An Evaluation of Thread-Safe and Contexts-Domains Features in Cray SHMEM

Naveen Namashivayam, David Knaak, Bob Cernohous, Nick Radcliffe, and
Mark Pagel

Cray Inc.

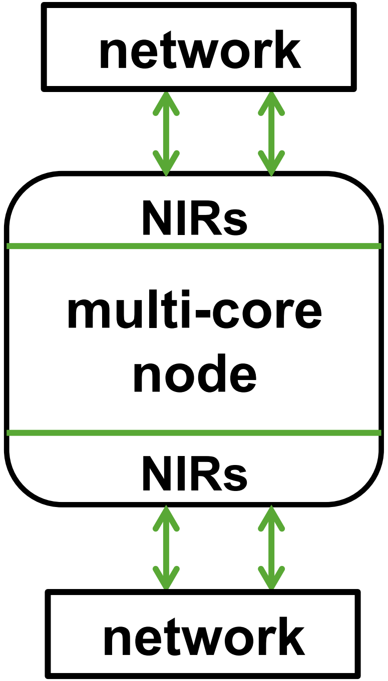
OpenSHMEM 2016: Third workshop on OpenSHMEM and Related Technologies

3-August-2016

Contents

Introduction - What is the Problem?

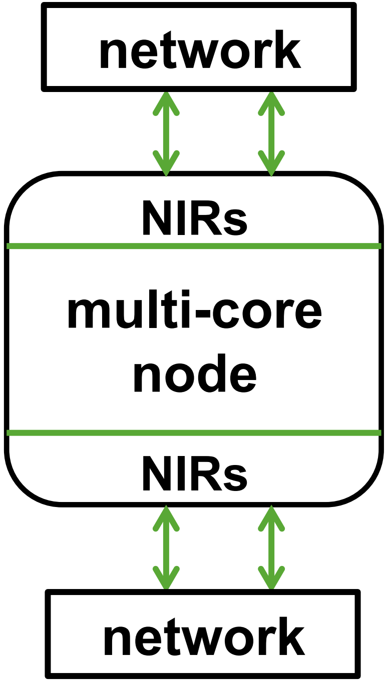
- Typical modern compute nodes have
  - Multiple cores for computation
  - Memory sharable by cores on node
  - Multiple network injection resources (NIR) for communication with other nodes
- We want an OpenSHMEM program to utilize as many HW resources as possible
- The OpenSHMEM API doesn’t support this



Introduction - What is the Problem?

In what way does the OpenSHMEM API not support this?

- Computation performance on-node can be improved with multithreading
  - This decreases the number of PEs as the number of threads increases:
  - nPEs * nThreads = nCores
- But only a PE can make SHMEM calls; so fewer NIRs are utilized
- Interaction between threads and OpenSHMEM routines is NOT yet standardized



Introduction - What is the Problem?

Architecture	Cores per Node	Aries NIRs per Node
Ivy Bridge	10+	~120
Haswell	28+	~120
Broadwell	36+	~120
Knights Landing	250+	~120

Introduction – Possible Solutions

What OpenSHMEM extensions can make it possible to maximum utilization of compute and network resources?

- Thread-Safe routines?
- Contexts-Domains routines?

We will evaluate two different proposals:

- "Thread-safe" proposal from Cray – Ticket #186
- "Contexts-Domains" proposal from Intel – Ticket #177

We will evaluate performance using implementations in Cray SHMEM

Contents

- Introduction – The Problem
- What is Cray SHMEM
- Multithreading in OpenSHMEM
- Thread-safe and Contexts-Domains Design in Cray SHMEM
- Experiments for Design Decisions
- Initial Application Level Evaluation
- Future Work and Conclusion

Cray SHMEM - Background

- Closed source vendor-specific OpenSHMEM implementation
- Part of Message Passing Toolkit (MPT) software stack from Cray Inc.
- OpenSHMEM specification **version-1.3** compliant
- Uses Cray **DMAPP** library as a low-level communication layer
- Apart from standard OpenSHMEM features, supports:
  - **Thread-safety**
  - Multiple-symmetric heaps for heterogeneous memory kinds
  - Flexible PE subsets (teams) creation and management
  - Alltoallv
  - Point-to-point Put with signal
  - Local shared-memory pointers
- Extra features are supported as SHMEMX-prefixed extensions

Contents

Multithreading in OpenSHMEM

- **Interaction between threads and OpenSHMEM routines are not standardized**
- **Multithreading API Objectives:**
  - Able to initiate OpenSHMEM communications from multiple threads
  - Provide the maximum possible utilization of:
    - Computational units (N) - cores and hyper-threads
    - Network Injection Resources (NIR)

Possible Utilizations	Example **Utilization Use Case**
N < NIR	Using Intel Broadwell nodes on Cray Aries Interconnect
N > NIR	Using Intel KNL nodes on Cray Aries Interconnect

  - Isolate users and OpenSHMEM routines from network and hardware resource details as much as possible
- **Two different approaches: “Thread-safe” and “Contexts-Domains”**

Thread-safe Proposal (Ticket #186)

- **Proposed by Cray to be a part of OpenSHMEM standards**
- **Extensions existing as SHMEMX-routines in Cray SHMEM**
- **Design Objective:**
  - Provide a fairly simple way to increase concurrency in multithreaded OpenSHMEM applications by allowing threads to make SHMEM calls and directly mapping threads to network injection resources
- **General Usage Flow:**

Create Threads using external threading library like OpenMP or pthreads → Register Threads with OpenSHMEM library → Enter threaded region → Make OpenSHMEM communication calls within threaded region from registered threads → Unregister and free registered threads

Basic Thread-safe Routines

- **int shmemx_init_thread (int required_threading_level);**
  - required_threading_level – SHMEM_THREAD_SINGLE, SHMEM_THREAD_MULTIPLE
  - Initiate and let the OpenSHMEM implementation know about multithreaded usage
- **void shmemx_thread_register (void);**
  - Register the thread with OpenSHMEM library and get network resource
- **No explicit thread-based RMA or AMO routines**
  - Normal RMA and AMO routines will implicitly be converted into thread-based routines when called by registered threads
- **void shmemx_thread_quiet / fence (void);**
  - Thread-based memory ordering operations
- **void shmemx_thread_unregister (void);**
  - Free the registered thread and release network resource

Contexts-Domains Proposal (Ticket #177)

- **Proposed by Dinan, *et al.* to be part of OpenSHMEM standards**
- **Extensions prototyped as SHMEMX-routines in Cray SHMEM**
- **Design Objectives:**
  - Increase concurrency with independent streams of communication
  - Separate message injection resources from remote completion tracking by introducing two new features: Contexts and Domains
- **Relation between Threads and Contexts-Domains**
  - Two Independent entities, no direct mapping
  - Contexts and Domains are OpenSHMEM objects
  - Contexts and Domains are mapped to network resources
  - Any thread can make use of these objects

Basic Contexts-Domains Routines

- typedef int shmem_ctx_t ; typedef int shmem_domain_t ;
  - Opaque handles for Context and Domain objects
- void shmemx_domain_create(int thread_level, int num_domain, shmem_domain_t domain[ ]) ;
- void shmemx_domain_destroy(int num_domain, shmem_domain_t domain[ ]) ;
  - Routines for creating and maintaining Domain objects
- int shmemx_ctx_create (shmem_domain_t domain, shmem_ctx_t *ctx);
- void shmemx_ctx_destroy (shmem_ctx_t ctx);
  - Routines for creating and maintaining Contexts objects
- void shmemx_ctx_fence / quiet (shmem_ctx_t ctx);
  - Context-based memory ordering routines
- void shmemx_ctx_TYPE_p(TYPE *addr , TYPE value , int pe, shmem_ctx_t ctx);
- void shmemx_ctx_getmem(void *dest , const void *source , size t nelems , int pe , shmem_ctx_t ctx);
- void shmemx_TYPE_inc(TYPE *dest , int pe , shmem_ctx_t ctx);
  - Sample Context-based RMA and AMO routines

Contents

Thread-Safe and Contexts-Domains Designs

3 Possible Solutions Evaluated:

- Thread-Safe Design
- Domain-based Contexts-Domains Design
- Context-based Contexts-Domains Design

Cray DMAPP Overview

- **Underlying low-level communication layer for Cray SHMEM**
- **Support for both Cray Aries and Cray Gemini interconnect**
- **Key Aries hardware features**
  - FMA – Fast Memory Access
  - BTE – Block Transfer Engine
  - CQ – Completion Queue

Network Injection Resources:
FMA – small data sizes
BTE – large data sizes

Event notification

FMA = ~120 **BTE = 2** **CQ = ~2K**

Events = PUT / GET / AMO

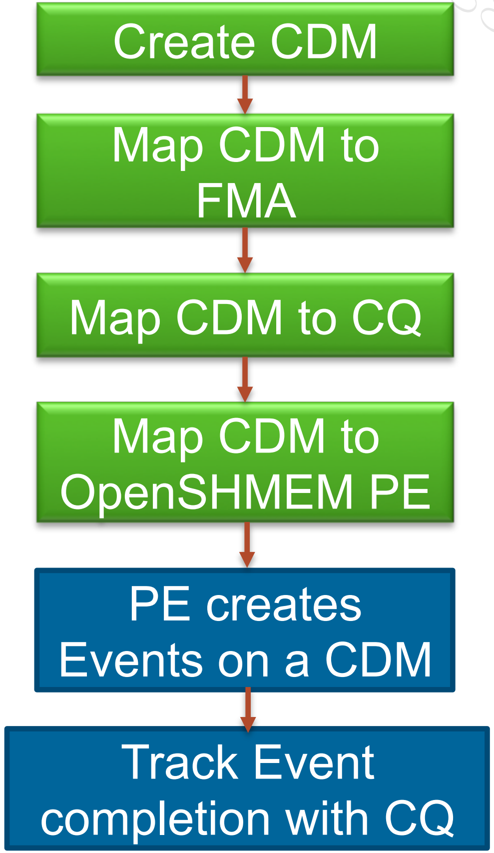
- **Key DMAPP software object for communication**
  - CDM – Communication Domain

DMAPP Design: Mapping to HW Resources

- **FMAs and CQs are key HW mechanisms for communication streams**
- **CDMs are SW objects to attach to FMAs and CQs**
  - CDM has 1-to-1 mapping with FMA
  - CDM has 1-to-1 mapping with CQ
    - Implicit: FMA has 1-to-1 mapping with CQ

Max number of CDMs per node = ~120

- **In single threaded OpenSHMEM Application**
  - 1 unique CDM per PE & PEs cannot share CDM
  - Use CQ for tracking remote completion or memory ordering – *shmem_quiet()* operation
  - PEs create events on a CDM using *cdm_handle*



Thread-Safe Design in Cray SHMEM

- Each registered thread (T) mapped to a CDM
  - T < CDM - Unique CDM per thread
  - T > CDM - CDMs shared by some threads
- The CDM associated with the thread is identified using a handle stored in Thread Local Storage (TLS)
- How is the *shmem_thread_quiet()* performed?
  - T < CDM – Using unique CQ associated with CDM
  - T > CDM – quiet operation is done on all threads that share the CDM, using the shared CQ associated with the CDM

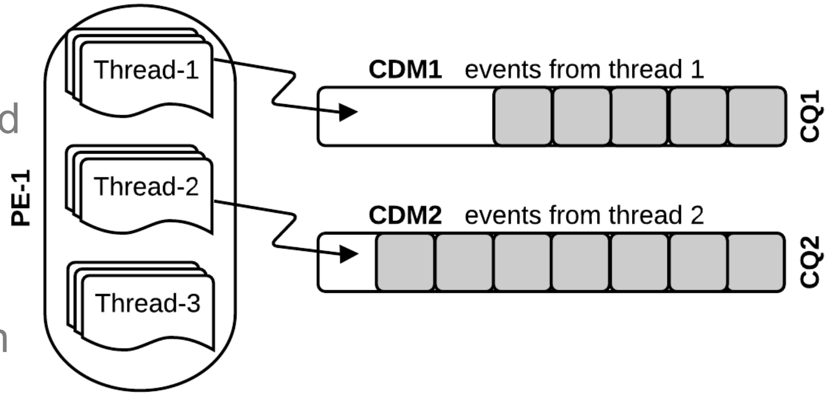


Fig: Thread-safe Design in Cray SHMEM

Domain-based Contexts-Domains Design in Cray SHMEM

- Each Domain object mapped to a CDM
- Each Domain can have multiple Contexts
- All Contexts in a Domain share the CQ
- Cannot use shared CQ to track events for each individual Context
- Each DMAPP event creates a unique *sync_id*
- Track *sync_id*'s as separate queues in SHMEM library level
- Track event completion using this *sync_id* queue
- *shmem_ctx_quiet()*

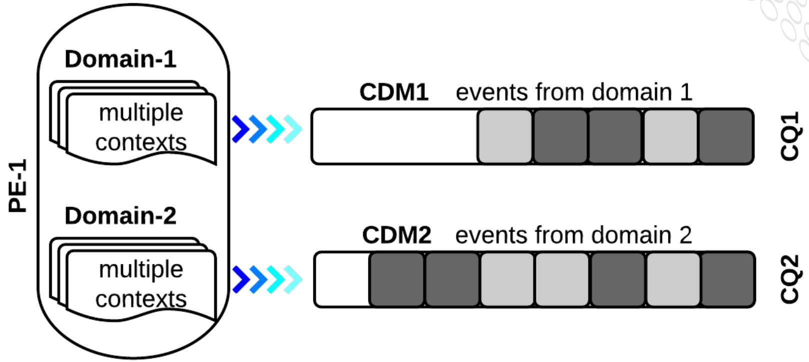


Fig: Domain-based Contexts-Domains Design in Cray SHMEM
(Only DMAPP level mapping are shown)

Context-based Contexts-Domains Design in Cray SHMEM

- Each Domain can have multiple Contexts
- Each Context mapped to a CDM based on the thread level of the Domain it is in
  - SHMEM_THREAD_SINGLE – Unique CDM
  - SHMEM_THREAD_MULTIPLE – Shared CDM
- How is *shmem_ctx_quiet()* performed?
  - Using CQ of the CDM for that Context
- What is the functionality of Domains in this design?
  - Track Context properties ????
  - Group Contexts efficiently for SHMEM_THREAD_MULTIPLE

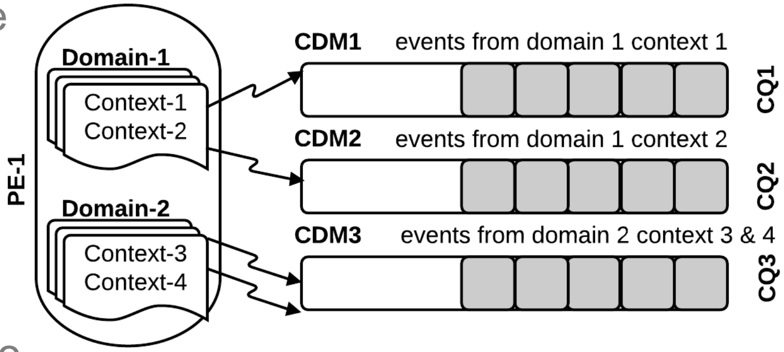


Fig: Context-based Contexts-Domains Design in Cray SHMEM

Contents

Experimental Setup

- **System Details**
  - Cray XC system
  - Cray Aries interconnect architecture
  - 32 core Intel Broadwell processors
  - 2 nodes, 1 PE per node, 32 threads per PE
- **Cray SHMEM version 7.4.0 plus modifications**
  - Used existing SHMEMX-prefixed Thread-safe extensions
  - Created the prototype version of Contexts-Domains extensions
- **Hybrid OpenSHMEM Microbenchmark**
  - Used OSU OpenSHMEM Microbenchmark tests and converted into multithreaded hybrid design
  - Used OpenMP along with OpenSHMEM for hybrid design

Experiment 1: Impact of Thread Local Storage - 1

- **Experiment specific to Thread-safe design**
- **For each thread to track its events, must store in TLS**
- **Performance Impact of using TLS for storing CDM handle**
  - USE_TLS version – use handle stored in TLS for all events
  - NO_TLS version – Explicitly pass handle as part of the event calls in a modified API to avoid TLS
- **Modified OSU Put Microbenchmark**
- **Large data size no change in performance**
- **Small data size less than 512 bytes – shows NO_TLS to perform 8% better than USE_TLS version**

Experiment 1: Impact of Thread Local Storage - 2

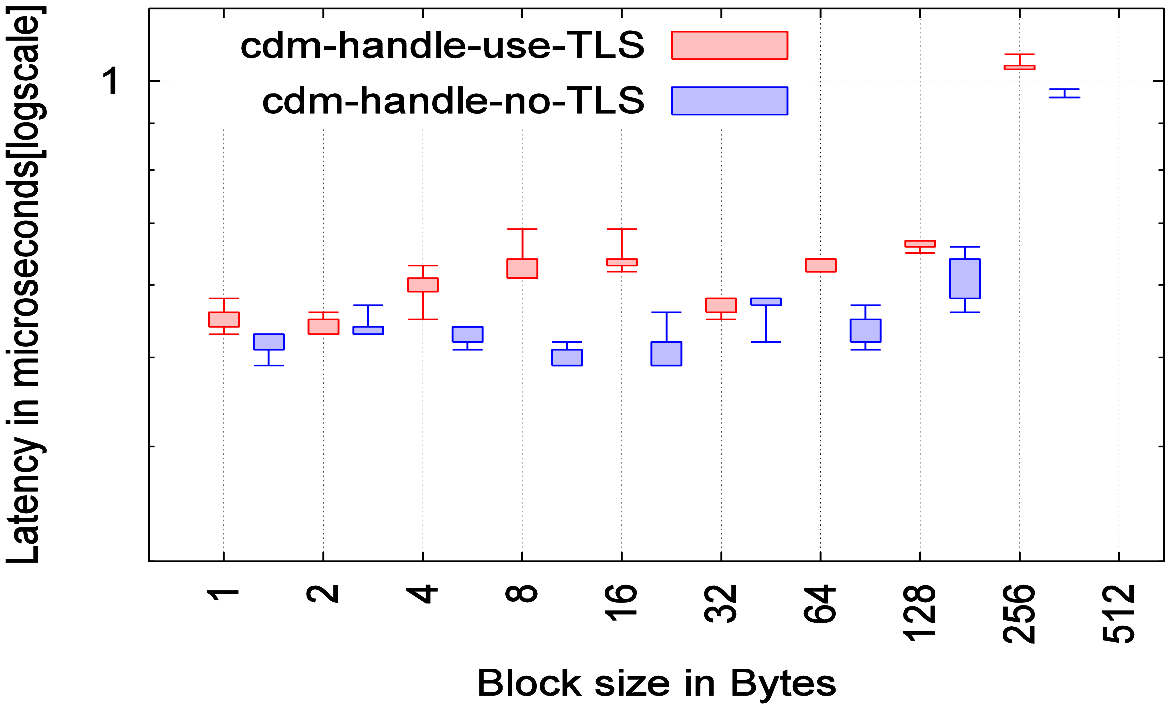
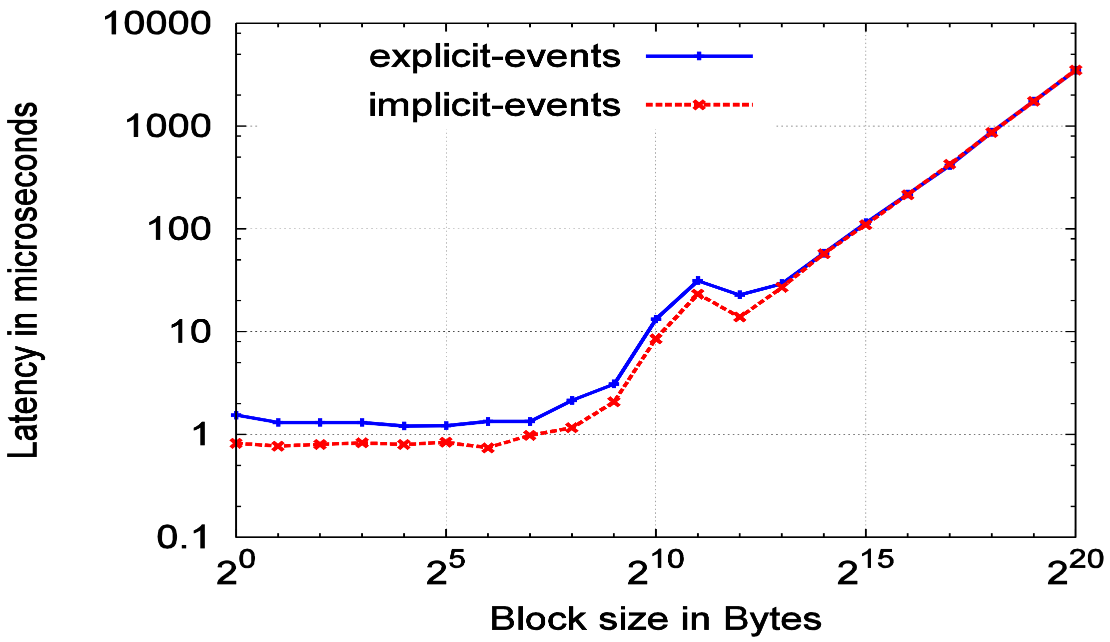


Fig: GCC Compiler 6.1 version

Experiment 2: Explicit or Implicit Non-Blocking Operations - 1

- **Experiment specific to Domain-based Contexts-Domains design**
  - Using *sync_id* for tracking event completion
  - *sync_id's* are not generated for all events
  - Only **Explicit NB events** create sync_id
  - All Domain-based events are Explicit NB
- **Performance Analysis**
  - Modified OSU Put Microbenchmark
  - Create 32 Context-objects and 32 Domains
  - 1 Context-object per Domain
    - All Context-objects have unique CQs

Experiment 2: Explicit or Implicit NB Operations - 1

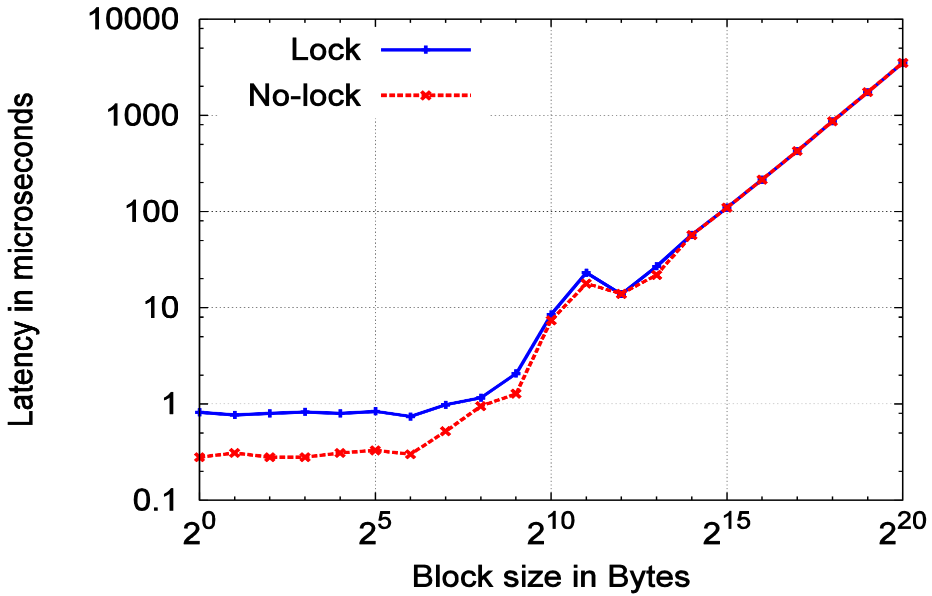


- Data size > 1MB – no performance change
- Data size < 1MB – Implicit events have latency 45% of Explicit events
- DMAPP has event chaining optimization for Implicit events

Experiment 3: Hierarchy of Threading Support - 1

- **Experiment specific to Thread-safe design**
- **Only two different types of thread-levels available now**
  - SHMEM_THREAD_SINGLE – No Lock
  - SHMEM_THREAD_MULTIPLE – Implicit Lock
- **Problem**
  - Even if Number of Threads < CDMs
    - SHMEM_THREAD_MULTIPLE has implicit locks
- **Cannot determine the number of registered threads to avoid implicit locking**
- **Major disadvantage in mapping threads directly to network resources**

Experiment 3: Hierarchy of Threading Support - 2



- 2 PEs – 1 PE per Node
- 32 registered threads per PE
- **No-lock has latency that is 25% of lock**

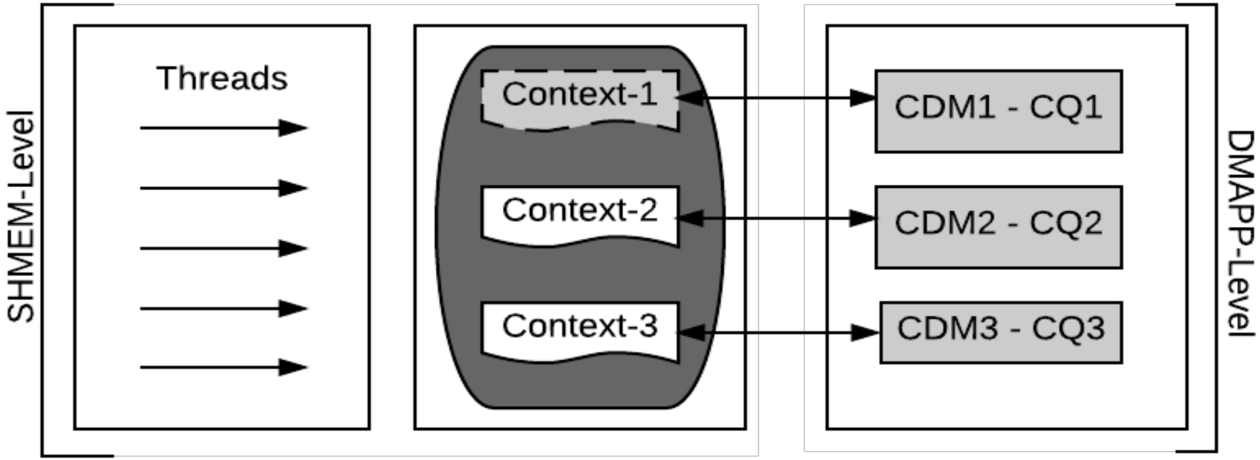
Efficient Network Resources Utilization - 1

- **Distinct trend in growing network resource demand w.r.t multi-core architectures**
- **Need for efficient resource mapping**
- **Problems in the Thread-safe design**
  - T < NIR – Excess streams are wasted
  - T > NIR – Insufficient hints for optimal mapping
    - Every thread gets equal performance priority
    - Even if over allocation is on a particular application module, performance is normalized in all the modules
    - SHMEM_MAX_NUM_THREADS is an insufficient hint because xxx

Efficient Network Resources Utilization - 2

- Contexts-Domains can better maximize use of CDMs
- Threads and Context-Domain objects are separate entities
- Context-Domain objects are mapped to CDMs
- Any thread can pick and use the objects
- T < NIR
  - Use multiple Context-objects per Thread for better CDM utilization
- T > NIR
  - Create priority on particular Context-objects
  - Useful for more unbalanced loads

Efficient Network Resources Utilization - 3



Registered or Unregistered Threads in the Application

Contexts and Domain objects created and handles made visible for threads

DMAPP Library Level - Resource Mapping based on either Context-based or domain-based design

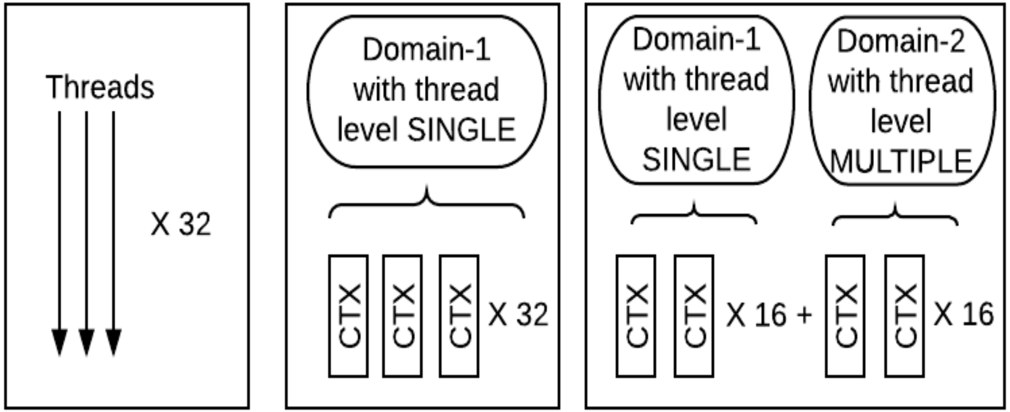
Contents

Initial Application Level Evaluation - 1

- **Analyze impact of efficient network resource mapping on application**
- **Multithreaded implementation of all-to-all collective communication pattern**
- **Three different version**
  - Thread_safe_version (TS) version
    - 32 registered thread per PE
  - Context_design_1 (CTX1)
    - 1 Domain, 32 Contexts, 32 Threads
    - All Contexts with SINGLE as property
    - Each thread use 1 Context-object
  - Context_design_2 (CTX2)
    - 2 Domains, 32 Contexts, 32 Threads
    - Domain-1: Property SINGLE with 16 Contexts
    - Domain-2: Property MULTIPLE with 16 Contexts

Initial Application Level Evaluation - 2

Alltoall done 3 different ways

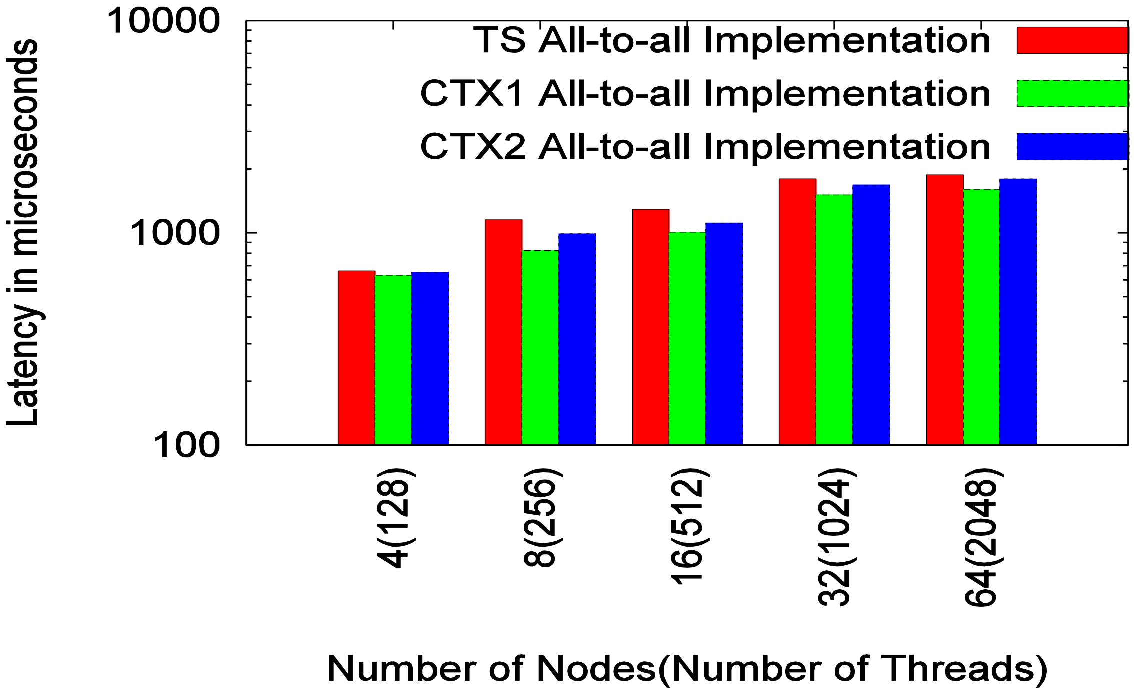


All-to-all Pattern Implementation with Thread-safe Extensions

Design-1 of All-to-all Pattern Implementation with Context-Domain Extensions

Design-2 of All-to-all Pattern Implementation with Context-Domain Extensions

Initial Application Level Evaluation - 3



CTX1 is 18% better than TS and 7% better than CTX2

Contents

- Introduction – The Problem
- What is Cray SHMEM
- Multithreading in OpenSHMEM
- Thread-safe and Contexts-Domains Design in Cray SHMEM
- Experiments for Design Decisions
- Initial Application Level Evaluation
- Future Work and Conclusion

Future Work

- **Analysis in this work are from implementer's perspective**
  - Identified the areas to tap complete utilization of the network resources and computational units
- **Evaluate these proposals more from a user's perspective**
  - Study on different usage scenarios w.r.t the suitability of using features from a particular proposal
  - Performance analysis with a balanced and unbalanced application
    - Balanced Application - Equal workload on all threads
    - Unbalanced Application - Unequal workloads on threads
  - Unequal workload on threads helps to identify the usage of Context objects with different properties

Conclusion

- We need an OpenSHMEM API that makes possible maximum utilization of HW compute and network injection resources
- Thread-Safe proposal is a simple API that can maximize utilization of cores but not necessarily NIRs
- Contexts-Domains proposal is somewhat more complicated but has better potential to maximize utilization of cores and NIRs
- Both extensions can be used in a single program but not in the same parallel region
- To get maximum utilization for different HW resource combinations requires some additional API
- Both proposals deserve attention by OpenSHMEM Committee

Thank You