

# Future of MPI

*Rajeev Thakur*

Deputy Director

Mathematics and Computer Science Division

Argonne National Laboratory

DOE Programming Models and Environments Workshop

March 9, 2015

# MPI Runs Successfully at Full Scale on all the Largest Systems of Today



# MPI on the Largest Systems

- Number of cores in largest systems

Tianhe-2 (China)	3,120,000 cores (mostly Xeon Phi cores)
Sequoia (LLNL)	1,572,864 cores
Mira (ANL)	786,432 cores
Blue Waters	773,632 cores (plus some GPU cores)
K computer	705,024 cores
Jülich BG/Q	458,752 cores
Titan (ORNL)	299,008 cores (plus GPU cores)

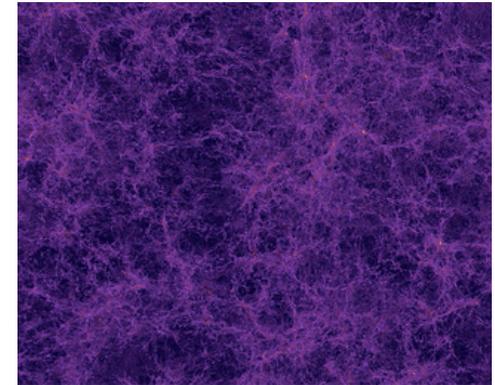
- Applications are running at full scale using either

- All MPI, or increasingly,
- MPI + X hybrid
  - X = threads (usually OpenMP) for regular cores
  - X = CUDA, OpenACC for GPU cores

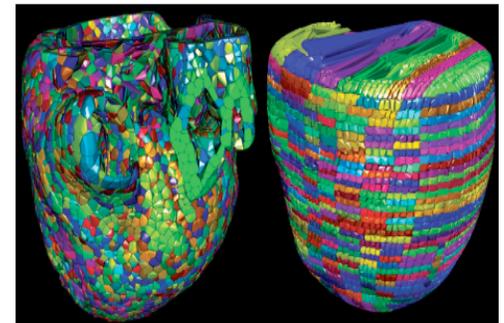


# Application Examples

- HACC Cosmology Code from Argonne (PI: Salman Habib)
  - 14 PFlops/s (70% of peak) on Sequoia
  - 16 MPI ranks \* 4 OpenMP threads per rank on each node
    - Matches h/w arch: 16 cores per node with 4 h/w threads each
  - ~ 6.3 million way concurrency: 1,572,864 MPI ranks \* 4 threads/rank



- Cardioid Cardiac Modeling Code (IBM and LLNL)
  - 12 PFlops/s on Sequoia
  - 1 MPI rank and 64 threads per node
  - OpenMP for thread creation only; other thread stuff uses custom code
  - Performance-critical communication done directly with IBM's SPI low-level layer; other communication with MPI



# Other Examples

- ROSS parallel discrete-event simulator (Chris Carothers, RPI)
  - Uses MPI-only, no threads
  - Ran on ~2 million cores of Sequoia and Vulcan systems combined
    - $1,966,080 \text{ cores} * 4 \text{ MPI ranks/core} = 7,864,320 \text{ MPI ranks}$
  - Achieved the highest event rate ever reported (504 billion events/sec) on PHOLD benchmark
- 100 million MPI processes using FG-MPI
  - Fine-Grain MPI: Prof. Alan Wagner, Univ. of British Columbia
  - An MPICH derivative that implements MPI processes as coroutines rather than regular OS processes
  - Ran 100 million MPI ranks on 6,480 processor cores of a cluster

=> Applications are flexibly choosing the number of MPI processes and threads as suits their application and architecture, and running at full scale



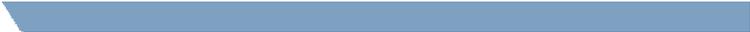
All benchmarks for the DOE CORAL procurement (2017/18) use MPI or MPI+X  
<https://asc.llnl.gov/CORAL-benchmarks/>

=> The application workload on the CORAL systems is expected to be largely MPI or MPI + X

=> MPI implementations will need to scale and run efficiently on these systems

Scalable Science Benchmarks	Priority Level	Lines of Code	Parallelism		Language			
			MPI	OpenMP/ Pthreads	Fortran	Python	C	C++
<a href="#">LSMS</a>	TR-1	200,000	X	X	X			X
<a href="#">QBOX</a>	TR-1	47,000	X	X				X
<a href="#">HIACC</a>	TR-1	35,000	X	X				X
<a href="#">Nekbone</a>	TR-1	48,000	X		X		X	
Throughput Benchmarks								
	Priority Level	Lines of Code	Parallelism		Language			
			MPI	OpenMP/ Pthreads	Fortran	Python	C	C++
<a href="#">CAM-SE</a>	TR-1	150,000	X	X	X		X	
<a href="#">UMT2013</a>	TR-1	51,000	X	X	X	X	X	X
<a href="#">AMG2013</a>	TR-1	75,000	X	X			X	
<a href="#">MCB</a>	TR-1	13,000	X	X			X	
<a href="#">QMCPACK</a>	TR-2	200,000	X	X			X	X
<a href="#">NAMD</a>	TR-2	180,000	X	X				X
<a href="#">LULESH</a>	TR-2	5,000	X	X			X	
<a href="#">SNAP</a>	TR-2	3,000	X	X	X			
<a href="#">miniFE</a>	TR-2	50,000	X	X				X
Data-Centric Benchmarks								
	Priority Level	Lines of Code	Parallelism		Language			
			MPI	OpenMP/ Pthreads	Fortran	Python	C	C++
<a href="#">Graph500</a>	TR-1			X			X	
<a href="#">Integer Sort</a>	TR-1	2,000	X		X		X	
<a href="#">Hash</a>	TR-1		X		X		X	
<a href="#">SPECint2006 "peak"</a>	TR-2				X		X	X
Skeleton Benchmarks								
	Priority Level	Lines of Code	Parallelism		Language			
			MPI	OpenMP/ Pthreads	Fortran	Python	C	C++
<a href="#">CLOMP</a>	TR-1			X			X	
<a href="#">IQR</a>	TR-1	4,000	X				X	
<a href="#">CORAL MPI benchmarks</a>	TR-1	1,000	X				X	
<a href="#">Memory benchmarks STREAM   STRIDE</a>	TR-1	1,500			X		X	
<a href="#">LCALS</a>	TR-1	5,000		X				X
<a href="#">Pynamic</a>	TR-2	12,000	X			X		X
<a href="#">HIACC IO</a>	TR-2	2,000	X					X
<a href="#">FTQ</a>	TR-2	1,000					X	
<a href="#">XSBench (mini OpenMC)</a>	TR-2	1,000		X			X	
<a href="#">MiniMADNESS</a>	TR-2	10,000	X	X				X





# Exascale Architectures and their Implications for MPI



# Reference: CAL Report

## Abstract Machine Models and Proxy Architectures for Exascale Computing

Rev 1.1

J.A. Ang<sup>1</sup>, R.F. Barrett<sup>1</sup>, R.E. Benner<sup>1</sup>, D. Burke<sup>2</sup>,  
C. Chan<sup>2</sup>, D. Donofrio<sup>2</sup>, S.D. Hammond<sup>1</sup>,  
K.S. Hemmert<sup>1</sup>, S.M. Kelly<sup>1</sup>, H. Le<sup>1</sup>, V.J. Leung<sup>1</sup>,  
D.R. Resnick<sup>1</sup>, A.F. Rodrigues<sup>1</sup>,  
J. Shalf<sup>2</sup>, D. Stark<sup>1</sup>, D. Unat<sup>2</sup>, N.J. Wright<sup>2</sup>



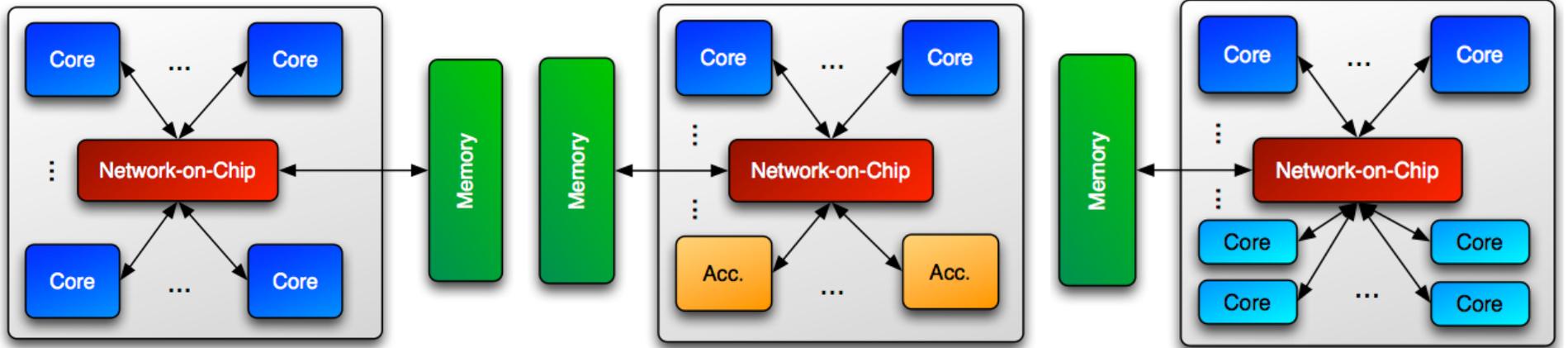
Sandia National Laboratories, NM<sup>1</sup>  
Lawrence Berkeley National Laboratory, CA<sup>2</sup>

May, 16 2014

Available from <http://www.cal-design.org/publications/publications2>



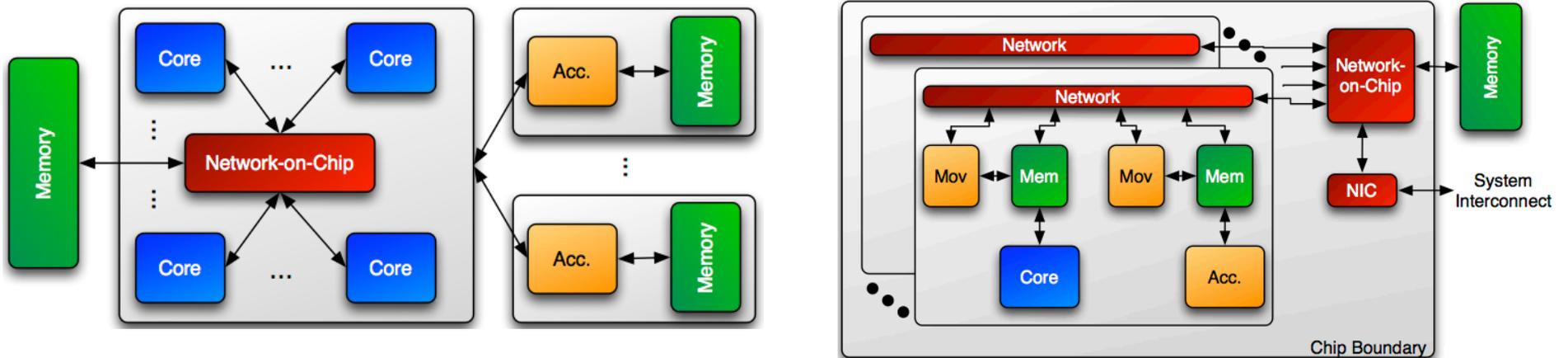
# Node Architecture Types



Homogeneous Many-core

Integrated CPU & Accelerators

Heterogeneous Multicore



Multicore CPU with Discrete Accelerators

Custom



# Design Parameters

	Processor Cores	GFlops/s per Proc Core	Accel Cores	Acc Count per Node	TFlops/s per Node	Node Count
Homogeneous M.C. Opt1	256	64	--	--	16	62,500
Homogeneous M.C. Opt2	64	250	--	--	16	62,500
Discrete Acc. Opt1	32	250	O(1000)	4	16C + 2A	55,000
Discrete Acc. Opt2	128	64	O(1000)	16	8C + 16A	41,000
Integrated Acc. Opt1	32	64	O(1000)	Integrated	30	33,000
Integrated Acc. Opt2	128	16	O(1000)	Integrated	30	33,000
Heterogeneous M.C. Opt1	16 / 192	250	--	--	16	62,500
Heterogeneous M.C. Opt2	32 / 128	64	--	--	16	62,500
Concept Opt1	128	50	128	Integrated	6	125,000
Concept Opt2	128	64	128	Integrated	8	125,000

Source: CAL Report

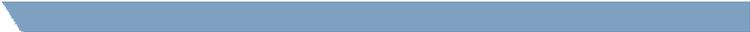


# Cores Per Node and Total Number of Nodes

- For design purposes, let's use the maximum numbers of cores per node and total number of nodes from previous slide, and assume 8 hardware threads per core

Total number of nodes	125,000
Processor cores per node	256
Hardware threads per core	8
Hardware threads per node	2048
Total concurrency	256 million





# Implications for MPI



# Main Differences from Today's Largest Systems (for MPI)

Total number of nodes	$O(125,000) \Rightarrow 1-4x$
Hardware threads per node	$O(2048) \Rightarrow 32-64x$
Total concurrency	$O(256 \text{ million}) \Rightarrow 40-120x$



# For the Total Number of Nodes

- 125,000 nodes is not a problem
  - Sequoia already has 98,304 nodes
  - Sequoia and Vulcan combined together have 122,880 nodes and have been run as a single system (see slide 5)
- Scaling the number of nodes requires scaling in the distributed memory sense.
  - Better, scalable collectives
  - Scalable data structures
  - Efficient RMA
  - Resilience
  - ...



# For the 32-64x Number of Hardware Threads/Node

- Needs MPI+X hybrid or even MPI+X+Y
  - Applications are already realizing this
- X/Y can be any of OpenMP, Pthreads, OpenACC, CUDA, ...
- X can also be MPI as some people have recognized
  - MPI-3 has added support for shared-memory programming
  - See **“MPI+MPI: A New, Hybrid Approach to Parallel Programming with MPI Plus Shared Memory Computing,”** Hoefler et al., *Computing*, 2013
- Needs better support from MPI for hybrid programming
  - Such as the “endpoints” proposal being discussed in the MPI Forum (see slide 17)



# For the 40-120x Total Concurrency

- It should be considered as two sub-problems and solved accordingly
  - How many MPI processes
  - How many threads per MPI process
- For example, 256 million total concurrency could be implemented as
  - 16 million MPI processes and 16 threads per process, or
  - 4 million MPI processes and 64 threads per process
- Both cases are manageable by
  - Improving MPI implementations to use memory scalably (memory-efficient data structures)
  - Using MPI-3 shared-memory constructs and the new endpoints proposal for efficient hybrid programming
  - For a particular application, picking the best performing combination of  $(n \times m)$ , where  $n$  = number of MPI processes,  $m$  = threads per process
- The new fault tolerance proposal in MPI will help support the increased need for resilience (see slide 18)



# Better Hybrid Programming: Extending MPI to Support Multiple Endpoints Per Process

- In MPI today, each process has a single communication endpoint (rank in `MPI_COMM_WORLD`)
- Multiple threads of a process communicate through that single endpoint, requiring the implementation to use locks etc., which are expensive
- MPI Forum is discussing a proposal (for MPI-4) that allows a process to have multiple endpoints
- Threads within a process can attach to different endpoints and communicate through those endpoints as if they are separate ranks
- The MPI implementation can avoid using locks if each thread communicates on a separate endpoint
- This allows the MPI standard to support “MPI + X” more efficiently without specifying what X is



# Improved Support for Fault Tolerance

- MPI always had support for error handlers and allows implementations to return an error code and remain alive
- MPI Forum working on additional support for MPI-4
- Current proposal handles fail-stop process failures (not silent data corruption or Byzantine failures)
  - If a communication operation fails because the other process has failed, the function returns error code `MPI_ERR_PROC_FAILED`
  - User can call `MPI_Comm_shrink` to create a new communicator that excludes failed processes
  - Collective communication can be performed on the new communicator
  - Lots of other details in the proposal...



# MPI is not BSP

- Some applications follow a BSP model and use MPI for communication
- But there are other applications that don't follow a BSP model and still use MPI

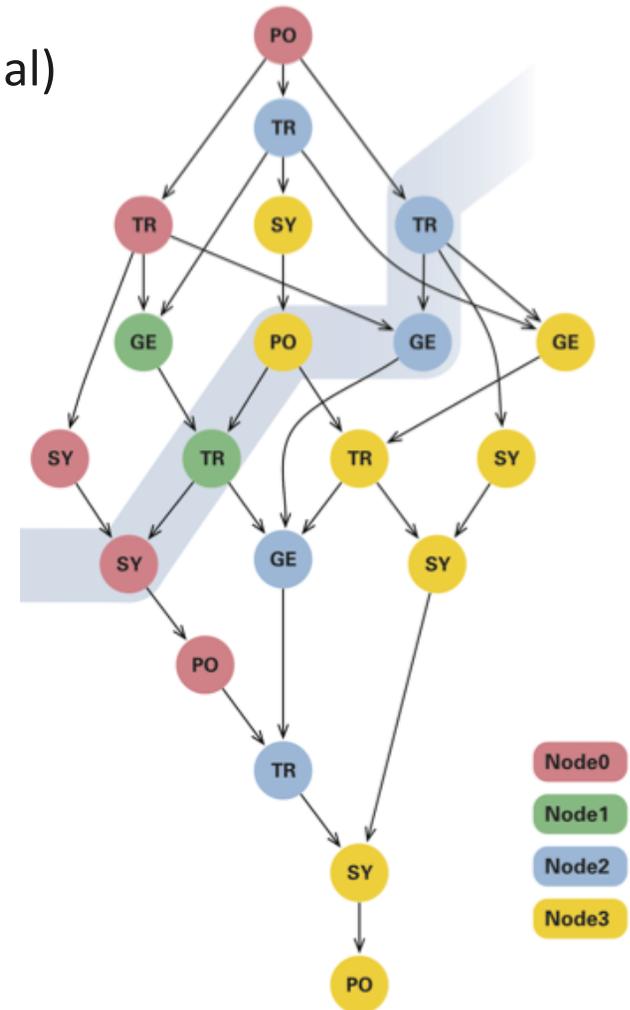
## *Examples*

- GFMC code, part of NUCLEI SciDAC-3 project
  - Uses a completely task-based, fully asynchronous programming model provided by the ADLB library, which is implemented on top of MPI
  - Distributed, global pool of tasks
  - Processes create work and (asynchronously) add it to the pool
    - Tasks have types, priorities
  - Other processes get work asynchronously from the pool
  - No global synchronizations
  - Has efficiently used 250,000 MPI ranks on Mira, Argonne's Blue Gene/Q



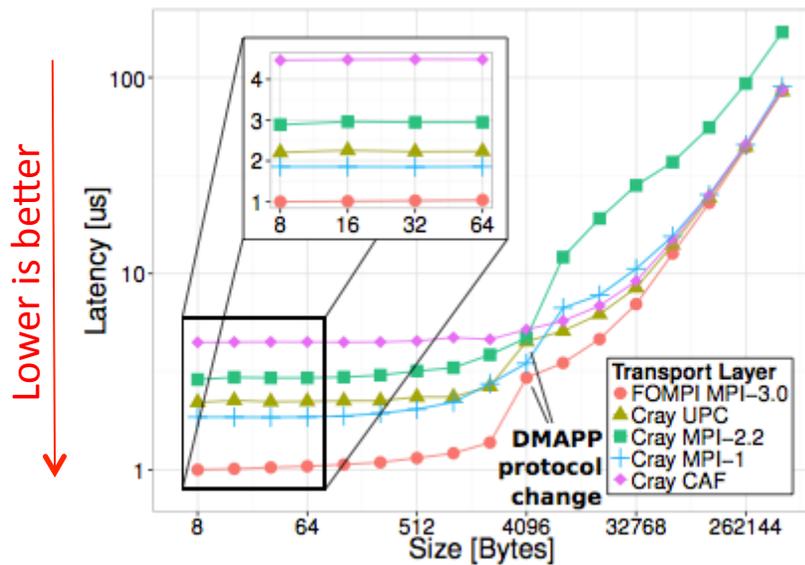
# Another non-BSP use of MPI

- PaRSEC Framework from UTK (Bosilca, Dongarra, et al)
- Generic framework for task-based programming on distributed many-core heterogeneous architectures
- Applications expressed as DAGs of tasks with edges representing data dependencies
- Entire DPLASMA dense linear algebra library implemented on top of PaRSEC
  - Linear system solvers, least squares, eigen value, level-3 BLAS, etc.
- PaRSEC uses MPI for communication

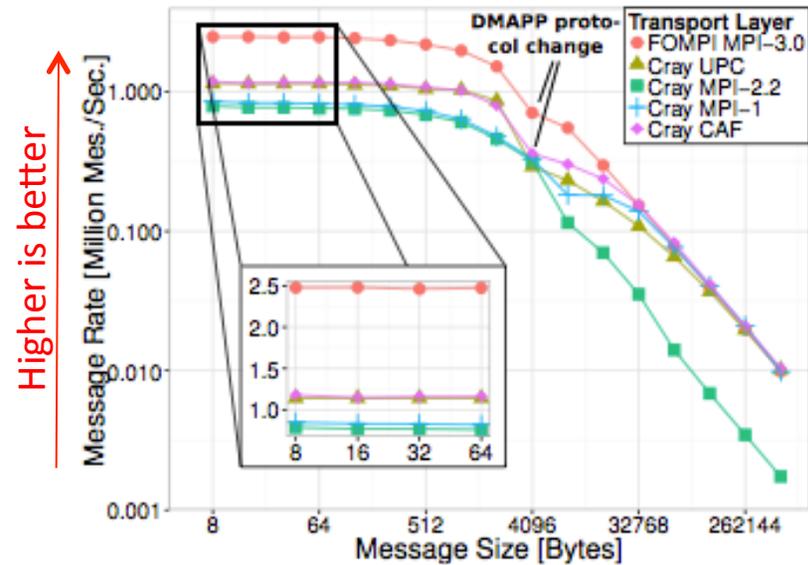


# MPI-3 RMA can be implemented efficiently

- “Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided” by Robert Gerstenberger, Maciej Besta, Torsten Hoefler (SC13 Best Paper Award)
- They implemented complete MPI-3 RMA for Cray Gemini (XK5, XE6) and Aries (XC30) systems on top of lowest-level Cray APIs
- Achieved better latency, bandwidth, message rate, and application performance than Cray’s UPC and Cray’s Fortran Coarrays



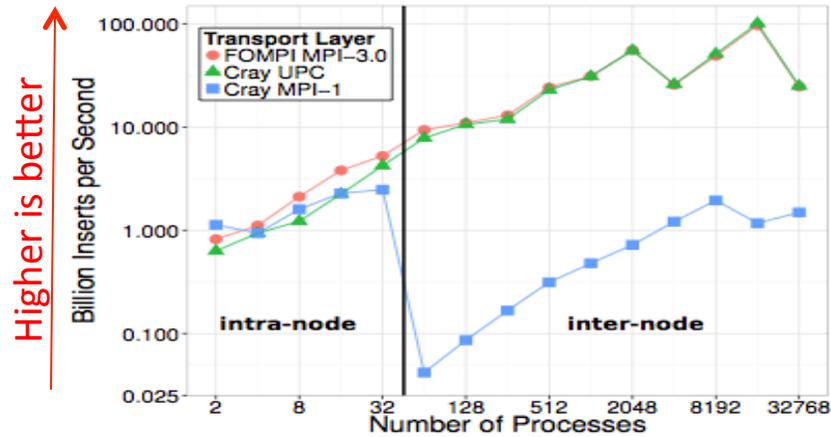
(a) Latency inter-node Put



(b) Message Rate inter-node

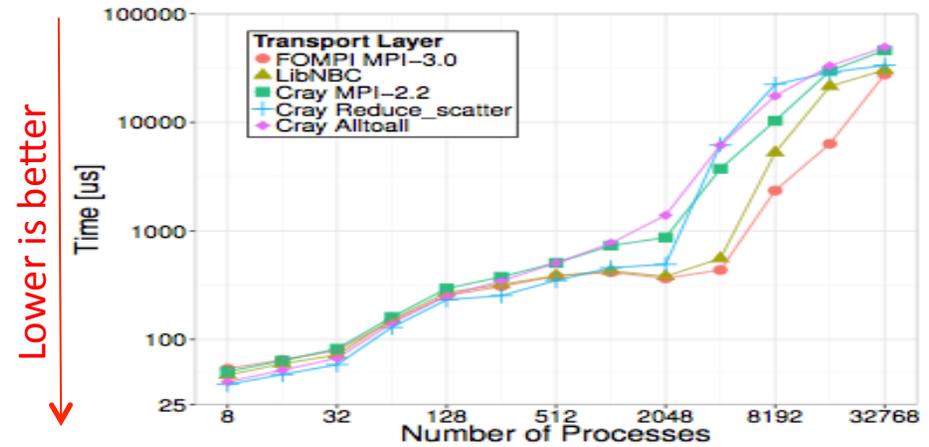


# Application Performance with Tuned MPI-3 RMA



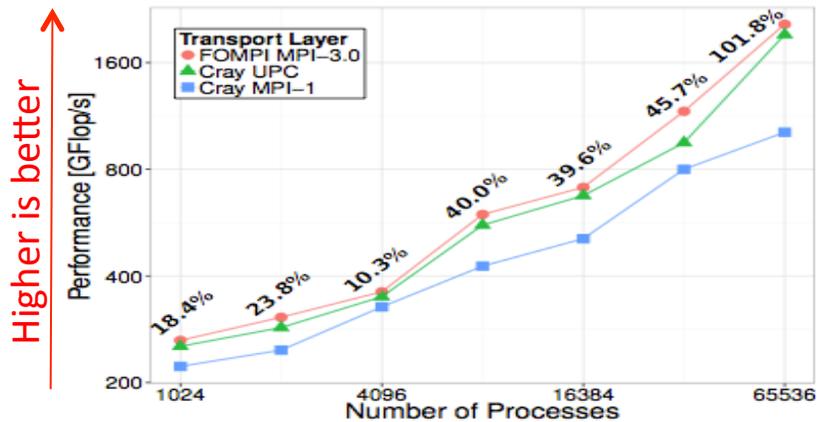
(a) Inserts per second for inserting 16k elements per process including synchronization.

Distributed Hash Table



(b) Time to perform one dynamic sparse data exchange (DSDE) with 6 random neighbors

Dynamic Sparse Data Exchange



(c) 3D FFT Performance. The annotations represent the improvement of FOMPI over MPI-1.

3D FFT

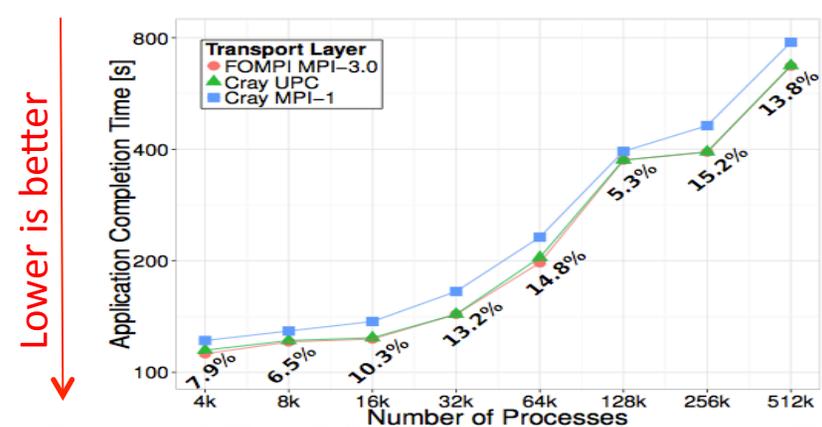


Figure 8: MILC: Full application execution time. The annotations represent the improvement of FOMPI and UPC over MPI-1.

MILC

# MPI RMA is Carefully and Precisely Specified

- To work on both cache-coherent and non-cache-coherent systems
  - Even though there aren't many non-cache-coherent systems, it is designed with the future in mind
- There even exists a *formal model* for MPI-3 RMA that can be used by tools and compilers for optimization, verification, etc.
  - See “Remote Memory Access Programming in MPI-3” by Hoefler, Dinan, Thakur, Barrett, Balaji, Gropp, Underwood. To appear in ACM TOPC, 2015.
  - <http://hlor.inf.ethz.ch/publications/index.php?pub=201>



# What's New in MPI-3

- Many enhancements for scalability, such as distributed graph topologies, support for symmetric memory allocation in RMA, nonblocking collectives
- Major improvements to one-sided communication, including
  - Atomic operations, such as compare-and-swap and fetch-and-add
  - New memory model with simplified consistency semantics
  - Support for allocating and accessing shared memory within a node
- Nonblocking collectives
  - MPI\_Ibcast, MPI\_Ibarrier, MPI\_Ireduce, and all other collectives
- Neighborhood collectives (and their nonblocking versions)
  - Communication among nearest neighbors (e.g., stencil) can be expressed as a collective communication
- Extensive interface for tools to portably access performance variables, or set control variables, in an MPI implementation
- Bindings for Fortran 2008
- Many other miscellaneous items...



# MPI-3 Implementations are already available

	MPICH	MVAPICH	Open MPI	Cray MPI	Tianhe MPI	Intel MPI	IBM BG/Q MPI <sup>1</sup>	IBM PE MPICH <sup>2</sup>	IBM Platform	SGI MPI	Fujitsu MPI	MS MPI
NB collectives	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	*
Neighborhood collectives	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	
RMA	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	
Shared memory	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	✓
Tools Interface	✓	✓	✓	✓	✓	✓	✓ <sup>1</sup>	✓	Q3 '15	✓	Q2 '15	*
Non-collective comm. create	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	
F08 Bindings	✓	✓	✓	✓	✓	Q2 '15	✓	Q? '15	Q3 '15	✓	Q2 '15	
New Datatypes	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	*
Large Counts	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	*
Matched Probe	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	✓	*

Release dates are estimates and are subject to change at any time.

Empty cells indicate no *publicly announced* plan to implement/support that feature.

<sup>1</sup> No MPI\_T variables exposed

\* Under development

Platform-specific restrictions might apply for all supported features

# MPI is an Evolving Standard

=> *If some feature is missing, it can be added.*

- MPI Forum is active and is working on various new features that will be useful for exascale
- The Forum meets about four times a year
  - March 2-5, 2015 (Portland)
  - June 1-4, 2015 (Chicago)
  - Sept 24-26, 2015 (Bordeaux, co-located with EuroMPI)
  - Dec 7-10, 2015 (San Jose)
- Anyone can participate
  - All meetings, working groups, working drafts, telecons are fully open
- Martin Schulz (LLNL) is the chair of the MPI Forum
- MPI 3.1 should be out by June this year



# Summary

- MPI has succeeded for many reasons. If I had to pick one reason:
  - It has enabled many highly tuned libraries (e.g., PETSc, Trilinos, FFTW, Chombo) that make life easier for application developers
  - “Productivity” comes from using these libraries
- MPI community needs to better understand the needs of advanced higher-level programming models so that MPI can be used as a basis for their implementation
- MPI can scale to exascale systems
  - But work is needed in both the MPI specification and in MPI implementations

