Partitioned Global Address Space (PGAS) Programming Models

Jahanzeb M. Hashmi
The Ohio State University
E-mail: Hashmi.29@osu.edu
http://web.cse.ohio-state.edu/~hashmi.29/
Overview of PGAS and Hybrid MPI+PGAS Models in the Context of OpenSHMEM
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.
- Additionally, OpenMP can be used to parallelize computation within the node
- Each model has strengths and drawbacks - suite different problems or applications
MPI Overview and History

- Message Passing Library standardized by MPI Forum
  - C and Fortran
- Goal: portable, efficient and flexible standard for writing parallel applications
- Not IEEE or ISO standard, but widely considered “industry standard” for HPC application
- Evolution of MPI
  - MPI-1: 1994
  - MPI-2: 1996
  - Next plan is for MPI 4.0
Partitioned Global Address Space (PGAS) Models

• Key features
  - Simple shared memory abstractions
  - Light weight one-sided communication
  - Easier to express irregular communication

• Different approaches to PGAS
  - Languages
    • Unified Parallel C (UPC)
    • Co-Array Fortran (CAF)
    • X10
    • Chapel
  - Libraries
    • OpenSHMEM
    • UPC++
    • Global Arrays
OpenSHMEM

- SHMEM implementations – Cray SHMEM, SGI SHMEM, Quadrics SHMEM, HP SHMEM, GSHMEM
- Subtle differences in API, across versions – example:

<table>
<thead>
<tr>
<th>SGI SHMEM</th>
<th>Quadrics SHMEM</th>
<th>Cray SHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>start_pes(0)</td>
<td>shmem_init</td>
</tr>
<tr>
<td>Process ID</td>
<td>_my_pe</td>
<td>my_pe</td>
</tr>
</tbody>
</table>

- Made application codes non-portable
- OpenSHMEM is an effort to address this:

  “A new, open specification to consolidate the various extant SHMEM versions into a widely accepted standard.” – OpenSHMEM Specification v1.0

  by University of Houston and Oak Ridge National Lab

  SGI SHMEM is the baseline
The OpenSHMEM Memory Model

• Symmetric data objects
  – Global Variables
  – Allocated using collective *shmalloc, shmemalign, shrealloc* routine

Symmetric Objects

• Globally addressable – objects have same
  – Type
  – Size
  – Same virtual address or offset at all PEs
  – Address of a remote object can be calculated based on info of local object

Virtual Address Space
Data Movement: Contiguous

- **Block Put and Get – Contiguous**
  - void shmem_TYPE_put (TYPE* target, const TYPE*source, size_t nelems, int pe)
    - TYPE can be char, short, int, long, float, double, longlong, longdouble
  - shmem_putSIZE – elements of SIZE: 32/64/128
  - shmem_putmem - bytes
  - Similar get operations

```c
int *b;
b = (int *) shmalloc (10*sizeof(int));
if (_my_pe() == 0) {
    shmem_int_put (b, b, 5, 1);
}
```
Collective Synchronization

- Barrier ensures completion of all previous operations
- Global Barrier
  - void shmem_barrier_all()
  - Does not return until called by all PEs
- Group Barrier
  - Involves only an "ACTIVE SET" of PEs
  - Does not return until called by all PEs in the "ACTIVE SET"
  - void shmem_barrier(int PE_start, /* first PE in the set */
                        int logPE_stride, /* distance between two PEs*/
                        int PE_size, /*size of the set*/
                        long *pSync /*symmetric work array*/);
  - pSync allows for overlapping collective communication
One-sided Synchronization

• Fence
  - void shmem_fence (void)
  - Enforces ordering on Put operations issued by a PE to each destination PE
  - Does not ensure ordering between Put operations to multiple PEs

• Quiet
  - void shmem_quiet (void)
  - Ensures remote completion of Put operations to all PEs

• Other point-to-point synchronization
  - shmem_wait and shmem_wait_until – poll on a local variable
Collective Operations and Atomics

- Broadcast – one-to-all
- Collect – allgather
- Reduction – allreduce (AND, OR, XOR; MAX, MIN; SUM, PRODUCT)
- Work on an active set – start, stride, count

- Unconditional - Swap Operation
  - long shmem_swap (long *target, long value, int pe)
  - TYPE shmem_TYPE_swap (TYPE *target, TYPE value, int pe)
  - TYPE can be int, long, longlong, float, double

- Conditional - Compare and Swap Operation
- Arithmetic – Fetch & Add, Fetch & Increment, Add, Increment
Remote Pointer Operations

- void *shmem_ptr (void *target, int pe)
  - Allows direct load/stores on remote memory
  - Useful when PEs are running on same node
  - Not supported in all implementations
  - Returns NULL if not accessible for loads/stores
A Sample code: Circular Shift

```c
#include <shmem.h>
int aaa, bbb;
int main (int argc, char *argv[]) {
    int target_pe;

    start_pes(0);
    target_pe = (_my_pe() + 1)% _num_pes();

    bbb = _my_pe() + 1
    shmem_barrier_all();

    shmem_int_get (&aaa, &bbb, 1, target_pe);
    shmem_barrier_all();
}
```
Architectures for Exascale Systems

Modern architectures have increasing number of cores per node, but have limited memory per core
- Memory bandwidth per core decreases
- Network bandwidth per core decreases
- Deeper memory hierarchy
- More parallelism within the node

Maturity of Runtimes and Application Requirements

- MPI has been the most popular model for a long time
  - Available on every major machine
  - Portability, performance and scaling
  - Most parallel HPC code is designed using MPI
  - Simplicity - structured and iterative communication patterns

- PGAS Models
  - Increasing interest in community
  - Simple shared memory abstractions and one-sided communication
  - Easier to express irregular communication

- Need for hybrid MPI + PGAS
  - Application can have kernels with different communication characteristics
  - Porting only part of the applications to reduce programming effort
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
Many possible ways to combine MPI

Focus on:
- Flat: One global address space
- Nested-multiple: Multiple global address spaces (UPC groups)

J. Dinan, P. Balaji, E. Lusk, P. Sadayappan and R. Thakur, Hybrid Parallel Programming with MPI and Unified Parallel C, ACM Computing Frontiers, 2010
Simple MPI + OpenSHMEM Hybrid Example

```c
int main(int c, char *argv[])
{
    int rank, size;

    /* SHMEM init */
    start_pes(0);

    /* fetch-and-add at root */
    shmem_int_fadd(&sum, rank, 0);

    /* MPI barrier */
    MPI_Barrier(MPI_COMM_WORLD);

    /* root broadcasts sum */
    MPI_Bcast(&sum, 1, MPI_INT, 0, MPI_COMM_WORLD);

    fprintf(stderr, "(%d): Sum: %d\n", rank, sum);

    shmem_barrier_all();
    return 0;
}
```

- OpenSHMEM atomic fetch-add
- MPI_Bcast for broadcasting result
Hybrid 2D Heat benchmark

Pure OpenSHMEM version

```java
while(true){
    <Gauss-Seidel Kernel>
    compute convergence locally
    sum_all =
    sumshmem_float_sum_to_all()
    Compute std. deviation
    shmem_broadcast(method to use)
}
```

- MPI Collectives have been optimized significantly
  - Performs better than OpenSHMEM collectives
- Improves performance of benchmark significantly

Hybrid MPI+OpenSHMEM version

```java
while(true){
    <Gauss-Seidel Kernel>
    compute convergence locally
    sum_all =
    MPI_Reduce()
    Compute std. deviation
    MPI_Bcast(method to use)
}
```
Exploiting and Evaluating OpenSHMEM on KNL Architecture

Jahanzeb M. Hashmi, Mingzhe Li, Hari Subramoni, and Dhabaleswar K. (DK) Panda

Fourth Workshop on OpenSHMEM and Related Technologies, Aug 2017
Intel Knights Landing Processor – Overview

- Multi-threaded cores
  - Up to 72 cores (model 7290)
- All cores divided into 36 Tiles
- Each tile contains two core
  - 2 VPU per core
  - 1MB shared L2 cache
- 512-bit wide vector registers
  - AVX-512 extensions
• On-package Multi Channel DRAM (MCDRAM)
  – 450 GB/s of theoretical bandwidth (4x of DDR)
  – Configurable in Flat, Cache, and Hybrid modes
Contributions

• Evaluate OpenSHMEM point-to-point, collective, and atomic operations on KNL architecture

• Evaluate OpenSHMEM application kernels
  – Discuss the impact of MCDRAM and vectorization
  – Comparison of KNL against Broadwell on core-by-core and node-by-node performance

• Discuss potential optimization for KNL architecture
Benchmarks and Configurations

- OpenSHMEM Microbenchmark Evaluation
  - Point-to-point (Put/Get), collectives (broadcast, reduce), and atomics
  - OSU Microbenchmark v5.3.2
- OpenSHMEM NAS Parallel Benchmark Kernel Kernels
  - MG, BT, EP, SP
- University of Houston OpenSHMEM test suite
  - Five application Kernels
    - 2D-Heat, Heat-Image, Matrix Multiplication, DAXPY, and ISx
- All experiments evaluate KNL 7250 against Broadwell
- Application evaluation on KNL also discuss MCDRAM and AVX-512 benefits
Experiment Setup

- Intel Xeon Phi KNL Cluster @ CSE, OSU (4 nodes)
  - KNL 7250 (1.4GHz) with 16GB MCDRAM and 96GB DDR
  - Mellanox EDR Connect-X HCAs (100 Gbps data rate)

- RI2 Cluster @ CSE, OSU (40 nodes)
  - Xeon E5-2680 v4 (2.40 GHz) with 128 GB DDR
  - Mellanox EDR Connect-X HCAs (100 Gbps data rate)

- Software stack
  - RHEL 6.3 with Mellanox OFED v2.2-1.0.0
  - MVAPICH2-X 2.2 with GCC v5.4.0
Broadwell shows about 3X better performance than KNL on large message

Muti-threaded memcpy routines on KNL could offset the degradation caused by the slower core on basic Put/Get operations

Shmem_putmem

Shmem_getmem
Microbenchmark Evaluations (Inter-node Put/Get)

- Inter-node small message latency is only 2X worse on KNL. While large message performance is almost similar on both KNL and Broadwell.
AVX-512 vectorized execution of BT kernel on KNL showed 30% improvement over default execution while EP kernel didn’t show any improvement.

Broadwell showed 20% improvement over optimized KNL on BT and 4X improvement over all KNL executions on EP kernel (random number generation).
Node-by-node Evaluation using Application Kernels

A single node of KNL is evaluated against a single node of Broadwell using all the available physical cores.

HeatImage and ISx kernels showed better performance than Intel Xeon.
Summary

Put/Get and Atomics Performance

Collectives Performance

Core-by-core Application Performance

Node-by-node Application Performance

- KNL (Default)
- KNL (AVX512)
- KNL (AVX512 + MCDRAM)
- Broadwell

(Closer to center is better)
Unifying UPC and MPI Runtimes: Experience with MVAPICH

Jithin Jose, Miao Luo, Sayantan Sur, and D. K. Panda

Fourth Conference on Partitioned Global Address Space Programming Model (PGAS '10), Oct. 2010
The Need for a Unified Runtime

- Deadlock when a message is sitting in one runtime, but application calls the other runtime
- Current prescription to avoid this is to barrier in one mode (either UPC or MPI) before entering the other
- Bad performance!!
Coercing UPC over MPI not Optimal

- MPI does not provide Active Messages
  - AMs critical to UPC compilation and performance
  - Simulating AMs over MPI leads to performance loss
  - Not going to be included in MPI-3
- MPI RMA model for non cache-coherent machines
  - Penalizes vast majority of cache coherent machines
  - MPI-3 considering a proposal to support both cache-coherent and non cache-coherent machines (will take time)
- MPI will not support instant teams
  - Communicators in MPI require group communication
- Path forward: unify runtimes, not programming models
Problem Statement

- Can we design a communication library for UPC?
  - Scalable on large InfiniBand clusters
  - Provides equal or better performance than existing runtime

- Can this library support both MPI and UPC?
  - Individually, both with great performance
  - Simultaneously, with great performance and less memory
Overall Approach

- Unified runtime provides APIs for MPI and GASNet
- **INCR** (Integrated Communication Runtime)
The INCR Interface

• Different AM APIs based on size for optimization
  – Send short AM without arguments
  – Short AM (no data payload)
  – Medium AM (bounce buffer using RDMA FP)
  – Large AM (RDMA Put, on-demand connections)

• GASNet Extended interface for efficient RMA
  – Inline put
  – Put (may be internally buffered)
  – Put bulk (send buffer will not be touched, no buffering)
  – Get (RDMA Read)
Unified Implementation

- All resources are shared between MPI and UPC
  - Connections, buffers, memory registrations
  - Schemes for establishing connections (fixed, on-demand)
  - RDMA for large AMs and for PUT, GET
Various Configurations for running UPC and MPI Applications

- **Pure MPI Applications**
  - MVAPICH - MPI Standard Interface
  - MVAPICH-Aptus Runtime
  - InfiniBand Network

- **GASNet-IBV**
  - Expansion of GASNet Interface and UPC Runtime
  - GASNet IBVerbs Runtime
  - InfiniBand Network

- **GASNet-MPI**
  - Expansion of GASNet Interface and UPC Runtime
  - GASNet-MPI Runtime
  - InfiniBand Network

- **GASNet-INCR**
  - Expansion of GASNet Interface and UPC Runtime
  - GASNet INCR Runtime
  - InfiniBand Network

- **Our Contribution**
  - MVAPICH INCR Implementation

---

Network Based Computing Laboratory

CSE 5194
Experiment Setup

- MVAPICH version 1.1 extended to support INCR
- Berkeley GASNet version 2.10.2 (--enable-pshm)
- Experimental Testbed
  - Type 1
    - Intel Nehalem (dual socket quad core Xeon 5500 2.4GHz)
    - ConnectX QDR InfiniBand
  - Type 2
    - Intel Clovertown (dual socket quad core Xeon 2.33GHz)
    - ConnectX DDR InfiniBand
  - Type 3
    - AMD Barcelona
    - Quad-socket quad-core Opteron 8530 processors
    - ConnectX DDR InfiniBand
Microbenchmarks: UPC-memput

- Cluster #1 used for these experiments
- GASNet-INCR performs identically with GASNet-IBV
- Comparatively GASNet-MPI performs much worse
- Mismatch of Active Message semantics
  - Message queue processing overheads
GASNet-INCR performs equal or better than GASNet-IBV
10% improvement for CG (B, 128)
23% improvement for MG (B, 128)
Cluster #3 used for these experiments
Hybrid FT

- Modified NAS FT UPC all-to-all pattern using MPI_Alltoall
- Truly hybrid program
- 34% improvement for FT (C, 128)
- Cluster #3 used for this experiment
Summary

- Integrated Communication Runtime (INCR): supports MPI and UPC simultaneously
- Promising: MPI communication not harmed and UPC communication not penalized
- No need for programmer to barrier between UPC and MPI modes, as is current practice
- Pure UPC NAS: 10% improvement CG (B, 128), 23% improvement MG (B, 128)
- MPI+UPC FT: 34% improvement for FT (C, 128)
Thank you!

hashmi.29@osu.edu

Network-Based Computing Laboratory

http://mvapich.cse.ohio-state.edu/