CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Network performance and multiple GPUs

Summer School 2018 – High Performance and Parallel GPU Computing

Maxime Martinasso, CSCS

July 24, 2018

Course Objectives

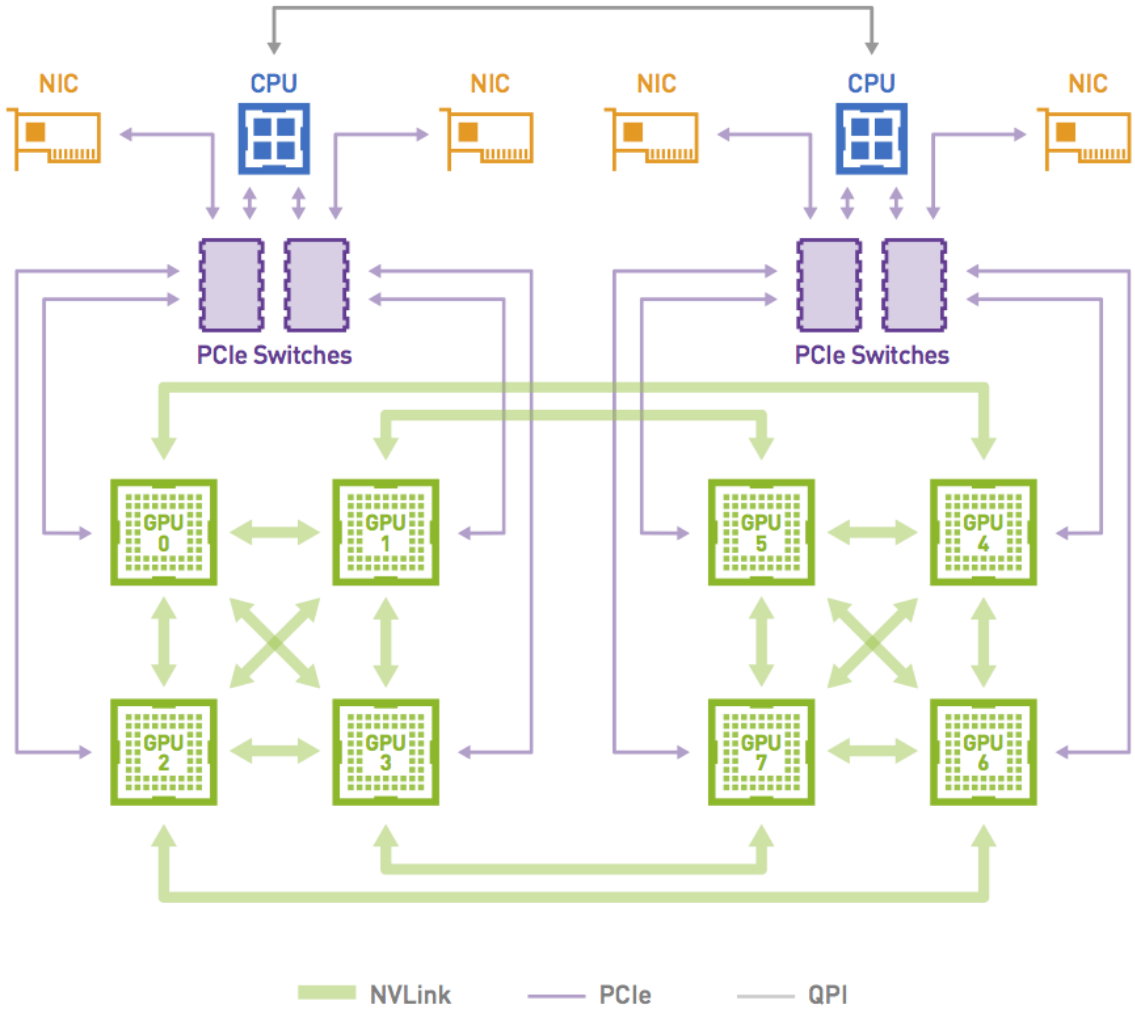
- Understand network performance and architecture, multiple GPUs

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

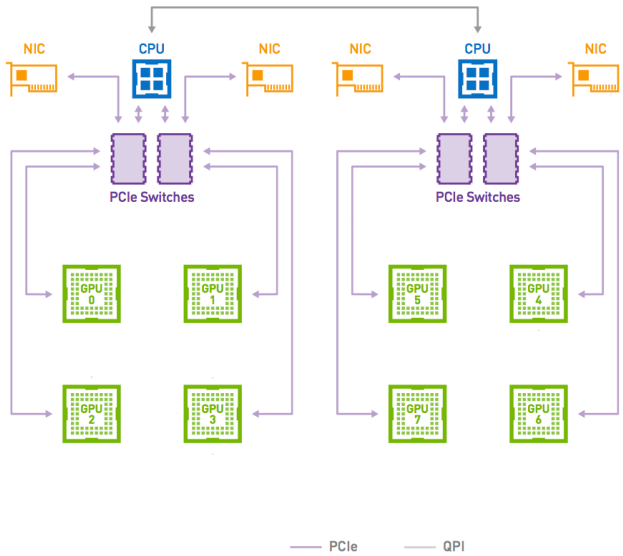
Network architecure

Muliple GPUs



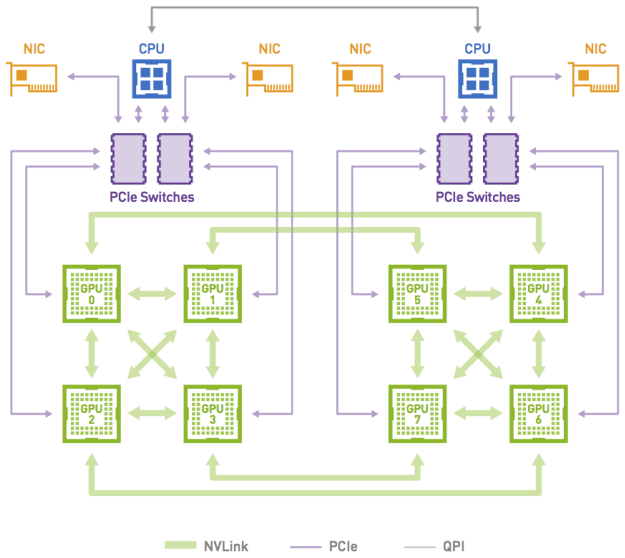
PCIe link: 16GB/s QPI: 25.6GB/s NVLink: 20GB/s

PCI express



- Connects GPU to CPU socket and GPUs among themselves
- Per link: Gen3 1GB/s, Gen4 2GB/s and Gen5 4GB/s
- GPUs and CPU sockets connected with 16 links (usually)
- Switches with different number of ports
- Cuda GPU PeerToPeer enable only on the same socket (root complex)

NVlink



- 4 links per GPUs
- 20 GB/s per link v1.0 and 25 GB/s for v2.0
- Several topology possible
- No automatic routing (manually in the application)
- NVSwitch fully connects 16 GPUs

HPC network

- InfiniBand: Fat-tree, FDR 7GB/s, EDR 12.5GB/s, HDR 25GB/s
- Cray Aries: Dragonfly, 12GB/s
- Others: Intel Omni-Path, 100 Gigabit Ethernet

OpenFabrics Interfaces (libfabric)

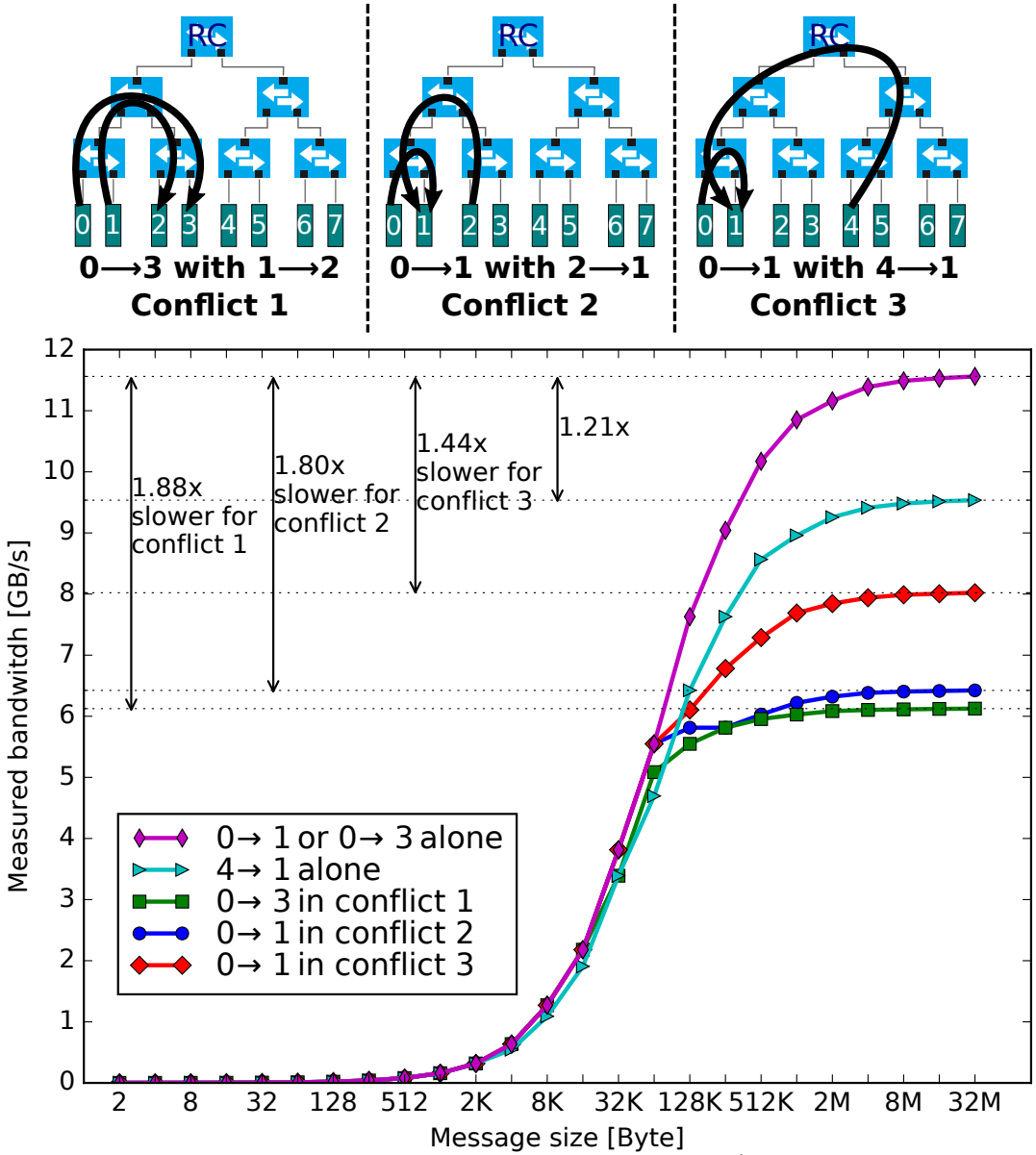
Performance of network

Bandwidth and latency

- Bandwidth [GB/s]: the rate of data transfer
- Latency [s]: network delay

Simple performance model: $T = L + M/Bw$

Example of performance variability - PCIe



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Communication model and performance

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

This approach can be very fast.

- 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

- GPUdirect Peer to Peer among GPUs on the same node
- GPUdirect RDMA among GPUs on different nodes
- MVAPICH, OpenMPI, 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 and Limitations

- 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 G2G

- 2D stencil with MPI in `08.MPI_Network/cxx/1.diffusion2d_mpi.cu`
- Implement the G2G version
    1. can you observe any performance differences?

Exercises: 2D Diffusion with MPI Results

Time for 10,000 time steps $128 \times 131{,}072$ 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

Communication model and software stack

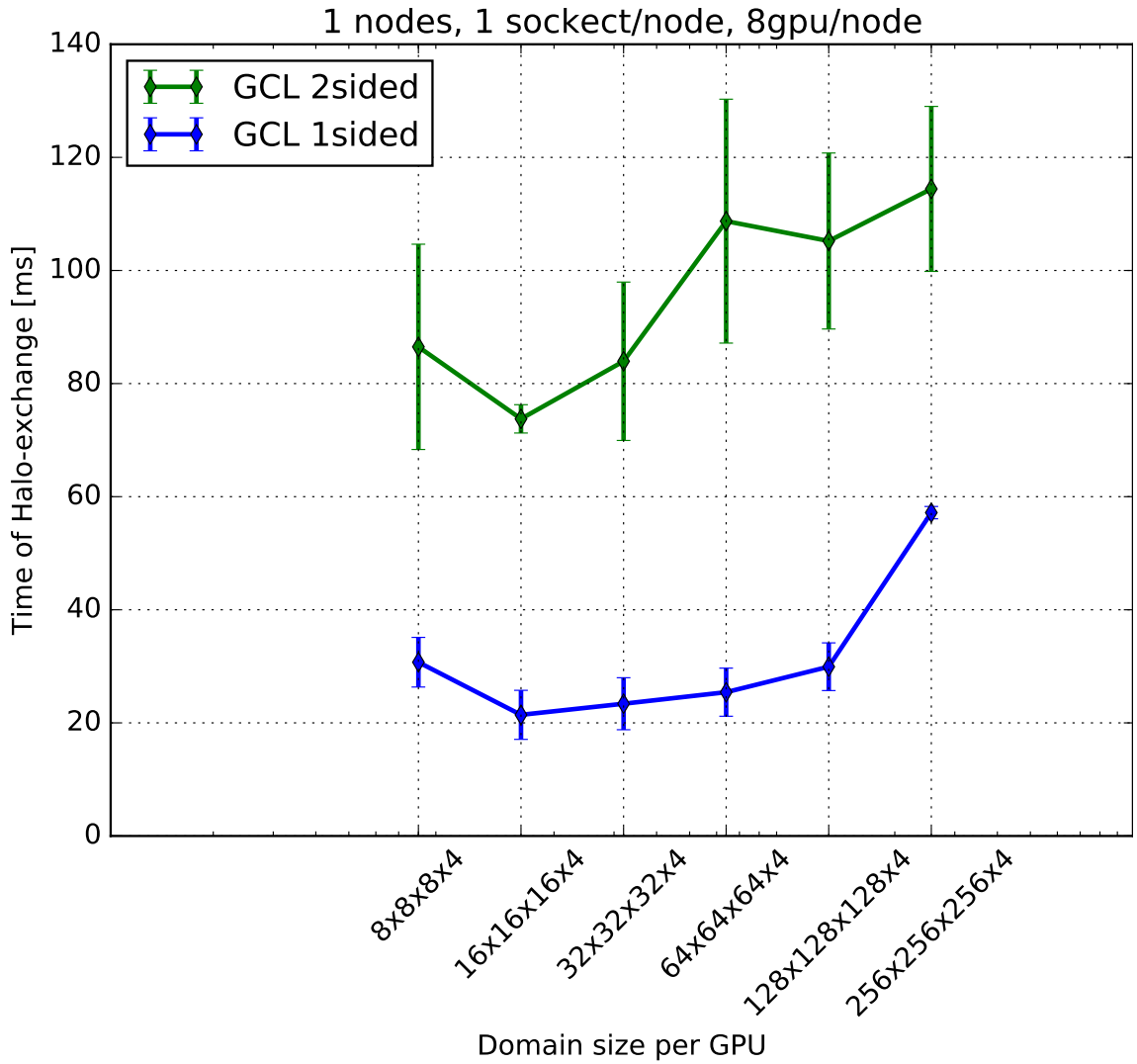
- MPI use cuda as a communication back-end
- Cuda library is one-sided one rank should know the source and target buffers:

```
cudaMemcpy(dst, src, count, kind)
```

What happens when using MPI two-sided?

- Uses IPC to communicate buffer adresses
- Lower down the performance

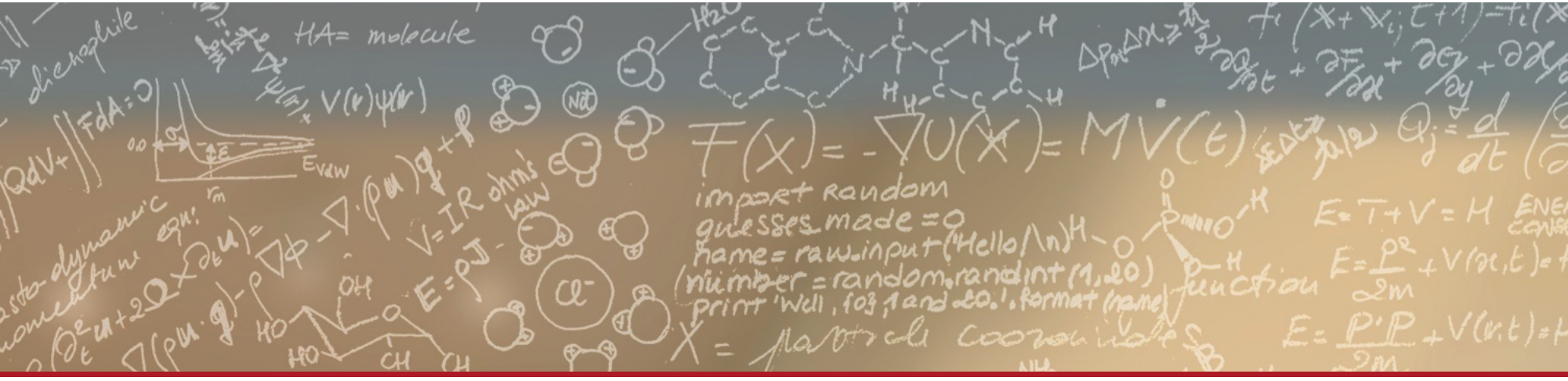
Example on halo-exchange



Exchanges 4 halo layers. Exchanges are periodic in all directions.

Misc.

- NVidia NCCL: Optimized primitives for collective multi-GPU communication: all-reduce, all-gather, reduce-scatter, reduce, broadcast
- OSU (Ohio State University) benchmarks for MPI



Thank you for your attention.