S9677 - NVSHMEM: A PARTITIONED GLOBAL ADDRESS SPACE LIBRARY FOR NVIDIA GPU CLUSTERS
Anshuman Goswami, Akhil Langer, Sreeram Potluri, NVIDIA
AGENDA

GPU Programming Models
Overview of NVSHMEM
Porting to NVSHMEM
Future Work
Conclusion and Future Work
GPU FOR COMPUTE OFFLOAD

**Compute** on GPU
**Communication** from CPU
Synchronization at boundaries
Offload latencies in critical path
Hiding increases code complexity
Avoids offload latencies

Compute - communication overlap

Easier to express algorithms with inline communication

Improving performance while making it easier to program
AGENDA

GPU Programming Models
Overview of NVSHMEM
Porting to NVSHMEM
Future Work
Conclusion and Future Work
WHAT IS NVSHMEM?

Experimental implementation of OpenSHMEM for NVIDIA GPUs, 1 PE/GPU

- shared memory: shmem_malloc
- private memory: cudaMalloc

shmem communication APIs: shared->shared or private->shared
**DEVICE-INITIATED COMMUNICATION**

Thread-level communication APIs

Allow finer grained control and overlap

Maps well onto NVLink fabric – DGX-1/DGX-2

```c
__global__ void stencil_single_step (float *u, ...)
{
int ix = threadIdx.x, iy = threadIdx.y;

//compute

//data exchange
if (iy == ny) {
    shmem_float_p (u + ny*nx + ix, u + ix, top_pe);
}
if (iy == 0) {
    shmem_float_p (u + nx + ix, u + (ny+1)*nx + ix, bottom_pe);
}
}
```
THREAD-GROUP COMMUNICATION

Operations can be issued by a WARP/CTA

Coarser, hence more efficient transfers over networks like IB

Still allows inter-warp/inter-block overlap

```c
__global__ void stencil_single_step (u, ...)
{
    //compute

    //data exchange
    shmem_float_put_block_nbi (u + ny*nx, u, nx, top_pe);
    shmem_float_put_block_nbi (u + nx, u + (ny+1)*nx, nx, bottom_pe);
}
```
IN-KERNEL SYNCHRONIZATION

Allows inter-PE synchronization

Can offload larger portions of application running CUDA kernels

```c
__global__ void stencil_uber(u, ...)
{
  while (iter=0; iter<N; iter++) {
    //compute
    //data exchange
    shmem_float_put_nbi_block (u + ny*nx, u, nx, top_pe);
    shmem_float_put_nbi_block (u + nx, u + (ny+1)*nx, nx, bottom_pe);
    shmem_barrier_all();
  }
}
```
### COLLECTIVE KERNEL LAUNCH

<table>
<thead>
<tr>
<th></th>
<th>GPUs supported</th>
<th>CUDA kernel launch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device-Initiated</td>
<td>Kepler or newer</td>
<td>regular &lt;&lt;&lt;&gt;&gt;&gt; or launch APIs</td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device-Initiated</td>
<td>Volta or newer</td>
<td><code>shmex_collective_launch</code></td>
</tr>
<tr>
<td>Synchronization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Provides progress when using device-side inter-kernel synchronization

Built on CUDA cooperative launch and requirement of 1PE/GPU
Not optimal to move all communication/synchronization into CUDA kernels

Inter-CTA synchronization latencies can be longer than kernel launch latencies

Allows mixing fine-grained communication + coarse-grained synchronization
NVLink or PCIe uses CUDA IPC under the hood

\texttt{shmem\_put/get} on device

\texttt{ld/store}

\texttt{shmem\_put/get\_on\_stream}

\texttt{cudaMemcpyAsync}
MULTI-NODE SUPPORT

Reverse offloads network transfers to the CPU

Avoids memory fences when signaling CPU

Uses standard IB verbs (Mellanox OFED for GPUDirect RDMA)
NVSHMEM STATUS

Research vehicle for designing and evaluating GPU-centric workloads

Early access (EA2) available - please reach out to nvshmem@nvidia.com

Main Features

- NVLink and PCIe support
- InfiniBand support (new)
- X86 and Power9 (new) support
- Interoperability with MPI and OpenSHMEM (new) libraries
AGENDA

GPU Programming Models
Overview of NVSHMEM
Porting to NVSHMEM
Future Work
Conclusion and Future Work
PORTING TO USE NVSHMEM FROM GPU

**Step I** : Only communication from inside the kernel (on Kepler or newer GPUs)

**Step II** : Both communication and synchronization from inside the kernel (on Pascal or newer Tesla GPUs)

Using Jacobi Solver, we will walk through I and II and compare with MPI version

Code available at : github.com/NVIDIA/multi-gpu-programming-models
EXAMPLE: JACOBI SOLVER

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

Apply periodic boundary conditions

Swap `a_new` and `a`

Next iteration
__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;
        real residue = new_val - a[ iy * nx + ix ];
        local_l2_norm += residue * residue;
    }
    atomicAdd( l2_norm, local_l2_norm );
}
HOST CODE - MPI

Once every n iterations
while (iter < iter_max) 
{
    //reset norm
    CUDA_RT_CALL(cudahostAsync(l2_norm_d, 0, sizeof(real), compute_stream));
    CUDA_RT_CALL(cudahostRecord(l2_norm_done, compute_stream));

    //compute boundary
    CUDA_RT_CALL(cudastreamWaitEvent(push_top_stream, l2norm_done, 0));
    launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, iy_start+1, nx, push_top_stream);
    CUDA_RT_CALL(cudahostRecord(push_top_done, push_top_stream));
    CUDA_RT_CALL(cudastreamWaitEvent(push_top_stream, reset_l2norm_done, 0));
    launch_jacobi_kernel(a_new, a, l2_norm_d, iy_end-1, iy_end, nx, push_bottom_stream);
    CUDA_RT_CALL(cudahostRecord(push_bottom_done, push_bottom_stream));

    //compute interior
    launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start+1, iy_end-1, nx, compute_stream);

    //Apply periodic boundary conditions
    CUDA_RT_CALL(cudastreamSynchronize(push_top_stream));
    MPI_CALL(MPI_Sendrecv(a_new+i*y_start*nx, nx, MPI_REAL_TYPE, top, 0, a_new+(iy_end-1)*nx, nx, MPI_REAL_TYPE, bottom, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE));
    CUDA_RT_CALL(cudastreamSynchronize(push_bottom_stream));
    MPI_CALL(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));

    //Periodic convergence check
    if ((iter % nccheck) == 0 || (csv && (iter % 100) == 0)) {
        CUDA_RT_CALL(cudastreamWaitEvent(compute_stream, push_top_done, 0));
        CUDA_RT_CALL(cudastreamWaitEvent(compute_stream, push_bottom_done, 0));
        CUDA_RT_CALL(cudahostAsync(l2_norm_h, l2_norm_d, sizeof(real), cudahostDeviceToHost, compute_stream));
        CUDA_RT_CALL(cudastreamSynchronize(compute_stream));
        MPI_CALL(MPI_Allreduce(l2_norm_h, &l2_norm, 1, MPI_REAL_TYPE, MPI_SUM, MPI_COMM_WORLD));
        l2_norm = std::sqrt(l2_norm);
    }
}
__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 ); }
}
a = (real *) shmem_malloc(nx*(chunk_size+2)*sizeof(real));

a_new = (real *) shmem_malloc(nx*(chunk_size+2)*sizeof(real));

...

while (iter < iter_max && l2_norm > tol ) {
    ...
    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 );

    shmex_barrier_all_on_stream(compute_stream);

    //convergence check
    if (((iter % nccheck) == 0) {
        cudaMemcpyAsync( l2_norm_h, l2_norm_d, sizeof(real), cudaMemcpyDeviceToHost, compute_stream );
        cudaStreamSynchronize( compute_stream );
        MPI_Allreduce( l2_norm_h, &l2_norm, 1, MPI_REAL_TYPE, MPI_SUM, MPI_COMM_WORLD );
        l2_norm = std::sqrt( l2_norm );
    }
}

...
CUDA KERNEL - NVSHMEM FOR COMMS + SYNC

```c
__global__ void jacobi_uber_kernel( ... ) {
    grid_group g = this_grid();
    //comms only ...
    g.sync();
    if ( (iter % nccheck) == 0 ) {
        //reduction across shmem pes
        if (!tid_x && !tid_y) {
            shmem_barrier_all();
            shmem_float_sum_to_all (l2_norm + 1, l2_norm, 1, 0, 0, npes, NULL, NULL);
            l2_norm[1] = (float) __frsqrt_rn( (float)l2_norm[1] );
            l2_norm[0] = 0;
        }
    }
    g.sync();
}
```
HOST CODE - NVSHMEM FOR COMMS + SYNC

```c
a = (real *) shmem_malloc(nx*(chunk_size+2)*sizeof(real));
a_new = (real *) shmem_malloc(nx*(chunk_size+2)*sizeof(real));
...
void *args[] = {&a_new, &a, &l2_norm_d, &iy_start, &iy_end, &nx,
             &top, &iy_end_top, &bottom, &iy_start_bottom};
shmemx_collective_launch ( jacobi_kernel, dim_grid,
                           dim_block, 0, compute_stream);
...```

JACOBI SOLVER (ON X86)

DGX-2 (1 node) - 18432 x 18432, 1000 iterations

Parallel Efficiency vs #GPUs

MPI Overlap  NVSHMEM

Benchmark setup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, Reported Runtime is the minimum of 5 repetitions
JACOBI SOLVER (ON POWER9)

Summit (64 nodes) - 16384 x 16384, 1000 iterations

Benchmark setup: Summit with RHEL 4.14.0, GCC 4.8.5, CUDA 9.2.148 with 396.64 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3)
AGENDA

GPU Programming Models
Overview of NVSHMEM
Porting to NVSHMEM
Future Work
Conclusion and Future Work
IN-HEADER IMPLEMENTATION

Put/Get translates to LD/ST on NVLink/PCIe

Each API results in

• Function call overhead
• Remote address recalculation

Avoid through in-header implementation

Benchmark setup: DGX-1 with OS 4.14.0, GCC 5.4.0, CUDA 10.0.130 with 418.39 Driver
COLLECTIVES OPTIMIZATION

Now use direct all-to-all communication
Works well over NVlink in a DGX-1/2
Limits scalability inter-node
Improving implementations for device-side ops
Leverage NCCL goodness for CPU/on-stream ops

Benchmark setup: Summit with RHEL 4.14.0, GCC 4.8.5, CUDA 9.2.148 with 396.64 Driver
OTHERS

Interoperability

Tied to OpenMPI for MPI interoperability

To make MPI-based bootstrap an external module

Strided transfers - multi-dimensional decomposition
SUMMARY

Allows design and experimentation with GPU-initiated communication

Early access (EA2) available

Support for P2P and Infiniband connected GPUs
x86 and P9 support
Interoperable with MPI and OpenSHMEM

Please reach out to nvshm@nvidia.com
application use case, how GPU-initiated may help