OpenSHMEM Extensions Towards Hybrid Programming and Heterogeneous Computing

David Knaak and Naveen Namashivayam
Cray Inc.
OpenSHMEM Workshop 2015
OpenSHMEM Extensions

Objectives for this talk

- Overview of 6 features being proposed for OpenSHMEM that are based on features currently in Cray SHMEM
- Our goal is to advance OpenSHMEM with new, desirable, and proven features
- Today’s audience:
  - For some, these are new concepts
  - For some, you are asking for these
  - For some, you are already involved in discussing
- See our paper for more details; see Redmine for full
- A starting point for discussion with OpenSHMEM community

August 2015
OpenSHMEM Extensions

What makes for a desirable, new OpenSHMEM feature?

- Improves ease of OpenSHMEM programming
- Improves performance of OpenSHMEM programs
- User friendly API
- Aids portability by hiding system differences in specific implementations
- Consistent with existing OpenSHMEM API
OpenSHMEM Extensions

Why do we need more than current API? Primarily due to trends in system architectures for exascale:

- Increased complexity
- Increasing number of cores in multi-core processors
- Memory hierarchies
  - Distributed and shared and high bandwidth
- Processor accelerators
- Increased network capabilities to offload communication work from compute processors
- Other new concepts that help programmability and performance
OpenSHMEM Extensions

The 6 Proposed Features (Redmine Ticket #)

- Alltoall Collectives
- Flexible PE Subsets, a.k.a. Teams
- Thread-Safety
- Local Shared Memory Pointers
- Put With Signal
- Non-Blocking Put and Get
**OpenSHMEM Extensions**

**Alltoall Collectives (Redmine #182, #183)**

- All-to-all pattern of communication is common in programs
- Each PE exchanging data with every other PE in the defined set
- Naive implementation usually far from optimal
- Sophisticated implementation complex and can be system-specific - best to hide it in the library
OpenSHMEM Extensions

Alltoall Collectives API – 3 routines

- `shmem_alltoall` - fixed size data
- `shmem_alltoallv` - variable size data and variable source/dest offsets
- `shmem_team_alltoall` - using teams syntax (using proposed Teams, see below)
OpenSHMEM Extensions

Alltoall Collectives Performance

(a) Both Source and Target are Same Variable

(b) Source and Target are Different Variables
OpenSHMEM Extensions

Flexible PE Subsets, a.k.a. Teams (Redmine #185)

- An alternative Teams proposal (Redmine #179)
- Current active set specification not flexible enough
- Proposed feature allows a set of PEs to be divided in arbitrary ways
- Similar to MPI and UPC teams: color and key
- Lots of issues to be hashed out
OpenSHMEM Extensions

Flexible PE Subsets API – 7 routines

- `shmem_team_split` - split existing team as needed using color and key
- `shmem_team_create_strided` - w/ stride argument, NOT power-of-2
- `shmem_team_translate_pe` - rank in one team to corresponding rank in another team
- `shmem_team_npes` - how many in this team
- `shmem_team_mype` - my rank in this team
- `shmem_team_barrier` - barrier for just this team
- `shmem_team_free` - release resources
Thread-Safety (Redmine #186)

- Hybrid programming such as SHMEM and OpenMP
- Execution by multiple threads per PE can be more efficient than by PEs alone
- Multiple threads per PE can directly access PE's symmetric memory
- OpenSHMEM needs to be thread safe
- Proposed support is basic - Puts, Gets, AMOs
- Can expand API in future as need arises
- Can be used with "Communications Contexts" (Redmine #177)
OpenSHMEM Extensions

Thread-Safety API – 6 routines

- \texttt{shmem\_init\_thread} – in place of \texttt{shmem\_init}
- \texttt{shmem\_query\_thread} – query current level
- \texttt{shmem\_thread\_register} – required before using a thread
- \texttt{shmem\_thread\_unregister} – when done using a thread
- \texttt{shmem\_thread\_quiet} – completion of outstanding communication
- \texttt{shmem\_thread\_fence} – ordering of communication
OpenSHMEM Extensions

Thread-Safety Performance

(a) Intra-Node Transfer

(b) Inter-Node Transfer
OpenSHMEM Extensions

Thread-Safety Performance

![Bar chart showing thread-safe Cray SHMEM + OpenMP vs only Cray SHMEM performance](chart)

- Time in Seconds
- Number of Processes/Threads

Legend:
- Red: thread-safe Cray SHMEM + OpenMP
- Green: only Cray SHMEM

- 16 processes/threads: Red (110), Green (105)
- 32 processes/threads: Red (90), Green (80)
- 64 processes/threads: Red (70), Green (65)
- 128 processes/threads: Red (60), Green (55)
- 256 processes/threads: Red (50), Green (45)
- 512 processes/threads: Red (40), Green (40)
OpenSHMEM Extensions

Local Shared-Memory Pointers (Redmine #70)

- `shmeme_local_ptr` is different from `shmeme_ptr` which supports off-node direct references
- `shmeme_local_ptr` is for on-node references
- Local in sense of on same node; define node:
  - group of processors, memory, and network components that acts as a network end point
  - the memory on a node is addressable by all processors on the node without having to go through the network
  - direct addressability can have lower latency and higher bandwidth
- Use when direct on-node references can be more efficient than through API calls
OpenSHMEM Extensions

Local Shared-Memory Pointers API – 3 routines

- `shmem_local_ptr` – returns address or NULL
- `shmem_local_npes` – how many PEs are local
- `shmem_local_pes` – which PEs are local
OpenSHMEM Extensions

Non-Blocking Put and Get (Redmine #113)

- Desirable to overlap communication between PEs and computation by PEs
- Current blocking Put and Get don't allow this
- Overlap by issuing non-blocking call, than later wait for completion
OpenSHMEM Extensions

Non-Blocking Put and Get API

- `shmem_<type>_put_nb`
- `shmem_put<size>_nb`
- `shmem_<type>_get_nb`
- `shmem_get<size>_nb`
OpenSHMEM Extensions

Put With Signal (Redmine #77)
- Combines sending data with sending a signal that data has arrived
- Easier to program
- Potential for better performance
- Blocking and non-blocking implicit versions
OpenSHMEM Extensions

Put With Signal API – many routines

- `shmem_<type>_put_signal`
- `shmem_put<size>_signal`
- `shmem_<type>_put_signal_nb`
- `shmem_put<size>_signal_nb`
OpenSHMEM Extensions

Put With Signal Performance

(a) Small Data Sizes $\leq 8192$ bytes

(b) Large Data Sizes $> 8192$ bytes

using USER DEFINED Put Signal routine
using Cray SHMEMX_PUT_SIGNAL() call
OpenSHMEM Extensions

Conclusion

- These 6 features have been implemented in Cray SHMEM and are already being used
- We believe these are valuable for many OpenSHMEM programs
- We request all 6 features be given careful consideration for OpenSHMEM API
- We will work within the OpenSHMEM community for consensus
OpenSHMEM Extensions

Acknowledgements

OpenSHMEM Extensions

Question?
OpenSHMEM Extensions