OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

BOF: LIBFABRICS AND HPC RUNTIME SYSTEMS

Yanfei Guo, Wesley Bland, Hajime Fujita, Howard Pritchard, James Dinan

Argonne National Laboratory, Intel Corporation, Los Alamos National Laboratory

[March 21, 2019]

AGENDA

Speakers

- Yanfei Guo (Argonne National Laboratory)
- Wesley Bland (Intel Corporation)
- Hajime Fujita (Intel Corporation)
- Howard Pritchard (Los Alamos National Laboratory)
- James Dinan (Intel Corporation)

Talks

- Libfabric in MPICH
- Libfabtic Usage in MPICH at Intel
- OpenMPI Use of OFI Libfabric
- Libfabric and OpenSHMEM

Discussion and Q&A

15th ANNUAL WORKSHOP 2019

STATUS OF LIBFABRIC IN MPICH

Yanfei Guo

Argonne National Laboratory

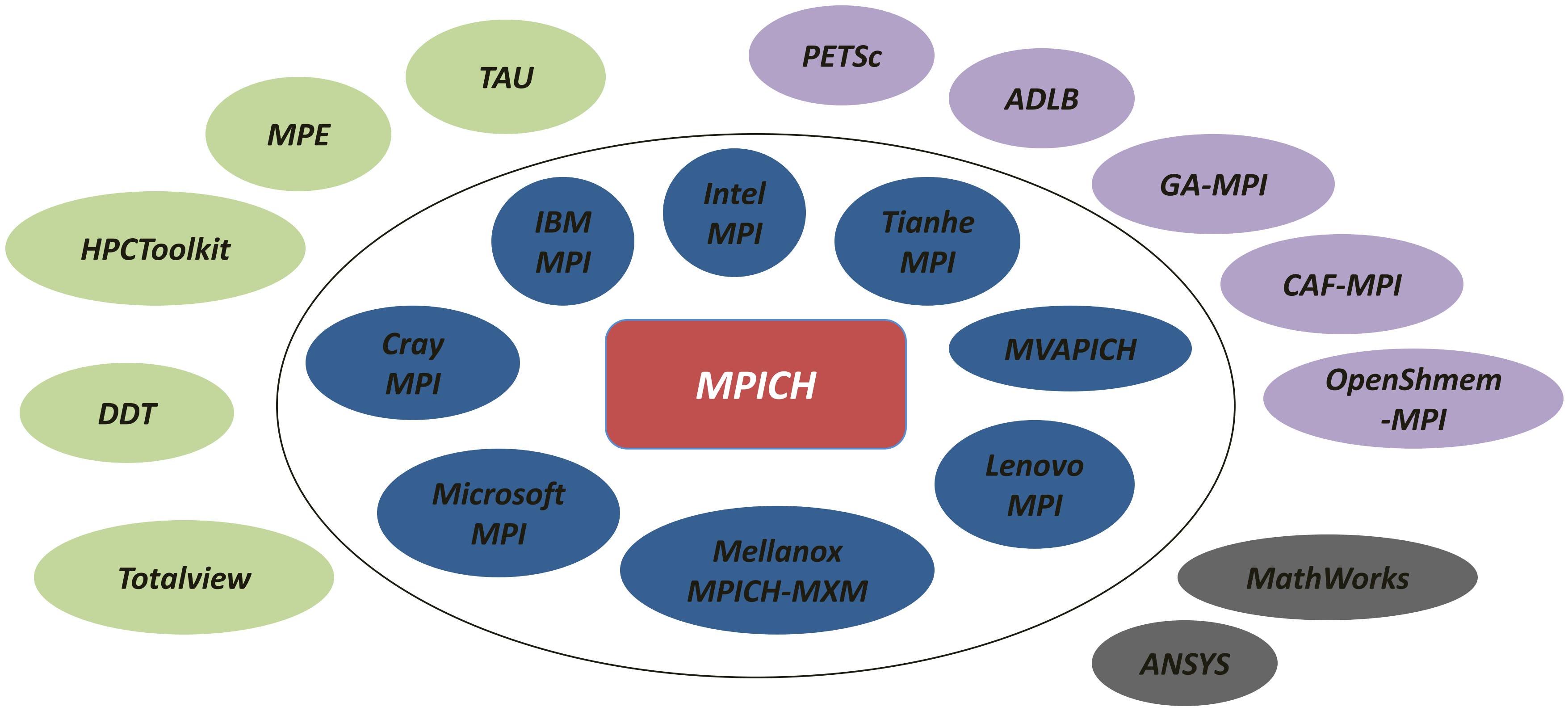
[March 21, 2019]

WHAT IS MPICH?

- **MPICH is a high-performance and widely portable open-source implementation of MPI**
- **It provides all features of MPI that have been defined so far (up to and include MPI-3.1)**
- **Active development lead by Argonne National Laboratory and University of Illinois at Urbana-Champaign**
  - Several close collaborators who contribute features, bug fixes, testing for quality assurance, etc.
    - IBM, Microsoft, Cray, Intel, Ohio State University, Queen’s University, Mellanox, RIKEN AICS and others
- **Current stable release is MPICH-3.2**
- **Latest release is MPICH-3.3a2**
- **www.mpich.org**

MPICH: GOAL AND PHILOSOPHY

- MPICH aims to be the preferred MPI implementation on the top machines in the world
- Our philosophy is to create an “MPICH Ecosystem”



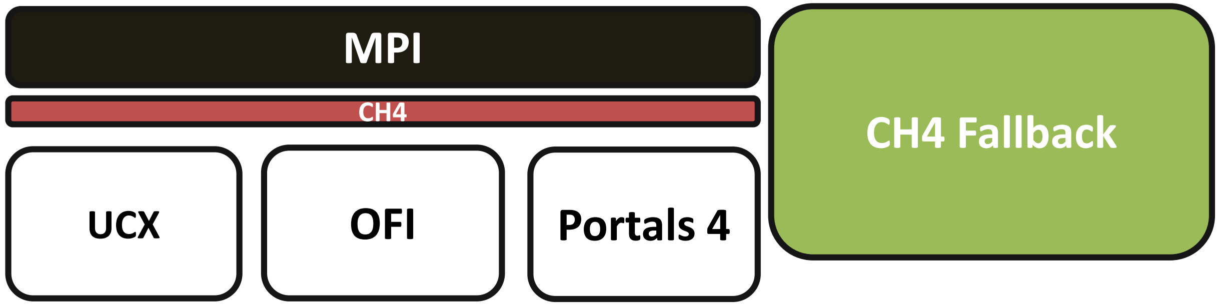
MOTIVATION

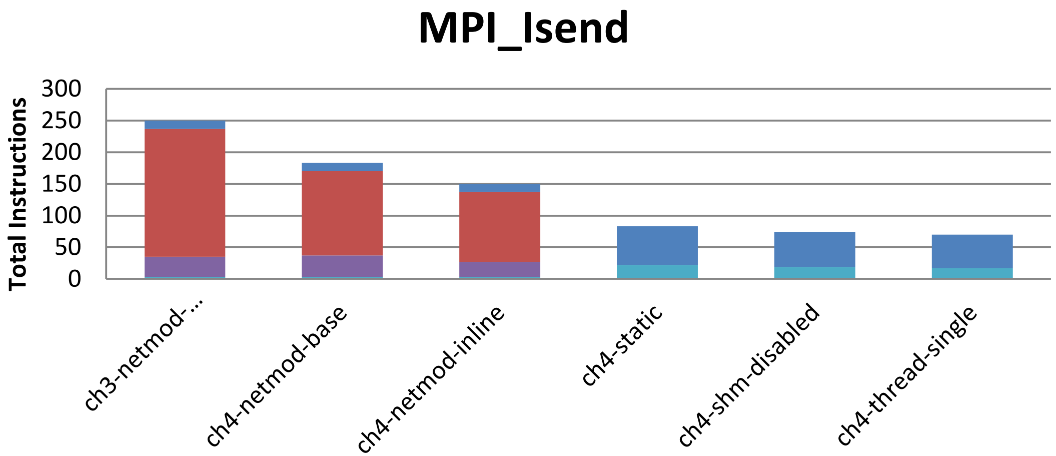
- **Why OFI/OFIWG?**
  - Support for diverse hardware through a common API
  - Actively, openly developed
    - Bi-weekly calls
    - Hosted on Github
  - Close abstraction for MPI
    - MPI community engaged from the start
  - Fully functional sockets provider
    - Prototype code on a laptop

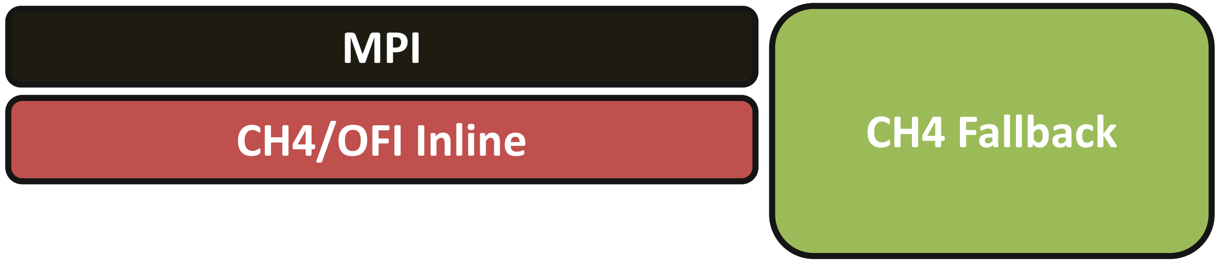
MPICH-3.3 SERIES

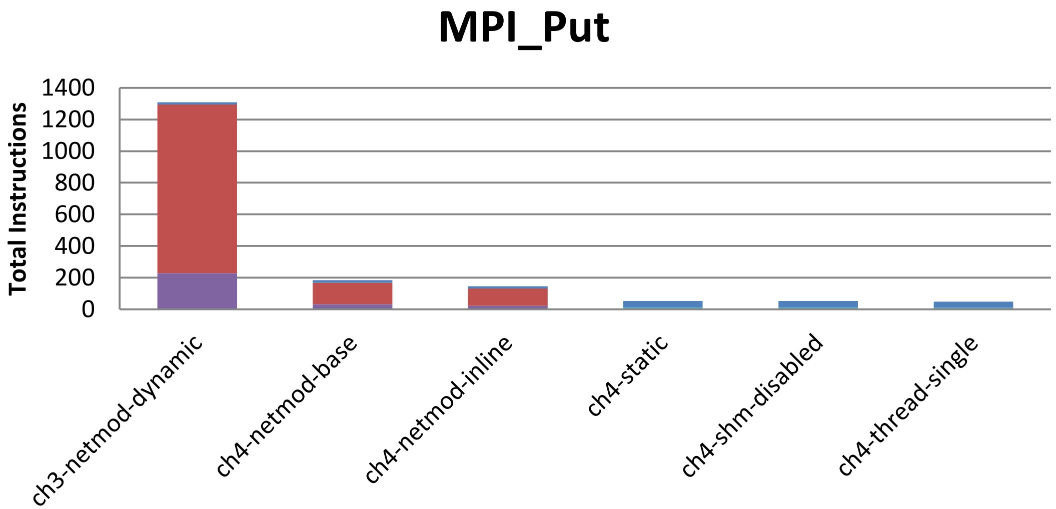
- **Introducing the CH4 device**
  - Replacement for CH3, but we will maintain CH3 till all of our partners have moved to CH4
  - Co-design effort
    - Weekly telecons with partners to discuss design and development issues
  - Two primary objectives:
    - Low-instruction count communication
      - Ability to support high-level network APIs (OFI, UCX, Portals 4)
      - E.g., tag-matching in hardware, direct PUT/GET communication
    - Support for very high thread concurrency
      - Improvements to message rates in highly threaded environments (MPI_THREAD_MULTIPLE)
      - Support for multiple network endpoints (THREAD_MULTIPLE or not)

REDUCING OVERHEAD



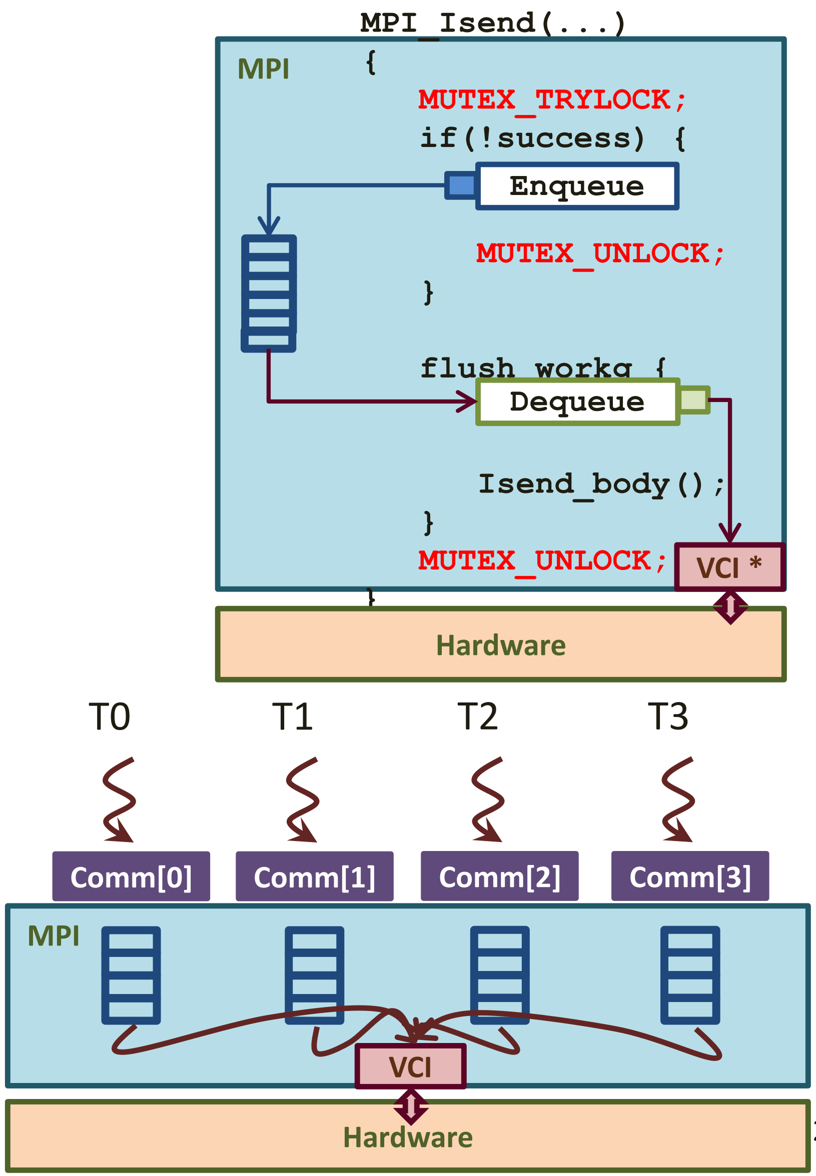






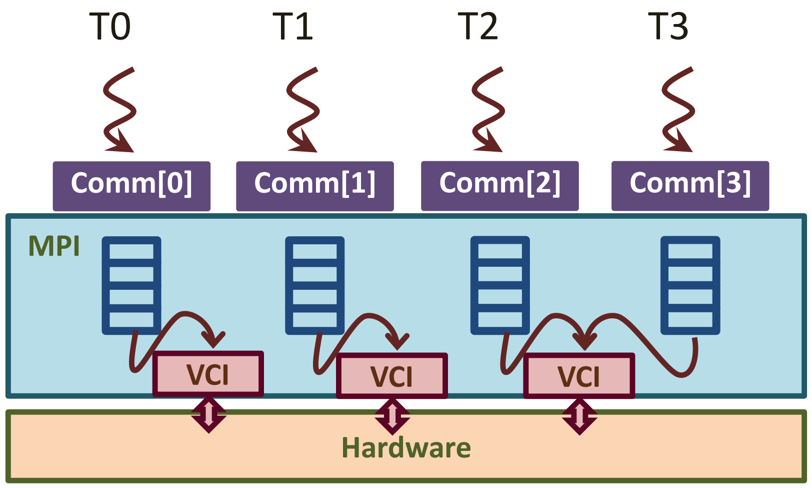
OPTIMIZATIONS FOR MPI+THREAD

- **Proposed solution: Work-Queue Model**
  - One or multiple **work-queues per endpoint**
  - Decouple blocking and nonblocking operations
  - Nonblocking operations enqueue **work descriptors** and leave if critical section held
  - Threads issue work on behalf of other threads when acquiring a critical section
  - Nonblocking operations **are truly** nonblocking
- **Multiple isolated work-queues**
  - Transparent to the user
  - E.g. one Work-Queue per communicator, per neighbor process (regular apps)



MULTIPLE VIRTUAL COMMUNICATION INTERFACE (VCI)

- **Virtual Communication Interface (VCI)**
  - Each VCI abstracts a set of network resources
  - Some networks support multiple VCIs: InfiniBand contexts, scalable endpoints over Intel Omni-Path
  - Traditional MPI implementation uses single VCI
    - Serializes all traffic
    - Does not fully exploit network hardware resources
- **Utilizing multiple VCIs to maximize independence in communication**
  - Separate VCIs per communicator or per RMA window
  - Distribute traffic between VCIs with respect to ranks, tags, and generally out-of-order communication
  - M-N mapping between Work-Queues and VCIs



CURRENT EXPERIENCE

- **Findings**
  - Several bottlenecks inside MPICH
  - Librabric over psm2 exhibit good scalability with independent TX, RX, and CQs
  - Almost 100% parallel efficiency within a NUMA Node
  - Around 85% parallel efficiency when crossing NUMA boundaries
- **Problems**
  - Performance anomalies with different COMM/context mapping

OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

THANK YOU

Yanfei Guo

Argonne National Laboratory

Argonne NATIONAL LABORATORY

OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

LIBFABRIC USAGE IN MPICH AT INTEL

Wesley Bland and Hajime Fujita

Intel Corporation

March 21, 2019

NOTICES AND DISCLAIMERS

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration.
- No product or component can be absolutely secure.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks .
- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/benchmarks .
- Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at http://www.intel.com/go/turbo.
- Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
- Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.
- Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
-
-
-

WHAT IS THIS TALK ABOUT?

- Not specifically an MPICH talk.
- Mostly talking about how MPICH uses libfabric
- How do we manage the feature set of libfabric vs. the needs of MPICH?
- Squarely in “Guru” territory from Sean Hefty’s 2015 OFA Workshop talk

SUPPORT EVERYTHING!

- **How can MPICH use libfabric for portable interconnects with maximum performance?**
  - 👍 Best part of libfabric: It can support almost everything (both inter- and intra-node communication)
  - 👎 Worst part of libfabric: It can support almost everything (both inter- and intra-node communication)
  - 😢 MPICH uses static inlines for most internal function calls (can't use function pointers).
- **How does MPICH deal with the huge number of available features in libfabric?**

Endpoint Types	bgq	gni	mlx	nd	psm	psm2	rxd	rxm
FI_EP_DGRAM	✗	✓	✗	✗	✓	✓	✗	✗
FI_EP_MSG	✗	✓	✗	✓	✗	✗	✗	✗
FI_EP_RDM								

Modes	bgq	gni	mlx	nd	psm	psm2	rxd	rxm	shm	sockets	tcp	udp	usnic	verbs
FI_ASYNC_IOV	R													
FI_BUFFERED_RECV														

Memory Registration Modes	bgq	gni	mlx	nd	psm	psm2	rxd	rxm	shm	sockets	tcp	udp	usnic	verbs
FI_MR_ALLOCATED				R				C						R
FI_MR_ENDPOINT														
FI_MR_LOCAL				R				C						R
FI_MR_PROV_KEY				R										
FI_MR_MMU_NOTIFY														
FI_MR_RAW														
FI_MR_RMA_EVENT														
FI_MR_VIRT_ADDR				R										
FI_MR_BASIC (compat)	✓	✓	✓	✓	✓									
FI_MR_SCALABLE (compat)	✓	✓	✗	✗	✓									

Secondary Capabilities
FI_FENCE
FI_LOCAL_COMM
FI_MULTI_RECV
FI_REMOTE_COMM
FI_RMA_EVENT
FI_RMA_PMEM
FI_SHARED_AV
FI_SOURCE
FI_SOURCE_ERR
FI_TRIGGER

Additional Features	bgq	gni	mlx	nd	psm	psm2	rxd	rxm	shm	sockets	tcp	udp	usnic	verbs
FABRIC_DIRECT	✓	✓	✗	✗	✗	✓		✗	✗	✓	✗	✗	✗	✗
FI_RM_ENABLED	✗	✓	✗		✓	✓		✓	✓	✓	✓	✓	✗	*
Scalable endpoints	✓	✓	✗	✗	✗	✓		✗	✗	✓	✗	✗	✗	✗
Counters (local operations)	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	*
Counters (remote operations)					✓	✓					✗	✗	✗	

CAPABILITY SETS

- **Capability Sets allow MPICH to detect (or be told by the user) the best provider and configuration for performance.**
- **A set of C macro definitions that determine which code path to use.**
- **Even if MPICH did use function pointers, there's so many options that it would probably be impractical to make one for each feature.**
- **Adding support for a new provider requires adding a set of definitions to ofi_capability_sets.h**

```
if (ENABLE_TAGGED) {
   fi_tsend(…);
} else {
   /* Send active message */
}
```

CAPABILITY SETS

- **If we know we will be using a particular provider, we can compile MPICH to only look for that provider's features.**
  - **This lets the compiler remove most of the branches as the macros become constants**
- **If we don't know which provider we'll use at compile time, most included providers have good detection of capabilities.**
  - **Includes the branches**
  - **Common for packagers or systems that want to be able to support multiple interconnects (e.g. sockets and PSM2)**

Best Case

```
if (ENABLE_TAGGED) {
   fi_tsend(…);
} else {
   /* Send active message */
}
```

Common Case

```
if (ENABLE_TAGGED) {
   fi_tsend(…);
} else {
   /* Send active message */
}
```

HIERARCHY – COMPILE TIME USER CHOICE

- **All of the branches go away**
- **Fastest performance**
- **Good for compiling on a specific production system**
- **Usually requires specific library and provider version**
  - No fallback if expected features are not found

Best

HIERARCHY – RUNTIME USER CHOICE

- **Compiles in support for all combinations of capabilities**
- **User specifies desired provider at startup**
  - MPIR_CVAR_OFI_USE_PROVIDER=psm2 mpiexec –n 16 ./a.out
- **Usually still requires specific library and provider version**
  - Will fallback to one of the lesser performance options if no match.

Still Pretty Good

HIERARCHY – RUNTIME USER CHOICE (WITH TWEAKS)

- **Compiles in support for all combinations of capabilities**
- **User specifies desired provider at startup**
  - MPIR_CVAR_OFI_USE_PROVIDER=psm2 mpiexec –n 16 ./a.out
- **Can add specific feature tweaks to turn capabilities on/off if desired**
  - MPIR_CVAR_OFI_USE_PROVIDER=psm2 MPIR_CVAR_OFI_ENABLE_TAGGED=0 mpiexec –n 16 ./a.out
- **Usually still requires specific library and provider version**
  - Will fallback to one of the lesser performance options if no match.

Still
Pretty
Good

HIERARCHY – NO USER CHOICE (KNOWN PROVIDER)

- **Mostly the same result as when the user chooses a provider**
- **As long as we can match a provider to a capability set, the performance is still fine**

Still Pretty Good

HIERARCHY – NO USER CHOICE (UNKNOWN GOOD PROVIDER)

- **Don’t know about the provider**
- **Still meets some good default requirements**
  - Tagged messages
  - Completion queue data
  - Some memory region keys
  - RMA
- **Doesn’t use some fancier features**
  - Scalable endpoints
  - Shared contexts
  - Auto progress

Could be better/worse

HIERARCHY – NO USER CHOICE (UNKNOWN MINIMAL PROVIDER)

- **Don’t know about the provider and doesn’t support default features**
- **Everything falls back to active messages**
- **Minimal requirements from libfabric**
  - FI_MSG
  - FI_MULTI_RECV
  - FI_RMA

Didn’t crash

LOTS OF CAPABILITIES DETECTED

- **AV_TABLE/AV_MAP**
- **Scalable Endpoints**
- **Shared Context**
- **Memory Region Bits**
  - VIRT_ADDR
  - PROV_KEY
  - ALLOCATED
- **FI_TAGGED**
- **FI_MSG**
- **FI_MULTI_RECV**
- **FI_DELIVERY_COMPLETE**
- **FI_ATOMICS**
- **AUTO_PROGRESS**
- **FI_TRIGGERED**
- **Library and provider versions**
- **FI_CONTEXT/FI_CONTEXT2**
- **MPICH-Specific combinations**

PROS / CONS

- 👍 **One library supports almost all interconnects (with shared memory up and coming).**
- 👍 **Moves all of the network-specific code (PSM2, Portals, etc.) out of MPICH**
- 👍 **Provides lots of customization options for users**

- 👎 **Latest and greatest hardware implements “optional” features.**
  - All providers don’t implement all features, requiring fallback code.
- 👎 **Keeping the capability sets up to date is a manual and error prone process.**
- 👎 **The startup code for MPICH became immensely complicated.**

CALL TO ACTION

- **Can utility providers help narrow the feature gap without loss of performance?**
- **How do applications handle features that *require* hardware support or will have bad performance?**
- **Could libfabric expose more information at compile time (e.g. through dynamically generated headers) to optimize application builds?**

OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

THANK YOU

Wesley Bland and Hajime Fujita

Intel Corporation

15th ANNUAL WORKSHOP 2019

OPEN MPI USE OF OFI LIBFABRIC

Howard Pritchard

Los Alamos National Laboratory

[March, 2019]

LA-UR 19-22193

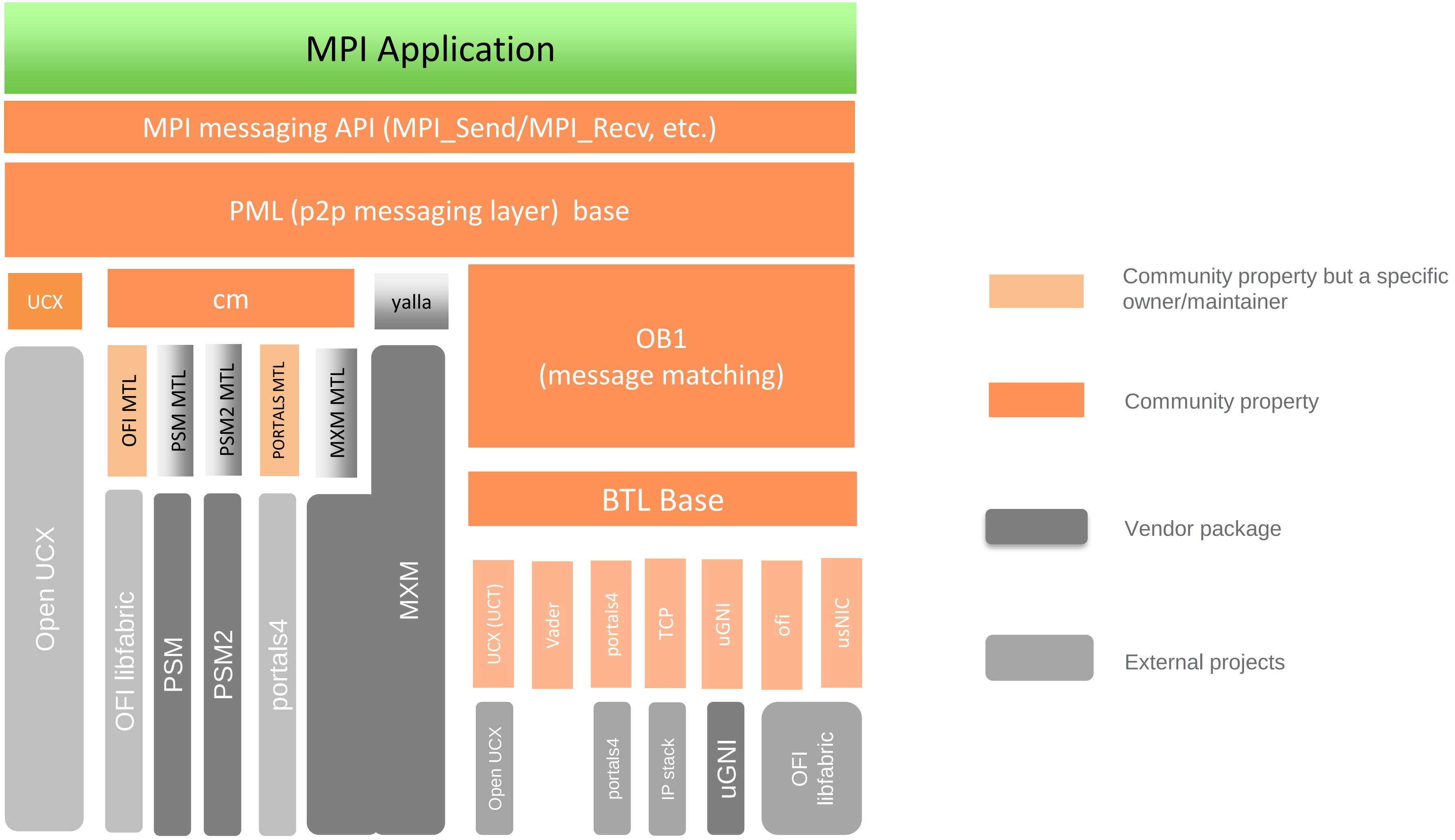
ACKNOWLEDGEMENTS

- **Most of development work for Open MPI’s OFI MTL and BTL components have been done by the following developers**
  - Aravind Gopalakrishnan
  - Neil Spruit
  - Matias A. Cabral
  - Thananon Patinyasakdikul

OUTLINE

- Open MPI's point to point messaging framework (PML) use of OFI libfabric
- Open MPI's RDMA byte transport framework (BTL) use of OFI libfabric
- Issues encountered using OFI libfabric
- Future work

OPEN MPI MESSAGING LAYER AND OFI LIBFABRIC



OPEN MPI'S USE OF LIBFABRIC – OFI MTL

- **Release series 3.0.x and 3.1.x**
  - FI_VERSION requested 1.0 or newer
  - Excluded vs included providers – excluded shm,sockets,tcp,udp,rstream
  - Endpoint types – FI_EP_RDM
  - Capabilities requested - FI_TAGGED
  - Don't request memory registration model (not needed for messaging)
- **Release series 4.0.x**
  - FI_VERSION requested 1.0 or newer
  - Excluded vs included providers – excluded shm,sockets,tcp,udp,rstream
  - Endoint types – FI_EP_RDM
  - Capabilities requested – FI_TAGGED, optional FI_DIRECTED_RECV and FI_RX_CQ_DATA (related to tag match expansion, we could use more than the 64 bit tag provided by libfabric API)
- **In the works (only on master)**
  - Same as 4.0.x but additionally –
  - Optional use of Scalable Endpoints (SEP)
  - Specialized function generation

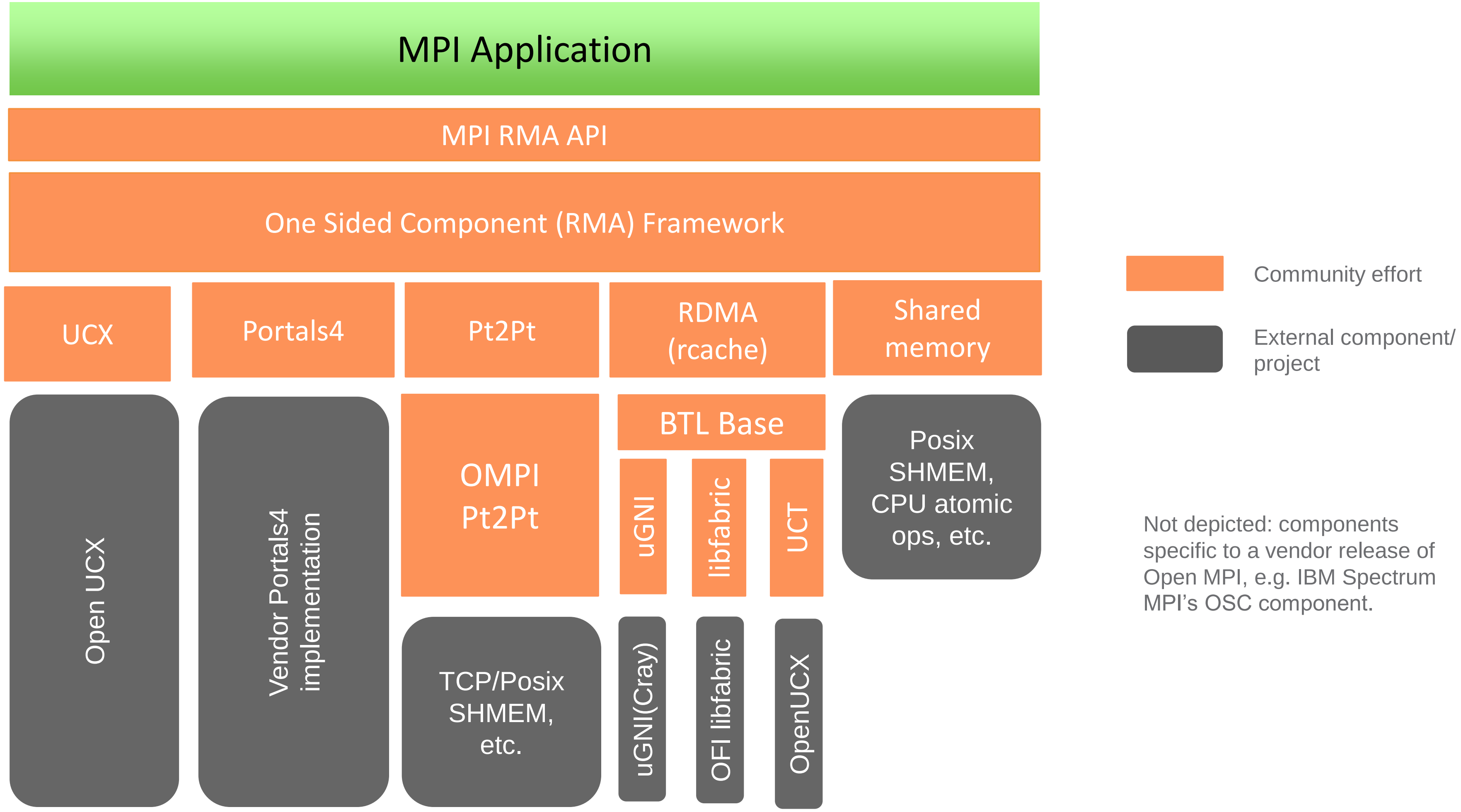
OFI MTL – TAG MATCHING IN 4.0.X

- **AUTO (default)**
  - Open MPI checks whether the selected OFI provider supports FI_REMOTE_CQ_DATA and FI_DIRECTED_RECV and if yes and user hasn't specified other tag format, use **ofi_tag_full**
- **ofi_tag_1**
  - Uses OFI 64 bit tag for source rank, CID, and MPI tag
  - 12 bits for CID, 18 bits for source rank, 32 bits for MPI tag
- **ofi_tag_2**
  - Like **ofi_tag_1** except 24 bits for CID, 18 bits for source rank, and 20 bits for MPI tag
- **ofi_tag_full**
  - Requires that provider support FI_RX_CQ_DATA and FI_DIRECTED_RECV
  - Source MPI rank available in the RX CQE is used
  - OFI tag bits are divided between 28 bits for CID, 32 bits for MPI tag. (may less if provider uses some tag bits)
- **Controlled via OMPI_MCA_ mtl_ofi_tag_mode environment variable (or mca parameter on mpirun command line)**

OFI MTL – SCALABLE ENDPOINTS

- **Targeting better mixed mode MPI+threads applications**
- **Disabled by default since in some cases regular FI_EP_RDM performs better**
- **When a new MPI communicator is created**
  - Check if number of tx/rx contexts created so far is less than max rx_ctx_cnt/tx_ctx_cnt
  - If yes, create new tx/rx context pair
  - Assign to the MPI communicator
  - Otherwise use round robin approach to assign MPI communicator to previously created tx/rx context pair
- **Need to set following MCA parameters (using environment variable method)**
  - `export OMPI_MCA_mtl_ofi_enable_sep=1`
  - `export OMPI_MCA_mtl_ofi_thread_grouping=1`
  - Optionally can request number of tx/rx/contexts to use:
    - `export OMPI_MCA_mtl_ofi_num_ctxts=N`
- **Progress**
  - Thread first progresses its context
  - If no CQ entries on thread’s libfabric CQs, progress other threads’ contexts.

OPEN MPI – MPI RMA (ONE-SIDED) AND OFI LIBFABRIC



Not depicted: components specific to a vendor release of Open MPI, e.g. IBM Spectrum MPI's OSC component.

OSC FRAMEWORK RDMA COMPONENT

- **Many enhancements over available MPI one-sided support in the 2.0.x and 2.1.x series**
- **Driven partly by ECP OMPI-X effort**
- **Emphasis on MPI-RMA performance for multi-threaded applications**
  - Lock scaling improvements for MPI_Win_lock_all
  - Much improved registration cache optimized for concurrent reads
    - Replaced with interval tree, relativistic ordering, there is now only a write lock to do node insertion, rotation, or deletion, so typical read-only lookup's are fast
    - Based on work by *P. W. Howard and J. Walpole(2014), Relativistic red-black trees, Concurrency Computat.: Pract. Exper., 26, pages 2684–2712.*
- **Improved scalability of memory use for MPI Windows**
- **Uses Open MPI BTL's that support RMA (fi_write, fi_read, fi_atomic, etc.) operations**
- **Decided it would be easier to write an OFI BTL rather than undertake writing an OFI OSC component**
- ***See N. Hjelm, et al. Improving MPI Multi-threaded RMA Communication Performance, ICPP 2018 Proceedings of the 47th International Conference on Parallel Processing***

OPEN MPI'S USE OF LIBFABRIC – OFI BTL

- **Only available in master**
  - FI_VERSION requested 1.5 or newer
  - No explicit exclusion of providers
  - Endpoint types – FI_EP_RDM
  - Capabilities requested - FI_RMA | FI_ATOMIC (optional FI_MSG), also FI_DELIVERY_COMPLETE
  - mr_mode - FI_MR_ALLOCATED | FI_MR_PROV_KEY | FI_MR_VIRT_ADDR
- **Makes use of scalable EP's if provider supports them**
- **Unfortunately doesn't work with OFI MTL for certain OFI providers, PSM2 provider in particular (providers that have limited ability to support multiple initializations are problematic)**

CHALLENGES USING OFI LIBFABRIC

- **The number of potential paths to support for functions in the critical path for message transmit/receive, etc. grows quickly as more libfabric functionality is optionally used**
  - Partially solved by specialized function generation (OFI MTL)
- **Some providers have characteristics that make life more difficult**
  - Performance of fi_send/fi_recv can be impacted by whether or not FI_RMA/FI_ATOMIC was requested as a capability
  - Some providers don’t support multiple fi_domain and/or fi_endpoint very well, resulting in problems having separate OFI MTL and OFI BTL components

FUTURE WORK

- Consider implementing a standalone OFI OSC component
- Consider implementing a OFI PML component (reduce dependence on CM infrastructure)
- Persistent memory awareness

OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

THANK YOU

Howard Pritchard

Los Alamos National Laboratory

OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

LIBFABRIC AND OPENSHMEM

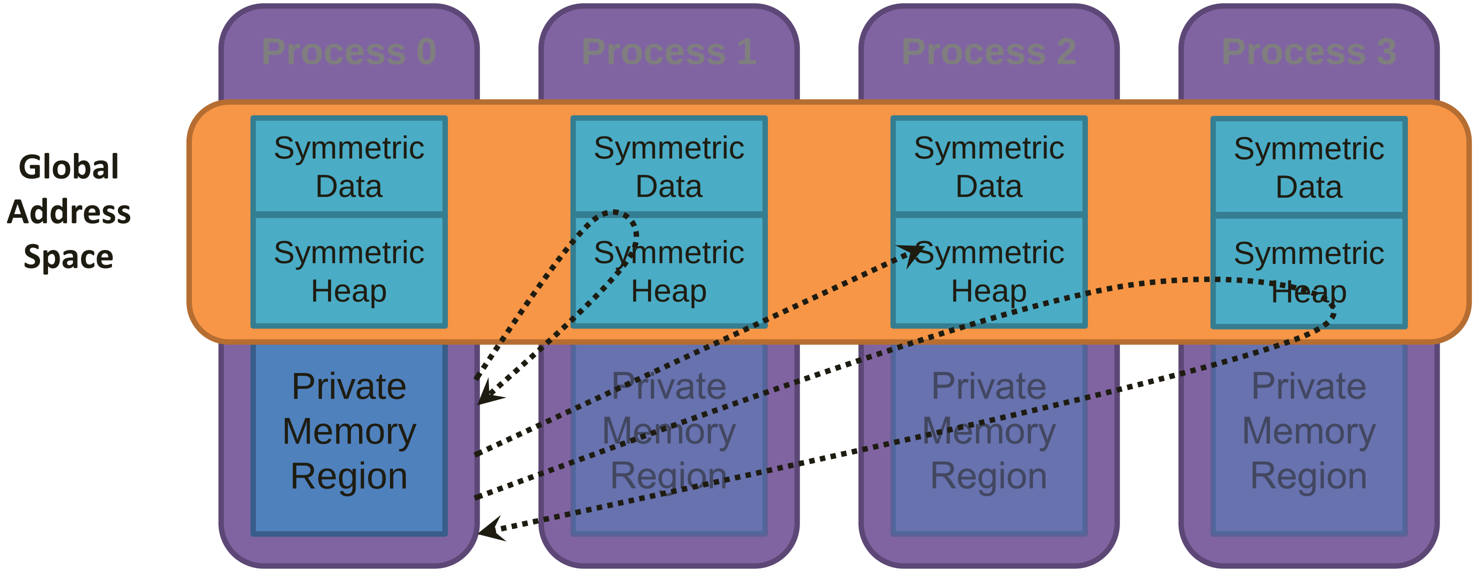
James Dinan

Intel Corporation

[March 21, 2019]

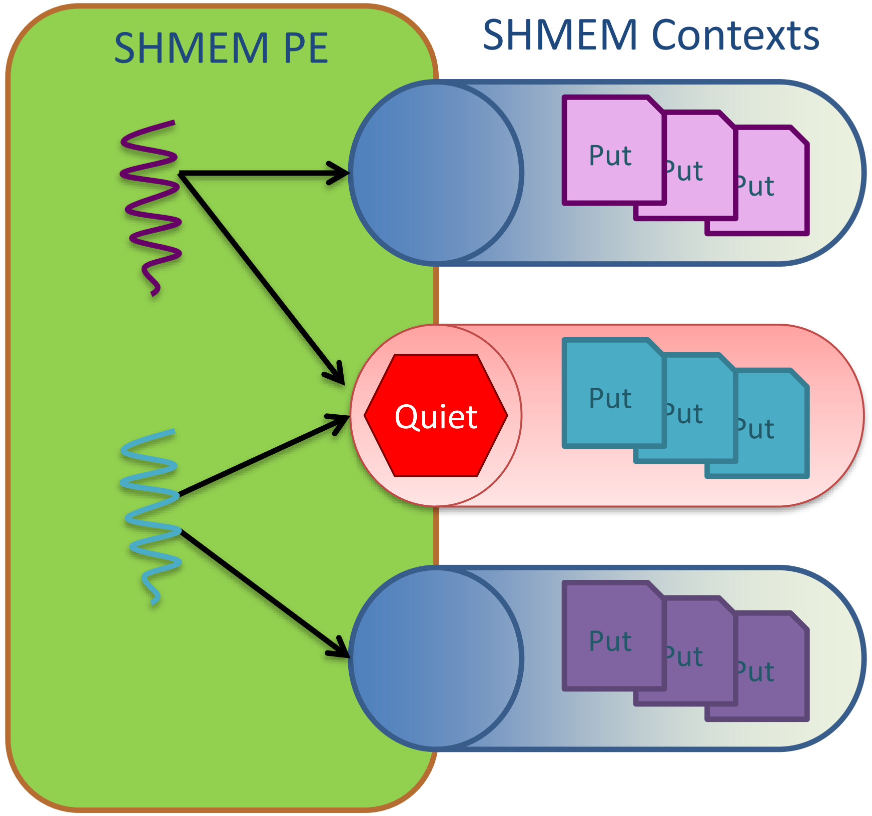
NOTICES AND DISCLAIMERS

WHAT IS OPENSHMEM?



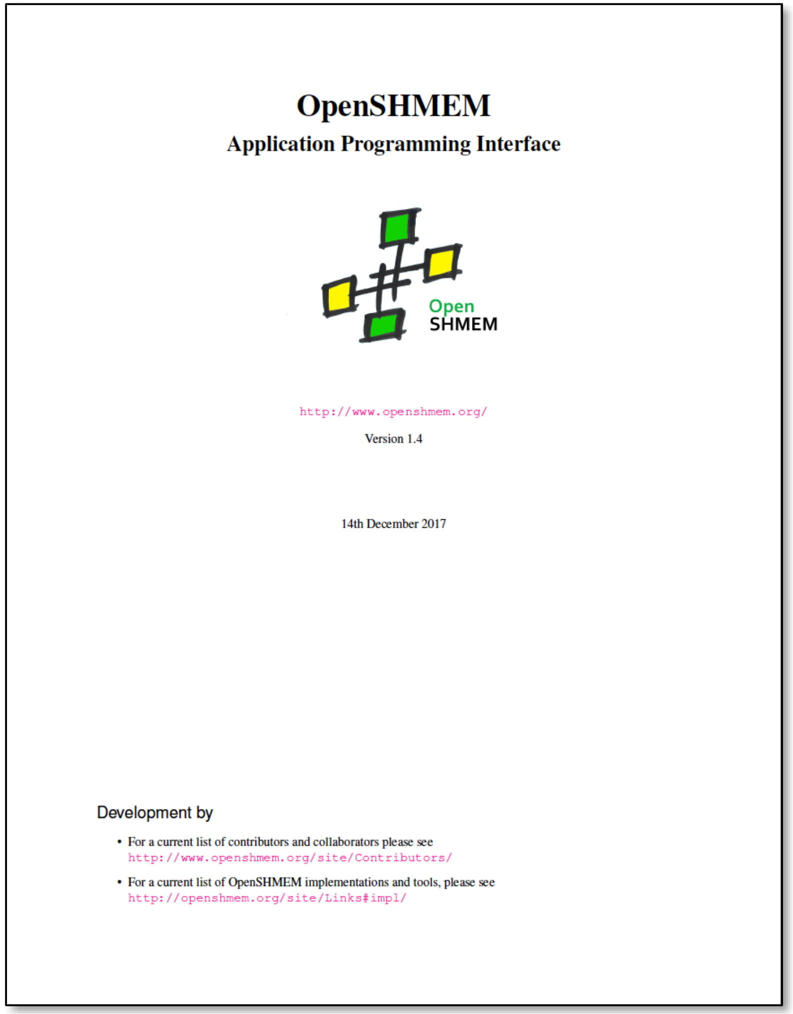
- **Partitioned Global Address Space (PGAS) memory model, SPMD execution**
- **Part of the memory in a process is exposed for remote access**
  - Remotely accessible memory is “symmetric”; it has the same size and layout at each process
  - Asynchronous read (get), write (put), and atomic update operations
- **Fence (ordering), quiet (remote completion), barrier (global sync), wait (pt-to-pt sync)**

OPENSHMEM AND THREADS: A UNIQUE COMMUNICATION MODEL



- **Library thread safety levels similar to MPI**
- **Communication is performed on a context**
- **Enables programmer to choose which operations are completed by quiet or ordered by fence**
  - Application-level communication/computation overlap
  - Eliminate interference between threads
- **Contexts can be private or shared**
  - Private, only usable by thread that created the context
  - Shared, usable by any thread
  - Want to manage OFI TX contexts to optimize private/shared
- **OpenSHMEM defines a “default” context**
  - Legacy API calls are tied to this context
  - It’s always shared

OPENSHMEM 1.4



OpenSHMEM
Application Programming Interface

Open SHMEM

http://www.openshmem.org/
Version 1.4

14th December 2017

Development by

- For a current list of contributors and collaborators please see http://www.openshmem.org/site/Contributors/
- For a current list of OpenSHMEM implementations and tools, please see http://openshmem.org/site/Links#impl/

- **Open standard for the SHMEM programming model**
- **Specification ratified Dec. 14, 2017**
  - Thread safety
  - Communication management API (contexts)
  - Test (pt-to-pt sync), sync (barrier), calloc (sym. memory)
  - Bitwise atomic operations
  - C11 generic selection bindings

OpenSHMEM

Application Programming Interface

Open SHMEM

http://www.openshmem.org/

~~Version 1.4~~

Version 1.5

~~14th December 2017~~

November, 2019

Development by

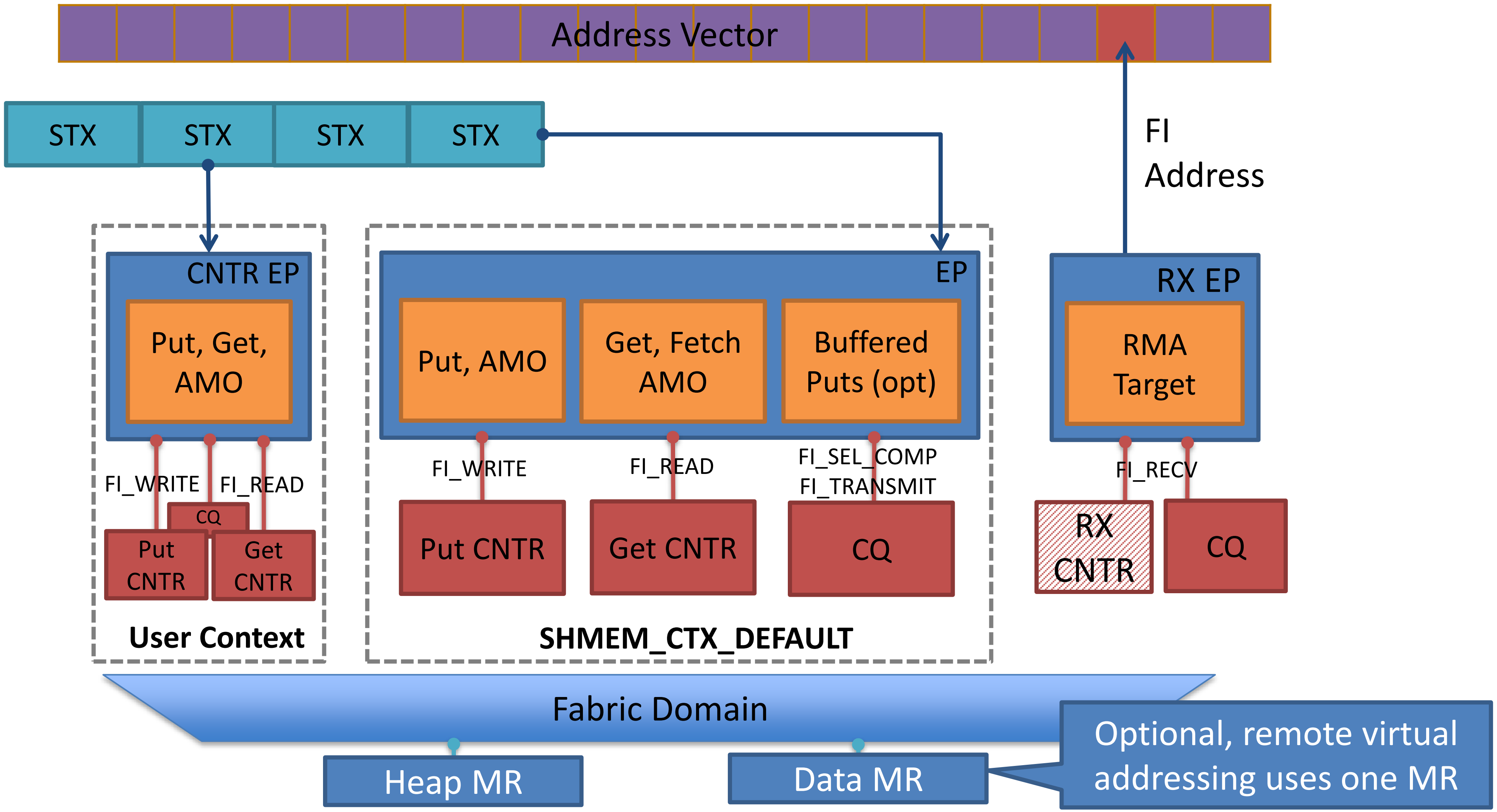
- For a current list of contributors and collaborators please see http://www.openshmem.org/site/Contributors/
- For a current list of OpenSHMEM implementations and tools, please see http://openshmem.org/site/Links#impl/

- **Hard at work on OpenSHMEM 1.5**
  - Add teams and refresh collectives API
    - Enable collectives acceleration
    - E.g. using OFI triggered operations
  - Improve point-to-point synchronization
    - Add test/wait – some/any/all
    - Add put-with-signal API (uses FI_FENCE)
  - Nonblocking atomic operations
  - Profiling interfaces
  - Memory model work to improve performance portability

- **Additional topics under discussion**
  - Support for symmetric objects in different memory kinds
  - Interoperability with MPI and other hybrid models
  - *Join us and add your topic here!*

- **Intel engaged in Sandia OpenSHMEM development**
  - Open source, supports Intel® Omni-Path Fabric
  - Supports OFI and Portals 4, first to support OpenSHMEM 1.3 and 1.4
  - https://github.com/Sandia-OpenSHMEM/SOS

SANDIA OPENSHMEM 1.4.X OFI TRANSPORT ARCHITECTURE



PROVIDER REQUIREMENTS TO SUPPORT OPENSHMEM

- **Reliable, connectionless communication model: FI_EP_RDM**
- **Communication operations: FI_RMA | FI_ATOMIC**
  - Require FI_INJECT for performance of scalar put and AMO, want FI_FENCE for put-with-signal operations
  - Collectives operate on symmetric buffers and can use RMA/ATOMIC (important difference from MPI)
    - Some implementations may use FI_TAGGED to share code with MPI (SOS currently does not)
- **Endpoint TX/RX are treated separately; RX at process level, TX at ctx/thread level**
  - STX (optional) allows SOS to optimize mapping of threads to TX resources
- **OpenSHMEM requires an asynchronous, “passive” target**
  - Ideally, FI_PROGRESS_AUTO without a progress thread
  - Can supplement with in-line progress, working on progress thread to support RXM
- **FI_THREAD_SAFE is good, in principle**
  - FI_THREAD_COMPLETION can provide better thread isolation, eliminating locking for private contexts
- **Memory registration: FI_MR_SCALABLE is efficient, no need to store/lookup keys**
  - Also support FI_MR_BASIC and FI_RMA_EVENT. Need to support others?

DISCUSSION

1. **OpenSHMEM *really* does require asynchronous progress**
   - Most applications have high messaging rates with small messages
   - On-loading auto (asynchronous) progress works for some communication patterns (benchmarks), but few apps
   - Inline manual progress works for even fewer communication patterns
2. **Corollary: On-loading atomics is a challenge for OpenSHMEM**
   - OpenSHMEM atomics are single-element and do not require atomicity across datatypes (e.g. int, long, etc.)
   - Hardware atomics only on 64-bit data with fetch-add and CAS still covers performance critical cases
   - Middleware should control whether to emulate atomics using HW or to use SW
   - User will choose tradeoff between latency of atomics and dedicating cores to on-loading
3. **Libfabric atomicity and memory model are underspecified**
   - Atomic across all datatypes and ops? Interaction with Read/write? What about different EPs on the same domain?
   - How can a process portably poll memory for an atomic update or write? FI_ATOMIC_READ … or?
   - How does a process ensure memory is consistent, e.g. flush an atomics cache in the NIC?
4. **Want to use collectives acceleration, including switch-based acceleration**
5. **Want one thread per core to provide the same performance as one process per core**
   - Scale up resources within a process and drive with multiple threads and low overheads

OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

THANK YOU

James Dinan

Intel Corporation