

# **SC14 OpenSHMEM TUTORIAL**

Presenters: Tony Curtis, Deepak Eachempati, Dounia Khaldi, Aaron Welch

University of Houston, Texas

Oscar Hernandez, Pavel Shamis, Manju Gorentla Venkata

Oak Ridge National Laboratory

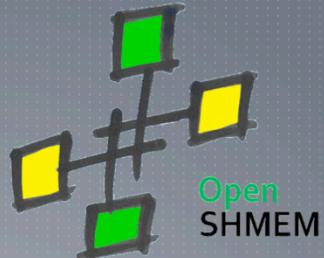
1

# **OpenSHMEM WORKSHOP**

## **OUTLINE**

- Prerequisites
- Background
- Concepts
- History and Implementations
- The OpenSHMEM Project
- OpenSHMEM routines
- OpenSHMEM and Hardware
- Developing OpenSHMEM Applications
- OpenSHMEM Implementations
- OpenSHMEM: The Future...

<http://www.openshmem.org/>



## OpenSHMEM PREREQUISITES

- Knowledge of C/Fortran
- Familiarity with parallel computing
- Linux/UNIX command-line
- Useful for hands-on
  - 64-bit Linux (native, VM or remote)
    - E.g. Fedora, CentOS, Ubuntu, ...
  - Installation of GASNet, “fast” segment configuration preferable
    - <http://gasnet.lbl.gov/>
  - OpenSHMEM download, test-suite & demo programs
    - <https://github.com/openshmem-org/openshmem>

3

These are the platform and software requirements (some of them, anyway) for installing the reference implementation. Precise package requirements are different on different distributions.

## **OpenSHMEM BACKGROUND (I)**

- ▶ Large applications require lots of compute power
- ▶ Various approaches to providing this
  - ▶ Mainframe
  - ▶ SMP
  - ▶ Cluster
- ▶ All involve
  - ▶ Multiple things happening at once
  - ▶ ...Which needs...
  - ▶ Programming methods to
    - ▶ Express this
    - ▶ Take advantage of systems

4

Talk about the setting for parallel computing and programming. There are different approaches (libraries/languages) to expressing parallelism.

## **OpenSHMEM BACKGROUND (2)**

- ▶ 2 main software paradigms
  - ▶ Threaded
  - ▶ Message-passing
- ▶ 2 main hardware paradigms
  - ▶ Single-image multiprocessor (SMP)
  - ▶ Distributed
    - ▶ Multiple machines with separate OS
    - ▶ Connected together

5

Talk about the different ways of getting parallel systems. The narrative is heading towards the 2<sup>nd</sup> point as being more interesting for OpenSHMEM.

## **OpenSHMEM BACKGROUND (3)**

- ▶ Programming environments provide abstraction
- ▶ In particular
  - ▶ A language or library can be used on many machine types
  - ▶ Implementation hides differences & leverages features
- ▶ 2 dominant models
  - ▶ MPI
  - ▶ OpenMP
- ▶ First, a little background for context...

6

## **OpenSHMEM BACKGROUND (4)**

- ▶ Concurrent
  - ▶ Multiple things logically happen at once
  - ▶ May be emulated
    - ▶ E.g. time slicing on shared machine
- ▶ Parallel
  - ▶ = Concurrent +
  - ▶ Things really happen independently
  - ▶ On separate processors
- ▶ Work is partitioned in some way across resources

7

## **OpenSHMEM BACKGROUND (5)**

- ▶ Different ways of partitioning work
  - ▶ different tasks \* different data
- ▶ MPMD = multiple program, multiple data
- ▶ SPMD = single program, multiple data
- ▶ SPSD = single program, single data
  - ▶ Just good old sequential

<http://en.wikipedia.org/wiki/SPMD>

8

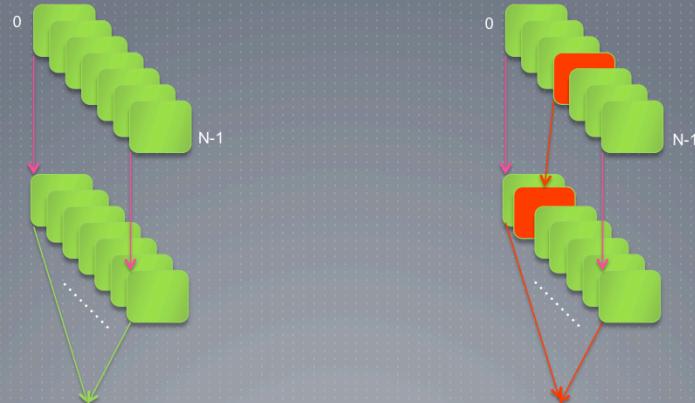
SPMD is the OpenSHMEM model. Diagram coming in a couple of slides.

## **OpenSHMEM BACKGROUND (6)**

- ▶ SPMD
  - ▶ Program launches many processes
  - ▶ Each starts with same code (SP)
  - ▶ And then typically operates on some specific part of the data (MD)
  - ▶ Processes may then communicate with each other
    - ▶ Share common data
    - ▶ Broadcast work
    - ▶ Collect results
- ▶ PVM and MPI are well-known examples
- ▶ OpenSHMEM is SPMD

9

## OpenSHMEM BACKGROUND (7)



10

The execution model has multiple instances of the same program running independently of each other, \*until\* they need to communicate and later synchronize to make sure variables have been fully updated. Synchronization introduces ordering points. It also can be a huge overhead...

## **OpenSHMEM BACKGROUND (8)**

- ▶ Address Spaces
  - ▶ Global vs. distributed
  - ▶ OpenMP has global (shared) space
  - ▶ MPI has partitioned space
    - ▶ Private data exchanged via messages
  - ▶ OpenSHMEM is “partitioned global address space”
    - ▶ PGAS
  - ▶ Has private *and* shared data
    - ▶ Shared data accessible directly by other processes

11

Where data lives in OpenSHMEM programs and how it is viewed across a program's memory space. Lead into the ecosystem of PGAS programming models.

## **OpenSHMEM BACKGROUND (9)**

- ▶ The PGAS family
  - ▶ Libraries include...
    - ▶ GASNet
    - ▶ ARMCI / Global Arrays
    - ▶ CCI
    - ▶ GASPI/GPI
  - ▶ Languages include...
    - ▶ Chapel
    - ▶ Titanium
    - ▶ X10
    - ▶ UPC
    - ▶ CAF
    - ▶ (Often built on these libraries)

12

## OpenSHMEM BACKGROUND (10)

- ▶ OpenSHMEM is SPMD parallel programming library
  - ▶ Library of functions similar in feel to using MPI (e.g. `shmemp_get()`)
- ▶ Available for C, C++ and Fortran
- ▶ Used for programs that
  - perform computations in separate address spaces and
  - explicitly pass data to and from different processes in the program.
- ▶ The processes participating in shared memory applications are referred to as processing elements (PEs).
- ▶ OpenSHMEM routines supply remote data transfer, work-shared broadcast and reduction, barrier synchronization, and atomic memory operations.

13

By “similar to MPI” we don’t mean OpenSHMEM is necessarily going to be a drop-in replacement, because the 1-sided nature of the calls and the way in which memory space is presented is different. But it’s the same idea of taking an existing language and adding library calls to get (distributed) parallelism.

## OpenSHMEM CONCEPTS (I)

### ▶ Symmetric Variables

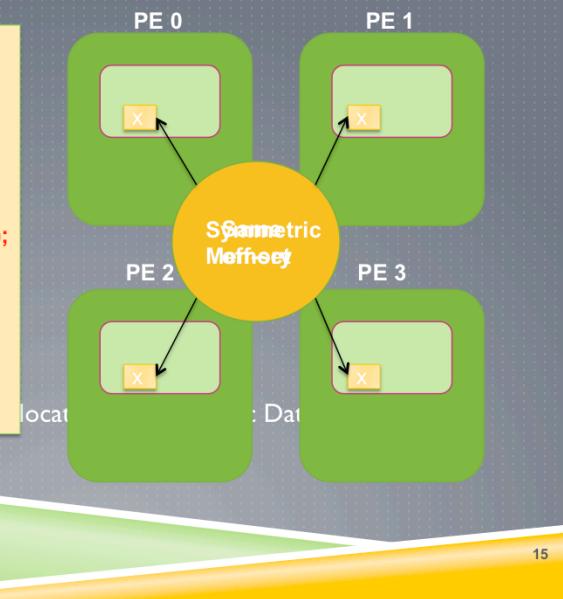
- ▶ Arrays or variables that exist with the **same size, type, and relative address** on all PEs.
- ▶ The following kinds of data objects are symmetric:
  - ▶ Fortran data objects in common blocks or with the **SAVE** attribute.
  - ▶ Non-stack C and C++ variables.
  - ▶ Fortran arrays allocated with **shmalloc**
  - ▶ C and C++ data allocated by **shmalloc**

14

**The local existence of a corresponding symmetric object implies that a data object is remotely accessible.**

## OPENSHMEM CONCEPTS (2)

```
#include <shmem.h>
int main (void)
{
    int *x;
    ...
    start_pes(0);
    ...
    x = (int*) shmalloc(sizeof(x));
    ...
    shmem_barrier_all();
    ...
    shfree(x);
    return 0;
}
```



Here's a snippet from a simple program that does execution-time dynamic allocation of memory. Full slide has an animation showing the progression of the program and memory being allocated. Need to talk about what symmetric means.

## OpenSHMEM HISTORY AND IMPLEMENTATIONS

- ▶ **Cray**
  - ▶ SHMEM first introduced by Cray Research Inc. in 1993 for Cray T3D
  - ▶ Platforms: Cray T3D, T3E, PVP, XT series
- ▶ **SGI**
  - ▶ Owns the “rights” for SHMEM
  - ▶ Baseline for OpenSHMEM development (Altix)
- ▶ **Quadrics** (company out of business)
  - ▶ Optimized API for QsNet
  - ▶ Platform: Linux cluster with QsNet interconnect
- ▶ **Others**
  - ▶ HP SHMEM, IBM SHMEM
  - ▶ GPSHMEM (cluster with ARMCI & MPI support, old)

Note: SHMEM was not defined by any one standard.

16

Read up on the Cray T3D.

## OpenSHMEM DIVERGENT IMPLEMENTATIONS (I)

- ▶ Many forms of initialization
  - ▶ Include header shmem.h to access the library
    - ▶ E.g. #include <shmem.h>, #include <mpp/shmem.h>
  - ▶ start\_pes, shmem\_init: Initializes the calling PE
  - ▶ my\_pe: Get the PE ID of local processor
  - ▶ num\_pes: Get the total number of PEs in the system

| SGI         |              | Quadratics | Cray        |             |
|-------------|--------------|------------|-------------|-------------|
| Fortran     | C/C++        | C/C++      | Fortran     | C/C++       |
| start_pes   | start_pes(0) | shmem_init | start_pes   | start_pes   |
|             |              |            | shmem_init  | shmem_init  |
| shmem_my_pe | shmem_my_pe  |            | shmem_my_pe | shmem_my_pe |
| shmem_n_pes | shmem_n_pes  |            | shmem_n_pes | shmem_n_pes |
| NUM_PES     | num_pes      | num_pes    | NUM_PES     |             |
| MY_PE       | my_pe        | my_pe      |             |             |

17

This is what happens when there's no standard. You can see the various different calls that showed up in different APIs from different vendors. Different routine names, and different parameter sets. In particular, the initialization call is completely different across different vendor SHMEMs.

## OpenSHMEM DIVERGENT IMPLEMENTATIONS (2)

### Hello World (SGI on Altix)

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

int main(void)
{
    int me, npes;
    start_pes(0);
    npes = _num_pes();
    me = _my_pe();
    printf("Hello from %d of %d\n", me, npes);
    return 0;
}
```

### Hello World (SiCortex)

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

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

18

The good old “hello world” program shows the different routines and header locations on different platforms. The programs will not compile with the other vendors’ SHMEMs.

## **OpenSHMEM**

### THE PROJECT

- ▶ <http://www.openshmem.org/>
- ▶ Standardized specification
- ▶ Reference Library of spec.
- ▶ Tutorials & other educational material
- ▶ Vendor products & information
- ▶ Community involvement, talk to each other!
- ▶ Tool-chain ecosystem

19

So enough people were interested in forging a standard for SHMEM, which ended up with the obvious name of OpenSHMEM. The website has quite a lot of things on it, not just software, but publications, vendor information and of course tutorials like this one.

## **OpenSHMEM ROUTINES**

- ▶ **Initialization and Program Query**
- ▶ **Data transfers**
- ▶ **Synchronization mechanisms**
- ▶ **Collective communication**
- ▶ **Atomic Memory Operations**
- ▶ **Address Manipulation, Data Cache control**
  - ▶ Not supported by all SHMEM implementations

20

These are the routines in the API. Need to talk about how these fit together and are actually used. Hands-on should give people a better feeling for usage. Slides tend to be just a laundry list that don't show context and there's not enough space on the screen to show interesting code.

## OpenSHMEM INITIALIZATION & QUERY

- ▶ `void start_pes(int n)`
  - ▶ Initialize the OpenSHMEM program
  - ▶ “n” means “number of PEs” but now ignored, set to 0
  - ▶ Number of PEs taken from invoking environment
    - ▶ E.g. from MPI or job scheduler
  - ▶ PEs numbered 0 .. (N - 1) in flat space
  
- ▶ `int _num_pes(void)`
- ▶ `int shmem_n_pes(void)`
  - ▶ **return number of PEs in this program**
- ▶ `int _my_pe(void)`
- ▶ `int shmem_my_pe(void)`
  - ▶ **return “rank” of calling PE**

21

# OpenSHMEM

## DATA TRANSFER (I)

### ► Put

- ▶ Single variable
  - ▶ `void shmem_TYPE_p(TYPE *target,TYPE value,int pe)`
    - ▶ TYPE = double, float, int, long, short
- ▶ Contiguous object
  - ▶ `void shmem_TYPE_put(TYPE *target,const TYPE *source,size_t nelems,int pe)`
    - ▶ For C:TYPE = double, float, int, long, longdouble, longlong, short
    - ▶ For Fortran:TYPE=complex, integer, real, character, logical
  - ▶ `void shmem_putSS(void *target,const void *source,size_t nelems,int pe)`
    - ▶ Storage Size (SS, bits) = 32, 64, 128, mem (any size)
- ▶ Target must be symmetric

22

## OpenSHMEM DATA TRANSFER (2)

- ▶ Example: Cyclic communication via puts

```
{  
    /*Initializations*/  
    int src;  
    int *dest;  
    ....  
    start_pes(0);  
    ....  
    src = me;  
    dest = (int *) shmalloc (sizeof (*dest));  
    nextpe = (me + 1) % npes; /*wrap around */  
  
    shmem_int_put (dest,&src, 1,nextpe);  
    more_work_goes_here (...)  
    shmem_barrier_all();  
    x = dest * 0.995 + 45 * y;  
    ....  
}
```

Automatic data element

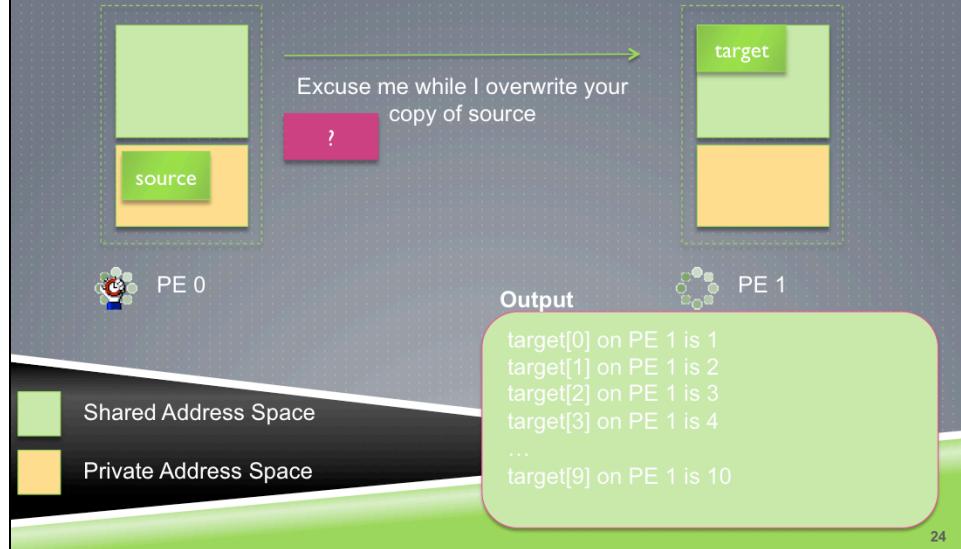
Symmetric data element

Synchronization before use

### Points To Remember

- 'Destination' has to be symmetric
- Consecutive puts are not guaranteed to finish in order
- Put returns after the data has been copied out of the source
- Completion guaranteed only after synchronization

## OpenSHMEM DATA TRANSFER (3): PUT



**shmem\_put():**

# OpenSHMEM

## DATA TRANSFER (4)

### ► Get

- ▶ Single variable
  - ▶ **TYPE shmem\_TYPE\_g(TYPE \*target,TYPE value, int pe)**
    - ▶ For C:TYPE = double, float, int, long, longdouble, longlong, short
    - ▶ For Fortran:TYPE=complex, integer, real, character, logical
- ▶ Contiguous object
  - ▶ **void shmem\_TYPE\_get(TYPE \*target, const TYPE \*source, size\_t nelems, int pe)**
    - ▶ For C:TYPE = double, float, int, long, longdouble, longlong, short
    - ▶ For Fortran:TYPE=complex, integer, real, character, logical
  - ▶ **void shmem\_getSS(void \*target, const void \*source, size\_t nelems, int pe)**
    - ▶ Storage Size (SS, bits) = 32, 64, 128, mem (any size)
- ▶ Source must be symmetric

25

## OpenSHMEM DATA TRANSFER (5)

- Example: Summation at PE 0

```
{  
    /*Initializations*/  
    int *src; Automatic data element  
    int target, sum;  
    ....  
    start_pes(0);  
    ....  
    src = (int *) shmalloc (sizeof (*src)); Symmetric data element  
    src = me;  
    sum=me;  
    if(me == 0){  
        for(int i = 1;i<num_pes();i++){  
            shmem_int_get(&target, src, i); No synchronization before use  
            sum = sum + target;  
        }  
    }  
    ....  
}
```

### Points To Remember

- 'Source' has to be remotely accessible
- Consecutive gets finish in order
- The routines return after the data has been delivered to the 'target' on the local PE

26

## OpenSHMEM DATA TRANSFER (6)

### ► Strided put/get

- ▶ `void shmem_TYPE_iwrite(TYPE *target, const TYPE *source,  
ptrdiff_t tst, ptrdiff_t sst,  
size_t nelems, int pe)`

- ▶ For C:TYPE = double, float, int, long, longdouble, longlong, short
- ▶ For Fortran:TYPE=complex, integer, real, character, logical
- ▶ tst and sst indicate stride between accesses of target and source resp.

- ▶ And the sized variants as for put/get

27

## OpenSHMEM DATA TRANSFER (7)

### ► Put vs. Get

- ▶ Put call completes when data is “being sent”
- ▶ Get call completes when data is “stored locally”
- ▶ Cannot assume put has written until later synchronization
  - ▶ Data still in transit
  - ▶ Partially written at target
  - ▶ Put order changed by e.g. network
- ▶ Puts allow overlap
  - ▶ Communicate
  - ▶ Compute
  - ▶ Synchronize

28

This slide is to stress that puts and gets behave differently. It is clearly a better idea to have puts in your program with deferred synchronization so you can generate more overlap of comms/comp or comms/comms. It’s about getting more work done in different places at the same time. Puts are an offload function.

## OpenSHMEM SYNCHRONIZATION (I)

- ▶ Active Sets
  - ▶ Way to specify a subset of PEs
  - ▶ A triple:
    - ▶ Start PE
    - ▶ Stride ( $\log_2$ )
    - ▶ Size of set
  - ▶ Limitations
    - ▶ Stride must be powers of 2
    - ▶ Only define 'regular' PE sub-groups

29

Every element of this array must be initialized with the value `_SHMEM_SYNC_VALUE` before any of the PEs in the active set enter the routine. OpenSHMEM \*currently\* only has these regular log2 active sets but there's a number of proposals out there for generalizing them. There are interesting headaches associated with separate memory allocations belonging to active sets, where we lose the idea of all allocations being globally symmetric.

## OpenSHMEM SYNCHRONIZATION (2)

- ▶ Quick look at Active Sets
  - ▶ Example 1
    - ▶ PE\_start = 0, logPE\_stride = 0, PE\_size = 4
    - ACTIVE SET?** PE 0, PE 1, PE 2, PE 3
  - ▶ Example 2
    - ▶ PE\_start = 0, logPE\_stride = 1, PE\_size = 4
    - ACTIVE SET?** PE 0, PE 2, PE 4, PE 6
  - ▶ Example 3
    - ▶ PE\_start = 2, logPE\_stride = 2, PE\_size = 3
    - ACTIVE SET?** PE 2, PE 6, PE 10

30

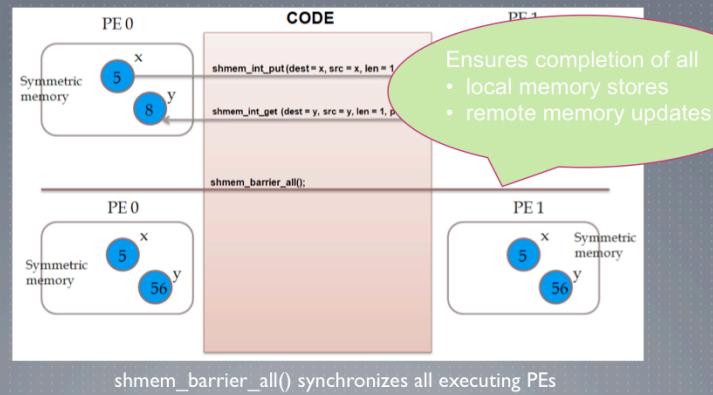
## OpenSHMEM SYNCHRONIZATION (3)

- ▶ Barrier (Group synchronization)
  - ▶ `void shmem_barrier_all()`
    - ▶ Suspend PE execution until all PEs call this function
  - ▶ `void shmem_barrier(int PE_start, int PE_stride, int PE_size, long *pSync)`
    - ▶ Barrier operation on subset of PEs
  - ▶ *pSync is a symmetric work array that allows different barriers to operate simultaneously*

31

People want the explicit pSync parameter to go away in future versions.

## OpenSHMEM SYNCHRONIZATION (4)



32

A barrier\_all is of course global. After memory allocation operations it is required but clearly could be a big slowdown on very large program runs. Tie this observation in with the idea of memory spaces in active sets.

## OpenSHMEM SYNCHRONIZATION (5)

- ▶ Conditional wait (P2P synchronization)
  - ▶ Suspend until local symmetric variable NOT equal to the value specified
  - ▶ `void shmem_wait(long *var, long value)`
  - ▶ `void shmem_TYPE_wait(TYPE *var, TYPE value)`
    - ▶ For C:TYPE = int, long, longdouble, longlong, short
    - ▶ For Fortran:TYPE = complex, integer, real, character, logical
- ▶ Specific conditional wait
  - ▶ Similar to the generic wait except the comparison can now be
    - ▶  $\geq, >, =, !=, <, \leq$
  - ▶ `void shmem_wait_until(long *var, int cond, long value)`
  - ▶ `void shmem_TYPE_wait_until(TYPE *var, int cond, TYPE value)`
    - ▶ TYPE = int, long, longlong, short

33

Waits are used as a synchronization tool that is a LOT lighter than a big barrier.

## OpenSHMEM SYNCHRONIZATION (6)

### Fence

**Ordering of outgoing write (put) operations to a single PE**

```
void shmem_fence()
```

### Quiet

**Ordering of all outgoing puts from the calling PE (on some implementations;  
fence = quiet)**

```
void shmem_quiet()
```

34

Fence can be implemented as quiet as it is a strict subset of quiet. But the opposite implementation is not true. Quiet affects all outbound puts. You can think of it as turning a valve on a compute node's network card and waiting for things to flush out before allowing more traffic...

# OpenSHMEM SYNCHRONIZATION (7)

## Example Fence

```
int main (int argc, char **argv)
{
    ...
    ...
    shmem_int_put (dest1, src1, l, nextpe);
    shmem_fence();
    shmem_inc_put (dest1, src2, l, nextpe);
    ...
    shmem_barrier_all ();
    shfree (dest);

    return 0;
}
```

## Example Quiet

```
int main (int argc, char **argv)
{
    ...
    ...
    shmem_int_put (dest1, src1, l, nextpe);
    shmem_int_put (dest2, src2, l, nextpe+1);
    shmem_quiet();

    ...
    shmem_barrier_all ();
    shfree (dest);

    return 0;
}
```

35

## OpenSHMEM COLLECTIVE COMMUNICATION (I)

- ▶ Broadcast
  - ▶ One-to-all symmetric communication
  - ▶ No update on root

- ▶ `void shmem_broadcastSS(void *target, void *source, size_t nelems,  
int PE_root, int PE_start, int PE_stride, int PE_size, long *pSync)`

Storage Size (SS, bits) = 32, 64

36

## OpenSHMEM COLLECTIVE COMMUNICATION (2)

37

```
...
...
int *target, *source;
target= (int *) shmalloc( sizeof(int) );
source= (int *) shmalloc( sizeof(int) );
*target= 0;
*source= 101;
if (me == 1) {
    *source = 222;
}
shmem_barrier_all();
shmem_broadcast32(target, source, 10, 0, 0, 4, pSync);
printf("target on PE %d is %d\n", _my_pe(), *target);
...
```

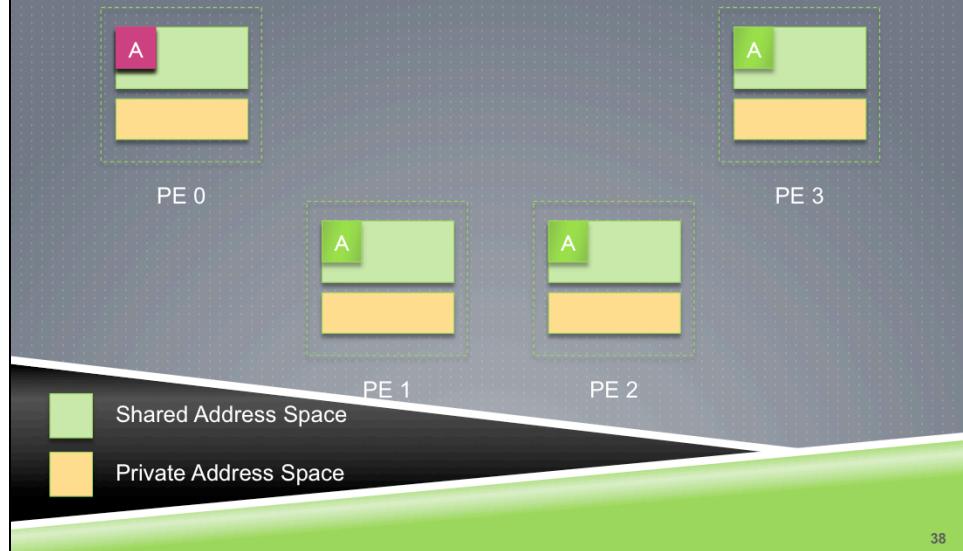
### Output

target on PE 0 is 0  
target on PE 1 is 222  
target on PE 2 is 222  
target on PE 3 is 222

of the PE  
active set

Code snippet showing working of shmem\_broadcast

## OpenSHMEM COLLECTIVE COMMUNICATION (3)



## OpenSHMEM COLLECTIVE COMMUNICATION (4)

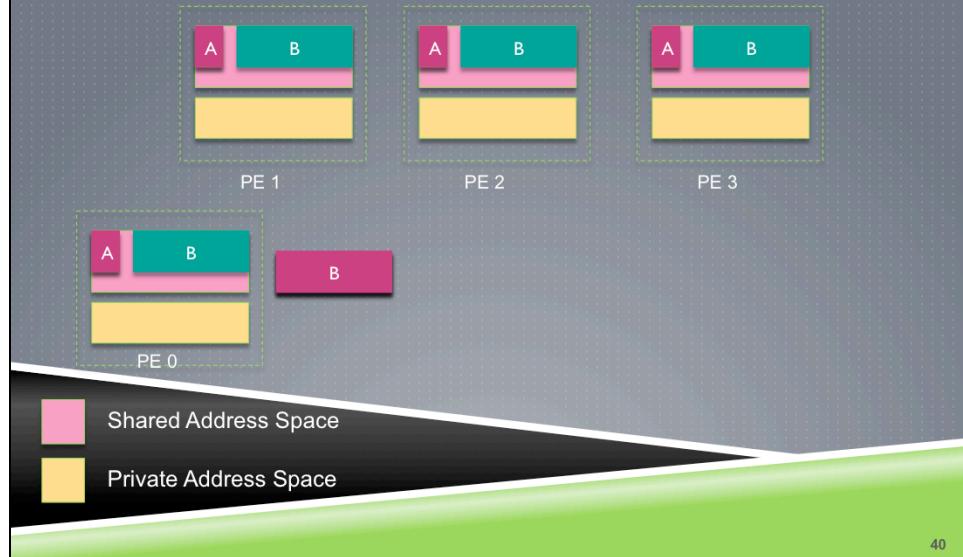
Storage Size (SS, bits) = 32, 64

### ► Collection

- ▶ Concatenates blocks of symmetric data from multiple PEs to an array in every PE
- ▶ Each PE can contribute different amounts
- ▶ `void shmem_collectSS(void *target, void *source, size_t nelems, int PE_start, int PE_stride, int PE_size, long *pSync)`
- ▶ Concatenation written on all participating PEs
- ▶ **shmem\_fcollect variant**
  - ▶ When all PEs contribute exactly same amount of data
  - ▶ PEs know exactly where to write data, so no offset lookup overhead

39

## OpenSHMEM COLLECTIVE COMMUNICATION (5)



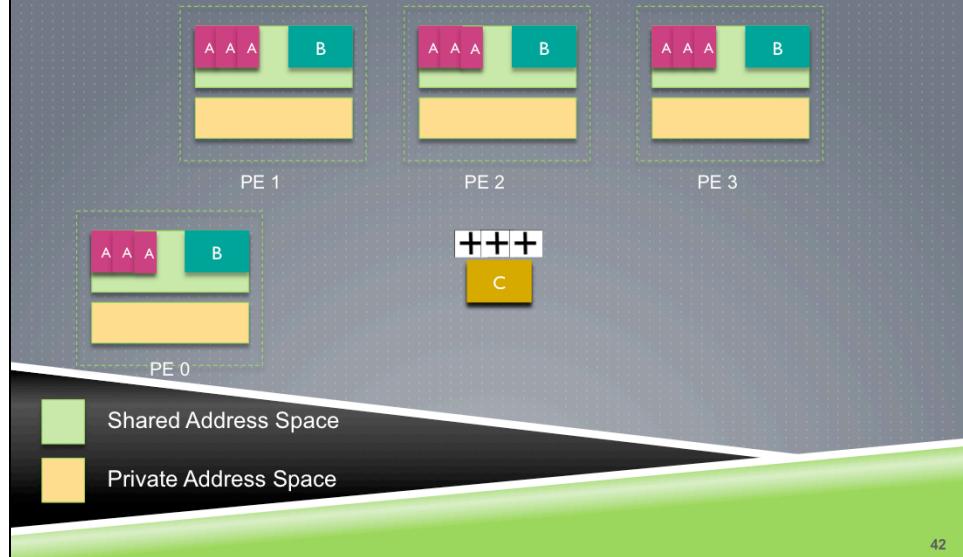
## OpenSHMEM COLLECTIVE COMMUNICATION (6)

### ► Reductions

- Perform commutative operation across symmetric data set
  - `void shmem_TYPE_OP_to_all(TYPE *target, TYPE *source, int nreduce, int PE_start, int PE_stride, int PE_size, TYPE *pWrk, long *pSync)`
    - Logical OP = and, or, xor
    - Extrema OP = max, min
    - Arithmetic OP = prod(uct), sum
    - TYPE = int, long, longlong, longdouble, short, complex
  - Reduction performed and stored on all participating PEs
  - pWrk and pSync allow interleaving
  - E.g. compute arithmetic mean across set of PEs
    - `sum_to_all / PE_size`

41

## OpenSHMEM COLLECTIVE COMMUNICATION (7)



## OpenSHMEM ATOMIC OPERATIONS (I)

- ▶ What does “atomic” mean anyway?
  - ▶ Indivisible operation on symmetric variable
  - ▶ No other operation can interpose during update
  - ▶ But “no other operation” actually means...?
    - ▶ No other atomic operation
    - ▶ Can’t do anything about other mechanisms interfering
      - ▶ E.g. thread outside of OpenSHMEM program
      - ▶ Non-atomic OpenSHMEM operation
  - ▶ Why this restriction?
    - ▶ Implementation in hardware

43

Offloading operations to hardware generates more overlap and allows the CPU(s) to do other useful things, rather than service network traffic. Vendors put atomics, collectives and other ops. onto network hardware. Arithmetic/logic ops can also be put onto memory controllers for example = do the ops where the data is, don’t incur cost of moving data back and forth (network, memory, ...)

## OpenSHMEM ATOMIC OPERATIONS (2)

### ► Atomic Swap

- ▶ Unconditional
  - ▶ `long shmem_swap(long *target, long value, int pe)`
  - ▶ `TYPE shmem_TYPE_swap(TYPE *target, TYPE value, int pe)`
    - ▶ `TYPE` = double, float, int, long, longlong
    - ▶ Return old value from symmetric target
- ▶ Conditional
  - ▶ `TYPE shmem_TYPE_cswap(TYPE *target, TYPE cond, TYPE value, int pe)`
    - ▶ `TYPE` = int, long, longlong
    - ▶ Only if "cond" matches value on target

44

## OpenSHMEM ATOMIC OPERATIONS (3)

### ► Arithmetic

- ▶ increment (= add 1) & add value
- ▶ `void shmem_TYPE_inc(TYPE *target, int pe)`
- ▶ `void shmem_TYPE_add(TYPE *target, TYPE value, int pe)`
  - ▶ `TYPE` = `int, long, longlong`
- ▶ Fetch-and-increment & fetch-and-add value
- ▶ `TYPE shmem_TYPE_finc(TYPE *target, int pe)`
- ▶ `TYPE shmem_TYPE_fadd(TYPE *target, TYPE value, int pe)`
  - ▶ `TYPE` = `int, long, longlong`
- ▶ Return previous value at target on PE

45

## OpenSHMEM ATOMIC OPERATIONS (4)

```
...
long *dest;
dest = (long *) shmalloc( sizeof(*dest) );
*dest= me;
shmem_barrier_all();
...
new_val = me;
if (me== 1) {
    swapped_val = shmem_long_swap(target, new_val, 0);
    printf("%d: target = %d, swapped = %d\n", me, *target, swapped_val);
}
shmem_barrier_all();
...
```

46

## OpenSHMEM ATOMIC OPERATIONS (5)

### ► Locks

- ▶ Symmetric variables
- ▶ Acquired and released to define mutual-exclusion execution regions
  - ▶ Only 1 PE can enter at a time
- ▶ `void shmem_set_lock(long *lock)`
- ▶ `void shmem_clear_lock(long *lock)`
- ▶ `int shmem_test_lock(long *lock)`
  - ▶ Acquire lock if possible, return whether or not acquired
  - ▶ But don't block...
- ▶ Initialize lock to 0. After that managed by above API
- ▶ Can be used for updating distributed data structures

47

Lock use should be minimized as they can turn into performance killers.

## OpenSHMEM ACCESSIBILITY

- ▶ `int shmem_pe_accessible(int pe)`
  - ▶ Can this PE talk to the given PE?
- ▶ `int shmem_addr_accessible(void *addr, int pe)`
  - ▶ Can this PE address the named memory location on the given PE?
- ▶ In SGI SHMEM used for mixed-mode MPI/SHMEM programs
  - ▶ In "pure" OpenSHMEM, could just return "!"
- ▶ Could in future be adapted for fault-tolerance

48

Fault-tolerance is a big deal. There are going to be lots of discussions about how and where to add FT and resilience capabilities to systems, and is a systemic issue, not just something to bolt onto OpenSHMEM.

## OpenSHMEM ADDRESSES & CACHE (I)

### ► Address manipulation

- ▶ `void *shmem_ptr(void *addr, int pc)`
  - ▶ Returns a pointer to a data object on a remote PE
  - ▶ Only of use on platforms where memory physically accessible
  - ▶ i.e. puts/gets are simple memory accesses

49

## OpenSHMEM ADDRESSES & CACHE (2)

### ► Cache control

- ▶ `shmem_clear_cache` - Clears the automatic cache coherency mode
- ▶ `shmem_set_cache` - Manual automatic cache coherence mode
- ▶ `shmem_set_cache_inv` - Enables automatic line cache invalidation mode
- ▶ `shmem_udflush` - Invalidates the entire user space cache coherence
- ▶ `shmem_udflushh` - Makes coherent a specific memory area

50

Access to cache lines was in one of the vendor SHMEMs. In OpenSHMEM, we're not going to deal with platform-specific cache needs.

## OpenSHMEM HARDWARE (I)

- ▶ Where is OpenSHMEM used?
  - ▶ Mainly clusters these days
  - ▶ Infiniband and similar networks
  - ▶ Why?
  
- ▶ Remote direct memory access (RDMA)
  - ▶ Network hardware writes directly into registered region of process memory
  - ▶ Without interrupting remote process(or)
  - ▶ Put symmetric memory areas here

Infiniband  
Myrinet  
Quadrics  
SeaStar  
RoCE

51

This is possibly the most important thing in this tutorial. RDMA means we really can offload and overlap things. If you can make visible (register) all your symmetric memory areas with the network hardware then you can go faster with zero-copy and zero-interrupt.

## OpenSHMEM HARDWARE (2)

- ▶ Offload
- ▶ Infiniband HCAs can do
  - ▶ Atomics
  - ▶ Collectives
  - ▶ Memory pinning
- ▶ Meaning CPU free to do other things
- ▶ Reduced software footprint (QPs)
- ▶ OpenSHMEM library issues offload instructions rather than doing atomics etc.

52

## Developing OpenSHMEM Applications

53

## **OpenSHMEM**

### LOOKING FOR OVERLAPS (I)

- ▶ How to identify overlap opportunities
  - ▶ Put is not an indivisible operation
    - ▶ Send local, reuse local, on-wire, stored
    - ▶ Can do useful work on other data in between

54

## OpenSHMEM

### LOOKING FOR OVERLAPS (2)

- ▶ How to identify overlap opportunities

- ▶ General principle:

- ▶ Identify independent tasks/data
    - ▶ Initiate action as early as possible
      - ▶ Put/barrier/collective
    - ▶ Interpose independent work
    - ▶ Synchronize as late as possible

55

Express ideas about how to look for overlap. This can mean that simple attempts to do MPI -> OpenSHMEM translations could miss lots of opportunities. We need tools that can look at source code and make suggestions or changes. This is a plug for the OpenSHMEM Analyzer.

## **OpenSHMEM**

### LOOKING FOR OVERLAPS (3)

- ▶ How to identify overlap opportunities
- ▶ How could we change OpenSHMEM to get even more overlap?
  - ▶ Divide application into distinct communication and computation phases to minimize synchronization points
  - ▶ Use of point-to-point synchronization as opposed to collective synchronization

56

## OPENSHMEM LOOKING FOR OVERLAPS (4)

- ▶ How to identify overlap opportunities
  - ▶ Shmalloc
    - ▶ Size check, allocate, barrier\_all
  - ▶ Opportunities to do other work after local allocation
  - ▶ Then wait in barrier later
  - ▶ Return handle for synch.

57

More detail about real apps and their OpenSHMEM implementations will come from the ORNL material. ORNL will also have real performance data about scaling.

## OPENSHMEM LOOKING FOR OVERLAPS (5)

- ▶ How to identify overlap opportunities
  - ▶ “\_nb” put/get calls
    - ▶ Local data not free for reuse on return
  - ▶ Return handle for later synch.

58

Talk about some of the proposed API extensions. Stress this is not “in camera” it is a public process.

## SOME OpenSHMEM IMPLEMENTATIONS

- ▶ Reference Library: University of Houston
  - ▶ On top of GASNet for portability
  - ▶ <http://www.openshmem.org/>
- ▶ ScalableSHMEM: Mellanox
  - ▶ For Mellanox Infiniband solutions
  - ▶ <http://www.mellanox.com/products/shmem>
- ▶ Portals-SHMEM: open-source
  - ▶ For Portals clusters
  - ▶ <http://code.google.com/p/portals-shmem/>
- ▶ Open-MPI
  - ▶ <http://www.open-mpi.org/>

59

We're not the only implementation out there. Other people have produced OpenSHMEMs.

## **OpenSHMEM**

### SUMMARY

- ▶ SPMD Library for C and Fortran programs
- ▶ Point-to-point data transfer
- ▶ Broadcast/collective transfer operations
- ▶ Synchronization
- ▶ Atomic operations

60

## OpenSHMEM REFERENCES

- ▶ Stephen W. Poole, Oscar Hernandez, Jeffery A. Kuehn, Galen M. Shipman, Anthony Curtis, and Karl Feind. [OpenSHMEM - Toward a Unified RMA Model](#). Published in Encyclopedia of Parallel Computing, Springer US. Pages 1379-1391, 2011
- ▶ [OpenSHMEM and Related Technologies, Experiences, Implementations, and Tools](#), First Workshop, OpenSHMEM 2014, Annapolis, MD, USA, March 4-6, 2014, Proceedings
- ▶ OpenSHMEM & Infiniband Research: visit DK Panda's group
  - ▶ <http://nowlab.cse.ohio-state.edu/>

61

## ACKNOWLEDGEMENTS



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