MPI Example: Ghost Exchange

Presenter: Bob Robey
Oct 21-23, 2025
AMD @ Tsukuba University

together we advance_

What is covered in the MPI Example: Ghost Exchange

- What is the Ghost Exchange example and how it can help scientific application developers familiarize with porting and running on GPUs
- How to compile and run the example
- Different implementations of the Ghost Exchange examples using OpenMP® or HIP
- Several programming improvements to better the overall performance of the example
- Examples of how to use affinity to further improve the performance of the Ghost Exchange example

MPI Example: Ghost Exchange – what is it?

- A simplified instance of what an actual scientific application code using MPI might look like
- The problem is discretized on a structured Cartesian grid where the solution is defined on a cell-wise fashion
- MPI is used to execute the computations in parallel: multiple processes handle partitions of the initial computational domain
- *Ghost cells* are cells assigned to a given process that are instead own by a different one: this means that a process can read the solution defined at the ghost cells but should not modify it
- The ghost cells exist because mathematical operators in discretized form need information on the neighboring cells to compute the values of physical fields on a given cell
- Ghost cells surround the cells owned by a process therefore forming a *halo* around the subdomain owned by the process. Note that boundary cells are also included in the halo in this example
- Values of the solution at the ghost cells need to be transferred from the process that owns the ghost cells to the process that is only reading their information: this is done through a *halo exchange* using MPI
- Boundary conditions are of *outflow* type, meaning that the value of the solution at the boundary cells is set equal to the neighboring interior cell.
- Boundary conditions are enforced prior to the halo exchanges

Ghost Cell Exchange in two-phase scheme

Example of the 2-step halo exchange, considering 9 processes, each owning a 4x4 subset of the mesh

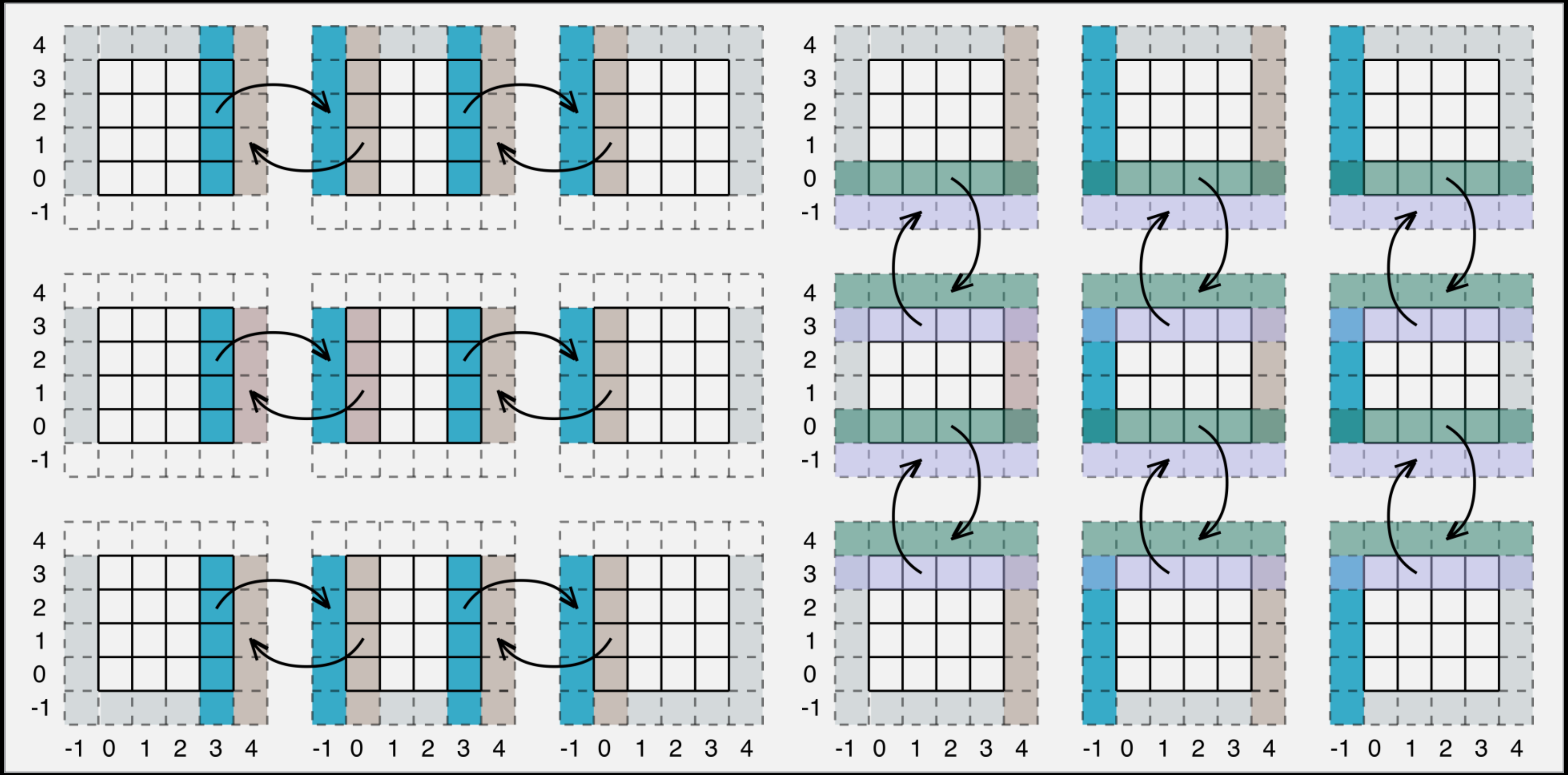

image from: *Parallel and High Performance Computing* by Robey and Zamora

Ghost Exchange: OpenMP® based versions

- OpenMP based implementations can be found at: https://github.com/amd/HPCTrainingExamples/tree/main/MPI-examples/GhostExchange/GhostExchange_ArrayAssign
- Multiple versions available:
  - Orig: CPU only implementation
  - Ver1: offload to GPU using OpenMP and unified shared memory, needs `` `export HSA_XNACK=1` `` (variation of Orig)
  - Ver2: added `roctx` markers for profiling (variation of Ver1)
  - Ver3: the communication buffers are allocated on the GPU using the OpenMP API (variation of Ver2)
  - Ver4: the communication buffers are dynamically allocated on the CPU with malloc only once at the beginning of the run instead of every time step the ghost exchange call is made (variation of Ver2)
  - Ver5: the solution arrays is unrolled from a 2D array into a 1D array (variation of Ver4)
  - Ver6: uses explicit memory management with OpenMP, can do `` `unset HSA_XNACK` `` (variation of Ver5)
  - Orig8: the computation of the averaging kernel is done in an asynchronous way, all on the CPU. The cells that do not need information from the cells in the ghost halos are advanced first. Then, the MPI communication is performed, which updates the value of the cells at the ghost cells, and then the solution is advanced on those cells that need information from cells on the ghost halos (variation of Orig)

AMD together we advance_

Ghost Exchange: HIP based versions

- HIP based implementations can be found at: HPCTrainingExamples/MPI-examples/GhostExchange/GhostExchange_ArrayAssign_HIP at main · amd/HPCTrainingExamples
- Multiple versions available:
  - Ver1: offload to GPU using HIP and unified shared memory, needs `` `export HSA_XNACK=1` `` (variation of Orig from the OpenMP® dir)
  - Ver1Cuda: same as Ver1 but in CUDA instead of HIP. This version is present to test hipify tools
  - Ver1WithBug: same as Ver1 but with a bug introduced in the thread grid launch parameters to have users debug it on their own
  - Ver2: added `roctx` markers for profiling (variation of Ver1)
  - Ver3: the communication buffers are allocated on the GPU using `hipMalloc`: GPU aware MPI is leveraged (variation of Ver2)
  - Ver4: the communication buffers are dynamically allocated on the CPU with malloc only once at the beginning of the run instead of every time step the ghost exchange call is made (variation of Ver2)
  - Ver5: the solution arrays is unrolled from a 2D array into a 1D array (variation of Ver4)
  - Ver6: the solution arrays and communication buffers are allocated on the GPU, can do `` `unset HSA_XNACK` `` (variation of Ver5)
  - Ver8: the computation advancing the solution happens on the GPU and overlaps with the MPI exchanges happening on the CPU. This feature is particularly valuable for the MI300A architecture since no copy and transfer of data has to be performed (variation of Orig8 from the OpenMP dir)

Run the Ghost Exchange CPU only version

- `module load rocm amdclang openmpi`
- `git clone https://github.com/AMD/HPCTrainingExamples`
- `cd HPCTrainingExamples/MPI-examples/GhostExchange/GhostExchange_ArrayAssign`
- `cd Orig`
- `mkdir build && cd build`
- `cmake ..`
- `make -j`
- `mpirun -n 4 ./GhostExchange -x 2  -y 2  -i 20000 -j 20000 -h 1 -c -I 100`
  - This will run 4 MPI ranks in a 2x2 processor grid (`-x 2 -y 2`)
  - The ghost cell halo will be 1 cells (-h 1) and the corners (`-c`)
  - Each process will have a 20,000 by 20,000 cell domain (`-i 20000 -j 20000`)
  - Problem will run for 100 timesteps (`-I 100`)

Ghost Exchange: CPU vs GPU Timings Comparison

Orig (CPU)

```
Solution Advancement: 24.927985
Boundary Condition Enforcement:  0.078748
Ghost Cell Update:  0.479542
Total: 25.786622
```

```
for (int j = 0; j < jsize; j++){
   for (int i = 0; i < isize; i++){
      xnew[j][i] = ( x[j][i] + x[j][i-1] + x[j][i+1] + x[j-1][i] + x[j+1][i] )/5.0;
   }
}
```

Ver1 HIP

```
Solution Advancement: 8.047540
Boundary Condition Enforcement:  0.571013
Ghost Cell Update:  1.992076
Total: 11.819252
```

Ver1 OpenMP® (GPU target offload)

```
Solution Advancement: 3.122815
Boundary Condition Enforcement:  0.585140
Ghost Cell Update:  0.462915
Total: 5.616494
```

← export HSA_XNACK=1 →

```
#pragma omp target teams distribute parallel for collapse(2)
for (int j = 0; j < jsize; j++){
   for (int i = 0; i < isize; i++){
      xnew[j][i] = ( x[j][i] + x[j][i-1] + x[j][i+1] + x[j-1][i] + x[j+1][i] )/5.0;
   }
}
```

```
__global__ void blur (double **x, double **xnew, int jsize, int isize)
{
    int tidx = threadIdx.x + blockIdx.x * blockDim.x;
    int tidy = threadIdx.y + blockIdx.y * blockDim.y;
    if (tidy < jsize && tidx < isize) {
        xnew[tidy][tidx] = ( x[tidy][tidx] + x[tidy][tidx-1] + x[tidy][tidx+1]
                           + x[tidy-1][tidx] + x[tidy+1][tidx] )/5.0;
    }
}
```

Run on MI210 with ROCm 6.4.2 and OpenMPI 5.0.7

Ghost Exchange: Ver1 vs Ver4 Timings Comparison

`export HSA_XNACK=1`

Ver1 OpenMP®

```
Solution Advancement: 3.122815
Boundary Condition Enforcement:  0.585140
Ghost Cell Update:  0.462915
Total: 5.616494
```

Ver1 HIP

```
Solution Advancement: 8.047540
Boundary Condition Enforcement:  0.571013
Ghost Cell Update:  1.992076
Total: 11.819252
```

Ver4 OpenMP®

```
Solution Advancement: 3.104172
Boundary Condition Enforcement:  0.445526
Ghost Cell Update:  0.489475
Total: 5.314528
```

Ver4 HIP

```
Solution Advancement: 7.120858
Boundary Condition Enforcement:  0.208986
Ghost Cell Update:  2.710922
Total: 11.053205
```

```
roctxRangePush("BufAlloc");
xbuf_left_send = (double *)malloc(bufcount*sizeof(double));
xbuf_rght_send = (double *)malloc(bufcount*sizeof(double));
xbuf_rght_recv = (double *)malloc(bufcount*sizeof(double));
xbuf_left_recv = (double *)malloc(bufcount*sizeof(double));
roctxRangePop(); //BufAlloc
```

Run on MI210 with ROCm 6.4.2 and OpenMPI 5.0.7

Ghost Exchange: Ver4 vs Ver6 Timings Comparison

Ver4 OpenMP®

```
Solution Advancement: 3.104172
Boundary Condition Enforcement:  0.445526
Ghost Cell Update:  0.489475
Total: 5.314528
```

export HSA_XNACK=1

Ver4 HIP

```
Solution Advancement: 7.120858
Boundary Condition Enforcement:  0.208986
Ghost Cell Update:  2.710922
Total: 11.053205
```

Ver6 OpenMP®

```
Solution Advancement: 1.210582
Boundary Condition Enforcement:  0.613608
Ghost Cell Update:  0.384211
Total: 2.227684
```

unset HSA_XNACK

Ver6 HIP

```
Solution Advancement: 5.505561
Boundary Condition Enforcement:  0.019133
Ghost Cell Update:  0.791291
Total: 6.637961
```

```
#pragma omp target enter data map(alloc: xbuf_left_send[0:bufcount], xbuf_rght_send[0:bufcount])
#pragma omp target enter data map(alloc: xbuf_rght_recv[0:bufcount], xbuf_left_recv[0:bufcount])
```

```
#pragma omp target enter data map(alloc: x[0:totcells], xnew[0:totcells])
```

```
roctxRangePush("BufAlloc");
HIP_CHECK(hipMalloc(&xbuf_left_send, bufcount*sizeof(double)));
HIP_CHECK(hipMalloc(&xbuf_rght_send, bufcount*sizeof(double)));
HIP_CHECK(hipMalloc(&xbuf_rght_recv, bufcount*sizeof(double)));
HIP_CHECK(hipMalloc(&xbuf_left_recv, bufcount*sizeof(double)));
roctxRangePop(); //BufAlloc
```

```
HIP_CHECK(hipMalloc(&x, totcells * sizeof(double)));
HIP_CHECK(hipMalloc(&xnew, totcells * sizeof(double)));
```

Run on MI210 with ROCm 6.4.2 and OpenMPI 5.0.7

Ghost Exchange Ver1 HIP (no affinity settings)

```
$ mpirun -n 4 ./GhostExchange -x 2  -y 2  -i 20000 -j 20000 -h 1 -c -I 100
```

```
Solution Advancement: 8.112173
Boundary Condition Enforcement:  0.493496
Ghost Cell Update:  1.985867
Total: 11.952207
```

```
MPI 003 - HWT 131 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID N/A
MPI 002 - HWT 130 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID N/A
MPI 000 - HWT 128 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID N/A
MPI 001 - HWT 129 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID N/A
```

MPI rank

Hardware thread process ran on

GPUs available to process

GPU bound to process

Ghost Exchange Ver1 HIP (GPU affinity)

```
$ mpirun -n 4 ../../set_gpu_device.sh ./GhostExchange -x 2  -y 2  -i 20000 -j 20000 -h 1 -c -I 100
```

```
Solution Advancement: 2.004534
Boundary Condition Enforcement:  0.020589
Ghost Cell Update:  0.661813
Total: 3.264538
```

3.66x speed-up

```
MPI 000 - HWT 128 - RT_GPU_ID 0 - GPU_ID 3
MPI 003 - HWT 131 - RT_GPU_ID 0 - GPU_ID 0
MPI 002 - HWT 002 - RT_GPU_ID 0 - GPU_ID 1
MPI 001 - HWT 129 - RT_GPU_ID 0 - GPU_ID 2
```

MPI rank

Hardware thread process ran on

GPUs available to process

GPU bound to process

Ghost Exchange example summary

- Porting simple MPI communication examples to the GPU can be straightforward
  - Use of complex MPI Datatypes can make it more difficult
- Some of the kernels for MPI communication and boundary cell conditions do not have a lot of work
  - On the MI300A, it might be better to do this on the CPU – see Ver8 HIP
- Additionally shown:
  - Taking advantage of Managed Memory (MI 200 series) and Unified Address (MI300A) for porting
  - Affinity and process placement
  - Avoiding memory allocations and transfers by allocating memory once on the GPU in the main routine

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED "AS IS" WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

AMD