UNIVERSITY OF HOUSTON

Documentation on OpenSHMEM Test Suite

A comprehensive set of tests for OpenSHMEM Specification 1.0

Acknowledgement

This work was supported by the United States Department of Defense & used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory.

Partial support for this work was provided by the National Science Foundation's Computer Systems Research program under Award No. CRI-0958464.

Disclaimer

Table of Contents

1. Introduction

OpenSHMEM API Specification 1.0 is based on SGI ‘s API. The tests in the OpenSHMEM test suite cover all SHMEM calls that must be supported by all implementations of OpenSHMEM 1.0. We divide the tests into feature tests, performance tests and examples. The feature tests check the completeness of the API supported by an OpenSHMEM library implementation and the performance tests give latency information for important OpenSHMEM calls. The examples directory contains C programs that show the working of one or more OpenSHMEM calls.

2. Tests

The tests are divided into feature tests, performance tests, and examples in the main 'test_suite' directory.

2.1 Feature Tests

The following lists of tests can be found in the folder 'feature_tests' for both C and Fortran;

i. test_shmem_put_shmalloc , test_shmem_put_globals

This tests

a. **elemental put calls**

C/C++ only:

shmem_double_p, shmem_float_p, shmem_int_p, shmem_long_p, shmem_short_p

b. **block put calls**

C/C++ and Fortran:

shmem_put32, shmem_put64, shmem_put128

C/C++ only:

shmem_double_put, shmem_float_put, shmem_int_put, shmem_real_put

c. **strided put calls**

C/C++ and Fortran:

shmem_iput32, shmem_iput64, shmem_iput128,

C/C++ only:

shmem_double_iput, shmem_float_iput, shmem_int_iput, shmem_long_iput, shmem_short_iput

Fortran only:

shmem_complex_iput, shmem_integer_iput, shmem_logical_iput, shmem_real_iput

d. **byte-granularity block put calls**

C/C++ and Fortran:

shmem_putmem

Additional information:

Test test_shmem_put_shmalloc uses shmalloc-ed variables that are allocated and managed by the OpenSHMEM library while the test_shmem_put_globals checks that

the same calls work with global variables. Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() OpenSHMEM call.

ii. test_shmem_get_shmalloc, test_shmem_get_globals

This tests

a. **elemental get calls**

C/C++ only:

shmem_double_p, shmem_float_p, shmem_int_p, shmem_long_p, shmem_short_p

b. **block get calls**

C/C++ and Fortran:

shmem_get32, shmem_get64, shmem_get128

C/C++ only:

shmem_double_get, shmem_float_get, shmem_int_get, shmem_real_get

c. **strided get calls**

C/C++ and Fortran:

shmem_iget32, shmem_iget64, shmem_iget128,

C/C++ only:

shmem_double_iget, shmem_float_iget, shmem_int_iget, shmem_long_iget, shmem_short_iget

Fortran only:

shmem_complex_iget, shmem_integer_iget, shmem_logical_iget, shmem_real_iget

d. **byte-granularity block get calls**

C/C++ and Fortran:

shmem_getmem

Additional information:

Test test_shmem_get_shmalloc uses shmalloc-ed variables that are allocated and managed by the OpenSHMEM library while the test_shmem_get_globals checks that the same calls work with global variables. Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() OpenSHMEM call.

iii. test_shmem_broadcast

This tests shmem_broadcast32, shmem_broadcast64 calls with different active sets (strided broadcast) available for C/C++ and Fortran.

Additional information:
Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() OpenSHMEM call.

iv. test_shmem_barrier

This tests shmem_barrier, shmem_barrier_all calls available for C/C++ and Fortran.
Additional information:
Correct output for this test also depends on a reliable implementation of the shmem_int_p() OpenSHMEM call.

v. test_shmem_reductions

This tests;
C/C++ only:
shmem_int_and_to_all, shmem_long_and_to_all, shmem_longlong_and_to_all, shmem_short_and_to_all, shmem_double_max_to_all, shmem_float_max_to_all, shmem_int_max_to_all, shmem_long_max_to_all, shmem_longdouble_max_to_all, shmem_longlong_max_to_all, shmem_short_max_to_all, shmem_double_min_to_all, shmem_float_min_to_all, shmem_int_min_to_all, smem_long_min_to_all, shmem_longdouble_min_to_all, shmem_longlong_min_to_all, shmem_short_min_to_all, shmem_double_sum_to_all, shmem_float_sum_to_all, shmem_int_sum_to_all, shmem_long_sum_to_all, shmem_longdouble_sum_to_all, shmem_longlong_sum_to_all, shmem_short_sum_to_all, shmem_double_prod_to_all, shmem_float_prod_to_all, shmem_int_prod_to_all, shmem_long_prod_to_all, shmem_longdouble_prod_to_all, shmem_longlong_prod_to_all, shmem_short_prod_to_all, shmem_int_or_to_all, shmem_long_or_to_all, shmem_longlong_or_to_all, shmem_short_or_to_all, shmem_int_xor_to_all, shmem_long_xor_to_all, shmem_longlong_xor_to_all, shmem_short_xor_to_all

Fortran only:
shmem_int4_and_to_all, shmem_int8_and_to_all, shmem_real4_max_to_all, shmem_real8_max_to_all, shmem_real16_max_to_all, shmem_int4_max_to_all, shmem_int8_max_to_all, shmem_real4_min_to_all, shmem_real8_min_to_all, shmem_real16_min_to_all, shmem_int4_min_to_all, shmem_int8_min_to_all, shmem_real4_sum_to_all, shmem_real8_sum_to_all, shmem_real16_sum_to_all, shmem_int4_sum_to_all, shmem_int8_sum_to_all, shmem_real4_prod_to_all, shmem_real8_prod_to_all, shmem_real16_prod_to_all, shmem_int4_prod_to_all, shmem_int8_prod_to_all, shmem_int4_or_to_all, shmem_int8_or_to_all, shmem_int4_xor_to_all, shmem_int8_xor_to_all

Additional information:
It also tests strided reductions which require at least 3 PEs to test. Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() OpenSHMEM call.

vi. test_shmem_atomic

This tests;
C/C++ only:
shmem_double_swap, shmem_float_swap, shmem_int_cswap, shmem_int_fadd, shmem_int_finc, shmem_int_swap, shmem_long_cswap, shmem_long_fadd, shmem_long_finc, shmem_long_swap, shmem_longlong_cswap, shmem_longlong_fadd, shmem_longlong_finc, shmem_longlong_swap

Fortran only:
shmem_int4_cswap, shmem_int4_fadd, shmem_int4_finc, shmem_int4_swap, shmem_int8_swap, shmem_real4_swap, shmem_real8_swap, shmem_int8_cswap shmem_int4_add, shmem_int4_inc
Additional information:
Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() and shmem_int_put OpenSHMEM calls.

vii. test_shmem_synchronization

This tests;
C/C++ only:
shmem_int_wait, shmem_int_wait_until, shmem_long_wait, shmem_long_wait_until, shmem_longlong_wait, shmem_longlong_wait_until, shmem_short_wait, shmem_short_wait_until
Fortran only:
shmem_int4_wait, shmem_int4_wait_until, shmem_int8_wait, shmem_int8_wait_until
Additional information:
Correct output for this test also depends on a reliable implementation of the shmem_barrier_all(), shmem_long_put and shmem_long_wait OpenSHMEM calls.

viii. test_shmem_accessible

This tests;
C/C++ and Fortran:
shmem_pe_accessible, shmem_addr_accessible
Additional information:

Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() OpenSHMEM call.

ix. test_shmem_collects

This tests;
C/C++ only:
shmem_collect32, shmem_collect64, shmem_fcollect32, shmem_fcollect64
Fortran only:
shmem_collect4, shmem_collect8, shmem_collect32, shmem_collect64,
shmem_fcollect4, shmem_fcollect8, shmem_fcollect32, shmem_fcollect64
Additional information:
Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() OpenSHMEM call.

x. test_shmem_lock

This tests;
C/C++ and Fortran:
shmem_clear_lock, shmem_set_lock, shmem_test_lock
Additional information:
Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() and shmem_quiet OpenSHMEM calls.

2.2 Performance Tests

2.2.1 Micro-benchmarks

The Micro-benchmark directory contains programs to measure latency of data transfer calls and collective calls in OpenSHMEM.

Performance tests measure the time taken for a OpenSHMEM call by finding the average over 10000 calls.

i. put_performance
ii. get_performance
iii. broadcast_performance
iv. barrier_performance
v. collects_performance

Additional information:

Correct output for this test also depends on a reliable implementation of the shmem_barrier_all() OpenSHMEM call.

2.3 Examples

These are C programs that test one or more OpenSHMEM calls. These are small and useful programs intended for beginners to SHMEM programming. Some programs in this category are described below.

a. hello.c – All PEs are initialized and print a "Hello" to screen
b. cpi.c – PI approximator
c. sping.c – Ping-pong test to check bandwidth utilization
d. shmem_matrix.c - Calculates the product of two matrices A and B, based on block distribution. This program is adapted from the MPI implementation of matrix muliplication based on 1D block-column distribution. In each iteration, the PE calculates the partial result of matrix-matrix multiply. After the multiplication, the PE sends the current portion of matrix A to its right neighbor and receives the next portion of matrix A from its left neighbor.
e. shmem_2dheat.c - Application for 2D heat transfer modeling using different methods. Adapted from the parallel MPI implementation of 2D heat conduction finite difference over a regular domain using the following methods, jacobi, Gauss-Siedel and SOR. Reference: URL of the MPI implementation http://www.cct.lsu.edu/~estrabd/2dheat.php
f. shmem_heat_image.c - Application solving heat conduction task based on row-based distribution of the matrix. The application distributes the matrix in rows among PEs and then exchanges the result of computation. After doing all the transfers the output can be written to a file in image format. Reference: URL of the original implementation at http://www.kiam.ru/MVS/documents/k100/examples/progrev_shmem_cpu.cpp
g. shmem_daxpy.c - A simple DAXPY like kernel with computation and communication. It simulates a typical application which uses one dimensional array for local computation and does a reduction collective operation of the result. Reference: http://parallel-for.sourceforge.net/shmem-proc-cpu-scalar.html
h. adjacent_32bit_amo.c – Contributed by SGI, this is a 32 bit Atomic Memory Operation (AMO) test that causes difficulties if the operation is implemented by InfiniBand alone since InfiniBand supports only 64 bit AMOs.
i. ptp.c – A Passive Target Progress test completes execution only if the atomic operations issued by other PEs complete in the absence of an OpenSHMEM call at the target PE.

3. Running Tests

Edit the Makefile or export values, such that they use the appropriate compiler, (SHMEM_FLAGS), run command (RUNCMD), run options (RUNOPT), command line options to control execution environment (NPROCOPT), NPROC (this parameter decides the number of PEs, default value is 4).

To compile use ‘make all’ and to execute use ‘make run’.

4. Expected Results and their interpretation

Example: Feature test for atomic operations

Execute test_shmem_atomics.c

Expected Result:

Test shmem_int_swap: Passed
Test shmem_float_swap: Passed
Test shmem_long_swap: Passed
Test shmem_double_swap: Passed
Test shmem_longlong_swap: Passed
Test shmem_int_cswap: Passed
Test shmem_long_cswap: Passed
Test shmem_longlong_cswap: Passed
Test shmem_int_fadd: Passed
Test shmem_long_fadd: Passed
Test shmem_longlong_fadd: Passed
Test shmem_int_finc: Passed
Test shmem_long_finc: Passed
Test shmem_longlong_finc: Passed

If the test says ‘Passed’ then the routines that are being tested behave in accordance with OpenSHMEM Specification 1.0 and the result produced (if applicable) is correct.