Advanced OpenMP®

Justin Chang, Leopold Grinberg, Mahdieh Ghazimirsaeed, Michael Klemm, Suyash Tandon, Bob Robey
AMD @HLRS
Sept 25-28th, 2023

AMD
together we advance_

Advanced OpenMP

1. Region Concept
2. Memory Management
   - Memory Management Capabilities
   - Optimizing Memory Movement Between Host and Device
3. Kernel Resources and Optimization
4. HIP and OpenMP Interoperability

Introduction

With GPU programming we have two considerations that must be addressed

1. **Memory and Data Management**
   1. Between the host and the device
   2. From GPU main memory to the Compute Unit
2. **Code Execution**
   - Managing compute resources
   - Which device to execute operation on
   - Expression of parallelism

- We'll tackle how to address each of these considerations in the following slides and exercises
- Then, we’ll discuss mixing HIP and OpenMP code within an application.

OpenMP® heavily relies on region concept

- What are regions?
  - A part of the code where a pragma applies
  - Default is the normal “block” of code following the directive
  - Can be specified by { }s in C or an end directive in Fortran

- What kinds of regions are there?
  - Data regions – data is on the GPU in this code region
  - Target regions – code in region is executed on the GPU
  - Parallel regions – code in region is executed in parallel

- Original OpenMP specification only had structured data regions
  - How to handle Object-oriented code and other patterns?

➔ Later standard version added unstructured data region concept

Structured vs Unstructured Data regions

Structured data region

```
#pragma omp target enter data map(tofrom: x[0:n])
{
#pragma omp target teams distribute parallel for simd
    for (int i = 0; i < n; n++){
       x[i] = 0.0;
    }
}
```

Unstructured data region

```
class myclass (int n) {
    myclass(){
        x=new double[n];
        #pragma omp target enter data map(alloc: x[0:n])
    }

    ~myclass(){
       #pragma omp target exit data map(delete: x[0:n])
       delete [] x;
       }
```

While object exists

Different Memory Management Capabilities

Explicit Memory Management

Requires explicit memory movement directives.

	Host	Device
x	0x000000000174b0e0	0x00007f617c434000
y	0x000000000175e970	0x00007f617c448000
z	0x0000000001772200	0x00007f617c420000

OpenMP® 5.0

Unified Shared Memory

#pragma omp requires unified_shared_memory

- The Operating System will move memory automatically between host and device.

	Host	Device
x	0x000000000174b0e0	0x00007f617c434000
y	0x000000000175e970	0x00007f617c448000
z	0x0000000001772200	0x00007f617c420000

OpenMP® 5.0

Single Memory address

#pragma omp requires unified_address

- a pointer will always refer to the same location in memory from all devices accessible through OpenMP

	Host/Device
x	0x000000000174b0e0
y	0x000000000175e970
z	0x0000000001772200

Optimizing Memory Movement Between Host and Device

Understanding the behavior of the memory movement pragmas

- The full set of examples is at https://github.com/AMD/HPCTrainingExamples in the HPCTrainingExamples/Pragma_Examples/OpenMP/CXX/memory_pragmas directory

- We'll experiment with different combinations of clauses in the pragmas. By setting LIBOMPTARGET_INFO=-1, we can see what the OpenMP® runtime does behind the scenes.

Summary of OpenMP® memory pragmas and what they do

OpenMP clause	Allocates/deletes device memory	Modifies reference counter	Copies data
Map to/from[1]	Yes	Yes	Yes
Map alloc/delete[2]	Yes[3]	Yes	No
Map release	If reference counter 0, delete	Decrements	No
Update to/from	No	No	Yes

Notes:

1. "Map to" checks if the memory is already allocated for the device.
   a. If not allocated, the device memory is allocated and the reference counter is set to one, and the data is copied to the device
   b. If allocated, the size is checked, and the reference counter is incremented

   Similar for "map from"
2. "Map alloc" checks if the memory is already allocated for the device.
   a. if not allocated, the device memory is allocated, and the reference counter is set to one
   b. if allocated, the size is checked, and the reference counter is incremented.

   "Map delete" will delete the memory and set the reference counter to zero
3. More generally, to cover single memory spaces, the memory must be available in the memory space.

Basic OpenMP® daxpy code (2 slides) – mem1.cc

```
int main(int argc, char* argv[])
{
   int num_iteration=NTIMERS;
   int n = 100000;
   double main_timer = 0.0;
   double main_start = omp_get_wtime();
   if (argc > 1) {
      n=atoi(argv[1]);
   }
   double a = 3.0;
   double *x = new double[n];
   double *y = new double[n];
   double *z = new double[n];

   for (int i = 0; i < n; i++) {
        x[i] = 2.0;
        y[i] = 1.0;
   }

   double * timers = (double *)calloc(num_iteration,sizeof(double));
   for (int iter=0;iter<num_iteration; iter++)
   {
        double start = omp_get_wtime();

        daxpy(n, a, x, y, z);

        timers[iter] = omp_get_wtime()-start;
   }

```

Adding alignment to the memory allocation is highly recommended. This will be discussed in the next section. The line of code with alignment specification is

```
   double *x = new (std::align_val_t(128) ) double[n];
```


together we advance_

Basic OpenMP® daxpy code (continued)

```
   double sum_time =  0.0;
   double max_time = -1.0e10;
   double min_time =  1.0e10;
   for (int iter=0; iter<num_iteration; iter++) {
        sum_time += timers[iter];
        max_time  = max(max_time,timers[iter]);
        min_time  = min(min_time,timers[iter]);
   }

   double avg_time = sum_time / (double)num_iteration;

   cout << "-Timing in Seconds: min=" << fixed << setprecision(6) << min_time << ", max=" <<max_time << ", avg=" << avg_time << endl;

   main_timer = omp_get_wtime()-main_start;
   cout << "-Overall time is " << main_timer << endl;

   cout << "Last Value: z[" << n-1 << "]=" << z[n-1] << endl;

   delete [] x;
   delete [] y;
   delete [] z;

   return 0;
}

void daxpy(int n, double a, double *__restrict__ x, double *__restrict__ y, double *__restrict__ z)
{
#pragma omp target teams distribute parallel for simd map(to: x[0:n], y[0:n]) map(from: z[0:n])
        for (int i = 0; i < n; i++)
                z[i] = a*x[i] + y[i];
}
```

Mem1.cc version

```
Map clause on pragma line just before computational loop
mem1.cc:89 #pragma omp target teams distribute parallel for simd map(to: x[0:n], y[0:n]) map(from: z[0:n])
```

Running this with LIBOMPTARGET_INFO=-1, we can see the memory operations. All occur from the pragma at line 89.

```
LIBOMPTARGET_INFO Report
Libomptarget info: Entering OpenMP kernel at mem1.cc:89:1 with 5 arguments:
Libomptarget info: firstprivate(n)[4] (implicit)   ← Note implicit firstprivate for scalar arguments
Libomptarget info: from(z[0:n])[80000]
Libomptarget info: firstprivate(a)[8] (implicit)
Libomptarget info: to(x[0:n])[80000]
Libomptarget info: to(y[0:n])[80000]
Libomptarget info: Creating new map entry with <...> TgtPtrBegin=0x00007f90b6a20000, Size=80000, DynRefCount=1, HoldRefCount=0, Name=z[0:n]
Libomptarget info: Creating new map entry with <...> TgtPtrBegin=0x00007f90b6a34000, Size=80000, DynRefCount=1, HoldRefCount=0, Name=x[0:n]
Libomptarget info: Copying data from host to device, HstPtr=0x0000000000c2f0e0, TgtPtr=0x00007f90b6a34000, Size=80000, Name=x[0:n]
Libomptarget info: Creating new map entry with <...> TgtPtrBegin=0x00007f90b6a48000, Size=80000, DynRefCount=1, HoldRefCount=0, Name=y[0:n]
Libomptarget info: Copying data from host to device, HstPtr=0x0000000000c42970, TgtPtr=0x00007f90b6a48000, Size=80000, Name=y[0:n]
Libomptarget info: Mapping exists with HstPtrBegin=0x0000000000c56200, TgtPtrBegin=0x00007f90b6a20000, Size=80000, DynRefCount=1 (update suppressed), HoldRefCount=0
Libomptarget info: Mapping exists with HstPtrBegin=0x0000000000c2f0e0, TgtPtrBegin=0x00007f90b6a34000, Size=80000, DynRefCount=1 (update suppressed), HoldRefCount=0
Libomptarget info: Mapping exists with HstPtrBegin=0x0000000000c42970, TgtPtrBegin=0x00007f90b6a48000, Size=80000, DynRefCount=1 (update suppressed), HoldRefCount=0
Libomptarget info: Mapping exists with HstPtrBegin=0x0000000000c42970, TgtPtrBegin=0x00007f90b6a48000, Size=80000, DynRefCount=0 (decremented, delayed deletion) <...>
Libomptarget info: Mapping exists with HstPtrBegin=0x0000000000c2f0e0, TgtPtrBegin=0x00007f90b6a34000, Size=80000, DynRefCount=0 (decremented, delayed deletion) <...>
Libomptarget info: Mapping exists with HstPtrBegin=0x0000000000c56200, TgtPtrBegin=0x00007f90b6a20000, Size=80000, DynRefCount=0 (decremented, delayed deletion) <...>
Libomptarget info: Copying data from device to host, TgtPtr=0x00007f90b6a20000, HstPtr=0x0000000000c56200, Size=80000, Name=z[0:n]
Libomptarget info: Removing map entry with HstPtrBegin=0x0000000000c42970, TgtPtrBegin=0x00007f90b6a48000, Size=80000, Name=y[0:n]
Libomptarget info: Removing map entry with HstPtrBegin=0x0000000000c2f0e0, TgtPtrBegin=0x00007f90b6a34000, Size=80000, Name=x[0:n]
Libomptarget info: Removing map entry with HstPtrBegin=0x0000000000c56200, TgtPtrBegin=0x00007f90b6a20000, Size=80000, Name=z[0:n]
```

Device memory allocated and Reference count incremented

Data copied

Device array deleted

Mem2.cc version -- Add enter/exit data alloc/delete when memory is created/freed

After new

```
mem2.cc:#pragma omp target enter data map(alloc: x[0:n], y[0:n], z[0:n])
```

Keep map on computational loop. The map to/from should check if the data exists. If not, it will allocate/delete it. Then it will do the copies to and from. This will increment the Reference Counter and decrement it at end of loop.

```
mem2.cc:#pragma omp target teams distribute parallel for simd map(to: x[0:n], y[0:n]) map(from: z[0:n])
```

Before delete

```
mem2.cc:#pragma omp target exit data map(delete: x[0:n], y[0:n], z[0:n])
```

LIBOMPTARGET_INFO Report

After new:

```
Libomptarget info: Creating new map entry with <…>TgtPtrBegin=0x00007ff58f020000, Size=80000, DynRefCount=1, HoldRefCount=0, Name=x[0:n]
```

Computational Loop:

```
   Libomptarget info: Mapping exists with HstPtrBegin=0x000000000161d200, TgtPtrBegin=0x00007ff58f048000, Size=80000, DynRefCount=2
(incremented), HoldRefCount=0, Name=z[0:n]
   Libomptarget info: Mapping exists with HstPtrBegin=0x000000000161d200, TgtPtrBegin=0x00007ff58f048000, Size=80000, DynRefCount=1
(decremented), HoldRefCount=0
```

After delete:

```
   Libomptarget device 0 info: Mapping exists with HstPtrBegin=0x000000000161d200, TgtPtrBegin=0x00007ff58f048000, Size=80000,
DynRefCount=0 (reset, delayed deletion), HoldRefCount=0
   Libomptarget device 0 info: Removing map entry with HstPtrBegin=0x000000000161d200, TgtPtrBegin=0x00007ff58f048000, Size=80000,
Name=z[0:n]
```

Mem3.cc version – replace map on computation loop with updates

LIBOMPTARGET_INFO Report

At update – check device array exists and copies data. Reference counter not incremented

```
   Libomptarget info: to(x[0:n])[80000]
   Libomptarget info: Mapping exists with HstPtrBegin=0x0000000000fe10e0, TgtPtrBegin=0x00007ff998a20000, Size=80000,
DynRefCount=1 (update suppressed), HoldRefCount=0
   Libomptarget info: Copying data from host to device, HstPtr=0x0000000000fe10e0, TgtPtr=0x00007ff998a20000, Size=80000,
Name=x[0:n]
```

At computational loop (no map directive) – note implicit checks, increments and decrements of reference counter

```
   Libomptarget device 0 info: use_address(x)[0] (implicit)
   Libomptarget device 0 info: Mapping exists (implicit) with HstPtrBegin=0x0000000000fe10e0, TgtPtrBegin=0x00007ff998a20000,
Size=0, DynRefCount=2 (incremented), HoldRefCount=0, Name=x
   Libomptarget device 0 info: Mapping exists with HstPtrBegin=0x0000000000fe10e0, TgtPtrBegin=0x00007ff998a20000, Size=0, Dy
   Libomptarget device 0 info: Mapping exists with HstPtrBegin=0x0000000000fe10e0, TgtPtrBegin=0x00007ff998a20000, Size=0,
DynRefCount=1 (decremented), HoldRefCount=0nRefCount=2 (update suppressed), HoldRefCount=0
```

Mem4.cc version – replace delete with release

LIBOMPTARGET_INFO Report is the same.

Reference counter is decremented to zero and device array is deleted

Mem7.cc version – add unified shared memory

Must set HSA_XNACK=1 environment variable during run

Add at top of every compilation unit

```
#pragma omp requires unified_shared_memory
```

Remove all memory movement from pragmas (2 computational loops)

```
#pragma omp target teams distribute parallel for simd
```

LIBOMPTARGET_INFO Report

```
Libomptarget device 0 info: Entering OpenMP kernel at mem7.cc:48:1 with 3 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: use_address(x)[0] (implicit)
Libomptarget device 0 info: use_address(y)[0] (implicit)
Libomptarget device 0 info: Entering OpenMP kernel at mem7.cc:91:1 with 5 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: use_address(z)[0] (implicit)
Libomptarget device 0 info: firstprivate(a)[8] (implicit)
Libomptarget device 0 info: use_address(x)[0] (implicit)
Libomptarget device 0 info: use_address(y)[0] (implicit)
-Timing in Seconds: min=0.000078, max=0.000078, avg=0.000078
-Overall time is 0.010562
Last Value: z[9999]=7.000000
```

Memory movement does not show up here. The operating system is doing the memory movement, not the OpenMP® runtime.

AMD
together we advance_

Mem8.cc version – add memory movement back for backwards compatibility

LIBOMPTARGET_INFO Report

```
Libomptarget device 0 info: Entering OpenMP data region at mem8.cc:50:1 with 3 arguments:
Libomptarget device 0 info: alloc(x[0:n])[80000]
Libomptarget device 0 info: alloc(y[0:n])[80000]
Libomptarget device 0 info: alloc(z[0:n])[80000]
Libomptarget device 0 info: Entering OpenMP kernel at mem8.cc:52:1 with 3 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: use_address(x)[0] (implicit)
Libomptarget device 0 info: use_address(y)[0] (implicit)
Libomptarget device 0 info: Entering OpenMP kernel at mem8.cc:97:1 with 5 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: use_address(z)[0] (implicit)
Libomptarget device 0 info: firstprivate(a)[8] (implicit)
Libomptarget device 0 info: use_address(x)[0] (implicit)
Libomptarget device 0 info: use_address(y)[0] (implicit)
-Timing in Seconds: min=0.000079, max=0.000079, avg=0.000079
-Overall time is 0.006449
Libomptarget device 0 info: Updating OpenMP data at mem8.cc:83:1 with 1 arguments:
Libomptarget device 0 info: from(z[0])[8]
Last Value: z[9999]=7.000000
Libomptarget device 0 info: Exiting OpenMP data region at mem8.cc:87:1 with 3 arguments:
Libomptarget device 0 info: alloc(x[0:n])[80000]
Libomptarget device 0 info: alloc(y[0:n])[80000]
Libomptarget device 0 info: alloc(z[0:n])[80000]
```

Memory movement does not show up here. The operating system is doing the memory movement, not the OpenMP® runtime.

AMD together we advance_

Mem9.cc version – Using std::vector with Unified Shared Memory

- One of the big advantages of Unified Shared Memory is it enables the use of std::vector.
- std::vector is pervasive in codes
- But when the vector class reallocates memory, the pointer changes and the device memory map is invalid
- Mem9.cc shows the use of std::vector in conjunction with Unified Shared Memory
  - This is a simple example and doesn't demonstrate a reallocation that would cause the code to fail

Mem10.cc version – Using std::valarray for backwards compatibility

- For backward compatibility, you might look at using valarray instead of vector
  - Valarray was introduced for HPC applications around 1998
  - Valarray does not resize automatically – a plus for HPC since it often leads to performance issues
- But when the vector class reallocates memory, the pointer changes and the device memory map is invalid
- Mem10.cc shows the use of std::valarray to keep backwards compatibility

Set LIBOMPTARGET_INFO flag at runtime

```
extern “C” void __tgt_set_info_flag(uint32_t);

<...>

__tgt_set_info_flag(-1);
#pragma omp target teams distribute parallel for simd map(to: x[0:n], y[0:n]) map(from: z[0:n])
for (int i = 0; i < n; i++)
   z[i] = a*x[i] + y[i];
__tgt_set_info_flag(0);
```

By setting this LIBOMPTARGET_INFO flag at runtime, you can see the detailed information at a particular point in your code

Optimizing Memory Bandwidth Utilization to GPU Main Memory

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(128)) 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 posix_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

Modifying base code to improve bandwidth from main memory

```
int main(int argc, char* argv[])
{
   int num_iteration=NTIMERS;
   int n = 100000;
   double main_timer = 0.0;
   double main_start = omp_get_wtime();
   if (argc > 1) {
      n=atoi(argv[1]);
   }
   double a = 3.0;
   double *x = new (std::align_val_t(128) ) double[n];
   double *y = new (std::align_val_t(128) ) double[n];
   double *z = new (std::align_val_t(128) ) double[n];

   for (int i = 0; i < n; i++) {
        x[i] = 2.0;
        y[i] = 1.0;
   }

   double * timers = (double *)calloc(num_iteration,sizeof(double));
   for (int iter=0;iter<num_iteration; iter++)
   {
        double start = omp_get_wtime();

        daxpy(n, a, x, y, z);

        timers[iter] = omp_get_wtime()-start;
   }

```

Basic OpenMP® daxpy code (continued)

```
   double sum_time =  0.0;
   double max_time = -1.0e10;
   double min_time =  1.0e10;
   for (int iter=0; iter<num_iteration; iter++) {
        sum_time += timers[iter];
        max_time  = max(max_time,timers[iter]);
        min_time  = min(min_time,timers[iter]);
   }

   double avg_time = sum_time / (double)num_iteration;

   cout << "-Timing in Seconds: min=" << fixed << setprecision(6) << min_time << ", max=" <<max_time << ", avg=" << avg_time << endl;

   main_timer = omp_get_wtime()-main_start;
   cout << "-Overall time is " << main_timer << endl;

   cout << "Last Value: z[" << n-1 << "]=" << z[n-1] << endl;

   delete [] x;
   delete [] y;
   delete [] z;

   return 0;
}

void daxpy(int n, double a, double *__restrict__ x, double *__restrict__ y, double *__restrict__ z)
{
#pragma omp target teams distribute parallel for simd map(to: x[0:n], y[0:n]) map(from: z[0:n])
        for (int i = 0; i < n; i++)
                z[i] = a*x[i] + y[i];
}
```

Memory alignment considerations

- Adding memory alignment to your programs is strongly recommended.
- 64 bytes is a minimum recommended and gives much better performance than the default of 16
- You may also try increasing the alignment to 128 or 256
- posix_memalign can be substituted for malloc
- Difficult to get vector and valarray classes to allocate desired memory alignment
  - But it is possible and recommended

Kernel traces and optimizations

Kernel Optimizations

- These examples are at https://github.com/AMD/HPCTrainingExamples in the HPCTrainingExamples/Pragma_Examples/OpenMP/CXX/kernel_pragmas directory

- We'll experiment with different optimizations with pragmas. By setting `LIBOMPTARGET_KERNEL_TRACE=1` or 2, we can see what the OpenMP® runtime does behind the scenes.
  - Setting to 1 shows the name of every kernel, number of teams, threads, and register usage.
  - Setting to 2 prints timing and data transfer information

- LIBOMPTARGET_DEBUG=1 will show more information about data transfer operations and kernel launch
- HPE/Cray
  - `CRAY_ACC_DEBUG=[1,2,3]`
  - `-hlist=aimd` at compile time

Kernel1.cc

- `export HSA_XNACK=1`
- `export LIBOMPTARGET_KERNEL_TRACE=1`
- `mkdir build && cd build`
- `CXX=amdclang++ cmake ..`
- `make`
- `./kernel1`

LIBOMPTARGET_KERNEL_TRACE Report

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 3 teamsXthrds:( 391X 256) reqd:(   0X   0) lds_usage:9784B sgpr_count:106 vgpr_count:58 sgpr_spill_count:39 vgpr_spill_count:0
tripcount:100000 rpc:1 n:__omp_offloading_3d_1a2def2_main_l52
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:( 391X 256) reqd:(   0X   0) lds_usage:9784B sgpr_count:106 vgpr_count:56 sgpr_spill_count:47 vgpr_spill_count:0
tripcount:100000 rpc:1 n:__omp_offloading_3d_1a2def2__Z5daxpyidPdS_S__l97
```


AMD together we advance_

kernel2.cc

- Change number of threads – add num_threads(64)

- New report

```
LIBOMPTARGET_KERNEL_TRACE Report
DEVID: 0 SGN:2 ConstWGSize:64   args: 3 teamsXthrds:( 416X  64) reqd:(   0X  64) lds_usage:9784B sgpr_count:106 vgpr_count:59 sgpr_spill_count:42 vgpr_spill_count:0
tripcount:100000 rpc:1 n:__omp_offloading_3d_1a2def6_main_l52
DEVID: 0 SGN:2 ConstWGSize:64   args: 5 teamsXthrds:( 416X  64) reqd:(   0X  64) lds_usage:9784B sgpr_count:106 vgpr_count:57 sgpr_spill_count:48 vgpr_spill_count:0
tripcount:100000 rpc:1 n:__omp_offloading_3d_1a2def6__Z5daxpyidPdS_S__l97
```

kernel3.cc

- Add thread limit for kernel – num_threads(64) thread_limit(64)

- New report

LIBOMPTARGET_KERNEL_TRACE Report
DEVID: 0 SGN:2 ConstWGSize:64 args: 3 teamsXthrds:(416X 64) reqd:(0X 64) lds_usage:9784B sgpr_count:106 vgpr_count:55 sgpr_spill_count:37 vgpr_spill_count:0 tripcount:100000 rpc:1 n:__omp_offloading_3d_1a2def8_main_l52
DEVID: 0 SGN:2 ConstWGSize:64 args: 5 teamsXthrds:(416X 64) reqd:(0X 64) lds_usage:9784B sgpr_count:106 vgpr_count:53 sgpr_spill_count:45 vgpr_spill_count:0 tripcount:100000 rpc:1 n:__omp_offloading_3d_1a2def8__Z5daxpyidPdS_S__l97

AMD together we advance_

Summary

OpenMP compute loop clauses	Workgroup size	LDS Usage	SGPR	VGPR	SGPR Spill	VGPR Spill
Simple parallel loop	256 256	9784 B 9784 B	106 106	58 56	39 47	0 0
num_threads(64)	64 64	9784 B 9784 B	106 106	59 57	42 48	0 0
num_threads(64) thread_limit(64)	64 64	9784 B 9784 B	106 106	55 53	37 45	0 0

- Note that reducing the threads does not reduce the VGPRs
- Adding the thread_limit clause does reduce the VGPRs

This is a very simple kernel. We are below the VGPR limit for occupancy restrictions. So the impact in this case is small. Try these changes on your larger kernels and see what it does there.

Register pressure and occupancy for MI250X

Note: When greater than 256, additional vector registers are stored in scratch, a slower memory. In most cases this should be avoided. We are below that number, but we are still concerned with the limit on the number of waves that can be scheduled.

This is the column that corresponds to the compiler and profiler report.

Num VGPRs	Occupancy per EU	Occupancy per CU
<= 64	8 waves	32 waves
<= 72	7 waves	28 waves
<= 80	6 waves	24 waves
<= 96	5 waves	20 waves
<= 128	4 waves	16 waves
<= 168	3 waves	12 waves
<= 256	2 waves	8 waves
> 256 (+ spilling to scratch)	1 waves	4 waves

There are 4 arithmetic Execution Units per Compute Unit. The compiler generates the VGPRs for each wavefront to be run on each Execution Unit. The scheduler can place 4 of the same wavefronts to execute on the CU or wavefronts from other tasks.

A brief word about kernel parallelization semantics

We used the kernel parallelization pattern below in our examples

```
#pragma omp target teams distribute parallel for simd
```

- The simd clause is only needed for Cray systems and will be dropped by them in the future
  - Most compilers are not implementing this level for the GPU
- Understanding the OpenMP approach – it is wordy
  - OpenMP is prescriptive and is designed so the user can control how parallelism is implemented for their application
  - OpenACC is descriptive – it gives the compiler more freedom on how to implement things
  - There is an extra level of parallelism provided by SIMD (think vector) that really isn't necessary for the GPU leading to differences in implementations

- OpenMP 5.0 standard has introduced the "loop" clause which replaces the parallel for simd construct and provides a descriptive alternative that gives the compiler more freedom on how to implement the parallelism
- It is safest for the time being to use the full syntax as shown in our examples for backwards compatibility.
- You may test alternative forms to see if they work on all the systems you are using.

Hip and OpenMP® Interoperability

HIP and OpenMP® Interoperability

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 @HLRS

(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];
    }
}
```

(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.

(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);
    hipDeviceSynchronize();
}
```

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

(2) OPENMP® TO HIP: FORTRAN AND DGEMM EX

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
```

Translation unit 1

ftn

```
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
```

```
#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

(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 together we advance_

REFERENCES

- AOMP: https://github.com/ROCm-Developer-Tools/aomp
  - AMD open-source Clang/LLVM based compiler with support for OpenMP® API
- HIPFORT: https://github.com/ROCmSoftwarePlatform/hipfort
  - Readily available FORTRAN interfaces to HIP/ROCm libraries

Disclaimer

AMD