Argonne NATIONAL LABORATORY

Next Generation MPICH: What to Expect - Lightweight communication and much more!

Ken Raffenetti

Software Development Specialist

Argonne National Laboratory

Email: raffenet@mcs.anl.gov

Web: http://www.mcs.anl.gov/~raffenet

U.S. DEPARTMENT OF ENERGY

Outline

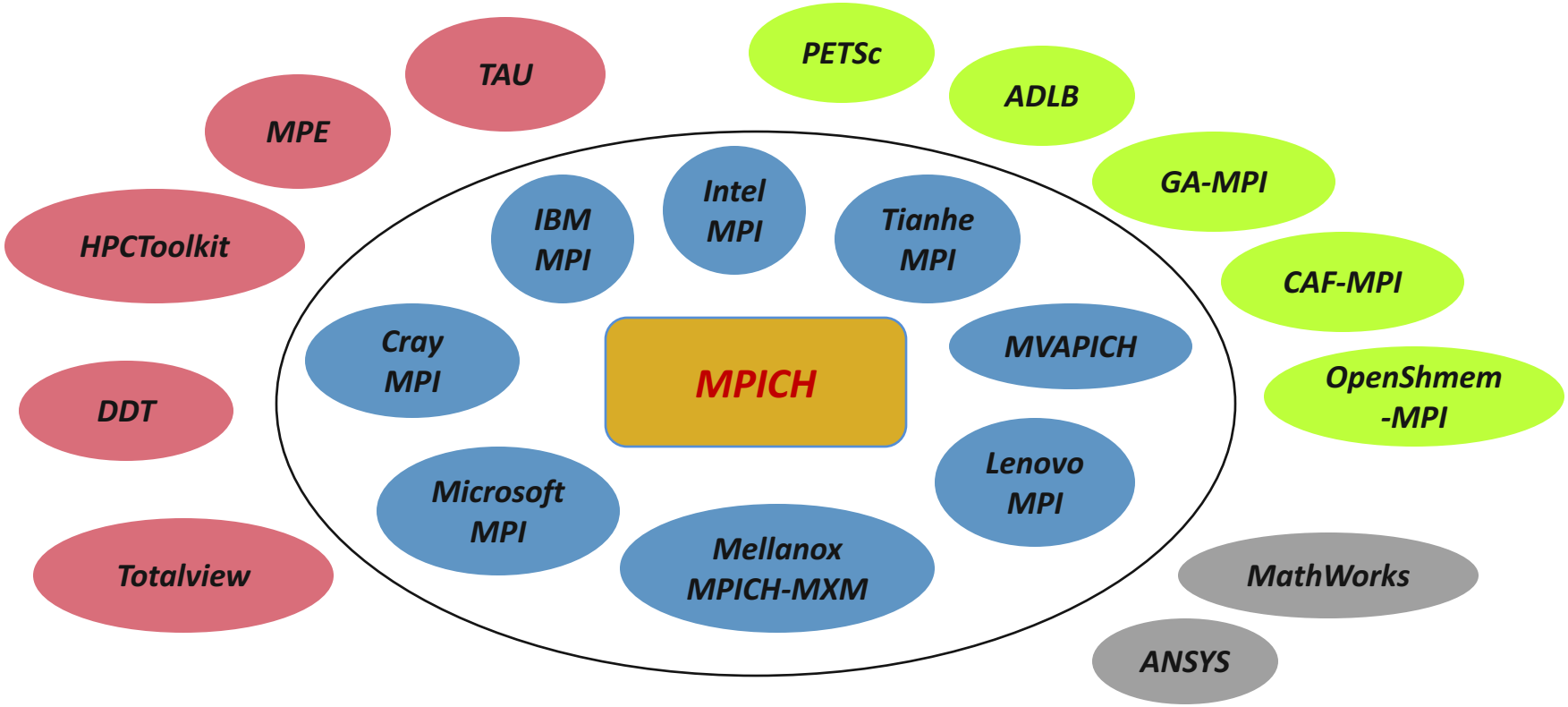
- Current MPICH
- MPICH-3.3 and beyond
  - Lightweight communication overhead
  - Memory scalability
  - Multi-threading
    - MPICH + User-level threads
  - ROMIO data logging
  - MPI derived datatypes
- Summary

MPICH Today

- 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
- www.mpich.org

MPICH: Goals and Philosophy

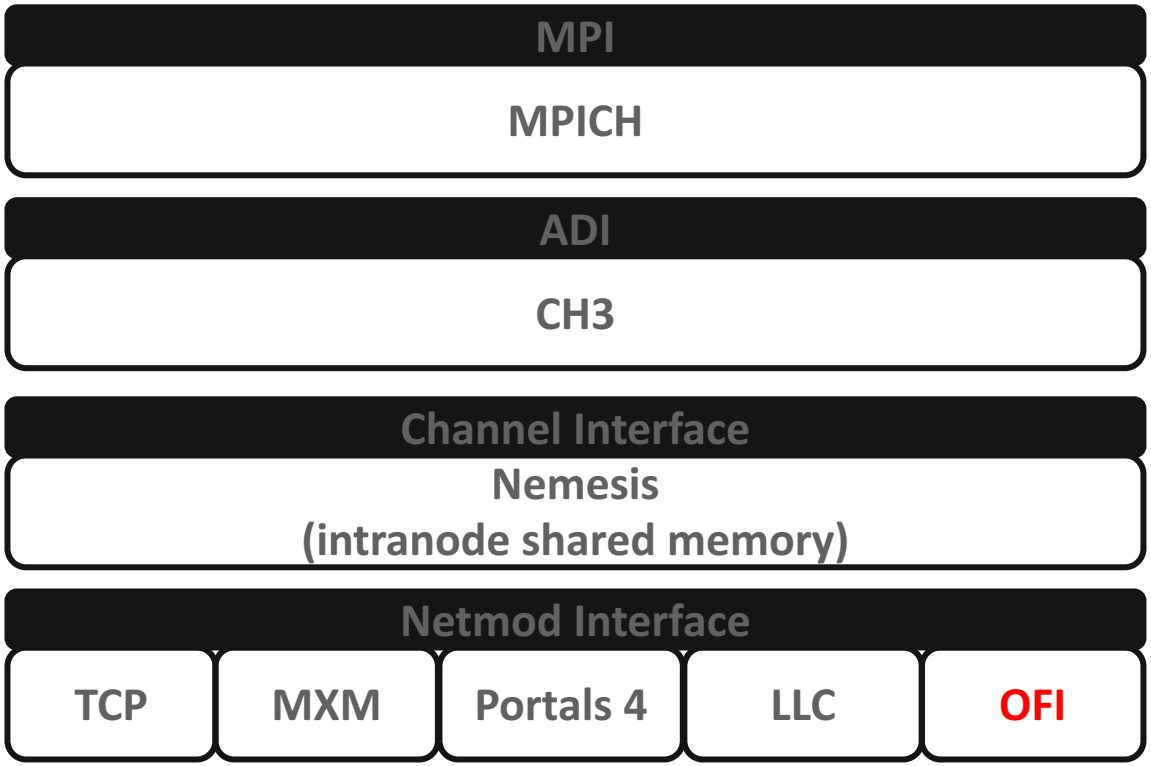
- MPICH aims to be the preferred MPI implementation on the top machines in the world
- Our philosophy is to create an "MPICH Ecosystem"



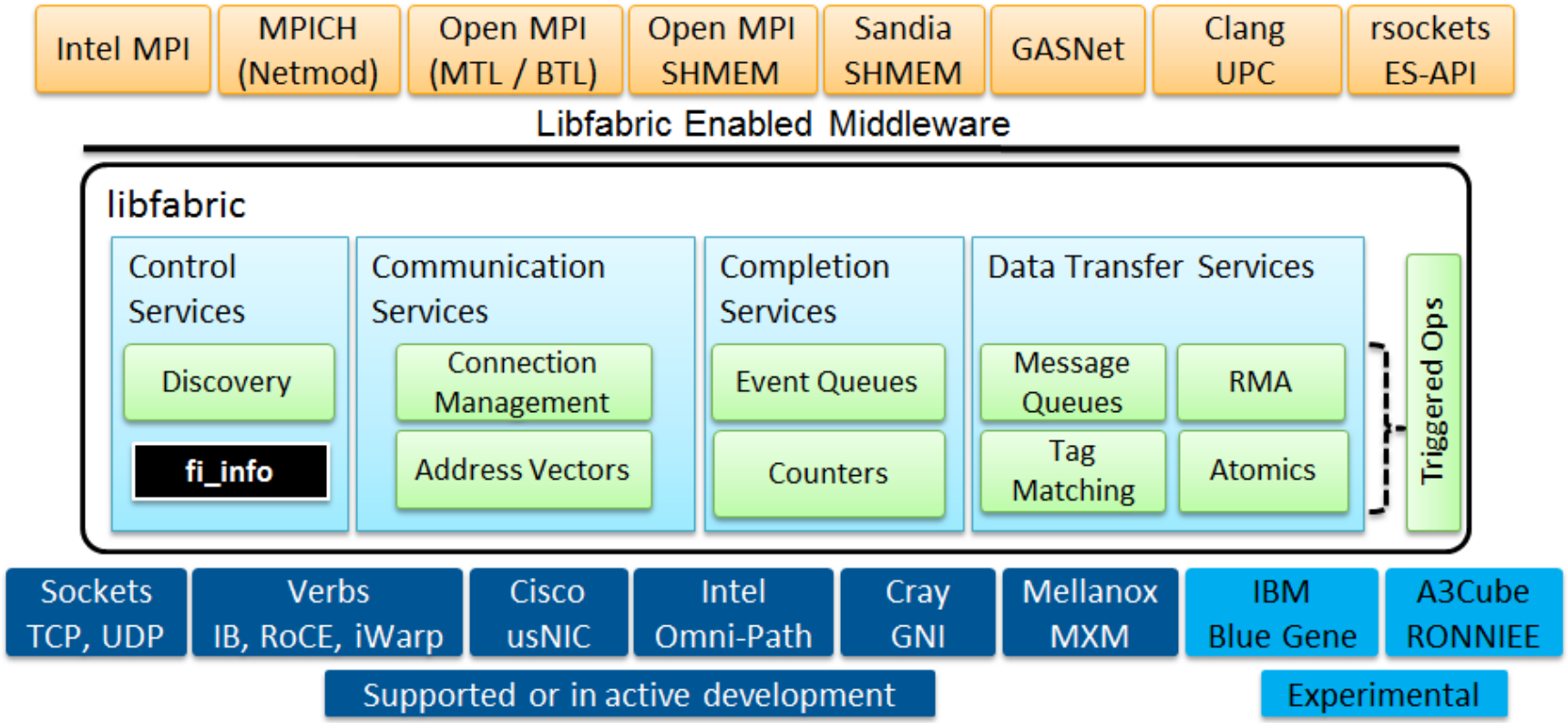
MPICH-3.2

- MPICH-3.2 is the latest major release series of MPICH
- Primary focus areas for mpich-3.2
  - Support for MPI-3.1 functionality (nonblocking collective I/O and others)
  - Fortran 2008 bindings
  - Support for the Mellanox MXM interface (thanks to Mellanox)
  - Support for the Mellanox HCOLL interface (thanks to Mellanox)
  - Support for the LLC interface for IB and Tofu (thanks to RIKEN)
  - Support for the OFI interface (thanks to Intel)
  - Improvements to MPICH/Portals 4
  - MPI-4 Fault Tolerance (ULFM – experimental)
  - Major improvements to the RMA infrastructure

MPICH-3.2


MPI
MPICH
ADI
CH3
Channel Interface
Nemesis
(intranode shared memory)
Netmod Interface
TCP
MXM
Portals 4
LLC
OFI

OFI - Libfabric



OFI Netmod in CH3

- All of MPI over `fi_tagged`
  - Hardware Send/Recv
  - MPI RMA emulation using MPICH packet headers
  - MPICH control messages
- Where to improve?
  - MPI RMA with `fi_rma`
  - Collectives with `fi_trigger`
  - Would require major infrastructure changes to CH3
    - Step back and look at CH3 as a whole…

Outline

- Current MPICH
- Next Generation MPICH
  - Lightweight communication overhead
  - Memory scalability
  - Multi-threading
    - MPICH + User-level threads
  - ROMIO data logging
  - MPI derived datatypes
- Summary

CH3 Shortcomings

Netmod API

- Passes down limited information and functionality to the network layer
  - `SendContig`
  - `SendNoncontig`
  - `iSendContig`
  - `iStartContigMsg`
  - `...`

Active Message First Design

- All communication involves a packet header + message payload
  - Requires a non-contiguous memory access for all messages
- Send/Recv override exists, but was somewhat clunky add-in

Singular Shared Memory Support

- Performant shared memory communication centrally managed by Nemesis
- Network library shared memory implementations are not well supported
  - Inhibits collective offload

Function Pointers Not Optimized By Compiler

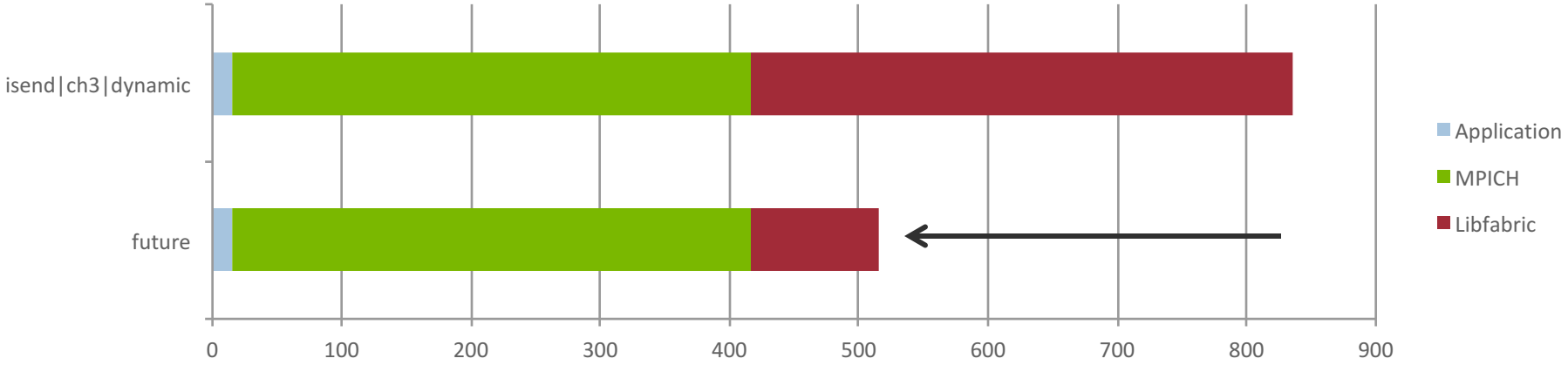
```
if (vc->comm_ops && vc->comm_ops->isend){
    mpi_errno =
        vc->comm_ops->isend(vc, buf, count, ...)
    goto fn_exit;
}
```

Non-scalable "Virtual Connections"

- 480 bytes * 1 million procs = 480MB(!) of VCs per process
- Connection-less networks emerging
  - VC and associated fields are overkill

Overheads

- With MPI features baked into next-generation hardware, we anticipate network library overheads will dramatically reduce.



- Message rate will come to be dominated by MPICH overheads

MPI on OFI

- Point-to-point data movement
  - Closely maps to fi_tsend/trecv functionality
  - How can MPICH get out of the way?

```
MPI_Isend(buf, count, datatype, dest, tag, comm, &req)

fi_tsend(gl_data.endpoint,        /* Local endpoint */
         send_buffer,             /* Packed or user */
         data_sz,                 /* Size of the send */
         gl_data.mr,              /* Dynamic memory region */
         to_addr(comm,dest),      /* Destination fabric address */
         match_bits(comm,tag),    /* Match bits */
         &req->ctx);              /* Context */
```

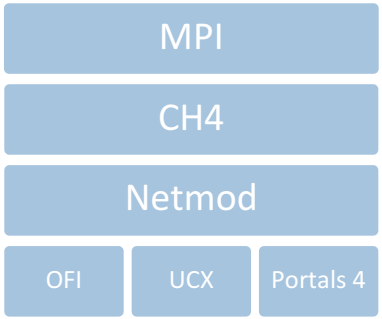
Addressing CH3's shortcomings

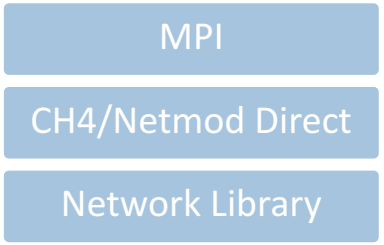
High-Level API

- Give more control to lower layers
  - `netmod_send`
  - `netmod_recv`
  - `netmod_put`
  - `netmod_get`
- Fallback to Active Message based communication when necessary
  - Operations not supported by the network

"Netmod Direct"

- Support two modes
  - Multiple netmods
    - Retains function pointer for flexibility
  - Single netmod with inlining into device layer
    - No function pointer





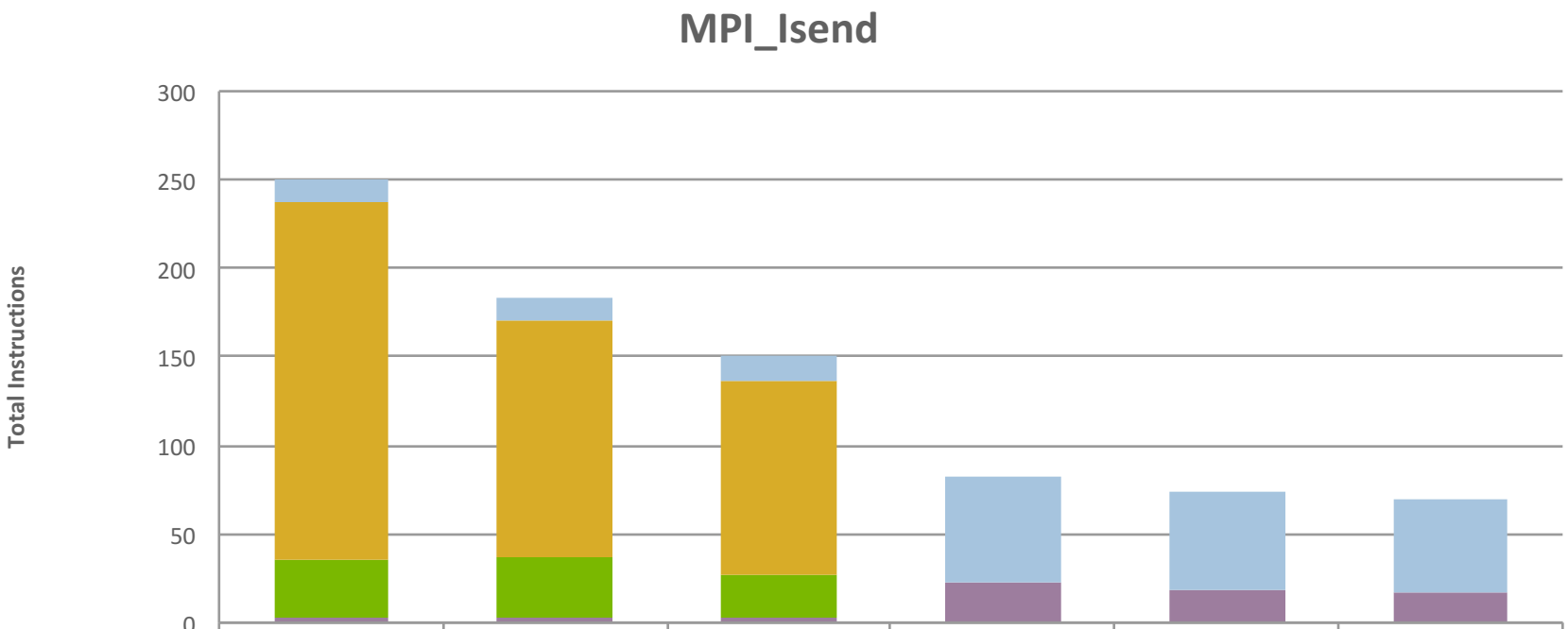
More configurable shared memory in CH4

- Involve the network layer in the decision
  - Support SHM aware algorithms
- One or more SHM transports (POSIX, XPMEM, CMA)

No Virtual Connection data structure

- Global address table (still O(p))
  - Contains all process addresses
  - Index into global table by translating (`rank+comm`)
- VCs can still be defined at the lower layers

MPI_Isend



	ch3-netmod-dynamic	ch4-netmod-base	ch4-netmod-inline	ch4-static	ch4-shm-disabled	ch4-thread-single
application-pre	13	13	13	61	55	53
mpi-pre	202	133	110	0	0	0
mpi-post	32	34	24	0	0	0
application-post	3	3	3	22	19	17

Outline

- Current MPICH Status
- MPICH-3.3
  - Lightweight communication overhead
  - Memory scalability
  - Multi-threading
    - MPICH + User-level threads
  - ROMIO data logging
  - MPI derived datatypes
- Summary

Overview of the New LPID/GPID Design
(Replacement for VC)

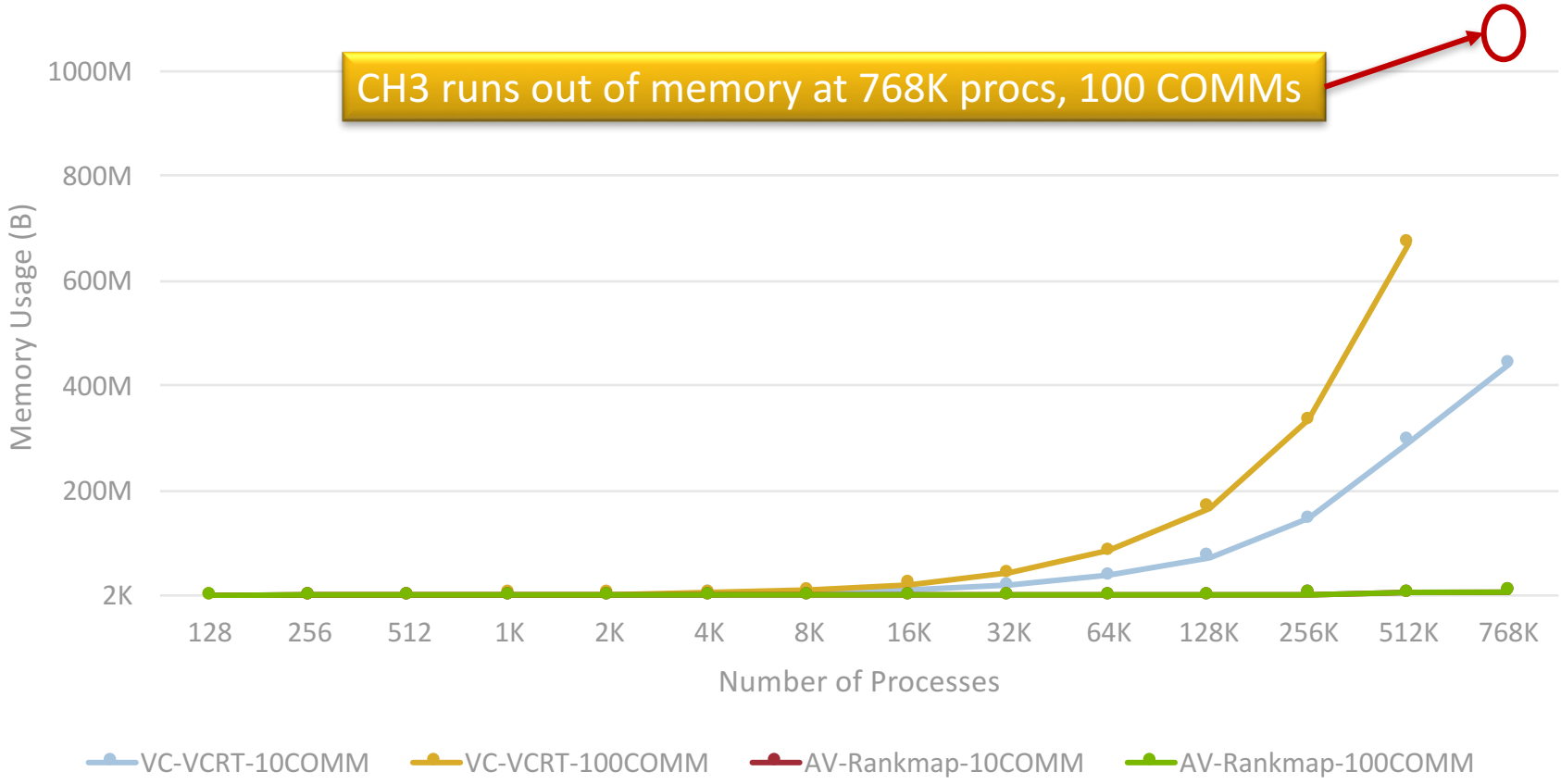
- Compressing VC (480Bytes -> 12Bytes)
  - Compressing Multi-transport Functionality
    - Function pointers are moved to a separate array
  - Deprioritizing Dynamic Processes
    - Process group information moved to COMM
- Regular/Irregular Rank Mapping Models
  - DIRECT/DIRECT_INTRA
  - OFFSET/OFFSET_INTRA
  - STRIDE/STRIDE_INTRA
  - STRIDE_BLOCK/STRIDE_BLOCK_INTRA
  - LUT/LUT_INTRA
  - MLUT
- Rank-Address Translate
  - (comm, rank) -> (avtid, lpid)

```
typedef struct {¬
    union {¬
        MPIDI_CH4_NETMOD_DEVADDR_DECL¬
    } netmod;¬
#ifdef MPIDI_BUILD_CH4_LOCALITY_INFO¬
    MPIDI_CH4I_locality_t is_local;¬
#endif¬
} MPIDI_CH4I_av_entry_t;¬
¬
typedef struct {¬
    MPIU_OBJECT_HEADER;¬
    int size;¬
    MPIDI_CH4I_av_entry_t table[0];¬
} MPIDI_CH4I_av_table_t;¬
¬
extern MPIDI_CH4I_av_table_t **MPIDI_CH4I_av_table;¬
extern MPIDI_CH4I_av_table_t *MPIDI_CH4I_av_table0;¬
```

All *_INTRA (avtid==0) model uses MPIDI_CH4I_av_table0 to save 1 memory reference during translation

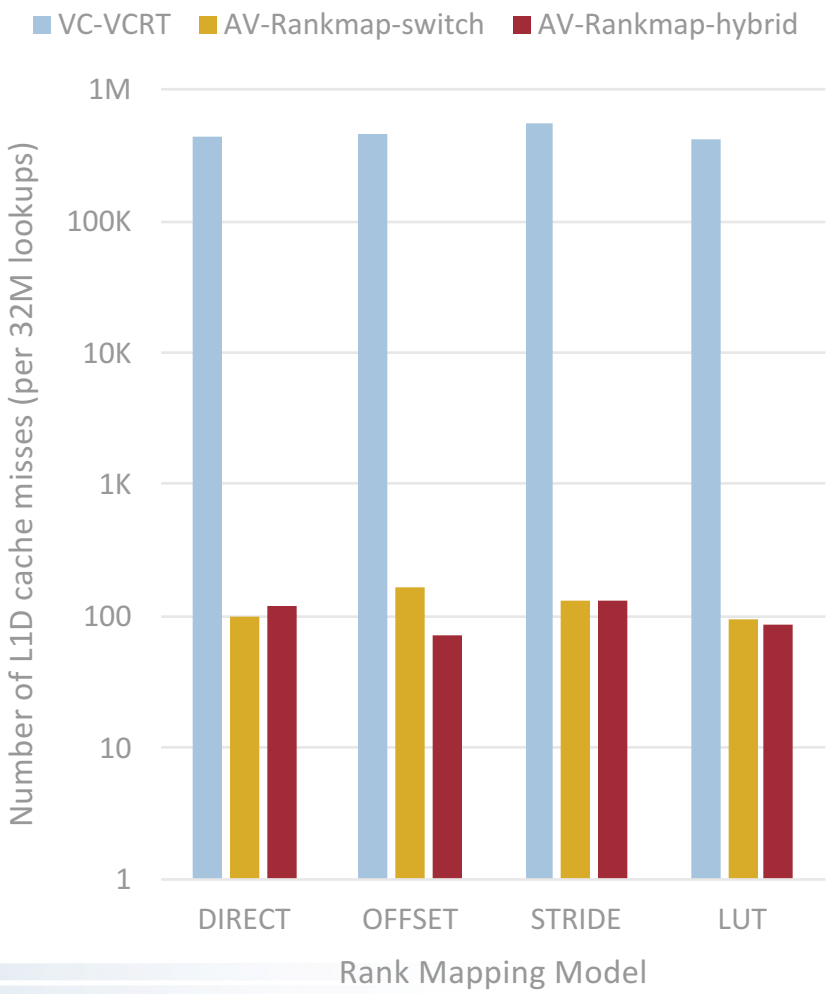
Memory Usage Reduction

MPI_Comm_split, 10 split COMM and 100 spilt COMM



L1D Cache Misses

- Compressing VC structure reduces the cache misses during communication
- Deduction in L1D cache misses compensated the overhead of additional instructions

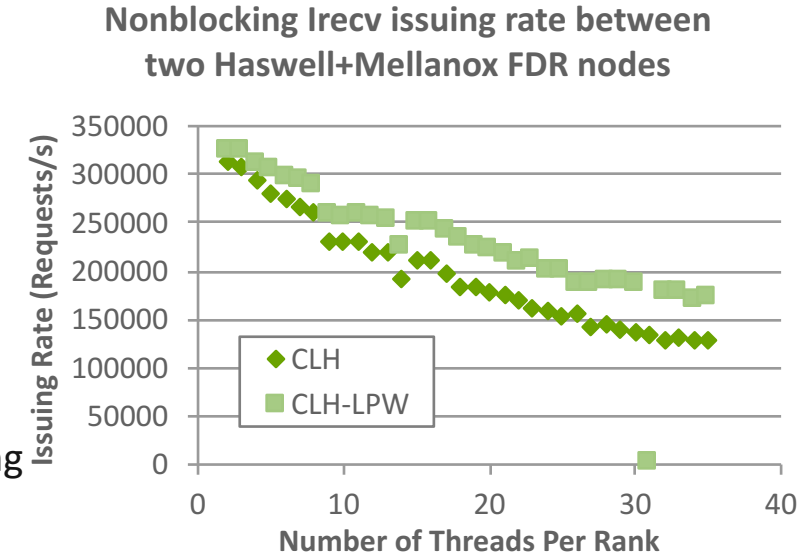


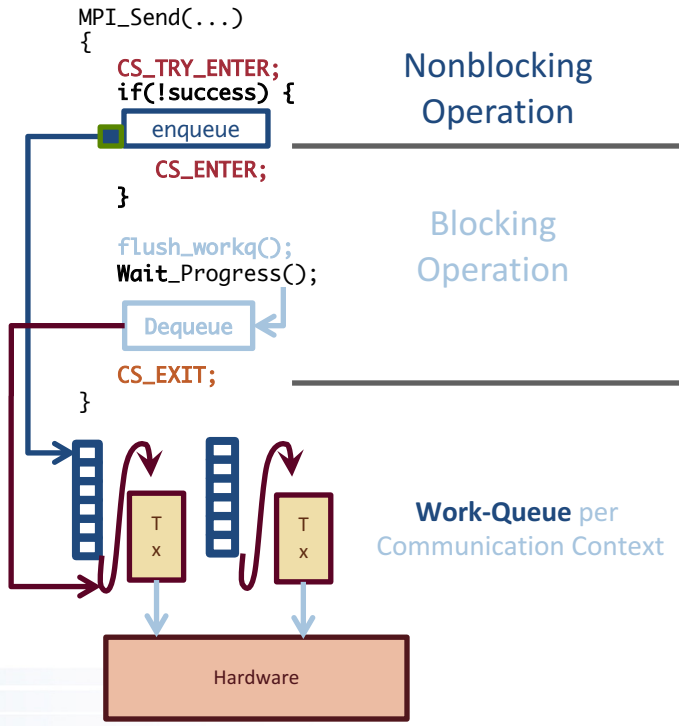
Outline

- Current MPICH Status
- MPICH-3.3
  - Lightweight communication overhead
  - Memory scalability
  - Multi-threading
    - MPICH + User-level threads
  - ROMIO data logging
  - MPI derived datatypes
- Summary

Multithreaded MPI Work-Queue Model

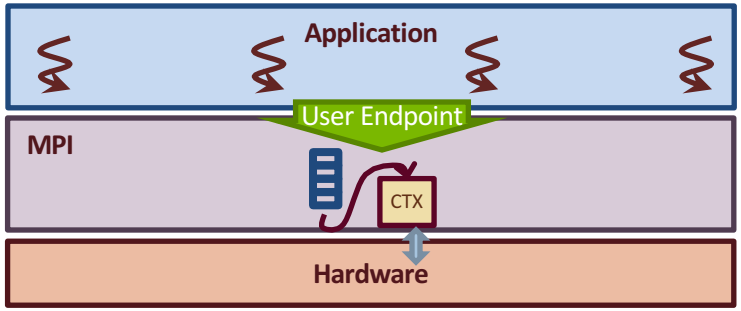
- **Context**
  - Existing lock-based MPI implementations **unconditionally** acquire locks
  - **Nonblocking** operations may **block** for a lock acquisition
    - **Not** truly nonblocking!
- **Consequences**
  - Nonblocking operations may be slowed by blocking ones from other threads
  - Pipeline stalls: higher latencies, lower thoughput, and less communication-computation overlapping
- **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



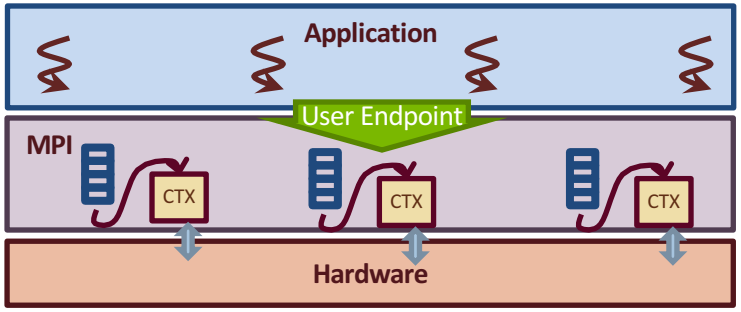


Work-Queue Model Through 3 Steps

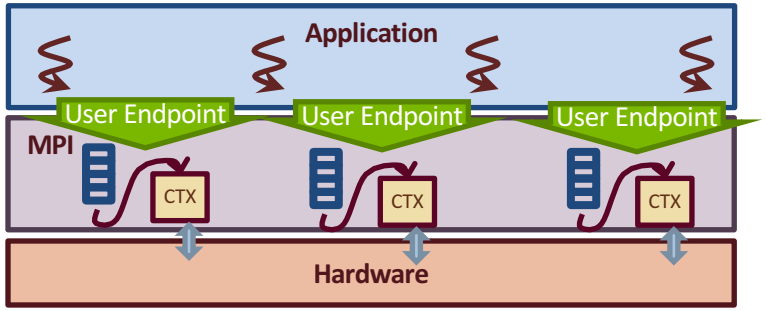
- **Step 1: Single Endpoint**
  - Current MPICH
  - Single endpoint per MPI process
  - Worst case contention



- **Step 2: Multiple User-Transparent Endpoints**
  - Multiple internal endpoints (BG/Q style)
  - Transparent to the user
  - E.g.: one endpoint per comm, per neighbor process (regular apps)



- **Step 3: Multiple User-Visible Endpoints**
  - MPI-4 Endpoints proposal
  - Multiple endpoints managed by the user



Current Work-Queue Model Implementation

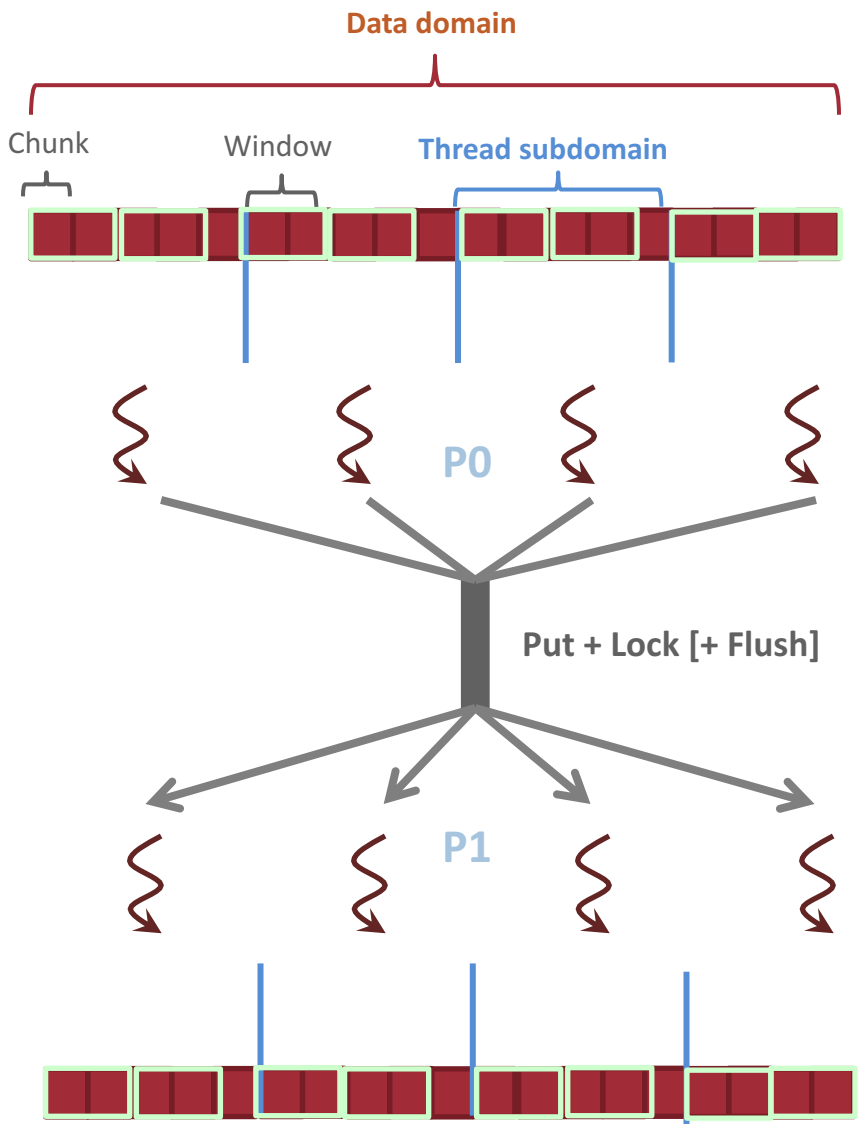
- MPIR Layer
  - All communication devices can take advantage
  - Single endpoint (endpoints have not been exposed yet)
- Progress semantics:
  - Nonblocking calls: flush queue if lock acquired
  - Blocking calls:
    - Flush work-queue at entry
    - Flush work-queue within the progress engine
- Unlimited work-queue
- Locked queue implementation
- **`Pthread mutex`** used for the global MPICH lock
- Work-queue: multiple implementations
  - Mutex locked queue
  - Michal Scott's lock-free queue
  - New Multi-Producer-Single-Consumer lock-free queues

```
MPI_Put(void* org_buf,...)
{
  CS_TRYENTER(&success);

  if(!success) {
      /* Enqueue my work */
      elem = {PUT, org_buf,…};
      enqueue(&work_queue, elem);
  }
  else{
      /* Flush the work queue */
      while(!empty(work_queue)) {
         elemt = dequeue(&work_queue);
         switch (elem.op){
            case PUT:
            MPID_Put(elem.org_buf, …);
            ...
              }
      }
      /* Issue my own op */
      MPID_Put(elem.org_buf, …);
   }
   if(success)
      CS_EXIT;
}
```

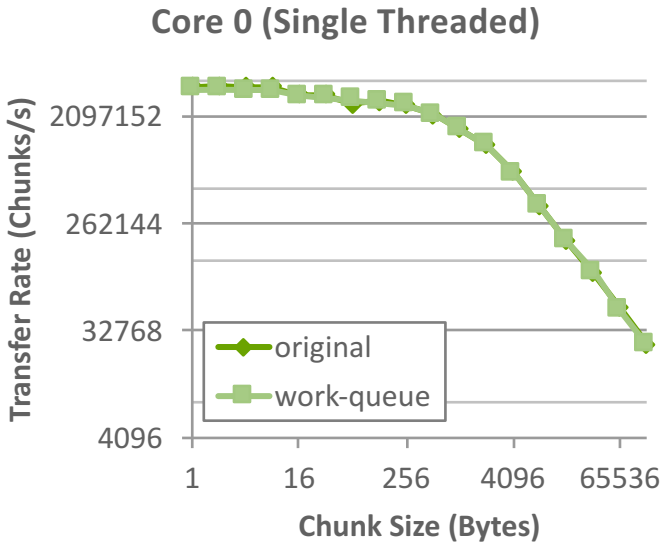
Data Transfer Rate with Threaded MPI RMA

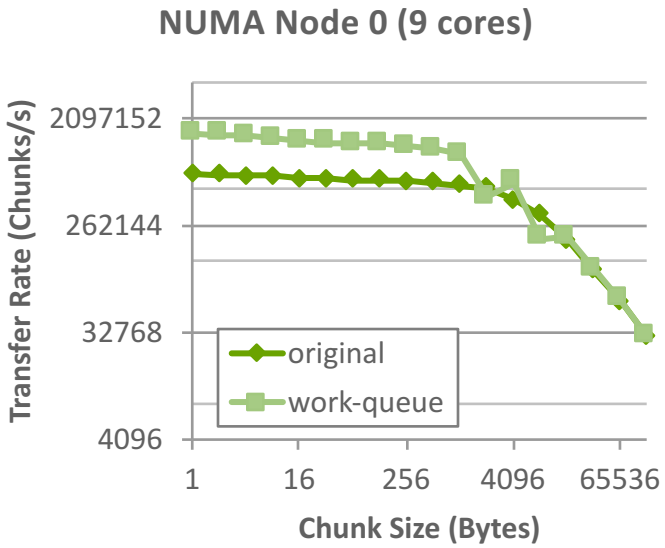
- Transfer data domain between two processes
- Stencil-like halo exchange (actual domain exchange, not like OSU benchmarks)
- Each thread gets a subdomain
- Transfer unit is a **`chunk`**
- Passive target synchronization
  - Master thread does **`Lock`**
  - All threads **`Put`** chunks
  - All threads do **`Flush`** every **`window_size`**
  - Master threads does **`Unlock`**

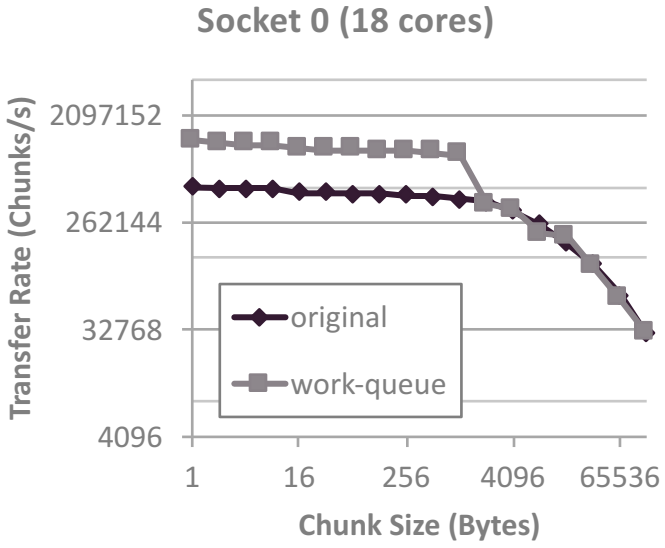


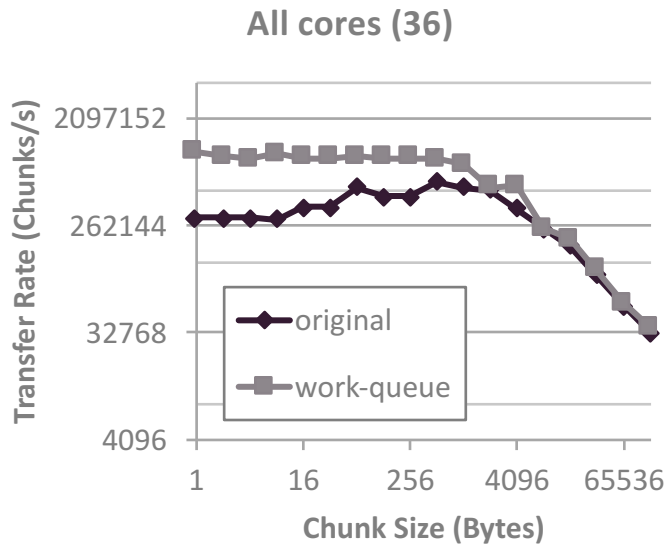
Put + Lock with a Mutex Work-queue (CH3+MXM)

No Concurrent Waiting Threads





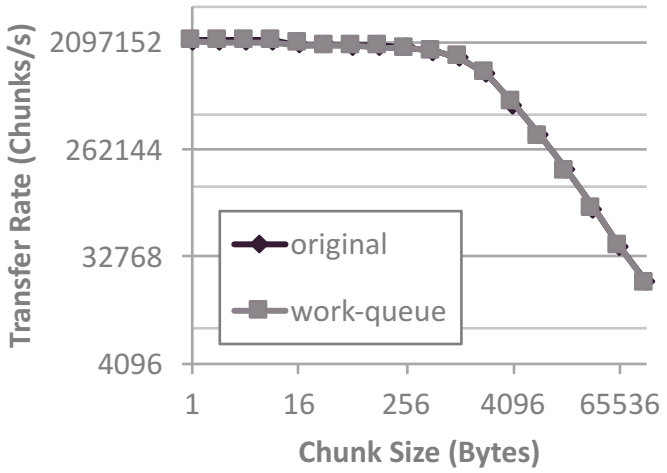




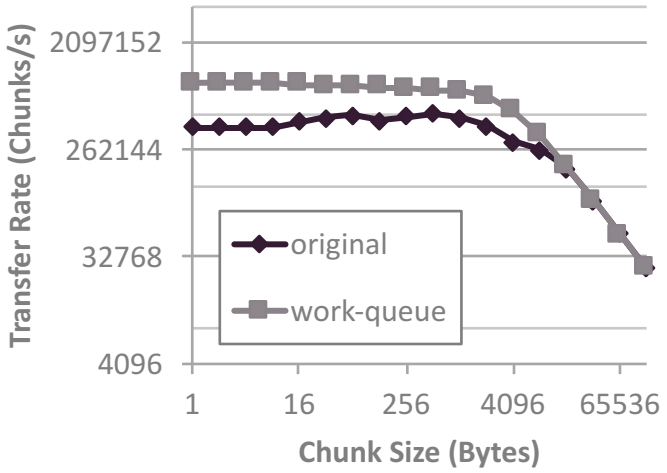
Put + Flush (w=64) + Lock with a Mutex Work-Queue

Concurrent Waiting Threads

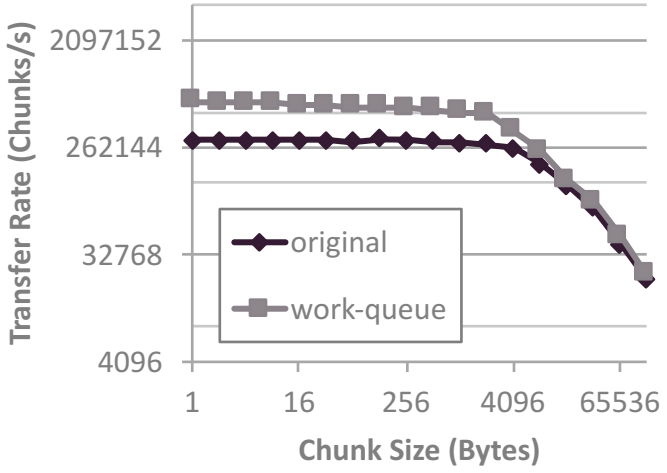
Core 0 (Single Threaded)



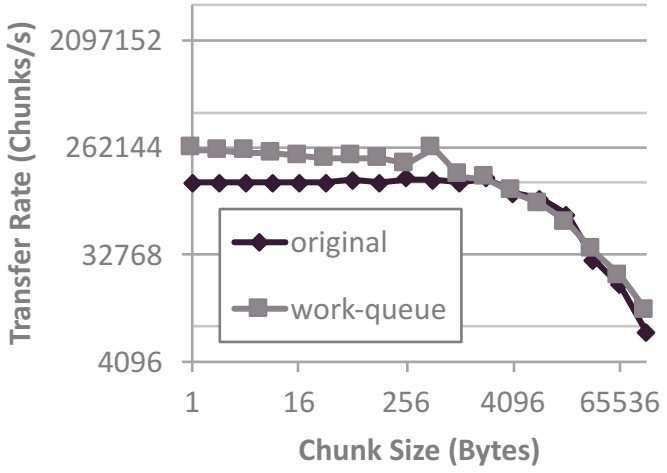
NUMA Node 0 (9 cores)



Socket 0 (18 cores)

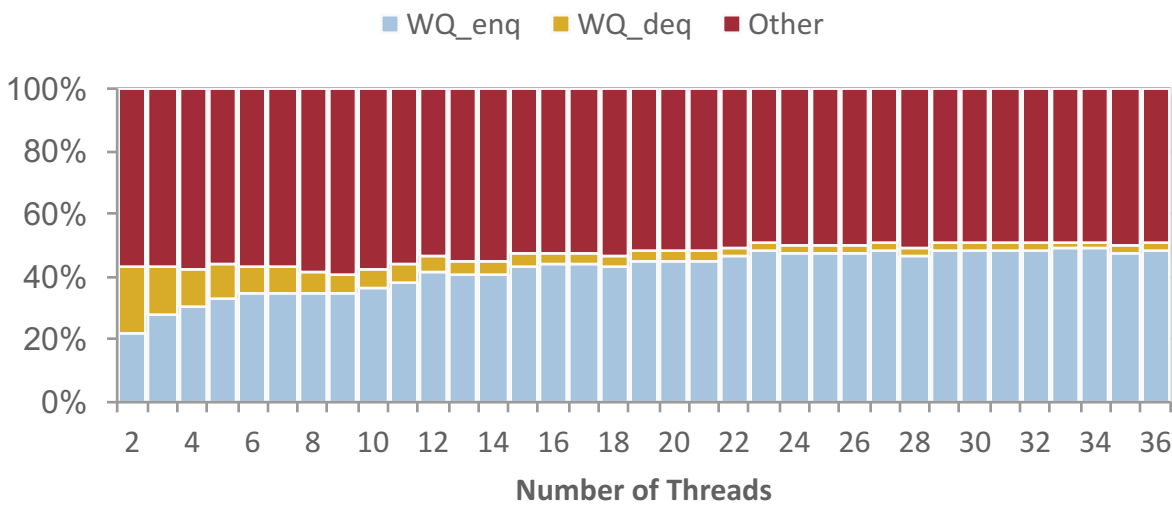


All cores (36)



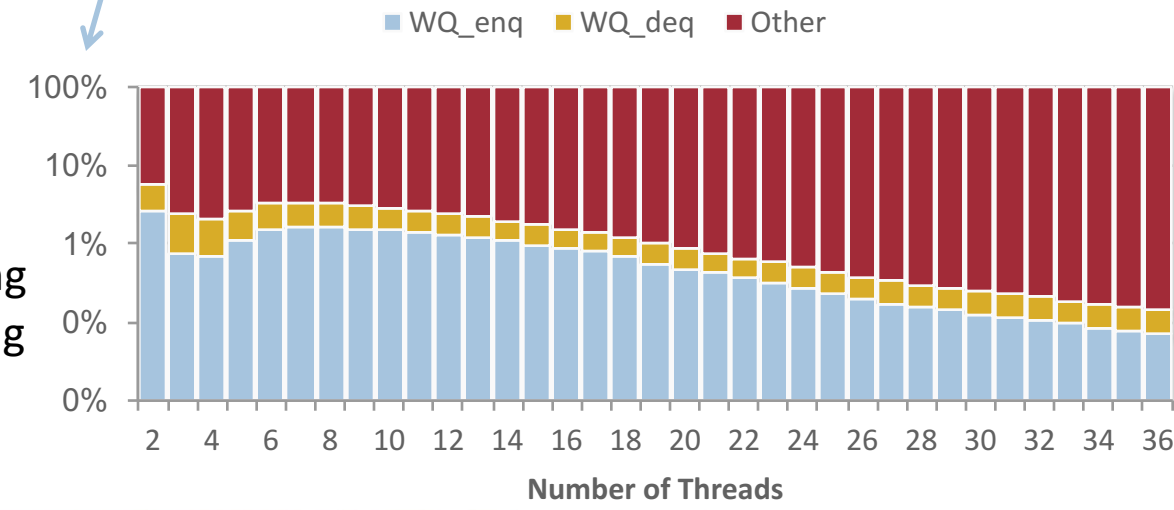
Breakdown Analysis

- **Put + Lock**
  - Queuing work is the major bottleneck!
  - Currently debugging a using faster lock-free queue
  - Goal ~ 0 overhead

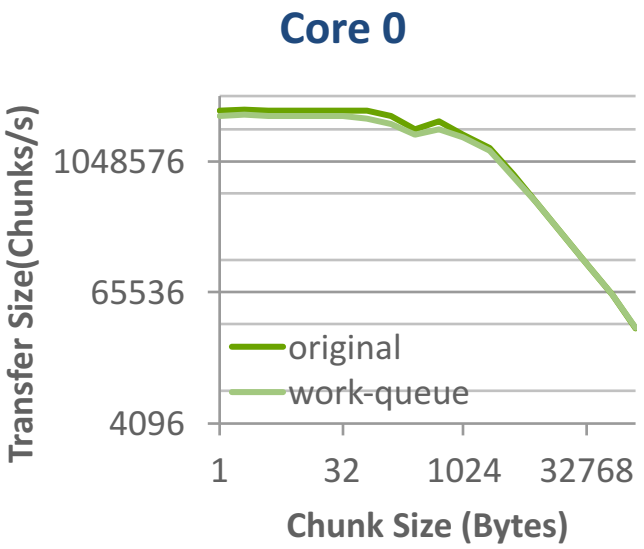


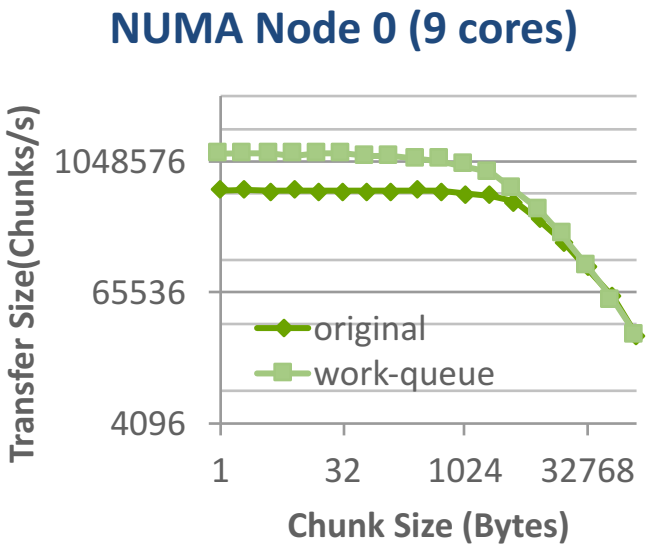
Watch out!
Log scale

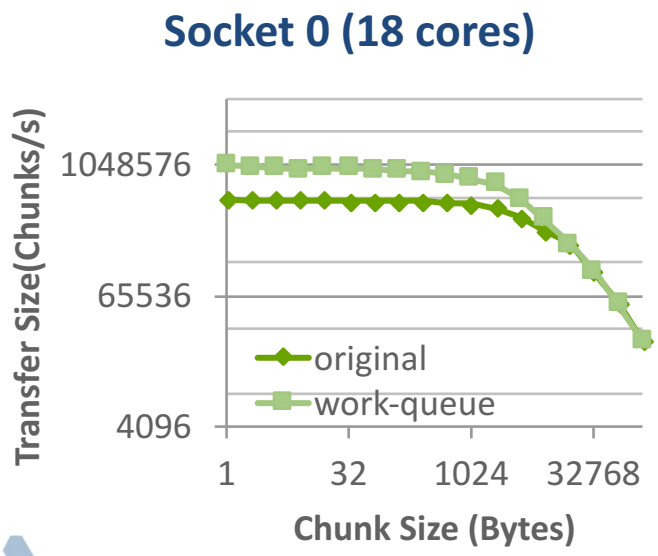
- **Put + Flush + Lock**
  - Queuing/Dequeuing work is negligible
  - Bottleneck somewhere else
  - Hypothesis: all threads waiting for completion without issuing (next slide)

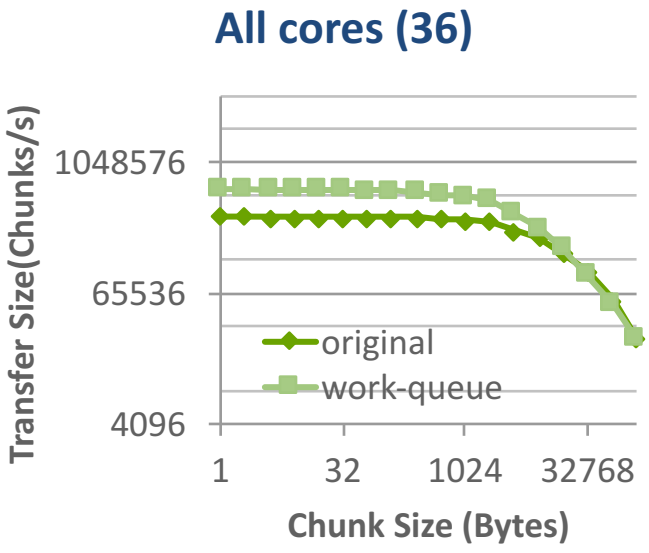


Point-to-Point Message Rate with a Mutex Work-Queue



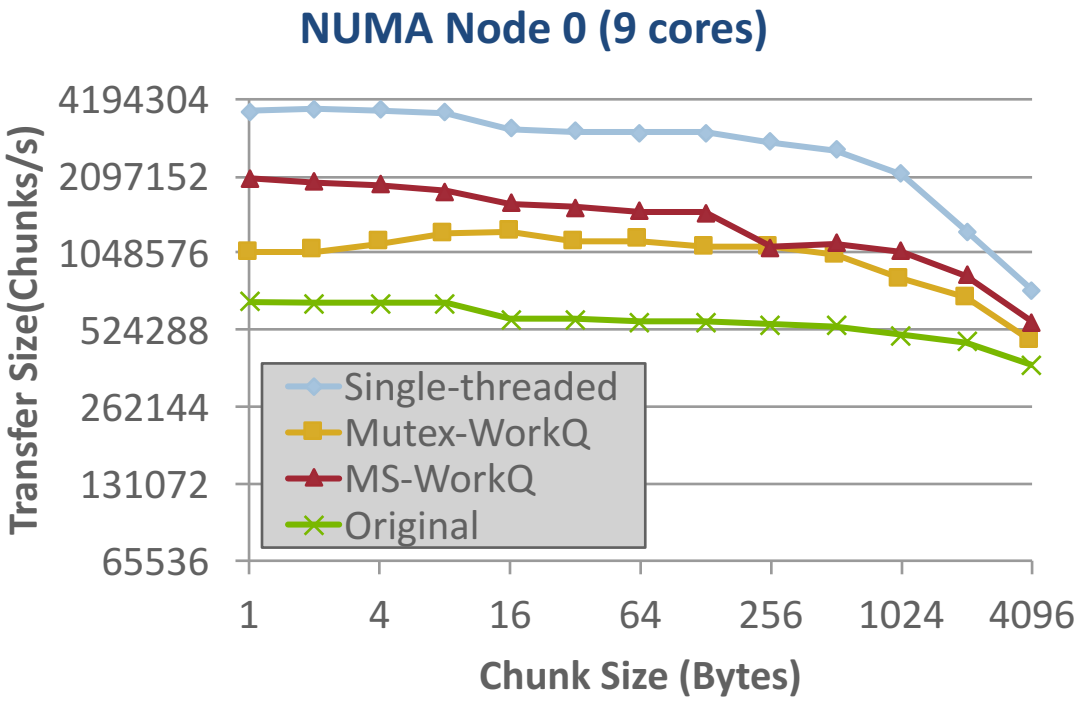






Results with a Lock-Free Work-Queue

- **Put + Lock**
  - Michael Scott's lock-free queue (MS-WorkQ)
  - Linked to TCMalloc
  - Still significantly below single-threaded
  - Working on faster lock-free queues

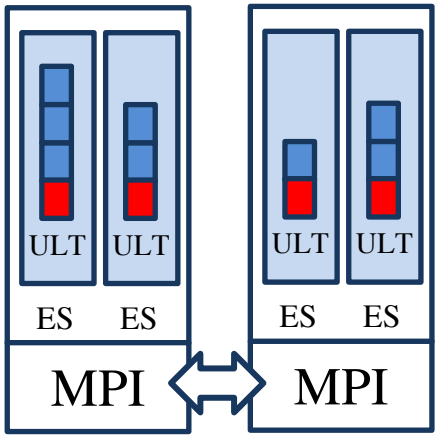


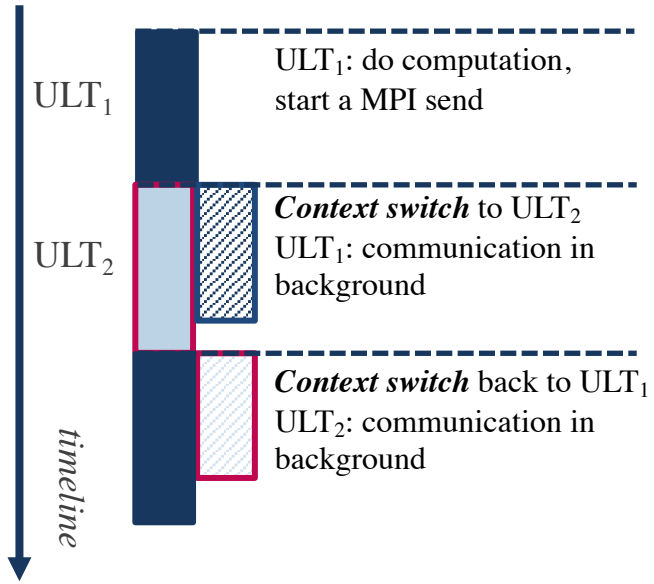
Outline

- Current MPICH
- MPICH-3.3 and beyond
  - Lightweight communication overhead
  - Memory scalability
  - Multi-threading
    - MPICH + User-level threads
  - ROMIO data logging
  - MPI derived datatypes
- Summary

Supporting User-level Threads in MPICH (Argobots)

- Motivation
  - Traditional MPI implementations are only aware of kernel threads
  - Thread-synchronization is costly to ensure thread-safety and progress requirement from MPI
  - Wasted resources if a kernel thread blocks for MPI communication
- **Argobots-aware MPICH**
  - Supports Argobots as another threading model
  - Lightweight context switching to overlap costly blocking operations
    - Communication, locks, etc.
  - Reduced thread-synchronization opportunities
    - Guaranteed consistency within an ES without locks or memory barriers





MPI+Argobots Execution Model

Outline

- Current MPICH Status
- MPICH-3.3
  - Lightweight communication overhead
  - Memory scalability
  - Multi-threading
    - MPICH + User-level threads
  - ROMIO data logging
  - MPI derived datatypes
- Summary

MPICH-OFI*

Wesley Bland
Senior Software Developer, Intel Corporation
Intel HPC Developers Conference
November 12, 2016

MPICH-OFI*

Open-source implementation based on MPICH

- Uses the new CH4 infrastructure
  - Co-designed with MPICH community
- Targets existing and new fabrics via next-gen Open Fabrics Interface (OFI)
  - Ethernet/sockets, Intel® Omni-Path, Cray Aries*, IBM BG/Q*, InfiniBand*
- Will be the default implementation available on the Aurora Supercomputer at Argonne National Laboratory

MPICH-OFI* Developments in 2016

Multiple hackathons with Argonne and internal development work to expand CH4 feature set:

- Capability sets
- Improved RMA support
- Reduce instruction overhead
- Support for improved internal concurrency

Capability Sets

Allows the user to compile-time select a set of OFI features to optimize lookups later in the execution.

Optimized for the best performance for each OFI provider.

Can enable runtime configuration to make things more flexible, if desired.

PSM2 Capability Set (subset)	
ENABLE_DATA	ON
ENABLE_AV_TABLE	ON
ENABLE_SCALABLE_ENDPOINTS	OFF

Sockets Capability Set (subset)	
ENABLE_DATA	ON
ENABLE_AV_TABLE	ON
ENABLE_SCALABLE_ENDPOINTS	ON

Improved RMA Support

Map MPI functionality directly to OFI features as much as possible

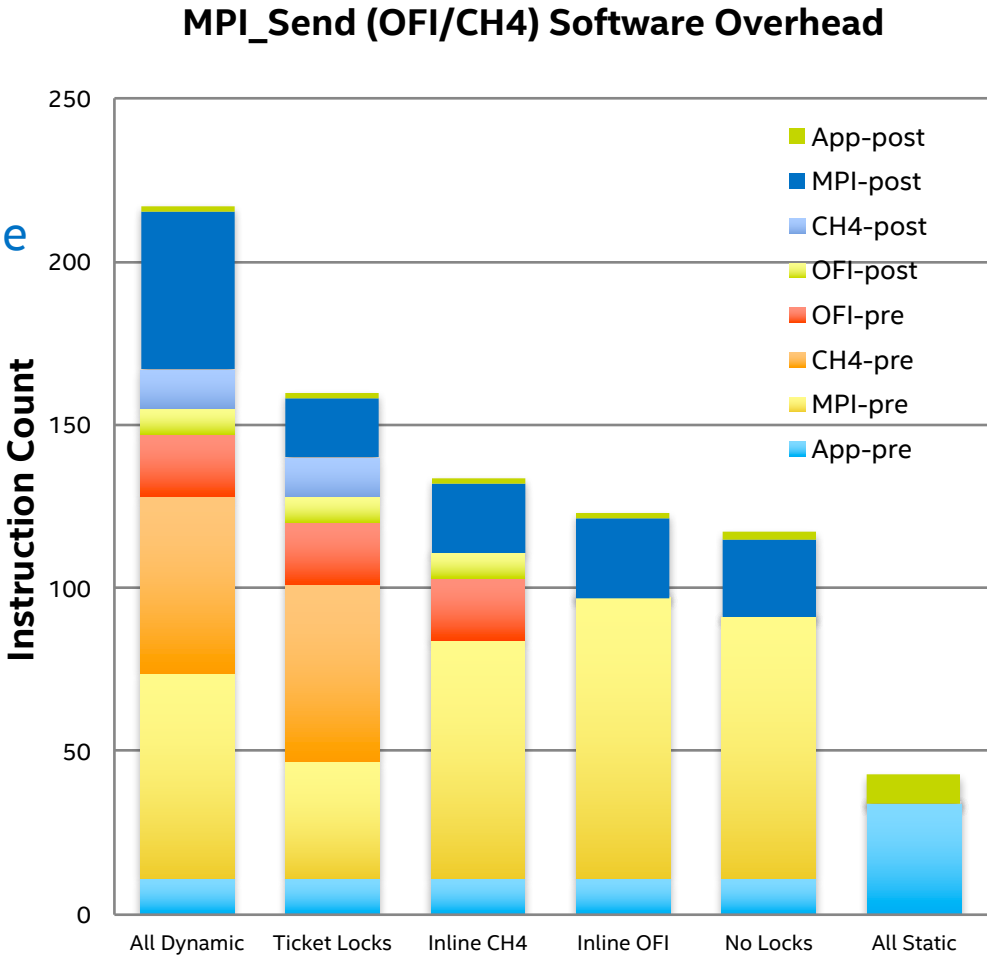
- OFI has direct support for Put, Get, Accumulate, etc.
- This may reduce software overhead and utilize underlying communication fabric better.

Reduced Instruction Overhead

As low as 43 instructions from application to OFI with all optimizations on

Reduce branching as much as possible

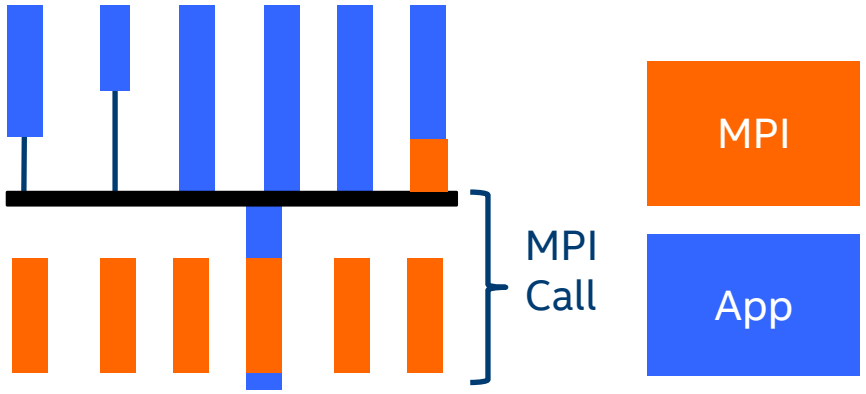
Reduce memory footprint



Support for improved internal concurrency

Parallel packing and unpacking using derived datatypes

- The approach shares threads between MPI and OpenMP*
  - MPI can steal application threads that are idle
  - MPI creates tasks that application threads can execute when idle
- MPI doesn't create additional threads.
  - No oversubscription.
- This model maps well to other runtimes, such as Intel® TBB or Cilk™ Plus



COMING SOON

Collective Selection

Improved Shared Memory Support

Collective Selection

Current MPICH

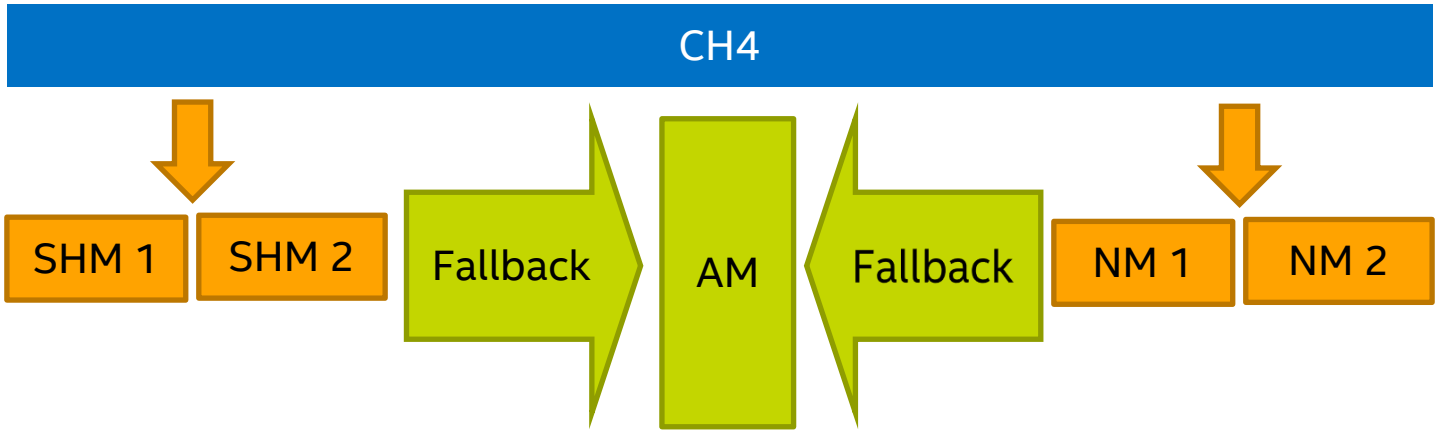
- Static selection of algorithms based on message size and communicator size at initialization

Proposal

- Introduce intelligent selection to determine optimal collective algorithm
  - Could be picked from a static configuration, runtime selection, etc.
  - Can use default algorithms or device/netmod/etc. specific algorithms

Improved Shared Memory Support

- Support multiple shmmods (pronounced shmem-mods)
  - Might implement a subset of the API and fall back to active messages for default support
- Similar architecture to current netmod design

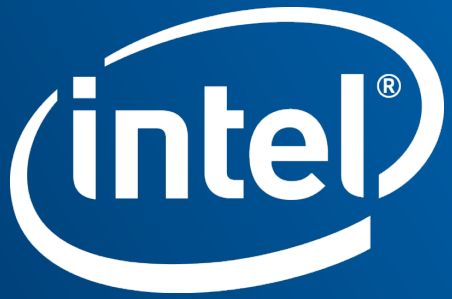


Legal Disclaimer & Optimization Notice

Optimization Notice

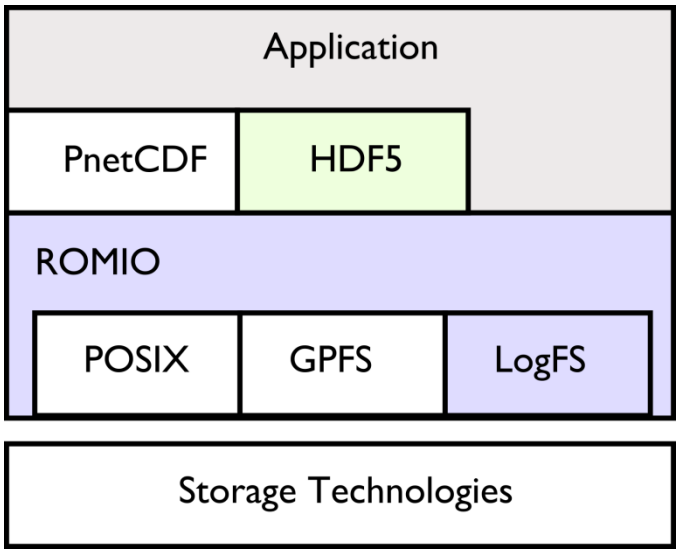
Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, 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.

Notice revision #20110804


intel®

ROMIO data logging

- WHO: Rob Latham (ANL)
- PROBLEM: How to make use of new layers in storage hierarchy?
- SOLUTION: “ad_logfs” maintains a log-structured record of all write activities
  - Sits below MPI-IO routines: transparent save for ‘logfs:’ prefix
  - Maintains one set of files per MPI process
    - Metadata, data, and global state
  - Can replay on close, explicit sync, or upon first read
- Intent: log all I/O to NVRAM or SSD, defer replay to parallel file system



Outline

- Current MPICH Status
- MPICH-3.3
  - Lightweight communication overhead
  - Memory scalability
  - Multi-threading
    - MPICH + User-level threads
  - ROMIO data logging
  - MPI derived datatypes
- Summary

DAME: a new engine for derived datatypes

- **Who:** Tarun Prabu, Bill Gropp (UIUC)
- **Why:** DAME is an improved engine for derived-datatypes
  - The Dataloop code (type processing today) effective, but requires many function calls (the “piece functions”) for each “leaf type”
  - Piece Functions (function pointers) are difficult for most (all?) compilers to inline, even with things like link-time optimizations
- **What:** DAME implements a new description of the MPI datatype, then transforms that description into efficient memory operations
- **Design Principles**:
  - Low processing overhead
  - Maximize ability of compiler to optimize code
  - Simplify partial packing
  - Enable memory access optimizations
- **Optimizations**:
  - Memory access optimizations can be done by shuffling primitives as desired. This is done at “commit” time.
  - Other optimizations such as normalization (e.g. an indexed with identical stride between elements), displacement sorting and merging can also easily be performed at commit-time.
- **More information at: https://wiki.mpich.org/mpich/index.php/DAME**

Outline

- Current MPICH
- MPICH-3.3 and beyond
  - Lightweight communication overhead
  - Memory scalability
  - Multi-threading
    - MPICH + User-level threads
  - ROMIO data logging
  - MPI derived datatypes
- Summary

MPICH-3.3 Next Major Release

- The CH4 device
  - Replacement for CH3
    - CH3 still supported and maintained for the time-being
  - Primary objectives
    - Lightweight communication overhead
      - Ability to support high-level network APIs (OFI, UCX, Portals 4)
      - E.g., tag-matching in hardware, direct PUT/GET communication
      - Low memory footprint
    - 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)

MPICH-3.3 Timeline

- CH4 code in main MPICH repo (recently moved to GitHub)
  http://github.com/pmodels/mpich
  - Some work-in-progress features in Pull Request branches
- MPICH-3.3a2 release out this week
  - Subsequent pre-releases as the code is stabilized, features added
- GA Release mid-2017

Thank you

- Questions?