Cray SHMEM Update

Monika ten Bruggencate
Cray Inc.
March 2014

Outline

- OpenSHMEM Compliance
- Incremental Extensions to Cray SHMEM
- Thread Safety Extensions in more Detail
- Future Work

OpenSHMEM Compliance

- Cray SHMEM has been compliant with OpenSHMEM 1.0 as of Cray Message Passing Toolkit (MPT) release 5.6.2, released Feb 2013.

Enhancements to Cray SHMEM

- We continue to enhance Cray SHMEM to meet our customers needs
- Exploring interfaces beyond the OpenShmem 1.0 base line in conjunction with our customers
- Each API extension implemented to be proposed to OpenSHMEM community for review and discussion
- Ultimately we would like inclusion of extensions in the spec, with modifications as necessary
- Avoid API bloat

Extension – Initialization and Finalization

```
void shmem_init(void)
void shmem_finalize(void)
```

- Naming of initialization and finalization routines more consistent with other APIs
- `shmem_init()` has same semantics as `start_pes(npes)`
- Use of `shmem_finalize()` is good practice and can be helpful for tools
- Supported on Cray XT, XE and XC

Extension – Non-blocking Puts and Gets

```
void shmem_put_nb(void *trg, const void *src, size_t
                  nelems, int pe, void **transfer_hnld)
void shmem_get_nb(void *trg, const void *src, size_t
                  nelems, int pe, void **transfer_hndl)
```

- Allow overlap of communication and computation
- Adopted Quadrics API
- No demand for explicit variants, no transfer hndl returned
- `shmem_quiet()` checks for completion
- All size and type variants are supported
- Supported on Cray XE, XC (`shmem_put_nb()` also on XT)

Extension – On-node shmem_ptr()

```
void *shmem_local_ptr(void *trg, size_t len, int pe)
int   shmem_local_npes(void)
void  shmem_local_pes(int *pes, int npes)
```

- `shmem_local_ptr()` has same effect as `shmem_ptr()` but for on-node use only
- Returns non-NULL for symmetric data objects on PEs for which data object can be accessed via memory loads /stores
- Allows direct access to symmetric memory of another PE on the node

Extension – Signalling Put

```
void shmem_put_signal[_nb](void *target,
     const void *source, size_t len,
     uint64_t *sig_target, uint64_t sig_val, int pe)
```

- Semantically equivalent to `shmem_put[_nb]()` PLUS after the Put transfer is complete, the signal value will be delivered to the signal location on the target PE
- The target buffer and signal location must reside in symmetric memory
- All data size and type variants are supported

Extension – SHMEM Teams

- Flexible way to manage PE subsets
- Current method to specify subgroups is restrictive
- Create, query, use, reuse, destroy arbitrary sets of PEs
- Follows the API adopted by UPC, which again follows the MPI communicator concept => consistency across PGAS
- Supports the existing triplet interface

Flexible Subsets – Team Interfaces

- `SHMEM_TEAM_WORLD, SHMEM_TEAM_NODE` created by `shmem_init()`
- Create /destroy teams (collective ops, subset determined by `color`, rank by key)

```
void shmem_team_split(shmem_team_t team, int color,
     int key, shmem_team_t *newteam)
void shmem_team_free(shmem_team_t *team)
void shmem_team_create_strided(int PE_start,
     int PE_stride, int PE_size,
     shmem_team_t *newteam)
```

Flexible Subsets – Team Interfaces

- Team info

```
int shmem_team_npes(shmem_team_t team)
int shmem_team_mype(shmem_team_t team)
int shmem_team_translate_pe(shmem_team_t team1,
    int team1_pe, shmem_team_t team2)
```

- Use in collective operations

```
void shmem_team_barrier(shmem_team_t team,
                        long *pSync)
```

Extension - Alltoall Collective Communication

```
void shmem_team_alltoall(void *trg, const void *src,
     size_t len, shmem_team_t team, long *pSync)
```

- Fixed size alltoall: each PE exchanges constant amount of data with all other PEs in team
- Allows reliable use and optimized implementation of alltoall communications for applications
- Mimics `MPI_Alltoall()`
- Also supported for existing subset triplet
- General rules for collectives apply

Alltoall Collective Communication

```
void shmem_team_alltoallv(void *trg, size_t *t_offsets,
     size_t *t_sizes, const void *src, size_t *s_offsets,
     size_t *s_sizes, shmem_team_t team, long *pSync)
void shmem_team_alltoallv_packed(void *trg, size_t t_len,
     size_t *t_size, const void *src, size_t *s_offsets,
     size_t *s_sizes, shmem_team_t team, long *pSync
```

- Variable size alltoall: each PE exchanges variable amount of data with all other PEs in team
- Also supported for existing subset triplet

Extension -Thread Safety

- **Motivation**
  - Customer requests for thread-safe SHMEM, especially thread-safe Puts, Gets, AMOs
  - Desire to achieve high per-process issue rates for small pt2pt operations in a multi-threaded environment

⇒**Two components to thread safety work**

  - Performance
  - API extensions necessary

- **Common Use Case**
  - 1 PE per node or NUMA node
  - No more than one thread/core or hyper-thread

Thread Safety – API

- To satisfy customer requirement, we came up with a set of short-term API extensions
- Keep API changes minimal, yet give user control to achieve good performance
- Some limitations are imposed
- It may be that only a subset of the threads of a process makes SHMEM calls
- Made some adjustments to initial proposal incorporating feedback from community
- Long-term solution needs to be discussed with community (endpoint, context)

Thread Safety – API Overview

- **Initialization**

```
int  shmem_init_thread(int required)
int  shmem_query_thread(void)
void shmem_thread_[un]register(void)
```

- **Per-thread issue of one-sided operations**
  - No change to existing functionality
  - Addition of per thread sync functions

```
void shmem_thread_fence(void)
void shmem_thread_quiet(void)
```

- **Per process collectives and symmetric heap functions**
  - No change to interface

Thread Safety - Initialization

```
int shmem_init_thread(int required)
```

- **Used instead of `start_pes()` when TS should be enabled**
- **Specify level of TS support desired**

  `SHMEM_THREAD_SINGLE` – no threading/one thread per process

  `SHMEM_THREAD_MULTIPLE` – processes may have multiple threads and any thread may issue SHMEM calls at any time (currently some restrictions apply)
- **Returns level of TS support enabled**
- **Specify max # of threads/process via env var (hints)**
- **Called by only one thread per process**
- **MPI interface is model**

Thread Safety – Query and Finalization

```
int shmem_query_thread(void)
```

- Query level of thread safety enabled
- Returns level of thread safety support enabled

- Finalization routine `shmem_finalize()` also restricted to being called by only one thread per process

Thread Safety – Thread Registration

```
void shmem_thread_register(void)
void shmem_thread_unregister(void)
```

- Allows a thread to register/unregister with SHMEM
- Any thread that wants to make SHMEM calls must register itself (optional for the thread which initialized thread safety) and later unregister
- Ease of use vs. performance
- Explicit registration minimizes overhead for pt2pt operations
- Allows threads to dynamically register/unregister
- Allows dedicated and optimized use of resources
- Very low overhead

Thread Safety – Pt2Pt Operations

- No change to semantics of Put, Get, AMO operations
- Multiple threads per process can simultaneously call pt2pt operations

Thread Safety – Synchronization

- No change to semantics of `shmem_quiet()` and `shmem_fence()`
- Concurrent calls to `shmem_quiet()` and pt2pt operations leads to undefined completion behavior
- Thread specific, lighter-weight synchronization is desired

```
void shmem_thread_fence(void)
void shmem_thread_quiet(void)
```

- Allow thread to order or wait for completion of its previously issued pt2pt ops

Thread Safety – Collectives

- No change in per-process collectives interface
- Only one collective operation per process can be active at a time
- No change to requirement that collectives have to be called in the same order across processes
- Symmetric heap management functions must be treated as collective operations and affect the entire process
- Currently no thread barrier function planned

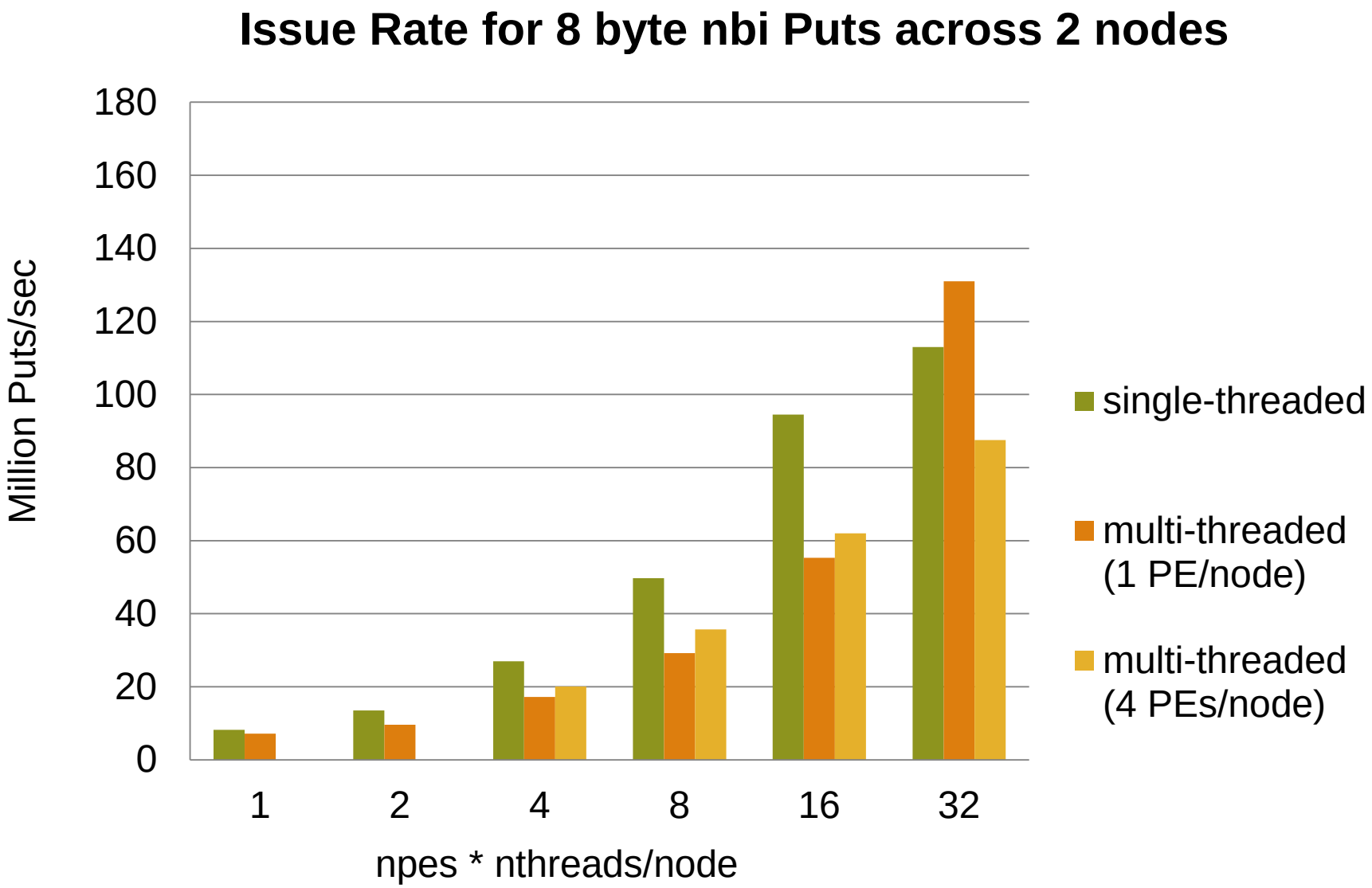
Thread Safety – Cray SHMEM Implementation

- Cray SHMEM thin layer over DMAPP
- Majority of thread safety work done in DMAPP
- More effective use of NIC resources: dedicated use w/o contention or shared use with limited contention
- Fine-grain locking
- Performance analysis in progress
- Further optimizations to be implemented in DMAPP for well-behaved apps: around NIC resources, possibly s/w resources
- API extensions in Cray SHMEM to be released

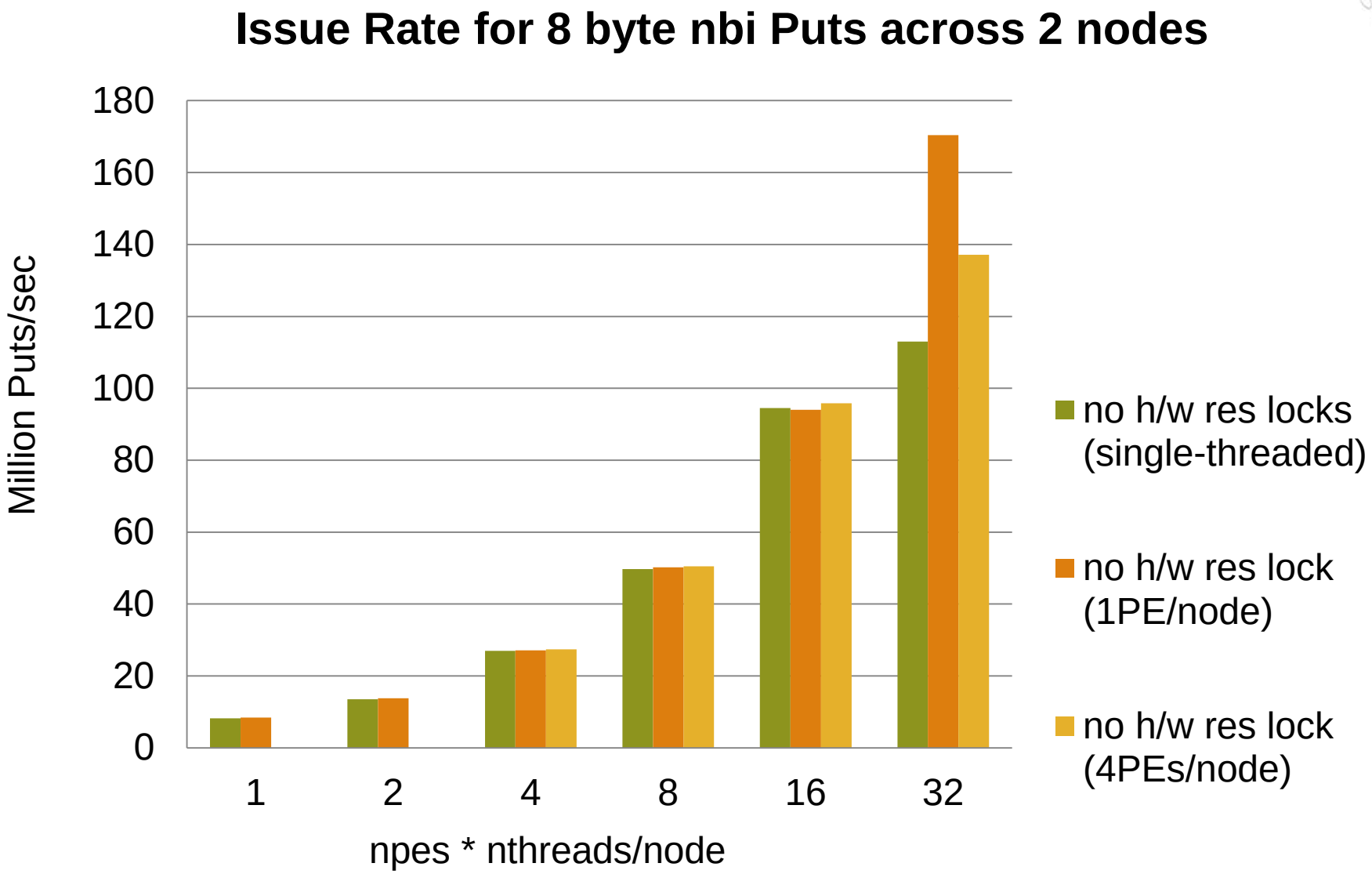
DMAPP Performance Data

- **Tested Put rates for single and multi-threaded tests**
  - Tests execute Puts to random PEs and target addresses
  - 8 byte to 1024 byte Puts
  - Multi-threaded tests mimic common use case
  - Non-blocking implicit transfers, 1024 ops outstanding
  - Running across two nodes
- **Cray XC 30 testbed**
  - Intel Sandybridge processors (32 hardware threads/node, 2.7 or 3.3 GHz)
  - CLE 5.1up01 DMAPP library

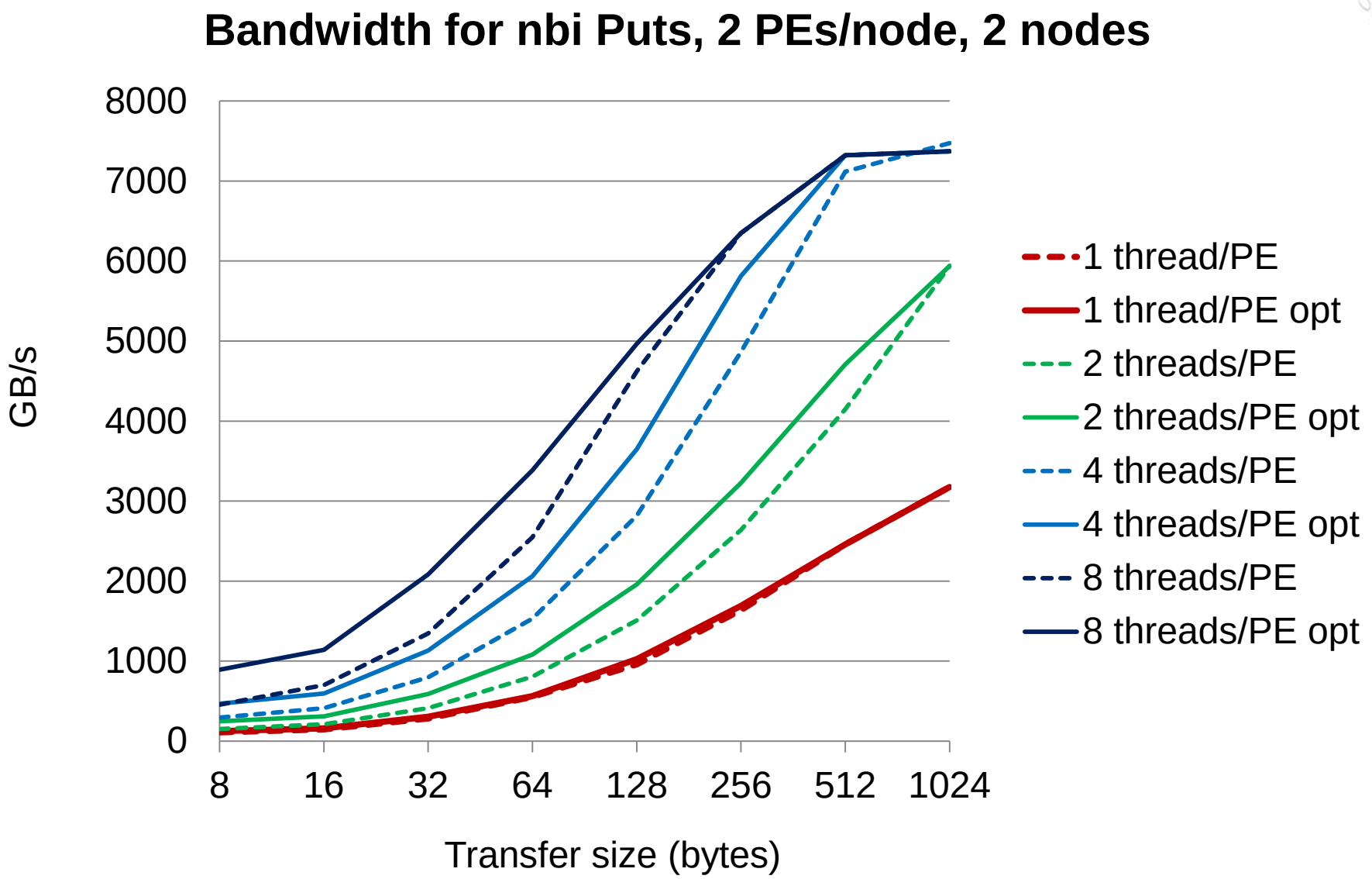
DMAPP Performance Data



Proto-type DMAPP Performance Data



DMAPP Performance Data



Future Work

- Implement thread safety h/w locks optimization
- Implement non-blocking implicit AMOs
- Work with OpenSHMEM community to integrate the Cray extensions into the OpenSHMEM spec, this may require adjustments to the extensions
- Work with community on long-term solutions for thread safety support, error handling, flexible subsets, API compaction etc.
- Remain OpenSHMEM compliant