Directive based GPU programming on AMD systems

Justin Chang, Member of Technical Staff
Leopold Grinberg, Principal Member of Technical Staff
Bob Robey, Principal Member of Technical Staff

[Public]

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

AMD

[Public]

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

AMD

[Public]

PART 1: HIP & OPENMP – HYBRID PROGRAMMING

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

AMD

[Public]

(1) OPENMP TO HIP: SAXPY EXAMPLE

```
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
        saxpy(n, a, x, y)
        compute_3(n, y);
    }
}
```

Allocate device memory for x and y, and specify directions of data transfers

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

```
void saxpy(size_t n, float a,
           float * x, float * y) {
#pragma omp target teams distribute \
                    parallel for …
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```


AMD

[Public]

(1) OPENMP TO HIP: HIP KERNEL FOR SAXPY()

A HIP version of the SAXPY kernel:

```
__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);
}
```

These are device pointers!

We need a way to translate the host pointer that was mapped by OpenMP directives and retrieve the associated device pointer.

AMD

[Public]

(1) OPENMP TO HIP: PUTTING IT TOGETHER

```
__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);
}
```

Translation unit 1

hipcc

```
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);
}
```

Translation unit 2

clang/cc

AMD

[Public]

(2) OPENMP TO HIP: FORTRAN AND DGEMM EXAMPLE

You can either create your own FORTRAN to HIP interface…

```
subroutine example
    use rocm_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
    ...

    !$OMP target enter data map(to:a,b,c)
    !$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
```

```
module rocm_interface
    interface
        subroutine init_rocblas(handle) bind(C)
            use iso_c_binding
            type(c_ptr)        :: handle
        end subroutine init_rocblas
        subroutine omp_dgemm(handle,ma,mb,m,n,k,alpha, &
            a,lda,b,ldb,beta,c,ldc) bind(C)
            use iso_c_binding
            type(c_ptr),value  :: a,b,c
            type(c_ptr)        :: handle
            integer(c_int)     :: ma,mb,m,n,k,lda,ldb,ldc
            real(c_double)     :: alpha,beta
        end subroutine omp_dgemm
    end interface
end module rocm_interface
```

Translation unit 1

ftn

```
#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);
    }
}
```

Translation unit 2

hipcc

… or build hipfort and use their readily available FORTRAN to HIP interface

https://github.com/ROCmSoftwarePlatform/hipfort

AMD

[Public]

(3) HIP TO OPENMP: BUFFER MANAGEMENT

```
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) * c
    HIPCALL(hipMemcpy(y_dev, y, sizeof(*y) * c

    saxpy_omp(count, a, x_dev, y_dev);

    HIPCALL(hipMemcpy(y, y_dev, sizeof(*y) * c
    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

AMD

[Public]

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

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

Single Precision	Default Threads * Teams	OpenMP/HIP ratio	Optimal Threads * Teams	Optimal OpenMP/HIP ratio
Read	256 * 480	1.48	-	-
Write	256 * 480	1.96	1024 * 1440	2.05
Copy	256 * 480	0.92	128 * 1920	0.97
Mul	256 * 480	0.92	128 * 1440	0.97
Add	256 * 480	0.89	128 * 1680	0.93
Triad	256 * 480	0.88	128 * 1440	0.92
Dot	256 * 480	0.57	64 * 1920	0.72
Double Precision	**Default Threads * Teams**	**OpenMP/HIP ratio**	**Optimal Threads * Teams**	**Optimal OpenMP/HIP ratio**
Read	256 * 480	1.01	1024 * 960	1.06
Write	256 * 480	0.90	1024 * 60	0.95
Copy	256 * 480	0.93	-	-
Mul	256 * 480	0.92	-	-
Add	256 * 480	0.93	64 * 1440	0.94
Triad	256 * 480	0.92	64 * 1440	0.94
Dot	256 * 480	0.64	256 * 960	0.76

AMD

[Public]

REFERENCES

- 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
- HIPFORT: https://github.com/ROCmSoftwarePlatform/hipfort
  - Readily available FORTRAN interfaces to HIP/ROCm libraries

AMD

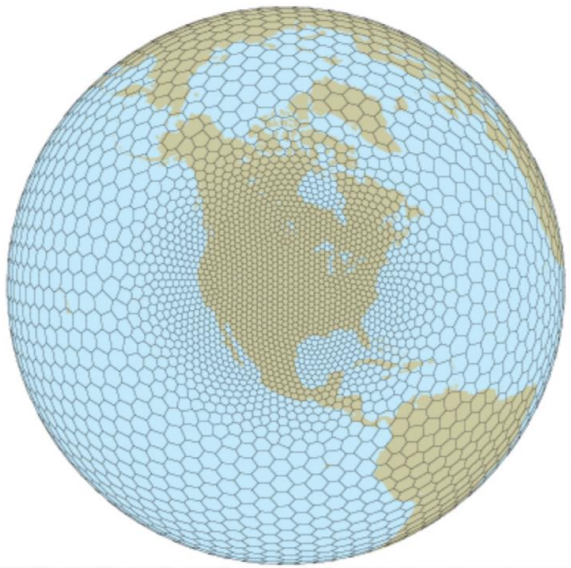
[Public]

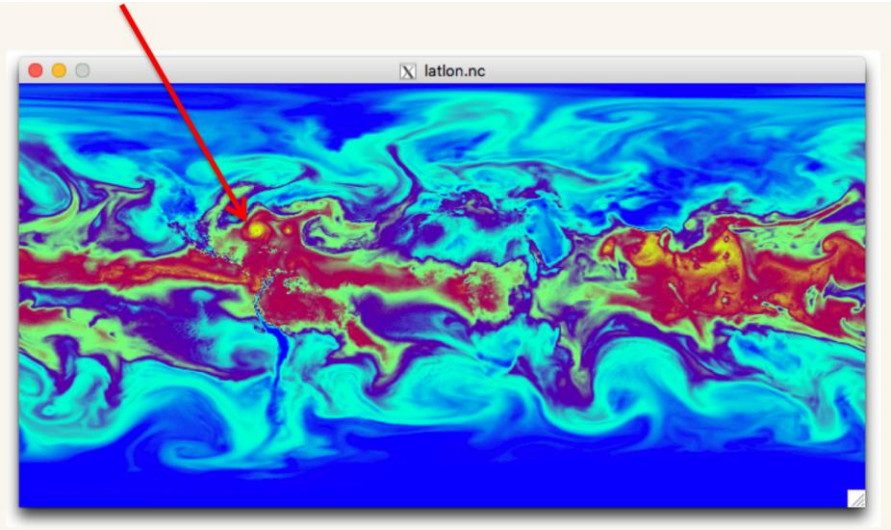
PART 2: PORTING MPAS TO SYSTEMS WITH AMD GPUS

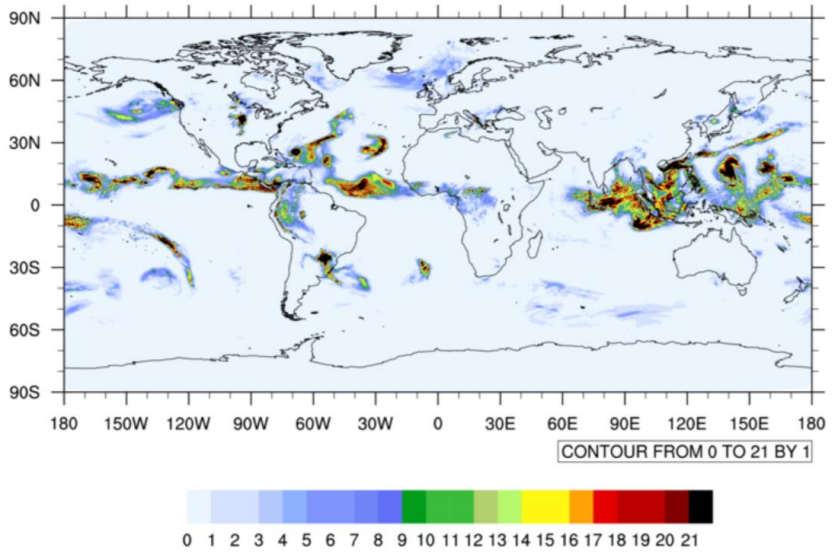
The Model for Prediction Across Scales (MPAS) is a collaborative project for developing atmosphere, ocean and other earth-system simulation components for use in climate, regional climate, and weather studies.

- 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







Above: The accumulated total precipitation between 2016-10-14 00 UTC and 2016-10-15 00 UTC on from MPAS with the 'mesoscale_reference' physics suite.

AMD

[Public]

MPAS MEMORY AND DATA MANAGEMENT

```
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_pool_get_array_2d_real_gpu(inPool, key, array, timeLevel)!{{{
    implicit none
    type (mpas_pool_type), intent(in) :: inPool
    character (len=*), intent(in) :: key
    real (kind=RKIND), dimension(:,:), pointer :: array
    integer, intent(in), optional :: timeLevel
    type (field2DReal), pointer :: field
    type (mpas_coupler_type), pointer :: coupler
    nullify(field)
    call mpas_pool_get_field_2d_real(inPool, key, field, timeLevel)
    nullify(array)
    if (associated(field)) then
        coupler => field%block%domain%mpas_cpl
        array => field % array
#ifdef CORE_ATMOSPHERE
        if (coupler % pole_includes(ROLE_INTEGRATE)) then
            !$acc enter data copyin(field%array)
        end if
#endif
    end if
end subroutine mpas_pool_get_array_2d_real_gpu!}}}
```

- 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.

```
subroutine mpas_run(domain)
    ! extract information from 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, 2)
        call mpas_pool_get_array_gpu(state, 'u',gpu_u_2, 2)
        ! and more...

        ! Compute...

        call mpas_update_gpu_data_on_host(domain)
    enddo
end subroutine
```

```
subroutine mpas_update_gpu_data_on_host(domain)
    ! extract information from domain ...
    call mpas_pool_get_array_gpu(state, 'w', w, 2)
    call mpas_pool_get_array_gpu(state, 'u', u, 2)
    !$acc update host(w, u)

end subroutine mpas_update_gpu_data_on_host
```


AMD

[Public]

EXAMPLE #1: OPENACC CODE

```
!$acc parallel vector_length(32)
!$acc loop gang
do iEdge=edgeStart,edgeEnd
    cell1 = cellsOnEdge(1,iEdge)
    cell2 = cellsOnEdge(2,iEdge)
    ! update edges for block-owned cells
    if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then
!DIR$ IVDEP
!$acc loop vector
        do k=1,nVertLevels
            pgrad = ((rtheta_pp(k,cell2)-rtheta_pp(k,cell1))*invDcEdge(iEdge) )/(.5*(zz(k,cell2)+zz(k,cell1)))
            pgrad = cqu(k,iEdge)*0.5*c2*(exner(k,cell1)+exner(k,cell2))*pgrad
            pgrad = pgrad + 0.5*zxu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2))
            ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad)
        end do
        ! accumulate ru_p for use later in scalar transport
!DIR$ IVDEP
!$acc loop vector
        do k=1,nVertLevels
            ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
        end do
    end if ! end test for block-owned cells
end do ! end loop over edges
!$acc end parallel
```

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

AMD

[Public]

EXAMPLE #1: OPENMP INITIAL PORT

```
!$omp target teams distribute
!$acc parallel vector_length(32)
!$acc loop gang
do iEdge=edgeStart,edgeEnd
    cell1 = cellsOnEdge(1,iEdge)
    cell2 = cellsOnEdge(2,iEdge)
    ! update edges for block-owned cells
    if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then
!$omp parallel do simd
!DIR$ IVDEP
!$acc loop vector
        do k=1,nVertLevels
            pgrad = ((rtheta_pp(k,cell2)-rtheta_pp(k,cell1))*invDcEdge(iEdge) )/(.5*(zz(k,cell2)+zz(k,cell1)))
            pgrad = cqu(k,iEdge)*0.5*c2*(exner(k,cell1)+exner(k,cell2))*pgrad
            pgrad = pgrad + 0.5*zxu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2))
            ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad)
        end do
!$omp end parallel do simd
        ! accumulate ru_p for use later in scalar transport
!$omp parallel do simd
!DIR$ IVDEP
!$acc loop vector
        do k=1,nVertLevels
            ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
        end do
!$omp end parallel do simd
    end if ! end test for block-owned cells
end do ! end loop over edges
!$acc end parallel
!$omp end target teams distribute
```

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

AMD

[Public]

EXAMPLE #1: OPENMP OPTIMIZATION – COLLAPSED DO LOOPS

```
#ifdef _OPENACC
#define GPUACC
#define GPUOMP !
#else
#define GPUACC !
#define GPUOMP
#endif
```

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

```
!$omp target teams distribute collapse(2)
!$acc parallel vector_length(32)
!$acc loop gang
do iEdge=edgeStart,edgeEn
GPUOMP do k=1,nVertLevels
    cell1 = cellsOnEdge(1,iEdge)
    cell2 = cellsOnEdge(2,iEdge)
    ! update edges for block-owned cells
    if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then
!$omp parallel do simd
!DIR$ IVDEP
!$acc loop vector
GPUACC  do k=1,nVertLevels
            pgrad = ((rtheta_pp(k,cell2)-rtheta_pp(k,cell1))*invDcEdge(iEdge) )/(.5*(zz(k,cell2)+zz(k,cell1)))
            pgrad = cqu(k,iEdge)*0.5*c2*(exner(k,cell1)+exner(k,cell2))*pgrad
            pgrad = pgrad + 0.5*zxu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2))
            ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad)
GPUOMP      ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
GPUACC  end do
!$omp end parallel do simd
        ! accumulate ru_p for use later in scalar transport
!$omp parallel do simd
!DIR$ IVDEP
!$acc loop vector
GPUACC  do k=1,nVertLevels
GPUACC      ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
GPUACC  end do
!$omp end parallel do simd
    end if ! end test for block-owned cells
GPUOMP end do
end do ! end loop over edges
!$acc end parallel
!$omp end target teams distribute
```


AMD

[Public]

EXAMPLE #2: OPENACC CODE

```
1   !$acc parallel vector_length(64)
2   !$acc loop gang private(wduz, tend_wk, eoe_w, we_w)
3   do iEdge=edgeSolveStart,edgeSolveEnd
4   !$acc cache(tend_wk,wduz,eoe_w,we_w)
5       cell1 = cellsOnEdge(1,iEdge)
6       cell2 = cellsOnEdge(2,iEdge)
7   !DIR$ IVDEP
8   !$acc loop vector
9       do k=1,nVertLevels
10          ! compute ...
11      end do
12      ! Compute ...
13  !$acc loop vector shortloop
14      do j = 1,nEdgesOnEdge(iEdge)
15          eoe_w(j) = edgesOnEdge(j,iEdge)
16          we_w(j)  = weightsOnEdge(j,iEdge)
17      end do
```

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

```
18  !DIR$ IVDEP
19  !$acc loop vector
20      do k=1,nVertLevels
21          q1 = pv_edge(k,iEdge)
22          q2 = 0.0
23  !$acc loop seq
24          do j = 1,nEdgesOnEdge(iEdge)
25              eoe = eoe_w(j)
26              workpv = 0.5 * (q1 + pv_edge(k,eoe))
27              q2 = q2 + we_w(j) * u(k,eoe) * workpv
28          end do
29          t_w = - rdzw(k)*(wduz(k+1)-wduz(k))
30          tend_u(k,iEdge) = t_w + rho_edge(k,iEdge) * &
31                                  (q2 - (ke(k,cell2) - ke(k,cell1)) *  &
32                                  invDcEdge(iEdge)) - tend_wk(k) * 0.5 * &
33                                  (h_divergence(k,cell1)+h_divergence(k,cell2))
34          end do
35      end do
36  !$acc end parallel
```

Collapsing inner loops not always possible

AMD

[Public]

EXAMPLE #2: OPENMP INITIAL PORT

```
!$omp target teams distribute parallel do thread_limit(64)
!$acc parallel vector_length(64)
!$acc loop gang private(wduz, tend_wk, eoe_w, we_w)
do iEdge=edgeSolveStart,edgeSolveEnd
!$acc cache(tend_wk,wduz,eoe_w,we_w)
    cell1 = cellsOnEdge(1,iEdge)
    cell2 = cellsOnEdge(2,iEdge)
!$omp parallel do simd
!DIR$ IVDEP
!$acc loop vector
    do k=1,nVertLevels
        ! compute ...
    end do
!$omp end parallel do simd
    ! Compute ...
!$omp parallel do simd
!$acc loop vector shortloop
    do j = 1,nEdgesOnEdge(iEdge)
        eoe_w(j) = edgesOnEdge(j,iEdge)
        we_w(j)  = weightsOnEdge(j,iEdge)
    end do
!$omp end parallel do simd
```

```
!DIR$ IVDEP
!$omp parallel do simd
!$acc loop vector
    do k=1,nVertLevels
        q1 = pv_edge(k,iEdge)
        q2 = 0.0
!$acc loop seq
        do j = 1,nEdgesOnEdge(iEdge)
            eoe = eoe_w(j)
            workpv = 0.5 * (q1 + pv_edge(k,eoe))
            q2 = q2 + we_w(j) * u(k,eoe) * workpv
        end do
        t_w = - rdzw(k)*(wduz(k+1)-wduz(k))
        tend_u(k,iEdge) = t_w + rho_edge(k,iEdge) * &
                          (q2 - (ke(k,cell2) - ke(k,cell1)) *  &
                          invDcEdge(iEdge)) - tend_wk(k) * 0.5 * &
                          (h_divergence(k,cell1)+h_divergence(k,cell2))
    end do
!$omp end parallel do simd
end do
!$acc end parallel
!$omp end target teams distribute
```

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?

AMD

[Public]

EXAMPLE #2: OPENMP OPTIMIZATION – REARRANGING AND SPLITTING LOOPS

```
!$omp target teams distribute parallel do thread_limit(64)
!$acc parallel vector_length(64)
!$acc loop gang private(wduz, tend_wk, eoe_w, we_w)
do iEdge=edgeSolveStart,edgeSolveEnd
!$acc cache(tend_wk,wduz,eoe_w,we_w)
    cell1 = cellsOnEdge(1,iEdge)
    cell2 = cellsOnEdge(2,iEdge)
!$omp parallel do simd
!DIR$ IVDEP
!$acc loop vector
    do k=1,nVertLevels
        ! compute ...
    end do
!$omp end parallel do simd
    ! Compute ...
!$omp parallel do simd
!$acc loop vector shortloop
    do j = 1,nEdgesOnEdge(iEdge)
        eoe_w(j) = edgesOnEdge(j,iEdge)
        we_w(j)  = weightsOnEdge(j,iEdge)
    end do
!$omp end parallel do simd
```

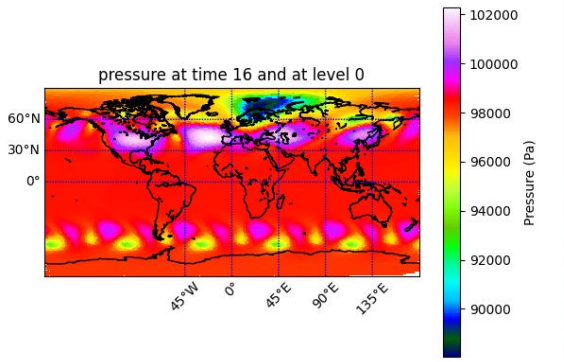
```
#ifdef _OPENACC
! ...
#else
!$omp parallel do simd
    do k=1,nVertLevels
        t_w = - rdzw(k)*(wduz(k+1)-wduz(k))
        tend_u(k,iEdge) = t_w + rho_edge(k,iEdge) * &
                          (-(ke(k,cell2) - ke(k,cell1))* invDcEdge(iEdge)) &
                          - tend_wk(k) * 0.5 * (h_divergence(k,cell1) &
                          + h_divergence(k,cell2))
    end do
!$omp end parallel do simd
    do j = 1,nEdgesOnEdge(iEdge)
        eoe = eoe_w(j)
!$omp parallel do simd
        do k=1,nVertLevels
            tend_u(k,iEdge) = tend_u(k,iEdge) + rho_edge(k,iEdge) * we_w(j) &
                              * u(k,eoe) * 0.5 * (pv_edge(k,iEdge) + pv_edge(k,eoe))
        end do
!$omp end parallel do simd
    end do
#endif
end do
!$acc end parallel
!$omp end target teams distribute
```

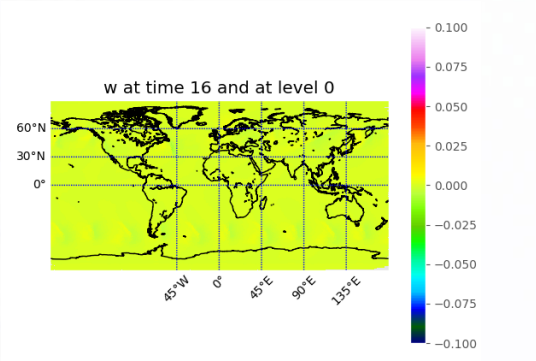
Rearrange order of the sequential and parallel do loops. Multiple parallel loops to minimize global memory reads/writes

AMD

[Public]

BENCHMARK: JW BAROCLINIC WAVE – PERFORMANCE ON 1 MI250X GPU





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

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 “mpas update GPU data on host” event will significantly increase as more physics/variables are ported
   - HMM can play a big role

AMD

[Public]

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.

https://www.kernel.org/doc/html/v5.0/vm/hmm.html

AMD

[Public]

OPENMP PROGRAMMING ON SYSTEMS WITH HMM

```
#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

AMD

[Public]

PERFORMANCE COMPARISON OF UNIFIED VS. NON UNIFIED MEMORY

With UNIFIED MEMORY

```
DEVID: 0 SGN:2 ConstWGSize:1024 args: 5 teamsXthrds:( 128X1024) reqd:( 128X1024) lds_usage:16716B sgpr_count:60 vgpr_count:58 sgpr_spill_count:0 vgpr_spill_count:0 tripcount:131072 rpc:0
n:__omp_offloading_10304_ac24ca_main_l50
Call __tgt_rtl_run_target_team_region_async:     10us         0 (         0, 0x00000203cf40, 0x00000203bd20, 0x00000203bd70,         5,         128,         1024,         131072, 0x7ffd4b4b40d0)
Call          __tgt_rtl_synchronize:    2207us         0 (         0, 0x7ffd4b4b40d0)
GPU par : dot loop time = 0.00224113 [s] effective read BW = 0.871489 [GB/s]

DEVID: 0 SGN:2 ConstWGSize:1024 args: 4 teamsXthrds:( 128X1024) reqd:( 128X1024) lds_usage:16452B sgpr_count:24 vgpr_count:44 sgpr_spill_count:0 vgpr_spill_count:0 tripcount:131072 rpc:0
n:__omp_offloading_10304_ac24ca_main_l57
Call __tgt_rtl_run_target_team_region_async:      7us         0 (         0, 0x00000203d1b0, 0x00000203d170, 0x00000203a940,         4,         128,         1024,         131072, 0x7ffd4b4b40d0)
Call          __tgt_rtl_synchronize:      18us         0 (         0, 0x7ffd4b4b40d0)
GPU par : read loop time = 4.1008e-05 [s] effective read BW = 47.6279 [GB/s]
```

NO UNIFIED MEMORY

```
Call          __tgt_rtl_data_alloc:       0us 0x7ff450408000 (         0,         8, 0x7ffe2056e990)
Call     __tgt_rtl_data_submit_async:      42us         0 (         0, 0x7ff450408000, 0x7ffe2056e990,         8, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_alloc:       0us 0x7ff450409000 (         0,         1048576, 0x0000025f7660)
Call     __tgt_rtl_data_submit_async:      62us         0 (         0, 0x7ff450409000, 0x0000025f7660,         1048576, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_alloc:       1us 0x7ff440200000 (         0,         1048576, 0x0000026f7670)
Call     __tgt_rtl_data_submit_async:      84us         0 (         0, 0x7ff440200000, 0x0000026f7670,         1048576, 0x7ffe2056e8c0)
DEVID: 0 SGN:2 ConstWGSize:1024 args: 5 teamsXthrds:( 128X1024) reqd:( 128X1024) lds_usage:16716B sgpr_count:60 vgpr_count:58 sgpr_spill_count:0 vgpr_spill_count:0 tripcount:131072 rpc:0
n:__omp_offloading_10304_ac24ca_main_l50
Call __tgt_rtl_run_target_team_region_async:     13us         0 (         0, 0x00000285bf10, 0x0000028816c0, 0x000002881710,         5,         128,         1024,         131072, 0x7ffe2056e8c0)
Call     __tgt_rtl_data_retrieve_async:     517us         0 (         0, 0x7ffe2056e990, 0x7ff450408000,         8, 0x7ffe2056e8c0)
Call          __tgt_rtl_synchronize:     501us         0 (         0, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_delete:       1us         0 (         0, 0x7ff440200000)
Call          __tgt_rtl_data_delete:       0us         0 (         0, 0x7ff450409000)
Call          __tgt_rtl_data_delete:       0us         0 (         0, 0x7ff450408000)
GPU par : dot loop time = 0.00127697 [s] effective read BW = 1.5295 [GB/s]
Call          __tgt_rtl_data_alloc:       0us 0x7ff450408000 (         0,         1048576, 0x0000025f7660)
Call     __tgt_rtl_data_submit_async:      39us         0 (         0, 0x7ff450408000, 0x0000025f7660,         1048576, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_alloc:       0us 0x7ff440200000 (         0,         1048576, 0x0000026f7670)
Call     __tgt_rtl_data_submit_async:      38us         0 (         0, 0x7ff440200000, 0x0000026f7670,         1048576, 0x7ffe2056e8c0)
DEVID: 0 SGN:2 ConstWGSize:1024 args: 4 teamsXthrds:( 128X1024) reqd:( 128X1024) lds_usage:16452B sgpr_count:24 vgpr_count:44 sgpr_spill_count:0 vgpr_spill_count:0 tripcount:131072 rpc:0
n:__omp_offloading_10304_ac24ca_main_l57
Call __tgt_rtl_run_target_team_region_async:      8us         0 (         0, 0x00000285c180, 0x00000285ad20, 0x00000285ca60,         4,         128,         1024,         131072, 0x7ffe2056e8c0)
Call          __tgt_rtl_synchronize:     400us         0 (         0, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_delete:       1us         0 (         0, 0x7ff440200000)
Call          __tgt_rtl_data_delete:       0us         0 (         0, 0x7ff450408000)
GPU par : read loop time = 0.00051403 [s] effective read BW = 3.79963 [GB/s]
```


AMD

[Public]

ONE STEP FURTHER TO ADVANCE PERFORMANCE: MEMORY ALIGNMENT

```
#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];
  X =  new (std::align_val_t(__STDCPP_DEFAULT_NEW_ALIGNMENT__)) double[N];
  if (N < 10)   Y =  new (std::align_val_t(16)) double[N];
  else          Y =  new (std::align_val_t(128)) 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;
}
```

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 possix_memalign is also an alternative

Alignment →	16	32	64	128	256	512
OpenMP: thread_limit(128)	540 GB/s	750 GB/s	750 GB/s	680 GB/s	870 GB/s	900 GB/s
OpenMP: thread_limit(1024)	990 GB/s	1000 GB/s	1010 GB/s	960 GB/s	1040 GB/s	1040 GB/s
HIP (blockDim 128-1024)	750 GB/s	1212 GB/s	1220 GB/s	1220 GB/s	1239 GB/s	1240 GB/s

AMD

[Public]

ONE STEP FURTHER TO ADVANCE PORTABILITY: PORTABLE TO SYSTEMS WITH AND WITHOUT UNIFIED 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;
}
```

```
#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;
}
```


AMD

[Public]

ADDITIONAL OPTIONS FOR MEMORY MANAGEMENT

```
#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

AMD

[Public]

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 **-faligned-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

AMD

[Public]

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

AMD

[Public]

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.


AMD