2022 ECP Community BOF Days

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

EXASCALE COMPUTING PROJECT

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

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

NNSA National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

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

	LLVM/Clang	AMD	HPE/Cray	Intel	NVIDIA	GNU (GCC 12)
Levels of parallelism	2 now, 3 under development	2 (teams, parallel)	2 (teams, parallel or simd)	3 (teams, parallel, simd)	2 (teams, parallel)	3 (teams, parallel, simd)
OpenMP directive						
target	✓	✓	✓	✓	✓	✓
declare target	✓	✓	✓	✓	✓	✓
map	✓	✓	✓	✓	✓	✓ (OMP 5.0)
target data	✓	✓	✓	✓	✓	✓
target enter/exit data	✓	✓	✓	✓	✓	✓
target update	✓	✓	✓	✓	✓	✓
teams	✓	✓	✓	✓	✓	✓
distribute	✓	✓	✓	✓	✓	✓
parallel	✓	✓	✓ (may be inactive)	✓	✓	✓
for/do	✓	✓	✓	✓	✓	✓
reduction	✓	✓	✓	✓	✓	✓
simd	✓, GPU under development	✓ (on host)	✓	✓	✓ (ignored)	✓
atomic	✓	✓	✓	✓	✓	✓ (OMP 5.1 ext)
critical	(✓)	✓	✓	✓	✗	✓
sections	✓	✓	✓	✓	✗	✓
master	✓	✓	✓	✓	✓	✓
single	(✓)	✓	✓	✓	✓	✓
barrier	✓	✓	✓	✓	✓	✓
loop directive	eventually	✓(recognize syntax)	✓ (Fortran only)	✓	✓	✓
collapse of a perfectly nested loop	✓	✓	✓	✓	✓	✓
collapse of an imperfectly nested loop	✓	✓ (c/c++)	✓	✗	✗	✗ (GCC13/OG12)
collapse of a non-rectangular nested loop	✓	✓ (c/c++)	✓	✗	✗	C/C++: ✓ / F90: ✗ (F90: GCC13/OG11)
loop transformation with tile	✓	✓	✓ (C/C++ only)	✓	✗	✗ (GCC13/OG12)
loop transformation with unroll	✓	✓	✓ (C/C++ only)	✓	✗	✗ (GCC13/OG12)
array reduction	✓	✓	✓	✓	✓	✓/ F90 array sections: ✗
scan	eventually	✓(recognize syntax)	✗	✗ (WIP)	✗	✓

MULTIPLE COMPILERS WILL SUPPORT A COMMON SET OF OPENMP DIRECTIVES ON GPUS (NON-EXHAUSTIVE LIST) (2/2)

As of 5/11/2022

✓	: yes
(✓)	: yes with caveats
✗	: no

	LLVM/Clang	AMD	HPE/Cray	Intel	NVIDIA	GNU (GCC 12)
requires unified_shared_memory	✓	✓	✓ (some platforms)	✓	✗ (unnecessary)	✗ (WIP for nvptx)
requires dynamic_allocators	(✓)	✗	✗	✓	✗	(✗) (GCC13/OG12)
declare reduction	✓	✓	✓ (C/C++ only)	✓	✗	✓
declare mapper	✓	✓	✓ (C/C++ only)	✓	✗	✗ (GCC13/OG12)
metadirective	✓	✓(c/c++)	(✓) (limited, OMP 5.0 only)	✗ (WIP)	partial	✗ (GCC13/OG11)
declare variant	✓	✓	(✓) (limited, OMP 5.0 only)	✓	partial	✓
“target nowait” supporting asynchronous execution	✓	✓	✓	✓	✓	(✓) (sync w/ in_reduction)
“target depend” supporting fine-grained dependencies	✓	✓	✓	✓	✗	✓
“target device” supporting multiple non-host devices per process	✓	✓	✗	✓	✓	✓
use_device_addr	✓	✓	✓	✓	✓	✓ (+ has_…)
detachable tasks: ”detach” clause and “omp_fulfill_event” runtime routine	(✓)	✓	✓	✗	✗	✓
Memory management APIs						
allocate directive for allocating variables in managed memory via allocator	(✓)	✓	✓ (extension)	✓	✗	✗ (GCC13/OG11)
allocate clause for allocating privatized variables in managed memory via allocator	(✓)	✗	✓ (extension)	✓	✗	(✓)
APIs for allocating/freeing memory via allocator	✓	✓ (limited support on device with predefined allocators)	✓	✓	✗	✓
APIs for defining new allocators with custom traits (e.g. pinned memory)	✓	✓ (only pinned)	✓	✓	✗	✓
Interop objects/directive and APIs	✓	✗	✗	✓	✗	✗
C++ attribute syntax	eventually	✓	✗	✗ (WIP)	✗	✓
Orphaned parallel regions (any limitations? e.g. serialized)	No limitations	✓	✓	✓	✓, parallel but slow	✓
Creating C++ objects containing virtual functions inside target regions (GPU)	✓	✗	✓	✗ (WIP)	✗	(✓) (if vtable+ methods emit.)
Mapping C++ objects containing virtual functions from host to the GPU	eventually	✗	✗	✗ (WIP)	✗	(✓) (no virt. calls)
printf/print support in a target region (GPU)	✓	✓	✓	✓	✓	✓/ F90 on nvptx: ✗ (GCC13/OG12)
Call CUDA/SYCL/HIP kernels in an OpenMP target region	✓	✗	✗	✓	CUDA works, but it depends on the details	(✓) (may work)

OPENMP RESOURCES

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

OpenMP Validation and Verification
- https://crpl.cis.udel.edu/ompvvsollve/

OpenMP YouTube Channel
- https://www.youtube.com/user/OpenMPARB/

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

Topics	Minutes	Presenter or Moderator
Introduction	3	JaeHyuk Kwack/ Colleen Bertoni
Roadmap Presentations		
LLVM	7	Johannes Doerfert
AMD	7	Carlo Bertolli
GNU	7	Tobias Burnus
HPE	7	Deepak Eachempati
Intel	7	Xinmin Tian
NVIDIA	7	Jeff Hammond
Panel discussion - Preselected questions - Questions/comments from audience (alternating)	45	Kalyan Kumaran and other panelists
Total time	90	

ROADMAP PRESENTATIONS

LLVM/OpenMP in HPC

A Brief Overview

Screenshot me :)

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

Screenshot me :)

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]

Screenshot me :)

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

Shared memory usage for scratchpads

Automatic guarding and synchronization

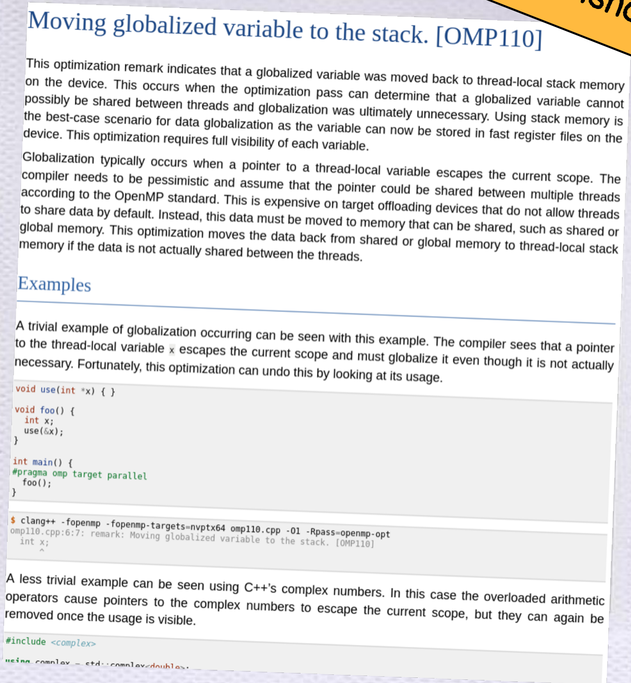
```
#pragma omp target teams
#pragma omp parallel
{
  double team_local_memory[M];
  #pragma omp allocate(team_local_memory) \
              allocator(omp_cgroup_mem_alloc)
  #pragma omp masked
  team_main_thread_only();
  #pragma omp barrier
  every_thread(team_local_memory);
}
```

OpenMP-Optimization Remarks & Assumptions

```
example.cpp:41:24: remark: Found thread data sharing on the
     GPU. Expect degraded performance due to data
     globalization. [OMP112] [-Rpass-missed=openmp-opt]
double device_function(float Arg) {
                                               ^
example.cpp:42:3: remark: Moving globalized variable to the
     stack. [OMP110] [-Rpass=openmp-opt]
double Lcl;
        ^
```

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`

Screenshot me :)



Moving globalized variable to the stack. [OMP110]

This optimization remark indicates that a globalized variable was moved back to thread-local stack memory on the device. This occurs when the optimization pass can determine that a globalized variable cannot possibly be shared between threads and globalization was ultimately unnecessary. Using stack memory is the best-case scenario for data globalization as the variable can now be stored in fast register files on the device. This optimization requires full visibility of each variable.

Globalization typically occurs when a pointer to a thread-local variable escapes the current scope. The compiler needs to be pessimistic and assume that the pointer could be shared between multiple threads according to the OpenMP standard. This is expensive on target offloading devices that do not allow threads to share data by default. Instead, this data must be moved to memory that can be shared, such as shared or global memory. This optimization moves the data back from shared or global memory to thread-local stack memory if the data is not actually shared between the threads.

Examples

A trivial example of globalization occurring can be seen with this example. The compiler sees that a pointer to the thread-local variable `x` escapes the current scope and must globalize it even though it is not actually necessary. Fortunately, this optimization can undo this by looking at its usage.

```
void use(int *x) { }

void foo() {
  int x;
  use(&x);
}

int main() {
#pragma omp target parallel
  foo();
}
```

```
$ clang++ -fopenmp -fopenmp-targets=nvptx64 omp110.cpp -O1 -Rpass=openmp-opt
omp110.cpp:6:7: remark: Moving globalized variable to the stack. [OMP110]
  int x;
      ^
```

A less trivial example can be seen using C++'s complex numbers. In this case the overloaded arithmetic operators cause pointers to the complex numbers to escape the current scope, but they can again be removed once the usage is visible.

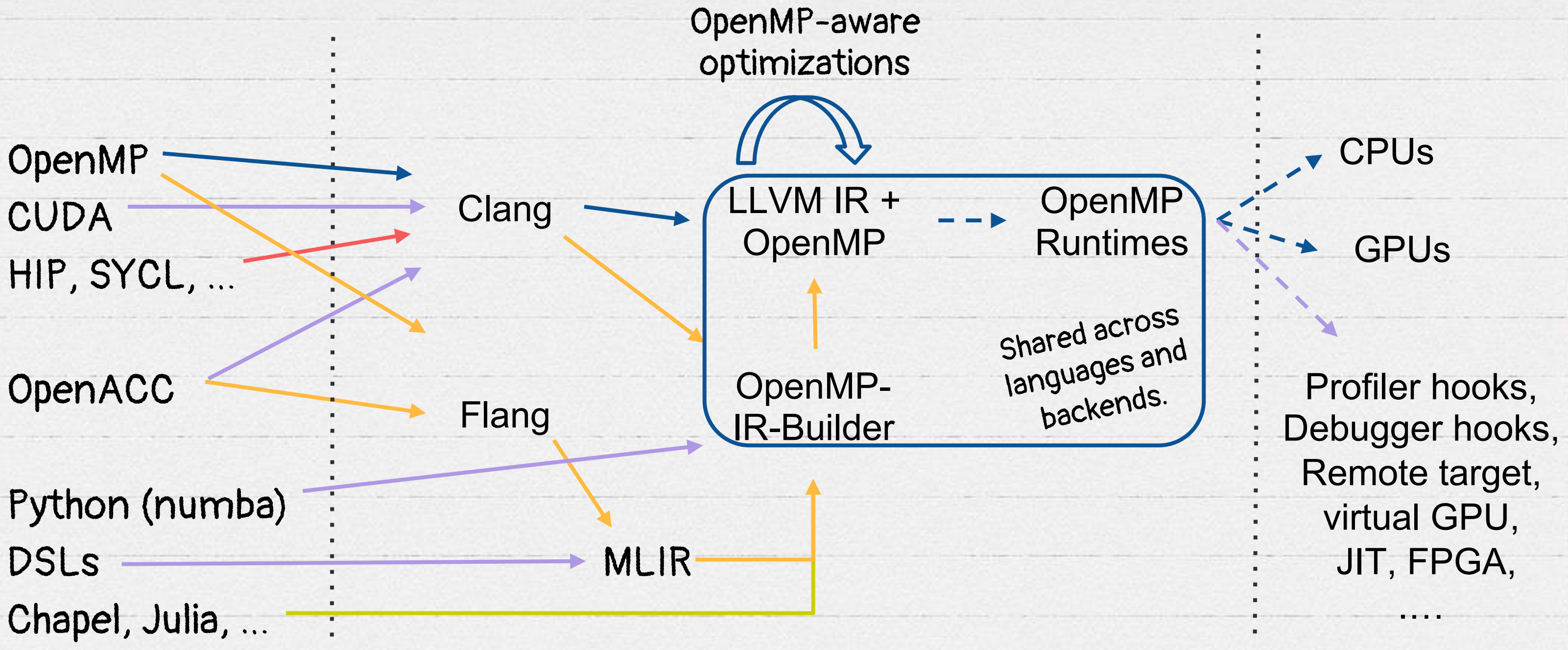
```
#include <complex>

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

Visit openmp.llvm.org for more!

https://openmp.llvm.org/remarks/OptimizationRemarks.html

Diverse Inputs/Outputs - One Pipeline

Screenshot me :)


OpenMP-aware optimizations
OpenMP
CUDA
HIP, SYCL, ...
OpenACC
Python (numba)
DSLs
Chapel, Julia, ...
Clang
Flang
MLIR
LLVM IR + OpenMP
OpenMP Runtimes
OpenMP-IR-Builder
Shared across languages and backends.
CPUs
GPUs
Profiler hooks, Debugger hooks, Remote target, virtual GPU, JIT, FPGA, ...

Production | Under Development | Research Prototype | Not Started | Proposed

Screenshot me :)

OpenMP offload Recommendations

- 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` and `-fopenmp-assume-no-thread-state`
- Use device-side LTO `-foffload-lto`

OpenMP® Support of ROCm™ v5.0 @ OpenMP RoadMap BoF

@2022 ECP Community BoF Day, 11th May 2022

- Carlo Bertolli & Saiyedul Islam

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

Option	Description
h	Print the help message.
a	Print values for all devices. Don't stop at first device found.
m	Print device code name (often found in pci.ids file).
n	Print numeric pci-id.
t	Print clang offload triple to use for the offload arch.
v	Verbose = -a -m -n -t For all devices, print codename, numeric value and triple
f <filename>	Print offload requirements including offload-arch for each compiled offload image built into an application binary file.
c	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.

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

```
#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

```
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

OpenMP in GCC Status & Tips

Tobias Burnus

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)

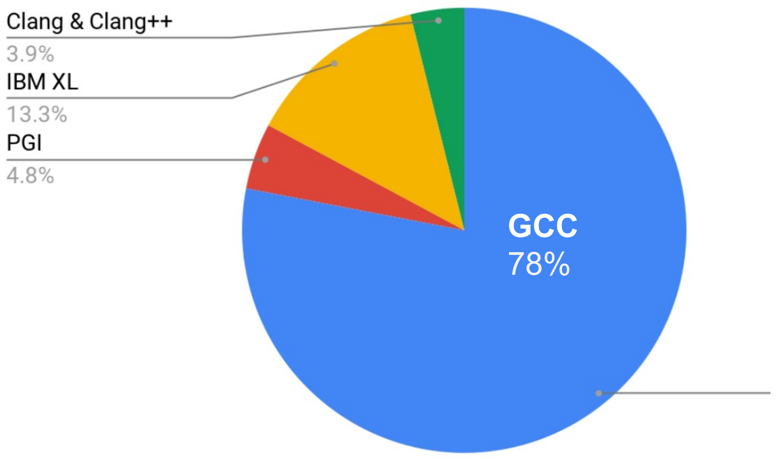
  → Linux distros use git branch directly, mainline also quite stable
  → Also avail: OG12 (= devel/omp/gcc-12) SIEMENS' public branch

https://gcc.gnu.org

On Summit (OLCF/ORNL) [by compute time]

Compiler Count By Family CY-2021

January - August, Total Count: ~3.4M



D. Bernholdt (ORNL) + T. Burnus, GCC,
https://openmpcon.org/conf2021/program-archive/

SIEMENS

OpenMP Now Supported & Implementation Status

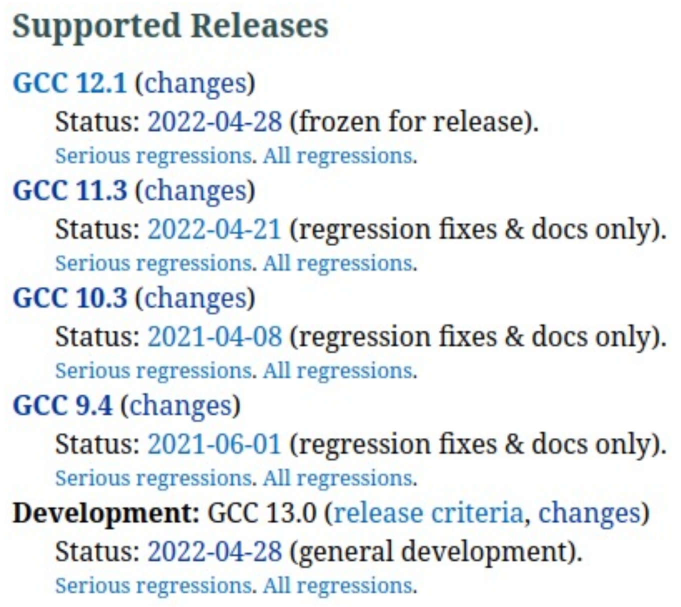
GCC 11

- 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

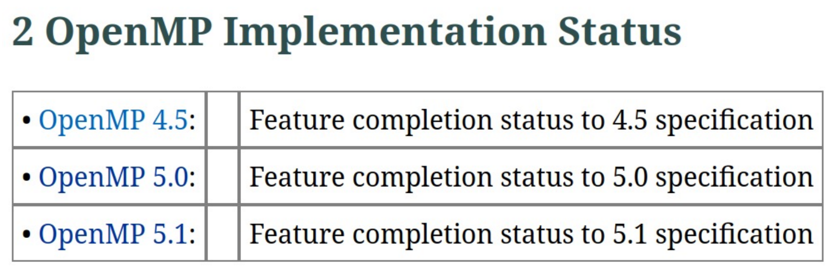
- OpenMP 5.1: C++ 11 attributes, masked/scope/error/nothing, atomic extensions, memory-allocation routines, strictly structured blocks
- OpenMP 5.0: affinity clause. Fortran: declare variant, depobj, mutexinoutset, iterator, defaultmap 5.0 ext., loop
- GCN: Debugging (ROCGDB), wavefronts per compute 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

Compiling

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

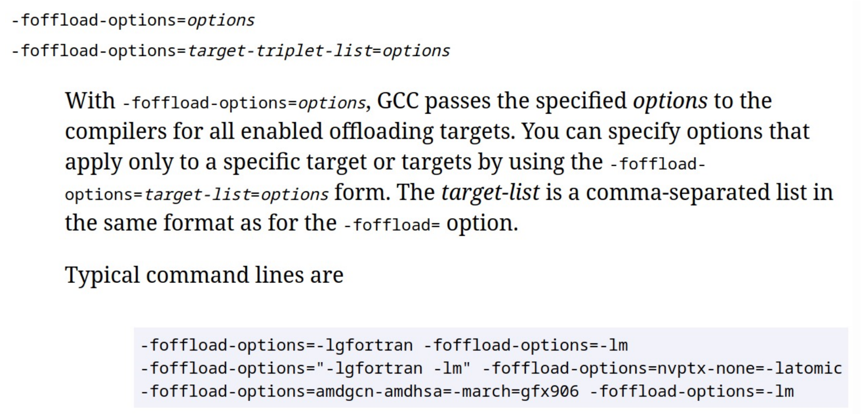
- -foffload-options=-lm   -foffload-options=nvptx-none=-latomic
  GCC <12: Use -foffload= instead (undocumented, has corner case)

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



`-foffload-options=`*`options`*

`-foffload-options=`*`target-triplet-list`*`=`*`options`*

With `-foffload-options=`*`options`*, GCC passes the specified *options* to the compilers for all enabled offloading targets. You can specify options that apply only to a specific target or targets by using the `-foffload-options=`*`target-list`*`=`*`options`* form. The *target-list* is a comma-separated list in the same format as for the `-foffload=` option.

Typical command lines are

```
-foffload-options=-lgfortran -foffload-options=-lm
-foffload-options="-lgfortran -lm" -foffload-options=nvptx-none=-latomic
-foffload-options=amdgcn-amdhsa=-march=gfx906 -foffload-options=-lm
```

(Since GCC 12) Manpage or
https://gcc.gnu.org/onlinedocs/gcc/

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

Hewlett Packard
Enterprise

HPE OPENMP COMPILER UPDATE

Deepak Eachempati
CCE OpenMP Compiler Group

May 11, 2022

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 complier
  - 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)
- deprecate 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
- atomics in simd
- simd in simd
- detachable tasks
- omp_control_tool
- OMPT
- OMPD
- 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)
- 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

NVIDIA	AMD	CCE Fortran OpenACC	CCE Fortran OpenMP	CCE C/C++ OpenMP	Clang C/C++ OpenMP
Threadblock	Work group	acc gang	omp teams	omp teams	omp teams
Warp	Wavefront	acc worker	omp simd	omp parallel or omp simd	omp parallel
Thread	Work item	acc vector			

- 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
deepak.eachempati@hpe.com

Intel® Compilers Update

Xinmin Tian

Intel Corporation

ECP OpenMP Community RoadMap BoF'2022

Notices & Disclaimers

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 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.

Intel Compiler	**Driver**	**Target***	**OpenMP Support**	**OpenMP Offload Support**	**Included in oneAPI Toolkit**
Intel® C++ Compiler Classic (ICC)	***icc***	CPU	Yes	No	HPC, IoT
Intel® oneAPI DPC++/C++ Compiler (ICX)	***dpcpp***	CPU, GPU, FPGA	Yes	Yes	Base
	icx	CPU GPU	Yes	Yes	Base
Intel® Fortran Compiler Classic (IFORT)	***ifort***	CPU	Yes	No	HPC
Intel® Fortran Compiler (Beta) (IFX)	***ifx***	CPU, GPU	Yes	Yes	HPC

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
!$omp end target teams distribute parallel do
!$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_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++*

```
#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

```
#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;
}
```

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

Leveraged community free agent helper thread support

```
xtian@scsel-cfl-12:$ icpx -fiopenmp -fopenmp-targets=spir64 target_nowait.cpp -o run.x
xtian@scsel-cfl-12:$ ./run.x

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

OpenMP SIMD for GPUs

```
#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] )
```

intel.

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
- Intel® oneAPI Base Toolkit and HPC toolkit- https://software.intel.com/content/www/us/en/develop/tools/oneapi/commercial-base-hpc.html
- Intel® oneAPI Base and IoT toolkit

  https://www.intel.com/content/www/us/en/developer/tools/oneapi/commercial-base- iot.html
- Porting Guide - https://software.intel.com/content/www/us/en/develop/articles/porting-guide-for-icc-users-to-dpcpp-or-icx.html
- ICX OpenMP features support

  https://www.intel.com/content/www/us/en/developer/articles/technical/openmp-features-and- extensions-supported-in-icx.html
- IFX OpenMP features support

  https://www.intel.com/content/www/us/en/developer/articles/technical/fortran-language-and-openmp-features-in-ifx.html

OpenMP in NVIDIA’s HPC Compilers

JEFF HAMMOND AND JEFF LARKIN

9 MAY 2022

NVIDIA Compiler and Language Support

Accelerated Standard Languages

```
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
```

Incremental Portable Optimization

```
#pragma acc data copy(x,y) {
...
#pragma acc parallel loop
for (i=0; i<n; i++) {
  y[i] += a * x[i];
}
...
}
```

```
#pragma omp target data map(x,y) {
...
#pragma omp target teams loop
for (i=0; i<n; i++) {
  y[i] += a * x[i];
}
...
}
```

Platform Specialization

```
__global__
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, ...);
```

Core	Math	Communication	Data Analytics	AI	Quantum

Acceleration Libraries

https://developer.nvidia.com/nvidia-hpc-sdk-downloads

NVIDIA HPC COMPILER

Using OpenMP

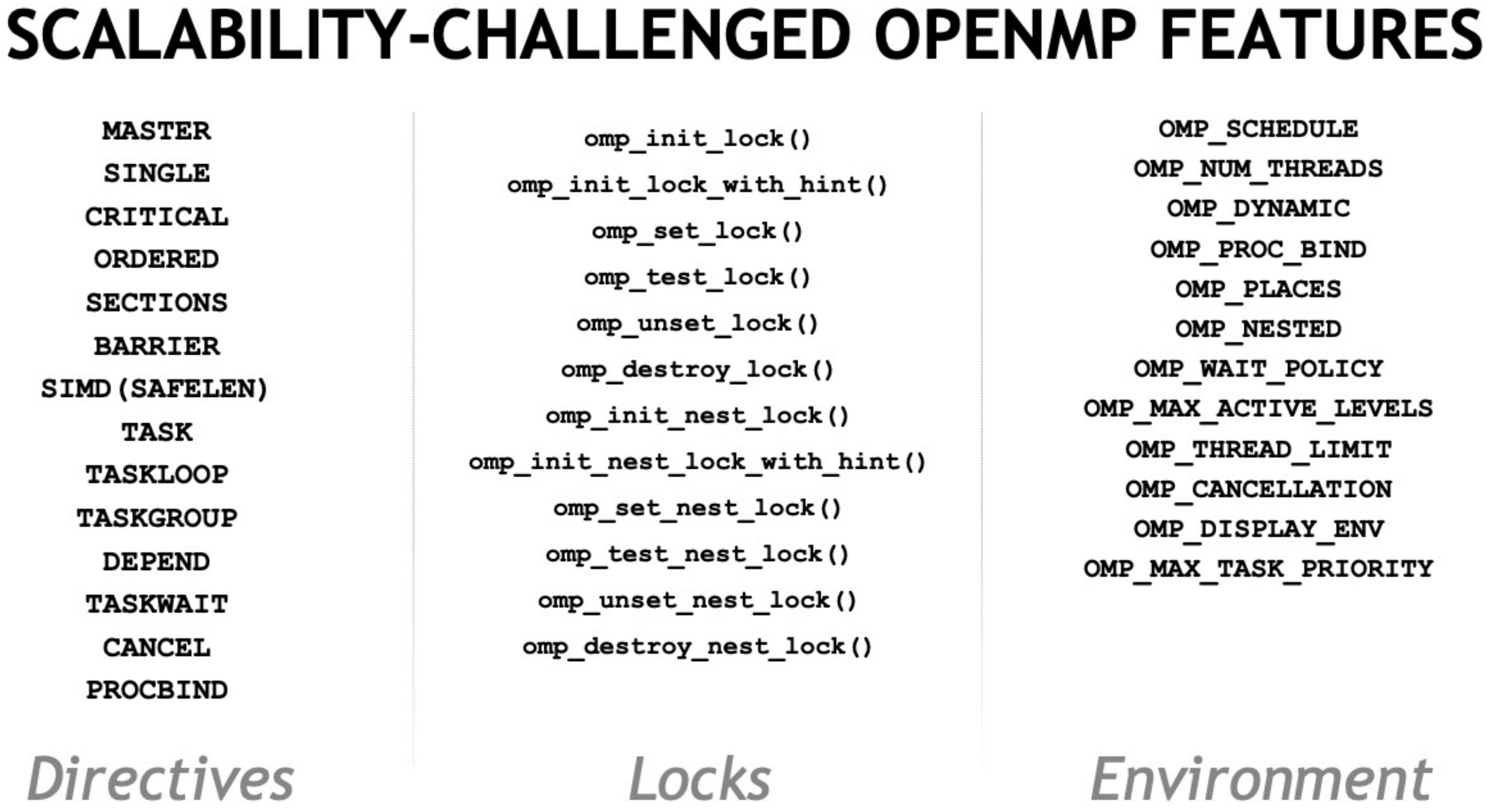
- OpenMP
  - -mp → Enable OpenMP targeting Multicore
  - -mp=gpu → Enable OpenMP targeting GPU and Multicore
- GPU Options
  - -gpu=ccXX → Set GPU target, specialize for one generation, or many
- Compiler Diagnostics
  - -Minfo=mp → Compiler diagnostics for OpenMP
- Environment variable for NOTIFY
  - 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

```
MASTER
SINGLE
CRITICAL
ORDERED
SECTIONS
BARRIER
SIMD(SAFELEN)
TASK
TASKLOOP
TASKGROUP
DEPEND
TASKWAIT
CANCEL
PROCBIND
```

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
```

Environment

https://developer.nvidia.com/gtc/2020/video/s21387

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

```
!$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
```

51% peak

```
!$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
```

12% peak

CASE STUDY: MATRIX TRANSPOSE

OpenMP descriptive parallelism

```
!$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
```

57% peak

```
!$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
```

13% peak

“teams loop” = more performance, less typing

CASE STUDY: MATRIX TRANSPOSE

Descriptive parallelism plus tiling

```
!$omp target teams loop collapse(2)
do jt=1,order,tile_size
  do it=1,order,tile_size
    !$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
      enddo
    enddo
  enddo
enddo
```

72% peak

```
!$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
  enddo
enddo
```

76% peak

CASE STUDY: AXPY

Memory management options

```
allocate(X,Y,Z)

X = 0
Y = 0
Z = 0

#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
```

MAP_ALLOC	MANAGED	allocate	data in	init
0	0	0.000015	2.367367	0.014560
1	1	0.348643	0.017112	3.049976
0	1	0.361456	0.018193	3.055903
1	0	0.000013	0.388539	0.020914

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

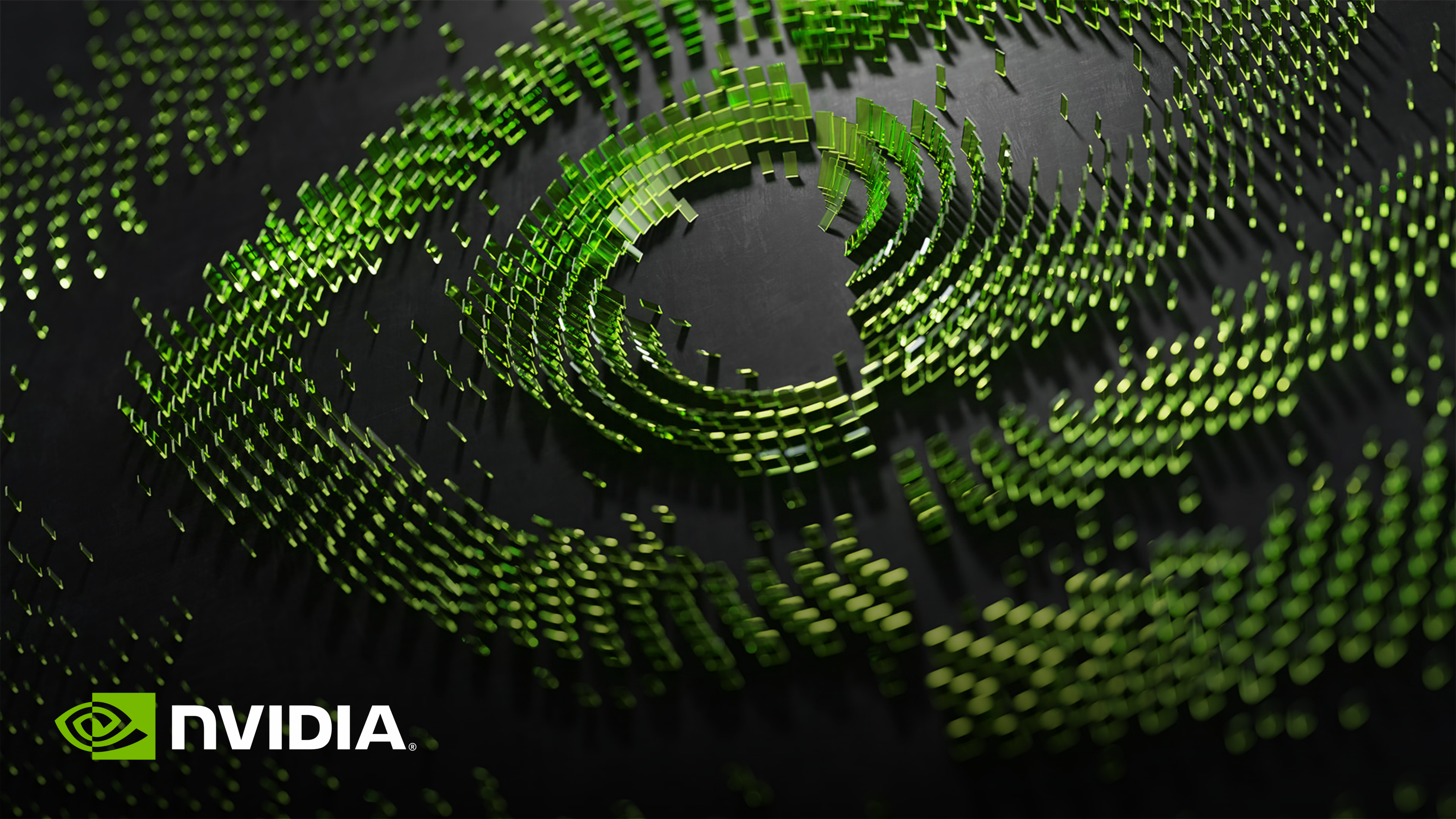
…or just skip the **`target data`** directive and use managed memory

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.


NVIDIA®

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)

As of 4/15/2021

MULTIPLE COMPILERS WILL SUPPORT A COMMON SET OF OPENMP DIRECTIVES ON GPUS (NON-EXHAUSTIVE LIST) (1/2)

	LLVM/Clang	AMD	HPE/Cray	IBM	Intel	NVIDIA	GNU
Levels of parallelism	2 (teams + parallel), eventually SIMD	2 (teams, parallel)	2 (teams, parallel or simd)	2 (teams, parallel)	3 (teams, parallel, simd)	2 (teams, parallel)	3 (teams, parallel, simd)
OpenMP directive							
target	✓	✓	✓	✓	✓	✓	✓
declare target	✓	✓	✓	✓	✓	✓	✓
map	✓	✓	✓	✓	✓	✓	✓
target data	✓	✓	✓	✓	✓	✓	✓
target enter/exit data	✓	✓	✓	✓	✓	✓	✓
target update	✓	✓	✓	✓	✓	✓	✓
teams	✓	✓	✓	✓	✓	✓	✓
distribute	✓	✓	✓	✓	✓	✓	✓
parallel	✓	✓	✓ (may be inactive)	✓	✓	✓	✓
for/do	✓	✓	✓	✓	✓	✓	✓
reduction	✓	✓	✓	✓	✓	✓	✓
simd	✓ (used for optimization, not for mapping)	✓ (on host)	✓	✓ (ignored)	✓	✓ (ignored)	✓
atomic	✓	✓	✓	✓	✓	✓	✓
critical	✓	✓	✓	✓	✓	✗	✓
sections	✓	✓	✓	✓	✓	✗	✓
master	✓	✓	✓	✓	✓	✓	✓
single	✓	✓	✓	✓	✓	✓	✓
barrier	✓	✓	✓	✓	✓	✓	✓
declare variant	✓	✓	✓ (C/C++ partial)	✗	✓	✗ (planned)	✓ (C/C++) ✗ (OG11)

As of 4/15/2021

MULTIPLE COMPILERS WILL SUPPORT A COMMON SET OF OPENMP DIRECTIVES ON GPUS (NON-EXHAUSTIVE LIST) (2/2)

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