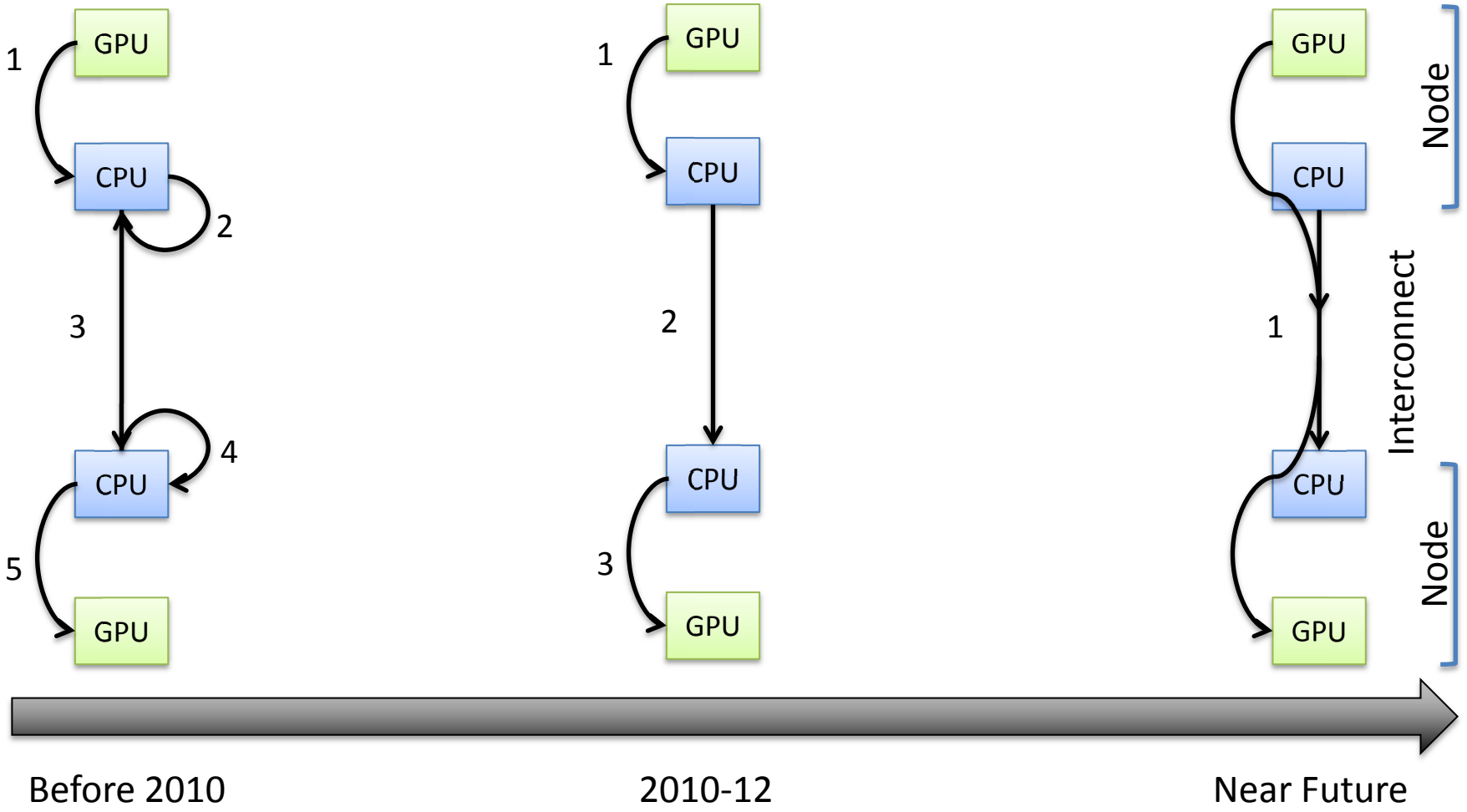
Use Cases and Status of GPUDirect:

A High Productivity & Performance Feature for GPU Supercomputing

Sadaf Alam and Mauro Bianco

Swiss National Supercomputing Centre (CSCS)

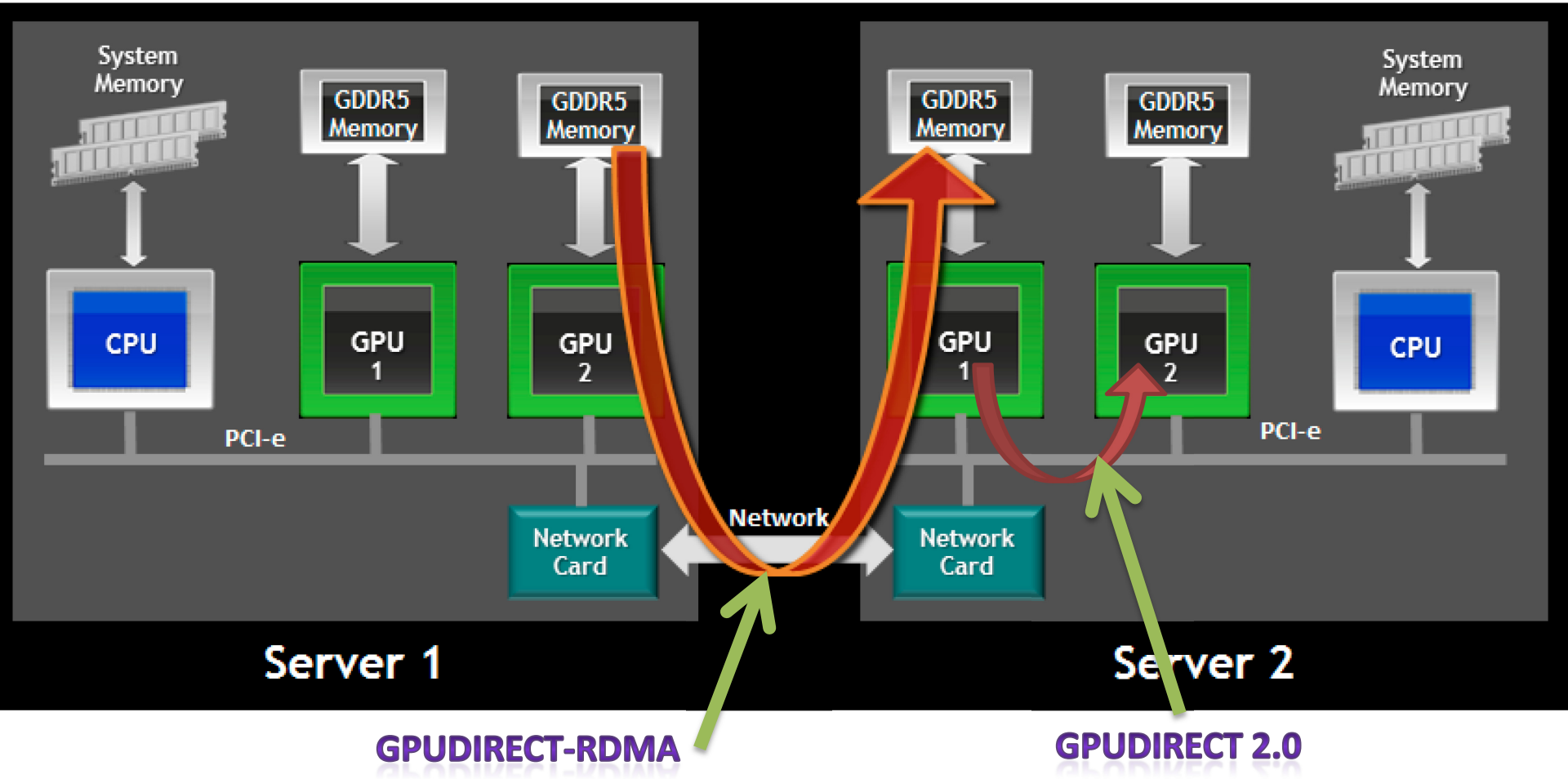
Evolution of Direct GPU Communication



Details at http://developer.nvidia.com/cuda/nvidia-gpudirect

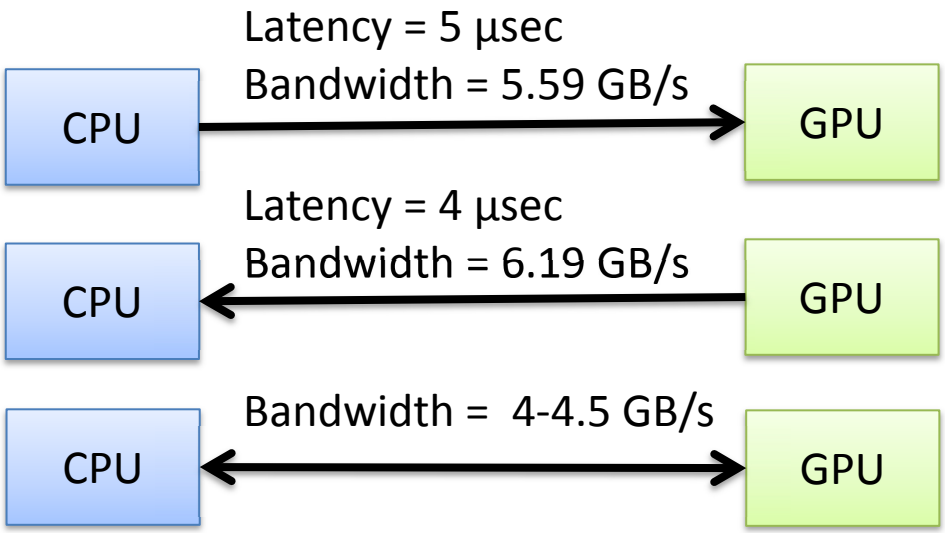
GPUDirect™ Support for RDMA, Introduced with CUDA 5

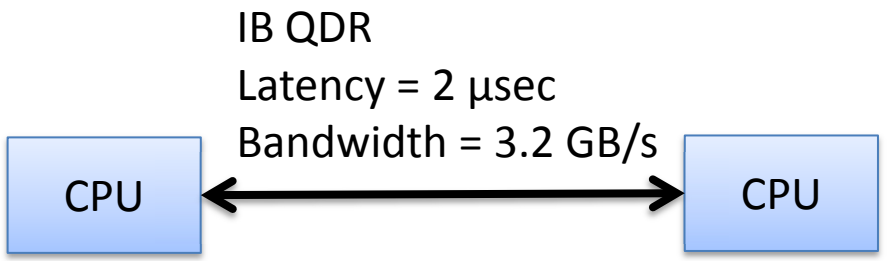
No implementation available to test yet!

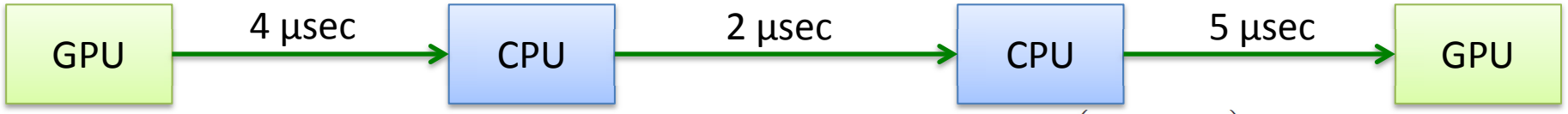


Details at http://developer.nvidia.com/cuda/nvidia-gpudirect

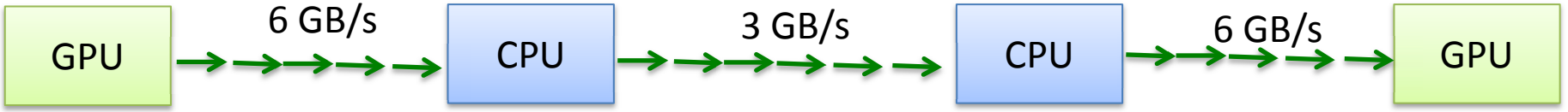
Revisiting Latencies and Bandwidth





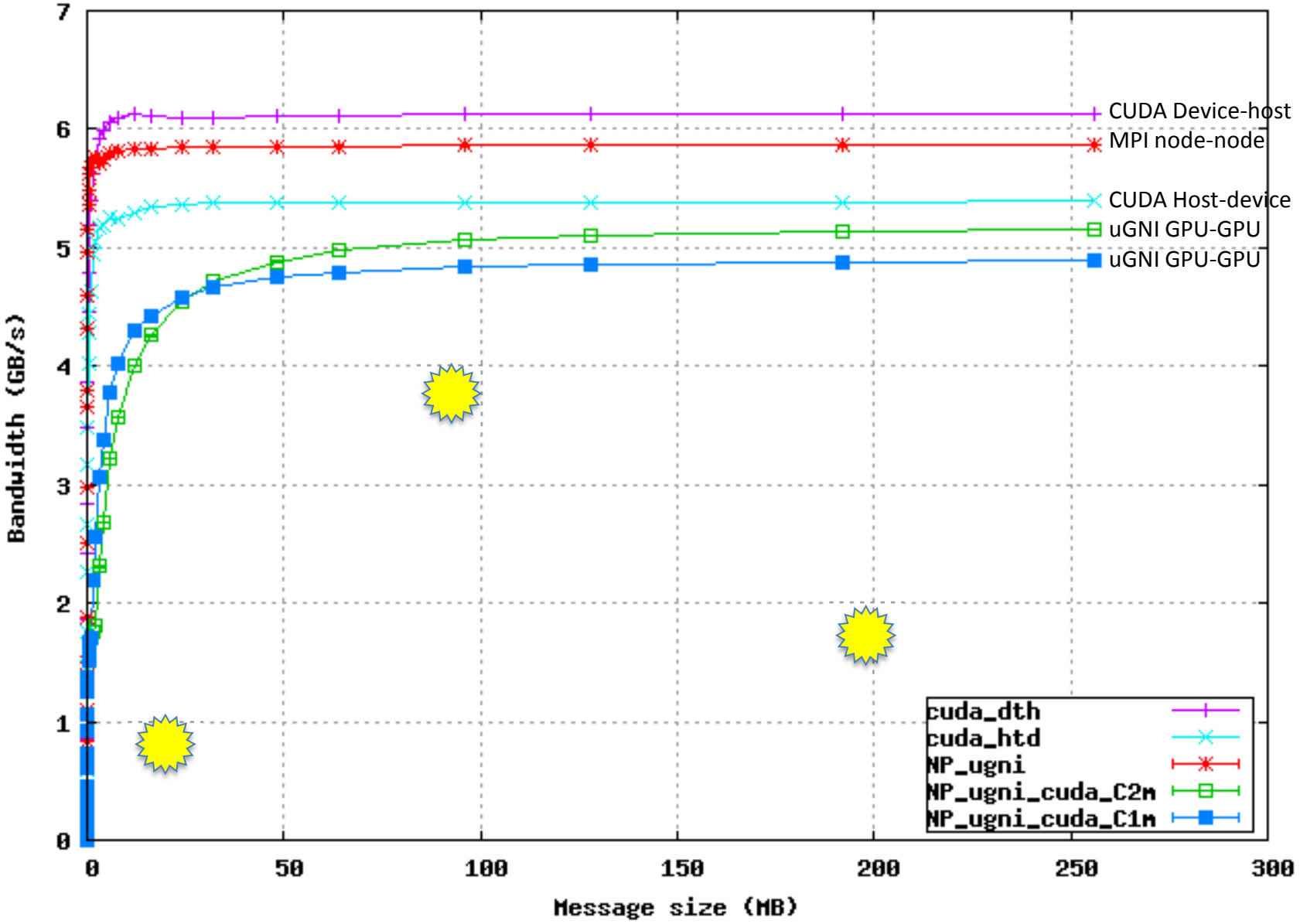


$$Latency_{total} = MIN\left(L_{\substack{CPU-GPU \\ pinned}}\right) + MIN\left(L_{\substack{GPU-CPU \\ pinned}}\right) + MIN\left(L_{CPU-CPU}\right)$$



$$Latency_{total} = MAX\left\{MIN\left(L_{CPU-GPU}\right), MIN\left(L_{GPU-CPU}\right), MIN\left(L_{CPU-CPU}\right)\right\}$$

Bandwidth between two GPU nodes on Cray XK6 platform



CUDA with GPUDirect MPI

Reducing Memory allocation

```
// allocate temperature array on host
temp1_h = (float *) malloc (sizeof (float) * ni * nj);
temp2_h = (float *) malloc (sizeof (float) * ni * nj);

// allocate buffer arrays on host
buf_left_in_h = (float *) malloc (sizeof (float) * (nj - 2));
buf_right_in_h = (float *) malloc (sizeof (float) * (nj - 2));
buf_left_out_h = (float *) malloc (sizeof (float) * (nj - 2));
buf_right_out_h = (float *) malloc (sizeof (float) * (nj - 2));

// allocate temperature arrays on device
cudaMalloc ((void **) &temp1_d, sizeof (float) * ni * nj);
cudaMalloc ((void **) &temp2_d, sizeof (float) * ni * nj);

// allocate buffer arrays on device
cudaMalloc ((void **) &buf_left_in_d, sizeof (float) * (nj - 2));
cudaMalloc ((void **) &buf_right_in_d, sizeof (float) * (nj - 2));
cudaMalloc ((void **) &buf_left_out_d, sizeof (float) * (nj - 2));
cudaMalloc ((void **) &buf_right_out_d, sizeof (float) * (nj - 2));
```

Reducing Extra Transfers

```
gather_gpu_kernel <<< grid_dim, block_dim >>> (nj - 2, buf_left_d,
                                               list_left_d, temp_d);
gather_gpu_kernel <<< grid_dim, block_dim >>> (nj - 2,
buf_right_d,   list_right_d, temp_d);

// copy buffers to cpu
cudaMemcpy ((void *) buf_left_h, (void *) buf_left_d,
        sizeof (float) * (nj - 2), cudaMemcpyDeviceToHost);
cudaMemcpy ((void *) buf_right_h, (void *) buf_right_d,
        sizeof (float) * (nj - 2), cudaMemcpyDeviceToHost);
```

Modifying MPI Calls

```
MPI_Irecv (buf_right_in_h, nj - 2, MPI_FLOAT, mpi_rank + 1,
           2 * mpi_rank + 1, MPI_COMM_WORLD, &req_in[1]);

MPI_Isend (buf_right_out_h, nj - 2, MPI_FLOAT, mpi_rank + 1,
           2 * (mpi_rank + 1), MPI_COMM_WORLD, &req_out[1]);
```

OpenACC with GPUDirect MPI (I)

Example # 1: Existing OpenACC Applications

```
MPI_Irecv (recv, n, MPI_FLOAT, dest, 0, MPI_COMM_WORLD,
&request);

// Run on the accelerator
#pragma acc parallel loop copyin(in[0:n]) copyout(out[0:n])
 for (i = 0; i < n; i++)
  {
   out[i] = in[i] * 4.0f;
  }

// Get expected results from receiving host

 MPI_Send (out, n, MPI_FLOAT, dest, 0, MPI_COMM_WORLD);

 MPI_Wait(&request,&status);
```

No Change

Profile Output

method=[memcpyHtoDasync] gputime=[177.312] cputime=[7.000]

method=[memcpyHtoDasync] gputime=[177.472] cputime=[4.000]

method=[memcpyHtoDasync] gputime=[177.952] cputime=[4.000]

method=[memcpyHtoDasync] gputime=[144.992] cputime=[4.000]

method=[main_72_gpu] gputime=[81.856] cputime=[13.000] occupancy=[1.000]

method=[memcpyDtoH] gputime=[1238.080] cputime=[2253.000]

Compiler Output

```
72, Accelerator kernel generated
      72, CC 1.0 : 7 registers; 44 shared, 0 constant, 0 local memory bytes
          CC 2.0 : 14 registers; 0 shared, 60 constant, 0 local memory bytes
      73, #pragma acc loop gang, vector(256) /* blockIdx.x threadIdx.x */
   72, Generating present_or_copyout(out[0:n])
      Generating present_or_copyin(in[0:n])
      Generating compute capability 1.0 binary
      Generating compute capability 2.0 binary
```

Note: All OpenACC experiments are performed using the PGI OpenACC compiler and MVAPHC2-GPU library. Behavior could be different with Cray and CAPS OpenACC compiler.

OpenACC with GPUDirect MPI (II)

Example # 2 : Converting non-accelerated code to GPU-enabled OpenACC code

```
#include <mpi.h>
#include "openacc.h"

…
 s_buf = (char *) acc_malloc(sizeof(char) * MYBUFSIZE);
 r_buf = (char *) acc_malloc(sizeof(char) * MYBUFSIZE);

…
#pragma acc parallel loop deviceptr(s_buf,r_buf)
        for(i = 0; i < size; i++) {
           s_buf[i] = 'a';
           r_buf[i] = 'b';
        }
…
MPI_Recv(r_buf, size, MPI_CHAR, 1, 1, MPI_COMM_WORLD, &reqstat);
…
MPI_Send(s_buf, size, MPI_CHAR, 0, 1, MPI_COMM_WORLD);
…

 acc_free(s_buf);
 acc_free(r_buf);
```

Note: All OpenACC experiments are performed using the PGI OpenACC compiler and MVAPHC2-GPU library. Behavior could be different with Cray and CAPS OpenACC compiler.

Experimental Setup

- System
  - IBM System x iDataPlex dx360 M3
  - Node: 2 Intel Westmere, 2 NVIDIA M2090
  - Interconnect: IB QDR
- GPU Aware MPI Library
  - MVAPICH2-GPU version 1.9a
- Compilers
  - CUDA 4.2
  - PGI 12.9 for OpenAC
  - CAPS 3.2.1

Use Cases

- CUDA accelerated code
  - LAMMPS (Molecular Dynamics)
  - GSCL (Generic Stencil Computation Library)
    - Communication layer for COSMO (a climate and weather modeling application)
- OpenACC accelerated code
  - GSCL for COSMO Physics
  - Hydro: PRACE Application Benchmark
  - MPI micro-benchmarks (IMB, OSU, …)

Latency and Bandwidth Measurements B/W Host (H) and Device (D)

memcpyDtoHasync memcpyDtoDasync

	H—H (1 Node)	H—D (1 Node)	D—H (1 Node)	D—D (1 Node)	H—H (2 Nodes)	D—D (2 Nodes)
Latency (8 Bytes)	0.26	21.49	21.44	14.51	2.01	40.88
Latency (512 Bytes)	0.38	23.56	23.63	14.52	4.42	43.00

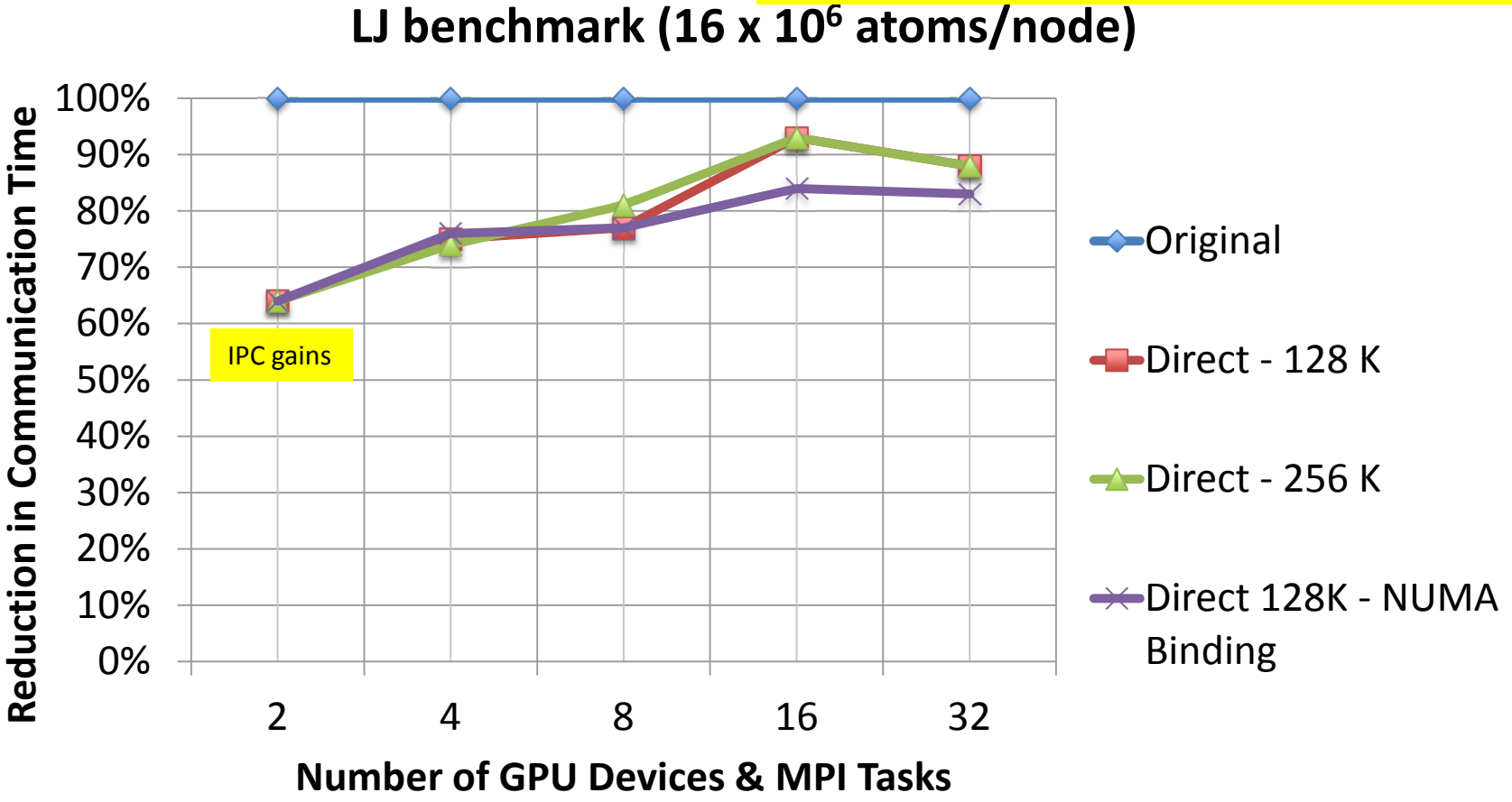
Latency µsec

	H—H (1 Node)	H—D (1 Node)	D—H (1 Node)	D—D (1 Node)	H—H (2 Nodes)	D—D (2 Nodes)
Bandwidth (2 KBytes)	4732.75	67.70	14.30	139.43	1641.45	55.69
Bandwidth (1 MBytes)	8760.59	3181.09	2833.32	5845.62	3205.57	2827.13

Bandwidth MB/s

LAMMPS Results

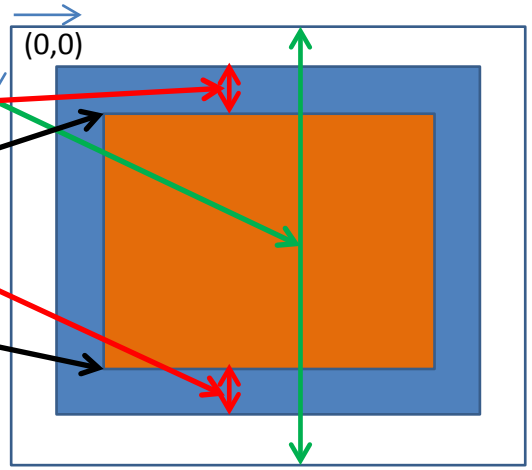
Total runtime improvement of up to 23%



Generic Communication Layer

- Provide halo exchange for regular grids
  - Used in COSMO (Dycore and Physics) and GSCL
- Based on `halo_descriptor`
  - For each dimension of the data:

```
struct halo_descriptor {
  halo_descriptor(int m, int p, int b, int e, int l) { … }
  …
  const int total_length() const { … }
  const int minus() const { … }
  const int plus() const { … }
  const int begin() const { … }
  const int end() const { … }
};
```



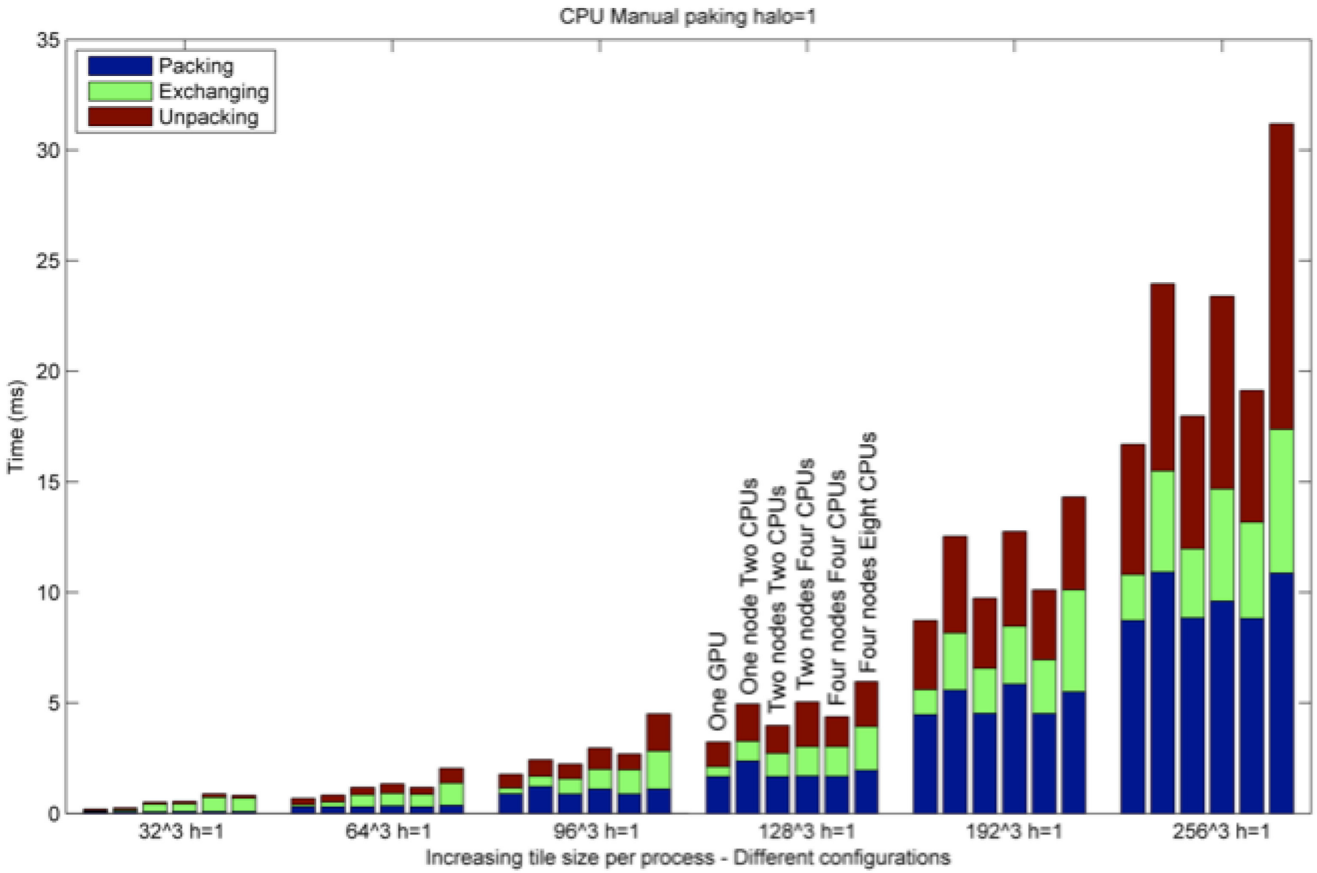
Halo Exchange Pattern

- Allow to specify at compile time
  - Data layout
  - Process grid orientation
  - Data location (CPU, GPU, …)
  - Packing/Unpacking version
    - Manual
    - MPI_Pack/Unpack
    - Single Datatype
  - Number of dimesions
  - Datatype (any POD)

```
template <typename layout_map,
          typename proc_map,
          typename DataType,
          int DIMS,
          typename Gcl_Arch,
          int version>
  class halo_exchange_dynamic {
  public:
    void add_halo( ... ) {...}
    void pack( ... ) {...}
    void unpack( ... ) {...}

    void start_exchange() {...}
    void wait() {...}
  };
```

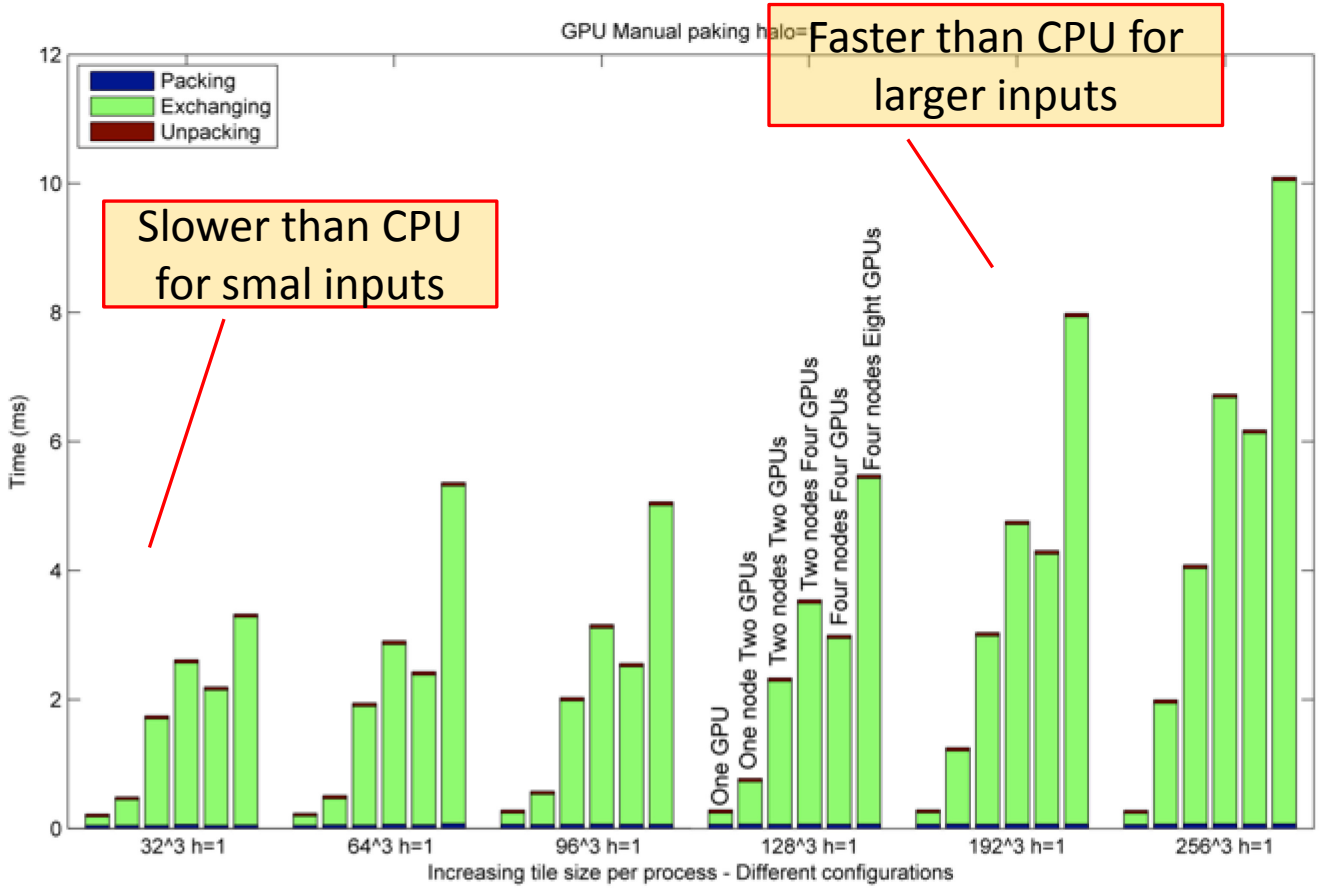
Manual Packing on CPU



~30 ms Manual CPU

- On this machins one process per node is better

Manual Packing on GPU



~30 ms Manual CPU

~10 ms Manual CPU

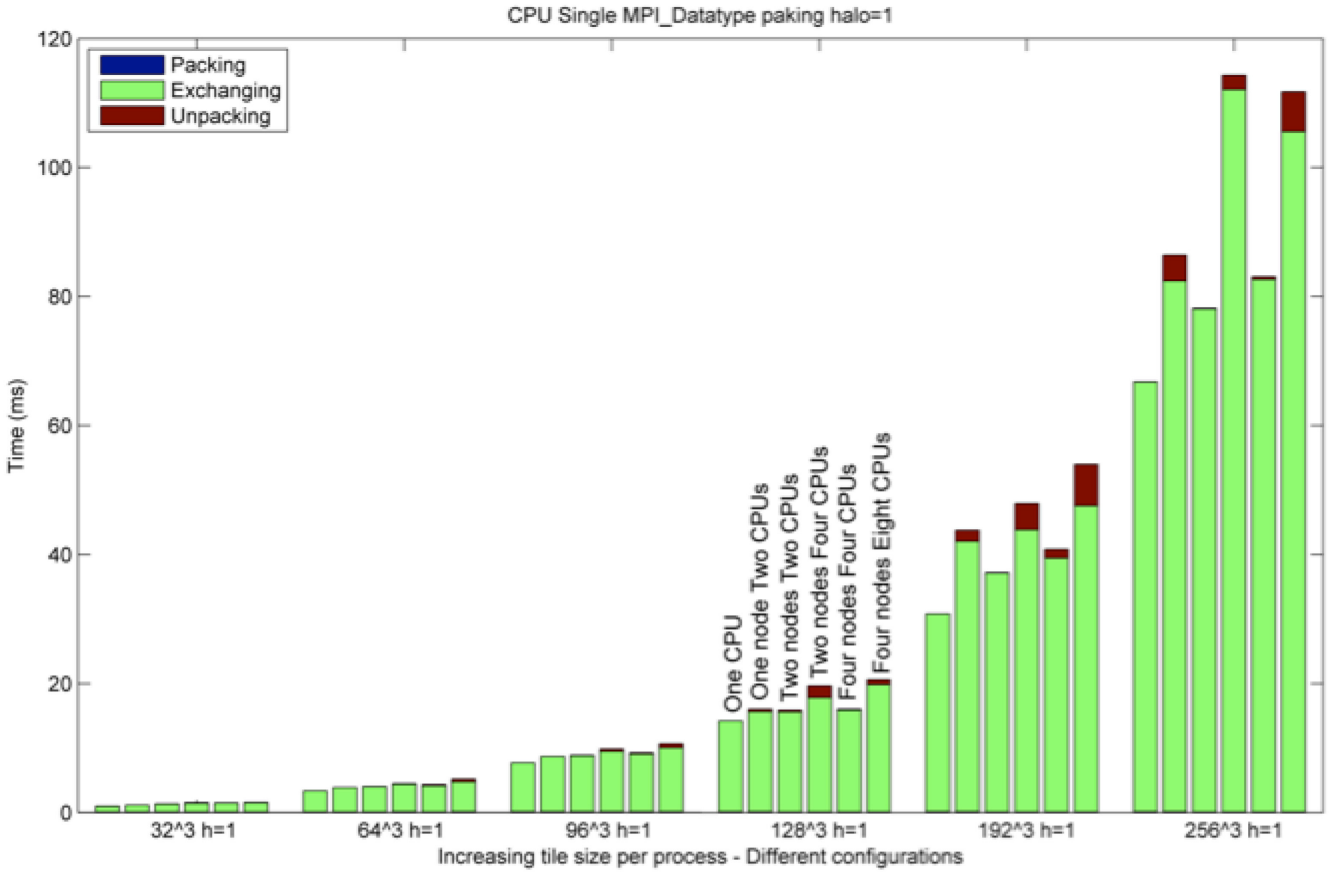
- Up to 6 CUDA kernels and 26 MPI calls

MPI Datatype on CPU

~120 ms MPI Datatypes CPU

~30 ms Manual CPU

~10 ms Manual CPU



- MPI Datatypes not fast as Manual (OpenMPI is actually faster)

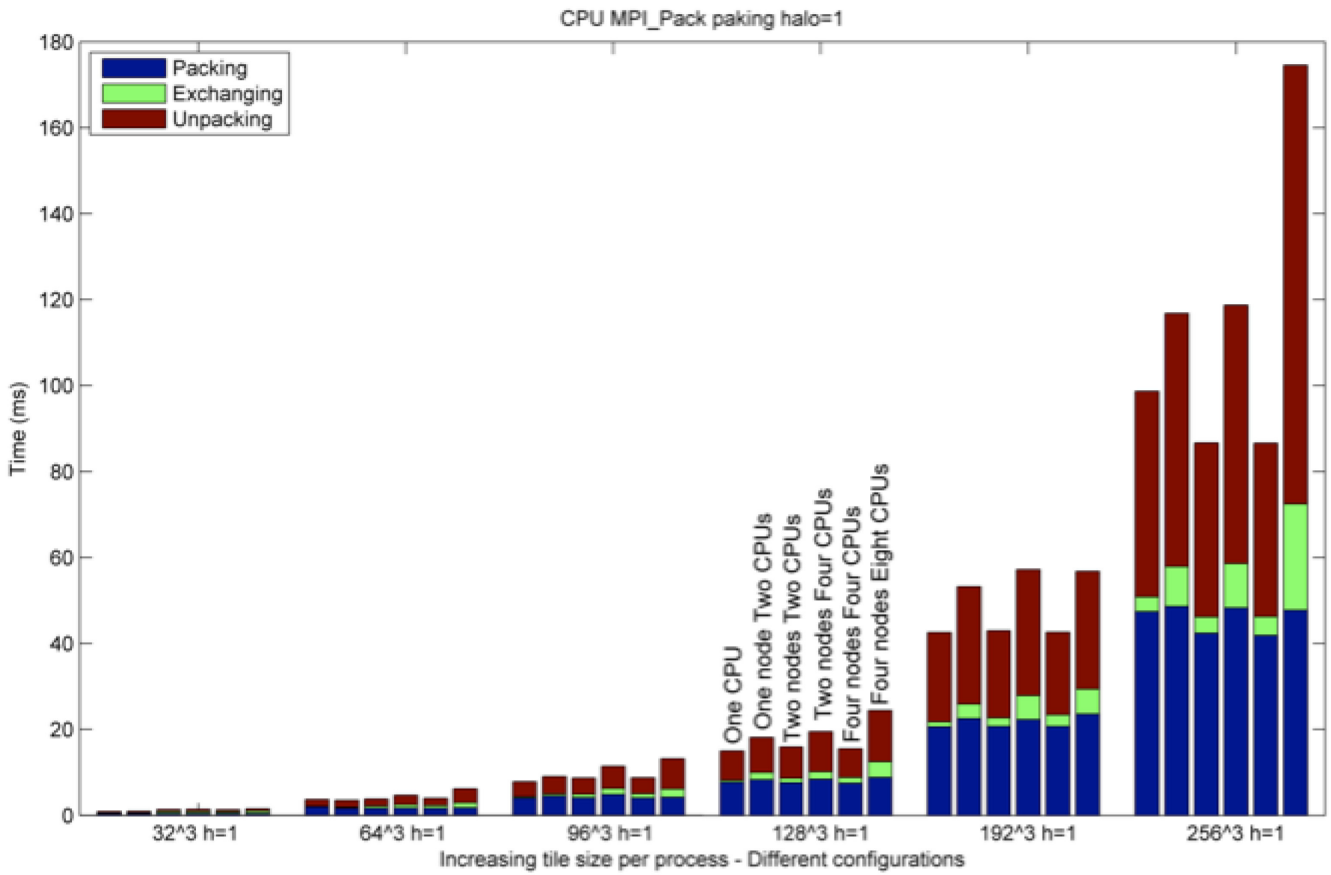
MPI Pack/Unpack on CPU

~180 ms MPI Pack/Unpack CPU

~120 ms MPI Datatypes CPU

~30 ms Manual CPU

~10 ms Manual CPU



MPI Datatype on GPU

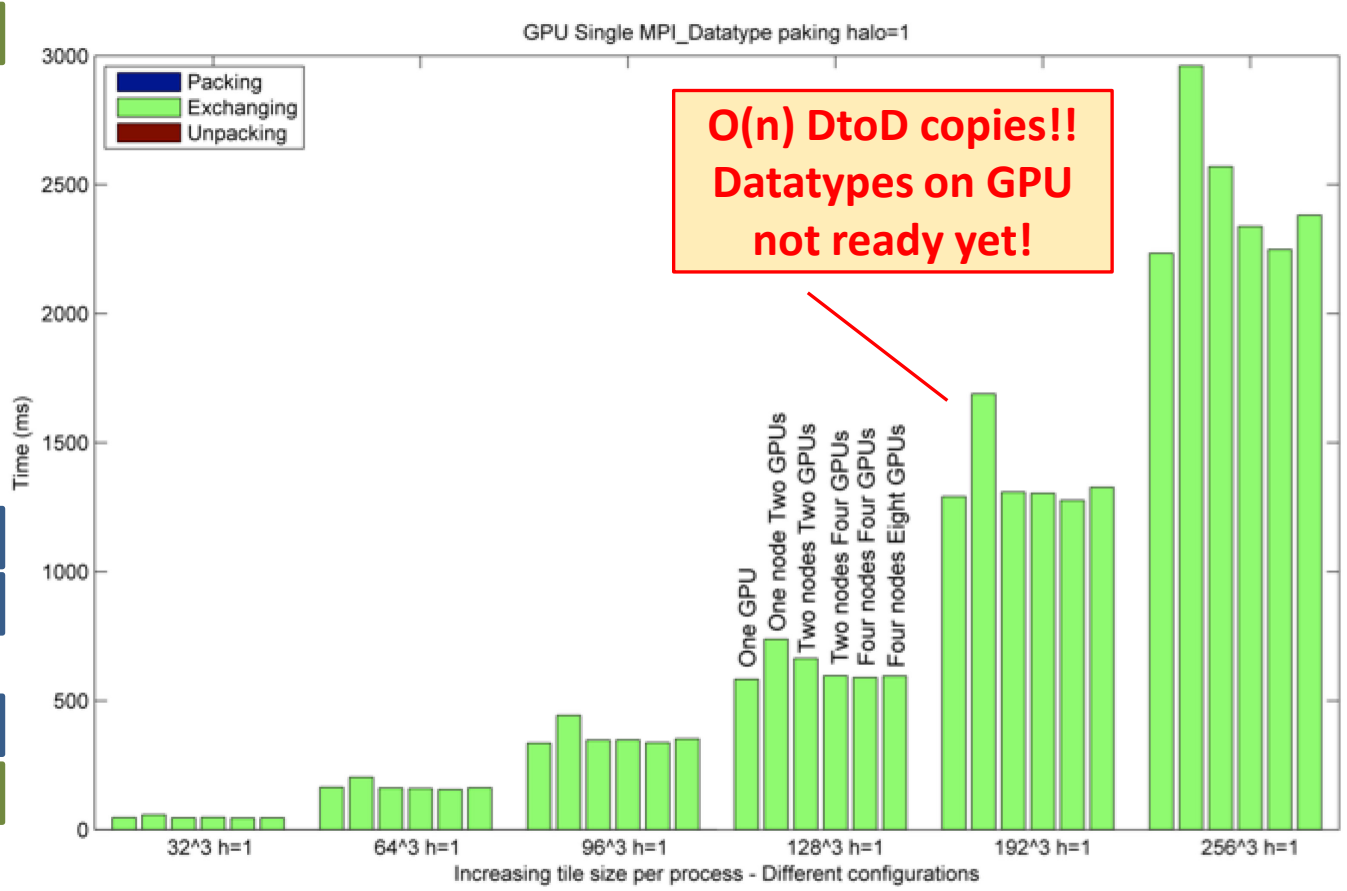
~3000 ms MPI Datatypes GPU

~180 ms MPI Pack/Unpack CPU

~120 ms MPI Datatypes CPU

~30 ms Manual CPU

~10 ms Manual CPU



MPI Pack/Unpack on GPU

~3000 ms MPI Datatypes GPU

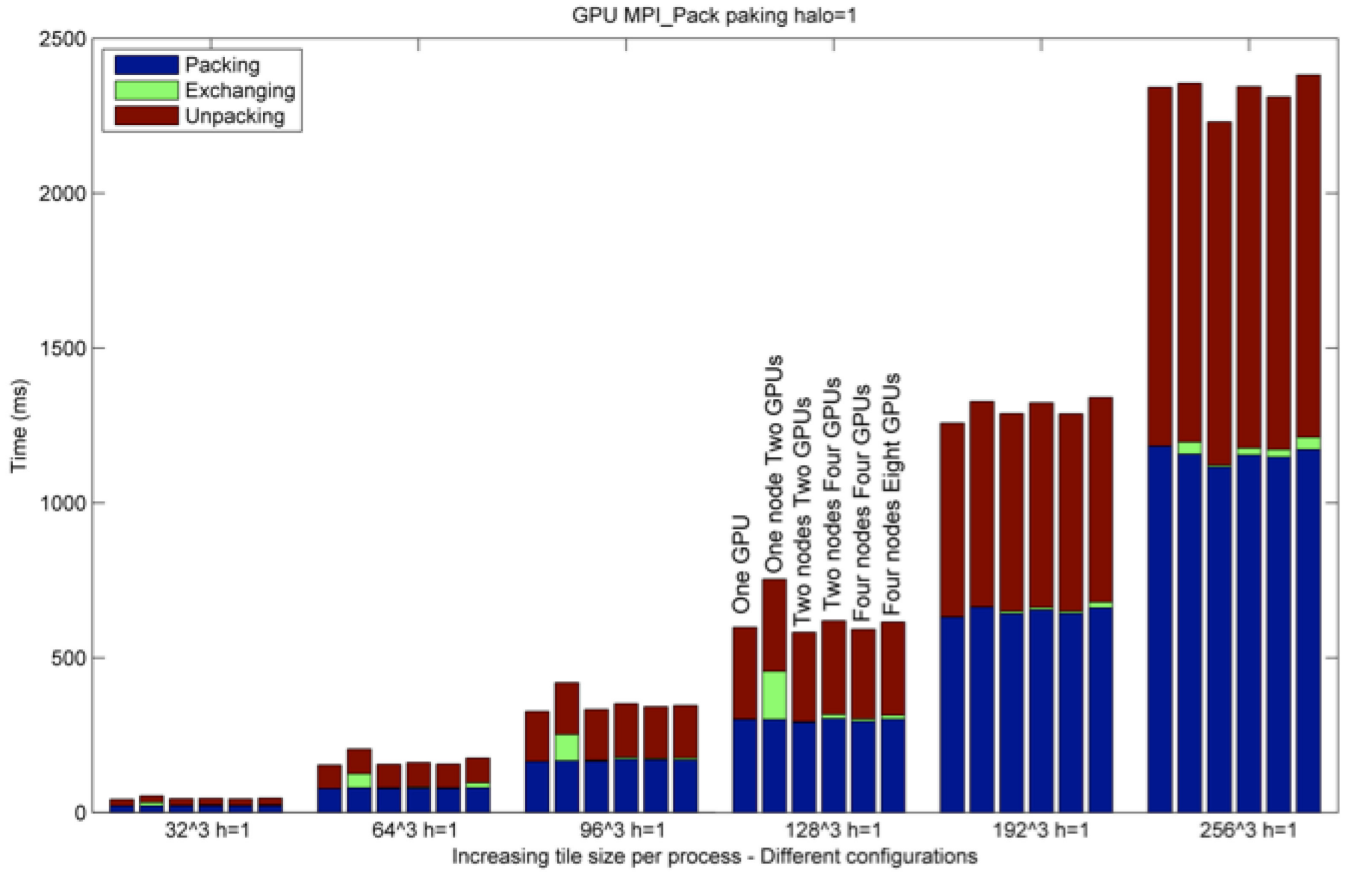
~2500 ms MPI Pack/Unpack GPU

~180 ms MPI Pack/Unpack CPU

~120 ms MPI Datatypes CPU

~30 ms Manual CPU

~10 ms Manual CPU



Major Issues and Challenges

- Using OpenACC in templated CUDA and C++ applications

NVIDIA & OpenACC Compilers

  - NVCC host_defines.h errors with Cray and PGI compilers
  - Uniform support for OpenACC standard

- MPI Datatypes performance

CRAY & MPI Libraries

  - Poor implementation in MPICH2, MVAPICH2 and Cray MPI
  - Portability and tuning constraints for manual packing and unpacking

Current Status & Dependencies

	Cray XK6/XK7	IB Cluster (Fermi)	IB Cluster (Kepler)*
GPUDirect Enabled Cray MPI	✗ ✔	✗ ✔	✗ ✔
GPUDirect Enabled MPI (MVAPICH2)	✗	✔	✔
Cray OpenACC Compilers	✔	✗	✗
PGI OpenACC Compilers	✔	✔	✔
CAPS OpenACC Compilers	✔	✔	✔
OpenCL Support	✔	✔	✔

* Does not exist at the moment

Final Thoughts

Today

Near Future

Acknowledgements

- Antonio J. Peña
  Technical University of Valencia

- DK Panda, Devendar Bureddy, Sreeram Potluri
  Ohio State University

THANK YOU