NVIDIA

Improving QoS for OpenSHMEM using in-network Priority

UCF 2022 Annual Meeting and Workshop

Vishwanath(Vish) Venkatesan, Manjunath Gorentla Venkata

Motivation and Agenda

QoS for Small message Operations

Motivation

- Achieving good Quality of Service(QoS) for short messages - desirable characteristic for OpenSHMEM applications
- Congestion due to contention for network switches, I/O subsystems etc., - cause adverse user experience and impact system performance
- Problem can exacerbate with increasing scale in leadership scale systems
- Network congestion can typically lead to prolonged tail latency
- This work studies how to achieve shorter tail latencies for short messages
  - Features available in NVIDIA infiniband hardware specifically Virtual Lanes
  - Exposing QoS prioritization to users through newly proposed OpenSHMEM queues API

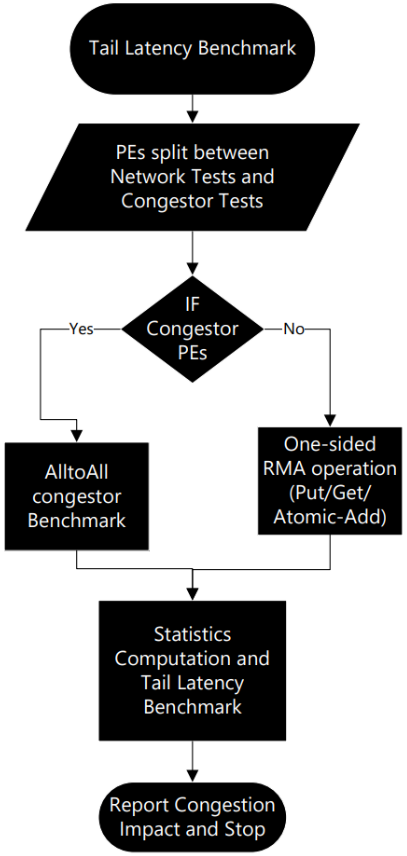
Agenda

- Introduce definition of tail latency and measurement strategy
- Design and Implementation for QoS based Interfaces in OpenSHMEM and UCX
- Experimental Evaluation
- Summary and Future work

Congestion and Tail Latency

Definition and Measurement Methodology

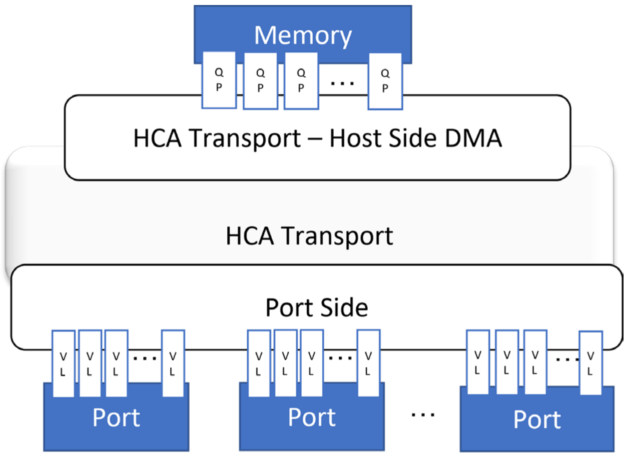
- Tail Latency
  - Small percentage of the transfer times in a system that take the longest in comparison to bulk of the transfers (typically p99)
  - P99 tail latency value marks that 99% samples are faster than or equal to the p99 value.
  - N/W congestion can be characterized with the help of tail latency
- Benchmark to measure tail latency
  - GPCNeT is a benchmark which measures congestion by running multiple types of communication patterns at the same time
  - Leverage essence of GPCNeT to create an OpenSHMEM benchmark to measure congestion
  - Create systematic congestion with communication traffic simultaneously
- Tail Latency benchmark Execution sequence
  - Divide SHMEM - PEs between isolated and congestor pattern tests. (1)
  - Measure isolated non-congestor communication latency and their tail latencies (2)
  - Start congestion traffic in PEs (3)
  - Measure congested latency and tail latency (4)
  - Repeat (2-4) for all isolated communication patterns based on user input

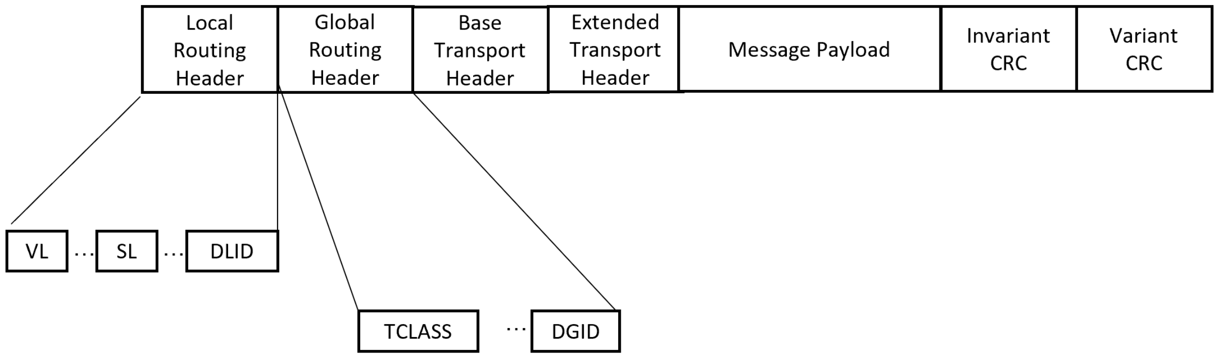


Infiniband Virtual and Service Lanes

Mitigation - Design (I)

- NVIDIA Infiniband architecture (IBA) includes Quality of Service and Congestion control features
  - All IBA packets contain a local route header (LRH) with SL information to forward packets within a subnet
  - All packets across subnets contain a global route header (GRH) which also carries SL information in the TCLASS field
  - IBA utilizes a memory-based user-level communication abstraction (Queue Pair) – a logical endpoint (host side)
  - Port side of channel adapter implements Virtual Lanes – enable multiple independent data flows sharing the same link
  - Service Lanes – permits a packet to operate at one of 16 SLs
  - IBA does not define a specific relationship between SL value and forwarding behavior
  - SL to VL mapping defined at the subnet layer allows QoS implementation on IBA





OpenSHMEM Queues

Mitigation - Design (II)

- Abstraction for communication aggregation, data aggregation and expressing QoS with ability to segregate traffic
- OpenSHMEM queue is an opaque object - users can interact with the queue through OpenSHMEM queue interfaces only.
- Aggregation
  - Communication queues are used for aggregating communication operations
  - Data queues enable aggregating of data before sending the data to the destination PEs
  - Aggregation should be agnostic of the underlying mechanisms
  - Aggregation should be agnostic of thread posting the operation
- QoS with Queues
  - Express QoS on the ability to segregate traffic
  - **Priority queues mapped to separate Service Lanes (SL) for short messages**
- Detailed API proposal of OpenSHMEM queues for the OpenSHMEM specification available in [1]

[1] OpenSHMEM Queues Architecture document: OpenSHMEM Queues for Aggregation · Issue #483 · openshmem-org/specification

https://github.com/manjugv/osh_public_docs/blob/master/gorentla_osh_queues_v0.1

OpenSHMEM Queues – QoS/Communication queue APIs

Mitigation - Design (III)

```
/** shmem_queue.h */
...SNIP...
typedef enum {
        SHMEM_QUEUE_COMM = 0,
        SHMEM_QUEUE_DATA = 1,
        SHMEM_QUEUE_QOS = 2
} shmem_queue_type_t;

typedef enum {
        SHMEM_QOS_HIGH_PRI = 0, /** Highest possible QoS on the system*/
        SHMEM_QOS_MEDIUM_PRI = 1, /** Best effort QoS - pick the best */
        SHMEM_QOS_LOW_PRI = 2, /** Lowest possible priority for messages */
} shmem_qos_class_t;

typedef struct shmem_queue_config {
        shmem_queue_type_t qtype;
        shmem_queue_sharing_t sharing_model;
        /* Exhaustive optional fields */
        uint64_t max_elems; /* Applicable for communication queues*/
        uint64_t max_bytes; /** Applicable for data queues */
        shmem_qos_class_t qos_class; /** Qos class to use */
        double timeout_flush;
        shmem_op_type_t op_type;
        shmem_pe_type_t pe_type;
} shmem_queue_config_t;
```

```
/** OpenSHMEM communication queue APIs */
int shmem_queue_comm_create(shmem_queue_t *queue,
                            shmem_queue_config_t *qconfig);
int shmem_queue_comm_destroy(shmem_queue_t queue);
int shmem_queue_TYPE_comm_push(shmem_queue_t queue, TYPE *dest, TYPE *src,
                               size_t nelems, int pe, uint64_t op);
int shmem_queue_progress(shmem_queue_t queue);
int shmem_queue_global_flush(shmem_queue_t queue);
```

- APIs for QoS priority queues and Communication queues
- Depending on the *shmem_queue_type_t* the appropriate queue is selected
- Communication queues enable aggregation for the communication operations such as (shmem_put_nbi, shmem_get_nbi and atomic operations) – collectives are not supported with this mechanism

[1] OpenSHMEM Queues Architecture document: https://github.com/manjugv/osh_public_docs/blob/master/gorentla_osh_queues_v0.1

Implementation

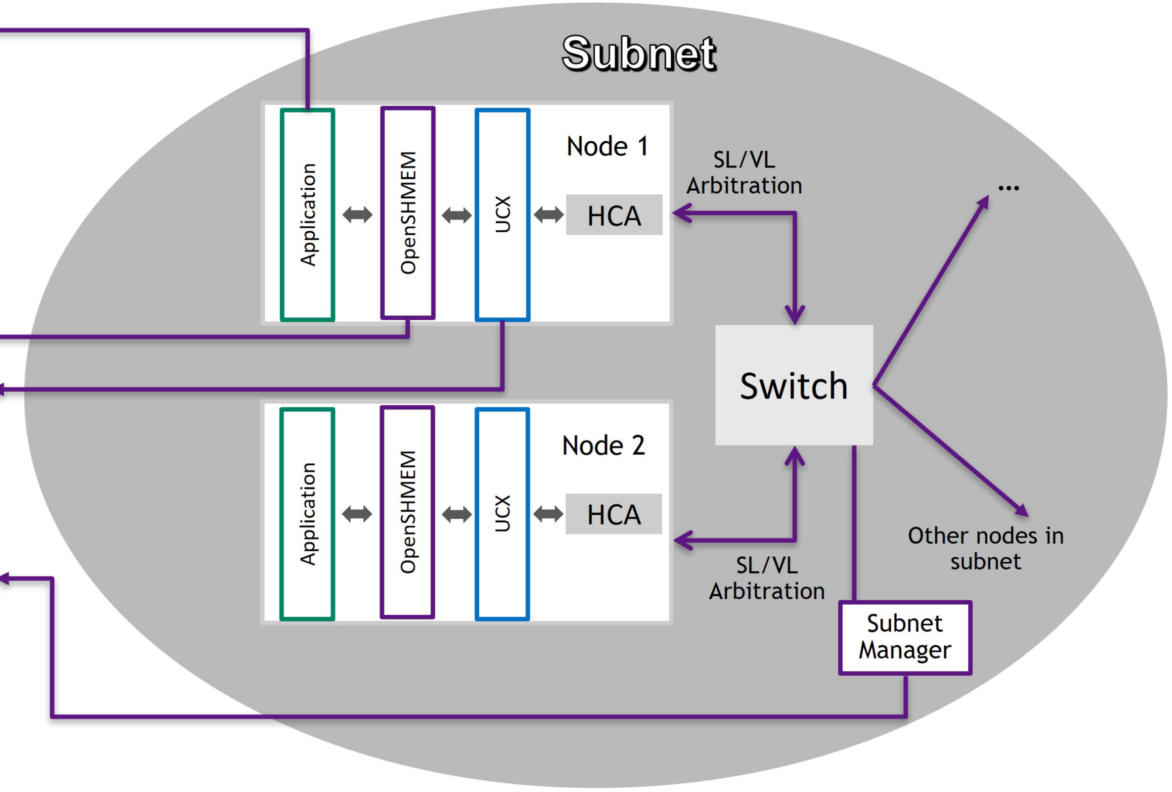
Application -
Case Study – Synthetic benchmark with short message and congestor traffic together

OpenSHMEM - Select SL for each Process (test/congestor) using OpenSHMEM queue API. (create a QoS priority queue)

UCX – UCP worker selects relevant IBA SL value based on User - priority

Subnet Manager Configuration
(three priority lanes)

qos TRUE
qos_max_vls 3
qos_high_limit 255
qos_vlarb_high 1:255 -> lane 1 for high priority
qos_vlarb_low 0:64, 2:64 -> lane 0 and 2 low priority
qos_sl2vl 0,1,2



Experimental Evaluation

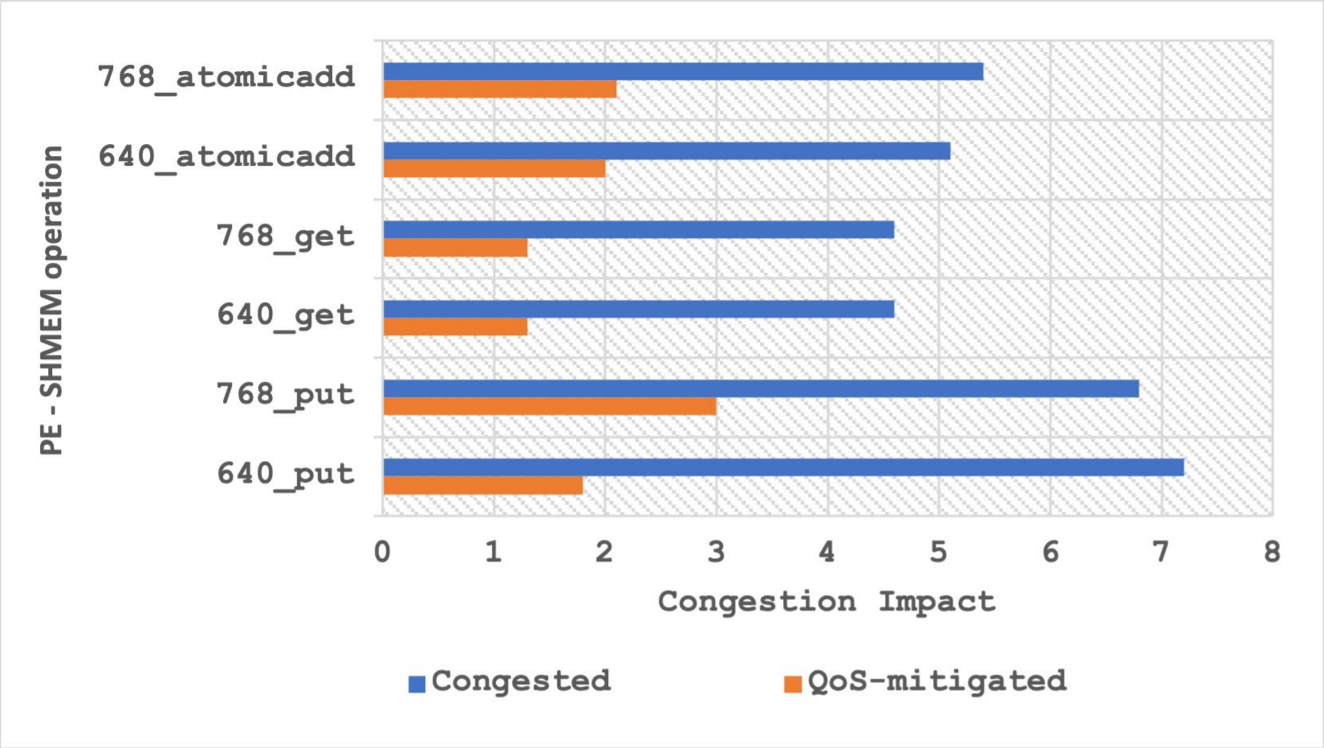
Setup

- IRIS Cluster – HPC Advisory Council
- Dual Socket Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz (Cascade Lake)
  - 32 node Cluster – 28 cores/node
- NVIDIA ConnectX-6 HDR100 100Gb/s InfiniBand/VPI adapters with Socket Direct
- NVIDIA HDR Quantum Switch QM7800 40-Port 200Gb/s HDR InfiniBand
- Memory: 187GB DDR4 2933 MHz RDIMMs per node
  - Exclusive access with control over Subnet manager for measurement
- Three service lanes created – 1 with highest priority for prioritizing network traffic

Results

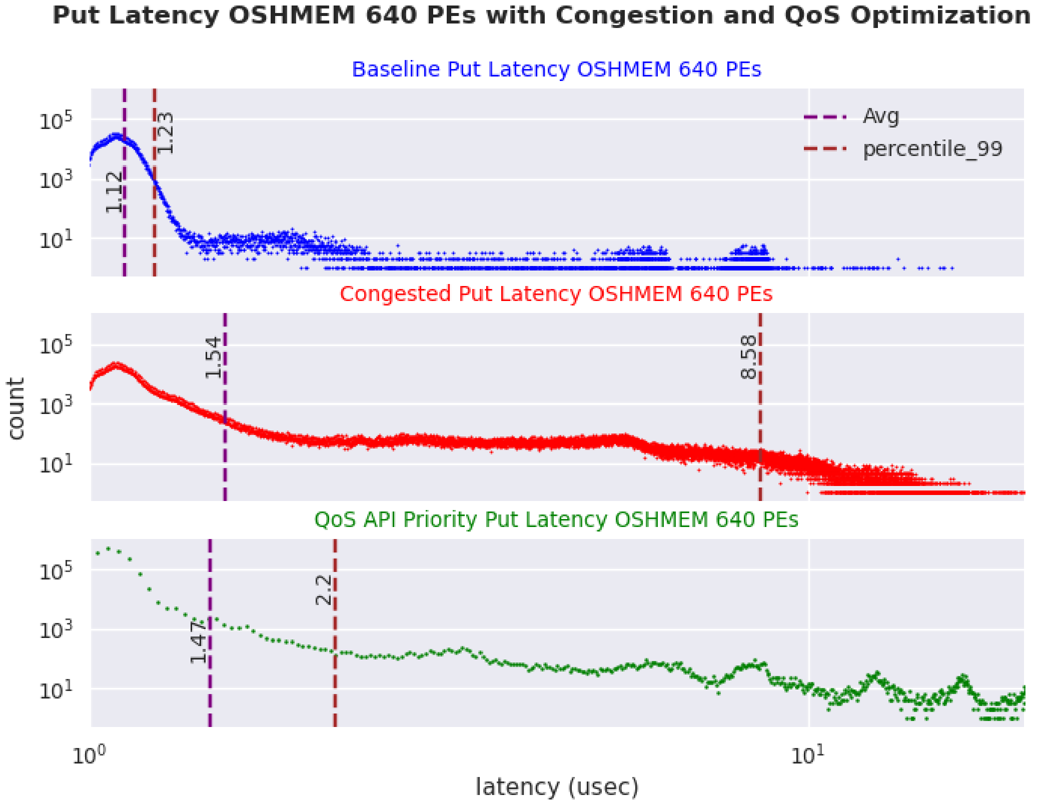
Congestion Impact – P99 Tail Latency

CI = p99_latency(congested)/p99_ latency(baseline)



Results (II)

Detailed Histogram

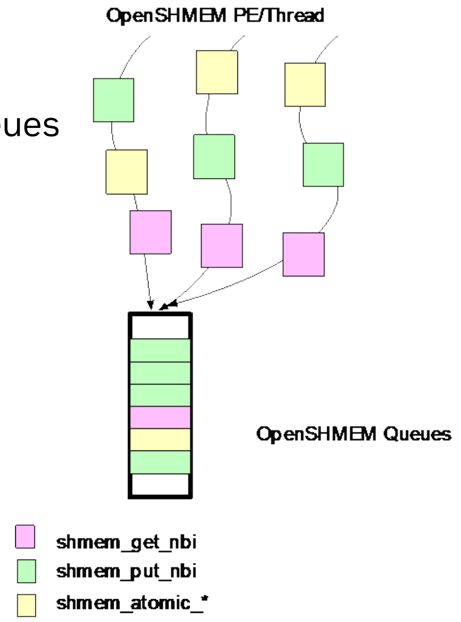


OpenSHMEM Queues – Aggregation (Work in Progress)

- Aggregated/Batched RMA operations lead to higher message rate and bandwidth
  - Hardware capability to achieve higher message rates specifically for small messages limited by software impediments
  - Bale Package - exstack, conveyors demonstrate the importance of aggregation for message rate

- Aggregating Puts/Gets/Atomics from multiple OpenSHMEM PEs using OpenSHMEM communication queues
  - Design Choices: **postlist**, optimizing h/w fences, UMR, DPUs

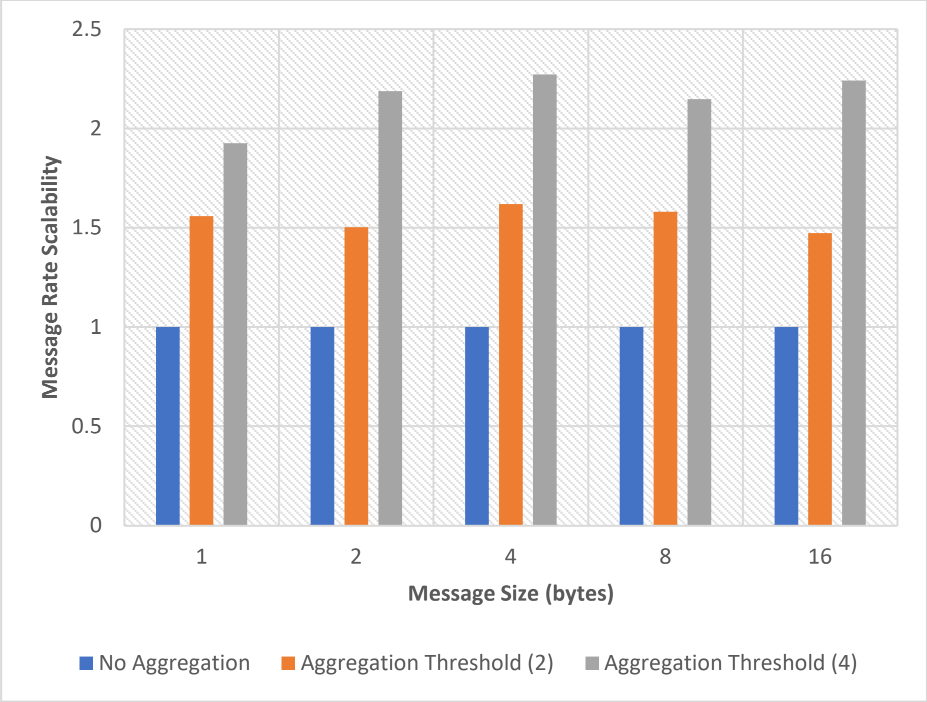
- OpenSHMEM communication queues for Aggregation
  - Current verbs Postlist based aggregation implementation
  - Can merge WR entries by passing a list through the ibv_sge structure.
  - Leverage IOV datatype in ucp_put/get_nbx instead of simple send/recv buffer semantics
    - Trigger aggregation by using the datatype argument in ucp_request_param_t
    - *rc verbs* provider in UCT layer is used for submitting a postlist for this implementation.
    - Constructs a sg_list based on the number of WRs and submitted through ibv_post_send



OpenSHMEM Queues – Aggregation (Early Results)

Work in progress

- System information
  - IRIS cluster from HPC Advisory Council machines
  - Dual Socket Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz (Cascade Lake)
  - 32 node Cluster – 28 cores/node
  - NVIDIA ConnectX-6 HDR100 100Gb/s InfiniBand/VPI adapters with Socket Direct
  - NVIDIA HDR Quantum Switch QM7800 40-Port 200Gb/s HDR InfiniBand
  - Memory: 187GB DDR4 2933 MHz RDIMMs per node
- Benchmark
  - OSU OpenSHMEM message rate benchmark
    - Modified with OpenSHMEM communication queue APIs
    - Threshold provided with environment variable
    - Aggregation shows around 2.2x improvement in message rate .
  - Higher threshold tests are in progress.



Summary and Future Work

- Congestion under heavy load can have a big impact on application performance.

- **OpenSHMEM priority queues helps achieve Shorter tail latency even with congestion**
  - Our Results Demonstrate > 2.5x reduction in tail latency with our approach

- The OpenSHMEM queues API also provides a neat abstraction for QoS
  - Allows users to prioritize communication traffic from OpenSHMEM applications when there are ordering concerns
  - Enables portability for new congestion mitigation approaches/hardware in future
    - Investigate in-network telemetry aware congestion mitigation from OpenSHMEM
    - Using Cache Stashing in supported processors as a design point to reduce tail latency.

- **Ongoing and Future work**
  - Improving Message rate with better aggregation mechanisms for OpenSHMEM queues
  - Support OpenSHMEM data queues aggregation
  - Explore offloading opportunities for progress and aggregation offload to DPUs

NVIDIA