OLCF Summit Early Science Workshop

Fortran OpenMP4.5 Topics

Lixiang Luo, David Appelhans

IBM

March 7th, 2018, Oak Ridge National Lab

List of Topics

- Asynchronous execution for offloading
- Multi-threaded asynchronous execution
- Implicit barriers inside a GPU kernel
- Other OpenACC/OpenMP4.5 migration considerations
  - Use of team index
  - Difference on “declare” directive for Data
  - Optimizations for Fortran array operations

Asynchronous Execution on GPUs

OpenACC

- Dependency resolution using streams – simpler for the implementer.
- OpenACC streams are mostly a abstraction of CUDA streams (not 1:1).
- Dependency between OpenACC and host OpenMP tasks needs additional coordination.

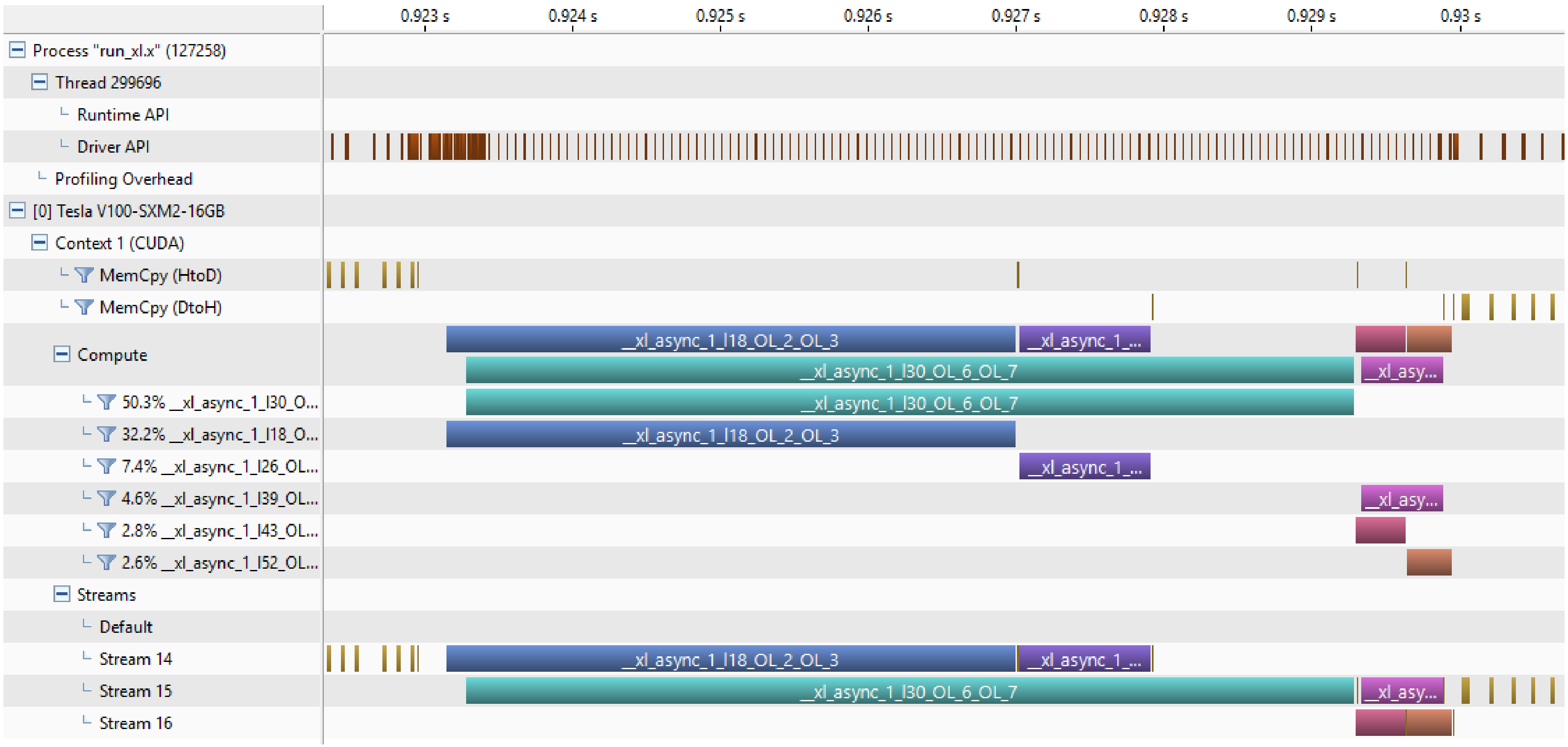
OpenMP4.5

- Task dependency based on data dependency – simpler for the user.
- There is no well-defined mapping to CUDA streams.
- Dependency between device and host tasks is natively supported.

A Comparison on Async

Enter data	**!$acc enter data copyin(a,v1,v2) create(v3,v4,v5)** **!$omp target enter data map(to:a,v1,v2) map(alloc:v3,v4,v5) depend(out:a,v1,v2)**
MxV – V3	**!$acc parallel loop independent gang vector collapse(2) async(1) default(present)** **!$omp target teams distribute parallel do simd collapse(2) nowait depend(in:a,v1) depend(out:v3)** do ie=1,Ne ; do i=1,Nd ; v3(i,ie)=dot_product(a(i,:,ie),v1(:,ie)) ; end do ; end do
Reduction – V3	mx3=0. **!$acc parallel loop gang vector async(3) wait(1) reduction(max:mx3) default(present)** **!$omp target teams distribute parallel do simd reduction(max:mx3) nowait depend(in:v3) depend(out:mx3)** do ie=1,Ne ; mx3 = max(dot_product(v3(:,ie),v3(:,ie)),mx3) ; end do
MxV – V4	**!$acc parallel loop independent gang vector collapse(2) async(2) default(present)** **!$omp target teams distribute parallel do simd collapse(2) nowait depend(in:a,v2) depend(out:v4)** do ie=1,Ne ; do i=1,Nd ; v4(i,ie)=dot_product(a(i,:,ie),v2(:,ie)) ; end do ; end do
Reduction – V4	mx4=0. **!$acc parallel loop gang vector async(4) wait(2) reduction(max:mx4) default(present)** **!$omp target teams distribute parallel do simd reduction(max:mx4) nowait depend(in:v4) depend(out:mx4)** do ie=1,Ne ; mx4 = max(dot_product(v4(:,ie),v4(:,ie)),mx4) ; end do
Add – V5	**!$acc parallel loop independent gang vector collapse(2) async(5) wait(1,2) default(present)** **!$omp target teams distribute parallel do simd collapse(2) nowait depend(in:v3,v4) depend(out:v5)** do ie=1,Ne ; do i=1,Nd ; v5(i,ie)=v3(i,ie)+v4(i,ie) ; end do ; end do
Reduction – V5	mx=0. **!$acc parallel loop gang vector wait(5) reduction(max:mx) default(present)** **!$omp target teams distribute parallel do simd reduction(max:mx) nowait depend(in:v5)** do ie=1,Ne ; mx = max(dot_product(v5(:,ie),v5(:,ie)),mx) ; end do
Exit data	**!$acc exit data copyout(v5)** **!$acc wait(3,4)** **!$omp target exit data map(from:v5) depend(in:v5,mx,mx3,mx4)**

NVPROF Timelines of OpenMP4.5 Async Example



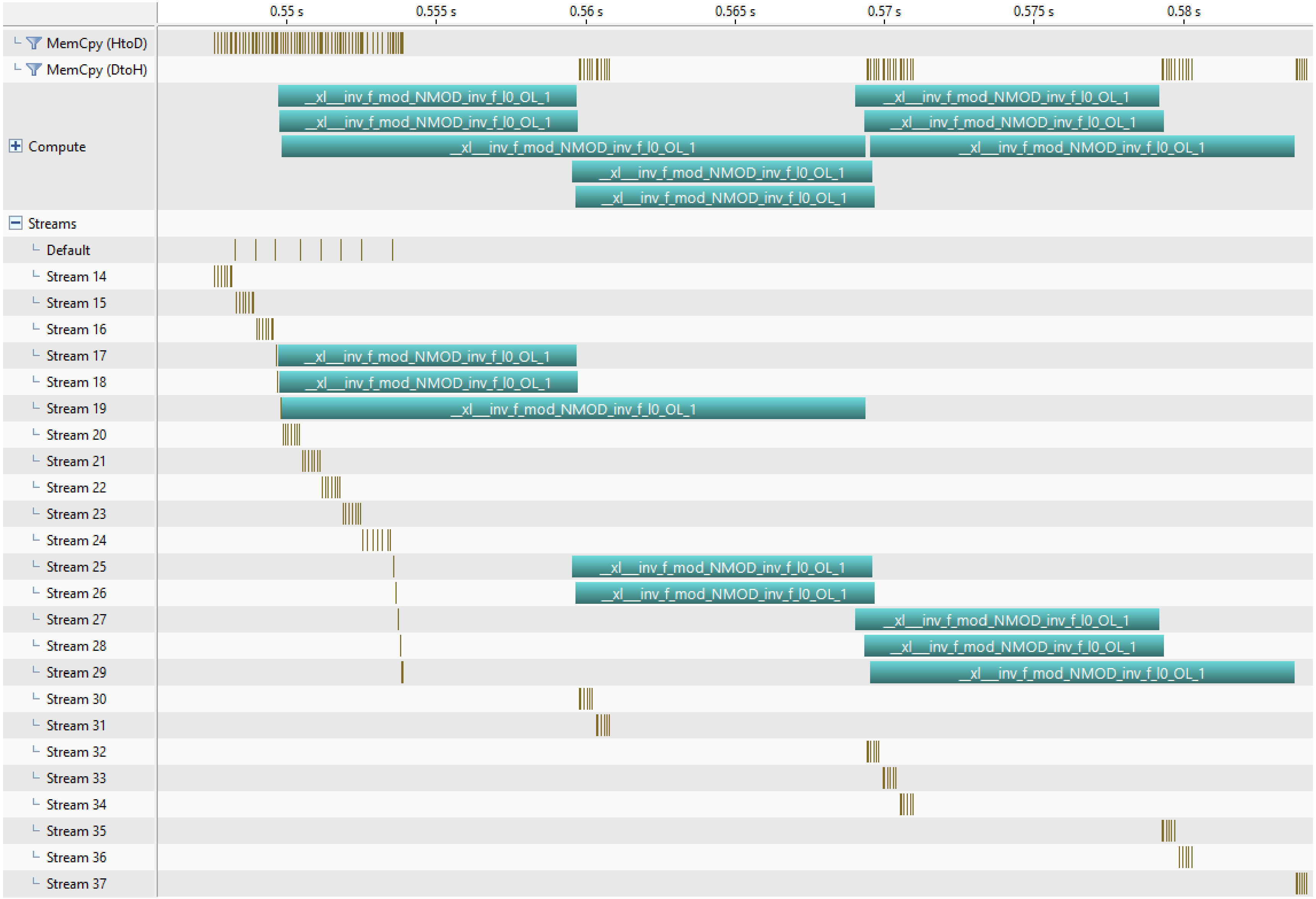
Multi-threaded Async Offloading

```
!$omp parallel do private(batch_start, batch_end, batch_size_this, thread_id, p_batch)
do ibatch=0,n_batch-1

  batch_start = ibatch*batch_size+1;
  batch_end = min((ibatch+1)*batch_size,Nmat_total);
  batch_size_this = batch_end-batch_start+1;
  thread_id = omp_get_thread_num();
  p_batch => matrix_array(:,:,batch_start:batch_end)

  !$acc enter data copyin(p_batch) async(thread_id)
  #ifdef OMP4OL
  !$omp target enter data map(to:p_batch) nowait depend(out:p_batch)
  #endif
  call inv_f(batch_size_this,N,p_batch,thread_id)
  !$acc exit data copyout(p_batch) async(thread_id)
  #ifdef OMP4OL
  !$omp target exit data map(from:p_batch) depend(in:p_batch)
  #endif

end do
!$acc wait
```

To run on Summit inside an interactive session using 8 threads:

```
$ jsrun -n1 -a1 -c8 -g1 -bpacked:8 ./fginv_xlomp4.x
```

NVPROF Timeline



Implicit Barriers inside a GPU Kernel

- In CUDA programming, “shared memory + thread synchronization” is one of the most fundamental concepts of kernel optimization.
- A similar technique can be applied in both OpenACC and OpenMP4.5.
- In OpenACC, this is achieved by multiple vector loops inside a “parallel gang” loop. An implicit barrier is imposed at the end of each vector loop.
- In OpenMP4.5, this is achieved by multiple “parallel do simd” loops inside a “teams distribute” loop. An implicit barrier is imposed at the end of each “parallel do” loop.

Implicit Barriers inside a GPU Kernel

```
!$omp target teams distribute nowait private(ibuf,ibuf_n,k,b,dep) depend(inout:dd)
!$acc parallel loop gang present(dd) private(dep) default(present) async(thread_id)
do ie=1,n_batch
   !$acc cache(dep)     ! Use CUDA shared memory

   !$omp parallel do simd collapse(2) private(s)
   !$acc loop vector collapse(2) private(s)
   do j=1,nv; do i=1,nv
     …
   end do; end do
   …
   do k=1,nv
      !$omp parallel do simd collapse(2) private(s,s1,s2,ieqk,jeqk)
      !$acc loop vector collapse(2) private(s,s1,s2,ieqk,jeqk)
      do j=1,nv; do i=1,nv
        …
      end do; end do
      …
   end do
end do
```

OpenACC/OpenMP4.5 Migration Considerations

- OpenMP4.5 kernel grid sizes are decided at runtime – OpenACC can decide the block size at compile time.
- OpenMP4.5 provides team index, while OpenACC does not.
- OpenMP4.5 supports explicit thread barriers inside each team, while OpenACC does not.
- OpenMP4.5 does not provide direct control for putting data in CUDA shared memory. It is implemented as an optimization if certain conditions are met.
- OpenACC "kernels" construct allows the compiler to do whatever as it sees fit. OpenMP4.5 has no equivalence, so manual refactoring may be necessary.

OpenACC/OpenMP4.5 Migration Considerations

- "!$acc declare" takes various mapping clauses and works for data of any scope, while "!$omp declare target" only works for procedures and global data. "!$acc declare" directives for data are usually translated into explicit OpenMP4.5 data mapping directives.
- Writing two paradigms in the same source files is possible. To prevent PGI from processing OpenMP4.5 offloading directives, these directives must be guarded by macros.
  - OpenMP5 will provide a more elegant solution.
- In a situation where device pointers must be shared between OpenACC and OpenMP4.5 (libraries written in different paradigms, for example), OpenMP4.5 runtime must be initialized first. This can be achieved by a dummy OpenMP4.5 target kernel before OpenACC initialization.

A (Bad) Team Index Example: Manual Privatization

OpenACC

```
real, dimension(L,N_elm) :: privarr

…

!$acc parallel loop gang create(privarr)
do ie=1,N_elm
   !$acc loop vector
   do i=1,L
     privarr(L,ie) = …
     …
   end do
end do
```

OpenMP4.5

```
real, dimension(L,N_team) :: privarr
Integer :: team_id
…
!$omp target teams map(alloc:privarr)
private(team_id)
team_id = omp_get_team_num()
!$omp distribute
do ie=1,N_elm
   !$omp parallel do simd
   do i=1,L
     privarr(L,team_id) = …
     …
   end do
end do
```

In OpenACC, one cannot make assumption on the gang length, and there is no gang identification. In OpenMP4.5, each team can be identified, which allows a much smaller manual privatization array.

Difference on “declare” Directive for Data

```
subroutine foo(someinput,…)
…
integer, intent(in) :: some_input
!$acc declare create ( some_input )   ! Mapping valid for procedure scope
! There is no direct equivalence in OpenMP4.5. Instead, do the following:

!$omp target enter data map ( alloc:some_input )

…

!$omp target exit data map ( delete:some_input )
…
end subroutine foo
```

Optimizations for Fortran Array Operations

OpenMP4.5 has no equivalence to "!$acc kernels". Manual refactorization is necessary in some cases. The OpenACC code one the left generates an efficient copy kernel which utilizes all GPU cores. For OpenMP4.5, a loop must be used instead. Similarly, matrix operations such as MATMUL and DOT_PRODUCT must be refactorized.

OpenACC

```
real, dimension(N) :: a,b
a = …

!$acc data copyin(a) copyout(b)
!$acc kernels
b = a
!$acc end kernels
…
!$acc end data
```

OpenMP4.5

```
real, dimension(N) :: a,b
a = …

!$omp target data map(to:a) map(from:b)
!$omp target teams distribute parallel do simd
thread_limit(256)
do i=1,N
   b(i) = a(i)
end do
…
!$omp end target data
```