Grid Computing

2011/12/5

金光浩(11D37060 松岡研究室)

Optimized Non-contiguous MPI Datatype Communication for GPU Clusters: Design, Implementation and Evaluation with MVAPICH2

Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Xiangyong O
uyang,Sayantan Sur, Dhabaleswar K. Panda

In *IEEE International Conference on Cluster Computing,*2011

Related

MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters

HaoWang · Sreeram Potluri · Miao Luo ·Ashish Kumar Singh · Sayanta
n Sur ·Dhabaleswar K. Panda

MVAPICH 2

http://mvapich.cse.ohio-state.edu/overview/mvapich2/

MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE

Network-Based Computing Laboratory
Department of Computer Science and Engineering

MVAPICH2 (MPI-2 over OpenFabrics-IB, OpenFabrics-iWARP, PSM, uDAPL and TCP/IP)

This is an MPI-2 implementation (conforming to MPI 2.2 standard) which includes all MPI-1 features. It is based on MPICH2 and MVICH.

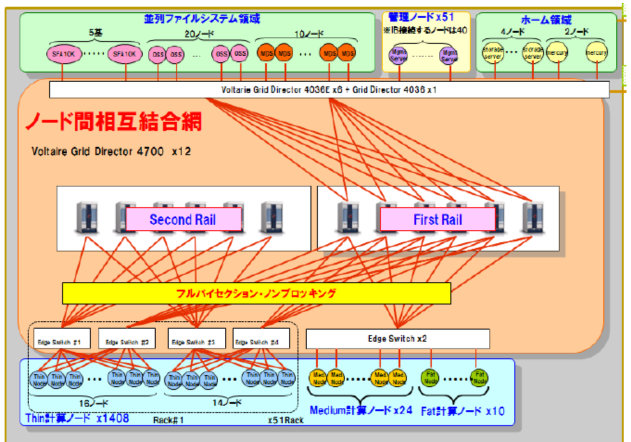
MVAPICH2 1.8 provides many features including high-performance communication

Support for NVIDIA GPU.

The computing nodes of TSUBAME2 are connected with

the InfiniBand device 'Grid Director 4700' made by Voltaire inc.

MVAPICH2: MVAPICH 1.5.1+intel



TSUBAME2.0ネットワーク全体図

Communication

Code without MPI integration

At Sender:

```
cudaMemcpy(s_buf, s_device, size, cudaMemcpyDeviceToHost);
MPI_Send(s_buf, size, MPI_CHAR, 1, 1, MPI_COMM_WORLD);
```

At Receiver:

```
MPI_Recv(r_buf, size, MPI_CHAR, 0, 1, MPI_COMM_WORLD, &req);
cudaMemcpy(r_device, r_buf, size, cudaMemcpyHostToDevice);
```

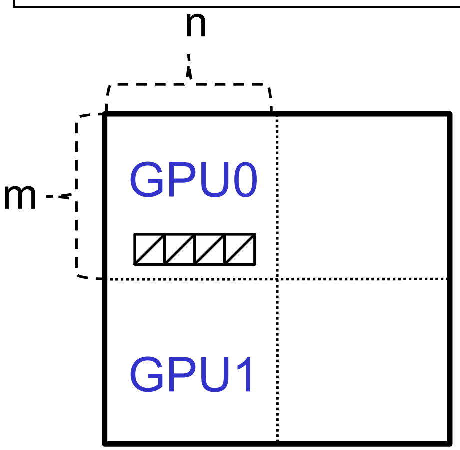
Code with MPI integration

At Sender:

```
MPI_Send(s_device, size, ...);
```

At Receiver:

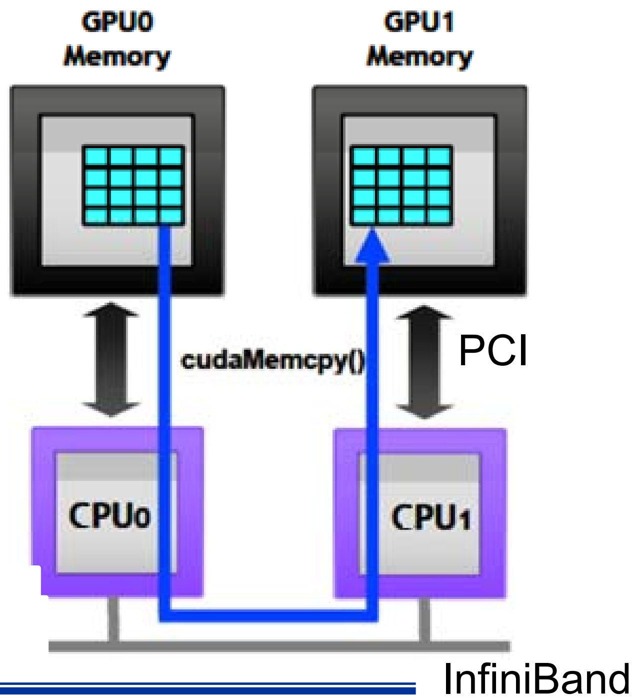
```
MPI_Recv(r_device, size, ...);
```



MPI_Send(&device[(m-1)*n],n, …)

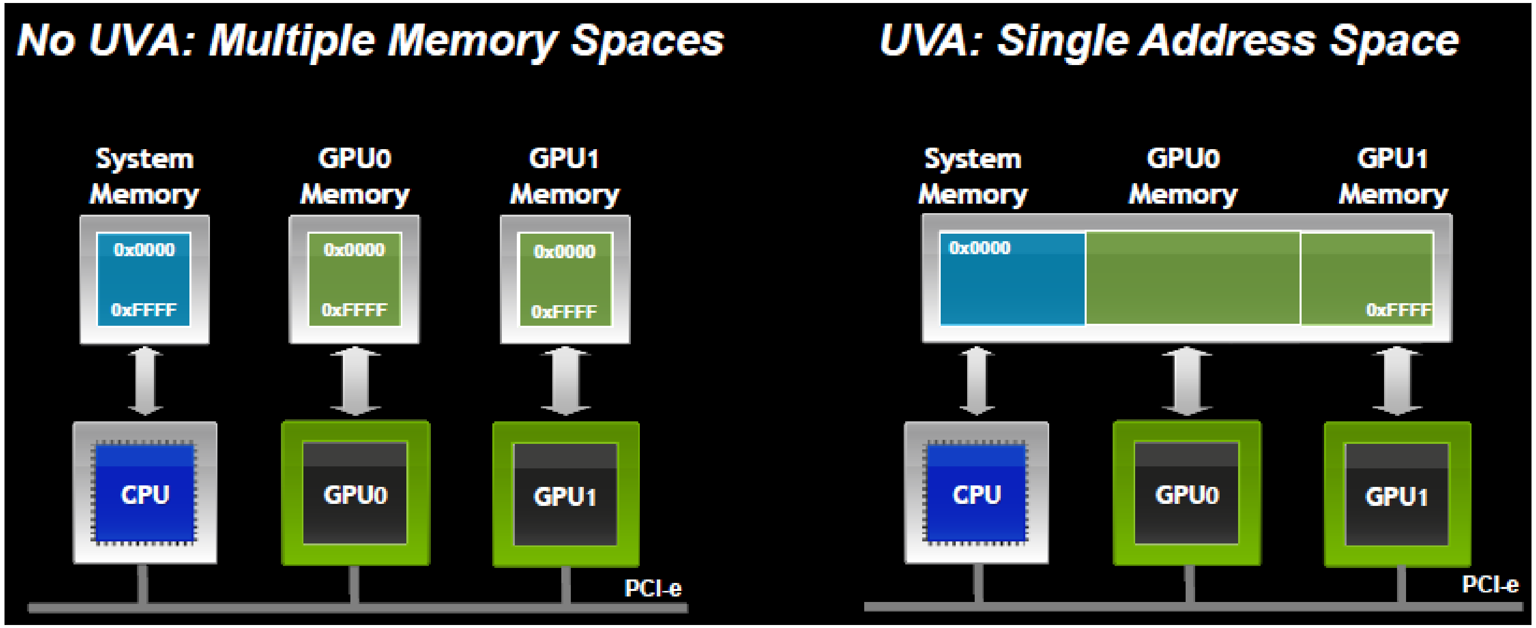
MVAPICH2 can further optimize the performance of GPU to GPU communication.This is achieved by pipelining tran sfers from GPU to host memory, host memory to remote h ost memory Via InfiniBand and finally from remote host to destination GPU memory.

Direct Transfers



CUDA 4.0 direct copy

Unified Virtual Addressing (UVA) :



UVA can also be used to find out if a particular buffer was allocated in the GPU memory or in the host memory.

One address space for all CPU and GPU memory

Determine physical memory location from pointer value.

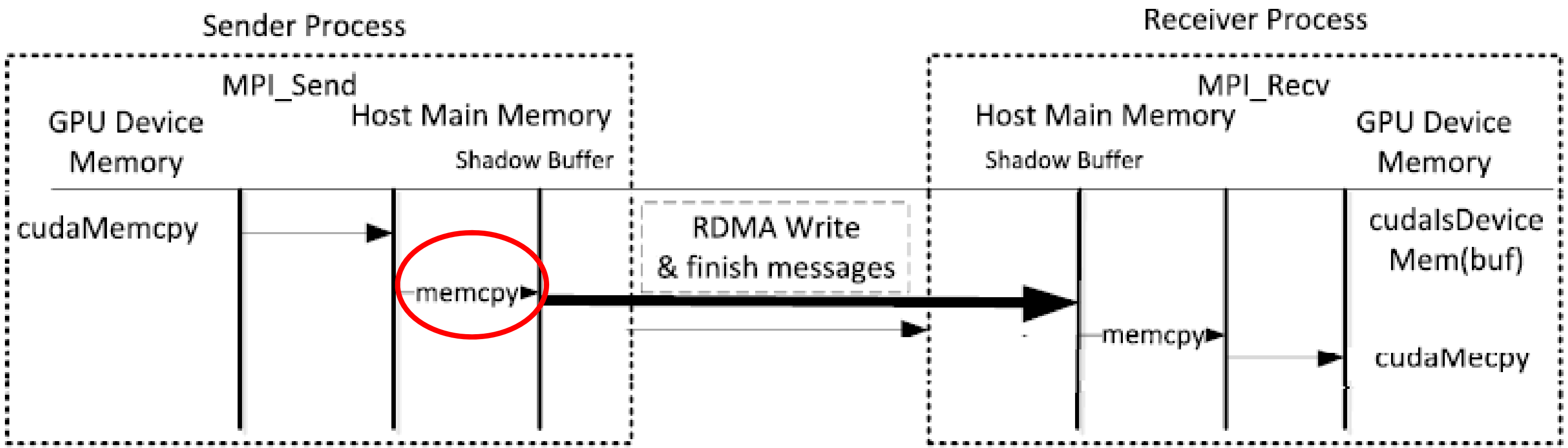
Supported on Tesla 20-series and other Fermi GPU

MPI libraries with support for NVIDIA GPUDirect

MPI transfer primitives copy data directly to/from GPU memory.

MPI library can differentiate between device memory and host memory without any hints from the user. **MPI library can find out whether the buffer was allocated in the GPU memory.**

MVAPICH2 MPI library for InfiniBand

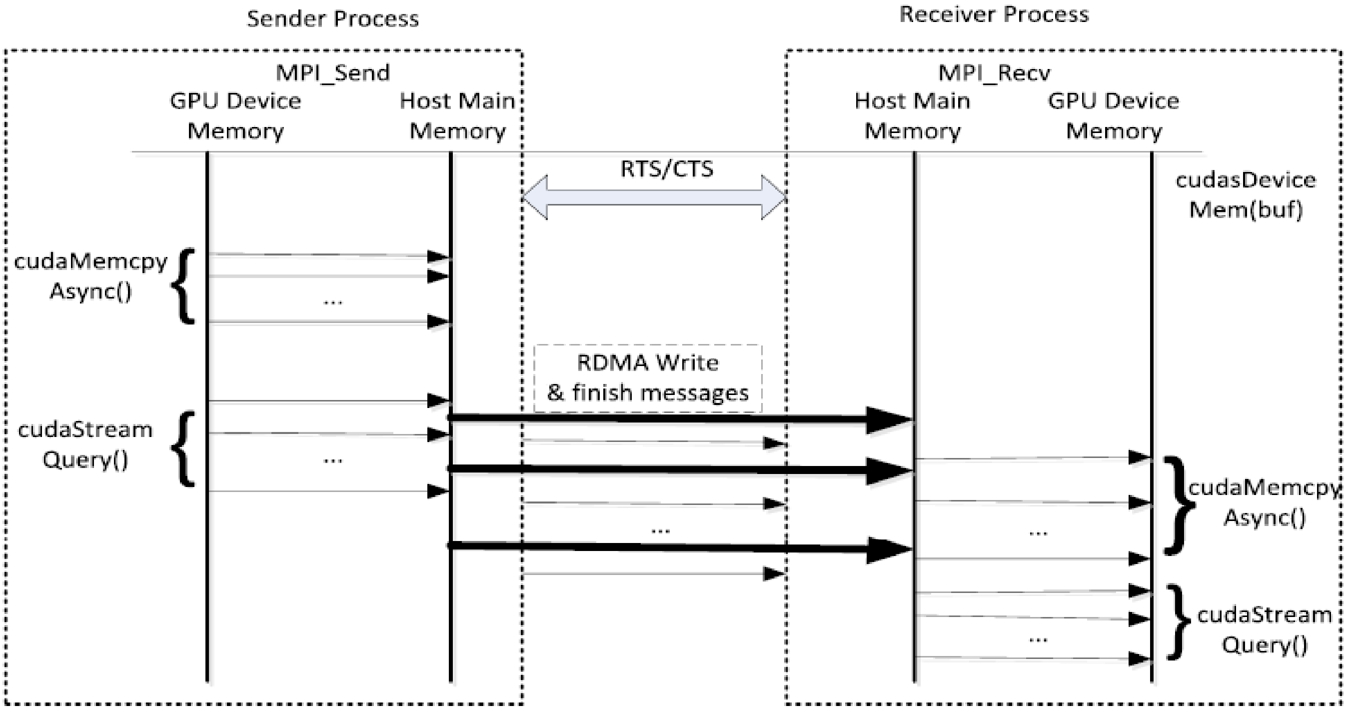


(b) MVAPICH2-GPU without GPU-Direct

GPU memory → Host memory (using cudaMemcpy())
→ Host memory for InfiniBand (using memcpy())→ InfiniBand Network.
InfiniBand requires communication memory to be registered.
Due to a limitation in the Linux kernel, it is not possible for two PCI devices to register the same page.

Using GPU Direct, both network adapter and GPU can pin down the same buffer.
Therefore, using GPU Direct, the following sequence is sufficient for network communication:
GPU memory → Host memory (using cudaMemcpy())→InfiniBand Network.
GPU Direct cuts down one step in the communication process.

MVAPICH2 MPI library for InfiniBand



(a) MVAPICH2-GPU with GPU-Direct

MVAPICH2 implements point-to-point messaging using RDMA. There are two protocols – RDMA Put and RDMA Get. In RDMA Put mechanism, the two processes perform a handshake using Request To Send (RTS) and Clear To Send (CTS) messages. When the sender receives CTS, it is able to issue RDMA write operation. Finally, the sender will send RDMA finish message to notify the receiver the RDMA write finish and the data is ready in the receive side.

MPI libraries with support for NVIDIA GPUDirect

The pipeline unit is presented as a configurable parameter of the MVAPICH2 library. Once the optimal value for the cluster is found, it can be placed in a configuration file (MVAPICH2 supports this), and end-users will transparently use this setting.

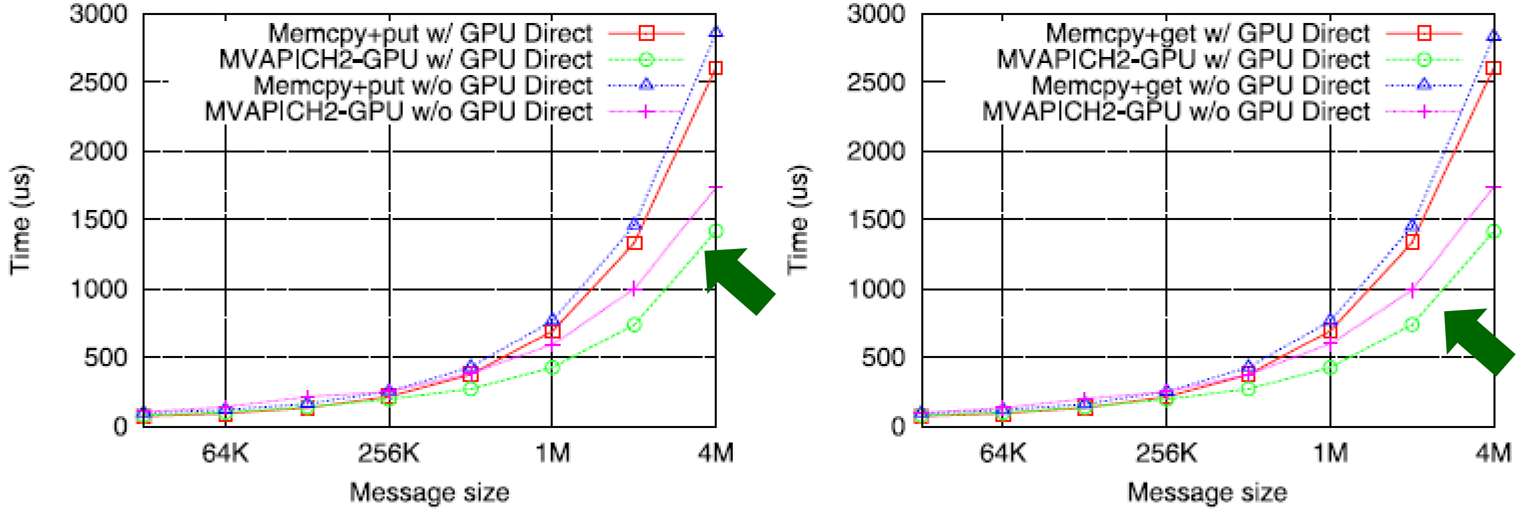
128 KB and 256 KB to be the optimal block size for the OSU cluster and TACC cluster

MVAPICH2 MPI library for InfiniBand

Fig. 6

One-sided latency

performance

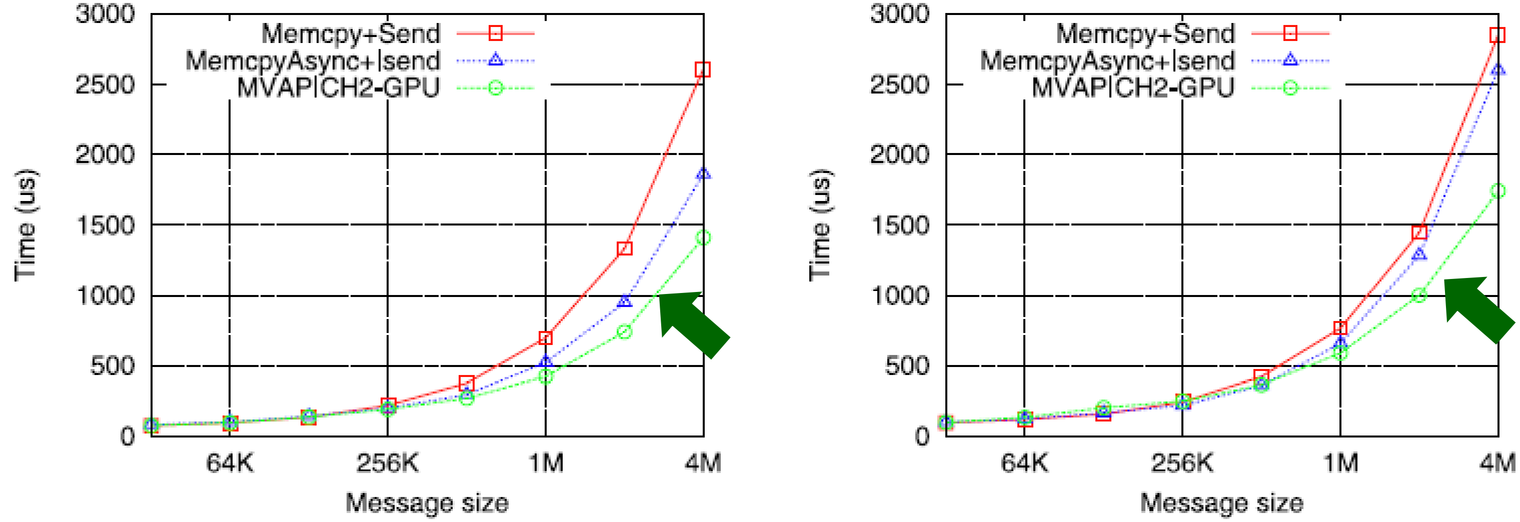


(a) MPI_Put latency (b) MPI_Get latency

Fig. 5

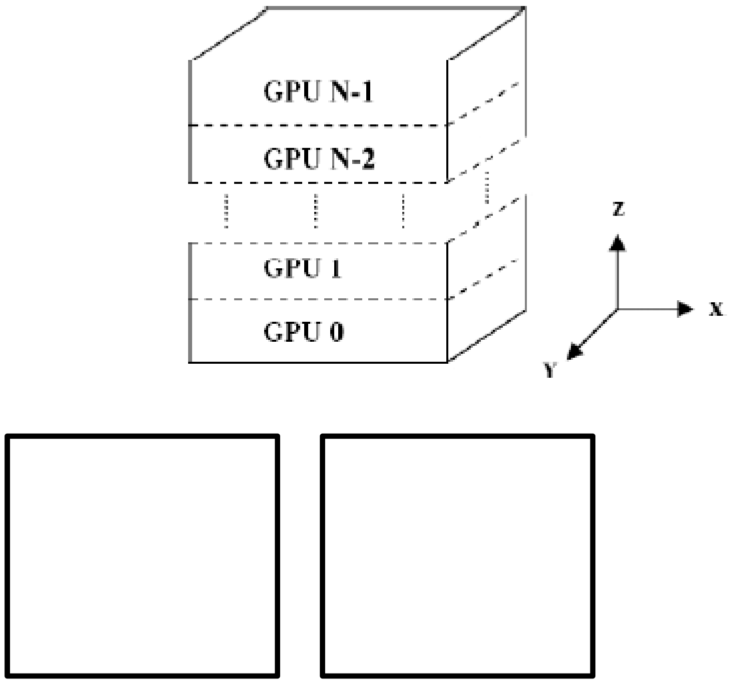
Two-sided latency

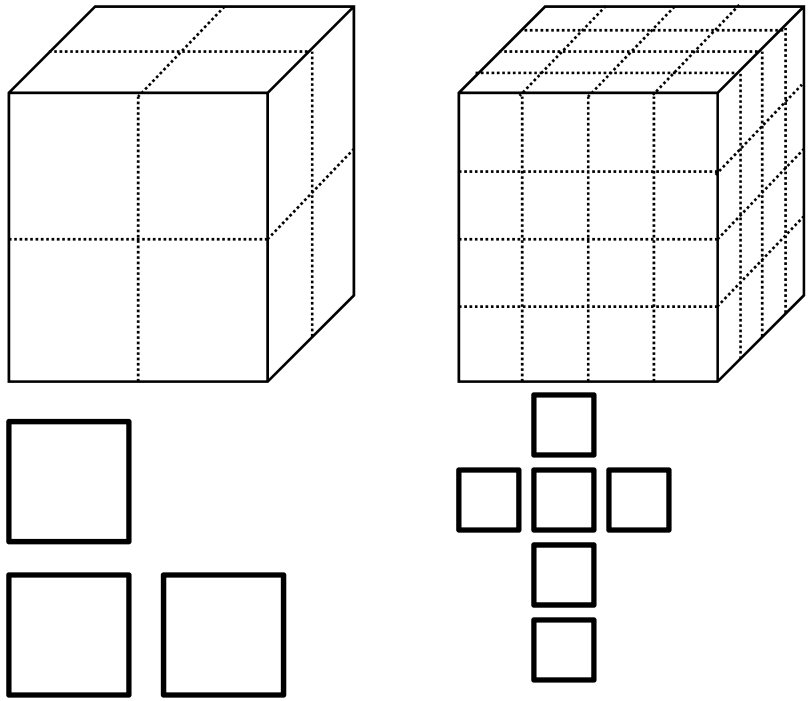
performance



(a) with GPU Direct (b) without GPU Direct

Decomposition

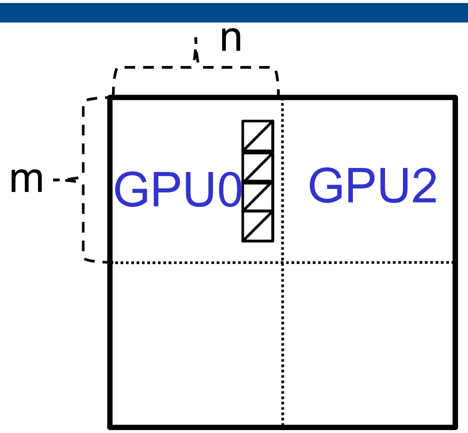




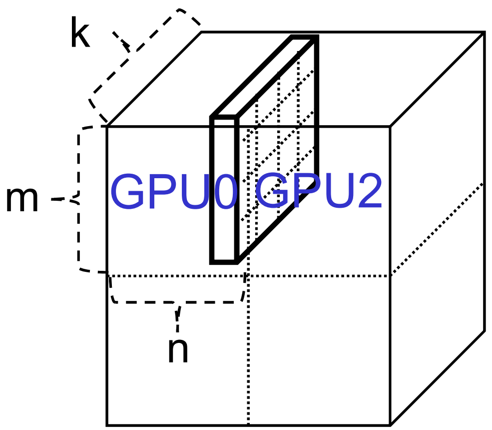
Max communication times is 6,
communication cost will be reduced

Currently, MVAPICH2 only supports contiguous datatype communication between GPUs.

Packing
Multidimensional structures data



device[n-1]
device[n+n-1]
…
…
device[(m-1)*n+n-1]



device[n-1]
device[n+n-1]
…
…
device[(m-1)*n+n-1]

device[m*n+n-1]
device[m*n+n+n-1]
…
…
device[m*n+(m-1)*n+n-1]

device[(k-1)*m*n+n-1]
device[(k-1)*m*n+n+n-1]
…
…
device[(k-1)*m*n+(m-1)*n+n-1]

This avoids multiple transfers over the network that can be prohibitively expensive

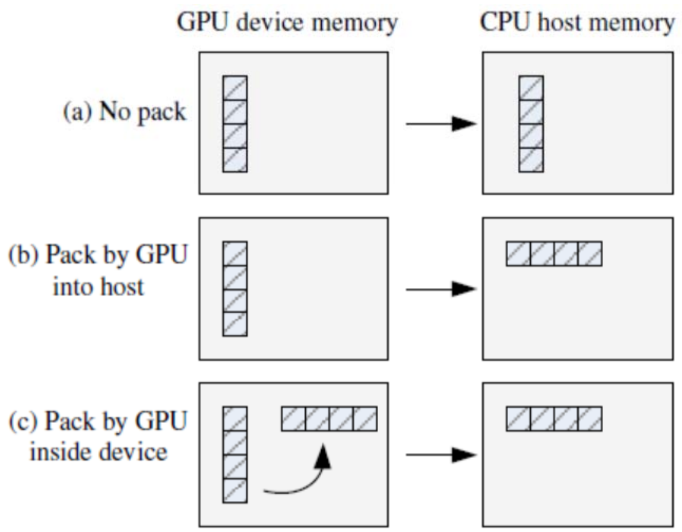
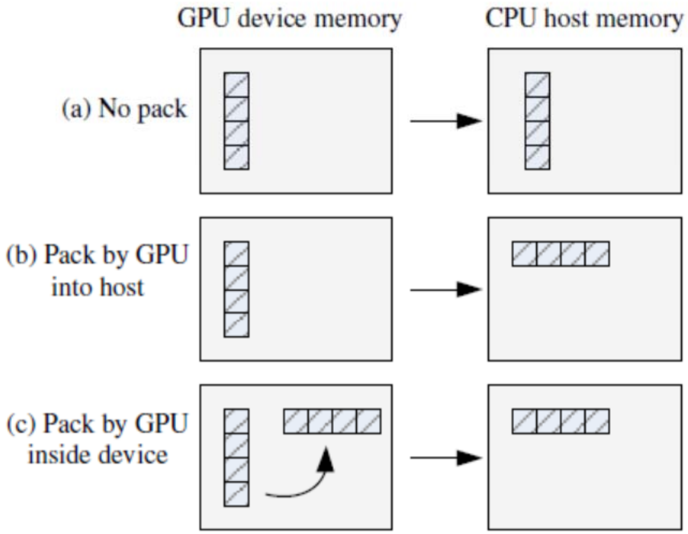


Figure 1. Data packing options for GPU based systems

Packing: evaluate GPU to CPU first

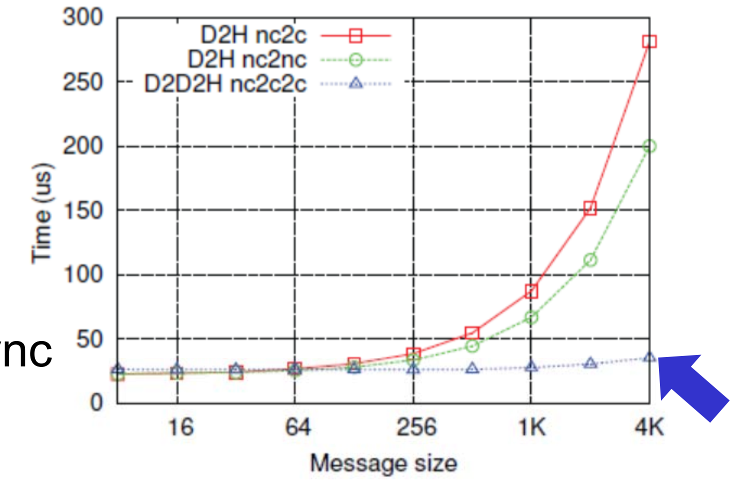


D2H nc2nc
cudaMemcpy2D

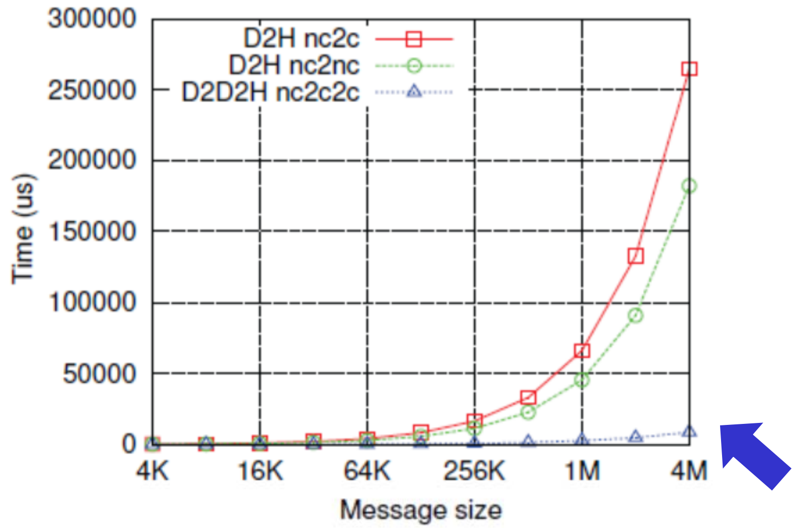
D2H nc2c
cudaMemcpy2DAsync

D2D2H nc2c2c
cudaMemcpy2DAsync
cudaMemcpy

Figure 1. Data packing options for GPU based systems



(a) Small Message



(b) Large Message

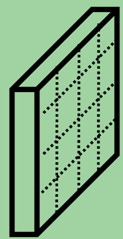
For non-contiguous data, latency of packing data in th
e GPU is always larger than the RDMA data transfer l
atency or time for contiguous data movement betwee
n device memory and main memory. So, data packin
g and unpacking latency will determine the pipeline p
erformance.

cudaMemcpy2D

```
cudaError_t cudaMemcpy2D ( void *                  dst,
                           size_t                  dpitch,
                           const void *            src,
                           size_t                  spitch,
                           size_t                  width,
                           size_t                  height,
                           enum cudaMemcpyKind     kind
                         )
```

Parameters:

- *dst* - Destination memory address
- *dpitch* - Pitch of destination memory
- *src* - Source memory address
- *spitch* - Pitch of source memory
- *width* - Width of matrix transfer (columns in bytes)
- *height* - Height of matrix transfer (rows)
- *kind* - Type of transfer



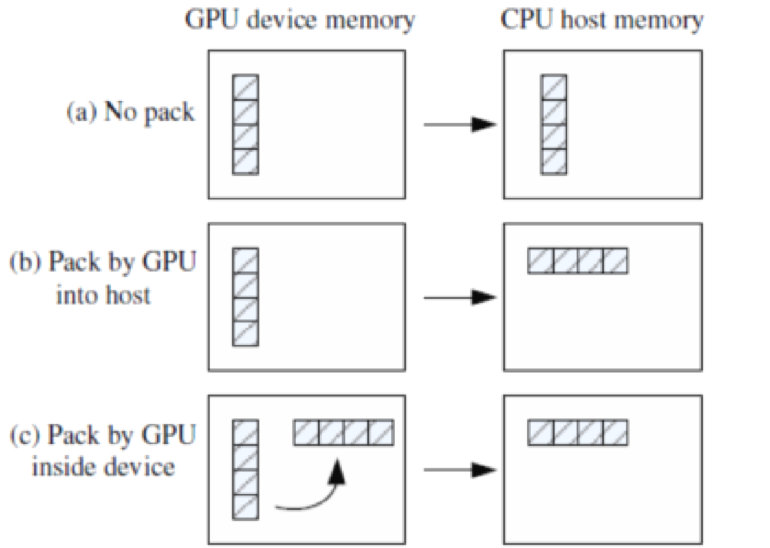


Figure 1. Data packing options for GPU based systems

Copies a matrix (`height` rows of `width` bytes each) from the memory area pointed to by `src` to the memory area pointed to by `dst`, where `kind` is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy. `dpitch` and `spitch` are the widths in memory in bytes of the 2D arrays pointed to by `dst` and `src`, including any padding added to the end of each row. The memory areas may not overlap. Calling **cudaMemcpy2D()** with `dst` and `src` pointers that do not match the direction of the copy results in an undefined behavior. **cudaMemcpy2D()** returns an error if `dpitch` or `spitch` is greater than the maximum allowed.

```
float source[4][4] =
     {1.0, 2.0, 3.0, 4.0,
      5.0, 6.0, 7.0, 8.0,
      9.0, 10.0, 11.0, 12.0,
     13.0, 14.0, 15.0, 16.0};
```

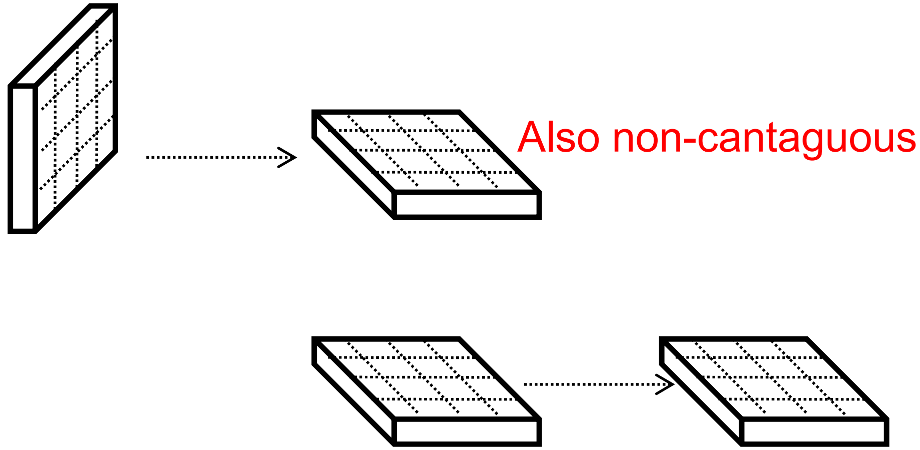
```
size_t  SIZE=1*sizeof(float);
cudaMemcpy2D(&dest[0][0], 1*SIZE,
             &source[0][0], 4*SIZE,
             1*SIZE, 4*SIZE,
             cudaMemcpyDefault);
```

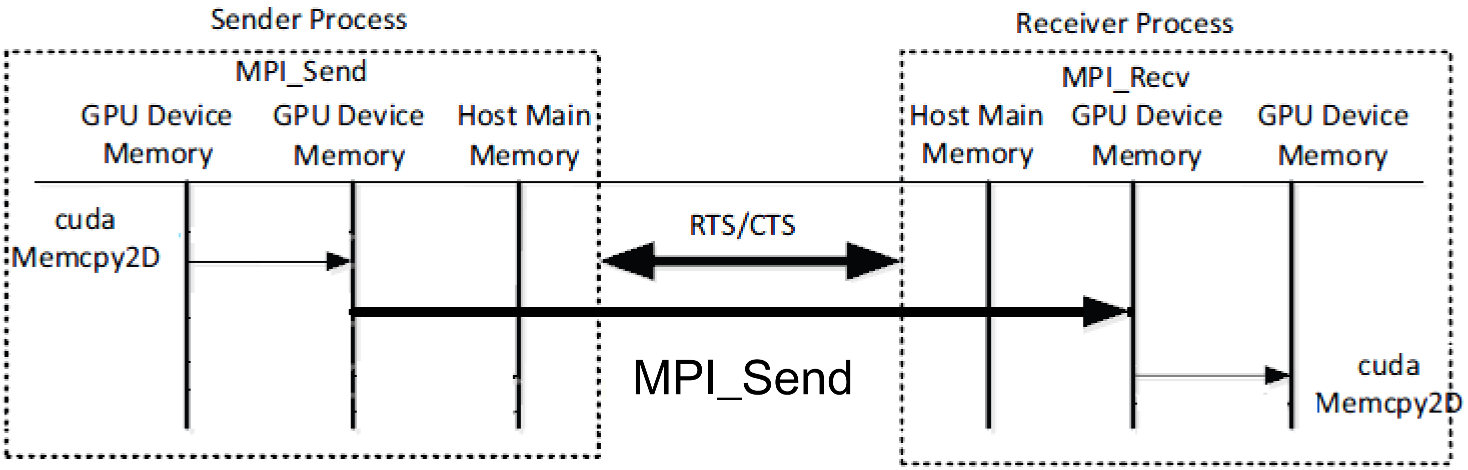
dest = {1.0, 5.0, 9.0, 13.0}

MPI and CUDA without pipelining

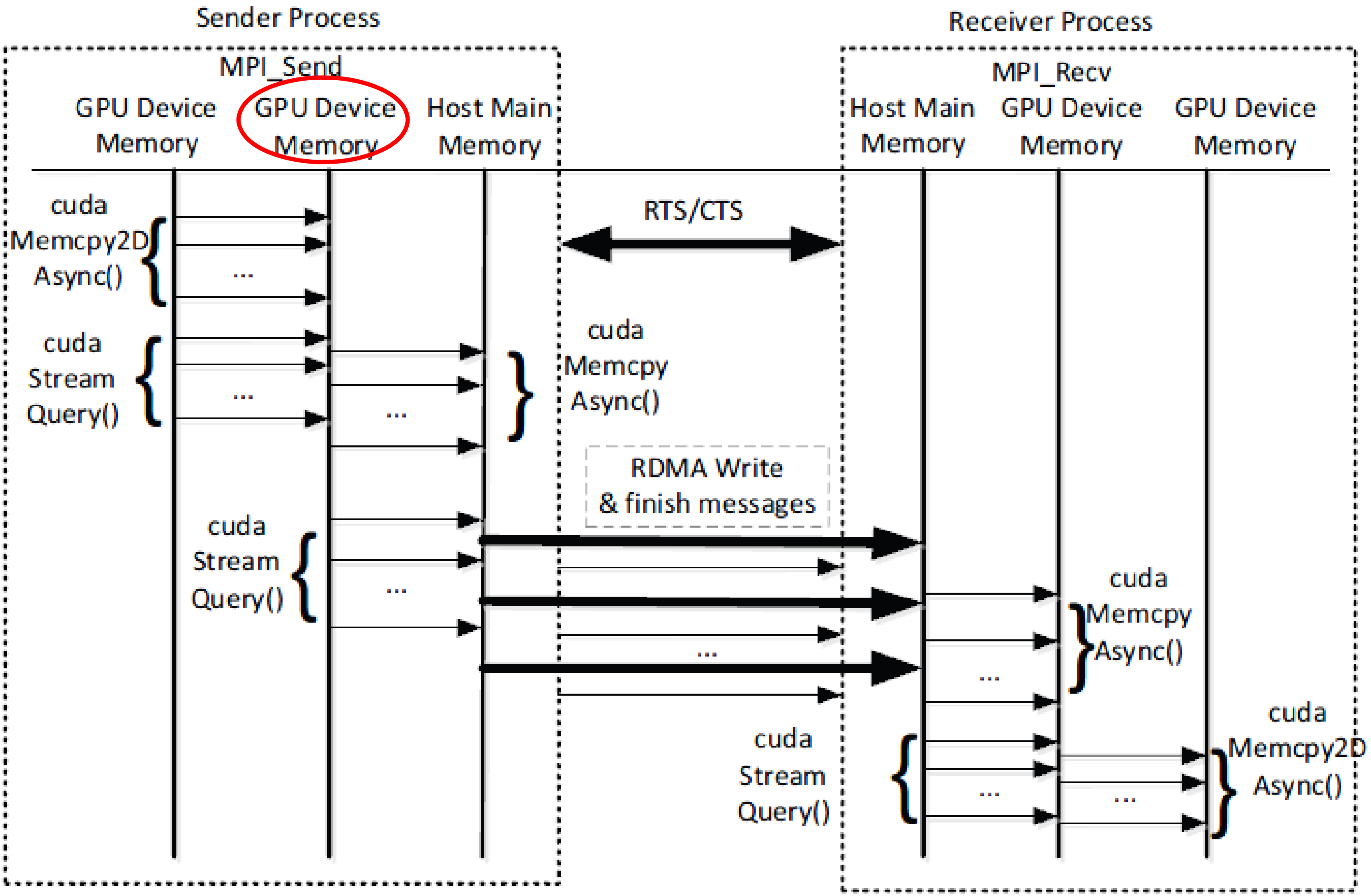
```
MPI_Type_vector();
MPI_Type_commit();
...
if (haveEastNeighbor) {
    // copy noncontiguous data from device to host
    cudaMemcpy2D(...);
    // send data with vector type to east neighbor
    MPI_Send(...);
    // receive data with vector type from east neighbor
    MPI_Recv(...);
    // copy noncontiguous data from host to device
    cudaMemcpy2D(...);
}
...
```

Prepare for the 3D case





Pipelining


Sender Process
MPI_Send
GPU Device Memory
GPU Device Memory
Host Main Memory
cuda Memcpy2D Async()
...
cuda Stream Query()
...
...
cuda Memcpy Async()
RTS/CTS
Receiver Process
MPI_Recv
Host Main Memory
GPU Device Memory
GPU Device Memory
RDMA Write & finish messages
cuda Stream Query()
...
...
cuda Memcpy Async()
...
cuda Stream Query()
...
...
cuda Memcpy2D Async()

Implementation

```
MPI_Type_vector();
MPI_Type_commit();
...
if (haveEastNeighbor) {
   for (i = 0; i < pipeline_length; i++) {
      // pack each block from non contiguous to contiguous in GPU
      cudaMemcpy2DAsync(…);
   }
   while (active_pack_stream || active_d2h_stream ) {
      if (active_pack_stream > 0) {
         if (cudaStreamQuery() == cudaSuccess) {
            // copy each block from device memory to host memory
            cudaMemcpyAsync(…);
         }
      }
      if (active_d2h_stream > 0) {
         if (cudaStreamQuery() == cudaSuccess) {
            // send each block to east neighbor from host memory
            MPI_Isend(...);
         }
      }
   }
   MPI_Waitall(...);
   for (j=0; j < pipeline_length; j++) {
      // receive each block from east neighbor to host memory
      MPI_Irecv(...);
   }
   while (active_recv > 0 || active_h2d_stream > 0) {
      if (active_recv > 0) {
         MPI_Test (...);
         // copy each block from host memory to device memory
         cudaMemcpyAsync (...);
      }
      if (active_h2d_stream > 0) {
         if (cudaStreamQuery()== cudaSuccess) {
            // unpack each block from contiguous to non contiguous in GPU
            cudaMemcpy2DAsync(...);
         }
      }
   }
}
...
```

MV2-GPU-NC

```
MPI_Type_vector();
MPI_Type_commit();
...
if (haveEastNeighbor) {
    // send data with vector type from device memory to east neighbor
    MPI_Send(...);
    // receive data with vector type to device memory from east neighbor
    MPI_Recv(...);
}
...
```

(c) MV2-GPU-NC (highest performance and productivity)

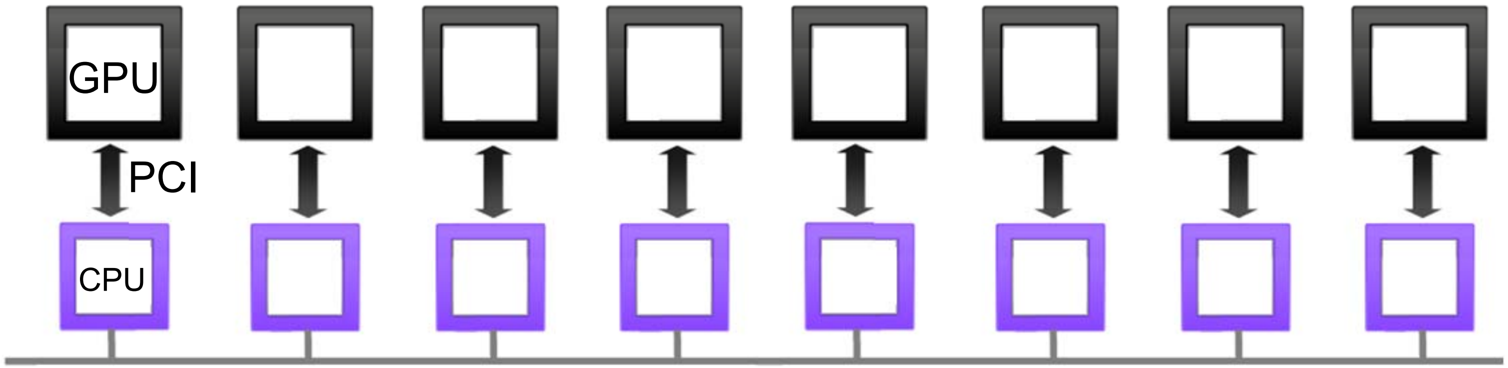
```
#define SIZE 4
float a[SIZE][SIZE] =
    {1.0, 2.0, 3.0, 4.0,
     5.0, 6.0, 7.0, 8.0,
     9.0, 10.0, 11.0, 12.0,
    13.0, 14.0, 15.0, 16.0};
MPI_Datatype columntype;
MPI_Type_vector(SIZE, 1, SIZE, MPI_FLOAT, &columntype);
MPI_Type_commit(&columntype);
MPI_Send(&a[0][0], 1, columntype,dest, tag, MPI_COMM_WORLD);
MPI_Recv(b, SIZE, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &stat);
MPI_Type_free(&columntype);
```

The design proposed in this paper has been integrated into MVAPICH2. MVAPICH2 natively supports direct GPU to GPU communication using NVIDIA CUDA 4.0.

Dest b= 1.0 5.0 9.0 13.0

Experiment

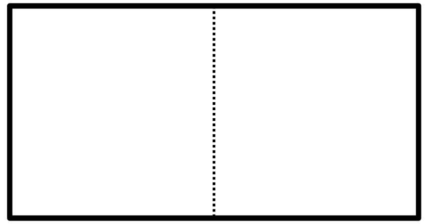
We used a cluster with eight nodes in our experimental evaluation. Each node is equipp ed with dual Intel Xeon Quad-core Westmere CPUs operating at 2.53 GHz, 12GB host memory, and Nvidia Tesla C2050 GPUs with 3GB DRAM. The InfiniBand HCAs used on this cluster are Mellanox QDR MT26428. Each node has Red Hat Linux 5.4, OFED 1.5.1, MVAPICH2-1.6RC2, and CUDA Toolkit 4.0. The MPI level evaluation is based on OSU Micro Benchmarks [20]. We modified Stencil2D application in SHOC 1.0.1 with MV2-GP U-NC to send and receive both contiguous and non-contiguous data in GPU device me mory. We run one process per node and use one GPU per process for all experiments.



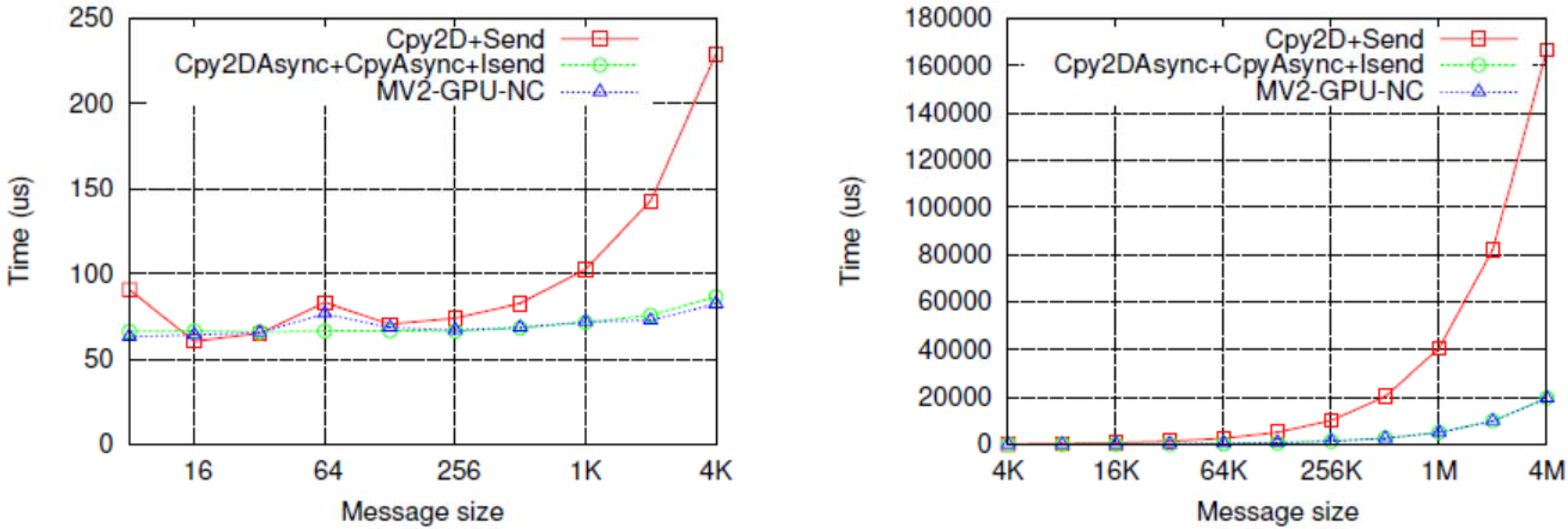
InfiniBand

Experiment

Communication size
Bigger than 64KB
Will get better performance



A. Performance Evaluation for Vector Data
   1x2 process grid for varying non-contiguous message sizes and a constant chunk size of 4 bytes (float).



(a) Small Message

(b) Large Message

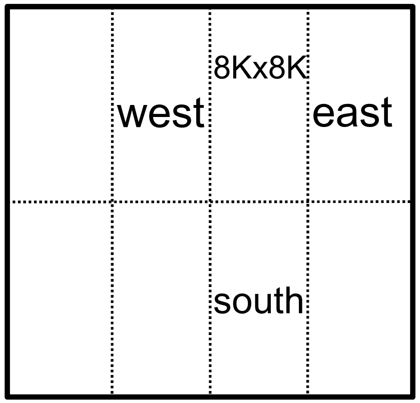
Figure 5. Vector Communication Latency

Experiment

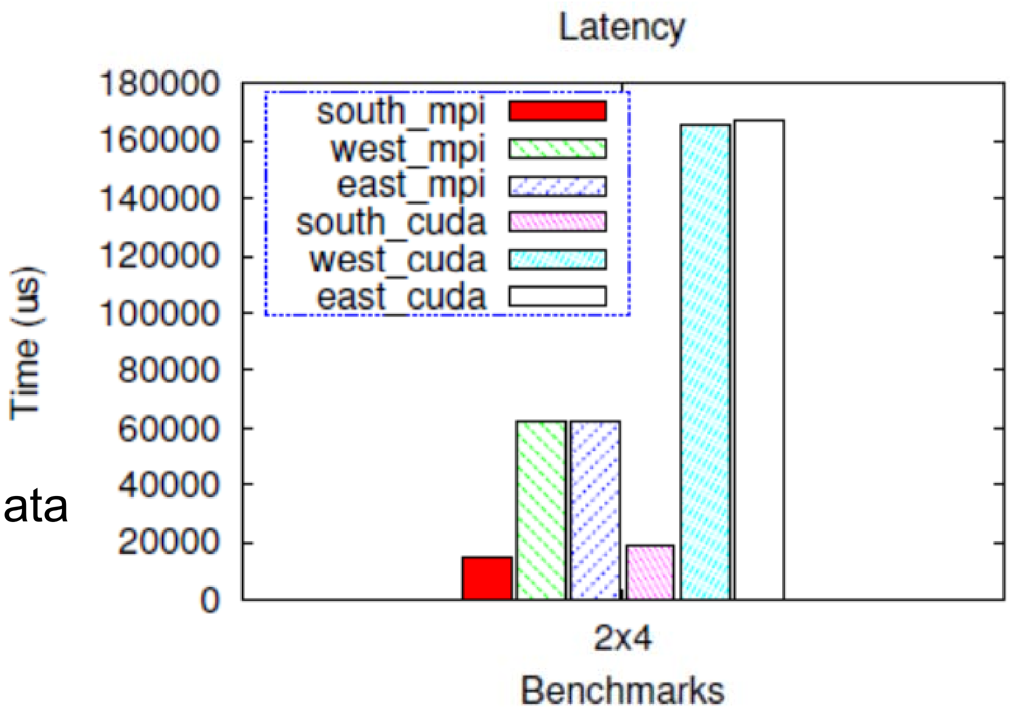
B. Performance evaluation for Stencil2D

2x4 process grid with a 8Kx8K single precision data set per process.

Not data size, it's amount



South is contiguous data
East and west are non-contiguous data



South mpi, west mpi and east mpi represent the time spent in mpi
South cuda,west cuda and east cuda represent the time spent in moving data betwee n device and main memory.

Experiment

B. Performance evaluation for Stencil2D

	Stencil2D-Def	Stencil2D-MV2-GPU-NC
Function calls	MPI_Irecv: 4 MPI_Send 4 MPI_Waitall: 2 cudaMemcpy: 4 cudaMemcpy2D: 4	MPI_Irecv: 4 MPI_Send: 4 MPI_Waitall: 2 cudaMemcpy: 0 cudaMemcpy2D: 0
Lines of Code	245	158

Table I
COMPARING COMPLEXITY OF EXISTING STENCIL2D CODE WITH MODIFIED CODE USING MV2-GPU-NC

Reduce : call function, check status of sending and receive, synchronize,

allocate space, parameter

Experiment

B. Performance evaluation for Stencil2D

Process Grid (Matrix Size/Process)	Stencil2D-Def	Stencil2D-MV2-GPU-NC	Improvement
1x8 (64k x 1k)	0.547788	0.314085	42%
8x1 (1k x 64k)	0.33474	0.272082	19%
2x4 (8k x 8k)	0.36016	0.261888	27%
4x2 (8k x 8k)	0.33183	0.258249	22%

Table II
COMPARING MEDIAN EXECUTION TIMES OF STENCIL2D - SINGLE PRECISION (SEC)

Process Grid (Matrix Size/Process)	Stencil2D-Def	Stencil2D-MV2-GPU-NC	Improvement
1x8 (64k x 1k)	0.780297	0.474613	39%
8x1 (1k x 64k)	0.563038	0.438698	22%
2x4 (8k x 8k)	0.57544	0.424826	26%
4x2 (8k x 8k)	0.546968	0.431908	21%

Table III
COMPARING MEDIAN EXECUTION TIMES OF STENCIL2D - DOUBLE PRECISION (SEC)

Thanks