Extending OpenSHMEM for GPU Computing

Sreeram Potluri
Devendar Bureddy
Hao Wang
Hari Subramoni
Dhabaleswar K. Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University
Outline

• Motivation

• Problem Statement

• OpenSHMEM Extensions for GPU Computing

• Designing a High-performance OpenSHMEM Runtime

• Performance Evaluation

• Conclusion and Future Work
Accelerator Era

- Accelerators are becoming common in high-end system architectures

- Increasing number of workloads are being ported to take advantage of GPUs

- As they scale to large GPU clusters with high compute density – higher the synchronization and communication overheads – higher the penalty

- Critical to minimize these overheads to achieve maximum performance

Top 100 – Nov 2012
(20% use Accelerators)

70% use GPUs
Partitioned Global Address Space (PGAS) Models

- PGAS models, an attractive alternative to traditional message passing
  - Simple shared memory abstractions
  - Lightweight one-sided communication
  - Flexible synchronization
  - Lower synchronization and communication overheads – fit the requirements for GPU computing?

- OpenSHMEM, a easy-to-use library based PGAS model that is gaining attention

- An effort to unify and standardize various proprietary SHMEM-like implementations

- Benefits from long standing SHMEM implementations and user base
The OpenSHMEM Memory Model

- Defines symmetric data objects that are globally addressable
  - Allocated using a collective `shmalloc` routine
  - Same type, size and offset address at all processes/processing elements (PEs)
  - Address of a remote object can be calculated based on info of local object

```c
int main (int c, char *v[]) {
    int *b;
    start_pes();
    b = (int *) shmalloc (sizeof(int));
    shmem_int_get (b, b, 1, 1);
    // (dst, src, count, pe)
}
```
Limitations of OpenSHMEM for GPU Computing

• OpenSHMEM memory model does not support disjoint memory address spaces - case with GPU clusters

Existing OpenSHMEM Model with CUDA

**PE 0**

```
host_buf = shmalloc (…)
cudaMemcpy (host_buf, dev_buf, . . .)
shmem_putmem (host_buf, host_buf, size, pe)
shmem_barrier (…)
```

**PE 1**

```
host_buf = shmalloc (…)
shmem_barrier ( . . .)
cudaMemcpy (dev_buf, host_buf, size, . . .)
```

• Copies severely limit the performance
• Synchronization negates the benefits of one-sided communication
Can we do better?

Proposed OpenSHMEM Model

- High performance – Preserve one-sided semantics
Problem Statement

Can we extend the OpenSHMEM memory model to allow direct communication from GPU device memory?

Can we design an OpenSHMEM runtime system to achieve the maximum performance for different GPU configurations?

Can the performance benefits offered by the OpenSHMEM runtime result in improvements in performance of applications?

Can the extensions be interoperable with both CUDA and OpenCL for wider acceptance in the GPU computing community?
Proposed Design Framework

HPC Applications & Kernels

OpenSHMEM

Extensions
(performance, interoperability)

CUDA
OpenCL

High Performance OpenSHMEM Runtime

Intra-Node

Intra-IOH

Inter-IOH

Inter-Node

Modern HPC System Architecture

Multi-core Nodes

InfiniBand Network

GPUs
Outline

• Motivation

• Problem Statement

• OpenSHMEM Extensions for GPU Computing

• Designing a High-performance OpenSHMEM Runtime

• Performance Evaluation

• Conclusion and Future Work
Proposed Design Framework
Evaluating Existing Alternatives

- Heap Selection (based on UPC extensions by Zheng et. al. and Luo et. al.)
  - Extensions to select memory domain before allocation
  - Shmalloc can return CPU and GPU buffers

**PE 0**
<set memory domain to GPU>
dev_buf = shmalloc (size)

shmem_putmem (dev_buf, dev_buf, size, pe)

---

**PE 1**
<set memory domain to GPU>
dev_buf = shmalloc (size)

- We saw limitations as we went into the details
Interoperability with CUDA and OpenCL

• Usability of `shmalloc`ed buffers in CUDA/OpenCL calls
  – OpenCL operates on data objects - require an address to object conversion

• Device buffer detection in OpenSHMEM communication calls
  – Not an issue with CUDA with Unified Virtual Addressing (UVA)
  – OpenCL standard does not offer a feature equivalent to UVA - user has to indicate the type of buffer with each call or OpenSHMEM runtime needs a way to differentiate

• Context Management
  – CUDA provides separate API calls to set context, user can select context before making OpenSHMEM calls
  – Context is required as a parameter in several calls, need a way for user and runtime to exchange context information

• These complications arise as OpenSHMEM handles allocation of GPU device buffers
Proposed Extension: shmmap

- Let users manage device buffer allocation
- Allow buffers to be mapped onto a symmetric address space for communication

```
void *shmmap (void *obj, size_t size, int obj type);
void shmunmap (void *ptr);
```

- **obj**
  - a pointer to OpenCL memory object
  - a device buffer pointer with CUDA
- **size**
- **type**
  - `OSHM_MEMTYPE_CUDA`
  - `OSHM_MEMTYPE_OPENCL`

- The address can be used in OpenSHMEM communication calls
- Note that the map address space is only virtual addresses
Proposed Extension and Interoperability

- Shmmap follows semantics similar to shmalloc routine – collective and symmetric
- Symmetric map allows for translation from local map address to remote map address
- The map address is then translated to the memory object or device address
- Buffer usability in CUDA/OpenCL calls
  - User has complete control of the allocated buffers or buffer objects
- Device buffer detection
  - A simple check owing to reserved virtual address space for symmetric mapping
- Context management
  - Users have complete control of context information
  - OpenSHMEM runtime can get OpenCL context information using clGetMemObjectInfo on the memory object
Outline

• Motivation

• Problem Statement

• OpenSHMEM Extensions for GPU Computing

• Designing a High-performance OpenSHMEM Runtime

• Performance Evaluation

• Conclusion and Future Work
Design Framework

HPC Applications & Kernels

OpenSHMEM

Extensions (performance, interoperability)

CUDA
OpenCL

High Performance OpenSHMEM Runtime

Intra-Node
Intra-IOH Inter-IOH

Inter-Node

Modern HPC System Architecture

Multi-core Nodes InfiniBand Network GPUs
Intra-IOH Communication with CUDA

- CUDA offers Inter-Process Communication (IPC), a host-bypass for GPU-GPU transfers
- One process can map another process’s device buffer into its address space
- Single copy to move data between processes
- Buffers are mapped during shmmap, data movement fits well with one-sided semantics in OpenSHMEM
- However, this works only when GPUs are on the same IOH or Socket
- We work around this limitations by using a “shadow context”
Inter-IOH Communication with CUDA: Shadow Context

• CUDA supports P2P transfers between GPUs on different IOHs within one process (staged through the host by the driver but still single copy call)

• Each process creates a “shadow” context on remote process’s GPU - during init

• In shmmmap call – process switches to shadow context, maps remote process’s buffer using CUDA IPC and switches back to original context

• Uses P2P transfers for GPU-GPU communication, one-sided
Design Framework

HPC Applications & Kernels

OpenSHMEM

Extensions
(performance, interoperability)

CUDA
OpenCL

High Performance OpenSHMEM Runtime

Intra-Node

Intra-IOH  Inter-IOH

Inter-Node

Modern HPC System Architecture

Multi-core Nodes  InfiniBand Network  GPUs

IPDPS 2013
Inter-Node Communication with CUDA

- Pipelined data transfers through host memory - overlap between CUDA copies and IB transfers
- Used light-weight CUDA events for synchronization instead of streams
- Service-thread offered by OpenSHMEM reference implementation for asynchronous and one-sided progress
- We follow similar designs with OpenCL and more designs discussed in the paper
Outline

• Motivation

• Problem Statement

• OpenSHMEM Extensions for GPU Computing

• Designing a High-performance OpenSHMEM Runtime

• Performance Evaluation

• Conclusion and Future Work
Experimental Setup

• Micro-benchmark level evaluation
  – A westmereep cluster, each node has
    • Each node has two Intel Xeon E5645 quad-core CPUs, 12GB RAM
    • Mellanox MT26428 QDR HCA
    • Two NVIDIA C2075 GPUs with 5GB Memory
    • Red Hat Linux 5.4, OFED 1.5.1 and CUDA 4.1

• Application Kernel level evaluation
  – XSEDE Keeneland-KIDS cluster
    • Two Intel Xeon X5560 six-core CPUs, 32GB RAM
    • Mellanox MT4099 IB FDR HCA
    • Three NVIDIA Tesla M2090 GPUs

• OpenSHMEM Reference Implementation v1.0b

• OSU Micro-Benchmarks & SHOC Benchmark Suite
Intranode Communication using CUDA

- CUDA IPC significantly improves intra-IOH GPU-GPU communication – 90% improvement for 4Byte and 72% for 4MByte messages
- “shadow context” and P2P copies provides single copy for inter-IOH transfers - 40% improvement for 4Byte and 45% improvement for 4MByte messages
Internode Communication using CUDA

shmem_getmem

- Small messages benefit from selective CUDA registration – 17% for 4Byte messages
- Large messages benefit from pipelined overlap – 42% for 4MByte messages
- Service thread enables one-sided communication
Internode Communication using OpenCL

shmem_putmem

- Naïve
- Enhanced

- Similar latency and benefits as with CUDA – 42% for 4MByte messages
Stencil2D Kernel

- Modified SHOC Stencil2D kernel to use OpenSHMEM for cluster level parallelism
- The enhanced version shows **65% improvement on 192 GPUs** with 4Kx4K problem size/GPU
- Using OpenSHMEM for GPU-GPU communication allows runtime to optimize non-contiguous transfers
BFS Kernel

- Extended SHOC BFS kernel to run on a GPU cluster using a level-synchronized algorithm and OpenSHMEM
- The enhanced version shows up to 12% improvement on 96 GPUs, a consistent improvement in performance as we scale from 24 to 96 GPUs.
Conclusion and Future Work

• Usability of OpenSHMEM on GPU clusters is severely limited

• Proposed extensions to the OpenSHMEM memory model to alleviate this, interoperable with both CUDA and OpenCL

• Presented a high-performance OpenSHMEM runtime including novel designs like shadow context

• Upto 90% improvement in inter-node GPU-GPU transfers and upto 42% improvement for inter-node GPU-GPU communication

• Demonstrated benefits using Stencil2D and BFS kernels
Conclusion and Future Work

- Current focus on runtime level optimizations using CUDA features like GPUDirect RDMA
- To re-design wider range of applications using the extended OpenSHMEM model
Thank You!

{potluri, bureddy, wanghai, subramon, panda}@cse.ohio-state.edu,

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page
http://mvapich.cse.ohio-state.edu/