MULTI-GPU PROGRAMMING MODELS

Jiri Kraus, Senior Devtech Compute
Jan Stephan, Intern Devtech Compute
MOTIVATION
Why use multiple GPUs?

Need to compute larger, e.g. bigger networks, car models, ...

Need to compute faster, e.g. weather prediction

Better energy efficiency with dense nodes with multiple GPUs
DGX-1V

Two fully connected quads, connected at corners

300GB/s per GPU bidirectional to Peers

Load/store access to Peer Memory

Full atomics to Peer GPUs

High speed copy engines for bulk data copy

PCIe to/from CPU
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
EXAMPLE: JACOBI SOLVER

Single GPU

While not converged

Do Jacobi step:

\[
\begin{align*}
  \text{for ( int iy = 1; iy < ny-1; iy++) } & \\
  \text{for ( int ix = 1; ix < nx-1; ix++) } & \\
  a_{\text{new}}[iy*nx+ix] &= -0.25 * \\
  & \quad -( a[ iy *nx+(ix+1)] + a[ iy *nx+ix-1] \\
  & \quad \quad + a[(iy-1)*nx+ ix ] + a[(iy+1)*nx+ix ] );
\end{align*}
\]

Apply periodic boundary conditions

Swap \texttt{a\_new} and \texttt{a}

Next iteration
DOMAIN DECOMPOSITION

Different Ways to split the work between processes:

Minimize number of neighbors:
Communicate to less neighbors
Optimal for latency bound communication

Minimize surface area/volume ratio:
Communicate less data
Optimal for bandwidth bound communication

Horizontal Stripes
Contiguous if data is row-major

Vertical Stripes
Contiguous if data is column-major

Tiles
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] = -0.25 *
   - ( a[ iy  *nx+(ix+1) ] + a[ iy  *nx+ix-1]
       + a[(iy-1)*nx+ ix ] + a[(iy+1)*nx+ix ] )
```

Apply periodic boundary conditions

Exchange halo with 2 neighbors

Swap `a_new` and `a`

Next iteration
while ( l2_norm > tol && iter < iter_max ) {
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
        const int top = dev_id > 0 ? dev_id - 1 : (num_devices-1); const int bottom = (dev_id+1)%num_devices;
        cudaSetDevice( dev_id );
        cudaMemcpyAsync( l2_norm_d[dev_id], 0 , sizeof(real) );
        jacobi_kernel<<<dim_grid,dim_block>>>( a_new[dev_id], a[dev_id], l2_norm_d[dev_id],
            iy_start[dev_id], iy_end[dev_id], nx );
        cudaMemcpyAsync( 12_norm_h[dev_id], 12_norm_d[dev_id], sizeof(real), cudaMemcpyDeviceToHost );
        cudaMemcpyAsync( a_new[top]+(iy_end[top]*nx), a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
        cudaMemcpyAsync( a_new[bottom], a_new[dev_id]+(iy_end[dev_id]-1)*nx, nx*sizeof(real), ...);
    }
    l2_norm = 0.0;
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
        cudaSetDevice( dev_id ); cudaDeviceSynchronize();
        12_norm += *(12_norm_h[dev_id]);
    }
    l2_norm = std::sqrt( 12_norm );
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) std::swap(a_new[dev_id],a[dev_id]);
    iter++;
}
cudaMemcpyAsync(
    a_new[top] + (iy_end[top] * nx),
    a_new[dev_id] + iy_start[dev_id] * nx, nx * sizeof(real), ...);
EXAMPLE JACOBI
Top/Bottom Halo

cudaMemcpyAsync(
    a_new[top]+(iy_end[top]*nx),
    a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
cudaMemcpyAsync(
    a_new[top]+(iy_end[top]*nx),
    a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);

cudaMemcpyAsync(
    a_new[bottom],
    a_new[dev_id]+(iy_end[dev_id]-1)*nx, nx*sizeof(real), ...);
SCALABILITY METRICS FOR SUCCESS

Serial Time: $T_s$: How long it takes to run the problem with a single process

Parallel Time: $T_p$: How long it takes to run the problem with multiple processes

Number of Processes: $P$: The number of Processes operating on the task at hand

Speedup: $S = \frac{T_s}{T_p}$: How much faster is the parallel version vs. serial. (optimal is $P$)

Efficiency: $E = \frac{S}{P}$: How efficient are the processors used (optimal is 1)
EXAMPLE: JACOBI SOLVER

Single GPU performance vs. problem size - Tesla V100 SXM2

Performance (Mcells/s) vs. Problem size (nx=ny)

- Performance (Mcells/s)
- Efficiency (%)
- Achieved Occupancy (%)
MULTI GPU JACOBI RUNTIME
DGX-1V - 7168 x 7168, 1000 iterations

Parallel Efficiency

Runtime (s)

#GPUs

Single Threaded Copy
Parallel Efficiency
MULTI GPU JACOBI NVVP TIMELINE

Single Threaded Copy 4 V100 on DGX-1V
Maximizes intra node inter GPU Bandwidth
Avoids Host memory and system topology bottlenecks
GPUDIRECT P2P
Enable P2P

```c
for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
    cudaSetDevice( dev_id );
    const int top = dev_id > 0 ? dev_id - 1 : (num_devices-1);
    int canAccessPeer = 0;
    cudaDeviceCanAccessPeer ( &canAccessPeer, dev_id, top );
    if ( canAccessPeer )
        cudaDeviceEnablePeerAccess ( top, 0 );
    const int bottom = (dev_id+1)%num_devices;
    if ( top != bottom ) {
        cudaDeviceCanAccessPeer ( &canAccessPeer, dev_id, bottom );
        if ( canAccessPeer )
            cudaDeviceEnablePeerAccess ( bottom, 0 );
    }
}
```
MULTI GPU JACOBI NVVP TIMELINE

Single Threaded Copy 4 V100 on DGX-1V with P2P

![Diagram of GPU operations](image-url)
MULTI GPU JACOBI RUNTIME

DGX-1V - 7168 x 7168, 1000 iterations
Halo updates for 1D domain decomposition with periodic boundary conditions

Unidirectional rings are important building block for collective algorithms
MAPPING 1D RING EXCHANGE TO DGX-1V
MAPPING 1D RING EXCHANGE TO DGX-1V

```bash
export CUDA_VISIBLE_DEVICES="0,3,2,1,5,6,7,4"
```
MULTI GPU JACOBI RUNTIME

DGX-1V - 7168 x 7168, 1000 iterations
MULTI GPU JACOBI NVVP TIMELINE

Single Threaded Copy 4 V100 on DGX-1V with P2P
```
int num_devices = 0;
cudaGetDeviceCount( &num_devices );
#pragma omp parallel num_threads( num_devices )
{
    int dev_id = omp_get_thread_num();
    cudaSetDevice( dev_id );
}
```
MULTI GPU JACOBI NVVP TIMELINE

Multi Threaded Copy 4 V100 on DGX-1V with P2P
MULTI GPU JACOBI RUNTIME

DGX1 - 1024 x 1024, 1000 iterations

Parallel Efficiency

#GPUs

- Single Threaded Copy P2P
- Multi Threaded Copy (no thread pinning)
GPU/CPU AFFINITY

GPU0 <-> GPU1
GPU2 <-> GPU3
GPU5 <-> GPU4
GPU7 <-> GPU6

CPU 0
0 - 19

CPU 1
20-39

thread 0 <-> thread 1 <-> thread 2 <-> thread 3
thread 4 <-> thread 5 <-> thread 6 <-> thread 7
## GPU/CPU Affinity

Querying system topology with `nvidia-smi topo -m`

<table>
<thead>
<tr>
<th>GPU0</th>
<th>GPU1</th>
<th>GPU2</th>
<th>GPU3</th>
<th>GPU4</th>
<th>GPU5</th>
<th>GPU6</th>
<th>GPU7</th>
<th>mlx5_0</th>
<th>mlx5_2</th>
<th>mlx5_1</th>
<th>mlx5_3</th>
<th>CPU Affinity</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>NV2</td>
<td>NV2</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>0–19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU1</td>
<td>NV1</td>
<td>X</td>
<td>NV2</td>
<td>NV1</td>
<td>SOC</td>
<td>NV2</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>0–19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU2</td>
<td>NV1</td>
<td>NV2</td>
<td>X</td>
<td>NV2</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU3</td>
<td>NV2</td>
<td>NV1</td>
<td>NV2</td>
<td>X</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>PHB</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU4</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>NV2</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU5</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>X</td>
<td>NV2</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU6</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>NV2</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>PIX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU7</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>NV2</td>
<td>X</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PIX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mlx5_0</td>
<td>PIX</td>
<td>PIX</td>
<td>PHB</td>
<td>PHB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mlx5_2</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIX</td>
<td>PIX</td>
<td>PHB</td>
<td>PHB</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mlx5_1</td>
<td>PHB</td>
<td>PHB</td>
<td>PIX</td>
<td>PIX</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mlx5_3</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>PHB</td>
<td>PIX</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
GPU/CPU AFFINITY
Using CUDA_VISIBLE_DEVICES and OpenMP env. vars.

export OMP_PROC_BIND=TRUE
export CUDA_VISIBLE_DEVICES="0,3,2,1,5,6,7,4"
export OMP_PLACES="0,1,2,3,20,21,22,23"
MULTI GPU JACOBI RUNTIME
DGX-1V - 7168 x 7168, 1000 iterations

Parallel Efficiency

#GPUs

Single Threaded Copy P2P  Multi Threaded Copy (no thread pinning)  Multi Threaded Copy
MULTI GPU JACOBI NVVP TIMELINE

Multi Threaded Copy 4 V100 on DGX-1V with P2P
COMMUNICATION + COMPUTATION OVERLAP

No Overlap

Process Whole Domain

Overlap

Boundary and inner domain processing can overlap

Process inner domain

Process boundary domain

Dependency

COMM

Possible gain
//Compute bulk
cudaStreamWaitEvent(compute_stream,push_top_done[(iter%2)][dev_id],0);
cudaStreamWaitEvent(compute_stream,push_bottom_done[(iter%2)][dev_id],0);
jacobi_kernel<<<dim_grid,dim_block,0,compute_stream>>>(a_new[dev_id],a,l2_norm_d,(iy_start+1),(iy_end[dev_id]-1),nx);

//Compute boundaries
cudaStreamWaitEvent(push_top_stream, reset_l2norm_done, 0 );
cudaStreamWaitEvent(push_top_stream, push_bottom_done[(iter%2)][top], 0 );
jacobi_kernel<<<nx/128+1,128,0,push_top_stream>>>( a_new[dev_id],a,l2_norm_d,iy_start,(iy_start+1),nx);
cudaStreamWaitEvent(push_bottom_stream,reset_l2norm_done,0);
cudaStreamWaitEvent(push_bottom_stream,push_top_done[(iter%2)][bottom], 0 );
jacobi_kernel<<<nx/128+1,128,0,push_bottom_stream>>>( a_new[dev_id],a,l2_norm_d,(iy_end[dev_id]-1),iy_end[dev_id],nx);

//Apply periodic boundary conditions and exchange halo
cudaMemcpyAsync(a_new[top]+(iy_end[top]*nx),a_new[dev_id]+iy_start*nx,nx*sizeof(real),cudaMemcpyDeviceToDevice,push_top_stream);
cudaEventRecord(push_top_done[((iter+1)%2)][dev_id],push_top_stream);
cudaMemcpyAsync(a_new[bottom],a_new[dev_id]+(iy_end[dev_id]-1)*nx,nx*sizeof(real),cudaMemcpyDeviceToDevice,push_bottom_stream);
cudaEventRecord(push_bottom_done[((iter+1)%2)][dev_id],push_bottom_stream);
 COMMUNICATION + COMPUTATION OVERLAP
High Priority Streams

```c
int leastPriority = 0;

int greatestPriority = leastPriority;

cudaDeviceGetStreamPriorityRange ( &leastPriority, &greatestPriority );

cudaStreamCreateWithPriority ( &compute_stream, cudaStreamDefault, leastPriority );

cudaStreamCreateWithPriority ( &push_top_stream, cudaStreamDefault, greatestPriority );

cudaStreamCreateWithPriority ( &push_bottom_stream, cudaStreamDefault, greatestPriority );
```
MULTI GPU JACOBI NVVP TIMELINE
Multi Threaded Copy Overlap 4 V100 on DGX-1V with P2P
MULTI GPU JACOBI RUNTIME
DGX-1V - 7168 x 7168, 1000 iterations
MULTI GPU JACOBI NVVP TIMELINE
Multi Threaded Copy Overlap 4 V100 on DGX-1V with P2P
while ( l2_norm > tol && iter < iter_max ) {
    cudaMemcpyAsync(l2_norm_d, 0, sizeof(real), compute_stream );
    #pragma omp barrier
    cudaStreamWaitEvent( compute_stream, compute_done[iter%2][top], 0 );
    cudaStreamWaitEvent( compute_stream, compute_done[iter%2][bottom], 0 );
    jacobi_kernel<<<dim_grid,dim_block,0,compute_stream>>>(
        a_new[dev_id], a, l2_norm_d, iy_start, iy_end[dev_id], nx,
        a_new[top], iy_end[top], a_new[bottom], 0 );
    cudaEventRecord( compute_done[(iter+1)%2][dev_id], compute_stream );
    cudaMemcpyAsync(l2_norm,l2_norm_d,sizeof(real),cudaMemcpyToDeviceToHost,compute_stream);
    // l2_norm reduction btw threads skipped ...
    #pragma omp barrier
    std::swap(a_new[dev_id],a); iter++;
}
MULTI THREADED MULTI GPU PROGRAMMING
Using OpenMP and P2P Mappings

```c
__global__ void jacobi_kernel( ... ) {
    const int ix = blockIdx.x*blockDim.x+threadsPerBlock.x;
    const int iy = blockIdx.y*blockDim.y+threadsPerBlock.y + iy_start;
    real local_l2_norm = 0.0;
    if ( iy < iy_end && ix >= 1 && ix < (nx-1) ) {
        const real new_val = 0.25 * ( a[ iy * nx + ix + 1 ] + a[ iy * nx + ix - 1 ]
                                      + a[ (iy+1) * nx + ix ] + a[ (iy-1) * nx + ix ] );
        a_new[ iy * nx + ix ] = new_val;
        if ( iy_start == iy ) a_new_top[ top_iy *nx + ix ] = new_val;
        if ( (iy_end - 1) == iy ) a_new_bottom[ bottom_iy*nx + ix ] = new_val;
        real residue = new_val - a[ iy * nx + ix ];
        local_l2_norm += residue * residue;
    }
    atomicAdd( l2_norm, local_l2_norm );
}
```
MULTI GPU JACOBI RUNTIME
DGX-1V - 7168 x 7168, 1000 iterations

Parallel Efficiency vs. #GPUs

- Single Threaded Copy P2P
- Multi Threaded Copy
- Multi Threaded Copy Overlap
- Multi Threaded P2P
MULTI GPU JACOBI NVVP TIMELINE

Multi Threaded P2P 4 V100 on DGX-1V with P2P
cudamemcpyAsync( l2_norm_h, l2_norm_d, sizeof(real), cudamemcpyDeviceToHost, compute_stream );
#pragma omp barrier
#pragma omp single
{ l2_norm = 0.0; }
#pragma omp barrier
cudaStreamSynchronize( compute_stream );
#pragma omp atomic
l2_norm += *(l2_norm_h);
#pragma omp barrier
#pragma omp single
{ l2_norm = std::sqrt( l2_norm ); }
MULTI THREADED MULTI GPU PROGRAMMING

Delayed L2 norm reduction

`cudaMemcpyAsync( l2_norm_h[ curr ], l2_norm_d[ curr ], sizeof( real ), cudaMemcpyDeviceToHost, compute_stream );`  # Issue H2D copy of current L2 norm

`cudaEventRecord( copy_done[ curr ], compute_stream );`

`cudaEventSynchronize( copy_done[ prev ] );`

`#pragma omp atomic`

`l2_norm[ prev ] += *( l2_norm_h[ prev ] );`

`#pragma omp barrier`

`l2_norm[ prev ] = std::sqrt( l2_norm[ prev ] );`

`#pragma omp barrier`

`l2_norm[ prev ] = 0.0;`  # Check L2 norm of last iteration (hidden)
MULTI GPU JACOBI NVVP TIMELINE

Multi Threaded P2P 4 V100 on DGX-1V with P2P
MULTI GPU JACOBI RUNTIME

DGX-1V - 7168 x 7168, 1000 iterations
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>
EXAMPLE JACOBI
Top/Bottom Halo

MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top, 0,
a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
EXAMPLE JACOBI
Top/Botton Halo

MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top, 0,
a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
EXAMPLE JACOBI
Top/Bottom Halo

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

MPI_Sendrecv(a_new+(iy_end-1)*nx, nx, MPI_FLOAT, bottom, 0,
             a_new, nx, MPI_FLOAT, top, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
HANDLING MULTIPLE MULTI GPU NODES
HANDLING MULTIPLE MULTI GPU NODES
How to determine the local rank? - MPI-3

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);
HANDLING MULTIPLE MULTI GPU NODES

- Shared Comm
- CPU 0 0-19
- CPU 1 20-39

0 1 ... 7
0 1 ... 7
0 1 ... 7
0 1 ... 7

8 9 ... 15
16 12 ... 23
24 25 ... 31
Use local rank:

```c
int local_rank = -1;
MPI_Comm_rank(local_comm,&local_rank);
int num_devices = 0;
cudaGetDeviceCount(&num_devices);
cudaSetDevice(local_rank % num_devices);
```
MULTI GPU JACOBI RUNTIME

DGX-1V - 7168 x 7168, 1000 iterations
MULTI GPU JACOBI NVVP TIMELINE

MPI 4 V100 on DGX-1V
COMMUNICATION + COMPUTATION OVERLAP

```c
launch_jacobi_kernel( a_new, a, 12_norm_d, iy_start, (iy_start+1), nx, push_top_stream );
launch_jacobi_kernel( a_new, a, 12_norm_d, (iy_end-1), iy_end, nx, push_bottom_stream );
launch_jacobi_kernel( a_new, a, 12_norm_d, (iy_start+1), (iy_end-1), nx, compute_stream );
const int top = rank > 0 ? rank - 1 : (size-1);
const int bottom = (rank+1)%size;
cudaStreamSynchronize( push_top_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 );
cudaStreamSynchronize( push_bottom_stream );
MPI_Sendrecv( a_new+(iy_end-1)*nx, nx, MPI_REAL_TYPE, bottom, 0,
    a_new, nx, MPI_REAL_TYPE, top, 0, MPI_COMM_WORLD,
    MPI_STATUS_IGNORE );
```
MULTI GPU JACOBI NVVP TIMELINE

MPI Overlapping 4 V100 on DGX-1V
MULTI GPU JACOBI RUNTIME

DGX-1V - 7168 x 7168, 1000 iterations
NVSHMEM

Implementation of OpenSHMEM, a Partitioned Global Address Space (PGAS) library

Defines API to (symmetrically) allocate memory that is remotely accessible

Defines API to access remote data

One-sided: e.g. shmem_putmem, shmem_getmem

Collectives: e.g. shmem_broadcast

NVSHMEM features

Symmetric memory allocations in device memory

Communication API calls on CPU (standard and stream-ordered)

Allows kernel-side communication (API and LD/ST*) between GPUs (within a single OS instance for the first release)

Interoperable with MPI
#include <shmem.h>
#include <shmemx.h>

int main(int argc, char *argv[]) {
    ...
    MPI_Comm mpi_comm;
    shmемx_init_attr_t attr;
    mpi_comm = MPI_COMM_WORLD;
    attr.mpi_comm = &mpi_comm;
    shmемx_init_attr (SHMEMX_INIT_WITH_MPI_COMM, &attr);
    int npes = shmem_n_pes();
    int mype = shmem_my_pe();
    ...
    return 0;
}
a = (real *) shm嫉_malloc(nx*(chunk_size+2)*sizeof(real));
a_new = (real *) shm嫉_malloc(nx*(chunk_size+2)*sizeof(real));
...
while ( l2_norm > tol && iter < iter_max ) {
   ...
   jacobi_kernel<<<dim_grid,dim_block,0,compute_stream>>>(
      a_new, a, l2_norm_d, iy_start, iy_end, nx,
      top, iy_end_top, bottom, iy_start_bottom );
   shm嫉_barrier_all_stream(compute_stream);
   ...
}
shm嫉_barrier_all_all();
shm嫉_free(a);
shm嫉_free(a_new);
__global__ void jacobi_kernel( ... ) {
    const int ix = bIdx.x*bDim.x+tIdx.x;
    const int iy = bIdx.y*bDim.y+tIdx.y + iy_start;
    real local_l2_norm = 0.0;
    if ( iy < iy_end && ix >= 1 && ix < (nx-1) ) {
        const real new_val = 0.25 * ( a[ iy * nx + ix + 1 ] + a[ iy * nx + ix - 1 ]
                                      + a[ (iy+1) * nx + ix ] + a[ (iy-1) * nx + ix ] );
        a_new[ iy * nx + ix ] = new_val;
        if ( iy_start == iy )
            shmem_float_p(a_new + top_iy*nx + ix, new_val, top_pe);
        if ( iy_end == iy )
            shmem_float_p(a_new + bottom_iy*nx + ix, new_val, bottom_pe);
        real residue = new_val - a[ iy * nx + ix ];
    }
    atomicAdd( l2_norm, local_l2_norm );
}
MULTI GPU JACOBI RUNTIME

DGX-1V - 7168 x 7168, 1000 iterations
NOT COVERED IN THIS TALK I

MPI with GPUDirect Async support (under development)

Enables MPI communication to follow CUDA stream ordering

Avoids unwanted CPU/GPU synchronization
NOT COVERED IN THIS TALK II
NCCL: Accelerating multi-GPU collective communications

GOAL:
• A library for collective communication using CUDA kernels for reduction and data movement.

APPROACH:
• Allreduce, Reduce, Broadcast, ReduceScatter and Allgather primitives, similar to MPI primitives.
• CUDA oriented : works on CUDA pointers only, enqueues operations to CUDA streams.
• Supports any mapping between processes, threads and GPUs per thread to integrate into any hybrid model.
## CONCLUSION

<table>
<thead>
<tr>
<th>Programming Models</th>
<th>GPU Direct P2P</th>
<th>Multi Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Threaded</td>
<td>CUDA</td>
<td>Improves Perf.</td>
</tr>
<tr>
<td>Multi Threaded</td>
<td>CUDA + OpenMP/TBB/...</td>
<td>Improves Perf.</td>
</tr>
<tr>
<td>Multi Threaded P2P</td>
<td>CUDA + OpenMP/TBB/...</td>
<td>Required</td>
</tr>
<tr>
<td>MPI</td>
<td>CUDA + MPI</td>
<td>Improves Perf.</td>
</tr>
<tr>
<td>NVSHMEM</td>
<td>CUDA + NVSHMEM (MPI interop.)</td>
<td>Required</td>
</tr>
</tbody>
</table>

*Initial version will only support a single OS instance.