14th ANNUAL WORKSHOP 2018

STATUS OF OFI SUPPORT IN MPICH

Yanfei Guo, Assistant Computer Scientist

Argonne

[April 11, 2018]

OUTLINE

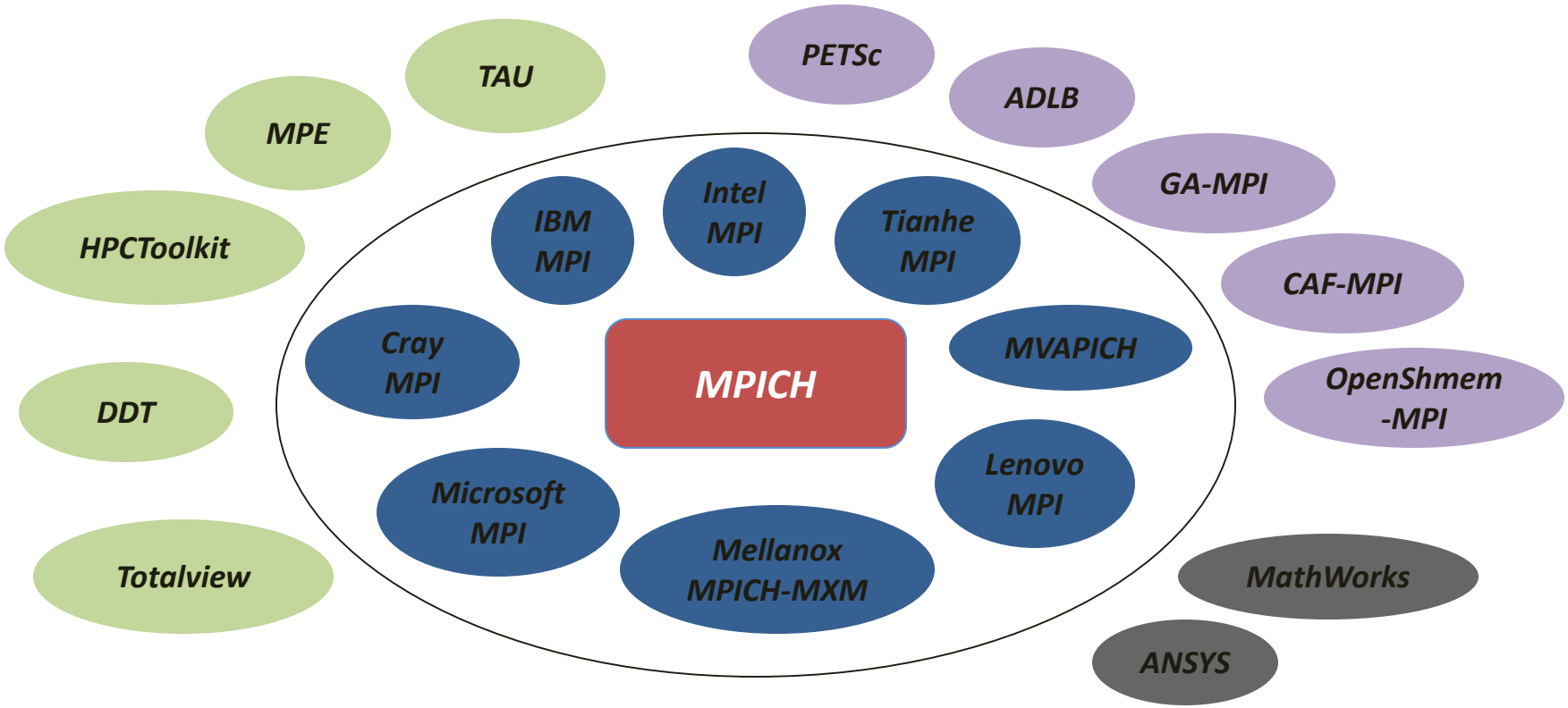
- **What is MPICH?**
- **Why OFI?**
- **Current support**
  - MPICH 3.2 series
  - MPICH 3.3 series (CH4)
- **Ongoing work**
  - Scalable Endpoints
  - Collectives

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"**



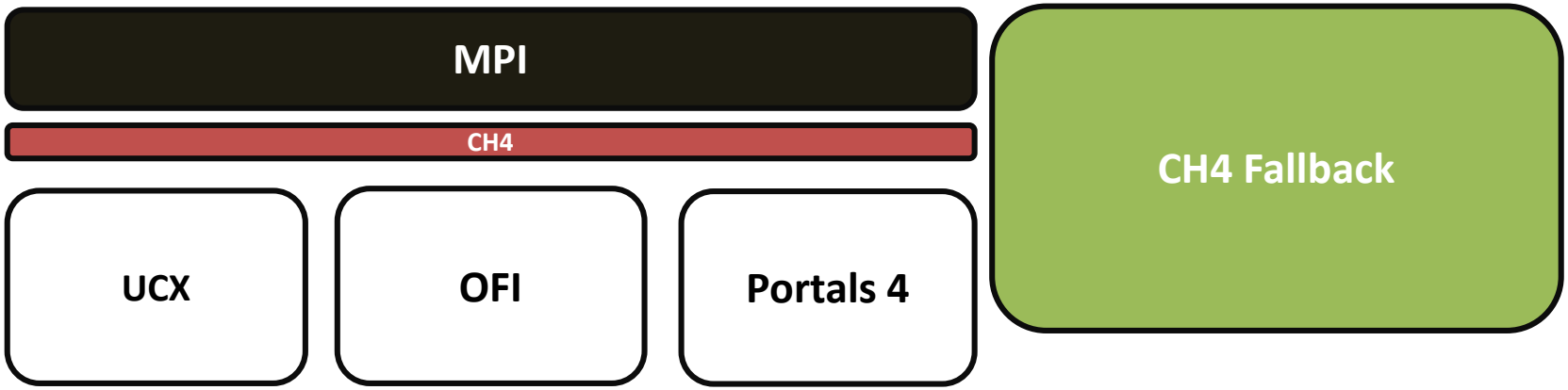
MOTIVATION

- **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

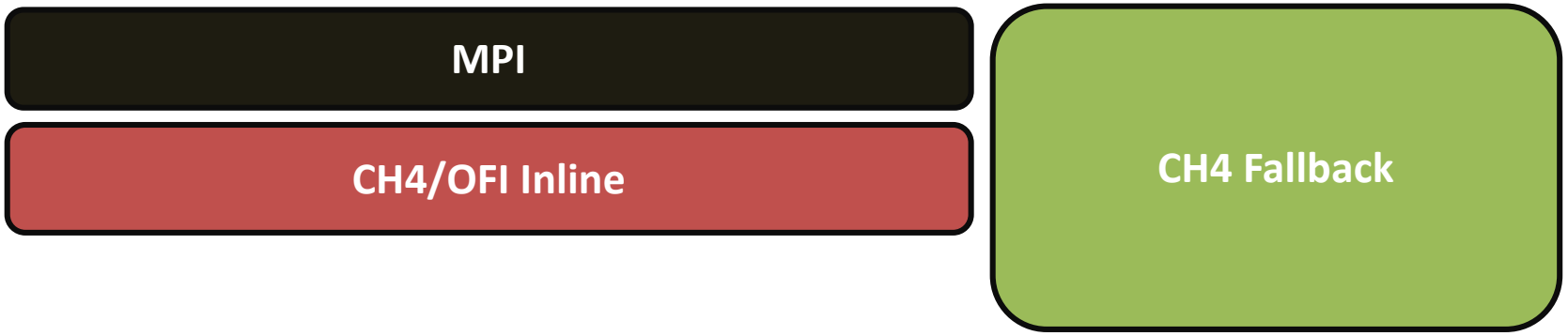
MPICH-3.3 SERIES

- **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



REDUCING OVERHEAD



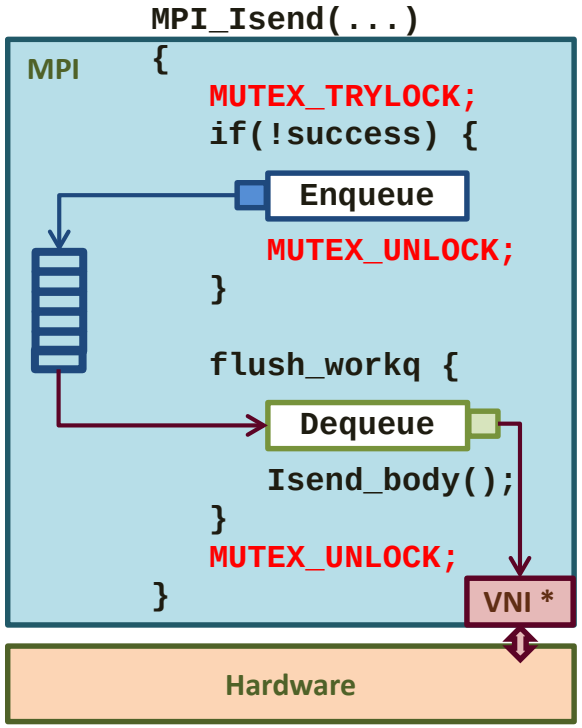
PERFORMANCE BOTTLENECK FOR MPI+THREAD

- **Current MPICH code**
  - Context
    - MPICH **unconditionally** acquires locks on critical paths
    - **Nonblocking** operations may **block** for a lock acquisition
    - **Not** truly nonblocking!
  - Consequences
    - Nonblocking operations may be slowed by blocking ones from other threads
    - Pipeline stalls: higher latencies, lower throughput, and less communication-computation overlapping

```
MPI_Isend(...) Nonblocking MPI call
{
  MUTEX_LOCK; /* Potentially blocking */
  Isend_body(); /* Interruptible */
  MUTEX_UNLOCK;
}
```

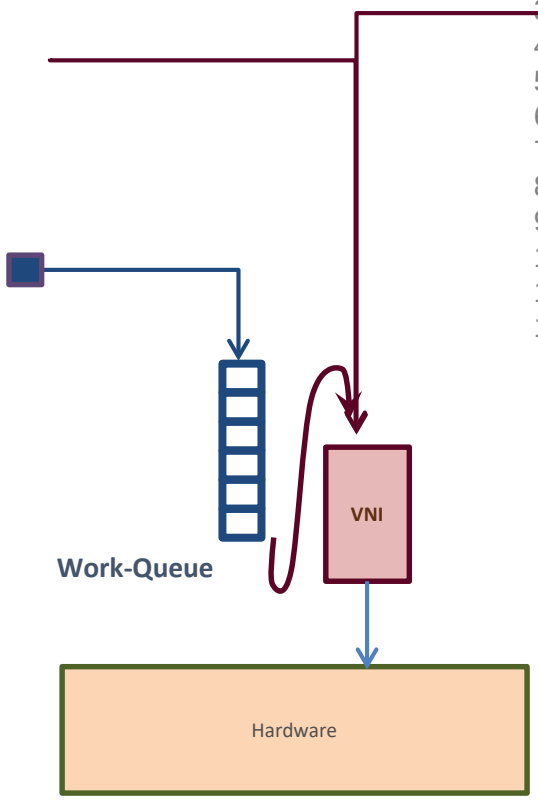
WORK-QUEUE MODEL

- **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



WORK-QUEUE MODEL

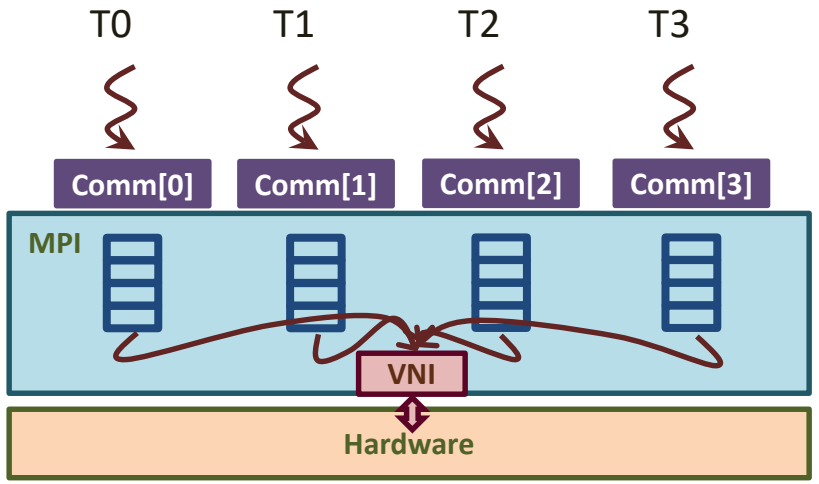
```
0  MPI_Isend (…,&req) {
1
2
3
4  lock_tryacquire(L, success);
5
6  if (success) {
7
8     flush_workq();
9     request_create(req);
10    network_isend(req);
      lock_release(L);
    } else {
      create_workq_desc(&d);
      workq_enqueue(d);
    }
  }
```




```
0  MPI_Wait (…,&req) {
1     lock_acquire(L);
2     while (!completed(req)) {
3        flush_workq();
4        network_progress();
5        if (!completed(req)) {
6           lock_release(L);
7           thread_yield();
8           lock_acquire(L);
9        }
10    }
11    free(req);
12    req = REQUEST_NULL;
      lock_release(L);
  }
```

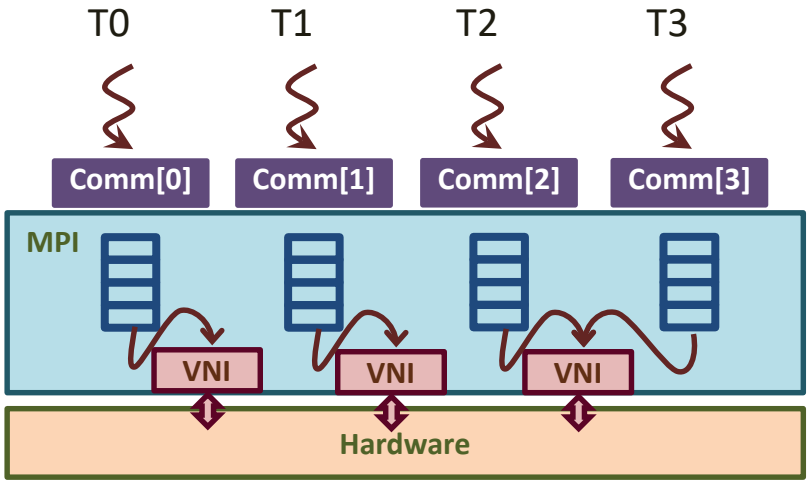
MULTIPLE WORK-QUEUES

- **Multiple isolated work-queues**
  - Transparent to the user
  - E.g. one Work-Queue per communicator, per neighbor process (regular apps)
- **Concurrency can be improved if the user program maximizes independence between threads (i.e., different communicator, peer_rank, or tag per thread)**

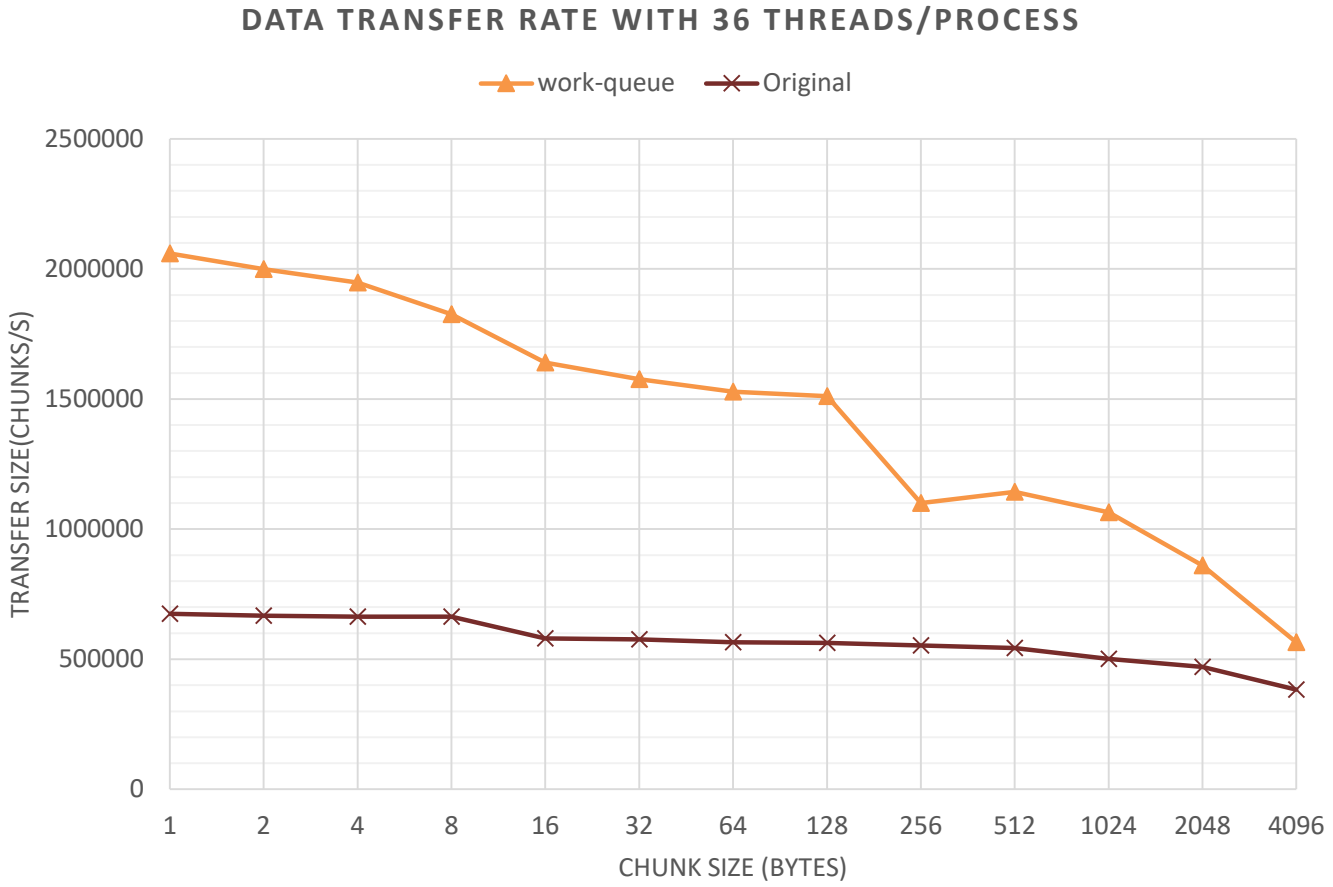


MULTIPLE VIRTUAL NETWORK INTERFACE (VNI)

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



PRELIMINARY RESULTS



Multithreaded MPI_PUT with 36 threads per MPI process between two Haswell nodes interconnected with a Mellanox QDR fabric

MPICH-3.3 ROADMAP

- **CH4 already in at http://github.com/pmodels/mpich**
- **MPICH-3.3b2 release has just come out**
  - MPICH-3.3b3 next month
- **Work-Queue Comes in 3.3.b3 release**
- **Multi-VNI is planned for 3.3 GA release**
- **GA Release Summer 2018**

14th ANNUAL WORKSHOP 2018

THANK YOU

Yanfei Guo, Assistant Computer Scientist

Argonne National Laboratory