Towards an Extreme Scale Multithreaded MPI

Abdelhalim Amer (Halim)
Postdoctoral Researcher
Argonne National Laboratory, IL, USA

Japan-Korea HPC Winter School
University of Tsukuba, Feb 17, 2016

Programming Models and Runtime Systems Group

Myself

Affiliation

- Mathematics and Computer Science Division, ANL

Group Lead

- Pavan Balaji (computer scientist)

Current Staff Members

- Sangmin Seo (Assistant Scientists)
- **Abdelhalim Amer (Halim)** (postdoc)
- Yanfei Guo (postdoc)
- Rob Latham (developer)
- Lena Oden (postdoc)
- Ken Raffenetti (developer)
- *Min Si (postdoc)*

Argonne
NATIONAL LABORATORY

The PMRS Group Research Areas

Communication Runtimes (MPICH)

Threading Models (OS-level, user-level: Argobots)

Data-Movement in Heterogeneous and Deep Memory Hierarchies

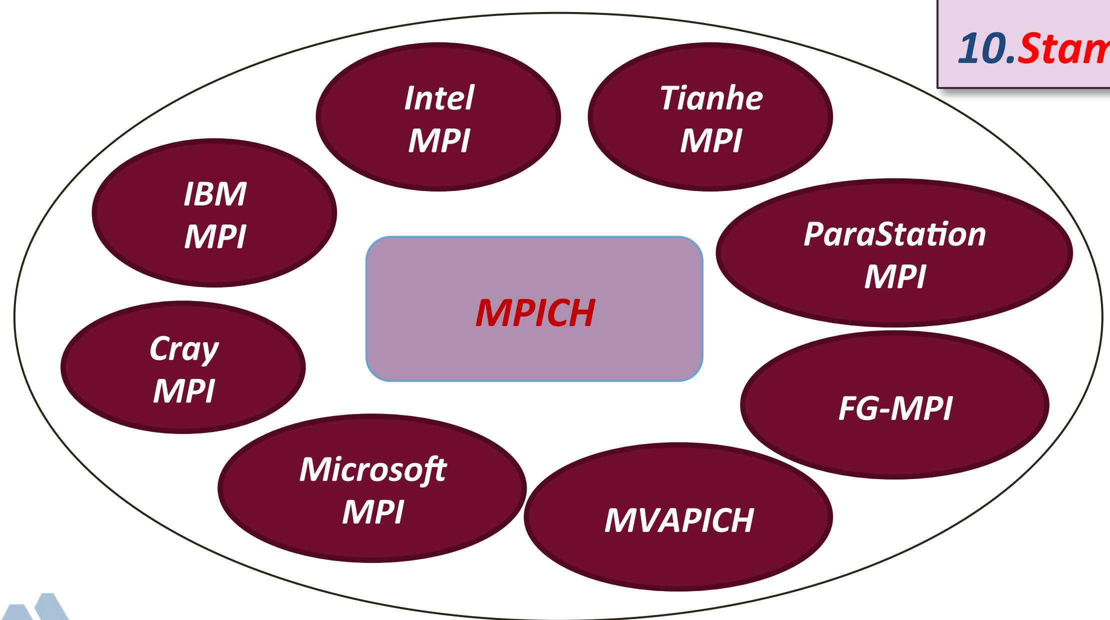
My focus:
Communication optimization in threading environments

The MPICH Project

- MPICH and its derivatives are the world's most widely used MPI implementations
- Funded by DOE for 23 years (turned 23 this month)
- Has been a key influencer in the adoption of MPI
- Award winning project
  - DOE R&D100 award in 2005

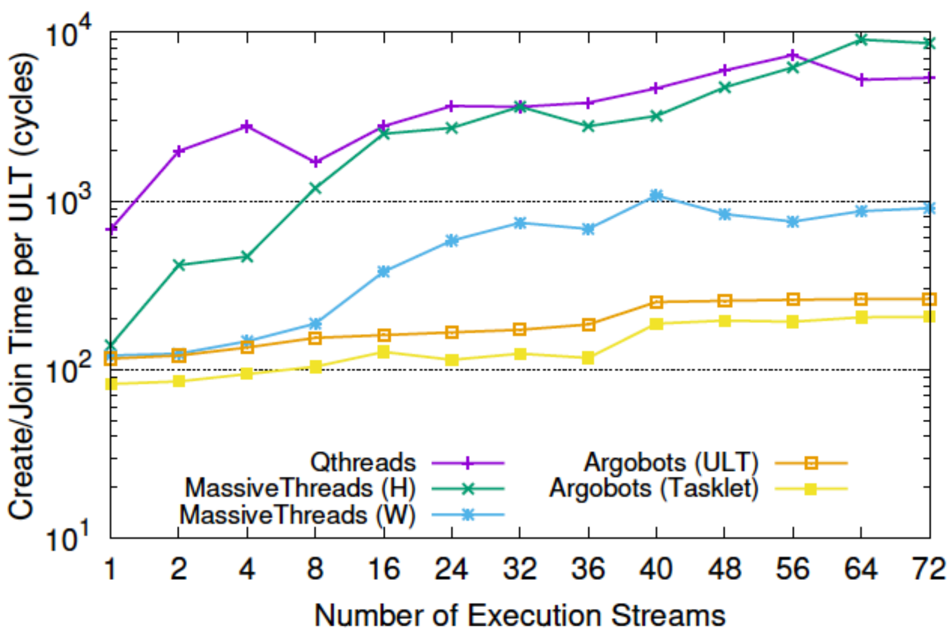
<u>MPICH and it derivatives in the Top 10</u>

1. ***Tianhe-2 (China): TH-MPI***
2. ***Titan (US): Cray MPI***
3. ***Sequoia (US): IBM PE MPI***
4. K Computer (Japan): Fujitsu MPI
5. ***Mira (US): IBM PE MPI***
6. ***Trinity (US): Cray MPI***
7. ***Piz Daint (Germany): Cray MPI***
8. ***Hazel Hen (Germany): Cray MPI***
9. ***Shaheen II (Saudi Arabia): Cray MPI***
10. ***Stampede (US): Intel MPI and MVAPICH***



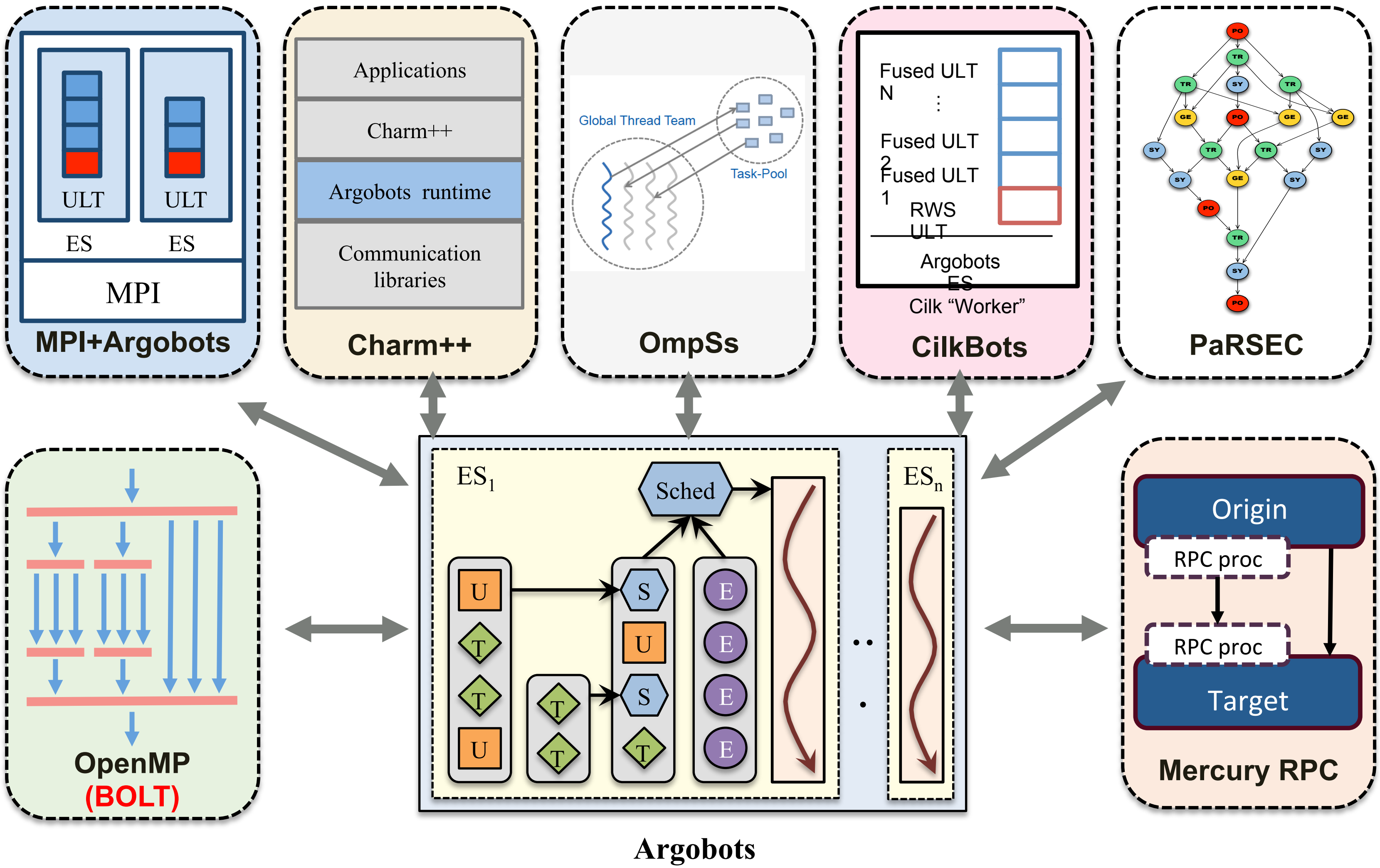
Threading Models

- Argobots
  - Lightweight low-level threading and tasking framework
  - Targets massive on-node parallelism
  - Fine-grained control over the execution, scheduling, synchronization, and data-movements
  - Exposes a rich set of features to higher programming systems and DSLs for efficient implementations
- BOLT (https://press3.mcs.anl.gov/bolt)
  - Based on the LLVM OpenMP runtime and the Clang



Sangmin Seo
Assistant Computer Scientist

The Argobots Ecosystem



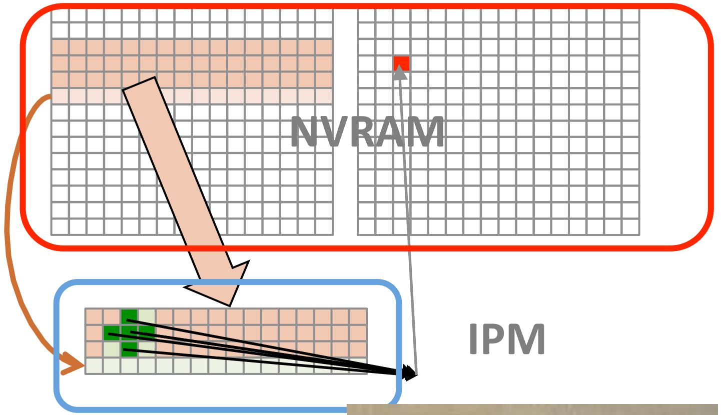
External Connections

GridFTP, Kokkos, RAJA, ROSE, TASCEL, XMP, etc.

Heterogeneous and Deep Memory Hierarchies

- Prefetching for deep memory hierarchies
- Currently focusing on MIC accelerator environments
- Compiler support
  - LLVM + user hints through pragma directives
- Runtime support
  - Prefetching between In-Package Memory (IPM) and NVRAM
  - Software-managed cache

Lena Oden
Postdoc



Yanfei Guo
Postdoc

The PMRS Group Research Areas

Communication Runtimes (MPICH)

Threading Models (OS-level, user-level: Argobots)

Data-Movement in Heterogeneous/ Hierarchical Memories

My focus:
Communication optimization in threading environments

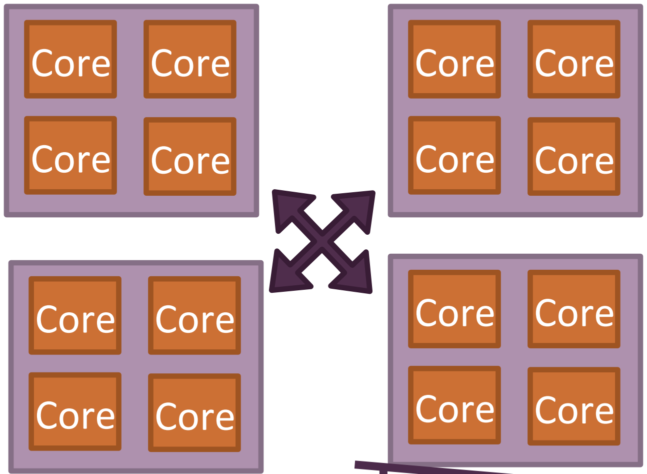
Today's Talk

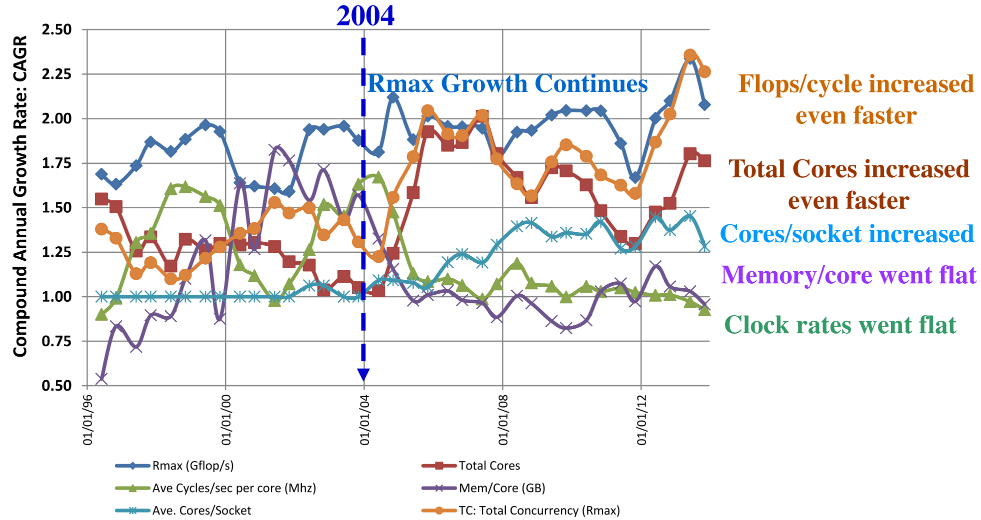
Outline

- **Introduction to hybrid MPI+threads programming**
  - Hybrid MPI programming
  - Multithreaded-Driven MPI communication
- **MPI + OS-Threading Optimization Space**
  - Optimizing the coarse-grained global locking model
  - Moving towards a fine-grained model
- **MPI + User-Level Threading Optimization Space**
  - Optimization Opportunities over OS-threading
  - Advanced coarse-grained locking in MPI
- **Summary**

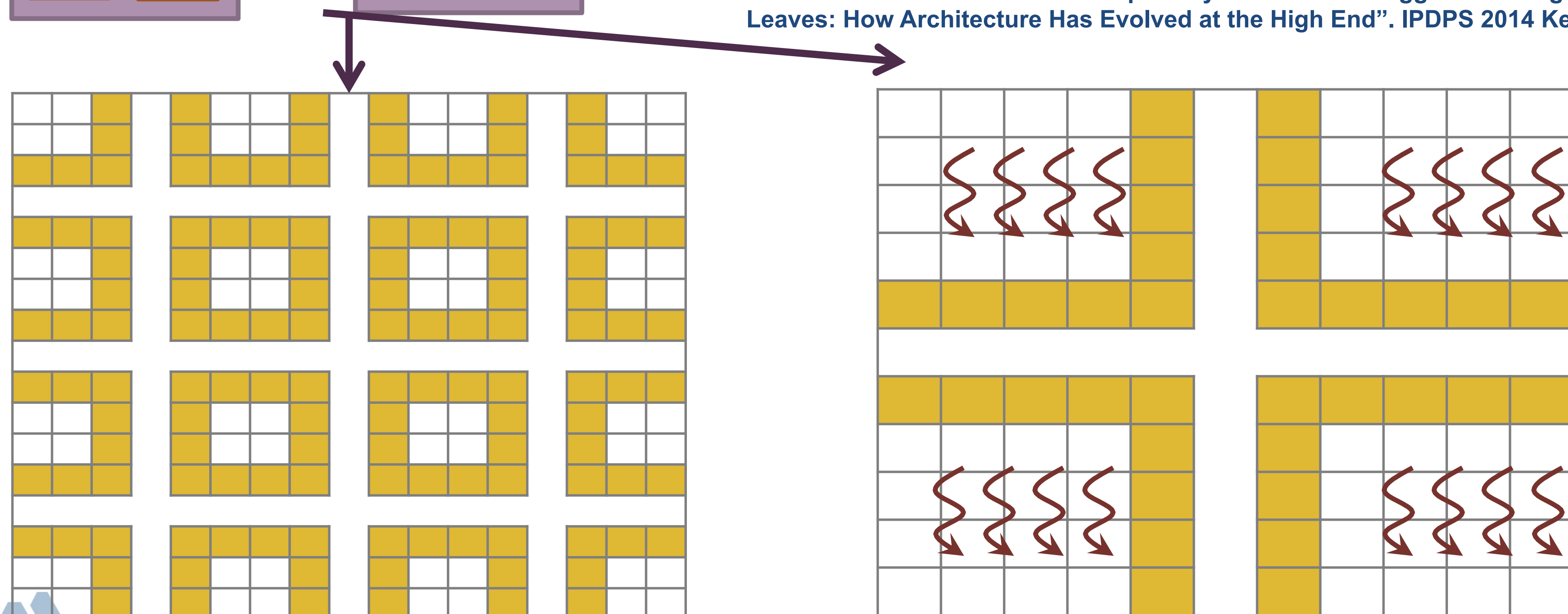
Introduction to Hybrid MPI Programming

Why Going Hybrid MPI + Shared-Memory (X) Programming





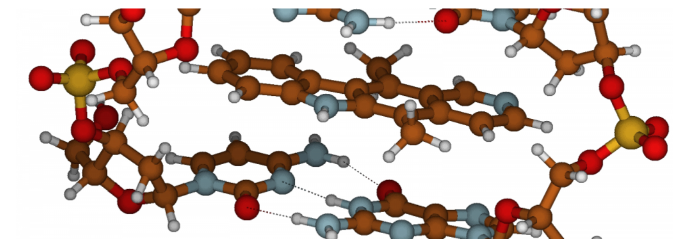
Growth of node resources in the Top500 systems. Peter Kogge: “Reading the Tea-Leaves: How Architecture Has Evolved at the High End”. IPDPS 2014 Keynote



Main Forms of Hybrid MPI Programming

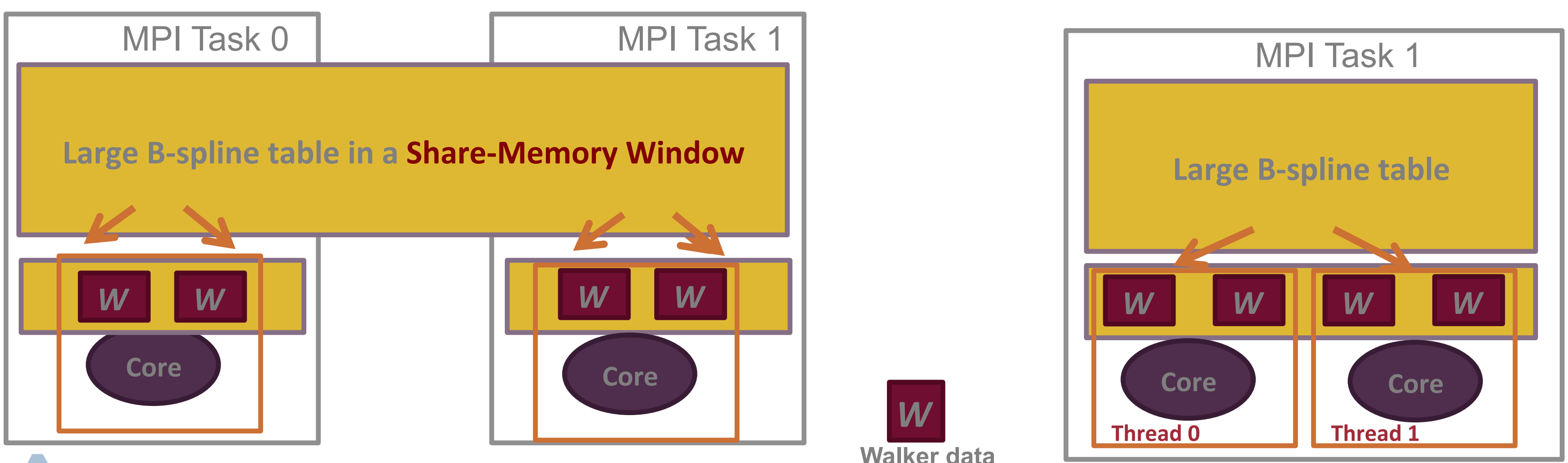
QMCPACK

- E.g. Quantum Monte Carlo simulations
- Two different models
  - MPI + shared-memory (X = MPI)
  - MPI + threads (e.g. X = OpenMP)
- Both models use direct load/store operations



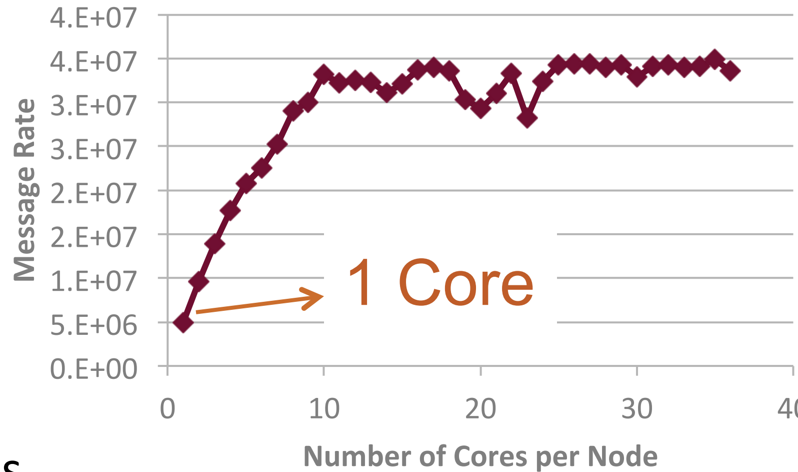
MPI + Shared-Memory (MPI 3.0~)
- Everything private by default
- Expose shared data explicitly

MPI + Theads
- Share everything by default
- Privatize data when necessary

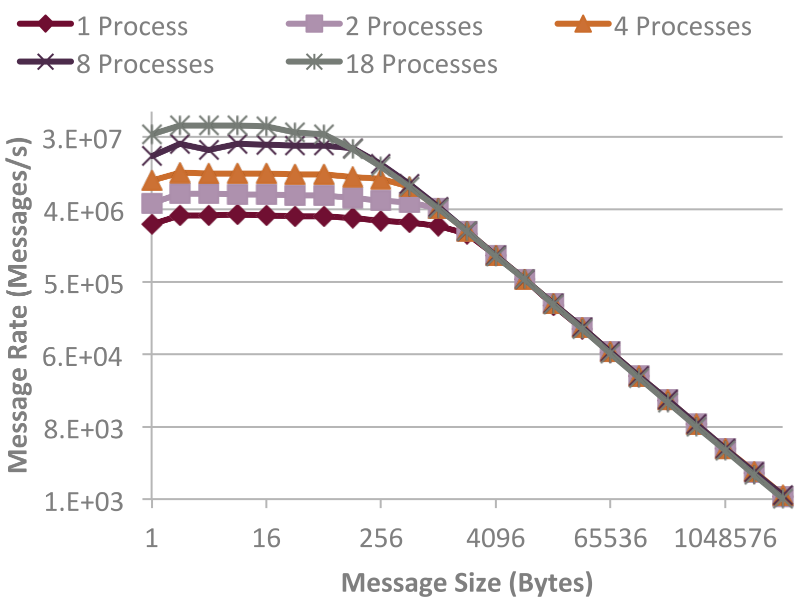


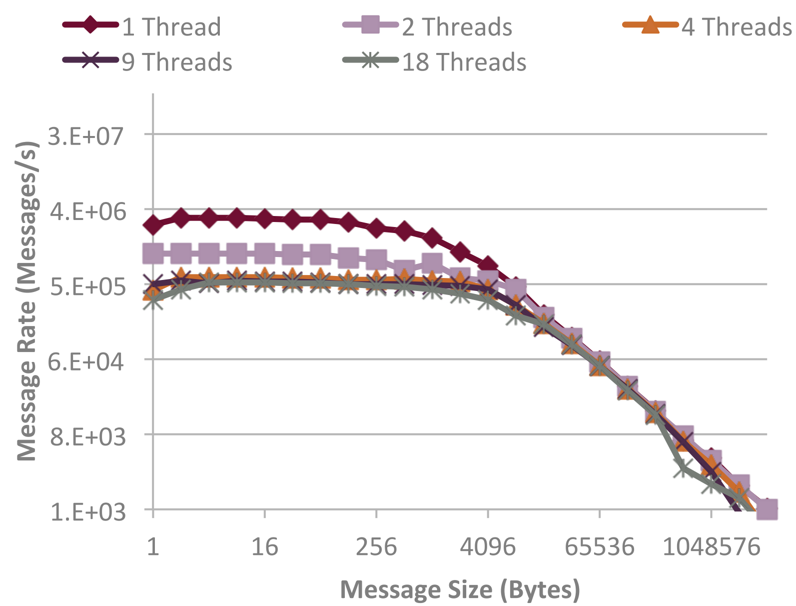
Multithreaded MPI Communication: Current Situation

- Becoming more difficult for a single-core to saturate the network
- Network fabrics are going parallel
  - E.g. BG/Q has 16 injection/reception FIFOs
- Core frequencies are reducing
  - Local message processing time increases
- Driving MPI communication through threads backfires



Node-to-node message rate (1B) on Haswell + Mellanox FDR Fabric

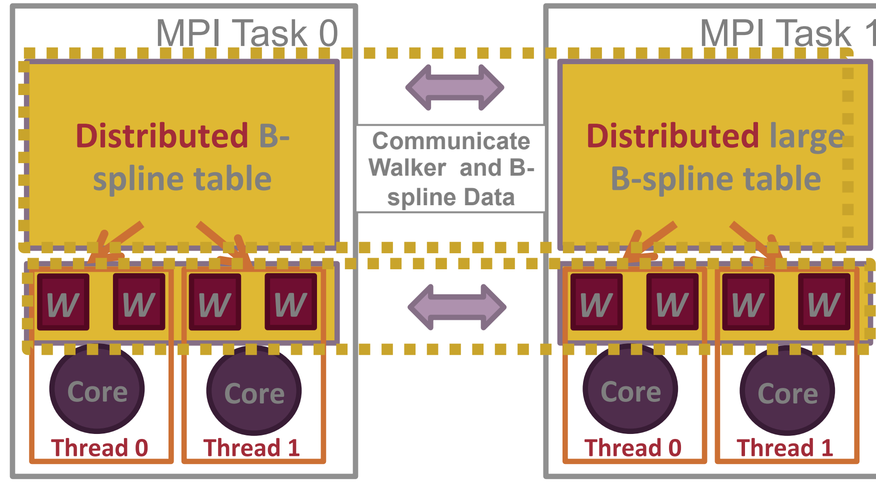




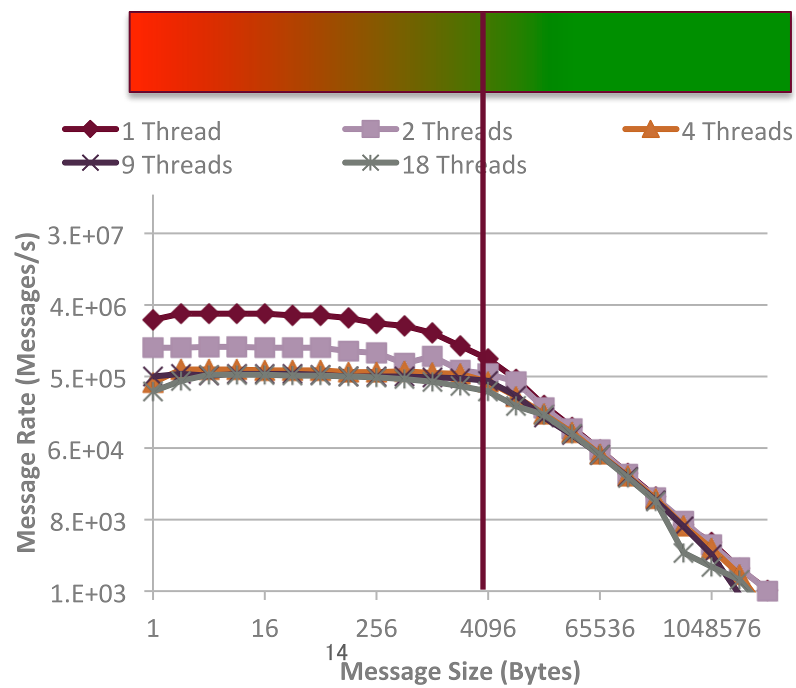
Node-to-node message rate on Haswell + Mellanox FDR Fabric: processes vs. threads

Applications to Exploit Multithreaded MPI Communication

- Several applications are moving to MPI +Threads models
- Most of them rely on `funneled communication`!
- Certain type of applications fit better multithreaded communication model
  - E.g. Task-parallel applications
- Several applications can scale well with current MPI runtimes if:
  - Data movements are not too fine-grained
  - Frequency of calling concurrently MPI is not too high
- A scalable multithreaded support is necessary for other applications



Distributed B-spline table if it cannot fit in memory fin Quantum Monte Carlo Simulations [1]

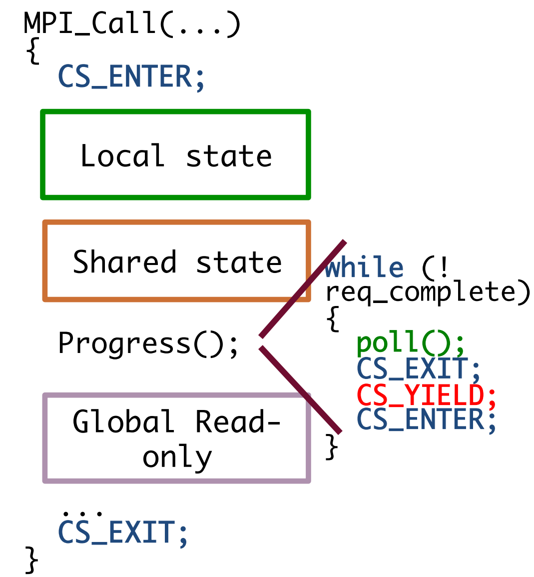


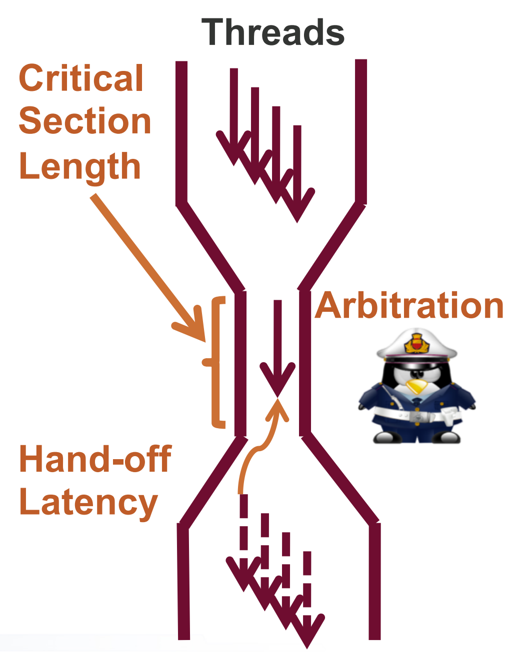
[1] Kim, Jeongnim, et al. "Hybrid algorithms in quantum Monte Carlo." Journal of Physics, 2012.

OS-Thread-Level Optimization Space

Challenges and Aspects Being Considered

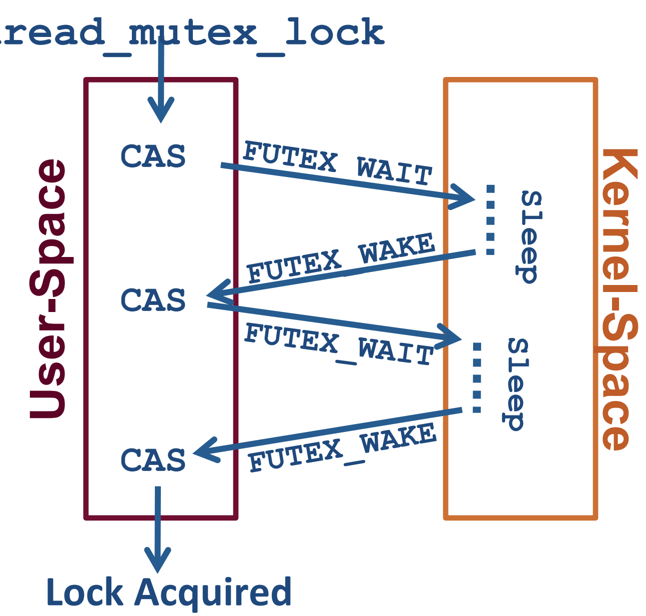
- **Granularity**
  - The current coarse-grained `lock/work/unlock` model is not scalable
  - Fully lock-free is not practical
    - Rank-wise progress constrains
    - Ordering constrains
    - Overuse of atomics and memory barriers can backfire
- **Arbitration**
  - Traditional `Pthread mutex` locking is biased by the hardware (NUCA) and the OS (slow wakeups)
  - Causes waste because of the lack of correlation between resource acquisition and work
- **Hand-off latency**
  - Slow hand-off latencies waste the advantages of fine-granularity and smart arbitration
  - Trade-off must be carefully addressed

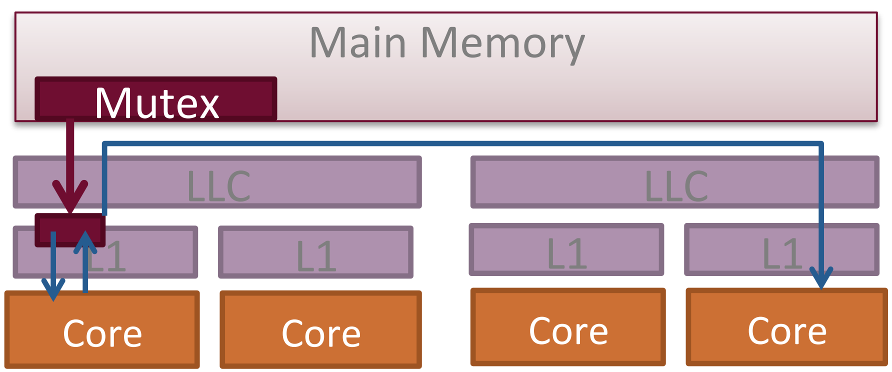




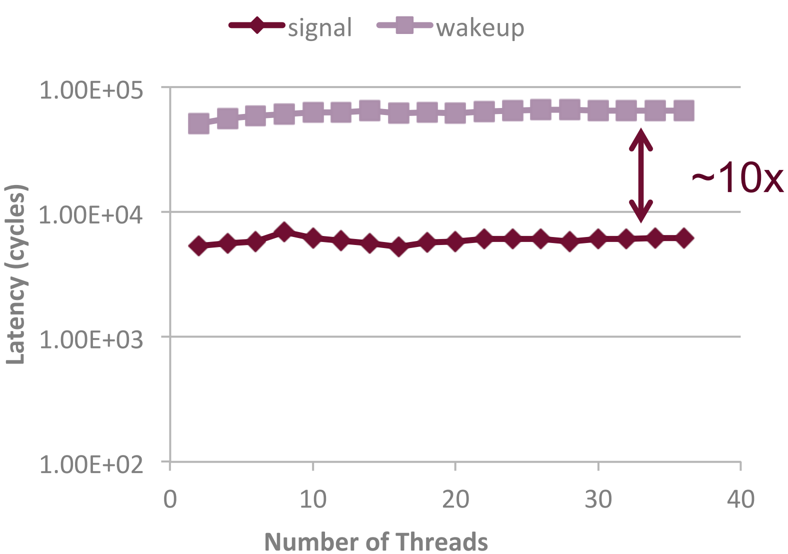
Production Thread-Safe MPI Implementations on Linux

- OS-thread-level synchronization
- Most implementations rely on **coarse-grained** critical sections
  - Only MPICH on BG/Q is known to implement fine-grained locking
- Most implementations rely on **Pthread** synchronization API (mutex, spinlock, …)
- Mutex in Native Posix Thread Library (NPTL)
  - Ships with glibc
  - Fast user-space locking attempt; usually with a Compare And Swap (CAS)
  - **Futex** wait/wake in contended cases





Lock acquisition hardware bias (e.g. CAS)



OS-bias from slow futex wakeups

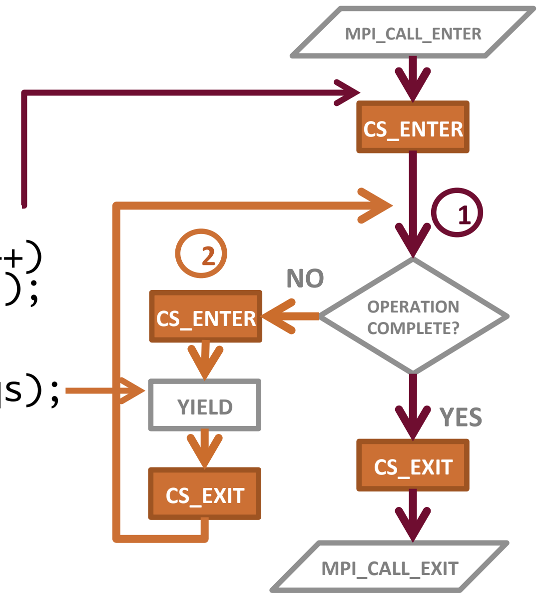
Analysis of MPICH on a Linux Cluster

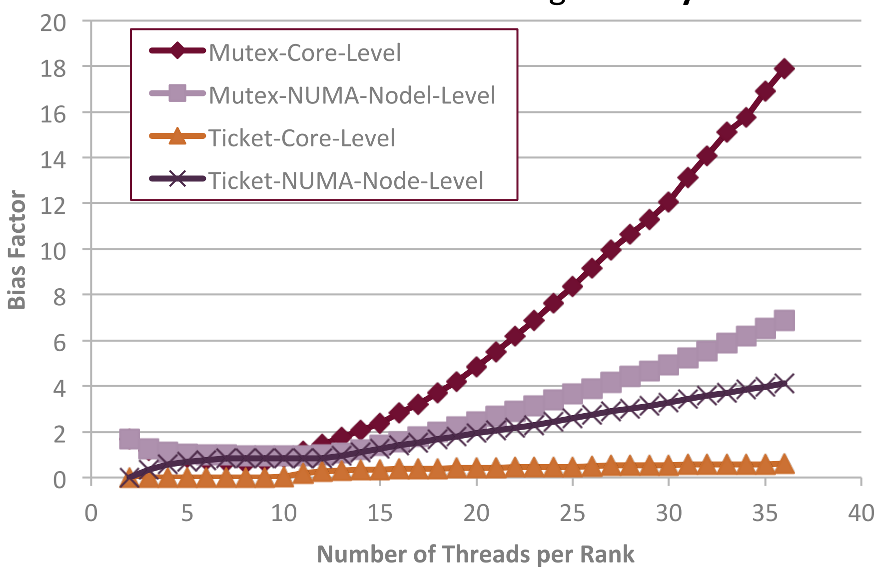
- Thread-safety in MPICH
  - Full `THREAD_MULTIPLE` support
  - OS-thread-level synchronization
  - **Single coarse-grained** critical section
  - Uses by default **Pthread mutex**
- Fairness analysis
  - Bias factor (BF): $\frac{P(same\_domain\_acquisition)}{P(random\_uniform)}$
  - BF <=1 → fair arbitration
- Progress analysis
  - Wasted progress polls = unsuccessful progress polls while other threads are waiting at **entry**

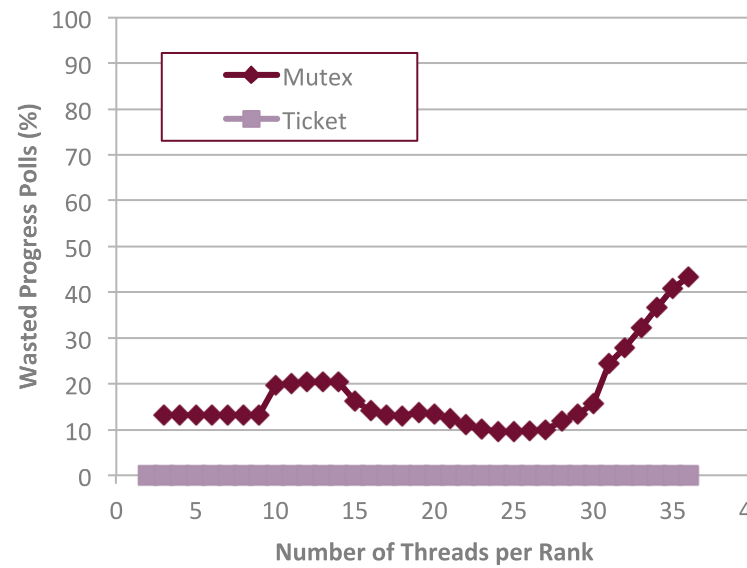
```
#pragma omp parallel
{

  for (i=0; i<nreq; i++)
    MPI_Irecv(&reqs[i]);

  MPI_Waitall(nreq,reqs);
}
```





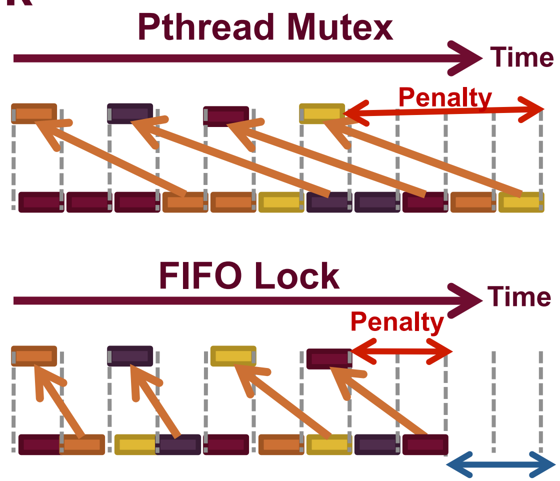


Fairness and progress analysis with a message rate benchmark on Haswell + Mellanox FDR fabric

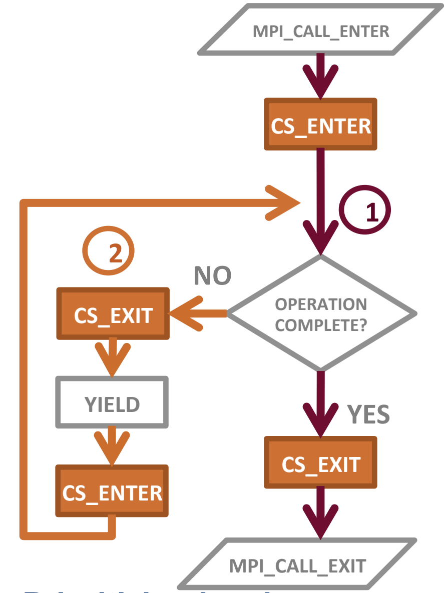
Smart Progress and Fast Hand-off

Adapt arbitration to maximize work

- FIFO locks overcome the shortcomings of mutexes
- Polling for progress can be wasteful (**waiting does not generate work**!)
- Prioritizing issuing operations
  - Feed the communication pipeline
  - Reduce chances of wasteful internal process (e.g. more requests on the fly ➔ higher chances of making progress)



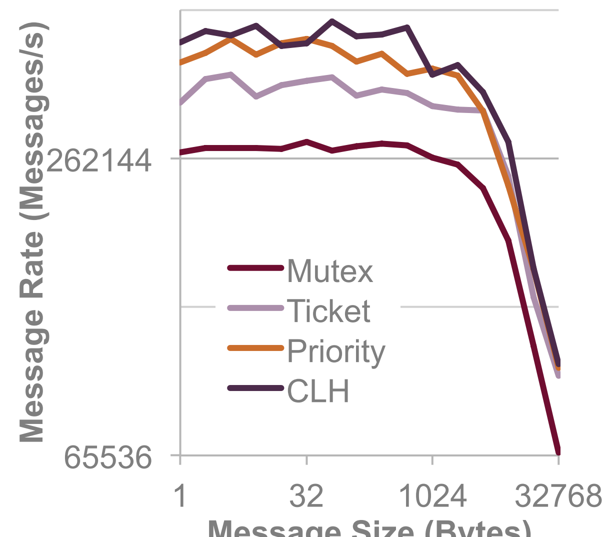
Fairness (FIFO) reduces wasted resource acquisitions



Prioritizing issuing ops over waiting for internal progress

Hand-off latency must be kept low

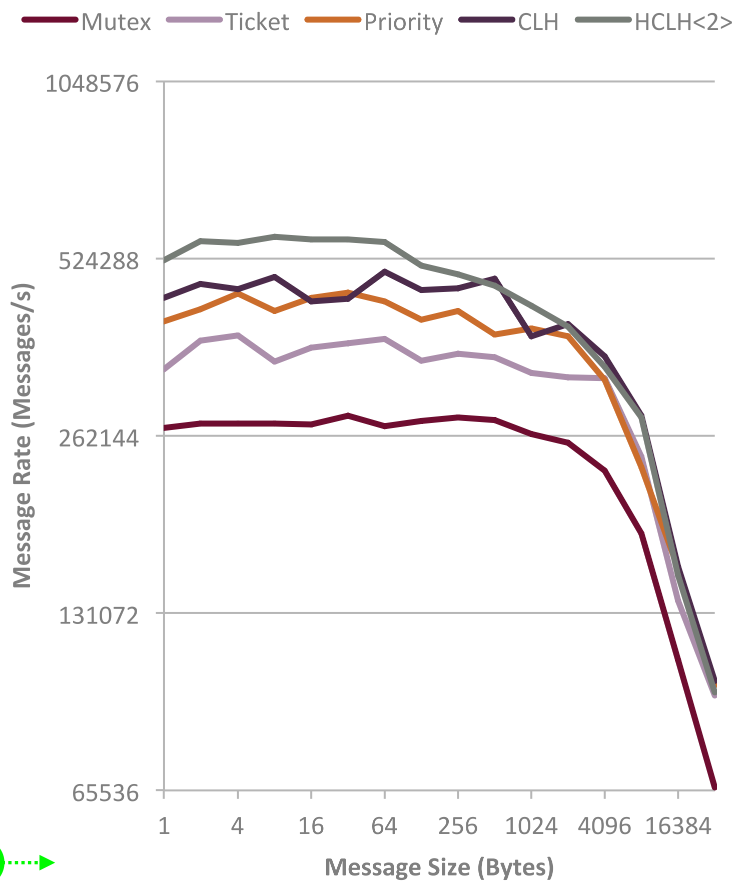
- Same arbitration cab be achieve with different locks (e.g. ticket, MCS, CLH, etc. are **FIFO)**
- Trade-off between desired arbitration and hand-off latency
  - E.g. for FIFO, CLH is more scalable than ticket
- Going through the kernel (e.g. using **futex**) is expensive and should not be done frequently
- A more scalable scalable **hierarchical lock** is being integrated

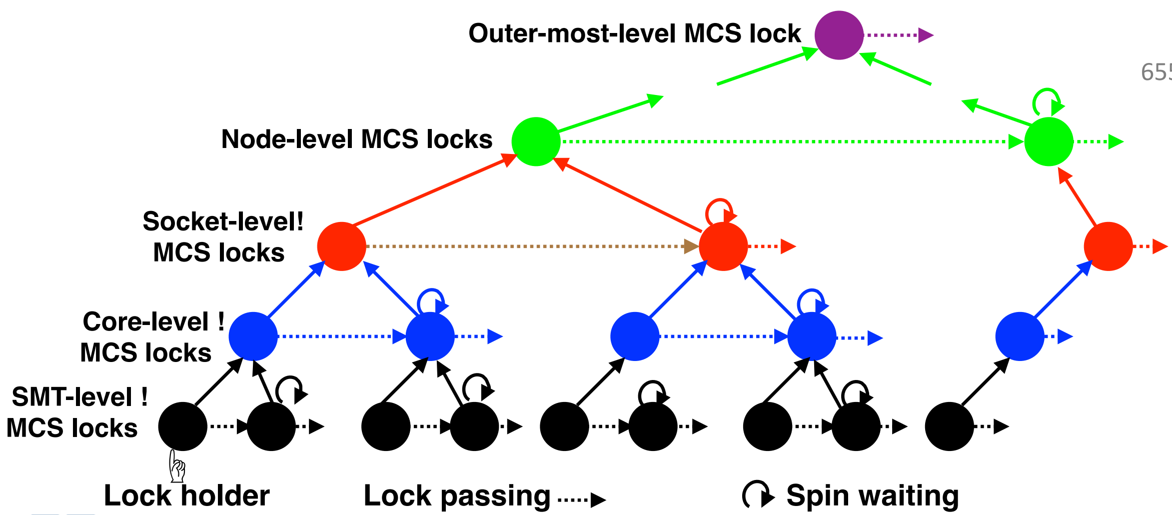


Pt2Pt message rate with 36 Haswell Cores and Mellanox FDR

Fairness and Cohort Locking

- Unfairness itself IS NOT EVIL
- **Unbounded** unfairness IS EVIL
- Unfairness can improve locality of reference
- Exploit locality of reference through cohorting
  - Threads sharing some level of the memory hierarchy
  - Local passing within a threshold



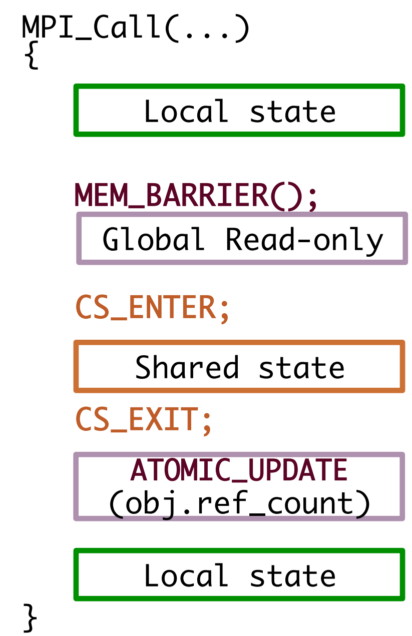


Hierarchical MCS Lock. Chabbi, Milind, Michael Fagan, and John Mellor-Crummey. "High performance locks for multi-level NUMA systems." Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM, 2015.

Granularity Optimization Being Considered

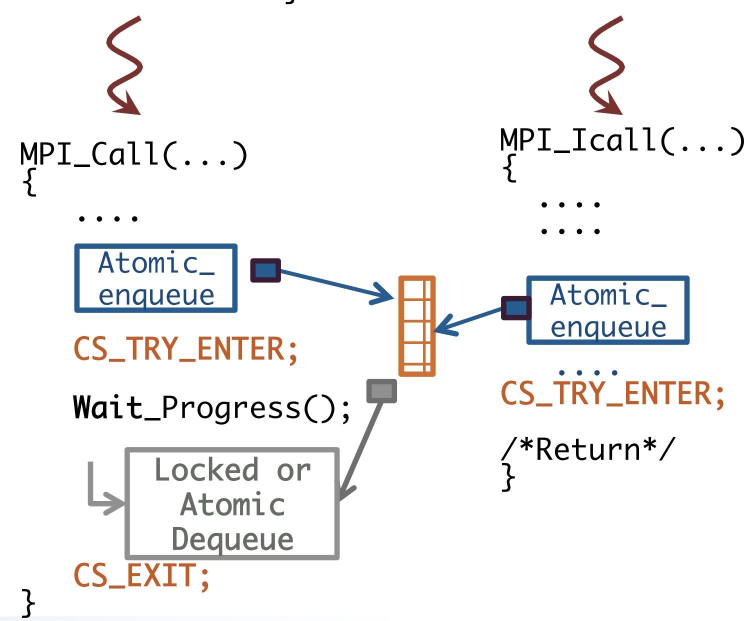
Combination of several methods to achieve fine granularity

- Locks and atomics are not always necessary
  - Memory barriers can be enough
  - E.g. only read barrier for `MPI_Comm_rank`
- Brief-global locking: only protect shared objects
- Lock-free wherever possible
  - Atomic reference count updates
  - Lock-free queue operations



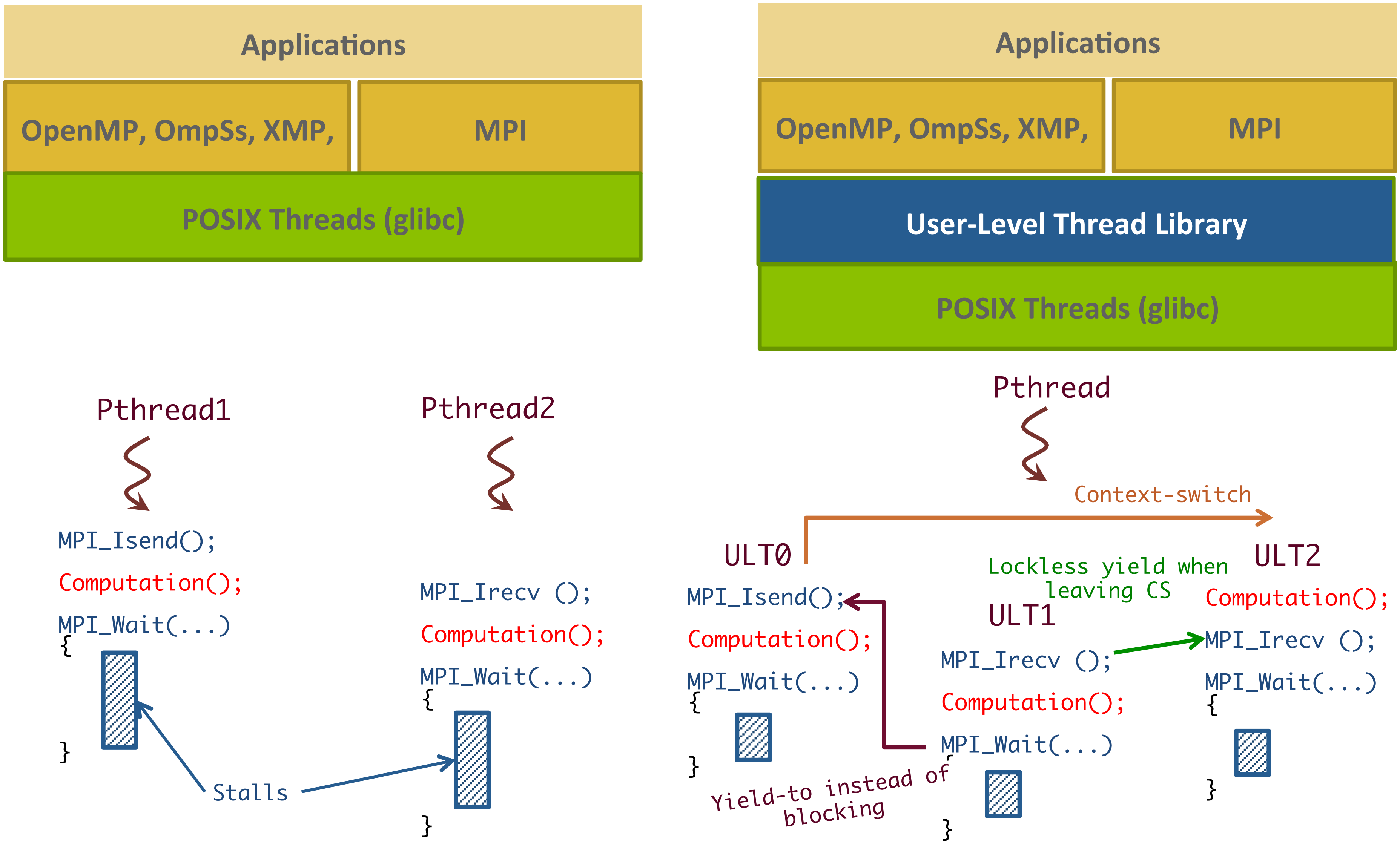
Moving towards a lightweight queue/dequeue model

- Reduce locking requirements and move to a mostly lock-free queue/dequeue model
- Reduce unnecessary progress polling for nonblocking operations
- Enqueue operations all atomic
- Dequeue operations
  - Atomic for unordered queues (RMA)
  - Fine-grained locking for ordered queues
- The result is a mostly lock-free model for nonblocking operations



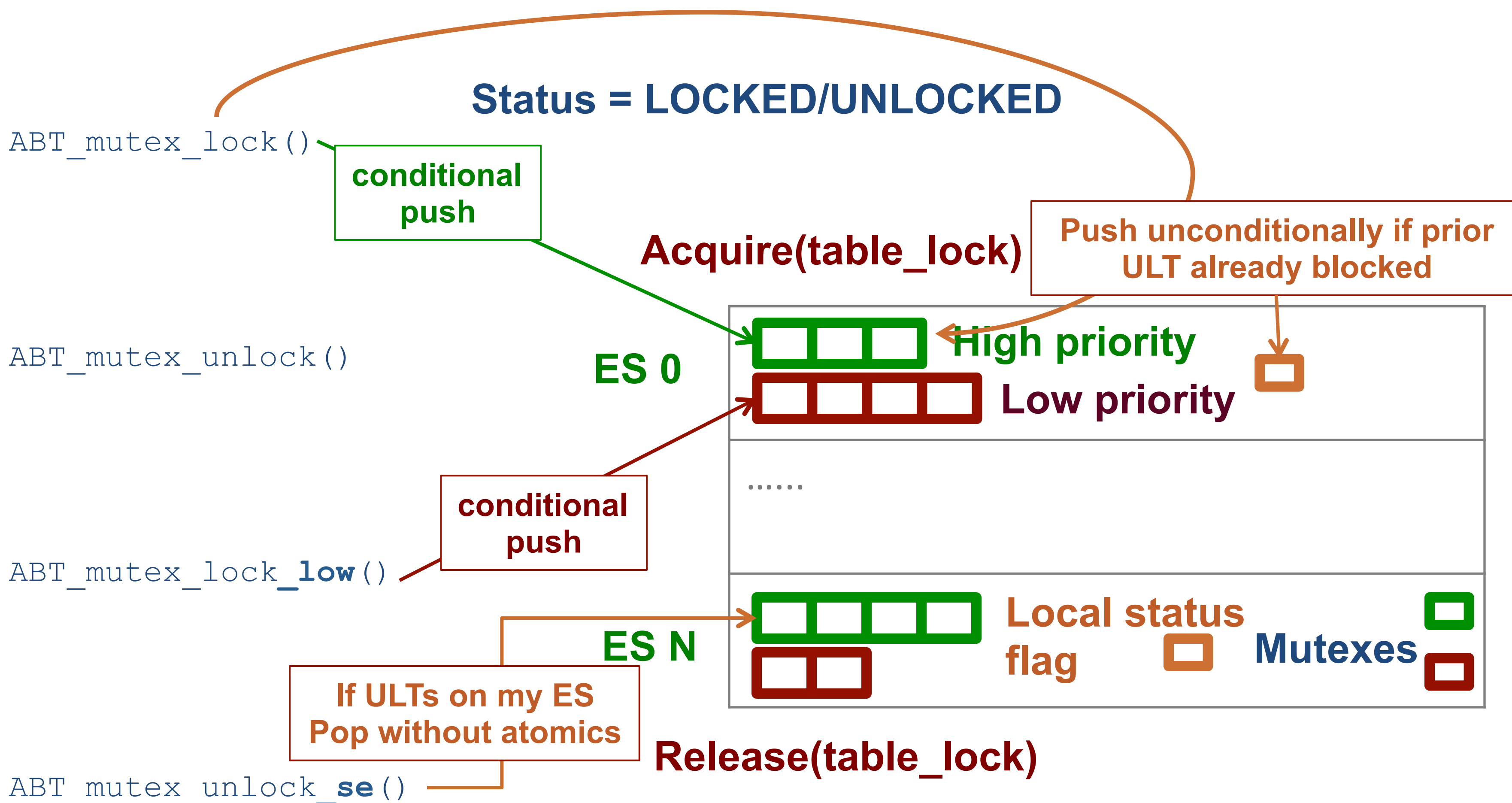
User-Level Threads Optimization Space

MPI+User-Level Threads Optimization Opportunities

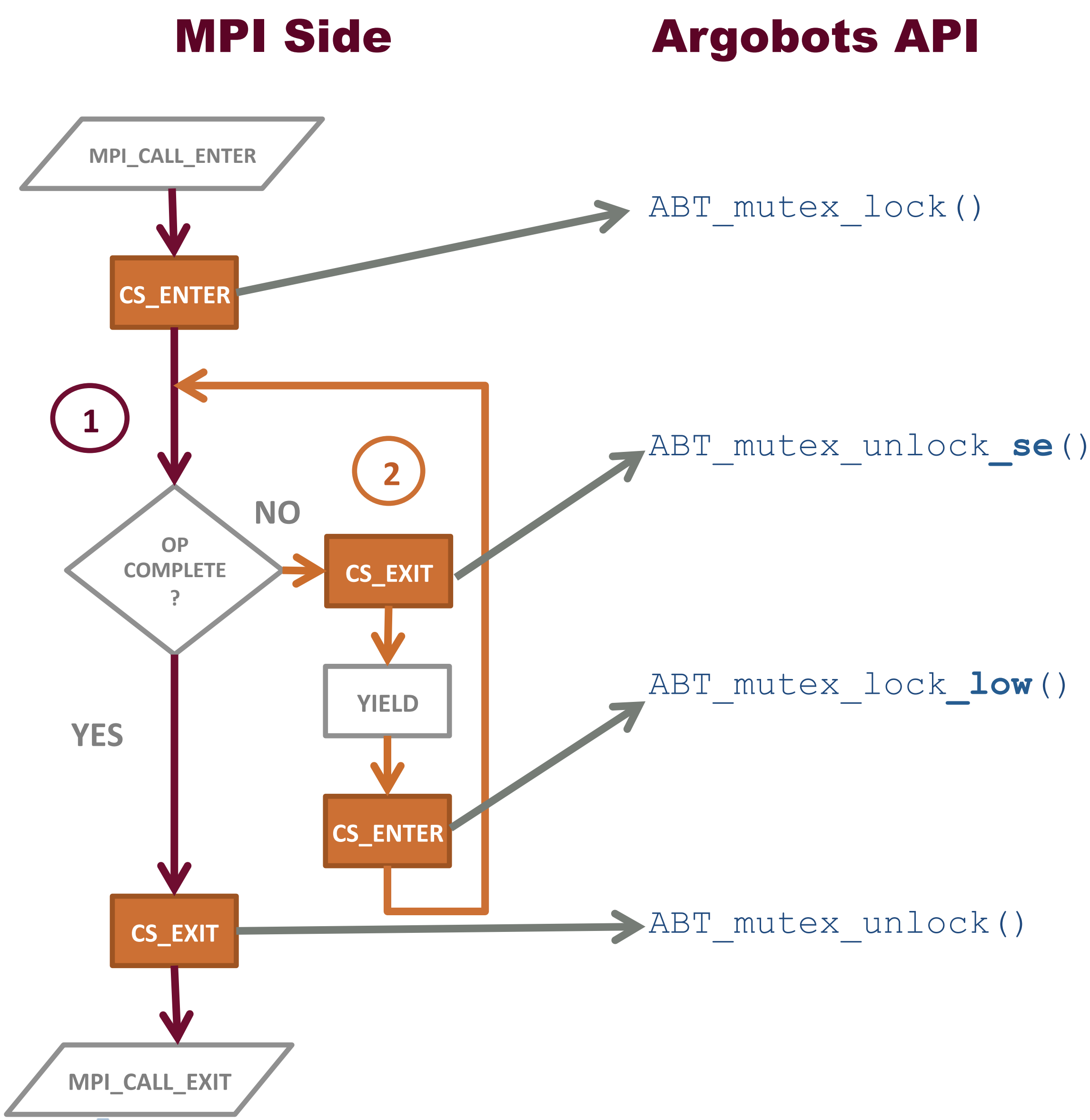


Advanced Argobots Locking Support

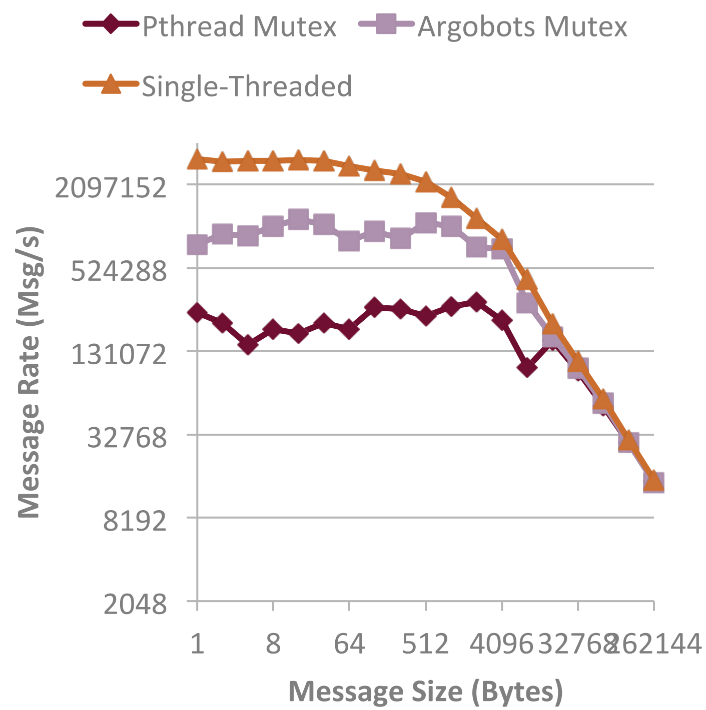
Argobots Mutex API **Argobots Mutex Data-Strctures**



Mapping Argobots Advanced Locking to MPI



Average of 4x improvement for fine-grained communication!



Pt2Pt message rate with 36 Haswell Cores and Mellanox FDR

Summary

To Take Out

- Why would you care about MPI+threads communication?
  - Applications, libraries, and languages are moving towards driving both computation and communication because of
    - Hardware constraints
    - Programmability
- The current situation
  - Multithreaded MPI communication is still not Exascale level
  - We improved upon existing runtimes but still more to go
- Insight into the future
  - Hopefully the fine-grained model will be enough for both OS-level and user-level threading
  - Otherwise, a combination of advanced thread-synchronization and threading runtimes will be necessary