Improving Application Scaling using OpenSHMEM for GPU-Initiated Communication

Sreeram Potluri
CUDA SW
GOAL

Informing the OpenSHMEM community

- Continued studies with GPU-Initiated Communication
- Single node and multi-node platforms
- Experiences with mini-apps
- Extensions to OpenSHMEM
- Being done as part of DoE Design Forward project
BROADER GOAL

Standard for data movement in a parallel context

- Move beyond host-managed communication
- Express communication within parallelism
- Taking advantage of the inherent GPU capabilities
- Use in applications and serve as a backend for higher-level models/languages
OUTLINE

- Motivation
- NVSHMEM and Experimental Platforms
- Experience with Mini-Apps
- OpenSHMEM Extensions
GPU Accelerator Redefined Parallel Computing in HPC

- **Tsubame**: World's First GPU Supercomputer
- **Top500**: 3 of Top 5 Supercomputers with Tesla GPUs
- **Oak Ridge TITAN**: World's Fastest Supercomputer
- **Summit & Sierra**: U.S. Announces Two Pre-Exascale Supercomputers Powered by GPU & NVLink
- **Breakthrough in HIV Research**: World's Largest Simulation of Virus Uncovers New Discovery
- **Deep Learning**: Univ. of Toronto Team Uses GPUs to Win Image-Net Competition, Google Acquires Team
- **NVIDIA Launches CUDA**
- **Univ. of Toronto Team Uses GPUs to Win Image-Net Competition, Google Acquires Team**
- **4**
Algorithm of choice is Hybrid Monte Carlo
  - Markov chain => no task parallelism
  - Parallelize over grid points

Presently running at 150 Tflops sustained on Titan
  - Volume=40^3x256 split over 1152 GPUs
  - At the limit of strong scaling using present algorithms (more on this later)

Physics goal is to run at Volume=128^3x512
  - 200x increase in compute power needed (scales super linearly with volume)
  - Need improved strong scaling and faster computers
Scalable multi-GPU solver
- CUDA streams to overlap communication and compute
- Separate interior halo-region update kernels
- Use MPI for node-to-node communication
CASE STUDY WITH QUADA

- Kernel launch blocked by serialized cudaMemcpy API overhead
- CUDA APIs cannot be called until MPI is complete
- Kernel launch latency
- Gaps in runtime are MPI

Device to Host
Interior update
Halo updates
Host to device
WHAT IS THE SOLUTION?

Possible solution with GPU-initiated communication
- Software overhead from calling CUDA API routines
  - Avoid API overhead by using GPU-driven communication
- Hardware overhead from launching a CUDA kernel
  - Use a single kernel for all updates (or fewer kernels)
- Inability to issue MPI / CUDA asynchronously with respect to each other
  - GPU kernel-level communication without having to synchronize with Host

Performance on the GPU
- Highly parallel architecture (lot of state to hide latencies)
- Scheduling
- Data coalescing
WHY SHMEM?

- Overheads of send/recv operations
  - Synchronization coupled with data movement
  - Request allocation and queuing
  - Message matching and unexpected messages
- Light-weight one-sided communication
  - Avoid synchronization and artificial serialization
  - Massively parallel and fine-grained communication
  - GPU-initiated SHMEM
OUTLINE

- Motivation
- NVSHMEM and Experimental Platforms
- Experience with Mini-Apps
- OpenSHMEM Extensions
NVSHMEM

- OpenSHMEM influenced - mostly identical but a subset
- Works with MPI/OpenSHMEM, in CUDA kernels or OpenACC regions

- initialization and cleanup (host)
  - nvstart_pes
- allocation and cleanup (host)
  - nvshmalloc
- nvshm_barrier_all (host)
- nvshm_get_ptr (host/GPU)
- put and get routines (GPU)
  - nvshm_(float/int)_(p/g)
  - nvshm_(float/int)_(put/get)
- loads/stores/atomics (GPU)
- nvshm_(quiet/fence) (GPU)
- nvshm_wait/wait_until (GPU)
CUDA IPC (Inter-process (P2P) Communication)

- Since CUDA 4.2
- Allows inter-process mapping of GPU buffers (similar to IPC in Linux)
  - GPUs should be in the same PCIe root complex
- Direct transfers between GPUs, bypassing CPU memory and from kernels
- Data can be moved using direct access (loads/store) or copy API (cudaMemcpy/cudaMemcpyAsync)

- Upto 8 GPUs - 2 per card - 4 cards under same PCIe root complex using raiser cards with PCIe switch
EXPERIMENTAL PLATFORMS - MULTI NODE

- ExpressFabric switches from Avago Technologies
- Base mode and Fabric mode
- Multi-host configuration
- Host-host communication capabilities
  - TWC - tunneled window connection (fine grained transfers)
    - Similar to non-transparent bridging
    - More scalable and secure

Source: http://www.avagotech.com/applications/datacenters/expressfabric/
OUTLINE

- Motivation
- NVSHMEM and Experimental Platforms
- Experience with Mini-Apps
- OpenSHMEM Extensions
2DSTENCIL

\[ u[i][j] = u[i][j] \]
\[ + (v[i+1][j] + v[i-1][j] + v[i][j+1] + v[i][j+1])/x \]
## CHANGE IN THE MODEL

<table>
<thead>
<tr>
<th>Traditional</th>
<th>Envisioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop {</td>
<td></td>
</tr>
<tr>
<td>Interior Compute (kernel launch)</td>
<td>Compute, Exchange and Synchronize (single kernel launch)</td>
</tr>
<tr>
<td>Pack Boundaries (kernel launch)</td>
<td>- Support SHMEM communication and synchronization primitives from inside GPU kernel</td>
</tr>
<tr>
<td>Stream Synchronize</td>
<td></td>
</tr>
<tr>
<td>Exchange (MPI/OpenSHMEM)</td>
<td></td>
</tr>
<tr>
<td>Unpack Boundaries (kernel launch)</td>
<td></td>
</tr>
<tr>
<td>Boundary Compute (kernel launch)</td>
<td></td>
</tr>
<tr>
<td>Stream/Device Synchronize</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

- Kernel launch overheads
- CPU based blocking synchronization
VISUAL PROFILE - TRADITIONAL

(Time marked for one step, Domain size/GPU - 1024, Boundary - 16, Ghost Width - 1)
VISUAL PROFILE - TRADITIONAL

(Time marked for one step, Domain size/GPU - 128, Boundary - 16, Ghost Width - 1)
VISUAL PROFILE - PERSISTENT KERNEL

(Time marked for complete run - 30 steps)
(Domain size/GPU- 128, Boundary - 16, Ghost Width - 1)
PERFORMANCE

### Time per Step (usec)

**Domain Size/GPU**

<table>
<thead>
<tr>
<th>Domain Size</th>
<th>Traditional</th>
<th>Persistent Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>195.33</td>
<td>13.88</td>
</tr>
<tr>
<td>128</td>
<td>193.7</td>
<td>21.32</td>
</tr>
<tr>
<td>256</td>
<td>193.18</td>
<td>39.77</td>
</tr>
<tr>
<td>512</td>
<td>220.28</td>
<td>132.61</td>
</tr>
<tr>
<td>1024</td>
<td>375.8</td>
<td>389.65</td>
</tr>
<tr>
<td>2048</td>
<td>1319.74</td>
<td>1312.59</td>
</tr>
<tr>
<td>4096</td>
<td>5299.23</td>
<td>4776.31</td>
</tr>
<tr>
<td>8192</td>
<td>21480.32</td>
<td>18394.88</td>
</tr>
</tbody>
</table>

**Time per Step (usec)**

(Ghost Width - 1; Boundary - 16)
(Threadsperblock - 512; blocks -15)
(4 Processes - 1 Process/GPU)

### GPU Count

<table>
<thead>
<tr>
<th>GPU Count</th>
<th>Traditional</th>
<th>Persistent Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>375</td>
<td>389</td>
</tr>
<tr>
<td>16</td>
<td>226</td>
<td>132</td>
</tr>
<tr>
<td>64</td>
<td>196</td>
<td>39</td>
</tr>
<tr>
<td>256</td>
<td>194</td>
<td>21</td>
</tr>
<tr>
<td>1K</td>
<td>192</td>
<td>13</td>
</tr>
<tr>
<td>4K</td>
<td>202</td>
<td>13</td>
</tr>
<tr>
<td>16K</td>
<td>193</td>
<td>12</td>
</tr>
<tr>
<td>64K</td>
<td>194</td>
<td>13</td>
</tr>
</tbody>
</table>

**Time per Step (usec)**

(Domain size - 2048; Ghost Width - 1; Boundary - 2)
(Extrapolation by reducing problem size per GPU, assuming constant exchange and synchronization time)

4 K40m GPUs connected on a Xeon E5-2690 socket using PLX switches
COMMUNICATION API PROFILE

Traditional MPI

Injection profile at process 0, for 10 iterations

NVSHMEM
MULTI-GPU TRANSPOSE OPERATION

Common algorithm in signal & image processing.
Similar to multi-node FFT algorithms
MULTI-GPU TRANSPOSE OPERATION

MPI Version
- Local transpose
- Data transfer using MPI
- Copy data to target (cudaMemcpy2D)

SHMEM Version
- Local transpose
- Write to destination
Performance

<table>
<thead>
<tr>
<th>Matrix Dim</th>
<th>Traditional</th>
<th>NVSHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>384</td>
<td>5.56</td>
<td>16.8</td>
</tr>
<tr>
<td>768</td>
<td>9.99</td>
<td>19.1</td>
</tr>
<tr>
<td>1536</td>
<td>13.6</td>
<td>20</td>
</tr>
<tr>
<td>3072</td>
<td>15.4</td>
<td>20</td>
</tr>
<tr>
<td>6144</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>12288</td>
<td>16</td>
<td>20</td>
</tr>
</tbody>
</table>

Bi-directional Bandwidth (GB/sec) (3 Processes - 1 Process/GPU)

- Reduced code complexity significantly
  - Complex pipelining in wrapper to Simple direct access
  - LOC for transpose function from 280 to 100

4 Tesla K40m - CUDA 6.5
CoMD

Proxy for Classical Moelcular Dynamics codes

- ExMatEx Co-Design Center

Short inter-atomic potentials

- each atom interacts with other atoms in its cell or in the twenty-six immediately neighboring cells

Force exchange

- MPI version - pack/unpack
  + send/recv
- NVSHMEM: Communication merged into compute kernels

Atoms exchange

- Also ported to use NVSHMEM

Source: [http://www.exmatex.org/comd.html](http://www.exmatex.org/comd.html)
PERFORMANCE

SHMEM vs. MPI Speedup

2 Tesla K40m - CUDA 6.5
CAFFE

Deep learning framework

Two phases

Train: (tagged) input data -> (apply operations) -> (tagged) signatures

Test: input data -> (apply operations) -> signature -> compare with tagged signatures to find relevant tags (by score)

Each batch is split over GPUs

Data Set N GBs
Replace Tree-based by Pipelined (not at scale)

- **CPU-Initiated**
  - Tree-based Bcast/Reduce
  - Few synchronizations

- **GPU-Initiated**
  - Pipelined Bcast/Reduce
  - Allow frequent synchronizations
SPEEDUP

Tree vs Pipeline

- Comm
- Compute

GPU count

Time (s)

0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

GPU count

Iterations / s

1
2
3
4
5
6
7
8
9
10

hsw231@PSG Cluster, 8x K80
Clocks : 2505,875
OUTLINE

- Motivation
- NVSHMEM and Experimental Platforms
- Experience with Mini-Apps
- OpenSHMEM Extensions
OPENSHMEM EXTENSIONS

- Support for thread safety
- Memory consistency model
- Isolated traffic flows
- Threaded collectives
- Memory locality
OPENSHMEM EXTENSIONS

- Support for thread safety
- Memory consistency model
  - Too strong ordering guarantees
  - Too weak ordering requirements
- Isolated traffic flows
- Threaded collectives
- Memory Locality
MP USING OPENSHMEM

All Put, AMOs, memory store operations to symmetric data objects are guaranteed to be completed and visible to all PEs

Message Passing Idiom

PE 0

shmem_int_p (&x, 1, 1)

shmem_int_p (&y, 1, 1)

PE 1

do {
    r1 = *((volatile int *) &y)
} while(r1 != 1)

r2 = x

r1=1, r2=1

Guaranteed in OpenSHMEM!!

shmem_fence would provide similar guarantees
**SHMEM_QUIET INSUFFICIENT**

- **shmем_quiet is not sufficient**
  - On weakly ordered machines like NVIDIA GPU, IBM Power, ARM
  - On Power: lwsync between L1 and L2 at PE1

Message Passing Idiom

```c
PE 0

shmем_int_p (&x, 1, 1)

shmем_quiet

shmем_int_p (&y, 1, 1)

shmем_quiet

PE 1

do {
    r1 = *((volatile int *) &y)
} while(r1 != 1)

lwsync

r2 = x

r1=1, r2=1
```
FIRST STEP SOLUTION

Address this in the OpenSHMEM specification explicitly
- Clarify what shmem_quiet guarantee
- Advise developers to use shmem_wait/shmem_wait_until

Message Passing Idiom

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>shmem_int_p (&amp;x, 1, 1)</td>
<td>shmem_wait_until(&amp;y, EQ, 1)</td>
</tr>
<tr>
<td>shmem_quiet</td>
<td></td>
</tr>
<tr>
<td>shmem_int_p (&amp;y, 1, 1)</td>
<td>r2=x</td>
</tr>
<tr>
<td>shmem_quiet</td>
<td></td>
</tr>
<tr>
<td>r2=1</td>
<td></td>
</tr>
</tbody>
</table>
shmem_ptr MEMORY

- shmem_ptr exposes remote memory for load/store
- Same ordering guarantees as put/get
  - Accesses to same address are not ordered
- Can cause unexpected behavior
  - Evident in presence of compiler optimizations
int *p = shmem_ptr(P, pe);

// a valid compiler optimization
*p = RAX; // Spill temporary value.
RAX = *p; // Restore temporary value.
*p = 1;

This works fine on regular memory (same address ordering)
May not be so on memory exposed by nvshmem_ptr
  Valid OpenSHMEM implementation
  Cannot be used as regular memory
Memory exposed by shmem_ptr cannot be used as real memory
POSSIBLE SOLUTIONS

- Not define behavior of memory exposed through shmem_ptr
- Provide same address ordering guarantee for memory exposed for loads/stores
  - all shared memory platforms today provide this
  - if there is a hypothetical platform that does not guarantee this, shmem_ptr can return NULL
OTHER ORDERING EXTENSIONS

- Identify the ordering variable/operation
  - Not done with shmem_fence/shmem_quiet today
  - Can allow optimizations
    - Ordering flag on network operation
    - Store release instruction on ARM

- Global ordering (operations to multiple PEs)
  - Only way is quiet - complete all operations before proceeding
  - Should not force an implementation

- SC-DRF - Sequential Consistency for Data Race Free
  - default model in Java and C++11
  - relaxed memory model
  - formal proof of support on Power, ARM
CONCLUDING REMARKS

- Strong scaling applications
- GPU-Initiated communication
  - Communication within parallelism
  - Taking advantage of GPU architecture
  - Improving programmability
- OpenSHMEM as the communication model
- On-going application studies
- On-going discussions on OpenSHMEM extensions