Introducing Cray OpenSHMEMX

A Modular Multi-Communication Layer OpenSHMEM Implementation

Naveen Namashivayam, Bob Cernohous, Dan Pou, and Mark Pagel

Overview

- **Cray supports two different proprietary OpenSHMEM implementations**
  - Cray SHMEM – current production-ready library
  - Cray OpenSHMEMX – early evaluation library
- **Cray OpenSHMEMX will supersede Cray SHMEM on future generation Cray systems**
- **In this presentation:**
  - Discuss on the need for a new OpenSHMEM implementation
  - Introduce Cray OpenSHMEMX
  - Overview on the design and performance impact comparing against Cray SHMEM

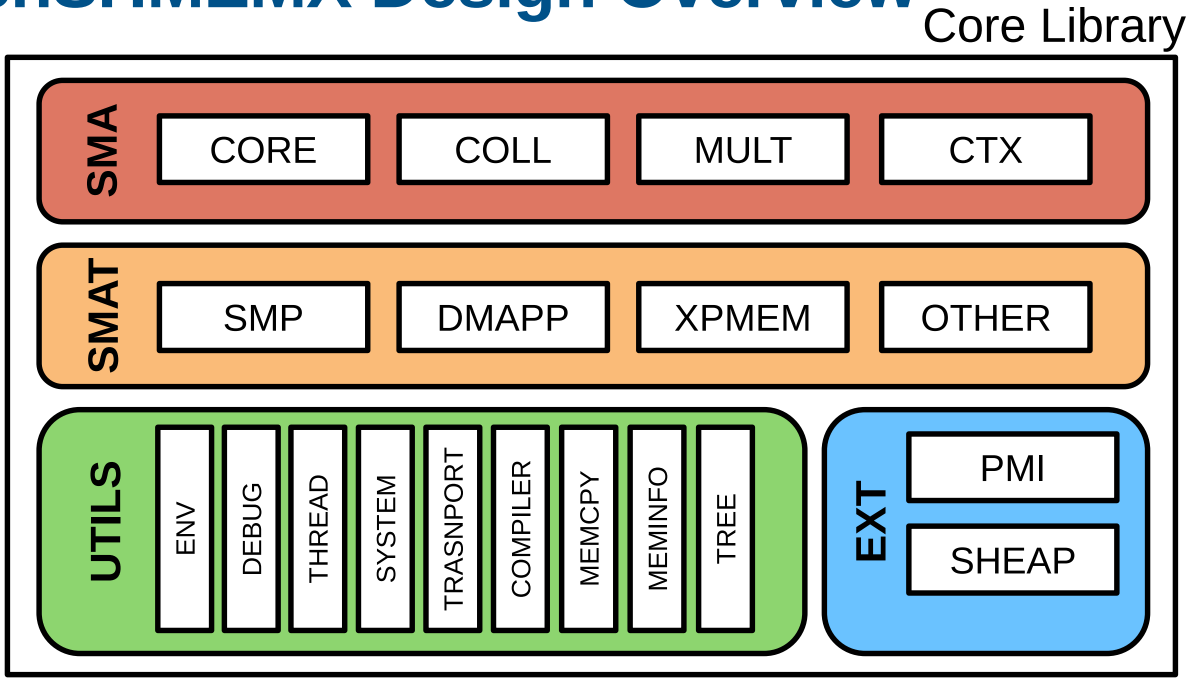
Cray SHMEM Background

- **Current generation production-ready proprietary implementation**
- **Highly optimized and tuned for Cray XC, Cray XK, and Urika-GX systems**
- **Implemented over DMAPP**
  - Underlying communication layer
  - Tightly coupled design
  - Can be considered as thin-wrapper over DMAPP
- **Supports Cray-developed Aries and Gemini Interconnects**
- **Supports x86_64 processors – Intel Xeon and Intel Xeon Phi**
- **Uses Cray PMI for process management**

Need for a New OpenSHMEM Implementation

- **Future system architectures from Cray for Exascale scalable systems and beyond**
  - Support multiple processor architectures and accelerators
    - x86_64, AArch64, GPU
  - Use different interconnects
    - Both Cray proprietary and other vendor interconnects
  - Diversify the application usage models
    - For example: Support for Multiple Application Per Node (MAPN) without networking
    - Cray SHMEM has a restriction on the maximum number of PPN – ~120 PEs
    - This restriction is extended even for single-node jobs – because of DMAPP registration at initialization

Cray OpenSHMEMX Design Overview



- **We use SMP-DMAPP transport layer for this work**
  - On-node data transfer handled by XPMEM
  - Off-node data transfer handled by DMAPP

Feature Comparison

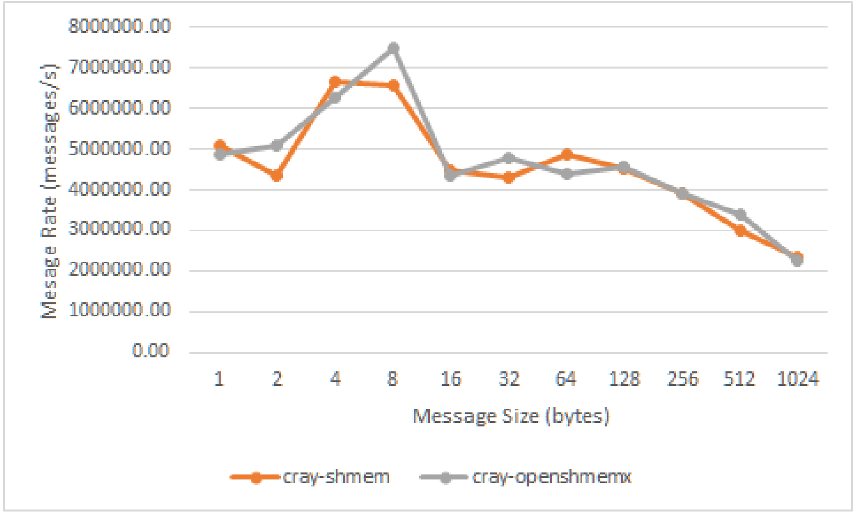
Library Features	Cray SHMEM 7.7.0	Cray OpenSHMEMX 8.0.1
OpenSHMEM (OSH) Compliance	OpenSHMEM 1.3	OpenSHMEM 1.4
- OSH 1.4 Contexts and Sync	✗	✓
- OSH 1.4 Extended Typed RMA and AMO	✓	✓
- OSH 1.4 *shmem_test ()* and *shmem_calloc ()*	✓	✓
- OSH 1.4 Bitwise AMOs	✓	✓
Cray specific Teams and Team-based Collectives	✓	✓
Cray specific Thread-hot multithreading features	✓	✓
Cray specific Symmetric Memory Partitions	✓	Future
Cray specific Non-blocking AMOs	✓	Future
Cray specific Put-with-Signal operations	✓	✓
Cray specific Alltoallv and Alltoallv_packed	✓	✓
Cray specific Local-node Queries	✓	✓
Cray specific Fortran 2008 Bind(C) Wrapper	✗	Future

Performance Regression Analysis

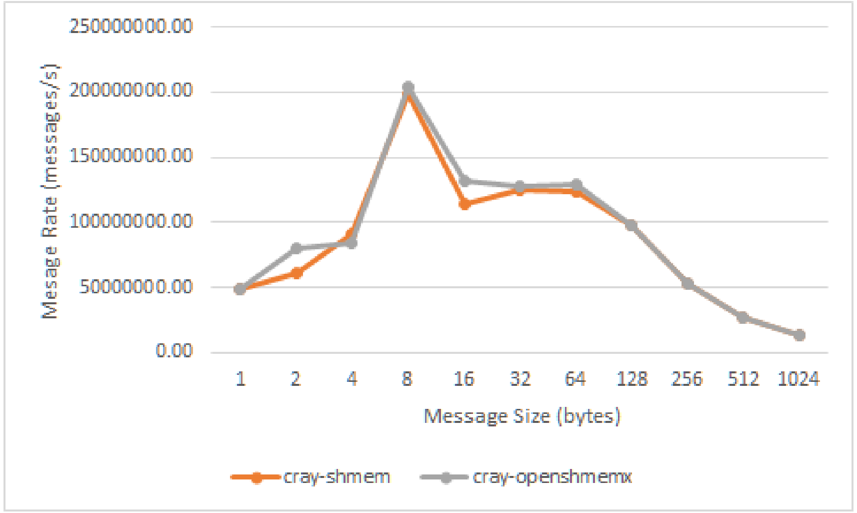
Component Name	Version Details
GCC Compiler	7.3.0
Cray SHMEM	7.7.0
Cray OpenSHMEMX	8.0.1
DMAPP	7.1.1
Cray Linux Environment (CLE)	CLE 6.0 update 06

- **Compared the performance of Cray SHMEM against Cray OpenSHMEMX using OSU Microbenchmarks**
- **Tested on two different processors – Intel Xeon (Broadwell) and Intel Xeon Phi (KNL)**
- **We report the performance analysis only on Non-blocking Put**
- **Similar analysis were performed on other SHMEM operations**

Inter-node NBI Put on Intel BDW – 2 Nodes



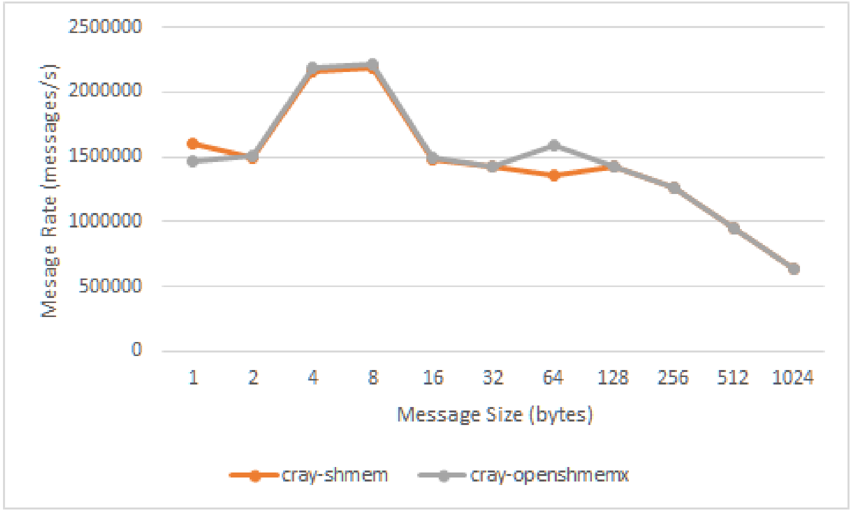
1 PPN



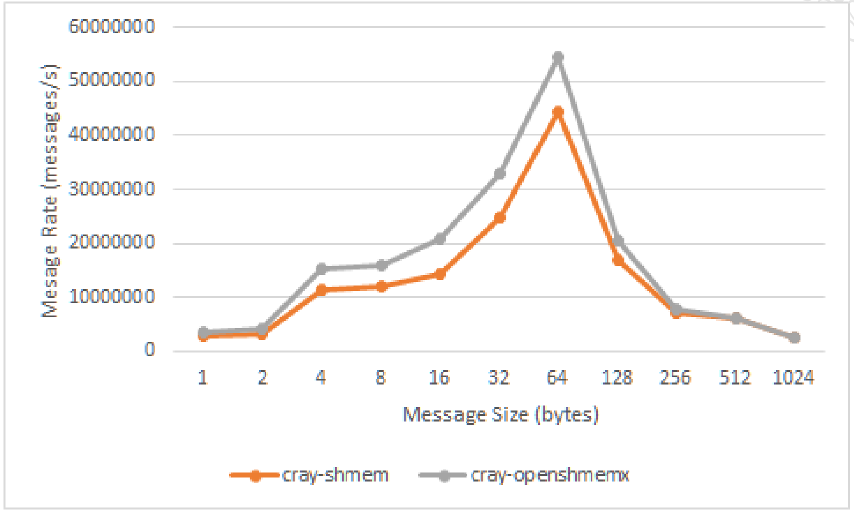
32 PPN

- **No Change in performance – within 3% variation**
- **Small Message Sizes less than 1024 bytes**

Inter-node NBI Put on Intel KNL – 2 Nodes



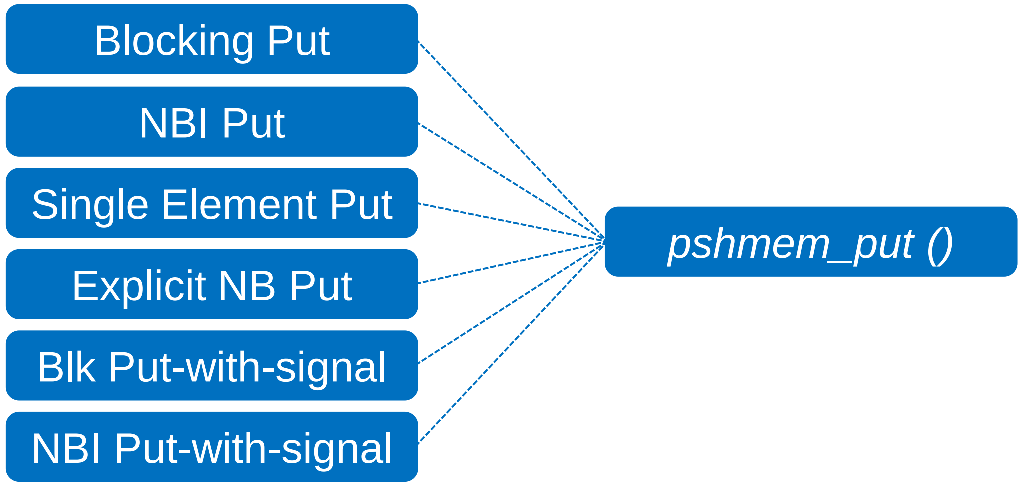
1 PPN



32 PPN

- **No Change in performance for 1PPN tests – within 3% variation**
- **Performance in Cray OpenSHMEMX improved by 16% for 32 PPN tests**
- **For small message sizes less than 1024 bytes**

NBI Put Code Path



- **Cray SHMEM:**
  - Different variants of the put operation targets the same internal put function
  - Multiple decision branches and unnecessary execution
- **Inference:1 Reach the main data transfer operation with few instructions**
- **Inference:2 Even try to support separate performance and debug builds**

Overview of Selected Features and Enhancements

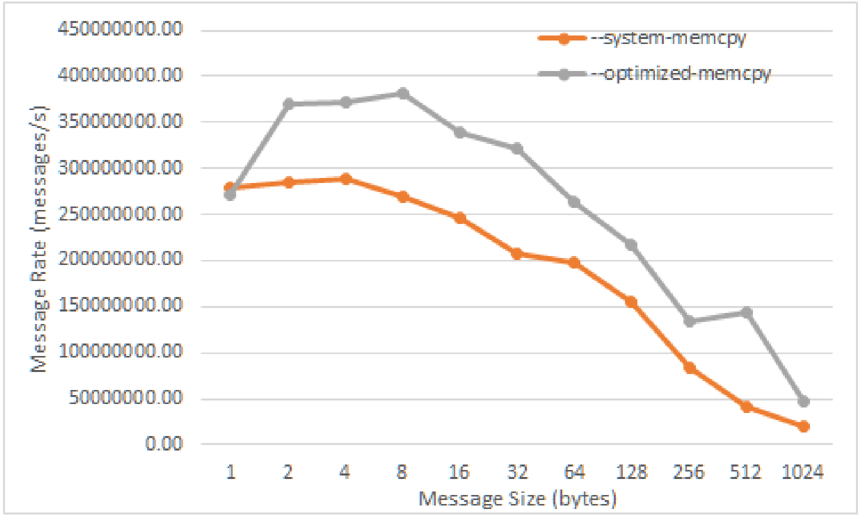
- **Detailed discussion on implementation of few important OpenSHMEM and Cray specific features in Cray OpenSHMEMX**
  - Supported Processor Architectures
  - Interoperability between OpenSHMEM Contexts and Cray Thread-hot
  - Support for Put-with-signal Operations

Supported Processor Architectures

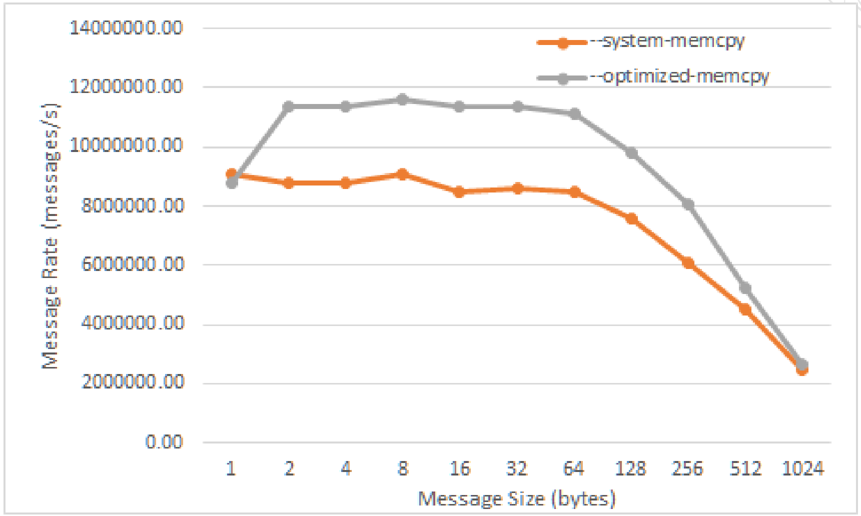
Processor Type	Cray SHMEM	Cray OpenSHMEMX
Intel Xeon (x86_64)	✓	✓
Intel Xeon Phi (x86_64)	✓	✓
Cavium Thunder X2 (AArch64)	✘	✓

- **Added support for Cavium Thunder X2**
- **System specific components are restricted to the UTILS layer**
- **On-node processor specific optimizations like optimized memcpy, on-node strided data transfers are added as part of the UTILS layer**

Intra-node NBI Put on Intel KNL – 1 Node



2 PPN



68 PPN

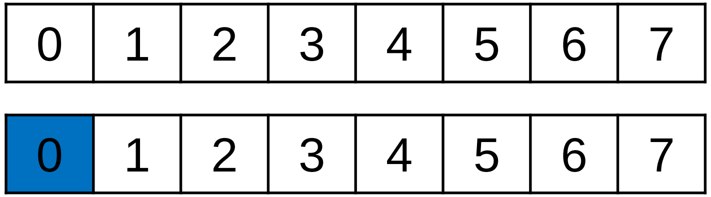
- **On-average 20% improved performance on using optimized memcpy compared against system memcpy for small data sizes less than 1024 bytes**
- **Enabled optimized memcpy as default in Cray OpenSHMEMX**

Interoperability between OpenSHMEM Contexts (CTX) and Cray Thread-hot (THS)

- **OpenSHMEM CTX – feature added in OpenSHMEM specification 1.4**
  - Multiple independent streams of communication within the same application
  - Provide opportunities for thread isolation by eliminating synchronization overhead on multithreaded application
- **Cray Thread-hot – Cray specific feature available in Cray SHMEM for a long time**
  - Similar functionality as CTX for multithreading but implicit resource management
  - Works with pthreads-based threading models
  - *shmemx_thread_register ()* and *shmemx_thread_unregister ()*
- **Current design in Cray OpenSHMEM**
  - Allows Interoperability between CTX and THS
  - Divide network resources (FMA) uniformly across all the PEs sharing the same node
  - Pre-initialize all the network resource available per PE during *shmem_init_thread ()*

Multithreading without CTX or THS

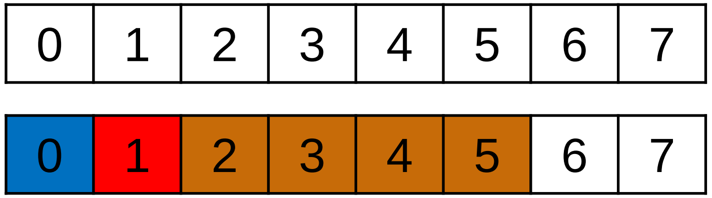
- Consider if each PE in the job has 8 resource and application doesn't make use of either CTX or THS feature
- RMA and AMO events initiated by all threads is still thread-safe
- All communication events use the same default network resource
- Events are serialized through locks



```
shmem_thread_init (SHMEM_THREAD_MULTIPLE, &avail);
# pragma omp parallel
{
    shmem_int_put (dst, src, nelems, pe);
}
```

Multithreading with Explicit CTX Creation

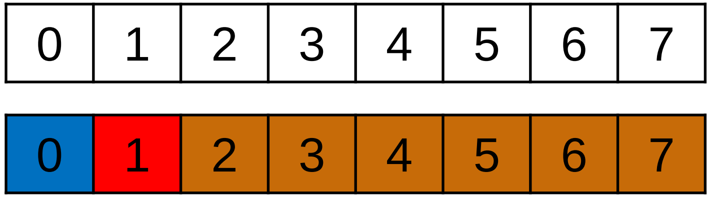
- **Expanding example from previous slide**
- **CTX1 is a shareable context**
- **CTX is a context with SHMEM_CTX_PRIVATE private**
  - Context creation with SHMEM_CTX_PRIVATE fails if there are no private resource



```
shmem_thread_init (SHMEM_THREAD_MULTIPLE, &avail);
shmem_ctx_t ctx1;
shmem_ctx_create (0, &ctx1);
# pragma omp parallel num_threads(4)
{
        shmem_ctx_t ctx;
        shmem_ctx_create (SHMEM_CTX_PRIVATE, &ctx);
}
```

Multithreading with both Explicit CTX Creation and Cray-specific THS registration

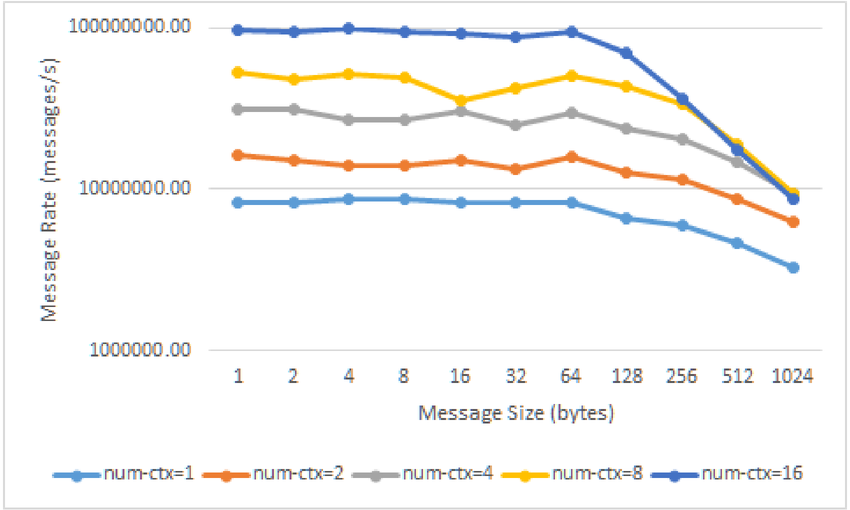
- **Modifying the example from previous slide**
- **CTX1 is a shareable context**
- **Instead of creating CTX with SHMEM_CTX_PRIVATE property**
  - Perform thread registrations on all the threads
  - In this example, 10 threads share 7 resources including CTX1 resource



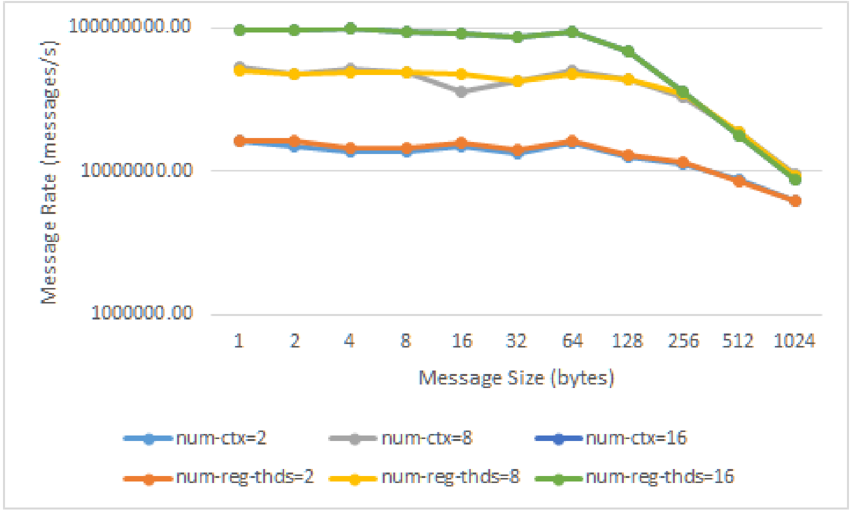
```
shmem_thread_init (SHMEM_THREAD_MULTIPLE, &avail);
shmem_ctx_t ctx1;
shmem_ctx_create (0, &ctx1);
# pragma omp parallel num_threads(10)
{
    shmemx_thread_register ();
}
```

Performance of CTX compared against THS

- Modified OSU Microbenchmarks and converted it to use CTX and THS
- Intel BDW – 2 Nodes with 1 PPN



OpenSHMEM CTX Message Rate



OpenSHMEM THS Message Rate

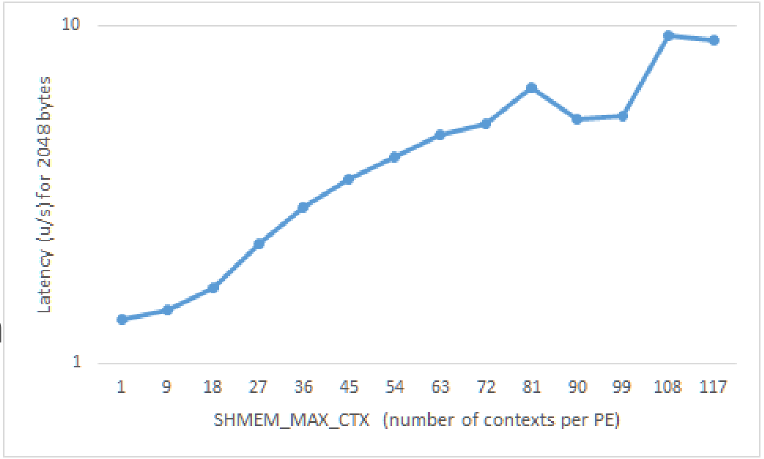
Inference from CTX and THS Resource Sharing Mechanisms

- **Though resources can be shared between CTX and THS – it becomes complicated**
  - When the number of CTX created or threads registered >> actual available resource
  - Initializing the right amount of resource at *shmem_init ()* impacts overall performance
  - Impacts the memory ordering semantics
    - shmem_quiet ()
    - shmem_ctx_quiet ()
    - shmemx_thread_quiet ()
    - As per OpenSHMEM shmem_quiet () ensures delivery only the default context
    - Modified the semantics to quiet both the default context and any registered thread

```
shmem_thread_init (SHMEM_THREAD_MULTIPLE, &avail);
shmem_quiet ();
#pragma omp parallel
{
    shmemx_thread_register ();
}
shmem_quiet ();
#pragma omp parallel
{
    shmemx_thread_unregister ();
}
shmem_quiet ();
shmem_finalize ();
```

SHMEM_MAX_CTX Environment Variable

- Knowing the number of Contexts at initialization improves performance
- SHMEM_MAX_CTX is different from SHMEM_MAX_THREADS environment variable
- For example – it helps in setting the correct DLA credits during initialization based on the SHMEM_MAX_CTX value
- Needs more study on similar performance issues
- Expose users with more query options to understand the resource utilization information
- Or Avoid pre-initialization of all resources



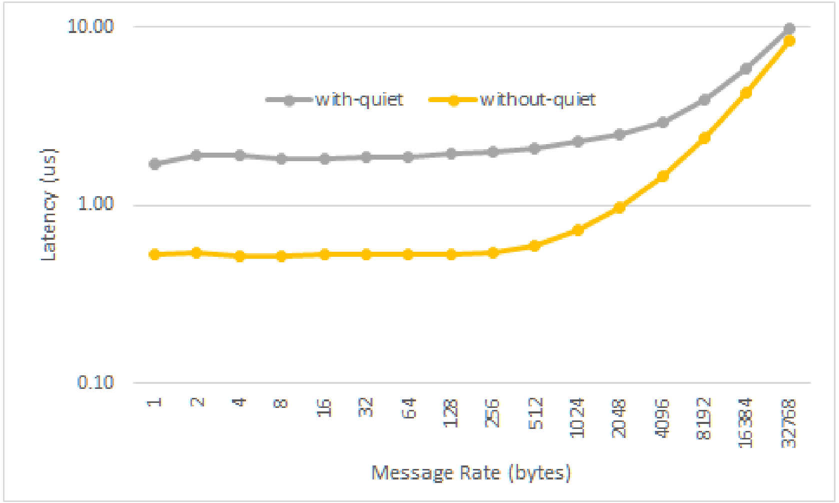
Latency of SHMEM_CTX_DEFAULT with different values for SHMEM_MAX_CTX – Intel BDW 2 Nodes 1 PPN

Put-with-Signal – Intel BDW 2 Nodes 1 PPN

```
shmem_put (*dst, *src, nelems, pe);
shmem_fence ();
shmem_p (*signal, value, pe);
```

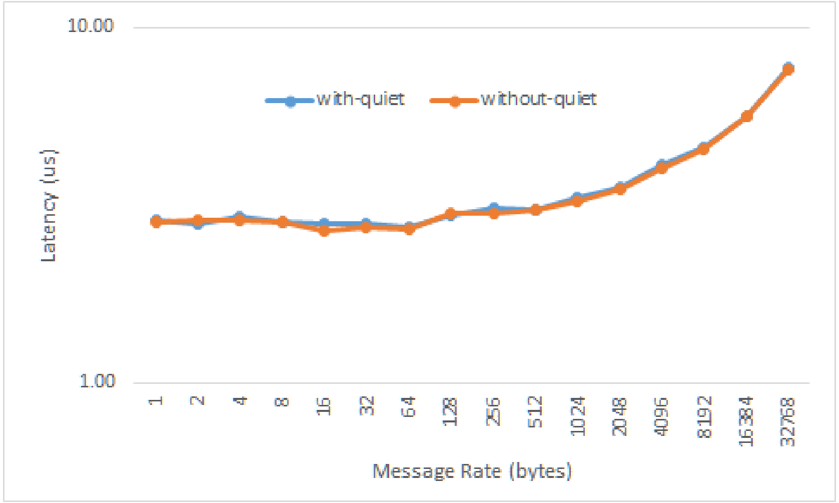
```
shmemx_put_signal (*dst, *src,
nelems, *signal, value, pe);
```

With Fence Implementation



Comparing put-with-signal with and without fence

Without Fence Implementation



Comparing put-with-signal when src and dest are in different segments

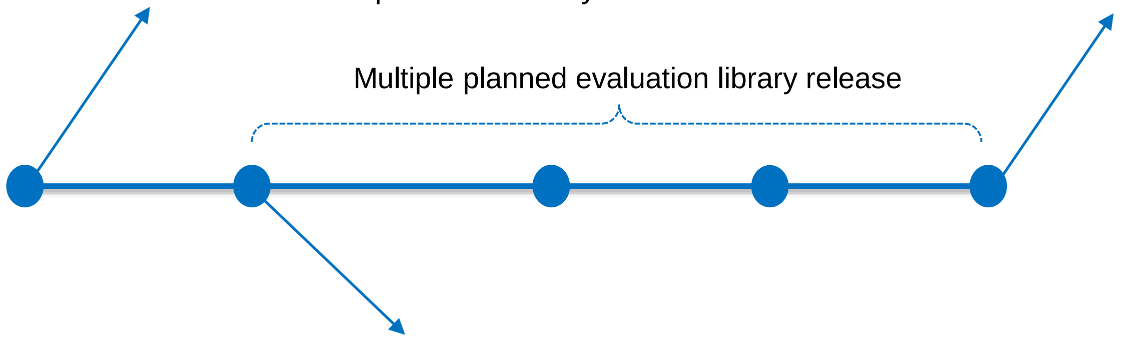
Conclusion

- **Introduced Cray OpenSHMEMX**
  - A new proprietary software library product from Cray Inc.
  - Planned to supersede Cray SHMEM on future Cray systems
  - Modular implementation to support both OpenSHMEM and Cray specific features
  - Designed to support different processor and interconnect architectures
- **Through early performance analysis in this work – we show:**
  - Cray OpenSHMEMX performs on-par or even better than Cray SHMEM on certain RMA and AMO features in current Cray XC systems
  - SMP-DMAPP is an evaluation library
  - Existing user applications are expected to migrate and start using the new library
- **Future Work**
  - Add new features
  - Look for exposing implementation specific optimization features efficiently to the users
  - Large scale performance study using collectives

- Release Information: https://pe-cray.github.io/cray-openshmemx/
- Related Whitepapers: https://pe-cray.github.io/whitepapers

Thank You

Cray OpenSHMEMX Availability



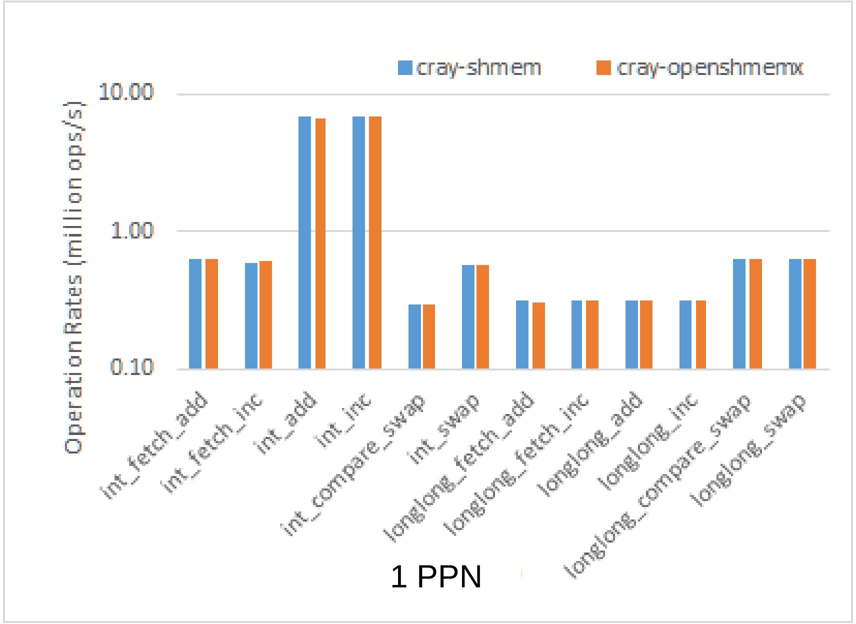
version: 8.0.0
- Support for Aarch64 and x86_64
- First official evaluation library release on x86_64
- First official production-ready release on Aarch64

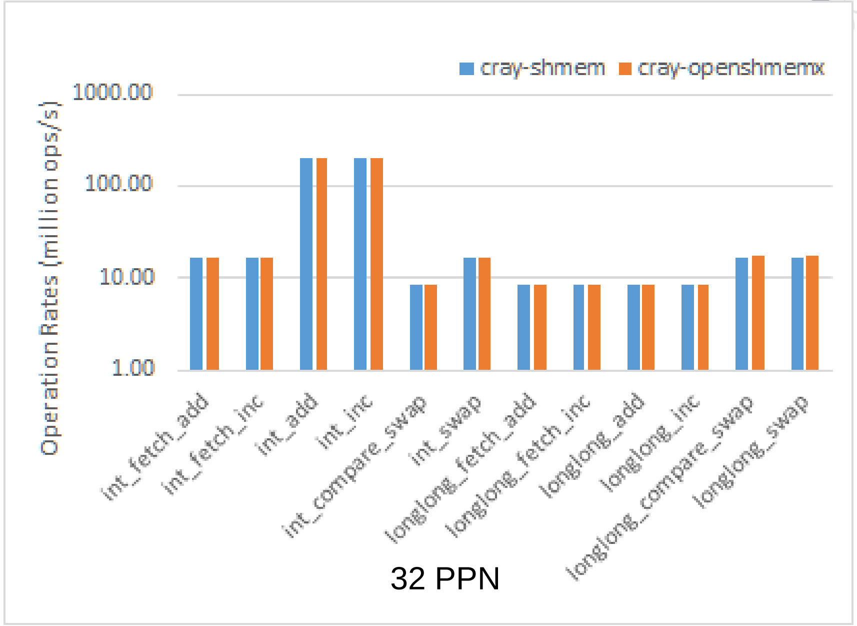
version: 9.0.0
- Anticipated first production-ready release
- Supersede Cray SHMEM as the production-ready library on all Cray systems – both x86_64 and Aarch64

version: 8.0.1
- Most recent evaluation library release on both x86_64 and Aarch64
- Interoperate CTX and Cray Thread-hot

- **Release Notes:** https://pe-cray.github.io/cray-openshmemx/
- **Related Whitepapers:** https://pe-cray.github.io/whitepapers

Inter-node AMO on Intel BDW – 2 Nodes





- **No Change in performance for both fetching and non-fetching – within 3% variance**