Outline

- Cray Compiling Environment (CCE) introduction
- CCE OpenMP overview
- CCE OpenMP offloading best practices
- Future CCE OpenMP roadmap
Cray Compilation Environment (CCE)

- Fortran compiler
  - Proprietary front end and optimizer; Cray-modified LLVM
  - Fortran 2018 support, excluding coarray teams
- Two C and C++ compiler options
  - Default option: Cray-modified Clang+LLVM compiler
  - “Classic” option: EDG front end; proprietary optimizer; Cray-modified LLVM
  - C11 and C++17 support
  - gcc 8.1 compatibility
  - UPC support
- OpenMP 4.5 support, including offloading for NVIDIA GPUs
CCE Motivation

• Why does Cray develop a compiler at all?
  • Performance
  • Customer support

• Why build on Clang/LLVM?
  • Clang leads C & C++ compilers in standards support and diagnostics
  • Clang’s compile time is much faster than Cray’s existing compiler
  • Cray can contribute non-differentiating work upstream
  • Cray can keep some differentiating work proprietary

A solid base compiler infrastructure frees up Cray compiler developers to focus more narrowly on HPC and customer-driven features and optimizations
CCE Clang Differentiation

- Loopmark
- Decompile
- CrayPat support
- Cray OpenMP runtime (CPU and GPU)
- Cray libm (optimized versions of select math functions)
- UPC
- Reveal support (in development)
- Enhanced FP trap support (in development)
CCE 9.0

- **Fortran Source**
  - Fortran FE
  - IPA
  - Optimizer
- **C and C++ Source**
  - C/C++ FE (EDG)
- **CCE-Classic**
  - Cray Libs (I/O, Math, ...) Listing Tools
  - Cray Libs (Math, OpenMP, PGAS) Listing Tools
- **CCE-Clang**
  - C/C++ FE (CLANG)
  - LLVM
    - CCE Differentiation:
      - Fast OpenMP 4.5
      - Vectorization
      - Loopmark
      - Reveal & CrayPat support
- **Object Files**
  - IPA
  - Optimizer
  - X86 CG (LLVM)
  - Aarch64 CG (LLVM)
  - GPU CG

**Notes**
- Cray developed
- 3rd party with Cray value added

© 2019 Cray, a Hewlett Packard Enterprise company
CCE Release and Versioning

• Moving to two major releases a year (~Q2 and ~Q4)
  • CCE codebase and version based off latest Clang major release
  • Monthly minor updates will continue for ~5 months after each major release

• CCE 9.0 – June 2019
  • Based on development Clang 9.0

• CCE 9.1 – Nov 2019
  • Based on released Clang 9.0

• CCE 10.0 – May 2020 (tentative)
  • Based on released Clang 10.0
Invoking CCE

- Different CCE modules for Clang or Classic C/C++
  - module load PrgEnv-cr
  - module avail cce
  - module swap cce/9.0.2
  - module swap cce/9.0.2-classic
- Fortran compiler is the same for both modules
- Invoke compiler through CrayPE driver
  - cc <CFLAGS> source.c
  - CC <CXXFLAGS> source.cpp
  - ftn <FFLAGS> source.f90
Common CCE Compiler Flags

<table>
<thead>
<tr>
<th>CCE Classic 9.0</th>
<th>CCE 9.0</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O0</td>
<td>-O0</td>
<td>(default)</td>
</tr>
<tr>
<td>-O1</td>
<td>-O1 -ffast-math</td>
<td>Light optimization</td>
</tr>
<tr>
<td>-O2 (default)</td>
<td>-O2 -ffast-math</td>
<td>Moderate optimization</td>
</tr>
<tr>
<td>-O3</td>
<td>-O3 -ffast-math</td>
<td>(recommended starting point)</td>
</tr>
<tr>
<td></td>
<td>-Ofast</td>
<td></td>
</tr>
<tr>
<td>-hpic, -hPIC</td>
<td>-fpic, -fPIC</td>
<td>Generate position-independent code</td>
</tr>
<tr>
<td>-hlist=m</td>
<td>-fsave-loopmark</td>
<td>Emit Loopmark (.lst)</td>
</tr>
<tr>
<td>-hlist=d</td>
<td>-fsave-decompile</td>
<td>Emit Decompile (.dc and .ll)</td>
</tr>
</tbody>
</table>

- CCE Clang is a drop-in replacement for upstream Clang
  - Cray adds options for new features but keeps existing options
  - Special –fno-cray option disables Cray changes
- CCE Clang is NOT a drop-in replacement for CCE Classic – options differ
CCE OPENMP
OVERVIEW
## CCE OpenMP Flags

<table>
<thead>
<tr>
<th>General OpenMP</th>
<th>CCE 8.7</th>
<th>CCE 9.0 Fortran and Classic</th>
<th>CCE 9.0 Clang C/C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable OpenMP</td>
<td>-homp (default)</td>
<td>-homp</td>
<td>-fopenmp (Cray runtime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-fopenmp (alias)</td>
<td>-fopenmp (Cray runtime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-fopenmp (alias)</td>
<td>-fopenmp (Cray runtime)</td>
</tr>
<tr>
<td>Disable OpenMP</td>
<td>-hnoomp</td>
<td>-hnoomp (default)</td>
<td>-fno-openmp (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-fno-openmp (alias)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### OpenMP Offloading

<table>
<thead>
<tr>
<th>All CCE Compilers (accel modules)</th>
<th>CCE 9.0 Clang C/C++ (optional flags)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Host CPU</td>
<td>(default without flags; no warning)</td>
</tr>
<tr>
<td>NVIDIA Pascal</td>
<td>-fopenmp-targets=nvptx64 -Xopenmp-target -march=sm_60</td>
</tr>
<tr>
<td>NVIDIA Volta</td>
<td>-fopenmp-targets=nvptx64 -Xopenmp-target -march=sm_70</td>
</tr>
<tr>
<td>AMD Vega20*</td>
<td>-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx906</td>
</tr>
</tbody>
</table>

*AMD GPU support under development*
CCE 9.0 Clang C/C++ OpenMP Plan

- Fully support OpenMP 4.5 (CPU and GPU)
  - NVIDIA support today, AMD GPU support under development
- Use CCE OpenMP runtime libraries
  - Offers interoperability with CCE Fortran (and Classic C/C++)
  - Provides a lightweight, HPC-optimized runtime
  - Requires a thin “adapter” layer in Cray’s runtime
- Implement Cray-optimized Clang code generation for accelerator regions
  - Mimics the CCE Classic implementation
  - Leverages compiler analysis/translation with ultra-lightweight device runtime
  - Offers significant performance advantage over upstream Clang
CCE OpenMP Limitations on GPU

• CCE currently handles these core constructs
  • target, teams, distribute, parallel, do/for, barrier, simd
• Support for other constructs is under development
  • E.g., master, single, critical
• Some constructs may be implemented to be “functionally compliant”
  • E.g., task, taskloop, cancel, doacross
• Our implementation is driven by user feedback
  • More feedback is always appreciated
The **libcraaymp** GNU interface is currently limited to OpenMP 3.1 constructs
Accesses to global variables and calls to functions cannot cross a “compiler boundary”
Additional pre-link steps may need to be added to the application build process if not linked by CCE.
The libcrayacc GNU offloading interface is not yet supported
OpenMP Offloading Strategy

1. Offload time-intensive parallel loops
   • Focus on functional correctness
   • Rely on “point-of-use” data transfers

2. Optimize kernel computation
   • Temporarily ignore data transfer overheads

3. Optimize data transfers
   • Add enclosing data regions and updates where necessary
   • Communicate directly to/from GPU memory

4. Use device asynchronously
   • Fill device “queue” with a “stream” of dependent work
   • Execute multiple kernels in parallel
Step 1: Offload Code to GPU with omp target

- “omp target” offloads region of code to an accelerator
  - Does not express any parallelism
- “Map” clauses move data to/from accelerator memory
  - Scalars are made “firstprivate” at default
  - Non-scalars are mapped “tofrom” at default
- “declare target” makes functions and global variables available on accelerator
- “nowait” and “depend” clauses express asynchronous operations

```c
$omp target
... ! This code runs serially on the GPU
$omp end target
```
Step 2: Express Parallelism on GPU
## GPU Terminology (Secret Decoder Ring)

<table>
<thead>
<tr>
<th>NVIDIA</th>
<th>AMD</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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/wavefront size is 32 threads  
• AMD warp/wavefront size is 64 threads |
| Thread          | Work item      | • Fine-grained, lock-step parallelism  
• Performs best with stride-1 data accesses  
• Performs best with non-divergent control flow |
omp teams

- Specifies fork/join parallelism, like “omp parallel”
  - API calls support querying “team num” and “num teams”
- Collective synchronization prohibited (i.e., no barriers)
  - Allows mapping directly to GPU threadblocks
- Must appear directly within “omp target”

```cpp
!$omp target
!$omp teams
... ! This code runs on the GPU using multiple “teams”
!$omp end teams
!$omp end target
```
omp distribute

• Loop partitioning construct, like “omp do/for”
  • Iterations are partitioned across “teams”

• There is no implied barrier at end of loop

• Best practice: combine as “target teams distribute”

```bash
!$omp target teams
!$omp distribute
do i=1,n
  ... ! Iterations are partitioned across “teams”
end do
!$omp end target teams
```
### OpenMP Construct Mapping to GPU

<table>
<thead>
<tr>
<th>Construct</th>
<th>CCE Classic</th>
<th>CCE 9.0 Clang</th>
<th>Upstream Clang</th>
</tr>
</thead>
</table>
| omp teams | • Threadblocks  
            • Threadblocks+threads | • Threadblocks  
            • Threadblocks+threads | • Threadblocks |
| omp parallel | • Unused (aggressive autothreading on GPU) | • Threads (with limitations) | • Threads |
| omp simd | • Threads | • Threads | • Unused |

- teams is required to map to GPU threadblocks, for all implementations
- parallel and simd are inconsistent across implementations
  - CCE parallel limitations: called functions are visible; only contains barrier, for with static schedule, and omp_get_thread_num / omp_get_num_threads queries
  - **Current Best practice:** specify the composite parallel do/for simd
- **Long term CCE goal:** let users express parallelism with any construct they think makes sense, and CCE will map to available hardware parallelism
omp parallel for simd

- Best practice: combine with “teams” to express two-level GPU parallelism
  - “teams” maps to GPU threadblocks
  - CCE-Classic maps “simd” to GPU threads (skips “parallel for”)  
  - Clang maps “parallel for” to GPU threads (skips “simd”)

```c
!$omp target teams distribute parallel for simd
do i=1,n
  ...
  ! 2-level parallelism
end do
!$omp end target teams
```

```c
!$omp target teams distribute parallel for simd
do i=1,n
  !$omp parallel for simd
do j=1,n
    ...
    ! 2-level parallelism
  end do
end do
```
16. #pragma omp target teams distribute
17. gG--< for (int i = 0 ; i < 1000 ; i++) {
18.     gG a[i] = b[i] + c[i];
19.     gG-->
20. }

$ cc -hmsgs -hlist=m -homp test.c

CC-6405 craycc: ACCEL File = test.c, Line = 17
A region starting at line 17 and ending at line 19 was placed on the accelerator.

CC-6430 craycc: ACCEL File = test.c, Line = 17
A loop was partitioned across the threadblocks and the 128 threads within a threadblock.

*CCE-Classic only; CCE-Clang is under development
#pragma omp target data map(from:a) map(alloc:b,c)
{
    #pragma omp target teams distribute
    for (int i = 0 ; i < 1000 ; i++) {
        b[i] = c[i] = 1.0;

    #pragma omp target teams distribute
    for (int i = 0 ; i < 1000 ; i++)
        a[i] = b[i] + c[i];
}
Debugging Data Directives

• CRAY_ACC_DEBUG
  • Recommended values: 1 or 2
  • Emits messages for all offloading operations (e.g., allocate, transfer, launch)
    • File/line/var info for CCE-Clang coming in CCE 9.1.0
  • Debug code with "host" accelerator target
    • All maps/updates become noops
$ CRAY_ACC_DEBUG=1 ./a.out

ACC: Transfer 3 items (to acc 12000 bytes, to host 0 bytes) from test.c:16

ACC: Execute kernel main$ck_L16_1 async(auto) from test.c:16

ACC: Wait async(auto) from test.c:18

ACC: Transfer 3 items (to acc 0 bytes, to host 4000 bytes) from test.c:18
$ CRAY_ACC_DEBUG=2 ./a.out
ACC: Start transfer 3 items from test.c:16
ACC: allocate, copy to acc 'a' (4000 bytes)
ACC: allocate, copy to acc 'b' (4000 bytes)
ACC: allocate, copy to acc 'c' (4000 bytes)
ACC: End transfer (to acc 12000 bytes, to host 0 bytes)
ACC: Execute kernel main$ck_L16_1 blocks:8 threads:128 async(auto) from test.c:16
ACC: Wait async(auto) from test.c:18
ACC: Start transfer 3 items from test.c:18
ACC: copy to host, free 'a' (4000 bytes)
ACC: free 'b' (4000 bytes)
ACC: free 'c' (4000 bytes)
ACC: End transfer (to acc 0 bytes, to host 4000 bytes)
MPI Communication on GPU Memory

```c
#pragma omp target data map(buffer[:n])
{
  #pragma omp target teams distribute
      for (int i = 0; i < n; i++) buffer[i] = ...;

#pragma omp target data use_device_ptr(buffer)
   {
      MPI_Send(buffer,n,...); // requires GPU-aware MPI
      // gpu_function(buffer, ...); // Or other GPU-aware function
   }
}
```
Step 4: Use GPU Asynchronously

```c
void target_nowait_example(int N, double *A, double *B, double *C) {
    #pragma omp target enter data map(alloca:A[:N], B[:N], C[:N])

    #pragma omp target teams distribute nowait depend(out:A)
    for(int i=0; i<N; i++) A[i] = i*4; // Kernel A

    #pragma omp target teams distribute nowait depend(out:B)
    for(int i=0; i<N; i++) B[i] = i*2; // Kernel B

    #pragma omp target teams distribute nowait depend(in:A,B) depend(out:C)
    for(int i=0; i<N; i++) C[i] = A[i] + B[i]; // Kernel C

    #pragma omp target exit data depend(in:C) map(from:C[:N]) map(delete:A[:N],B[:N])
}
```
CCE OPENMP ROADMAP
CCE OpenMP Roadmap

• 2019 Jun: **CCE 9.0**
  • Alpha CCE Clang OpenMP 4.5 for NVIDIA GPUs

• 2019 Nov: **CCE 9.1**
  • Full CCE Clang OpenMP 4.5 for NVIDIA GPUs

• 2020 Feb: **CCE 9.1.x**
  • Pre-alpha OpenMP 4.5 for AMD GPUs (*CORAL-2 only*)

• 2020 May: **CCE 10.0**
  • Partial OpenMP 5.0 support

• 2020 Nov: **CCE 11.0**
  • Full OpenMP 5.0 support
FORWARD LOOKING STATEMENTS

This presentation may contain forward-looking statements that involve risks, uncertainties and assumptions. If the risks or uncertainties ever materialize or the assumptions prove incorrect, the results of Hewlett Packard Enterprise Company and its consolidated subsidiaries (“Hewlett Packard Enterprise”) may differ materially from those expressed or implied by such forward-looking statements and assumptions. All statements other than statements of historical fact are statements that could be deemed forward-looking statements, including but not limited to any statements regarding the expected benefits and costs of the transaction contemplated by this presentation; the expected timing of the completion of the transaction; the ability of HPE, its subsidiaries and Cray to complete the transaction considering the various conditions to the transaction, some of which are outside the parties’ control, including those conditions related to regulatory approvals; projections of revenue, margins, expenses, net earnings, net earnings per share, cash flows, or other financial items; any statements concerning the expected development, performance, market share or competitive performance relating to products or services; any statements regarding current or future macroeconomic trends or events and the impact of those trends and events on Hewlett Packard Enterprise and its financial performance; any statements of expectation or belief; and any statements of assumptions underlying any of the foregoing. Risks, uncertainties and assumptions include the possibility that expected benefits of the transaction described in this presentation may not materialize as expected; that the transaction may not be timely completed, if at all; that, prior to the completion of the transaction, Cray’s business may not perform as expected due to transaction-related uncertainty or other factors; that the parties are unable to successfully implement integration strategies; the need to address the many challenges facing Hewlett Packard Enterprise’s businesses; the competitive pressures faced by Hewlett Packard Enterprise’s businesses; risks associated with executing Hewlett Packard Enterprise’s strategy; the impact of macroeconomic and geopolitical trends and events; the development and transition of new products and services and the enhancement of existing products and services to meet customer needs and respond to emerging technological trends; and other risks that are described in our Fiscal Year 2018 Annual Report on Form 10-K, and that are otherwise described or updated from time to time in Hewlett Packard Enterprise’s other filings with the Securities and Exchange Commission, including but not limited to our subsequent Quarterly Reports on Form 10-Q. Hewlett Packard Enterprise assumes no obligation and does not intend to update these forward-looking statements.
THANK YOU

QUESTIONS?

cray.com
@cray_inc
linkedin.com/company/cray-inc-/

TM
"THE FUTURE IS Seldom the same as the past."

SEYMOUR CRAY