr>
<tr>
<td>collapse of a non-rectangular nested loop</td>
<td>✓</td>
<td>✓ (c/c++)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>array reduction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>requires unified_shared_memory</td>
<td>✓</td>
<td>X</td>
<td>✓ (WIP CCE 13)</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X (planned)</td>
</tr>
<tr>
<td>requires dynamic_allocators</td>
<td>eventually</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X (OG11)</td>
</tr>
<tr>
<td>declare reduction</td>
<td>eventually</td>
<td>✓</td>
<td>✓ (C/C++ only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>declare mapper</td>
<td>✓</td>
<td>X</td>
<td>X (WIP CCE 13)</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X (OG12)</td>
</tr>
<tr>
<td>metadirective</td>
<td>LLVM 13</td>
<td>X</td>
<td>✓ (Fortran only)</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X (planned)</td>
</tr>
<tr>
<td>“target nowait” supporting asynchronous execution</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>“target depend” supporting fine-grained dependencies</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>“target device” supporting multiple non-host devices per process</td>
<td>✓</td>
<td>X</td>
<td>X (WIP CCE 13)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>use_device_addr</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>detachable tasks: &quot;detach&quot; clause and &quot;omp_fulfill_event&quot; runtime routine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Memory management APIs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allocate directive for allocating variables in managed memory via allocator</td>
<td>✓</td>
<td>X</td>
<td>✓ (extension)</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X (OG11)</td>
</tr>
<tr>
<td>allocate clause for allocating privatized variables in managed memory via allocator</td>
<td>✓</td>
<td>X</td>
<td>✓ (extension)</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X (OG11)</td>
</tr>
<tr>
<td>APIs for allocating/freeing memory via allocator</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>APIs for defining new allocators with custom traits (e.g. pinned memory)</td>
<td>✓ (not fully implemented)</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>(OG11)</td>
</tr>
<tr>
<td>Interop objects and APIs</td>
<td>✓</td>
<td>X</td>
<td>✓ (planned CCE 13)</td>
<td>X</td>
<td>X</td>
<td>X (WIP)</td>
<td>X (OG12)</td>
</tr>
</tbody>
</table>

As of 4/15/2021