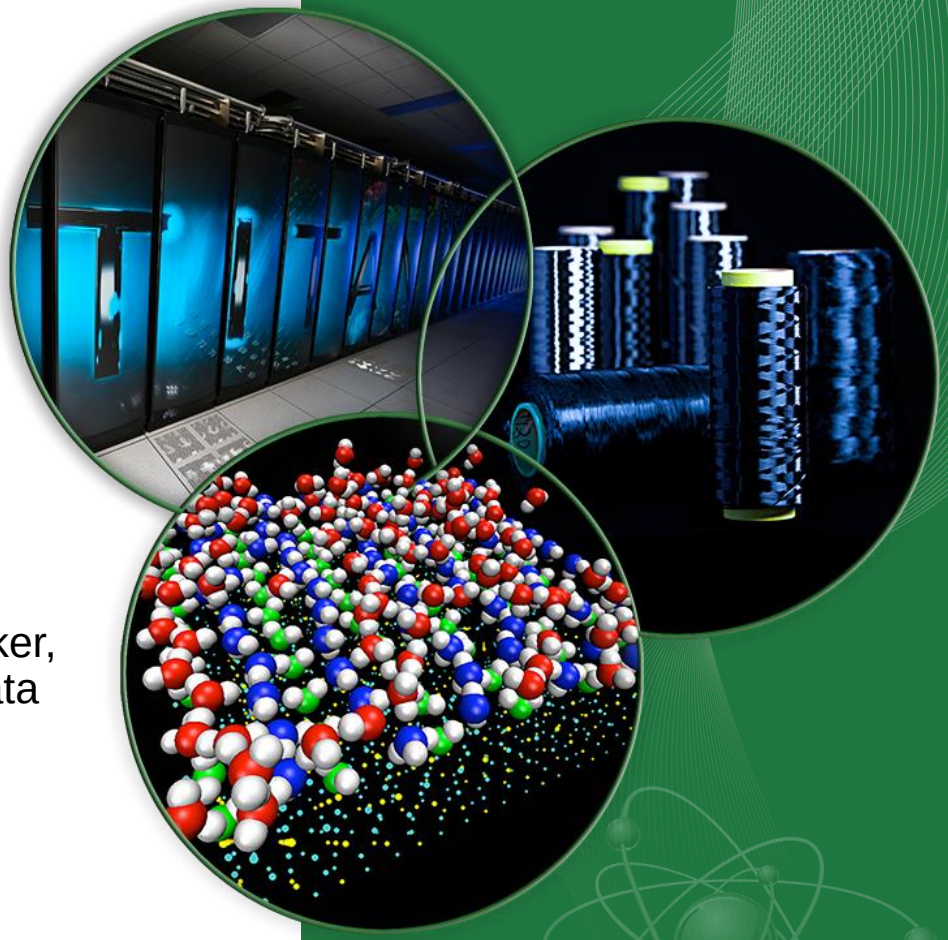
Oak Ridge National Laboratory
Computing and Computational Sciences Directorate

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

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory

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)

Benchmark	Single Threaded	Multithreaded
Graph500	YES	YES
SSCA1	YES	YES
NPB	YES	NO
GUPS	YES	YES
SHOMS	YES	N/A

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^{scale_factor}
    - Number of edges = 2 x *edge_factor*
  - Memory required:
    - Example: (2^{24} x (2 x **16**)) x 8 (bytes) = 4MB

scale_factor = 24 *edge_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

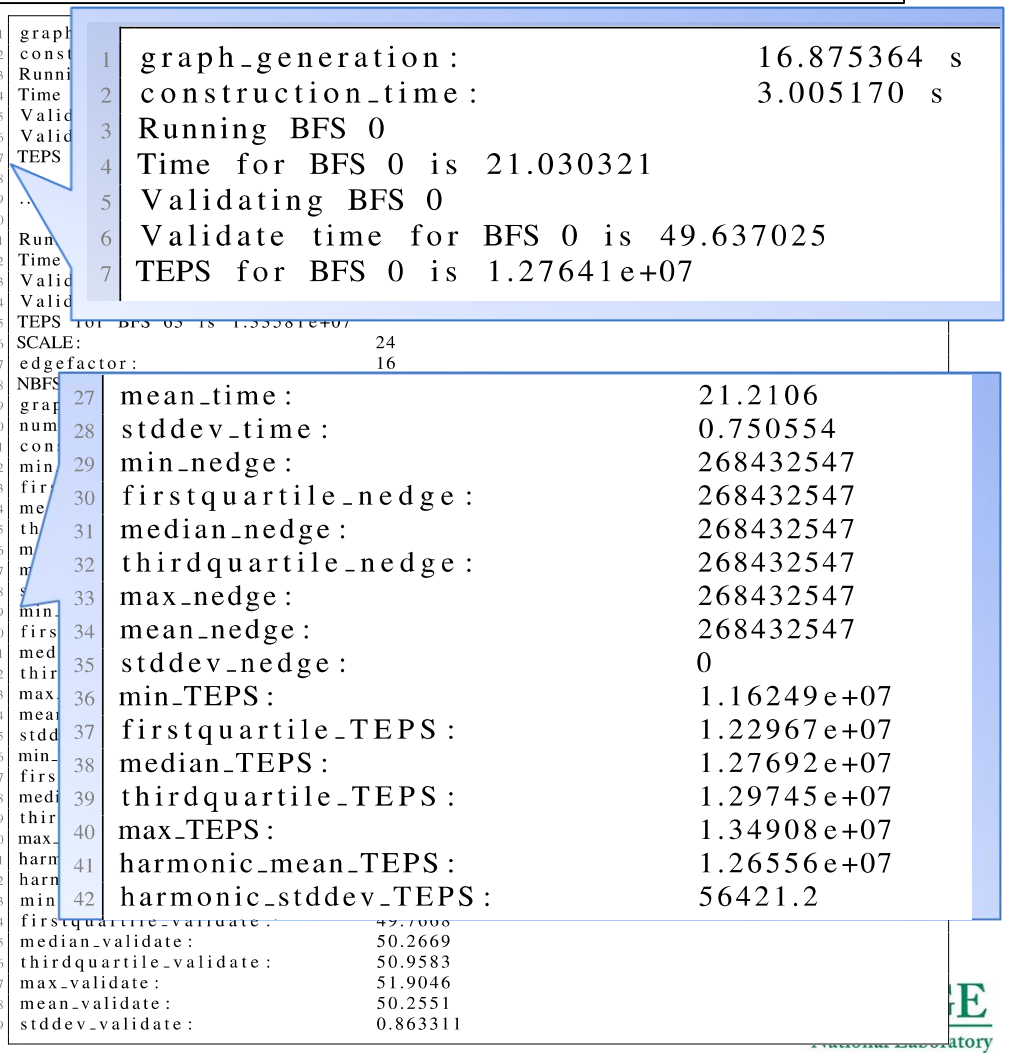
Graph500 OpenSHMEM

- 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

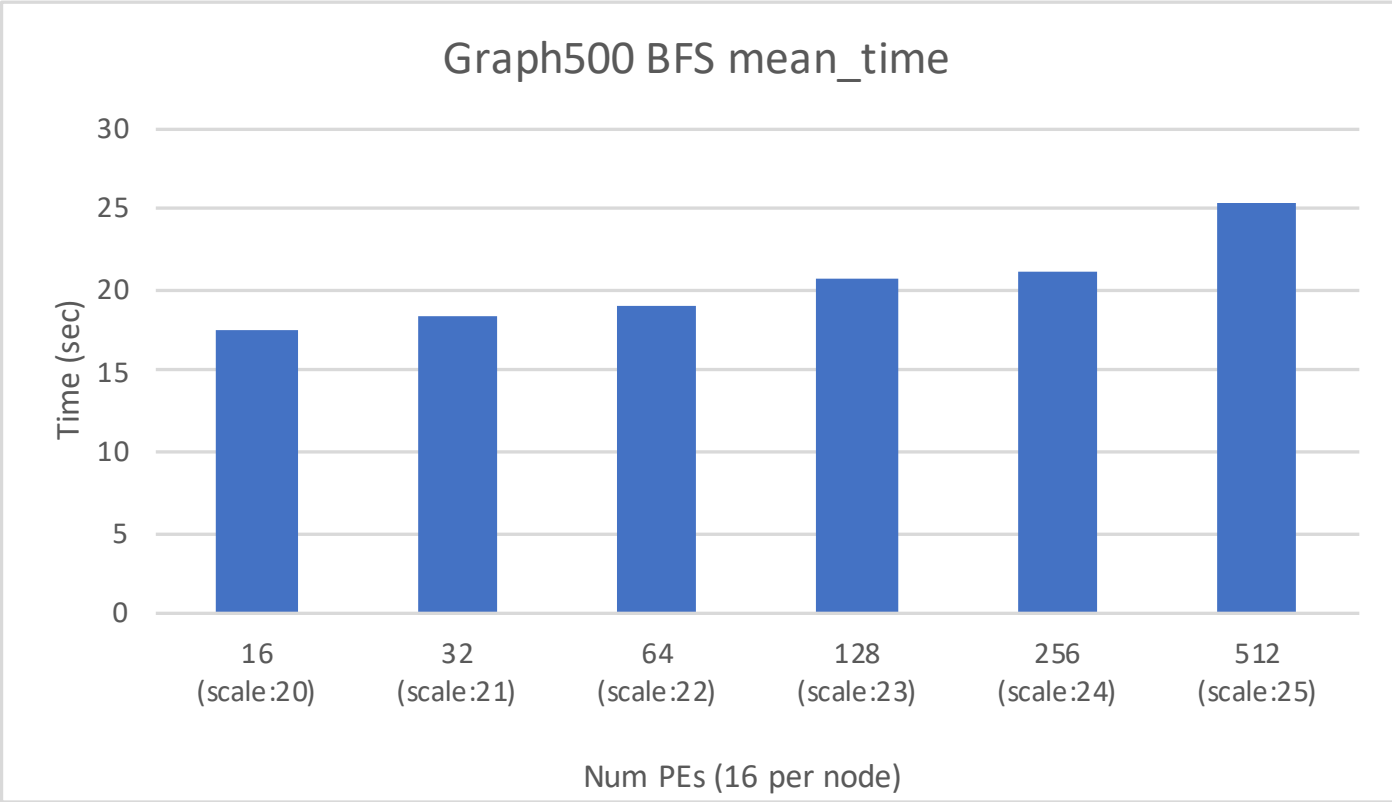
Graph500 Usage & Results

To Execute:
cd mpi/
oshrun –np 32 ./graph500_shmem_one_sided 24 16

- Environment variables
  - SHORT_VALIDATION (8 BFS)
  - SKIP_VALIDATION



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)

SSCA1

- 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:

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

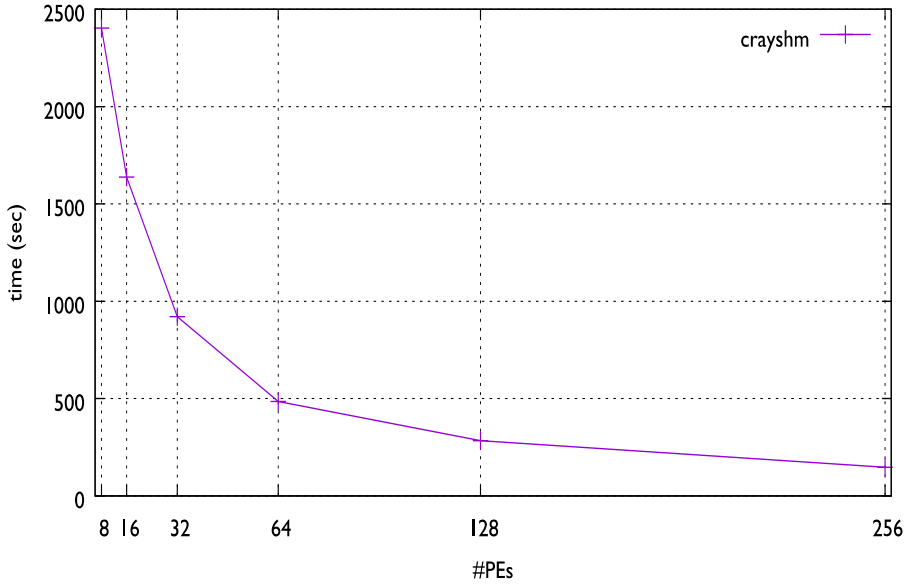
```
Running with OpenMP, thread count: 8
Running with OpenMP, thread count: 8
Running with OpenMP, thread count: 8
Running with OpenMP, thread count: 8
Runni
...<s
Begin

   Ela


Begin

   Ela


Found

Start
posit


veri
   10
  230

verify
  138
     gagaacgactga      13856
  37081  *STARTGAPMIDST—END*  tgaagcacggcgaggacg————
     atgatagacagcacgggggcgccggagaacgactga      37098

...<snip>...
```

```
verifyAlignment 0, succeeded; score 55:
  26590  *IDENTICAL*  tgaatagacgagaacacgatatgcgcgctgt
  29129  *IDENTICAL*  tgaatagacgagaacacgatatgcgcgctgt

verifyAlignment 1, succeeded; score 54:
   1039  *MISQRMATCHES*  tgaatgataagccagaggatggcgacg
  23093  *MISQRMATCHES*  tgaatgataagcaggcagatggcgacg

verifyAlignment 2, succeeded; score 53:
  13839  *STARTGAPMIDST—END*  tgaagcacggcgaggacgggg
     gagaacgactga      13856
  37081  *STARTGAPMIDST—END*  tgaagcacggcgaggacg——
     atgatagacagcacgggggcgccggagaacgactga      37098
```

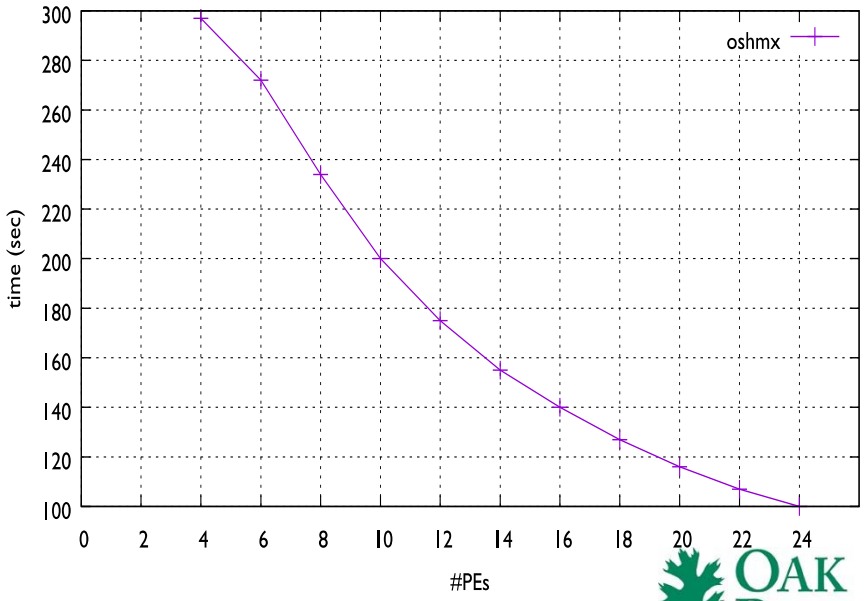
SSCA1 Usage & Results

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

EOS SSCA1



Turing SSCA1-threaded



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

NPB

- 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

NPB Usage & Results

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"
```

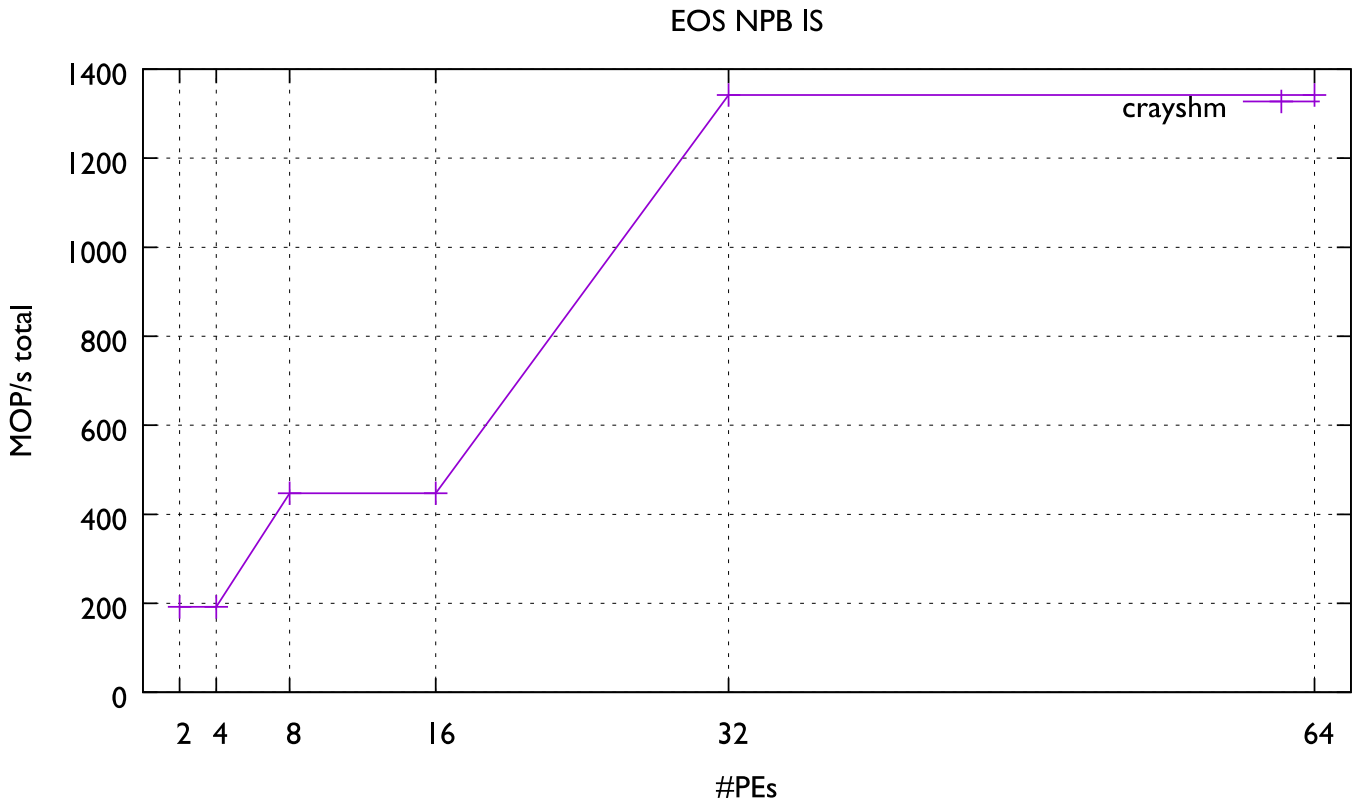
Example:

```
oshrun –np 32 ./bin/is.C.32
```

```
 9  IS Benchmark Completed
10  Class             =                        C
11  Size              =                134217728
12  Iterations        =                       10
13  Time in seconds   =                     1.00
14  Total processes   =                       32
15  Compiled procs    =                       32
16  Mop/s total       =                  1342.18
17  Mop/s/process     =                    41.94
...
CMPI_LIB      = (none)
CMPI_INC      = (none)
CFLAGS        = -O3 -g
CLINKFLAGS    = -O3 -g
...<snip>...
```

NPB Usage & Results

```
# cray-shmem with ALPS launcher

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



GUPS

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

GUPS Usage & Results

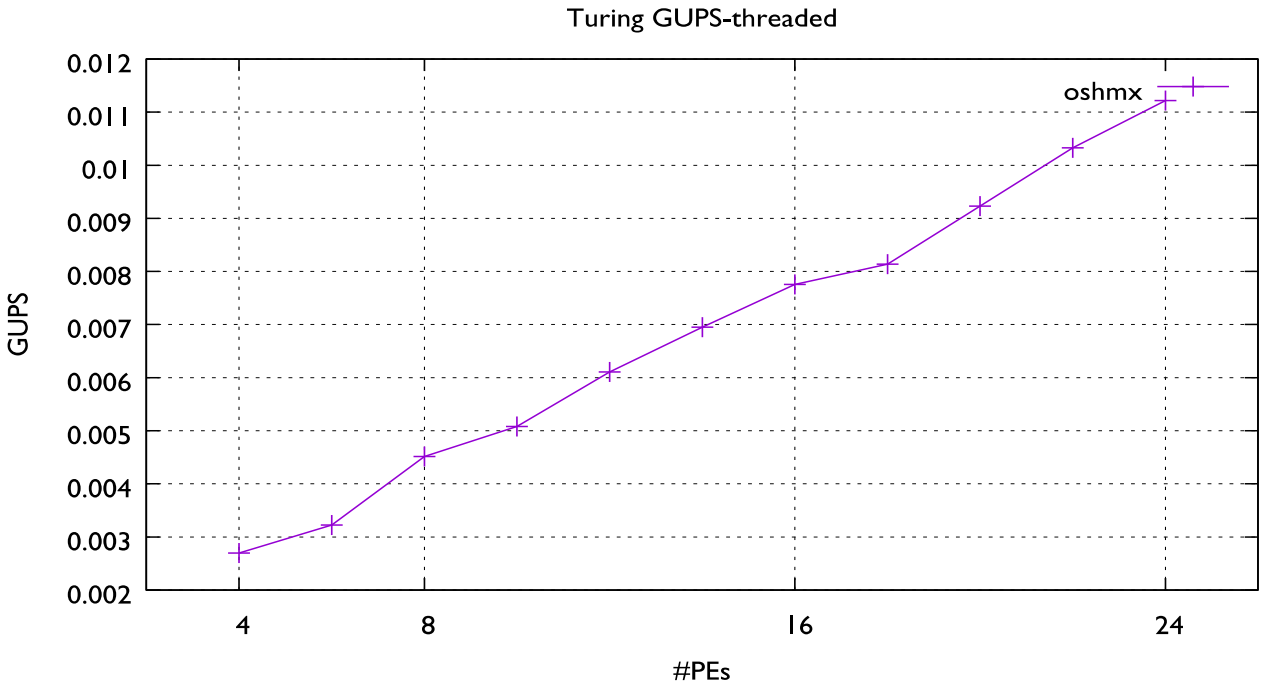
To Execute:
oshrun –np 32 ./gups

```
5 Real time used = 58.660473 seconds
6 0.004576087 Billion(10^9) Updates per second [GUP/s]
7 0.000143003 Billion(10^9) Updates/PE per second [GUP/s]
```

```
1 Runni
2 Total
3 PE Ma
4 Default number of                    = 268435456      and actually done =
      26843545
5 Real time used = 58.660473 seconds
6 0.004576087 Billion(10^9) Updates     per second [GUP/s]
7 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 \                    # Num PEs per node
        –np $NPROCS \                           # Num PEs (total)
        –x OMP_NUM_THREADS=10 \                 # Num Threads per PE
       ./gups
```



SHOMS

- 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

SHOMS Usage & Results

To Execute:
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 a 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.

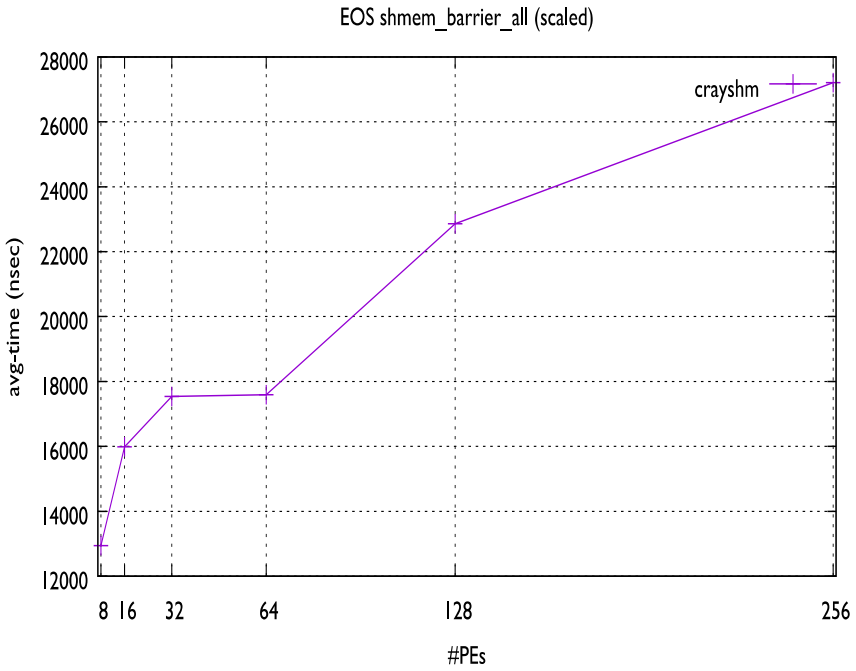
SHOMS Usage & Results

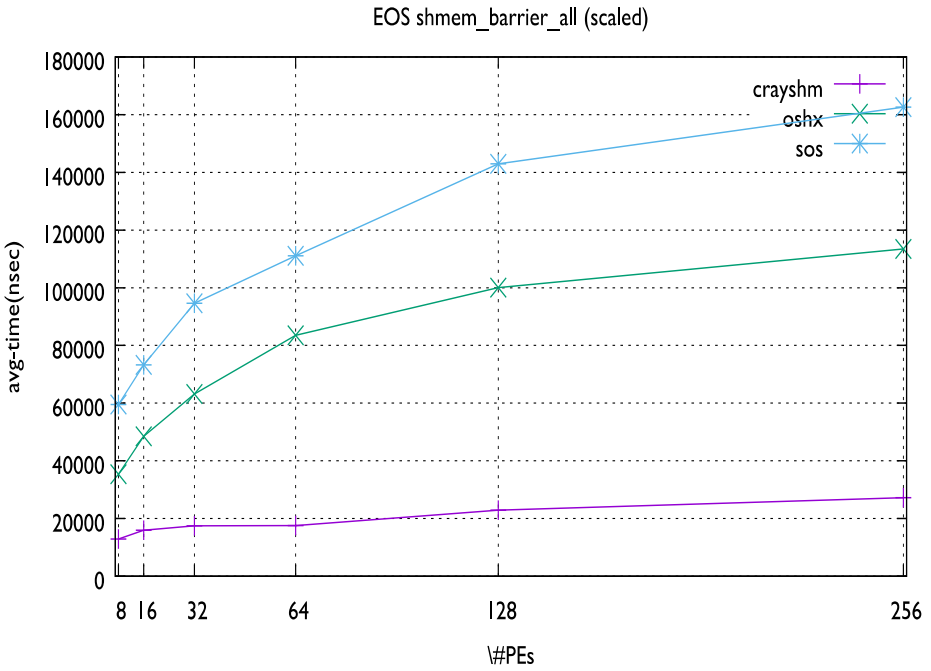
```
Using OpenSHMEM version 1.3
Created all test list.
Will be running with 128 different tests
Will be running with 22 different size configurations
Using OpenSHMEM version 1.3
Running tests

...<snip>...

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

- [2] M. Baker, F. Aderholdt, M. Gorentla Venkata, and P. Shamis. **OpenSHMEM-UCX: Evaluation of UCX for Implementing OpenSHMEM Programming Model**. OpenSHMEM'16.
- [3] M. Baker, A. Welch, and M. Gorentla Venkata. **Parallelizing the Smith-Waterman Algorithm Using OpenSHMEM and MPI-3 One-Sided Interfaces**. OpenSHMEM'15.
- [4] A. Bouteiller, S. Pophale, S. Boehm, M. Baker, and M. Gorentla Venkata. **Evaluating Contexts in OpenSHMEM-X Reference Implementation**. OpenSHMEM'17.
- [6] E. F. D'Azevedo and N. Imam. **Graph500 in OpenSHMEM**. OpenSHMEM'15.
- [11] S. Pophale, R. Nanjegowda, H. Jin, B. Chapman, A. Curtis, S. Poole, and J. A. Kuehn. **OpenSHMEM Performance and Potential: A NPB Experimental Study**. Proc. of PGAS-12, 2012.
- [14] P. Shamis, M. Gorentla Venkata, S. Poole, A. Welch, and T. Curtis. **Designing a High Performance OpenSHMEM Implementation Using Universal Common Communication Substrate as a Communication Middleware**. OpenSHMEM'14.

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?