IT00CG19-3002

GPU Programming

Slide set #5: Multi-GPU Programming

Fall 2023, period 1

Jan Westerholm
Åbo Akademi University

Multi-GPU Program Design and Implementation

- Many supercomputers and computer clusters consist of nodes with 2 CPU processors dies, each with tens of CPU cores, and 4 or even 8 GPUs.
- Running a GPU program on one of these nodes therefore has direct access to 4 or 8 GPUs.
  - Distributed memory programming with e.g. MPI not needed but may be helpful
- The relevant questions are now: how do we access (access = copy data to/from, run programs on) these GPUs and how do we subdivide our computational task between the GPUs?

Multi-GPU Program Design

- Central idea: Distribute the thread blocks

```
  kernel<<<threadblocks,threads_in_block>>>(…);
```

- as smaller threadblock bunches among the GPUs:

```
bunch = (threadblocks + #ofGPUs - 1)/#ofGPUs
 for ( int i = 0; i < #ofGPUs; ++i )
     {
     cudaSetDevice(i);
     //copy all data to device i
     cudaMemcpy(…, cudaMemcpyHostToDevice)
     int tbstart = i*bunch;
     kernel<<<bunch,threads_in_block>>>(…,tbstart);
     }
```

Multi-GPU Program Design

- Change your GPU program

```
blockIdx.x ⇒ blockIdx.x + tbstart
```

- Copy data back to CPU

```
for ( int i = 0; i < #ofGPUs; ++i )
{
cudaSetDevice(i);
// copy result back to CPU and aggregate
 cudaMemcpy(…, cudaMemcpyDeviceToHost)
}
```

- This can also be made to work for #ofGPUs = 1.
- Now the program can be made to run on all available GPUS, cudaGetDeviceCount (int* count)

Multi-GPU Program Design

- This design can be extended to MPI programs, where the GPU program is subdivided to work on GPUs on separate nodes (two levels: first which node, then which GPU!)
- The full data should now be subdivided among all GPUs

$$(\#\ \text{of nodes}) * (\#\ \text{of GPUs per node})$$

using the rank of the MPI program, one MPI process per GPU, as a global GPU ID.