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

Naveen N Ravichandrasekaran, David Knaak, Bob Cernohous, Nick Radcliffe, and Mark Pagel

Programming Env. – Message Passing Toolkit
Cray Inc.

Intel PGAS Tech Session
1-September-2016

Introduction

- **What is OpenSHMEM?**
  - Partitioned Global Address Space (PGAS) library interface **specification**
  - Aims to provide standard API for SHMEM libraries
  - Cray SHMEM is a SHMEM library implementation from Cray Inc. which follows the OpenSHMEM standards
- **Multithreading in OpenSHMEM**
  - Interaction between threads and OpenSHMEM routines are **NOT** yet standardized
  - Two different proposals:
    - "Thread-safe" proposal from Cray Inc. – Ticket #186 and #218
    - "Contexts-Domains" proposal from Intel – Ticket #177
- **What is this presentation about?**
  - Early evaluation of the two different proposals using Cray SHMEM
  - Study mostly on the resource mapping

Contents

- **Problem Statement**
- Multithreading in OpenSHMEM standards – Background
- Thread-safe and Contexts-Domains Design in Cray SHMEM
- Experiments for Design Decisions
- Initial Application Level Evaluation
- Future Work and Conclusion

Problem Statement – Current Scenario

- **Typical modern compute nodes**
  - Multiple computational units (N) – cores and threads
  - Memory shared by computational units on node
  - Multiple network injection resource (NIR) for communication across nodes
- **We want OpenSHMEM program to utilize as many HW resources as possible**

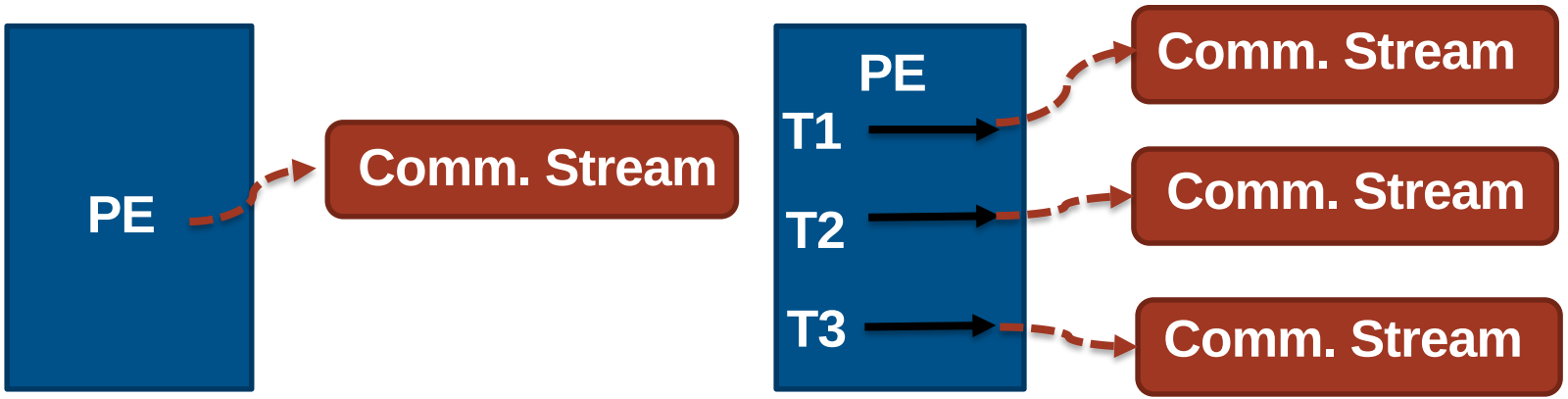
Architecture	Threads per Node	Aries NIR per Node
Ivy Bridge	40+	~120
Haswell	56+	~120
Broadwell	70+	~120
Knights Landing	250+	~120

N < NIR

N > NIR

Problem Statement – OpenSHMEM API

- **Current Standard – Allows one communication stream per PE**
  - Example: Single-threaded scenario
  - NIR underutilized incase of N < NIR (e.g. Broadwell, Haswell)
- **Required – Multiple communication streams per PE**
  - Possibly one communication stream per thread – Multi-threaded scenario



Single-threaded application **Multi-threaded application**

Multithreading in OpenSHMEM

- **Able to initiate OpenSHMEM communications from multiple threads**
- **Provide the maximum possible utilization of**
  - Computational unit, and
  - Network resources
- **Provide possible abstraction for users from network and hardware resource details**
- **Two different approaches:**
  - “Thread-safe” and
  - “Contexts-Domains”

Contents

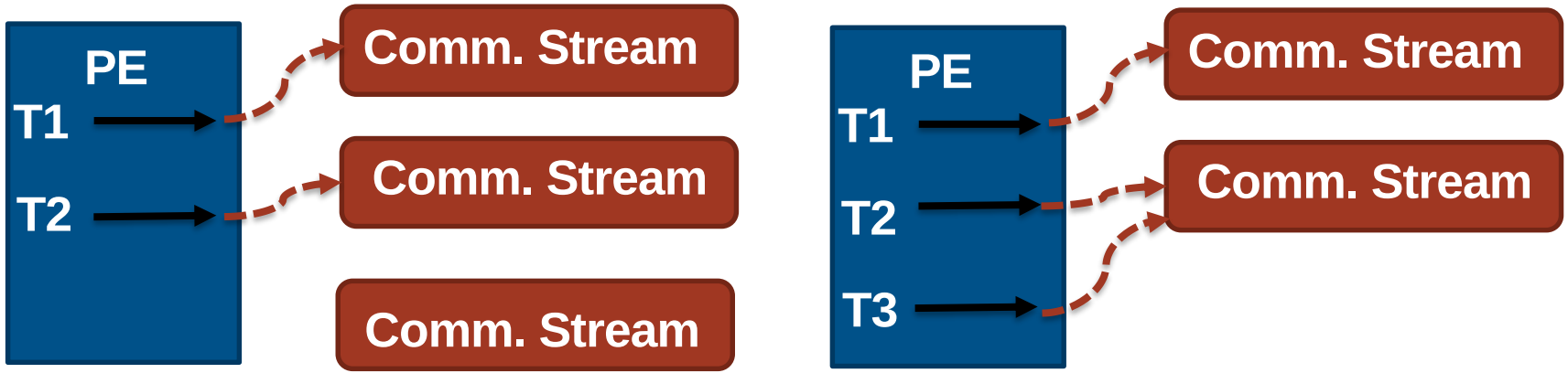
- Problem Statement
- **Multithreading OpenSHMEM proposals – Background**
- Thread-safe and Contexts-Domains Design in Cray SHMEM
- Experiments for Design Decisions
- Initial Application Level Evaluation
- Future Work and Conclusion

Cray SHMEM - Background

- Closed source vendor-specific OpenSHMEM implementation
- Part of Message Passing Toolkit (MPT) software stack from Cray Inc.
- Use DMAPP (Distributed Shared Memory Application) library as a low-level communication layer
- OpenSHMEM specification version-1.3 compliant
- Apart from standard OpenSHMEM features, supports:
  - Thread-safe extensions
  - Support for multiple-symmetric heap for heterogeneous memory kinds
  - Flexible PE subsets creation and management – OpenSHMEM Teams
  - Point-to-point put operation with signal, and
  - Local shared-memory pointers
- Extra features are supported as SHMEMX-prefixed extensions

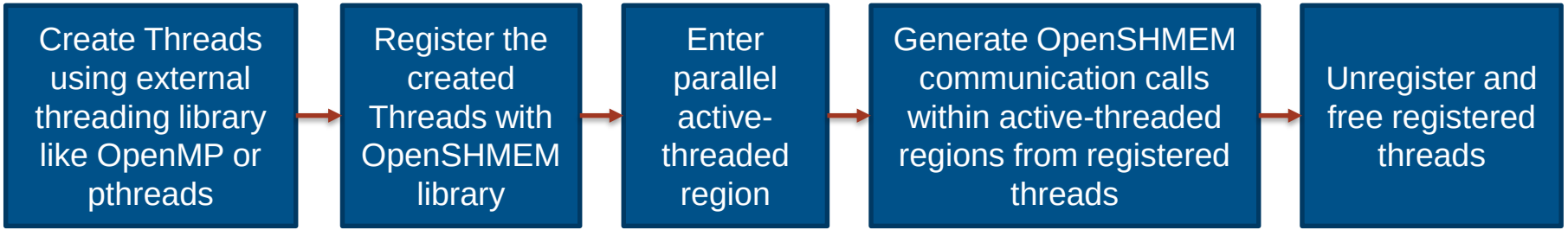
Thread-safe Proposal(Ticket #186)

- **Extensions are available as SHMEMX-routines in Cray SHMEM**
- **Design Objective:**
  - Provide a fairly simple way to increase communication concurrency in multithreaded OpenSHMEM applications by **directly mapping threads** to network resources
- **Basic design Overview:**
  - If threads < NIR – each thread gets a unique NIR
  - If threads > NIR – some threads are forced to share a NIR



Thread-safe Proposal(Ticket #186)

- **General Usage Directions:**
  - Initiating OpenSHMEM communication from multiple threads



- **Discussions beyond the scope for this presentation:**
  - Does this model cover all nested OpenMP scenarios?
  - Does this model work with all threading models?
    - Seems to identify “shepherds” in Qthreads
  - How do we handle multiple initialization and finalize calls?
  - Should we make collectives as thread-safe calls?

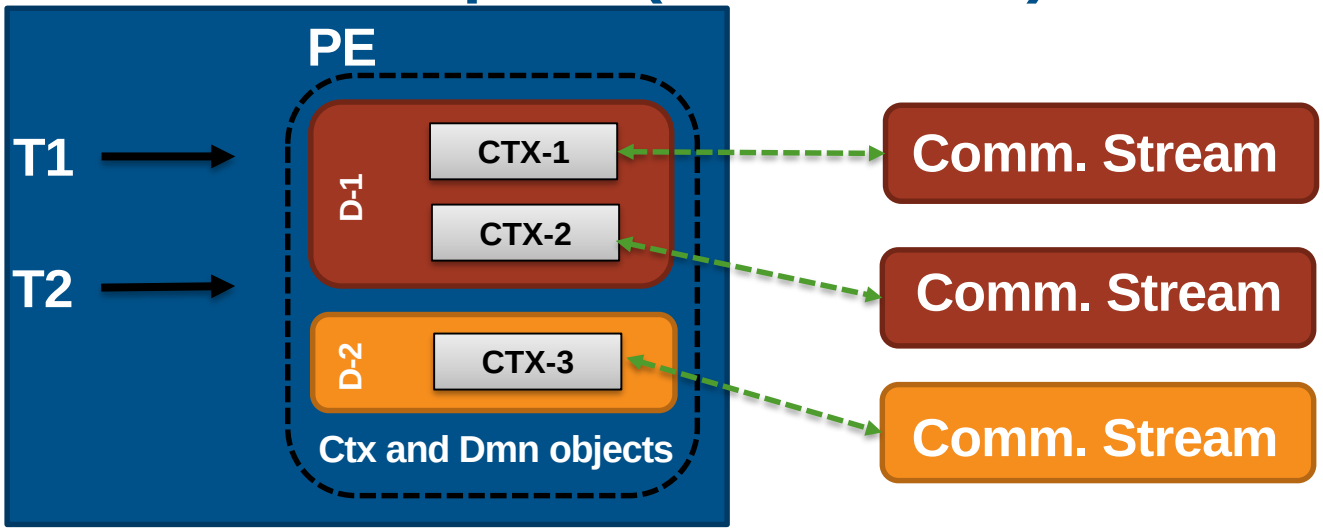
Basic Thread-safe Extensions

- **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
- **void shmemx_thread_unregister (void);**
  - Free the registered thread, and release network resource
- **void shmemx_thread_quiet / fence (void);**
  - Thread based memory ordering operations
- **No explicit thread-based RMA, or AMO routines**
  - Normal RMA, and AMO routines will implicitly be converted into thread-based routines on used from registered threads

Contexts-Domains Proposal(Ticket #177)

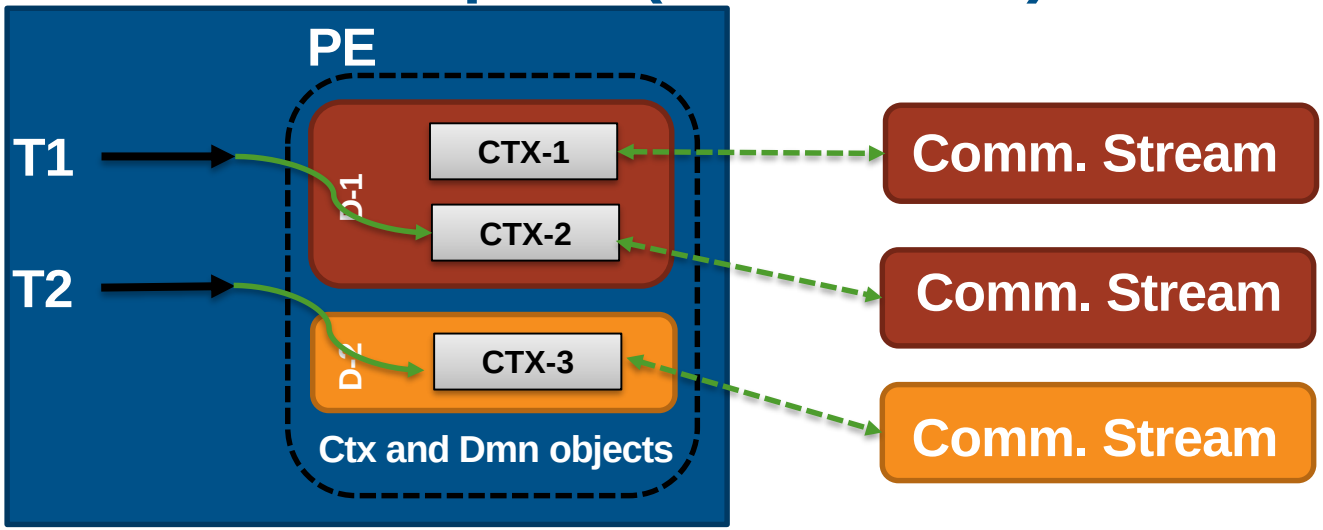
- **Proposed by Dinan, *et al.* to be part of OpenSHMEM standards**
- **Extensions prototyped as SHMEMX-routines in Cray SHMEM**
- **Design Objective:**
  - Increase concurrency with independent streams of communication, and
  - Separate message injection from remote completion tracking by introducing two new features in OpenSHMEM: Contexts and Domains
- **Context is a separate communication stream**
  - Can perform memory ordering on (only) the Contexts
- **Domain is a group of contexts which share a same property**
  - All properties are not yet defined
  - Example property: Thread-level - SHMEM_THREAD_SINGLE/MULTIPLE

Contexts-Domains Proposal(Ticket #177)



- **Relation between Threads and Contexts-Domains**
  - **Two Independent entities, no direct mapping**
  - Contexts-Domains are another OpenSHMEM objects – made visible to all threads
  - Contexts-Domains are mapped to network resources
  - Any threads can make use of these objects based on their property
- **No thread registration required – any thread can use Contexts-Domains objects**

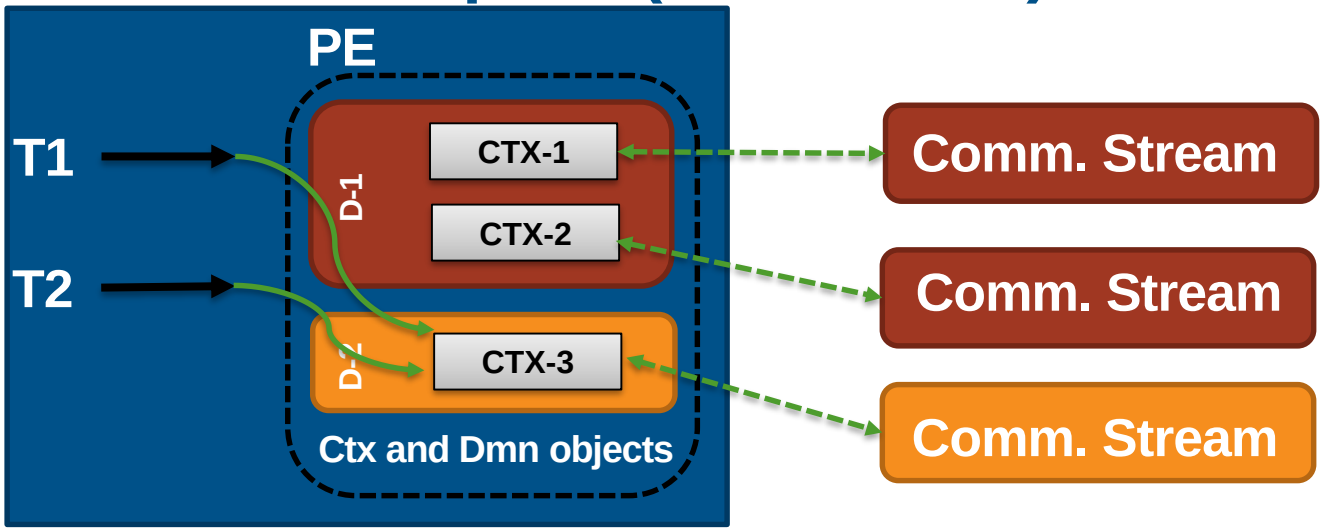
Contexts-Domains Proposal(Ticket #177)



Threads should use Context objects based on the object property

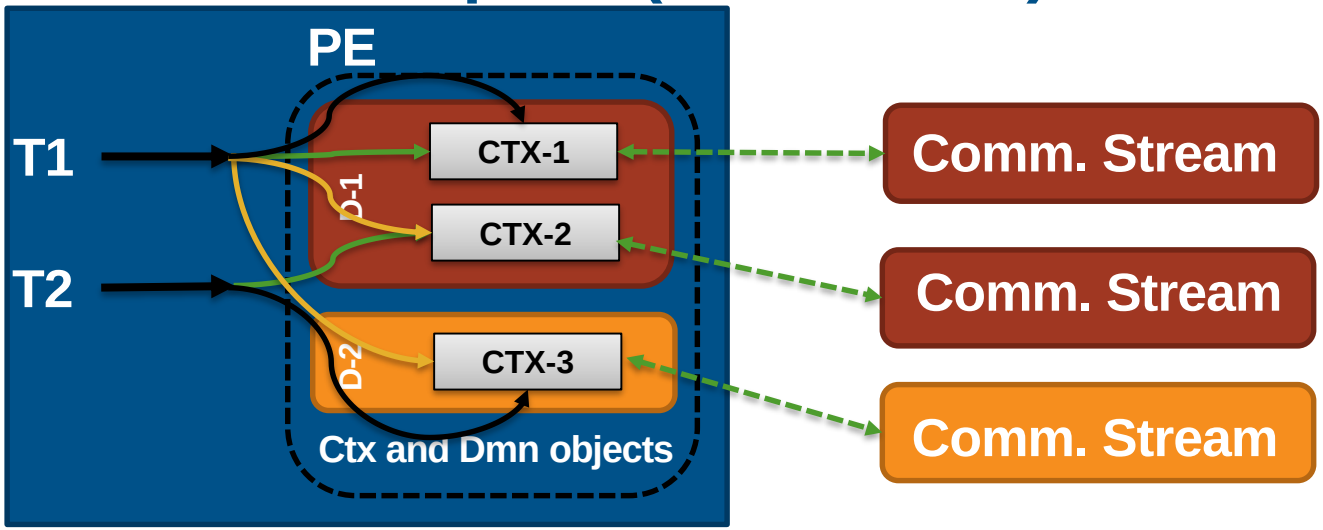
- **SHMEM_THREAD_SINGLE – only one thread should access Context object at a time**
- Consider CTX-1, CTX-2, and CTX-3 to be SHMEM_THREAD_SINGLE
- T1 using CTX-2 and T2 using CTX-3 is correct
- T1 and T2 simultaneously using CTX-2 is wrong

Contexts-Domains Proposal(Ticket #177)



- Threads should use Context objects based on the object property
  - **SHMEM_THREAD_MULTIPLE – multiple threads can access the same Context object concurrently**
  - Consider CTX-1, CTX-2, and CTX-3 to be SHMEM_THREAD_MULTIPLE
  - T1 and T2 can simultaneously use CTX-3

Contexts-Domains Proposal(Ticket #177)



- **Threads and Domain-object relation**
  - Threads are not mapped to Domains
  - No restriction on the usage
  - All combinations are allowed in accessing a Context object belonging to a Domain group, provided the Context property is satisfied

Basic Contexts-Domains Extensions

- 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_ctx_TYPE_inc(TYPE *dest , int pe , shmem_ctx_t ctx);
  - Sample Context-based AMO, and RMO operations

Contents

- Problem Statement
- Multithreading OpenSHMEM proposals – Background
- **Thread-safe and Contexts-Domains Design in Cray SHMEM**
- Experiments for Design Decisions
- Initial Application Level Evaluation
- Future Work and Conclusion

DMAPP Overview

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

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

→ **Event notification mechanism**

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

Events = PUT / GET / AMO

- Key DMAPP software object for communication
  - CDM – Communication Domains

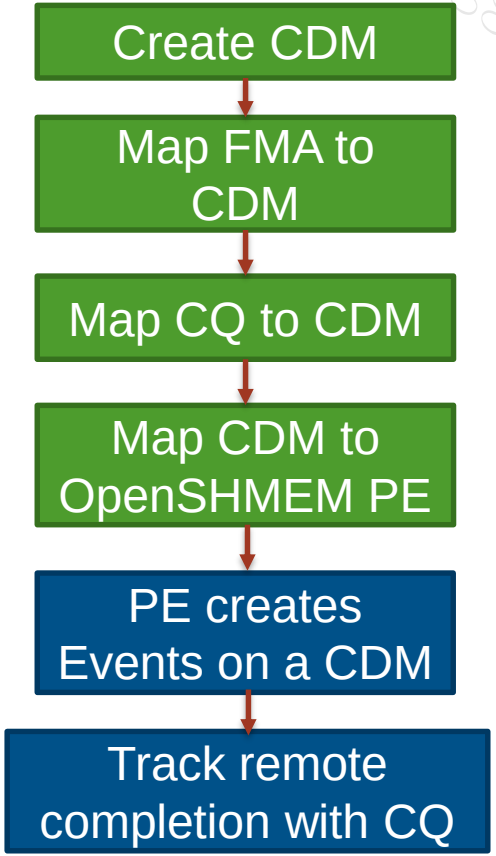
* Values are approximated and would possibly change with MPI-hybrid, IO usage. Refer to Aries Software development guide for complete details

DMAPP Design and OpenSHMEM Mapping

- **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

Number of max CDM 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 *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 corresponding 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

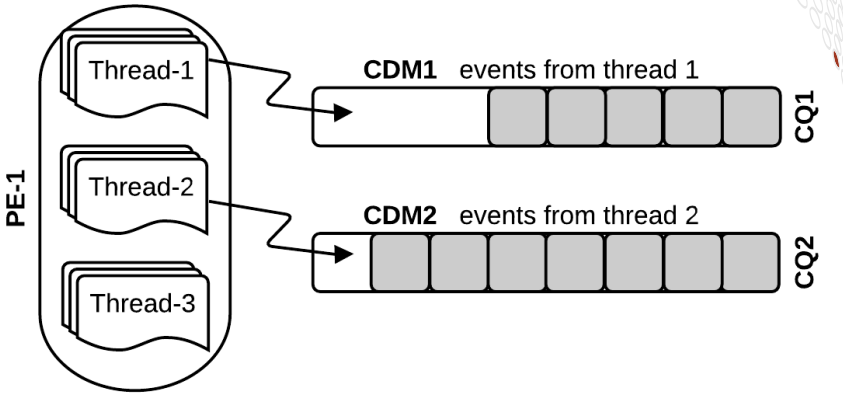


Fig: Thread-safe Design in Cray SHMEM

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 same CQ
- Cannot use CQ to track events for each individual Context
- **Each DMAPP events creates a unique *sync_id***
  - **Track *sync_id*'s as separate queues in SHMEM library level**
  - Track event completion using this *sync_id* queue for *shmem_ctx_quiet()*
- Issues:
  - Multiple threads can't use contexts that belong to the same Domain concurrently on SHMEM_THREAD_SINGLE

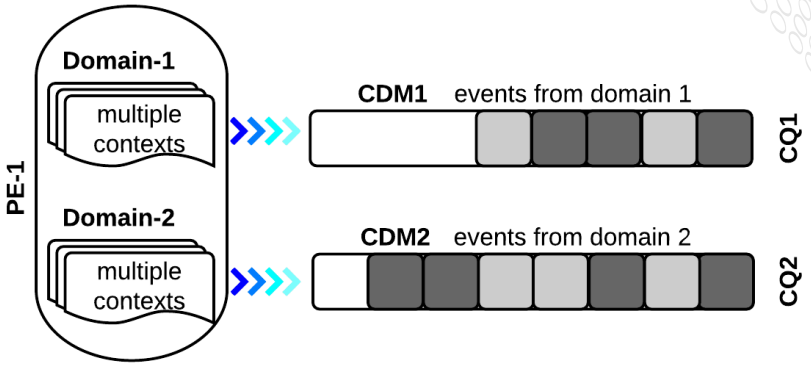


Fig: Domain-based Contexts-Domains Design in Cray SHMEM
(*Only DMAPP level mapping are shown*)

Context-based Contexts-Domains Design in Cray SHMEM

- Each Domain can have multiple Contexts
- Each Context is 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 object
- Issues:
  - Need for Domains – Group Contexts efficiently for SHMEM_THREAD_MULTIPLE
  - *Without* some config knowledge we can't make good decisions
  - Intermediate solution
    - SINGLE -> Unique CDM
    - MULTIPLE -> Shared CDM

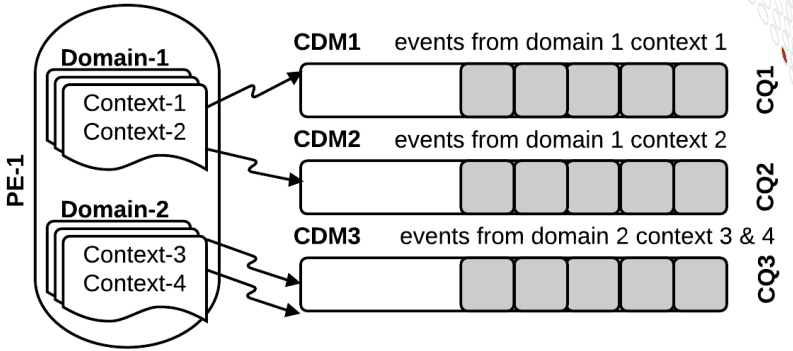


Fig: Context-based Contexts-Domains Design in Cray SHMEM

Contents

- Problem Statement
- Multithreading OpenSHMEM proposals – Background
- Thread-safe and Contexts-Domains Design in Cray SHMEM
- **Experiments for Design Decisions**
- Initial Application Level Evaluation
- Future Work and Conclusion

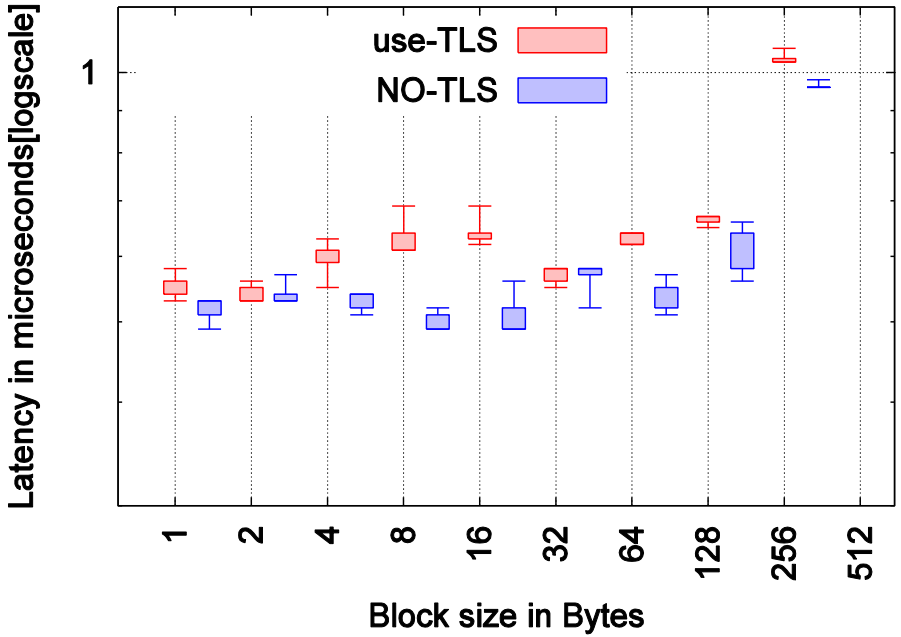
Experimental Setup

- **System Details**
  - Cray XC system
  - Cray Aries interconnect architecture
  - 32 core Intel Broadwell processors per node
  - 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 – Context-based and Thread-safe-based
  - Used OpenMP along with OpenSHMEM for hybrid design

Impact of Thread Local Storage(TLS) – 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 handle to access CDM**
  - 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 lookup
- **Large data size no change in performance**
- **Modified OSU Put Microbenchmark**

Impact of Thread Local Storage(TLS) – 2



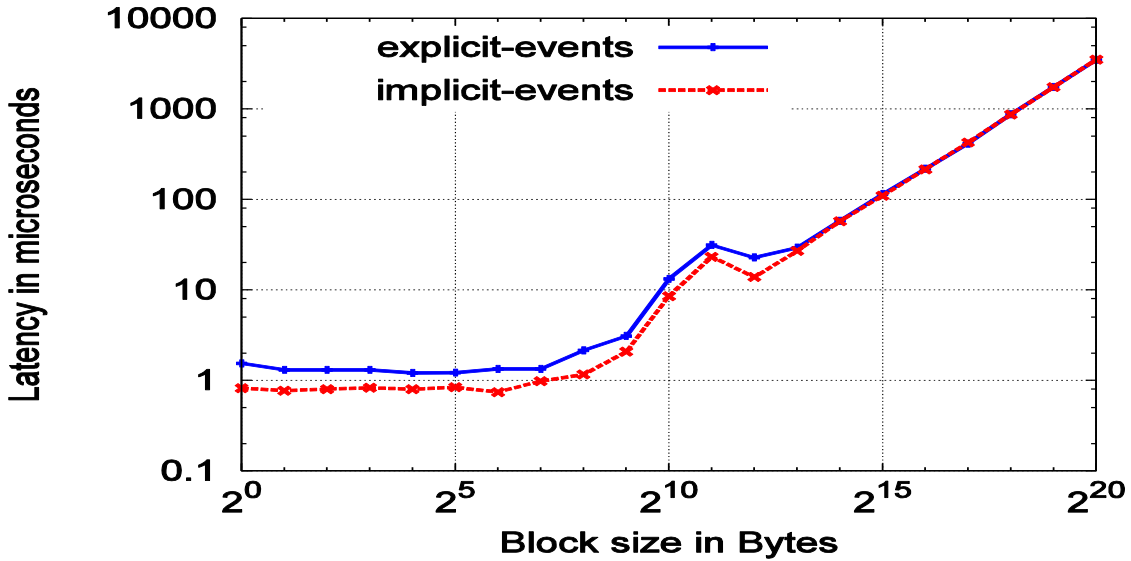
- Small data size less than 512 bytes – Shows NO_TLS to perform with 8% better latency than USE_TLS version

Fig: GCC Compiler 6.1 version

Usage of Explicit and Implicit NB 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

Usage of Explicit and Implicit NB Operations – 2

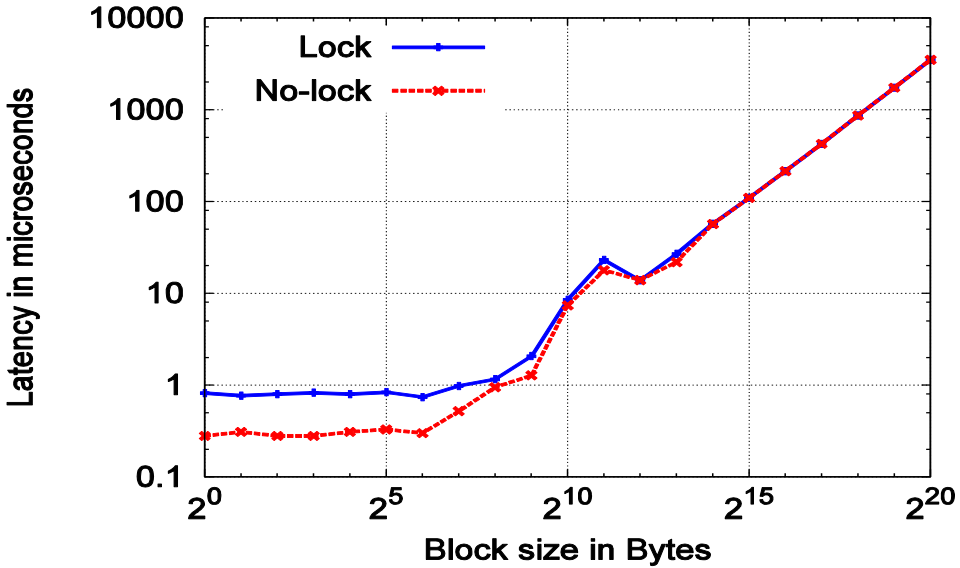


- Data size > 1MB – no perf change
- Data size < 1MB – Implicit events have 45% better latency than Explicit events
- DMAPP has event chaining optimization for Implicit events

Hierarchy of Threading Support – 1

- **Experiment specific to Thread-safe design**
- **Major disadvantage in mapping threads directly to network resources**
- **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**

Hierarchy of Threading Support – 2



- 2 PEs – 1 PE per Node
- 32 registered threads per PE
- Modified OSU Put Microbenchmark
- **No-lock has 25% better latency than Implicit-lock based design**

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

Efficient Network Resources Utilization – 2

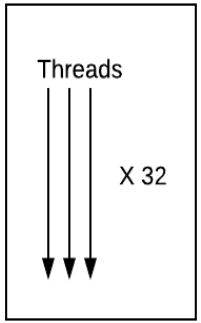
- **Contexts-Domains can better maximize use of CDMs**
- **Threads and Contexts-Domains objects are separate entities**
- **Contexts-Domains objects are mapped to CDMs**
- **Any thread can pick and use the objects**
- **T < NIR**
  - Use multiple Context-objects per Thread for better CDM utilization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

Contents

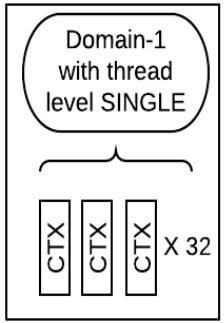
- Problem Statement
- Multithreading OpenSHMEM proposals – Background
- Thread-safe and Contexts-Domains Design in Cray SHMEM
- Experiments for Design Decisions
- **Initial Application Level Evaluation**
- Future Work and Conclusion

Initial Application Level Evaluation

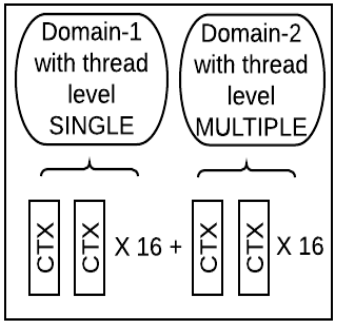
- Analyze impact of efficient network resource mapping
- Multithreaded implementation of all-to-all collective communication pattern
- Three different version
  - Thread_safe_version(TS) version
    - 32 registered thread per PE
  - Context_design_1(CTX1)
    - 1 Domain, 32 Contexts, 32 Threads
    - 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



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

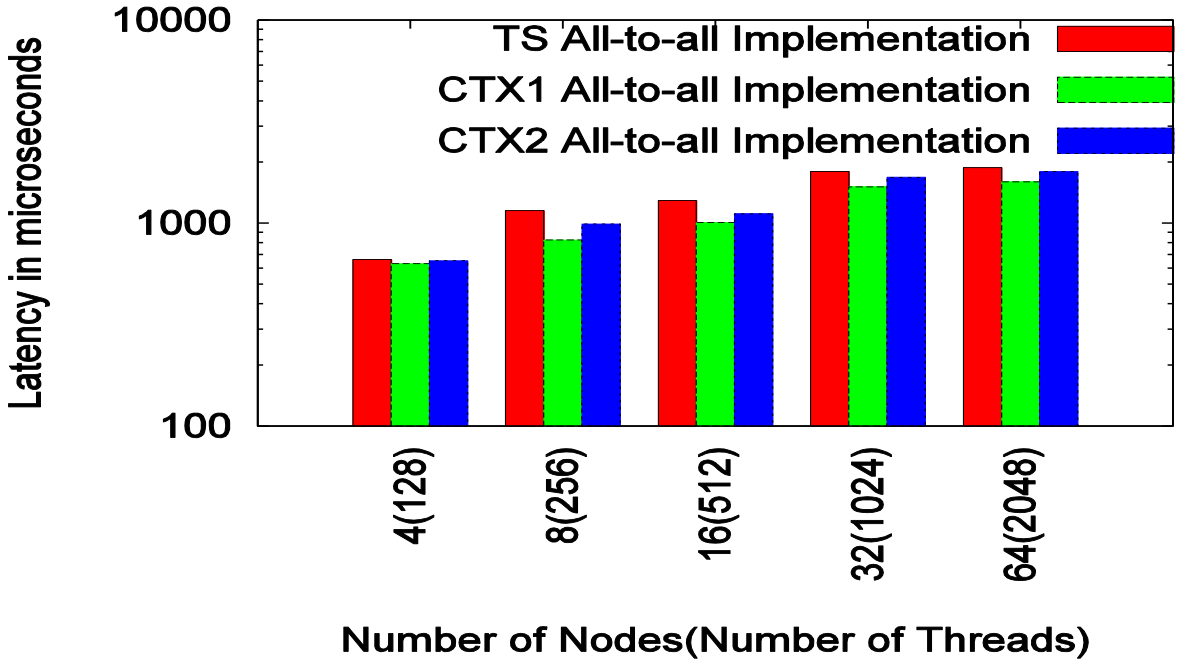


Design-1 of All-to-all Pattern Implementation with Context-Domain Extensions



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

Initial Application Level Evaluation



CTX1 to be 18% better than TS, and 7% better than CTX2

Contents

- Problem Statement
- Multithreading OpenSHMEM proposals – Background
- Thread-safe and Contexts-Domains Design in Cray SHMEM
- Experiments for Design Decisions
- Initial Experimental 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**
  - Separating users from network resource details
  - Using some kind of configuration hints – **shmemx_domain_config**
  - 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**
  - Introduce users to a new layer of network properties
  - Explicit control for resource allocation
  - Obtain performance as close to the underlying communication layers
- **Both proposals deserve attention from OpenSHMEM Committee**
  - Interaction between user and library for resource mapping
  - A right level of abstraction is necessary along with sufficient hints

Thank You

Backup

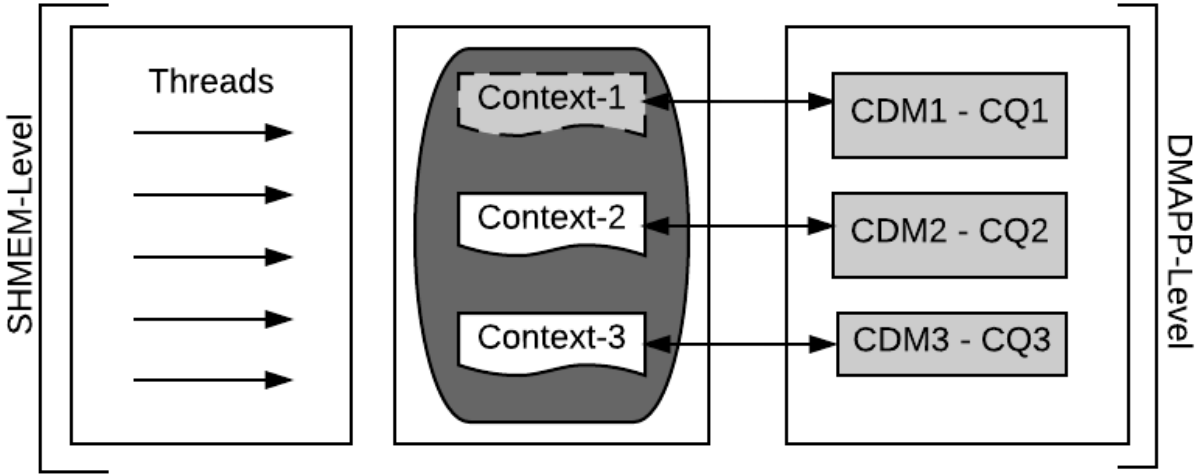
Possible Usage Scenarios

- **Thread-safe**
  - Balanced Workload + over allocation(N > NIR)
- **Context-Domain**
  - Balanced Workload + under-allocation(N < NIR)
    - Multiple Context per Domain design – Possibility for multiple synchronization points with Compute + Communication overlap
    - One Context per Domain and assign each Domain to a thread
  - Unbalanced Workload + any kind of allocation
    - Possibility to select threads for high-priority work loads

Problems in Contexts-Domains Prototype

- Without knowing the number of Domains that will be created before the actual Domain creation call – not possible to come up with an efficient resource allocation design
- Need for shmemx_domain_config(..) just a hint routine

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