SHMEM TUTORIAL

Presenters: Swaroop Pophale and Tony Curtis

University of Houston, Texas

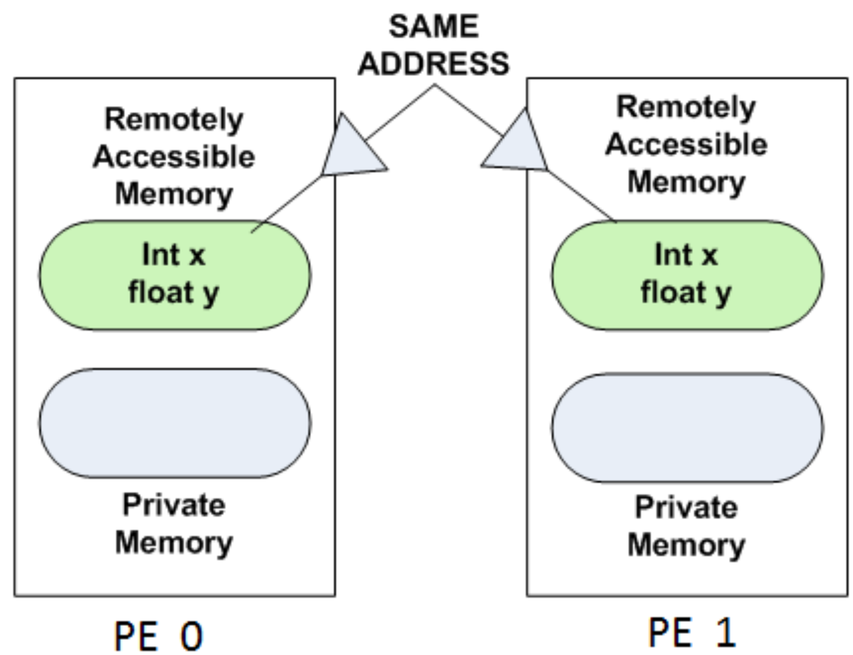
Outline

- ✓ Background
- ✓ History and Implementations
- ✓ SHMEM routines
- ✓ Getting started
  - ✓ Code Example
  - ✓ Closer look
- ✓ Performance
- ✓ Conclusions
- ✓ References

Background
What is SHMEM?

- SHared MEMory library (SPMD model)
  - Library of functions similar to MPI (e.g. *shmem_get()*)
- Available for C / 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).
- Shmem routines supply remote data transfer, work-shared broadcast and reduction, barrier synchronization, and atomic memory operations.

- Symmetric Variables
  - Arrays or variables that exist with the same size, type, and relative address on all PEs.
  - Data allocated and managed by shmem
  - C
    - Non-stack variables
      - Global
      - Local static
  - Fortran
    - Variables in common blocks
    - Variables with the SAVE attribute



History and Implementations

- Cray SHMEM
  - SHMEM first introduced by Cray Research Inc. in 1993 for Cray T3D
  - Platforms: Cray T3D, T3E, PVP, XT series
- SGI SHMEM
  - SGI bought CRI and SHMEM was incorporated in SGI's Message Passing Toolkit (MPT)
  - Owns the "rights" for SHMEM
  - Platform support: SGI Irix, Origin, Altix
  - SGI was bought by Rackable Systems in May 2009
- Quadrics SHMEM (company out of business)
  - Optimized API for QsNet
  - PSHMEM support available via joint effort from HCS Lab & Quadrics
  - Platform: Linux cluster with QsNet interconnect
- Others
  - HP SHMEM, IBM SHMEM (used internally only)
  - GPSHMEM (cluster with ARMCI & MPI support, dead)

Note: SHMEM is not defined by any one standard.

SHMEM Routines

- **Data transfers**
  - One sided *puts* and *gets*
- **Synchronization mechanisms**
  - Barrier, Fence, quiet
- **Collective communication**
  - Broadcast, Collection, Reduction
- **Atomic Memory Operations**
  - Provide mechanisms to implement mutual exclusion
  - Swap, Add, Increment
- **Address Manipulation, Data Cache control and Locks**
  - Not supported by all SHMEM implementations

Getting Started

- Initialization
  - Include header shmem.h to access the library
    - E.g. #include <shmem.h> , #include <mpp/shmem.h>
  - start_pes, shmem_init: Initializes the caller and then synchronizes the caller with the other processes.
  - 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		

Implementation Comparison

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;
}
```

Implementation Differences

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;
}
```

Closer Look
Data Transfer (1)

- Put
  - Single variable
    - **void shmem_TYPE_p(TYPE *addr, TYPE value, int pe)**
      - TYPE = double, float, int, long, short
  - Contiguous object
    - **void shmem_put(void *target, const void *source, size_t len, int pe)**
    - **void shmem_TYPE_put(TYPE *target, const TYPE*source, size_t len, 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 len, int pe)**
      - Storage Size (SS, bits) = 32, 64,128, mem (any size)

Data Transfer (2)

- Get
  - Single variable
    - **void shmem_TYPE_g(TYPE *addr, 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_get(void *target, const void *source, size_t len, int pe)**
    - **void shmem_TYPE_get(TYPE *target, const TYPE*source, size_t len, 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 len, int pe)**
      - Storage Size (SS, bits) = 32, 64,128, mem (any size)

Synchronization (1)

- Barrier (Group synchronization)
  - *pSync is a* symmetric work array used to prevent overlapping collective communication
  - **void shmem_barrier_all()**
    - Suspend all operations 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

- Conditional wait (P2P synchronization)
  - Generic conditional wait
    - Suspend until local shared 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 = double, float, int, long, longdouble, longlong, short
      - For Fortran: TYPE=complex, integer, real, character, logical

Synchronization (2)

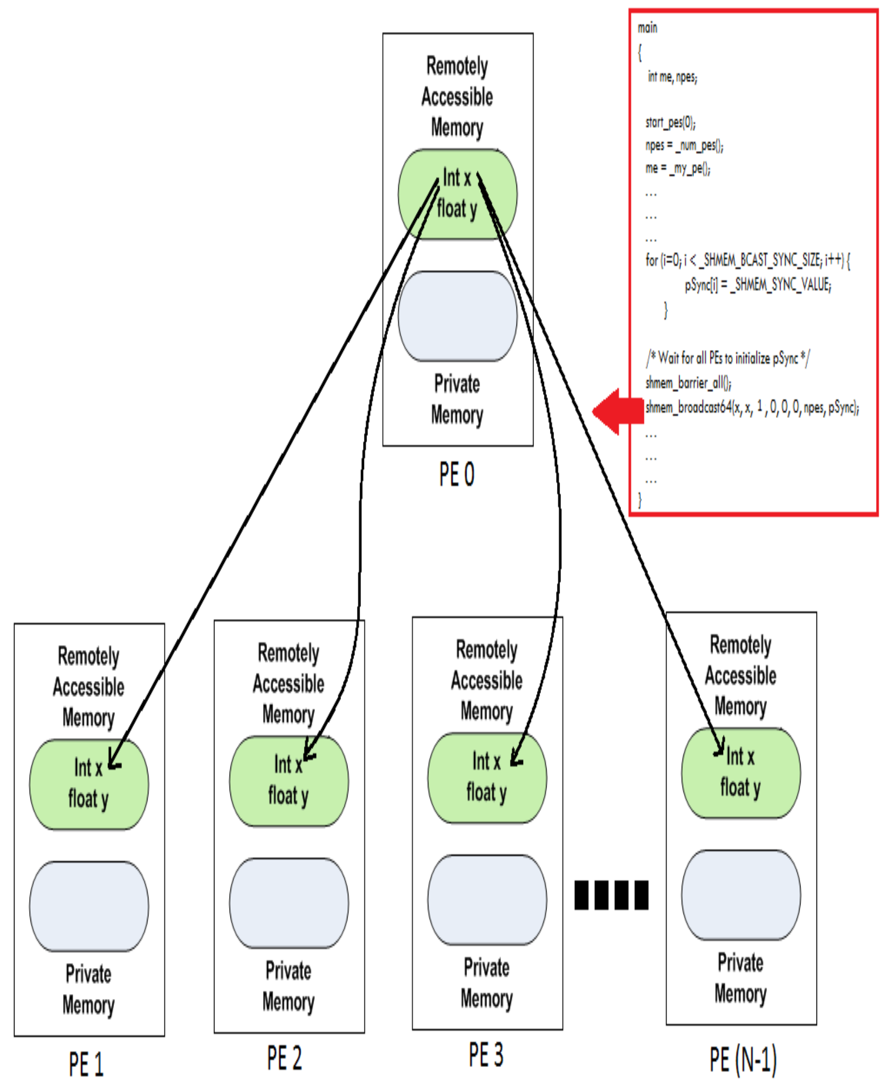
- Specific conditional wait
  - Similar to the generic wait except the comparison can now be >=, >, =, !=, <, <=
  - **void shmem_wait_until(long *var, int cond, long value)**
  - **void shmem_TYPE_wait_until(TYPE *var, int cond, TYPE value)**
    - TYPE = int, long, longlong, short

- 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()**

Collective Communication (1)

- Broadcast
  - One-to-all communication
  - **void shmem_broadcast(void *target, void *source, int nlong, int PE_root, int PE_start, int PE_stride, int PE_size, long *pSync)**
  - **void shmem_broadcastSS(void *target, void *source, int nlong, int PE_root, int PE_start, int PE_stride, int PE_size, long *pSync)**



Collective Communication (2)

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

- Collection
  - Concatenates blocks of data from multiple PEs to an array in every PE
  - **void shmem_collect(void *target, void *source, int nlong, int PE_start, int PE_stride, int PE_size, long *pSync)**
  - **void shmem_collectSS(void *target, void *source, int nlong, int PE_start, int PE_stride, int PE_size, long *pSync)**

- Reductions
  - Logical, Statistical and Arithmetic
    - **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, Statistical OP = max, min, Arithmetic OP = product, sum
      - TYPE = int, long, longlong, short

Atomic Operations

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

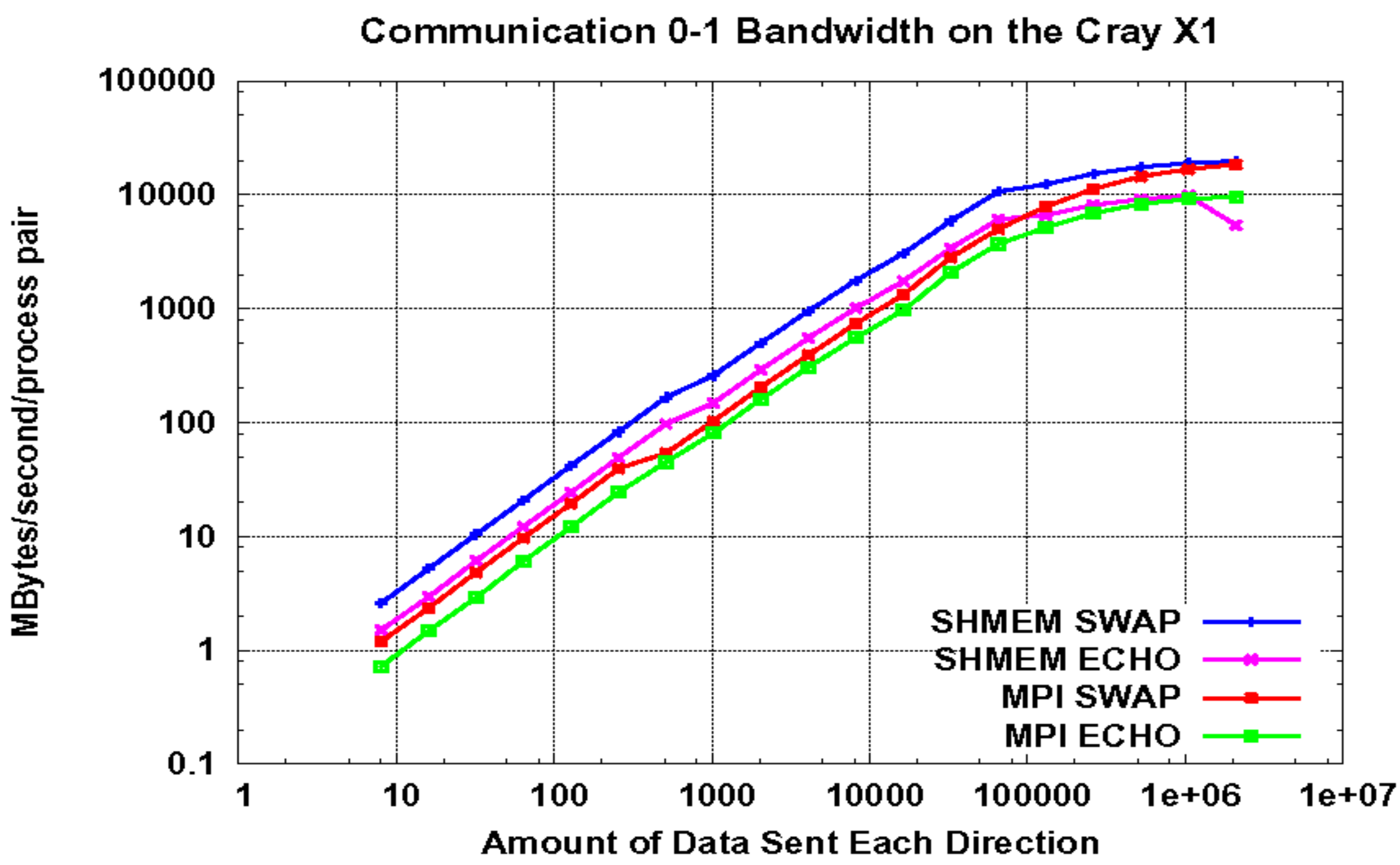
- Arithmetic
  - **TYPE shmem_TYPE_OP(TYPE *target, TYPE value, int pe)**
    - OP = fadd, finc
    - TYPE = int, long, longlong, short

Addresses & Cache

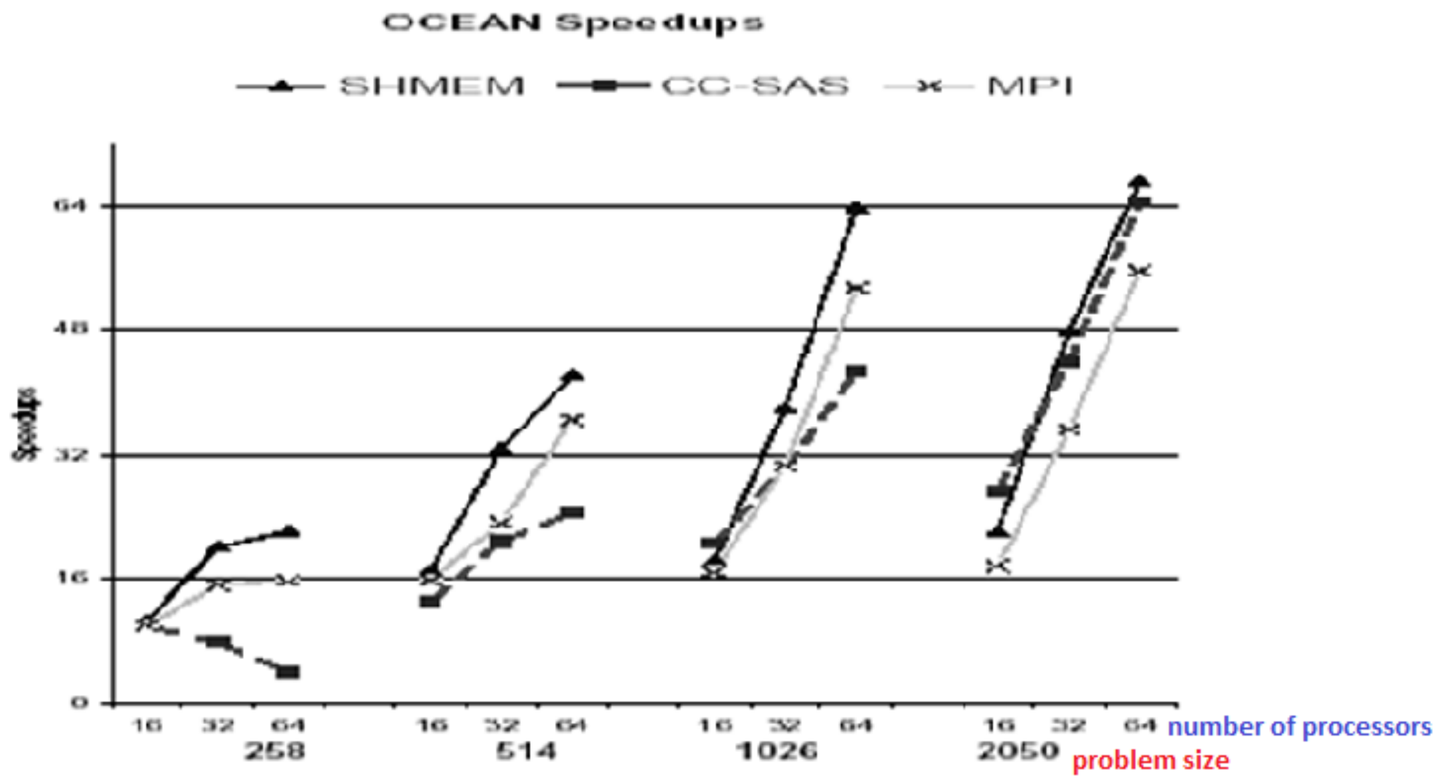
- Address manipulation
  - **shmem_ptr** - Returns a pointer to a data object on a remote PE

- Cache control
  - **shmem_clear_cache_inv** - Disables automatic cache coherency mode
  - **shmem_set_cache_inv** - Enables automatic cache coherency mode
  - **shmem_set_cache_line_inv** - Enables automatic line cache coherency mode
  - **shmem_udcflush** - Makes the entire user data cache coherent
  - **shmem_udcflush_line** - Makes coherent a cache line

Performance – Bandwidth



Performance – Speedups



On SGI Origin 2000

Conclusions

- **Pros**
  - Simpler one-sided style of communication
  - Can take advantage of high performance interconnects
    - low latency
    - hardware assist; e.g. rDMA, collective support, remote CPU not interrupted during transfers
- **Cons**
  - Not standardized
    - Different implementation have different APIs
    - Effort underway to develop a standardization.

Summary and Related Work

SHMEM

- Library for C and Fortran programs
- Provides calls for data transfer, collective operations, synchronization and atomic operations
- Requires explicit put/get calls to communicate using symmetric data

UPC

- Language extension for ANSI C
- Provides extensions for declaring global shared variables, communicating global shared variables, synchronization and work sharing
- No syntactic difference between accesses to a shared and accesses to a private variable

Summary and Related Work

- Related & Future Work
  - Compiler side
    - Develop SHMEM-aware compilers and tools to analyze source code
    - E.g. code-motion to provide better communication/computation overlaps, transfer coalescing…
  - Runtime
    - Error detection, recovery

- Related Work, e.g. from Iowa State:
  - Compiler side
    - Evaluating Error Detection Capabilities of UPC Compilers
  - Runtime
    - Error detection, recovery

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 Leucke et. al., *The Performance and Scalability of SHMEM and MPI-2 One-Sided Routines on a SCI 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 for reading!