Hewlett Packard
Enterprise

HPE COMPILER GPU OFFLOADING

Jeff Sandoval
CCE OpenMP Tech Lead

May 20, 2021

OUTLINE

- General compiler overview
- Offloading models
  - OpenMP
  - OpenACC
  - HIP
- Offloading best practices

GENERAL COMPILER OVERVIEW

HPE CRAY COMPILING ENVIRONMENT (CCE)

- A major part of the broader HPE Cray Programming Environment (CPE) supported on HPE systems
  - Compilers + Math & Communication Libraries + Debuggers + Performance Analysis Tools
- 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
- Offloading support
  - NVIDIA GPUs – XC and CS systems only
  - AMD GPUs – Shasta and Apollo systems only
  - OpenMP 4.5 and partial 5.0
  - OpenACC 2.0 – Fortran only
  - HIP – AMD GPUs only

CCE COMPILER RELEASE AND VERSIONING

- Two major releases a year (~Q2 and ~Q4)
  - CCE codebase and version based off latest Clang major release (lag by ~2 months)
- Monthly minor updates in between
  - Continue for 4 months after each major release
- Examples
  - CCE 11.0 – based on Clang 11.0 – Nov 2020
  - CCE 12.0 – based on Clang 12.0 – Jun 2021 (tentative)
  - CCE 13.0 – based on Clang 13.0 – Nov 2021 (tentative)
- *Release cadence and versioning changed in CCE 10.0*
  - *Older versions of CCE do not correspond to Clang/LLVM version numbers*

CCE COMPILER DOCUMENTATION

- Man pages of interest
  - cc, CC, ftn – CCE compiler driver documentation
  - craycc, crayCC, crayftn – CCE C, C++, and Fortran compiler documentation
  - intro_openmp – CCE OpenMP documentation
  - intro_openacc – CCE OpenACC documentation
  - intro_directives – CCE compiler directives

CCE OFFLOADING MODELS

CCE OPENMP SUPPORT

- Uses proprietary CCE OpenMP runtime libraries
  - Allows cross-language and cross-vendor interoperability
- Implements HPE-optimized code generation for OpenMP offload regions
- OpenMP 5.0 – implemented over several CCE releases
  - See release notes and intro_openmp man page for full list of supported features in each release
  - Many remaining features in progress (unified_shared_memory, declare mapper, multiple GPUs)
  - Full OpenMP 5.0 planned for CCE 13.0 (Nov 2021)
- OpenMP 5.1 – implementation planned over several CCE releases
  - High-priority features planned for CCE 13.0 (e.g., interop construct, “masked” construct)
  - Other clarifications and features will be implemented as schedule permits
  - Full OpenMP 5.1 support is planned for CCE 14.0 (May 2022)

CCE OPENMP 5.0 IMPLEMENTATION 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, tentative)

- 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++)
- uses_allocators (C/C++)
- scan (C/C++)
- lvalue list items for depend
- mutexinoutset (C/C++)
- taskloop cancellation (C/C++)

CCE 13.0 (Nov 2021, tentative)

- close modifier (Fortran)
- extend defaultmap (Fortran)
- uses_allocators (Fortran)
- concurrent maps
- taskloop cancellation (Fortran)
- scan (Fortran)
- mutexinoutset (Fortran)
- metadirectives (C/C++)
- loop construct (C/C++)
- task reduction (Fortran)
- task modifier (Fortran)
- target task reduction
- mapper
- non-rectangular loop collapse (Fortran)
- declare variant (C/C++)
- iterator in depend (Fortran)
- simd if (Fortran)
- depobj for depend (Fortran)

Refer to CCE release notes or intro_openmp man page for current implementation status

CCE OPENACC SUPPORT

- CCE only supports OpenACC for Fortran
- C/C++ support was dropped in CCE 10.0
- OpenACC 2.0 support available today
- OpenACC 3.1 support planned over next 12-18 months
- CCE OpenMP and OpenACC implementations share a common codebase
  - Significant overlap in both compiler and runtime library
  - Same performance should be achievable with either model

CCE HIP SUPPORT

- Heterogeneous-Compute Interface for Portability (HIP) is AMD’s “CUDA-like” offloading model
- CCE 11.0 (Nov 2020) introduced support for compiling HIP source files targeting AMD GPUs
  - CCE HIP support relies on AMD’s open-source HIP implementation in upstream Clang/LLVM
- CCE does not provide HIP header files or runtime libraries
  - Header files and runtime libraries are needed from a standard AMD ROCm install
- CCE HIP will maintain compatibility with upstream HIP implementation whenever possible

CCE OPENMP/OPENACC FLAGS

Capability	CCE Fortran Flags	CCE C/C++ Flags
Enable/Disable OpenMP (disabled at default)	-f[no-]openmp -h[no]omp	-f[no-]openmp
Enable/Disable OpenACC (enabled at default)	-h[no]acc	N/A
Enable HIP	N/A	-x hip --rocm-path=$ROCM_PATH –L $ROCM_PATH/lib –lamdhip64

Offloading Target	All CCE Compilers (accel modules)	CCE C/C++ (optional flags)
Native Host CPU	craype-accel-host	(default without flags; no warning)
NVIDIA Volta[1]	craype-accel-nvidia70	-fopenmp-targets=nvptx64 -Xopenmp-target -march=sm_70
AMD MI60[2]	craype-accel-amd-gfx906	-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx906
AMD MI100[2]	craype-accel-amd-gfx908	-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx908

[1] NVIDIA GPU support limited to XC and CS systems
[2] AMD GPU support limited to Shasta and Apollo systems

CCE OFFLOADING BEST PRACTICES

THE MULTIPLE DIMENSIONS OF GPU PARALLELISM

NVIDIA	AMD	Description
Threadblock / CTA	Work group	• Loosely-coupled, course-grained parallelism • **Collective synchronization prohibited** • Performs best with massive parallelism • Performance scales with more powerful GPUs
Warp	Wavefront	• Fine-grained, independent parallelism • NVIDIA warp size is 32 threads • AMD wavefront size is 64 work items
Thread	Work item	• Fine-grained, lock-step parallelism • Performs best with stride-1 data accesses • Performs best with non-divergent control flow

OPENACC/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` `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

RUNTIME OFFLOADING MESSAGES

- Environment variable CRAY_ACC_DEBUG=[1-3]
- Emits runtime debug messages for offload activity (allocate, free, transfer, kernel launch, etc)

```
program main
 integer :: aaa(1000)
 aaa = 0
 !$omp target teams distribute map(aaa)
 do i=1,1000
  aaa(i) = 1
 end do

 if ( sum(abs(aaa)) .ne. 1000 ) then
  print *, "FAIL"
  call exit(-1)
 end if
 print *, "PASS"
end program main
```

```
ACC: Version 4.0 of HIP already initialized, runtime
version 3241
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 1 items from hello_gpu.f90:4
ACC:       allocate, copy to acc 'aaa(:)' (4000 bytes)
ACC: End transfer (to acc 4000 bytes, to host 0 bytes)
ACC: Execute kernel main_$ck_L4_1 blocks:8 threads:128
from hello_gpu.f90:4
ACC: Start transfer 1 items from hello_gpu.f90:7
ACC:       copy to host, free 'aaa(:)' (4000 bytes)
ACC: End transfer (to acc 0 bytes, to host 4000 bytes)
 PASS
```

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 are not yet optimized well (overly conservative locking)

UNIFIED MEMORY CAPABILITIES

- CCE supports OpenMP 5.0 allocator mechanisms
  - "pinned" allocator trait maps to *cudaMallocHost* or *hipMallocHost*
  - Vendor-specific allocator maps to *cudaMallocManaged* or *hipMallocManaged*
    - "cray_omp_get_managed_memory_allocator_handle()" returns a custom allocator handle (available in CCE 12.0)
- Environment variable, CRAY_ACC_USE_UNIFIED_MEM=1
  - CCE offloading runtime library will auto-detect user-allocations of pinned or managed memory
  - No explicit allocations or transfers will be issued for such memory
  - Original pointers passed directly into GPU kernels
  - CRAY_ACC_DEBUG runtime messages reflect this capability

THANK YOU

Jeff Sandoval
jeffrey.sandoval@hpe.com