MULTI GPU PROGRAMMING
WITH CUDA AND MPI

Jiri Kraus, Senior Devtech Compute, April 25th 2017
//MPI rank 0
MPI_Send(sbuf_d, size, MPI_DOUBLE, n-1, tag, MPI_COMM_WORLD);

//MPI rank n-1
MPI_Recv(rbuf_d, size, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
USING MPI FOR INTER GPU COMMUNICATION
MESSAGE PASSING INTERFACE - MPI

Standard to exchange data between processes via messages

  Defines API to exchanges messages

    Point to Point: e.g. MPI_Send, MPI_Recv

    Collectives: e.g. MPI_Reduce

Multiple implementations (open source and commercial)

  Bindings for C/C++, Fortran, Python, ...

  E.g. MPICH, OpenMPI, MVAPICH, IBM Platform MPI, Cray MPT, ...
#include <mpi.h>

int main(int argc, char *argv[]) {
    int rank, size;
    /* Initialize the MPI library */
    MPI_Init(&argc, &argv);
    /* Determine the calling process rank and total number of ranks */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    /* Call MPI routines like MPI_Send, MPI_Recv, ... */
    ...
    /* Shutdown MPI library */
    MPI_Finalize();
    return 0;
}
MPI
Compiling and Launching

$ mpicc -o myapp myapp.c
$ mpirun -np 4 ./myapp <args>

rank = 0
myapp

rank = 1
myapp

rank = 2
myapp

rank = 3
myapp
EXAMPLE: JACOBI SOLVER

Solves the 2D-Laplace Equation on a rectangle

\[ \Delta u(x, y) = 0 \ \forall \ (x, y) \in \Omega \setminus \delta \Omega \]

Dirichlet boundary conditions (constant values on boundaries) on left and right boundary

Periodic boundary conditions on top and bottom boundary

Domain decomposition with stripes

Horizontal Stripes
EXAMPLE: JACOBI SOLVER

Single GPU

While not converged

Do Jacobi step:

```c
for (int iy = 1; iy < (ny-1); ++iy)
  for (int ix = 1; ix < (nx-1); ++ix)
    a_new[iy*nx+ix] = 0.25f*(
      a[iy*nx+ix+1] + a[iy*nx+ix-1] +
      a[(iy+1)*nx+ix] + a[(iy-1)*nx+ix]);
```

Apply periodic boundary conditions

swap a_new and a

Next iteration
EXAMPLE: JACOBI SOLVER

Multi GPU

While not converged

Do Jacobi step:

```c
for (int iy = iy_start; iy < iy_end; ++iy)
    for (int ix = 1; ix < (nx-1); ++ix)
        a_new[iy*nx+ix] = 0.25f*((a[iy*nx+ix+1]+a[iy*nx+ix-1] + a[(iy+1)*nx + ix] + a[(iy-1)*nx + ix]));
```

Apply periodic boundary conditions and exchange halo with 2 neighbors

swap a_new and a

Next iteration
EXAMPLE JACOBI
Top/Bottom Halo

MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_DOUBLE, top, 0,
a_new+iy_end*nx, nx, MPI_DOUBLE, bottom, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
EXAMPLE JACOBI

Top/Bottom Halo

```c
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_DOUBLE, top, 0,
             a_new+iy_end*nx, nx, MPI_DOUBLE, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```
EXAMPLE JACOBI

Top/Bottom Halo

MPI_Sendrecv

\[ a_{\text{new}} + iy_{\text{start}} \times nx, \text{nx, MPI\_DOUBLE, top, 0,} \]
\[ a_{\text{new}} + iy_{\text{end}} \times nx, \text{nx, MPI\_DOUBLE, bottom, 0,} \]
\[ \text{MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE}; \]

MPI_Sendrecv

\[ a_{\text{new}} + (iy_{\text{end}} - 1) \times nx, \text{nx, MPI\_DOUBLE, bottom, 1,} \]
\[ a_{\text{new}} + (iy_{\text{start}} - 1) \times nx, \text{nx, MPI\_DOUBLE, top, 1,} \]
\[ \text{MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE}; \]
HANDLING MULTI GPU NODES

GPU-affinity

Use local rank:

```c
int local_rank = //determine local rank
int num_devices = 0;

cudaGetDeviceCount(&num_devices);

cudaSetDevice(local_rank % num_devices);
```
HANDLING MULTI GPU NODES
How to determine the local rank? - MPI-3

```c
MPI_Comm local_comm;
MPI_Info info;
MPI_Info_create(&info);

MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, info, &local_comm);

int local_rank = -1;

MPI_Comm_rank(local_comm,&local_rank);

MPI_Comm_free(&local_comm);

MPI_Info_free(&info);
```
PROFILING OF MPI+CUDA APPS
PROFILING MPI+CUDA APPLICATIONS

Using nvprof+NVVP

Embed MPI rank in output filename, process name, and context name

```bash
mpirun -np $np nvprof --output-profile profile.%q{OMPI_COMM_WORLD_RANK} \
    --process-name "rank %q{OMPI_COMM_WORLD_RANK}" \
    --context-name "rank %q{OMPI_COMM_WORLD_RANK}" 
```

Alternative:

Only save the textual output (`--log-file`)

OpenMPI: OMPI_COMM_WORLD_RANK

MVAPICH2: MV2_COMM_WORLD_RANK
PROFILING MPI+CUDA APPLICATIONS

Using \texttt{nvprof+NVVP}
PROFILING MPI+CUDA APPLICATIONS

Using nvprof+NVVP

Use the import Wizard

Multiple parallel profiling tools are CUDA-aware

Score-P
Vampir
Tau

These tools are good for discovering MPI issues as well as basic CUDA performance inhibitors.
OVERLAPPING MPI AND COMPUTATION
COMMUNICATION + COMPUTATION OVERLAP

OpenMPI 1.10.2 - 2 Tesla K40

![Runtime vs Problem Size Graph]

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>No overlap</th>
<th>Ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096x4096</td>
<td>Green</td>
<td>Cyan</td>
</tr>
<tr>
<td>2048x2048</td>
<td>Yellow</td>
<td>Teal</td>
</tr>
<tr>
<td>1024x1024</td>
<td>Red</td>
<td>Blue</td>
</tr>
</tbody>
</table>
COMMUNICATION + COMPUTATION OVERLAP

No Overlap
- Process Whole Domain

Overlap
- Process boundary domain
- Process inner domain
  - Boundary and inner domain processing can overlap
  - Possible gain

Dependency
- MPI

MPI
Asynchronous execution with CUDA streams

launch_jacobi_kernel( a_new, a, 12_norm_m, iy_start, (iy_start+1), nx, halo_stream );
launch_jacobi_kernel( a_new, a, 12_norm_m, (iy_end-1), iy_end, nx, halo_stream );
launch_jacobi_kernel( a_new, a, 12_norm_m, (iy_start+1), (iy_end-1), nx, compute_stream );
int top = rank > 0 ? rank - 1 : (size-1); int bottom = (rank+1)%size;

//Apply periodic boundary conditions
cudaStreamSynchronize( halo_stream );
MPI_Sendrecv( a_new+iy_start*nx, nx, MPI_REAL_TYPE, top , 0,
    a_new+(iy_end*nx), nx, MPI_REAL_TYPE, bottom, 0,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE ));
MPI_Sendrecv( a_new+(iy_end-1)*nx, nx, MPI_REAL_TYPE, bottom, 0,
    a_new+(iy_start-1)*nx, nx, MPI_REAL_TYPE, top, 0,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE ));
cudaStreamSynchronize( compute_stream );
COMMUNICATION + COMPUTATION OVERLAP

OpenMPI 1.10.2 - 2 Tesla K40
MPI AND UNIFIED MEMORY
MPI AND UNIFIED MEMORY

CAVEAT

Using Unified Memory with a non Unified Memory-aware MPI might break in some cases, e.g. when registering memory for RDMA, or even worse silently produce wrong results.

Use a Unified Memory-aware MPI with Unified Memory and MPI

Unified Memory-aware: CUDA-aware MPI with support for Unified Memory
Available Unified Memory-aware MPI implementations

- OpenMPI (since 1.8.5)
- MVAPICH2-GDR (since 2.2b)
  - Performance improvements with 2.2RC1 for Intranode GPU to GPU communication

Currently both treat all Unified Memory as Device Memory

Good performance if all buffers used in MPI are touched mainly on the GPU.
MPI AND UNIFIED MEMORY

Without Unified Memory-aware MPI

Only use non Unified Memory Buffers for MPI: cudaMalloc, cudaMallocHost or malloc

Application managed non Unified Memory Buffers also allow to work around current missing cases in Unified Memory-aware MPI Implementations.
HANDS-ON
HANDS-ON MENU

4 tasks to choose from

Task 0: Using MPI

Task 1: Handle GPU Affinity

Task 2: Apply Domain Decomposition

Task 3: Overlap MPI and Compute
**TASK 0: USING MPI**

**task0**

Determine rank (MPI_Comm_rank) and size (MPI_Comm_size)

Add MPI_Barrier to ensure correct timing

Num GPUs: 1.
2048x2048: 1 GPU: 18.9308 s, 1 GPUs: 18.8297 s, speedup:
Num GPUs: 1.
2048x2048: 1 GPU: 18.9888 s, 1 GPUs: 18.8055 s, speedup:

Look for TODOs

Make Targets:
- run: run jacobi with $NP procs.
- build jacobi bin (default)
- memcheck: profile with cuda-memcheck
- profile: profile with nvprof
- Solution is in solution0

https://www.open-mpi.org/doc/current/
**TASK 1: HANDLING GPU AFFINITY**

Handle GPU affinity with `MPI_COMM_TYPE_SHARED`

Run and report the performance

```plaintext
900, 0.061306
Num GPUs: 2.
2048x2048: 1 GPU: 18.9678 s, 2 GPUs: 18.8432 s, speedup: 1.01
```

Look for TODOs

Make Targets:
- `run`: run `jacobi` with `$NP` procs.
- `jacobi`: build `jacobi` bin (default)
- `memcheck`: profile with `cuda-memcheck`
- `profile`: profile with `nvprof`
- Solution is in `solution1`

[https://www.open-mpi.org/doc/current/](https://www.open-mpi.org/doc/current/)
TASK 2: APPLY DOMAIN DECOMPOSITION

Calculate first \((iy\_start)\) and last \((iy\_end)\) row to be processed by each rank

Use `MPI_Sendrecv` to handle halo updates and periodic boundary conditions

Use `MPI_Allreduce` to calculate global L2 norm

900, 0.061306
Num GPUs: 2.
2048x2048: 1 GPU: 8.7768 s, 2 GPUs: 8.7579 s, speedup: 1.00

https://www.open-mpi.org/doc/current/

Make Targets:
- run: run jacobi with $NP$ procs.
- jacobi: build jacobi bin (default)
- memcheck: profile with cuda-memcheck
- profile: profile with nvprof
Solution is in solution2

Look for TODOs
Use `cudaStreamCreate` to create halo processing stream

Split jacobi step in top boundary, bottom boundary and bulk part

Launch top and bottom boundary part in halo processing stream

900, 0.061310
Num GPUs: 2.
2048x2048: 1 GPU: 8.8221 s, 2 GPUs: 4.4896 s, speedup: 1.97

https://www.open-mpi.org/doc/current/

Make Targets:
- run: run jacobi with $NP$ procs.
- jacobi: build jacobi bin (default)
- memcheck: profile with cuda-memcheck
- profile: profile with nvprof

Solution is in solution3
SOLUTIONS
int main(int argc, char * argv[]) {
    int rank = 0;
    int size = 1;
    MPI_CALL( MPI_Init(&argc,&argv) );
    MPI_CALL( MPI_Comm_rank(MPI_COMM_WORLD,&rank) );
    MPI_CALL( MPI_Comm_size(MPI_COMM_WORLD,&size) );
    //...
    MPI_CALL( MPI_Barrier(MPI_COMM_WORLD) );
    double start = MPI_Wtime();
    while ( l2_norm > tol && iter < iter_max )
    //...
    MPI_CALL( MPI_Finalize() );
    return result_correct == 1 ? 0 : 1; }

TASK 0: USING MPI
Solution
```c
int dev_id = -1;
{
    MPI_Comm local_comm;
    MPI_Info info;
    MPI_CALL( MPI_Info_create(&info) );
    MPI_CALL( MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED,
                                   rank, info, &local_comm) );
    MPI_CALL( MPI_Comm_rank(local_comm,&dev_id) );
    MPI_CALL( MPI_Comm_free(&local_comm) );
    MPI_CALL( MPI_Info_free(&info) );
}
CUDA_RT_CALL( cudaSetDevice( dev_id ) );
```
// Ensure correctness if ny%size != 0
int chunk_size = std::ceil( (1.0*ny)/size );
int iy_start = rank*chunk_size;
int iy_end = iy_start+chunk_size;
// Do not process boundaries
iy_start = std::max( iy_start, 1 );
iy_end = std::min( iy_end, ny - 1 );
TASK 2: APPLY DOMAIN DECOMPOSITION

Solution II

//Apply periodic boundary conditions
CUDA_RT_CALL(cudaStreamSynchronize(compute_stream));
PUSH_RANGE("MPI", 5)
MPI_CALL(MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_REAL_TYPE, top, 0,
a_new+(iy_end*nx), nx, MPI_REAL_TYPE, bottom, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE));
MPI_CALL(MPI_Sendrecv(a_new+(iy_end-1)*nx, nx, MPI_REAL_TYPE, bottom, 0,
a_new+(iy_start-1)*nx, nx, MPI_REAL_TYPE, top, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE));
POP_RANGE
CUDA_RT_CALL( cudaStreamSynchronize( compute_stream ) );

MPI_CALL( MPI_Allreduce( l2_norm_m, &l2_norm, 1, MPI_REAL_TYPE,
                         MPI_SUM, MPI_COMM_WORLD ) );

l2_norm = std::sqrt( l2_norm );
launch_jacobi_kernel( a_new, a, l2_norm_d,
    iy_start, (iy_start+1), nx, halo_stream );
launch_jacobi_kernel( a_new, a, l2_norm_d,
    (iy_end-1), iy_end, nx, halo_stream );
launch_jacobi_kernel( a_new, a, l2_norm_d, (iy_start+1),
    (iy_end-1), nx, compute_stream );

int top = rank > 0 ? rank - 1 : (size-1); int bottom = (rank+1)%size;

//Apply periodic boundary conditions
CUDA_RT_CALL( cudaStreamSynchronize( halo_stream ) );
PUSH_RANGE("MPI",5)
MPI_CALL( MPI_Sendrecv( a_new+iy_start*nx, ...
TASK 3: OVERLAP MPI AND COMPUTE

Solution