OpenSHMEM Application Programming Interface



http://www.openshmem.org/

Version 1.5

17th August 2018

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
- 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

10		
11	1	The OpenSHMEM Effort
12	2	Programming Model Overview
12	3	Memory Model
13		3.1 Atomicity Guarantees
14	4	Execution Model
15		4.1 Progress of OpenSHMEM Operations
16	5	Language Bindings and Conformance
17	6	Library Constants
18	7	Library Handles
19	8	Environment Variables
20	9	OpenSHMEM Library API
20		9.1 Library Setup, Exit, and Query Routines
21		9.1.1 SHMEM_INIT
22		9.1.2 SHMEM_MY_PE 14
23		9.1.3 SHMEM_N_PES
24		9.1.4 SHMEM_FINALIZE 16
25		9.1.5 SHMEM_GLOBAL_EXIT
26		9.1.6 SHMEM_PE_ACCESSIBLE
27		9.1.7 SHMEM_ADDR_ACCESSIBLE
28		9.1.8 SHMEM_PTR
20		9.1.9 SHMEM_INFO_GET_VERSION
20		9.1.10 SHMEM_INFO_GET_NAME
30		9.1.11 START_PES 23
31		9.2 Thread Support
32		9.2.1 SHMEM_INIT_THREAD 25
33		9.2.2 SHMEM_QUERY_THREAD
34		9.3 Memory Management Routines 26
35		9.3.1 SHMEM_MALLOC, SHMEM_FREE, SHMEM_REALLOC, SHMEM_ALIGN 27
36		9.3.2 SHMEM_CALLOC 28
37		9.3.3 SHPALLOC 29
38		9.3.4 SHPCLMOVE
20		9.3.5 SHPDEALLC 31
39		9.4 Team Management Routines 32
40		9.4.1 SHMEM_TEAM_MY_PE 32
41		9.4.2 SHMEM_TEAM_N_PES
42		9.4.3 SHMEM_TEAM_CONFIG_T
43		9.4.4 SHMEM_TEAM_GET_CONFIG
44		9.4.5 SHMEM_TEAM_TRANSLATE
45		9.4.6 SHMEM_TEAM_SPLIT_STRIDED
46		9.4.7 SHMEM_TEAM_SPLIT_2D 37
47		9.4.8 SHMEM_TEAM_DESTROY
40		9.5 Communication Management Routines 40
48		9.5.1 SHMEM_CTX_CREATE 40

	9.5.2 SHMEM_TEAM_CREATE_CTX	41
	9.5.3 SHMEM CTX DESTROY	42
	9.5.4 SHMEM CTX GET TEAM	45
9.6	Remote Memory Access Routines	45
	961 SHMEM PUT	46
	962 SHMEM P	48
	963 SHMFM IPIT	50
	064 SHMEM CET	52
	0.65 SHMEM C	54
	0.66 SHMEM ICET	55
07	Non blocking Remote Memory Access Routines	57
9.1	0.7.1 SHMEM DUT NRI	57
	0.7.2 SHMEM_CET_NBI	50
0.8	9.7.2 SHMEM_GET_NDI Atomic Memory Operations	59
9.0		61
	9.0.1 SHMEM_ATOMIC_FETCH	62
	9.0.2 SHMEM_ATOMIC_SET	65
	9.6.5 SHMEM_ATOMIC_COMPARE_SWAP	600
	9.8.4 SHMEM_ATOMIC_SWAP	60
	9.8.5 SHMEM_ATOMIC_FETCH_INC	08
	9.8.6 SHMEM_ATOMIC_INC	/0
	9.8.7 SHMEM_ATOMIC_FETCH_ADD	/1
	9.8.8 SHMEM_ATOMIC_ADD	13
	9.8.9 SHMEM_ATOMIC_FETCH_AND	15
	9.8.10 SHMEM_ATOMIC_AND	76
	9.8.11 SHMEM_ATOMIC_FETCH_OR	76
	9.8.12 SHMEM_ATOMIC_OR	77
	9.8.13 SHMEM_ATOMIC_FETCH_XOR	78
	9.8.14 SHMEM_ATOMIC_XOR	79
9.9	Collective Routines	79
	9.9.1 SHMEM_BARRIER_ALL	80
	9.9.2 SHMEM_BARRIER	81
	9.9.3 SHMEM_SYNC_ALL	83
	9.9.4 SHMEM_SYNC	84
	9.9.5 SHMEM_TEAM_BROADCAST	86
	9.9.6 SHMEM_BROADCAST	87
	9.9.7 SHMEM_COLLECT, SHMEM_FCOLLECT	89
	9.9.8 SHMEM_TEAM_COLLECT, SHMEM_TEAM_FCOLLECT	92
	9.9.9 SHMEM_REDUCTIONS	93
	9.9.9.1 AND	94
	9.9.9.2 MAX	94
	9.9.9.3 MIN	94
	9.9.9.4 SUM	95
	9.9.9.5 PROD	96
	9.9.9.6 OR	97
	9.9.9.7 XOR	97
	9.9.10 SHMEM_ALLTOALL 1	101
. ·	9.9.11 SHMEM_ALLTOALLS 1	104
9.10	Point-To-Point Synchronization Routines	107
	9.10.1 SHMEM_WAIT_UNTIL	107
	9.10.2 SHMEM_TEST 1	10
9.11	Memory Ordering Routines	11
	9.11.1 SHMEM_FENCE 1	11
	9.11.2 SHMEM_QUIET 1	13
	9.11.3 Synchronization and Communication Ordering in OpenSHMEM 1	14

v

	9.12 Distributed Locking Routines	118 118
	9.13 Cache Management	119
A	Writing OpenSHMEM Programs	121
B	Compiling and Running Programs	124
	1 Compilation 2 Running Programs	124 124
С	Undefined Behavior in OpenSHMEM	126
D	Interoperability with other Programming Models 1 Message Passing Interface (MPI) Interoperability	127 127
E	History of OpenSHMEM	128
F	OpenSHMEM Specification and Deprecated API 1 Overview 2 Deprecation Rationale 2.1 Header Directory: mpp 2.2 C/C++: start_pes 2.3 Implicit Finalization 2.4 C/C++: _my_pe, _num_pes, shmalloc, shfree, shrealloc, shmemalign 2.5 Fortran: START_PES, MY_PE, NUM_PES 2.6 Fortran: SHMEM_PUT 2.7 SHMEM_CACHE 2.8 _SHMEM_* Library Constants 2.9 SMA_* Environment Variables 2.10 C/C++: shmem_wait 2.11 C/C++: shmem_wait_until 2.12 CI1 and C/C++: shmem_fetch, shmem_set, shmem_cswap, shmem_swap, shmem_finc, shmem_shmem_fadd, shmem_add 2.13 Fortran API	129 129 130 130 130 130 131 131 131 131
G	Changes to this Document 1 Version 1.5 2 Version 1.4 3 Version 1.3 4 Version 1.2 5 Version 1.1	133 133 133 135 136 137
In	dex	139

1 The OpenSHMEM Effort

OpenSHMEM is a *Partitioned Global Address Space* (PGAS) library interface specification. OpenSHMEM aims to provide a standard *Application Programming Interface* (API) for SHMEM libraries to aid portability and facilitate uniform predictable results of OpenSHMEM programs by explicitly stating the behavior and semantics of the Open-SHMEM library calls. Through the different versions, OpenSHMEM will continue to address the requirements of the PGAS community. As of this specification, many existing vendors support OpenSHMEM-compliant implementations and new vendors are developing OpenSHMEM library implementations to help the users write portable OpenSHMEM code. This ensures that programs can run on multiple platforms without having to deal with subtle vendor-specific implementation differences. For more details on the history of OpenSHMEM please refer to the History of OpenSHMEM section.

The OpenSHMEM¹ effort is driven by the DoD with continuous input from the OpenSHMEM community. To see all of the contributors and participants for the OpenSHMEM API, please see: http://www.openshmem.org/site/Contributors. In addition to the specification, the effort includes a reference OpenSHMEM implementation, validation and verification suites, tools, a mailing list and website infrastructure to support specification activities. For more information please refer to: http://www.openshmem.org/.

2 Programming Model Overview

OpenSHMEM implements PGAS by defining remotely accessible data objects as mechanisms to share information among OpenSHMEM processes or *Processing Elements* (PEs), and private data objects that are accessible by only the PE itself. The API allows communication and synchronization operations on both private (local to the PE initiating the operation) and remotely accessible data objects. The key feature of OpenSHMEM is that data transfer operations are *one-sided* in nature. This means that a local PE executing a data transfer routine does not require the participation of the remote PE to complete the routine. This allows for overlap between communication and computation to hide data transfer latencies, which makes OpenSHMEM ideal for unstructured, small/medium size data communication patterns. The OpenSHMEM library routines have the potential to provide a low-latency, high-bandwidth communication API for use in highly parallelized scalable programs.

The OpenSHMEM interfaces can be used to implement *Single Program Multiple Data* (SPMD) style programs. It provides interfaces to start the OpenSHMEM PEs in parallel and communication and synchronization interfaces to access remotely accessible data objects across PEs. These interfaces can be leveraged to divide a problem into multiple sub-problems that can be solved independently or with coordination using the communication and synchronization interfaces. The OpenSHMEM specification defines library calls, constants, variables, and language bindings for *C* and *Fortran*². The *C*++ interface is currently the same as that for *C*. Unlike Unified Parallel C, *Fortran 2008*, Titanium, X10, and Chapel, which are all PGAS languages, OpenSHMEM relies on the user to use the library calls to implement the correct semantics of its programming model.

An overview of the OpenSHMEM routines is described below:

1. Library Setup and Query

- (a) *Initialization*: The OpenSHMEM library environment is initialized, where the PEs are either single or multithreaded.
- (b) Query: The local PE may get the number of PEs running the same program and its unique integer identifier.
- (c) *Accessibility*: The local PE can find out if a remote PE is executing the same binary, or if a particular symmetric data object can be accessed by a remote PE, or may obtain a pointer to a symmetric data object on the specified remote PE on shared memory systems.

2. Symmetric Data Object Management

(a) *Allocation*: All executing PEs must participate in the allocation of a symmetric data object with identical arguments.

¹The OpenSHMEM specification is owned by Open Source Software Solutions Inc., a non-profit organization, under an agreement with HPE. ²As of OpenSHMEM 1.4, the *Fortran* interface has been deprecated.

- (b) *Deallocation*: All executing PEs must participate in the deallocation of the same symmetric data object with identical arguments.
- (c) *Reallocation*: All executing PEs must participate in the reallocation of the same symmetric data object with identical arguments.

3. Communication Management

(a) *Contexts*: Contexts are containers for communication operations. Each context provides an environment where the operations performed on that context are ordered and completed independently of other operations performed by the application.

4. Remote Memory Access

- (a) *Put*: The local PE specifies the *source* data object (private or symmetric) that is copied to the symmetric data object on the remote PE.
- (b) *Get*: The local PE specifies the symmetric data object on the remote PE that is copied to a data object (private or symmetric) on the local PE.

5. Atomics

- (a) *Swap*: The PE initiating the swap gets the old value of a symmetric data object from a remote PE and copies a new value to that symmetric data object on the remote PE.
- (b) *Increment*: The PE initiating the increment adds 1 to the symmetric data object on the remote PE.
- (c) *Add*: The PE initiating the add specifies the value to be added to the symmetric data object on the remote PE.
- (d) *Bitwise Operations*: The PE initiating the bitwise operation specifies the operand value to the bitwise operation to be performed on the symmetric data object on the remote PE.
- (e) *Compare and Swap*: The PE initiating the swap gets the old value of the symmetric data object based on a value to be compared and copies a new value to the symmetric data object on the remote PE.
- (f) *Fetch and Increment*: The PE initiating the increment adds 1 to the symmetric data object on the remote PE and returns with the old value.
- (g) *Fetch and Add*: The PE initiating the add specifies the value to be added to the symmetric data object on the remote PE and returns with the old value.
- (h) *Fetch and Bitwise Operations*: The PE initiating the bitwise operation specifies the operand value to the bitwise operation to be performed on the symmetric data object on the remote PE and returns the old value.

6. Synchronization and Ordering

- (a) *Fence*: The PE calling fence ensures ordering of *Put*, AMO, and memory store operations to symmetric data objects with respect to a specific destination PE.
- (b) *Quiet*: The PE calling quiet ensures remote completion of remote access operations and stores to symmetric data objects.
- (c) *Barrier*: All or some PEs collectively synchronize and ensure completion of all remote and local updates prior to any PE returning from the call.

7. Collective Communication

- (a) *Broadcast*: The *root* PE specifies a symmetric data object to be copied to a symmetric data object on one or more remote PEs (not including itself).
- (b) *Collection*: All PEs participating in the routine get the result of concatenated symmetric objects contributed by each of the PEs in another symmetric data object.
- (c) *Reduction*: All PEs participating in the routine get the result of an associative binary routine over elements of the specified symmetric data object on another symmetric data object.

3. MEMORY MODEL

(d) *All-to-All*: All PEs participating in the routine exchange a fixed amount of contiguous or strided data with all other PEs in the active set.

8. Mutual Exclusion

- (a) Set Lock: The PE acquires exclusive access to the region bounded by the symmetric lock variable.
- (b) *Test Lock*: The PE tests the symmetric *lock* variable for availability.
- (c) Clear Lock: The PE which has previously acquired the lock releases it.

- deprecation start -

9. Data Cache Control

(a) Implementation of mechanisms to exploit the capabilities of hardware cache if available.

3 Memory Model

- 1	PE 0	PE 1		PE N-1
Symmetric	Global and Static Variables	Global and Static Variables		Global and Static Variables
cessible (ta Object	Variable: X	Variable: X	X = shmem_malloc(sizeof(long))	Variable: X
Remotely Ac Dat	Symmetric Heap	Symmetric Heap	000	Symmetric Heap
Private Data Objects	Local Variables	Local Variables		Local Variables

Figure 1: OpenSHMEM Memory Model

An OpenSHMEM program consists of data objects that are private to each PE and data objects that are remotely accessible by all PEs. Private data objects are stored in the local memory of each PE and can only be accessed by the PE itself; these data objects cannot be accessed by other PEs via OpenSHMEM routines. Private data objects follow the memory model of *C* or *Fortran*. Remotely accessible objects, however, can be accessed by remote PEs using OpenSHMEM routines. Remotely accessible data objects are called *Symmetric Data Objects*. Each symmetric data object has a corresponding object with the same name, type, and size on all PEs where that object is accessible via the OpenSHMEM API³. (For the definition of what is accessible, see the descriptions for *shmem_pe_accessible* and *shmem_addr_accessible* in sections 9.1.6 and 9.1.7.) Symmetric data objects accessed via typed and type-generic

deprecation end -

³For efficiency reasons, the same offset (from an arbitrary memory address) for symmetric data objects might be used on all PEs. Further discussion about symmetric heap layout and implementation efficiency can be found in section 9.3.1

OpenSHMEM interfaces are required to be naturally aligned based on their type requirements and underlying architecture. In OpenSHMEM the following kinds of data objects are symmetric:

• — deprecation start –

Fortran data objects in common blocks or with the *SAVE* attribute. These data objects must not be defined in a dynamic shared object (DSO).

- Global and static C and C++ variables. These data objects must not be defined in a DSO.
- — deprecation start ______ *Fortran* arrays allocated with *shpalloc*

- deprecation end —

deprecation end —

• C and C++ data allocated by OpenSHMEM memory management routines (Section 9.3)

OpenSHMEM dynamic memory allocation routines (*shpalloc* and *shmem_malloc*) allow collective allocation of *Symmetric Data Objects* on a special memory region called the *Symmetric Heap*. The Symmetric Heap is created during the execution of a program at a memory location determined by the implementation. The Symmetric Heap may reside in different memory regions on different PEs. Figure 1 shows how OpenSHMEM implements a PGAS model using remotely accessible symmetric objects and private data objects when executing an OpenSHMEM program. Symmetric data objects are stored on the symmetric heap or in the global/static memory section of each PE.

3.1 Atomicity Guarantees

OpenSHMEM contains a number of routines that operate on symmetric data atomically (Section 9.8). These routines guarantee that accesses by OpenSHMEM's atomic operations with the same datatype will be exclusive, but do not guarantee exclusivity in combination with other routines, either inside OpenSHMEM or outside.

For example: during the execution of an atomic remote integer increment operation on a symmetric variable X, no other OpenSHMEM atomic operation may access X. After the increment, X will have increased its value by I on the destination PE, at which point other atomic operations may then modify that X. However, access to the symmetric object X with non-atomic operations, such as one-sided *put* or *get* operations, will invalidate the atomicity guarantees.

4 Execution Model

An OpenSHMEM program consists of a set of OpenSHMEM processes called PEs that execute in an SPMD-like 35 model where each PE can take a different execution path. For example, a PE can be implemented using an OS process. 36 The PEs may be either single or multithreaded. The PEs progress asynchronously, and can communicate/synchro-37 nize via the OpenSHMEM interfaces. All PEs in an OpenSHMEM program should start by calling the initialization 38 routine *shmem_init*⁴ or *shmem_init_thread* before using any of the other OpenSHMEM library routines. An Open-39 SHMEM program concludes its use of the OpenSHMEM library when all PEs call shmem_finalize or any PE calls 40 shmem_global_exit. During a call to shmem_finalize, the OpenSHMEM library must complete all pending commu-41 nication and release all the resources associated to the library using an implicit collective synchronization across PEs. Calling any OpenSHMEM routine after shmem_finalize leads to undefined behavior. 42

The PEs of the OpenSHMEM program are identified by unique integers. The identifiers are integers assigned in a monotonically increasing manner from zero to one less than the total number of PEs. PE identifiers are used for Open-SHMEM calls (e.g. to specify *put* or *get* routines on symmetric data objects, collective synchronization calls) or to dictate a control flow for PEs using constructs of *C* or *Fortran*. The identifiers are fixed for the life of the OpenSHMEM program.

12

14 15

16

17

18

19

20 21 22

23 24

25

26

27

28

29

30 31 32

33 34

48

⁴

⁴start_pes has been deprecated as of OpenSHMEM 1.2

4.1 Progress of OpenSHMEM Operations

The OpenSHMEM model assumes that computation and communication are naturally overlapped. OpenSHMEM programs are expected to exhibit progression of communication both with and without OpenSHMEM calls. Consider a PE that is engaged in a computation with no OpenSHMEM calls. Other PEs should be able to communicate (*put*, *get*, *atomic*, etc) and complete communication operations with that computationally-bound PE without that PE issuing any explicit OpenSHMEM calls. One-sided OpenSHMEM communication calls involving that PE should progress regardless of when that PE next engages in an OpenSHMEM call.

Note to implementors:

- An OpenSHMEM implementation for hardware that does not provide asynchronous communication capabilities may require a software progress thread in order to process remotely-issued communication requests without explicit program calls to the OpenSHMEM library.
- High performance implementations of OpenSHMEM are expected to leverage hardware offload capabilities and provide asynchronous one-sided communication without software assistance.
- Implementations should avoid deferring the execution of one-sided operations until a synchronization point where data is known to be available. High-quality implementations should attempt asynchronous delivery whenever possible, for performance reasons. Additionally, the OpenSHMEM community discourages releasing Open-SHMEM implementations that do not provide asynchronous one-sided operations, as these have very limited performance value for OpenSHMEM programs.

5 Language Bindings and Conformance

OpenSHMEM provides ISO *C* and *Fortran 90* language bindings. As of OpenSHMEM 1.4, the *Fortran* API is deprecated. For rationale and considerations of future *Fortran* use of OpenSHMEM, see Section 2.13.

Any implementation that provides both C and *Fortran* bindings can claim conformance to the specification. Alternatively, an implementation may claim conformance only with respect to one of those languages. For example, an implementation that provides only a C interface may claim to conform to the OpenSHMEM specification with respect to the C language, but not to *Fortran*, and should make this clear in its documentation. The OpenSHMEM header files *shmem.h* for C and *shmem.fh* for *Fortran* must contain only the interfaces and constant names defined in this specification.

OpenSHMEM APIs can be implemented as either routines or macros. However, implementing the interfaces using macros is strongly discouraged as this could severely limit the use of external profiling tools and high-level compiler optimizations. An OpenSHMEM program should avoid defining routine names, variables, or identifiers with the prefix *SHMEM*_(for *C* and *Fortran*), *_SHMEM*_(for *C*) or with OpenSHMEM API names.

All OpenSHMEM extension APIs that are not part of this specification must be defined in the *shmemx.h* and *shmemx.fh* include files for *C* and *Fortran* language bindings, respectively. These header files must exist, even if no extensions are provided. Any extensions shall use the *shmemx*_ prefix for all routine, variable, and constant names.

6 Library Constants

The OpenSHMEM library provides a set of compile-time constants that may be used to specify options to API routines, provide implementation-specific parameters, or return information about the implementation. All constants that start with *_SHMEM_** are deprecated, but provided for backwards compatibility.

Constant	Description	
C/C++: SHMEM_THREAD_SINGLE	The OpenSHMEM thread support level which specifies that the program must not be multithreaded. See Section 9.2 for more detail about its use.	

Constant	Description
C/C++: SHMEM_THREAD_FUNNELED	The OpenSHMEM thread support level which specifies that the program may be multithreaded but must ensure that only the main thread invokes the OpenSHMEM interfaces. See Section 9.2 for more detail about its use.
C/C++: SHMEM_THREAD_SERIALIZED	The OpenSHMEM thread support level which specifies that the program may be multithreaded but must ensure that the OpenSHMEM interfaces are not invoked concurrently by multiple threads. See Section 9.2 for more detail about its use.
C/C++: SHMEM_THREAD_MULTIPLE	The OpenSHMEM thread support level which specifies that the program may be multithreaded and any thread may in- voke the OpenSHMEM interfaces. See Section 9.2 for more detail about its use.
C/C++: SHMEM_TEAM_NOCOLLECTIVE	The team creation option which specifies that the new team will not be initialized with support for team collective operations. See Section 9.4 for more detail about its use.
C/C++: SHMEM_TEAM_NULL	Predefined constant that can be compared against handles of type <i>shmem_team_t</i> to determine if they refer to a valid team. See Section 9.4 for more detail about its use.
C/C++: SHMEM_CTX_SERIALIZED	The context creation option which specifies that the given context is shareable but will not be used by multiple threads concurrently. See Section 9.5.1 for more detail about its use.
C/C++: SHMEM_CTX_PRIVATE	The context creation option which specifies that the given context will be used only by the thread that created it. See Section 9.5.1 for more detail about its use.
C/C++: SHMEM_CTX_NOSTORE	The context creation option which specifies that quiet and fence operations performed on the given context are not required to enforce completion and ordering of memory store operations. See Section 9.5.1 for more detail about its use.
C/C++: SHMEM_SYNC_VALUE — deprecation start	The value used to initialize the elements of $pSync$ arrays. The value of this constant is implementation specific. See Section 9.9 for more detail about its use.
C/C++: _SHMEM_SYNC_VALUE	
Fortran: SHMEM_SYNC_VALUE	
deprecation end —	

6. LIBRARY CONSTANTS

Constant	Description
C/C++: SHMEM_SYNC_SIZE — deprecation start —	Length of a work array that can be used with any SHMEM collective communication operation. Work arrays sized for specific operations may consume less memory. The value of this constant is implementation specific. See Section 9.9 for more detail about its use.
Fortran: SHMEM_SYNC_SIZE	
deprecation end —	
C/C++: SHMEM_BCAST_SYNC_SIZE — deprecation start —	Length of the <i>pSync</i> arrays needed for broadcast routines. The value of this constant is implementation specific. See Section $9.9.6$ for more detail about its use.
C/C++: SHMEM_BCAST_SYNC_SIZE	
Fortran: SHMEM_BCAST_SYNC_SIZE	
deprecation end —	
C/C++: SHMEM_REDUCE_SYNC_SIZE — deprecation start —	Length of the work arrays needed for reduction routines. The value of this constant is implementation specific. See Section 9.9.9 for more detail about its use.
C/C++: _SHMEM_REDUCE_SYNC_SIZE	
Fortran: SHMEM_REDUCE_SYNC_SIZE	
deprecation end	
C/C++: SHMEM_BARRIER_SYNC_SIZE	Length of the work array needed for barrier routines. The value of this constant is implementation specific. See Section 9.9.2 for more detail about its use.
- deprecation start	
C/C++: SHMEM_BARRIER_SYNC_SIZE	
Fortran: SHMEM_BARRIER_SYNC_SIZE	
deprecation end —	

Constant	Description
C/C++: SHMEM_COLLECT_SYNC_SIZE	Length of the work array needed for collect routines. The value of this constant is implementation specific. See Section 9.9.7 for more detail about its use.
- deprecation start	
C/C++: _SHMEM_COLLECT_SYNC_SIZE	
Fortran: SHMEM_COLLECT_SYNC_SIZE	
deprecation end —	
C/C++: SHMEM_ALLTOALL_SYNC_SIZE — deprecation start —	Length of the work array needed for <i>shmem_alltoall</i> rou- tines. The value of this constant is implementation specific. See Section 9.9.10 for more detail about its use.
Fortran: SHMEM_ALLTOALL_SYNC_SIZE deprecation end	
C/C++: SHMEM_ALLTOALLS_SYNC_SIZE — deprecation start —	Length of the work array needed for <i>shmem_alltoalls</i> routines. The value of this constant is implementation specific. See Section 9.9.11 for more detail about its use.
Fortran: SHMEM_ALLTOALLS_SYNC_SIZE 	
C/C++: SHMEM_REDUCE_MIN_WRKDATA_SIZE	Minimum length of work arrays used in various collective routines.
- deprecation start	
C/C++: SHMEM_REDUCE_MIN_WRKDATA_SIZE	
Fortran: SHMEM_REDUCE_MIN_WRKDATA_SIZE	

6. LIBRARY CONSTANTS

Constant	Description
C/C++: SHMEM_MAJOR_VERSION	Integer representing the major version of OpenSHMEM Specification in use.
deprecation start	
C/C++: _SHMEM_MAJOR_VERSION	
Fortran: SHMEM_MAJOR_VERSION	
deprecation end —	
C/C++: SHMEM_MINOR_VERSION	Integer representing the minor version of OpenSHMEM Specification in use.
C/C++: SHMEM_MINOR_VERSION	
Fortran: SHMEM_MINOR_VERSION	
deprecation end —	
C/C++: SHMEM_MAX_NAME_LEN - deprecation start C/C++: SHMEM_MAX_NAME_LEN	Integer representing the maximum length of <i>SHMEM_VENDOR_STRING</i> .
Fortran: SHMEM_MAX_NAME_LEN 	
C/C++: SHMEM_VENDOR_STRING — deprecation start C/C++:	String representing vendor defined information of size at most <i>SHMEM_MAX_NAME_LEN</i> . In <i>C/C++</i> , the string is terminated by a null character. In <i>Fortran</i> , the string of size less than <i>SHMEM_MAX_NAME_LEN</i> is padded with blank characters up to size <i>SHMEM_MAX_NAME_LEN</i> .
_SHMEM_VENDOR_STRING Fortran: SHMEM_VENDOR_STRING deprecation end —	

Constant	Description
C/C++: SHMEM_CMP_EQ	An integer constant expression corresponding to the "equal to" comparison operation. See Section 9.10 for more detail about its use.
deprecation start	
C/C++: _SHMEM_CMP_EQ	
Fortran: SHMEM_CMP_EQ	
deprecation end —	
C/C++: SHMEM_CMP_NE	An integer constant expression corresponding to the "not equal to" comparison operation. See Section 9.10 for more detail about its use.
deprecation start	
C/C++: _SHMEM_CMP_NE	
Fortran: SHMEM_CMP_NE	
deprecation end —	
C/C++: SHMEM_CMP_LT	An integer constant expression corresponding to the "less than" comparison operation. See Section 9.10 for more detail about its use.
— deprecation start —	
C/C++: _SHMEM_CMP_LT	
Fortran: SHMEM_CMP_LT	
deprecation end —	
C/C++: SHMEM_CMP_LE	An integer constant expression corresponding to the "less than or equal to" comparison operation. See Section 9.10 for more detail about its use.
deprecation start	
C/C++: SHMEM_CMP_LE	
Fortran: SHMEM_CMP_LE	
deprecation end —	

Constant	Description
C/C++: SHMEM_CMP_GT	An integer constant expression corresponding to the "greater than" comparison operation. See Section 9.10 for more detail about its use.
deprecation start	
C/C++: SHMEM_CMP_GT	
Fortran: SHMEM_CMP_GT	
deprecation end —	
C/C++: SHMEM_CMP_GE	An integer constant expression corresponding to the "greater than or equal to" comparison operation. See Section 9.10 for more detail about its use.
deprecation start	
_SHMEM_CMP_GE	
Fortran: SHMEM_CMP_GE	
deprecation end —	

7 Library Handles

The OpenSHMEM library provides a set of predefined named constant handles. All named constants can be used in initialization expressions or assignments, but not necessarily in array declarations or as labels in C switch statements. This implies named constants to be link-time but not necessarily compile-time constants.

Handle	Description
	Handle of type <i>shmem_team_t</i> that corresponds to the de-
C/C++:	fault team of all PEs in the OpenSHMEM program. All
SHMEM_TEAM_WORLD	point-to-point communication operations and synchroniza-
	tions that do not specify a team are performed on the default
	team. See Section 9.4 for more detail about its use.
	Handle of type <i>shmem_team_t</i> that corresponds a team of
C/C++: SHMEM_TEAM_NODE	PEs which share node level resources, such as shared mem-
	ory, network interfaces, etc. When this handle is used by
	some PE, it will refer to the node level team containing that
	PE. See Section 9.4 for more detail about its use.
<i>a</i> / <i>a</i>	Handle of type <i>shmem_ctx_t</i> that corresponds to the default
C/C++:	communication context. All point-to-point communication
SHMEM_CTX_DEFAULT	operations and synchronizations that do not specify a con-
	text are performed on the default context. See Section 9.5
	for more detail about its use.

8 Environment Variables

The OpenSHMEM specification provides a set of environment variables that allows users to configure the Open-SHMEM implementation, and receive information about the implementation. The implementations of the specification are free to define additional variables. Currently, the specification defines four environment variables. All environment variables that start with *SMA_** are deprecated, but currently supported for backwards compatibility. If both *SHMEM_*and *SMA_*-prefixed environment variables are set, then the value in the *SHMEM_-*prefixed environment variable establishes the controlling value. Refer to the *SMA_** Environment Variables deprecation rationale for more details.

Variable	Value	Description
SHMEM_VERSION	Any	Print the library version at start-up
SHMEM_INFO	Any	Print helpful text about all these environment variables
SHMEM_SYMMETRIC_SIZE	Non-negative integer	Number of bytes to allocate for symmetric heap
SHMEM_DEBUG	Any	Enable debugging messages

9 OpenSHMEM Library API

9.1 Library Setup, Exit, and Query Routines

The library setup and query interfaces that initialize and monitor the parallel environment of the PEs.

9.1.1 SHMEM_INIT

A collective operation that allocates and initializes the resources used by the OpenSHMEM library.

SYNOPSIS

C/C++:	
<pre>void shmem_init(void);</pre>	
- deprecation start	
FORTRAN:	
CALL SHMEM_INIT()	
	deprecation end

DESCRIPTION

Arguments None.

API description

shmem_init allocates and initializes resources used by the OpenSHMEM library. It is a collective operation that all PEs must call before any other OpenSHMEM routine may be called. At the end of the OpenSHMEM program which it initialized, the call to *shmem_init* must be matched with a call to *shmem_finalize*. After the first call to *shmem_init*, a subsequent call to *shmem_init* or *shmem_init_thread* in the same program results in undefined behavior.

Return Values

None.

Notes

As of OpenSHMEM 1.2, the use of *start_pes* has been deprecated and calls to it should be replaced with calls to *shmem_init*. While support for *start_pes* is still required in OpenSHMEM libraries, users are encouraged to use *shmem_init*. An important difference between *shmem_init* and *start_pes* is that multiple calls to *shmem_init* within a program results in undefined behavior, while in the case of *start_pes*, any subsequent calls to *start_pes* after the first one results in a no-op.

EXAMPLES

The following *shmem_init* example is for *C11* programs:

```
#include <stdio.h>
#include <shmem.h>
int main(void) {
    static int targ = 0;
```

```
1
                 shmem_init();
2
                 int me = shmem_my_pe();
3
                 int receiver = 1 % shmem_n_pes();
                 if (me == 0) {
                     int src = 33;
                     shmem_put(&targ, &src, 1, receiver);
                 }
                 shmem_barrier_all(); /* Synchronizes sender and receiver */
9
                 if (me == receiver)
10
                     printf("PE %d targ=%d (expect 33)\n", me, targ);
11
                 shmem_finalize();
12
                 return 0;
13
14
15
      9.1.2 SHMEM_MY_PE
16
17
      Returns the number of the calling PE.
18
19
      SYNOPSIS
20
            C/C++:
21
            int shmem_my_pe(void);
22
            - deprecation start -
23
            FORTRAN:
24
            INTEGER SHMEM_MY_PE, ME
25
            ME = SHMEM_MY_PE()
26
                                                                                                    deprecation end -
27
28
29
      DESCRIPTION
30
31
            Arguments
32
                   None.
33
34
            API description
35
36
                 This routine returns the PE number of the calling PE. It accepts no arguments. The result is an integer
37
                 between 0 and npes - 1, where npes is the total number of PEs executing the current program.
38
39
40
            Return Values
41
                 Integer - Between 0 and npes - 1
42
43
            Notes
44
                 Each PE has a unique number or identifier. As of OpenSHMEM 1.2 the use of _my_pe has been dep-
45
                 recated. Although OpenSHMEM libraries are required to support the call, users are encouraged to use
46
                 shmem_my_pe instead. The behavior and signature of the routine shmem_my_pe remains unchanged from
47
                 the deprecated <u>my_pe</u> version.
48
```

9.1.3 SHMEM_N_PES

Returns the number of PEs running in a program.

SYNOPSIS

C/C++: int shmem_n_pes(void);

— deprecation start FORTRAN: INTEGER SHMEM_N_PES, N_PES N_PES = SHMEM_N_PES()

DESCRIPTION

Arguments

None.

API description

The routine returns the number of PEs running in the program.

Return Values

Integer - Number of PEs running in the OpenSHMEM program.

Notes

As of OpenSHMEM 1.2 the use of _*num_pes* has been deprecated. Although OpenSHMEM libraries are required to support the call, users are encouraged to use *shmem_n_pes* instead. The behavior and signature of the routine *shmem_n_pes* remains unchanged from the deprecated _*num_pes* version.

EXAMPLES

The following *shmem_my_pe* and *shmem_n_pes* example is for C/C++ programs:

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

int main(void)
{
    shmem_init();
    int me = shmem_my_pe();
    int npes = shmem_n_pes();
    printf("I am #%d of %d PEs executing this program\n", me, npes);
    shmem_finalize();
    return 0;
}
```

deprecation end -

9.1.4 SHMEM_FINALIZE

A collective operation that releases all resources used by the OpenSHMEM library. This only terminates the Open-SHMEM portion of a program, not the entire program.

SYNOPSIS

C/C+	+:
void	<pre>shmem_finalize(void);</pre>
— dej	precation start —
FOR	TRAN:
CALL	SHMEM_FINALIZE()
	deprecation end —
ESCRIPT	ION
Argu	ments
	None.
API d	lescription
2	shmem_finalize is a collective operation that ends the OpenSHMEM portion of a program previously ini-
5 1 1 1 1 1 1 1 1	This collective operation requires all PEs to participate in the call. There is an implicit global barrier in <i>shmem_finalize</i> to ensure that pending communications are completed and that no resources are released until all PEs have entered <i>shmem_finalize</i> . This routine destroys all shareable contexts. The user is responsible for destroying all contexts with the <i>SHMEM_CTX_PRIVATE</i> option enabled prior to calling this routine; otherwise, the behavior is undefined. <i>shmem_finalize</i> must be the last OpenSHMEM library call encountered in the OpenSHMEM portion of a program. A call to <i>shmem_finalize</i> will release all resources nitialized by a corresponding call to <i>shmem_init</i> or <i>shmem_init_thread</i> . All processes that represent the PEs will still exist after the call to <i>shmem_finalize</i> returns, but they will no longer have access to resources
t	hat have been released.

Return Values

None.

Notes

shmem_finalize releases all resources used by the OpenSHMEM library including the symmetric memory heap and pointers initiated by *shmem_ptr*. This collective operation requires all PEs to participate in the call, not just a subset of the PEs. The non-OpenSHMEM portion of a program may continue after a call to *shmem_finalize* by all PEs.

EXAMPLES

The following finalize example is for C11 programs: #include <stdio.h>
#include <shmem.h>
int main(void)
{

```
static long x = 10101;
long y = -1;
shmem_init();
int me = shmem_my_pe();
int npes = shmem_n_pes();
if (me == 0)
   y = shmem_g(&x, npes-1);
printf("%d: y = %ld\n", me, y);
shmem_finalize();
return 0;
```

9.1.5 SHMEM_GLOBAL_EXIT

A routine that allows any PE to force termination of an entire program.

SYNOPSIS

C11: _Noreturn void shmem_global_exit(int status);

status

C/C++:

void shmem_global_exit(int status);

- deprecation start

FORTRAN:

INTEGER STATUS
CALL SHMEM_GLOBAL_EXIT(status)

DESCRIPTION

Arguments IN

The exit status from the main program.

API description

shmem_global_exit is a non-collective routine that allows any one PE to force termination of an Open-SHMEM program for all PEs, passing an exit status to the execution environment. This routine terminates the entire program, not just the OpenSHMEM portion. When any PE calls *shmem_global_exit*, it results in the immediate notification to all PEs to terminate. *shmem_global_exit* flushes I/O and releases resources in accordance with *C/C++/Fortran* language requirements for normal program termination. If more than one PE calls *shmem_global_exit*, then the exit status returned to the environment shall be one of the values passed to *shmem_global_exit* as the status argument. There is no return to the caller of *shmem_global_exit*; control is returned from the OpenSHMEM program to the execution environment for all PEs. 2

9

10

11 12 13

14 15

16 17

18

19

20

21

22 23

24

25

26

27 28 29

30 31

32

33 34

35

37

38

39

40

41

42

43

44 45 46

47

48

deprecation end -

Note	tes		
	shmem_global_exit may be used in situations where	nere one or more PEs have determ	nined that the program
	has completed and/or should terminate early. Acc	ordingly, the integer status argume	ent can be used to pass
	any information about the nature of the exit; e.g	., that the program encountered a	n error or found a so-
	lution. Since <i>shmem global exit</i> is a non-collect	tive routine, there is no implied sy	nchronization, and all
	PEs must terminate regardless of their current exe	cution state. While I/O must be flu	ished for standard lan-
	guage I/O calls from $C/C + +/Fortran$ it is impler	nentation dependent as to how I/C) done by other means
	(e.g. third party I/O libraries) is handled. Similar	arly resources are released accord	ing to $C/C \pm \pm/Fortran$
	standard language requirements, but this may not i	include all resources allocated for t	the OpenSUMEM pro
	gram However a quality implementation will mal	here a best affort to flush all I/O and c	leon up all resources
	gram. However, a quanty implementation win mai	te a best enort to nush an 1/O and c	iean up an resources.
EXAMPL	LES		
#inc	clude <stdio.h></stdio.h>		
#inc	clude <stdlib.h></stdlib.h>		
#inc	clude <shmem.h></shmem.h>		
int	main(void)		

```
FILE *fp = fopen("input.txt", "r");
if (fp == NULL) { /* Input file required by program is not available */
    shmem_global_exit(EXIT_FAILURE);
    }
    /* do something with the file */
    fclose(fp);
    }
    shmem_finalize();
    return 0;
}
```

9.1.6 SHMEM_PE_ACCESSIBLE

shmem_init();

if (me == 0) {

int me = shmem_my_pe();

Determines whether a PE is accessible via OpenSHMEM's data transfer routines.

SYNOPSIS

{

34	
35	C/C++:
36	<pre>int shmem_pe_accessible(int pe);</pre>
37	— deprecation start —
38	FORTRAN:
39	LOGICAL LOG, SHMEM_PE_ACCESSIBLE
40	INTEGER pe
40	LOG = SHMEM_PE_ACCESSIBLE(pe)
41	deprecation end —
12	
43	
44	DESCRIPTION
45	DESCRIPTION
46	A
47	
48	IN <i>pe</i> Specific PE to be checked for accessibility from the local PE.

API description

shmem_pe_accessible is a query routine that indicates whether a specified PE is accessible via Open-SHMEM from the local PE. The *shmem_pe_accessible* routine returns a value indicating whether the remote PE is a process running from the same executable file as the local PE, thereby indicating whether full support for symmetric data objects, which may reside in either static memory or the symmetric heap, is available.

Return Values

C/C++: The return value is 1 if the specified PE is a valid remote PE for OpenSHMEM routines; otherwise, it is 0.

Fortran: The return value is *.TRUE*. if the specified PE is a valid remote PE for OpenSHMEM routines; otherwise, it is *.FALSE*..

Notes

This routine may be particularly useful for hybrid programming with other communication libraries (such as MPI) or parallel languages. For example, when an MPI job uses *Multiple Program Multiple Data* (MPMD) mode, multiple executable MPI programs are executed as part of the same MPI job. In such cases, OpenSHMEM support may only be available between processes running from the same executable file. In addition, some environments may allow a hybrid job to span multiple network partitions. In such scenarios, OpenSHMEM support may only be available between PEs within the same partition.

9.1.7 SHMEM_ADDR_ACCESSIBLE

Determines whether an address is accessible via OpenSHMEM data transfer routines from the specified remote PE.

SYNOPSIS

C/C++:

int shmem_addr_accessible(const void *addr, int pe);

deprecation start
FORTRAN:
LOGICAL LOG, SHMEM_ADDR_ACCESSIBLE
INTEGER pe
LOG = SHMEM_ADDR_ACCESSIBLE(addr, pe)

deprecation end -

DESCRIPTION

Arguments	
IN	

addrData object on the local PE.peInteger id of a remote PE.

API description

IN

shmem_addr_accessible is a query routine that indicates whether a local address is accessible via Open-SHMEM routines from the specified remote PE.

This routine verifies that the data object is symmetric and accessible with respect to a remote PE via Open-SHMEM data transfer routines. The specified address *addr* is a data object on the local PE.

1	
2	
3	Return Values
4	C/C++: The return value is 1 if addr is a symmetric data object and accessible via OpenSHMEM routines
5	from the specified remote PE; otherwise, it is 0.
6	Fortran: The return value is .TRUE. if addr is a symmetric data object and accessible via OpenSHMEM
7	routines from the specified remote PE; otherwise, it is .FALSE
8	
9	Notes
10	This routine may be particularly useful for hybrid programming with other communication libraries (such
11	as MPI) or parallel languages. For example, when an MPI job uses MPMD mode, multiple executable MPI
12	programs may use OpenSHIVIEM routines. In such cases, static memory, such as a <i>Fortran</i> common block or C global variable, is symmetric between processes running from the same executable file, but is not
13	symmetric between processes running from different executable files. Data allocated from the symmetric
14	heap (shmem_malloc or shnalloc) is symmetric across the same or different executable files
15	neup (sumeni_manoe of supanoe) is symmetric across the same of american execution mest
16	
17 01	2 SHMEM DTD
18	, SHIVIEN_IIK
19 Retu	Irns a local pointer to a symmetric data object on the specified PE.
20	
	VOPSIS
22	C/C++:
22	<pre>void *shmem_ptr(const void *dest, int pe);</pre>
2.5	- deprecation start
24	FORTRAN:
25	POINTER (PTR, POINTEE)
20	INTEGER pe
27	PTR = SHMEM_PTR(dest, pe)
28	deprecation end —
29	
30	
³¹ DES	SCRIPTION
32	Arguments
33	IN <i>dest</i> The symmetric data object to be referenced
34	IN <i>ne</i> An integer that indicates the PE number on which <i>dest</i> is to be accessed.
35	When using <i>Fortran</i> , it must be a default integer value.
36	
37	API description
38	
39	<i>shmem_ptr</i> returns an address that may be used to directly reference <i>dest</i> on the specified PE. This address
40	can be assigned to a pointer. After that, ordinary loads and stores to this remote address may be performed.
41	The <i>shmem_ptr</i> routine can provide an efficient means to accomplish communication, for example when a
42	sequence of reads and writes to a data object on a remote PE does not match the access pattern provided in
43	an OpenSHMEM data transfer routine like <i>snmem_put</i> or <i>shmem_iget</i> .
44	
45	
46	Return Values
47	The address of the <i>dest</i> data object is returned when it is accessible using memory loads and stores. Other-

wise, a null pointer is returned.

When calling *shmem_ptr*, *dest* is the address of the referenced symmetric data object on the calling PE.

EXAMPLES

This *Fortran* program calls *shmem_ptr* and then PE 0 writes to the *BIGD* array on PE 1:

```
PROGRAM REMOTEWRITE
INCLUDE "shmem.fh"
INTEGER BIGD (100)
SAVE BIGD
INTEGER POINTEE (*)
POINTER (PTR, POINTEE)
CALL SHMEM_INIT()
IF (SHMEM_MY_PE() .EQ. 0) THEN
   ! initialize PE 1's BIGD array
  PTR = SHMEM_PTR(BIGD, 1)  ! get address of PE 1's BIGD
                                 !
                                    array
   DO I=1,100
       POINTEE(I) = I
   ENDDO
ENDIF
CALL SHMEM_BARRIER_ALL
IF (SHMEM_MY_PE() .EQ. 1) THEN
  PRINT*,'BIGD on PE 1 is: '
  PRINT*,BIGD
ENDIF
END
This is the equivalent program written in C11:
#include <stdio.h>
#include <shmem.h>
int main(void)
   static int dest[4];
   shmem_init();
   int me = shmem_my_pe();
   if (me == 0) { /* initialize PE 1's dest array */
      int* ptr = shmem_ptr(dest, 1);
      if (ptr == NULL)
         printf("can't use pointer to directly access PE 1's dest array\n");
      else
         for (int i = 0; i < 4; i++)</pre>
            *ptr++ = i + 1;
   }
   shmem_barrier_all();
   if (me == 1)
      printf("PE 1 dest: %d, %d, %d, %d\n",
        dest[0], dest[1], dest[2], dest[3]);
   shmem_finalize();
   return 0;
```

9.1.9 SHMEM_INFO_GET_VERSION

Returns the major and minor version of the library implementation.

2

6

7

9

10

11

12

13

14 15

16

17

18

19

20

21 22

23

24

25

26

27

28

29

30

31 32

33

34

35

36

37

38

39

40

41

42

43

44 45 46

SYNOPSIS		
C/C++:		
<pre>void shmem_info_</pre>	get_version(ir	<pre>nt *major, int *minor);</pre>
— deprecation star FORTRAN:	t	
INTEGER MAJOR, M	INOR	
CALL SHMEM_INFO_	GET_VERSION (MA	AJOR, MINOR)
		deprecation end —
DESCRIPTION		
Arguments		
OUT	maior	The major version of the OpenSHMEM Specification in use
OUT	minor	The minor version of the OpenSHMEM Specification in use.
API description		
This routine	eturns the majo	r and minor version of the OpenSHMEM Specification in use. For a given
library impler constants <i>SH</i>	nentation, the m MEM_MAJOR_	ajor and minor version returned by these calls are consistent with the library <i>VERSION</i> and <i>SHMEM_MINOR_VERSION</i> .
Return Values		
None		
i tone.		
Notos		
None		
Tone.		
9.1.10 SHMEM_INFO	D_GET_NAME	S
This routine returns the ve	endor defined nar	me string that is consistent with the library constant SHMEM VENDOR STRI
		v
SYNOPSIS		
C/C++:		
<pre>void shmem_info_</pre>	get_name(char	*name);
- deprecation star	t	
FORTRAN:		
CHARACTER * (*) NA	ME	
CALL SHMEM_INFO_	GET_NAME (NAME))
		deprecation end —
		*
DESCRIPTION		
Arguments		
ATT	name	The vendor defined string

API description

This routine returns the vendor defined name string of size defined by the library constant *SHMEM_MAX_NAME_LEN*. The program calling this function provides the *name* memory buffer of at least size *SHMEM_MAX_NAME_LEN*. The implementation copies the vendor defined string of size at most *SHMEM_MAX_NAME_LEN* to *name*. In *C/C++*, the string is terminated by a null character. In *Fortran*, the string of size less than *SHMEM_MAX_NAME_LEN* is padded with blank characters up to size *SHMEM_MAX_NAME_LEN*. If the *name* memory buffer is provided with size less than *SHMEM_MAX_NAME_LEN*, behavior is undefined. For a given library implementation, the vendor string returned is consistent with the library constant *SHMEM_VENDOR_STRING*.

Return Values

None.

Notes

None.

9.1.11 START_PES

Called at the beginning of an OpenSHMEM program to initialize the execution environment. This routine is deprecated and is provided for backwards compatibility. Implementations must include it, and the routine should function properly and may notify the user about deprecation of its use.

API description

The *start_pes* routine initializes the OpenSHMEM execution environment. An OpenSHMEM program must call *start_pes*, *shmem_init*, or *shmem_init_thread* before calling any other OpenSHMEM routine. Unlike *shmem_init* and *shmem_init_thread*, *start_pes* does not require a call to *shmem_finalize*. Instead, the OpenSHMEM library is implicitly finalized when the program exits. Implicit finalization is collective and includes a global synchronization to ensure that all pending communication is completed before resources are released.

1	Return Values
2	None.
3	
4	Notes
5	If any other OpenSHMEM call occurs before <i>start_pes</i> , the behavior is undefined. Although it is recom-
6	mended to set <i>npes</i> to 0 for <i>start_pes</i> , this is not mandated. The value is ignored. Calling <i>start_pes</i> mor
7	than once has no subsequent effect.
8	As of OpenSHMEM 1.2 the use of <i>start_pes</i> has been deprecated. Although OpenSHMEM libraries ar
9	required to support the call, users are encouraged to use <i>shmem_init</i> or <i>shmem_init_thread</i> instead.
10	
11 12	EXAMPLES
13	This is a simple program that calls <i>start_pes</i> :
14	
15	INCLUDE "shmem.fh"
16	
17	COMMON /T/ TARG
18	PARAMETER (RECEIVER=1)
19	CALL START_PES(0)
20	IF (SHMEM_MY_PE() .EQ. 0) THEN
21	SRC = 33
22	ENDIF
23	
24	CALL SHMEM_BARRIER_ALL ! SYNCHRONIZES SENDER AND RECEIVER
25	IF (SHMEM_MY_PE() .EQ. RECEIVER) THEN
26	PRINT*,'PE ', SHMEM_MY_PE(),' TARG=',TARG,' (expect 33)' ENDIF
27	END
28	
29	
30	9.2 Thread Support
31	
32	This section specifies the interaction between the OpenSHMEM interfaces and user threads. It also describes the interfaces and user threads are found and the section of th
33	routines that can be used for initializing and querying the thread environment. There are four levels of threadin defined by the OpenSHMEM enceifaction
34	defined by the OpenSHWEW specification.
35	SHMEM THREAD SINCLE
36	The OpenSHMEM program must not be multithreaded
37	The openetimeter program must not be multimended.
38	SHMEM_THREAD_FUNNELED
39	The OpenSHMEM program may be multithreaded. However, the program must ensure that only the mai
40	thread invokes the OpenSHMEM interfaces. The main thread is the thread that invokes either <i>shmem_init</i> of

shmem_init_thread.

41 42

43

44 45

46

47

SHMEM_THREAD_SERIALIZED

The OpenSHMEM program may be multithreaded. However, the program must ensure that the OpenSHMEM interfaces are not invoked concurrently by multiple threads.

SHMEM_THREAD_MULTIPLE

The OpenSHMEM program may be multithreaded and any thread may invoke the OpenSHMEM interfaces.

⁴⁸ The following semantics apply to the usage of these models:

9. OPENSHMEM LIBRARY API

- 1. In the *SHMEM_THREAD_FUNNELED*, *SHMEM_THREAD_SERIALIZED*, and *SHMEM_THREAD_MULTIPLE* thread levels, the *shmem_init* and *shmem_finalize* calls must be invoked by the same thread.
- 2. Any OpenSHMEM operation initiated by a thread is considered an action of the PE as a whole. The symmetric heap and symmetric variables scope are not impacted by multiple threads invoking the OpenSHMEM interfaces. Each PE has a single symmetric data segment and symmetric heap that is shared by all threads within that PE. For example, a thread invoking a memory allocation routine such as *shmem_malloc* allocates memory that is accessible by all threads of the PE. The requirement that the same symmetric heap operations must be executed by all PEs in the same order also applies in a threaded environment. Similarly, the completion of collective operations is not impacted by multiple threads. For example, *shmem_barrier_all* is completed when all PEs enter and exit the *shmem_barrier_all* call, even though only one thread in the PE is participating in the collective call.
- 3. Blocking OpenSHMEM calls will only block the calling thread, allowing other threads, if available, to continue executing. The calling thread will be blocked until the event on which it is waiting occurs. Once the blocking call is completed, the thread is ready to continue execution. A blocked thread will not prevent progress of other threads on the same PE and will not prevent them from executing other OpenSHMEM calls when the thread level permits. In addition, a blocked thread will not prevent the progress of OpenSHMEM calls performed on other PEs.
- 4. In the *SHMEM_THREAD_MULTIPLE* thread level, all OpenSHMEM calls are thread-safe. Any two concurrently running threads may make OpenSHMEM calls and the outcome will be as if the calls executed in some order, even if their execution is interleaved.
- 5. In the *SHMEM_THREAD_SERIALIZED* and *SHMEM_THREAD_MULTIPLE* thread levels, if multiple threads call collective routines, including the symmetric heap management routines, it is the programmer's responsibility to ensure the correct ordering of collective calls.

9.2.1 SHMEM_INIT_THREAD

Initializes the OpenSHMEM library, similar to *shmem_init*, and performs any initialization required for supporting the provided thread level.

SYNOPSIS

C/C++:

int shmem_init_thread(int requested, int *provided);

DESCRIPTION

Arguments		
IN	requested	The thread level support requested by the user.
OUT	provided	The thread level support provided by the OpenSHMEM implementa-
		tion.

API description

shmem_init_thread initializes the OpenSHMEM library in the same way as *shmem_init*. In addition, *shmem_init_thread* also performs the initialization required for supporting the provided thread level. The argument *requested* is used to specify the desired level of thread support. The argument *provided* returns the support level provided by the library. The allowed values for *provided* and *requested* are *SHMEM_THREAD_SINGLE*, *SHMEM_THREAD_FUNNELED*, *SHMEM_THREAD_SERIALIZED*, and *SHMEM_THREAD_MULTIPLE*.

An OpenSHMEM program is initialized either by *shmem_init* or *shmem_init_thread*. Once an Open-SHMEM library initialization call has been performed, a subsequent initialization call in the same program

results in undefined behavior. If the call to <i>shmem_init_thread</i> is unsuccessful in allocating and initializin resources for the OpenSHMEM library, then the behavior of any subsequent call to the OpenSHMEE library is undefined.
Return Values shmem_init_thread returns 0 upon success; otherwise, it returns a non-zero value.
Notes The OpenSHMEM library can be initialized either by <i>shmem_init</i> or <i>shmem_init_thread</i> . If the Ope
SHMEM library is initialized by <i>shmem_init</i> , the library implementation can choose to support any one the defined thread levels.
9.2.2 SHMEM_QUERY_THREAD
Returns the level of thread support provided by the library.

SYNOPSIS

C/C++: void shmem_query_thread(int *provided);

provided

DESCRIPTION

Arguments

OUT

The thread level support provided by the OpenSHMEM implementation.

API description

The *shmem_query_thread* call returns the level of thread support currently being provided. The value returned will be same as was returned in *provided* by a call to *shmem_init_thread*, if the OpenSHMEM library was initialized by *shmem_init_thread*. If the library was initialized by *shmem_init*, the implementation can choose to provide any one of the defined thread levels, and *shmem_query_thread* returns this thread level.

Return Values

None.

Notes

None.

Memory Management Routines 9.3

47 OpenSHMEM provides a set of APIs for managing the symmetric heap. The APIs allow one to dynamically allocate, 48 deallocate, reallocate and align symmetric data objects in the symmetric heap.

1

2

9

10

11

16

17 18

19 20

21

22 23

24 25

26

27

28 29

30 31

32

33

34

35

36 37 38

39

40 41

42

43 44 45

9.3.1 SHMEM_MALLOC, SHMEM_FREE, SHMEM_REALLOC, SHMEM_ALIGN

Collective symmetric heap memory management routines.

SYNOPSIS

```
C/C++:
void *shmem_malloc(size_t size);
void shmem_free(void *ptr);
void *shmem_realloc(void *ptr, size_t size);
void *shmem_align(size_t alignment, size_t size);
```

DESCRIPTION

Arguments		
IN	size	The size, in bytes, of a block to be allocated from the symmetric heap.
		This argument is of type <i>size_t</i>
IN	ptr	Pointer to a block within the symmetric heap.
IN	alignment	Byte alignment of the block allocated from the symmetric heap.

API description

The *shmem_malloc*, *shmem_free*, *shmem_realloc*, and *shmem_align* routines are collective operations that require participation by all PEs.

The *shmem_malloc* routine returns a pointer to a block of at least *size* bytes, which shall be suitably aligned so that it may be assigned to a pointer to any type of object. This space is allocated from the symmetric heap (in contrast to *malloc*, which allocates from the private heap). When *size* is zero, the *shmem_malloc* routine performs no action and returns a null pointer.

The *shmem_align* routine allocates a block in the symmetric heap that has a byte alignment specified by the *alignment* argument. The value of *alignment* shall be a multiple of *sizeof(void *)* that is also a power of two. Otherwise, the behavior is undefined. When *size* is zero, the *shmem_align* routine performs no action and returns a null pointer.

The *shmem_free* routine causes the block to which *ptr* points to be deallocated, that is, made available for further allocation. If *ptr* is a null pointer, no action is performed.

The *shmem_realloc* routine changes the size of the block to which *ptr* points to the size (in bytes) specified by *size*. The contents of the block are unchanged up to the lesser of the new and old sizes. If the new size is larger, the newly allocated portion of the block is uninitialized. If *ptr* is a null pointer, the *shmem_realloc* routine behaves like the *shmem_malloc* routine for the specified size. If *size* is 0 and *ptr* is not a null pointer, the block to which it points is freed. If the space cannot be allocated, the block to which *ptr* points is unchanged.

The *shmem_malloc*, *shmem_align*, *shmem_free*, and *shmem_realloc* routines are provided so that multiple PEs in a program can allocate symmetric, remotely accessible memory blocks. These memory blocks can then be used with OpenSHMEM communication routines. When no action is performed, these routines return without performing a barrier. Otherwise, each of these routines includes at least one call to a procedure that is semantically equivalent to *shmem_barrier_all: shmem_malloc* and *shmem_align* call a barrier on exit; *shmem_free* calls a barrier on entry; and *shmem_realloc* may call barriers on both entry and exit, depending on whether an existing allocation is modified and whether new memory is allocated, respectively. This ensures that all PEs participate in the memory allocation, and that the memory on other PEs can be used as soon as the local PE returns. The implicit barriers performed by these routines quiet the default context. It is the user's responsibility to ensure that no communication operations involving the given memory block are pending on other contexts prior to calling the *shmem_free* and *shmem_realloc* routines. The user is also responsible for calling these routines with identical argument(s) on all PEs; if differing *ptr*,

size, or alignment arguments are used, the behavior of the call and any subsequent OpenSHMEM calls is

2	undefined.			
3				
4				
5	Datum Valuas			
6	Return values			
7	The <i>shmem_malloc</i> routine returns a pointer to the allocated space; otherwise, it returns a null pointer.			
,	The <i>shmem_free</i> routine i	eturns no value.		
8 9	The <i>shmem_realloc</i> routine returns a pointer to the allocated space (which may have moved); otherwise all PEs return a null pointer.			
10 11	The <i>shmem_align</i> routine returns a pull pointer	returns an aligned pointer whose value is a multiple of <i>alignment</i> ; otherwise, it		
12	returns a nun pointer.			
13				
15	Notes			
14	As of OpenSHMEM 1.2 the use of <i>shmalloc</i> , <i>shmemalign</i> , <i>shfree</i> , and <i>shrealloc</i> has been deprecated. A though OpenSHMEM libraries are required to support the calls, users are encouraged to use <i>shmem_malloc</i>			
16 17	shmem_align, shmem_free, and shmem_realloc instead. The behavior and signature of the routines re mains unchanged from the deprecated versions.			
18	The total size of the symmetric heap is determined at job startup. One can specify the size of the heap usin the <i>SHMEM_SYMMETRIC_SIZE</i> environment variable (where available).			
19	The shmem malloc, shmem free, and shmem realloc routines differ from the private heap allocation routines			
20	tines in that all PEs in a program must call them (a barrier is used to ensure this).			
21	When the <i>ntr</i> argument in a call to <i>shmem</i> realloc corresponds to a buffer allocated using <i>shmem</i> align			
22 23	the buffer returned by <i>shmem_realloc</i> is not guaranteed to maintain the alignment requested in the origina			
24	call to shmem_aligh.			
25				
26	Note to implementors			
27	The symmetric heap allocation routines always return a pointer to corresponding symmetric objects acros			
28	all PEs. The OpenSHMEM specification does not require that the virtual addresses are equal across all PEs.			
29	nevertheless, the implementation must avoid costly address translation operations in the communication path including $Q(N)$ memory translation tables, where N is the number of PEs. In order to avoid address			
30	paul, including $O(N)$ memory translation tables, where N is the number of PEs. In order to avoid addres translations, the implementation may re-map the allocated block of memory based on agreed virtual ad			
31	dress Additionally some operating systems provide an option to disable virtual address randomization			
32	which enables predictable allocation of virtual memory addresses			
33				
34				
35				
36	9.3.2 SHMEM_CALLOC			
27	Allocate a zeroed block of symmetric memory			
37	Anocate a zeroed block of symmetric	memory.		
38	SYNOPSIS			
39				
40	C/C++:			
41	<pre>void *shmem_calloc(size_t</pre>	count, size_t size);		
42				
43				
44	DESCRIPTION			
45	A			
46	Arguments			
47	IN count	The size is better of each clear at the line to		
	IIN SIZE	The size in dytes of each element to allocate.		

1

API description

The *shmem_calloc* routine is a collective operation that allocates a region of remotely-accessible memory for an array of *count* objects of *size* bytes each and returns a pointer to the lowest byte address of the allocated symmetric memory. The space is initialized to all bits zero.

If the allocation succeeds, the pointer returned shall be suitably aligned so that it may be assigned to a pointer to any type of object. If the allocation does not succeed, or either *count* or *size* is 0, the return value is a null pointer.

The values for *count* and *size* shall each be equal across all PEs calling *shmem_calloc*; otherwise, the behavior is undefined.

When *count* or *size* is 0, the *shmem_calloc* routine returns without performing a barrier. Otherwise, this routine calls a procedure that is semantically equivalent to *shmem_barrier_all* on exit.

Return Values

The *shmem_calloc* routine returns a pointer to the lowest byte address of the allocated space; otherwise, it returns a null pointer.

Notes

None.

9.3.3 SHPALLOC

Allocates a block of memory from the symmetric heap.

SYNOPSIS

— deprecation start FORTRAN: POINTER (addr, A(1)) INTEGER length, errcode, abort CALL SHPALLOC(addr, length, errcode, abort)

deprecation end -

DESCRIPTION

ArgumentsFirst word address of the allocated block.OUTaddrFirst word address of the allocated block.INlengthNumber of words of memory requested. One word is 32 bits.OUTerrcodeError code is 0 if no error was detected; otherwise, it is a negative integer code for the type of error.INabortAbort code; nonzero requests abort on error; 0 requests an error code.

API description

SHPALLOC allocates a block of memory from the program's symmetric heap that is greater than or equal to the size requested. To maintain symmetric heap consistency, all PEs in an program must call *SHPALLOC* with the same value of length; if any PEs are missing, the program will hang.

By using the *Fortran POINTER* mechanism in the following manner, array A can be used to refer to the block allocated by *SHPALLOC: POINTER* (*addr*, A())

2				
3	Return Values			
4				
5	Error Code		Condition	
6	-1		Length is not an integer greater than 0	
7	-2		No more memory is available from the system (checked if the	
8			request cannot be satisfied from the available blocks on the sym-	
9			metric neap).	
10	Notes			
11	The total size of the symmetric heap is determined at job startup. One may adjust the size of the heap using			
12	the SHMEM_SYMMETRIC_SIZE environment variable (if available).			
13				
14				
15	Note to implementors			
16	The symmetric neap allocation routines always return a pointer to corresponding symmetric objects across all DEs.			
17	all PES. The OpenSHMEM specification does not require that the virtual addresses are equal across all PES.			
18	not including order N (where N is the number of PFs) memory translation tables. In order to avoid ad-			
19	dress translations, the implementation may re-map the allocated block of memory based on agreed virtual			
20	address. Additionally, some operating systems provide an option to disable virtual address randomization,			
21	which enables predictable allocation of virtual memory addresses.			
22				
22				
23	9.3.4 SHPCLMOVE			
25				
26	Extends a symmetric hear	block or copies	the contents of the block into a larger block.	
20	GUNIODAIA			
27	SYNOPSIS			
28	— deprecation start	t		
29	FORTRAN:			
30	POINTER (addr, A(1))			
31	INTEGER length, status, abort			
32	CALL SHPCLMOVE (ad	ddr, length, s	tatus, abort)	
33			deprecation end —	
34				
35				
36	DESCRIPTION			
37				
38	Arguments			
39	INOUT	addr	On entry, first word address of the block to change; on exit, the new	
40	The	7 .7	address of the block if it was moved.	
41	IN	length	Requested new total length in words. One word is 32 bits.	
42	OUT	status	Status is 0 if the block was extended in place, 1 if it was moved, and a	
43	IN	1	negative integer for the type of error detected.	
44	LIN	abort	Adort code. Nonzero requests abort on error; U requests an error code.	
45				
46	API description			
40				
1.1				

The SHPCLMOVE routine either extends a symmetric heap block if the block is followed by a large enough free block or copies the contents of the existing block to a larger block and returns a status code indicating
9. OPENSHMEM LIBRARY API

that the block was moved. This routine also can reduce the size of a block if the new length is less than the old length. All PEs in a program must call SHPCLMOVE with the same value of addr to maintain symmetric heap consistency; if any PEs are missing, the program hangs.

Return Values

Error Codo		Condition
		Length is not an integer greater than 0
-1 -2		No more memory is available from the system (checked if the
-2		request cannot be satisfied from the available blocks on the sym-
		metric heap).
-3		Address is outside the bounds of the symmetric heap
-4		Block is already free.
-5		Address is not at the beginning of a block.
Notes		
None.		
1,010		
035 SHDDEALLC		
7.3.3 SIII DEALLC		
Returns a memory block to t	the symmetric l	heap.
	-	
SYNOPSIS		
depression start		
FORTRAN.		
POINTER (addr. A(1)	1)	
INTEGER errcode, at	port.	
CALL SHPDEALLC (add:	r, errcode, a	abort)
	, , , , , , , , , , , , , , , , , , , ,	depression and
		deprecation end —
DESCRIPTION		
Arguments		*
IN	addr	First word address of the block to deallocate.
OUT	errcode	Error code is 0 if no error was detected; otherwise, it is a negative inte-
		ger code for the type of error.
IN	abort	Abort code. Nonzero requests abort on error; 0 requests an error code.

API description

SHPDEALLC returns a block of memory (allocated using SHPALLOC) to the list of available space in the symmetric heap. To maintain symmetric heap consistency, all PEs in a program must call SHPDEALLC with the same value of *addr*; if any PEs are missing, the program hangs.

Return Values

Error Code

Condition

-1	Length is not an integer greater than 0
-2	No more memory is available from the system (checked if the request cannot be satisfied from the available blocks on the symmetric heap).
-3	Address is outside the bounds of the symmetric heap.
-4	Block is already free.
-5	Address is not at the beginning of a block.

Notes

None.

9.4 Team Management Routines

The PEs in an OpenSHMEM program can communicate either using point-to-point routines that specify the PE number of the target PE or using collective routines which operate over some predefined set of PEs. Teams in OpenSHMEM allow programs to group subsets of PEs for collective communications and provide a contiguous reindexing of the PEs within that subset that can be used in point-to-point communication.

An OpenSHMEM team is a set of PEs defined by calling a specific team split routine with a parent team argument and other arguments to further specify how the parent team is to be split into one or more new teams. A team created by a *shmem_team_split_** routine can be used as the parent team for a subsequent call to a team split routine. A team persists and can be used for multiple collective routine calls until it is destroyed by *shmem_team_destroy*.

Every team must have a least one member. Any attempt to create a team over an empty set of PEs will result in no new team being created.

A "team handle" is an opaque object with type *shmem_team_t* that is used to reference a defined team. Team handles are created by one of the team split routines and destroyed by the team destroy routine. Team handles have local semantics only. That is, team handles should not be stored in shared variables and used across other PEs. Doing so will result in undefined behavior.

By default, OpenSHMEM creates predefined teams that will be available for use once the routine *shmem_init* has been called. See Section 7 for a description of all predefined team handles provided by OpenSHMEM. Predefined *shmem_team_t* handles can be used as the parent team when creating new OpenSHMEM teams.

Every PE is a member of the default team, which may be referenced through the team handle *SHMEM_TEAM_WORLD*, and its number in the default team is equal to the value of its PE number as returned by *shmem_my_pe*.

A special team handle value, *SHMEM_TEAM_NULL*, may be used to indicate that a returned team handle is not valid. This value can be tested against to check for successful split operations and can be assigned to user declared team handles as a sentinel value.

Teams that are created by a *shmem_team_split_** routine may be provided a configuration argument that specifies options that may affect a team's capabilities and may allow for optimized performance. This configuration argument is of type *shmem_team_config_t*, which is detailed further in Section 9.4.3.

9.4.1 SHMEM_TEAM_MY_PE

Returns the number of the calling PE within the provided team.

SYNOPSIS

```
45 C/C++:
46 int shmem_team_my_pe(shmem_team_t team);
```

DESCRIPTION

Arguments

IN

A valid OpenSHMEM team handle.

API description

The *shmem_team_my_pe* function returns the number of calling PE within the provided team. The number will be a value between 0 and N-1, for a team of size N. Each member of the team has a unique number. For the team *SHMEM_TEAM_WORLD*, this will return the same value as *shmem_my_pe*. Error checking will be done to ensure a valid team handle is provided. Errors will result in a return value less than 0.

- design feedback requested -

team

Return Values

The number of the calling PE within the provided team, or a value less than θ if the team handle is invalid.

Notes

None.

9.4.2 SHMEM_TEAM_N_PES

Returns the total number of PEs in the provided team.

team

SYNOPSIS

C/C++:
int shmem_team_n_pes(shmem_team_t team);

DESCRIPTION

Arguments

IN

A valid OpenSHMEM team handle.

API description

The *shmem_team_n_pes* function returns the number of PEs in the team. This will always be a value between 1 and N, where N is the total number of PEs accessible to the OpenSHMEM program. For the team *SHMEM_TEAM_WORLD*, this will return the same value as *shmem_n_pes*.

All PEs in the team will get back the same value for the team size.

Error checking will be done to ensure a valid team handle is provided. Errors will result in a return value less than 0.

- design feedback requested

Return Values

Total number of PEs in the provided team, or a value less than 0 if the team handle is invalid.

Notes None. 33

A structure type representing team configuration arguments SYNOPSIS C/C++: typedef struct (f f distribute_collocations; int return_local_limit; int return_limit; int return_local_limit; int return_local_limit; int return_local_limit; int return_local_limit; int return_local_limit; int return_limit; int return_lint return; int return_limit; int return; int return;	9.4.3 SHMEM_TEAM_CONF	_ T
<pre>SYNOPSIS C/C++: typedef stratup { find disable_collectives; int resum_ticoal_limit; int resum_ticoad_limit; int resum_t</pre>	A structure type representing team	nfiguration arguments
<pre>CVC+:: typedef struct { fit (isbble_collectives; for (isbble_collective; for (isbble_collective;</pre>	SYNOPSIS	
<pre>typedef struct { int diable_collectives; int (int cliable_collectives; int (int cliable_collectives; int (int cliable_collectives; int (int cliable_collectives; int (int cliable_collective); int (int cliablecollective); int (int cliable_collective); int (int cliable_coll</pre>	C/C++:	
<pre>int diable.collectives; ist num_threads; ist num_thr</pre>	typedef struct {	
<pre>int roturn_local_limit; int num_thread; int num_thread; int num_thread; int num_thread; int num_thread configuration argument acts as both input and output to the shneen_team_split* rotuines. As an input, it specifies the requested capabilities of the team to be created. As an output, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is not output, the specifies the requested capabilities of the team to be created. When onscence, the team will not support collective operations, which allows implementations to reduce team creation overheads. The return_local_limit member controls whether, after a failed team creation, the team configuration argument is updated with the locally restrictive parameters). The return_local_limit member controls whether, after a failed team creation, the team configuration argument is updated with the locally restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Suppose Cuc+: [Cuc+: [Cuc+: [Must ream_unt_team_tteam_tteam_config_t +team]; [Must reamconfig_t teamducconfig_t shnem_tteam_config_t +team]; [Must reamconfig_t shnem_tteam_config_t +team]; [Must reamconfig_t shnem_tteam_config_t shn</pre>	int disable_collective	
<pre>pitr run_threads; j shmm_team_config_t; API description A can configuration argument acts as both input and output to the shmem_team_split_* outputs. As an input, it specifies the requested capabilities of the team to be created. As an output, the configuration argument is confluentionally updated on whether team creation or successful, it is configuration argument is not modified; if unsuccessful, it is updated to specify the limiting configuration parameter(s). The disable_collectives member allows for teams to be created without support for collective communica- tions, which allows implementations to reduce team creation overheads. The <i>neutrom_local_limit</i> member controls whether, after a failed team creation, the team configuration argu- ment is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameter(s) across the PEs of the new team. The most restrictive parameter(s) or the most restrictive parameter(s). Act senter Act a sinter Act senter Act senter</pre>	<pre>int return_local_limit</pre>	
<pre>> shmen_team_config_t; API description A team configuration argument acts as both input and output to the shmem_team_split_* routines. As an input, it specifies the requested capabilities of the team to be created. As an output, the configuration argument is not modified; if unsuccessful, it is updated to specify the limiting configuration parameter(s). The disable_collectives member allows for teams to be created without support for collective communica- tions, which allows implementations to reduce team creation overheads. The team will not support collective operations, which allows implementations to reduce team creation overheads. The return_local_limit member controls whether, after a failed team creation, whet nean will not support collective operations, which allows implementations to reture team creation overheads. The return_local_limit member controls whether, after a failed team creation, whet is updated to specify the parameters are returned; otherwise, the locally restrictive parameters are returned; of therwise, the locally restrictive parameters are returned; otherwise, the locally restrictive parameters are</pre>	<pre>int num_threads;</pre>	
API description A team configuration argument acts as both input and output to the shinkem_team_split_* routines. As an input, it specifies the requested capabilities of the team to be created. As an output, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is not modified; if unsuccessful, it is updated to specify the limiting configuration parameters. In <i>elivable_collectives</i> member allows for teams to be created. When nonzero, the team will not support collective parameters and the team to certain overheads. The return_local_limit member controls whether, after a failed team creation, the team configuration argument is updated with the locally restrictive parameters) or the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned; otherwise, the local support and the new team. If many that are an instead and the restrictive parameters are returned; otherwise, the local support and the restrictive parameters are returned; otherwise, the local support and the rest are an instead and the restrictive parameters are returned; otherwise, the local support and the restrictive parameters are returned; otherwise, the local supportex and the restrictive parameters are retu	<pre>} shmem_team_config_t;</pre>	
A team configuration argument acts as both input and output to the <i>shmem_team_split_*</i> routines. As an input, it specifies the requested capabilities of the team to be created. As an output, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration parameter(s). The <i>disable_collectives</i> member allows for teams to be created without support for collective communications, which allows implementations to reduce team creation overheads. When nonzero, the team will not support collective operations, which allows implementations to reduce team creation overheads. The <i>return_local_limit</i> member controls whether, after a failed team creation, the team configuration argument is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameter(s) or connective start of the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes Notes Notes Note: VC++: Void shmem_team_get_config(shmem_team_t team, shmem_team_config_t +team); DESCRIPTION Arguments IN team Get_config (shmem_team_t team, shmem_team_config_t +team); DESCRIPTION Arguments IN team Availd OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. Arguments IN team to config and Availd OpenSHMEM team handle. DUT config A pointer to the configuration parameters for the new team.	API description	
<pre>a input, it specifies the requested capabilities of the team to be created. As an output, the configuration argument is not modified; if unsuccessful, it is updated to specify the limiting configuration parameter(s). The disable_collectives member allows for teams to be created without support for collective communica- tions, which allows implementations to reduce team creation overheads for those teams. When its value is zero, it specifies that the team should have collectives enabled. When nonzero, the team will not support collective operations, which allows implementations to reduce team creation overheads. The return_local_limit member controls whether, after a failed team creation overheads. The return_local_limit member controls whether, after a failed team creation, the team will not support collective operameters) across the PEs of the new team. When its value is zero, the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. Arguments N team Avalid OpenSHMEM team handle. OUT config A valid OpenSHMEM team handle. Notes Note</pre>	A team configuration a	ument acts as both input and output to the shmem team split * routines. As
<pre>argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is conditionally updated on whether team creation is successful. If successful, the configuration argument is conditionally updated on whether team creation is updated to specify the limiting configuration parameter(s). The disable_collectives member allows inplementations to reduce team creation overheads. The return_local_limit member controls whether, after a failed team creation, the team configuration argument is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The unn_hreads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. C/C++: Void shmem_team_get_config(shmem_team_t team, shmem_team_config_t +team); ESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description</pre>	an input it specifies th	requested canabilities of the team to be created. As an output, the configuration
<pre>argument is not modified; if unsuccessful, it is updated to specify the limiting configuration parameter(s). The disable_collectives member allows for teams to be created without support for collective communica- tions, which allows implementations to reduce team creation overheads for those teams. When its value is zero, it specifies that the team should have collectives enabled. When nonzero, the team oull not support collective operations, which allows implementations to reduce team creation overheads for those teams. When its value is zero, it specifies that the team should have collectives enabled. When nonzero, the team oull not support collective operations, which allows implementations to reduce team creation overheads for the east east. The return_local_limit member controls whether, after a failed team creation, the team configuration argu- ment is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. VA.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SynOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description </pre>	argument is conditional	undated on whether team creation is successful. If successful, the configuration
The disable_collectives member allows for teams to be created without support for collective communica- tions, which allows implementations to reduce team creation overheads for those teams. When its value is zero, it specifies that the team should have collectives enabled. When nonzero, the team will not support collective operations, which allows implementations to reduce team creation overheads. The <i>return_local_limit</i> member controls whether, after a failed team creation, the team configuration argu- ment is updated with the locally restrictive parameter(s) or the most restrictive parameters) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The <i>num_threads</i> member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. OLA4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t +team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	argument is not modifie	if unsuccessful, it is updated to specify the limiting configuration parameter(s).
It is unable_contents inferiors incluses for tore actacted without support for concerner to university of the concerner of the value is zero, it specifies that the team should have collectives enabled. When nonzero, the team will not support collective operations, which allows implementations to reduce team creation overheads. The return_local_limit member controls whether, after a failed team creation, the team configuration argument is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes Notes None. C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t +team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team.	The disable collective	the allows for teams to be created without support for collective communication
<pre>build, which arrows implementations to reduce realifected or brieded with the set any should have collectives enabled. When nonzero, the team will not support collective operations, which allows implementations to reduce team creation overheads. The return_local_limit member controls whether, after a failed team creation, the team configuration argu- ment is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. 4.4.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team YNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description</pre>	tions which allows im	mentations to reduce team creation overheads for those teams. When its value is
Letter is spectres in the team should have concluses enhanced, which holzero, the team configuration argument is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. V4.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team.	tions, which anows fill	the neurons to reduce team creation overheads for those teams, when its value is
The return_local_limit member controls whether, after a failed team creation overheads. The return_local_limit member controls whether, after a failed team creation the team configuration argu- ment is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. V.4.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t +team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	collective operations.	ch allows implementations to reduce team creation overheads
The return_local_timit member controls whether, after a failed team oreation, the team configuration argu- ment is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The <i>num_threads</i> member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. 2.4.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t +team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description		
<pre>ment is updated with the locally restrictive parameter(s) or the most restrictive parameter(s) across the PEs of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description</pre>	The return_local_limit	ember controls whether, after a failed team creation, the team configuration argu-
of the new team. When its value is zero, the most restrictive parameters are returned; otherwise, the locally restrictive parameters are returned. The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. .4.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team YNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team OUT config A pointer to the configuration parameters for the new team.	ment is updated with th	ocally restrictive parameter(s) or the most restrictive parameter(s) across the PEs
restrictive parameters are returned. The <i>num_threads</i> member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. None. A.4.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	of the new team. When	s value is zero, the most restrictive parameters are returned; otherwise, the locally
The num_threads member specifies the number of threads that will create contexts from the new team. It must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. A.4.4 SHMEM_TEAM_GET_CONFIG Attem the configuration parameters of a given team YNOPSIS C/C++: void shmem_team_get_config(shmem_team_t_team, shmem_team_config_t +team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	restrictive parameters a	returned.
must have a nonnegative value. See Section 9.5 for more on communication contexts and Section 9.5.2 for team-based context creation. Notes None. 4.4 SHMEM_TEAM_GET_CONFIG eturn the configuration parameters of a given team YNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); ESCRIPTION Arguments IN team OUT config A pointer to the configuration parameters for the new team.	The <i>num_threads</i> mem	r specifies the number of threads that will create contexts from the new team. It
team-based context creation. Notes None. A.4. SHMEM_TEAM_GET_CONFIG eturn the configuration parameters of a given team YNOPSIS C/C++: void shmem_team_get_config (shmem_team_t team, shmem_team_config_t *team); ESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team.	must have a nonnegativ	value. See Section 9.5 for more on communication contexts and Section 9.5.2 for
Notes None. A.4. SHMEM_TEAM_GET_CONFIG eturn the configuration parameters of a given team EVOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); ESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	team-based context cre	on.
Notes None. A.4 SHMEM_TEAM_GET_CONFIG Leturn the configuration parameters of a given team EVENUAL SHOPSIS C/C++: Void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); ESECRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description		
None. P.4.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN LN OUT config A pointer to the configuration parameters for the new team. API description	Notes	
A.4 SHMEM_TEAM_GET_CONFIG eturn the configuration parameters of a given team YNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); ESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	None	
A.4. SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	Tone.	
A.4. SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description		
A.4 SHMEM_TEAM_GET_CONFIG Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description		
Return the configuration parameters of a given team SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	0.4.4 SHMEM_TEAM_GET_	DNFIG
SYNOPSIS C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	Return the configuration parameter	of a given team
C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	SVNOPSIS	
C/C++: void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description	51101010	
void shmem_team_get_config(shmem_team_t team, shmem_team_config_t *team); DESCRIPTION Arguments IN team A valid OpenSHMEM team handle. OUT OUT config A pointer to the configuration parameters for the new team. API description Image: Shmem_team_team_team_config_t *team);	C/C++:	
DESCRIPTION Arguments IN team OUT config A pointer to the configuration parameters for the new team.	void shmem_team_get_conf	<pre>(shmem_team_t team, shmem_team_config_t *team);</pre>
Arguments A valid OpenSHMEM team handle. IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description Image: Second Seco		
Arguments IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team. API description Image: Configuration parameters for the new team.		
ArgumentsA valid OpenSHMEM team handle.INteamA valid OpenSHMEM team handle.OUTconfigA pointer to the configuration parameters for the new team.API descriptionImage: Configuration parameters for the new team.	DESCRIPTION	
ArgumentsINteamOUTconfigA pointer to the configuration parameters for the new team.API description		
IN team A valid OpenSHMEM team handle. OUT config A pointer to the configuration parameters for the new team.	Arguments	
OUTconfigA pointer to the configuration parameters for the new team.API description	IN team	A valid OpenSHMEM team handle.
API description	OUT conf	A pointer to the configuration parameters for the new team.
API description		
	API description	
	Ai i ucocription	

- design feedback requested -

A library implementation must apply all requested options to a team, even in the event that the library does not make optimizations based on these options. For example, suppose library implementation must always create teams with the same overhead, no matter if the program disables collective support during team creation. The library must still enable the *SHMEM_TEAM_NOCOLLECTIVE* option when it is requested, so that the OpenSHMEM program will be portable across implementations.

All PEs in the team will get back the same parameter values for the team options. If the *team* argument does not specify a valid team, the behavior is undefined.

Return Values

None.

Notes

A use case for this function is to determine whether a given team can support collective operations by testing for the *SHMEM_TEAM_NOCOLLECTIVE* option. When teams are created without support for collectives, they may still use point to point operations to communicate and synchronize. So programmers may wish to design frameworks with functions that provide alternative algorithms for teams based on whether they do or do not support collectives.

9.4.5 SHMEM_TEAM_TRANSLATE

Translates a given PE number to the corresponding PE number in another team.

SYNOPSIS

C/C++: int shmem_team_translate_pe(shmem_team_t src_team, int src_pe, shmem_team_t dest_team);

DESCRIPTION

rguments	
IN	src_team
IN	src_pe
IN	dest_team

A valid SHMEM team handle. A PE number in src_team. A valid SHMEM team handle.

API description

The *shmem_team_translate_pe* function will translate a given PE number to the corresponding PE number in another team. Specifically, given the *src_pe* in *src_team*, this function returns that PE's number in *dest_team*. If *src_pe* is not a member of both the *src_team* and *dest_team*, a value less than 0 is returned. If *SHMEM_TEAM_WORLD* is provided as the *dest_team* parameter, this function acts as a global PE number translator and will return the corresponding *SHMEM_TEAM_WORLD* number. This may be useful when performing point-to- point operations between PEs in a subset, as point-to-point operations that do not take a context argument require the global *SHMEM_TEAM_WORLD* PE number. Error checking will be done to ensure valid team handles are provided. Errors will result in a return value less than 0.

predefined

teams

be

The

2 — design feedback requested **Return Values** The specified PE's number in the *dest_team*, or a value less than 0 if any team handle arguments are invalid or the *src_pe* is not in both the source and destination teams. Notes None. 10 11 12 9.4.6 SHMEM_TEAM_SPLIT_STRIDED 13 14 Create a new OpenSHMEM team from a subset of the existing parent team PEs, where the subset is defined by the PE 15 triplet (*PE_start*, *PE_stride*, and *PE_size*) supplied to the function. 16 17 **SYNOPSIS** 18 C/C++: 19 void shmem_team_split_strided(shmem_team_t parent_team, int PE_start, int PE_stride, 20 int PE_size, shmem_team_config_t *config, long config_mask, shmem_team_t *new_team); 21 22 DESCRIPTION 23 24 Arguments 25 valid SHMEM IN parent_team А 26 *SHMEM_TEAM_WORLD* or *SHMEM_TEAM_NODE* may 27 used, or any team created by the user. 28 The lowest PE number of the subset of PEs from the parent team that IN PE_start 29 will form the new team 30 IN PE stride The stride between team PE numbers in the parent team that comprise 31 the subset of PEs that will form the new team. The number of PEs from the parent team in the subset of PEs that will IN PE_size 33 form the new team. 34 35 **INOUT** config A pointer to the configuration parameters for the new team. 36 IN The bitwise mask representing the set of configuration parameters to config_mask 37 use from config. 38 OUT A new OpenSHMEM team handle, representing a PE subset of all the new_team 39 PEs in the parent team that is created from the PE triplet provided. 40

API description

41 42

43 44

45

46

47

48

The shmem team split strided function is a collective routine. It creates a new OpenSHMEM team from a subset of the existing parent team, where the subset is defined by the PE triplet (*PE_start*, *PE_stride*, and PE_size) supplied to the function.

team.

This function must be called by all processes contained in the PE triplet specification. It may be called by additional PEs not included in the triplet specification, but for those processes a *new_team* value of *SHMEM_TEAM_NULL* is returned. All calling processes must provide the same values for the PE triplet. This function will return a *new_team* containing the PE subset specified by the triplet, and ordered by the existing global PE number. None of the parameters need to reside in symmetric memory.

The *config* argument specifies team configuration parameters, which are described in Section 9.4.3.

The *config_mask* argument is a bitwise mask representing the set of configuration parameters to use from *config_mask* value of 0 indicates that all the field members of *config* should be used. Individual field masks can be combined through a bitwise OR operation of the following library constants:

SHMEM_TEAM_NOCOLLECTIVEThe team should be created using the value of the dis-
able_collectives member of the configuration parameter config.SHMEM_TEAM_LOCAL_LIMITThe team should be created using the value of the re-
turn_local_limit member of the configuration parameter config.SHMEM_TEAM_NUM_THREADSThe team should be created using the value of the num_threads
member of the configuration parameter config.

Error checking will be done to ensure a valid PE triplet is provided, and also to determine whether a valid team handle is provided for the parent team.

If *parent_team* is equal to *SHMEM_TEAM_NULL*, then *new_team* will be assigned the value *SHMEM_TEAM_NULL*. Otherwise, if *parent_team* is an invalid team handle, the behavior is undefined. If *new_team* cannot be created, it will be assigned the value *SHMEM_TEAM_NULL*.

Return Values

None.

Notes

It is important to note the use of the less restrictive *PE_stride* argument instead of *logPE_stride*. This method of creating a team with an arbitrary set of PEs is inherently restricted by its parameters, but allows for many additional use-cases over using a *logPE_stride* parameter, and may provide an easier transition for existing OpenSHMEM programs to create and use OpenSHMEM teams.

See the description of team handles and predefined teams at the top of Section 9.4 for more information about semantics and usage.

EXAMPLES

9.4.7 SHMEM_TEAM_SPLIT_2D

Create two new teams by splitting an existing parent team into two subsets based on a 2D Cartesian space defined by the *xrange* argument and a *y* dimension derived from *xrange* and the parent team size. These ranges describe the Cartesian space in *x*- and *y*-dimensions.

SYNOPSIS

C/C++:

```
void shmem_team_split_2d(shmem_team_t parent_team, int xrange,
    shmem_team_config_t *xaxis_config, long xaxis_mask, shmem_team_t *xaxis_team,
    shmem_team_config_t *yaxis_config, long yaxis_mask, shmem_team_t *yaxis_team);
```

1	Arguments		
2	IN	parent_team	A valid OpenSHMEM team. Any predefined teams, such as
3			SHMEM_TEAM_WORLD, may be used, or any team created by the
4			user.
5	IN	xrange	A nonnegative integer representing the number of elements in the first
6			dimension.
7	INOUT	varia config	A pointer to the configuration perspectate for the new x axis team
8	INOUT	xaxis_conjig	A pointer to the configuration parameters for the new x-axis team.
9	IN	xaxis mask	The bitwise mask representing the set of configuration parameters to
10			use from <i>xaxis</i> config.
11		•	
12	OUT	xaxis_team	A new PE team handle representing a PE subset consisting of all the DEs that have the same accordinate along the x axis as the calling DE
13			PES that have the same coordinate along the x-axis as the canning PE.
14	INOUT	yaxis_config	A pointer to the configuration parameters for the new y-axis team.
15			
16	IN	yaxis_mask	The bitwise mask representing the set of configuration parameters to
17			use from <i>yaxis_config</i> .
19	OUT	vaxis team	A new PE team handle representing a PE subset consisting of all the
10			PEs that have the same coordinate along the y-axis as the calling PE.
19			

API description

The *shmem_team_split_2d* routine is a collective routine. It creates two new teams by splitting an existing parent team into up to two subsets based on a 2D Cartesian space. The user provides the size of the *x* dimension, which is then used to derive the size of the *y* dimension based on the size of the parent team. The size of the *y* dimension will be equal to $ceiling(N \div xrange)$, where *N* is the size of the parent team. In other words, $xrange \times yrange \ge N$, so that every PE in the parent team has a unique (x, y) location the 2D Cartesian space.

After the split operation, each of the new teams will contain all PEs that have the same coordinate along the *x*-axis and *y*-axis, respectively, as the calling PE. The PEs are numbered in the new teams based on the position of the PE along the given axis.

Any valid **OpenSHMEM** team can be used as the parent team. This routine must be called by all PEs in the parent team. The value of *xrange* must be nonnegative and all PEs in the parent team must pass the same value for *xrange*. None of the parameters need to reside in symmetric memory.

The *xaxis_config* and *yaxis_config* arguments specify team configuration parameters for the *x*- and *y*-axis teams, respectively. These parameters are described in Section 9.4.3. All PEs that will be in the same resultant team must specify the same configuration parameters. The PEs in the parent team *do not* have to all provide the same parameters for new teams.

The *xaxis_mask* and *xaxis_mask* arguments are a bitwise masks representing the set of configuration parameters to use from *xaxis_config* and *yaxis_config*, respectively. A mask value of 0 indicates that all the field members of the configuration parameter argument should be used. Individual field masks can be combined through a bitwise OR operation of the following library constants:

SHMEM_TEAM_NOCOLLECTIVE	The team should be created using the value of the dis-
	able_collectives member of the respective configuration param-
	eter.
SHMEM_TEAM_LOCAL_LIMIT	The team should be created using the value of the re-
	<i>turn_local_limit</i> member of the respective configuration param-
	eter.
SHMEM_TEAM_NUM_THREADS	The team should be created using the value of the <i>num_threads</i>
	member of the respective configuration parameter.

If *parent_team* is equal to *SHMEM_TEAM_NULL*, both *xaxis_team* and *yaxis_team* will be assigned the value *SHMEM_TEAM_NULL*. Otherwise, if *parent_team* is an invalid team handle, the behavior is undefined. If either team cannot be created, that team will be assigned the value *SHMEM_TEAM_NULL*.

Return Values

None.

Notes

Since the split may result in a 2D space with more points than there are members of the parent team, there may be a final, incomplete row of the 2D mapping of the parent team. This means that the resultant *x*-axis teams may vary in size by up to 1 PE, and that there may be one resultant *y*-axis team of smaller size than all of the other *y*-axis teams.

The following grid shows the 12 teams that would result from splitting a parent team of size 10 with *xrange* of 3. The numbers in the grid cells are the PE numbers in the parent team. The rows are the *y*-axis teams. The columns are the *x*-axis teams.

	x=0	x=1	x=2	
y=0	0	1	2	
y=1	3	4	5	
y=2	6	7	8	
y=3	9			

It would be legal, for example, if PEs 0, 3, 6, 9 specified a different value for *xaxis_config* than all of the other PEs, as long as the configuration parameters match for all PEs in each of the new teams.

See the description of team handles and predefined teams at the top of section 9.4 for more information about team handle semantics and usage.

EXAMPLES

9.4.8 SHMEM_TEAM_DESTROY

Destroys existing team.

SYNOPSIS

C/C++:

int shmem_team_destroy(shmem_team_t team);

team

DESCRIPTION

Arguments

IN

A valid OpenSHMEM team handle.

API description

The *shmem_team_destroy* function destroys an existing team. This is a collective call, in which every member of the team being destroyed needs to participate. This will free all internal memory structures associated with the team and invalidate the team handle. Upon return, the team handle can no longer be used for team API calls.

It is considered erroneous to free SHMEM_TEAM_WORLD or any other predefined team. Error checking will be done to ensure a valid team handle is provided. Errors will result in a return value less than 0.

- design feedback requested -

Return Values

On success, the function will return 0. Otherwise a value less than θ will be returned.

Notes

None.

9.5 Communication Management Routines

All OpenSHMEM RMA, AMO, and memory ordering routines are performed on a communication context. The communication context defines an independent ordering and completion environment, allowing users to manage the overlap of communication with computation and also to manage communication operations performed by separate threads within a multithreaded PE. For example, in single-threaded environments, contexts may be used to pipeline 20 communication and computation. In multithreaded environments, contexts may additionally provide thread isolation, eliminating overheads resulting from thread interference.

Context handles are of type *shmem_ctx_t* and are valid for language-level assignment and equality comparison. A handle to the desired context is passed as an argument in the C shmem_ctx_* and type-generic API routines. API routines that do not accept a context argument operate on the default context. The default context can be used explicitly through the SHMEM CTX DEFAULT handle.

26 Every communication context is associated with a team. This association is established at context creation. Com-27 munication contexts created by shmem_ctx_create are associated with the default team, while contexts created by shmem team create ctx are associated with a team specified at context creation. The default context is associated 28 with the default team. A context's associated team specifies the set of PEs over which PE-specific routines that operate 29 on a communication context, explicitly or implicitly, are performed. All point-to-point routines that operate on this 30 context will do so with respect to the team-relative PE numbering of the associated team. 31

9.5.1 SHMEM_CTX_CREATE

Create a communication context locally.

SYNOPSIS

C/C++:

```
39
40
41
```

42 43

int shmem_ctx_create(long options, shmem_ctx_t *ctx);

DESCRIPTION

44	Arguments		
45	IN	options	The set of options requested for the given context. Multiple options
46			may be requested by combining them with a bitwise OR operation; otherwise, 0 can be given if no options are requested.
47	OUT	ctx	A handle to the newly created context.
48			-

2

10

11

16

17

18

19

21

22

23

24

25

32

33 34

> 35 36

API description

The *shmem_ctx_create* routine creates a new communication context and returns its handle through the *ctx* argument. If the context was created successfully, a value of zero is returned; otherwise, a nonzero value is returned. An unsuccessful context creation call is not treated as an error and the OpenSHMEM library remains in a correct state. The creation call can be reattempted with different options or after additional resources become available.

A newly created communication context has an initial association with the default team. All OpenSHMEM routines that operate on this context will do so with respect to the associated PE team. That is, all point-to-point routines operating on this context will use team-relative PE numbering.

By default, contexts are *shareable* and, when it is allowed by the threading model provided by the Open-SHMEM library, they can be used concurrently by multiple threads within the PE where they were created. The following options can be supplied during context creation to restrict this usage model and enable performance optimizations. When using a given context, the application must comply with the requirements of all options set on that context; otherwise, the behavior is undefined. No options are enabled on the default context.

SHMEM_CTX_SERIALIZED	The given context is shareable; however, it will not be used by multiple threads concurrently. When the <i>SHMEM_CTX_SERIALIZED</i> option is set, the user must ensure that operations involving the given context are serialized by the application.
SHMEM_CTX_PRIVATE	The given context will be used only by the thread that created it.
SHMEM_CTX_NOSTORE	Quiet and fence operations performed on the given context are not required to enforce completion and ordering of memory store operations. When ordering of store operations is needed, the ap- plication must perform a synchronization operation on a context without the <i>SHMEM_CTX_NOSTORE</i> option enabled.
Return Values	
Zero on success and nonzero otherw	vise
Notes None.	
9.5.2 SHMEM_TEAM_CREATE_CTX	
Create a communication context from a team.	
SYNOPSIS	
C/C++:	
<pre>int shmem_team_create_ctx(shmem_tea</pre>	<pre>am_t team, long options, shmem_ctx_t *ctx);</pre>
DESCRIPTION	
Arguments	
IN team	A handle to the specified PE team.

1 2 3	IN	options	The set of options requested for the given context. Multiple options may be requested by combining them with a bitwise OR operation; otherwise, 0 can be given if no options are requested.
4	OUT	ctx	A handle to the newly created context.
5	API description		
7 8 9 10 11	The <i>shmem_i</i> the <i>ctx</i> argum The <i>shmem_</i> <i>num_threads</i> the team was	team_create_ctx in the second	routine creates a new communication context and returns its handle through is created from the team specified by the <i>team</i> argument. routine must be called by no more threads than were specified by the <i>chmem_team_config_t</i> configuration parameters that were specified when
12 13 14	In addition to the same retu	the team, the <i>shn</i> rn conditions as the	<i>nem_team_create_ctx</i> routine accepts the same arguments and provides all ne <i>shmem_ctx_create</i> routine.
15 16 17 18 19	Return Values Zero on succe	ess and nonzero of	therwise.
20	Notes None		
21 22 23			
24 9. 25 D 26 D 27 S	.5.3 SHMEM_CTX_ restroy a locally created YNOPSIS	DESTROY	ontext.
29	C/C++:		
30 31	<pre>void shmem_ctx_c</pre>	lestroy(shmem_ct	<pre>tx_t ctx);</pre>
32 33 D 34 35	ESCRIPTION Arguments IN	ctx	Handle to the context that will be destroyed
36 37		CIA	mandie to the context that will be desitoyed.
38	API description		
39 40 41 42	shmem_ctx_a shmem_team been destroye	<i>destroy</i> destroys _ <i>create_ctx</i> . It is ed, for example wh	a context that was created by a call to <i>shmem_ctx_create</i> or a the user's responsibility to ensure that the context is not used after it has then the destroyed context is used by multiple threads. This function performs
43 44 45	an implicit qu If <i>ctx</i> is a han	the to the default	ne given context before it is freed. context, the behavior is undefined.
46 47 48	Return Values None.		

Notes

It is invalid to pass *SHMEM_CTX_DEFAULT* to this routine. Destroying a context makes it impossible for the user to complete communication operations that are pending on that context. This includes nonblocking communication operations, whose local buffers are only returned to the user after the operations have been completed. An implicit quiet is performed when freeing a context to avoid this ambiguity.

A context with the *SHMEM_CTX_PRIVATE* option enabled must be destroyed by the thread that created it.

EXAMPLES

The following example demonstrates the use of contexts in a multithreaded *C11* program that uses OpenMP for threading. This example shows the shared counter load balancing method and illustrates the use of contexts for thread isolation.

```
#include <stdio.h>
#include <shmem.h>
long pwrk[SHMEM_REDUCE_MIN_WRKDATA_SIZE];
long psync[SHMEM_REDUCE_SYNC_SIZE];
long task_cntr = 0; /* Next task counter */
long tasks_done = 0; /* Tasks done by this PE */
long total_done = 0; /* Total tasks done by all PEs */
int main(void) {
    int tl, i;
    long ntasks = 1024; /* Total tasks per PE
    for (i = 0; i < SHMEM_REDUCE_SYNC_SIZE; i++</pre>
        psync[i] = SHMEM_SYNC_VALUE;
    shmem_init_thread(SHMEM_THREAD_MULTIPLE, &tl);
    if (tl != SHMEM_THREAD_MULTIPLE) shmem_global_exit(1);
    int me = shmem_my_pe();
    int npes = shmem_n_pes();
#pragma omp parallel reduction (+:tasks_done)
    {
        shmem_ctx_t ctx;
        int task_pe = me, pes_done = 0;
        int ret = shmem_ctx_create(SHMEM_CTX_PRIVATE, &ctx);
        if (ret != 0) {
            printf("%d: Error creating context (%d)\n", me, ret);
            shmem_global_exit(2);
        }
        /* Process tasks on all PEs, starting with the local PE. After
         * all tasks on a PE are completed, help the next PE. */
        while (pes_done < npes) {</pre>
            long task = shmem_atomic_fetch_inc(ctx, &task_cntr, task_pe);
            while (task < ntasks) {</pre>
                /* Perform task (task_pe, task) */
                tasks_done++;
                task = shmem_atomic_fetch_inc(ctx, &task_cntr, task_pe);
            }
            pes_done++:
            task_pe = (task_pe + 1) % npes;
        }
        shmem_ctx_destroy(ctx);
    }
```

2

3

4

5

6

9

10 11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

36

37

38

39

40

41

42

43

44

45

46

47

```
shmem_long_sum_to_all(&total_done, &tasks_done, 1, 0, 0, npes, pwrk, psync);
int result = (total_done != ntasks * npes);
shmem_finalize();
return result;
}
```

The following example demonstrates the use of contexts in a single-threaded C11 program that performs a summation reduction where the data contained in the *in_buf* arrays on all PEs is reduced into the *out_buf* arrays on all PEs. The buffers are divided into segments and processing of the segments is pipelined. Contexts are used to overlap an all-to-all exchange of data for segment *p* with the local reduction of segment *p*-1.

```
#include <stdio.h>
#include <stdlib.h>
#include <shmem.h>
#define LEN 8192 /* Full buffer length */
#define PLEN 512 /* Length of each pipeline stage */
int in_buf[LEN], out_buf[LEN];
int main(void) {
   int i, j, *pbuf[2];
   shmem_ctx_t ctx[2];
   shmem init();
   int me = shmem_my_pe();
   int npes = shmem_n_pes();
   pbuf[0] = shmem_malloc(PLEN * npes * sizeof(int));
   pbuf[1] = shmem_malloc(PLEN * npes * sizeof(int));
   int ret_0 = shmem_ctx_create(0, &ctx[0]);
   int ret_1 = shmem_ctx_create(0, &ctx[1]);
   if (ret_0 || ret_1) shmem_global_exit(1);
   for (i = 0; i < LEN; i++) {</pre>
        in_buf[i] = me; out_buf[i] = 0;
    }
   int p_idx = 0, p = 0; /* Index of ctx and pbuf (p_idx) for current pipeline stage (p) */
   for (i = 1; i <= npes; i++)</pre>
        shmem_put_nbi(ctx[p_idx], &pbuf[p_idx][PLEN*me], &in_buf[PLEN*p],
                      PLEN, (me+i) % npes);
    /* Issue communication for pipeline stage p, then accumulate results for stage p-1 */
   for (p = 1; p < LEN/PLEN; p++) {</pre>
        p_idx ^= 1;
        for (i = 1; i <= npes; i++)</pre>
            shmem_put_nbi(ctx[p_idx], &pbuf[p_idx][PLEN*me], &in_buf[PLEN*p],
                           PLEN, (me+i) % npes);
        shmem_ctx_quiet(ctx[p_idx^1]);
        shmem_sync_all();
        for (i = 0; i < npes; i++)</pre>
            for (j = 0; j < PLEN; j++)</pre>
                out_buf[PLEN*(p-1)+j] += pbuf[p_idx^1][PLEN*i+j];
    }
   shmem_ctx_quiet(ctx[p_idx]);
   shmem_sync_all();
   for (i = 0; i < npes; i++)</pre>
        for (j = 0; j < PLEN; j++)</pre>
            out_buf[PLEN*(p-1)+j] += pbuf[p_idx][PLEN*i+j];
   shmem_finalize();
   return 0:
```

1

2

4

6

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29 30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

}

9.5.4 SHMEM_CTX_GET_TEAM

Retrieve the team associated with the communication context.

SYNOPSIS

C/C	++:	
int	<pre>shmem_ctx_get_team(shmem_ctx_t ctx,</pre>	<pre>shmem_team_t *team);</pre>

DESCRIPTION

Arguments		
IN	ctx	A handle to a communication context.
OUT	team	A pointer to a handle to the associated PE team.

API description

The *shmem_ctx_get_team* routine returns a handle to the team associated with the specified communication context *ctx*. The team handle is returned through the pointer argument *team*. If *ctx* is the default context or one created by a call to *shmem_ctx_create*, the returned team is the default team.

If *ctx* is an invalid context, the behavior is undefined.

If *team* is a null pointer, a value of *-1* is returned.

Return Values

Zero on success; otherwise, nonzero.

Notes None.

9.6 Remote Memory Access Routines

The *Remote Memory Access* (RMA) routines described in this section are one-sided communication mechanisms of the OpenSHMEM API. While using these mechanisms, the user is required to provide parameters only on the calling side. A characteristic of one-sided communication is that it decouples communication from the synchronization. One-sided communication mechanisms transfer the data but do not synchronize the sender of the data with the receiver of the data.

OpenSHMEM RMA routines are all performed on the symmetric objects. The initiator PE of the call is designated as *source*, and the PE in which memory is accessed is designated as *dest*. In the case of the remote update routine, *Put*, the origin is the *source* PE and the destination PE is the *dest* PE. In the case of the remote read routine, *Get*, the origin is the *dest* PE and the destination is the *source* PE.

Where appropriate compiler support is available, OpenSHMEM provides type-generic one-sided communication interfaces via C11 generic selection (C11 §6.5.1.1⁵) for block, scalar, and block-strided put and get communication. Such type-generic routines are supported for the "standard RMA types" listed in Table 3.

The standard RMA types include the exact-width integer types defined in *stdint.h* by *C99*⁶ §7.18.1.1 and *C11* §7.20.1.1. When the *C* translation environment does not provide exact-width integer types with *stdint.h*, an OpenSHMEM implementation is not required to provide support for these types.

	TYPE	TYPENAME	
	float	float	
	double	double	
	long double	longdouble	
	char	char	
	signed char	schar	
	short	short	
	int	int	
	long	long	
	long long	longlong	
	unsigned char	uchar	
	unsigned short	ushort	
	unsigned int	uint	
	unsigned long	ulong	
	unsigned long long	ulonglong	
	int8_t	int8	
	int16_t	int16	
	int32_t	int32	
	int64_t	int64	
	uint8_t	uint8	
	uint16_t	uint16	
	uint32_t	uint32	
	uint64_t	uint64	
	size_t	size	
Ť	ptrdiff_t	ptrdiff	

Table 3: Standard RMA Types and Names

9.6.1 SHMEM_PUT

The put routines provide a method for copying data from a contiguous local data object to a data object on a specified PE.

SYNOPSIS

40	C11:
41	<pre>void shmem_put(TYPE *dest, const TYPE *source, size_t nelems, int pe);</pre>
42	<pre>void shmem_put(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, size_t nelems, int pe);</pre>
43	where TYPE is one of the standard RMA types specified by Table 3.
44	C/C++:
45	<pre>void shmem_<typename>_put(TYPE *dest, const TYPE *source, size_t nelems, int pe);</typename></pre>
	<pre>void shmem_ctx_<typename>_put(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, size_t</typename></pre>
46	<pre>nelems, int pe);</pre>
17	

⁵Formally, the C11 specification is ISO/IEC 9899:2011(E).
⁶Formally, the C99 specification is ISO/IEC 9899:1999(E).

- deprecation start -

FORTRAN:

```
CALL SHMEM_CHARACTER_PUT(dest, source, nelems, pe)
CALL SHMEM_COMPLEX_PUT(dest, source, nelems, pe)
CALL SHMEM_DOUBLE_PUT(dest, source, nelems, pe)
CALL SHMEM_INTEGER_PUT(dest, source, nelems, pe)
CALL SHMEM_LOGICAL_PUT(dest, source, nelems, pe)
CALL SHMEM_PUT4(dest, source, nelems, pe)
CALL SHMEM_PUT8(dest, source, nelems, pe)
CALL SHMEM_PUT32(dest, source, nelems, pe)
CALL SHMEM_PUT64(dest, source, nelems, pe)
CALL SHMEM_PUT64(dest, source, nelems, pe)
CALL SHMEM_PUT128(dest, source, nelems, pe)
CALL SHMEM_PUT128(dest, source, nelems, pe)
CALL SHMEM_PUT128(dest, source, nelems, pe)
CALL SHMEM_PUTMEM(dest, source, nelems, pe)
CALL SHMEM_PUTMEM(dest, source, nelems, pe)
```

```
deprecation end -
```

DESCRIPTION

Arguments IN OUT	ctx dest	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> . Data object to be updated on the remote PE. This data object must be remotely accessible
IN IN IN	source nelems pe	 Data object containing the data to be copied. Number of elements in the <i>dest</i> and <i>source</i> arrays. <i>nelems</i> must be of type <i>size_t</i> for <i>C</i>. When using <i>Fortran</i>, it must be a constant, variable, or array element of default integer type. PE number of the remote PE. <i>pe</i> must be of type integer. When using <i>Fortran</i>, it must be a constant, variable, or array element of default integer type.
API description		

The routines return after the data has been copied out of the *source* array on the local PE. The delivery of data words into the data object on the destination PE may occur in any order. Furthermore, two successive put routines may deliver data out of order unless a call to *shmem_fence* is introduced between the two calls.

The dest and source data objects must conform to certain typing constraints, which are as follows:

		47
Routine	Data type of <i>dest</i> and <i>source</i>	49

1	shmem_putmem	<i>Fortran</i> : Any noncharacter type. <i>C</i> : Any data type. nelems is scaled in bytes.
3	shmem_put4, shmem_put32	Any noncharacter type that has a storage size equal to 32 bits.
4	shmem_put8	C: Any noncharacter type that has a storage size equal to 8 bits.
5		Fortran: Any noncharacter type that has a storage size equal to
6		64 bits.
7	shmem_put64 shmem_put128	Any noncharacter type that has a storage size equal to 04 bits.
8	SHMEM CHARACTER PUT	Elements of type character. <i>nelems</i> is the number of characters
9		to transfer. The actual character lengths of the <i>source</i> and <i>dest</i>
10		variables are ignored.
11	SHMEM_COMPLEX_PUT	Elements of type complex of default size.
12	SHMEM_DOUBLE_PUT	Elements of type double precision.
13	SHMEM_INTEGER_PUT	Elements of type Integer.
14	SHMEM_EOOICAL_FUT	Elements of type regl
15		Exements of type real.
16		
17	Return Values	
18	None.	
20		
20	Notes	
22	When using <i>Fortran</i> , data types mus	t be of default size. For example, a real variable must be declared as
23	REAL, REAL*4, or REAL(KIND=KIN	<i>VD(1.0)</i>). As of OpenSHMEM 1.2, the <i>Fortran</i> API routine <i>SHMEM_PUT</i>
24	has been deprecated, and entitel SHM	<i>EM_FOTS</i> of <i>SHMEM_FOT04</i> should be used in its place.
25		
26	EXAMPLES	
27		
28	The following <i>shmem_put</i> example is for 0	C11 programs:
29	<pre>#include <stdio.h> #include <shmom h=""></shmom></stdio.h></pre>	
30	#Include (Sinicht.if)	
31	int main (void)	
32	long source[10] = { 1, 2, 3, 4, 5	5, 6, 7, 8, 9, 10 };
33	<pre>static long dest[10]; shmem_init();</pre>	
34	<pre>int me = shmem_my_pe();</pre>	
35	<pre>if (me == 0) /* put 10 words into shmem put (dest, source, 10, 1)</pre>	o dest on PE 1 */
36	shmem_pur(dest, source, io, i) shmem_barrier_all(); /* sync send	der and receiver */
3/	printf("dest[0] on PE %d is %ld\r shmem finalize():	n", me, dest[0]);
20	return 0;	
40	}	
41		
42	9.6.2 SHMEM_P	
43		
44	Copies one data item to a remote PE.	
45	SYNOPSIS	
46	C11:	
47	<pre>void shmem_p(TYPE *dest, TYPE value,</pre>	, int pe);
48	void shmem_p(shmem_ctx_t ctx, TYPE	<pre>*dest, TYPE value, int pe);</pre>

1

3

6 7

18 19

20

21

26 27 28

29 30 31

32 33

34

36

37 38

39

40

41

42

43

44

45

46

47

48

where *TYPE* is one of the standard RMA types specified by Table 3.

C/C++:

void	l shmem_< TYPENAME >_p(TYPE *dest, TYPE value, int pe);	
void	<pre>i shmem_ctx_<typename>_p(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);</typename></pre>	

where TYPE is one of the standard RMA types and has a corresponding TYPENAME specified by Table 3.

DESCRIPTION

Arguments		
IN	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
OUT	dest	The remotely accessible array element or scalar data object which will receive the data on the remote PE.
IN	value	The value to be transferred to <i>dest</i> on the remote PE.
IN	pe	The number of the remote PE.

API description

These routines provide a very low latency put capability for single elements of most basic types.

As with *shmem_put*, these routines start the remote transfer and may return before the data is delivered to the remote PE. Use *shmem_quiet* to force completion of all remote *Put* transfers.

Return Values

None.

Notes

None.

EXAMPLES

The following example uses *shmem_p* in a *C11* program.

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

int main(void)

```
const double e = 2.71828182;
const double epsilon = 0.00000001;
static double f = 3.1415927;
shmem_init();
int me = shmem_my_pe();
if (me == 0)
   shmem_p(&f, e, 1);
shmem_barrier_all();
if (me == 1)
   printf("%s\n", (fabs(f - e) < epsilon) ? "OK" : "FAIL");
shmem_finalize();
return 0;
```

9.6.3 SHMEM_IPUT

Copies strided data to a specified PE.

SYNOPSIS

C11:	
void	<pre>shmem_iput(TYPE *dest, const TYPE *source, ptrdiff_t dst, ptrdiff_t sst, size_t nelems,</pre>
i	int pe);
void	<pre>shmem_iput(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, ptrdiff_t dst, ptrdiff_t</pre>
S	sst, size_t nelems, int pe);
where	e <i>TYPE</i> is one of the standard RMA types specified by Table 3.
C/C+	+:
void	<pre>shmem_<typename>_iput(TYPE *dest, const TYPE *source, ptrdiff_t dst, ptrdiff_t sst,</typename></pre>
s	<pre>size_t nelems, int pe);</pre>
void	<pre>shmem_ctx_<typename>_iput(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, ptrdiff_t dst, ptrdiff_t sst, size_t nelems, int pe);</typename></pre>
where	e TYPE is one of the standard RMA types and has a corresponding TYPENAME specified by Table 3.
void	<pre>shmem_iput<size>(void *dest, const void *source, ptrdiff_t dst, ptrdiff_t sst, size_t</size></pre>
r	nelems, int pe);
void	<pre>shmem_ctx_iput<size>(shmem_ctx_t ctx, void *dest, const void *source, ptrdiff_t dst,</size></pre>
F	<pre>ptrdiff_t sst, size_t nelems, int pe);</pre>
where	e SIZE is one of 8, 16, 32, 64, 128.
— de	precation start
FOR	TRAN:
INTEG	ER dst, sst, nelems, pe
CALL	SHMEM_COMPLEX_IPUT(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_DOUBLE_IPUT(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_INTEGER_IPUT(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_IPUT4(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_IPUT8(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_IPUT32(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_IPUT64(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_IPUT128(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_LOGICAL_IPUT(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_REAL_IPUT(dest, source, dst, sst, nelems, pe)
	deprecation end –

DESCRIPTION

40	Arguments		
41	IN	ctx	The context on which to perform the operation. When this argument is
42			not provided, the operation is performed on SHMEM_CTX_DEFAULT.
43	OUT	dest	Array to be updated on the remote PE. This data object must be re-
44			motely accessible.
45	IN	source	Array containing the data to be copied.
46	IN	dst	The stride between consecutive elements of the dest array. The stride
47			is scaled by the element size of the <i>dest</i> array. A value of 1 indicates
48			contiguous data. <i>dst</i> must be of type <i>ptrdiff_t</i> . When using <i>Fortran</i> , it
			must be a default integer value.

IN	sst	The stride between consecutive elements of the <i>source</i> array. The stride is scaled by the element size of the <i>source</i> array. A value of <i>1</i> indicates contiguous data. <i>sst</i> must be of type <i>ptrdiff_t</i> . When using <i>Fortran</i> , it must be a default integer value.
IN	nelems	Number of elements in the <i>dest</i> and <i>source</i> arrays. <i>nelems</i> must be of type <i>size_t</i> for <i>C</i> . When using <i>Fortran</i> , it must be a constant, variable, or array element of default integer type.
IN	pe	PE number of the remote PE. <i>pe</i> must be of type integer. When using <i>Fortran</i> , it must be a constant, variable, or array element of default integer type.

API description

The *iput* routines provide a method for copying strided data elements (specified by *sst*) of an array from a *source* array on the local PE to locations specified by stride *dst* on a *dest* array on specified remote PE. Both strides, *dst* and *sst*, must be greater than or equal to *1*. The routines return when the data has been copied out of the *source* array on the local PE but not necessarily before the data has been delivered to the remote data object.

The dest and source data objects must conform to typing constraints, which are as follows:

Routine	Data type of <i>dest</i> and <i>source</i>
shmem_iput4, shmem_iput32	Any noncharacter type that has a storage size equal to 32 bits.
shmem_iput8	C: Any noncharacter type that has a storage size equal to 8 bits.
	<i>Fortran</i> : Any noncharacter type that has a storage size equal to 64 bits.
shmem_iput64	Any noncharacter type that has a storage size equal to 64 bits.
shmem_iput128	Any noncharacter type that has a storage size equal to 128 bits.
SHMEM_COMPLEX_IPUT	Elements of type complex of default size.
SHMEM_DOUBLE_IPUT	Elements of type double precision.
SHMEM_INTEGER_IPUT	Elements of type integer.
SHMEM_LOGICAL_IPUT	Elements of type logical.
SHMEM_REAL_IPUT	Elements of type real.
ırn Vəluos	
None	
None.	

Notes

When using *Fortran*, data types must be of default size. For example, a real variable must be declared as *REAL*, *REAL**4 or *REAL*(*KIND*=*KIND*(1.0)). See Section 3 for a definition of the term remotely accessible.

EXAMPLES

Consider the following *shmem_iput* example for *C11* programs.

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

int main(void)

```
short source[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
2
              static short dest[10];
              shmem_init();
              int me = shmem_my_pe();
              if (me == 0) /* put 5 elements into dest on PE 1 */
                 shmem_iput(dest, source, 1, 2, 5, 1);
              shmem_barrier_all(); /* sync sender and receiver */
              if (me == 1) {
                 printf("dest on PE %d is %hd %hd %hd %hd %hd\n", me,
                     dest[0], dest[1], dest[2], dest[3], dest[4]);
9
              shmem_finalize();
10
              return 0;
11
12
13
     9.6.4 SHMEM_GET
14
     Copies data from a specified PE.
15
16
      SYNOPSIS
17
18
           C11:
19
           void shmem_get(TYPE *dest, const TYPE *source, size_t nelems, int pe);
           void shmem_get(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, size_t nelems, int pe);
20
           where TYPE is one of the standard RMA types specified by Table 3.
21
22
           C/C++:
23
           void shmem_<TYPENAME>_get(TYPE *dest, const TYPE *source, size_t nelems, int pe);
           void shmem_ctx_<TYPENAME>_get(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, size_t
24
               nelems, int pe);
25
           where TYPE is one of the standard RMA types and has a corresponding TYPENAME specified by Table 3.
26
           void shmem_get<SIZE>(void *dest, const void *source, size_t nelems, int pe);
27
           void shmem_ctx_get<SIZE>(shmem_ctx_t ctx, void *dest, const void *source, size_t nelems,
28
                int pe);
29
           where SIZE is one of 8, 16, 32, 64, 128.
30
           void shmem_getmem(void *dest, const void *source, size_t nelems, int pe);
31
           void shmem_ctx_getmem(shmem_ctx_t ctx, void *dest, const void *source, size_t nelems, int
32
               pe);
33

    deprecation start

34
           FORTRAN:
35
           INTEGER nelems, pe
           CALL SHMEM_CHARACTER_GET(dest, source, nelems, pe)
36
           CALL SHMEM_COMPLEX_GET(dest, source, nelems, pe)
37
           CALL SHMEM_DOUBLE_GET(dest, source, nelems, pe)
38
           CALL SHMEM_GET4(dest, source, nelems, pe)
39
           CALL SHMEM_GET8 (dest, source, nelems, pe)
40
           CALL SHMEM_GET32(dest, source, nelems, pe)
41
           CALL SHMEM_GET64(dest, source, nelems, pe)
42
           CALL SHMEM_GET128(dest, source, nelems, pe)
43
           CALL SHMEM_GETMEM(dest, source, nelems, pe)
44
           CALL SHMEM_INTEGER_GET(dest, source, nelems, pe)
45
           CALL SHMEM_LOGICAL_GET(dest, source, nelems, pe)
46
           CALL SHMEM_REAL_GET(dest, source, nelems, pe)
47
                                                                                             deprecation end -
48
```

DESCRIPTION

Arguments		
IN	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
OUT	dest	Local data object to be updated.
IN	source	Data object on the PE identified by <i>pe</i> that contains the data to be copied. This data object must be remotely accessible.
IN	nelems	Number of elements in the <i>dest</i> and <i>source</i> arrays. <i>nelems</i> must be of type <i>size_t</i> for <i>C</i> . When using <i>Fortran</i> , it must be a constant, variable, or array element of default integer type.
IN	pe	PE number of the remote PE. <i>pe</i> must be of type integer. When using <i>Fortran</i> , it must be a constant, variable, or array element of default integer type.

API description

The get routines provide a method for copying a contiguous symmetric data object from a different PE to a contiguous data object on the local PE. The routines return after the data has been delivered to the *dest* array on the local PE.

The dest and source data objects must conform to typing constraints, which are as follows:

Routine	Data type of <i>dest</i> and <i>source</i>	
shmem_getmem	Fortran: Any noncharacter type. C: Any data type. nelems is scaled in bytes.	
shmem_get4, shmem_get32 shmem_get8	Any noncharacter type that has a storage size equal to 32 bits. <i>C</i> : Any noncharacter type that has a storage size equal to 8 bits. <i>Fortran</i> : Any noncharacter type that has a storage size equal to 64 bits.	
shmem_get64 shmem_get128 SHMEM_CHARACTER_GET	Any noncharacter type that has a storage size equal to 64 bits. Any noncharacter type that has a storage size equal to 128 bits. Elements of type character. <i>nelems</i> is the number of characters to transfer. The actual character lengths of the <i>source</i> and <i>dest</i> variables are ignored	
SHMEM_COMPLEX_GET SHMEM_DOUBLE_GET SHMEM_INTEGER_GET SHMEM_LOGICAL_GET SHMEM_REAL_GET	Elements of type complex of default size. <i>Fortran</i> : Elements of type double precision. Elements of type integer. Elements of type logical. Elements of type real.	
rn Values None		

Notes

See Section 3 for a definition of the term remotely accessible. When using *Fortran*, data types must be of default size. For example, a real variable must be declared as *REAL*, *REAL**4, or *REAL*(*KIND=KIND*(1.0)).

```
EXAMPLES
1
2
3
            Consider this example for Fortran.
4
            PROGRAM REDUCTION
            INCLUDE "shmem.fh"
5
6
            REAL VALUES, SUM
            COMMON /C/ VALUES
            REAL WORK
            CALL SHMEM_INIT()
                                              ! ALLOW ANY NUMBER OF PES
9
            VALUES = SHMEM_MY_PE()
                                                    ! INITIALIZE IT TO SOMETHING
            CALL SHMEM_BARRIER_ALL
10
            SUM = 0.0
11
            DO I = 0, SHMEM_N_PES() -1
               CALL SHMEM_REAL_GET(WORK, VALUES, (SHMEM_N_PES()()-1), I)
12
               SUM = SUM + WORK
13
            ENDDO
14
            PRINT*, 'PE ', SHMEM_MY_PE(), ' COMPUTED SUM=', SUM
            CALL SHMEM_BARRIER_ALL
15
            END
16
17
18
      9.6.5 SHMEM G
19
      Copies one data item from a remote PE
20
21
      SYNOPSIS
22
23
            C11:
24
            TYPE shmem_g(const TYPE *source, int pe);
25
            TYPE shmem_g(shmem_ctx_t ctx, const TYPE *source, int pe);
26
            where TYPE is one of the standard RMA types specified by Table 3.
27
            C/C++:
28
            TYPE shmem_<TYPENAME>_g(const TYPE *source, int pe);
29
            TYPE shmem_ctx_<TYPENAME>_g(shmem_ctx_t ctx, const TYPE *source, int pe);
30
            where TYPE is one of the standard RMA types and has a corresponding TYPENAME specified by Table 3.
31
32
33
      DESCRIPTION
34
35
            Arguments
                                                   The context on which to perform the operation. When this argument is
36
                  IN
                                   ctx
                                                   not provided, the operation is performed on SHMEM_CTX_DEFAULT.
37
                  IN
                                                   The remotely accessible array element or scalar data object.
                                   source
38
                  IN
                                                   The number of the remote PE on which source resides.
                                   pe
39
40
41
            API description
42
43
                 These routines provide a very low latency get capability for single elements of most basic types.
44
45
46
            Return Values
47
                 Returns a single element of type specified in the synopsis.
48
```

Notes

None.

EXAMPLES

The following *shmem_g* example is for *C11* programs:

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

int main(void)

```
long y = -1;
static long x = 10101;
shmem_init();
int me = shmem_my_pe();
int npes = shmem_n_pes();
if (me == 0)
    y = shmem_g(&x, npes-1);
printf("%d: y = %ld\n", me, y);
shmem_finalize();
return 0;
```

9.6.6 SHMEM_IGET

Copies strided data from a specified PE.

SYNOPSIS

C11:

```
void shmem_iget(TYPE *dest, const TYPE *source, ptrdiff_t dst, ptrdiff_t sst, size_t nelems,
    int pe);
void shmem_iget(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, ptrdiff_t dst, ptrdiff_t
    sst, size_t nelems, int pe);
where TYPE is one of the standard RMA types specified by Table 3.
```

C/C++:

```
void shmem_<TYPENAME>_iget(TYPE *dest, const TYPE *source, ptrdiff_t dst, ptrdiff_t sst,
    size_t nelems, int pe);
void shmem_ctx_<TYPENAME>_iget(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, ptrdiff_t
    dst, ptrdiff_t sst, size_t nelems, int pe);
where TYPE is one of the standard RMA types and has a corresponding TYPENAME specified by Table 3.
```

```
void shmem_iget<SIZE>(void *dest, const void *source, ptrdiff_t dst, ptrdiff_t sst, size_t
    nelems, int pe);
void shmem_ctx_iget<SIZE>(shmem_ctx_t ctx, void *dest, const void *source, ptrdiff_t dst,
    ptrdiff_t sst, size_t nelems, int pe);
```

where SIZE is one of 8, 16, 32, 64, 128.

- deprecation start -

FORTRAN:

```
INTEGER dst, sst, nelems, pe
CALL SHMEM_COMPLEX_IGET(dest, source, dst, sst, nelems, pe)
CALL SHMEM_DOUBLE_IGET(dest, source, dst, sst, nelems, pe)
CALL SHMEM_IGET4(dest, source, dst, sst, nelems, pe)
CALL SHMEM_IGET8(dest, source, dst, sst, nelems, pe)
CALL SHMEM_IGET32(dest, source, dst, sst, nelems, pe)
```

10

12

13

14

15

16

17

18 19 20

21 22

23 24

25

26

27

28

29

30

31

32

33

34

35

37

38

39

40

41

42

43

44

45

46

47

CALL	SHMEM IGET64 (dest source det est pelems pe)
CALL	Simila_Iobio(dest, Source, dst, Sst, nerens, pc)
CALL	SHMEM_IGET128(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_INTEGER_IGET(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_LOGICAL_IGET(dest, source, dst, sst, nelems, pe)
CALL	SHMEM_REAL_IGET(dest, source, dst, sst, nelems, pe)
	deprecation and -

DESCRIPTION

10	Arguments		
11	IN	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
12	OUT	dest	Array to be updated on the local PE.
13	IN	source	Array containing the data to be copied on the remote PE.
14	IN	dst	The stride between consecutive elements of the <i>dest</i> array. The stride
15			is scaled by the element size of the <i>dest</i> array. A value of 1 indicates
16			contiguous data. <i>dst</i> must be of type <i>ptrdiff</i> t. When using <i>Fortran</i> , it
17			must be a default integer value.
18	IN	sst	The stride between consecutive elements of the <i>source</i> array. The stride
19			is scaled by the element size of the <i>source</i> array. A value of <i>1</i> indicates
20			contiguous data. <i>sst</i> must be of type <i>ptrdiff_t</i> . When using <i>Fortran</i> , it
21			must be a default integer value.
21	IN	nelems	Number of elements in the dest and source arrays. nelems must be of
22			type <i>size_t</i> for <i>C</i> . When using <i>Fortran</i> , it must be a constant, variable,
23			or array element of default integer type.
24	IN	ре	PE number of the remote PE. pe must be of type integer. When us-
25		*	ing Fortran, it must be a constant, variable, or array element of default
26			integer type.

API description

The iget routines provide a method for copying strided data elements from a symmetric array from a specified remote PE to strided locations on a local array. The routines return when the data has been copied into the local dest array.

The *dest* and *source* data objects must conform to typing constraints, which are as follows:

Routine	Data type of <i>dest</i> and <i>source</i>
shmem_iget4, shmem_iget32 shmem_iget8	Any noncharacter type that has a storage size equal to 32 bits. <i>C</i> : Any noncharacter type that has a storage size equal to 8 bits. <i>Fortran</i> : Any noncharacter type that has a storage size equal to 64 bits.
shmem_iget64 shmem_iget128 SHMEM_COMPLEX_IGET SHMEM_DOUBLE_IGET SHMEM_INTEGER_IGET SHMEM_LOGICAL_IGET SHMEM_REAL_IGET	 Any noncharacter type that has a storage size equal to 64 bits. Any noncharacter type that has a storage size equal to 128 bits. Elements of type complex of default size. <i>Fortran</i>: Elements of type double precision. Elements of type integer. Elements of type logical. Elements of type real.

Return Values

None.

Notes

When using *Fortran*, data types must be of default size. For example, a real variable must be declared as REAL, REAL*4, or REAL(KIND=KIND(1.0)).

EXAMPLES

The following example uses *shmem_logical_iget* in a *Fortran* program.

```
PROGRAM STRIDELOGICAL
INCLUDE "shmem.fh"
```

```
LOGICAL SOURCE(10), DEST(5)
SAVE SOURCE ! SAVE MAKES IT REMOTELY ACCESSIBLE
DATA SOURCE /.T.,.F.,.T.,.F.,.T.,.F.,.T.,.F.,.T.,.F./
DATA DEST / 5*.F. /
CALL SHMEM_INIT()
IF (SHMEM_MY_PE() .EQ. 0) THEN
   CALL SHMEM_LOGICAL_IGET(DEST, SOURCE, 1, 2, 5, 1)
  PRINT*,'DEST AFTER SHMEM_LOGICAL_IGET:',DEST
ENDIF
CALL SHMEM_BARRIER_ALL
```

9.7 Non-blocking Remote Memory Access Routines

9.7.1 SHMEM_PUT_NBI

The nonblocking put routines provide a method for copying data from a contiguous local data object to a data object on a specified PE.

SYNOPSIS

```
C11:
                                                                                                       31
void shmem_put_nbi(TYPE *dest, const TYPE *source, size_t nelems, int pe);
void shmem_put_nbi(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, size_t nelems, int pe);
                                                                                                       32
                                                                                                        33
where TYPE is one of the standard RMA types specified by Table 3.
                                                                                                       34
C/C++:
                                                                                                       35
void shmem_<TYPENAME>_put_nbi(TYPE *dest, const TYPE *source, size_t nelems, int pe);
                                                                                                        36
void shmem_ctx_<TYPENAME>_put_nbi(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, size_t
    nelems, int pe);
                                                                                                        37
                                                                                                        38
where TYPE is one of the standard RMA types and has a corresponding TYPENAME specified by Table 3.
void shmem_put<SIZE>_nbi(void *dest, const void *source, size_t nelems, int pe);
                                                                                                        39
void shmem_ctx_put<SIZE>_nbi(shmem_ctx_t ctx, void *dest, const void *source, size_t nelems,
                                                                                                        40
    int pe);
                                                                                                       41
where SIZE is one of 8, 16, 32, 64, 128.
                                                                                                       42
void shmem_putmem_nbi(void *dest, const void *source, size_t nelems, int pe);
                                                                                                       43
void shmem_ctx_putmem_nbi(shmem_ctx_t ctx, void *dest, const void *source, size_t nelems,
                                                                                                       44
    int pe);
                                                                                                        45
- deprecation start -
FORTRAN:
                                                                                                       47
CALL SHMEM_CHARACTER_PUT_NBI(dest, source, nelems, pe)
                                                                                                       48
CALL SHMEM_COMPLEX_PUT_NBI(dest, source, nelems, pe)
```

10

11

12

13

14

15

16

17

18

19

20 21 22

23 24

25

26

27 28

> 29 30

1	CALL	SHMEM_DOUBLE_PUT_NBI(dest, source, nelems, pe)
2	CALL	SHMEM_INTEGER_PUT_NBI(dest, source, nelems, pe)
3	CALL	SHMEM_LOGICAL_PUT_NBI(dest, source, nelems, pe)
4	CALL	SHMEM_PUT4_NBI(dest, source, nelems, pe)
5	CALL	SHMEM_PUT8_NBI(dest, source, nelems, pe)
6	CALL	SHMEM_PUT32_NBI(dest, source, nelems, pe)
7	CALL	SHMEM_PUT64_NBI(dest, source, nelems, pe)
•	CALL	SHMEM_PUT128_NBI(dest, source, nelems, pe)
	CALL	SHMEM_PUTMEM_NBI(dest, source, nelems, pe)
9	CALL	SHMEM REAL PUT NBI(dest, source, nelems, pe)

deprecation end -

DESCRIPTION

Arguments		
IN	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
OUT	dest	Data object to be updated on the remote PE. This data object must be remotely accessible.
IN	source	Data object containing the data to be copied.
IN	nelems	Number of elements in the <i>dest</i> and <i>source</i> arrays. <i>nelems</i> must be of type <i>size_t</i> for <i>C</i> . When using <i>Fortran</i> , it must be a constant, variable, or array element of default integer type.
IN	pe	PE number of the remote PE. <i>pe</i> must be of type integer. When using <i>Fortran</i> , it must be a constant, variable, or array element of default integer type.

API description

The routines return after posting the operation. The operation is considered complete after a subsequent call to shmem_quiet. At the completion of shmem_quiet, the data has been copied into the dest array on the destination PE. The delivery of data words into the data object on the destination PE may occur in any order. Furthermore, two successive put routines may deliver data out of order unless a call to shmem_fence is introduced between the two calls.

The dest and source data objects must conform to certain typing constraints, which are as follows:

Routine	Data type of <i>dest</i> and <i>source</i>
shmem_putmem_nbi	<i>Fortran</i> : Any noncharacter type. <i>C</i> : Any data type. nelems is scaled in bytes.
shmem_put4_nbi, shmem_put32_nbi	Any noncharacter type that has a storage size equal to 32 bits.
shmem_put8_nbi	<i>C</i> : Any noncharacter type that has a storage size equal to 8 bits. <i>Fortran</i> : Any noncharacter type that has a storage size equal to 64 bits.
shmem_put64_nbi shmem_put128_nbi SHMEM_CHARACTER_PUT_NBI	Any noncharacter type that has a storage size equal to 64 bits. Any noncharacter type that has a storage size equal to 128 bits. Elements of type character. <i>nelems</i> is the number of characters to transfer. The actual character lengths of the <i>source</i> and <i>dest</i> variables are ignored.

ç

SHMEM_COMPLEX_PUT_NBI	Elements of type complex of default size.
SHMEM_DOUBLE_PUT_NBI	Elements of type double precision.
SHMEM_INTEGER_PUT_NBI	Elements of type integer.
SHMEM_LOGICAL_PUT_NBI	Elements of type logical.
SHMEM_REAL_PUT_NBI	Elements of type real.
Return Values	

None.

Notes

None.

9.7.2 SHMEM_GET_NBI

The nonblocking get routines provide a method for copying data from a contiguous remote data object on the specified PE to the local data object.

SYNOPSIS

C11:

void shmem_get_nbi(TYPE *dest, const TYPE *source, size_t nelems, int pe); void shmem_get_nbi(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, size_t nelems, int pe);

where TYPE is one of the standard RMA types specified by Table 3.

C/C++:

```
void shmem_<TYPENAME>_get_nbi(TYPE *dest, const TYPE *source, size_t nelems, int pe);
void shmem_ctx_<TYPENAME>_get_nbi(shmem_ctx_t ctx, TYPE *dest, const TYPE *source, size_t
    nelems, int pe);
where TYPE is one of the standard RMA types and has a corresponding TYPENAME specified by Table 3.
void shmem_get<SIZE>_nbi(void *dest, const void *source, size_t nelems, int pe);
void shmem_ctx_get<SIZE>_nbi(shmem_ctx_t ctx, void *dest, const void *source, size_t
    nelems, int pe);
where SIZE is one of 8, 16, 32, 64, 128.
void shmem_getmem_nbi(void *dest, const void *source, size_t nelems, int pe);
void shmem_ctx_getmem_nbi(shmem_ctx_t ctx, void *dest, const void *source, size_t nelems,
    int pe);
- deprecation start
FORTRAN:
INTEGER nelems, pe
CALL SHMEM_CHARACTER_GET_NBI(dest, source, nelems, pe)
CALL SHMEM_COMPLEX_GET_NBI(dest, source, nelems, pe)
CALL SHMEM_DOUBLE_GET_NBI(dest, source, nelems, pe)
CALL SHMEM_GET4_NBI(dest, source, nelems, pe)
CALL SHMEM_GET8_NBI(dest, source, nelems, pe)
CALL SHMEM_GET32_NBI(dest, source, nelems, pe)
CALL SHMEM_GET64_NBI(dest, source, nelems, pe)
CALL SHMEM_GET128_NBI(dest, source, nelems, pe)
CALL SHMEM_GETMEM_NBI(dest, source, nelems, pe)
CALL SHMEM_INTEGER_GET_NBI(dest, source, nelems, pe)
CALL SHMEM_LOGICAL_GET_NBI(dest, source, nelems, pe)
CALL SHMEM_REAL_GET_NBI(dest, source, nelems, pe)
```

10

11

12 13 14

15 16

17 18

19 20

21

22 23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

47

- deprecation end —

DESCRIPTION

Arguments

Ι	N	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
(DUT	dest	Local data object to be updated.
Ι	N	source	Data object on the PE identified by <i>pe</i> that contains the data to be copied. This data object must be remotely accessible.
Ι	N	nelems	Number of elements in the <i>dest</i> and <i>source</i> arrays. <i>nelems</i> must be of type <i>size_t</i> for <i>C</i> . When using <i>Fortran</i> , it must be a constant, variable, or array element of default integer type.
Ι	N	pe	PE number of the remote PE. <i>pe</i> must be of type integer. When using <i>Fortran</i> , it must be a constant, variable, or array element of default integer type.

API description

The get routines provide a method for copying a contiguous symmetric data object from a different PE to a contiguous data object on the local PE. The routines return after posting the operation. The operation is considered complete after a subsequent call to *shmem_quiet*. At the completion of *shmem_quiet*, the data has been delivered to the *dest* array on the local PE.

The dest and source data objects must conform to typing constraints, which are as follows:

Routine	Data type of <i>dest</i> and <i>source</i>
shmem_getmem_nbi	Fortran: Any noncharacter type. C: Any data type. nelems is scaled in bytes.
shmem_get4_nbi,	Any noncharacter type that has a storage size equal to 32 bits.
shmem_get32_nbi	
shmem_get8_nbi	C: Any noncharacter type that has a storage size equal to 8 bits.
	<i>Fortran</i> : Any noncharacter type that has a storage size equal to 64 bits.
shmem_get64_nbi	Any noncharacter type that has a storage size equal to 64 bits.
shmem_get128_nbi	Any noncharacter type that has a storage size equal to 128 bits.
SHMEM_CHARACTER_GET_NB	I Elements of type character. <i>nelems</i> is the number of characters
	to transfer. The actual character lengths of the <i>source</i> and <i>dest</i> variables are ignored.
SHMEM_COMPLEX_GET_NBI	Elements of type complex of default size.
SHMEM_DOUBLE_GET_NBI	Fortran: Elements of type double precision.
SHMEM_INTEGER_GET_NBI	Elements of type integer.
SHMEM_LOGICAL_GET_NBI	Elements of type logical.
SHMEM_REAL_GET_NBI	Elements of type real.

Return Values None.

Notes

See Section 3 for a definition of the term remotely accessible. When using *Fortran*, data types must be of default size. For example, a real variable must be declared as *REAL*, *REAL**4, or *REAL*(*KIND*=*KIND*(1.0)).

9.8 Atomic Memory Operations

An *Atomic Memory Operation* (AMO) is a one-sided communication mechanism that combines memory read, update, or write operations with atomicity guarantees described in Section 3.1. Similar to the RMA routines, described in Section 9.6, the AMOs are performed only on symmetric objects. OpenSHMEM defines two types of AMO routines:

• The *fetching* routines return the original value of, and optionally update, the remote data object in a single atomic operation. The routines return after the data has been fetched from the target PE and delivered to the calling PE. The data type of the returned value is the same as the type of the remote data object.

The fetching routines include: *shmem_atomic_{fetch, compare_swap, swap}* and *shmem_atomic_fetch_{inc, add, and, or, xor}*.

• The *non-fetching* routines update the remote data object in a single atomic operation. A call to a non-fetching atomic routine issues the atomic operation and may return before the operation executes on the target PE. The *shmem_quiet, shmem_barrier*, or *shmem_barrier_all* routines can be used to force completion for these non-fetching atomic routines.

The non-fetching routines include: *shmem_atomic_{set, inc, add, and, or, xor}*.

Where appropriate compiler support is available, OpenSHMEM provides type-generic AMO interfaces via *C11* generic selection. The type-generic support for the AMO routines is as follows:

- *shmem_atomic_{compare_swap, fetch_inc, inc, fetch_add, add}* support the "standard AMO types" listed in Table 4,
- *shmem_atomic_[fetch, set, swap]* support the "extended AMO types" listed in Table 5, and
- *shmem_atomic_{fetch_and, and, fetch_or, or, fetch_xor, xor}* support the "bitwise AMO types" listed in Table 6.

The standard, extended, and bitwise AMO types include some of the exact-width integer types defined in *stdint.h* by *C99* §7.18.1.1 and *C11* §7.20.1.1. When the *C* translation environment does not provide exact-width integer types with *stdint.h*, an OpenSHMEM implementation is not required to provide support for these types.

9.8.1 SHMEM_ATOMIC_FETCH

Atomically fetches the value of a remote data object.

SYNOPSIS

C11:

TYPE shmem_atomic_fetch(const TYPE *source, int pe);

TYPE shmem_atomic_fetch(shmem_ctx_t ctx, const TYPE *source, int pe);

where *TYPE* is one of the extended AMO types specified by Table 5.

C/C++:

TYPE shmem_<TYPENAME>_atomic_fetch(const TYPE *source, int pe);

TYPE shmem_ctx_<TYPENAME>_atomic_fetch(shmem_ctx_t ctx, const TYPE *source, int pe);

where *TYPE* is one of the extended AMO types and has a corresponding *TYPENAME* specified by Table 5.

- deprecation start

C11:

— deprecation end —

ТҮРЕ	TYPENAME
int	int
long	long
long long	longlong
unsigned int	uint
unsigned long	ulong
unsigned long long	ulonglong
int32_t	int32
int64_t	int64
uint32_t	uint32
uint64_t	uint64
size_t	size
ptrdiff_t	ptrdiff

Table 4: Standard AMO Types and Names

TYPENAME
float
double
int
long
longlong
uint
ulong
ulonglong
int32
int64
uint32
uint64
size
ptrdiff

Table 5: Extended AMO Types and Names

where *TYPE* is one of {*float*, *double*, *int*, *long*, *long* long}.

C/C++:

TYPE shmem_<TYPENAME>_fetch(const TYPE *source, int pe);

where *TYPE* is one of {*float*, *double*, *int*, *long*, *long long*} and has a corresponding *TYPENAME* specified by Table 5.

- deprecation start -

FORTRAN: integer pe

 43
 INTEGER pe

 44
 INTEGER*4 SHMEM_INT4_FETCH, ires_i4

45 ires_i4 = SHMEM_INT4_FETCH(source, pe)

46 INTEGER*8 SHMEM_INT8_FETCH, ires_i8

ires_i8 = SHMEM_INT8_FETCH(source, pe)

REAL*4 SHMEM_REAL4_FETCH, res_r4

48 res_r4 = SHMEM_REAL4_FETCH(source, pe)

TYPE shmem_fetch(const TYPE *source, int pe);

TYPENAME
uint
ulong
ulonglong
int32
int64
uint32
uint64

Table 6: Bitwise AMO Types and Names

REAL*8 SHMEM_REAL8_FETCH, res_r8 res_r8 = SHMEM_REAL8_FETCH(source, pe)

DESCRIPTION

Arguments

IN	ctx	The context on which to perform the operation. When this argument is
		not provided, the operation is performed on SHMEM_CTX_DEFAULT.
IN	source	The remotely accessible data object to be fetched from the remote PE.
IN	pe	An integer that indicates the PE number from which source is to be
	^	fetched.

API description

shmem_atomic_fetch performs an atomic fetch operation. It returns the contents of the *source* as an atomic operation.

Return Values

The contents at the *source* address on the remote PE. The data type of the return value is the same as the type of the remote data object.

Notes

None.

9.8.2 SHMEM_ATOMIC_SET

Atomically sets the value of a remote data object.

SYNOPSIS

C11:

```
void shmem_atomic_set(TYPE *dest, TYPE value, int pe);
void shmem_atomic_set(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);
```

deprecation end -

	where <i>TTPE</i> is one of the extended AMO types specified by Table 5.					
	C/C++:					
	<pre>void shmem_<typename>_atomic_set(TYPE *dest, TYPE value, int pe);</typename></pre>					
	<pre>void shmem_ctx_<typename>_atomic_set(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);</typename></pre>					
	where <i>TYPE</i> is one of the extended AMO types and has a corresponding <i>TYPENAME</i> specified by Table 5.					
	deprecation start					
	C11:					
	<pre>void shmem_set(TYPE *dest, TYPE value, int pe);</pre>					
	where <i>TYPE</i> is one of { <i>float</i> , <i>double</i> , <i>int</i> , <i>long</i> , <i>long long</i> }.					
	C/C++:					
	<pre>void shmem_<typename>_set(TYPE *dest, TYPE value, int pe);</typename></pre>					
	where <i>TYPE</i> is one of { <i>float</i> , <i>double</i> , <i>int</i> , <i>long</i> , <i>long long</i> } and has a corresponding <i>TYPENAME</i> specified by Table 5.					
	deprecation end —					
	- deprecation start					
	FORTRAN:					
	INTEGER pe					
	INTEGER*4 SHMEM_INT4_SET, value_i4					
	CALL SHMEM_INT4_SET(dest, value_14, pe)					
	INTEGER*8 SHMEM_INT8_SET, Value_18					
	CALL SHMEM_INTS_SET(dest, value_18, pe)					
	REAL*4 SHMEM_REAL4_SEI, Value_r4					
	CALL SHMEM_REAL4_SET(dest, value_r4, pe)					
	REAL*8 SHMEM_REAL8_SET, value_r8					
	REAL*8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET(dest, value_r8, pe)					
	REAL*8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET(dest, value_r8, pe) 					
ESC	REAL*8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET(dest, value_r8, pe) deprecation end —					
ESC	REAL*8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET(dest, value_r8, pe) CRIPTION Arguments					
ESC	REAL*8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET (dest, value_r8, pe) deprecation end CRIPTION Arguments IN ctx OUT dest IN value The context on which to perform the operation. When this argument is not provided, the operation is performed on SHMEM_CTX_DEFAULT. The remotely accessible data object to be set on the remote PE. The value to be atomically written to the remote PE.					
ESC	REAL*8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET (dest, value_r8, pe) CRIPTION Arguments IN ctx OUT dest IN value IN value IN value OUT dest IN value IN value IN value IN value IN pe					
esc	REAL*8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET (dest, value_r8, pe) CRIPTION Arguments IN ctx OUT dest IN value IN value IN pe Arguments The context on which to perform the operation. When this argument is not provided, the operation is performed on SHMEM_CTX_DEFAULT. The remotely accessible data object to be set on the remote PE. The value to be atomically written to the remote PE. An integer that indicates the PE number on which dest is to be updated. API description					
ESC	REAL+8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET (dest, value_r8, pe) CRIPTION Arguments IN ctx OUT dest IN value IN value IN value Arguments The context on which to perform the operation. When this argument is not provided, the operation is performed on SHMEM_CTX_DEFAULT. The remotely accessible data object to be set on the remote PE. The value to be atomically written to the remote PE. An integer that indicates the PE number on which dest is to be updated. API description shmem_atomic_set performs an atomic set operation. It writes the value into dest on pe as an atomic operation.					
ESC	REAL+8 SIMEM_REAL8_SET, value_r8 CALL SIMEM_REAL8_SET (dest, value_r8, pe) CRIPTION Arguments IN ctx OUT dest IN value IN pe An integer that indicates the PE number on which dest is to be updated. API description shmem_atomic_set performs an atomic set operation. It writes the value into dest on pe as an atomic operation. Return Values					
DESC	REAL+80 SHMEM_REAL8_SET, value_r8 CRIPTION Arguments IN ctx OUT dest IN value IN pe API description An integer that indicates the PE number on which dest on pe as an atomic operation. Return Values None.					
DESC	REAL+8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET (dest, value_r8, pe) CRIPTION Arguments IN etx OUT dest IN value IN pe An integer that indicates the PE number on which dest is to be updated. API description shmem_atomic_set performs an atomic set operation. It writes the value into dest on pe as an atomic operation. Return Values None.					
ESC	REAL+8 SHMEM_REAL8_SET, value_r8 CALL SHMEM_REAL8_SET (dest, value_r8, pe) deprecation end CRIPTION Arguments IN etx OUT dest IN value IN value IN pe Arguments The context on which to perform the operation. When this argument is not provided, the operation is performed on SHMEM_CTX_DEFAULT. The remotely accessible data object to be set on the remote PE. The value to be atomically written to the remote PE. An integer that indicates the PE number on which dest is to be updated. API description shmem_atomic_set performs an atomic set operation. It writes the value into dest on pe as an atomic operation. Return Values None. Notes Nore.					

9.8.3 SHMEM_ATOMIC_COMPARE_SWAP

Performs an atomic conditional swap on a remote data object.

SYNOPSIS

C11:

TYPE shmem_atomic_compare_swap(TYPE *dest, TYPE cond, TYPE value, int pe); TYPE shmem_atomic_compare_swap(shmem_ctx_t ctx, TYPE *dest, TYPE cond, TYPE value, int pe);

where TYPE is one of the standard AMO types specified by Table 4.

C/C++:

```
TYPE shmem_<TYPENAME>_atomic_compare_swap(TYPE *dest, TYPE cond, TYPE value, int pe);
TYPE shmem_ctx_<TYPENAME>_atomic_compare_swap(shmem_ctx_t ctx, TYPE *dest, TYPE cond, TYPE
value, int pe);
```

where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4.

- deprecation start -

C11:

TYPE shmem_cswap(**TYPE** *dest, **TYPE** cond, **TYPE** value, **int** pe);

where *TYPE* is one of {*int*, *long*, *long long*}.

C/C++:

TYPE shmem_<TYPENAME>_cswap(TYPE *dest, TYPE cond, TYPE value, int pe);

where TYPE is one of {*int*, *long*, *long long*} and has a corresponding TYPENAME specified by Table 4.

		deprecation end –
	$\overline{\mathbf{V}}$	
INTEGER pe		
INTEGER * 4 SHMEM_INT4_CSWAP, cond_i4, value	e_i4, ires_i4	
<pre>ires_i4 = SHMEM_INT4_CSWAP(dest, cond_i4,</pre>	value_i4, pe)	
<pre>INTEGER*8 SHMEM_INT8_CSWAP, cond_i8, value</pre>	e_i8, ires_i8	
<pre>ires_i8 = SHMEM_INT8_CSWAP(dest, cond_i8,</pre>	value_i8, pe)	
		deprecation end –

DESCRIPTION

Arguments		
IN	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
OUT	dest	The remotely accessible integer data object to be updated on the remote PE.
IN	cond	<i>cond</i> is compared to the remote <i>dest</i> value. If <i>cond</i> and the remote <i>dest</i> are equal, then <i>value</i> is swapped into the remote <i>dest</i> ; otherwise, the remote <i>dest</i> is unchanged. In either case, the old value of the remote <i>dest</i> is returned as the routine return value. <i>cond</i> must be of the same data type as <i>dest</i> .
IN	value	The value to be atomically written to the remote PE. <i>value</i> must be the same data type as <i>dest</i> .
IN	pe	An integer that indicates the PE number upon which <i>dest</i> is to be up- dated. When using <i>Fortran</i> , it must be a default integer value.

1	AI	PI description				
2						
3		The conditional swap routines conditionally update a <i>dest</i> data object on the specified PE and return the prior contents of the data object in one atomic operation				
4		prior contents of the data object in one atomic operation.				
6		When using Fortran, a	When using Fortran, dest, cond, and value must be of the following type:			
7		Routine		Data type of <i>dest</i> , <i>co</i>	nd, and value	
8					,	
9		CUMENTA INTA CO	WAD	1 harda interne		
10		SHMEM_INT4_CS	WAP WAP	<i>4</i> -byte integer.		
11		SINVIENT_INTO_CS	W/ 11	o byte integer.		
12						
13	Da	Anna Malana				
14	ке	The contents that had i	been in the <i>dest</i> of	lata object on the remo	te PE prior to the condit	ional swan. Data type
15		is the same as the <i>dest</i>	data type.	iata object on the remo		ionai swap. Data type
10			51			
17	No	ites				
19	110	None.				•
20						
21						
22	EXAMP	PLES				
23						
24	Th	e following call ensures t	hat the first PE to	o execute the condition	al swap will successfull	y write its PE number
25	to	<i>race_winner</i> on PE 0.				
26	#i:	nclude <stdio.h></stdio.h>			*	
27	#i1	nclude <shmem.h></shmem.h>				
28	in	t main(void)				
29	{	static int race winn	r = -1			
30		<pre>shmem_init();</pre>	ci - i,			
31		<pre>int me = shmem_my_pe int oldwal = shmem_a</pre>	();	august (frage winner	1	
32		if (oldval == -1) pr	intf("PE %d wa	<pre>is first\n", me);</pre>	-1, me, 0),	
33		<pre>shmem_finalize();</pre>				
34	}	return 0;				
35						
36						
37	9.8.4 S	HMEM_ATOMIC_SW	AP			
38	Performs	an atomic swan to a rem	ote data object			
39	1 errorins	in atomic swap to a ten	ole data object.			
40 41	SYNOP	SIS				
42	C1	1:				
43	TY	PE shmem_atomic_swap(TYPE *dest, TY	TPE value, int pe);		
44	TY	PE shmem_atomic_swap(shmem_ctx_t ct	x, TYPE *dest, TYPE	E value, int pe);	
45 46	wh	here <i>TYPE</i> is one of the end	xtended AMO ty	pes specified by Table	5.	
47	C/	C++:				
48	TY	PE shmem_ <typename>_a</typename>	tomic_swap(TYP	E *dest, TYPE value	e, int pe);	
	TY	PE shmem_ctx_< TYPENAM	E >_atomic_swap	(shmem_ctx_t ctx, '	TYPE *dest, TYPE val	ue, int pe);
where TYPE is one of the extended AMO types and has a corresponding TYPENAME specified by Table 5.

 de	nre	ecat	tior		tart
 uc	μι	Jua	uoi	1.5	ιaιι

C11:

TYPE shmem_swap(**TYPE** *dest, **TYPE** value, **int** pe);

where *TYPE* is one of {*float*, *double*, *int*, *long*, *long* long}.

C/C++:

TYPE shmem_<TYPENAME>_swap(TYPE *dest, TYPE value, int pe);

where TYPE is one of {float, double, int, long, long long} and has a corresponding TYPENAME specified by Table 5.

	– deprecation end –
deprecation start	
FORTRAN:	
INTEGER SHMEM_SWAP, value, pe	
ires = SHMEM_SWAP(dest, value, pe)	
<pre>INTEGER*4 SHMEM_INT4_SWAP, value_i4, ires_i4</pre>	
ires_i4 = SHMEM_INT4_SWAP(dest, value_i4, pe)	
<pre>INTEGER*8 SHMEM_INT8_SWAP, value_i8, ires_i8</pre>	
ires_i8 = SHMEM_INT8_SWAP(dest, value_i8, pe)	
REAL *4 SHMEM_REAL4_SWAP, value_r4, res_r4	
res_r4 = SHMEM_REAL4_SWAP(dest, value_r4, pe)	
REAL *8 SHMEM_REAL8_SWAP, value_r8, res_r8	
res_r8 = SHMEM_REAL8_SWAP(dest, value_r8, pe)	
	 – deprecation end –

DESCRIPTION



API description

shmem_atomic_swap performs an atomic swap operation. It writes value into dest on PE and returns the previous contents of *dest* as an atomic operation.

When using *Fortran*, *dest* and *value* must be of the following type:

Routine	Data type of <i>dest</i> and <i>value</i>	
		46
SHMEM SWAP	Integer of default kind	47
SHMEM_INT4_SWAP	4-byte integer	48

1	SHMEM_INT8_SWAP 8-byte integer
2	SHMEM_KEAL4_SWAP 4-byte real
3	SITIVIEW_REALO_SWAP 8-byte teat
4	
5	Datum Valuas
6	The content that had been at the <i>dest</i> address on the remote PE prior to the swap is returned.
7	
0	Notes
10	None.
11	
12	
13	EXAMPLES
14	The example below swaps values between odd numbered PEs and their right (modulo) neighbor and outputs the result of swap
16	tinglude setdie by
17	<pre>#include <stufic.h> #include <shmem.h></shmem.h></stufic.h></pre>
18	int main (maid)
19	
20	<pre>static long dest; shown init();</pre>
21	<pre>shmem_fift(); int me = shmem_my_pe();</pre>
22	<pre>int npes = shmem_n_pes(); dost = mo;</pre>
23	<pre>shmem_barrier_all();</pre>
24	<pre>long new_val = me; if (ma < 1) (</pre>
25	<pre>light (me & 1) { long swapped_val = shmem_atomic_swap(&dest, new_val, (me + 1) % npes);</pre>
26	<pre>printf("%d: dest = %ld, swapped = %ld\n", me, dest, swapped_val);</pre>
27	<pre>} shmem_finalize();</pre>
28	return 0;
29	}
30	
31	9.8.5 SHMEM_ATOMIC_FETCH_INC
32	Performs an atomic fatch and increment operation on a remote data object
33	renomis an atomic reten-and-increment operation on a remote data object.
34 35	SYNOPSIS
36	C11:
37	TYPE shmem_atomic_fetch_inc(TYPE *dest, int pe);
38	TYPE shmem_atomic_fetch_inc(shmem_ctx_t ctx, TYPE *dest, int pe);
39	where TYPE is one of the standard AMO types specified by Table 4.
40	C/C++:
41	TYPE shmem_ <typename>_atomic_fetch_inc(TYPE *dest, int pe);</typename>
42	where TVDE is one of the standard AMO types and has a corresponding TVDENAME specified by Table 4
43	where THE is one of the standard Aivio types and has a corresponding TH EIVAME specified by fable 4.
44	- deprecation start
45	
40	TIPE SHMEM_IINC(TIPE *dest, Int pe);
-+7	where <i>IYPE</i> is one of { <i>int</i> , <i>long</i> , <i>long long</i> }.
+0	C/C++:

TYPE shmem_<TYPENAME>_finc(TYPE *dest, int pe);

where *TYPE* is one of {*int*, *long*, *long long*} and has a corresponding *TYPENAME* specified by Table 4.

	deprecation end
- deprecation start	
FORTRAN:	
INTEGER pe	
INTEGER*4 SHMEM_INT4_FINC, ires_i4	
ires_i4 = SHMEM_INT4_FINC(dest, pe)	
INTEGER * 8 SHMEM_INT8_FINC, ires_i8	
ires_i8 = SHMEM_INT8_FINC(dest, pe)	
	deprecation end —

DESCRIPTION

Arguments

IN	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
OUT	dest	The remotely accessible integer data object to be updated on the remote PE. The type of <i>dest</i> should match that implied in the SYNOPSIS sec-
IN	pe	tion. An integer that indicates the PE number on which <i>dest</i> is to be updated. When using <i>Fortran</i> , it must be a default integer value.

API description

These routines perform a fetch-and-increment operation. The *dest* on PE *pe* is increased by one and the routine returns the previous contents of *dest* as an atomic operation.

When using Fortran, dest must be of the following type:

Routine	Data type of <i>dest</i>
SHMEM_INT4_FINC SHMEM_INT8_FINC	4-byte integer 8-byte integer

Return Values

The contents that had been at the *dest* address on the remote PE prior to the increment. The data type of the return value is the same as the *dest*.

Notes

None.

EXAMPLES

The following *shmem_atomic_fetch_inc* example is for *C11* programs:

```
#include <stdio.h>
            #include <shmem.h>
2
            int main(void)
               int old = -1;
               static int dst = 22;
               shmem_init();
               int me = shmem_my_pe();
               if (me == 0)
                  old = shmem_atomic_fetch_inc(&dst, 1);
               shmem_barrier_all();
9
               printf("%d: old = %d, dst = %d\n", me, old, dst);
10
               shmem_finalize();
               return 0;
11
12
13
14
      9.8.6 SHMEM_ATOMIC_INC
15
16
      Performs an atomic increment operation on a remote data object.
17
18
      SYNOPSIS
19
            C11:
20
            void shmem_atomic_inc(TYPE *dest, int pe);
21
            void shmem_atomic_inc(shmem_ctx_t ctx, TYPE *dest, int pe);
22
            where TYPE is one of the standard AMO types specified by Table 4.
23
            C/C++:
24
            void shmem_<TYPENAME>_atomic_inc(TYPE *dest, int pe);
25
            void shmem_ctx_<TYPENAME>_atomic_inc(shmem_ctx_t ctx, TYPE *dest, int pe);
26
            where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4.
27
28
            - deprecation start -
29
            C11:
30
            void shmem_inc(TYPE *dest, int pe);
31
            where TYPE is one of {int, long, long long}.
32
            C/C++:
33
            void shmem_<TYPENAME>_inc(TYPE *dest, int pe);
34
35
            where TYPE is one of {int, long, long long} and has a corresponding TYPENAME specified by Table 4.
36

deprecation end —

37

    deprecation start

38
            FORTRAN:
39
            INTEGER pe
40
            CALL SHMEM_INT4_INC(dest, pe)
41
            CALL SHMEM_INT8_INC (dest, pe)
42
                                                                                                  deprecation end -
43
44
45
      DESCRIPTION
46
47
            Arguments
48
```

IN	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
OUT	dest	The remotely accessible integer data object to be updated on the remote PE. The type of <i>dest</i> should match that implied in the SYNOPSIS section.
IN	pe	An integer that indicates the PE number on which <i>dest</i> is to be updated. When using <i>Fortran</i> , it must be a default integer value.

API description

These routines perform an atomic increment operation on the dest data object on PE.

When using Fortran, dest must be of the following type:

Routine		Data type of <i>dest</i>	15
			10
SHMEM INT	A INC	4-byte integer	18
SHMEM_INT	T8_INC	8-byte integer	19
			20
			21
Return Values			22
None.			23
			24
Notes			25
None.			26
			27
EXAMPLES			28
			29
The following shmem	_atomic_inc example	is for C11 programs:	30
<pre>#include <stdio.h></stdio.h></pre>			31
<pre>#include <shmem.h></shmem.h></pre>			32
<pre>int main(void)</pre>			33
{	· · · · ·		34
<pre>static int dst shmem init();</pre>	= 74;		35
<pre>int me = shmem_</pre>	my_pe();		36
<pre>if (me == 0) shmom atomic</pre>	inc(Edst 1).		37
shmem_barrier_a	ll();		38
printf("%d: dst	= %d\n", me, dst);	;	39
snmem_Iinalize(return 0;);		40
}			41
			42

9.8.7 SHMEM_ATOMIC_FETCH_ADD

Performs an atomic fetch-and-add operation on a remote data object.

SYNOPSIS

C11:

TYPE shmem_atomic_fetch_	add(TYPE *dest, TYPE value, int pe);	
IFE SIMEM_atomic_fetch_	add (shinem_ctx_t ctx, fife *dest, fife value, fit pe);	
where <i>TYPE</i> is one of the standard AMO types specified by Table 4.		
C/C++:	and fatal add myng daat myng aalaa int aala	
TYPE snmem_ <typename>_at</typename>	<pre>omic_retch_add(TYPE *dest, TYPE value, int pe); > atomic fetch add(shmem ctx t ctx TYPE *dest TYPE value int pe);</pre>	
where TVPE is one of the st	anderd AMO types and has a corresponding TVPENAME specified by Table 4	
depression start	indard AMO types and has a corresponding <i>TTPENAME</i> specified by Table 4.	
C11.		
CII: TYPE shmem fadd(TYPE *de	st, TYPE value, int pe);	
where <i>TYPE</i> is one of { <i>int</i> 1	ang lang lang}	
C/C++: TYPE shmem <typename> fa</typename>	dd(TYPE +dest. TYPE value int pe).	
where TVDE is one of (int 1	and long long) and has a corresponding TVDENAME aposition by Table 4	
where <i>TYPE</i> is one of { <i>int</i> , <i>l</i>	<i>Sng</i> , <i>long long</i> } and has a corresponding <i>TTPENAME</i> specified by Table 4.	
	deprecation end —	
- deprecation start		
FORTRAN:		
INTEGER pe		
INTEGER * 4 SHMEM_INT4_FAL	D, ires_i4, value_i4	
ires_i4 = SHMEM_INT4_FA	DD(dest, value_i4, pe)	
INTEGER * 8 SHMEM_INT8_FAL	D, ires_i8, value_i8	
ires_i8 = SHMEM_INT8_FA	DD(dest, value_i8, pe)	
	deprecation end —	
CRIPTION		
Arguments		
IN ctx	The context on which to perform the operation. When this argument is	
	not provided, the operation is performed on SHMEM_CTX_DEFAULT	
OUT dest	The remotely accessible integer data object to be updated on the remote	
	PE. The type of <i>dest</i> should match that implied in the SYNOPSIS sec	
	tion.	
IN valu	<i>e</i> The value to be atomically added to <i>dest</i> . The type of <i>value</i> should	
	match that implied in the SYNOPSIS section.	
IN pe	An integer that indicates the PE number on which <i>dest</i> is to be updated	
	When using <i>Fortran</i> , it must be a default integer value.	
A DL description		
API description		
-less	all mutines as from an etamic fatch and add encention. An etamic fatch and add	
snmem_alomic_jelcn_	<i>dua</i> fournes perform an atomic fetch-and-add operation. An atomic fetch-and-add	
the <i>dest</i> between the til	a dest and adds value to dest without the possibility of another atomic operation on the of the fatch and the undate. These routines add value to dest on ne and raturn the	
pravious contents of de	the of the fetch and the update. These fournes and value to dest on pe and feturit the	
previous contents of <i>ae</i>		
	and and a share much have fight for the fight of the fight	
when using Fortran, d	est and value must be of the following type:	

<pre>SHMEM_INT8_FADD 4-byte integer SHMEM_INT8_FADD 4-byte integer Notes Notes Notes The following shmem_atomic_fetch_add example is for C11 programs: #include catdio.by #include ca</pre>		Routine	Data type of <i>dest</i> and <i>value</i>
<pre>Return Values The contents that had been at the dest address on the remote PE prior to the atomic addition operation. The data type of the return value is the same as the dest. Notes None. KAMPLES The following shmem_atomic_fetch_add example is for C11 programs: #include <stdio().>> #include <stdio().>>> #include <stdio().>> #include <stdio().>>>>>>>>>>>>>>>></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></stdio().></pre>		SHMEM_INT4_FADD SHMEM_INT8_FADD	4-byte integer 8-byte integer
<pre>Notes None. AMPLES The following shmem_atomic_fetch_add example is for C11 programs: finclude <stmem_h> finclude <stmem_h+ finclude finclude <br< td=""><td>Retu</td><td>rn Values The contents that had been at the data type of the return value is</td><td>he <i>dest</i> address on the remote PE prior to the atomic addition operation. The the same as the <i>dest</i>.</td></br<></stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h+ </stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></stmem_h></pre>	Retu	rn Values The contents that had been at the data type of the return value is	he <i>dest</i> address on the remote PE prior to the atomic addition operation. The the same as the <i>dest</i> .
<pre>None. AMPLES The following shmem_atomic_fetch_add example is for C11 programs: #include <ation.b> #i</ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></ation.b></pre>	Note	s	
<pre>SAMPLES The following shmem_atomic_fetch_add example is for C11 programs: #include <stdio.h> #include <stdio.h< td=""> #include <stdio.h> #inc</stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h<></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></pre>		None.	
<pre>The following shmem_atomic_fetch_add example is for C11 programs: finclude <stdio.h> finclude <stdio.h< td=""> finclude finclude <stdio.h< td=""> finclude finclude <stdio.h< td=""> finclude finclude <stdio.h< td=""> finclude finclude <stdio.h< td=""> finclude finclude finclude <stdio.h< td=""> finclude finclude <stdio.h< td=""> finclude finclude finclude <stdio.h< td=""> finclude finclude finclude finclude finclude <stdio.h< td=""> finclude</stdio.h<></stdio.h<></stdio.h<></stdio.h<></stdio.h<></stdio.h<></stdio.h<></stdio.h<></stdio.h<></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></pre>	KAMPL	ES	
<pre>#include <stdio.h> #include <stdio.h< #include="" <stdio.h<="" td=""> #include <stdio.h< #inc<="" #include="" <stdio.h<="" td=""><td>The f</td><td>following shmem_atomic_fetch</td><td><i>a_add</i> example is for <i>C11</i> programs:</td></stdio.h<></stdio.h<></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></pre>	The f	following shmem_atomic_fetch	<i>a_add</i> example is for <i>C11</i> programs:
<pre>int min(void) { int old = -1; static int dot = 22; shmem_init(); int me = shmem_my_pe(); if (me == 1) old = shmem_atomic_fetch_add(sdst, 44, 0); shmem_barrier_all(); printf("%d; old = %d, dst = %d\n", me, old, dst); shmem_finalize(); return 0; } S.8 SHMEM_ATOMIC_ADD forms an atomic add operation on a remote symmetric data object. NOPSIS C11: void shmem_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE value, int pe); void shmem_cTYPENAME>_atomic_add(shmem_ctx_t ctx, TYPE value, int pe); void shmem_ctx_TYPENAME>_atomic_add(shmem_ctx_t ctx, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. - deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. - deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. - deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {int, long, long long}. </pre>	#inc #inc	<pre>lude <stdio.h> lude <shmem.h></shmem.h></stdio.h></pre>	
<pre>int old = -1; static int dst = 22; shmem_init(); int me = shmem_my_pe(); if (me == 1) old = shmem_atomic_fetch_add(&dst, 44, 0); shmem_barrier_all(); printf("%d: old = %d, dst = %d\n", me, old, dst); shmem_finalize(); return 0; } 3.8 SHMEM_ATOMIC_ADD forms an atomic add operation on a remote symmetric data object. NOPSIS C11: void shmem_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types specified by Table 4. C/C++: void shmem_ctx_cTYPENAME>_atomic_add(shmem_ctx_t ctx, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. - deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. - deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {int, long, long long}.</pre>	int 1 {	main(void)	
<pre>shmem_barrier_all(); printf("%d: old = %d, dst = %d\n", me, old, dst); shmem_finalize(); return 0; } S.8 SHMEM_ATOMIC_ADD forms an atomic add operation on a remote symmetric data object. NOPSIS C11: void shmem_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types specified by Table 4. C/C++: void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4 deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {int, long, long long}. </typename></pre>	i: s: s: i: i:	<pre>nt old = -1; tatic int dst = 22; hmem_init(); nt me = shmem_my_pe(); f (me == 1) old = shmem atomic fetch</pre>	add(&dst. 44, 0):
<pre>.3 SHMEM_ATOMIC_ADD forms an atomic add operation on a remote symmetric data object. NOPSIS C11: void shmem_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types specified by Table 4. C/C++: void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4 deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {<i>int, long, long long</i>}.</typename></pre>	si p: si r (}	<pre>hmem_barrier_all(); rintf("%d: old = %d, dst = hmem_finalize(); eturn 0;</pre>	%d\n", me, old, dst);
<pre>forms an atomic add operation on a remote symmetric data object. NOPSIS C11: void shmem_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types specified by Table 4. C/C++: void shmem_ctx_<typename>_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4</typename></typename></pre>	8 SHI	MEM ATOMIC ADD	
<pre>torms an atomic add operation on a remote symmetric data object. NOPSIS C11: void shmem_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types specified by Table 4. C/C++: void shmem_ctx_<typename>_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4</typename></typename></pre>	.0 511		
NOPSIS C11: void shmem_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types specified by Table 4. C/C++: void shmem_ <typename>_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. - deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {int, long, long long}.</typename></typename>	torms a	n atomic add operation on a ren	note symmetric data object.
<pre>C11: void shmem_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types specified by Table 4. C/C++: void shmem_<typename>_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {int, long, long long}.</typename></typename></pre>	NOPSIS	5	
<pre>where TYPE is one of the standard AMO types specified by Table 4. C/C++: void shmem_<typename>_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {int, long, long long}.</typename></typename></pre>	C11: void void	<pre>shmem_atomic_add(TYPE *de shmem atomic add(shmem ct.</pre>	st, TYPE value, int pe); x t ctx, TYPE *dest, TYPE value, int pe);
C/C++: void shmem_ <typename>_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where <i>TYPE</i> is one of the standard AMO types and has a corresponding <i>TYPENAME</i> specified by Table 4. - deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where <i>TYPE</i> is one of {<i>int</i>, <i>long</i>, <i>long long</i>}.</typename></typename>	wher	e TYPE is one of the standard A	AMO types specified by Table 4.
<pre>void shmem_<typename>_atomic_add(TYPE *dest, TYPE value, int pe); void shmem_ctx_<typename>_atomic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe); where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4.</typename></typename></pre>	C/C+	++:	
<pre>where TYPE is one of the standard AMO types and has a corresponding TYPENAME specified by Table 4. deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {int, long, long long}.</pre>	void void	<pre>shmem_<typename>_atomic_a shmem_ctx_<typename>_atom</typename></typename></pre>	dd(TYPE *dest, TYPE value, int pe); ic_add(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);
<pre>— deprecation start C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of {int, long, long long}.</pre>	wher	e TYPE is one of the standard A	AMO types and has a corresponding <i>TYPENAME</i> specified by Table 4.
C11: void shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of { <i>int</i> , <i>long</i> , <i>long long</i> }.	— de	eprecation start	
wold shmem_add(TYPE *dest, TYPE value, int pe); where TYPE is one of { <i>int</i> , <i>long</i> , <i>long long</i> }.	C11:		
where I IFE is one of { <i>int, long, long long</i> }.	void	shmem_add(TYPE *dest, TYP:	E value, int pe);
	wher	e IYPE is one of {int, long, lon	ig long }.

void shmem_	<typename>_add(TYPE</typename>	<pre>*dest, TYPE value, int pe);</pre>
where TYPE	is one of { int, long, long	g long } and has a corresponding TYPENAME specified by Table 4.
		deprecation end —
		deprecation ond
- deprecation	on start —	
FURIKAN:		
INTEGER pe		
CALL SUMEM	Value_14	vid po)
INTEGER + 8 V	alue i8	2_14, pe)
CALL SHMEM	INT8 ADD (dest. value	i8. pe)
		depresention and
DESCRIPTION		
Arguments	atre	The context on which to perform the operation. When this argument is
111	CIX	not provided the operation is performed on SHMEM CTX_DEFAULT
OUT	dast	The remotely accessible integer data object to be undeted on the remote
001	uesi	PF When using $C/C++$ the type of dest should match that implied in
		the SYNOPSIS section.
IN	value	The value to be atomically added to <i>dest</i> . When using $C/C++$, the type
		of value should match that implied in the SYNOPSIS section. When
		using Fortran, it must be of type integer with an element size of dest.
IN	pe	An integer that indicates the PE number upon which dest is to be up-
		dated. When using Fortran, it must be a default integer value.
API descript	tion	
The she	man atomio add nouti	a performs on stamic add aparation. It adds usive to dest on DE as and
atomica	nem_atomic_add foutil	hout returning the value
atonnea	iny updates the desi with	nout returning the value.
When u	sing Fortran dest and w	value must be of the following type:
when u	ising Forman, aest and v	alue must be of the following type.
Routin	ne	Data type of <i>dest</i> and <i>value</i>
OTD 4		1 huto integen
SHMI	EM_INT& ADD	4-byte integer
SHMI	EIVI_IIN I 8_ADD	o-byte integer
_		
Return Valu	es	
None.		
Notes		
None.		
EXAMPLES		

```
#include <stdio.h>
#include <shmem.h>
int main(void)
{
    static int dst = 22;
    shmem_init();
    int me = shmem_my_pe();
    if (me == 1)
        shmem_atomic_add(&dst, 44, 0);
    shmem_barrier_all();
    printf("%d: dst = %d\n", me, dst);
    shmem_finalize();
    return 0;
}
```

9.8.9 SHMEM_ATOMIC_FETCH_AND

Atomically perform a fetching bitwise AND operation on a remote data object.

SYNOPSIS

C11:

TYPE shmem_atomic_fetch_and(TYPE *dest, TYPE value, int pe);
TYPE shmem_atomic_fetch_and(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);

where TYPE is one of the bitwise AMO types specified by Table 6.

C/C++:

```
TYPE shmem_<TYPENAME>_atomic_fetch_and(TYPE *dest, TYPE value, int pe);

TYPE shmem_ctx_<TYPENAME>_atomic_fetch_and(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);

where TYPE is one of the bitwise AMO types and has a corresponding TYPENAME specified by Table 6.
```

DESCRIPTION

Arguments

IN	ctx	The context on which to perform the operation. When this argument is
		not provided, the operation is performed on SHMEM_CTX_DEFAULT.
OUT	dest	A pointer to the remotely accessible data object to be updated.
IN	value	The operand to the bitwise AND operation.
IN	pe	An integer value for the PE on which <i>dest</i> is to be updated.

API description

shmem_atomic_fetch_and atomically performs a fetching bitwise AND on the remotely accessible data object pointed to by *dest* at PE *pe* with the operand *value*.

Return Values

The value pointed to by *dest* on PE pe immediately before the operation is performed.

Notes

None.

1	9.8.10 SHMEM_ATO	OMIC_AND			
2 3	Atomically perform a non-fetching bitwise AND operation on a remote data object.				
4 5	SYNOPSIS				
6 7 8	C11: void shmem_atom: void shmem_atom:	ic_and(TYPE *de	est, TYPE value, int pe); :x_t ctx, TYPE *dest, TYPE value, int pe);		
9	where TYPE is one	e of the bitwise A	MO types specified by Table 6.		
10	C/C++:				
11 12	void shmem_< TYP void shmem_ctx_<	ENAME >_atomic_a < TYPENAME >_atom	and(TYPE *dest, TYPE value, int pe); nic_and(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);		
13	where TYPE is one	e of the bitwise A	MO types and has a corresponding <i>TYPENAME</i> specified by Table 6.		
14 15 16	DESCRIPTION				
17 18	Arguments				
19 20	IN	ctx	The context on which to perform the operation. When this argument is not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .		
21	OUT	dest	A pointer to the remotely accessible data object to be updated.		
22	IN	value	The operand to the bitwise AND operation.		
23	IN	pe	An integer value for the PE on which <i>dest</i> is to be updated.		
24 25					
26	API description				
27					
28	shmem_atom	<i>iic_and</i> atomical	Ity performs a non-retching bitwise AND on the remotely accessible data		
30	object pointe	d to by desi di 11	pe with the operated value.		
31					
32	Return Values				
33	None.				
34			*		
35	Notes				
36	None.				
37					
39	9.8.11 SHMEM_ATO	OMIC_FETCH_	OR		
40 41	Atomically perform a fet	tching bitwise OF	R operation on a remote data object.		
42 43	SYNOPSIS				
44	C11:				
45	TYPE shmem_atom:	ic_fetch_or(TYE	PE *dest, TYPE value, int pe);		
46	TYPE shmem_atom:	ic_fetch_or(shn	<pre>nem_ctx_t ctx, TYPE *dest, TYPE value, int pe);</pre>		
4/	where <i>TYPE</i> is one of the bitwise AMO types specified by Table 6.				
+0	C/C++:				

TYPE charge (TYP	ENAME: atomic i	fatch an (MVDE ideat MVDE value int value	1
TYPE shmem_ctx_	<pre>CONTE</pre>	<pre>mic_fetch_or(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);</pre>	2
where TYPE is on	e of the bitwise A	MO types and has a corresponding <i>TYPENAME</i> specified by Table 6.	3
			4
			5
DESCRIPTION			6
Arguments			7
IN	etr	The context on which to perform the operation. When this argument is	0 Q
111	CIX	not provided, the operation is performed on SHMEM CTX DEFAULT.	10
OUT	dest	A pointer to the remotely accessible data object to be updated.	11
IN	value	The operand to the bitwise OR operation.	12
IN	pe	An integer value for the PE on which <i>dest</i> is to be updated.	13
			13
			14
API description			15
			10
shmem ator	nic fetch or ator	nically performs a fetching bitwise OR on the remotely accessible data object	17
pointed to by	y <i>dest</i> at PE <i>pe</i> wi	ith the operand <i>value</i> .	10
1	× 1		19
			20
Roturn Volues			21
The value po	ninted to by <i>dest</i> of	on PE n_e immediately before the operation is performed	22
The value pe		si i diperininediately before ale operation is performed.	23
Notes			24
None			25
rone.			26
			27
9 12 SUMEM ATC	MIC OP		28
.0.12 SHIVIEWI_AIQ	JWIIC_OK		29
Atomically perform a no	on-fetching bitwis	e OR operation on a remote data object.	30
			31
SYNOPSIS			32
C11:			33
void shmem atom	ic or (TYPE + des	st. TYPE value. int ne).	34
void shmem_atom	ic_or(shmem_ct	<pre>x_t ctx, TYPE *dest, TYPE value, int pe);</pre>	35
where TYPE is on	e of the bitwise A	MO types specified by Table 6	36
			37
U/U++:	ENAMES atomic	or (TYDE + doct TYDE walue int po).	38
void shmem_ <iip< td=""><td><typename> ator</typename></td><td>nic or(shmem ctx t ctx. TYPE *dest. TYPE value. int pe):</td><td>39</td></iip<>	<typename> ator</typename>	nic or(shmem ctx t ctx. TYPE *dest. TYPE value. int pe):	39
where TYPE is on	$=$ of the bitwise Δ	MO types and has a corresponding TVPENAME specified by Table 6	40
where I II E 18 01	te of the oftwise A	Two types and has a corresponding TH ENAME specified by fault 0.	41
			42
DESCRIPTION			43
			44
Arguments			45

IN	ctx	The context on which to perform the operation. When this argument is
		not provided, the operation is performed on SHMEM_CTX_DEFAULT.
OUT	dest	A pointer to the remotely accessible data object to be updated.

1	IN IN	value pe	The operand to the bitwise OR operation. An integer value for the PE on which <i>dest</i> is to be updated.
3		-	
4			
5	API description		
6	•		
7	shmem_atom	<i>ic_or</i> atomically	performs a non-fetching bitwise OR on the remotely accessible data object
8	pointed to by	<i>dest</i> at PE <i>pe</i> wi	th the operand <i>value</i> .
9			
10			
12	Return Values		
13	None.		
14	Nistan		
15	Notes		
16	rone.		
17			
18	9.8.13 SHMEM ATO	MIC FETCH	XOR
19			
20	Atomically perform a feto	ching bitwise ex	clusive OR (XOR) operation on a remote data object.
21	SYNODSIS		
22	SYNOPSIS		
23	C11:		
24	TYPE shmem_atomi	c_fetch_xor(T	<pre>YPE *dest, TYPE value, int pe);</pre>
25	TYPE shmem_atomi	c_fetch_xor(s)	<pre>hmmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);</pre>
20	where TYPE is one	of the bitwise A	AMO types specified by Table 6.
28	C/C++:		
29	TYPE shmem_ <type< th=""><th>NAME>_atomic_:</th><th><pre>fetch_xor(TYPE *dest, TYPE value, int pe); mic fetch vor(shmem ctv t ctv TYPE *dest TYPE value int pe);</pre></th></type<>	NAME>_atomic_:	<pre>fetch_xor(TYPE *dest, TYPE value, int pe); mic fetch vor(shmem ctv t ctv TYPE *dest TYPE value int pe);</pre>
30	where TVDE is one	of the bituring	MO types and has a corresponding TYPENAME specified by Table 6
31	where <i>TTFL</i> is one	of the bitwise P	Two types and has a corresponding <i>TTPENAME</i> specified by Table 6.
32			
33	DESCRIPTION		Y
34			·
35	Arguments		
36	IN	ctx	The context on which to perform the operation. When this argument is
37			not provided, the operation is performed on <i>SHMEM_CTX_DEFAULT</i> .
38	OUT	dest	A pointer to the remotely accessible data object to be updated.
39	IN	value	The operand to the bitwise XOR operation.
40	IN	pe	An integer value for the PE on which dest is to be updated.
42			
43			
44	API description		
45	alaa	in fatale and it	omically performs a fatabing bitwice VOD on the super-table second in the
46	snmem_atom	ic_ <i>jeicn_xor</i> at to by <i>dest</i> at Pl	Sincary performs a recentling only set AOK on the remotery accessible data $E pe$ with the operand <i>value</i>
47	object pointee		2 pe mai die operand vanae.
48			

Return Values

The value pointed to by dest on PE pe immediately before the operation is performed.

Notes

None.

9.8.14 SHMEM_ATOMIC_XOR

Atomically perform a non-fetching bitwise exclusive OR (XOR) operation on a remote data object.

SYNOPSIS

C11:

```
void shmem_atomic_xor(TYPE *dest, TYPE value, int pe);
void shmem_atomic_xor(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);
where TYPE is one of the bitwise AMO types specified by Table 6.
```

C/C++:

```
void shmem_<TYPENAME>_atomic_xor(TYPE *dest, TYPE value, int pe);
void shmem_ctx_<TYPENAME>_atomic_xor(shmem_ctx_t ctx, TYPE *dest, TYPE value, int pe);
where TYPE is one of the bitwise AMO types and has a corresponding TYPENAME specified by Table 6.
```

DESCRIPTION

Arguments

IN	ctx	The context on which to perform the operation. When this argument is
		not provided, the operation is performed on SHMEM_CTX_DEFAULT.
OUT	dest	A pointer to the remotely accessible data object to be updated.
IN	value	The operand to the bitwise XOR operation.
IN	pe	An integer value for the PE on which <i>dest</i> is to be updated.

API description

shmem_atomic_xor atomically performs a non-fetching bitwise XOR on the remotely accessible data object pointed to by *dest* at PE *pe* with the operand *value*.

Return Values

None.

Notes

None.

9.9 Collective Routines

Collective routines are defined as communication or synchronization operations on a group of PEs called an active set. The collective routines require all PEs in the active set to simultaneously call the routine. A PE that is not in the active

set calling the collective routine results in undefined behavior. All collective routines have an active set as an input parameter except *shmem_barrier_all* and *shmem_sync_all*. Both *shmem_barrier_all* and *shmem_sync_all* must be called by all PEs of the OpenSHMEM program.

The active set is defined by the arguments PE_start , $logPE_stride$, and PE_size . PE_start specifies the starting PE number and is the lowest numbered PE in the active set. The stride between successive PEs in the active set is 2^{logPE_stride} and $logPE_stride$ must be greater than or equal to zero. PE_size specifies the number of PEs in the active set and must be greater than zero. The active set must satisfy the requirement that its last member corresponds to a valid PE number, that is $0 \le PE_start + (PE_size - 1) * 2^{logPE_stride} < npes$. All PEs participating in the collective routine must provide the same values for these arguments. If any of these requirements are not met, the behavior is undefined.

Another argument important to collective routines is *pSync*, which is a symmetric work array. All PEs participating in a collective must pass the same *pSync* array. On completion of a collective call, the *pSync* is restored to its original contents. The user is permitted to reuse a *pSync* array if all previous collective routines using the *pSync* array have been completed by all participating PEs. One can use a synchronization collective routine such as *shmem_barrier* to ensure completion of previous collective routines. The *shmem_barrier* and *shmem_sync* routines allow the same *pSync* array to be used on consecutive calls as long as the PEs in the active set do not change.

All collective routines defined in the Specification are blocking. The collective routines return on completion. The collective routines defined in the OpenSHMEM Specification are:

- shmem_barrier_all
- shmem_barrier
- shmem_sync_all
- shmem_sync
- shmem_broadcast{32, 64}
- shmem_collect{32, 64}
- shmem_fcollect{32, 64}
- Reductions for the following operations: AND, MAX, MIN, SUM, PROD, OR, XOR
- shmem_alltoall{32, 64}
- shmem_alltoalls{32, 64}

9.9.1 SHMEM_BARRIER_ALL

Registers the arrival of a PE at a barrier and blocks the PE until all other PEs arrive at the barrier and all local updates and remote memory updates on the default context are completed.

SYNOPSIS

40	C/C++:	
41	<pre>void shmem_barrier_all(void);</pre>	
42	— deprecation start —	
43	FORTRAN:	
44	CALL SHMEM_BARRIER_ALL	
45		deprecation end —
46		
47		

DESCRIPTION

Arguments

None.

API description

The *shmem_barrier_all* routine registers the arrival of a PE at a barrier. Barriers are a mechanism for synchronizing all PEs at once. This routine blocks the PE until all PEs have called *shmem_barrier_all*. In a multithreaded OpenSHMEM program, only the calling thread is blocked.

Prior to synchronizing with other PEs, *shmem_barrier_all* ensures completion of all previously issued memory stores and remote memory updates issued on the default context via OpenSHMEM AMOs and RMA routine calls such as *shmem_int_add*, *shmem_put32*, *shmem_put_nbi*, and *shmem_get_nbi*.

Return Values

None.

Notes

The *shmem_barrier_all* routine can be used to portably ensure that memory access operations observe remote updates in the order enforced by initiator PEs.

Calls to *shmem_ctx_quiet* can be performed prior to calling the barrier routine to ensure completion of operations issued on additional contexts.

EXAMPLES

The following *shmem_barrier_all* example is for C11 programs:

```
#include <stdio.h>
#include <shmem.h>
int main(void)
{
    static int x = 1010;
```

shmem_init();
int me = shmem_my_pe();

```
int npes = shmem_n_pes();
/* put to next PE in a circular fashion */
shmem_p(&x, 4, (me + 1) % npes);
/* synchronize all PEs */
shmem_barrier_all();
printf("%d: x = %d\n", me, x);
shmem_finalize();
return 0;
```

}

9.9.2 SHMEM_BARRIER

Performs all operations described in the *shmem_barrier_all* interface but with respect to a subset of PEs defined by the active set.

C/C++:		
void shmem_barrie	er(int PE_start,	<pre>int logPE_stride, int PE_size, long *pSync);</pre>
- deprecation start		
FORTRAN:		
INTEGER PE_start,	, logPE_stride, E	PE_size
INTEGER pSync(SHI	MEM_BARRIER_SYNC_	SIZE)
CALL SHMEM_BARRII	ER(PE_start, logE	PE_stride, PE_size, pSync)
		deprecation end –
CRIPTION		
Arguments		
IN	PE start	The lowest PE number of the active set of PEs. <i>PE</i> start must be
	—	type integer. When using <i>Fortran</i> , it must be a default integer value.
IN	logPE_stride	The log (base 2) of the stride between consecutive PE numbers in the
		active set. logPE_stride must be of type integer. When using Fortra
		it must be a default integer value.
IN	PE_size	The number of PEs in the active set. <i>PE_size</i> must be of type intege
IN	en Course o	when using <i>Fortran</i> , it must be a default integer value.
110	psync	A symmetric work array of size SHMEM_DARKIER_STIVE_SIZE.
		<i>pSync</i> must be an array of elements of default integer type. Every el
		ment of this array must be initialized to SHMEM_SYNC_VALUE befo
		any of the PEs in the active set enter <i>shmem_barrier</i> the first time.
API description		
r i i i i i i i i i i i i i i i i i i i		
shmem_barrie	er is a collective	synchronization routine over an active set. Control returns from
shmem_barrie	er after all PEs in t	he active set (specified by PE_start, logPE_stride, and PE_size) have
called shmem_	_barrier.	
As with all O	penSHMEM collect	tive routines, each of these routines assumes that only PEs in the activ
set call the rol	utine. If a PE not in	the active set calls an OpenSHMEM collective routine, the behavior i
The values of	orguments DE star	et log DE stride and DE size must be the same value on all DEs in the
active set. The	arguments <i>TE_star</i> same work array n	ust be passed in <i>pSync</i> to all PEs in the active set.
shmem barrie	er ensures that all t	previously issued stores and remote memory updates, including AMO
and RMA ope	erations, done by an	y of the PEs in the active set on the default context are complete befor
returning.		•
The same <i>pSy</i>	nc array may be reu	sed on consecutive calls to <i>shmem_barrier</i> if the same active set is used
Return Values		
None.		
Notes		
If the <i>pSync</i>	array is initialized	at the run time, all PEs must be synchronized before the first call t
shmem_barrie	er (e.g., by shmem_	<i>_barrier_all</i>) to ensure the array has been initialized by all PEs befor
it is used.		

	additional synchronization beyond that implied by <i>shmem_barrier</i> itself is necessary in this case.
	The <i>shmem_barrier</i> routine can be used to portably ensure that memory access operations observe remote updates in the order enforced by initiator PEs.
	Calls to <i>shmem_ctx_quiet</i> can be performed prior to calling the barrier routine to ensure completion of operations issued on additional contexts.
	No team-based barrier is provided by OpenSHMEM, as a team may have any number of communication contexts associated with the team. Applications seeking such an idiom should call <i>shmem_ctx_quiet</i> on the desired context, followed by a call to <i>shmem_team_sync</i> on the desired team.
EXAM	PLES
Т	The following barrier example is for <i>C11</i> programs:
# #	<pre>include <stdio.h> include <shmem.h></shmem.h></stdio.h></pre>
i {	nt main (void)
	<pre>static int x = 10101; static long pSync[SHMEM_BARRIER_SYNC_SIZE]; for (int i = 0; i < SHMEM_BARRIER_SYNC_SIZE; i++) pSync[i] = SHMEM_SYNC_VALUE;</pre>
	<pre>shmem_init(); int me = shmem_my_pe(); int npes = shmem_n_pes();</pre>
	<pre>if (me % 2 == 0) { /* put to next even PE in a circular fashion */ shmem_p(&x, 4, (me + 2) % npes); /* synchronize all even pes */ shmem_barrier(0, 1, (npes / 2 + npes % 2), pSync); } printf("%d: x = %d\n", me, x); ebmom finalize();</pre>
}	return 0;
9.9.3	SHMEM_SYNC_ALL
Registe	rs the arrival of a PE at a barrier and suspends PE execution until all other PEs arrive at the barrier.
SYNO	PSIS
v	C/C++: oid shmem_sync_all(void);
DESCI	RIPTION
A	arguments
	None.

If the active set does not change, *shmem_barrier* can be called repeatedly with the same *pSync* array. No

of

1	API description		
3 4 5 6 7 8 9	The <i>shmem_sy</i> synchronizing multithreaded In contrast wit of previously i OpenSHMEM	<i>ync_all</i> routine regiall PEs at once. Th OpenSHMEM prog th the <i>shmem_barri</i> ssued memory store routines.	sters the arrival of a PE at a barrier. Barriers are a fast mechanism for is routine blocks the PE until all PEs have called <i>shmem_sync_all</i> . In a ram, only the calling thread is blocked. <i>er_all</i> routine, <i>shmem_sync_all</i> only ensures completion and visibility es and does not ensure completion of remote memory updates issued via
10	B aturn Values		
11	None.		
12 13 14 15 16 17	Notes The <i>shmem_sy</i> mote updates i of remote upda	<i>ync_all</i> routine can n the order enforce ates with a call to <i>sh</i>	be used to portably ensure that memory access operations observe re- d by the initiator PEs, provided that the initiator PE ensures completion <i>mem_quiet</i> prior to the call to the <i>shmem_sync_all</i> routine.
18 19	0.0.4 SHMEM SVNC		
20 21 22	Performs all operations der team or active set.	scribed in the shme	<i>m_sync_all</i> interface but with respect to a subset of PEs defined by the a
23 24	SYNOPSIS		
25	C11:		
26	void shmem_sync(s	<pre>shmem_team_t team</pre>);
27	C/C++:	nt PF start int	logPE stride int PE size long +pSync).
28	void shmem_team_s	sync(shmem_team_t	team);
30			
31	DESCRIPTION		
32	Arguments		
34			
35	IN	team	The team over which to perform the operation.
36	IN	PE_start	The lowest PE number of the active set of PEs. <i>PE_start</i> must be of two integer
37 38	IN	logPE_stride	The log (base 2) of the stride between consecutive PE numbers in the active set <i>logPE</i> stride must be of type integer
39	IN	PE size	The number of PEs in the active set. <i>PE size</i> must be of type integer.
40	IN	pSync	A symmetric work array. In $C/C++$, pSync must be of type long
41 42			and size <i>SHMEM_BARRIER_SYNC_SIZE</i> . Every element of this array must be initialized to <i>SHMEM_SYNC_VALUE</i> before any of the PEs in
43			the active set enter <i>shmem_sync</i> the first time.
44			
45	API description		
46			
47 48	shmem_sync is shmem_sync a	s a collective syncl after all PEs in the	pronization routine over a team or an active set. Control returns from e specified team or active set (specified by <i>PE start</i> , <i>logPE stride</i> .

and *PE_size*) have called *shmem_sync*. An active set is specified by the triple of values: *PE_start*, *logPE_stride*, and *PE_size*.

As with all **OpenSHMEM** active set-based collective routines, each of these routines assumes that only PEs in the active set call the routine. If a PE not in the active set calls an OpenSHMEM collective routine, the behavior is undefined.

The values of arguments *PE_start*, *logPE_stride*, and *PE_size* must be equal on all PEs in the active set. The same work array must be passed in *pSync* to all PEs in the active set.

In contrast with the *shmem_barrier* routine, *shmem_sync* only ensures completion and visibility of previously issued memory stores and does not ensure completion of remote memory updates issued via Open-SHMEM routines.

The same *pSync* array may be reused on consecutive calls to *shmem_sync* if the same active set is used.

Return Values

None.

Notes

If the *pSync* array is initialized at run time, another method of synchronization (e.g., *shmem_sync_all*) must be used before the initial use of that *pSync* array by *shmem_sync*.

If the active set does not change, *shmem_sync* can be called repeatedly with the same *pSync* array. No additional synchronization beyond that implied by *shmem_sync* itself is necessary in this case.

The *shmem_sync* routine can be used to portably ensure that memory access operations observe remote updates in the order enforced by the initiator PEs, provided that the initiator PE ensures completion of remote updates with a call to *shmem_quiet* prior to the call to the *shmem_sync* routine.

EXAMPLES

The following *shmem_sync_all* and *shmem_sync* example is for *C11* programs:

```
#include <stdio.h>
#include <shmem.h>
int main(void)
  static int x = 10101;
  static long pSync[SHMEM_BARRIER_SYNC_SIZE];
   shmem_init();
  int me = shmem_my_pe();
  int npes = shmem_n_pes();
   for (int i = 0; i < SHMEM_BARRIER_SYNC_SIZE; i++)</pre>
     pSync[i] = SHMEM_SYNC_VALUE;
   shmem_sync_all();
  if (me % 2 == 0) {
      /* put to next even PE in a circular fashion */
      shmem_p(\&x, 4, (me + 2) % npes);
      /* synchronize all even pes */
      shmem_quiet();
      shmem_sync(0, 1, (npes / 2 + npes % 2), pSync);
  printf("%d: x = %d n", me, x);
  shmem_finalize();
  return 0:
```

2

3

4

6

9

10

11 12 13

14

15 16

17

18

19

20

21

22

23

24 25

26 27

28

29

30 31

32

33

34

35

36 37

38

39

40

41

42

43

44

45

46

47

9.9.5 SHMEM_TEAM_BROADCAST

Broadcasts a block of data from one PE in a team to all other PEs in the team

SYNOPSIS

C/C++: void shmem_team_broadcast32(shmem_team_t team, void *dest, const void *source, size_t nelems, int PE_root); void shmem_team_broadcast64(shmem_team_t team, void *dest, const void *source, size_t nelems, int PE_root);

DESCRIPTION

Arguments		
IN	team	A valid OpenSHMEM team handle to a team which has been created without disabling support for collective operations.
OUT	dest	A symmetric data object. See the table below in this description for allowable types
IN	source	A symmetric data object that can be of any data type that is permissible for the <i>dest</i> argument.
IN	nelems	The number of elements in <i>source</i> . For <i>shmem_team_broadcast32</i> , this is the number of 32-bit halfwords. nelems must be of type <i>size_t</i> .
IN	PE_root	Zero-based ordinal of the PE, with respect to the team, from which the data is copied. <i>PE_root</i> must be of type <i>int</i> .

API description

OpenSHMEM team broadcast routines are collective routines over an existing team. They copy data object *source* on the processor specified by *PE_root* and store the values at *dest* on the other PEs that are members of the team. The data is not copied to the *dest* area on the root PE.

If the team has been created with the *SHMEM_TEAM_NOCOLLECTIVE* option, it will not have the required support structures to complete this routine. If such a team is passed to this or any other team collective routine, the behavior is undefined.

As with all OpenSHMEM routines where the operation occurs over a given team, PE numbering is relative to the team. The specified root PE must be a valid PE number for the team, between 0 and N-1, where N is the size of the team.

The values of the argument *PE_root* must be the same value on all PEs in the team. The same *dest* and *source* data objects must be passed by all PEs in the team.

Upon return from a broadcast routine, the following are true for the local PE:

- If the current PE is not the root PE, the *dest* data object is updated.
- The source data object may be safely reused.

Error checking will be done to detect a value of *SHMEM_TEAM_NULL* passed for the team argument. In that case, the program will abort with an informative error message. If an invalid team handle is passed to the routine, the behavior is undefined.

The *dest* and *source* data objects must conform to certain typing constraints, which are as follows:

	Routine		Data type of <i>dest</i> and <i>source</i>
5	shmem_team_b	proadcast64	Any noncharacter type that has an element size of 64 bits. C/C++ structures are NOT allowed.
S	shmem_team_b	proadcast32	Any noncharacter type that has an element size of 32 bits. C/C++ structures are NOT allowed.
Return	Values		
N	one.		
Notes			
9.9.6 SHMI	EM_BROADC	AST	
Broadcasts a	block of data fro	om one PE to o	ne or more destination PEs.
SYNOPSIS			
010			
void si PE void si	• hmem_broadcas _start, int l hmem_broadcas	t32(void *des ogPE_stride, t64(void *des	st, const void *source, size_t nelems, int PE_root, int int PE_size, long *pSync); st, const void *source, size_t nelems, int PE_root, int
PE	_start, int l	ogPE_stride,	<pre>int PE_size, long *pSync);</pre>
— depr FORTI	recation start — RAN:		
INTEGE	R nelems, PE_	root, PE_star	rt, logPE_stride, PE_size
INTEGE	R pSync(SHMEM	_BCAST_SYNC_S	JIZE)
CALL SI CALL SI CALL SI	HMEM_BROADCAS' HMEM_BROADCAS' HMEM_BROADCAS'	T4(dest, sour T8(dest, sour T32(dest, sou	cce, nelems, PE_root, PE_start, logPE_stride, PE_size, pSync) cce, nelems, PE_root, PE_start, logPE_stride, PE_size, pSync) arce, nelems, PE_root, PE_start, logPE_stride, PE_size,pSync)
CALL S	HMEM_BROADCAS	T64(dest, sou	<pre>arce, nelems, PE_root, PE_start, logPE_stride, PE_size,pSync)</pre>
			deprecation end —
DESCRIPTI	ON		
Argum	ents		
(OUT	dest	A symmetric data object.
]	IN	source	A symmetric data object that can be of any data type that is permissible for the <i>dest</i> argument.
]	IN	nelems	The number of elements in <i>source</i> . For <i>shmem_broadcast32</i> and <i>shmem_broadcast4</i> , this is the number of 32-bit halfwords. nelems must be of type <i>size_t</i> in <i>C</i> . When using <i>Fortran</i> , it must be a default integer value
]	IN	PE_root	Zero-based ordinal of the PE, with respect to the active set, from which the data is copied. Must be greater than or equal to 0 and less than
			<i>PE_size. PE_root</i> must be of type integer. When using <i>Fortran</i> , it must

be a default integer value.

1	IN	PE_start	The lowest PE number of the active set of PEs. <i>PE_start</i> must be of type integer. When using <i>Fortron</i> it must be a default integer value
2 3	IN	logPE_stride	The log (base 2) of the stride between consecutive PE numbers in the
4			active set. <i>log_PE_stride</i> must be of type integer. When using <i>Fortran</i> , it must be a default integer value
5	IN	PE_size	The number of PEs in the active set. <i>PE_size</i> must be of type integer. When using <i>Fortran</i> it must be a default integer value.
7	IN	pSvnc	A symmetric work array of size SHMEM BCAST SYNC SIZE. In
8		r - y	C/C++, pSync must be an array of elements of type long. In
9			Fortran, pSync must be an array of elements of default integer
10			type. Every element of this array must be initialized with the value
11			shmem broadcast.
12			
14			
15	API description		
16	OpenSHME	M broadcast routines	s are collective routines. They copy data object <i>source</i> on the proces-
17	sor specified	by <i>PE_root</i> and sto <i>PE_size</i> The data	re the values at <i>dest</i> on the other PEs specified by the triplet <i>PE_start</i> , is not conject to the <i>dest</i> area on the root PE.
19	As with all C	OpenSHMEM collect	tive routines, each of these routines assumes that only PEs in the active
20	set call the re	outine. If a PE not in	the active set calls an OpenSHMEM collective routine, the behavior is
21	undefined.		
22	The values of	f arguments <i>PE_root</i>	, <i>PE_start</i> , <i>logPE_stride</i> , and <i>PE_size</i> must be the same value on all PEs
23	all PFs in the	set. The same <i>dest</i> at	id source data objects and the same psync work array must be passed by
24	Before any P	E calls a broadcast ro	putine, the following conditions must be ensured:
25	• The <i>n</i> Sv	<i>nc</i> array on all PEs in	the active set is not still in use from a prior call to a broadcast routine
26	• The desi	t array on all PEs in t	he active set is ready to accept the broadcast data.
27	Otherwise, th	ne behavior is undefir	ned.
28	Upon return	from a broadcast rou	tine, the following are true for the local PE:
29	• If the cu	rrent PE is not the ro	ot PE, the <i>dest</i> data object is updated.
31	• The source	<i>rce</i> data object may b	be safely reused.
32	• The value	ies in the <i>pSync</i> array	v are restored to the original values.
33			
34			
35	The <i>dest</i> and	source data objects 1	nust conform to certain typing constraints, which are as follows:
36	Routine		Data type of <i>dest</i> and <i>source</i>
37			51
38	shmem bro	padcast8,	Any noncharacter type that has an element size of 64 bits. No
39	shmem_bro	padcast64	<i>Fortran</i> derived types or $C/C++$ structures are allowed.
40	shmem_bro	padcast4,	Any noncharacter type that has an element size of 32 bits. No
41	shmem_bro	padcast32	Fortran derived types or $C/C++$ structures are allowed.
42			
45	Return Values		
45	None.		
46			
47	Notes		
48	All OpenSHI routines that	use the same <i>pSync</i> a	nes restore <i>psync</i> to its original contents. Multiple calls to OpenSHMEM array do not require that <i>pSync</i> be reinitialized after the first call.

The user must ensure that the *pSync* array is not being updated by any PE in the active set while any of the PEs participates in processing of an OpenSHMEM broadcast routine. Be careful to avoid these situations: If the *pSync* array is initialized at run time, before its first use, some type of synchronization is needed to ensure that all PEs in the active set have initialized *pSync* before any of them enter an OpenSHMEM routine called with the *pSync* synchronization array. A *pSync* array may be reused on a subsequent OpenSHMEM broadcast routine only if none of the PEs in the active set are still processing a prior OpenSHMEM broadcast routine call that used the same *pSync* array. In general, this can be ensured only by doing some type of synchronization.

EXAMPLES

In the following examples, the call to shmem_broadcast64 copies source on PE 4 to dest on PEs 5, 6, and 7.

```
C/C++ example:
#include <stdio.h>
#include <stdlib.h>
#include <shmem.h>
int main(void)
   static long pSync[SHMEM_BCAST_SYNC_SIZE];
   for (int i = 0; i < SHMEM_BCAST_SYNC_SIZE; i++)</pre>
      pSync[i] = SHMEM_SYNC_VALUE;
   static long source[4], dest[4];
   shmem_init();
   int me = shmem_my_pe();
   int npes = shmem_n_pes();
   if (me == 0)
      for (int i = 0; i < 4; i++)</pre>
        source[i] = i;
   shmem_broadcast64(dest, source, 4, 0, 0, 0, npes, pSync);
   printf("%d: %ld, %ld, %ld\n", me, dest[0], dest[1], dest[2], dest[3]);
   shmem_finalize();
   return 0;
}
Fortran example:
INCLUDE "shmem.fh"
INTEGER PSYNC (SHMEM_BCAST_SYNC_SIZE)
INTEGER DEST, SOURCE, NLONG, PE_ROOT, PE_START,
& LOGPE_STRIDE, PE_SIZE, PSYNC
COMMON /COM/ DEST, SOURCE
DATA PSYNC /SHMEM_BCAST_SYNC_SIZE*SHMEM_SYNC_VALUE/
CALL SHMEM_BROADCAST64 (DEST, SOURCE, NLONG, 0, 4, 0, 4, PSYNC)
```

9.9.7 SHMEM_COLLECT, SHMEM_FCOLLECT

Concatenates blocks of data from multiple PEs to an array in every PE.

SYNOPSIS

C/C++:

2

3

4

5

6

7

9 10

11 12

13

14

15

16

17

18

19

20

21

22

23

24 25

26

27

28

29

30

31 32

33

34

36

37

38

39 40

41 42 43

44

45 46

1	<pre>void shmem_collect32(void *dest, const void *source, size_t nelems, int PE_start, int</pre>
2	<pre>logPE_stride, int PE_size, long *pSync);</pre>
3	<pre>void shmem_collect64(void *dest, const void *source, size_t nelems, int PE_start, int</pre>
1	logPE_stride, int PE_size, long *pSync);
	<pre>void shmem_fcollect32(void *dest, const void *source, size_t nelems, int PE_start, int</pre>
5	logPE_stride, int PE_size, long *pSync);
5	<pre>void shmem_fcollect64(void *dest, const void *source, size_t nelems, int PE_start, int</pre>
7	logPE_stride, int PE_size, long *pSync);
3	deprecation start
)	FORTRAN:
0	INTEGER nelems
1	INTEGER PE_start, logPE_stride, PE_size
2	INTEGER pSync(SHMEM_COLLECT_SYNC_SIZE)
3	CALL SHMEM_COLLECT4(dest, source, nelems, PE_start, logPE_stride, PE_size, pSync)
4	CALL SHMEM_COLLECT8(dest, source, nelems, PE_start, logPE_stride, PE_size, pSync)
-	CALL SHMEM_COLLECT32(dest, source, nelems, PE_start, logPE_stride, PE_size, pSync)
3	CALL SHMEM_COLLECT64(dest, source, nelems, PE_start, logPE_stride, PE_size, pSync)
6	CALL SHMEM_FCOLLECT4(dest, source, nelems, PE_start, logPE_stride, PE_size, pSync)
7	CALL SHMEM_FCOLLECT8(dest, source, nelems, PE_start, logPE_stride, PE_size, pSync)
8	CALL SHMEM_FCOLLECT32(dest, source, nelems, PE_start, logPE_stride, PE_size, pSync)
9	CALL SHMEM_FCOLLECT64(dest, source, nelems, PE_start, logPE_stride, PE_size, pSync)

deprecation end —

DESCRIPTION

20

21 22

23 24

Arguments

25	·	
26	OUT	dest
27		
28		
29	-	
30		
31		
32		
33		
34	IN	source
35		
36	IN	nelems
37	IN	
38	IIN	PE_start
39	IN	logPE_stride
40		
41		
42	IN	PE_size
43		
44	IN	pSync
45		
46		
47		
48		

A symmetric array. The *dest* argument must be large enough to accept the concatenation of the *source* arrays on all participating PEs. The data types are as follows: For *shmem_collect8*, *shmem_collect64*, *shmem_fcollect8*, and *shmem_fcollect64*, any data type with an element size of 64 bits. *Fortran* derived types, *Fortran* character type, and C/C++ structures are not permitted. For *shmem_collect32*, any data type with an element size of 32 bits. *Fortran* derived types, *Fortran* character type, and C/C++ structures are not permitted.

A symmetric data object that can be of any type permissible for the *dest* argument.

The number of elements in the *source* array. *nelems* must be of type $size_t$ for C. When using *Fortran*, it must be a default integer value.

The lowest PE number of the active set of PEs. *PE_start* must be of type integer. When using *Fortran*, it must be a default integer value.

The log (base 2) of the stride between consecutive PE numbers in the active set. *logPE_stride* must be of type integer. When using *Fortran*, it must be a default integer value.

The number of PEs in the active set. *PE_size* must be of type integer. When using *Fortran*, it must be a default integer value.

A symmetric work array of size SHMEM_COLLECT_SYNC_SIZE. In C/C++, pSync must be an array of elements of type long. In Fortran, pSync must be an array of elements of default integer type. Every element of this array must be initialized with the value SHMEM_SYNC_VALUE before any of the PEs in the active set enter shmem_collect or shmem_fcollect.

API description

OpenSHMEM *collect* and *fcollect* routines concatenate *nelems* 64-bit or 32-bit data items from the *source* array into the *dest* array, over the set of PEs defined by *PE_start*, *log2PE_stride*, and *PE_size*, in processor number order. The resultant *dest* array contains the contribution from PE *PE_start* first, then the contribution from PE *PE_start* + *PE_stride* second, and so on. The collected result is written to the *dest* array for all PEs in the active set.

The *fcollect* routines require that *nelems* be the same value in all participating PEs, while the *collect* routines allow *nelems* to vary from PE to PE.

As with all OpenSHMEM collective routines, each of these routines assumes that only PEs in the active set call the routine. If a PE not in the active set and calls this collective routine, the behavior is undefined.

The values of arguments *PE_start*, *logPE_stride*, and *PE_size* must be the same value on all PEs in the active set. The same *dest* and *source* arrays and the same *pSync* work array must be passed by all PEs in the active set.

Upon return from a collective routine, the following are true for the local PE: The *dest* array is updated and the *source* array may be safely reused. The values in the *pSync* array are restored to the original values.

Return Values

None.

Notes

All OpenSHMEM collective routines reset the values in *pSync* before they return, so a particular *pSync* buffer need only be initialized the first time it is used.

The user must ensure that the *pSync* array is not being updated on any PE in the active set while any of the PEs participate in processing of an OpenSHMEM collective routine. Be careful to avoid these situations: If the *pSync* array is initialized at run time, some type of synchronization is needed to ensure that all PEs in the working set have initialized *pSync* before any of them enter an OpenSHMEM routine called with the *pSync* synchronization array. A *pSync* array can be reused on a subsequent OpenSHMEM collective routine only if none of the PEs in the active set are still processing a prior OpenSHMEM collective routine call that used the same *pSync* array. In general, this may be ensured only by doing some type of synchronization.

The collective routines operate on active PE sets that have a non-power-of-two *PE_size* with some performance degradation. They operate with no performance degradation when *nelems* is a non-power-of-two value.

EXAMPLES

The following *shmem_collect* example is for *C/C*++ programs:

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

int main(void)
{
   static long lock = 0;
   static long pSync[SHMEM_COLLECT_SYNC_SIZE];
   for (int i = 0; i < SHMEM_COLLECT_SYNC_SIZE; i++)
        pSync[i] = SHMEM_SYNC_VALUE;
   shmem_init();
   int me = shmem_my_pe();
</pre>
```

```
int npes = shmem_n_pes();
               int my_nelem = me + 1; /* linearly increasing number of elements with PE */
2
              int total_nelem = (npes * (npes + 1)) / 2;
               int * source = (int *) shmem_malloc(npes*sizeof(int)); /* symmetric alloc */
               int* dest = (int*) shmem_malloc(total_nelem*sizeof(int));
               for (int i = 0; i < my_nelem; i++)</pre>
                  source[i] = (me * (me + 1)) / 2 + i;
               for (int i = 0; i < total_nelem; i++)</pre>
                  dest[i] = -9999;
9
               shmem_barrier_all(); /* Wait for all PEs to update source/dest */
10
               shmem_collect32(dest, source, my_nelem, 0, 0, npes, pSync);
11
12
               shmem_set_lock(&lock); /* Lock prevents interleaving printfs
13
              printf("%d: %d", me, dest[0]);
               for (int i = 1; i < total_nelem; i++)</pre>
14
                  printf(", %d", dest[i]);
15
              printf("\n");
               shmem_clear_lock(&lock);
16
               shmem_finalize();
17
               return 0;
18
19
           The following SHMEM_COLLECT example is for Fortran programs:
20
           INCLUDE "shmem.fh"
21
22
           INTEGER PSYNC (SHMEM COLLECT SYNC SIZE)
           DATA PSYNC /SHMEM_COLLECT_SYNC_SIZE*SHMEM_SYNC_VALUE/
23
24
           CALL SHMEM_COLLECT4 (DEST, SOURCE, 64, PE_START, LOGPE_STRIDE,
25
           & PE_SIZE, PSYNC)
26
27
     9.9.8 SHMEM_TEAM_COLLECT, SHMEM_TEAM_FCOLLECT
28
29
      Concatenates blocks of data from multiple PEs in a team to an array in every PE in the team.
30
31
      SYNOPSIS
32
33
           C/C++:
           void shmem_team_collect32(shmem_team_t team, void *dest, const void *source, size_t nelems);
34
           void shmem_team_collect64(shmem_team_t team, void *dest, const void *source, size_t nelems);
35
           void shmem_team_fcollect32(shmem_team_t team, void *dest, const void *source, size_t nelems);
36
           void shmem_team_fcollect64(shmem_team_t team, void *dest, const void *source, size_t nelems);
37
38
39
      DESCRIPTION
40
41
           Arguments
42
                 IN
                                                 A valid OpenSHMEM team handle to a team which has been created
43
                                 team
                                                 without disabling support for collective operations.
44
                 OUT
                                 dest
                                                 A symmetric array large enough to accept the concatenation of the
45
                                                 source arrays on all PEs in the team. See table below in this description
46
                                                 for allowable data types.
47
                  IN
                                 source
                                                 A symmetric data object that can be of any type permissible for the dest
48
                                                 argument.
```

API description

OpenSHMEM *team_collect* and *team_fcollect* are collective routines over an existing team. These routines concatenate *nelems* 64-bit or 32-bit data items from the *source* array into the *dest* array, over all PEs in the specified *team* in processor number order. The resultant *dest* array contains the contribution from the first PE in the *team*, then the contribution from the second PE in the *team*, and so on. The collected result is written to the *dest* array for all PEs in the team.

The *fcollect* routines require that all PEs in the team provide the same value for *nelems*, while the *collect* routines allow *nelems* to vary from PE to PE.

If the team has been created with the *SHMEM_TEAM_NOCOLLECTIVE* option, it will not have the required support structures to complete this routine. If such a team is passed to this or any other team collective routine, the behavior is undefined.

The same dest and source data objects must be passed by all PEs in the team.

Upon return from a collective routine, the following are true for the local PE:

- The *dest* array is updated.
- The *source* array may be safely reused.

Error checking will be done to detect a value of *SHMEM_TEAM_NULL* passed for the team argument. In that case, the program will abort with an informative error message. If an invalid team handle is passed to the routine, the behavior is undefined.

The *dest* and *source* data objects must conform to certain typing constraints, which are as follows:

Routine	Data type of <i>dest</i> and <i>source</i>
shmem_team_collect64,	Any noncharacter type that has an element size of 64 bits.
shmem_team_fcollect64	C/C++ structures are NOT allowed.
shmem_team_collect32,	Any noncharacter type that has an element size of 32 bits.
shmem_team_fcollect32	C/C++ structures are NOT allowed.

Return Values

None.

Notes

All OpenSHMEM team collective routines use symmetric data structures associated with the team to synchronize and share data. By default, new teams that result from split operations will have these structures.

EXAMPLES

9.9.9 SHMEM_REDUCTIONS

The following functions perform reduction operations across all PEs in a set of PEs.

SYNOPSIS

<pre>roid shmem_short_and_to_all(short *c</pre>	lest, const short *source, int nreduce, int PE_s
logPE_stride, int PE_size, short	: *pWrk, long *pSync);
logDE strids int DE size int	const int *source, int nreduce, int PE_start,
roid shown long and to all (long the	(pwik, long *psylic);
logPE stride int PE size long	trikk long the source, int frequee, int PE_star
roid shmem longlong and to all (long	*pwik, long *psylic),
PE start int logPE stride int	PE size long long +pWrk long +pSync):
TE_Start, Inc Togre_Stride, Inc	TE_SIZE, IONG IONG APWIK, IONG APSync/,
- deprecation start	
FORTRAN:	
CALL SHMEM_INT4_AND_TO_ALL(dest, sou	<pre>irce, nreduce, PE_start, logPE_stride, PE_size,</pre>
pSync)	
CALL SHMEM_INT8_AND_TO_ALL(dest, sou	<pre>irce, nreduce, PE_start, logPE_stride, PE_size,</pre>
pSync)	
	deprec
).9.9.2 MAX Performs a maximum-va	lue reduction across a set of PEs.
C/C++:	
<pre>roid shmem_short_max_to_all(short *c</pre>	<pre>lest, const short *source, int nreduce, int PE_s</pre>
<pre>logPE_stride, int PE_size, short</pre>	: *pWrk, long *pSync);
<pre>roid shmem_int_max_to_all(int *dest,</pre>	<pre>const int *source, int nreduce, int PE_start,</pre>
logPE_stride, int PE_size, int ;	<pre>wpWrk, long *pSync);</pre>
<pre>roid shmem_double_max_to_all(double</pre>	*dest, const double *source, int nreduce, int H
<pre>int logPE_stride, int PE_size, c</pre>	<pre>iouble *pWrk, long *pSync);</pre>
<pre>roid shmem_float_max_to_all(float *c</pre>	<pre>lest, const float *source, int nreduce, int PE_s</pre>
logPE_stride, int PE_size, float	: *pWrk, long *pSync);
rold shmem_long_max_to_all(long *des	t, Const long *source, int nreduce, int PE_star
rogPE_stride, int PE_size, long	*pwrk, long *psync);
int PE start int logPE stride	int PE size long double +pWrk long +pSunc):
void shmem longlong max to all (long	long *dest, const long long *source, int preduc
PE start, int logPE stride, int	PE size. long long *pWrk. long *pSync):
- deprecation start	
FORTRAN:	
CALL SHMEM_INT4_MAX_TO_ALL(dest, sou	<pre>irce, nreduce, PE_start, logPE_stride, PE_size,</pre>
pSync)	
CALL SHMEM_INT8_MAX_TO_ALL(dest, sou	<pre>irce, nreduce, PE_start, logPE_stride, PE_size,</pre>
pSync)	
CALL SHMEM_REAL4_MAX_TO_ALL(dest, sc	ource, nreduce, PE_start, logPE_stride, PE_size,
pSync)	
CALL SHMEM_REAL8_MAX_TO_ALL(dest, sc	ource, nreduce, PE_start, logPE_stride, PE_size,
pSync)	
CALL SHMEM_REAL16 MAX TO ALL(dest. s	source, nreduce, PE start, logPE stride. PE size
pSvnc)	
P01.00)	
	deprec
) 0 0 3 MIN Performs a minimum valu	le reduction across a set of PEs
).9.9.3 MIN Performs a minimum-valu	e reduction across a set of PEs.

<pre>void shmem_int_min_to_all(int *dest, const int *source, int nreduce, int PE_start, int</pre>
logPE_stride, int PE_size, int *pWrk, long *pSync);
<pre>void shmem_double_min_to_all(double *dest, const double *source, int nreduce, int PE_start,</pre>
<pre>int logPE_stride, int PE_size, double *pWrk, long *pSync);</pre>
<pre>void shmem_float_min_to_all(float *dest, const float *source, int nreduce, int PE_start, int</pre>
logPE_stride, int PE_size, float *pWrk, long *pSync);
<pre>void shmem_long_min_to_all(long *dest, const long *source, int nreduce, int PE_start, int</pre>
logPE_stride, int PE_size, long *pWrk, long *pSync);
<pre>void shmem_longdouble_min_to_all(long double *dest, const long double *source, int nreduce,</pre>
<pre>int PE_start, int logPE_stride, int PE_size, long double *pWrk, long *pSync);</pre>
<pre>void shmem_longlong_min_to_all(long long *dest, const long long *source, int nreduce, int</pre>
<pre>PE_start, int logPE_stride, int PE_size, long long *pWrk, long *pSync);</pre>
- deprecation start
FORTRAN:
CALL SHMEM_INT4_MIN_T0_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
CALL SHMEM INT8 MIN TO ALL(dest, source, nreduce, PE start, logPE stride, PE size, pWrk,
pSvnc)
(ALL SHMEM REALA MIN TO ALL (dest source preduce PE start logPE stride PE size pWrk
prune)
(NIT CUMEM DEALS MIN TO ALL doct courses produce DE stort logDE stride DE size purk
CALL SHMEM_REALS_MIN_IO_ALL(dest, Source, Hreduce, Pr_start, TogPr_stride, Pr_size, pwrk,
CALL SHMEM_REALI6_MIN_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
deprecation end -
0.0.0.4 SUM Derforms a sum reduction serves a set of DEs
CICLLA
UC++:
void snmem_complexd_sum_to_all(double_complex *dest, const double_complex *source, int
nreduce, int PE_start, int logPE_stride, int PE_size, double _Complex *pwrk, long
*psync);
void snmem_complexi_sum_to_all(iloat _Complex *dest, const iloat _Complex *source, int
<pre>nreduce, int PE_start, int logPE_stride, int PE_size, float _Complex *pWrk, long </pre>
*psync);
void snmem_short_sum_to_all(short *dest, const short *source, int nreduce, int PE_start, int
logPE_stride, int PE_size, short *pWrk, long *pSync);

- void shmem_int_sum_to_all(int *dest, const int *source, int nreduce, int PE_start, int logPE_stride, int PE_size, int *pWrk, long *pSync);
- void shmem_float_sum_to_all(float *dest, const float *source, int nreduce, int PE_start, int logPE_stride, int PE_size, float *pWrk, long *pSync);
- void shmem_long_sum_to_all(long *dest, const long *source, int nreduce, int PE_start, int logPE_stride,int PE_size, long *pWrk, long *pSync);
- void shmem_longdouble_sum_to_all(long double *dest, const long double *source, int nreduce, int PE_start, int logPE_stride, int PE_size, long double *pWrk, long *pSync);
- void shmem_longlong_sum_to_all(long long *dest, const long long *source, int nreduce, int
 PE_start, int logPE_stride, int PE_size, long long *pWrk, long *pSync);

- deprecation start -

FORTRAN:

- CALL SHMEM_COMP4_SUM_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, pSync)
- CALL SHMEM_COMP8_SUM_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, pSync)

CALL SHMEM_INT4_SUM_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
CALL SHMEM_INT8_SUM_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
CALL SHMEM_REAL4_SUM_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
CALL SHMEM_REAL8_SUM_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
CALL SHMEM_REAL16_SUM_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
deprecation end —
9.9.9.5 PROD Performs a product reduction across a set of PEs.
void snmem_complexd_prod_to_all(double _complex *dest, const double _complex *source, int
nreduce, int PE_start, int logPE_stride, int PE_size, double _complex *pwrk, long
<pre>*psync), woid shmem complexing to all (float Complex +dest const float Complex +source int</pre>
nreduce. int PE start. int logPE stride. int PE size. float. Complex *DWrk. long
*pSync):
void shmem short prod to all(short *dest, const short *source, int nreduce, int PE start,
<pre>int logPE_stride, int PE_size, short *pWrk, long *pSync);</pre>
<pre>void shmem_int_prod_to_all(int *dest, const int *source, int nreduce, int PE_start, int</pre>
logPE_stride, int PE_size, int *pWrk, long *pSync);
<pre>void shmem_double_prod_to_all(double *dest, const double *source, int nreduce, int PE_start,</pre>
<pre>int logPE_stride, int PE_size, double *pWrk, long *pSync);</pre>
<pre>void shmem_float_prod_to_all(float *dest, const float *source, int nreduce, int PE_start,</pre>
<pre>int logPE_stride, int PE_size, float *pWrk, long *pSync);</pre>
<pre>void shmem_long_prod_to_all(long *dest, const long *source, int nreduce, int PE_start, int</pre>
<pre>logPE_stride, int PE_size, long *pWrk, long *pSync);</pre>
void shmem_longdouble_prod_to_all(long double *dest, const long double *source, int nreduce,
<pre>int PE_start, int logPE_stride, int PE_size, iong double *pwrk, iong *psync); reid show logplane word to all (long long ideat area long long ist and use int</pre>
Void shmem_longlong_prod_to_all(long long *dest, const long long *source, int hreduce, int
TE_state, inc togre_stilde, inc te_size, iong iong spwik, iong spsyne),
- deprecation start
FORTRAN:
CALL SHMEM_COMP4_PROD_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
CALL SHMEM_COMP8_PROD_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
CALL SHMEM_INT4_PROD_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)
CALL SHMEM_INT8_PROD_TO_ALL(dest, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSvnc)
CALL SHMEM REAL4 PROD TO ALL(dest, source, preduce, PE start, logPE stride, PE size, pWrk
pSync)
CALL SHMEM REALS PROD TO ALL (dest. source preduce PF start logPF stride PF size nurk
pSync)
CALL SHMEM REALLS PROD TO ALL (dest source produce DE start logDE stride DE size rwyk
Division of the second source, meduce, re_state, togre_stitue, re_stree, pwik,
bolue)

— deprecation end —

void shmem shor	t or to all (ch	ort +dest. const short +source, int preduce int PF start int
logPE strid	le. int PE size	. short *pWrk. long *pSync):
void shmem int	or to all(int ,	*dest, const int *source, int nreduce, int PE start, int
logPE_strid	le, int PE_size	<pre>, int *pWrk, long *pSync);</pre>
void shmem_long	_or_to_all(lon	g *dest, const long *source, int nreduce, int PE_start, int
logPE_strid	le, int PE_size	<pre>, long *pWrk, long *pSync);</pre>
void shmem_long	long_or_to_all	(long long *dest, const long long *source, int nreduce, int
PE_start, i	.nt logPE_stride	e, int PE_size, long long *pWrk, long *pSync);
— deprecation sta	ırt	A
FORTRAN:		
CALL SHMEM_INT4	_OR_TO_ALL (dest	t, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)		
CALL SHMEM_INT8	_OR_TO_ALL (dest	t, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,
pSync)		
		deprecation end —
9.9.9.7 XOR H	Performs a bitwise	e exclusive OR (XOR) reduction across a set of PEs.
()/(·++•		
0/0111		
void shmem_shor	t_xor_to_all(s	<pre>hort *dest, const short *source, int nreduce, int PE_start, int</pre>
void shmem_shor logPE_strid	t_xor_to_all(s le, int PE_size	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync);</pre>
<pre>void shmem_shor logPE_strid void shmem_int_;</pre>	t_xor_to_all(s de, int PE_size, xor_to_all(int	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size,	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync);</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long.</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int</pre>
<pre>void shmem_shor logPE_strid void shmem_int_ logPE_strid void shmem_long logPE_strid</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size,	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync);</pre>
<pre>void shmem_shor logPE_strid void shmem_int_ logPE_strid void shmem_long logPE_strid void shmem_long</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size, long_xor_to_all	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long. logPE_strid void shmem_long PE_start, i</pre>	t_xor_to_all(sh de, int PE_size, xor_to_all(int de, int PE_size, _xor_to_all(lor de, int PE_size, long_xor_to_all .nt logPE_stride	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync);</pre>
<pre>void shmem_shor logPE_strid void shmem_int_ logPE_strid void shmem_long logPE_strid void shmem_long PE_start, i deprecation sta</pre>	t_xor_to_all(sh de, int PE_size, xor_to_all(int de, int PE_size, _xor_to_all(lor de, int PE_size, long_xor_to_all .nt logPE_stride	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync);</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long logPE_strid void shmem_long PE_start, i — deprecation sta FORTRAN:</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size, long_xor_to_all .nt logPE_stride	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync);</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long logPE_strid void shmem_long PE_start, i deprecation sta FORTRAN: CALL SHMEM_INT4</pre>	t_xor_to_all(sh he, int PE_size, xor_to_all(int he, int PE_size, _xor_to_all(lor he, int PE_size, long_xor_to_all int logPE_stride urt _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long logPE_strid void shmem_long PE_start, i deprecation sta FORTRAN: CALL SHMEM_INT4 pSync)</pre>	t_xor_to_all(sh de, int PE_size, xor_to_all(int de, int PE_size, _xor_to_all(lor de, int PE_size, long_xor_to_all .nt logPE_stride urt _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long logPE_strid void shmem_long PE_start, i — deprecation sta FORTRAN: CALL SHMEM_INT4 pSync) CALL SHMEM_INT8</pre>	t_xor_to_all(sh de, int PE_size, xor_to_all(int de, int PE_size, _xor_to_all(lor de, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,</pre>
<pre>void shmem_shor logPE_strid void shmem_int_ logPE_strid void shmem_long logPE_strid void shmem_long PE_start, i deprecation sta FORTRAN: CALL SHMEM_INT4 pSync) CALL SHMEM_INT8 pSync)</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,</pre>
<pre>void shmem_shor logPE_strid void shmem_int_ logPE_strid void shmem_long PE_start, i deprecation sta FORTRAN: CALL SHMEM_INT4 pSync) CALL SHMEM_INT8, pSync)</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long PE_start, i — deprecation sta FORTRAN: CALL SHMEM_INT4, pSync) CALL SHMEM_INT8, pSync)</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long. logPE_strid void shmem_long PE_start, i deprecation sta FORTRAN: CALL SHMEM_INT4. pSync) CALL SHMEM_INT8. pSync)</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long PE_start, i - deprecation sta FORTRAN: CALL SHMEM_INT4 pSync) CALL SHMEM_INT8 pSync)</pre>	t_xor_to_all(sh de, int PE_size, xor_to_all(int de, int PE_size, _xor_to_all(lor de, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk,</pre>
<pre>void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long DogPE_strid void shmem_long PE_start, i - deprecation sta FORTRAN: CALL SHMEM_INT4: pSync) CALL SHMEM_INT8: pSync) CALL SHMEM_INT8: pSync)</pre>	t_xor_to_all(sh de, int PE_size, xor_to_all(int de, int PE_size, _xor_to_all(lor de, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, deprecation end —</pre>
<pre>void shmem_shor logPE_strid void shmem_int_ logPE_strid void shmem_long PE_start, i deprecation sta FORTRAN: CALL SHMEM_INT4 pSync) CALL SHMEM_INT8. pSync)</pre>	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, deprecation end —</pre>
void shmem_shor logPE_strid void shmem_int_ logPE_strid void shmem_long PE_start, i — deprecation sta FORTRAN: CALL SHMEM_INT4 pSync) CALL SHMEM_INT8 pSync) CALL SHMEM_INT8 pSync)	t_xor_to_all(sh le, int PE_size, xor_to_all(int le, int PE_size, _xor_to_all(lor le, int PE_size, long_xor_to_all int logPE_stridd _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, deprecation end</pre>
void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long PE_start, i - deprecation sta FORTRAN: CALL SHMEM_INT4 pSync) CALL SHMEM_INT8 pSync) CALL SHMEM_INT8 pSync)	t_xor_to_all(st de, int PE_size, xor_to_all(int de, int PE_size, _xor_to_all(lor de, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, deprecation end</pre>
void shmem_shor logPE_strid void shmem_int_: logPE_strid void shmem_long PE_start, i — deprecation sta FORTRAN: CALL SHMEM_INT4 pSync) CALL SHMEM_INT8 pSync) CALL SHMEM_INT8 pSync) CALL SHMEM_INT8 pSync)	t_xor_to_all(st de, int PE_size, xor_to_all(int de, int PE_size, _xor_to_all(lor de, int PE_size, long_xor_to_all .nt logPE_stride _XOR_TO_ALL(des _XOR_TO_ALL(des	<pre>hort *dest, const short *source, int nreduce, int PE_start, int , short *pWrk, long *pSync); *dest, const int *source, int nreduce, int PE_start, int , int *pWrk, long *pSync); ng *dest, const long *source, int nreduce, int PE_start, int , long *pWrk, long *pSync); l(long long *dest, const long long *source, int nreduce, int e, int PE_size, long long *pWrk, long *pSync); st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, st, source, nreduce, PE_start, logPE_stride, PE_size, pWrk, deprecation end</pre>

A symmetric array, of length *nreduce* elements, that contains one ele-IN source ment for each separate reduction routine. The source argument must have the same data type as *dest*. IN nreduce The number of elements in the dest and source arrays. nreduce must be of type integer. When using Fortran, it must be a default integer value. IN The lowest PE number of the active set of PEs. PE_start must be of PE_start type integer. When using Fortran, it must be a default integer value.

1 2	IN	logPE_stride	The log (base 2) of the stride between consecutive PE numbers in the active set. <i>logPE_stride</i> must be of type integer. When using <i>Fortran</i> , it must be a default integer value
3	IN	PE_size	The number of PEs in the active set. <i>PE_size</i> must be of type integer. When using <i>Fortran</i> , it must be a default integer value.
5 6	IN	pWrk	A symmetric work array of size at least max(<i>nreduce</i> /2 + 1, <i>SHMEM_REDUCE_MIN_WRKDATA_SIZE</i>) elements.
7 8 9	IN	pSync	A symmetric work array of size <i>SHMEM_REDUCE_SYNC_SIZE</i> . In <i>C/C++</i> , <i>pSync</i> must be an array of elements of type <i>long</i> . In
10 11			type. Every element of this array must be initialized with the value SHMEM_SYNC_VALUE before any of the PEs in the active set enter
12 13			the reduction routine.
14	API description		
16	OpenSHMEM PEs. A reduc	M reduction routines ation performs an asso	compute one or more reductions across symmetric arrays on multiple ociative binary routine across a set of values.
19 20	The <i>nreduce</i> PEs in the ac the <i>dest</i> array	argument determines tive set provides one on all PEs in the act	s the number of separate reductions to perform. The <i>source</i> array on all element for each reduction. The results of the reductions are placed in ive set. The active set is defined by the <i>PE_start</i> , <i>logPE_stride</i> , <i>PE_size</i>
22	The <i>source</i> and	nd <i>dest</i> arrays may be	e the same array, but they may not be overlapping arrays.
23 24 25	As with all C set call the ro undefined.	DenSHMEM collect Dutine. If a PE not in	tive routines, each of these routines assumes that only PEs in the active the active set calls an OpenSHMEM collective routine, the behavior is
26 27 28	The values o active set. Th all PEs in the	f arguments <i>nreduce</i> the same <i>dest</i> and <i>source</i> active set.	, <i>PE_start</i> , <i>logPE_stride</i> , and <i>PE_size</i> must be equal on all PEs in the <i>rce</i> arrays, and the same <i>pWrk</i> and <i>pSync</i> work arrays, must be passed to
29	Before any P	E calls a reduction ro	outine, the following conditions must be ensured:
30 31	• The <i>pWr</i> OpenSH	k and <i>pSync</i> arrays of MEM routine.	n all PEs in the active set are not still in use from a prior call to a collective
32 33	• The <i>dest</i> Otherwise, th	array on all PEs in t be behavior is undefir	he active set is ready to accept the results of the <i>reduction</i> .
34 35	Upon return f the <i>source</i> an	from a reduction rout ray may be safely reu	ine, the following are true for the local PE: The <i>dest</i> array is updated and used. The values in the <i>pSync</i> array are restored to the original values.
36 37 38	The complex environment support for th	-typed interfaces are does not support com nese complex-typed i	only provided for sum and product reductions. When the C translation pplex types ⁷ , an OpenSHMEM implementation is not required to provide nterfaces.
39 40			
41 42	When calling	trom <i>Fortran</i> , the <i>de</i>	est date types are as follows:
43	Koutine		Data type
44 45 46	shmem_int shmem_int shmem_cou	8_and_to_all 4_and_to_all np8_max_to_all 4_max_to_all	Integer, with an element size of 8 bytes. Integer, with an element size of 4 bytes. Complex, with an element size equal to two 8-byte real values. Integer, with an element size of 4 bytes
48			

⁷That is, under C language standards prior to C99 or under C11 when <u>STDC_NO_COMPLEX</u> is defined to 1

shmem_int8_max_to_all shmem_real4_max_to_all shmem_real16_max_to_all shmem_int4_min_to_all shmem_int8_min_to_all shmem_real4_min_to_all shmem_real8_min_to_all shmem_real16_min_to_all shmem_comp4_sum_to_all shmem_comp8_sum_to_all shmem_int4_sum_to_all shmem_int8_sum_to_all shmem real4 sum to all shmem_real8_sum_to_all shmem real16 sum to all shmem_comp4_prod_to_all shmem_comp8_prod_to_all shmem_int4_prod_to_all shmem_int8_prod_to_all shmem_real4_prod_to_all shmem_real8_prod_to_all shmem_real16_prod_to_all shmem_int8_or_to_all shmem_int4_or_to_all shmem_int8_xor_to_all shmem_int4_xor_to_all

Integer, with an element size of 8 bytes.
Real, with an element size of 4 bytes.
Real, with an element size of 16 bytes.
Integer, with an element size of 4 bytes.
Integer, with an element size of 8 bytes.
Real, with an element size of 4 bytes.
Real, with an element size of 8 bytes.
Real, with an element size of 16 bytes.
Complex, with an element size equal to two 4-byte real values.
Complex, with an element size equal to two 8-byte real values.
Integer, with an element size of 4 bytes.
Integer, with an element size of 8 bytes
Real, with an element size of 4 bytes.
Real, with an element size of 8 bytes.
Real, with an element size of 16 bytes.
Complex, with an element size equal to two 4-byte real values.
Complex, with an element size equal to two 8-byte real values.
Integer, with an element size of 4 bytes.
Integer, with an element size of 8 bytes.
Real, with an element size of 4 bytes.
Real, with an element size of 8 bytes.
Real, with an element size of 16 bytes.
Integer, with an element size of 8 bytes.
Integer, with an element size of 4 bytes.
Integer, with an element size of 8 bytes.
Integer, with an element size of 4 bytes.

Return Values

None.

Notes

All OpenSHMEM reduction routines reset the values in pSync before they return, so a particular pSync buffer need only be initialized the first time it is used. The user must ensure that the pSync array is not being updated on any PE in the active set while any of the PEs participate in processing of an OpenSHMEM reduction routine. Be careful to avoid the following situations: If the pSync array is initialized at run time, some type of synchronization is needed to ensure that all PEs in the working set have initialized pSync before any of them enter an OpenSHMEM routine called with the pSync synchronization array. A pSync or pWrk array can be reused in a subsequent reduction routine call only if none of the PEs in the active set are still processing a prior reduction routine call that used the same pSync or pWrk arrays. In general, this can be assured only by doing some type of synchronization.

EXAMPLES

This *Fortran* reduction example statically initializes the *pSync* array and finds the logical *AND* of the integer variable *FOO* across all even PEs.

INCLUDE "shmem.fh"

```
INTEGER PSYNC(SHMEM_REDUCE_SYNC_SIZE)
DATA PSYNC /SHMEM_REDUCE_SYNC_SIZE*SHMEM_SYNC_VALUE/
PARAMETER (NR=1)
INTEGER*4 PWRK(MAX(NR/2+1,SHMEM_REDUCE_MIN_WRKDATA_SIZE))
INTEGER FOO, FOOAND
SAVE FOO, FOOAND, PWRK
```

2

3

6

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24 25

26

27 28

29

30

31

32

33

34

35

36

37 38 39

40 41

42

43

44

45

46

47

```
1
           INTRINSIC SHMEM_MY_PE()
2
           FOO = SHMEM_MY_PE()
           IF ( MOD(SHMEM_MY_PE() .EQ. 0) THEN
                IF ( MOD (SHMEM_N_PES()(),2) .EQ. 0) THEN
                   CALL SHMEM_INT8_AND_TO_ALL(FOOAND, FOO, NR, 0, 1, NPES/2, &
               PWRK, PSYNC)
                ELSE
                   CALL SHMEM_INT8_AND_TO_ALL(FOOAND, FOO, NR, 0, 1, NPES/2+1, &
               PWRK, PSYNC)
                ENDIF
9
                PRINT*, 'Result on PE ', SHMEM_MY_PE(), ' is ', FOOAND
10
           ENDIF
11
           This Fortran example statically initializes the pSync array and finds the maximum value of real variable FOO
12
           across all even PEs.
13
           INCLUDE "shmem.fh"
14
15
           INTEGER PSYNC (SHMEM_REDUCE_SYNC_SIZE)
           DATA PSYNC /SHMEM_REDUCE_SYNC_SIZE*SHMEM_SYNC_VALUE/
16
           PARAMETER (NR=1)
17
           REAL FOO, FOOMAX, PWRK (MAX (NR/2+1, SHMEM_REDUCE_MIN_WRKDATA_SIZE) )
18
           COMMON /COM/ FOO, FOOMAX, PWRK
           INTRINSIC SHMEM_MY_PE()
19
20
           IF ( MOD (SHMEM_MY_PE() .EQ. 0) THEN
                   CALL SHMEM_REAL8_MAX_TO_ALL (FOOMAX, FOO, NR, 0, 1, NSPES/2,
21
              PWRK, PSYNC)
            8
22
                   PRINT*, 'Result on PE ', SHMEM_MY_PE(), ' is ', FOOMAX
           ENDIF
23
24
           This Fortran example statically initializes the pSync array and finds the minimum value of real variable FOO
25
           across all the even PEs.
26
           INCLUDE "shmem.fh"
27
           INTEGER PSYNC (SHMEM_REDUCE_SYNC_SIZE)
28
           DATA PSYNC /SHMEM_REDUCE_SYNC_SIZE*SHMEM_SYNC_VALUE/
29
           PARAMETER (NR=1)
30
           REAL FOO, FOOMIN, PWRK(MAX(NR/2+1, SHMEM_REDUCE_MIN_WRKDATA_SIZE))
           COMMON /COM/ FOO, FOOMIN, PWRK
31
           INTRINSIC SHMEM_MY_PE()
32
           IF ( MOD (SHMEM_MY_PE() .EQ. 0) THEN
33
                   CALL SHMEM_REAL8_MIN_TO_ALL (FOOMIN, FOO, NR, 0, 1, N$PES/2,
34
               PWRK, PSYNC)
                   PRINT*, 'Result on PE ', SHMEM_MY_PE(), ' is ', FOOMIN
35
           ENDIF
36
37
           This Fortran example statically initializes the pSync array and finds the sum of the real variable FOO across all
38
           even PEs.
39
           INCLUDE "shmem.fh"
40
           INTEGER PSYNC (SHMEM_REDUCE_SYNC_SIZE)
41
           DATA PSYNC /SHMEM_REDUCE_SYNC_SIZE*SHMEM_SYNC_VALUE/
42
           PARAMETER (NR=1)
           REAL FOO, FOOSUM, PWRK (MAX (NR/2+1, SHMEM_REDUCE_MIN_WRKDATA_SIZE))
43
           COMMON /COM/ FOO, FOOSUM, PWRK
44
           INTRINSIC SHMEM_MY_PE()
45
           IF ( MOD (SHMEM_MY_PE() .EQ. 0) THEN
46
                   CALL SHMEM_INT4_SUM_TO_ALL(FOOSUM, FOO, NR, 0, 1, N$PES/2,
47
               PWRK, PSYNC)
            8
                   PRINT*, 'Result on PE ', SHMEM_MY_PE(), ' is ', FOOSUM
48
           ENDIF
```

This *Fortran* example statically initializes the *pSync* array and finds the *product* of the real variable *FOO* across all the even PEs.

```
INCLUDE "shmem.fh"
     INTEGER PSYNC (SHMEM REDUCE SYNC SIZE)
     DATA PSYNC /SHMEM_REDUCE_SYNC_SIZE*SHMEM_SYNC_VALUE/
     PARAMETER (NR=1)
     REAL FOO, FOOPROD, PWRK (MAX (NR/2+1, SHMEM_REDUCE_MIN_WRKDATA_SIZE))
     COMMON /COM/ FOO, FOOPROD, PWRK
     INTRINSIC SHMEM_MY_PE()
     IF ( MOD (SHMEM_MY_PE() .EQ. 0) THEN
              CALL SHMEM_COMP8_PROD_TO_ALL(FOOPROD, FOO, NR, 0, 1, N$PES/2,
     &
        PWRK, PSYNC)
              PRINT*, 'Result on PE ', SHMEM_MY_PE(),' is ', FOOPROD
     ENDIF
     This Fortran example statically initializes the pSync array and finds the logical OR of the integer variable FOO
     across all even PEs.
     INCLUDE "shmem.fh"
     INTEGER PSYNC(SHMEM_REDUCE_SYNC_SIZE)
     DATA PSYNC /SHMEM_REDUCE_SYNC_SIZE * SHMEM_SYNC_VALUE /
     PARAMETER (NR=1)
     REAL PWRK (MAX (NR/2+1, SHMEM_REDUCE_MIN_WRKDATA_SIZE))
     INTEGER FOO, FOOOR
     COMMON /COM/ FOO, FOOOR, PWRK
     INTRINSIC SHMEM_MY_PE()
     IF ( MOD (SHMEM_MY_PE() .EQ. 0) THEN
              CALL SHMEM_INT8_OR_TO_ALL (FOOOR, FOO, NR, 0, 1, N$PES/2,
        PWRK, PSYNC)
     &
              PRINT*, 'Result on PE ', SHMEM_MY_PE(), ' is ', FOOOR
     ENDIF
     This Fortran example statically initializes the pSync array and computes the exclusive XOR of variable FOO
     across all even PEs.
     INCLUDE "shmem.fh"
     INTEGER PSYNC (SHMEM_REDUCE_SYNC_SIZE)
     DATA PSYNC /SHMEM_REDUCE_SYNC_SIZE*SHMEM_SYNC_VALUE/
     PARAMETER (NR=1)
     REAL FOO, FOOXOR, PWRK (MAX (NR/2+1, SHMEM_REDUCE_MIN_WRKDATA_SIZE))
     COMMON /COM/ FOO, FOOXOR, PWRK
     INTRINSIC SHMEM_MY_PE()
     IF ( MOD (SHMEM_MY_PE() .EQ. 0) THEN
             CALL SHMEM_REAL8_XOR_TO_ALL(FOOXOR, FOO, NR, 0, 1, N$PES/2,
     &
        PWRK, PSYNC)
             PRINT*, 'Result on PE ', SHMEM_MY_PE(), ' is ', FOOXOR
     ENDIF
9.9.10 SHMEM ALLTOALL
shmem_alltoall is a collective routine where each PE exchanges a fixed amount of data with all other PEs in the active
set.
SYNOPSIS
```

```
void shmem_alltoall32(void *dest, const void *source, size_t nelems, int PE_start, int
logPE_stride, int PE_size, long *pSync);
```

10

11

12

13

14

15

16 17

18

19

20

21

22

23

24

25

26 27

28 29

30

31

32

33

34

35

36

37

38

39 40

41 42

43

44

45 46

47

void shmem_alltoa	ill64 (void *dest,	<pre>const void *source, size_t nelems, int PE_start, int prg +pSync):</pre>			
demonster start,	depresention start				
FORTRAN:	— deprecation start — FORTRAN: INTEGER pSync(SHMEM ALLTOALL SYNC SIZE)				
INTEGER pSync(SHM					
INTEGER PE_start,	logPE_stride, F	PE_size, nelems			
CALL SHMEM_ALLTOA	LL32(dest, sourc	ce, nelems, PE_start, logPE_stride, PE_size, pSync)			
CALL SHMEM_ALLTOA	LL64(dest, sourc	ce, nelems, PE_start, logPE_stride, PE_size, pSync)			
		deprecation end —			
ESCRIPTION					
Arguments					
OUT	dest	A symmetric data object large enough to receive the combined total of <i>nelems</i> elements from each PE in the active set.			
IN	source	A symmetric data object that contains <i>nelems</i> elements of data for each PE in the active set, ordered according to destination PE.			
IN	nelems	The number of elements to exchange for each PE. <i>nelems</i> must be of type size_t for $C/C++$. When using <i>Fortran</i> , it must be a default integer value.			
IN	PE_start	The lowest PE number of the active set of PEs. <i>PE_start</i> must be o type integer. When using <i>Fortran</i> , it must be a default integer value.			
IN	logPE_stride	The log (base 2) of the stride between consecutive PE numbers in the active set. <i>logPE_stride</i> must be of type integer. When using <i>Fortran</i> it must be a default integer value.			
IN	PE_size	The number of PEs in the active set. <i>PE_size</i> must be of type integer When using <i>Fortran</i> , it must be a default integer value.			
IN	pSync	A symmetric work array of size SHMEM_ALLTOALL_SYNC_SIZE			
		In C/C++, pSync must be an array of elements of type long. In			
		Fortran, pSync must be an array of elements of default intege			
		super synce value before any of the PEs in the active set enter			
		the routine.			
API description					

The shmem alltoall routines are collective routines. Each PE in the active set exchanges nelems data elements of size 32 bits (for shmem_alltoall32) or 64 bits (for shmem_alltoall64) with all other PEs in the set. The data being sent and received are stored in a contiguous symmetric data object. The total size of each PEs source object and dest object is nelems times the size of an element (32 bits or 64 bits) times PE_size. The source object contains PE_size blocks of data (the size of each block defined by nelems) and each block of data is sent to a different PE. Given a PE i that is the $k^{\text{th}}PE$ in the active set and a PE i that is the $l^{h}PE$ in the active set, PE *i* sends the l^{h} block of its *source* object to the k^{h} block of the *dest* object of PE j.

As with all OpenSHMEM collective routines, this routine assumes that only PEs in the active set call the routine. If a PE not in the active set calls an OpenSHMEM collective routine, the behavior is undefined.

The values of arguments *nelems*, *PE_start*, *logPE_stride*, and *PE_size* must be equal on all PEs in the active set. The same *dest* and *source* data objects, and the same *pSync* work array must be passed to all PEs in the active set.

Before any PE calls a *shmem_alltoall* routine, the following conditions must be ensured:

37

38

39

40

41

42

43

44

45

46

47
- The *pSync* array on all PEs in the active set is not still in use from a prior call to a *shmem_alltoall* routine.
- The dest data object on all PEs in the active set is ready to accept the shmem_alltoall data.

Otherwise, the behavior is undefined.

Upon return from a *shmem_alltoall* routine, the following is true for the local PE: Its *dest* symmetric data object is completely updated and the data has been copied out of the *source* data object. The values in the *pSync* array are restored to the original values.

The *dest* and *source* data objects must conform to certain typing constraints, which are as follows:

Routine	Data type of <i>dest</i> and <i>source</i>
shmem_alltoall64	64 bits aligned.
shmem_alltoall32	32 bits aligned.

Return Values

None.

Notes

This routine restores *pSync* to its original contents. Multiple calls to OpenSHMEM routines that use the same *pSync* array do not require that *pSync* be reinitialized after the first call. The user must ensure that the *pSync* array is not being updated by any PE in the active set while any of the PEs participates in processing of an OpenSHMEM *shmem_alltoall* routine. Be careful to avoid these situations: If the *pSync* array is initialized at run time, some type of synchronization is needed to ensure that all PEs in the active set have initialized *pSync* array may be reused on a subsequent OpenSHMEM *shmem_alltoall* routine only if none of the PEs in the active set are still processing a prior OpenSHMEM *shmem_alltoall* routine call that used the same *pSync* array. In general, this can be ensured only by doing some type of synchronization.

EXAMPLES

This example shows a *shmem_alltoall64* on two long elements among all PEs.

```
#include <stdio.h>
#include <inttypes.h>
#include <shmem.h>
int main (void)
   static long pSync[SHMEM_ALLTOALL_SYNC_SIZE];
  for (int i = 0; i < SHMEM_ALLTOALL_SYNC_SIZE; i++)</pre>
     pSync[i] = SHMEM_SYNC_VALUE;
  shmem_init();
  int me = shmem_my_pe();
  int npes = shmem_n_pes();
  const int count = 2;
  int64_t* dest = (int64_t*) shmem_malloc(count * npes * sizeof(int64_t));
   int64_t* source = (int64_t*) shmem_malloc(count * npes * sizeof(int64_t));
   /* assign source values */
  for (int pe = 0; pe < npes; pe++) {</pre>
     for (int i = 0; i < count; i++) {</pre>
```

```
source[(pe * count) + i] = me + pe;
      dest[(pe * count) + i] = 9999;
   }
}
/* wait for all PEs to update source/dest */
shmem_barrier_all();
/* alltoall on all PES */
shmem_alltoall64(dest, source, count, 0, 0, npes, pSync);
/* verify results */
for (int pe = 0; pe < npes; pe++) {</pre>
   for (int i = 0; i < count; i++) {</pre>
      if (dest[(pe * count) + i] != pe + me) {
         printf("[%d] ERROR: dest[%d]=%" PRId64 ", should be %d\n",
            me, (pe * count) + i, dest[(pe * count) + i], pe + me);
}
shmem_free(dest);
shmem_free(source);
shmem_finalize();
return 0;
```

9.9.11 SHMEM_ALLTOALLS

shmem_alltoalls is a collective routine where each PE exchanges a fixed amount of strided data with all other PEs in the active set.

```
SYNOPSIS
26
```

```
C/C++:
27
           void shmem_alltoalls32(void *dest, const void *source, ptrdiff_t dst, ptrdiff_t sst, size_t
28
               nelems, int PE_start, int logPE_stride, int PE_size, long *pSync);
29
           void shmem_alltoalls64(void *dest, const void *source, ptrdiff_t dst, ptrdiff_t sst, size_t
               nelems, int PE_start, int logPE_stride, int PE_size, long *pSync);
30
31

deprecation start —

           FORTRAN:
32
           INTEGER pSync (SHMEM_ALLTOALLS_SYNC_SIZE)
33
           INTEGER dst, sst, PE_start, logPE_stride, PE_size
34
           INTEGER nelems
35
           CALL SHMEM_ALLTOALLS32(dest, source, dst, sst, nelems, PE_start, logPE_stride, PE_size,
36
               pSync)
37
           CALL SHMEM_ALLTOALLS64(dest, source, dst, sst, nelems, PE_start, logPE_stride, PE_size,
38
               pSync)
39
                                                                                            deprecation end -
40
41
42
     DESCRIPTION
43
44
           Arguments
```

46	OUT	dest	A symmetric data object large enough to receive the combined total of
17			nelems elements from each PE in the active set.
40	IN	source	A symmetric data object that contains <i>nelems</i> elements of data for each
48			PE in the active set, ordered according to destination PE.

2

9

10

11

12 13

14 15

16

17

18 19 20

21 22

23

24

25

IN	dst	The stride between consecutive elements of the <i>dest</i> data object. The stride is scaled by the element size. A value of <i>1</i> indicates contiguous data. <i>dst</i> must be of type <i>ptrdiff_t</i> . When using <i>Fortran</i> , it must be a defoult integer value.
IN	sst	The stride between consecutive elements of the <i>source</i> data object. The stride is scaled by the element size. A value of <i>1</i> indicates contiguous data. <i>sst</i> must be of type <i>ptrdiff_t</i> . When using <i>Fortran</i> , it must be a default integer value.
IN	nelems	The number of elements to exchange for each PE. <i>nelems</i> must be of type size_t for $C/C++$. When using <i>Fortran</i> , it must be a default integer value.
IN	PE_start	The lowest PE number of the active set of PEs. <i>PE_start</i> must be of type integer. When using <i>Fortran</i> , it must be a default integer value.
IN	logPE_stride	The log (base 2) of the stride between consecutive PE numbers in the active set. <i>logPE_stride</i> must be of type integer. When using <i>Fortran</i> , it must be a default integer value.
IN	PE_size	The number of PEs in the active set. <i>PE_size</i> must be of type integer. When using <i>Fortran</i> , it must be a default integer value.
IN	pSync	A symmetric work array of size <i>SHMEM_ALLTOALLS_SYNC_SIZE</i> . In <i>C/C++</i> , <i>pSync</i> must be an array of elements of type <i>long</i> . In <i>Fortran</i> , <i>pSync</i> must be an array of elements of default integer type. Every element of this array must be initialized with the value <i>SHMEM_SYNC_VALUE</i> before any of the PEs in the active set enter the routine.

API description

The *shmem_alltoalls* routines are collective routines. Each PE in the active set exchanges *nelems* strided data elements of size 32 bits (for *shmem_alltoalls32*) or 64 bits (for *shmem_alltoalls64*) with all other PEs in the set. Both strides, *dst* and *sst*, must be greater than or equal to 1. Given a PE *i* that is the k^{th} PE in the active set and a PE *j* that is the l^{th} PE in the active set, PE *i* sends the *sst*l*thblock of the *source* data object to the *dst*k*thblock of the *dest* data object on PE *j*.

As with all OpenSHMEM collective routines, these routines assume that only PEs in the active set call the routine. If a PE not in the active set calls an OpenSHMEM collective routine, undefined behavior results. The values of arguments *dst*, *sst*, *nelems*, *PE_start*, *logPE_stride*, and *PE_size* must be equal on all PEs in the active set. The same *dest* and *source* data objects, and the same *pSync* work array must be passed to all PEs in the active set.

Before any PE calls a *shmem_alltoalls* routine, the following conditions must be ensured:

- The *pSync* array on all PEs in the active set is not still in use from a prior call to a *shmem_alltoall* routine.
- The dest data object on all PEs in the active set is ready to accept the shmem_alltoalls data.

Otherwise, the behavior is undefined.

Upon return from a *shmem_alltoalls* routine, the following is true for the local PE: Its *dest* symmetric data object is completely updated and the data has been copied out of the *source* data object. The values in the *pSync* array are restored to the original values.

The dest and source data objects must conform to certain typing constraints, which are as follows:

Routine	Data type of <i>dest</i> and <i>source</i>	47
		48

1	shmem_alltoalls64	64 bits aligned.
2	shmem_alltoalls32	32 bits aligned.
3		
4		
5	Return Values	
6	None.	
7		
8	Notes	
9	This routine restores <i>pSync</i> to	its original contents. Multiple calls to OpenSHMEM routines that use the
10	same <i>pSync</i> array do not requir	re that $pSync$ be reinitialized after the first call. The user must ensure that the
11	pSync array is not being update	ed by any PE in the active set while any of the PEs participates in processing
12	of an OpenSHMEM shmem_c	<i>ultoalls</i> routine. Be careful to avoid these situations: If the <i>pSync</i> array is
13	initialized at run time, some ty	pe of synchronization is needed to ensure that all PEs in the active set have
14	initialized <i>pSync</i> before any of	them enter an OpenSHMEM routine called with the <i>pSync</i> synchronization
15	afray. A <i>psync</i> array may be r	still processing a prior OpenSHMEM shmem_alloadus routine call that used
16	the same <i>p</i> Sv <i>pc</i> array. In gener	al this can be ensured only by doing some type of synchronization
17	the sume poyne array. In gener	ar, this can be choused only by doing some type of synchronization.
18		
10	EXAMPLES	
20		
20	This example shows a <i>shmem_allton</i>	alls64 on two long elements among all PEs.
21	#include <stdio.h></stdio.h>	
22	#include <inttypes.h></inttypes.h>	
25	#include <shmem.h></shmem.h>	
24	<pre>int main(void)</pre>	
25	{	
26	static long pSync[SHMEM_ALL	TUALLS_SYNC_SIZE ;

```
for (int i = 0; i < SHMEM_ALLTOALLS_SYNC_SIZE; i++)</pre>
27
                  pSync[i] = SHMEM_SYNC_VALUE;
28
               shmem_init();
29
               int me = shmem_my_pe();
30
               int npes = shmem_n_pes();
31
               const int count = 2;
32
               const ptrdiff_t dst = 2;
               const ptrdiff_t sst = 3;
33
               int64_t* dest = (int64_t*) shmem_malloc(count * dst * npes * sizeof(int64_t));
34
               int64_t* source = (int64_t*) shmem_malloc(count * sst * npes * sizeof(int64_t));
35
               /* assign source values */
36
               for (int pe = 0; pe < npes; pe++) {
   for (int i = 0; i < count; i++) {</pre>
37
                      source[sst * ((pe * count) + i)] = me + pe;
38
                      dest[dst * ((pe * count) + i)] = 9999;
39
                  }
               }
40
               /* wait for all PEs to update source/dest */
41
               shmem_barrier_all();
42
               /* alltoalls on all PES */
43
               shmem_alltoalls64(dest, source, dst, sst, count, 0, 0, npes, pSync);
44
               /* verify results */
45
               for (int pe = 0; pe < npes; pe++) {</pre>
46
                  for (int i = 0; i < count; i++) {</pre>
                      int j = dst * ((pe * count) + i);
```

printf("[%d] ERROR: dest[%d]=%" PRId64 ", should be %d\n",

if (dest[j] != pe + me) {

47

```
me, j, dest[j], pe + me);
}
shmem_free(dest);
shmem_free(source);
shmem_finalize();
return 0;
```

9.10 Point-To-Point Synchronization Routines

The following section discusses OpenSHMEM APIs that provide a mechanism for synchronization between two PEs based on the value of a symmetric data object. The point-to-point synchronization routines can be used to portably ensure that memory access operations observe remote updates in the order enforced by the initiator PE using the *shmem_fence* and *shmem_quiet* routines.

Where appropriate compiler support is available, OpenSHMEM provides type-generic point-to-point synchronization interfaces via *C11* generic selection. Such type-generic routines are supported for the "point-to-point synchronization types" identified in Table 7.

The point-to-point synchronization types include some of the exact-width integer types defined in *stdint.h* by C99 §7.18.1.1 and C11 §7.20.1.1. When the *C* translation environment does not provide exact-width integer types with *stdint.h*, an OpenSHMEM implementation is not required to provide support for these types.

ТҮРЕ	TYPENAME
short	short
int	int
long	long
long long	longlong
unsigned short	ushort
unsigned int	uint
unsigned long	ulong
unsigned long long	ulonglong
int32_t	int32
int64_t	int64
uint32_t	uint32
uint64_t	uint64
size_t	size
ptrdiff_t	ptrdiff

Table 7: Point-to-Point Synchronization Types and Names

The point-to-point synchronization interface provides named constants whose values are integer constant expressions that specify the comparison operators used by OpenSHMEM synchronization routines. The constant names and associated operations are presented in Table 8. For Fortran, the constant names of Table 8 shall be identifiers for integer parameters of default kind corresponding to the associated comparison operation.

9.10.1 SHMEM_WAIT_UNTIL

Wait for a variable on the local PE to change.

SYNOPSIS

C11:

					-	
1		Constant	Name	Comparison	_	
2		SHMEM_	_CMP_EQ	Equal		
3		SHMEM_	_CMP_NE	Not equal		
4		SHMEM_	_CMP_GT	Greater than		
5		SHMEM_	_CMP_GE	Greater than or equal to		
6		SHMEM_	_CMP_LT	Less than		
7		SHMEM_	_CMP_LE	Less than or equal to	_	
8		Table 8: 1	Point-to-Poi	nt Comparison Constants		
0						
1	void shmem_wait_	until(TYPE *ivar	, int cmp,	TYPE cmp_value);		
12	where TYPE is one	of the point-to-poir	nt synchroni	zation types specified by T	able 7.	
14	C/C++:					
15	<pre>void shmem_<type< pre=""></type<></pre>	NAME>_wait_until	(TYPE *iva:	r, int cmp, TYPE cmp_v	alue);	
15	where TYPE is one	e of the point-to-poi	nt synchron	ization types and has a co	rresponding TYPENAME speci	ified
16	by Table 7.					
17	— deprecation star	t				
18	void shmem_wait_	until(long *ivar	, int cmp,	<pre>long cmp_value);</pre>		
19	void shmem_wait(<pre>long *ivar, long</pre>	cmp_value);		
20	void shmem_ <type< td=""><td>NAME>_wait(TYPE</td><td>*ivar, TYP</td><td>z cmp_value);</td><td></td><td></td></type<>	NAME>_wait(TYPE	*ivar, TYP	z cmp_value);		
21	where TYPE is one	of {short, int, long	long long	and has a corresponding T	YPENAME specified by Table	7.
22		()) ()		1 0	deprecation en	d
23						u —
24	— deprecation star	t				
25	FORTRAN:					
26	CALL SHMEM_INT4_	WAIT(ivar, cmp_va	alue)	3		
27	CALL SHMEM_INT4_	WAIT_UNTIL(ivar,	cmp, cmp_	value)		
28	CALL SHMEM_INI8_	WAII(IVar, Cmp_va	aiue)			
29	CALL SHMEM_INIO_	WAII_UNIIL(IVaI,	cmp, cmp_	value)		
30	CALL SHMEM_WAIT(INTIL (ivar cmp				
31	CALL SHMEM_WAIT_	owill(ival, cmp,	cmp_varue)		
32 33 34 DES	SCRIPTION				deprecation en	d —
35	Arguments					
37 38	OUT	ivar	A remote <i>ivar</i> shou	ly accessible integer varial ld match that implied in th	ble. When using $C/C++$, the type SYNOPSIS section.	pe of
39 40 41	IN	стр	The comp <i>Fortran</i> , t	pare operator that compare it must be of default kind.	s <i>ivar</i> with <i>cmp_value</i> . When when using <i>C/C++</i> , it must	using be of
42 43 44 45 46	IN	cmp_value	cmp_valu cmp_valu using For as ivar.	<i>ue</i> must be of type integen <i>ue</i> should match that implie <i>tran</i> , cmp_value must be a	When using $C/C++$, the typed in the SYNOPSIS section. We an integer of the same size and	pe of When kind
47 48	API description					

shmem_wait and *shmem_wait_until* wait for *ivar* to be changed by a write or an atomic operation issued by a PE. These routines can be used for point-to-point direct synchronization. A call to *shmem_wait* does not return until a PE writes a value not equal to *cmp_value* into *ivar* on the waiting PE. A call to *shmem_wait_until* does not return until a PE changes *ivar* to satisfy the condition implied by *cmp* and *cmp_value*. The *shmem_wait* routines return when *ivar* is no longer equal to *cmp_value*. The *shmem_wait_until* routines return when the compare condition is true. The compare condition is defined by the *ivar* argument compared with the *cmp_value* using the comparison operator *cmp*.

When using *Fortran*, *ivar* must be a specific sized integer type according to the routine being called, as follows:

Routine	Data type
shmem_wait, shmem_wait_until	default INTEGER
shmem_int4_wait,	INTEGER*4
shmem_int4_wait_until	
shmem_int8_wait,	INTEGER*8
shmem_int8_wait_until	
Return Values	
None.	
Notes	
As of OpenSHMEM 1.4, the shme	em_wait routine is deprecated, however, shmem_wait is equivalent to
shmem_wait_until where cmp is Sh	HMEM_CMP_NE.
Note to implementors	
Implementations must ensure that	shmem wait and shmem wait until do not return before the update of
the memory indicated by <i>ivar</i> is ful	ly complete. Partial updates to the memory must not cause <i>shmem_wait</i>
or shmem_wait_until to return.	
EXAMPLES	
The following call returns when variable	iver is not equal to 100.
	war is not equal to 100.
INCLUDE "Snmem.In"	
INTEGER*8 IVAR	
CALL SHMEM_INT8_WAIT (IVAR, INTEGER	**8(100))
The following call to SHMEM_INT8_W	AIT_UNTIL is equivalent to the call to SHMEM_INT8_WAIT in exam-
ple 1:	-
INCLUDE "shmem.fh"	
CALL SHMEM INT8 WAIT UNTIL(IVAR, S	HMEM CMP NE, INTEGER*8(100))
The following $C/C++$ call waits until the	e value in <i>ivar</i> is set to be less than zero by a transfer from a remote PE:
#include <stdio.h></stdio.h>	
<pre>#include <shmem.h></shmem.h></pre>	
<pre>int ivar;</pre>	
<pre>shmem_int_wait_until(&ivar, SHMEM_</pre>	_CMP_LT, 0);

```
1
            The following Fortran example is in the context of a subroutine:
2
            INCLUDE "shmem.fh"
3
            SUBROUTINE EXAMPLE()
            INTEGER FLAG_VAR
            COMMON/FLAG/FLAG_VAR
             . . .
            FLAG_VAR = FLAG_VALUE
                                         ! initialize the event variable
            IF (FLAG_VAR .EQ. FLAG_VALUE) THEN
                      CALL SHMEM_WAIT(FLAG_VAR, FLAG_VALUE)
9
            ENDIF
            FLAG_VAR = FLAG_VALUE
                                         ! reset the event variable for next time
10
11
            END
12
13
      9.10.2 SHMEM_TEST
14
15
      Test whether a variable on the local PE has changed.
16
17
      SYNOPSIS
18
            C11:
19
            int shmem_test(TYPE *ivar, int cmp, TYPE cmp_value);
20
            where TYPE is one of the point-to-point synchronization types specified by Table 7.
21
22
            C/C++:
23
            int shmem_<TYPENAME>_test(TYPE *ivar, int cmp, TYPE cmp_value);
24
            where TYPE is one of the point-to-point synchronization types and has a corresponding TYPENAME specified
            by Table 7.
25
26
27
      DESCRIPTION
28
29
            Arguments
30
31
                   OUT
                                                    A pointer to a remotely accessible data object.
                                   ivar
32
                   IN
                                   cmp
                                                    The comparison operator that compares ivar with cmp_value.
                   IN
                                                    The value against which the object pointed to by ivar will be compared.
33
                                   cmp_value
34
35
36
            API description
37
38
                 shmem_test tests the numeric comparison of the symmetric object pointed to by ivar with the value
39
                 cmp_value according to the comparison operator cmp.
40
41
42
            Return Values
43
                 shmem_test returns 1 if the comparison of the symmetric object pointed to by ivar with the value cmp_value
44
                 according to the comparison operator cmp evaluates to true; otherwise, it returns 0.
45
46
            Notes
47
                 None.
48
```

9. OPENSHMEM LIBRARY API

EXAMPLES

The following example demonstrates the use of *shmem_test* to wait on an array of symmetric objects and return the index of an element that satisfies the specified condition.

```
#include <stdio.h>
#include <shmem.h>
int user_wait_any(long *ivar, int count, int cmp, long value)
 int idx = 0;
 while (!shmem_test(&ivar[idx], cmp, value))
   idx = (idx + 1) % count;
 return idx;
int main(void)
 shmem_init();
 const int mype = shmem_my_pe();
 const int npes = shmem_n_pes();
 long *wait_vars = shmem_calloc(npes, sizeof(long));
 if (mype == 0)
   int who = user_wait_any(wait_vars, npes, SHMEM_CMP_NE, 0);
   printf("PE %d observed first update from PE %d\n", mype, who);
 else
    shmem_p(&wait_vars[mype], mype, 0);
 shmem_free(wait_vars);
 shmem_finalize();
 return 0;
```

9.11 Memory Ordering Routines

The following section discusses OpenSHMEM APIs that provide mechanisms to ensure ordering and/or delivery of Put, AMO, memory store, and non-blocking Put and Get routines to symmetric data objects.

9.11.1 SHMEM_FENCE

Assures ordering of delivery of Put, AMO, memory store, and nonblocking Put routines to symmetric data objects.

SYNOPSIS

C/C++:	
<pre>void shmem_fence(void);</pre>	
<pre>void shmem_ctx_fence(shmem_ctx_t ctx);</pre>	
- deprecation start	
CALL SHMEM_FENCE	
	 —— deprecation end —

DESCRIPTION

Arguments IN

ctx

The context on which to perform the operation. When this argument is not provided, the operation is performed on SHMEM_CTX_DEFAULT. 2

6

9

10 11

12

13

14

15

16

17

18

19

20

21 22

23

24 25 26

27 28

29

30 31

32

33 34

45 46

47

API description

This routine assures ordering of delivery of *Put*, AMO, memory store, and nonblocking *Put* routines to symmetric data objects. All *Put*, AMO, memory store, and nonblocking *Put* routines to symmetric data objects issued to a particular remote PE on the given context prior to the call to *shmem_fence* are guaranteed to be delivered before any subsequent *Put*, AMO, memory store, and nonblocking *Put* routines to symmetric data objects to the same PE. *shmem_fence* guarantees order of delivery, not completion. It does not guarantee order of delivery of nonblocking *Get* routines.

Return Values

None.

Notes

shmem_fence only provides per-PE ordering guarantees and does not guarantee completion of delivery. *shmem_fence* also does not have an effect on the ordering between memory accesses issued by the target PE. *shmem_wait_until, shmem_test, shmem_barrier, shmem_barrier_all* routines can be called by the target PE to guarantee ordering of its memory accesses. There is a subtle difference between *shmem_fence* and *shmem_quiet*, in that, *shmem_quiet* guarantees completion of *Put*, AMO, memory store, and nonblocking *Put* routines to symmetric data objects which makes the updates visible to all other PEs.

The *shmem_quiet* routine should be called if completion of *Put*, AMO, memory store, and nonblocking *Put* routines to symmetric data objects is desired when multiple remote PEs are involved.

In an OpenSHMEM program with multithreaded PEs, it is the user's responsibility to ensure ordering between operations issued by the threads in a PE that target symmetric memory (e.g. *Put*, AMO, memory stores, and nonblocking routines) and calls by threads in that PE to *shmem_fence*. The *shmem_fence* routine can enforce memory store ordering only for the calling thread. Thus, to ensure ordering for memory stores performed by a thread that is not the thread calling *shmem_fence*, the update must be made visible to the calling thread according to the rules of the memory model associated with the threading environment.

EXAMPLES

The following example uses *shmem_fence* in a *C11* program:

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

```
int main(void)
```

```
int src = 99;
long source[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
static long dest[10];
static int targ;
shmem_init();
int me = shmem_my_pe();
if (me == 0) {
   shmem_put(dest, source, 10, 1); /* put1 */
   shmem_put(dest, source, 10, 2); /* put2 */
   shmem_fence();
   shmem_put(&targ, &src, 1, 1); /* put3 */
   shmem_put(&targ, &src, 1, 2); /* put4 */
shmem_barrier_all(); /* sync sender and receiver */
printf("dest[0] on PE %d is %ld\n", me, dest[0]);
shmem finalize();
return 0;
```

Put1 will be ordered to be delivered before *put3* and *put2* will be ordered to be delivered before *put4*.

1

2

3

9 10

11

12

14

15

16

17

18

19

20

21

22

23

24

25

26 27 28

29 30

31

32

33

34

35

36

37

38

39

40

41

42

43 44

45

46

9.11.2 SHMEM_QUIET

Waits for completion of all outstanding *Put*, AMO, memory store, and nonblocking *Put* and *Get* routines to symmetric data objects issued by a PE.

SYNOPSIS

C/C++:
void shmem_quiet(void);
void shmem_ctx_quiet(shmem_ctx_t ctx);

ctx

deprecation start	
CALL SHMEM_QUIET	
	deprecation end –

DESCRIPTION

Arguments

IN

The context on which to perform the operation. When this argument is not provided, the operation is performed on *SHMEM_CTX_DEFAULT*.

API description

The *shmem_quiet* routine ensures completion of *Put*, AMO, memory store, and nonblocking *Put* and *Get* routines on symmetric data objects issued by the calling PE on the given context. All *Put*, AMO, memory store, and nonblocking *Put* and *Get* routines to symmetric data objects are guaranteed to be completed and visible to all PEs when *shmem_quiet* returns.

Return Values

None.

Notes

shmem_quiet is most useful as a way of ensuring completion of several Put, AMO, memory store, and non-blocking Put and Get routines to symmetric data objects initiated by the calling PE. For example, one might use shmem_quiet to await delivery of a block of data before issuing another Put or nonblocking Put routine, which sets a completion flag on another PE. shmem_quiet is not usually needed if shmem_barrier_all or shmem_barrier are called. The barrier routines wait for the completion of outstanding writes (Put, AMO, memory stores, and nonblocking Put and Get routines) to symmetric data objects on all PEs.

In an OpenSHMEM program with multithreaded PEs, it is the user's responsibility to ensure ordering between operations issued by the threads in a PE that target symmetric memory (e.g. *Put*, AMO, memory stores, and nonblocking routines) and calls by threads in that PE to *shmem_quiet*. The *shmem_quiet* routine can enforce memory store ordering only for the calling thread. Thus, to ensure ordering for memory stores performed by a thread that is not the thread calling *shmem_quiet*, the update must be made visible to the calling thread according to the rules of the memory model associated with the threading environment.

A call to *shmem_quiet* by a thread completes the operations posted prior to calling *shmem_quiet*. If the user intends to also complete operations issued by a thread that is not the thread calling *shmem_quiet*, the user must ensure that the operations are performed prior to the call to *shmem_quiet*. This may require the use of a synchronization operation provided by the threading package. For example, when using POSIX Threads, the user may call the *phread_barrier_wait* routine to ensure that all threads have issued operations before a thread calls *shmem_quiet*.

shmem_quiet does not have an effect on the ordering between memory accesses issued by the target PE. *shmem_wait_until, shmem_test, shmem_barrier, shmem_barrier_all* routines can be called by the target PE to guarantee ordering of its memory accesses.

EXAMPLES

```
The following example uses shmem_quiet in a C11 program:
#include <stdio.h>
#include <shmem.h>
int main(void)
   static long dest[3];
   static long source[3] = { 1, 2, 3 };
   static int targ;
   static int src = 90;
   long x[3] = \{0\};
   int y = 0;
   shmem_init();
   int me = shmem_my_pe();
   if (me == 0) {
      shmem_put(dest, source, 3, 1); /* put1
      shmem_put(&targ, &src, 1, 2);
                                     /* put2 *
      shmem_quiet();
      shmem_get(x, dest, 3, 1); /* gets updated value from dest on PE 1 to local array x */
      shmem_get(&y, &targ, 1, 2); /* gets updated value from targ on PE 2 to local variable
    y */
      printf("x: { %ld, %ld }\n", x[0], x[1], x[2]); /* x: { 1, 2, 3 } */
      printf("y: %d\n", y); /* y: 90 */
      shmem_put(&targ, &src, 1, 1); /* put3 */
      shmem_put(&targ, &src, 1, 2); /* put4 */
   shmem_finalize();
   return 0;
Put1 and put2 will be completed and visible before put3 and put4.
```

9.11.3 Synchronization and Communication Ordering in OpenSHMEM

⁴⁵ When using the OpenSHMEM API, synchronization, ordering, and completion of communication become critical. The ⁴⁶ updates via *Put* routines, AMOs, stores, and nonblocking *Put* and *Get* routines on symmetric data cannot be guaranteed ⁴⁷ until some form of synchronization or ordering is introduced in the user's program. The table below gives the different ⁴⁷ synchronization and ordering choices, and the situations where they may be useful.

1

2

9

10 11 12

13 14 15

16 17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33





9. OPENSHMEM LIBRARY API



completed before any PE returns from the call. Additionally no PE shall return from the barrier until all PEs have entered the same *shmem_barrier_all* call. This routine should be used when synchronization as well as completion of all stores and remote memory updates via OpenSHMEM is required over all PEs.

9.12 Distributed Locking Routines

The following section discusses OpenSHMEM locks as a mechanism to provide mutual exclusion. Three routines are available for distributed locking, *set, test* and *clear*.

9.12.1 SHMEM_LOCK

Releases, locks, and tests a mutual exclusion memory lock.

SYNOPSIS

C/C++	+:					
void a	shmem_clear_1	<pre>lock(long *lock);</pre>				
void a	shmem_set_lo	ck(long *lock);				
int sh	hmem_test_lo	ck(long *lock);				
— dep	precation start -					
FORI	IRAN:					
INTEGH	ER lock, SHMI	EM_TEST_LOCK				
CALL S	SHMEM_CLEAR_1	LOCK(lock)				
CALL S	SHMEM_SET_LO	CK(lock)				
I = SH	HMEM_TEST_LO	CK(lock)				
						deprecation end —
DESCRIPT	ION					
Argun	nents					
	IN	lock	A symmetric data of	pject that is a scal	lar variable or	an array of length I

API description

The *shmem_set_lock* routine sets a mutual exclusion lock after waiting for the lock to be freed by any other PE currently holding the lock. Waiting PEs are assured of getting the lock in a first-come, first-served manner. The *shmem_clear_lock* routine releases a lock previously set by *shmem_set_lock* after ensuring that all local and remote stores initiated in the critical region are complete. The *shmem_test_lock* routine sets a mutual exclusion lock only if it is currently cleared. By using this routine, a PE can avoid blocking on a set lock. If the lock is currently set, the routine returns without waiting. These routines are appropriate for protecting a critical region from simultaneous update by multiple PEs.

This data object must be set to 0 on all PEs prior to the first use. lock

must be of type long. When using Fortran, it must be of default kind.

Return Values

The *shmem_test_lock* routine returns 0 if the lock was originally cleared and this call was able to set the lock. A value of 1 is returned if the lock had been set and the call returned without waiting to set the lock.

Notes

The term symmetric data object is defined in Section 3. The lock variable should always be initialized to zero and accessed only by the OpenSHMEM locking API. Changing the value of the lock variable by other means without using the OpenSHMEM API, can lead to undefined behavior.

EXAMPLES

The following example uses *shmem_lock* in a *C11* program.

```
#include <stdio.h>
#include <shmem.h>
int main(void)
  static long lock = 0;
   static int count = 0;
  shmem_init();
  int me = shmem_my_pe();
  shmem_set_lock(&lock);
  int val = shmem_g(&count, 0); /* get count value on PE 0 */
  printf("%d: count is %d\n", me, val);
  val++; /* incrementing and updating count on PE 0 */
   shmem_p(&count, val, 0);
   shmem_quiet();
   shmem_clear_lock(&lock);
   shmem_finalize();
   return 0;
```

9.13 Cache Management

All of these routines are deprecated and are provided for backwards compatibility. Implementations must include all items in this section, and the routines should function properly and may notify the user about deprecation of their use.

9.13.1 SHMEM_CACHE

Controls data cache utilities.

SYNOPSIS

- deprecation start -

C/C++:

```
void shmem_clear_cache_inv(void);
void shmem_set_cache_inv(void);
void shmem_clear_cache_line_inv(void *dest);
void shmem_set_cache_line_inv(void *dest);
void shmem_udcflush(void);
void shmem_udcflush_line(void *dest);
```

deprecation end -

deprecation end -

- deprecation start

```
FORTRAN:
```

CALL SHMEM_CLEAR_CACHE_INV CALL SHMEM_SET_CACHE_INV CALL SHMEM_SET_CACHE_LINE_INV(dest) CALL SHMEM_UDCFLUSH CALL SHMEM_UDCFLUSH_LINE(dest) 2

9

10

11

12

13

14

15

16

17 18 19

20 21

22

23 24

25 26

27 28

29

30 31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

1	Arguments		
2 3	IN	dest	A data object that is local to the PE. <i>dest</i> can be of any noncharacter
4			type. when using <i>Forman</i> , it can be of any kind.
5			
6	API description		
8			
9	shmem_set_c	ache_inv enable	es automatic cache coherency mode.
10	shmem_set_co the address of	ache_line_inv e dest only.	enables automatic cache coherency mode for the cache line associated with
11	shmem_clear_ shmem_set_c	_cache_inv di ache _inv or shr	sables automatic cache coherency mode previously enabled by <i>nem_set_cache_line_inv</i> .
13	shmem_udcflı	ush makes the en	ntire user data cache coherent.
14	shmem_udcfli	ush_line makes	coherent the cache line that corresponds with the address specified by <i>dest</i> .
15			
16			
17	Return Values		
19	None.		
20	.		
21	Notes These routine	s have been ret	ained for improved backward compatibility with leavey architectures. They
22	are not require	ed to be support	ed by implementing them as <i>no-ops</i> and where used, they may have no effect
23	on cache line	states.	
24			
25	EVAMDI ES		
26	EAAMI LES		
27	None.		
28			
30			
31			
32			
33			
34			*
35			
36			
37			
38			
40			
41			
42			
43			
44			
45			
46			
47			
48			

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* or *shmem.fh* for *Fortran*.
- Add the initialization call *shmem_init*.
- Use OpenSHMEM calls to query the local 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

```
#include <stdio.h>
1
2
   #include <shmem.h> /* The OpenSHMEM header file */
3
4
   int main (void)
5
6
      shmem_init();
7
      int me = shmem_my_pe();
8
      int npes = shmem_n_pes();
9
      printf("Hello from %d of %d\n", me, npes);
10
      shmem_finalize();
11
      return 0;
12
```

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

 1
 Hello from 0 of 4

 2
 Hello from 2 of 4

 3
 Hello from 3 of 4

 4
 Hello from 1 of 4

	deprecation start
Oj	penSHMEM also provides a Fortran API. Listing A.3 shows a similar program written in Fortran.
	Listing A.3: "Hello World" example program in Fortran
1	program hello
23	include "shmem.fh"
4	integer :: shmem_my_pe, shmem_n_pes
6	integer :: npes, me
7 8	call shmem init ()
9	npes = shmem_n_pes ()
10 11	me = shmem_my_pe ()
12	write (*, 1000) me, npes
13 14	1000 format ('Hello from', 1X, I4, 1X, 'of', 1X, I4)
15	and program hells
10	end program herro
	Listing A.4: Possible ordering of expected output with 4 PEs from the program in Listing A.3
1	Hello from 0 of 4 Hello from 2 of 4
3	Hello from 3 of 4
4	Hello from 1 of 4
	deprecation end

ANNEX A. WRITING OPENSHMEM PROGRAMS

The example in Listing A.5 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.5: Example program with symmetric data objects

```
1
   #include <stdio.h>
2
   #include <shmem.h>
3
   #define SIZE 16
4
5
6
   int main (void)
7
8
       short source[SIZE];
9
       static short dest[SIZE];
10
       static long lock = 0;
11
       shmem_init();
12
       int me = shmem_my_pe();
13
       int npes = shmem_n_pes();
      if (me == 0) {
14
15
          /* initialize array */
16
          for (int i = 0; i < SIZE; i++)</pre>
17
            source[i] = i;
18
          /* local, not symmetric */
          /* static makes it symmetric */
19
20
          /* put "size" words into dest on each PE */
21
          for (int i = 1; i < npes; i++)</pre>
22
             shmem_put(dest, source, SIZE, i);
23
24
       shmem_barrier_all(); /* sync sender and receiver */
25
       if (me != 0) {
26
          shmem set lock(&lock);
27
          printf("dest on PE %d is \t", me);
28
          for (int i = 0; i < SIZE; i++)</pre>
            printf("%hd \t", dest[i]);
29
30
          printf("\n");
31
          shmem_clear_lock(&lock);
32
33
       shmem finalize();
34
       return 0;
35
```

Listing A.6: Possible ordering of expected output with 4 PEs from the program in Listing A.5

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

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

 3
 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:

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

Where the $\langle \text{compiler options} \rangle$ 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++**.

Programs written in Fortran

— deprecation start

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

oshfort <compiler options> -o myprogram myprogram.f

Where the $\langle \text{compiler options} \rangle$ are options understood by the underlying *Fortran* compiler called by **oshfort**.

- deprecation end —

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:

$\langle runner options \rangle$	Options passed to the underlying launcher.
-np $\langle \# \rangle$	The number of PEs to be used in the execution.
(program)	The program executable to be launched.
$\langle program arguments \rangle$	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	In an OpenSHMEM program where the initialization routines
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	The symmetric memory management routines are collectives. For
symmetric memory	example, all PEs in the program must call <i>shmem_malloc</i> with the
	same <i>size</i> argument. Program behavior after a mismatched
	<i>shmem_malloc</i> call is undefined.
Use of null pointers with non-zero len specified	In any OpenSHMEM routine that takes a pointer and <i>len</i> describing the number of elements in that pointer, a null pointer may not be given unless the corresponding <i>len</i> is also specified as zero. Otherwise, the resulting behavior is undefined. The following cases summarize this behavior:
	• <i>len</i> is 0, pointer is null: supported.
	• <i>len</i> is not 0, pointer is null: undefined behavior.
	• <i>len</i> is 0, pointer is non-null: supported.
	• <i>len</i> is not 0, pointer is non-null: supported.

Annex D

Interoperability with other Programming Models

1 MPI Interoperability

OpenSHMEM routines may be used in conjunction with MPI routines in the same program. For example, on Silicon Graphics International (SGI) systems, programs that use both MPI and OpenSHMEM routines call *MPI_Init* and *MPI_Finalize* but omit the call to the *shmem_init* routine. OpenSHMEM PE numbers are equal to the MPI rank within the *MPI_COMM_WORLD* environment variable. Note that this indexing precludes use of OpenSHMEM routines between processes in different *MPI_COMM_WORLD*s. For example, MPI processes started using the *MPI_Comm_spawn* routine cannot use OpenSHMEM routines to communicate with their parent MPI processes.

On SGI systems where MPI jobs use *Transmission Control Protocol* (TCP)/sockets for inter-host communication, OpenSHMEM routines may be used to communicate with processes running on the same host. The *shmem_pe_accessible* routine should be used to determine if a remote PE is accessible via OpenSHMEM communication from the local PE. When running an MPI program involving multiple executable files, OpenSHMEM routines may be used to communicate with processes running from the same or different executable files, provided that the communication is limited to symmetric data objects. On these systems, static memory—such as a *Fortran* common block or *C* global variable—is symmetric between processes running from the same executable file, but is not symmetric between processes running from the symmetric heap (e.g., *shmem_malloc, shpalloc*) is symmetric across the same or different executable files. The *shmem_addr_accessible* routine should be used to determine if a local address is accessible via OpenSHMEM communication from a remote PE.

Another important feature of these systems is that the *shmem_pe_accessible* routine returns *TRUE* only if the remote PE is a process running from the same executable file as the local PE, indicating that full OpenSHMEM support (static memory and symmetric heap) is available. When using OpenSHMEM routines within an MPI program, the use of MPI memory-placement environment variables is required when using non-default memory-placement options.

Annex E

1

9

15

16

17 18

19

20 21

22

23 24

25

26

27

28

29

30

31

33 34

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, 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	1.2	Current	(#0#0)
Fortran: SHMEM_CLEAR_CACHE_INV	1.5	Current	(none)
C/C++: shmem_clear_cache_line_inv	1.3	Current	(none)
C/C++: shmem_set_cache_inv	1.2	Current	(nona)
Fortran: SHMEM_SET_CACHE_INV	1.5	Current	(none)
C/C++: shmem_set_cache_line_inv	13	Current	(none)
Fortran: SHMEM_SET_CACHE_LINE_INV	1.5	Current	(none)
C/C++: shmem_udcflush	13	Current	(none)
Fortran: SHMEM_UDCFLUSH	1.5	Current	(none)
C/C++: shmem_udcflush_line	13	Current	(none)
Fortran: SHMEM_UDCFLUSH_LINE	110	Current	
_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

1	Deprecated API	Deprecated Since	Last Version Supported	Replaced By
	_SHMEM_CMP_GT	1.3	Current	SHMEM_CMP_GT
2	_SHMEM_CMP_GE	1.3	Current	SHMEM_CMP_GE
3	SMA_VERSION	1.4	Current	SHMEM_VERSION
-	SMA_INFO	1.4	Current	SHMEM_INFO
4	SMA_SYMMETRIC_SIZE	1.4	Current	SHMEM_SYMMETRIC_SIZE
	SMA_DEBUG	1.4	Current	SHMEM_DEBUG
5	C/C++: shmem_wait C/C++: shmem_< TYPENAME >_wait	1.4	Current	See Notes for <i>shmem_wait_until</i>
	C/C++: shmem_wait_until	1.4	Current	C11: shmem_wait_until, C/C++: shmem_long_wait_until
7	C11: shmem_fetch C/C++: shmem_< TYPENAME >_fetch	1.4	Current	shmem_atomic_fetch
8 9	C11: shmem_set C/C++: shmem_< TYPENAME >_set	1.4	Current	shmem_atomic_set
10	C11: shmem_cswap C/C++: shmem_< TYPENAME >_cswap	1.4	Current	shmem_atomic_compare_swap
11	C11: shmem_swap C/C++: shmem_< TYPENAME >_swap	1.4	Current	shmem_atomic_swap
12	C11: shmem_finc C/C++: shmem_< TYPENAME >_finc	1.4	Current	shmem_atomic_fetch_inc
13	C11: shmem_inc C/C++: shmem_< TYPENAME >_inc	1.4	Current	shmem_atomic_inc
15	C11: shmem_fadd C/C++: shmem_< TYPENAME >_fadd	1.4	Current	shmem_atomic_fetch_add
16	C11: shmem_add C/C++: shmem_< TYPENAME >_add	1.4	Current	shmem_atomic_add
17	Entire Fortran API	1.4	Current	(none)

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

27 #include <mpp/shmem.h>

#include <mpp/shmemx.h>

²⁹ and in *Fortran* as

30 include 'mpp/shmem.fh'

31 include 'mpp/shmemx.fh'

³² 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.

40 41

42

43

44 45

46

22 23

24

25

26

28

34

35

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 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

<i>C/C</i> ++:	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
<pre>shmem_clear_cache_line_inv</pre>	

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	Ε	_SHMEM_CMP_LE
6	_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.

4				
5	2 12	C11 and $C/C++$	shmem fetch sl	hmem set shmem cswap shmem swap shmem finc
7		shmem inc. shme	em fadd. shmem	add
8				
9	The CI	I and $C/C++$ interface	es for	
10			<i>C11</i> :	<i>C/C</i> ++:
11			shmem_fetch	shmem_ <typename>_fetch</typename>
12			shmem_set shmem_cswap	shmem_ <iiiename>_set shmem_<typename>_cswap</typename></iiiename>
13			shmem_swap	shmem_ <typename>_swap</typename>
14			shmem_finc	shmem_ <typename>_finc</typename>
16			shmem_inc	shmem_ <typename>_inc shmem_<typename>_fadd</typename></typename>
17			shmem_jaaa shmem_add	shmem <typename> add</typename>
18				
19	were d	eprecated and replaced	with similarly name	d interfaces within the <i>shmem_atomic_</i> * namespace in order to more
20	"fadd"	were expanded for clar	rity to "compare sw	ap", "fetch inc", and "fetch add".
21		·····		
22	2 13	Fortran API		
23	2.1.0			
25	The en	tire OpenSHMEM For	tran API was deprec	ated because of a general lack of use and a lack of conformance with
26	OpenS	HMEM Specification's	eu of an extensive up C APL through the	bate of the <i>Fortran</i> AP1, <i>Fortran</i> users are encouraged to leverage the <i>Fortran</i> -C interoperability initially standardized by <i>Fortran</i> 2003 ¹
27	openo	invitivi opeemeuton s	e mi i unough the	ionnan e meroperaonny mitany standardized by forman 2005.
28				
29				
30				
31				
32				
34				
35				
36				
37				
38				
39				
40				
41				
43				
44				
45				
46				
47				
48	1			

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*.

132

2.11

1

2

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

Annex G

Changes to this Document

Version 1.5 1

Major changes in OpenSHMEM 1.5 include ...

The following list describes the specific changes in OpenSHMEM 1.5:

• This item is a template for changelist entries and should be deleted before this document is published. See Annex G.

Version 1.4 2

Major changes in OpenSHMEM 1.4 include multithreading support, *contexts* for communication management, *shmem_sync*, ²⁶ 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.

> 15 16

> 17

18 19

20

21 22 23

24

27

28

29

30

31

32 33

34

35

36

37

38 39

40

41 42

43

44

45

47

48

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 9.5.
- New API shmem_sync_all and shmem_sync to provide PE synchronization without completing pending communication operations. See Sections 9.9.3 and 9.9.4.
- · Clarified that the OpenSHMEM extensions header files are required, even when empty. See Section 5.
- Clarified that the SHMEM_GET64 and SHMEM_GET64_NBI routines are included in the Fortran language bindings. See Sections 9.6.4 and 9.7.2.
- Clarified that *shmem init* must be matched with a call to *shmem finalize*. See Sections 9.1.1 and 9.1.4.
- Added the SHMEM_SYNC_SIZE constant. See Section 6.
- Added type-generic interfaces for *shmem_wait_until*. See Section 9.10.1.

1 2 3 4	• Removed the <i>volatile</i> qualifiers from the <i>ivar</i> arguments to <i>shmem_wait</i> routines and the <i>lock</i> arguments in the lock API. <i>Rationale: Volatile qualifiers were added to several API routines in OpenSHMEM 1.3; however, they were later found to be unnecessary.</i> See Sections 9.10.1 and 9.12.1.
5 6	 Deprecated the SMA_* environment variables and added equivalent SHMEM_* environment variables. See Section 8.
7 8 9	• Added the <i>C11_Noreturn</i> function specifier to <i>shmem_global_exit</i> . See Section 9.1.5.
10 11	• Clarified ordering semantics of memory ordering, point-to-point synchronization, and collective synchronization routines.
12 13	 Clarified deprecation overview and added deprecation rationale in Annex F. See Section F.
15 16	• Deprecated header directory <i>mpp</i> . See Section F.
17 18	 Deprecated the <i>shmem_wait</i> functions and the <i>long</i>-typed C/C++ <i>shmem_wait_until</i> function. See Section 9.10.
19 20 21	• Added the <i>shmem_test</i> functions. See Section 9.10.
22 23	• Added the <i>shmem_calloc</i> function. See Section 9.3.2.
24 25	• Introduced the thread safe semantics that define the interaction between OpenSHMEM routines and user threads. See Section 9.2.
26 27 28 29	 Added the new routine <i>shmem_init_thread</i> to initialize the OpenSHMEM library with one of the defined thread levels. See Section 9.2.1.
30 31 32	 Added the new routine <i>shmem_query_thread</i> to query the thread level provided by the OpenSHMEM implementation. See Section 9.2.2.
33 34	 Clarified the semantics of <i>shmem_quiet</i> for a multithreaded OpenSHMEM PE. See Section 9.11.2
35 36 37	 Revised the description of <i>shmem_barrier_all</i> for a multithreaded OpenSHMEM PE. See Section 9.9.1
38 39	 Revised the description of <i>shmem_wait</i> for a multithreaded OpenSHMEM PE. See Section 9.10.1
40 41	• Clarified description for <i>SHMEM_VENDOR_STRING</i> . See Section 6.
42 43 44	• Clarified description for <i>SHMEM_MAX_NAME_LEN</i> . See Section 6.
45 46	• Clarified API description for <i>shmem_info_get_name</i> . See Section 9.1.10.
47 48	• Expanded the type support for RMA, AMO, and point-to-point synchronization operations. See Tables 3, 4, 5, and 7

- Renamed AMO operations to use *shmem_atomic_** prefix and deprecated old AMO routines. See Section 9.8.
- Added fetching and non-fetching bitwise AND, OR, and XOR atomic operations. See Section 9.8.
- 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 9.9.9.
- Clarified that complex-typed reductions in C are optionally supported. See Section 9.9.9.

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:

20 • Clarified implementation of PEs as threads. 21 22 • Added *const* to every read-only pointer argument. 23 24 • Clarified definition of Fence. See Section 2. 25 26 • Clarified implementation of symmetric memory allocation. 27 See Section 3. 28 29 • Restricted atomic operation guarantees to other atomic operations with the same datatype. See Section 3.1. 31 • Deprecation of all constants that start with _SHMEM_*. 32 See Section 6. 33 34 • Added a type-generic interface to OpenSHMEM RMA and AMO operations based on C11 Generics. 35 See Sections 9.6, 9.7 and 9.8. 36 New nonblocking variants of remote memory access, SHMEM_PUT_NBI and SHMEM_GET_NBI. 37 See Sections 9.7.1 and 9.7.2. 38 39 • New atomic elemental read and write operations, SHMEM FETCH and SHMEM SET. 40 See Sections 9.8.1 and 9.8.2 41 42 • New alltoall data exchange operations, SHMEM_ALLTOALL and SHMEM_ALLTOALLS. 43 See Sections 9.9.10 and 9.9.11. 44 • Added volatile to remotely accessible pointer argument in SHMEM_WAIT and SHMEM_LOCK. 45 See Sections 9.10.1 and 9.12.1. 47 • Deprecation of SHMEM_CACHE. 48 See Section 9.13.1.

10 11

12

13 14

> 15 16

17

18

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 4.
 - New library constants to support API that query version and name information. See Section 6.
 - New API *shmem_init* to provide mechanism to start an OpenSHMEM program and replace deprecated *start_pes*. See Section 9.1.1.
 - Deprecation of _*my_pe* and _*num_pes* routines. See Sections 9.1.2 and 9.1.3.
 - New API *shmem_finalize* to provide collective mechanism to cleanly exit an OpenSHMEM program and release resources.
 See Section 9.1.4.
 - New API *shmem_global_exit* to provide mechanism to exit an OpenSHMEM program. See Section 9.1.5.
 - Clarification related to the address of the referenced object in *shmem_ptr*. See Section 9.1.8.
 - New API to query the version and name information. See Section 9.1.9 and 9.1.10.
 - OpenSHMEM library API normalization. All *C* symmetric memory management API begins with *shmem_*. See Section 9.3.1.
 - Notes and clarifications added to *shmem_malloc*. See Section 9.3.1.
 - Deprecation of *Fortran* API routine *SHMEM_PUT*. See Section 9.6.1.
 - Clarification related to *shmem_wait*. See Section 9.10.1.
 - 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 Open-SHMEM Specification.
- See Annex F.

10 11

12

13

14

15

17

18

20 21

22

23

24 25

26

27

28

29

31

32 33

34

35

36

37 38

39

40 41

42

43

44

45 46

47

48

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 readability 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 9.11.
- Clarification of the completion semantics of memory load and store operations in context of *shmem_barrier_all* and *shmem_barrier* routines. See Section 9.9.1 and 9.9.2.
- Clarification of the completion and ordering semantics of *shmem_quiet* and *shmem_fence*. See Section 9.11.2 and 9.11.1.
- Clarifications of the completion semantics of RMA and AMO routines. See Sections 9.6 and 9.8
- Clarifications of the memory model and the memory alignment requirements for symmetric data objects. See Section 3.
- Clarification of the execution model and the definition of a PE. See Section 4
- Clarifications of the semantics of *shmem_pe_accessible* and *shmem_addr_accessible*. See Section 9.1.6 and 9.1.7.
- 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 6 and 9.10.1.
- Added API calls: *shmem_char_p*, *shmem_char_g*. See Sections 9.6.2 and 9.6.5.
- Removed API calls: *shmem_char_put, shmem_char_get*. See Sections 9.6.1 and 9.6.4.
- The usage of *ptrdiff_t*, *size_t*, and *int* in the interface signature was made consistent with the description. See Sections 9.9, 9.6.3, and 9.6.6.
- Revised *shmem_barrier* example. See Section 9.9.2.
- Clarification of the initial value of *pSync* work arrays for *shmem_barrier*. See Section 9.9.2.
- Clarification of the expected behavior when multiple *start_pes* calls are encountered. See Section 9.1.11.
- Corrected the definition of atomic increment operation. See Section 9.8.6.
- Clarification of the size of the symmetric heap and when it is set. See Section 9.3.1.

10

11

12

13

14 15

16

17

18

19

20

21 22

23

24

25

26

27

29

31

33

34

35

36

37

38

39 40

41

42

43

44

45

47

1 2	• Clarification of the integer and real sizes for <i>Fortran</i> API. See Sections 9.8.8, 9.8.3, 9.8.4, 9.8.5, 9.8.6, and 9.8.7.
3	• Clarification of the expected behavior on program <i>exit</i> . See Section 4, Execution Model.
6 7	 More detailed description for the progress of OpenSHMEM operations provided. See Section 4.1.
8 9	• Clarification of naming convention for non-standard interfaces and their inclusion in <i>shmemx.h.</i> See Section 5.
10	• Various fixes to OpenSHMEM code examples across the Specification to include appropriate header files.
12 13 14	 Removing requirement that implementations should detect size mismatch and return error information for <i>shmalloc</i> and ensuring consistent language. See Sections 9.3.1 and Annex C.
15 16	• <i>Fortran</i> programming fixes for examples. See Sections 9.9.9 and 9.10.1.
17 18 19	• Clarifications of the reuse <i>pSync</i> and <i>pWork</i> across collectives. See Sections 9.9, 9.9.6, 9.9.7 and 9.9.9.
20 21	Name changes for UV and ICE for SGI systems. See Annex E.
22	
23	
24	
26	
27	
28	
29	
31	
32	
33	
34	
35	
36	
37	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
Index

_SHMEM_BARRIER_SYNC_SIZE, 7, 129 _SHMEM_BCAST_SYNC_SIZE, 7, 129 _SHMEM_CMP_EQ, 10, 129 _SHMEM_CMP_GE, 11, 130 _SHMEM_CMP_GT, 11, 130 _SHMEM_CMP_LE, 10, 129 _SHMEM_CMP_LT, 10, 129 _SHMEM_CMP_NE, 10, 129 _SHMEM_COLLECT_SYNC_SIZE, 8, 129 _SHMEM_MAJOR_VERSION, 9, 129 _SHMEM_MAX_NAME_LEN, 9, 129 _SHMEM_MINOR_VERSION, 9, 129 _SHMEM_REDUCE_MIN_WRKDATA_SIZE, 8, 129 _SHMEM_REDUCE_SYNC_SIZE, 7, 129 _SHMEM_SYNC_VALUE, 6, 129 SHMEM VENDOR STRING, 9, 129 _my_pe, 129 _num_pes, 129

Bitwise AMO Types and Names, 63

Constants, 5

Deprecated API, 129

Environment Variables, 12 Extended AMO Types and Names, 62

Handles, 11

Library Constants, 5 Library Handles, 11

MY_PE, 129

NUM_PES, 129

Point-to-Point Comparison Constants, 108 Point-to-Point Synchronization Types and Names, 107

shfree, 129
shmalloc, 129
shmem_<TYPENAME>_add, 74, 130
shmem_<TYPENAME>_atomic_add, 73
shmem_<TYPENAME>_atomic_and, 76
shmem_<TYPENAME>_atomic_compare_swap, 65
shmem_<TYPENAME>_atomic_fetch, 61

shmem_<TYPENAME>_atomic_fetch_add, 72 shmem_<TYPENAME>_atomic_fetch_and, 75 shmem_<TYPENAME>_atomic_fetch_inc, 68 shmem <TYPENAME> atomic fetch or, 77 shmem_<TYPENAME>_atomic_fetch_xor, 78 shmem **<TYPENAME>** atomic inc, 70 shmem_<TYPENAME>_atomic_or, 77 shmem_<TYPENAME>_atomic_set, 64 shmem **<TYPENAME>** atomic swap, 66 shmem **<TYPENAME>** atomic xor, 79 shmem_<TYPENAME>_cswap, 65, 130 shmem_<TYPENAME>_fadd, 72, 130 shmem_<TYPENAME>_fetch, 62, 130 shmem_<TYPENAME>_finc, 69, 130 shmem_<TYPENAME>_g, 54 shmem <TYPENAME> get, 52 shmem_<TYPENAME>_get_nbi, 59 shmem_<TYPENAME>_iget, 55 shmem <TYPENAME> inc, 70, 130 shmem_<TYPENAME>_iput, 50 shmem_<TYPENAME>_p, 49 shmem_<TYPENAME>_put, 46 shmem **<TYPENAME>** put nbi, 57 shmem **<TYPENAME>** set, 64, 130 shmem_<TYPENAME>_swap, 67, 130 shmem_<TYPENAME>_test, 110 shmem_<TYPENAME>_wait, 108, 130 shmem_<TYPENAME>_wait_until, 108 shmem add, 73, 130 SHMEM ADDR ACCESSIBLE, 19 shmem_addr_accessible, 19 shmem_align, 27 SHMEM_ALLTOALL32, 102 shmem alltoall32, 101 SHMEM ALLTOALL64, 102 shmem_alltoall64, 102 SHMEM_ALLTOALL_SYNC_SIZE, 8 SHMEM_ALLTOALLS32, 104 shmem_alltoalls32, 104 SHMEM_ALLTOALLS64, 104 shmem alltoalls64, 104 SHMEM_ALLTOALLS_SYNC_SIZE, 8 shmem_atomic_add, 73 shmem_atomic_and, 76 shmem_atomic_compare_swap, 65

shmem_atomic_fetch, 61 shmem atomic fetch add, 72 2 shmem_atomic_fetch_and, 75 shmem atomic fetch inc, 68 shmem_atomic_fetch_or, 76 shmem_atomic_fetch_xor, 78 shmem atomic inc, 70 shmem_atomic_or, 77 shmem_atomic_set, 63 9 shmem_atomic_swap, 66 shmem_atomic_xor, 79 10 SHMEM BARRIER, 82 11 shmem barrier, 82 12 SHMEM BARRIER ALL, 80 13 shmem_barrier_all, 80 14 SHMEM_BARRIER_SYNC_SIZE, 7 15 SHMEM_BCAST_SYNC_SIZE, 7 16 SHMEM_BROADCAST32, 87 17 shmem_broadcast32, 87 SHMEM_BROADCAST4, 87 18 SHMEM BROADCAST64, 87 19 shmem broadcast64, 87 20 SHMEM BROADCAST8, 87 21 shmem_calloc, 28 22 SHMEM_CHARACTER_GET, 52 23 SHMEM_CHARACTER_GET_NBI, 59 24 SHMEM_CHARACTER_PUT, 47 25 SHMEM_CHARACTER_PUT_NBI, 57 SHMEM CLEAR CACHE INV, 119, 129 26 shmem_clear_cache_inv, 119, 129 27 shmem_clear_cache_line_inv, 119, 129 28 SHMEM CLEAR LOCK, 118 29 shmem clear lock, 118 30 SHMEM_CMP_EQ, 10, 108 31 SHMEM_CMP_GE, 11, 108 32 SHMEM_CMP_GT, 11, 108 33 SHMEM_CMP_LE, 10, 108 34 SHMEM CMP LT, 10, 108 SHMEM_CMP_NE, 10, 108 35 SHMEM COLLECT32, 90 36 shmem_collect32, 90 37 SHMEM_COLLECT4, 90 38 SHMEM_COLLECT64, 90 39 shmem_collect64, 90 40 SHMEM_COLLECT8, 90 41 SHMEM_COLLECT_SYNC_SIZE, 8 42 SHMEM COMP4 PROD TO ALL, 96 SHMEM COMP4 SUM TO ALL, 95 43 SHMEM COMP8 PROD TO ALL, 96 44 SHMEM_COMP8_SUM_TO_ALL, 95 45 SHMEM_COMPLEX_GET, 52 46 SHMEM_COMPLEX_GET_NBI, 59 47 SHMEM_COMPLEX_IGET, 55 48 SHMEM_COMPLEX_IPUT, 50

SHMEM_COMPLEX_PUT, 47 SHMEM COMPLEX PUT NBI, 57 shmem_complexd_prod_to_all, 96 shmem_complexd_sum_to_all, 95 shmem_complexf_prod_to_all, 96 shmem_complexf_sum_to_all, 95 shmem cswap, 65, 130 shmem_ctx_<TYPENAME>_atomic_add, 73 shmem_ctx_<TYPENAME>_atomic_and, 76 shmem_ctx_<TYPENAME>_atomic_compare_swap, 65 shmem_ctx_<TYPENAME>_atomic_fetch, 61 shmem ctx <TYPENAME> atomic fetch add, 72 shmem_ctx_<TYPENAME>_atomic_fetch_and, 75 shmem ctx **<TYPENAME>** atomic fetch inc, 68 shmem_ctx_<TYPENAME>_atomic_fetch_or, 77 shmem_ctx_<TYPENAME>_atomic_fetch_xor, 78 shmem_ctx_<**TYPENAME**>_atomic_inc, 70 shmem_ctx_<TYPENAME>_atomic_or, 77 shmem_ctx_<TYPENAME>_atomic_set, 64 shmem_ctx_<TYPENAME>_atomic_swap, 66 shmem_ctx_<TYPENAME>_atomic_xor, 79 shmem_ctx_<TYPENAME>_g, 54 shmem_ctx_<TYPENAME>_get, 52 shmem_ctx_<TYPENAME>_get_nbi, 59 shmem_ctx_<TYPENAME>_iget, 55 shmem_ctx_<TYPENAME>_iput, 50 shmem_ctx_<TYPENAME>_p, 49 shmem_ctx_<TYPENAME>_put, 46 shmem_ctx_<TYPENAME>_put_nbi, 57 shmem_ctx_create, 40 SHMEM_CTX_DEFAULT, 11, 40 shmem ctx destroy, 42 shmem ctx fence, 111 shmem_ctx_get<SIZE>, 52 shmem_ctx_get<SIZE>_nbi, 59 shmem_ctx_get_team, 45 shmem_ctx_getmem, 52 shmem ctx getmem nbi, 59 shmem_ctx_iget<SIZE>, 55 shmem ctx iput<SIZE>, 50 SHMEM_CTX_NOSTORE, 6, 41 SHMEM_CTX_PRIVATE, 6, 41 shmem_ctx_put<SIZE>, 47 shmem_ctx_put<SIZE>_nbi, 57 shmem_ctx_putmem, 47 shmem_ctx_putmem_nbi, 57 shmem ctx quiet, 113 SHMEM CTX SERIALIZED, 6, 41 SHMEM DEBUG, 12 SHMEM_DOUBLE_GET, 52 SHMEM_DOUBLE_GET_NBI, 59 SHMEM_DOUBLE_IGET, 55 SHMEM_DOUBLE_IPUT, 50 shmem_double_max_to_all, 94

INDEX

shmem_double_min_to_all, 95 shmem double prod to all, 96 SHMEM DOUBLE PUT, 47 SHMEM_DOUBLE_PUT_NBI, 58 shmem_double_sum_to_all, 95 shmem_fadd, 72, 130 SHMEM FCOLLECT32, 90 shmem_fcollect32, 90 SHMEM_FCOLLECT4, 90 SHMEM_FCOLLECT64, 90 shmem_fcollect64, 90 SHMEM FCOLLECT8, 90 SHMEM_FENCE, 111 shmem fence, 111 shmem_fetch, 62, 130 SHMEM_FINALIZE, 16 shmem_finalize, 16 shmem_finc, 68, 130 shmem_float_max_to_all, 94 shmem_float_min_to_all, 95 shmem_float_prod_to_all, 96 shmem float sum to all, 95 shmem free, 27 shmem_g, 54 shmem_get, 52 SHMEM_GET128, 52 SHMEM_GET128_NBI, 59 SHMEM GET32, 52 SHMEM GET32 NBI, 59 SHMEM_GET4, 52 SHMEM_GET4_NBI, 59 SHMEM GET64, 52 SHMEM GET64 NBI, 59 SHMEM_GET8, 52 SHMEM_GET8_NBI, 59 shmem_get<SIZE>, 52 shmem_get<SIZE>_nbi, 59 shmem get nbi, 59 SHMEM_GETMEM, 52 shmem getmem, 52 SHMEM_GETMEM_NBI, 59 shmem_getmem_nbi, 59 SHMEM_GLOBAL_EXIT, 17 shmem_global_exit, 17 shmem_iget, 55 SHMEM_IGET128, 56 SHMEM IGET32, 55 SHMEM IGET4, 55 SHMEM IGET64, 56 SHMEM_IGET8, 55 shmem_iget<SIZE>, 55 shmem_inc, 70, 130 SHMEM_INFO, 12 SHMEM_INFO_GET_NAME, 22

shmem_info_get_name, 22 SHMEM INFO GET VERSION, 22 shmem_info_get_version, 22 SHMEM INIT, 13 shmem_init, 13 shmem_init_thread, 25 SHMEM INT4 ADD, 74 SHMEM_INT4_AND_TO_ALL, 94 SHMEM_INT4_CSWAP, 65 SHMEM_INT4_FADD, 72 SHMEM_INT4_FETCH, 62 10 SHMEM INT4 FINC, 69 11 SHMEM_INT4_INC, 70 12 SHMEM INT4 MAX TO ALL, 94 13 SHMEM_INT4_MIN_TO_ALL, 95 14 SHMEM_INT4_OR_TO_ALL, 97 15 SHMEM_INT4_PROD_TO_ALL, 96 16 SHMEM_INT4_SET, 64 17 SHMEM_INT4_SUM_TO_ALL, 96 SHMEM_INT4_SWAP, 67 18 SHMEM INT4 WAIT, 108 19 SHMEM INT4 WAIT UNTIL, 108 20 SHMEM_INT4_XOR_TO_ALL, 97 21 SHMEM_INT8_ADD, 74 22 SHMEM_INT8_AND_TO_ALL, 94 23 SHMEM_INT8_CSWAP, 65 24 SHMEM_INT8_FADD, 72 25 SHMEM_INT8_FETCH, 62 SHMEM_INT8_FINC, 69 26 SHMEM_INT8_INC, 70 27 SHMEM_INT8_MAX_TO_ALL, 94 28 SHMEM_INT8_MIN_TO_ALL, 95 29 SHMEM INT8 OR TO ALL, 97 30 SHMEM_INT8_PROD_TO_ALL, 96 31 SHMEM_INT8_SET, 64 32 SHMEM_INT8_SUM_TO_ALL, 96 33 SHMEM_INT8_SWAP, 67 SHMEM INT8 WAIT, 108 34 SHMEM_INT8_WAIT_UNTIL, 108 35 SHMEM_INT8_XOR_TO_ALL, 97 36 shmem_int_and_to_all, 94 37 shmem_int_max_to_all, 94 38 shmem_int_min_to_all, 95 39 shmem_int_or_to_all, 97 40 shmem_int_prod_to_all, 96 41 shmem_int_sum_to_all, 95 shmem int xor to all, 97 42 SHMEM INTEGER GET, 52 43 SHMEM INTEGER GET NBI, 59 44 SHMEM_INTEGER_IGET, 56 45 SHMEM_INTEGER_IPUT, 50 46 SHMEM_INTEGER_PUT, 47 47 SHMEM_INTEGER_PUT_NBI, 58 48 shmem_iput, 50

SHMEM_IPUT128, 50 SHMEM IPUT32, 50 2 SHMEM IPUT4, 50 SHMEM IPUT64, 50 SHMEM_IPUT8, 50 shmem_iput<SIZE>, 50 SHMEM LOGICAL GET, 52 SHMEM_LOGICAL_GET_NBI, 59 SHMEM_LOGICAL_IGET, 56 9 SHMEM_LOGICAL_IPUT, 50 SHMEM_LOGICAL_PUT, 47 10 SHMEM LOGICAL PUT NBI, 58 11 shmem_long_and_to_all, 94 12 shmem_long_max_to_all, 94 13 shmem_long_min_to_all, 95 14 shmem_long_or_to_all, 97 15 shmem_long_prod_to_all, 96 16 shmem_long_sum_to_all, 95 17 shmem_long_xor_to_all, 97 shmem_longdouble_max_to_all, 94 18 shmem_longdouble_min_to_all, 95 19 shmem longdouble prod to all, 96 20 shmem_longdouble_sum_to_all, 95 21 shmem_longlong_and_to_all, 94 22 shmem_longlong_max_to_all, 94 23 shmem_longlong_min_to_all, 95 24 shmem_longlong_or_to_all, 97 25 shmem_longlong_prod_to_all, 96 shmem_longlong_sum_to_all, 95 26 shmem_longlong_xor_to_all, 97 27 SHMEM_MAJOR_VERSION, 9 28 shmem malloc, 27 29 SHMEM MAX NAME LEN, 9 30 SHMEM_MINOR_VERSION, 9 31 SHMEM_MY_PE, 14 32 shmem_my_pe, 14 33 SHMEM_N_PES, 15 34 shmem n pes, 15 shmem_p, 48 35 shmem_pe_accessible, 18 36 SHMEM_PTR, 20 37 shmem_ptr, 20 38 SHMEM_PUT, 129 39 shmem_put, 46 40 SHMEM_PUT128, 47 41 SHMEM_PUT128_NBI, 58 42 SHMEM PUT32, 47 SHMEM PUT32 NBI, 58 43 SHMEM PUT4, 47 44 SHMEM PUT4 NBI, 58 45 SHMEM_PUT64, 47 46 SHMEM_PUT64_NBI, 58 47 SHMEM_PUT8, 47 48 SHMEM_PUT8_NBI, 58

shmem_put<SIZE>, 47 shmem put<SIZE> nbi, 57 shmem_put_nbi, 57 SHMEM PUTMEM, 47 shmem_putmem, 47 SHMEM_PUTMEM_NBI, 58 shmem putmem nbi, 57 shmem_query_thread, 26 SHMEM_QUIET, 113 shmem_quiet, 113 SHMEM_REAL16_MAX_TO_ALL, 94 SHMEM REAL16 MIN TO ALL, 95 SHMEM_REAL16_PROD_TO_ALL, 96 SHMEM REAL16 SUM TO ALL, 96 SHMEM_REAL4_FETCH, 62 SHMEM_REAL4_MAX_TO_ALL, 94 SHMEM_REAL4_MIN_TO_ALL, 95 SHMEM_REAL4_PROD_TO_ALL, 96 SHMEM_REAL4_SET, 64 SHMEM_REAL4_SUM_TO_ALL, 96 SHMEM_REAL4_SWAP, 67 SHMEM REAL8 FETCH, 63 SHMEM_REAL8_MAX_TO_ALL, 94 SHMEM_REAL8_MIN_TO_ALL, 95 SHMEM_REAL8_PROD_TO_ALL, 96 SHMEM_REAL8_SET, 64 SHMEM_REAL8_SUM_TO_ALL, 96 SHMEM_REAL8_SWAP, 67 SHMEM REAL GET, 52 SHMEM_REAL_GET_NBI, 59 SHMEM_REAL_IGET, 56 SHMEM REAL IPUT, 50 SHMEM REAL PUT, 47 SHMEM_REAL_PUT_NBI, 58 shmem_realloc, 27 SHMEM_REDUCE_MIN_WRKDATA_SIZE, 8 SHMEM_REDUCE_SYNC_SIZE, 7 shmem set, 64, 130 SHMEM_SET_CACHE_INV, 119, 129 shmem set cache inv, 119, 129 SHMEM_SET_CACHE_LINE_INV, 119, 129 shmem_set_cache_line_inv, 119, 129 SHMEM_SET_LOCK, 118 shmem_set_lock, 118 shmem_short_and_to_all, 94 shmem_short_max_to_all, 94 shmem short min to all, 94 shmem short or to all, 97 shmem_short_prod_to_all, 96 shmem_short_sum_to_all, 95 shmem_short_xor_to_all, 97 SHMEM_SWAP, 67 shmem_swap, 67, 130 SHMEM_SYMMETRIC_SIZE, 12

shmem_sync, 84 shmem sync all, 83 SHMEM_SYNC_SIZE, 7 SHMEM_SYNC_VALUE, 6 shmem_team_broadcast32, 86 shmem_team_broadcast64, 86 shmem_team_collect32, 92 shmem_team_collect64, 92 shmem_team_create_ctx, 41 shmem_team_destroy, 39 shmem_team_fcollect32, 92 shmem team fcollect64, 92 shmem_team_get_config, 34 SHMEM_TEAM_LOCAL_LIMIT, 37, 38 shmem_team_my_pe, 32 shmem_team_n_pes, 33 SHMEM_TEAM_NOCOLLECTIVE, 6, 35, 37, 38, 86, 93 SHMEM_TEAM_NODE, 11, 36 SHMEM_TEAM_NULL, 6, 32, 37, 39, 86, 93 SHMEM_TEAM_NUM_THREADS, 37, 38 shmem_team_split_2d, 37 shmem_team_split_strided, 36 shmem_team_sync, 84 shmem_team_translate_pe, 35 SHMEM_TEAM_WORLD, 11, 32, 33, 35, 36, 38, 40 shmem_test, 110 SHMEM_TEST_LOCK, 118 shmem_test_lock, 118 SHMEM_THREAD_FUNNELED, 6, 24 SHMEM_THREAD_MULTIPLE, 6, 24 SHMEM_THREAD_SERIALIZED, 6, 24 SHMEM THREAD SINGLE, 5, 24 SHMEM_UDCFLUSH, 119, 129 shmem_udcflush, 119, 129 SHMEM_UDCFLUSH_LINE, 119, 129 shmem_udcflush_line, 119, 129 SHMEM_VENDOR_STRING, 9 SHMEM_VERSION, 12 SHMEM_WAIT, 108 shmem_wait, 108, 130 SHMEM_WAIT_UNTIL, 108 shmem_wait_until, 108, 130 shmemalign, 129 SHPALLOC, 29 SHPCLMOVE, 30 SHPDEALLC, 31 shrealloc, 129 SMA DEBUG, 130 SMA_INFO, 130 SMA_SYMMETRIC_SIZE, 130 SMA_VERSION, 130 Standard AMO Types and Names, 62 Standard RMA Types and Names, 46

START_PES, 23, 129	1
start_pes, 23, 129	2
Tables	3
Bitwise AMO Types and Names, 63	4
Constants, 5	6
Deprecated API, 129 Environment Variables 12	7
Extended AMO Types and Names, 62	8
Handles, 11	9
Library Constants, 5	10
Library Handles, 11	11
Point-to-Point Comparison Constants, 108	12
Standard AMO Types and Names 62	13
Standard RMA Types and Names, 46	14
	15
	16
	17
	18
	19
	20
	21
	22
	23
	24
	25
	20
	28
	29
	30
	31
	32
	33
	34
	35
	36
	37
	38
	39
	40
	41
	42
	43
	45
	46
	47
	48