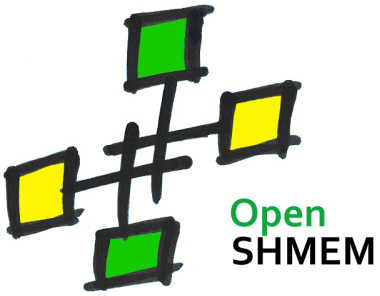
OpenSHMEM

Application Programming Interface



http://www.openshmem.org/

Version 1.5

29th October 2019

Development by

- For a current list of contributors and collaborators please see
  http://www.openshmem.org/site/Contributors/
- For a current list of OpenSHMEM implementations and tools, please see
  http://openshmem.org/site/Links#impl/

Sponsored by

- U.S. Department of Defense (DoD)
  http://www.defense.gov/
- Oak Ridge National Laboratory (ORNL)
  http://www.ornl.gov/
- Los Alamos National Laboratory (LANL)
  http://www.lanl.gov/

Current Authors and Collaborators

- Matthew Baker, ORNL
- Swen Boehm, ORNL
- Aurelien Bouteiller, University of Tenneesee at Knoxville (UTK)
- Barbara Chapman, Stonybrook University (SBU)
- Robert Cernohous, Cray Inc.
- James Culhane, LANL
- Tony Curtis, SBU
- James Dinan, Intel
- Mike Dubman, Mellanox
- Karl Feind, Hewlett Packard Enterprise (HPE)
- Manjunath Gorentla Venkata, ORNL
- Megan Grodowitz, Arm Inc.
- Max Grossman, Rice University
- Khaled Hamidouche, Advanced Micro Devices (AMD)
- Jeff Hammond, Intel
- Yossi Itigin, Mellanox
- Bryant Lam, DoD
- David Knaak, Cray Inc.
- Jeff Kuehn, LANL
- Jens Manser, DoD
- Tiffany M. Mintz, ORNL
- David Ozog, Intel
- Nicholas Park, DoD
- Steve Poole, Open Source Software Solutions (OSSS)
- Wendy Poole, OSSS

- Swaroop Pophale, ORNL
- Sreeram Potluri, NVIDIA
- Howard Pritchard, LANL
- Naveen Ravichandrasekaran, Cray Inc.
- Michael Raymond, HPE
- James Ross, Army Research Laboratory (ARL)
- Pavel Shamis, Arm Inc.
- Sameer Shende, University of Oregon (UO)
- Lauren Smith, DoD

Alumni Authors and Collaborators

- Amrita Banerjee, University of Houston (UH)
- Monika ten Bruggencate, Cray Inc.
- Eduardo D'Azevedo, ORNL
- Oscar Hernandez, ORNL
- Gregory Koenig, ORNL
- Graham Lopez, ORNL
- Ricardo Mauricio, UH
- Ram Nanjegowda, UH
- Aaron Welch, ORNL

Acknowledgments

The OpenSHMEM specification belongs to Open Source Software Solutions, Inc. (OSSS), a non-profit organization, under an agreement with HPE. For a current list of Contributors and Collaborators, please see http://www.openshmem.org/site/Contributors/. We gratefully acknowledge support from Oak Ridge National Laboratory's Extreme Scale Systems Center and the continuing support of the Department of Defense.

We would also like to acknowledge the contribution of the members of the OpenSHMEM mailing list for their ideas, discussions, suggestions, and constructive criticism which has helped us improve this document.

OpenSHMEM 1.4 is dedicated to the memory of David Charles Knaak. David was a highly involved colleague and contributor to the entire OpenSHMEM project. He will be missed.

Contents

Annex A

Writing OpenSHMEM Programs

Incorporating OpenSHMEM into Programs

The following section describes how to write a “Hello World" OpenSHMEM program. To write a “Hello World" OpenSHMEM program, the user must:

- Include the header file *shmem.h* for *C*.
- Add the initialization call *shmem_init*.
- Use OpenSHMEM calls to query the local *Processing Element* (PE) number (*shmem_my_pe*) and the total number of PEs (*shmem_n_pes*).
- Add the finalization call *shmem_finalize*.

In OpenSHMEM, the order in which lines appear in the output is not deterministic because PEs execute asynchronously in parallel.

Listing A.1: “Hello World” example program in *C*

```c
#include <stdio.h>
#include <shmem.h> /* The OpenSHMEM header file */

int main (void)
{
   shmem_init();
   int me = shmem_my_pe();
   int npes = shmem_n_pes();
   printf("Hello from %d of %d\n", me, npes);
   shmem_finalize();
   return 0;
}
```

Listing A.2: Possible ordering of expected output with 4 PEs from the program in Listing A.1

```
Hello from 0 of 4
Hello from 2 of 4
Hello from 3 of 4
Hello from 1 of 4
```

The example in Listing A.3 shows a more complex OpenSHMEM program that illustrates the use of symmetric data objects. Note the declaration of the *static short dest* array and its use as the remote destination in *shmem_put*.

The *static* keyword makes the *dest* array symmetric on all PEs. Each PE is able to transfer data to a remote *dest* array by simply specifying to an OpenSHMEM routine such as *shmem_put* the local address of the symmetric data object that will receive the data. This local address resolution aids programmability because the address of the *dest* need not be exchanged with the active side (PE *0*) prior to the *Remote Memory Access* (RMA) routine.

Conversely, the declaration of the *short source* array is asymmetric (local only). The *source* object does not need to be symmetric because *Put* handles the references to the *source* array only on the active (local) side.

Listing A.3: Example program with symmetric data objects

```
#include <stdio.h>
#include <shmem.h>

#define SIZE 16

int main(void)
{
   short source[SIZE];
   static short dest[SIZE];
   static long lock = 0;
   shmem_init();
   int me = shmem_my_pe();
   int npes = shmem_n_pes();
   if (me == 0) {
      /* initialize array */
      for (int i = 0; i < SIZE; i++)
         source[i] = i;
      /* local, not symmetric */
      /* static makes it symmetric */
      /* put "size" words into dest on each PE */
      for (int i = 1; i < npes; i++)
         shmem_put(dest, source, SIZE, i);
   }
   shmem_barrier_all(); /* sync sender and receiver */
   if (me != 0) {
      shmem_set_lock(&lock);
      printf("dest on PE %d is \t", me);
      for (int i = 0; i < SIZE; i++)
         printf("%hd \t", dest[i]);
      printf("\n");
      shmem_clear_lock(&lock);
   }
   shmem_finalize();
   return 0;
}
```

Listing A.4: Possible ordering of expected output with 4 PEs from the program in Listing A.3

```
dest on PE 1 is 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
dest on PE 2 is 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
dest on PE 3 is 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Annex B

Compiling and Running Programs

The OpenSHMEM Specification does not specify how OpenSHMEM programs are compiled, linked, and run. This section shows some examples of how wrapper programs are utilized in the OpenSHMEM Reference Implementation to compile and launch programs.

1 Compilation

Programs written in *C*

The OpenSHMEM Reference Implementation provides a wrapper program, named **oshcc**, to aid in the compilation of *C* programs. The wrapper may be called as follows:

```
oshcc <compiler options> -o myprogram myprogram.c
```

Where the ⟨compiler options⟩ are options understood by the underlying *C* compiler called by **oshcc**.

Programs written in *C++*

The OpenSHMEM Reference Implementation provides a wrapper program, named **oshc++**, to aid in the compilation of *C++* programs. The wrapper may be called as follows:

```
oshc++ <compiler options> -o myprogram myprogram.cpp
```

Where the ⟨compiler options⟩ are options understood by the underlying *C++* compiler called by **oshc++**.

2 Running Programs

The OpenSHMEM Reference Implementation provides a wrapper program, named **oshrun**, to launch OpenSHMEM programs. The wrapper may be called as follows:

```
oshrun <runner options> -np <#> <program> <program arguments>
```

The arguments for **oshrun** are:

⟨runner options⟩	Options passed to the underlying launcher.
-np ⟨#⟩	The number of PEs to be used in the execution.
⟨program⟩	The program executable to be launched.
⟨program arguments⟩	Flags and other parameters to pass to the program.

Annex C

Undefined Behavior in OpenSHMEM

The OpenSHMEM Specification formalizes the expected behavior of its library routines. In cases where routines are improperly used or the input is not in accordance with the Specification, the behavior is undefined.

Inappropriate Usage	**Undefined Behavior**
Uninitialized library	If the OpenSHMEM library is not initialized, calls to non-initializing OpenSHMEM routines have undefined behavior. For example, an implementation may try to continue or may abort immediately upon an OpenSHMEM call into the uninitialized library.
Multiple calls to initialization routines	In an OpenSHMEM program where the initialization routines *shmem_init* or *shmem_init_thread* have already been called, any subsequent calls to these initialization routines result in undefined behavior.
Accessing non-existent PEs	If a communications routine accesses a non-existent PE, then the OpenSHMEM library may handle this situation in an implementation-defined way. For example, the library may report an error message saying that the PE accessed is outside the range of accessible PEs, or may exit without a warning.
Use of non-symmetric variables	Some routines require remotely accessible variables to perform their function. For example, a *Put* to a non-symmetric variable may be trapped where possible and the library may abort the program. Another implementation may choose to continue execution with or without a warning.
Non-symmetric allocation of symmetric memory	The symmetric memory management routines are collectives. For example, all PEs in the program must call *shmem_malloc* with the same *size* argument. Program behavior after a mismatched *shmem_malloc* call is undefined.
Use of null pointers with non-zero *len* specified	In any OpenSHMEM routine that takes a pointer and *len* describing the number of elements in that pointer, a null pointer may not be given unless the corresponding *len* is also specified as zero. Otherwise, the resulting behavior is undefined. The following cases summarize this behavior: • *len* is 0, pointer is null: supported. • *len* is not 0, pointer is null: undefined behavior. • *len* is 0, pointer is non-null: supported. • *len* is not 0, pointer is non-null: supported.

Annex D

Interoperability with Other Programming Models

OpenSHMEM routines may be used in conjunction with the routines of other communication libraries or parallel languages in the same program. This section describes the interoperability with other programming models, including clarification of undefined behaviors caused by mixed use of different models, and advice to OpenSHMEM library users and developers that may improve the portability and performance of hybrid programs.

1 MPI Interoperability

OpenSHMEM and MPI are two commonly used parallel programming models for distributed-memory systems. The user can choose to utilize both models in the same program to efficiently and easily support various communication patterns.

A vendor may implement the OpenSHMEM and MPI libraries in different ways. For instance, one may implement both OpenSHMEM and MPI as standalone libraries, each of which allocates and initializes fully isolated communication resources. Another approach is to implement both OpenSHMEM and MPI interfaces within the same software system in order to share a communication resource when possible.

To improve interoperability and portability in OpenSHMEM + MPI hybrid programming, we clarify the relevant semantics in the following subsections.

1.1 Initialization

In order to ensure that a hybrid program can be portably performed with different vendor implementations, the OpenSHMEM environment of the program must be initialized by a call to *shmem_init* or *shmem_init_thread* and be finalized by a call to *shmem_finalize*; the MPI environment of the program must be initialized by a call to *MPI_Init* or *MPI_Init_thread* and be finalized by a call to *MPI_Finalize*.

Note to implementors

Portable implementations of OpenSHMEM and MPI must ensure that the initialization calls can be made in an arbitrary order within a program; the same rule also applies to the finalization calls. A software runtime that utilizes a shared communication resource for OpenSHMEM and MPI communication may maintain an internal reference counter in order to ensure that the shared resource is initialized only once and thus no shared resource is released until the last finalization call is made.

1.2 Dynamic Process Creation

MPI defines a dynamic process model that allows creation of processes after an MPI application has started (e.g., by calling *MPI_Comm_spawn*) and connection to independent processes (e.g., through *MPI_Comm_accept* and *MPI_Comm_connect*). It provides a mechanism to establish communication between the newly created processes and the existing MPI application (see MPI standard version 3.1, Chapter 10). Unlike MPI, OpenSHMEM starts all processes at once and requires all PEs to collectively allocate and initialize resources (e.g., symmetric heap) used by the OpenSHMEM library before any other OpenSHMEM routine may be called. OpenSHMEM does not support communication with dynamically created or connected processes. In such a scenario, MPI can be used to communicate with these processes.

1.3 Thread Safety

Both OpenSHMEM and MPI define the interaction with user threads in a program with routines that can be used for initializing and querying the thread environment. A hybrid program may request different thread levels at the initialization calls of OpenSHMEM and MPI environments; however, the returned support level provided by the OpenSHMEM or MPI library might be different from that returned in an non-hybrid program. For instance, the former initialization call in a hybrid program may initialize a resource with the requested thread level, but the supported level cannot be updated by a subsequent initialization call if the underlying software runtime of OpenSHMEM and MPI share the same internal communication resource. The program should always check the *provided* thread level returned at the corresponding initialization call or query the level of thread support after initialization to portably ensure thread support in each communication environment.

Both OpenSHMEM and MPI define similar thread levels, namely, *THREAD_SINGLE*, *THREAD_FUNNELED*, *THREAD_SERIALIZED*, and *THREAD_MULTIPLE*. When requesting threading support in a hybrid program, however, the following additional rules are applied if the implementations of OpenSHMEM and MPI share the same internal communication resource. It is strongly recommended to always follow these rules to ensure program portability.

- The *THREAD_SINGLE* thread level requires a single-threaded program. Hence, a hybrid program should not request *THREAD_SINGLE* at the initialization call of either OpenSHMEM or MPI but request a different thread level at the initialization call of the other model.

- The *THREAD_FUNNELED* thread level allows only the main thread to make communication calls. A hybrid program using the *THREAD_FUNNELED* thread level in both OpenSHMEM and MPI should ensure that the same main thread is used in both communication environments.

- The *THREAD_SERIALIZED* thread level requires the program to ensure that communication calls are not made concurrently by multiple threads. If a hybrid program uses *THREAD_SERIALIZED* in one communication environment and *THREAD_SERIALIZED* or *THREAD_FUNNELED* in the other one, it should also guarantee that the OpenSHMEM and MPI calls are not made concurrently from two distinct threads.

1.4 Mapping Process Identification Numbers

Similar to the PE number in OpenSHMEM, MPI defines rank as the identification number of a process in a communicator. Both the OpenSHMEM PE and the MPI rank are unique integers assigned from zero to one less than the total number of processes. In a hybrid program, the OpenSHMEM PE number in *SHMEM_TEAM_WORLD* and the MPI rank in *MPI_COMM_WORLD* of a process can be equal. This feature, however, may be provided by only some of the OpenSHMEM and MPI implementations (e.g., if both environments share the same underlying process manager) and is not portably guaranteed. A portable program should always use the standard functions in each model, namely, *shmem_team_my_pe* in OpenSHMEM and *MPI_Comm_rank* in MPI, to query the process identification numbers in each communication environment and manage the mapping of identifiers in the program when necessary.

Example

The following example demonstrates how to manage the mapping between OpenSHMEM PE numbers and MPI ranks in *MPI_COMM_WORLD* in a hybrid OpenSHMEM and MPI program.

```
#include <stdio.h>
#include <shmem.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);
    shmem_init();

    int mype = shmem_team_my_pe(SHMEM_TEAM_WORLD);
    int npes = shmem_team_n_pes(SHMEM_TEAM_WORLD);

    static int myrank;
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    int *mpi_ranks = shmem_calloc(npes, sizeof(int));

    shmem_int_collect(SHMEM_TEAM_WORLD, mpi_ranks, &myrank, 1);
    if (mype == 0)
        for (int i = 0; i < npes; i++)
            printf("PE %d's MPI rank is %d\n", i, mpi_ranks[i]);

    shmem_free(mpi_ranks);

    shmem_finalize();
    MPI_Finalize();

    return 0;
}
```

1.5 RMA Programming Models

OpenSHMEM and MPI each define similar one-sided communication models; however, a portable program should not assume interoperability between these models. For instance, OpenSHMEM guarantees the atomicity only of concurrent OpenSHMEM AMO operations that operate on symmetric data with the same datatype. Access to the same symmetric object with MPI atomic operations, such as an *MPI_Fetch_and_op*, may result in an undefined result. A hybrid program should avoid situations where MPI and OpenSHMEM one-sided operations perform concurrent accesses to the same memory location; otherwise, the behavior is undefined.

1.6 Communication Progress

OpenSHMEM promises the progression of communication both with and without OpenSHMEM calls and requires the software progress mechanism in the implementation (e.g., a progress thread) when the hardware does not provide asynchronous communication capabilities. In MPI, however, a weak progress semantics is applied. That is, an MPI communication call is guaranteed only to complete in finite time. For instance, an *MPI_Put* may be completed only when the remote process makes an MPI call that internally triggers the progress of MPI, if the underlying hardware does not support asynchronous communication. A hybrid program should not assume that the OpenSHMEM library also makes progress for MPI. It can explicitly manage the asynchronous communication of MPI in order to prevent any deadlock or performance degradation.

Annex E

History of OpenSHMEM

SHMEM has a long history as a parallel-programming model and has been extensively used on a number of products since 1993, including the Cray T3D, Cray X1E, Cray XT3 and XT4, Silicon Graphics International (SGI) Origin, SGI Altix, Quadrics-based clusters, and InfiniBand-based clusters.

- SHMEM Timeline
  - Cray SHMEM
    - SHMEM first introduced by Cray Research, Inc. in 1993 for Cray T3D
    - Cray was acquired by SGI in 1996
    - Cray was acquired by Tera in 2000 (MTA)
    - Platforms: Cray T3D, T3E, C90, J90, SV1, SV2, X1, X2, XE, XMT, XT
  - SGI SHMEM
    - SGI acquired Cray Research, Inc. and SHMEM was integrated into SGI's Message Passing Toolkit (MPT)
    - SGI currently owns the rights to SHMEM and OpenSHMEM
    - Platforms: Origin, Altix 4700, Altix XE, ICE, UV
    - SGI was acquired by Rackable Systems in 2009
    - SGI and OSSS signed a SHMEM trademark licensing agreement in 2010
    - HPE acquired SGI in 2016

A listing of OpenSHMEM implementations can be found on http://www.openshmem.org/.

Annex F

OpenSHMEM Specification and Deprecated API

1 Overview

For the OpenSHMEM Specification, deprecation is the process of identifying API that is supported but no longer recommended for use by users. The deprecated API **must** be supported until clearly indicated as otherwise by the Specification. This chapter records the API or functionality that have been deprecated, the version of the OpenSHMEM Specification that effected the deprecation, and the most recent version of the OpenSHMEM Specification in which the feature was supported before removal.

Deprecated API	Deprecated Since	Last Version Supported	Replaced By
Header Directory: *mpp*	1.1	Current	(none)
C/C++: *start_pes*	1.2	Current	*shmem_init*
Fortran: *START_PES*	1.2	Current	*SHMEM_INIT*
Implicit finalization	1.2	Current	*shmem_finalize*
C/C++: *_my_pe*	1.2	Current	*shmem_my_pe*
C/C++: *_num_pes*	1.2	Current	*shmem_n_pes*
Fortran: *MY_PE*	1.2	Current	*SHMEM_MY_PE*
Fortran: *NUM_PES*	1.2	Current	*SHMEM_N_PES*
C/C++: *shmalloc*	1.2	Current	*shmem_malloc*
C/C++: *shfree*	1.2	Current	*shmem_free*
C/C++: *shrealloc*	1.2	Current	*shmem_realloc*
C/C++: *shmemalign*	1.2	Current	*shmem_align*
Fortran: *SHMEM_PUT*	1.2	Current	*SHMEM_PUT8* or *SHMEM_PUT64*
C/C++: *shmem_clear_cache_inv* *Fortran*: *SHMEM_CLEAR_CACHE_INV*	1.3	Current	(none)
C/C++: *shmem_clear_cache_line_inv*	1.3	Current	(none)
C/C++: *shmem_set_cache_inv* *Fortran*: *SHMEM_SET_CACHE_INV*	1.3	Current	(none)
C/C++: *shmem_set_cache_line_inv* *Fortran*: *SHMEM_SET_CACHE_LINE_INV*	1.3	Current	(none)
C/C++: *shmem_udcflush* *Fortran*: *SHMEM_UDCFLUSH*	1.3	Current	(none)
C/C++: *shmem_udcflush_line* *Fortran*: *SHMEM_UDCFLUSH_LINE*	1.3	Current	(none)
_SHMEM_SYNC_VALUE	1.3	Current	*SHMEM_SYNC_VALUE*
_SHMEM_BARRIER_SYNC_SIZE	1.3	Current	*SHMEM_BARRIER_SYNC_SIZE*
_SHMEM_BCAST_SYNC_SIZE	1.3	Current	*SHMEM_BCAST_SYNC_SIZE*
_SHMEM_COLLECT_SYNC_SIZE	1.3	Current	*SHMEM_COLLECT_SYNC_SIZE*
_SHMEM_REDUCE_SYNC_SIZE	1.3	Current	*SHMEM_REDUCE_SYNC_SIZE*
_SHMEM_REDUCE_MIN_WRKDATA_SIZE	1.3	Current	*SHMEM_REDUCE_MIN_WRKDATA_SIZE*
_SHMEM_MAJOR_VERSION	1.3	Current	*SHMEM_MAJOR_VERSION*
_SHMEM_MINOR_VERSION	1.3	Current	*SHMEM_MINOR_VERSION*
_SHMEM_MAX_NAME_LEN	1.3	Current	*SHMEM_MAX_NAME_LEN*
_SHMEM_VENDOR_STRING	1.3	Current	*SHMEM_VENDOR_STRING*
_SHMEM_CMP_EQ	1.3	Current	*SHMEM_CMP_EQ*
_SHMEM_CMP_NE	1.3	Current	*SHMEM_CMP_NE*
_SHMEM_CMP_LT	1.3	Current	*SHMEM_CMP_LT*
_SHMEM_CMP_LE	1.3	Current	*SHMEM_CMP_LE*

Deprecated API	Deprecated Since	Last Version Supported	Replaced By
_SHMEM_CMP_GT	1.3	Current	*SHMEM_CMP_GT*
_SHMEM_CMP_GE	1.3	Current	*SHMEM_CMP_GE*
SMA_VERSION	1.4	Current	*SHMEM_VERSION*
SMA_INFO	1.4	Current	*SHMEM_INFO*
SMA_SYMMETRIC_SIZE	1.4	Current	*SHMEM_SYMMETRIC_SIZE*
SMA_DEBUG	1.4	Current	*SHMEM_DEBUG*
C/C++: *shmem_wait* *C/C++*: *shmem_<**TYPENAME**>_wait*	1.4	Current	See **Notes** for *shmem_wait_until*
C/C++: *shmem_wait_until*	1.4	Current	*C11*: *shmem_wait_until*, *C/C++*: *shmem_long_wait_until*
C11: *shmem_fetch* *C/C++*: *shmem_<**TYPENAME**>_fetch*	1.4	Current	*shmem_atomic_fetch*
C11: *shmem_set* *C/C++*: *shmem_<**TYPENAME**>_set*	1.4	Current	*shmem_atomic_set*
C11: *shmem_cswap* *C/C++*: *shmem_<**TYPENAME**>_cswap*	1.4	Current	*shmem_atomic_compare_swap*
C11: *shmem_swap* *C/C++*: *shmem_<**TYPENAME**>_swap*	1.4	Current	*shmem_atomic_swap*
C11: *shmem_finc* *C/C++*: *shmem_<**TYPENAME**>_finc*	1.4	Current	*shmem_atomic_fetch_inc*
C11: *shmem_inc* *C/C++*: *shmem_<**TYPENAME**>_inc*	1.4	Current	*shmem_atomic_inc*
C11: *shmem_fadd* *C/C++*: *shmem_<**TYPENAME**>_fadd*	1.4	Current	*shmem_atomic_fetch_add*
C11: *shmem_add* *C/C++*: *shmem_<**TYPENAME**>_add*	1.4	Current	*shmem_atomic_add*
Entire *Fortran* API	1.4	Current	(none)
C/C++: *shmem_barrier*	1.5	Current	*shmem_quiet*; *shmem_sync*
C/C++: Active set based *shmem_sync*	1.5	Current	Team based *shmem_sync*
C/C++: *shmem_broadcast[32,64]*	1.5	Current	*shmem_broadcast*
C/C++: *shmem_collect[32,64]*	1.5	Current	*shmem_collect*
C/C++: *shmem_fcollect[32,64]*	1.5	Current	*shmem_fcollect*
C/C++: *shmem*_TYPENAME_*OP_to_all*	1.5	Current	*shmem*_TYPENAME_*OP_reduce*
C/C++: *shmem_alltoall[32,64]*	1.5	Current	*shmem_alltoall*
C/C++: *shmem_alltoalls[32,64]*	1.5	Current	*shmem_alltoalls*

2 Deprecation Rationale

2.1 Header Directory: *mpp*

In addition to the default system header paths, OpenSHMEM implementations must provide all OpenSHMEM-specified header files from the *mpp* header directory such that these headers can be referenced in *C/C++* as

```
#include <mpp/shmem.h>
#include <mpp/shmemx.h>
```

and in *Fortran* as

```
include 'mpp/shmem.fh'
include 'mpp/shmemx.fh'
```

for backwards compatibility with SGI SHMEM.

2.2 *C/C++*: *start_pes*

The *C/C++* routine *start_pes* includes an unnecessary initialization argument that is remnant of historical *SHMEM* implementations and no longer reflects the requirements of modern OpenSHMEM implementations. Furthermore, the naming of *start_pes* does not include the standardized *shmem_* naming prefix. This routine has been deprecated and OpenSHMEM users are encouraged to use *shmem_init* instead.

2.3 Implicit Finalization

Implicit finalization was deprecated and replaced with explicit finalization using the *shmem_finalize* routine. Explicit finalization improves portability and also improves interoperability with profiling and debugging tools.

2.4 *C/C++*: *_my_pe*, *_num_pes*, *shmalloc*, *shfree*, *shrealloc*, *shmemalign*

The *C/C++* routines *_my_pe*, *_num_pes*, *shmalloc*, *shfree*, *shrealloc*, and *shmemalign* were deprecated in order to normalize the OpenSHMEM *Application Programming Interface* (API) to use *shmem_* as the standard prefix for all routines.

2.5 *Fortran*: *START_PES*, *MY_PE*, *NUM_PES*

The *Fortran* routines *START_PES*, *MY_PE*, and *NUM_PES* were deprecated in order to minimize the API differences from the deprecation of *C/C++* routines *start_pes*, *_my_pe*, and *_num_pes*.

2.6 *Fortran*: *SHMEM_PUT*

The *Fortran* routine *SHMEM_PUT* is defined only for the *Fortran* API and is semantically identical to *Fortran* routines *SHMEM_PUT8* and *SHMEM_PUT64*. Since *SHMEM_PUT8* and *SHMEM_PUT64* have defined equivalents in the *C/C++* interface, *SHMEM_PUT* is ambiguous and has been deprecated.

2.7 SHMEM_CACHE

The *SHMEM_CACHE* API

C/C++:	*Fortran*:
shmem_clear_cache_inv	*SHMEM_CLEAR_CACHE_INV*
shmem_set_cache_inv	*SHMEM_SET_CACHE_INV*
shmem_set_cache_line_inv	*SHMEM_SET_CACHE_LINE_INV*
shmem_udcflush	*SHMEM_UDCFLUSH*
shmem_udcflush_line	*SHMEM_UDCFLUSH_LINE*
shmem_clear_cache_line_inv	

was originally implemented for systems with cache-management instructions. This API has largely gone unused on cache-coherent system architectures. *SHMEM_CACHE* has been deprecated.

2.8 *_SHMEM_** Library Constants

The library constants

_SHMEM_SYNC_VALUE	*_SHMEM_MAX_NAME_LEN*
_SHMEM_BARRIER_SYNC_SIZE	*_SHMEM_VENDOR_STRING*
_SHMEM_BCAST_SYNC_SIZE	*_SHMEM_CMP_EQ*
_SHMEM_COLLECT_SYNC_SIZE	*_SHMEM_CMP_NE*
_SHMEM_REDUCE_SYNC_SIZE	*_SHMEM_CMP_LT*
_SHMEM_REDUCE_MIN_WRKDATA_SIZE	*_SHMEM_CMP_LE*
_SHMEM_MAJOR_VERSION	*_SHMEM_CMP_GT*
_SHMEM_MINOR_VERSION	*_SHMEM_CMP_GE*

do not adhere to the *C* standard's reserved identifiers and the *C++* standard's reserved names. These constants were deprecated and replaced with corresponding constants of prefix *SHMEM_* that adhere to *C/C++* and *Fortran* naming conventions.

2.9 *SMA_** Environment Variables

The environment variables *SMA_VERSION*, *SMA_INFO*, *SMA_SYMMETRIC_SIZE*, and *SMA_DEBUG* were deprecated in order to normalize the OpenSHMEM API to use *SHMEM_* as the standard prefix for all environment variables.

2.10 *C/C++*: *shmem_wait*

The *C/C++* interface for *shmem_wait* and *shmem_<**TYPENAME**>_wait* was identified as unintuitive with respect to the comparison operation it performed. As *shmem_wait* can be trivially replaced by *shmem_wait_until* where *cmp* is *SHMEM_CMP_NE*, the *shmem_wait* interface was deprecated in favor of *shmem_wait_until*, which makes the comparison operation explicit and better communicates the developer's intent.

2.11 *C/C++*: *shmem_wait_until*

The *long*-typed *C/C++* routine *shmem_wait_until* was deprecated in favor of the *C11* type-generic interface of the same name or the explicitly typed *C/C++* routine *shmem_long_wait_until*.

2.12 *C11* and *C/C++*: *shmem_fetch*, *shmem_set*, *shmem_cswap*, *shmem_swap*, *shmem_finc*, *shmem_inc*, *shmem_fadd*, *shmem_add*

The *C11* and *C/C++* interfaces for

C11:	*C/C++*:
shmem_fetch	*shmem_<**TYPENAME**>_fetch*
shmem_set	*shmem_<**TYPENAME**>_set*
shmem_cswap	*shmem_<**TYPENAME**>_cswap*
shmem_swap	*shmem_<**TYPENAME**>_swap*
shmem_finc	*shmem_<**TYPENAME**>_finc*
shmem_inc	*shmem_<**TYPENAME**>_inc*
shmem_fadd	*shmem_<**TYPENAME**>_fadd*
shmem_add	*shmem_<**TYPENAME**>_add*

were deprecated and replaced with similarly named interfaces within the *shmem_atomic_** namespace in order to more clearly identify these calls as performing atomic operations. In addition, the abbreviated names "cswap", "finc", and "fadd" were expanded for clarity to "compare_swap", "fetch_inc", and "fetch_add".

2.13 *Fortran* API

The entire OpenSHMEM *Fortran* API was deprecated in OpenSHMEM 1.4 and removed in OpenSHMEM 1.5 because of a general lack of use and a lack of conformance with legacy *Fortran* standards. In lieu of an extensive update of the *Fortran* API, *Fortran* users are encouraged to leverage the OpenSHMEM Specification's *C* API through the *Fortran–C* interoperability initially standardized by *Fortran 2003*[1].

2.14 Active-set-based collective routines

With the addition of OpenSHMEM teams, the previous methods for performing collective operations has been superseded by a more readable, flexible method for organizing and communicating between groups of PEs. All collective routines which previously indicated subgroups of PEs with a list of parameters to describe the subgroup composition should be phased out in favor of using collective operations with a team parameter.

When moving from active set routines to teams based routines, the fixed-size versions of the routines, e.g. *shmem_broadcast32*, were not carried forward. Instead, all teams based collective routines use standard *C* types with the option to use generic *C11* functions for more portable and maintainable implementations.

2.15 *C/C++*: *shmem_barrier*

Each OpenSHMEM team might be associated with some number of communication contexts. The *shmem_barrier* functions imply that the default context is quiesced after synchronizing some set of PEs. Since teams may have some

[1] Formally, *Fortran 2003* is known as ISO/IEC 1539-1:2004(E).

number of contexts associated with the team, it becomes less clear which context would be the "default" context for that particular team. Rather than continue to support *shmem_barrier* for active-sets or teams, programs should use a call to *shmem_quiet* followed by a call to *shmem_sync* in order to explicitly indicate which context to quiesce.

Annex G

Changes to this Document

1 Version 1.5

Major changes in OpenSHMEM 1.5 include ...

The following list describes the specific changes in OpenSHMEM 1.5:

- Added support for nonblocking *Atomic Memory Operation* (AMO) functions.
  See Section **??**.
- Added support for blocking *put-with-signal* functions.
  See Section **??**.
- Added support for nonblocking *put-with-signal* functions.
  See Section **??**.
- Clarified that point-to-point synchronization routines preserve the atomicity of OpenSHMEM AMOs.
  See Section **??**.
- Clarified that symmetric variables used as *ivar* arguments to point-to-point synchronization routines must be updated using OpenSHMEM AMOs.
  See Section **??**.
- Removed the entire OpenSHMEM *Fortran* API.
- Added support for multipliers in *SHMEM_SYMMETRIC_SIZE* environment variables.
  See Section **??**.
- Added support for a multiple-element point-to-point synchronization API with the functions: *shmem_wait_until_all*, *shmem_wait_until_any*, *shmem_wait_until_some*, *shmem_test_all*, *shmem_test_any*, and *shmem_test_some*.
  See Sections **??**, **??**, **??**, **??**, **??**, and **??**.
- Added support for vectorized comparison values in the multiple-element point-to-point synchronization API with the functions: *shmem_wait_until_all_vector*, *shmem_wait_until_any_vector*, *shmem_wait_until_some_vector*, *shmem_test_all_vector*, *shmem_test_any_vector*, and *shmem_test_some_vector*.
  See Sections **??**, **??**, **??**, **??**, **??**, and **??**.
- Added OpenSHMEM profiling interface.
  See Section **??**.
- Specified the validity of communication contexts, added the constant *SHMEM_CTX_INVALID*, and clarified the behavior of *shmem_ctx_** routines on invalid contexts.
  See Section **??**.

- Clarified PE active set requirements.
  See Section **??**.

- Clarified that when the *size* argument is zero, symmetric heap allocation routines perform no action and return a null pointer; that symmetric heap management routines that perform no action do not perform a barrier; and that the *alignment* argument to *shmem_align* must be power of two multiple of *sizeof(void*)*.
  See Section **??**.

- Clarified that the OpenSHMEM lock API provides a non-reentrant mutex and that *shmem_clear_lock* performs a quiet operation on the default context.
  See Section **??**

- Clarified the atomicity guarantees of the OpenSHMEM memory model.
  See Section **??**.

2 Version 1.4

Major changes in OpenSHMEM 1.4 include multithreading support, *contexts* for communication management, *shmem_sync*, *shmem_calloc*, expanded type support, a new namespace for atomic operations, atomic bitwise operations, *shmem_test* for nonblocking point-to-point synchronization, and *C11* type-generic interfaces for point-to-point synchronization.

The following list describes the specific changes in OpenSHMEM 1.4:

- New communication management API, including *shmem_ctx_create*; *shmem_ctx_destroy*; and additional RMA, AMO, and memory ordering routines that accept *shmem_ctx_t* arguments.
  See Section **??**.

- New API *shmem_sync_all* and *shmem_sync* to provide PE synchronization without completing pending communication operations.
  See Sections **??** and **??**.

- Clarified that the OpenSHMEM extensions header files are required, even when empty.
  See Section **??**.

- Clarified that the *SHMEM_GET64* and *SHMEM_GET64_NBI* routines are included in the *Fortran* language bindings.
  See Sections **??** and **??**.

- Clarified that *shmem_init* must be matched with a call to *shmem_finalize*.
  See Sections **??** and **??**.

- Added the *SHMEM_SYNC_SIZE* constant.
  See Section **??**.

- Added type-generic interfaces for *shmem_wait_until*.
  See Section **??**.

- Removed the *volatile* qualifiers from the *ivar* arguments to *shmem_wait* routines and the *lock* arguments in the lock API. *Rationale: Volatile qualifiers were added to several API routines in OpenSHMEM 1.3; however, they were later found to be unnecessary.*
  See Sections **??** and **??**.

- Deprecated the *SMA_** environment variables and added equivalent *SHMEM_** environment variables.
  See Section **??**.

- Added the *C11 _Noreturn* function specifier to *shmem_global_exit*.
  See Section **??**.

- Clarified ordering semantics of memory ordering, point-to-point synchronization, and collective synchronization routines.
- Clarified deprecation overview and added deprecation rationale in Annex F.
  See Section F.
- Deprecated header directory *mpp*.
  See Section F.
- Deprecated the *shmem_wait* functions and the *long*-typed *C/C++* *shmem_wait_until* function.
  See Section **??**.
- Added the *shmem_test* functions.
  See Section **??**.
- Added the *shmem_calloc* function.
  See Section **??**.
- Introduced the thread safe semantics that define the interaction between OpenSHMEM routines and user threads.
  See Section **??**.
- Added the new routine *shmem_init_thread* to initialize the OpenSHMEM library with one of the defined thread levels.
  See Section **??**.
- Added the new routine *shmem_query_thread* to query the thread level provided by the OpenSHMEM implementation.
  See Section **??**.
- Clarified the semantics of *shmem_quiet* for a multithreaded OpenSHMEM PE.
  See Section **??**
- Revised the description of *shmem_barrier_all* for a multithreaded OpenSHMEM PE.
  See Section **??**
- Revised the description of *shmem_wait* for a multithreaded OpenSHMEM PE.
  See Section **??**
- Clarified description for *SHMEM_VENDOR_STRING*.
  See Section **??**.
- Clarified description for *SHMEM_MAX_NAME_LEN*.
  See Section **??**.
- Clarified API description for *shmem_info_get_name*.
  See Section **??**.
- Expanded the type support for RMA, AMO, and point-to-point synchronization operations.
  See Tables **??**, **??**, **??**, and **??**
- Renamed AMO operations to use *shmem_atomic_** prefix and deprecated old AMO routines.
  See Section **??**.
- Added fetching and non-fetching bitwise AND, OR, and XOR atomic operations.
  See Section **??**.
- Deprecated the entire *Fortran* API.
- Replaced the *complex* macro in complex-typed reductions with the *C99* (and later) type specifier *_Complex* to remove an implicit dependence on *complex.h*.
  See Section **??**.
- Clarified that complex-typed reductions in C are optionally supported.
  See Section **??**.

3 Version 1.3

Major changes in OpenSHMEM 1.3 include the addition of nonblocking RMA operations, atomic *Put* and *Get* operations, all-to-all collectives, and *C11* type-generic interfaces for RMA and AMO operations.

The following list describes the specific changes in OpenSHMEM 1.3:

- Clarified implementation of PEs as threads.
- Added *const* to every read-only pointer argument.
- Clarified definition of *Fence*.
  See Section **??**.
- Clarified implementation of symmetric memory allocation.
  See Section **??**.
- Restricted atomic operation guarantees to other atomic operations with the same datatype.
  See Section **??**.
- Deprecation of all constants that start with *_SHMEM_*.
  See Section **??**.
- Added a type-generic interface to OpenSHMEM RMA and AMO operations based on *C11* Generics.
  See Sections **??**, **??** and **??**.
- New nonblocking variants of remote memory access, *SHMEM_PUT_NBI* and *SHMEM_GET_NBI*.
  See Sections **??** and **??**.
- New atomic elemental read and write operations, *SHMEM_FETCH* and *SHMEM_SET*.
  See Sections **??** and **??**
- New alltoall data exchange operations, *SHMEM_ALLTOALL* and *SHMEM_ALLTOALLS*.
  See Sections **??** and **??**.
- Added *volatile* to remotely accessible pointer argument in *SHMEM_WAIT* and *SHMEM_LOCK*.
  See Sections **??** and **??**.
- Deprecation of *SHMEM_CACHE*.
  See Section **??**.

4 Version 1.2

Major changes in OpenSHMEM 1.2 include a new initialization routine (*shmem_init*), improvements to the execution model with an explicit library-finalization routine (*shmem_finalize*), an early-exit routine (*shmem_global_exit*), namespace standardization, and clarifications to several API descriptions.

The following list describes the specific changes in OpenSHMEM 1.2:

- Added specification of *pSync* initialization for all routines that use it.
- Replaced all placeholder variable names *target* with *dest* to avoid confusion with *Fortran*'s *target* keyword.
- New Execution Model for exiting/finishing OpenSHMEM programs.
  See Section **??**.
- New library constants to support API that query version and name information.
  See Section **??**.

- New API *shmem_init* to provide mechanism to start an OpenSHMEM program and replace deprecated *start_pes*.
  See Section **??**.
- Deprecation of *_my_pe* and *_num_pes* routines.
  See Sections **??** and **??**.
- New API *shmem_finalize* to provide collective mechanism to cleanly exit an OpenSHMEM program and release resources.
  See Section **??**.
- New API *shmem_global_exit* to provide mechanism to exit an OpenSHMEM program.
  See Section **??**.
- Clarification related to the address of the referenced object in *shmem_ptr*.
  See Section **??**.
- New API to query the version and name information.
  See Section **??** and **??**.
- OpenSHMEM library API normalization. All *C* symmetric memory management API begins with *shmem_*.
  See Section **??**.
- Notes and clarifications added to *shmem_malloc*.
  See Section **??**.
- Deprecation of *Fortran* API routine *SHMEM_PUT*.
  See Section **??**.
- Clarification related to *shmem_wait*.
  See Section **??**.
- Undefined behavior for null pointers without zero counts added.
  See Annex C
- Addition of new Annex for clearly specifying deprecated API and its support across versions of the OpenSHMEM Specification.
  See Annex F.

5 Version 1.1

Major changes from OpenSHMEM 1.0 to OpenSHMEM 1.1 include the introduction of the *shmemx.h* header file for non-standard API extensions, clarifications to completion semantics and API descriptions in agreement with the SGI SHMEM specification, and general readabilty and usability improvements to the document structure.

The following list describes the specific changes in OpenSHMEM 1.1:

- Clarifications of the completion semantics of memory synchronization interfaces.
  See Section **??**.
- Clarification of the completion semantics of memory load and store operations in context of *shmem_barrier_all* and *shmem_barrier* routines.
  See Section **??** and **??**.
- Clarification of the completion and ordering semantics of *shmem_quiet* and *shmem_fence*.
  See Section **??** and **??**.
- Clarifications of the completion semantics of RMA and AMO routines.
  See Sections **??** and **??**

- Clarifications of the memory model and the memory alignment requirements for symmetric data objects.
  See Section **??**.
- Clarification of the execution model and the definition of a PE.
  See Section **??**
- Clarifications of the semantics of *shmem_pe_accessible* and *shmem_addr_accessible*.
  See Section **??** and **??**.
- Added an annex on interoperability with MPI.
  See Annex D.
- Added examples to the different interfaces.
- Clarification of the naming conventions for constant in *C* and *Fortran*.
  See Section **??** and **??**.
- Added API calls: *shmem_char_p*, *shmem_char_g*.
  See Sections **??** and **??**.
- Removed API calls: *shmem_char_put*, *shmem_char_get*.
  See Sections **??** and **??**.
- The usage of *ptrdiff_t*, *size_t*, and *int* in the interface signature was made consistent with the description.
  See Sections **??**, **??**, and **??**.
- Revised *shmem_barrier* example.
  See Section **??**.
- Clarification of the initial value of *pSync* work arrays for *shmem_barrier*.
  See Section **??**.
- Clarification of the expected behavior when multiple *start_pes* calls are encountered.
  See Section **??**.
- Corrected the definition of atomic increment operation.
  See Section **??**.
- Clarification of the size of the symmetric heap and when it is set.
  See Section **??**.
- Clarification of the integer and real sizes for *Fortran* API.
  See Sections **??**, **??**, **??**, **??**, **??**, and **??**.
- Clarification of the expected behavior on program *exit*.
  See Section **??**, Execution Model.
- More detailed description for the progress of OpenSHMEM operations provided.
  See Section **??**.
- Clarification of naming convention for non-standard interfaces and their inclusion in *shmemx.h*.
  See Section **??**.
- Various fixes to OpenSHMEM code examples across the Specification to include appropriate header files.
- Removing requirement that implementations should detect size mismatch and return error information for *shmalloc* and ensuring consistent language.
  See Sections **??** and Annex C.
- *Fortran* programming fixes for examples.
  See Sections **??** and **??**.

- Clarifications of the reuse *pSync* and *pWork* across collectives.
  See Sections **??**, **??**, **??** and **??**.
- Name changes for UV and ICE for SGI systems.
  See Annex E.

Index