Pipelining in OpenACC
Parallelism vs. Pipelining

- In MPI
  - Parallel execution on MPI ranks without pipelining:
    - MPI_Send(1 GB); Compute(on 1GB);
  - Parallel execution on MPI ranks with pipelining:
    - MPI_Send(1 MB); Compute(on 1MB); send(); compute(); ...

- Pipelining in CUDA (two streams are required to run communication and computation in parallel)
  - MemcpyAsync(comm_stream); Event(); Kernel<<<krnl_stream>>>
  - MemcpyAsync(comm_stream); Event(); Kernel<<<krnl_stream>>>
  - MemcpyAsync(comm_stream); Event(); Kernel<<<krnl_stream>>>

- To make pipelining possible, large tasks have to be decomposed
  - This may not be possible for some algorithms
Pipelining Transfers and Overlapping with Kernels

memcpy to device  kernel  memcpy to host  memcpy to device  kernel  memcpy to host
memcpy to device  kernel  memcpy to host
memcpy to device  kernel  memcpy to host
memcpy to device  kernel  memcpy to host
memcpy to device  kernel  memcpy to host
OpenACC Pipelining Basics

• Non-pipelined code
  - #pragma acc data copy(array)
  - #pragma acc parallel loop
  - For (k = ...N)
    • compute_on_device(array)

• Pipelined code
  - #pragma acc data create(array_section)
  - For (outer_K = ...N/blk_size)
    • #pragma acc update device(array_section)
    • #pragma acc parallel loop
    • For (k = ...blk_size)
      • compute_on_device(array_section)
    • #pragma acc update host(array_section) self(array_section)
      • /* “host” is deprecated, use “self” instead */
Pipelining Transfers and Overlapping with Kernels
Asynchronous Execution in OpenACC

- OpenACC has a notion of streams that are called work queues
  - They are numbered with integers
  - Operations in a work queue are executed in-order just as in a stream
  - Running operations from multiple work queues may be in parallel
- Two clauses manage queues
  - `async(k)` launches the following code asynchronously in work queue `k`
    - Without `k` specified, the default work queue is used
  - `wait(k)` blocks the host until all operations in queue `k` completed
    - Without `k` specified, the default work queue is used
/* launch first kernel asynchronously */
#pragma acc parallel loop async(1)
for (int j=0; j<N; ++j)
    /*compute loop */

/* launch second kernel asynchronously */
#pragma acc parallel loop async(1)
for (int j=0; j<N; ++j)
    /*compute loop */

/* wait for work queue 1 */
#pragma acc wait(1)

/* wait for all work queues */
#pragma acc wait
Multi-GPU Asynchronous Code in OpenACC

```c
#pragma omp parallel num_threads(acc_get_num_devices(acc_device_nvidia))
{
    int myid = omp_get_thread_num();
    acc_set_device_num(myid, acc_device_nvidia);
    int queue = 1;

#pragma acc data create(array_section)
{
    #pragma omp for schedule(static)
    for (int blk = 0; blk < allblks; ++blk)
    {
        #pragma acc update device(array_section) async(queue)
        #pragma acc parallel loop collapse(2) gang vector async(queue)
        for (int j=0; j<blksize; ++j)
        {
            for (int i=0; i<blksize; ++i)
                compute(array_section);
        }
        #pragma acc update self(array_section) async(queue)
        queue = (queue % allqueues)+1;
    }

    /* wait for all the queued work to finish */
    #pragma acc wait
}
```

Set a device separately for each thread. CUDA also associates each thread with a device.

There are multiple work queues for each device to pipeline on individual devices.
Additional OpenACC Information

- OpenACC Course Recordings
  - https://developer.nvidia.com/openacc-courses
- PGI Website
  - http://www.pgroup.com/resources
- OpenACC on StackOverflow
  - http://stackoverflow.com/questions/tagged/openacc
- OpenACC Toolkit
- Parallel Forall Blog
- GPU Technology Conference
  - http://www.gputechconf.com/
- OpenACC Website
  - http://openacc.org/