Oak Ridge OpenSHMEM Benchmark Suite

Thomas Naughton, Ferrol Aderholdt, Matt Baker, Swaroop Pophale, Manjunath Gorentla Venkata and Neena Imam

Oak Ridge National Laboratory

August 22, 2018

OpenSHMEM 2018 Workshop, Baltimore, MD, USA
Talk Summary

• Introduce a set of benchmarks to aid assessment of OpenSHMEM by users & implementors

• Outline
  – Overview of benchmarks in suite
  – Highlight multithreaded enhancements
  – Basic usage and results for illustration
OpenSHMEM Benchmark (OSB) Suite

• Collection of codes ported to use OpenSHMEM
  – Micro-benchmarks
  – Mini-applications / compute kernels

• Target users
  – System implementors
  – Application developers

• Example use cases
  – Assess effects of different implementation strategies
  – Assess performance of different library implementations
OSB Suite

“Mini” Applications
• Graph500 (search/graphs)
• SSCA1 (search/text)
• NPB (compute kernels)

Synthetic Benchmarks
• GUPS (memory)
• SHOMS (oshmem API)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Single Threaded</th>
<th>Multithreaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph500</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>SSCA1</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>NPB</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>GUPS</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>SHOMS</td>
<td>YES</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Experimental Environments

• “Turing”
  – 16-node Linux cluster
  – RHEL 7.4
  – Intel Xeon E5-2660 2.6Ghz
  – 64GB memory / node
  – Mellanox ConnectX-5

• “EOS”
  – 736-node Cray XC30
  – CrayOS 5.2.82
  – Intel Xeon E5-2670
  – 64GB memory / node
  – Aries network

• OpenSHMEMs
  – Cray-shmem 7.7.0
  – OpenSHMEM-X “devel” (w/ ucx)
  – SOS 1.4.1 (w/ libfabric-cray)
Graph500

• Benchmark to represent data intensive workloads
  – Breadth-First Search (BFS) on large undirected graphs
  – Fine-grained communication
  – Sparse spatial & temporal locality

• Input parameters
  – Problem (graph) size : scale_factor & edge_factor
    • Number of vertices = $2^{\text{scale}\_\text{factor}}$
    • Number of edges = $2 \times \text{edge}\_\text{factor}$
  – Memory required:
    • Example: $(2^{24} \times (2 \times 16)) \times 8$ (bytes) = 4MB
      
      \[ \text{scale}\_\text{factor} = 24 \quad \text{edge}\_\text{factor} = 16 \]
Graph500

• Roughly three phases
  1. Graph edges generated (Kronecker algorithm)
     • Parameters: scale_factor & edge_factor
  2. BFS - Randomly designate 64 vertices as “root” vertices, build tree from the “root” vertex,
  3. BFS is validated for correctness
     – Time measured for all three phases
• Metric
  – TEPS = Number of Traversed Edges Per Second
  – * Also mean_time for BFS steps
Graph500 OpenSHMEM

• OpenSHMEM version
  – Adapted from MPI version
  – Graph in symmetric heap & partitioned among PEs
  – Vertices & edges accessible to all PEs via OSHMEM
  – During BFS tracks vertex status (visited/discovered/...)
    • Use shmem_put & AMO instead of MPI_Accumulate for updating queues of vertices (e.g., discovered vertices)

• Multithreaded
  – OpenMP threads parallelize workload of BFS
  – Partition discovered vertices among threads
  – Thread executes BFS on vertices in its partition
  – OpenSHMEM context per thread to separate operations
Graph500 Usage & Results

To Execute:
```bash
cd mpi/
oshrun -np 32 ./graph500_shmem_one_sided 24 16
```

- Environment variables
  - SHORT_VALIDATION (8 BFS)
  - SKIP_VALIDATION

---

```plaintext
1 graph_generation: 16.875364 s
2 construction_time: 3.005170 s
3 Running BFS 0
4 Time for BFS 0 is 21.030321
5 Validating BFS 0
6 Validate time for BFS 0 is 49.637025
7 TEPS for BFS 0 is 1.27641e+07

16.875364
3.005170
21.030321
49.637025
1.27641e+07

mean_time: 21.2106
stddev_time: 0.750554
min_nedge: 268432547
first_quartile_nedge: 268432547
median_nedge: 268432547
third_quartile_nedge: 268432547
max_nedge: 268432547
mean_nedge: 268432547
stddev_nedge: 0
min_TEPS: 1.16249e+07
first_quartile_TEPS: 1.22967e+07
median_TEPS: 1.27692e+07
third_quartile_TEPS: 1.29745e+07
max_TEPS: 1.34908e+07
harmonic_mean_TEPS: 1.26556e+07
harmonic_stddev_TEPS: 56421.2
```
Graph500 Usage & Results

- Time for BFS scaling up #PEs (16-512) & scale_factor (20-25)
- Using OpenSHMEM-X on Turing
SSCA1

• SSCA1: Scalable Synthetic Compact Applications 1
  – Sequence alignment algorithm with gap scoring
  – Implemented as dynamic programming algorithm
  – Similarity matrix simulates DNA codon to protein encoding
  – Compare characters in text strings for matches (score)
  – Scoring sequence based on presence of a gap

• Input parameters
  – Problem size: SCALE environment variable (integer)

• Metric
  – Time to solution (elapsed time)
• Variants
  – Single threaded: OpenSHMEM or MPI-3 one sided
  – Multithreaded: OpenSHMEM specific

• Application workflow
  – Structure: Outer / Inner loop
  – Many small messages (puts & gets) in inner loop
    • Inner loop: 5 small gets (must finish on each inner loop)
    • Inner loop: 3 small puts (must finish before starting outer loop)

• OpenSHMEM Multithread
  – Outer loop solving a diagonal in matrix (not parallelizable)
  – Inner (parallel) loop solves each entry in the diagonal
  – Inner loop using OpenMP threads
SSCA1 Usage & Results

To Execute:

```bash
export SCALE=31
export OMP_NUM_THREADS=4           # If using threaded variant
oshrun –np 256 ./ssca1
```

```verbatim
Running with OpenMP, thread count: 8
Running with OpenMP, thread count: 8
Running with OpenMP, thread count: 8
Running with OpenMP, thread count: 8

... < snip > ...

verbatim
verifyAlignment 0, succeeded; score 55:
  26590 *IDENTICAL* tgaatagacgagacgagatgagcgcgtg
  29129 *IDENTICAL* tgaatagacgagacgagatgagcgcgtg

verbatim
verifyAlignment 1, succeeded; score 54:
  1039 *MISQRMATCHES* tgaatgataagccagaggatgagcgc
  23093 *MISQRMATCHES* tgaatgataagccagaggatgagcgc

verbatim
verifyAlignment 2, succeeded; score 53:
  13839 *STARTGAPMIDST—END* tgaagcagcgggaggacg gagaagctg
  13856 gagaacgcgtg
  37081 *STARTGAPMIDST—END* tgaagcagcgggaggacg —atgatacagcccttcgcggagacgactg
  37098

... < snip > ...
```
SSCA1 Usage & Results

# OpenSHMEM-X execution line (Turing SSCA1-threaded)
orterun \
  –hostfile hosts \
  –bind-to socket \
  –map-by ppr:2:node \
  –np 4 \
  –x OMP_NUM_THREADS=8 \
  –x SCALE=32 \
./ssca1

# Num PEs per node
# Num PEs (total)
# Num Threads per PE
# Problem size

![Graphs showing performance metrics for EOS and Turing SSCA1-threaded configurations.](image)
NPB

• NPB: NAS Parallel Benchmarks
  – Application kernels for common scientific algorithms
  – OpenSHMEM versions adapted from MPI variants
  – Single threaded OpenSHMEM versions

• Benchmarks / Mini-apps
  – BT: Block Tri-diagonal solver, CFD mini-app (Fortran)
  – SP: Scalar Penta-diagonal solver, CFD mini-app (Fortran)
  – MG: Multi-Grid, long/short distance communication, memory intensive (Fortran)
  – IS: Integer Sort – random memory access (C)

• Metric
  – MOPS – Millions of Operations Per Second
• **OpenSHMEM details**
  - Adapted from MPI variants
  - **IS: Integer Sort**
    • Bucket sort, each process sorts random set of keys in their range
    • Uses put/get to simulate MPI AlltoAll/AlltoAllv to communicate keys
  - **MG: Multi-Grid**
    • Due to 1-sided communication, only require synchronization (barrier_all) to ensure updates are visible at all PEs & ensures all PEs at same stage
  - **BT: Block Tri-diagonal solver**
    • Uses gets when solving block tridiagonal equations
  - **SP: Scalar Penta-diagonal**
    • In OSHMEM case SP has better comp/comm overlap b/c synchronize only when communicated data is used
To Execute:
oshrun –np <nprocs> ./bin/<benchmark-name>..<class>..<nprocs>
    # where
    #   <benchmark-name> is "is", "mg", "bt" or "sp"
    #   <nprocs> is the number of processes
    #   <class> is "S", "W", "A", "B", "C", or "D"

Example:
oshrun –np 32 ./bin/is.C.32
# cray-shmem with ALPS launcher

`aprun -d 16 -S 1 -n $NPROCS ./bin/is.C.$NPROCS`

![Graph showing MOP/s total vs. #PEs for EOS NPB IS, with a linear increase from 2 to 32 #PEs, and then a flat line for 32 to 64 #PEs. The graph is labeled crayshm.](image-url)
GUPS

- **GUPS**: Giga Updates per Second
  - Adapted from Random Access Benchmark
  - Randomly generate address & PE where update occurs
  - Number memory locations randomly updated in 1 sec/1 billion
    - “Randomly” - no relationship between locations in address space
    - “Update” – read-modify-write on table (HPCC_Table) of 64 bit words

- **Input parameters**
  - (None)
    - Number of PEs used to automatically calculate table size

- **Metric**
  - GUPS – Giga Updates Per Second
GUPS

• OpenSHMEM implementation
  – Table (HPCC_Table) shared via symmetric heap
  – Modify uses get/put/quiet to ensure visible on remote PE
  – Based on spec v1.3 – feedback that v1.4 has atomic XOR

• Multithread variant
  – Use OpenMP for the threading
  – Each thread performs random updates of HPCC_Table
  – Use OpenSHMEM contexts to manage thread specific location information
To Execute:
oshrun –np 32 ./gups

```
Real time used = 58.660473 seconds
0.004576087 Billion (10^9) Updates per second [GUP/s]
0.000143003 Billion (10^9) Updates/PE per second [GUP/s]
```
GUPS Usage & Results

# OpenSHMEM-X execution line (Turing GUPS-threaded)
orterun \
  --hostfile hosts \
  --bind-to socket \
  --map-by ppr:2:node \
  --np $NPROCS \n  x OMP_NUM_THREADS=10 \n./gups

Turing GUPS-threaded

![Graph showing the relationship between number of PEs and GUPS performance](image)

- GUPS: 0.002 to 0.012
- #PEs: 4 to 24
- The graph illustrates the performance of GUPS with increasing number of PEs, showing a trend of increasing GUPS values with more PEs.
• SHOMS Micro-benchmark testing suite
  – Based on UOMS benchmark for UPC micro operations
  – Tests OpenSHMEM API
  – Minimal test of each function

• Metric(s)
  – Tests report latencies (min/max/avg)
  – Tests report bandwidth (when function transfers data)

• Highlights
  – Strictly test performance, not test correctness
  – "Affinity Mode": subset of tests run on 2 nodes to identify if a core is favored by OpenSHMEM on a particular node
To Execute:
```bash
oshrun --np 2 ./shoms [--input test-FEATURE.txt] [FLAGS…]
```

**SHOMS flags**

--off_cache: Shifts the data in the symmetric heap in an effort to disable the effects of caches on CPU.

--warmup: Do N/10 untimed iterations before doing N iterations in the main loop. Warms up hardware/caches.

--msglen: Points to a file with a set of message lengths. Put one number per line in the file and it will use N bytes per message for each line in the file.

--minsize: Starts tests at N bytes. Scales up by N*2 bytes on each iteration until it goes above maxsize. Default minsize is 8.

--maxsize: Ends when N*2 is greater than maxsize. Default maxsize is 16777216

--time: Soft limit of N seconds for each iteration. This will not interrupt network operations.

--output: File to write results to. Default stdout. Will truncate existing files.

--input: File that lists tests to perform. By default SHOMS will run all tests available. List one test per line.

--affinity: Run affinity test mode.
Using OpenSHMEM version 1.3
Created all test list.
Will be running with 128 different tests
Will be running with 22 different size configurations
Using OpenSHMEM version 1.3
Running tests

...<snip>...

# Benchmarking shmem_barrier_all
# #processes = 8
#----------------------------------------
# bytes  #repetitions  t_min[nsec]  t_max[nsec]  t_avg[nsec]  Bw_aggregated[MB/sec]
Bw_aggregated[MB/sec]
   N/A        1000    12345        12937.99    43411
   N/A

...<snip>...
SHOMS Usage & Results

To Execute:
aprun -d 16 -S 1 -n $NPES ./shoms --input barrier.txt --maxsize 8
Summary

• Overview of OpenSHMEM Benchmark (OSB) suite
  – Included details on usage & example outputs/metrics
  – Used OSB with different OpenSHMEM implementations

• Highlight enhancements for multithreaded variants
  – Graph500, GUPS & SSCA1

• OSB Suite publicly available
  – Encourage community use & improvements

https://github.com/ornl-languages/osb
Publications where OSB codes appeared


Acknowledgements

This work was supported by the United States Department of Defense (DoD) and used resources of the Computational Research and Development Programs at Oak Ridge National Laboratory.

This work was sponsored by the U.S. Department of Energy’s Office of Advanced Scientific Computing Research.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
Questions?