GPU Teaching Kit

Accelerated Computing

Module 14 – Efficient Host-Device Data Transfer

Lecture 14.4 – CUDA Unified Memory

Objective

- -To learn the important concepts involved in an efficient use of CUDA Unified Memory
  - Prefetching
  - Memory Advisor

Data Prefetching in CUDA Unified Memory

- Main purpose being to avoid page faults
- Establishes data locality, providing a mechanism to improve the performance of the application
- Used to migrate data to a device or the host and map page tables onto the processor before it begins using data
- Most useful when the data is accessed from a single processor

Data Prefetching CUDA API

- cudaMemPrefetchAsync()
  - Four parameters
    - Unified Memory Address of the memory region to prefetch
    - Size of the region to prefetch in terms of bytes
    - Destination processor
    - CUDA stream

- Prefetching is an asynchronous operation with respect to the device, however the call may not be fully asynchronous with respect to the host .
- The destination processor must be a valid device ID or *cudaCpuDeviceId,* the later will prefetch the memory to the host.
- If the prefetch processor is a GPU, the device property cudaDevAttrConcurrentManagedAccess must be nonzero.

Putting it all together

```
void function(int * data, cudaStream_t stream) {
    // data must've been allocated with cudaMallocManaged( (void**) &data, N);
    init(data, N);                                                    // Init data on the host
    cudaMemPrefetchAsync(data, N * sizeof(int), myGpuId, stream);     // Prefetch to the device
    kernel<<<..., stream>>>(data, ...);                               // Execute on the device
    cudaMemPrefetchAsync(data, N * sizeof(int), cudaCpuDeviceId, stream);  // Prefetch to the host
    cudaStreamSynchronize(stream);
    hostFunction(data, N);
}
```

CUDA Unified Memory - Memory Advisor

- Hints that can be provided to the driver on how data will be used during runtime

- One example of when we can give an advice is when data will be accessed from multiple processors at the same time

Memory Advisor CUDA API

- cudaMemAdvise()
  - Four parameters
    - Unified Memory Address of the memory region to advise
    - Size of the region to advise in terms of bytes
    - Memory advise
    - Destination processor
- The destination processor must be a valid device ID or *cudaCpuDeviceId.*
- Available memory hints are:
  - cudaMemAdviseSetReadMostly
  - cudaMemAdviseSetPreferredLocation
  - cudaMemAdviseSetAccessedBy

  To unset the advice:
  - cudaMemAdviseUnsetReadMostly
  - cudaMemAdviseUnsetPreferredLocation
  - cudaMemAdviseUnsetAccessedBy

Memory Advisor CUDA API

- cudaMemAdviseSetReadMostly:
  - This tells the UM driver that the memory region is mostly for reading purposes and occasionally writing.
  - All read accesses from a processor will create a read only copy to be accessed.
  - The device argument is ignored.
  - When used in conjunction with *cudaMemPrefetchAsync()* a read only copy will be created in the specified device.
- cudaMemAdviseSetPreferredLocation:
  - This tells the UM driver the preferred location of the data.
  - However this does not cause immediate data migration to the location, it only guides the migration policy for the UM driver.
  - If the destination processor is a GPU, then that GPU needs a nonzero value for the device flag *cudaDevAttrConcurrentManagedAccess*.

Memory Advisor CUDA API

- cudaMemAdviseSetAccessedBy:
  - This tell the UM driver that the data will be accessed by the specified processor.
  - This advice does not cause immediate data migration to the location, it only causes data to always be mapped in the specified processor’s pages.
  - It is useful to avoid page faulting, like when in a multi-GPU system one device wants to access another GPU's memory but migrating the data may be more expensive than just reading it through the PCIE link.

Putting it all together

```
void function(int * data, cudaStream_t stream) {
    // data must be addressable by the Unified Memory driver.
    init(data, N);        // Init data on the host
    cudaMemAdvise(data, N * sizeof(int), cudaMemAdviseSetReadMostly , 0 );    // Set the advice for read only.
    cudaMemPrefetchAsync(data, N * sizeof(int), myGpuId1, stream1);           // Prefetch to the 1st device.
    cudaMemPrefetchAsync(data, N * sizeof(int), myGpuId2, stream2);           // Prefetch to the 2nd device.
    cudaSetDevice(myGpuId1);                                                  // Execute read only operations
    kernel<<<..., stream>>>(data, ...);                                       // on the 1st device.
    cudaSetDevice(myGpuId2);                                                  // Execute read only operations
    kernel<<<..., stream>>>(data, ...);                                       // on the 2nd device.
}
```

GPU Teaching Kit

Accelerated Computing