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

OpenSHMEM WORKSHOP OUTLINE

http://www.openshmem.org/

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

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

OpenSHMEM BACKGROUND (1)

- 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

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

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…

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

OpenSHMEM BACKGROUND (5)

- Different ways of partitioning work
  - different tasks * different data

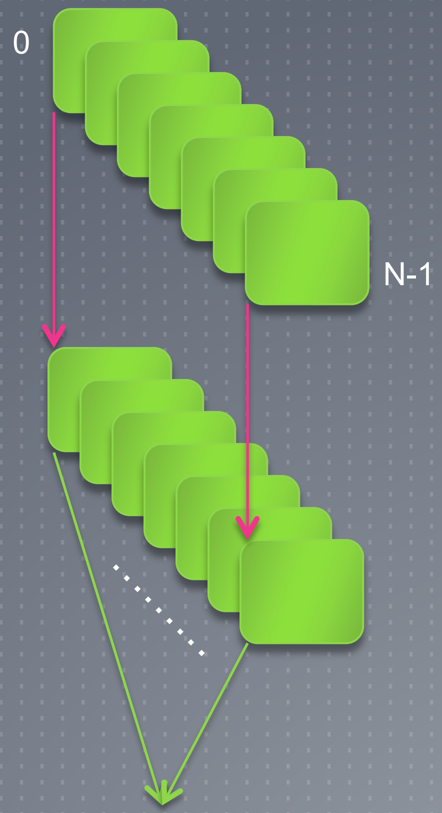
- MPMD = multiple program, multiple data
- SPMD = single program, multiple data
- SPSD = single program, single data
  - Just good old sequential

`http://en.wikipedia.org/wiki/SPMD`

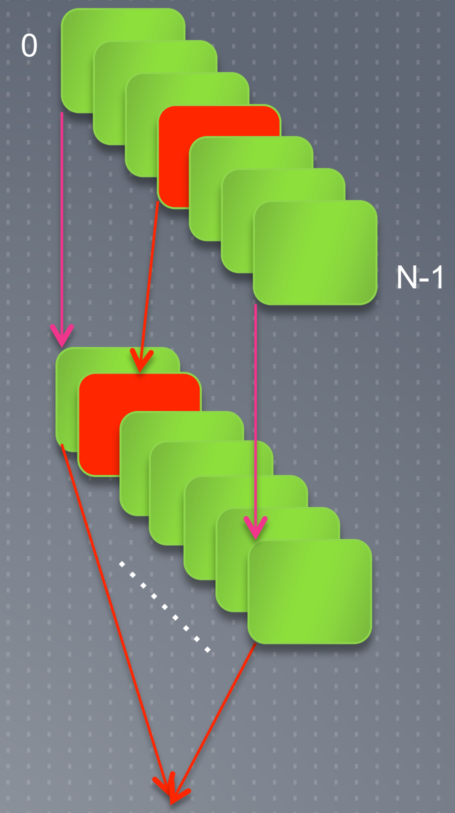
OpenSHMEM BACKGROUND (6)

- 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

OpenSHMEM BACKGROUND (7)



Independent execution



Communication/Synchronization

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

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)

OpenSHMEM BACKGROUND (10)

- OpenSHMEM is SPMD parallel programming library
  - Library of functions similar in feel to using MPI (e.g. *shmem_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.

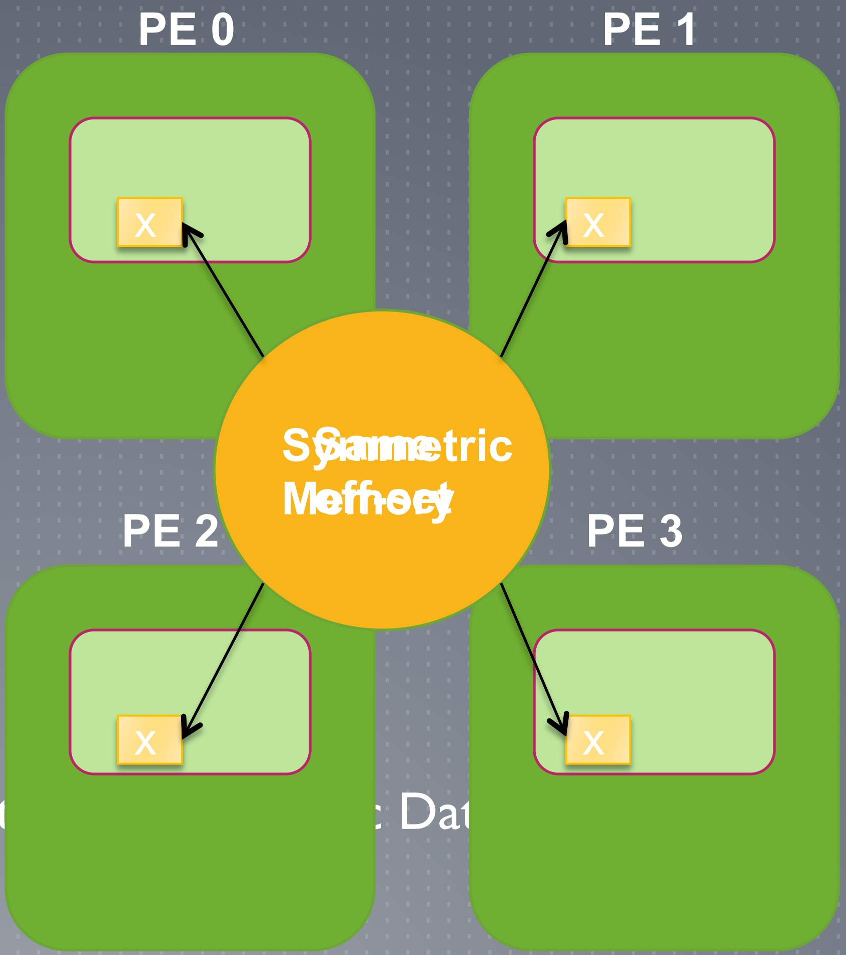
OpenSHMEM CONCEPTS (1)

- 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 **shpalloc**
    - C and C++ data allocated by **shmalloc**

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;
}
```



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.

OpenSHMEM
DIVERGENT IMPLEMENTATIONS (1)

- 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		Quadrics	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		

OpenSHMEM DIVERGENT IMPLEMENTATIONS (2)

Hello World (SGI on Altix)

```
#include <stdio.h>
#include <mpp/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;
}
```

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

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

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**

OpenSHMEM
DATA TRANSFER (1)

- 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

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

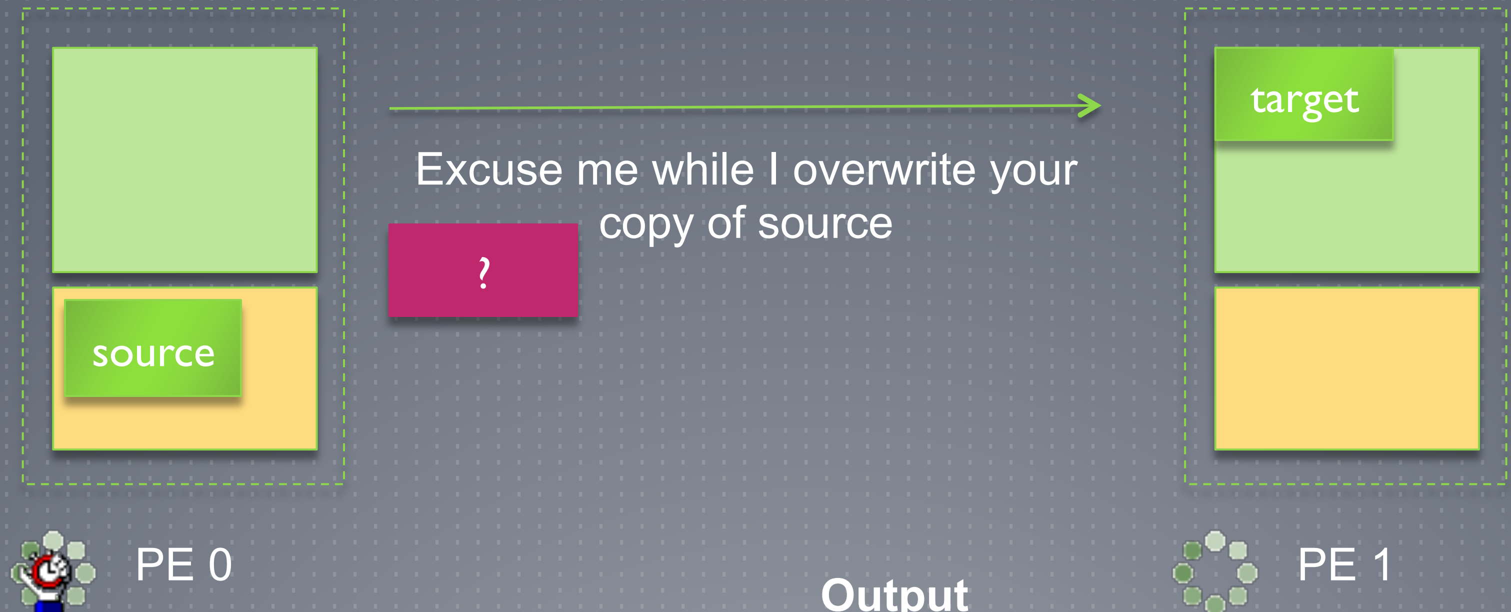
Symmetric data element

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



Output

target[0] on PE 1 is 1
target[1] on PE 1 is 2
target[2] on PE 1 is 3
target[3] on PE 1 is 4
…
target[9] on PE 1 is 10

Shared Address Space

Private Address Space

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

OpenSHMEM DATA TRANSFER (5)

- Example: Summation at PE 0

```
{
    /*Initializations*/
    int *src;
    int target, sum;
    …..
    start_pes(0);
    ….
    src  = (int *) shmalloc (sizeof (*src));
    src = me;
    sum=me;
    if(me == 0){
        for(int i = 1,i<num_pes();i++){
          shmem_int_get(&target, src, 1
          sum = sum + target;
        }
    }
    …
}
```

Automatic data element

Symmetric data element

No synchronization before use

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

OpenSHMEM
DATA TRANSFER (6)

- Strided put/get
  - **void shmem_TYPE_iput(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

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

OpenSHMEM SYNCHRONIZATION (1)

- 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

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

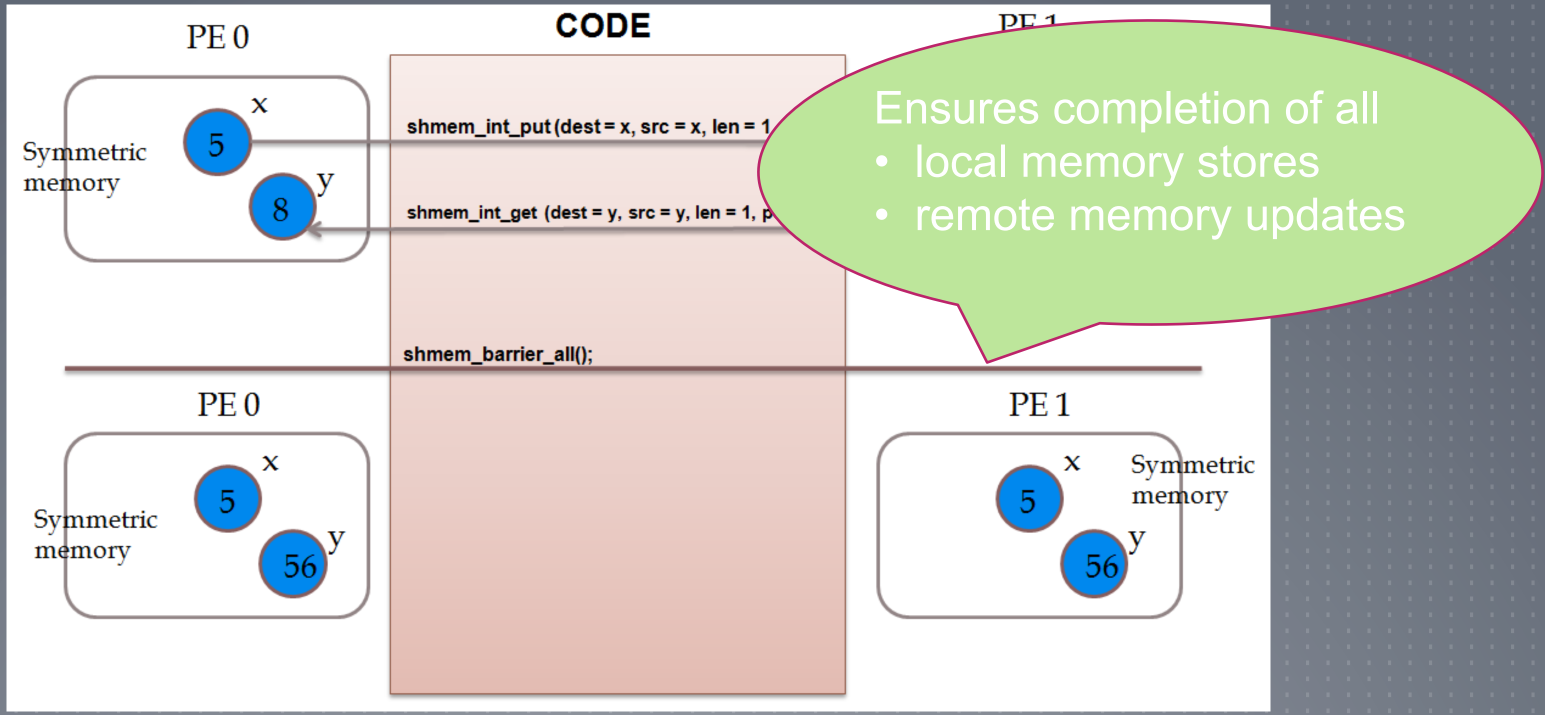
    ACTIVE SET? **PE 0, PE 2, PE 4, PE 6**
  - Example 3
    - PE_start = 2, logPE_stride = 2, PE_size = 3

    ACTIVE SET? **PE 2, PE 6, PE 10**

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

OpenSHMEM SYNCHRONIZATION (4)



shmem_barrier_all() synchronizes all executing PEs

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
      - >=, >, =, !=, <, <=
    - **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

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()

OpenSHMEM SYNCHRONIZATION (7)

Example Fence

```
int main (int argc, char **argv)
{
….
….
  shmem_int_put (dest1, src1, 1, nextpe);
  shmem_fence();
  shmem_int_put (dest1, src2, 1, nextpe);
  ….
  shmem_barrier_all ();
  shfree (dest);

  return 0;
}
```

Example Quiet

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

  return 0;
}
```

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

OpenSHMEM
COLLECTIVE COMMUNICATION (2)

```
…
…
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, 1 0, 0, 0, 4, pSync);

 printf("target on PE %d is %d\n", _my_pe(), *target);
…
```

collective operation

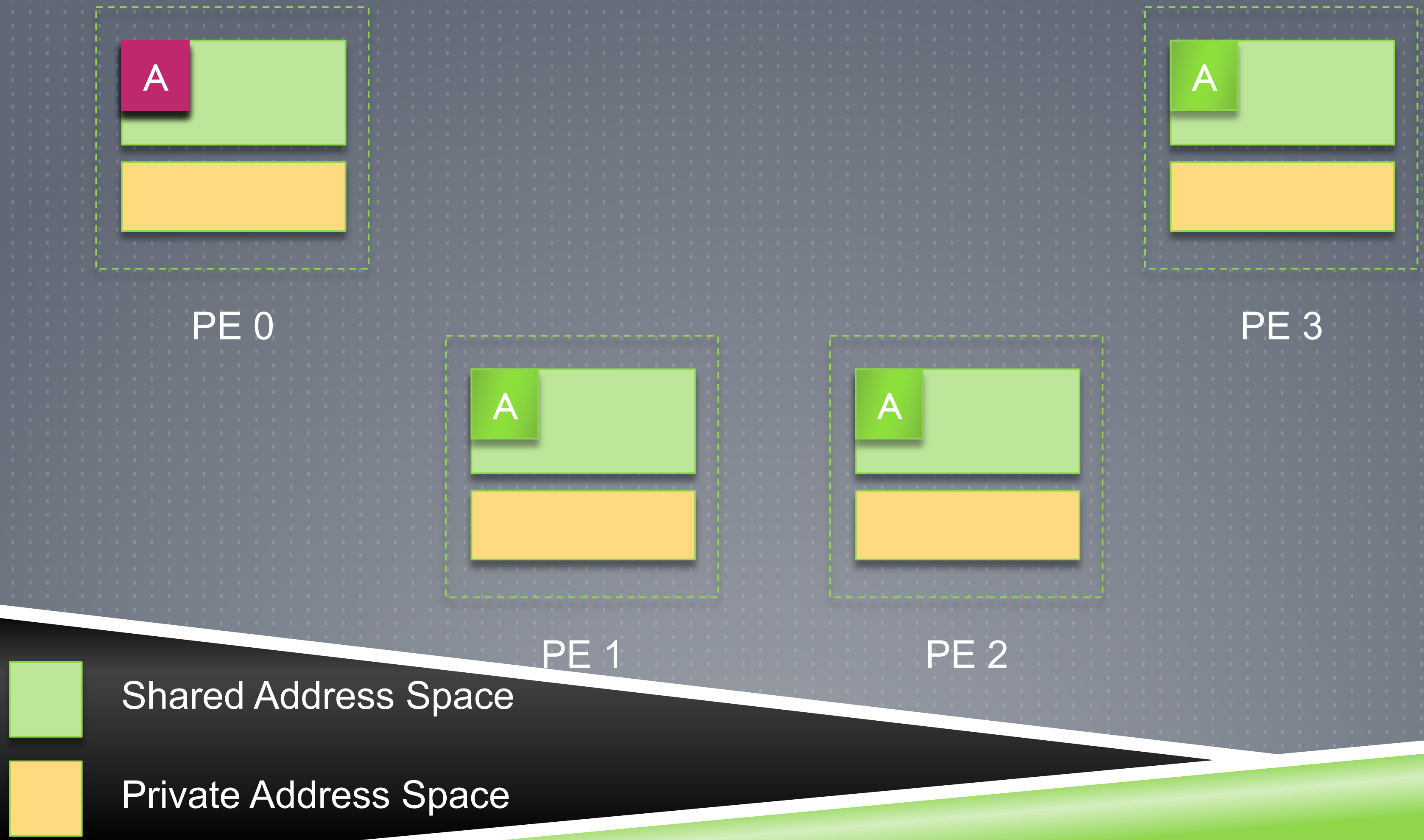
Output

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)

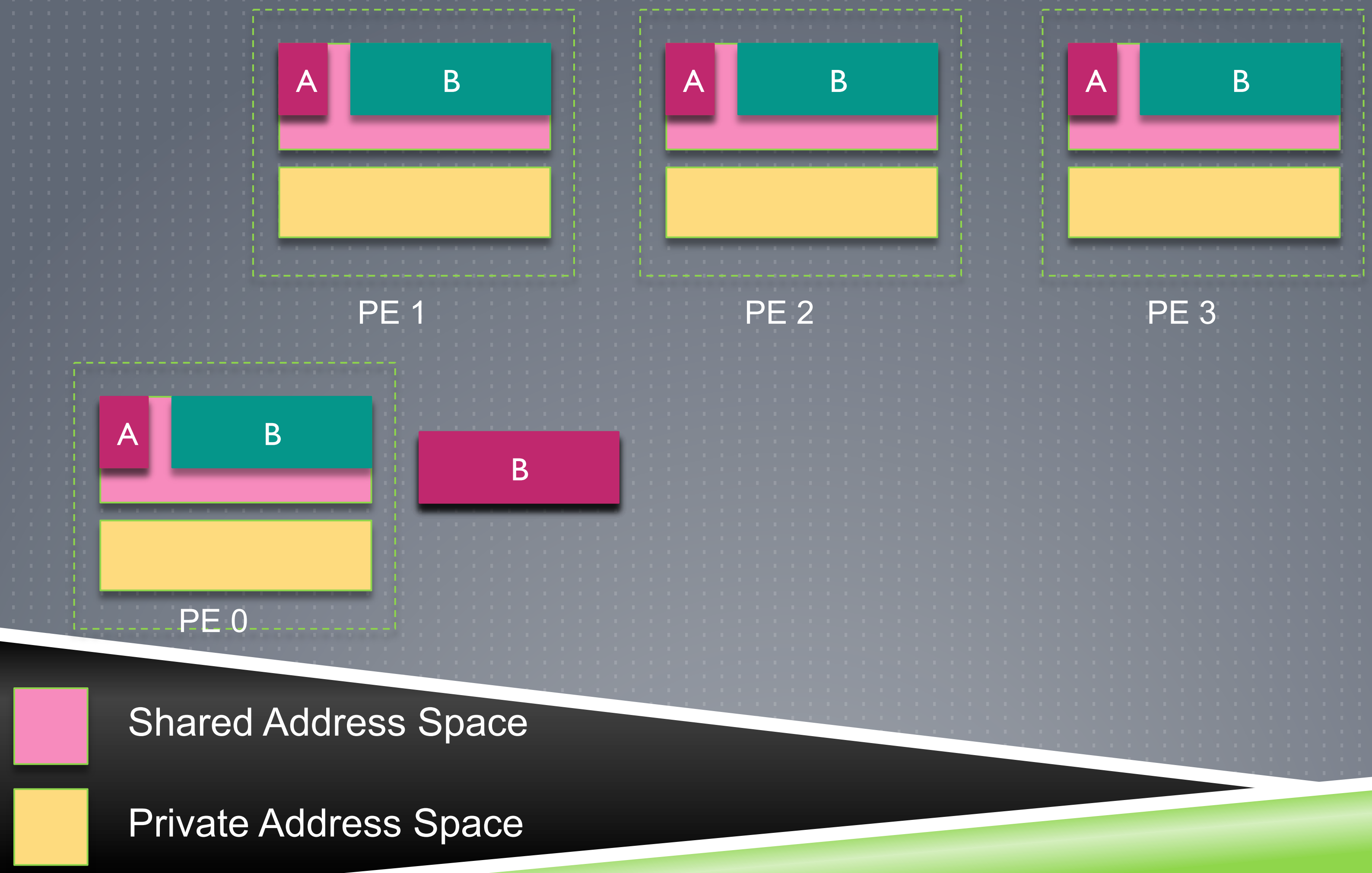


OpenSHMEM COLLECTIVE COMMUNICATION (4)

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

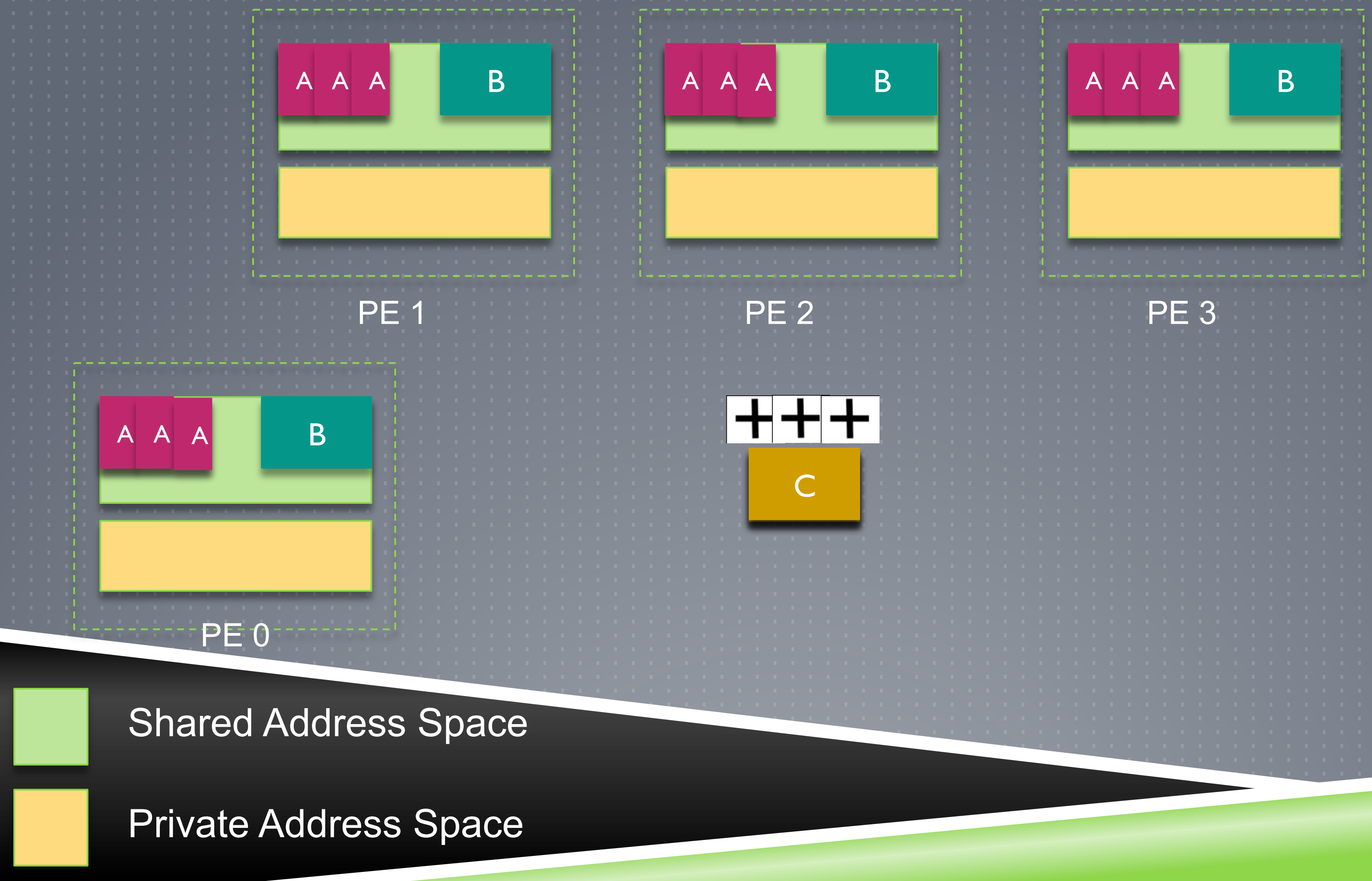
OpenSHMEM
COLLECTIVE COMMUNICATION (5)



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

OpenSHMEM
COLLECTIVE COMMUNICATION (7)



OpenSHMEM
ATOMIC OPERATIONS (1)

- 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

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

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

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();
…
```

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

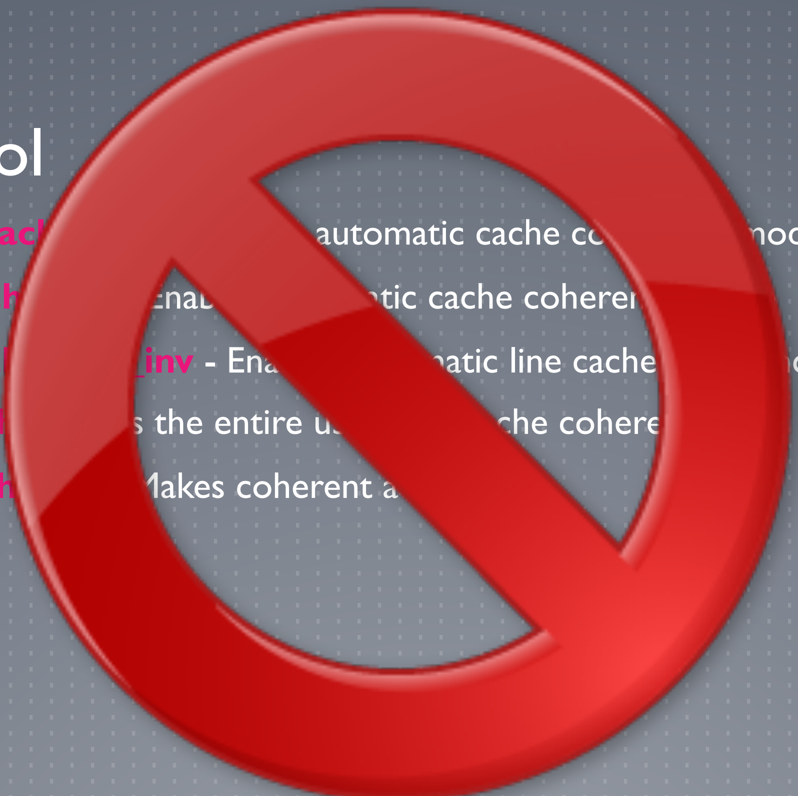
OpenSHMEM ACCESSIBILITY

- **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 "1"
- Could in future be adapted for fault-tolerance

OpenSHMEM
ADDRESSES & CACHE (1)

- Address manipulation
  - **void *shmem_ptr(void *addr, int pe)**
    - 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

OpenSHMEM
ADDRESSES & CACHE (2)



- Cache control
  - shmem_clear_cac automatic cache co mode
  - shmem_set_cach Enab tic cache coherei
  - shmem_set_cac inv - Ena atic line cache cy mode
  - shmem_udcflus s the entire u che cohere
  - shmem_udcflush akes coherent a

OpenSHMEM
HARDWARE (1)

- 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

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.

Developing OpenSHMEM Applications

OpenSHMEM
LOOKING FOR OVERLAPS (1)

- 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

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

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

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.

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.

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/

OpenSHMEM
SUMMARY

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

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/

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.