Hewlett Packard
Enterprise

AN UPDATE ON THE HPE/CRAY COMPILER

Deepak Eachempati
CCE OpenMP Team

September 13, 2021

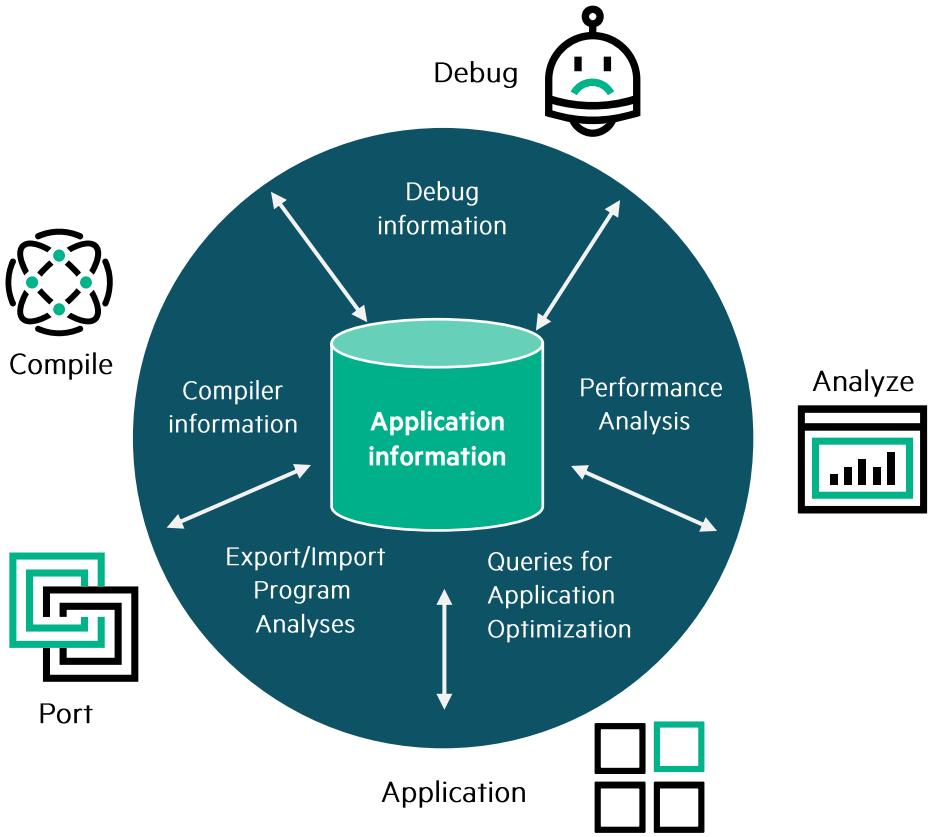
OUTLINE

- Overview of the Cray Compiling Environment (CCE)
- OpenMP Current Status
- OpenMP Implementation Specifics

OVERVIEW OF THE CRAY COMPILING ENVIRONMENT (CCE)

HPE CRAY PROGRAMMING ENVIRONMENT (CPE)

- Fully integrated software suite designed to increase programmer productivity, scalability, and performance
- Components:
  - **Cray Compiling Environment (CCE)**
  - Cray Performance Measurement & Analysis Tools (CPMAT)
  - Cray Debugging Support Tools (CDST)
  - Cray Scientific and Math Libraries (CSML)
  - Cray Message Passing Toolkit (CMPT)
  - Cray Environment Setup and Compiling Support (CENV)



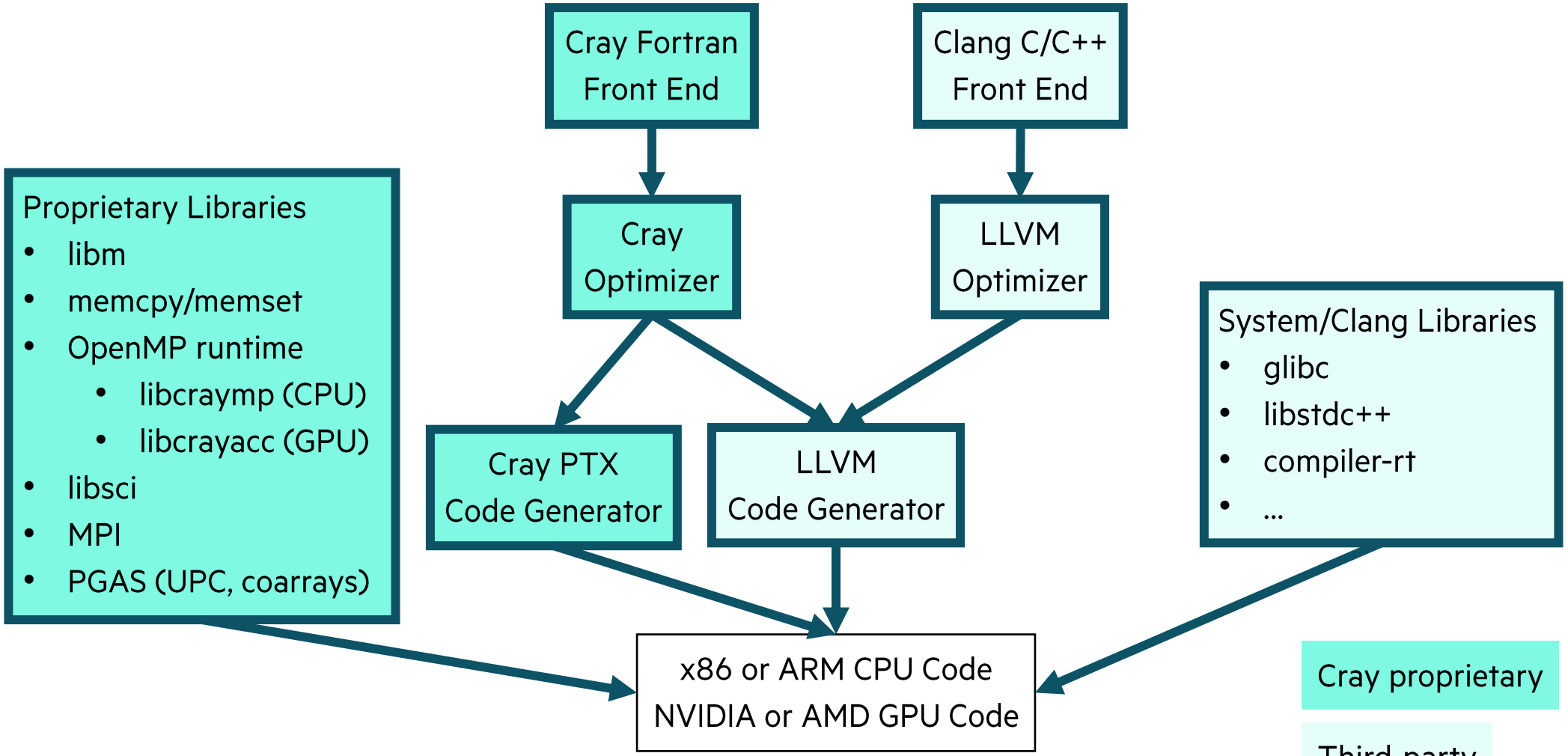
CRAY COMPILING ENVIRONMENT (CCE)

- Fortran, C, and C++ compilers
  - Integrated parallel programming model support (OpenMP, UPC, Fortran coarrays)
  - Automatic scalar, vector, and threading optimizations
  - HPC-optimized runtime libraries
  - Supporting utilities like linker, objdump, sanitizers, etc.

- CCE evolving technology
  - Focus shift from custom Cray architectures to high performance versions of commodity CPUs and GPUs
  - Increasing focus on C++ in HPC applications
  - Continue to support key customers segments heavily invested in Fortran

- Accelerator offloading support for AMD and NVIDIA GPUs
  - OpenMP 4.5 and partial 5.0
  - OpenACC 2.0 and partial 2.6 (Fortran only)
  - HIP (AMD GPUs only)

CURRENT CCE ARCHITECTURE (2018 TO PRESENT)



OVERALL CLANG/LLVM APPROACH

- Provide customers with the benefits of the base upstream Clang/LLVM compiler
  - Quick adoption of new language standards
  - Fast compilation
  - Extensive diagnostic messages
  - Industry standard compiler flags
  - Good performance
- Collaborate with LLVM community to improve the open-source project
  - The success of CCE depends on the success of the LLVM project
- Enhance CCE Clang in areas that add value to our customers

CCE CLANG DIFFERENTIATION

- Enhanced features
  - Loopmark (intuitive visual representation of key optimizations)
  - Decompile (for advanced users who desire the details of how their code was transformed)
  - UPC (Unified Parallel C)
  - Limited support for Cray pragmas
- Improved performance
  - Proprietary CCE OpenMP runtime (CPU and GPU)
  - Proprietary Clang OpenMP offload handling
  - Proprietary CCE libm (optimized versions of select math functions)
  - New optimizations and improvements to existing optimizations
  - Adjusted tuning parameters
- Integration with the rest of the programming environment
  - Cray Perftools
  - Cray Parallelization Assistant (Reveal)

COMMON CCE COMPILER FLAGS

Fortran	C/C++	Purpose
-O0	-O0 *(default)*	No optimization
-O1	-O1 -ffast-math	Light optimization
-O2 *(default)*	-O2 -ffast-math	Moderate optimization
-O3 -Ofast	-O3 -ffast-math -Ofast *(recommended starting point)*	Heavy optimization
-hpic, -hPIC	-fpic, -fPIC	Generate position-independent code
-hlist=m	-fsave-loopmark	Emit Loopmark (.lst)
-hlist=d	-fsave-decompile	Emit Decompile (.dc and .ll)

- CCE C/C++ compiler is a drop-in replacement for upstream Clang
  - HPE adds options for new features but keeps existing options
  - Special *–fno-cray* option disables HPE/Cray changes
- Only common CCE Fortran and C/C++ flags are equivalent

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
  - CCE 13.0 – based on Clang 13.0 – Nov 2021
  - CCE 14.0 – based on Clang 14.0 – May 2022 (tentative)
- Release cadence and versioning changed in CCE 10.0
- Older versions of CCE do not correspond to Clang/LLVM version numbers

HPE COMPILER DOCUMENTATION

- Online documentation at https://support.hpe.com
  - HPE **Cray Compiling Environment Release Overview** (12.0) (S-5212)
  - Cray Fortran Reference Manual (12.0) (**S-3901**)
  - HPE Cray Clang C and C++ Quick Reference (12.0) (**S-2179**)

- man pages of interest (installed with compiler)
  - 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

OPENMP CURRENT STATUS

CCE FLAGS FOR OPENMP AND ACCELERATORS

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 MI60	craype-accel-amd-gfx906	-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx906
AMD MI100	craype-accel-amd-gfx908	-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx908

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)

- 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, tentative)

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

CCE OPENMP 5.0 IMPLEMENTATION LIMITATIONS

- Several hints are parsed and accepted, but no action is taken
  - simd nontemporal, atomic hints, nonmonotonic loop scheduling, and the close modifier
- The **loop** construct “bind” semantics are honored, but no additional optimization is performed
- The **order(concurrent)** clause is accepted, but is not used for additional optimization
- Pause resource initially only causes the CCE offloading runtime library to relinquish GPU memory
- The “requires” directive is parsed, but any clauses that are not yet supported produce compile errors
  - This is allowed by the OpenMP specification
  - “dynamic_allocators”, “reverse_offload” and “unified_shared_memory” currently result in compile time errors

OPENMP IMPLEMENTATION SPECIFICS

RUNTIME OFFLOADING MESSAGES

- Environment variable CRAY_ACC_DEBUG=[1-3]
- Emits runtime debug messages for offload activity (allocate, free, transfer, kernel launch, etc)
- Enhanced information for pointer attachments and unified memory for OpenMP 5.0

```fortran
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)
- host-to-device dependences are not yet optimized well (overly conservative synchronization)
- Multi-threaded use of GPU are not yet optimized well (overly conservative locking)

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

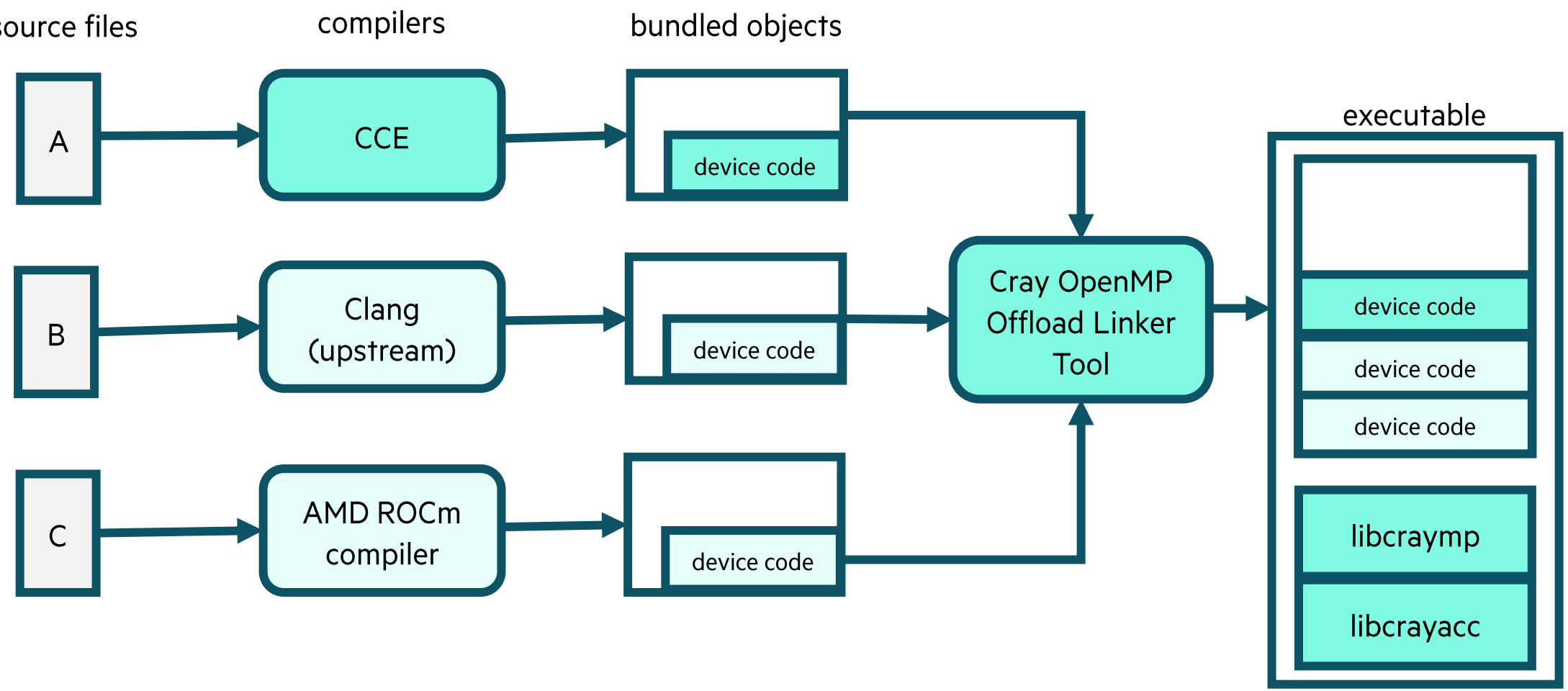
Long-term goal: let users express parallelism with any construct they think makes sense, and CCE will map to available hardware parallelism

OPENMP 5.0 MEMORY MANAGEMENT SUPPORT

Allocation Mechanism	Use Case	Description
user-defined allocator with **pinned** trait set to true	Allocated “pinned” host memory that is accessible for NVIDIA or AMD GPU	Results in underlying call to **cudaMallocHost** or **hipMallocHost**
omp_cgroup_mem_alloc predefined allocator	Allocate private variables for **teams** construct in “shared” GPU memory	Result in static allocation in fast local “shared” memory for the launched kernel. Fortran only.
cray_omp_get_managed_memory_allocator_handle() (extension)	“Managed” memory allocation.	Results in underlying call to **cudaMallocManaged** or **hipMallocManaged**

- 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

OPENMP COMPILER INTEROPERABILITY



SUPPORT FOR OPENMP PROFILING

- Proprietary compiler-based Cray OpenMP trace points for CrayPAT

- OMPT profiling tool interface supported as of CCE 11
  - Supports all mandatory events on CPU (host device)
  - Supports events relating to device initialization and target offload
  - Support events for synchronization regions, reductions, and some worksharing constructs
  - Does not currently support device tracing interface
  - Will support the “external monitoring interface” for device tracing from OpenMP 5.1 in an upcoming release

SUPPORT FOR OPENMP DEBUGGING

- OMPD library shipped with CCE as of CCE 11
  - enables a debugger to access information about threads, tasks, and teams managed by the OpenMP runtime.
  - permits OpenMP runtime frame filtering
  - Full support on CPU
  - Support for GPUs are in progress

Default back trace

```
(gdb) bt
#0  fib (n=0) at fibonacci.c:11
#1  0x0000000000202c26 in fib_cray$mt$p0002 () at fibonacci.c:17
#2  0x000015554c083e82 in clang_task_trampoline() () at …/libcraymp.so.1
#3  0x000015554c06e251 in __cray$mt_execute_task () at …/libcraymp.so.1
#4  0x000015554c06f7dc in __cray$mt_taskwait () at …/libcraymp.so.1
#5  0x000015554c0871ed in __kmpc_omp_taskwait () at …/libcraymp.so.1
#6  0x0000000000202a35 in fib (n=2) at fibonacci.c:18
#7  0x0000000000202c26 in fib_cray$mt$p0002 () at fibonac
#8  0x000015554c083e82 in clang_task_trampoline() () at …
#9  0x000015554c06e251 in __cray$mt_execute_task () at …/
#10 0x000015554c06f7dc in __cray$mt_taskwait () at …/libc
#11 0x000015554c0871ed in __kmpc_omp_taskwait () at …/lib
#12 0x0000000000202a35 in fib (n=4) at fibonacci.c:18
#13 0x0000000000202c26 in fib_cray$mt$p0002 () at fibonac
#14 0x000015554c083e82 in clang_task_trampoline() () at …
#15 0x000015554c06e251 in __cray$mt_execute_task () at …/
#16 0x000015554c06f7dc in __cray$mt_taskwait () at …/libc
#17 0x000015554c0871ed in __kmpc_omp_taskwait () at …/lib
#18 0x0000000000202a35 in fib (n=6) at fibonacci.c:18
#19 0x0000000000202c26 in fib_cray$mt$p0002 () at fibonac
#20 0x000015554c083e82 in clang_task_trampoline() () at …
#21 0x000015554c06e251 in __cray$mt_execute_task () at …/
...
```

OMPD-enhanced back trace

```
(gdb) bt
#0  @thread 3: fib (n=0) at fibonacci.c:12
#1  0x0000000000202b86 in @thread 3: "#pragma omp task" () at fibonacci.c:17
#6  0x0000000000202a2c in @thread 3: fib (n=2) at fibonacci.c:18
#7  0x0000000000202b86 in @thread 3: "#pragma omp task" () at fibonacci.c:17
#12 0x0000000000202a2c in @thread 3: fib (n=4) at fibonacci.c:18
#13 0x0000000000202b86 in @thread 3: "#pragma omp task" () at fibonacci.c:17
#18 0x0000000000202a2c in @thread 3: fib (n=6) at fibonacci.c:18
#19 0x0000000000202b86 in @thread 3: "#pragma omp task" () at fibonacci.c:17
#24 0x0000000000202a2c in @thread 3: fib (n=8) at fibonacci.c:18
#25 0x0000000000202b86 in @thread 3: "#pragma omp task" () at fibonacci.c:17
#32 0x000000000020326a in @thread 3: .omp_outlined._debug__ (.global_tid.=0x155530098858,
.bound_tid.=0x15553009885c, n=@0x7fffffffa4dc: 10) at fibonacci.c:29
#33 0x0000000000203295 in @thread 3: .omp_outlined..24 (.global_tid.=0x155530098858,
.bound_tid.=0x15553009885c, n=@0x7fffffffa4dc: 10) at fibonacci.c:27
#34 0x00000000002032ac in @thread 3: "#pragma omp parallel" () at fibonacci.c:27
#### switching to thread 1 ####
#31 0x000000000020319f in @thread 1: main (argc=1, argv=0x7fffffffa5d8) at fibonacci.c:27
```

CCE OPENMP SUPPORT RECAP

- 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
  - Full OpenMP 5.0 planned for CCE 13 (Nov 2021)

- OpenMP 5.1 – implementation planned over several CCE releases
  - Some minor features, clarifications planned for CCE 13
  - Full OpenMP 5.1 planned for CCE 14 (May 2022)

THANK YOU
QUESTIONS?

Deepak Eachempati
deepak.eachempati@hpe.com