Please adjust group name in tumuser.sty.
Department of Electrical and Computer Engineering
Technical University of Munich

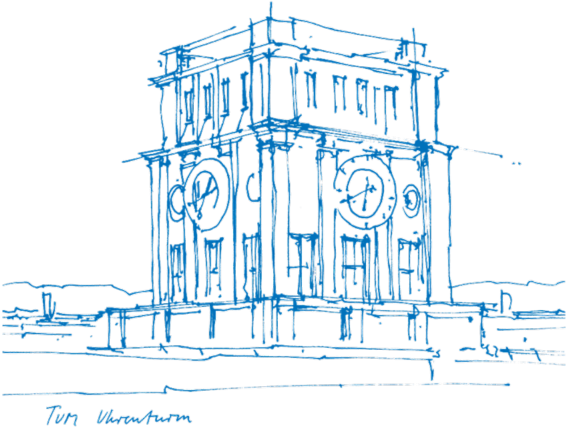
HPC-lab

Final Project
CUDA-Aware MPI with UVA

Phuong Nguyen, Zirong Cai

Please adjust group name in tumuser.sty.
Department of Electrical and Computer Engineering
Technical University of Munich

November 8th, 2021



Outline

SWE with CUDA - Current Implementation

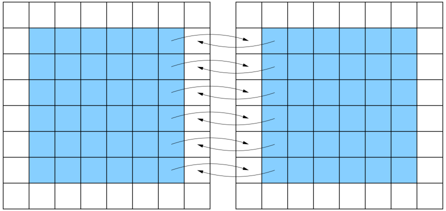
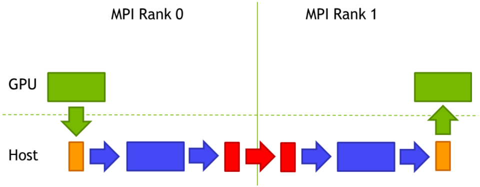


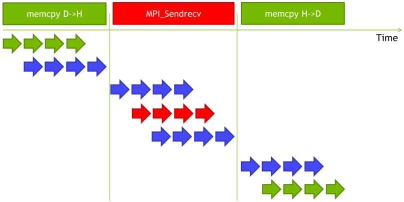
Figure 1 Caption

```
While ()
{
exchangeGhostLayers();           // On Host
setGhostLayer();
memcpy H-> D;
computeNumericalFluxes();                 // On Device
MPI_Allreduce(&l_maxTimeStep, \
        &l_maxTimeGlobal, ...);
updateUnknowns(l_maxTimeGlobal);          // On Device
memcpy D-> H;
iteration++;
}
```

- Allocate data on both Host and Device memory.
- Need to copy data between Host - Device explicitly.
- Additional data structures are required for exchanging ghost layer with top & bottom neighbors (*topLayer*, *bottomLayer*).

SWE CUDA with MPI





CUDA Aware MPI

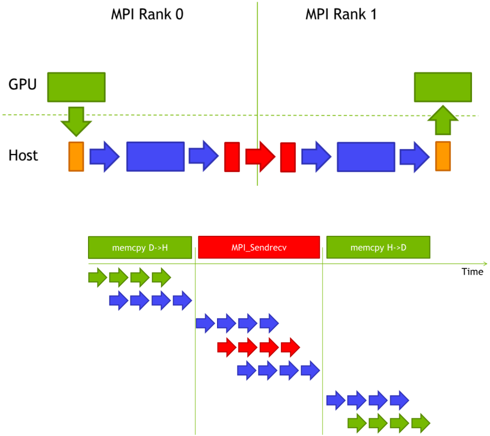


Figure 2 CUDA MPI

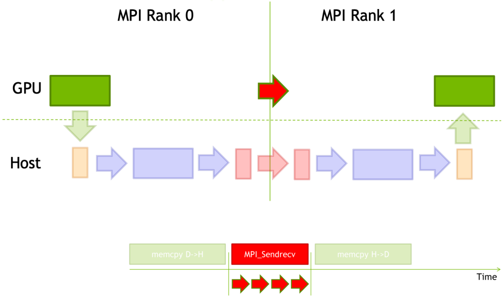


Figure 3 CUDA-Aware MPI

Unified Memory

- Unified Memory is a single memory address space accessible from any processor in a system.
- CUDA system software and/or the hardware takes care of migrating memory

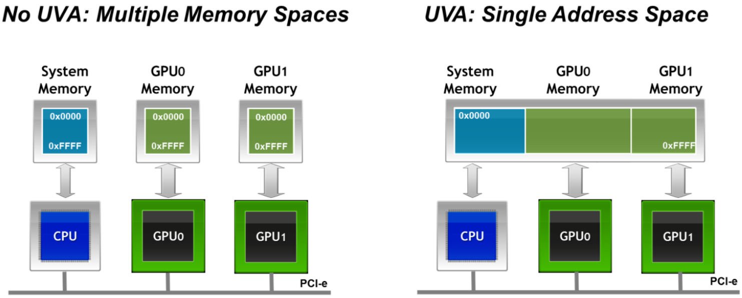


Figure 4 No Unified Memory Vs. Unified Memory

GPUDiect

NVIDIA GPUDirect technologies provide high-bandwidth, low-latency communications with NVIDIA GPUs.

- GPUDirect P2P
- GPUDirect RDMA

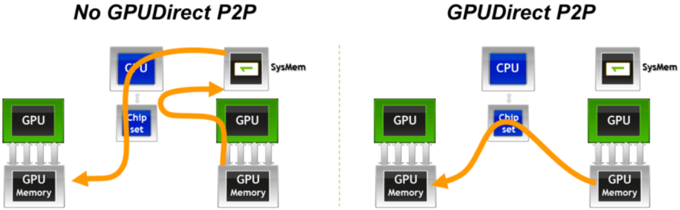


Figure 5

Outline

SWE with CUDA-Aware MPI

Implementing Unified Virtual Address (UVA):

- *h*, *hv*, *hu* are allocated on UVA.
- Other helper variables for kernel functions: still on device memory.
- No explicit Copy data between Host - Device.

MPI implementation:

- Taking UVA pointers.
- Pack data directly from the grid data (which is on UVA).
- For top/bottom layer exchange: strided data access is possible, using
  MPI_Type_vector(l_nXLocal + 2, 1, l_nYLocal + 2, MPI_FLOAT, &l_mpiRow);

Error Test

Error test implementation:

- *-DENABLE_TEST*: prints *h* to an output file.
- *test/error_check.py*: calculates the absolute difference between 2 input data files.
- *test/run_test.sh*: executable script (compilation, runs, error check.

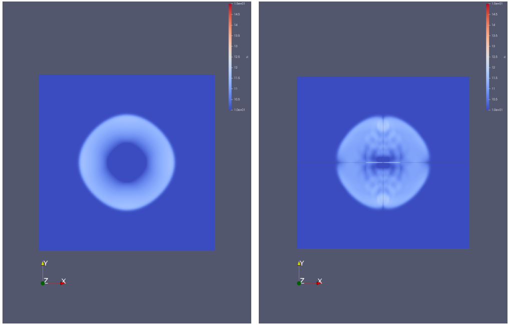
```
// 2D subarray Datatype for MPI_File_set_view
MPI_Type_create_subarray(ndims,  gsizes,\
        lsizes, starts, MPI_ORDER_C,\
        MPI_FLOAT, &subarr_type);
...
// Datatype for printing data
MPI_Type_vector(l_nXLocal, l_nYLocal, \
        l_nYLocal + 2, MPI_FLOAT, &print_type);
...
MPI_File_open(MPI_COMM_WORLD, filename.c_str(),\
MPI_MODE_WRONLY + MPI_MODE_CREATE,\
MPI_INFO_NULL, &file);

MPI_File_set_view(file, 0, MPI_FLOAT, subarr_type,\
        "native", MPI_INFO_NULL);
MPI_File_write_all(file, &h_test[1][1], 1,\
        print_type, &status);

MPI_File_close(&file);
```

Listing 1 Parallel IO for printing *h*

SWE-CUDA Issue (1)



(a) 1 MPI **(b)** 4 MPI-CUDA

Figure 6 Visualisation of *WaterHeight*

	1 cuda	4 mpi	4 cuda
1 mpi	0.0275	0.0373	29929
1 cuda	0.0000	0.0648	29929

Table 1 Abs difference of *h*

SWE-CUDA Correction (2)

- *USE_MPI* was not defined.
  $\rightarrow$ No calls *synchCopyLayerBeforeRead*.
  $\rightarrow$ No *memcpy D->H*

- No calls *synchGhostLayerAfterWrite*
  $\rightarrow$ No *memcpy H->D*

	1 cuda	4 mpi	4 cuda
1 mpi	0.02754	1.28397	0.02754
1 cuda	0.00000	1.31151	0.00000

Table 2 Abs difference of *h* after fixing bugs

Outline

Validation

Runs job on multiple GPUs with the *run_gpu.sh* script

```bash
#!/bin/bash
export CUDA_VISIBLE_DEVICES=\
        $OMPI_COMM_WORLD_LOCAL_RANK
case $OMPI_COMM_WORLD_LOCAL_RANK in
[0]) cpus=0;;
[1]) cpus=1;;
[2]) cpus=2;;
[3]) cpus=3;;
esac
numactl --physcpubind=$cpus $@
```

```
$ mpirun -np 4 ../gpu_bind.sh ./swe-cuda
                -x ${NX} -y ${NY} -c ${STEPS} -o .
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63.01    Driver Version: 470.63.01    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:01:00.0 Off |                  N/A |
| 30%   35C    P2   246W / 320W |    901MiB / 10018MiB |     89%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce ...  On   | 00000000:41:00.0 Off |                  N/A |
| 30%   33C    P2   234W / 320W |    500MiB / 10018MiB |     88%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA GeForce ...  On   | 00000000:81:00.0 Off |                  N/A |
| 30%   37C    P2   230W / 320W |    538MiB / 10018MiB |     88%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA GeForce ...  On   | 00000000:C1:00.0 Off |                  N/A |
| 30%   36C    P2   233W / 320W |    526MiB / 10018MiB |     88%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    307874      C   ./build/ns                        313MiB |
|    0   N/A  N/A   2268977      C   ./build/swe-mpi                   467MiB |
|    1   N/A  N/A   2268982      C   ./build/swe-mpi                   467MiB |
|    2   N/A  N/A   2268978      C   ./build/swe-mpi                   467MiB |
|    3   N/A  N/A   2268981      C   ./build/swe-mpi                   467MiB |
+-----------------------------------------------------------------------------+
```

Figure 7 GPU Monitoring

Validation

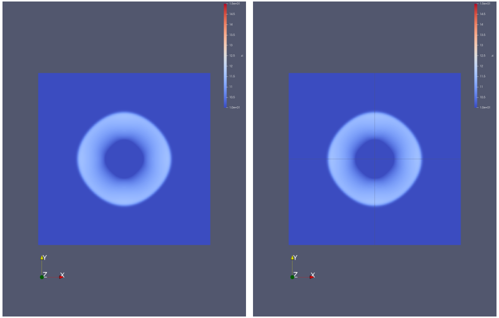


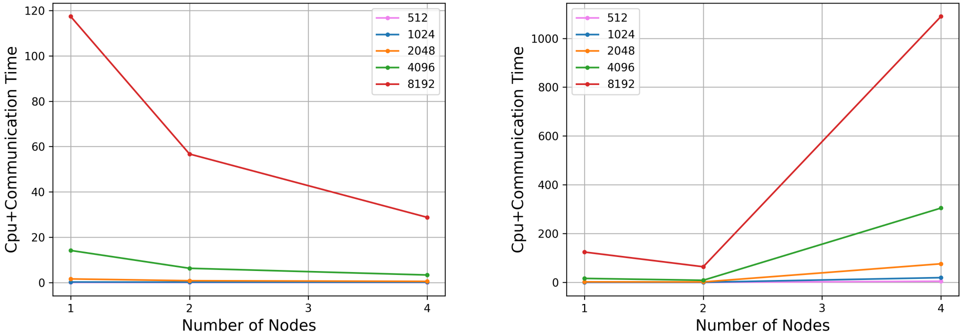
Figure 8 Pure-MPI vs CUDA-Aware-MPI

CUDA-Aware MPI implementation produced correct results.

	1 cuda-aware	4 cuda-aware
1 mpi	0.0275	0.0275
1 cuda	0.0000	0.0000
4 cuda	0.0000	0.0000

Table 3 Error table

Initial Performance



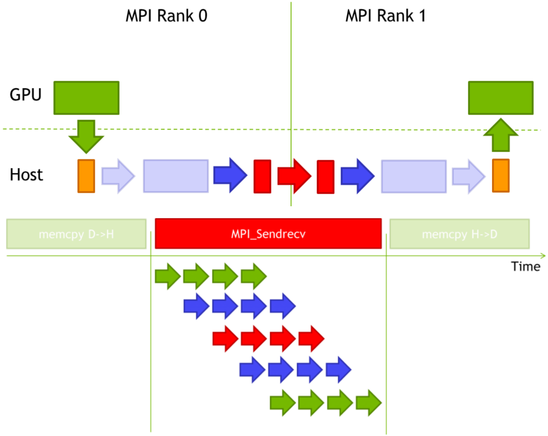
(a) normal CUDA+MPI

(b) CUDA-Aware-MPI

Figure 9 Runtime comparison of the two implementations

Hardware specs

NVIDIA GeForce RTX™ 3080 ?
→ No NVLINK
→ No GPUDirect support



Profiling Output

```
Time(%)  Time (ms)     Name
-------  ---------   ----------------------
   76.4        292   cudaMallocManaged
   13.9         53   cudaDeviceSynchronize
    4.8         18   cudaLaunchKernel
    1.7          6   cudaMemcpyAsync

Time(%)   Time (ms)     Operation
-------   ---------   ----------------------
   47.1        2   [CUDA Unified Memory memcpy HtoD]
   39.4        2   [CUDA Unified Memory memcpy DtoH]
   13.5        1   [CUDA memcpy DtoH]


Time(%)   Time (ms)         Range
-------   ---------   ----------------------
   44.8        171   MPI:MPI_Init
   28.8        110   MPI:MPI_Finalize
   15.4         58   MPI:MPI_Sendrecv
   11.0         42   MPI:MPI_Allreduce
```

Listing 2 2 MPI

```
Time(%)   Time (ns)     Name
-------   ---------   ----------------------
   62.7       3342   cuMemcpyAsync
   26.2        139   cuStreamSynchronize
    9.4        499   cudaMallocManaged
    1.1         57   cudaDeviceSynchronize

Time(%)   Time (ns)     Operation
-------   ---------   ----------------------
   52.6     527   [CUDA memcpy HtoD]
   46.6     467   [CUDA memcpy DtoH]
    0.4       4   [CUDA Unified Memory memcpy HtoD]
    0.3       3   [CUDA Unified Memory memcpy DtoH]

Time(%)   Time (ns)         Range
-------   ---------   ----------------------
   92.4       7147   MPI:MPI_Sendrecv
    3.7        288   MPI:MPI_Init
    2.4        187   MPI:MPI_Allreduce
    1.5        115   MPI:MPI_Finalize
```

Listing 3 4 MPI

Outline

Data Packing

```
l_waveBlock->setGhostLayer();
while (iteration) // time integration loop
{
exchangeGhostLayers();

synchGhostLayerAfterWrite(); //data unpacking;
l_waveBlock->computeNumericalFluxes();

MPI_Allreduce(&maxTimeStep, &maxTimeGlobal,...);

l_waveBlock->updateUnknowns(l_maxTimeGlobal);
synchCopyLayerBeforeRead() // data packing

iteration++;
}
```

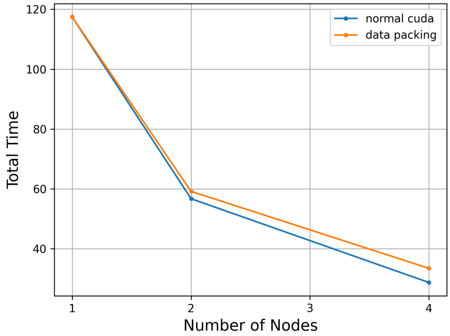


Figure 11 Performance improvement with Packing

Data Migration

- *cudaMemAdvise*
    - *cudaMemAdviseSetPreferredLocation*
    - *cudaMemAdviseSetAccessedBy*
- *cudaMemPrefetchAsync*

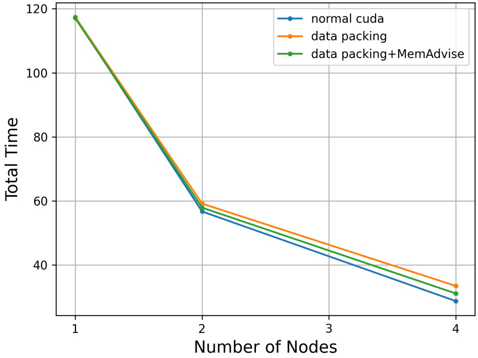


Figure 12 Performance improvement with MemAdvise

Non Blocking Communication

- *MPI_Isend* and *MPI_Irecv*
- Only communication overlap, no overlap with computations

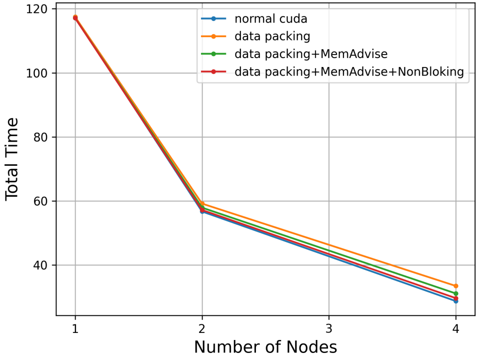


Figure 13 Performance improvement with Non Blocking

Outline

Future Work

- Testing the implementation with different Runtime optimization: GPU Direct RMDA, GPU Direct P2P, MPS, etc
- Overlapping Computations with Communications.

Conclusion

- Successfully implemented CUDA-Aware MPI for SWE
- Fixed some bugs of the original code
- Achieved almost the same performance as the normal cuda version using CUDA-Aware-MPI without GPUDiect support

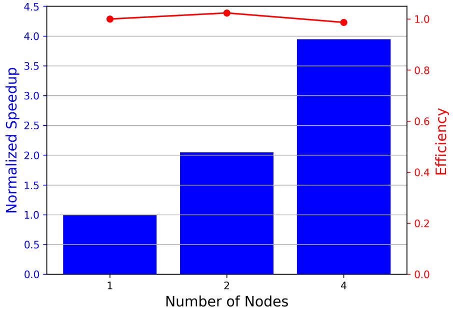


Figure 14 CUDA-Aware MPI - Final performance