Programming in CUDA

Part VI. CUDA & MPI

Frédéric S. Masset

Lecture on CUDA

Outline

Outline

Main idea: one GPU = one host process

- We want GPUs to work collectively as we wanted host processes to work collectively through MPI on a “normal” cluster.
- Therefore, now we want one GPU to correspond to one MPI process.
- The run is spawn through `mpirun -np ... my_exec`. Each instance (each rank) then has to select its own device.

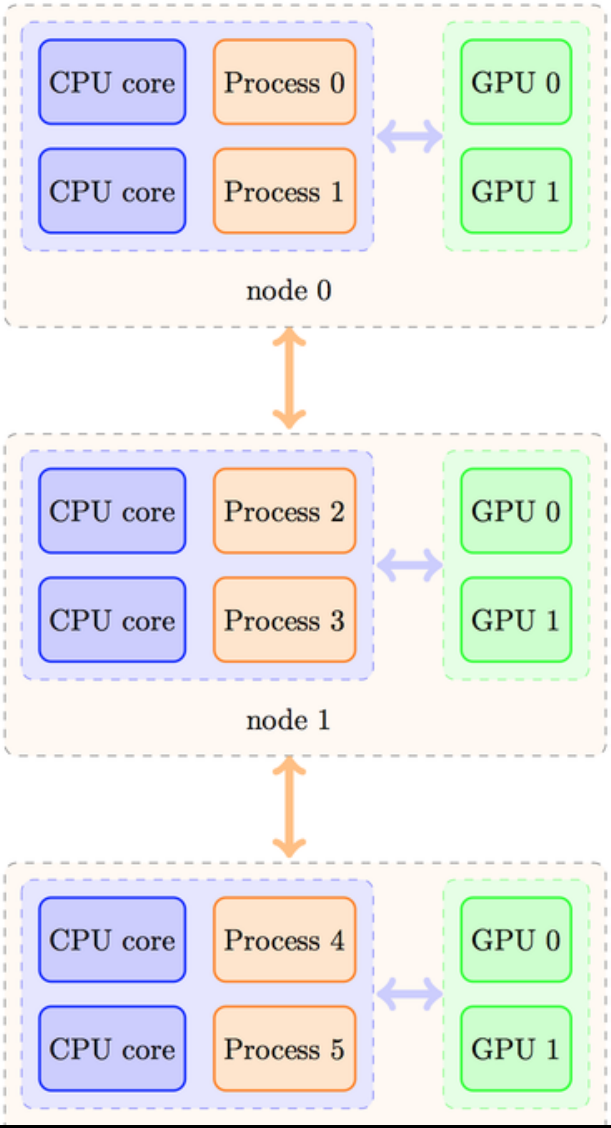
Device selection from the host

If there are several GPUs on a host, they have different logical numbers, starting from 0 (hence on Kepler's nodes we have devices 0 to 3).
In order to select manually a device at runtime, we issue:

Device selection

```
cudaSetDevice (int device_number)
```

Configuration example



On this cluster each node has two GPUs. We there want to run TWO processes per node (no more !) so that each process has access to its own GPU.

Here, we want the process 0 to select GPU 0, process 1 to select GPU 1, process 2 to select GPU 0, process 3 to select GPU 1, etc...

What is the general rule, here ?

Outline

How do we communicate data between GPUs ?

- Either we communicate through the hosts: get the data down to the hosts (D2H), send it to other processes, then upload it to the devices (H2D).
- Recent versions of CUDA and MPI allow direct transmission !

Standard, "old" method

The standard method (go through the host) is straightforward in its principle. It splits completely the CUDA part from the MPI part:

- CUDA and the calculation on board the GPU is not aware of other processes and potential communications with them
- MPI communications are performed on the host memory, as always.

Both levels of parallelism are completely “orthogonal”, and this can be achieved with any version of CUDA and MPI, respectively.

CUDA aware MPI

Modern versions of CUDA (>4.0) support UVA mode (*Unified Virtual Addressing*): the compiler knows whether a pointer points to device or host memory.
If a recent version of MPI (OpenMPI >= 1.7, MVAPICH2) is used, it can take advantage of UVA: communications can be done directly on the device, so that we can have:

CUDA aware MPI

```
MPI_Send (gpu_ptr1, ... )
MPI_Recv (gpu_ptr2, ... )
```

Advantages:

- No need to transfer the data through the host!
- Transfer is optimized transparently: it takes the fastest way

CUDA aware MPI

Note

This does not (yet) work for reduction type MPI instructions, such as MPI_Scan().

Another advantage of UVA

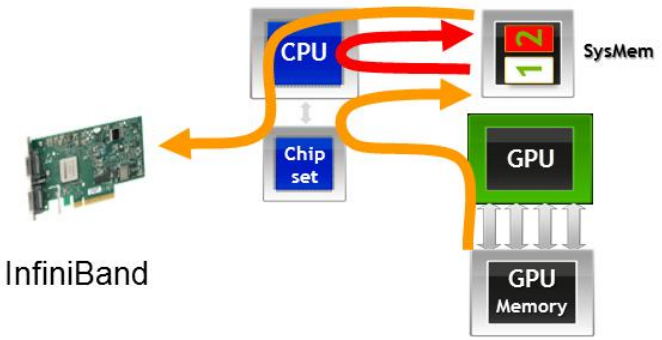
Since the compiler knows where the data resides (host / device), then the qualifiers `CudaMemcpyDeviceToHost`, `CudaMemcpyHostToDevice`, etc. become obsolete !
There is only one qualifier: `CudaMemcpyDefault` !

Data transfers can be optimized with GPU Direct

Without GPUDirect

Same data copied three times:

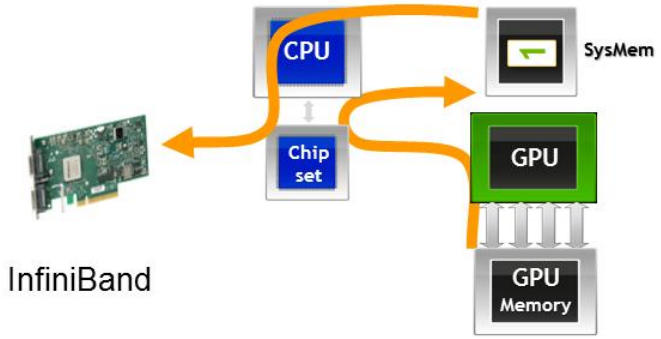
1. **GPU writes to pinned sysmem1**
2. **CPU copies from sysmem1 to sysmem2**
3. **InfiniBand driver copies from sysmem2**



With GPUDirect

Data only copied twice

Sharing pinned system memory makes sysmem-to-sysmem copy unnecessary



Optimal transfers are achieved transparently. No specific action is required from the user. All is needed is CUDA aware MPI version.

Outline

GPU selection: a problem

- With ANY CUDA aware MPI implementation (OpenMPI-1.7, MVAPICH2), each process must have selected its GPU when `MPI_Init()` is executed. This is a requirement so that later invocations of pointers on board the devices by MPI functions know where to seek the data.
- We have seen earlier that the GPUs are selected, on a given platform, according to the rank of the process. We need to do that BEFORE `MPI_Init ()`.
- But... how do we know the rank *before* invoking `MPI_Init ()` ??

GPU selection: a problem

- With ANY CUDA aware MPI implementation (OpenMPI-1.7, MVAPICH2), each process must have selected its GPU when `MPI_Init()` is executed. This is a requirement so that later invocations of pointers on board the devices by MPI functions know where to seek the data.
- We have seen earlier that the GPUs are selected, on a given platform, according to the rank of the process. We need to do that BEFORE `MPI_Init ()`.
- But... how do we know the rank *before* invoking `MPI_Init ()` ??

GPU selection: a problem

- With ANY CUDA aware MPI implementation (OpenMPI-1.7, MVAPICH2), each process must have selected its GPU when `MPI_Init()` is executed. This is a requirement so that later invocations of pointers on board the devices by MPI functions know where to seek the data.
- We have seen earlier that the GPUs are selected, on a given platform, according to the rank of the process. We need to do that BEFORE `MPI_Init ()`.
- But... how do we know the rank *before* invoking `MPI_Init ()` ??

Know the rank BEFORE invoking MPI_Init()

How is this possible ? We need to do this manually. We use an environment variable that depends on the implementation.

- For OpenMPI, this variable is `OMPI_COMM_WORLD_RANK`
- For MVAPICH2, this variable is `NV2_COMM_WORLD_RANK`

How do we use it ? We read it from the process(es) and for each process it has the value of the rank that *will be* allocated later on during `MPI_Init ()`.

Example of use

We assume that we run OpenMPI. We would do the following:

```
char rank_string[512];
int my_future_rank;
rank_string = getenv("OMPI_COMM_WORLD_RANK");
my_future_rank = atoi (rank_string);
cudaSelectDevice (my_future_rank % 2);
MPI_Init (...);
```

Even better solution: *local* rank

There is something even simpler: get the *local* rank directly, *i.e.* the rank within the node (hence local).

```
char rank_string[512];
int local_rank;
rank_string = getenv("OMPI_COMM_WORLD_LOCAL_RANK");
local_rank = atoi (rank_string);
cudaSelectDevice (local_rank); // No need for %
MPI_Init (...);
```

Exercise: a very simple GPU-GPU communication

Write from scratch a simple piece of code that does the following:

- Initialize a short vector with a different value on each process
- Send it to the device, in each process
- Reset to zero the vector on the host
- Do a send of the DEVICE vector from process 0 to process 1, with MPI
- Transfer the vectors back to the host
- Display the values on screen

Different steps

- In a first step, you can have all your processes on the same GPU.
- In a second step, use the early rank attribution to select different GPUs for your processes and rerun your code.

Setting the environment

On the kepler machine, you'll need to set, in your `.bashrc` file:

if you want to use OpenMpi:

```
export PATH=/share/apps/openmpi/bin:$PATH
export LD_LIBRARY_PATH=/share/apps/openmpi/lib:$LD_LIBRARY_PATH
```

or, if you prefer to use MVAPICH2:

```
export PATH=/share/apps/mvapich2/bin:$PATH
export LD_LIBRARY_PATH=/share/apps/mvapich2/lib:$LD_LIBRARY_PATH
export MV2_USE_CUDA=1
```