Hewlett Packard
Enterprise

OPENFAM: PROGRAMMING DISAGGREGATED MEMORY

Sharad Singhal, Hewlett Packard Labs

Cray Users' Group (CUG'22), May, 2022

The OpenFAM library was developed by

Clarete Crasta
Chinmay Gosh
Cynara Justine
Faizan Barmawer
Gautham Bhat
Mashood Abdulla
Ramya Ahobala Rao
Rishikesh Rajak
Sherin George
Soumya P N
Vrashi Ponnappa

OUTLINE

- Motivation
- OpenFAM: programming fabric-attached memory
  - Overview and concepts
  - API methods
  - Performance
- Summary & future work

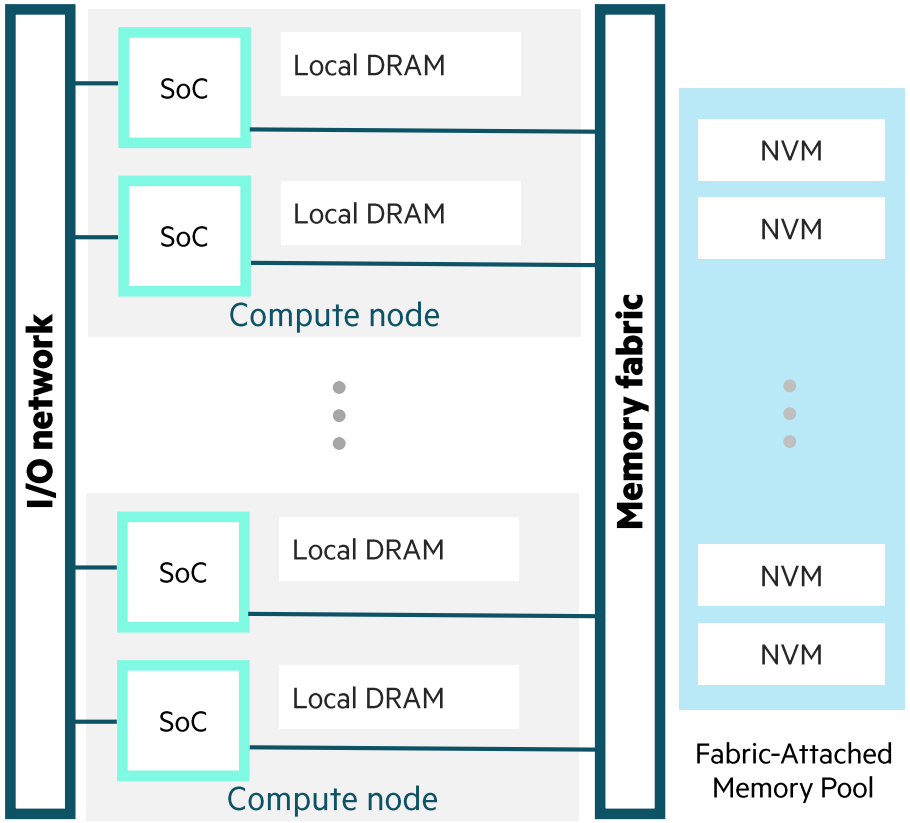
MOTIVATING USE CASES

- Large scale graphs
  - Applications in
    - Security: website reputation, malware detection
    - Social networks: Community detection, link prediction
    - Advertising: brand reputation, click-through prediction
    - Internet of things: traffic management, risk detection
  - Fabric-attached memory (FAM) provides a balance between
    - Scale-out compute
    - Scale-up memory

- HPC workflows/pipelines such as those in genomics
  - Applications to transform data in a workflow with large intermediate data sets
  - FAM provides fast intermediate storage for staging applications
    - Much faster than SSD/file system
    - Not tied to compute nodes

FABRIC-ATTACHED (PERSISTENT) MEMORY

- **Benefits**
  - **Independent scaling** of compute and memory
    - Improved utilization, reduced overprovisioning
    - Decoupling of failure domains
  - **Reduced depth of I/O stack** for large workloads
    - Byte addressable NVM replaces disks/SSDs
    - No conversion from in-memory formats
- **Challenges**
  - **Coherence domains are limited** to individual nodes
    - Transparent caching or paging limits scalability
  - **Fabric latency** is large compared to local memory
    - Software has to be (usually) refactored for performance
  - **Few easy-to-use APIs** available for developers
    - A number of efforts[1,2] underway
      - Generally use object store or file system abstractions



1. Intel DAOS: https://github.com/daos-stack/daos
2. M. Grodowitz, P. Shamis, and S. Poole, "OpenSHMEM I/O Extensions for Fine-Grained Access to Persistent Memory Storage," in *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI*, vol. 1315, J. Nichols, B. Verastegui, A. 'Barney' Maccabe, O. Hernandez, S. Parete-Koon, and T. Ahearn, Eds. Cham: Springer International Publishing, 2020, pp. 318–333; https://github.com/openucx/shmem-opensnapi

OPENFAM: API FOR PROGRAMMING FABRIC-ATTACHED MEMORY

- **Inspired by**
  - OpenSHMEM[1]
  - HPE's work on The Machine program
  - Increasing availability of persistent memory
- The API[2] largely parallels the OpenSHMEM 1.3 API
  - Get/put, scatter/gather, atomics, memory ordering (fence/quiet)
  - Collectives are not included, except for barrier
  - Allocation API is different

- **Differences** between OpenFAM and OpenSHMEM environments:
  - **Focus is on fabric-attached memory** (FAM) instead of remote processing element (PE) memory
    - Node memory is treated as private, while FAM is shared
    - Any PE can access FAM-resident data independently of other PEs
  - FAM-resident **data can outlive the application**
    - Named data items, directory operations (e.g., lookup)
    - Permissions associated with allocations
  - FAM is **not associated with any PE**
    - Allocation/deallocation can be done from any PE
    - Any PE can access any allocation

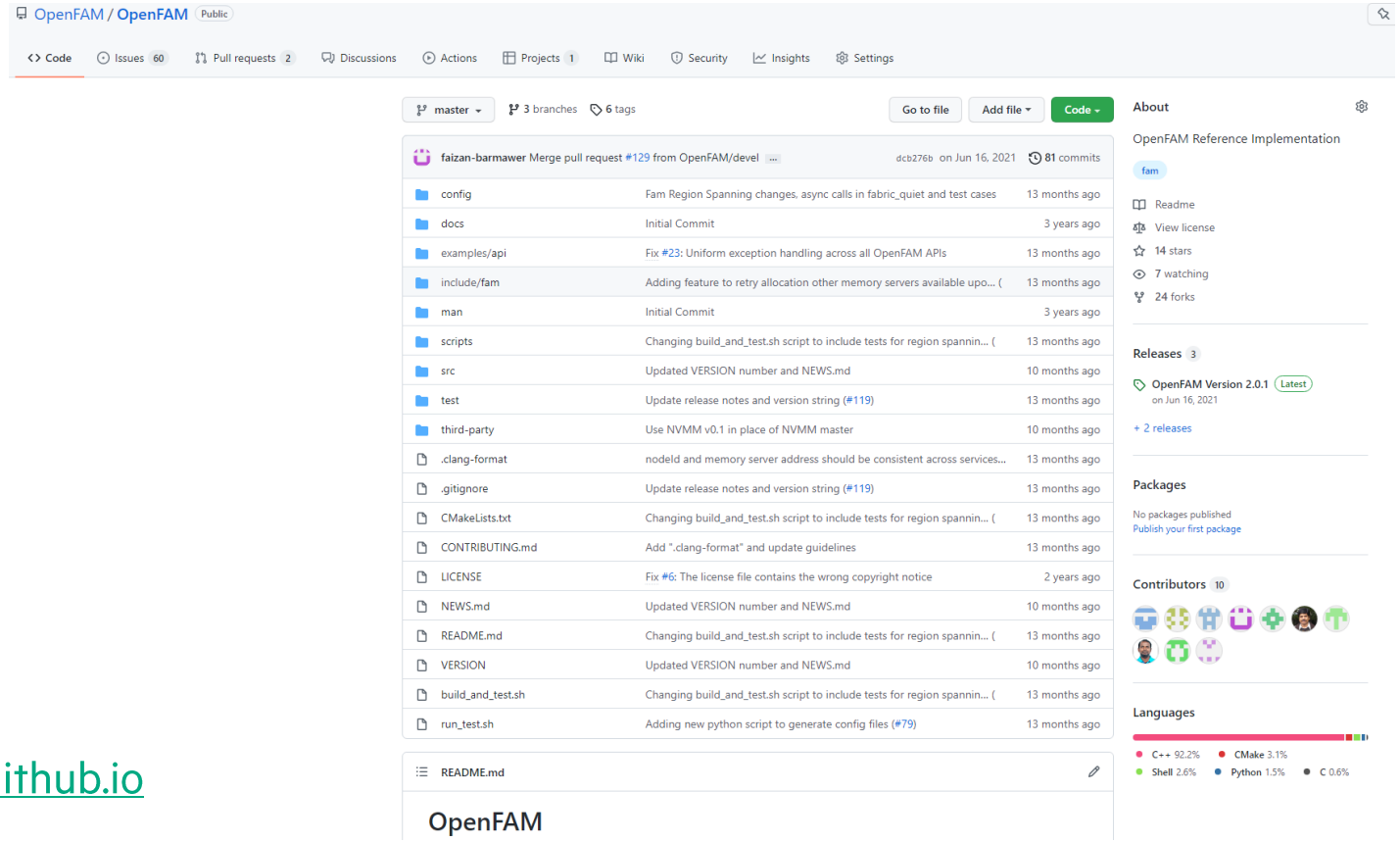
1. OpenSHMEM : http://www.openshmem.org/site/Specification
2. Keeton K., Singhal S., Raymond M. (2019) *The OpenFAM API: A Programming Model for Disaggregated Persistent Memory*. In: Pophale S., Imam N., Aderholdt F., Gorentla Venkata M. (eds) OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Extreme Heterogeneity. OpenSHMEM 2018. Lecture Notes in Computer Science, vol 11283. Springer, Cham

OPENFAM REFERENCE IMPLEMENTATION

Availability: version 2.0.1

- BSD 3-Clause license to enable broad use
- OpenFAM code availability
  - https://github.com/OpenFAM/OpenFAM
  - https://github.com/OpenFAM/OpenFAM_ATL
  - C++ API in
    - include/fam.h
    - include/fam_exception.h
  - Tests include
    - Unit and regression tests
    - Code coverage
    - Micro-benchmarks
    - Example applications
      - Page Rank
      - SpMV
  - Scripts to build third-party dependencies
- API documentation available at https://OpenFAM.github.io

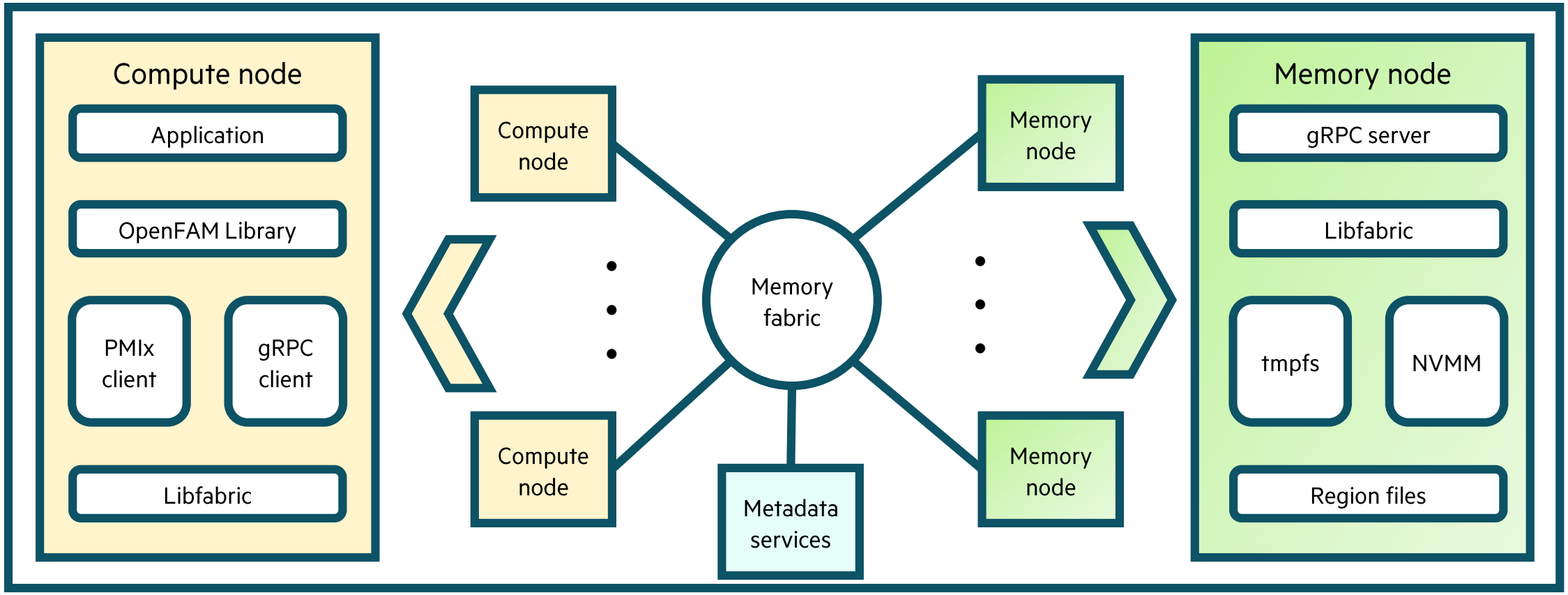
- Built and tested on Ubuntu 18.04 LTS, CentOS
  - Tested on Superdome X, Omnipath and InfiniBand clusters
  - Currently testing with SLES 15 SP3, Ubuntu 20.04 LTS



All OpenFAM code development is open-source

- *The **master** branch contains the last stable version (2.0.1)*
- *The **devel** branch contains current development (3.0)*

OPENFAM REFERENCE IMPLEMENTATION
Architecture



FAM ADDRESSING

- FAM is addressed using descriptors
  - Opaque read-only data structures that uniquely identify a location in FAM
  - Descriptors are accessible across OS instances and applications
    - Use base + offset addressing
  - While it is possible to map FAM to a “flat address”, descriptors carry additional information

- General API pattern:

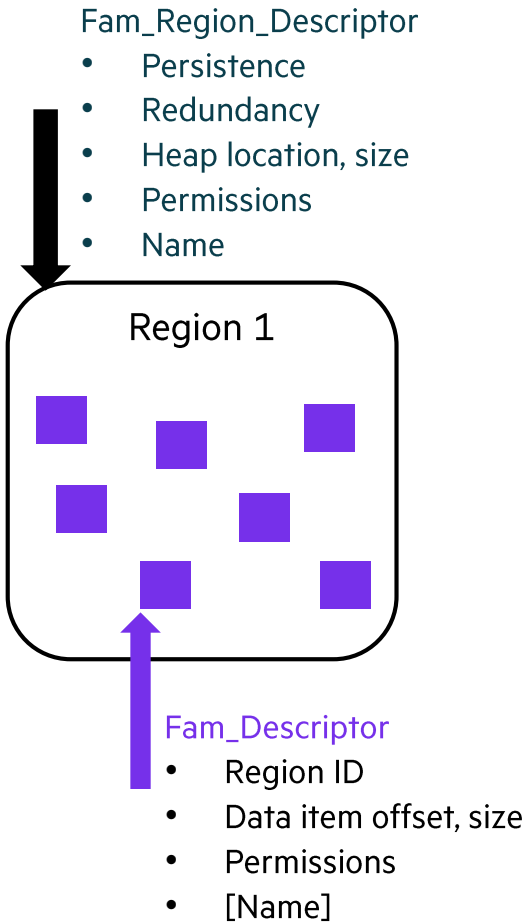
```
void fam_get_blocking(void *local, Fam_Descriptor
*descriptor, uint64_t offset, uint64_t nbytes);
```

Local DRAM buffer

FAM data item descriptor

Byte offset within the data item

Number of bytes to transfer



OPENFAM API SUMMARY[1]

API

- **Data path operations**
  - Get, put, gather, scatter
    - Blocking, non-blocking, atomic variants
- **Map/unmap**
  - Map FAM to process address space
    - Available only on scale-up environments
- **FAM atomics**
  - Fetching and non-fetching atomics
- **Memory ordering and collectives**
  - Quiet, fence, barrier
  - Thread safety and contexts (OpenFAM 3.0)
  - Near-memory operations (OpenFAM 3.0)
- **Memory management**
  - Region creation, resizing, destruction
  - Data item allocation, deallocation
- **Miscellaneous**
  - FAM to FAM copy
  - Permission management
  - Data item/region lookup by name
  - FAM backup and restore (OpenFAM 3.0)

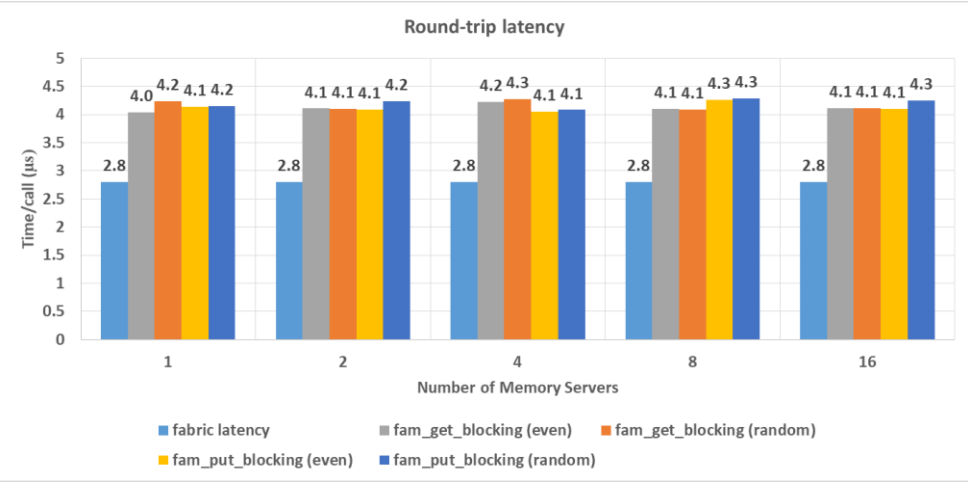
Test configurations

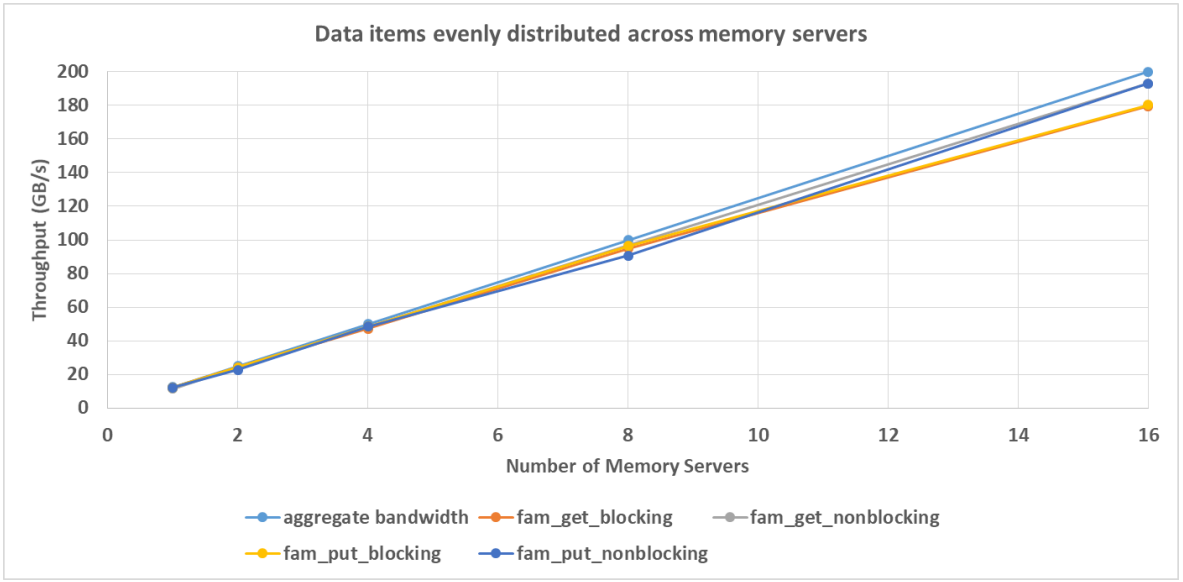
- **Scale-out cluster**
  - 48 nodes from a 96-node InfiniBand cluster
    - 12.5 GBps link bandwidth
    - Fat tree configuration
    - 24 nodes placed within a switch
  - Compute nodes
    - 40 Xeon Gold 6248 cores
    - 188 GB DRAM
      - Serves memory when used as memory node
    - RHEL 8.3
- **Scale-up (MC990x)**
  - Intel(R) Xeon(R) CPU E7-8890 v3 @ 2.50GHz
  - 16 sockets, 18 cores per socket
  - 11.13 TB memory
  - Centos Linux Server 7.6

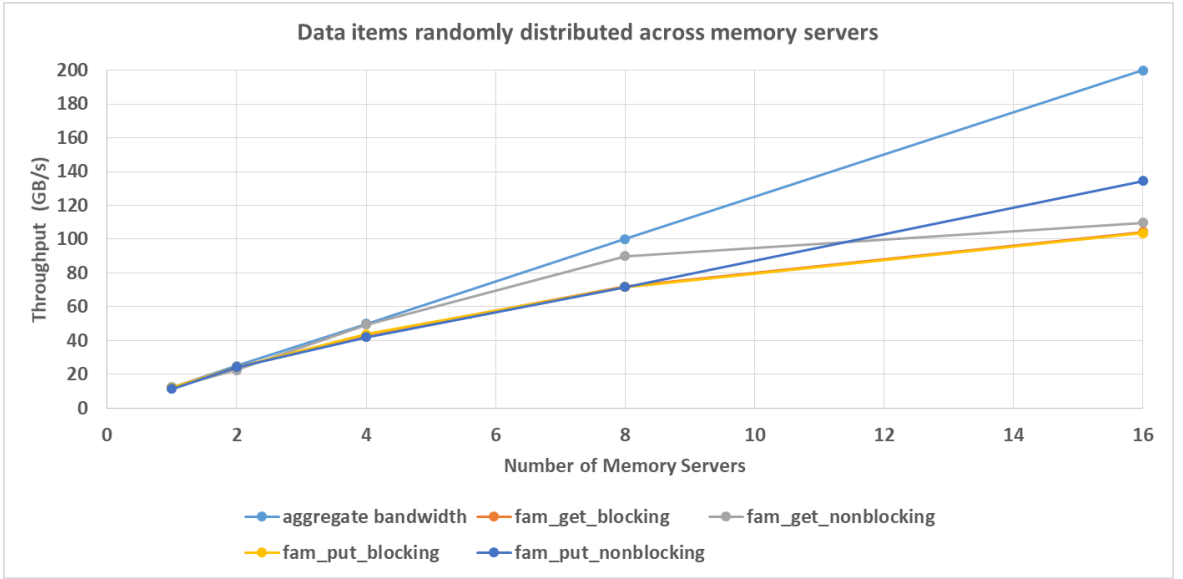
1. The complete 2.0 API is documented at https://OpenFAM.github.io

DATA PATH OPERATIONS
fam_get/fam_put

- **Purpose**
  - Copy data from (to) FAM to (from) DRAM
- **Performance**[1]
  - 16 PEs, each accesses 112 data items 11,200 times
    - 4 MiB transfers for throughput
    - 256 byte transfers for latency
  - Results
    - *Linear scaling close to available bandwidth for balanced system*
    - *Shows in-cast at memory servers for random access*
    - *Short transfer latency < 5 µs*



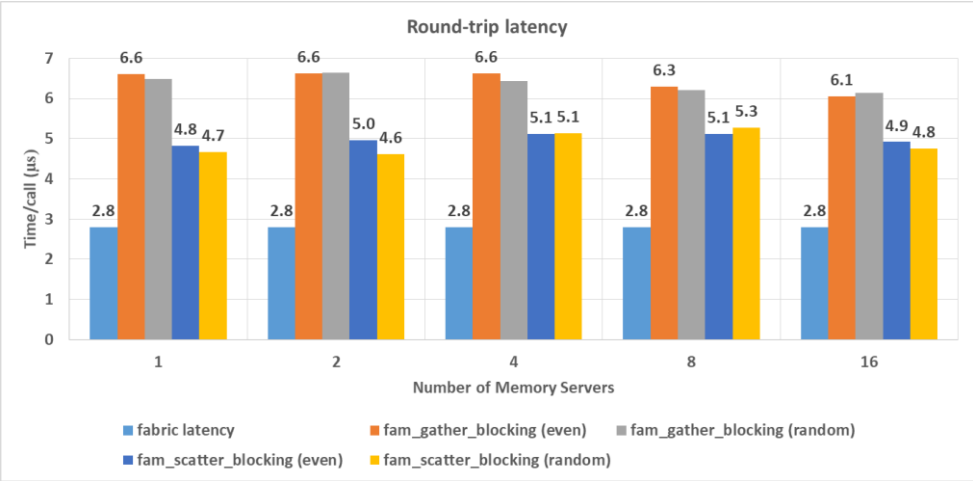


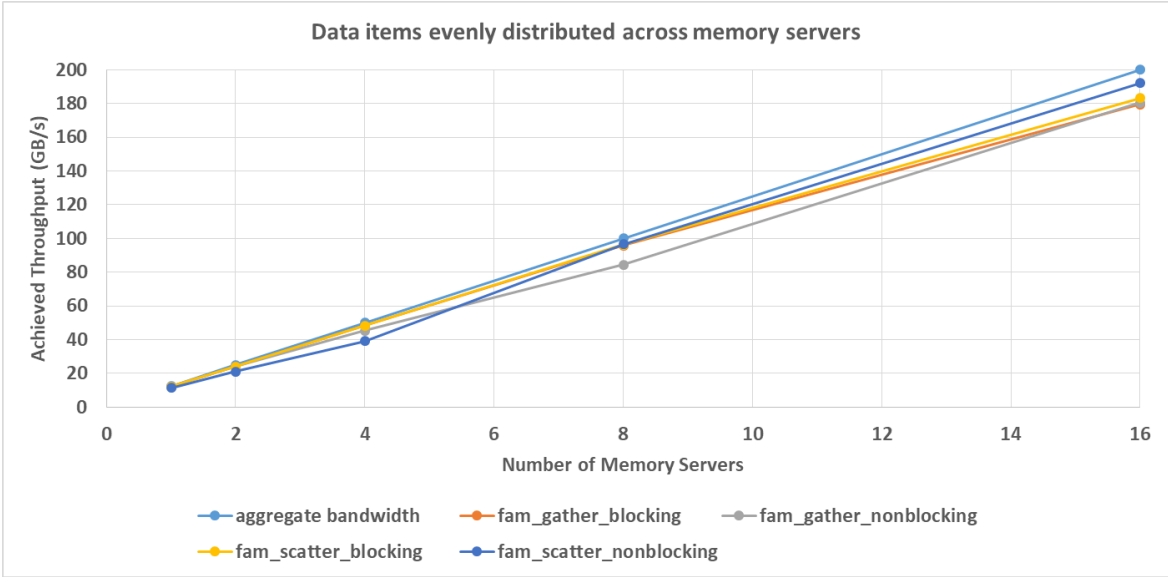


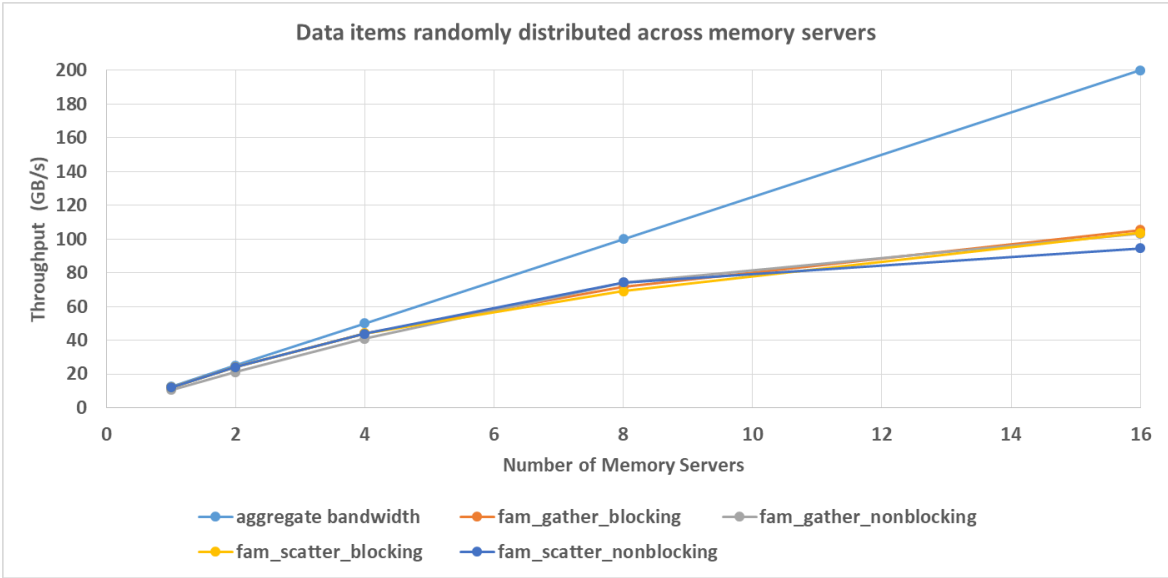
1. S. Singhal et. al., OpenFAM: A library for programming disaggregated memory, in OpenSHMEM and related technologies 2021 workshop, http://www.openshmem.org/workshops/openshmem2021/program.html

DATA PATH OPERATIONS
fam_gather/fam_scatter

- **Purpose**
  - Copy data from (to) FAM to (from) DRAM
- **Performance**[1]
  - 16 PEs, each accesses 112 data items 11,200 times
    - 4 MiB transfers for throughput
    - 256 byte transfers for latency
  - Results
    - *Linear scaling close to available bandwidth for balanced system*
    - *Shows in-cast at memory servers for random access*
    - *Short transfer latency < 10 µs*



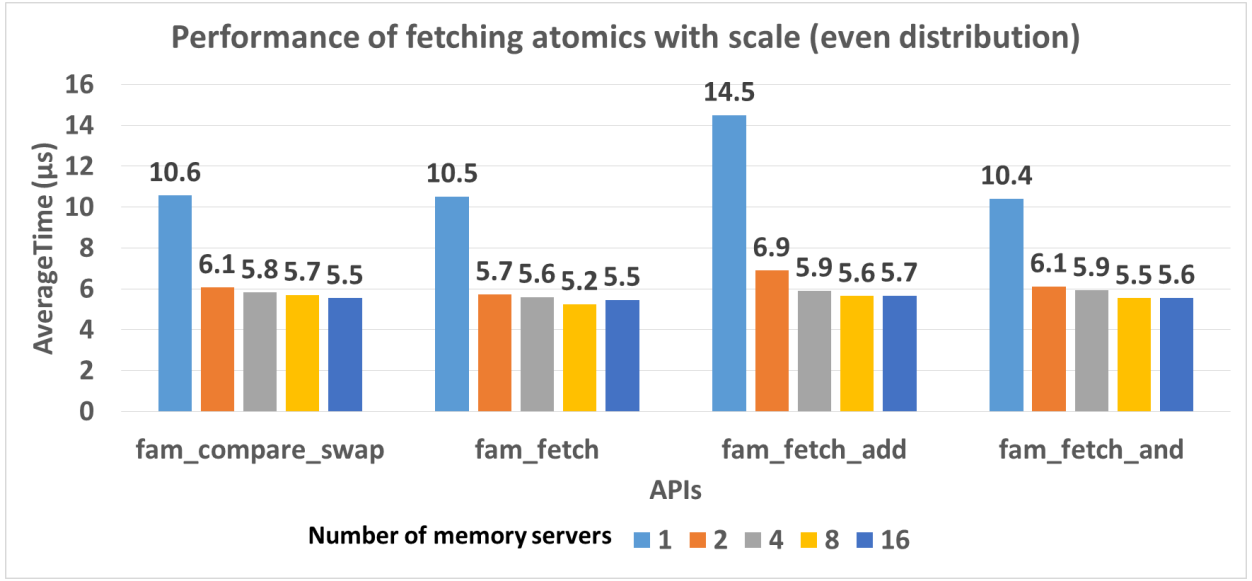


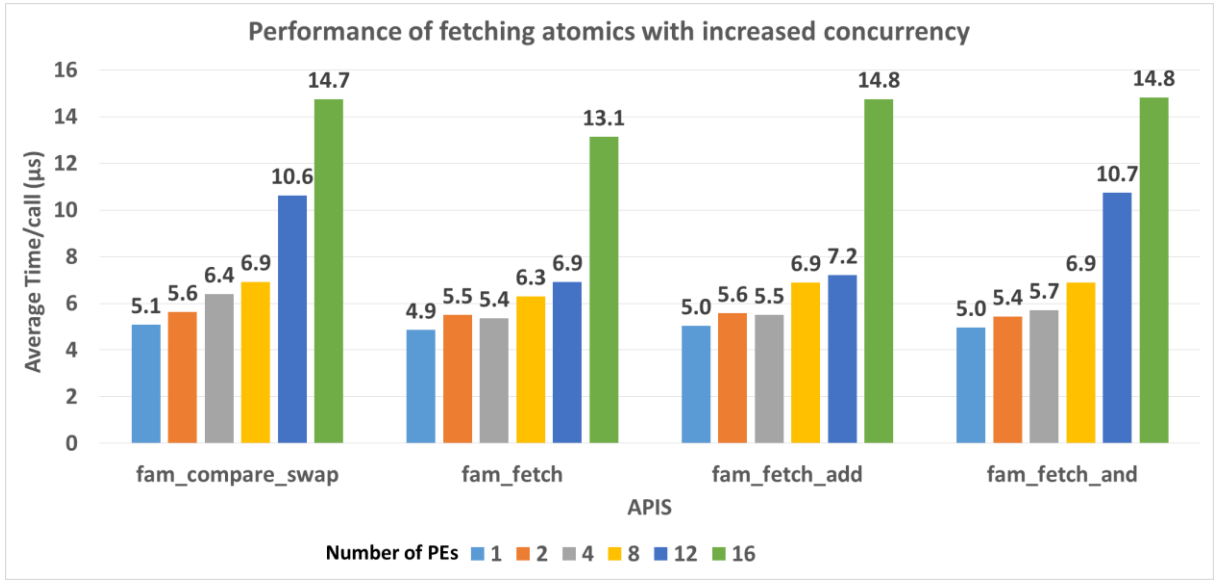


1. S. Singhal et. al., OpenFAM: A library for programming disaggregated memory, in OpenSHMEM and related technologies 2021 workshop, http://www.openshmem.org/workshops/openshmem2021/program.html

FAM ATOMICS OPERATIONS

- Purpose
  - Perform atomics operations on FAM
- Variants
  - Fetching
    - fetch, swap, add, subtract, min, max, and, or, xor, cas
  - Non-fetching
    - Set, add, subtract, min, max, and, or, xor
- Performance
  - Scaling with number of memory servers
    - 16 PEs, 112 data items, 11,200 operations/PE
    - *Round-trip times are comparable to small transfer performance (< 10 µs)*
  - Scaling with concurrency
    - Single memory server
    - 16 PEs, 112 data items, 11,200 operations/PE
    - *Round-trip times increase as concurrency is increased past 8 memory servers*



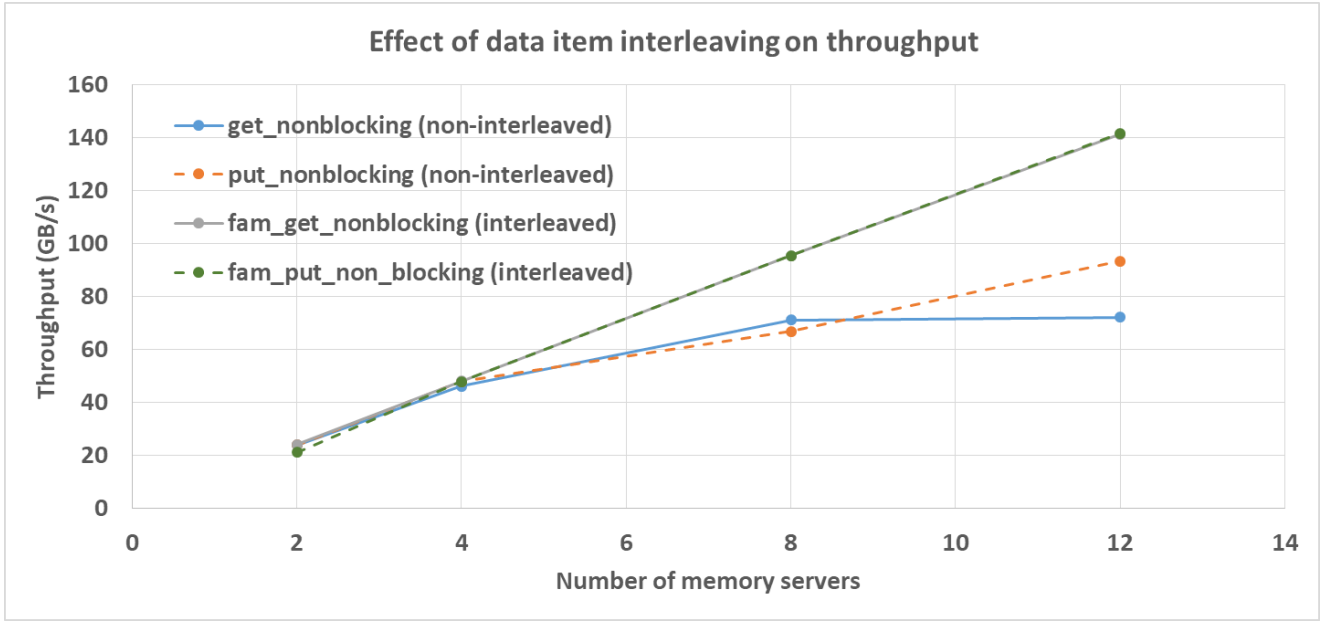


1. S. Singhal et. al., OpenFAM: A library for programming disaggregated memory, in OpenSHMEM and related technologies 2021 workshop, http://www.openshmem.org/workshops/openshmem2021/program.html

DATA PATH OPERATIONS

Data item Interleaving

- Purpose
  - Copy data from (to) FAM to (from) DRAM
- Performance
  - 12 PEs, each accesses independent data items
    - 1 GiB transfers for throughput
    - 128 KiB interleave size
    - 256 byte transfers for latency
  - Results
    - Linear scaling close to available bandwidth for balanced system
    - With interleaving
      - Random distribution no longer shows lower performance
      - Short transfer latency ≈ 4 µs
      - Atomic operations latency ≈ 5-6 µs

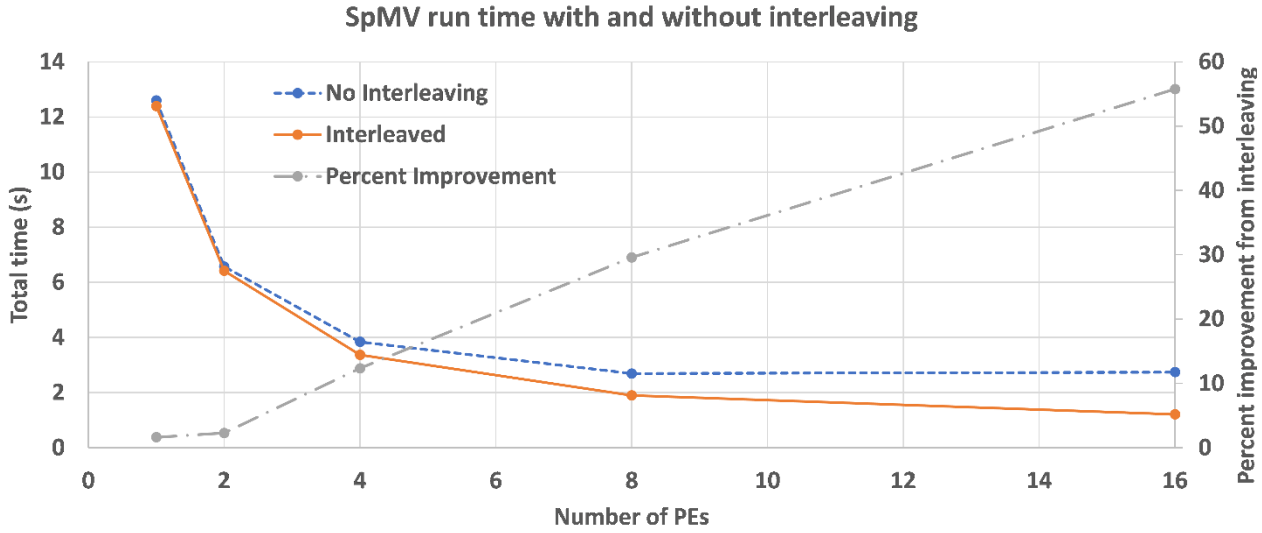


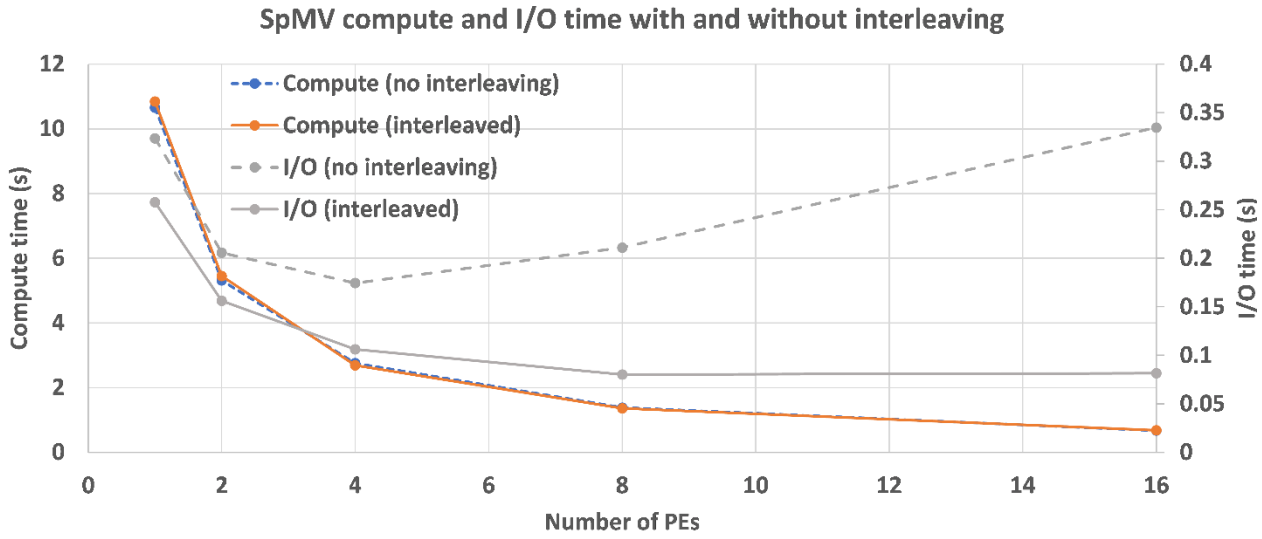

```
Fam_Region_Descriptor *fam_create_region(char *name, uint64_t
size, mode_t permissions, Fam_Region_Attributes*
regionAttributes);

typedef struct {
            Fam_Redundancy_Level redundancyLevel;
            Fam_Memory_Type memoryType;
            Fam_Interleave_Enable interleaveEnable;
} Fam_Region_Attributes;
```

SPARSE MATRIX VECTOR MULTIPLICATION (SPMV)

Interleaving benefits

- Sparse matrix stored in CSR format in FAM
  - Matrix size 2^{27} x 2^{27}, edge factor 4
- Each PE (in parallel)
  - Reads dense vector
  - Loop while <more rows>
    - Read next 2^{20} rows
    - Compute part of result vector
    - Write part of result vector
- Results
  - Data resides on 8 memory servers
  - PEs varied from 1-16
  - Compute bound job at this scale; normal strong scaling behavior
  - ~56% improvement over non-interleaved code at 16 PEs





DATA PATH OPERATIONS
Contexts and multi-threading

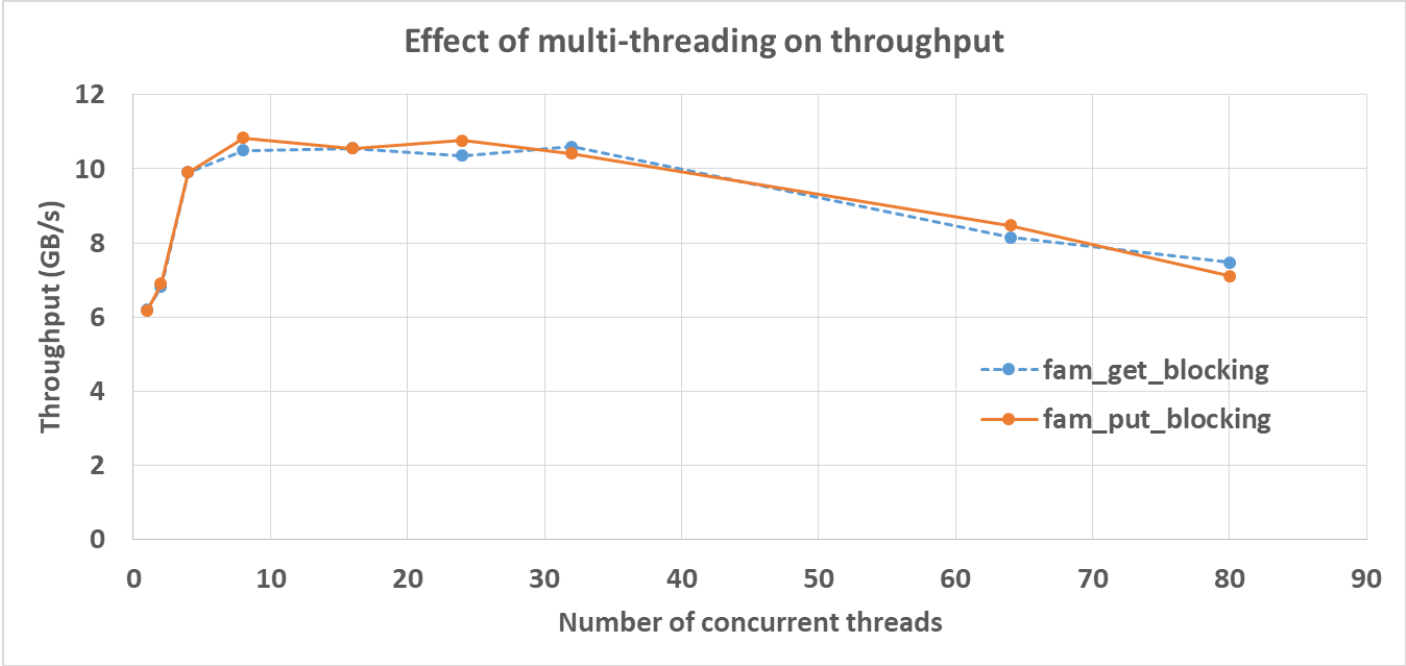
- Contexts allow partitioning of I/O operations into sub-groups
  - A `quiet()` operation on a contexts only waits for completion within the context
  - A `progress()` operation allows the application to check the number of pending I/O operations
- Enables the application to efficiently
  - Overlap computation and communication with multi-threaded PEs
  - Handle more I/O contexts than the number of available cores

```
fam_context* fam->fam_context_open();
void fam->fam_context_close(fam_context*);
uint64_t  fam_progress();
```

```
…
fam *myFam = new fam();
…
fam_context  *myFamCtx = myFam->fam_context_open();

myFamCtx->fam_get_nonblocking(local1, descriptor1,
6*sizeof(int), 10*sizeof(int));
myFamCtx->fam_get_nonblocking(local2, descriptor1,
26*sizeof(int), 10*sizeof(int));
…

myFamCtx->fam_quiet();
myFam->fam_context_close(myFamCtx);
…
```

DATA PATH OPERATIONS
Multi-threading & contexts

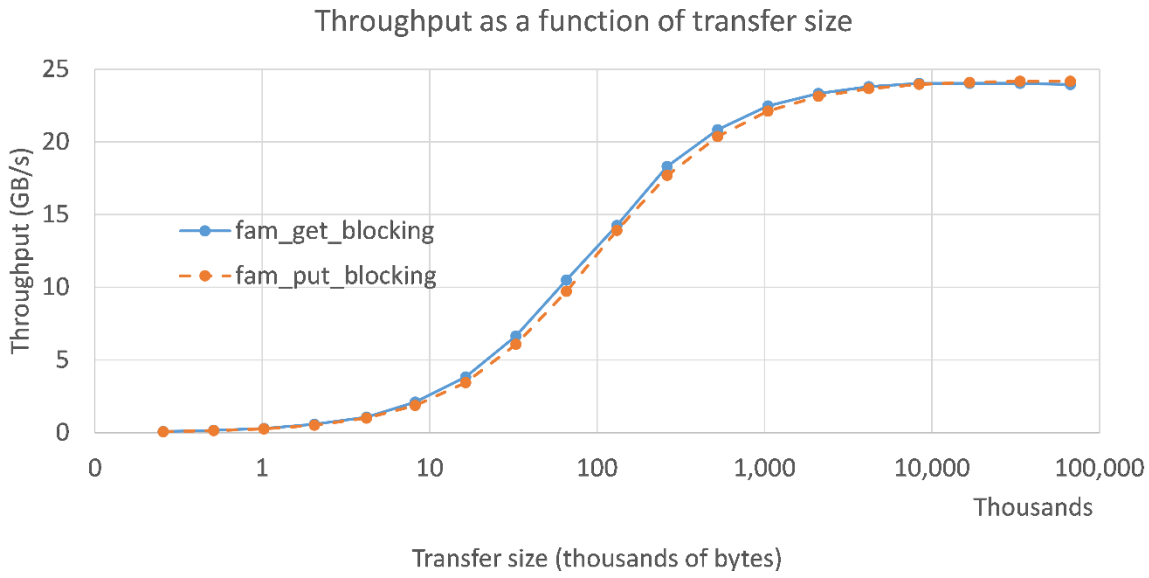
- Purpose
  - Understand effect of multi-threading
  - 40 cores (80 hyperthreads) in the compute node
- Performance
  - 1 memory server, 1 PE
    - 64 KiB data transfer
    - 10,000 transfers/thread averaged
  - Results
    - ~8 threads needed to drive network to saturation
    - Performance drops beyond 32 threads
      - Thread interference



DATA PATH OPERATIONS

Initial results using Slingshot

- Single PE connected to a single memory server across a switch
  - 25 GB/s bandwidth
- Get_blocking & put_blocking calls with different message sizes
  - Round-trip latency ≈ 3.7 µs with 256 byte messages
  - We see expected S-shape behavior as message size is increased from 256 bytes to 64 MiB
    - 23.9 GB/s for get_blocking (64 MiB)
    - 24.1 GB/s for put_blocking (64 MiB)
  - Expect the curve to move left with multi-threading



SUMMARY & FUTURE WORK

- OpenFAM API is a programming model for disaggregated persistent fabric-attached memory
  - Focuses on fabric-attached memory (FAM) instead of remote processing element (PE) memory
  - Introduces naming, permissions and other operations to accommodate persistent FAM-resident data
  - Employs two-level allocation scheme to manage large FAM capacity and scale
  - Uses opaque descriptors for portable references to FAM-resident data across OS instances
- Library is open-source
  - Available at https://github.com/OpenFAM
  - Contributions are welcome

- Current work
  - Added APIs for multiple contexts and multi-threading
  - Slingshot support
  - FAM resilience
  - Integration with other programming APIs
    - C, OpenSHMEM, HDF5, Chapel, …