Hewlett Packard Enterprise

HPE COMPILER GPU OFFLOADING

Steve Abbott
HPE Cray Programming Environment
& CORAL-2 Centers of Excellence

April 29, 2022

SOME RELEVANT PREVIOUS TALKS WITH SOME USER EXPERIENCE

Ordered chronologically

- **“Experiences in Implementing OpenMP offload support in Fortran”** by Kostas Makrides (HPE) & Aaron Black (LLNL).
- “**Asynchronous 3-D FFTs using OpenMP offload for extreme problem sizes**” by Kiran Ravikumar (Georgia Tech) et al
- “**Using OpenMP to Harness GPUs for Core-Collapse Supernova Simulations with GenASiS**” by Reuben Budiardja (ORNL).
- “**OpenMP experiences with Thornado**” by Austin Harris (ORNL).

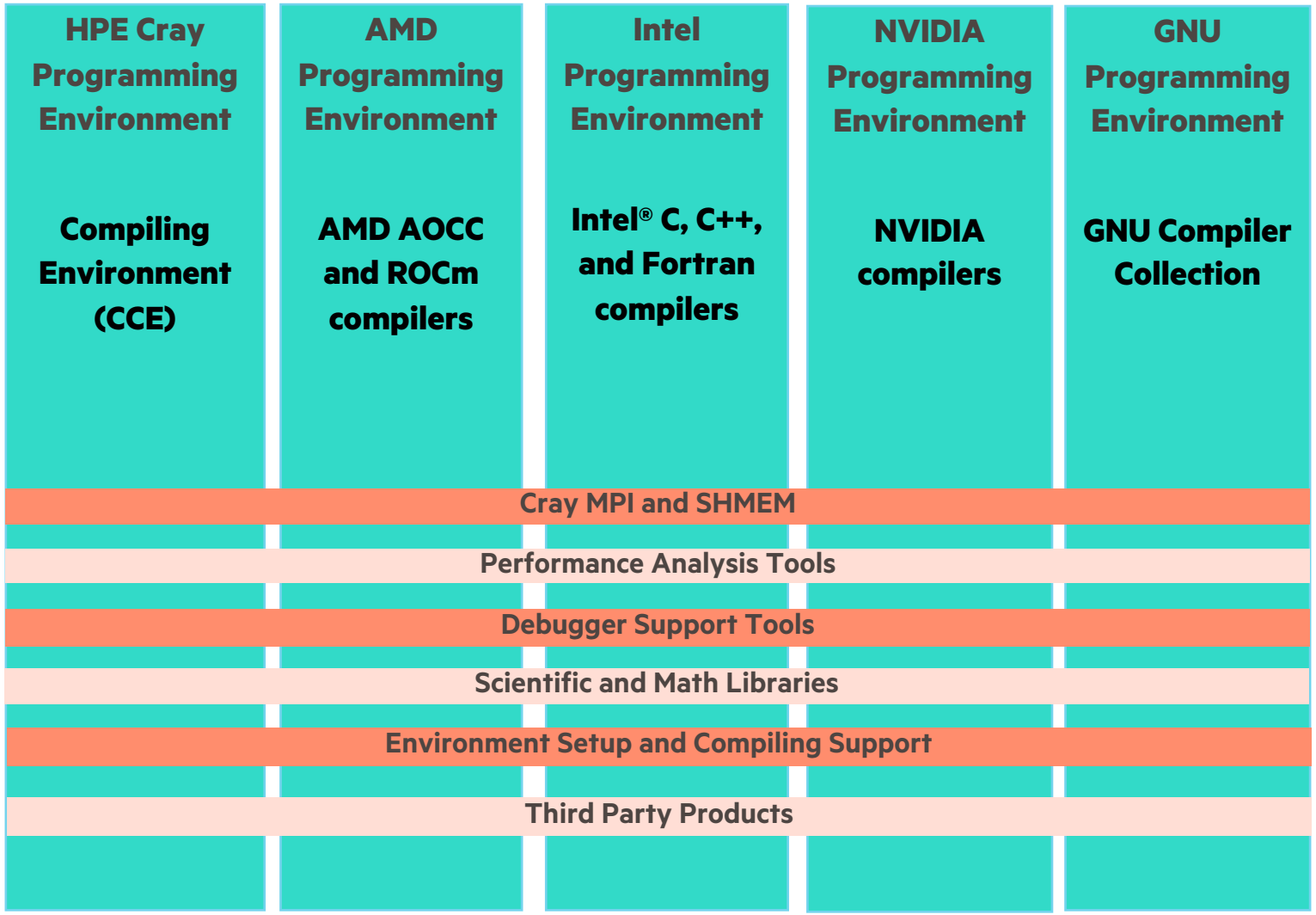
OUTLINE

- Broader HPE Cray PE accelerator support
- General compiler overview
- Offloading models
- Offloading feature highlights and best practices

PROVIDING THE USER WITH COMPILER CHOICE

Provides compiler and library choice, performance, and programmability

- Multiple programming environments
- Compiler interoperability
- Automatically uses our math, scientific, and communication libraries with chosen compiler
- Can use debug and profiling tools with chosen compiler



HPE CRAY PROGRAMMING ENVIRONMENT

Comprehensive set of tools for developing, porting, debugging, and tuning of HPC applications on HPE & HPE Cray systems

Development

Programming Models

- HPE Cray MPI
- SHMEM
- OpenMP | OpenACC
- AMD ROCm HIP | NVIDIA CUDA
- UPC | Fortran co-arrays
- Chapel

Programming Environments

- Compiling Environment
  - GNU
  - Intel Programming Environment
  - AMD Programming Environment
  - NVIDIA HPC SDK

Programming Languages

- C
- C++
- Fortran
- Python
- Chapel
- R

Optimized Libraries

- LibSci (BLAS)
- LAPACK & ScaLAPACK
- LibSci_ACC
- IRT
- FFTW

I/O Libraries

- NetCDF
- HDF5

DL / AI Tools

- Deep Learning Plug-in

Debugging

- **Comparative Debugger**
  Compare two versions of an application
- **GDB for HPC**
  Parallelized gdb for HPC
- **Valgrind for HPC**
  Memory debugging at scale
- **Stack Trace Analysis Tool**
  Stack tracing at scale
- **Tool for Abnormal Termination Processing**
  Manage core files at scale
- TotalView
- DDT

Performance Analysis & Optimization

- **Performance Analysis Tool (PAT)**
  Whole program performance analysis, exposing wide set of indicators, identifying bottlenecks and automatically generating suggestions to improve performance.
- **Visualization Tool**
  Complements text reports with summary of performance data in graphs and charts, allowing users to drill down and resolve issues
- **Code Parallelization Assistant**
  Reveal hidden potential of an application via code restructuring

Setup & Runtime

Environment Setup

- Tool Enablement
  (for Spack, CMake, EasyBuild, etc)
- Modules / Lmod

Supported systems:

- HPE Cray EX supercomputers
- HPE Apollo 2000
- HPE Apollo 6500
- HPE ProLiant DL systems
- Legacy Cray systems

HPE –authored | HPE Added-value to 3rd party | 3rd party

COMPILING ENVIRONMENT ADVANTAGES

Performance and programmability

Classic Cray Fortran compiler

Fortran 2018 (w. co-arrays)

Proprietary front end, optimizer; HPE-modified LLVM backend

C and C++ compiler

C11 and C++17; UPC

HPE-modified closed-source build of Clang+LLVM compiler

Offloading support

- OpenMP 5.0 and partial 5.1
- OpenACC 2.7 – focus on Fortran, 3.0 in 2022
- HIP – AMD GPUs only

Fully integrated heterogeneous optimization capability

- Providing consistency across all HPE and Cray HPC systems

  NVIDIA GPUs – Cray XC and CS systems today, Cray EX in 2022

  AMD GPUs – HPE Cray EX and Apollo systems
- Supporting:
  - x86-64 (both Intel and AMD) processors
  - ARM-based processors
  - NVIDIA accelerators
  - AMD accelerators

Integration with Program Development Tools

- Performance analysis tools exploit compiler's whole program analysis
- Interfaces through CCE's Program Library technology, an application-wide repository
- Code Parallelization Assistant leverages compiler analyses
- Compiler optimization feedback also supplied by performance reports for application tuning

Focus on application portability and investment protection

Focus on compliance and language support:

- Languages: Fortran, C/C++
- Programming models: OpenMP, OpenACC, and PGAS
- Encourage coding safety with strict standards compliance
- Support current versions of specifications

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
  - AMD GPUs
  - OpenMP 4.5 and near-complete 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 12.0 – based on Clang 12.0 – Jun 2021
  - CCE 13.0 – based on Clang 13.0 – Nov 2021
  - CCE 14.0 – based on Clang 14.0 – May 2022
  - CCE 15.0 – based on Clang 15.0 – Nov 2022 (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
- PDF manuals
  - Search at: https://support.hpe.com/hpesc/public/home
    - S-2179 for the release overview
    - S-3901 for the Fortran reference manual
    - S-5212 for the C/C++ quick reference guide
- Release information
  - module help cce/X.y.z

CCE OFFLOADING MODELS

CCE OPENMP SUPPORT

- Uses proprietary OpenMP runtime libraries
- Supports cross-language and cross-vendor OpenMP interoperability
- Implements HPE-optimized code generation for OpenMP offload regions
- Supports asynchronous ”nowait” GPU operations with ”depend” clauses
- Supports OpenMP allocators (e.g., CPU “pinned”, GPU “shared” and “managed”)
- Full OpenMP 4.5 support for Fortran, C, and C++
- OpenMP 5.x – in progress, implementation phased in 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 14.0 (May 2022)
  - 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 INTEROPERABILITY

- OpenMP CPU interoperability
  - CCE's libcraymp behaves as drop-in replacement for Clang's libomp and GNU's libgomp
  - GNU OpenMP interface support is currently limited to OpenMP 3.1 constructs
- OpenMP GPU interoperability
  - CCE's libcrayacc behaves as drop-in replacement for Clang's libomptarget
  - No planned support for GNU OpenMP offload interface
  - Device code relies on each vendor's device runtime library
  - Each vendor's device code is linked into a separate "device image"
  - CCE OpenMP offload linker tool handles device unbundling and linking
  - **Requires linking with CCE, or manually invoking the CCE OpenMP offload linker tool**

CCE OPENACC SUPPORT

- CCE supports OpenACC 2.0+ for Fortran
- C/C++ OpenACC support was dropped in CCE 10.0
- Full OpenACC 3.2 support planned for a future CCE release
- 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 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	craype-accel-nvidia70	-fopenmp-targets=nvptx64 -Xopenmp-target -march=sm_70
AMD MI100	craype-accel-amd-gfx908	-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx908
AMD MI250X	craype-accel-amd-gfx90a	-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx90a

CCE - ROCM COMPATIBILITY/INTEROPERABILITY

- CCE HIP offloading relies on ROCm headers, host libraries, and device bitcode libraries
- CCE OpenMP offloading relies on ROCm host libraries and device bitcode libraries
- Device bitcode libraries require a matching LLVM version between CCE and ROCm
- CCE OpenMP interoperability relies on compatible Clang OpenMP runtime ABI

	HIP/OpenMP (CCE Only)	OpenMP Interop (CCE + ROCm)
CCE 13.0.0	ROCm 4.1 – 4.5	ROCm 4.2 – 4.3
CCE 13.0.1	ROCm 4.1 – 4.5	ROCm 4.2 – 4.5
CCE 13.0.x	ROCm 4.1 – 4.5	ROCm 4.2 – 4.5
CCE 14.0.0	ROCm 5.0 – 5.1	ROCm 5.0 – 5.1

CCE - CUDA COMPATIBILITY

- CUDA API stability means compatibility is less constrained
- CCE runtime uses the CUDA Driver API
- Offload regions are compiled to PTX and passed to NVIDIA toolchain for assembly
- Testing may be limited to what's supported by underlying system software releases

CCE HIP SUPPORT

- CCE 11.0 (Nov 2020) introduced support for compiling HIP source files targeting AMD GPUs
- CCE HIP support leverages AMD's open-source HIP implementation in upstream Clang/LLVM
- CCE relies on HIP header files and runtime libraries from a standard AMD ROCm install
- CCE does not provide a "hipcc" wrapper – invoke the "CC" compiler driver directly

CCE HIP Flag	Description
-x hip	Enables HIP compilation for subsequent input files (avoid on link line or follow with "-x none")
--offload-arch=gfx90a	Specifies the MI250X offload target architecture
--rocm-path=<ROCM_PATH>	Specifies the location of a ROCm install; not required when $ROCM_PATH environment variable is set
-f[no-]gpu-rdc	Enables (disables) relocatable device code, producing bundled HIP offload object files and allowing cross-file references in HIP device code (default: -fno-gpu-rdc)
--hip-link	Enables device linking for bundled HIP offload object files; required when compiling with -fgpu-rdc
-mllvm -amdgpu-early-inline-all=true -mllvm -amdgpu-function-calls=false	Optimization flags that AMD's "hipcc" wrapper script provides; may provide additional performance benefit

CCE OFFLOADING FEATURE HIGHLIGHTS AND BEST PRACTICES

THE MULTIPLE DIMENSIONS OF GPU PARALLELISM

AMD	NVIDIA	Description
Work group	Threadblock / CTA	• Loosely-coupled, course-grained parallelism • **Collective synchronization prohibited** • Performs best with massive parallelism • Performance scales with more powerful GPUs
Wavefront	Warp	• Fine-grained, independent parallelism • NVIDIA warp size is 32 threads • AMD wavefront size is 64 work items
Work item	Thread	• 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 parallel`
Thread	Work item	`acc vector`		`omp simd`	

- 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 is optimized as of CCE 13.0 (Nov 2021) – relaxed locking strategy

CCE OPENMP UNIFIED MEMORY DETECTION

- CCE's default runtime behavior for OpenMP map clauses is to allocate/transfer GPU memory
- Dynamically enable GPU managed memory for OpenMP map clauses
  - Set env var CRAY_ACC_USE_UNIFIED_MEM=1
  - Triggers runtime check to skip explicit allocate/transfer for managed memory

CCE OPENMP UNIFIED SHARED MEMORY SUPPORT FOR AMD MI250X

- Dynamically enable GPU unified memory for OpenMP map clauses
  - Set env vars CRAY_ACC_USE_UNIFIED_MEM=1 and HSA_XNACK=1
  - Skips explicit allocate/transfer for all system memory
  - Global ”declare target” variables will still be allocated separately (compiler statically emits a device copy)
- Statically enable GPU unified memory for OpenMP map clauses
  - Compile with “requires unified_shared_memory” directive
  - Set env var HSA_XNACK=1

HPE COMPILER SUMMARY

- CCE is one component of the broader HPE Cray Programming Environment
- Consistent development environment across a wide variety of CPU and GPU targets
- Support for the latest base language standards
  - Fortran 2018 support (including coarray teams)
  - C11 and C++17 support
- Support for several on-node parallel/offloading models
  - OpenMP 4.5, working towards 5.2
  - OpenACC 2.0, working towards 3.2
  - HIP
- Please reach out or file bugs if you have questions or encounter issues

THANK YOU

Steve Abbott
stephen.abbott@hpe.com

CCE OPENMP ALLOCATOR SPECIALIZATION

Use Case	Allocator Mechanism	Notes
“Pinned” CPU memory	Allocator with “pinned” trait set	• Maps to hipMallocHost
“Shared” GPU memory	omp_cgroup_mem_alloc predefined allocator	• Maps to static allocation in LDS memory • Must be lexically specified on “allocate” clause on “teams” construct • Currently supported for Fortran only
“Managed” memory	cray_omp_get_managed_memory_allocator_handle()	• Maps to hipMallocManaged • CCE-specific extension • Topic of interest for OpenMP committee