Context-Domain Prototype Design in Cray SHMEM

Naveen Namashivayam, David Knaak, Bob Cernohous

OpenSHMEM Multithreading WG
June 14, 2016

Introduction

What is this presentation about ?

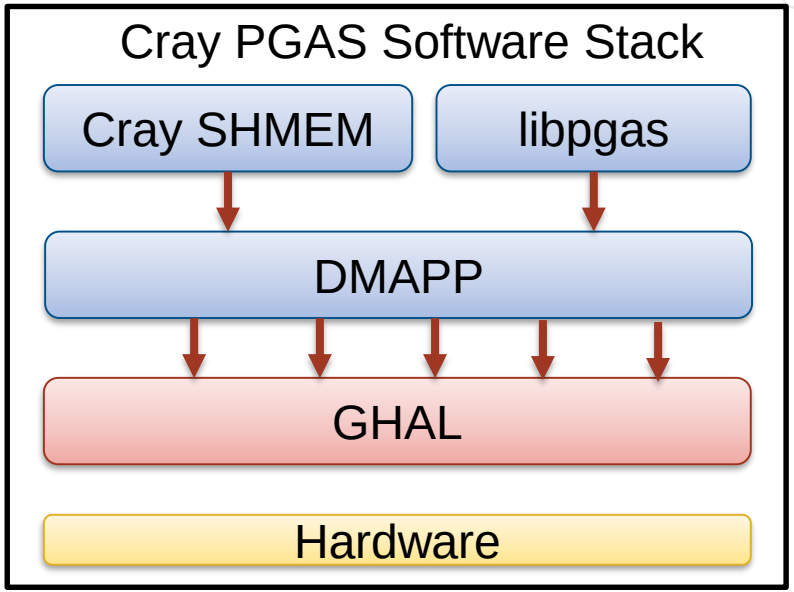
- **Brief Introduction to Cray SHMEM over DMAPP**
- **Initial Context-Domain Feature Analysis**
- **What is the real problem that Context-Domain Proposal Address ?**
- **Context-Domain Prototype Design(s) in Cray SHMEM for Aries Interconnect**
  - Performance Impact of Explicit vs Implicit Events
  - Resource Mapping over Contexts
  - Resource Mapping over Domains
- **Suggestions for Effective Resource Mapping**

What is this presentation not about ?

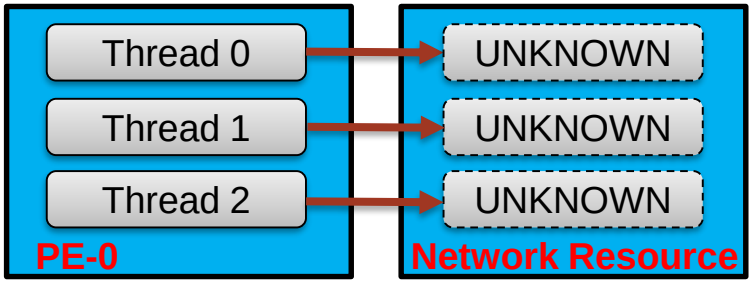
- **Comparison between Thread-safe (#186, #218) and Context features (#177)**

DMAPP Overview

- **Underlying communication layer for Cray SHMEM**
- **Make use of GNI APIs**
- **Has support for –**
  - One-sided RMA operations
    - Strided and Contiguous Data transfer
  - Scatter/Gather operations
  - Atomic Memory Operations
  - Synchronization Events
  - Collective operations, and
  - Symmetric Heap Management

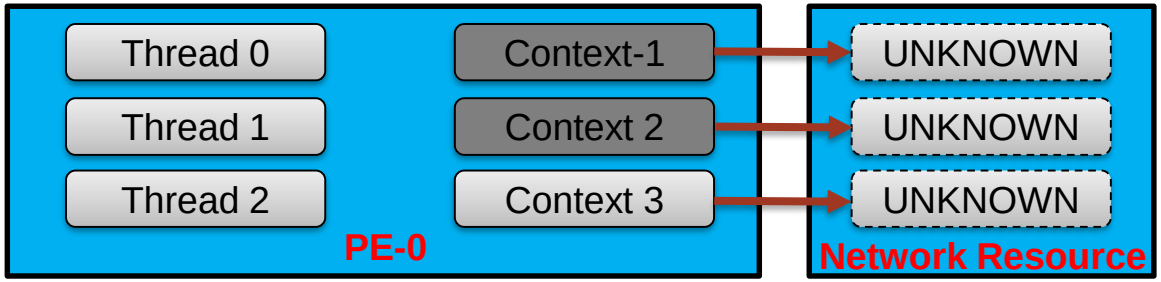


Brief Overview of Thread-safe Design (#186)



- **Network Resource is a black box**
  - User Perspective – Network Resources are Unknown Variables
  - Users doesn't know the thread mapping?
  - Users have no Control over the mapping
  - Each thread may have its own resource or threads can share resources
- **Positives**
  - SHMEM is not a low-level library
  - Users shouldn't be forced to understand the varying Network Architectures
- **Negatives**
  - Limit the optimization level for the implementation to perform efficient resource mapping
  - Only hint : SHMEM_MAX_THREADS env variable & Thread-level (single, multiple, serial, funnel)

Context-Domain as Intermediate Layer



- **Still Network Resource is a black-box**
- **Relationship between Threads and Context-Domain**
  - Two separate independent entities
  - Any thread can create a context and make context objects available for usage by other threads
  - But, usage depends on the properties by which the contexts are created
  - Users have complete control on mapping threads to contexts/domains at application level
  - Provides sufficient hints to the implementation for better resource mapping
- **Being Cautious with this Design**
  - Should decide on the right level of abstraction for the users
  - Not expose the complete network resource mapping to the users

Context-Domain Proposal Overview

- **Not just an OpenSHMEM proposal for Multithreading**
- **Two Important Features:**
  - **Contexts –** Splits up and separates Message Injection and Remote completion tracking
  - **Domains –** Group Contexts and provide better hints for Resource Mapping
- **Separates Message Injection and Remote completion tracking**
  - Fine-grain Synchronization with separate Communication streams called **Contexts**
  - Context based quiet using shmem_ctx_quiet()
  - Even in a single-threaded application you can get better synchronization
- **Efficient Resource Mapping**
  - **Domain = Context Group**
    - Nomenclature similarities and confusions
    - Domains in libfabrics, or DMAPP refer to something different
  - **void** shmem_domain_create(**int** thread_level, **int** num_domains, **shmem_domain_t** domain_hndls[])
    - thread_level is the only possible grouping being discussed for this current proposal

High-level Usage Scenarios

- **Threads are not bound to Domains or Contexts**
- **Any thread can create a Domain or Context**
- **If Domain, and Context handles are visible – Any thread can access as per thread level**
- **Group of Contexts with similar thread level forms a Domain**

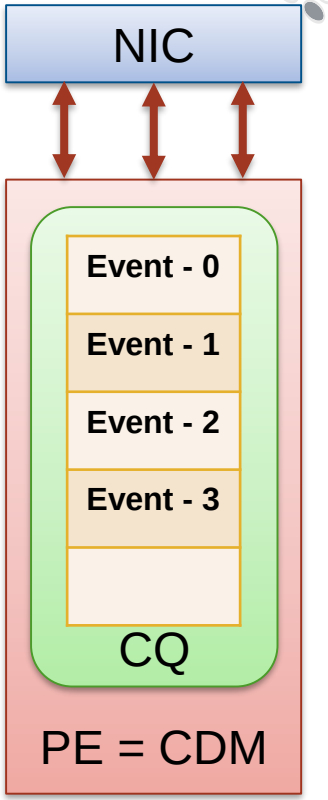
Example Usage Scenarios

- **Thread-1 Creates Context-1 with Thread level as SHMEM_THREAD_SINGLE**
  - Resource is not locked – Users are responsible for the correct usage
  - Only Thread-1 uses Context-1 throughout the lifetime of Context-1
  - Any Thread, but only one Thread at a time uses Context-1
- **Thread-1 Creates Context-1 with Thread level as SHMEM_THREAD_MULTIPLE**
  - Resources are locked
  - Any thread can make use of this Context-1 at the same time

Single-thread Design Overview

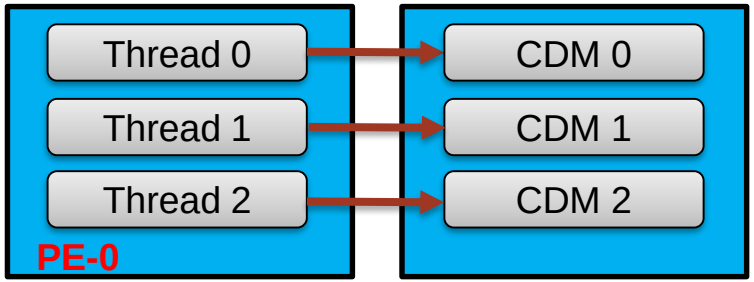
Communication Establishment Steps on Each PE

1. **Create Communication Domain(CDM)**
2. **Map PE as logical end-points to CDM**
3. **Attach CDM to NIC device**
   - Hardware limits on the number of available NIC devices per node
   - Aries Interconnect the limit is 120
   - Split equally among PEs in that Node
   - Unique CDM per PE
4. **Create Completion Queue(CQ) per CDM**
   - No shared CQs feature – Each CDM has just 1 CQ
5. **FMA- or BTE-based communication between end-point**
   - Events = Put/Get/AMO
   - FMA – Small Messages, BTE – Large Messages
   - ***shmem_quiet()*** is on all pending events per CQ
6. **CDM – Message Injection, CQ – Remote Completion Tracking**

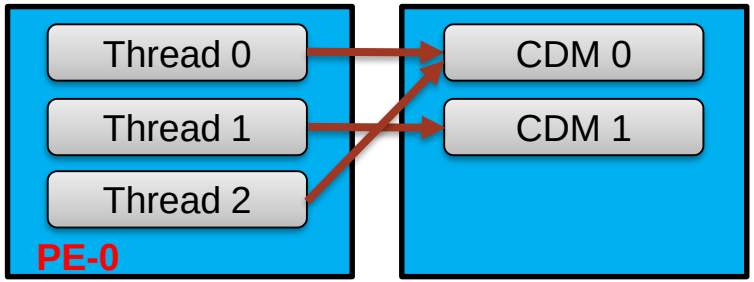


Generic Network Mapping

- **Single Threaded Model – SHMEM_THREAD_SINGLE**
  - 1CDM per PE – No Locks
- **Thread-safe Multi Threaded Model – SHMEM_THREAD_MULTIPLE**
  - Map CDM to Threads Directly – With Locks
- **Current Scenarios:**
  - Broadwell 36 cores per Node – Aries Network Resources 120 per Node
    - There should be No Locks for this scenario
  - Future Architectures 250+ cores per Node – Aries Network Resources 120 per Node
    - There should definitely be Locks for this scenario



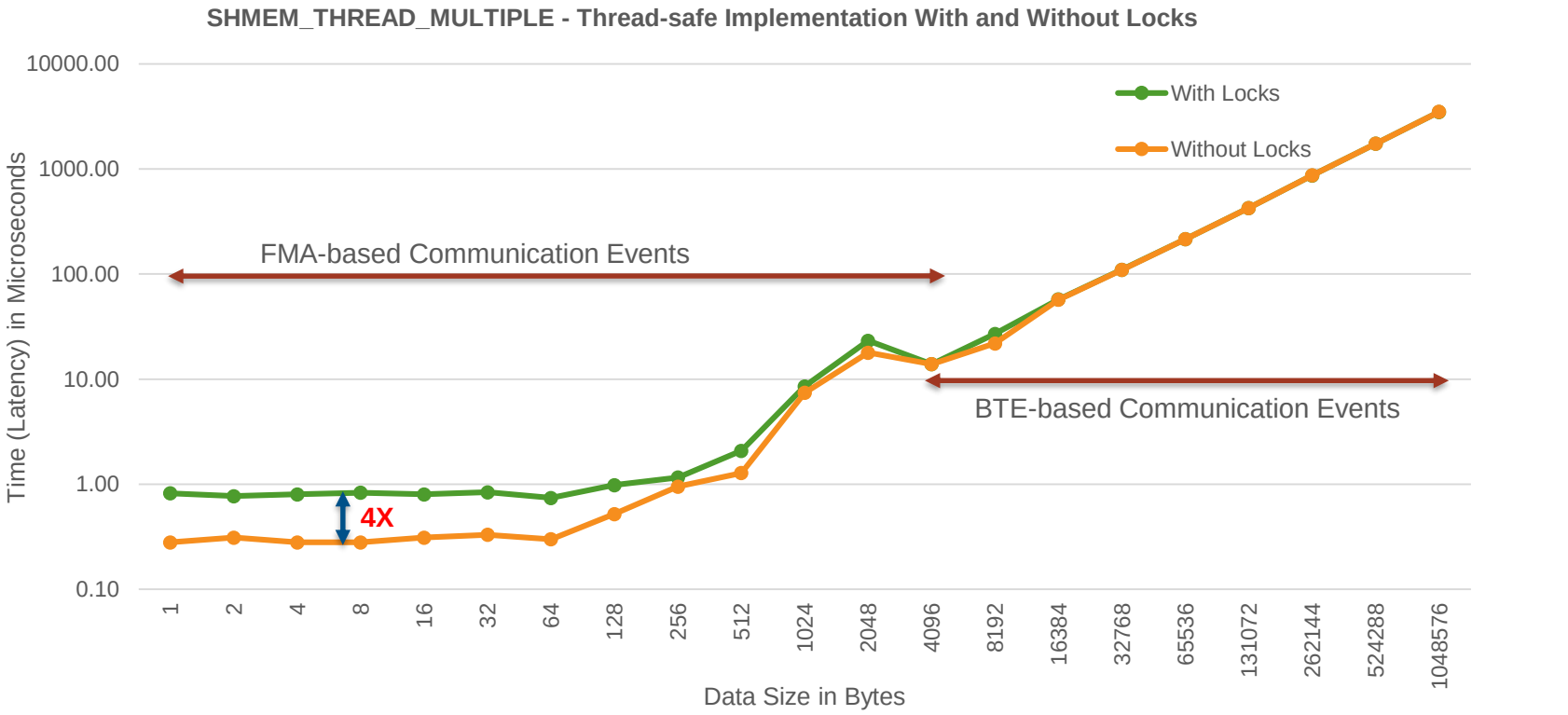
No Lock – Sufficient CDMs



With Locks – Insufficient CDMs

Lock vs. No Lock Performance Comparison

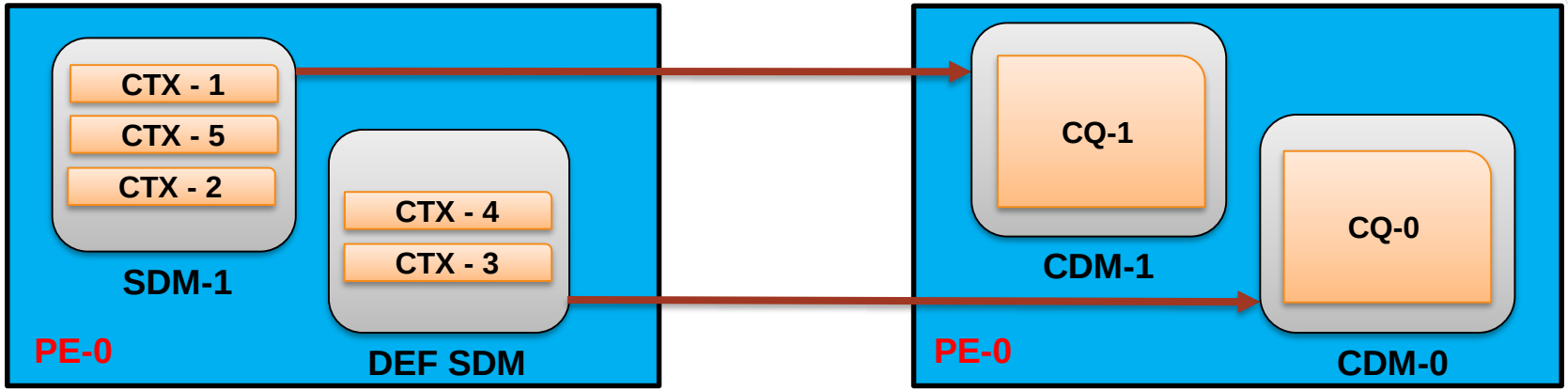
- Modified OSU PUT Microbenchmark – Broadwell 36 Cores, 2 PEs, 1 PE per Node, & 36 threads per PE



Context-Domain Prototype(s) in Cray SHMEM

- **Multiple Ways to map Network Resources(CDM, CQ) to Context-Domain(CTX, SDM)**
- **Design-0 – Explicit Event Tracking**
  - Basic Design with Explicit Event Tracking
  - Performance Comparison with Implicit vs. Explicit Events
- **Design-1 – Domain-based Mapping**
  - Domain-based Mapping
  - Handle Events Remote Completion tracking Internally per Domain
  - Limitations of this model in DMAPP
  - Not sure about other Implementations
- **Design-2 – Context-based Mapping**
  - Context-based Mapping
  - Work-in-progress

Explicit Event Tracking(Design-0)



SHMEM level logical end points

DMAPP level end points

- Each SHMEM-Domain has its own CDM
- Every Context in that Domain share a single CQ mapped to that CDM

Design Problem:

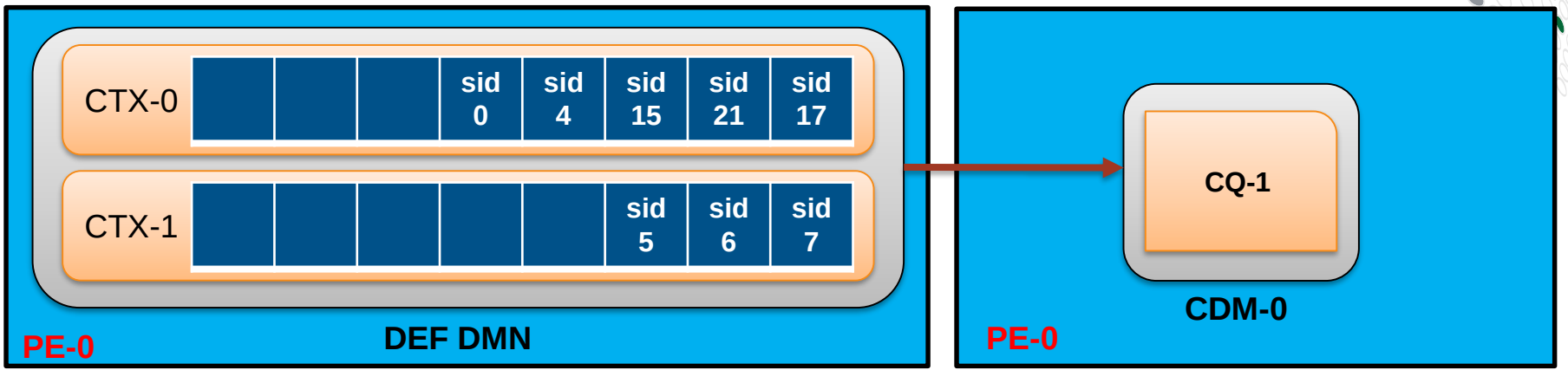
- shmem_ctx_quiet() is not possible - CQ tracks all events from all Contexts in that Domain

Explicit vs. Implicit Communication Events

- **Communication Events = PUT, GET, and AMO**
- **Types of events based on Data Sizes – FMA, and BTE**
- **Further Classification based on sync_ID – Explicit, and Implicit**

- **sync_ID – Handle to track the Remote Completion of Particular Event**
- **Explicit events**
  - Every event returns sync_ID
  - Can perform quiet/fence operation on a particular sync_ID
- **Implicit events**
  - Don't return any sync_ID
  - Optimizations(like event chaining) on FMA-based communications
  - Performs better than explicit events on FMA-based communications
  - No performance difference on BTE-based communications

Design for tracking Context-based events



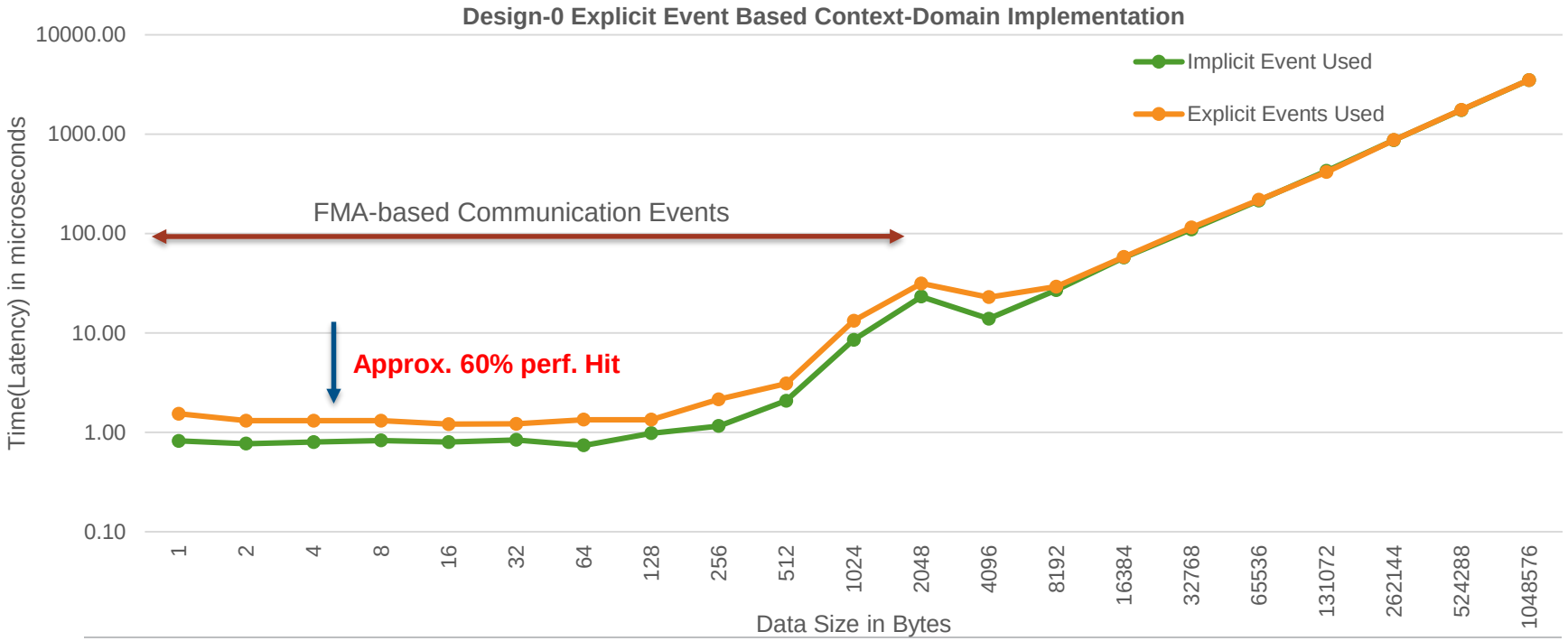
- Implement all Context-based events as Explicit events
- Track queue of sync_ID for each context in SHMEM-level
- Still all Contexts per Domain use a single CQ in DMAPP-level

Design Problem:

- Lose all Performance optimizations on Implicit-FMA-based events

Performance Analysis – Implicit vs. Explicit

- Modified OSU PUT Microbenchmark – Broadwell 36 Cores
- 2 PEs, 1 PE per Node, & 36 Domains with 1 Context per Domain in a PE
- All Domains are with thread level SHMEM_THREAD_MULTIPLE



Explicit Event Tracking(Design-0) Summary

- **Context-based quiet needs conversion of all Context-events into Explicit events**
- **Performance penalty (about 60%) in this design**
- **Not an ideal solution to use explicit events**
- **Problem: Support for multiple CQs per CDM**
  - In DMAPP – Performance is Optimized, but is this a design specific to DMAPP?
  - How does UCX, or Libfabrics track Completion?
  - Support for Multiple Remote Completion Tracking per Injection Point?

- **Intermediate Fix for DMAPP**
  - Not a blocker for Domain-based Mapping(Design-1)
  - Design-1 uses the similar resource mapping
  - Divert all FMA events from different Contexts into a default CQ Bank as Implicit Events
  - Track only BTE events separately with sync_ID

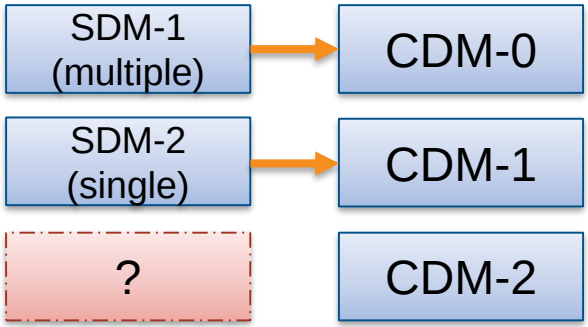
Domain-based Mapping(Design-1)

- **Derivative of Design-0, similar Resource Mapping**
  - CDMs mapped directly to SHMEM-Domains
  - Handled the Explicit event performance penalty
  - Divert all events from different Contexts into a default CQ as Implicit Events
- **Major Improvisations from Design-0**
  - Use thread_level argument from shmem_domain_create as hints to efficiently share resources
  - SHMEM_THREAD_MULTIPLE domains – share resources – with locks
  - SHMEM_THREAD_SINGLE domains – try to have unique resources – No locks

Design Problem:

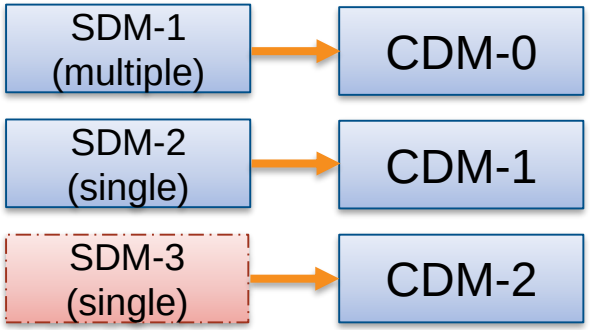
- **shmem_domain_create – dynamic, no limit on the max number of domain create calls in the application**
- **Can't fix on the optimized resource allocation without the complete picture of the application**

Domain Resource Mapping Scenarios



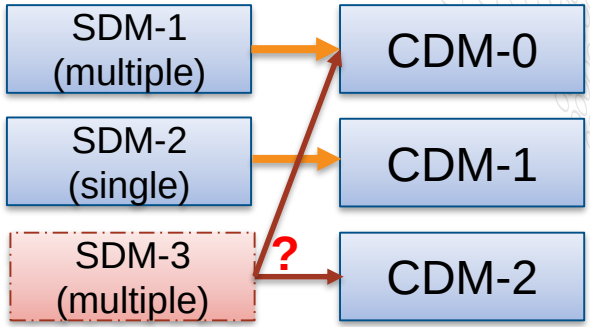
Example:

- There are 3 CDMs available
- Only 3 Domains(SDM) will be created
- SDM-1 created with thread level MULTIPLE
- SDM-2 created with thread level SINGLE
- Implementation has no clue on the next Domain level



Scenario:1

- SDM-3 is created with thread level SINGLE
- SDM-3 has its own unshared CDM



Scenario:2

- SDM-3 is created with thread level MULTIPLE
- Conservative Design:

Share CDM with previous MULTIPLE

- Efficient Design:

Assign SDM-3 with its own unshared CDM

Possible Hints for the Domain Configuration

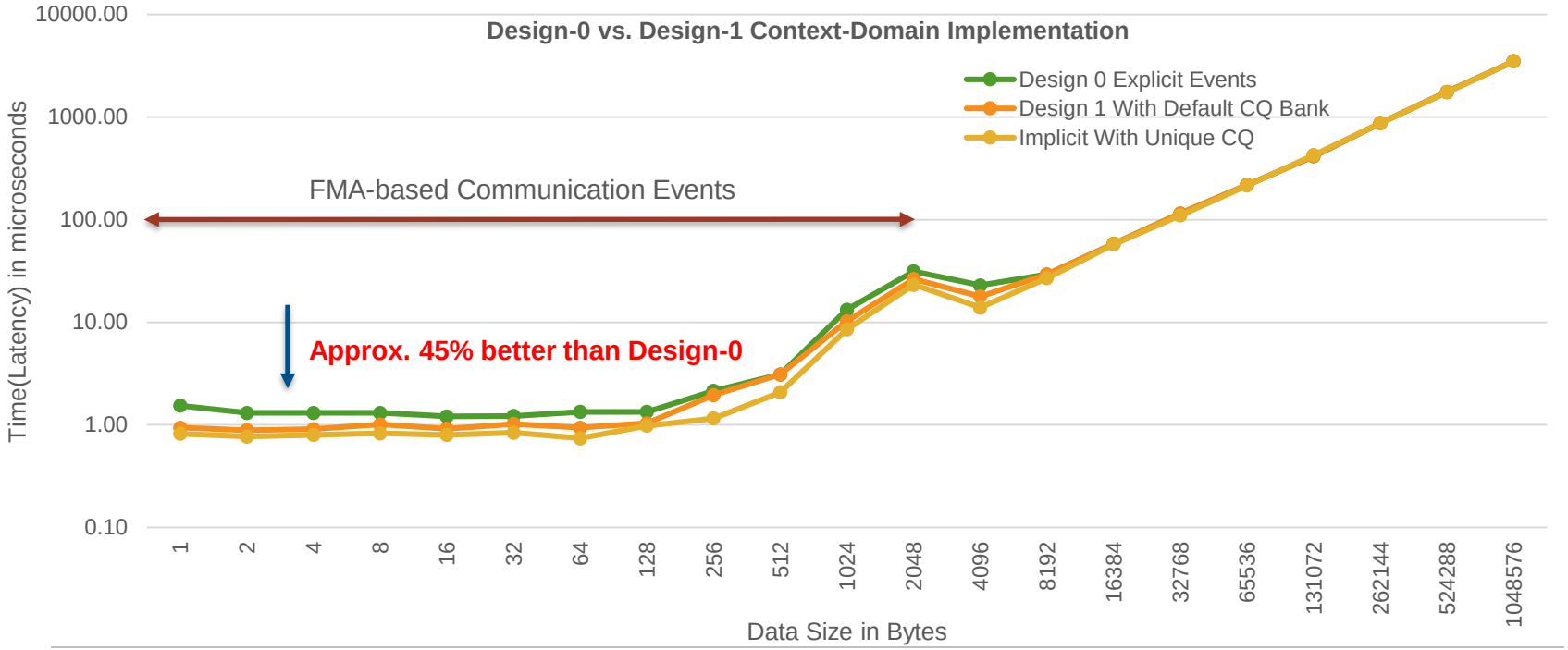
```
void shmem_domain_config(shmem_domain_type *thread_level_list, int num_thread_level, int state)

struct shmem_domain_type {
        shmem_thread_level thread_level;
        int                num_domains };
enum shmem_thread_level {
        SHMEM_THREAD_SINGLE,
        SHMEM_THREAD_MULTIPLE };
```

- Optional Routine – Just to provide hints to the Implementation
- Hints can help on fixing the most optimized resource mapping
- Assume that applications will follow the hint, not a strict rule
- Can use **shmem_domain_config** any number of times, and change hints dynamically. But, destroy all resources before reconfiguration
- **state –** Argument decides whether the call is global or local(Needs separate discussion)

PS: Proposed based on initial study, not a final proposal. Needs further discussion on the Domain-configuration. Do we need domains different thread levels, or can we just use context-group with the locking mechanism as a level?

Performance Analysis – Design-1

- Modified OSU PUT Microbenchmark – Broadwell 36 Cores
- 2 PEs, 1 PE per Node, & 36 Domains with 1 Context per Domain in a PE
- No Hints needed for this micro-benchmark



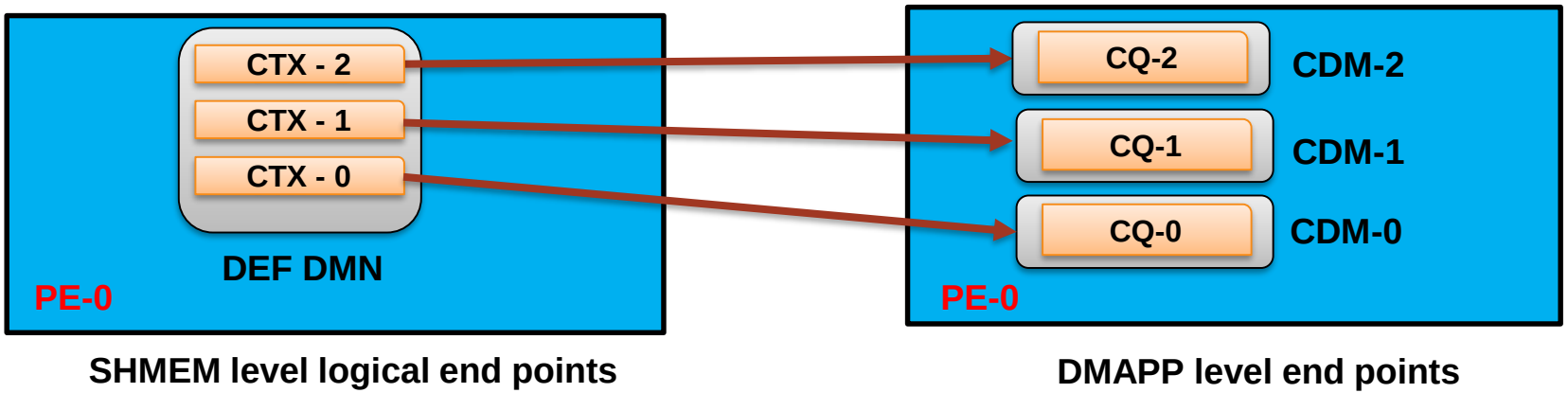
Domain-based Mapping(Design-1) Summary

- **Possible to avoid Explicit Events and Map Resources Directly to Domains**
- **Performance Better than Explicit Event Tracking(Design-0)**
- **Overlapping completion per Context within a thread - Not efficient for FMA-events**
- **Efficient Resource Mapping Needs Hints from Users**
  - Number of Domains to be Created with threading level
  - If usage matches hints, over-allocation of resources can be handled efficiently

- **Unanswered Questions**
  - What is the Correct level of abstractions for the users ?
  - Are these hints feasible from applications ?
  - Will these hints be useful in other implementations ?
  - Is this the best Design for Context-Domain Implementation?
    - Intermediate design, till hardware or communication layer supports multiple CQ per CDM
    - Still working on the performance enhancements

Context-based Mapping(Design-2) – WIP

- **Design-0, and Design-1**
  - Maps resources(CDMs) directly to Domains and Contexts share CQs
- **Design-2 - WIP**
  - Map CDMs directly to Contexts, each Contexts will have separate CQs
  - Group similar Contexts with same thread-levels as Domain(Only 2-domains will be necessary)
  - No interference between threads (no locks/syncs, completely independent) – good performance
  - Even within a thread, possible better overlapping completion per Context than Design-1



Conclusion

- **Context-Domain Prototype Design in Cray SHMEM is a WIP**
- **Important Problems that Context-Domain Features target:**
  - Splits up and separates Message Injection and Remote completion tracking
  - Solve the Limitation on the Number of Resources per Node by better Resource Mapping
- **Multiple Designs based on different Resource Mapping**
  - No Explicit Event Tracking Design for Implementing Context-Domain Features
  - Design-1: Map SHMEM-Domains Directly to Network Resources
    - Performance hit on the overlapping completion per Context within a Thread
    - Need for Multiple remote completion tracking within the same Injection points
    - Need for hints from applications about the number of Domains
  - Design-2: Map SHMEM-Contexts Directly to Network Resources
    - Avoids interference between threads(almost similar to Design-1)
    - Better Overlapping completion per context, even within a thread
- **Work on applications, and usage scenarios before settling on the final API**
- **Correct level of abstraction to the users**

Backup Slides

Sample Context Creation and Usage

```
NUMBER_OF_THREADS = 3 + master_thread
int main(void) {
        int i;
        shmem_int_thread (SHMEM_THREAD_MULTIPLE);
        shmem_ctx_t *ctx;
        ctx = (shmem_ctx_t *) malloc(sizeof(shmem_ctx_t)*4);
        shmem_ctx_create(SHMEM_DEFAULT_DOMAIN, ctx[0]);
        shmem_ctx_create(SHMEM_DEFAULT_DOMAIN, ctx[1]);
        shmem_ctx_create(SHMEM_DEFAULT_DOMAIN, ctx[2]);
        shmem_ctx_create(SHMEM_DEFAULT_DOMAIN, ctx[3]);
```

- Contexts created on the default domain
- Contexts created by master thread and the context objects made visible for other threads to use
- Number of contexts created based on the number of threads being used inside the parallel region

```
        #pragma omp parallel for shared(i)
        {
                int tid = get_omp_thread_num();
                for (i = 0; i < 4; i++) {
                        shmem_ctx_put(ctx[tid], ….);
                }
        }
```

- Each thread uses a unique contexts objects
- Though the contexts are created by the master thread, other threads are allowed to use this
- The order in which the context objects are accessed by the thread might ne modified in the next parallel region