CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

GPUs in Distributed Computing

CSCS Summer School 2023

Andreas Jocksch, Prashanth Kanduri, Radim Janalik & Ben Cumming

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
  - Ex: `MPI_Send`, `MPI_Recv`, `MPI_Sendrecv`, etc.
- Supports global collectives
  - Ex: `MPI_Barrier`, `MPI_Gather`, `MPI_Reduce`, etc.

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
  - Ex: `MPI_Send`, `MPI_Recv`, `MPI_Sendrecv`, etc.
- Supports global collectives
  - Ex: `MPI_Barrier`, `MPI_Gather`, `MPI_Reduce`, etc.

When one starts 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

```c
#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;
}
```

A Basic MPI Application

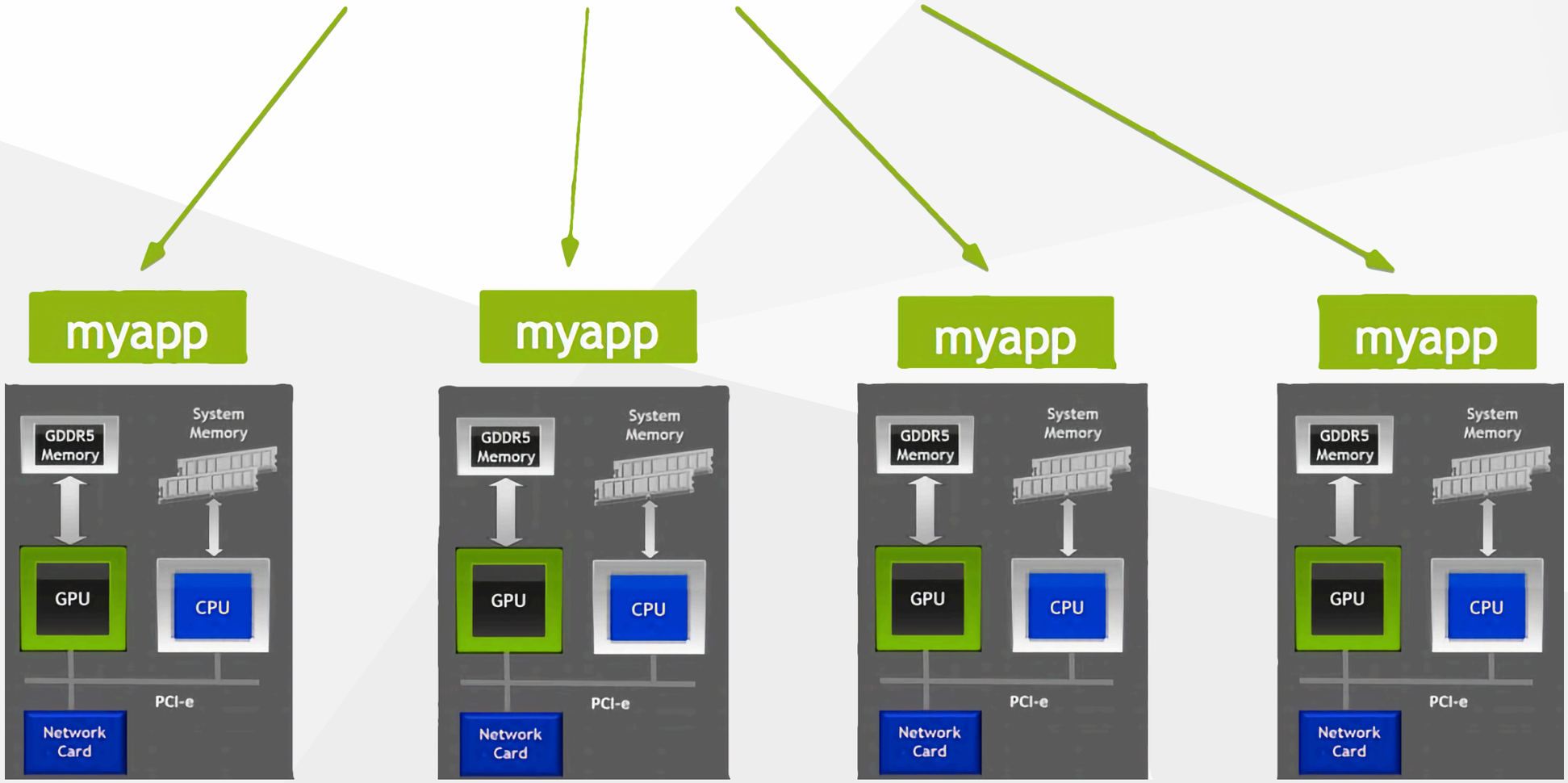
```
#include <mpi.h>
#include <unistd.h>
#include <cstdio>

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 the 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
```



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 $\rightarrow$ host memory;
3. Perform MPI communication with host buffers;
4. Copy received data from host $\rightarrow$ device memory.

This approach can be very fast.

- Have a CPU thread dedicated to asynchronous host $\leftrightarrow$ 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`
  - If not set, MPI assumes that all pointers are to host memory, and your application will probably crash with segmentation faults
- 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);
```

Capabilities & Limitiations

- Support for most MPI API calls (point-to-point, collectives, etc)
- Robust support for common MPI API calls
  - i.e. point-to-point operations
- No support for user-defined MPI data types

Exercise: MPI with GPU-to-GPU (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?
- **Extra++**: find the nasty performance bug...

```
# launch with 2 MPI ranks
MPICH_RDMA_ENABLED_CUDA=1 srun -C gpu -n2 -N2 --reservation=summeruni diffusion2d_mpi 8

# plot the solution
python plotting.py
```