Advanced Features Overview

Sebastian Keller, Javier Otero, Prashanth Kanduri
and Ben Cumming, CSCS

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Using MPI with GPUs

What is MPI

MPI (**Message Passing Interface**) is a standardised library for message passing

- Highly portable: it is implemented on every HPC system available today.
- Has C, C++ and Fortran bindings.
- Supports point to point communication
  - `MPI_Send`, `MPI_Recv`, `MPI_Sendrecv`, etc.
- Supports global collectives
  - `MPI_Barrier`, `MPI_Gather`, `MPI_Reduce`, etc.

When you start an MPI application

- N copies of the application are launched.
- Each copy is given a **rank** $\in \{0, 1, \ldots, N-1\}$.

A basic MPI application

Example MPI application myapp.cpp

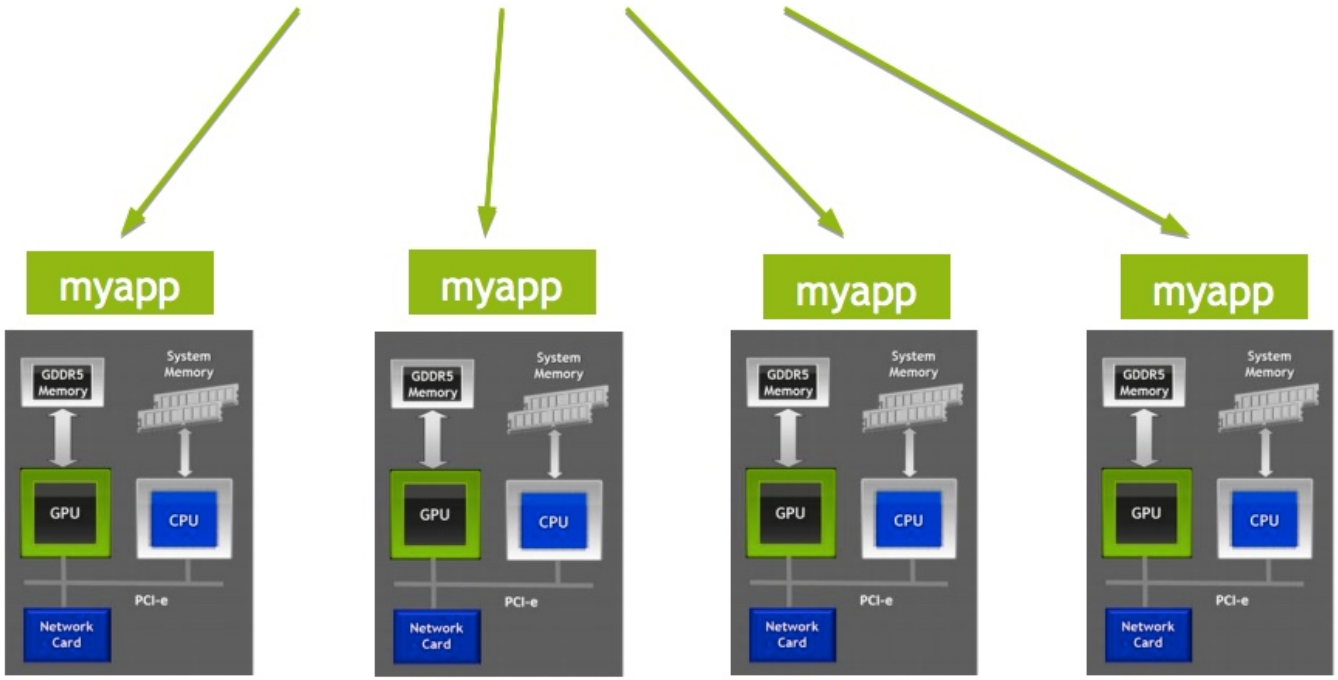
```
#include <mpi.h>
#include <unistd.h>
#include <cstdio>

int main(int argc, char** argv) {
    // initialize MPI on this rank
    MPI_Init(&argc, &argv);
    // get information about our place in the world
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    // print a message
    char name[128]; gethostname(name, sizeof(name));
    printf("hello world from %d of %d on %s\n", rank, size, name);
    // close down MPI
    MPI_Finalize();
    return 0;
}
```

MPI applications are compiled with a **compiler wrapper**:

```
> CC myapp.cpp -o myapp # the Cray C++ wrapper is CC
```

Running our basic MPI application

```
# run myapp 4 ranks (-n) on 4 nodes (-N)
> srun -n4 -N4 ./myapp
hello world from 0 of 4 on nid02117
hello world from 1 of 4 on nid02118
hello world from 2 of 4 on nid02119
hello world from 3 of 4 on nid02120
```



©NVIDIA Corporation

MPI with data in device memory

Use GPUs to parallelize on-node computation and MPI for communication between nodes.

To use with data that is in buffers in GPU memory:

1. Allocate buffers in host memory;
2. Manually copy from device→host memory;
3. Perform MPI communication with host buffers;
4. Copy received data from host→device memory.

A potential optimization:

- Have a CPU thread dedicated to asynchronous host↔device and MPI communication

GPU-aware MPI

GPU-aware MPI implementations can automatically handle MPI transactions with pointers to GPU memory

- MVAPICH 2.0
- OpenMPI since version 1.7.0
- Cray MPI

How it works

- Each pointer passed to MPI is checked to see if it is in host or device memory. If not set, MPI assumes that all pointers are to host memory, and your application will probably crash with segmentation faults
- Small messages between GPUs (up to ≈8 k) are copied directly with **RDMA**
- Larger messages are **pipelined** via host memory

How to use G2G communication

- Set the environment variable `export MPICH_RDMA_ENABLED_CUDA=1`
  - MPI will assume that all pointers are to host memory if this is not set
- Experiment with the environment variable `MPICH_G2G_PIPELINE`
  - Sets the maximum number of 512 kB message chunks that can be in flight (default 16)

MPI with G2G example

```
MPI_Request srequest, rrequest;
auto send_data = malloc_device<double>(100);
auto recv_data = malloc_device<double>(100);

// call MPI with GPU pointers
MPI_Irecv(recv_data, 100, MPI_DOUBLE, source, tag, MPI_COMM_WORLD,
    &rrequest);
MPI_Isend(send_data, 100, MPI_DOUBLE, target, tag, MPI_COMM_WORLD,
    &srequest);
```

Exercise: MPI with G2G

- 2D stencil with MPI in `diffusion/diffusion2d_mpi.cu`
- Implement the G2G version
  1. can you observe any performance differences?
  2. why are we restricted to just 1 MPI rank per node?
- Implement a version that uses managed memory
  - what happens if you don't set `MPICH_RDMA_ENABLED_CUDA`?

```
# load modules require to compile MPI
module swap PrgEnv-cray PrgEnv-gnu
module load cudatoolkit

# launch with 2 MPI ranks
MPICH_RDMA_ENABLED_CUDA=1 srun -C gpu -n2 -N2 --reservation=summer
        diffusion2d_mpi.cuda 8

# plot the solution
source ../../../../scripts/plot.sh

# once it gets the correct results:
sbatch job.batch
```

Exercises: 2D Diffusion with MPI Results

Time for 10,000 time steps 128×131,072 on P100 GPUs

nodes	G2G off	G2G on
1	5.579	5.580
2	3.083	2.811
4	1.909	1.426
8	1.203	0.737
16	0.836	0.399

Using Unified Memory with MPI

- To pass a managed pointer to MPI you must use a GPU-aware MPI distribution
- Even if the managed memory is on the host at time of calling

- The MPI implementation uses page-locked (pinned) memory for RDMA.
- If not aware of unified memory you get
  - if lucky: crashes
  - if unlucky: unexpected behavior