Enabling Efficient Use of UPC and OpenSHMEM PGAS models on GPU Clusters

Presentation at GTC 2014

by

Dhabaleswar K. (DK) Panda
The Ohio State University
E-mail: panda@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~panda
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
Parallel Programming Models Overview

- Programming models provide abstract machine models
- Models can be mapped on different types of systems
  - e.g. Distributed Shared Memory (DSM), MPI within a node, etc.
- Each model has strengths and drawbacks - suite different problems or applications
Outline

• Overview of PGAS models (UPC and OpenSHMEM)
• Limitations in PGAS models for GPU computing
• Proposed Designs and Alternatives
• Performance Evaluation
• Exploiting GPUDirect RDMA
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

• Different approaches to PGAS
  - Libraries
    • OpenSHMEM
    • Global Arrays
    • Chapel
  - Languages
    • Unified Parallel C (UPC)
    • Co-Array Fortran (CAF)
    • X10
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));
    shmemp_int_get (b, b, 1, 1); // (dst, src, count, pe)
}
```

```c
int main (int c, char *v[]) {
    int *b;
    start_pes();
    b = (int *) shmalloc (sizeof(int));
}
```
Global Shared Space: can be accessed by all the threads

Private Space: holds all the normal variables; can only be accessed by the local thread

Example:

```c
shared int A1[THREADS];  //shared variable
int main() {
    int y;  //private variable
    A1[0] = 0;  //local access
    A1[1] = 1;  //remote access
}
```
MPI+PGAS for Exascale Architectures and Applications

• Gaining attention in efforts towards Exascale computing

• Hierarchical architectures with multiple address spaces

• (MPI + PGAS) Model
  – MPI across address spaces
  – PGAS within an address space

• MPI is good at moving data between address spaces

• Within an address space, MPI can interoperate with other shared memory programming models

• Re-writing complete applications can be a huge effort

• Port critical kernels to the desired model instead
Hybrid (MPI+PGAS) Programming

• Application sub-kernels can be re-written in MPI/PGAS based on communication characteristics

• Benefits:
  – Best of Distributed Computing Model
  – Best of Shared Memory Computing Model

• Exascale Roadmap*:
  – “Hybrid Programming is a practical way to program exascale systems”

MVAPICH2-X for Hybrid MPI + PGAS Applications

- Unified communication runtime for MPI, UPC, OpenSHMEM available with MVAPICH2-X 1.9 (2012) onwards! : http://mvapich.cse.ohio-state.edu

- Feature Highlights
  - Supports MPI(+OpenMP), OpenSHMEM, UPC, MPI(+OpenMP) + OpenSHMEM, MPI(+OpenMP) + UPC
  - MPI-3 compliant, OpenSHMEM v1.0 standard compliant, UPC v1.2 standard compliant
  - Scalable Inter-node and intra-node communication – point-to-point and collectives

- Effort underway for support on NVIDIA GPU clusters
Outline

• Overview of PGAS models (UPC and OpenSHMEM)
• Limitations in PGAS models for GPU computing
• Proposed Designs and Alternatives
• Performance Evaluation
• Exploiting GPUDirect RDMA
Limitations of PGAS models for GPU Computing

- PGAS memory models do not support disjoint memory address spaces - case with GPU clusters

- OpenSHMEM case

  Existing OpenSHMEM Model with CUDA

  **PE 0**
  ```
  host_buf = shmalloc (...)
  cudaMemcpy (host_buf, dev_buf, ...)
  shmputmem (host_buf, host_buf, size, pe)
  shmbarrier (...)
  ```

  **PE 1**
  ```
  host_buf = shmalloc (...)
  shmbarrier (...)
  cudaMemcpy (dev_buf, host_buf, size, ...)
  ```

- Copies severely limit the performance
- Synchronization negates the benefits of one-sided communication
- Similar limitations in UPC
Outline

• Overview of PGAS models (UPC and OpenSHMEM)
• Limitations in PGAS models for GPU computing
• Proposed Designs and Alternatives
• Performance Evaluation
• Exploiting GPUDirect RDMA
Global Address Space with Host and Device Memory

- **Extended APIs:**
  - `heap_on_device/heap_on_host`
  - A way to indicate location on heap
- Can be similar for dynamically allocated memory in UPC

```c
heap_on_device();
/*allocated on device*/
dev_buf = shmalloc(sizeof(int));

heap_on_host();
/*allocated on host*/
host_buf = shmalloc(sizeof(int));
```
CUDA-aware OpenSHMEM and UPC runtimes

• After device memory becomes part of the global shared space:
  – Accessible through standard UPC/OpenSHMEM communication APIs
  – Data movement transparently handled by the runtime
  – Preserves one-sided semantics at the application level

• Efficient designs to handle communication
  – Inter-node transfers use host-staged transfers with pipelining
  – Intra-node transfers use CUDA IPC

• Service-thread for asynchronous and one-sided progress in

• Goal: Enabling High performance one-sided communications with GPU devices
Outline

• Overview of PGAS models (UPC and OpenSHMEM)
• Limitations in PGAS models for GPU computing
• Proposed Designs and Alternatives
• Performance Evaluation
• Exploiting GPU Direct RDMA
Shmem_putmem Inter-node Communication

- Small messages benefit from selective CUDA registration – 22% for 4Byte messages
- Large messages benefit from pipelined overlap – 28% for 4MByte messages
- Service thread enables one-sided communication

Shmem_putmem Intra-node Communication

- Using IPC for intra-node communication
- Small messages – 73% improvement for 4Byte messages
- Large messages – 85% for 4MByte messages

Based on MVAPICH2X-2.0b + Extensions
Intel WestmereEP node with 8 cores
2 NVIDIA Tesla K20c GPUs, Mellanox QDR HCA
CUDA 6.0RC1
Application Kernel Evaluation: Stencil2D

- 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

![Graph showing performance comparison between naive and enhanced versions of Stencil2D kernel. The graph illustrates the total execution time (msec) for different problem sizes (512x512, 1Kx1K, 2Kx2K, 4Kx4K, 8Kx8K) and number of GPUs (48, 96, 192). The enhanced version shows a significant reduction in execution time, indicated by a 65% improvement.](image-url)
Application Kernel Evaluation: BFS

- 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.
Outline

• Overview of PGAS models (UPC and OpenSHMEM)
• Limitations in PGAS models for GPU computing
• Proposed Designs and Alternatives
• Performance Evaluation
• Exploiting GPUDirect RDMA
Exploiting GPUDirect RDMA

- In OpenSHMEM (Preliminary results)
  - GDR for small message sizes
  - Host-staging for large message to avoid PCIe bottlenecks
  - Hybrid design brings best of both
  - 3.3\text{us} latency for 4 bytes

**GPU features will be available in future releases of MVAPICH2-X!!**

Based on MVAPICH2X-2.0b + Extensions

Intel Sandy Bridge (E5-2670) node with 16 cores

NVIDIA Tesla K40c GPU, Mellanox Connect-IB Dual-FDR HCA

CUDA 5.5, Mellanox OFED 2.1 with GPU-Direct-RDMA Plugin
Hangout with the Speaker

Come know and discuss how we make it easier to use MPI and PGAS models on NVIDIA GPU clusters

S4951 – Hangout: GTC Speakers
Tuesday – 03/25
13:00 – 14:00 (Now)
Concourse Pod B
Talk on advances in CUDA-aware MPI and Hybrid HPL

1) S4517 - Latest Advances in MVAPICH2 MPI Library for NVIDIA GPU Clusters with InfiniBand

Tuesday, 03/25 (Today)
15:00 – 15:25
Room LL21A

2) S4535 - Accelerating HPL on Heterogeneous Clusters with NVIDIA GPUs

Tuesday, 03/25 (today)
17:00 – 17:25
Room LL21A
Thank You!

panda@cse.ohio-state.edu

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

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