OPENFABRICS ALLIANCE

PSM Tag Matching API

Author: Todd Rimmer
Date: April 2011

Agenda

- What is PSM?
- How does PSM differ from Verbs?
- PSM Advanced Features
- PSM Performance/Scalability

What is PSM?

- PSM=Performance Scaled Messaging
- API designed specifically for HPC
  - Analyzed needs of various MPI Channel Interfaces
  - Designed an API to perfectly match the needs of those Interfaces
- Carefully selected division of responsibility
  - MPI handles higher level capabilities
    - The wide array of MPI functions, etc
  - PSM focused on interconnect specific details
    - Data movement strategies and tuning
    - Advanced features – QoS, dispersive routing, resiliency, etc
- Implemented as a user space library

PSM is now included in OFED as of 1.5.2

A Bit of InfiniBand History

OPENFABRICS ALLIANCE

Early 2000's

- InfiniBand Inception
  - Designed for the enterprise data center market and an IO paradigm
  - Backbone network as a replacement for Ethernet and Fibre Channel
  - Incorporate best data center features of all interconnects and protocols
  - Cluster sizes: 100s of nodes
  - Performance Req: 1M IOPs

IB Verbs Design Point

Mid-2000's

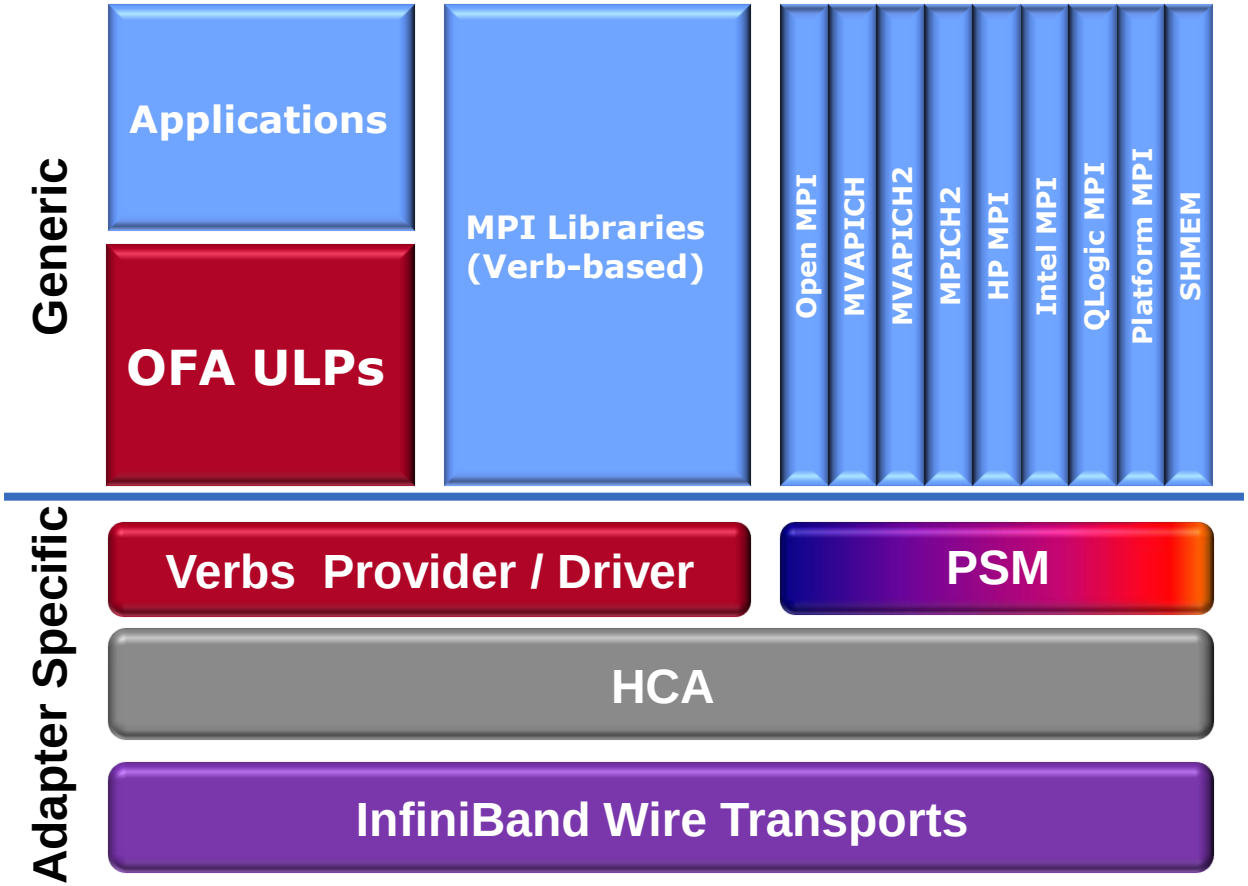
- InfiniBand Finds Its Niche
  - High Performance Computing Clusters market
  - Low-Latency / High Bandwidth advantages
  - Primary message paradigm – MPI
  - Cluster sizes: 1000s of nodes
  - Performance Req: tens of millions of messages per second

PSM Design Point

PSM Design Focus

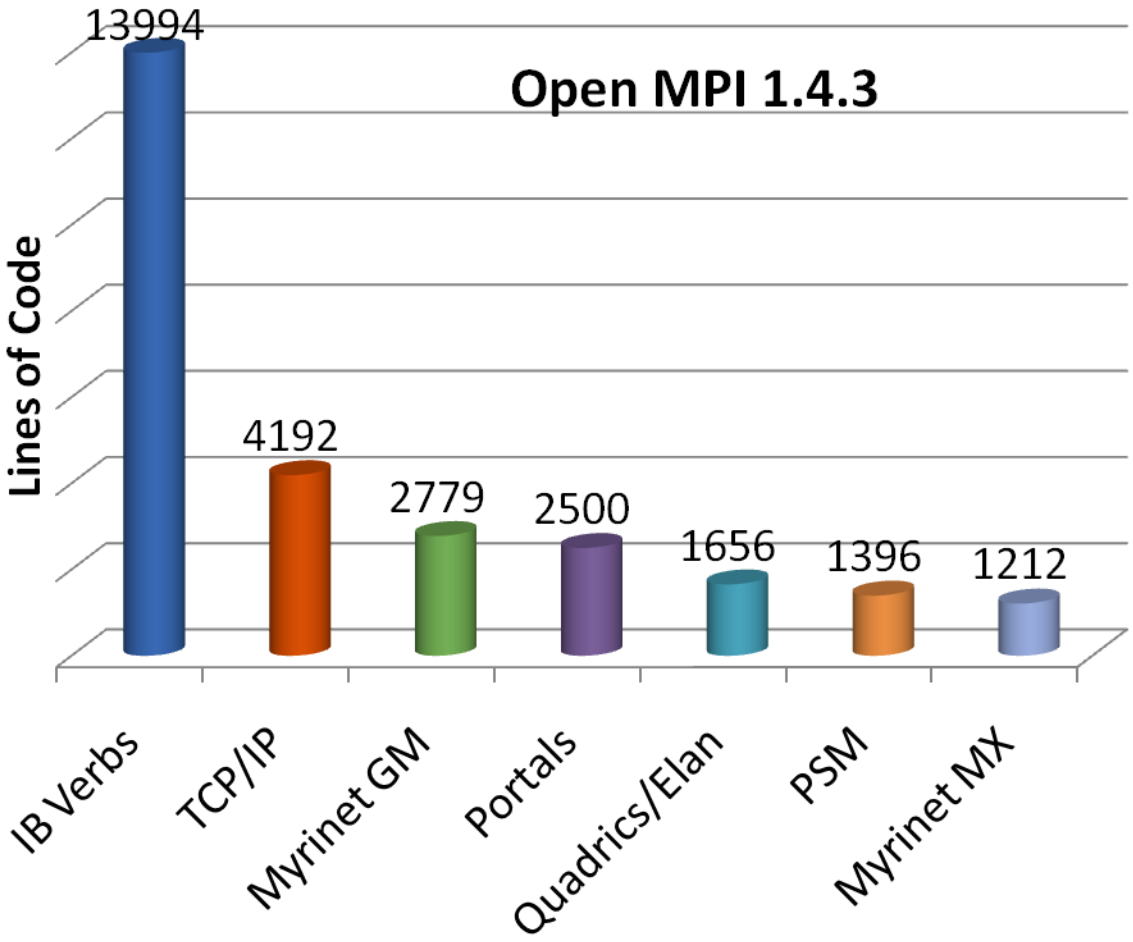
- **Focus on the needs of MPI and HPC Compute**
- **Design for very high HPC messaging rate, scalable latency**
  - Allow full support of all MPI collective implementations
- **Maintain a minimal memory footprint**
  - No adapter state per connection, no caches in adapter
  - Minimal memory footprint per end point
  - Scale out to large job size
- **Provide the high degree of resiliency needed by HPC**
  - More sophisticated retry/timeout mechanisms
  - More persistence
  - (IBTA Verbs limited to 7 retries at fixed timeout interval)
- **Overcome weaknesses in IO oriented IB transport**
  - Out of order packet handling, dispersive send of individual messages, etc
  - While interoperating with standard IB link and network layers
- **Centralize implementation of sophisticated features**
  - Allow all MPIs to easily take advantage

Integration of PSM with Open Fabrics



Comparison of Impedance Match OpenMPI MTL and BTL sizes

- Interfacing to PSM is comparable to HPC focused interconnects
  - Such as Quadrics, Myrinet
- Relative sizes are similar for other MPIs
  - mvapich, mvapich2, etc



Example of using PSM MPI Receive in OpenMPI 1.4.3

```
int ompi_mtl_psm_irecv(…)
{
    …
    mca_mtl_psm_request_t * mtl_psm_request = (mca_mtl_psm_request_t*) mtl_request;

    ret = ompi_mtl_datatype_recv_buf(convertor, &mtl_psm_request->buf, &length,
                                                &mtl_psm_request->free_after);
    if (OMPI_SUCCESS != ret) return ret;

    mtl_psm_request->length = length;
    mtl_psm_request->convertor = convertor;
    mtl_psm_request->type = OMPI_MTL_PSM_IRECV;

    PSM_MAKE_TAGSEL(src, tag, comm->c_contextid, mqtag, tagsel);

    err = psm_mq_irecv(ompi_mtl_psm.mq, mqtag,   tagsel,  0,  mtl_psm_request->buf,    length,
                                mtl_psm_request,  &mtl_psm_request->psm_request);
    if (err)
        return OMPI_ERROR;
    return OMPI_SUCCESS;
}
```

Example of using PSM MPI Send in OpenMPI 1.4.3

```
int ompi_mtl_psm_isend(…)
{
   …
   mca_mtl_psm_request_t * mtl_psm_request = (mca_mtl_psm_request_t*) mtl_request;
   …
   mqtag = PSM_MAKE_MQTAG(comm->c_contextid, comm->c_my_rank, tag);

   ret = ompi_mtl_datatype_pack(convertor,  &mtl_psm_request->buf,   &length,
                                            &mtl_psm_request->free_after);
   if (OMPI_SUCCESS != ret) return ret;

   mtl_psm_request->length= length;
   mtl_psm_request->convertor = convertor;
   mtl_psm_request->type = OMPI_MTL_PSM_ISEND;

   if (mode == MCA_PML_BASE_SEND_SYNCHRONOUS)
      flags |= PSM_MQ_FLAG_SENDSYNC;

   psm_error = psm_mq_isend(ompi_mtl_psm.mq,   psm_endpoint->peer_addr,   flags,  mqtag,
                       mtl_psm_request->buf,  length,  mtl_psm_request,  &mtl_psm_request->psm_request);

   return psm_error == PSM_OK ? OMPI_SUCCESS : OMPI_ERROR;
}
```

Features Handled Inside PSM

- MPI tag matching
- Per message selection of data movement approach
  - eager, rendezvous, etc
- Multi-Rail
- Dispersive routing
  - Concurrent transmission of large message segments on different paths
  - Rotation of path usage to balance traffic patterns in fabric core
- Scalable resiliency algorithms
  - Flow control
  - Retry/timeout
  - Degree of persistence in face of congestion or fabric instabilities
  - Tolerance for OS jitter
- Memory locking

Features Handled Inside PSM

- GPU integration
- End point establishment
- Path resolution
  - LID exchange for simple fabrics
  - QoS and partitioning interaction for clusters with Virtual Fabrics
  - Scalable SA PathRecord query for complex fabrics (Torus, Mesh, etc)
- QoS
  - QoS level per job
  - Multiple QoS level in same job
    - Separate QoS levels for control messages vs bulk data transfers
- Optional CPU affinity
- HPC Oriented Interconnect statistics and debug hooks

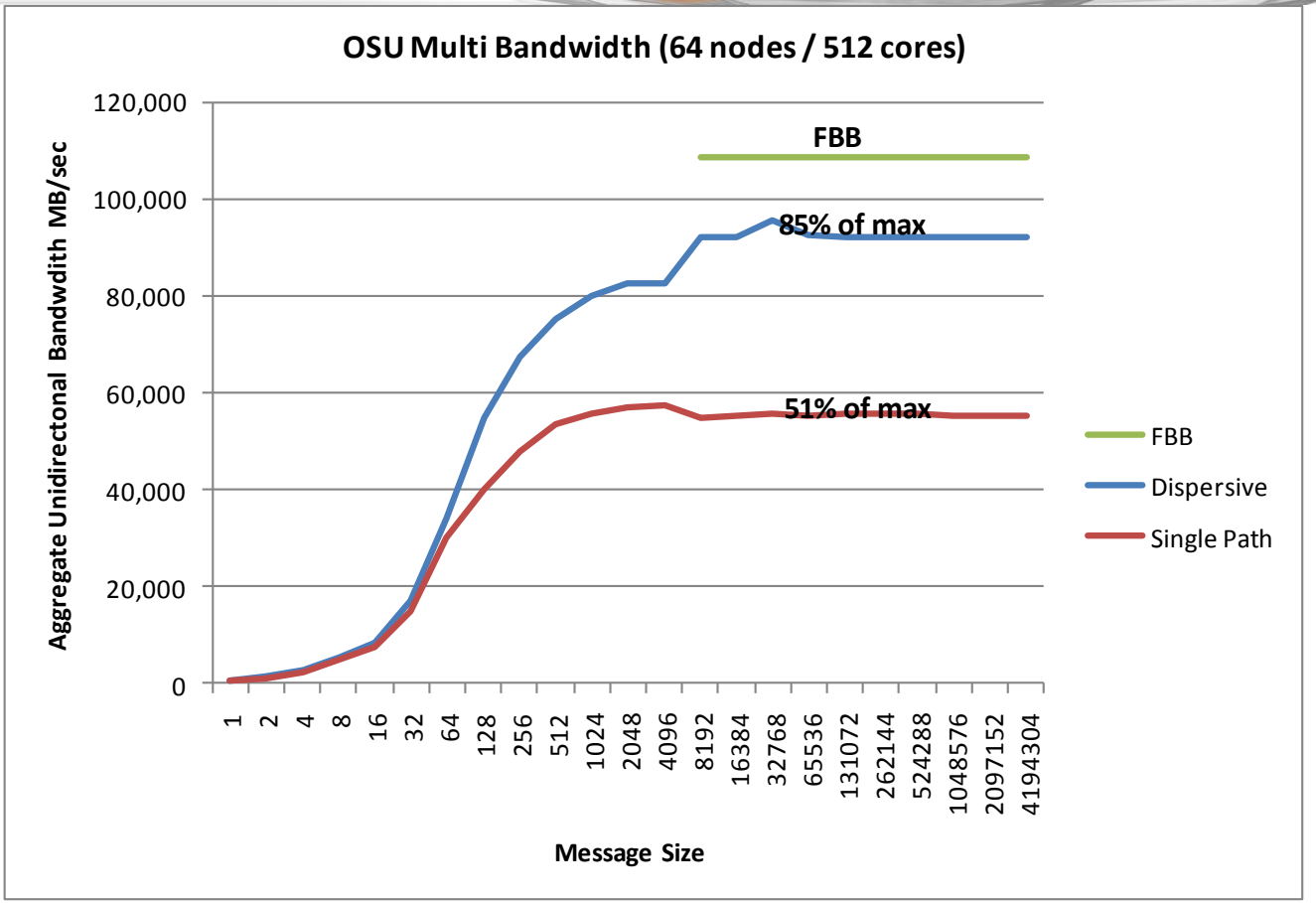
Localization of features makes them instantly available for all MPIs

Advanced PSM Feature Example: Dispersive Routing

- Multiple routes exist between pairs of sources and destinations
  - LMC enabled
- Choice of route is source-directed
  - Short and control messages
    - Follow a single path to maintain ordering
  - Long messages
    - Rendezvous transfers, once set up, can be un-ordered with respect to each other
    - Round-robin path selection
- Dispersive routing load balances traffic across these paths to reduce hotspots and congestion
  - Improved performance
  - More deterministic/reproducible performance

PSM Dispersive Routing

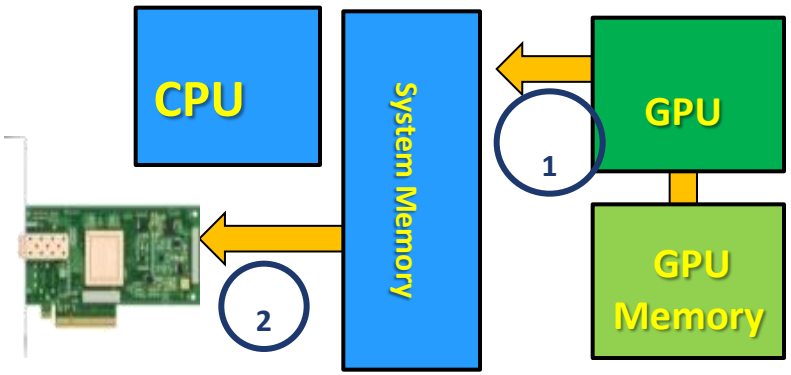
- Traditional InfiniBand fabrics use static routes thru the fabric based on the destination LID
- Deterministic oblivious routing does not reach full bisectional bandwidth for random messaging patterns
- Fat-Tree networks deliver <60% of peak



Dispersive Routing Delivers Higher Bandwidth

Advanced PSM Feature Example: GPUDirect Integration

- GPUDirect
  - Optimizes memory access and transfers for communication between GPU nodes across InfiniBand
  - Provides for faster communications across a GPU based cluster

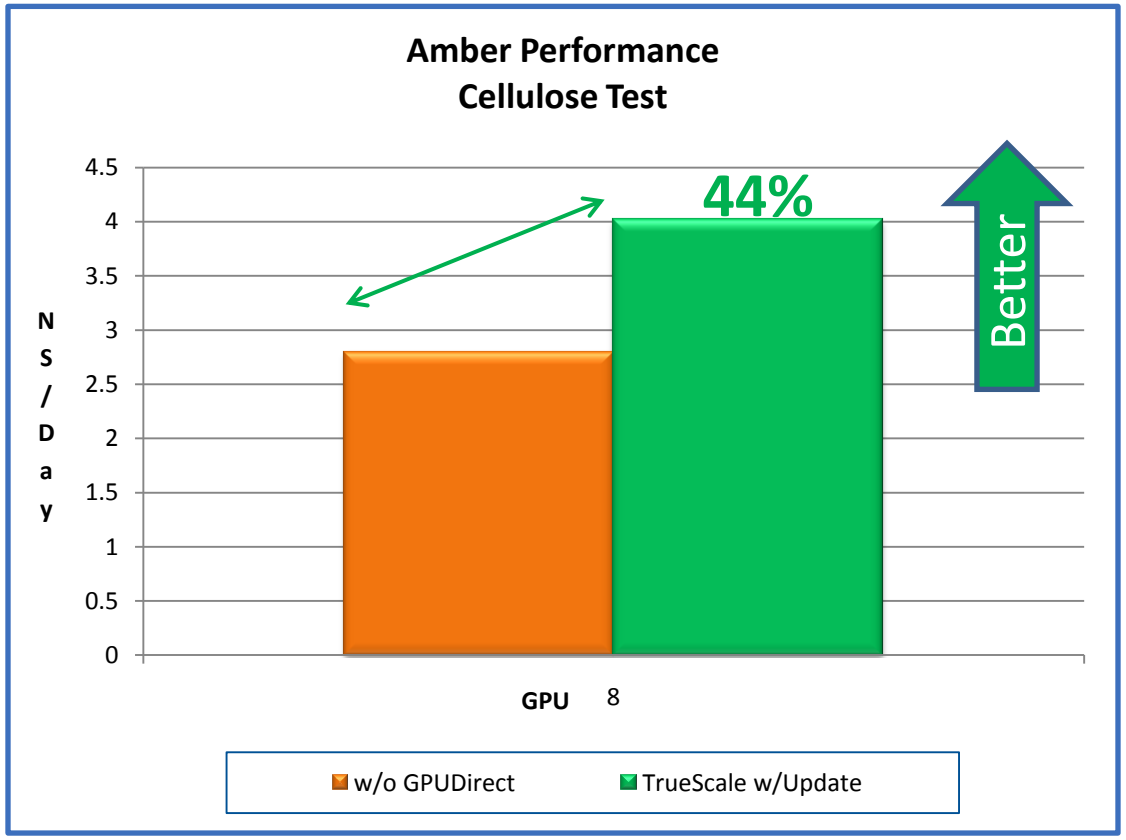


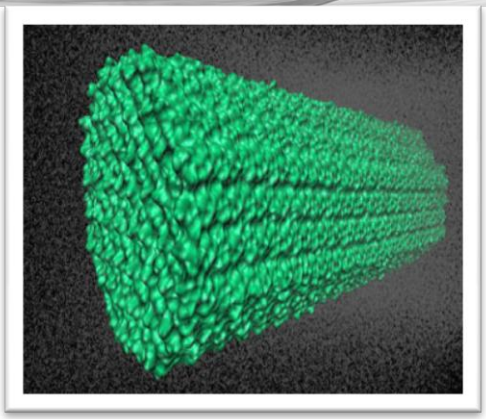
PSM Solution

- **No memory region conflicts**
- **No impact on GPGPU performance**
- **Maintains latency and message rate performance**

44% Performance improvement with PSM GPUDirect

Performance with and without GPUDirect





Test Configuration

- 8 GPU Cluster
- Tesla M2050 GPU (2/Server)
- Intel Servers –
  - -- Dual X5570 2.93 GHz
  - -- 24GB Memory/Server

- **44% Performance improvement with PSM and GPUDirect**

NDC Based Results

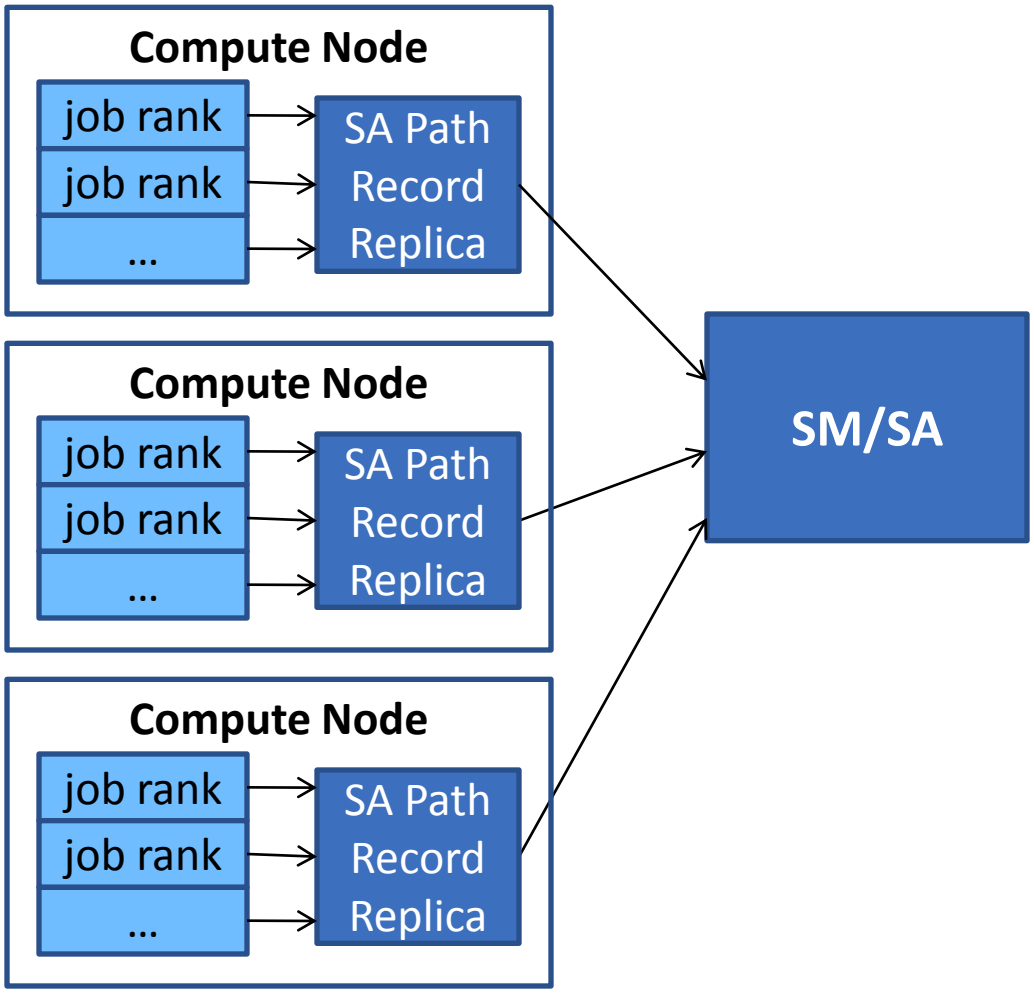
Advanced PSM Feature Example: Path Resolution

- For simple fabric designs (such as Fat Tree) LID exchange is sufficient
  - Basic QoS and Partitioning with a single PKey and SL for entire job
- However, more advanced fabrics need full Path Records
  - Mesh/Torus
  - SL can vary per PathRecord, fabric disruption handling can also alter SL
  - Fabrics using virtual Fabrics to separate various MPI jobs
  - Fabrics using virtual Fabrics to separate control vs data in a single MPI job
- PSM Hides this complexity from MPI
  - Provides simple parameters to enable/disable the capability
  - Jobs can be assigned ServiceIds to identify job in PathRecord queries

PSM Enables a Very Scalable Solution

Scalable Path Resolution

- Each node retains and synchronizes a PathRecord replica with the SM/SA
  - Automatic update on fabric change
- Replica persists beyond life of jobs
  - Shared by all ranks on node
- Replica allows >1 Million PathRecord query/sec per node
- Permits very rapid job startup and avoids SA being a bottleneck in large fabrics

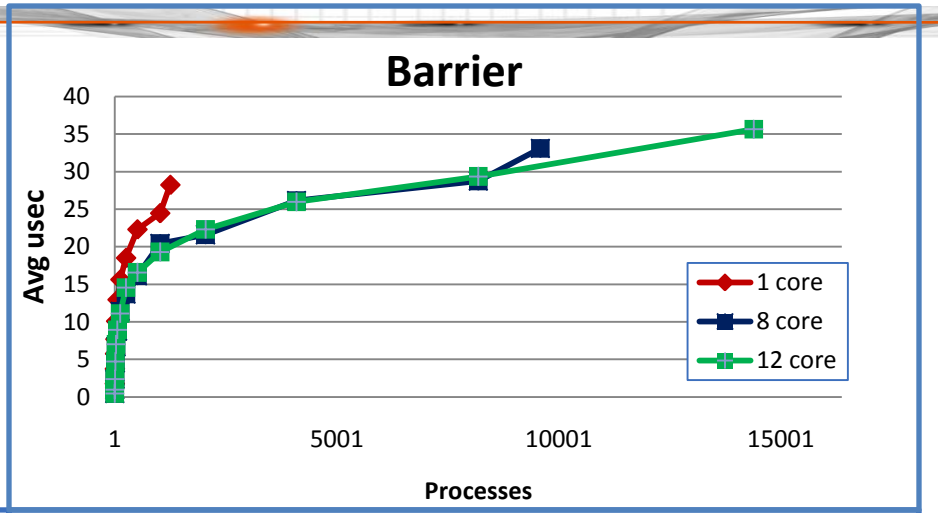


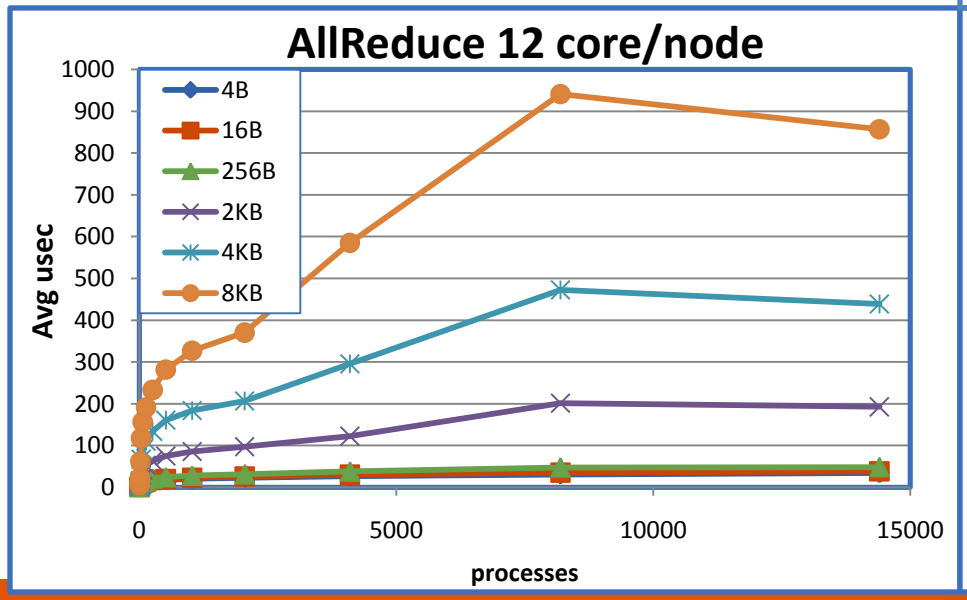
PSM Performance Focus

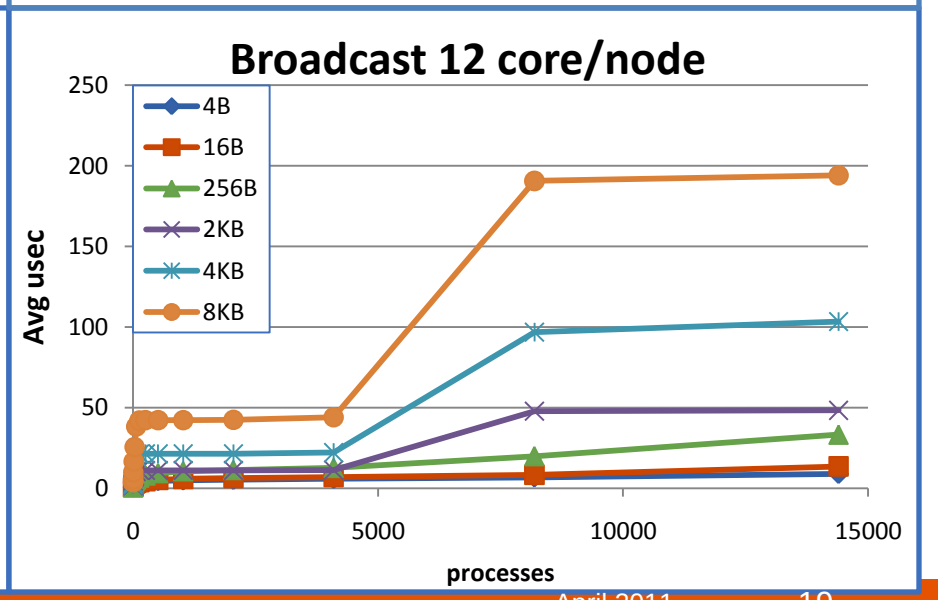
- MPI Collectives are Critical to Scalability and Performance of most applications
- At the Heart of Collectives is Scalable Latency and Messaging Rate
  - A critical design focus for PSM
- PSM's Performance allows MPI's Native Collectives to Scale and Perform
  - No need for hardware acceleration
- Full MPI Collective Performance is available for all Collectives, all MPIs, all Collective Algorithms

MPI Collectives Performance at Scale

- 1200+ node Westmere System
- LLNL Muir Cluster
- QLogic QDR HCA and Switches
- Open MPI with PSM MTL
- Default OpenMPI Collectives algorithms

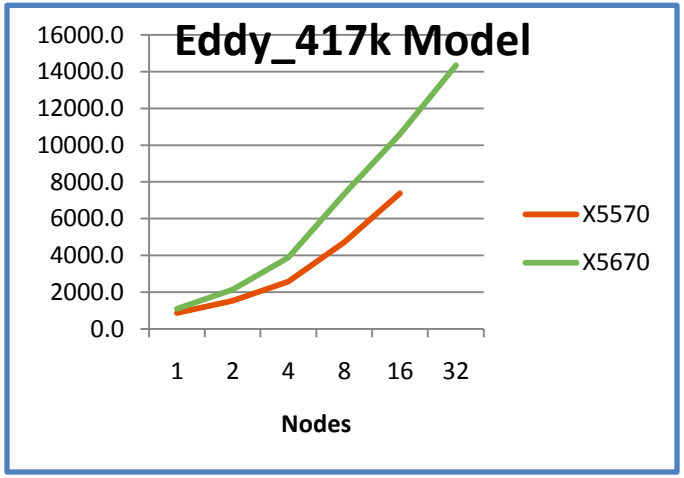


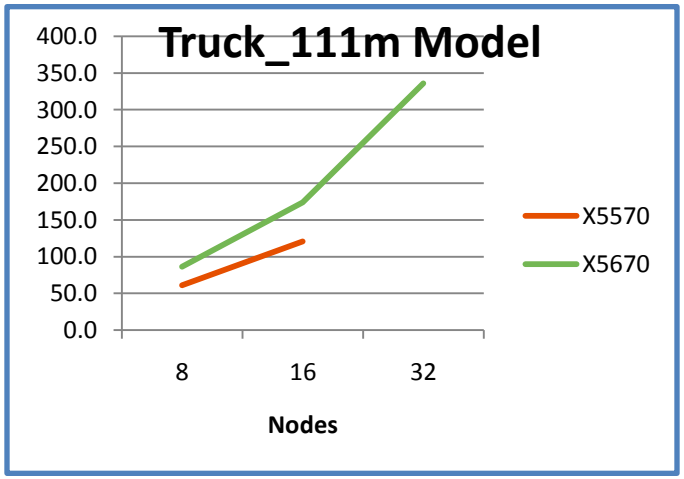




Highly Scalable Application Performance

- **Scalability and Collectives Performance results in application scalability**
- **PSM Solutions Scale with Node Count**
- **PSM Solutions Scale with Core Count**
- **PSM takes advantage of performance advances in CPUs and GPUs**





http://www.ansys.com/Support/Platform+Support/Benchmarks+Overview

Summary

- PSM is now included in OFED
- PSM is a highly optimized API Focused on the needs of MPI and HPC
- PSM provides the right abstraction layer
  - Relieves MPI from all the interconnect details
  - Allows many advanced interconnect specific optimizations in PSM

- PSM provides excellent scalability
- PSM provides excellent messaging rate, scalable latency
- PSM's inherent performance translates to excellent collectives performance

- PSM results in application scalability and performance for all MPIs

OPENFABRICS ALLIANCE

Thank You

Author: Todd Rimmer
Date: April 2011