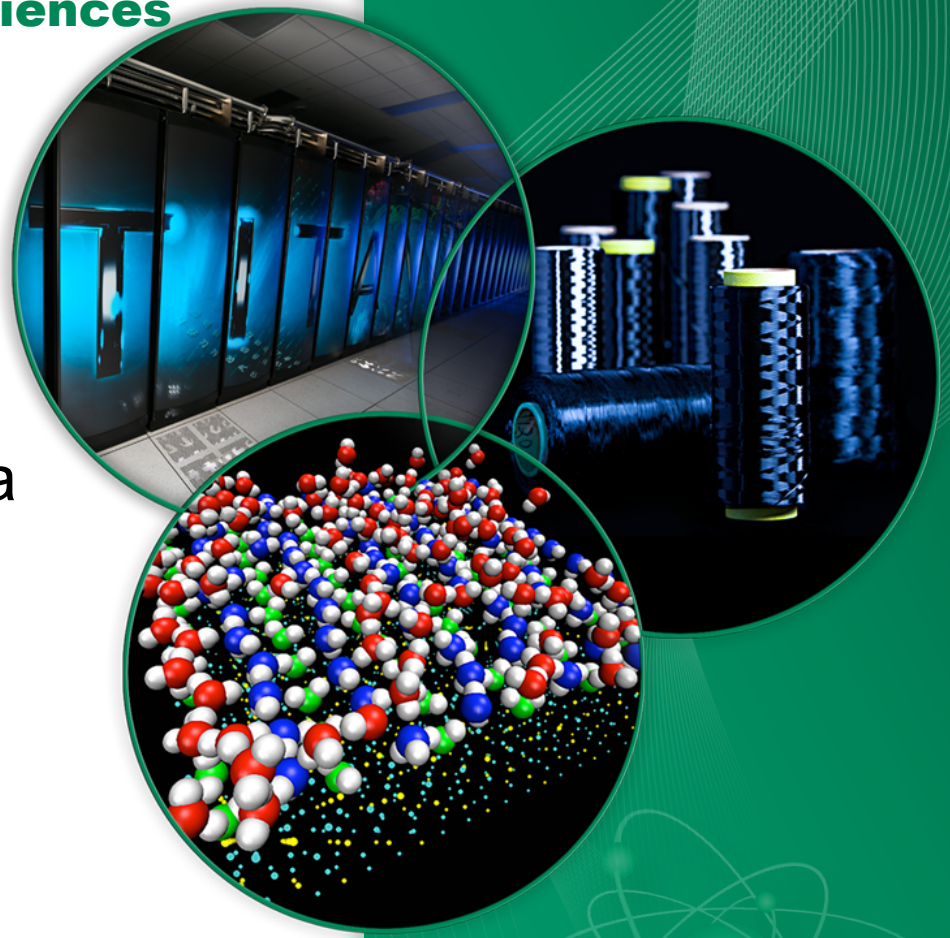
Oak Ridge National Laboratory
Computing and Computational Sciences



OpenSHMEM Extensions and a Vision for Its Future Direction

Manjunath Gorentla Venkata

Authors

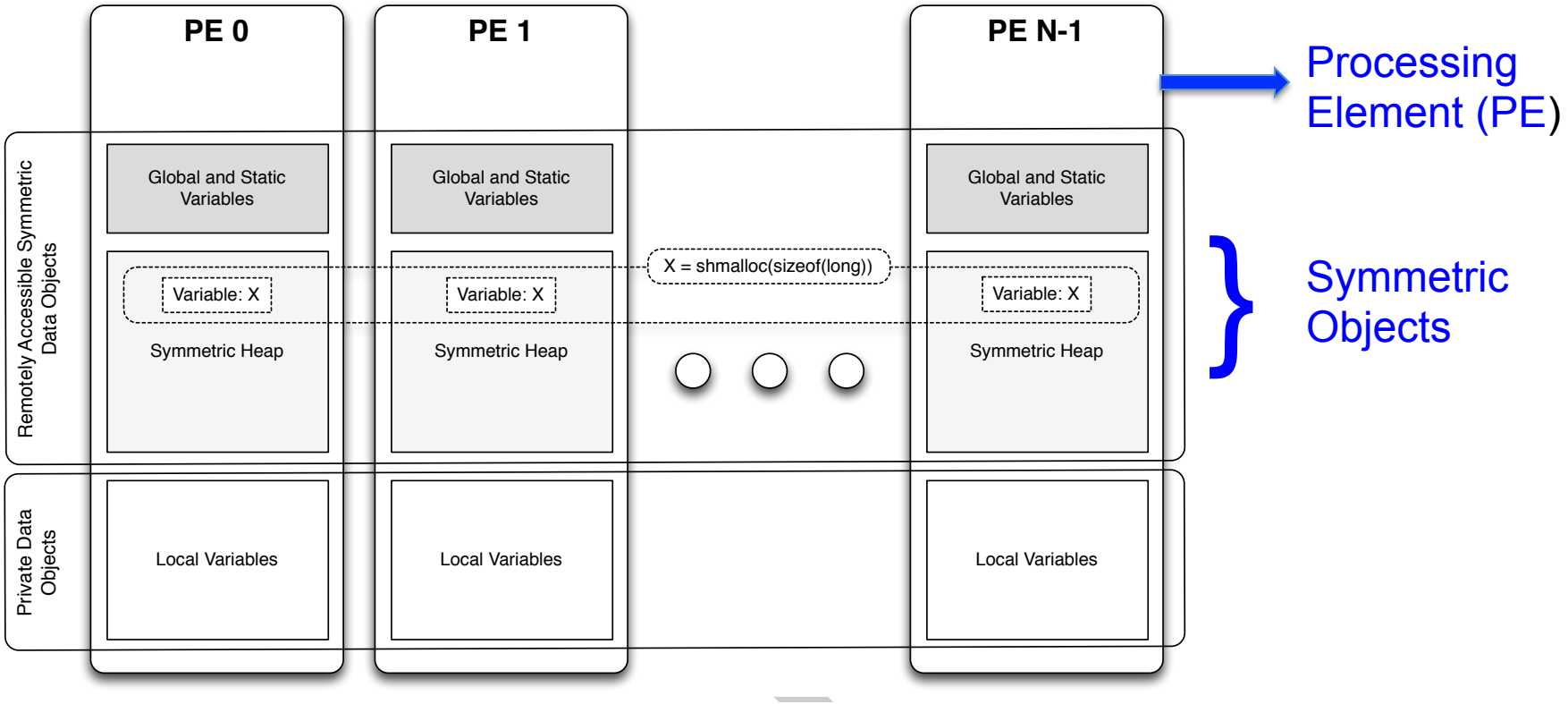
- Tony Curtis
- Manjunath Gorentla Venkata
- Oscar Hernandez
- Chung-Hsing Hsu
- Gregory Koenig
- Jens Manser
- Steve Poole
- Nick Parks
- Swaroop Pophale
- Pavel Shamis
- Donald Aaron Welch

These are some initial ideas to start the conversation…

What is OpenSHMEM ?

- OpenSHMEM is a PGAS library interface specification
- Two versions
  - 1.0: Derivate of SGI SHMEM man pages (released in 2011)
  - 1.1(Draft): Errata, improve readability, and clarify semantics (open for comments)

OpenSHMEM: Basic Concepts



- Provides routines enabling parallel programming
- Parallel processes are called Processing Element(PE)s
- Symmetric objects have same address on all PEs
  - Static and global variables
  - Symmetric Heap: Memory allocated by the shmalloc routine

OpenSHMEM Routines

- Remote Memory
- Atomic Memory
- Collective
- Point-to-point Synchronization
- Point-to-point Ordering
- Distributed Locking

What does OpenSHMEM Lack ?

- Light-weight generic group creation routines
- Non-blocking routines
  - Collectives, Remote Memory, Atomics ..
- Graceful library shutdown
- Error codes, Threading model, Communication context, I/O interfaces, Fault tolerance, Profiling interfaces and more ..

Explicit Active Sets

- Active sets: A lightweight “abstraction” to express a group of PEs on which collective operations are executed
- Limitations:
  - Supports only *log* strides
  - Lifetime is limited to only one collective call
- Explicit Active Sets (ASET) (Extension)
  - Active sets are encapsulated in an opaque object
  - Lifetime extended beyond a single collective call
  - Supports more than log strides between the PEs in an Active Set

Example Interfaces for Creating and Using ASET

Create a strided active set.	shmem_aset *shmem_create_strided_aset(int PE_start, int PE_stride, int PE_size)
Create a log-strided active set.	shmem_aset *shmem_create_log_strided_aset(int PE_start, int PE_log_stride, int PE_size, int stride_base)
Create a user-defined active set.	shmem_aset *shmem_create_custom_aset(int PE_start, shmem_offset_fn offset, int PE_size, void *const_params)
Create a generic active set.	shmem_aset *shmem_create_generic_aset(int *PE_list, int PE_size)
Check if a PE is in an active set.	int shmem_in_aset(shmem_aset *aset)
Query the size of an active set.	int shmem_aset_size(shmem_aset *aset)
Delete an active set.	void shmem_delete_aset(shmem_aset *aset);

Creating ASET using the Stride Interface

```
1  int main(int argc, char **argv) {
2      shmem_aset *aset;
3      ...
4      /* creates an active set containing PEs 0, 3, 6, 9, ... */
5      aset = shmem_create_strided_aset(0, 3, npes / 3);
6      /* equivalent to the OpenSHMEM 1.0-style (me % 3 == 0) */
7      if (shmem_in_aset(aset)) {
8          shmem_barrier_aset(aset, pSync);
9      }
10 }
```

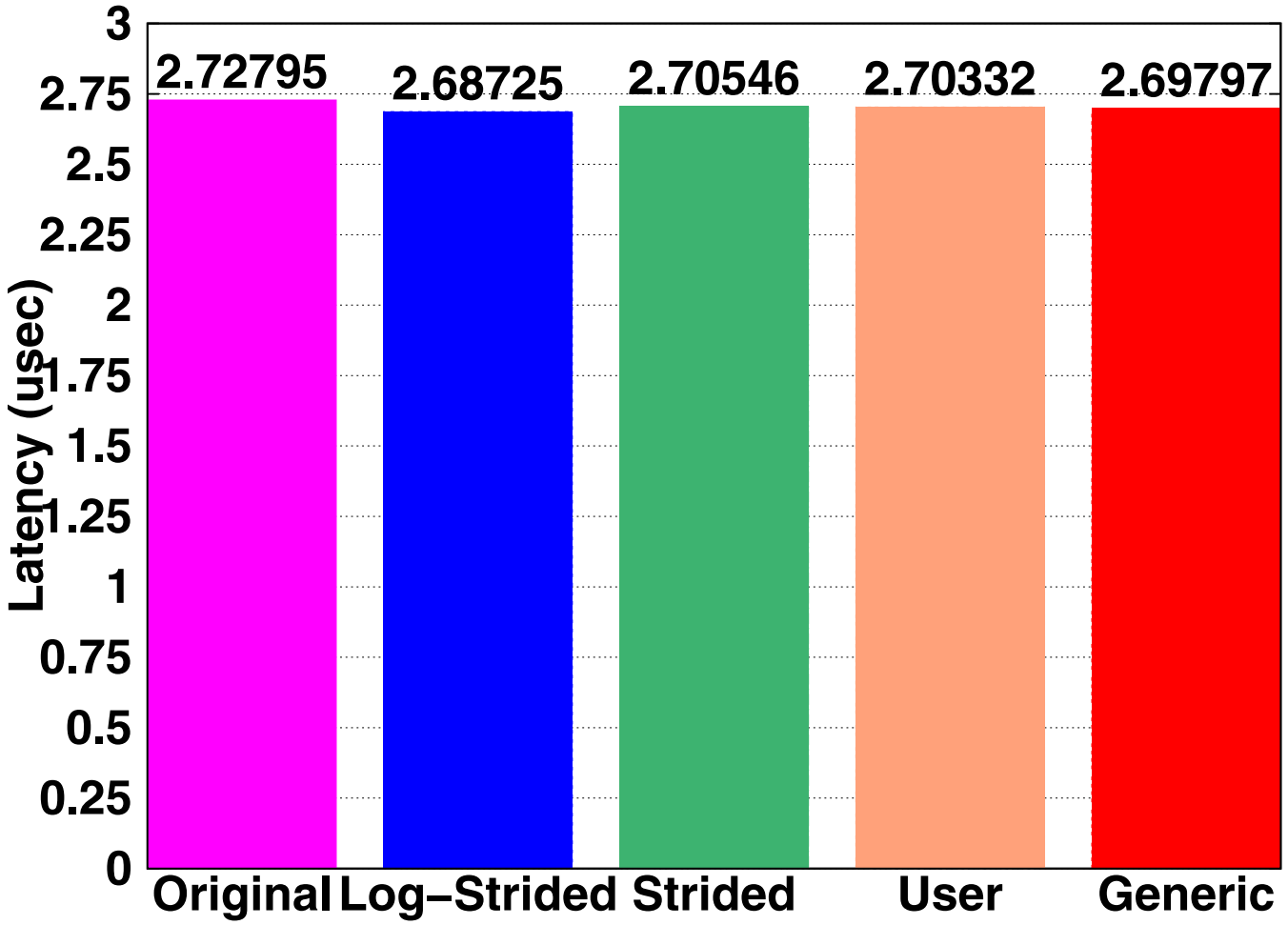
- PEs that are not a part of the ASET return NULL

Creating ASET using the Custom Function Interface

```
1  int my_custom_index_fn(int PE_index, int PE_start, int PE_size, void
       *const_params) {
2     return PE_index * PE_index + PE_start;
3  }
4  int main(int argc, char **argv) {
5     shmem_aset *aset;
6     ...
7     /* creates an active set containing PEs 2, 3, 6, 11 */
8     aset = shmem_create_custom_aset(2, &my_custom_index_fn, 4, NULL);
9     if (shmem_in_aset(aset)) {
10       shmem_barrier_aset(aset, pSync);
11    }
12 }
```

- Custom function maps a PE number to an index
- PEs that are not a part of ASET return NULL

Performance Comparison of ASET Creation Routines



Non-blocking Operations (Extension)

- Splits the invocation and completion of an operation
  - Completion/Synchronization decoupled from the invocation
  - Allows multiple outstanding operations
  - Out of order completion
  - Enables asynchronous progress (not guaranteed)
  - Enables computation-communication overlap

Non-blocking Remote Memory Access Operations

- shmem_put_nb /shmem_get_nb
- On return of this call
  - The Put/Get has been successfully posted (not necessarily on hardware)
  - Buffer not reusable (In OpenSHMEM 1.0, it is reusable)
  - No guarantee that the data is transferred to the remote PE
- Requires calling *shmem_wait_request/ shmem_wait_test* to learn the status of the operation

Example Usage of Non-blocking Get Operations

```
1   int main(int argc, char *argv[]){
2       shmem_request_handle_t request1;
3       ...
4       start_pes(0);
5       ...
6       shmem_int_get_nb(target, source, 1, me+1, &request1);
7       //some useful work
8       //call wait before the value is required by local PE
9       shmem_wait_request(request1);
10      x = target + 0.25 * y;
11      ...
12      return 0;
13  }
```

Non-blocking Atomic Operations

- Performs read and update operations on symmetric data objects
- Semantics: Similar to non-blocking Put/Get operations
- non-fetch operations are already non-blocking
  - This operation is explicit - status of the operation can be learnt using the request handle.

What about shmem_quiet/shmem_fence ?

- No change in semantics of shmem_quiet and shmem_fence
  - shmem_quiet does not guarantee the completion of prior invoked non-blocking RMA or Atomic Operations
  - shmem_fence does not order non-blocking RMA or Atomic Operations

Non-blocking Collective Operations

- barrier, barrier_all, broadcast, collect, and reductions
- The invocation and completion semantics are similar to the non-blocking RMA and Atomic operations
  - Outstanding operations
  - Out of order completion
  - Requires wait/test routine to be called for learning the status and completion

Ordering of Non-blocking Collectives

- Order of non-blocking collectives on an ASET should be the same

```
switch(pe) {
        case 0:
        shmem_barrier_nb(aset, pSync,&request1);
        shmem_broadcast32_nb(&y,&x,1,aset,pSync1, &request2);
        shmem_wait_request(request1); break;

        case 1:
        shmem_broadcast32_nb(&y,&x,1,aset,pSync1, &request2);
        shmem_barrier_nb(aset, pSync,&request1);
        shmem_wait_request(request1); break;
}
```

Graceful Library Shutdown (Extension)

- shmem_finalize
  - All PEs should call shmem_finalize (collective)
  - All non-blocking operations should be completed
  - All request handles should be released
  - Implementation may (should) release the resources
  - Using OpenSHMEM routines would require calling start_pes by all participating PEs
- shmem_abort
  - Any PE can call shmem_abort
  - Aborts the execution of OpenSHMEM program
  - Best effort to abort other PEs
  - OS Process can continue ..

Summary

- Evolutionary steps
  - Encapsulating Active sets in an opaque object and removing the constraints
  - Non-blocking operations
  - Graceful library shutdown

- Prototype implementation for the extensions (except non-blocking collectives)

Wishlist

- Add a memory context: pSync and pWork integrated into ASET
  - Performance - library optimizations
  - Productivity - library manages memory

- Add a communication context
  - Provides library isolation
  - Will this make ASETs heavy weight ?
  - Are we moving away from global symmetric memory ?

Wishlist

- Threading model
- I/O interfaces
- Error codes
- Profiling / Tools interface
- Fault tolerance

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.

Empowering the Mission