Directive based GPU programming on AMD systems

Justin Chang, Member of Technical Staff
Leopold Grinberg, Principal Member of Technical Staff
DIRECTIVE BASED PROGRAMMING (HIGH LEVEL VIEW)

With OpenMP and OpenACC directives developers can:

a) Specify to the compiler code regions that need to be compiled for a HOST and for a DEVICE (typically CPU and GPU)

b) Manage memory and data (allocate memory on HOST and DEVICE and copy data between the two)

c) Express parallelism and potentially mapping different work items to different threads

d) Compiler directives reduce the effort and development time for porting codes to highly parallel devices such as multi-core CPUs and GPUs
AGENDA

In this talk, we will demonstrate:

1. OpenMP offloading is both compatible and competitive with HIP

2. A weather modeling application (MPAS) can run on AMD discrete GPUs

3. Heterogeneous Memory Management (HMM) is ready to use on AMD systems
Hybrid programming here stands for the interaction of OpenMP with a lower-level programming model like HIP. In other words, one can program with OpenMP in the style one might program with HIP.

OpenMP supports the following interactions:

- Calling low-level HIP kernels from OpenMP application code
- Calling HIP/ROCM math libraries (rocBLAS, rocFFT, etc.) from OpenMP application code
- Calling OpenMP kernels from low-level HIP application code
void example() {
    float a = 2.0;
    float * x;
    float * y;

    #pragma omp target data map(to:x[0:count]) map(tofrom:y[0:count])
    {
        compute_1(n, x);
        compute_2(n, y);
        #pragma omp target update to(x[0:count]) to(y[0:count])
        saxpy(n, a, x, y)
        compute_3(n, y);
    }
}

Let's assume that we want to implement the saxpy() function in a low-level language.

Allocate device memory for x and y, and specify directions of data transfers
A HIP version of the SAXPY kernel:

```c
__global__ void saxpy_kernel(size_t n, float a, float * x, float * y) {
    size_t i = threadIdx.x + blockIdx.x * blockDim.x;
    y[i] = a * x[i] + y[i];
}

void saxpy_hip(size_t n, float a, float * x, float * y) {
    assert(n % 256 == 0);
    saxpy_kernel<<<n/256,256,0,NULL>>>(n, a, x, y);
}
```

We need a way to translate the host pointer that was mapped by OpenMP directives and retrieve the associated device pointer.
(1) OPENMP TO HIP: PUTTING IT TOGETHER

```c
__global__ void saxpy_kernel(size_t n, float a, float * x, float * y) {
    size_t i = threadIdx.x + blockIdx.x * blockDim.x;
    y[i] = a * x[i] + y[i];
}

void saxpy_hip(size_t n, float a, float * x, float * y) {
    assert(n % 256 == 0);
    saxpy_kernel<<<n/256,256,0,NULL>>>(n, a, x, y);
}

void example() {
    float a = 2.0;
    float * x = ...;   // assume: x = 0xabcd
    float * y = ...;

    // allocate the device memory
    #pragma omp target data map(to:x[0:count]) map(tofrom:y[0:count])
    {
        compute_1(n, x);  // mapping table: x:[0xabcd,0xef12], x = 0xabcd
        compute_2(n, y);
        #pragma omp target update to(x[0:count]) to(y[0:count])  // update x and y on the target
        #pragma omp target data use_device_ptr(x,y)
        {
            saxpy_hip(n, a, x, y) // mapping table: x:[0xabcd,0xef12], x = 0xef12
        }
    }
    compute_3(n, y);
}
```
(2) OPENMP TO HIP: FORTRAN AND DGEMM EXAMPLE

subroutine example
use rocml_interface
use iso_c_binding
implicit none
real(8), allocatable, target, dimension(:,:) :: a, b, c
type(c_ptr) :: rocblas_handle
...
allocate(da(M,N), db(N,K), dc(M,K))
call init_matrices(da, db, dc, M, N, K)
! Initialize matrices
call init_rocblas(rocblas_handle)
! Initialize rocBLAS
...
#ifdef C
$OMP target enter data map(to:a,b,c)
$OMP target data use_device_ptr(a,b,c)
call omp_dgemm(rocblas_handle, modea, modeb, M, N, K, alpha, &
c_loc(a), lda, c_loc(b), ldb, beta, c_loc(c), ldc)
$OMP end target data
$OMP target update from(c)
$OMP target exit data map(delete:a,b,c)
...
end subroutine example

... or build hipfort with flang and use their readily available FORTRAN to HIP interface
https://github.com/ROCmSoftwarePlatform/hipfort

include <rocblas.h>
extern "C" {
    void omp_dgemm(void *ptr, int modeA, int modeB, int m, int n,
                int k, double alpha, double *A, int lda,
                double *B, int ldb, double beta, double *C, int ldc) {
        rocblas_handle *handle = (rocblas_handle *) ptr;
        rocblas_dgemm(*handle, convert(modeA), convert(modeB), m, n, k,
                        &alpha, A, lda, B, ldb, &beta, C, ldc);
    }
    void init_rocblas(void *ptr) {
        rocblas_handle *handle = (rocblas_handle *) ptr;
        rocblas_create_handle(handle);
    }
}
void example() {
    HIPCALL(hipSetDevice(0));

    compute_1(n, x);
    compute_2(n, x);

    HIPCALL(hipMalloc(&x_dev, sizeof(*x_dev) * count));
    HIPCALL(hipMalloc(&y_dev, sizeof(*y_dev) * count));
    HIPCALL(hipMemcpy(x_dev, x, sizeof(*x) * count, hipMemcpyHostToDevice));
    HIPCALL(hipMemcpy(y_dev, y, sizeof(*y) * count, hipMemcpyHostToDevice));

    saxpy_omp(count, a, x_dev, y_dev);

    HIPCALL(hipMemcpy(y, y_dev, sizeof(*y) * count, hipMemcpyDeviceToHost));
    HIPCALL(hipFree(x_dev));
    HIPCALL(hipFree(y_dev));

    compute_3(n, y);
}

void saxpy_omp(size_t n, float a, float * x, float * y) {
    #pragma omp target teams distribute parallel for num_threads(256) num_teams(480)
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}

num_threads and num_teams optional. Default for MI100 is 256 x 480
## OPENMP OFFLOADING VS HIP: BABELSTREAM CASE STUDY

Full comparison of OpenMP Offloading vs HIP for all kernels in single precision and double precision

All experiments performed on a single Instinct MI100 using AOMP 13.0-6

Default Threads * Teams configuration already optimal for some kernels

<table>
<thead>
<tr>
<th>Single Precision</th>
<th>Default Threads * Teams</th>
<th>OpenMP/HIP ratio</th>
<th>Optimal Threads * Teams</th>
<th>Optimal OpenMP/HIP ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>256 * 480</td>
<td>1.48</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Write</td>
<td>256 * 480</td>
<td>1.96</td>
<td>1024 * 1440</td>
<td>2.05</td>
</tr>
<tr>
<td>Copy</td>
<td>256 * 480</td>
<td>0.92</td>
<td>128 * 1920</td>
<td>0.97</td>
</tr>
<tr>
<td>Mul</td>
<td>256 * 480</td>
<td>0.92</td>
<td>128 * 1440</td>
<td>0.97</td>
</tr>
<tr>
<td>Add</td>
<td>256 * 480</td>
<td>0.89</td>
<td>128 * 1680</td>
<td>0.93</td>
</tr>
<tr>
<td>Triad</td>
<td>256 * 480</td>
<td>0.88</td>
<td>128 * 1440</td>
<td>0.92</td>
</tr>
<tr>
<td>Dot</td>
<td>256 * 480</td>
<td>0.57</td>
<td>64 * 1920</td>
<td>0.72</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Double Precision</th>
<th>Default Threads * Teams</th>
<th>OpenMP/HIP ratio</th>
<th>Optimal Threads * Teams</th>
<th>Optimal OpenMP/HIP ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>256 * 480</td>
<td>1.01</td>
<td>1024 * 960</td>
<td>1.06</td>
</tr>
<tr>
<td>Write</td>
<td>256 * 480</td>
<td>0.90</td>
<td>1024 * 60</td>
<td>0.95</td>
</tr>
<tr>
<td>Copy</td>
<td>256 * 480</td>
<td>0.93</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mul</td>
<td>256 * 480</td>
<td>0.92</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Add</td>
<td>256 * 480</td>
<td>0.93</td>
<td>64 * 1440</td>
<td>0.94</td>
</tr>
<tr>
<td>Triad</td>
<td>256 * 480</td>
<td>0.92</td>
<td>64 * 1440</td>
<td>0.94</td>
</tr>
<tr>
<td>Dot</td>
<td>256 * 480</td>
<td>0.64</td>
<td>256 * 960</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Optimization for BabelStream would require a different number of Threads * Teams for each of the sub-benchmarks
REFERENCES

- **AOMP**: [https://github.com/ROCm-Developer-Tools/aomp](https://github.com/ROCm-Developer-Tools/aomp)
  - AMD open-source Clang/LLVM based compiler with support for OpenMP API


- **BabelStream**: [https://github.com/UoB-HPC/BabelStream](https://github.com/UoB-HPC/BabelStream)

- **HIPFORT**: [https://github.com/ROCmSoftwarePlatform/hipfort](https://github.com/ROCmSoftwarePlatform/hipfort)
  - Readily available FORTRAN interfaces to HIP/ROCm libraries
PART 2: PORTING MPAS TO SYSTEMS WITH AMD GPUs

The Model for Prediction Across Scales (MPAS) is a collaborative project for developing atmosphere, ocean and other earth-system simulation components for use in climate, regional climate, and weather studies.

- Finite volume solver for non-hydrostatic atmospheric equations.
- Written in FORTRAN. Uses directives for GPU acceleration
  - ~2.5k lines of !$acc code, still an ongoing effort
  - AMD approach: OpenMP directives

See https://mpas-dev.github.io/ and https://github.com/MPAS-Dev/MPAS-Model for more information
MPAS MEMORY AND DATA MANAGEMENT

• All GPU memory buffers allocated are the first time step and is reused for subsequent time steps.
• Updating the host from device occurs at the end of every time step.
• Now we can strictly focus on porting and optimizing the compute kernels.

Program mpas

use mpas_subdriver
use mpas_derived_types, only : core_type, domain_type
implicit none
type (core_type), pointer :: corelist => null()
type (domain_type), pointer :: domain => null()
call mpas_init(corelist, domain) // Allocate domain and host arrays
call mpas_run(domain)
call mpas_finalize(corelist, domain)
end program mpas

Subroutine mpas_run(domain)

do itimestep=1,Ntimestep

! Allocate GPU arrays

call mpas_pool_get_array_gpu(state,'theta_m',gpu_theta_m,1,1)
call mpas_pool_get_array_gpu(state,'scalars',gpu_scalars,1,1)
call mpas_pool_get_array_gpu(diag,'pressure_p',gpu_pressure_p)
call mpas_pool_get_array_gpu(diag,'rtheta_p',gpu_rtheta_p)
call mpas_pool_get_array_gpu(state,'w',gpu_w,2)
call mpas_pool_get_array_gpu(state,'u',gpu_u,2)
!
and more...
!
Compute...

call mpas_update_gpu_data_on_host(domain)
eddo

end subroutine

Subroutine mpas_update_gpu_data_on_host(domain)

! extract information from domain ...

call mpas_pool_get_array_gpu(state,'w',gpu_w,2)
call mpas_pool_get_array_gpu(state,'u',gpu_u,2)
!
and more...
!
Compute...

call mpas_update_gpu_data_on_host(domain)
eddo

end subroutine
The existing OpenACC code serves as a rough guideline for our OpenMP offloading port.

First step of the porting & optimization process is to add existing OpenMP directives on top of the OpenACC directives.
EXAMPLE #1: OPENMP INITIAL PORT

Note: number of vertical levels (nVertLevels) depends on mesh. (e.g., nVertLevels = 26 in the JW Baroclinic Wave benchmark)

This may be okay for a hardware with shorter SIMD (warp). With warp size exceeding the nVertLevels use of recourses will be suboptimal

How to ensure most threads are doing useful work for these smaller meshes? One approach could be to collapse the inner do loops
EXAMPLE #1: OPENMP OPTIMIZATION – COLLAPSED DO LOOPS

Macro-definitions added at the top of each source file to distinguish do loops for the OpenACC backend from do loops for the new OpenMP offloading backend.

End goal is to have one code with few adaptations for optimal use of different directive-based programming models.

```c
#pragma omp target teams distribute collapse(2)
#pragma acc parallel vector_length(32)
#pragma acc loop gang
do iEdge=edgeStart,edgeEnd
  GPUOMP do k=1,nVertLevels
    cell1 = cellsOnEdge(1,iEdge)
    cell2 = cellsOnEdge(2,iEdge)
    if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then
      // Update edges for block-owned cells
      // End goal is to have one code with few adaptations for optimal use of different directive-based programming models
  %omp parallel do simd
  vDxEdge = (vDx(k,iEdge) + vDx(k,1Edge)) / 2.0
  vDyEdge = (vDy(k,iEdge) + vDy(k,1Edge)) / 2.0
  vDzEdge = (vDz(k,iEdge) + vDz(k,1Edge)) / 2.0
  vDwEdge = (vDw(k,iEdge) + vDw(k,1Edge)) / 2.0
  if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then
    // Update edges for block-owned cells
    // End goal is to have one code with few adaptations for optimal use of different directive-based programming models
  endif
  endif
enddo
```

OpenMP on AMD GPUs
EXAMPLE #2: OPENACC CODE

Caches local arrays into shared memory
No OpenMP equivalent for !$acc cache

Collapsing inner loops not always possible
EXAMPLE #2: OPENMP INITIAL PORT

Suboptimal performance of RHS loop – register spills and scratch usage according to rocprof.
Collapsing the loops not possible because of the reduction of q2 variable.
However, can we rearrange the order of the parallel and sequential loops?
EXAMPLE #2: OPENMP OPTIMIZATION – REARRANGING AND SPLITTING LOOPS

Rearrange order of the sequential and parallel do loops. Multiple parallel loops to minimize global memory reads/writes.
## BENCHMARK: JW BAROCLINIC WAVE – PERFORMANCE ON 1 MI250X GPU

<table>
<thead>
<tr>
<th>Events</th>
<th>Initial port (secs)</th>
<th>Initial optimization (secs)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>time integration</td>
<td>334.99</td>
<td>92.94</td>
<td>3.60</td>
</tr>
<tr>
<td>atm_rk_integration_setup</td>
<td>1.95</td>
<td>0.54</td>
<td>3.61</td>
</tr>
<tr>
<td>atm_compute_moist_coefficients</td>
<td>20.28</td>
<td>0.63</td>
<td>32.19</td>
</tr>
<tr>
<td>physics_get_tend</td>
<td>0.57</td>
<td>0.4</td>
<td>1.41</td>
</tr>
<tr>
<td>atm_compute_vert_imp_coefs</td>
<td>4.03</td>
<td>1.68</td>
<td>2.40</td>
</tr>
<tr>
<td>atm_compute_dyn_tend</td>
<td>128.90</td>
<td>20.68</td>
<td>6.23</td>
</tr>
<tr>
<td>small_step_prep</td>
<td>2.83</td>
<td>1.86</td>
<td>1.52</td>
</tr>
<tr>
<td>atm_advance_acoustic_step</td>
<td>113.55</td>
<td>31.23</td>
<td>3.64</td>
</tr>
<tr>
<td>atm_divergence_damping_3d</td>
<td>8.97</td>
<td>3.1</td>
<td>2.89</td>
</tr>
<tr>
<td>atm_recover_large_step_variables</td>
<td>8.68</td>
<td>4.76</td>
<td>1.82</td>
</tr>
<tr>
<td>atm_compute_solve_diagnostics</td>
<td>13.66</td>
<td>6.53</td>
<td>2.09</td>
</tr>
<tr>
<td>atm_rk_dynamics_substep_finish</td>
<td>0.66</td>
<td>0.49</td>
<td>1.35</td>
</tr>
<tr>
<td>atm_rk_reconstruct</td>
<td>2.18</td>
<td>0.86</td>
<td>2.54</td>
</tr>
<tr>
<td>atm_rk_summary</td>
<td>2.78</td>
<td>2.77</td>
<td>1.00</td>
</tr>
<tr>
<td>mpass update GPU data on host</td>
<td>6.46</td>
<td>6.33</td>
<td>1.02</td>
</tr>
</tbody>
</table>

### Notes:
1. Overall GPU port (including the OpenACC backend) still in progress
2. Only a couple variables copied back to the host – about ~7% of time integration
   - The “mpass update GPU data on host” event will significantly increase as more physics/variables are ported
   - HMM can play a big role
PART 3: HETEROGENEOUS MEMORY MANAGEMENT (HMM)

HMM allows the same pointer to an object to be used both by the CPU and a device [GPU] even if the physical location of the object were moved by the operating system or device driver. Furthermore, the device driver can control the policy of whether the current physical location of the object is in CPU or device memory.

OPENMP PROGRAMMING ON SYSTEMS WITH HMM

```c
#pragma omp requires unified_shared_memory

int main(){
    double * X, * Y, *Z;
    size_t N = (size_t) 1024*1024*1024/sizeof(double);
    X = new double[N];
    Y = new double[N];

    #pragma omp target teams distribute parallel for if(target:N > 2000)
    for (size_t i = 0; i < N; ++i)
        X[i] = 0.000001*i;

    #pragma omp target teams distribute parallel for if(target:N > 2000)
    for (size_t i = 0; i < N; ++i)
        Y[i] = X[i]

    delete[] X; delete[] Y;
    return 0;
}
```

Highlights:

1. Uses system memory allocators.

2. “Pointer is a pointer” data can be accessed by threads running on any device, regardless of the current physical location of the data.

3. HMM allows OS, driver, and HW to manage physical memory location, while OpenMP directives are used primarily for expressing parallelism and execution space (HOST, DEVICE 0, DEVICE 1, etc.).

4. Footnotes: manual management of data, memory location, and expression of parallelism (for example using HIP programming models) may provide higher performance. Some performance optimizations may also be done via using additional directives, clauses, and APIs.
PERFORMANCE COMPARISON OF UNIFIED VS. NON UNIFIED MEMORY

**With UNIFIED MEMORY**

GPU par : dot loop time = 0.00224113 [s] effective read BW = 0.871489 [GB/s]

**NO UNIFIED MEMORY**

GPU par : read loop time = 4.1008e-05 [s] effective read BW = 47.6279 [GB/s]
The default memory alignment obtained with “new” is 16 bytes. Such alignment is not optimal for computing on GPUs.

C++ offers ways to specify memory alignment using default parameter set at the compilation time (-faligned-allocation -fnew-alignment=64) or at run time as shown in the example. Use system memory allocators such as posix_memalign is also an alternative.

<table>
<thead>
<tr>
<th>Alignment</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP: thread_limit(128)</td>
<td>540 GB/s</td>
<td>750 GB/s</td>
<td>750 GB/s</td>
<td>680 GB/s</td>
<td>870 GB/s</td>
<td>900 GB/s</td>
</tr>
<tr>
<td>OpenMP: thread_limit(1024)</td>
<td>990 GB/s</td>
<td>1000 GB/s</td>
<td>1010 GB/s</td>
<td>960 GB/s</td>
<td>1040 GB/s</td>
<td>1040 GB/s</td>
</tr>
<tr>
<td>HIP (blockDim 128-1024)</td>
<td>750 GB/s</td>
<td>1212 GB/s</td>
<td>1220 GB/s</td>
<td>1220 GB/s</td>
<td>1239 GB/s</td>
<td>1240 GB/s</td>
</tr>
</tbody>
</table>
ONE STEP FURTHER TO ADVANCE PORTABILITY:
PORTABLE TO SYSTEMS WITH AND WITHOUT UNIFIED MEMORY

```c
int main(){
  double * X, * Y, *Z;
  size_t N = (size_t) 1024*1024*1024/sizeof(double);
  X = new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];
  Y = new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];
  #pragma omp target enter data map(alloc:X[0:N], Y[0:N])
  #pragma omp target teams distribute parallel for if(target:N>2000)
    map(from:X[0:N])
    for (size_t i = 0; i < N; ++i)
      X[i] = 0.000001*i;
  #pragma omp target teams distribute parallel for if(target:N>2000)
    map(to:X[0:N]) map(from:Y[0:N])
    for (size_t i = 0; i < N; ++i)
      Y[i] = X[i];
  #pragma omp target exit data map(release:X[0:N], Y[0:N])
  delete[] X;  delete[] Y;
  return 0;
}
```

```c
#pragma omp requires unified_shared_memory
int main(){
  double * X, * Y, *Z;
  size_t N = (size_t) 1024*1024*1024/sizeof(double);
  X = new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];
  Y = new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];
  #pragma omp target enter data map(alloc:X[0:N], Y[0:N])
  #pragma omp target teams distribute parallel for if(target:N>2000)
    map(from:X[0:N])
    for (size_t i = 0; i < N; ++i)
      X[i] = 0.000001*i;
  #pragma omp target teams distribute parallel for if(target:N>2000)
    map(to:X[0:N]) map(from:Y[0:N])
    for (size_t i = 0; i < N; ++i)
      Y[i] = X[i];
  #pragma omp target exit data map(release:X[0:N], Y[0:N])
  delete[] X;  delete[] Y;
  return 0;
}
```
ADDITIONAL OPTIONS FOR MEMORY MANAGEMENT

```c
#include <hip/hip_runtime.h>
#include <omp.h>

#pragma omp requires unified_shared_memory
int main(){

double * X, * Y, *Z;
size_t N = (size_t) 1024*1024*1024/sizeof(double);
X = new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];
Y = new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];

hipMemPrefetchAsync(X,nbytes,omp_get_default_device());
hipMemAdvise((void*) X,sizeof(double)*N,hipMemAdviseSetPreferredLocation,omp_get_default_device());
hipMemAdvise((void*) Y,sizeof(double)*N,hipMemAdviseSetPreferredLocation,omp_get_default_device());

#pragma omp target teams distribute parallel for if(target:N>2000)
for (size_t i = 0; i < N; ++i)
    X[i] = 0.000001*i;

#pragma omp target teams distribute parallel for if(target:N>2000)
for (size_t i = 0; i < N; ++i)
    Y[i] = X[i];

delete[] X; delete[] Y;
return 0;
}
```

Adding a proper API for prefetching buffers to HOST or DEVICE memory can increase performance.
IMPACT OF C++ MEMORY ALLOCATORS ON PERFORMANCE

X = new double[N];  Y = new double[N];

hipcc -fopenmp -O3 test_omp_CPU_GPU_HIP.cpp

Output:
CPU memory initialization: loop time  = 0.0454931 BW = 65.944[GB/s]  
X = 0x7f47fffff010 __STDCPP_DEFAULT_NEW_ALIGNMENT__ = 16, MOD(X,16) = 0  
Y = 0x7f47fffff010 __STDCPP_DEFAULT_NEW_ALIGNMENT__ = 16, MOD(Y,16) = 0  
using device ID: 0  
GPU: loop time  = 0.00305295 BW = 655.104[GB/s]  
GPU: loop time  = 0.00266623 BW = 750.121[GB/s]

X = new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];  
Y = new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];

hipcc -faliged-allocation -fnew-alignment=256 -fopenmp -O3 test_omp_CPU_GPU_HIP.cpp

Output:
CPU memory initialization: loop time  = 0.04548 BW = 65.963[GB/s]  
CPU memory initialization: loop time  = 0.0455599 BW = 65.8474[GB/s]  
X = 0x7fbffffff100 __STDCPP_DEFAULT_NEW_ALIGNMENT__ = 256, MOD(X,256) = 0  
Y = 0x7fbffffff100 __STDCPP_DEFAULT_NEW_ALIGNMENT__ = 256, MOD(Y,256) = 0  
using device ID: 0  
GPU: loop time  = 0.00218606 BW = 914.888[GB/s]  
GPU: loop time  = 0.00161409 BW = 1239.09[GB/s]

Alignment → 8 16 32 64 128 256 512
| blockDim.x=256 | 733 GB/s | 733 GB/s | 1220 GB/s | 1215 GB/s | 1220 GB/s | 1248 GB/s | 1240 GB/s |
| blockDim.x=128 | 751 GB/s | 750 GB/s | 1212 GB/s | 1220 GB/s | 1220 GB/s | 1239 GB/s | 1240 GB/s |
| blockDim.x=64  | 738 GB/s | 738 GB/s | 740 GB/s  | 740 GB/s  | 740 GB/s  | 740 GB/s  | 740 GB/s  |

27 OpenMP on AMD GPUs
SUMMARY

- OpenMP offloading is compatible and competitive with HIP
- OpenMP can interface to ROCm/HIP math libraries
- Performance of OpenMP regions can be tuned by modifying the number of teams or threads
- OpenACC applications like MPAS can be ported to OpenMP to run on AMD GPUs
- Performance tuning of application code may benefit from minimal modification to the source code
- Heterogeneous Memory Management (HMM) is available to use on AMD systems
Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© 2022 Advanced Micro Devices, Inc and OpenMP® Architecture Review Board. All rights reserved.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.