OpenSHMEM TUTORIAL

Presenters: Swaroop Pophale and Tony Curtis

University of Houston, Texas

Acknowledgement

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

Outline

- About us
- Background
- History and Implementations
- The OpenSHMEM Effort
- OpenSHMEM API
- Porting
- A look ahead…
- References

OpenSHMEM Tutorial
Introductory Material

Dr. Barbara Chapman

Tony Curtis

Swaroop Pophale

Ricardo Mauricio

Ram Nanjegowda

OpenSHMEM Tutorial Introductory Material

- http://www.cs.uh.edu/~hpctools/
- Research in a number of areas
  - Focused on large scale parallelism
  - Exascale
- ~ 20 MS & PhD students
- 3 senior staff and assistant professor

OpenSHMEM Tutorial
Introductory Material

- So what is it our group researches?

- OpenMP
  - Extreme scale
  - Distributed systems
  - Locality

OpenSHMEM Tutorial
Introductory Material

- Compiler technology
  - OpenUH (based on Open64)

OpenSHMEM Tutorial
Introductory Material

- Heterogeneous Computing
  - Power-aware
  - OpenMP
  - MCA
  - Accelerators

OpenSHMEM Tutorial
Introductory Material

- PGAS Languages and Libraries
  - UPC
  - CAF
  - Both supported in OpenUH

OpenSHMEM Tutorial
Introductory Material

- PGAS Languages and Libraries
  - SHMEM
  - Chapel

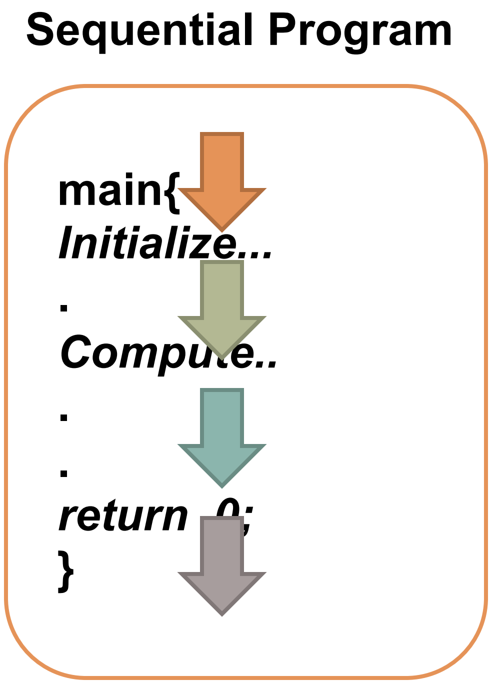
11

NEW TO PARALLEL COMPUTING?

WE WILL TAKE IT SEQUENTIALLY..

Background
What is Parallel Computing?

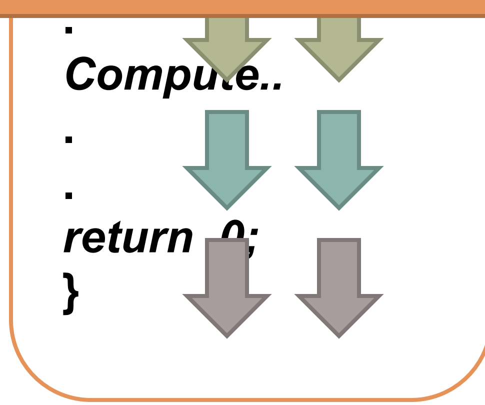
- **Parallel computing** is the simultaneous use of multiple compute resources to solve a computational problem.



Background
What is Parallel Computing?

- **Parallel computing** is the simultaneous use of multiple compute resources to solve a computational problem.

Concurrent ≠ Parallel



Background
Different types Parallel Programming

- Single Program Multiple Data (SPMD)
  - All processes are doing the same thing with different data items
- Multiple Program Multiple Data (MPMD)
  - All process are executing different programs and using different data items

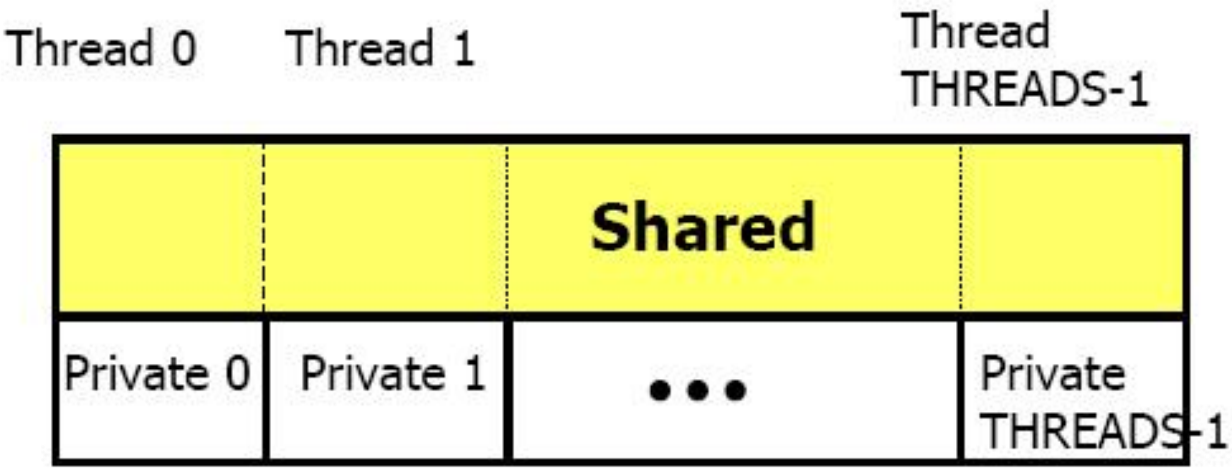
Background
What is a Programming Model?

- A view of data and execution
- Where architecture and applications meet
- Can be viewed as a "contract"
  - Everyone knows the rules
  - Better understanding of performance considerations
- Benefits
  - Application - independence from architecture
  - Architecture - independence from applications

Background
Programming Models

- Data Parallel Model
  - HPF
- Communication Centric Model
  - MPI
- Shared Memory Model
  - OpenMP
- Distributed-Shared Memory Model or the Partitioned Global Address Space Model
  - UPC, CAF, SHMEM

Background
PGAS Programming Model



Logical Layout of PGAS Programming Model

Background
UPC

- Unified Parallel C
- Language defines a "physical" association between shared data items and UPC threads called ”*affinity*”.
  - All scalar data has affinity with thread 0.
  - Arrays may have *cyclic* (per element), *blocked-cyclic* (user-defined) or *blocked* (run-time) affinity.
- All thread interaction is explicitly managed by the programmer through language primitives: locks, barriers, memory fences.
- Work sharing using “forall”

Background
CAF

- The number of images is fixed and each image has its own index, retrievable at run-time.
- Each image executes the same program independently of the others.
- The programmer inserts explicit synchronization and branching as needed.
- An “object” has the same name in each image.
- Each image works on its own local data.
- An image moves remote data to local data through, and only through, explicit CAF syntax.

20

NEW TO SHMEM?

Introduction
What is SHMEM?

- Symmetric Hierarchical MEMory library
  - For Single Program Multiple Data style of programming
  - Available for C , C++, and Fortran
- Used for programs that
  - perform computations in separate address spaces and
  - explicitly communicate data to and from different processes in the program.
- The processes participating in SHMEM applications are referred to as processing elements (PEs).
- SHMEM routines supply remote one-sided data transfer, broadcast, reduction, synchronization, and atomic memory operations.

Introduction
History of SHMEM

- Cray SHMEM
  - SHMEM first introduced by Cray Research Inc. in 1993 for Cray T3D
  - Platforms: Cray T3D, T3E, PVP, XT series
- SGI SHMEM
  - SGI incorporated Cray SHMEM in their Message Passing Toolkit (MPT)
  - Owns the "rights" for SHMEM
- Quadrics SHMEM (company out of business)
  - Optimized API for QsNet
  - Platform: Linux cluster with QsNet interconnect
- Others
  - GSHMEM, University of Florida
  - HP SHMEM, IBM SHMEM (used internally only)
  - GPSHMEM (cluster with ARMCI & MPI support, dead)

Note: SHMEM was not defined by any one standard.

The Problem: Differences in SHMEM Implementations (1)

- Initialization
  - Include header shmem.h
    - E.g. #include <shmem.h> , #include <mpp/shmem.h>
  - start_pes, shmem_init: Initializes the library
  - my_pe: Get the PE ID of local processor (0 to N-1)
  - num_pes: Get the total number of PEs in the program

SGI		Quadrics	Cray	
Fortran	C/C++	C/C++	Fortran	C/C++
start_pes(0)	**start_pes(0)**	**shmem_init**	**start_pes**	**start_pes**
			shmem_init	**shmem_init**
NUM_PES	**_num_pes**	**num_pes**	**NUM_PES**	
MY_PE	**_my_pe**	**my_pe**		

The Problem: Differences in SHMEM 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;
}
```

The Problem: Differences in SHMEM Implementations (2)

Hello World on 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 on 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;
}
```

The Solution:

OpenSHMEM

DEPARTMENT OF DEFENSE • UNITED STATES OF AMERICA •

UNIVERSITY of HOUSTON

Open Source Software Solutions Inc.

OAK RIDGE National Laboratory

What is OpenSHMEM?

- An effort to create a standardized SHMEM library API and defining expected behavior
- Aims at bringing together hardware vendors and SHMEM library developers
- Discuss and extend standard with important new capabilities

SGI's SHMEM API is the baseline for OpenSHMEM Specification 1.0

OpenSHMEM Outreach

- Community web site (under construction)
- Wiki
- Documentation
  - OpenSHMEM 1.0 Specification
  - FAQ
  - Cheat sheet
- Training material and tutorials
- Mailing list
  - https://email.ornl.gov/mailman/listinfo/openshmem

OpenSHMEM Participation

- **PGAS'10**
  - Workshop and paper
- **SC'10 New Orleans**
  - Booth presence (PGAS, Oak Ridge National Laboratory, Gulf Coast Academic Supercomputing)
  - BOF Session
  - GCAS booth presentation
- **ICS 2011**
  - Poster Presentation

OpenSHMEM Participation

- **PGAS'11**
  - Workshop
- **SC'11**
  - Poster
  - BOF

Key Concept
Remote Direct Memory Access

- RDMA lets one PE access certain variables of another PE without interrupting the other PE
- SHMEM can take advantage of hardware RDMA
- SHMEM's data transfer uses **symmetric variables**

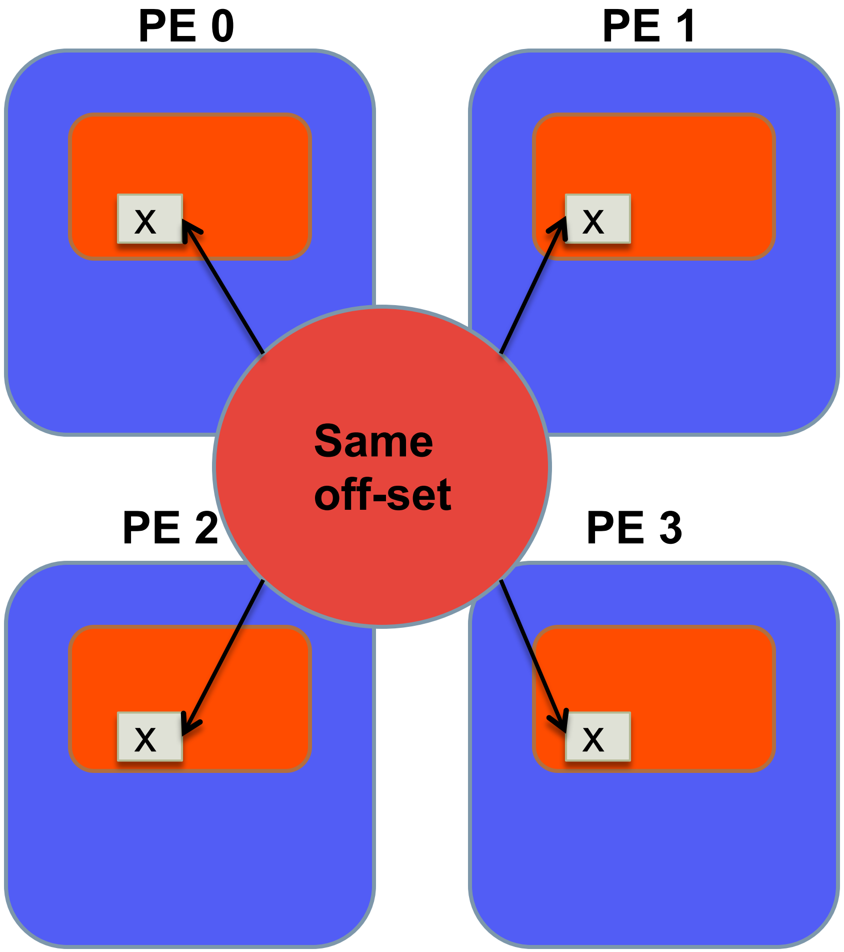
Key Concept
Symmetric Variables

- Symmetric Variables
  - Scalars or arrays that exist with the **same size, type, and relative address** on all PEs.
- There are two types of Symmetric Variables
  - Globals
  - Dynamically allocated and maintained by the SHMEM library
- 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**

Run time allocation on *symmetric heap*

Key Concept Symmetric Variables

```
int main (void)
{
  int *x;
  …
  start_pes(0);
  …
  x = (int*) shmalloc(sizeof(x));
  …
  …
  shmem_barrier_all();
  …
  shfree(x);
  return  0;
}
```



Dynamic allocation of Symmetric Data

OpenSHMEM Routines

- **Data transfers**
  - One sided *puts* and *gets*
- **Synchronization**
  - Barrier, Fence, quiet
- **Collective communication**
  - Broadcast, Collection, Reduction
- **Address Manipulation and Data Cache control**
  - Not supported by all SHMEM implementations (Deprecated in OpenSHMEM 1.0)
- **Atomic Memory Operations**
  - Provide mechanisms to implement mutual exclusion
  - Swap, Add, Increment, fetch
- **Distributed Locks**
  - Set, free and query
- **Accessibility Query Routines**
  - PE accessible, Data accessible

OpenSHMEM API
Data Transfer (1)

- Put
  - Single value
    - double, float, int, long, short, longlong, longdouble, char
  - Contiguous object
    - For C: TYPE = double, float, int, long, longdouble, longlong, short, 32, 64, 128, mem
    - For Fortran: TYPE=complex, integer, real, character, logical
  - Strided
    - For C: TYPE = double, float, int, long, longdouble, longlong, short, 32, 64, 128, mem
    - For Fortran: TYPE=complex, integer, real, character, logical

OpenSHMEM API Data Transfer (2): Put

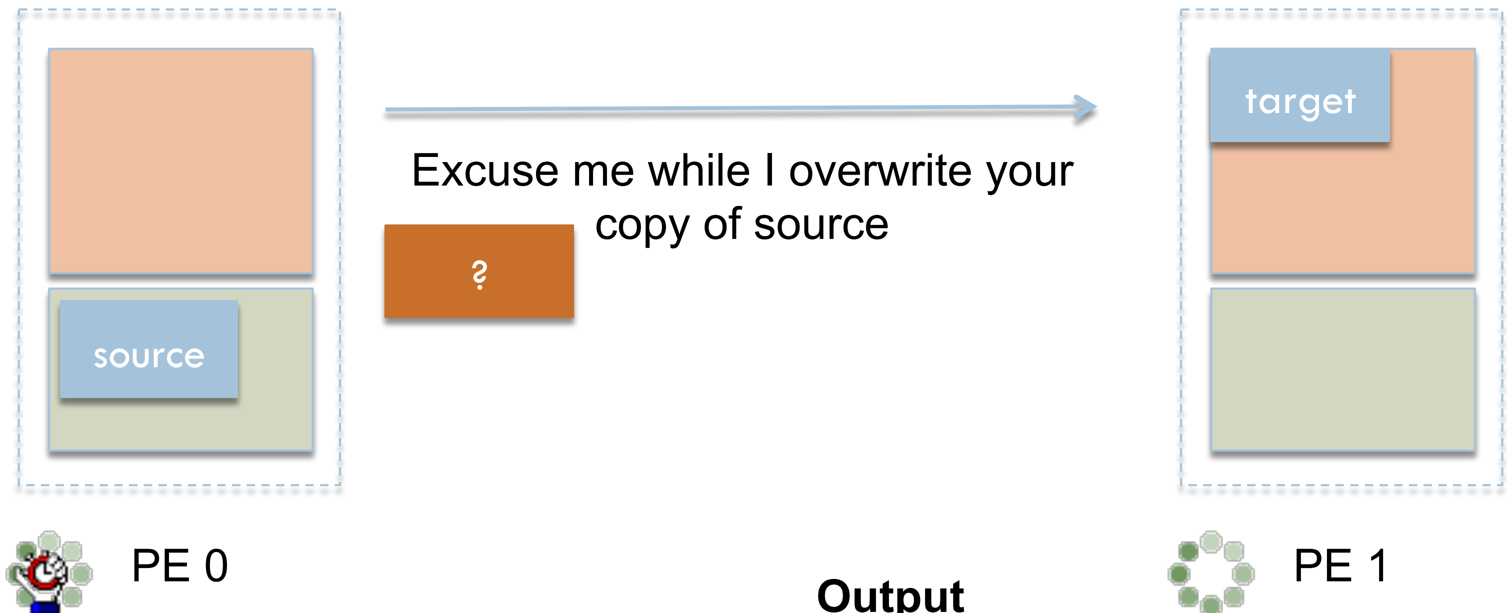
```
..
long source[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
static long target[10];

start_pes(0);
if (_my_pe() == 0) {
   /* put 10 words into target on PE 1 */
   shmem_long_put(target, source, 10, 1);
}
shmem_barrier_all(); /* sync sender and receiver */

if (_my_pe() == 1) {
  for(i=0;i<10;i++)
    printf("target[0] on PE %d is %d\n", _my_pe(), target[0]);
}
…
```

Code snippet showing a put from PE 0 to PE 1

OpenSHMEM API
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 API
Data Transfer (4)

- Get
  - Single value
    - double, float, int, long, short, longlong, longdouble, char
  - Contiguous object
    - For C: TYPE = double, float, int, long, longdouble, longlong, short, 32, 64, 128, mem
    - For Fortran: TYPE=complex, integer, real, character, logical
  - Strided
    - For C: TYPE = double, float, int, long, longdouble, longlong, short, 32, 64, 128, mem
    - For Fortran: TYPE=complex, integer, real, character, logical

OpenSHMEM API
Data Transfer (4): Get

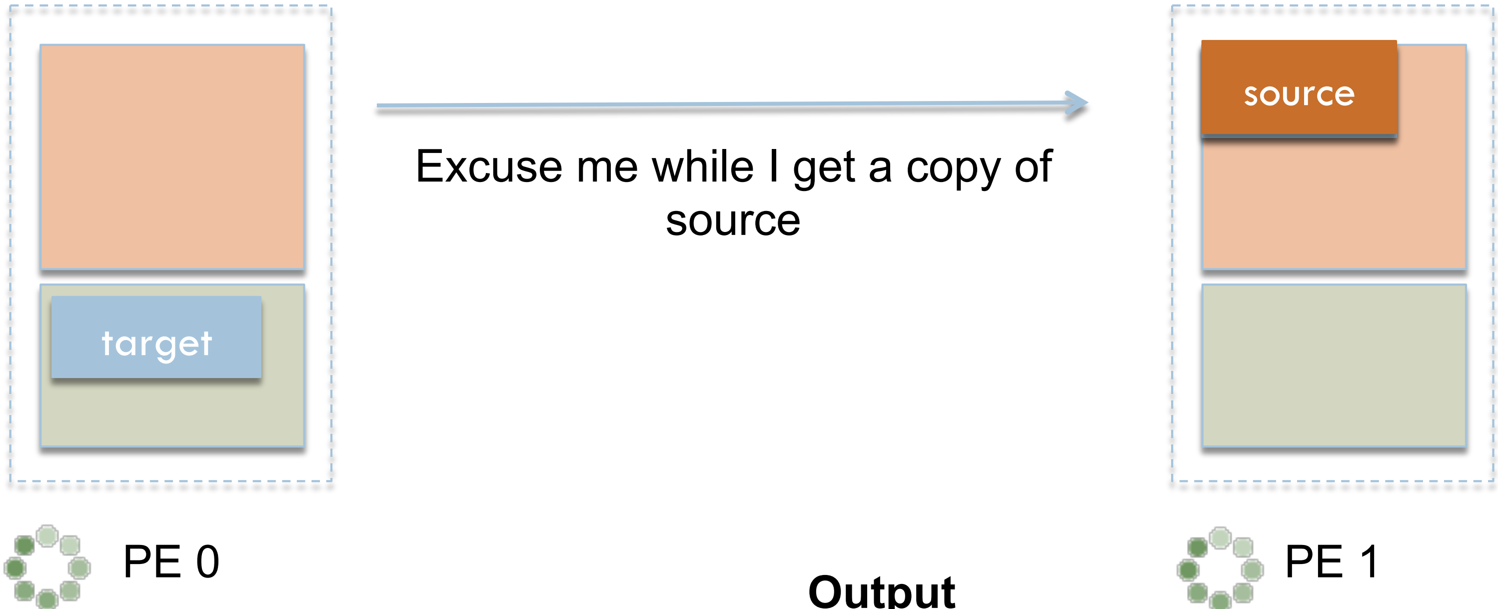

```
..
static long source[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
long target[10];

start_pes(0);
if (_my_pe() == 1) {
   /* get 10 words into target from PE 0 */
   shmem_long_get(target, source, 10, 0);
}

if (_my_pe() == 1) {
  for(i=0;i<10;i++)
    printf("target[0] on PE %d is %d\n", _my_pe(), target[0]);
}
…
```

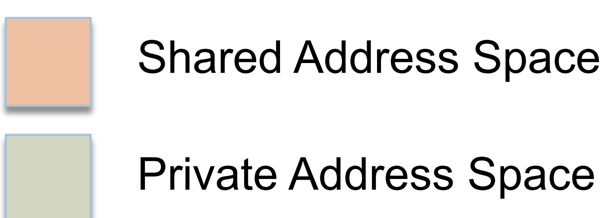
Code snippet showing PE 1 get data from PE 0

OpenSHMEM API
Data Transfer (5): Get



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



OpenSHMEM Collective API
Group Synchronization

- Barrier
  - *pSync is a* symmetric work array that enables overlapping collective communication
  - **void shmem_barrier_all()**
    - All PEs wait until every PE calls this function
  -

NEW CONCEPT

"ACTIVE SET"

Subset of PEs defined by Start_PE, logPE_stride and PE_sizes

OpenSHMEM Collective's Concept Active Sets

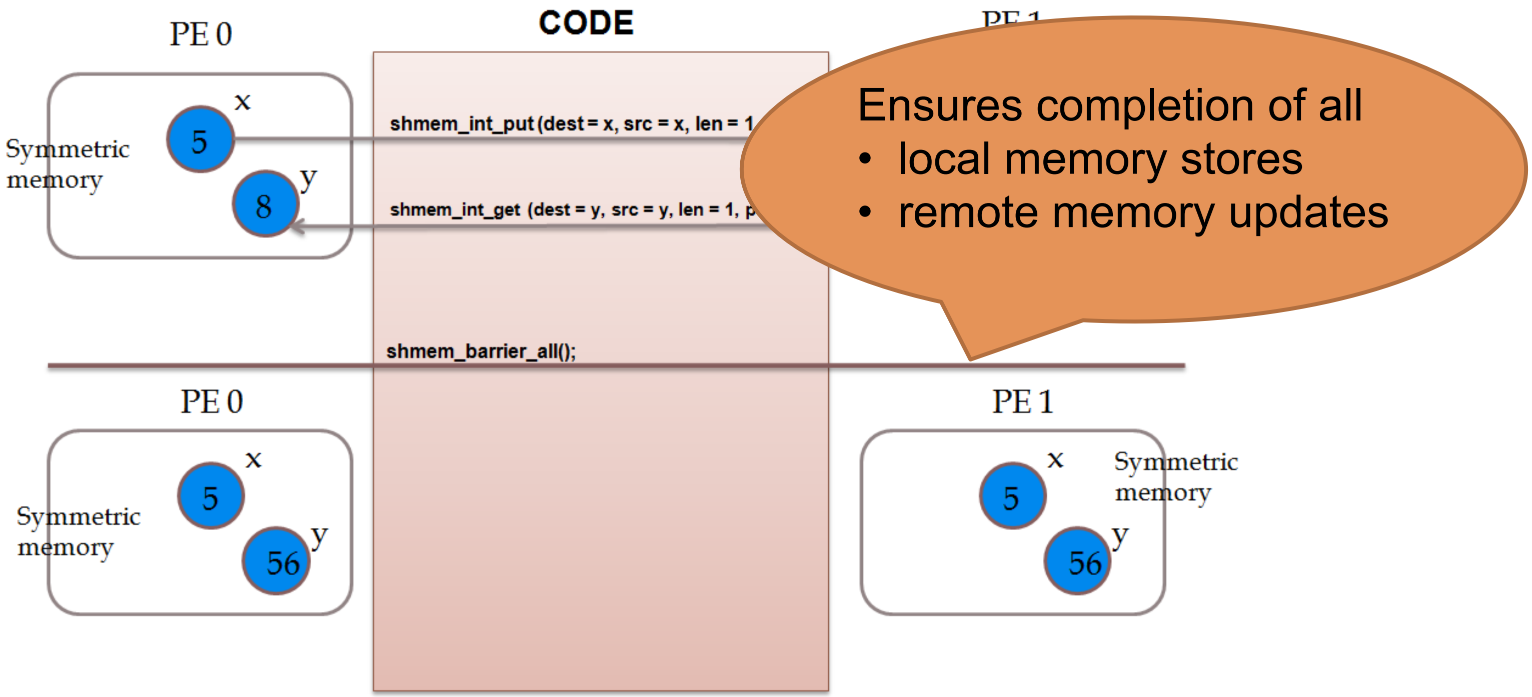
- Quick look at Active Sets
  - Example 1
    - PE_start = 0, logPE_stride = 0, PE_size = 4

    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

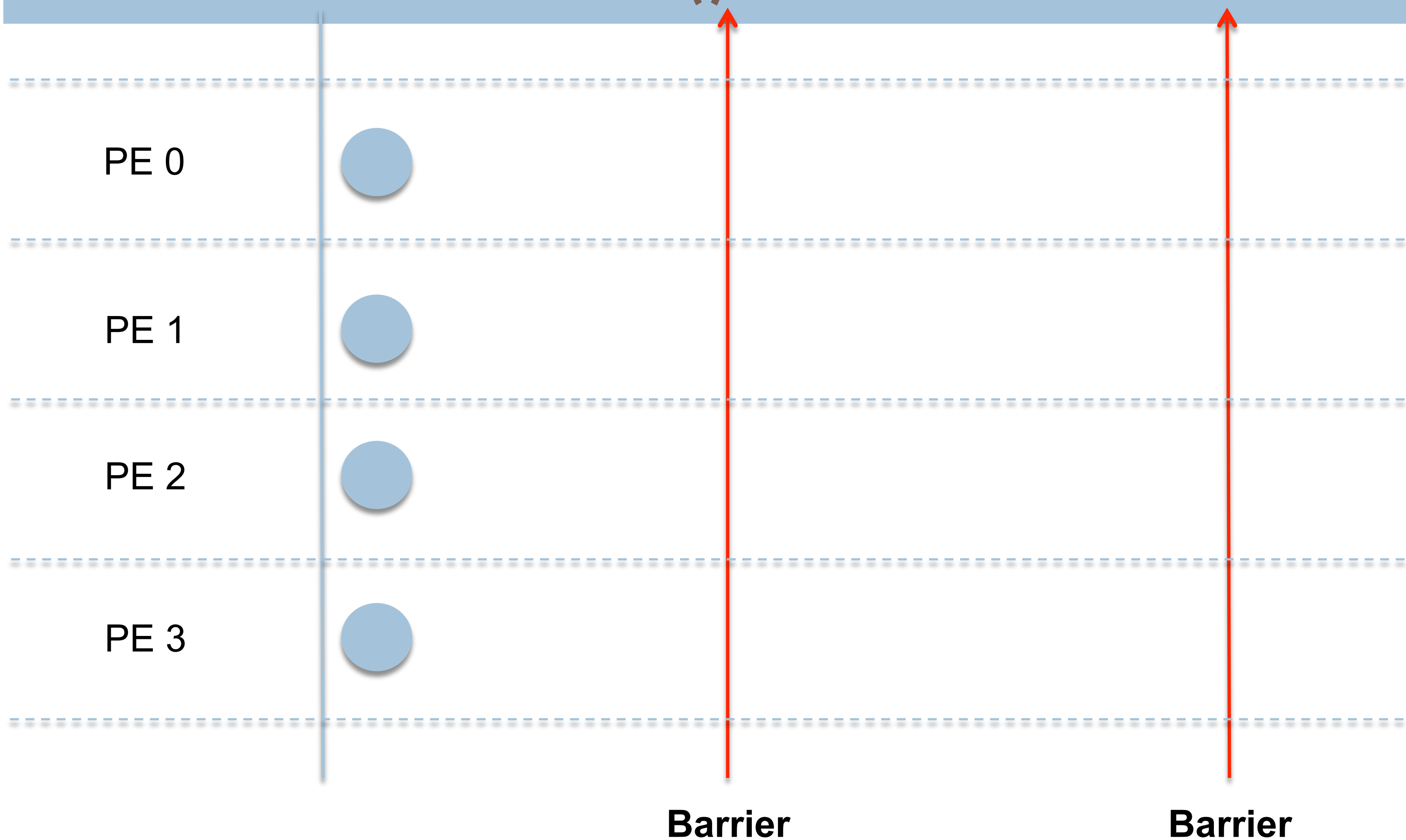
    ACTIVE SET? **PE 2, PE 6, PE 10**

Group Synchronization (1): shmem_barrier_all()

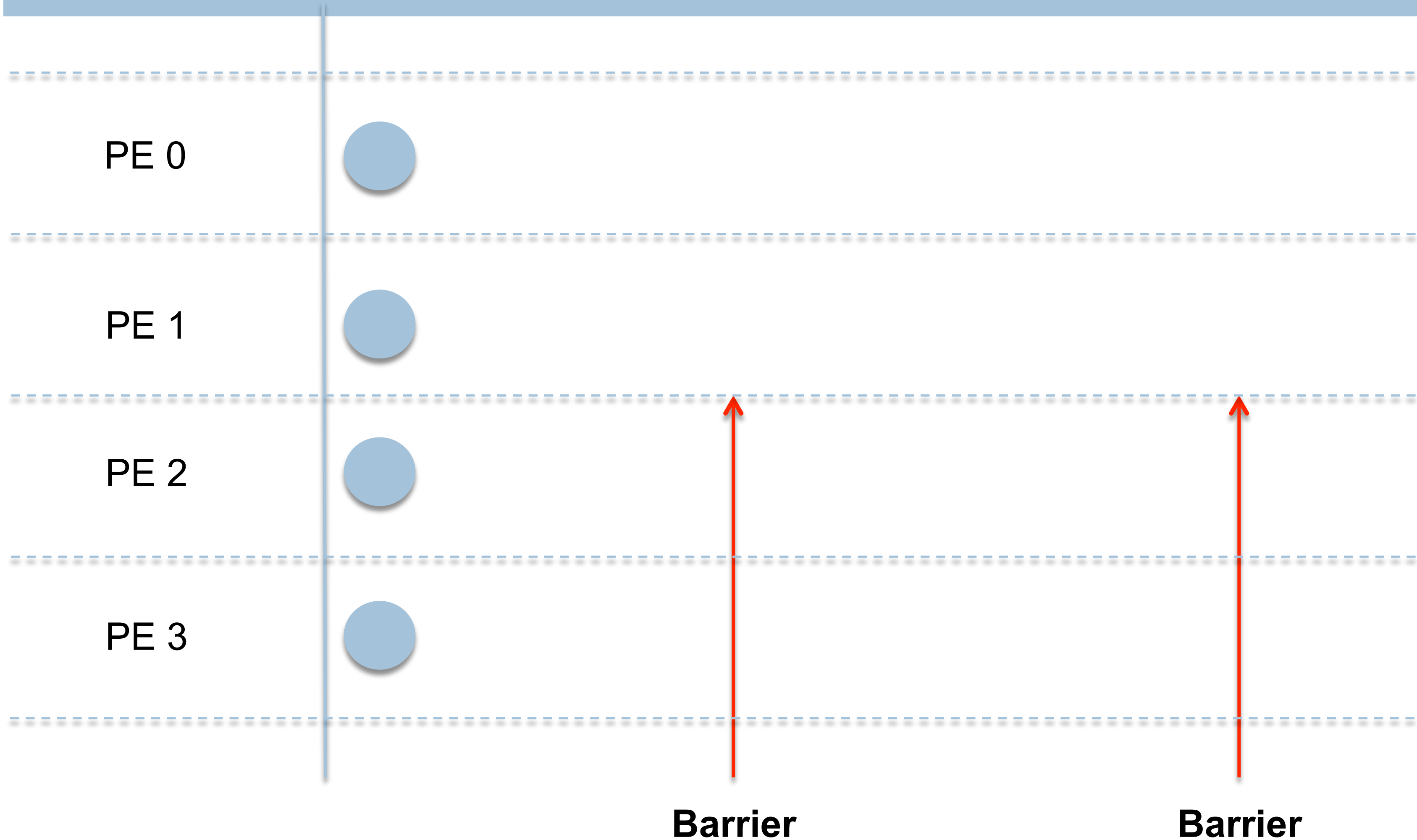


shmem_barrier_all() synchronizes all executing PEs

Group Synchronization (2): shmem_barrier_all()



Group Synchronization (3): shmem_barrier(...)



OpenSHMEM API
Point-to-Point Synchronization (1)

- Point-to-Point synchronization
  - Wait
  - Wait Until
    - Equal, Not equal, Greater than, Less than or equal to, Less than, Greater than or equal to

  - For C: TYPE = double, float, int, long, longdouble, longlong, short
  - For Fortran: TYPE=complex, integer, real, character, logical

OpenSHMEM API
Point-to-Point Synchronization (2)

PE 0

```
…
…
long *dest;
dest = (long *) shmalloc( sizeof(*dest) );
*dest = 9L;
shmem_barrier_all();
….
if (me == 1) {
   shmem_long_w
}
…
if (me == 0) {
  src=101;
  shmem_long_put(dest, &src, 1, 1);
}
shmem_barrier_all();
…
```

PE 1 waits

PE 1

Code snippet showing operation of shmem_wait

OpenSHMEM API
Point-to-Point Synchronization (3)

- Fence (data transfer sync.)
  - Ensures ordering of outgoing write (put) operations to a single PE
  - **void shmem_fence()**

- Quiet (data transfer sync.)
  - Waits for completion of all outstanding remote writes initiated from the calling PE (on some implementations fence = quiet)
  - **void shmem_quiet()**

OpenSHMEM Collective API Broadcast (1)

- One-to-all communication
  - **void shmem_broadcastSS(void *target, void *source, int nelems, int PE_root, int PE_start, int PE_stride, int PE_size, long *pSync)**
  - Storage Size (SS, bits) = 32/4, 64/8

OpenSHMEM Collective API Broadcast (2)

```
…
…
int *target, *source;
target= (int *) shmalloc( sizeof(int) );
source= (int *) shmalloc( sizeof(int) );
*target= 0;
*source= 101;
shmem_barrier_all();
if (me == 1) {
  *source = 2
}
shmem_broadcast32(target, source, 1, 0, 0, 0, 4, pSync);

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

collective operation

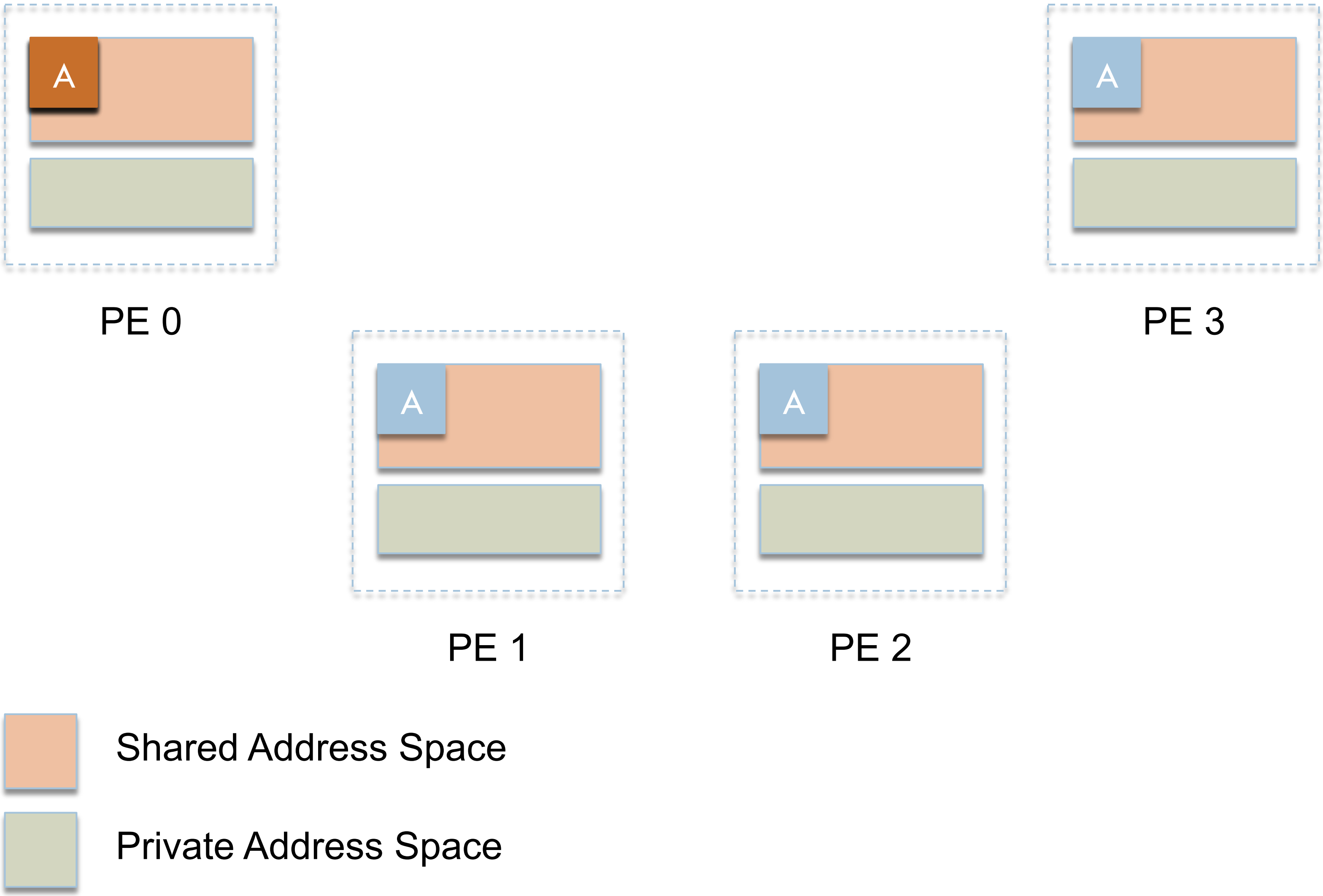
Output

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 operation of shmem_broadcast

OpenSHMEM Collective API Broadcast (3): Working



OpenSHMEM Collective API Broadcast (4): Root & Active Set

- Example 1
  - PE_root = 0, PE_start = 0, logPE_stride = 0, PE_size = 4

  PE 0 broadcasts to PE 1, PE 2 and PE 3
- Example 2
  - PE_root = 2, PE_start = 2, logPE_stride = 0, PE_size = 4

  PE 4 broadcasts to PE 2, PE 3 and PE 5
- Example 3
  - PE_root = 1, PE_start = 0, logPE_stride = 1, PE_size = 4

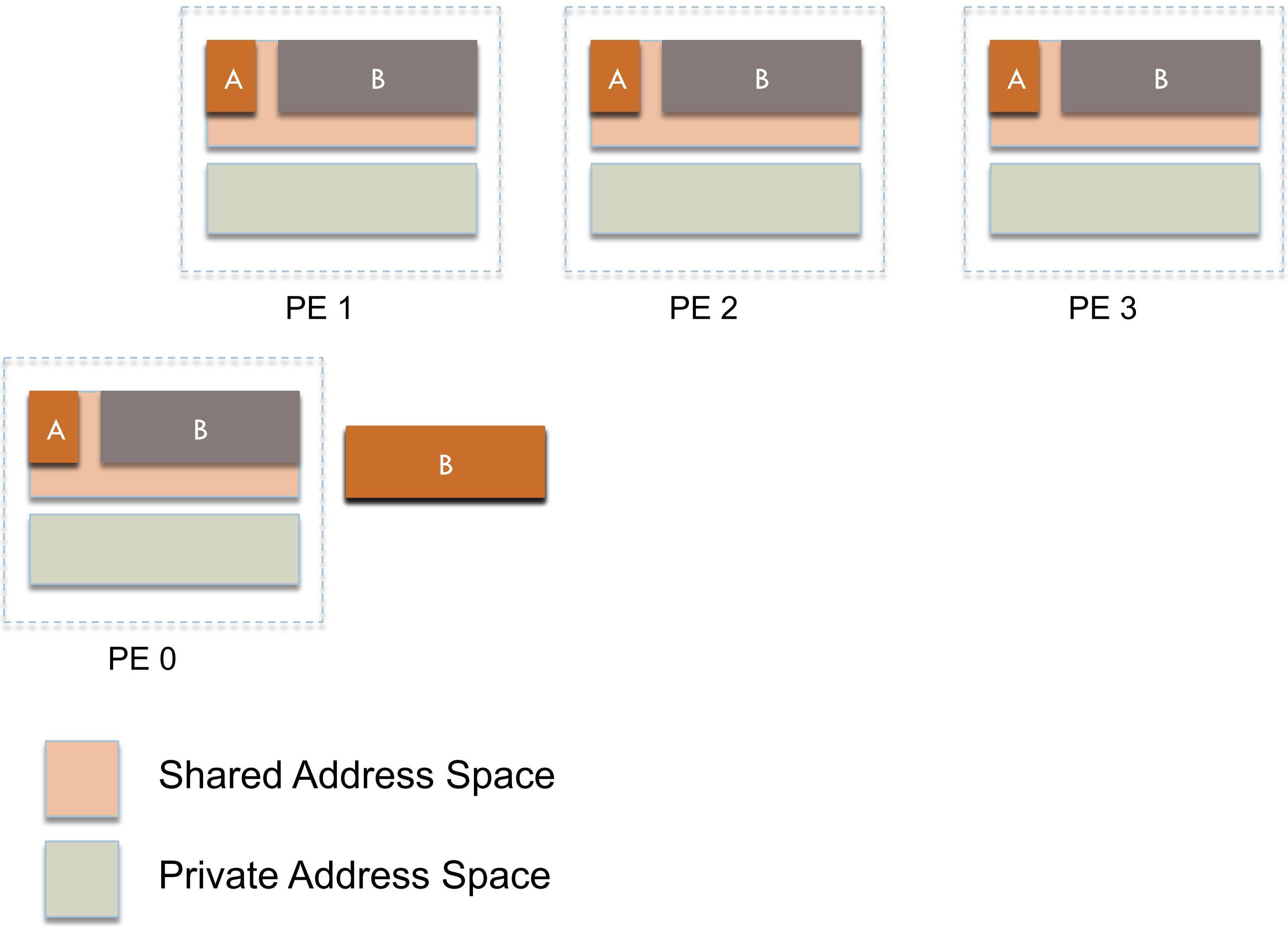
  PE 2 broadcasts to PE 0, PE 4 and PE 6

OpenSHMEM Collective API Collect (1)

Storage Size (SS, bits) = 32, 64 (default)

- Collect
  - Concatenates blocks of data from multiple PEs to an array in every PE
  - **void shmem_collectSS(void *target, void *source, int nelems, int PE_start, int PE_stride, int PE_size, long *pSync)**
  - Storage Size (SS, bits) = 32, 64,128, mem (any size)
- Fixed Collect
  - **void shmem_fcollectSS(void *target, void *source, int nelems, int PE_start, int PE_stride, int PE_size, long *pSync)**

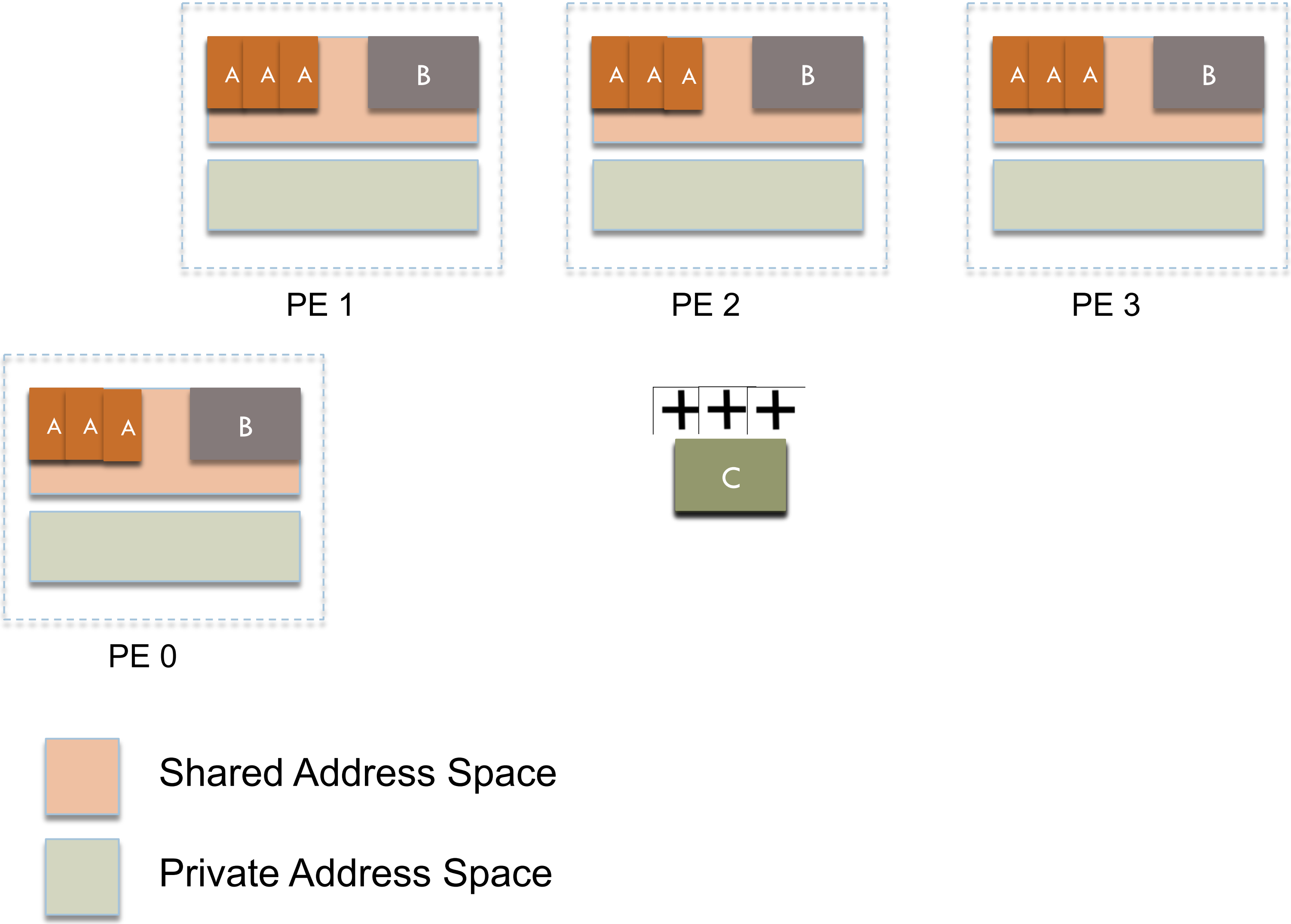
OpenSHMEM Collective API Collect (2): Working of Collect



OpenSHMEM Collective API Reductions (1)

- Logical
  - and, or, xor
- Extrema
  - max, min
- Arithmetic
  - product, sum
- TYPE = int, long, longlong, short

OpenSHMEM Collective API Reductions (2): Working



OpenSHMEM API
Atomic Operations (1)

- 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, short
  - Conditional
    - **TYPE shmem_TYPE_cswap(TYPE *target, int cond, TYPE value, int pe)**
      - TYPE = int, long, longlong, short

- Arithmetic
  - **TYPE shmem_TYPE_OP(TYPE *target, TYPE value, int pe)**
    - OP = fadd, finc
    - TYPE = int, long, longlong, short

OpenSHMEM API
Atomic Operations (2)


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

Output

PE 1: dest = 1, swapped = 0

OpenSHMEM API Accessibility

- **shmem_pe_accessible**
  - Determines whether a processing element (PE) is accessible via SHMEM data transfer operations

- **shmem_addr_accessible**
  - Determines whether an address is accessible via SHMEM data transfers operations from the specified remote processing element (PE)

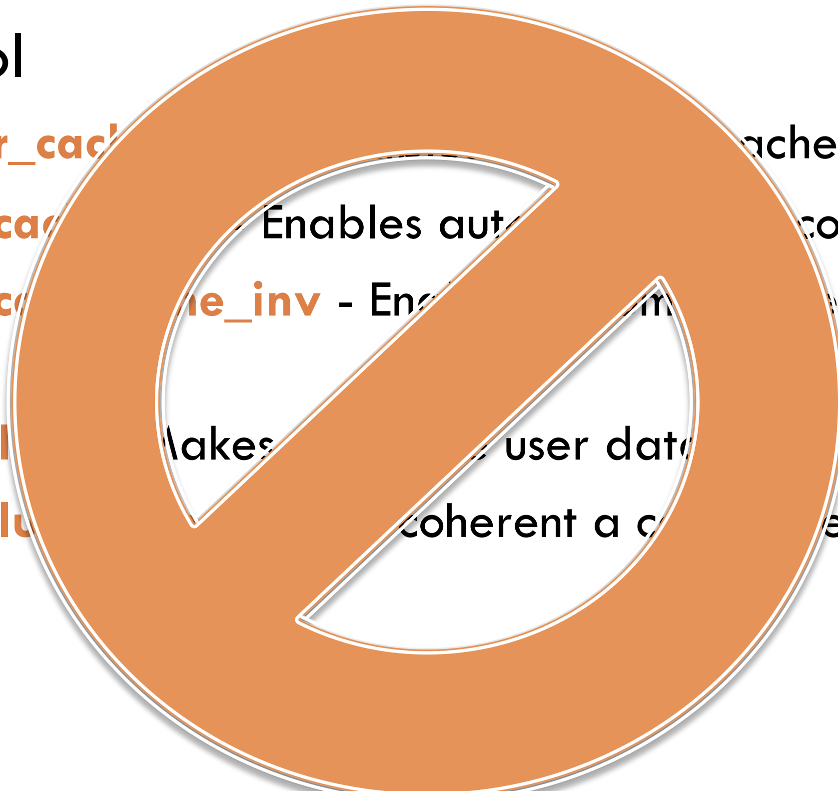
OpenSHMEM API
Mutual Exclusion: Locks

- **Set lock**
  - first-come, first-served manner
- **Clear lock**
  - ensuring that all local and remote stores initiated in the critical region are complete before releasing the lock
- **Test lock**
  - avoid blocking
  - function returns without waiting

OpenSHMEM API
Address Manipulation and Cache

- Address manipulation
  - **shmem_ptr** - Returns a pointer to a data object on a remote PE

- Cache control
  - **shmem_clear_cac** ache coherency mode
  - **shmem_set_cac** Enables aut coherency mode
  - **shmem_set_c e_inv** - En cache coherency mode
  - **shmem_udcfl** akes user dat coherent
  - **shmem_udcflu** coherent a c e



Sequential to Parallel using OpenSHMEM

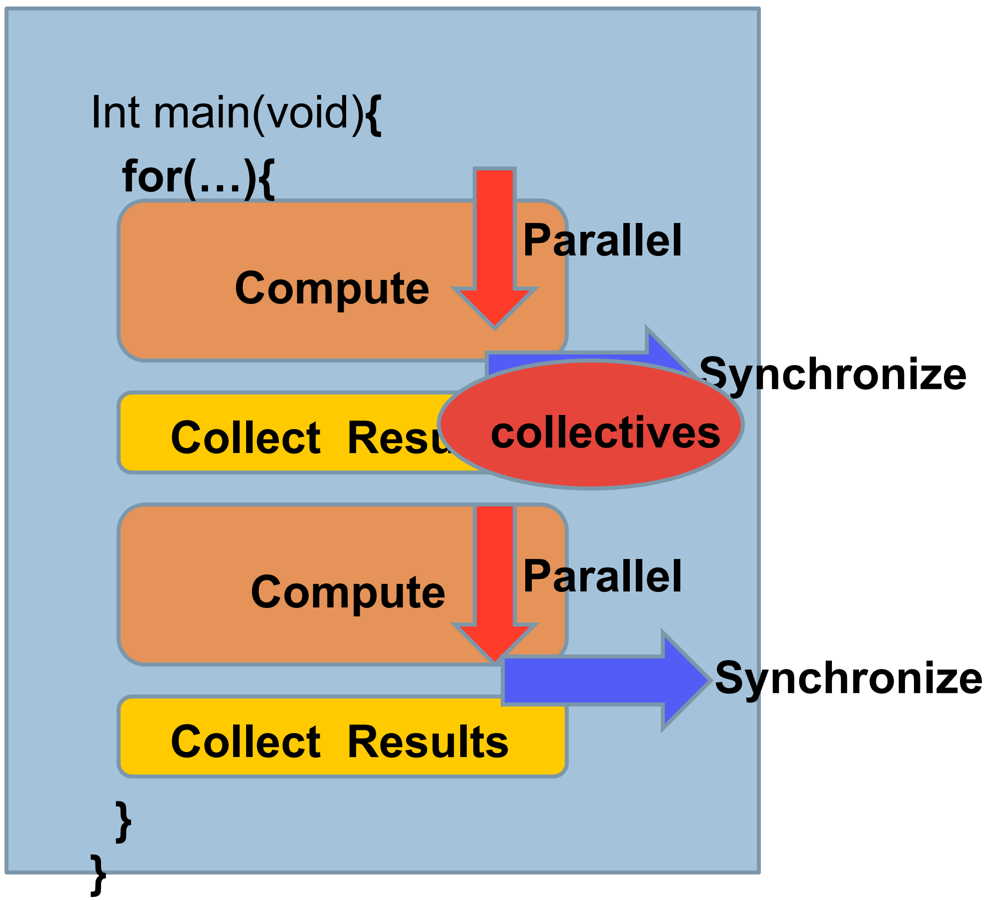
Parallelization using OpenSHMEM Step 1

- Preparation
  - Code Analysis
    - To determine what the code does and how it does it
    - Should fit SPMD style of programming
  - Dependency Analysis
    - Data dependencies
      - True dependency, input dependency
    - Control dependencies
      - To determine the sections of code that can run in parallel and those that must be executed sequentially.

Parallelization using OpenSHMEM Step 2

- Decide what variables need to be symmetric
  - only variables that need to be communicated
- Add shmem API calls for communication and computation
- Add shmem synchronization
  - To insure updates
  - Separate different stages

Parallelization using OpenSHMEM Example



Parallelization using OpenSHMEM Example:SSCA3

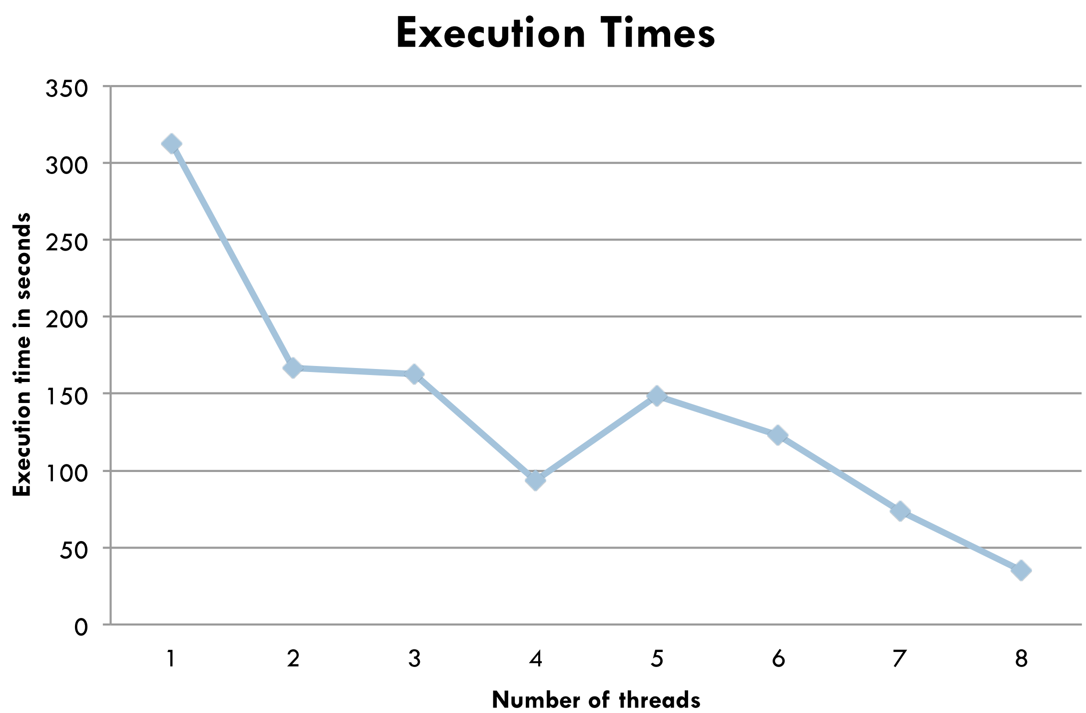
- **Image Processing and Data I/O Application benchmark developed for High Productivity Computing Systems (HPCS).**

- SSCA3 benchmark has essentially two stages;
  - front-end
  - back-end

Parallelization using OpenSHMEM Example:SSCA3, Our observations

	UPC	SHMEM
SYNTACTIC COMPLEXITY	LOW	HIGH
CONCEPTUAL COMPLEXITY	LOW	MEDIUM
MAXIMUM SPEED-UP	6.27	8.94

Parallelization using OpenSHMEM Example:SSCA3



PORTING APPLICATIONS: MPI 1.0 TO SHMEM

MPI 1.0 to OpenSHMEM Incremental Porting

- **Step 1**: Replace initialization calls
- **Step 2**: Replace MPI send-receive pair by a single put/get with appropriate synchronization
- **Step 3**: Replace MPI collective calls with SHMEM collective calls
- **Step 4**: For calls that do not have corresponding OpenSHMEM calls

MPI 1.0 to OpenSHMEM Incremental Porting: Stage1 (2)

Example: Stage 1 (Initialization)

Include shmem.h

```
#include <mpp/shmem.h>
int main(int argc, char *argv[]){
MPI_Init( &argc, &argv );
start_pes(0);
comm_size = _num_pes();
MPI_Comm_rank( MPI_COMM_WORLD, &my_rank );
my_rank = _my_pe();
MPI_Comm_size( MPI_COMM_WORLD, &comm_size );
....
MPI_Finalize();
```

MPI 1.0 to OpenSHMEM Incremental Porting: Unmatched calls


```
MPI_Alltoall( send_count, 1, MPI_INT, recv_count, 1,
MPI_INT, MPI_COMM_WORLD );
```

Replace by

```
for(i=0; i<npes; i++){
    shmem_int_put(&recv_count, &send_count, 1, i);
}
```

MPI 1.0 to OpenSHMEM Incremental Porting: Matrix Multiplication

MPI Code

Distribute blocks of COLUMNS to each process

```
np = size; // number of processes
blocksize = COLUMNS/np; // block size
B_matrix_displacement = rank * blocksize ;
```

Allocate local arrays

```
a_local = (double **)malloc(ROWS*sizeof(dou  le *));
b_local = (double **)malloc(ROWS*sizeof(d
c_local = (double **)malloc(ROWS*sizeof(do
```

Initialize local arrays

```
for(i=0; i<ROWS; i++) {
      a_local[i] = (double *)malloc(blocksize*sizeof(double));
..
}
```

OpenSHMEM Code

Distribute blocks to COLUMNS to each process

```
np = size; // number of processes
blocksize = COLUMNS/np; // block size
B_matrix_displacement = rank * blocksize ;
```

Allocate SHMEM arrays

```
shmem_barrier_all();
 _local = (double **)shmalloc(ROWS*sizeof(double *));
  local = (double **)shmalloc(ROWS*sizeof(double *));
   cal = (double **)shmalloc(ROWS*sizeof(double *));
```

Initialize arrays

```
for(i=0; i<ROWS; i++) {
      a_local[i] = (double
      *)shmalloc(blocksize*sizeof(double));
…

}
```

MPI 1.0 to OpenSHMEM Incremental Porting: Matrix Multiplication

MPI Code

Send the Local block of matrix a to process on right

```
    MPI_Barrier(MPI_COMM_WORLD);
    if(rank == np-1)
        MPI_Isend (&a_local[i][0],blocksize,MPI_DOUBLE,
    0,
        1,MPI_COMM_WORLD,&req[0]);
    else
        MPI_Isend (&a_local[i]
    [0],blocksize,MPI_DOUBLE,rank+1,
        1,MPI_COMM_WORLD,&req[1]);
    if(rank == 0)
        MPI_Recv(&a_local[i]
    [0],blocksize,MPI_DOUBLE,np-1,
        1,MPI_COMM_WORLD,&status);
    else
        MPI_Recv(&a_local[i]
    [0],blocksize,MPI_DOUBLE,rank-1,
        1,MPI_COMM_WORLD,&status);
```

Compute the local displacement

…

OpenSHMEM Code

Send the Local block of matrix 'a' to process on right

```
    shmem_barrier_all();
    if(rank == np-1)
      shmem_double_put(&a_local[i][0],&a_local[i]
    [0],blocksize,0);
    else
        shmem_double_put(&a_local[i][0],&a_local[i]
    [0],blocksize,rank+1);
    shmem_barrier_all();
```

Compute the local displacement (REMAINS SAME AS MPI)

…

MPI 1.0 to OpenSHMEM Direct Replacement (1)

MPI calls	Possible OpenSHMEM calls
MPI_Init(&argc, &argv)	start_pes(0)
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank)	_my_pe()
MPI_Comm_size(MPI_COMM_WORLD, &comm_size)	_num_pes()
MPI_Barrier(comm)	shmem_barrier_all()
MPI_Allreduce(bucket_size, bucket_size_totals, SIZE, MPI_INT, MPI_SUM, MPI_COMM_WORLD)	shmem_int_sum_to_all(bucket_size_totals,bucket_size,SIZE, 0,0,comm_size,ipWrk,pSync)
MPI_Bcast(It, 1, MPI_INTEGER, 0, MPI_COMM_WORLD)	shmem_broadcast4(It, It, 1, 0, 0, 0, nprocs, pSync)

MPI 1.0 to OpenSHMEM Direct Replacement (2)

MPI calls	Possible OpenSHMEM calls
MPI_Send(send_buff, buff_len,MPI_DOUBLE,to_rank....)	shmem_double_put(recv_buff,send_buff,buff_len, to_rank)
MPI_Recv(recv_buff, buff_len,dp_type, from_rank....)	shmem_double_get(recv_buff,send_buff,buff_len, from_rank)
MPI_Wait(request,status)	shmem_wait(variable, value)
MPI_reduce(t, tmax, 1,MPI_REAL, MPI_MAX,root, mpi_comm_world)	shmem_int_max_to_all(tmax,t, 1,0,0,nprocs,pwrk,psync)
MPI_Scatter(src,count,MPI_INT,dst,count, MPI_INT, 0, comm_world)	shmem_broadcast(dst, src, count, 0, 0, 0, size, pSync)
MPI_Gather(src,count,MPI_INT,dst,count, MPI_INT, 0, comm_world)	shmem_collect32(dst, src, count, 0, 0, 0, size, pSync)

MPI 1.0 to OpenSHMEM
Equivalent OpenSHMEM calls

MPI calls	Possible OpenSHMEM calls

MPI_AlltoAll	for(j1=0;j1<comm_size;j1++){
	shmem_int_put(&recv_count[my_rank],
	&send_count[j1],1,j1);
	}

MPI_AlltoAllv	for(j1=0;j1<comm_size;j1++){
	int k1 = send_displ[j1];
	static int k2;
	shmem_int_get(&k2,&recv_displ[my_rank]
	,1,j1);
	shmem_int_put(key_buff2+k2,key_buff1+k
	1,send_count[j1],j1);
	}

MPI_Comm and MPI_Group calls	NA
MPI_Finalize	NA

78

OpenSHMEM vs. MPI 2.0

OpenSHMEM vs. MPI 2.0 Symmetric memory allocation

Collective Call

MPI_Win_create(var1, window1)

MPI_Win_create(var2, window2)

Process 0	**Process1**
……	…..
MPI_Put(window1)	MPI_Put(window2)
…….	……

MPI Window semantics

- All processes which intend to use the window must participate in window creation
- Many or all the local allocations/objects should be coalesced within a single window creation.

Symmetric Data

Global variables

Static local or global variables

shmalloc() memory

Process 0	**Process 1**
shmem_get(&var1)	shmem_put(&var2)
…….	………

SHMEM semantics

- All global and static data are by default accessible to all process.
- Local allocations/objects can be made remotely accessible using shmalloc instead of malloc

OpenSHMEM vs. MPI 2.0 Synchronization (1)

Process 0 (Source)	Process 1 (Dest)
MPI_Fence	MPI_Fence
……	……
If(rank==0)	If(rank==0)
MPI_Put	MPI_Put
…….	…….
MPI_Fence	MPI_Fence

MPI_Win_fence

- Fence is a collective call.
- Need 2 fence calls, one to separate and another one to complete.
- So it mostly functions like barrier

Process 0 (Source)	Process 1 (Dest)
shmem_fence()	shmem_fence()
……	……
shmem_put()	shmem_put()
shmem_fence()	shmem_fence()
shmem_put()	shmem_put()
…….	…….

shmem_fence

- shmem fence is just meant for ordering of puts.
- It does not separate the processes nor does it mean completion
- Ensures there are no pending puts to be delivered to the same target before the next put

OpenSHMEM vs. MPI 2.0 Synchronization (2)

Process 0 (Source)	Process 1 (Dest)
MPI_Start	MPI_Post
……	….
MPI_Put	Cannot do anything
MPI_Put	
…….	…….
MPI_Complete	MPI_Wait

- Point to point synchronization.
- Sender does Start and waits for Post from receiver
- The receiver does Post and waits for the data.
- The sender Puts the data and signals completion to receiver

Process 0 (Source)	Process 1 (Dest)
shmem_put(data)	shmem_wait(flag1)
shmem_put(flag1)	……
shmem_wait(flag2)	shmem_put()
…….	shmem_fence()
…….	shmem_put(flag2)
	…….

- The receiver can directly wait for the data using shmem_wait on a event flag.
- The sender puts the data and sets the event flag to signal the receiver.
- Both post and complete are implicit inside the wait and put operation.

OpenSHMEM vs. MPI 2.0 Synchronization (3)

Process 0 (Source)

```
If(rank == 0) {
MPI_Win_lock
MPI_Put
MPI_Win_unlock
}
```

Process 1 (Dest)

```
If(rank == 0) {
MPI_Win_lock
MPI_Put
MPI_Win_unlock
}
```

- No mutual exclusion
- Lock is not real lock, but begin RMA
- Unlock means end RMA
- Only the source calls lock

Process 0 (Source)

```
shmem_set_lock
……..
shmem_put(data)
…….
shmem_clear_lock()
```

Process 1 (Dest)

```
shmem_set_lock
…….
shmem_put(data)

shmem_clear_lock()
```

- Enforces mutual exclusion
- The PE which acquires lock does put
- The waiting PE gets the lock on first come first served basis

OpenSHMEM vs. MPI 2.0
Difficulties using MPI 2.0

- Window creation is a collective operation
- May restrict the use of passive-target RMA operations to only work on memory allocated using MPI_Alloc_mem
- It is erroneous to have concurrent conflicting RMA get/put (or local load/store)
- Multiple windows are allowed to include overlapping memory regions, however it is erroneous to use concurrent operations to distinct overlapping windows

OpenSHMEM and Hardware

- OpenSHMEM is intended to be a specification that
  - Standardizes current efforts
  - Doesn't restrict implementors
- Want to allow freedom for innovation on hardware
  - E.g. collectives/atomics on NICs
  - Emerging manycore architectures
    - MIC, Bluegene/Q
    - Embedded systems with DMA engines
  - Heterogeneous architectures
    - E.g. Convey, "ceepee-geepee"

References

1. Hongzhang Shan and Jaswinder Pal Singh, *A Comparison of MPI, SHMEM and Cache-coherent Shared Address Space Programming Models on the SGI Origin2000*

2. SHMEM tutorial by Hung-Hsun Su, HCS Research Laboratory,University of Florida

3. Evaluating Error Detection Capabilities of UPC Compilers and Runtime Error detection by Iowa Sate University http://hpcgroup.public.iastate.edu/CTED/

4. *Quadrics SHMEM Programming Manual* http://www.psc.edu/~oneal/compaq/ShmemMan.pdf

5. Glenn Luecke et. al., *The Performance and Scalability of SHMEM and MPI-2 One-Sided Routines on a SGI Origin 2000 and a Cray T3E-600* http://dsg.port.ac.uk/Journals/PEMCS/papers/paper19.pdf

6. Patrick H. Worley, *CCSM Component Performance Benchmarking and Status of the CRAY X1 at ORNL* http://www.csm.ornl.gov/~worley/talks/index.html

7. Karl Feind, *Shared Memory Access (SHMEM) Routines*

8. *Galen M. Shipman and Stephen W. Poole, Open-SHMEM: Towards a Unified RMA Model*

Thanks!

Questions?