MPI usage at NERSC: Present and Future

Alice Koniges, Brandon Cook, Jack Deslippe, Thorston Kurth, Hongzhang Shan
Lawrence Berkeley National Laboratory, Berkeley, CA USA

Abstract: We describe how MPI is used at the National Energy Research Scientific Computing Center (NERSC). NERSC is the production high-performance computing center for the US Department of Energy, with more than 5000 users and 800 distinct projects. We also compare the usage of MPI to exascale developmental programming models such as UPC++ and HPX, with an eye on what features and extensions to MPI are plausible and useful for NERSC users. We also discuss perceived shortcomings of MPI, and why certain groups use other parallel programming models. We believe our results tend to represent the advanced MPI NERSC user base, rather than an extensive survey of the entire NERSC code group population.

Special thanks to Dr. Rolf Rabenseifner and colleagues at Stuttgart High Performance Computing Center for input in making the MPI NERSC User Survey.

NERSC Hardware

Cori is NERSC's newest supercomputer system (NERSC-8). It is named after American biochemist Gerty Cori, the first woman to win a Nobel Prize in Physiology or Medicine.

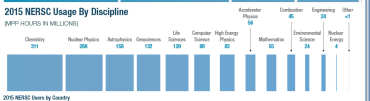
Cori Phase 1 system is a Cray XC based on the Intel Haswell multi-core processor. Cori Phase 2 is based on the second generation of the Intel® Xeon Phi™ family of products, called the Knights Landing (KNL) Many Integrated Core (MIC) Architecture. Cori Phase 1 & Cori Phase II integration begins ~September 19, 2016 (for ~6 weeks).

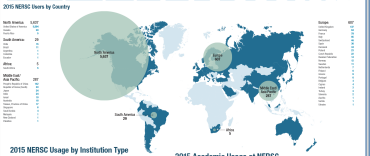
- Cori is a Cray XC40 supercomputer
- Theoretical Peak performance: Phase I Haswell: 1.92 PFlops/sec; Phase II KNL: 27.9 PFlops/sec.
- Total compute nodes: Phase I Haswell: 1,630 computes nodes, 52,160 cores in total (32 cores per node);
  Phase II KNL: 9,304 compute nodes, 632,672 cores in total (68 cores per node).
- Cray Aries high-speed interconnect with Dragonfly topology (0.25 µs to 3.7 µs MPI latency, ~8GB/sec MPI bandwidth)
- Aggregate memory: Phase I Haswell partition: 203 TB; Phase II KNL partition: 1 PB.
- Scratch storage capacity: 30 PB

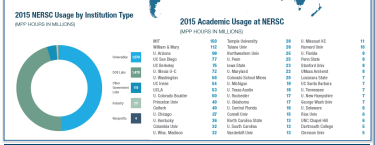
Edison has been the NERSC's primary supercomputer system for a number of years. It is named after U.S. inventor and businessman Thomas Alva Edison.

Edison, a Cray XC30, has a peak performance of 2.57 petaflops/sec, 133,824 compute cores for running scientific applications, 357 Terabytes of memory, and 7.56 Petabytes of online disk storage with a peak I/O bandwidth of 168 gigabytes (GB) per second.

NERSC Research Areas and Users







Snapshot of User Comparisons of MPI to other Programming Models at NERSC

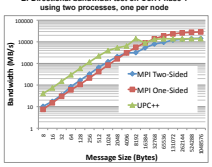
NERSC users are trying a variety of programming models and often comparing them to MPI. While these comparisons are obviously implementation and application dependent, we give here some current results. (See links in MPI Alternatives Survey for more results.)

UPC++: A PGAS Extension for C++

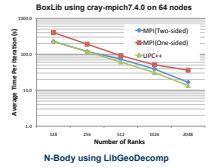
Features	UPC++
Global Pointer (type T)	global_ptr<T>
Global Memory Allocation	allocate<T>(rank_size)
Contiguous Data Copy	async_copy(src, dest, count)
Remote operation	Async_after(rank, handler)
Global Synchronization	barrier()
Shared Memory support	global_ptr<T> p_is_local()
Remote/Signal Data Copy	async_copy_after_()
Group Synchronization	sync_neighbors(list, length)

Zheng, Yili, et al. "UPC++: a PGAS Extension for C++." Parallel and Distributed Processing Symposium, 2014

Bi-Directional bandwidth test on Cori Phase 1 using two processes, one per node



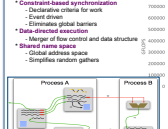
BoxLib using cray-mpich7.4.0 on 64 nodes



BoxLib: An AMR Software Framework for building parallel adaptive mesh refinement (AMR) codes solving multi-scale/multi-physics partial differential equations
Available at https://github.com/BoxLib-Codes
Managed by Weiqun Zhang, et al.
C++ and Fortran versions
MPI + OpenMP (+ UPC++)

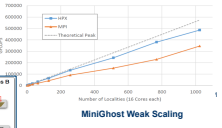
High Performance ParalleX (HPX)

- **Lightweight multi-threading**
  - Divides work into smaller tasks
  - Increases concurrency
- **Message-driven computation**
  - Move work to data
  - Keeps work local, stops blocking
- **Constraint-based synchronization**
  - Declarative criteria for work
  - Event driven
  - Eliminates global barriers
- **Data-directed execution**
  - Merger of flow control and data structure
- **Shared name space**
  - Global address space
  - Simplifies random gathers

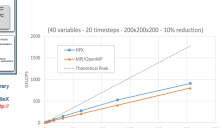


HPX-5 is the High Performance ParalleX runtime library from Indiana University. (http://hpx.crest.iu.edu).
HPX-3 is the C++11/14 implementation of ParalleX execution model from Louisiana State University (http://stellar-group.org/libraries/hpx/).

N-Body using LibGeoDecomp



MiniGhost Weak Scaling

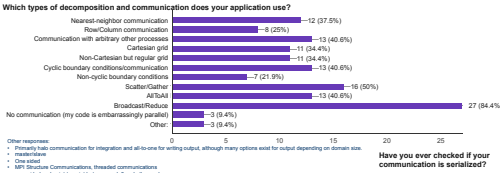


Comments on Alternatives to MPI

- In addition to UPC++ and HPX, NERSC users are using UPC, Charm++, Kokkos, Chapel, Coarray Fortran, OCR – based, Legion, Parallel Python, Hadoop/MapReduce, SHMEM, OpenMP, GASNet, Global Arrays , and GPI-2 / GASPI
- Performance benefit from using these parallel programming model(s)/approach(s) is generally not dramatic and varies with the application, the programming model, implementation and the hardware but can show definite improvement
- As seen in our survey on MPI Alternatives there are additional reasons people are citing for trying other methods
- MPI is being used in combination with a wider range of other parallel approaches as seen in our MPI User Survey
- Experiences gained by using MPI alternatives and MPI + X can be used to improve future versions of MPI
- Results with these models as well as the one-sided implementation of MPI are preliminary and implementation dependent

How MPI is used at NERSC

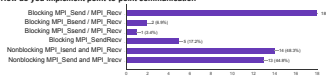
Which types of decomposition and communication does your application use?



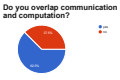
Other responses:
- Primarily halo communication for integration and all-to-one for writing output, although many options exist for output depending on domain size.
- master/slave
- One sided
- MPI Structure Communications, threaded communications
- one-sided and point-to-point between rank 0 and other ranks
- Graph traversal / random access
- I master PE for I/O ; generally this is minor amount of time
- Ideally all meaningful patterns are benchmarked

Have you ever checked if your communication is serialized?

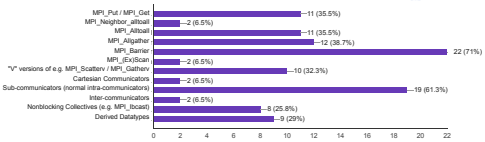
How do you implement point-to-point communication



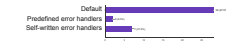
Do you overlap communication and computation?



Do you use any of these features?

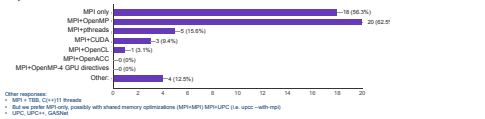


What error handling do you use?



How do you do MPI initialization?

Do you use MPI+X?



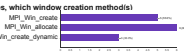
Other responses:
- MPI + TBB, CU++11 threads
- But we prefer MPI-only, possibly with shared memory optimizations (MPI+MPI) MPI+UPC (i.e. upcc --with-mpi)
- UPC, UPC++, GASNet

If you combine MPI with another communication model, e.g., MPI + Co-Array Fortran, MPI + UPC, etc., which models and why?

- Trying to switch completely to Coarray Fortran for ease of programming.
- MPI + SHMEM
- load balancing; outside of NERSC to allow reduced memory footprint to allow execution on Xeon Phi (KNL)
- MPI + Global Arrays
- OpenMP does not work across nodes
- UPC mainly used to test interoperability and due to UPC's better one-sided performance and productivity
- Ideally I'd use MPI just to get the processes spread across nodes, and use simpler internal to python multiprocessing for additional parallelism

Do you use one-sided communication (normal windows)?

If yes, which window creation method(s)

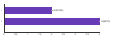


If yes, do you use overlapping windows?

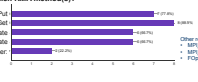
If you use overlapping windows, please specify the reason

- See Casper papers
- I use Casper for RMA asynchronous progress which requires overlapped windows to offload user RMA communication to background processes.

If yes to one-sided communication with normal windows, which RMA memory model?



If yes to one-sided communication with normal windows, which RMA method(s)?



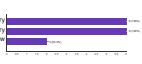
Other responses:
- MPI_Fetch_and_op
- MPI_Fetch_and_op
- FOp and CAS

If yes to one-sided communication with normal windows, which synchronization methods?

Other lock/flush/sync routines responses:
- Lock all, unlock all, ush local, ush, win sync, unlock all
- MPI_Win_lock_all/unlock_all/ ush_all/ ush_local_all
- MPI_Win_ush(_local)_all()

Do you use MPI shared memory windows (allocated with MPI_Win_allocate_shared)?

If yes, do you use

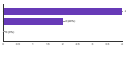


Which synchronization method do you use?

- MPI_Barrier() in MPI and sync all in CAF
- win sync or ush, as appropriate
- MPI_Win_lock_all/unlock_all/ ush_all/ ush_local_all
- MPI_Barrier, plus MPI_Send/Recv of empty messages
- MPI_WAIT MPI_WAITALL
- MPI_Win_sync for shared memory windows
- multiple methods

Do you use hints via the info object?

If yes, do you

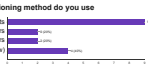


Do you use MPI-IO?

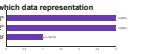
If yes, can you specify which MPI_File_write/read routines

- read/write_at_all() and read/write_at()
- MPI_FILE_WRITE_ALL
- MPI_File_read_all and MPI_File_write_all
- write_at_all_write, read at
- MPI_File_write_at, MPI_File_read_at
- MPI_FILE_WRITE_AT_ALL

If yes, which positioning method do you use



If yes, which data representation



Other response: Native

Do you use an I/O package like HDF5?

If yes, do you use, HDF5, NetCDF, or other

- 11 of 16 responses use HDF5
- 7 of 16 responses use NetCDF
- No response specify other

Which tools, if any, do you use to optimize or debug the MPI portion of your application?

- Allinea DDT
- Totalview, print statements
- Lots of print statements to debug!
- write(*,*) for debugging :)
- TAU, own tools
- Intel Vtune, Allinea Forge
- ITAC
- ddt
- Valgrind
- gdb
- Casper for optimizing asynchronous progress of MPI RMA, DDT for debug.
- Sometimes VTune, Intel Trace Analyzer and Collector
- Compare results with serial code.
- GDB for debugging. Recently started performing instruction count analysis using Intel SDE
- Tau
- internal timers + MPI_P*

Portability issues wrt different MPI implementations

In survey responses of 32 code groups, 10 gave the following comments on portability

- Have noticed that MPI implementations seem to differ in how they implement read/write_at_all() and read/write_at(). On NERSC's machines, I need to use read/write_at_all(), while on NCAR's machine, the same code prefers read/write_at().
- Open MPI: thread multiple and RMA usually broken, blue gene has no MPI-3
- Compilation problem with the Cray compiler
- Cray mpi sometimes crashes with alltoallv at large process counts
- We write MPI code following the MPI standard, so generally no portability issues exist. But sometimes we get unexpected RMA bugs in CrayMPI releases (in both regular mode and DMAPP mode).
- Not directly. Co-workers experienced problems with one-sided MPI (Intel MPI/MPICH)
- Old versions of MPI did not have MPI_INTEGER8 data type. Newer versions do.
- MPI_Comm_split deadlocking, MPI_Bcast deadlocking, MPI_Comm_dup using linear memory on each call, incorrect attribute caching destructor semantics.
- MPI_IO external32 vs. native encodings with OpenMPI & MPICH
- Too many issues over 15+ years to try to list them here.

Comments/concerns/suggestions/requests wrt MPI

In survey responses 5 code groups gave additional information

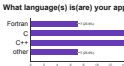
- don't break things!!!
- Lack of good open source MPI for Cray makes lots of things hard. So many problems would have been solved much quicker with source.
- It would be good to document automatic potential protocol switches in specific MPI implementations (e.g. Cray MPI) for small messages. That could explain some observed communication anomalies.
- Neighborhood collectives and nonblocking collectives are the most relevant optimizations. Implementation of "endpoints" would be huge for interoperability between packages that use OpenMP or a similar threading system and those that use MPI only (perhaps with shared memory windows).
- For progress of RMA operations, I use a process-based progress engine for MPI RMA, namely Casper.

Application codes used by MPI survey responders

- NIMROD
- NCAR LES
- GINGER FEL code
- WRF
- im3dshape, cosmosis (my own codes)
- WRF, LMWM, HYCOM, ESMF
- CSU global atmospheric model
- benchmarks
- NWChem
- Parallel Research Kernels
- ALPS/Core
- own code
- Applications that use PETSc
- GTFock, Elemental
- NWChem, Global Arrays
- Our own code, adapted for MPI
- NWChem, CTF
- epiSNP
- Ixam
- NIMROD
- MPI-3.1
- ISx
- FEniCS
- Ray, HipMer
- MILC QCD simulations codes
- PSI-Tet (finite element based MHD code)
- WARPI/PICSAR + own applications
- HipMER, Combinatorial BLAS, SuperLU
- idealized climate simulations using code named EULAG
- MUMPS, SuperLU_DIST, STRUMPACK, PETSc
- Lattice QCD, RHMC, Complex Langevin, Conjugate Gradient
- desispec and related DESI-specific custom code

Responses from Survey of MPI Alternatives

What language(s) is(are) your application written in?



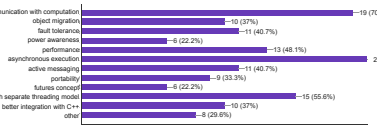
Other responses:
- UPC, python, bash UPC, UPC++
- python
- Java, Scala, Python, PERL, UPC
- Python, Javascript
- R, haskell, ocaml, python

Are you working on a proxy app for a major application?

If yes, what is not in the proxy app that is in full application? (Please name full application.)

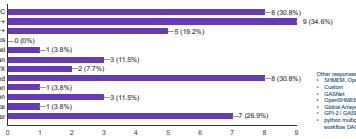
- computation is greatly simplified
- NEK5000 is the full app. At least, proper local matrix integration is missing.

What issue(s) caused you to consider approaches other than MPI?



Other responses:
- complex DAG-like workflow dependencies; familiarity and clarity of multiprocessing vs. MPI
- complex transformations on communication operations
- Binary code without access to expertise to modify source
- Much less syntax to accomplish the same thing using Coarray Fortran, resulting in more readable / maintainable code
- Simpler programming (productivity)
- Tended to implement non-MPI models
- Load balancing
- MPI is too low level.

Which parallel programming model(s)/approach(s) do you use?



Other responses:
- SHMEM, OpenMP
- Custom
- GASNet
- OpenSHMEM
- Global Arrays
- GPI-2 / GASPI
- python multiprocessing; custom workflow DAG management

Do you have versions in more than one different programming model?

If yes, which models?

- MPI
- MPI
- MPI, OpenMP
- MPI, OpenMP
- MPI and multiprocessing
- most of them
- MPI and Coarray Fortran
- benchmarks in multiple models
- MPI, GA, UPC++
- Charm++, HPX
- MPI+OpenMP, OCR, UPC, UPC++
- MPI and SHMEM
- MPI vs OCR

Have you compared your approach to MPI?

If yes, are there results available, e.g., paper, talk, link?

- http://gasnet.lbl.gov/performance/
- http://dx.doi.org/10.1007/978-3-319-41321-1_17, https://github.com/ParRes/Kernels
  https://dx.doi.org/10.1177/1094342016658110
- See http://upc.lbl.gov/publications/
- https://www.computer.org/csdl/proceedings/ipdps/2016/2140/00/2140a453-abs.html
  http://leeexplore.ieee.org/document/7161562/?reload=true&anumber=7161562
- http://dx.doi.org/10.1109/IPDPS.2013.78
- http://dx.doi.org/10.1109/IPDPSW.2014.83, http://www.gpi-site.com/gpi2/benchmarks/

If yes, was your imolementation in MPI optimized?

If yes, on which architecture(s) was it optimized?

- Cray
- Cray
- Cray XC (Edison/Cori)
- See individual papers for details
- many – Intel CPU cluster, Cray, etc.
- x86_64
- x86
- x86 / InfiniBand

How do you think MPI could be improved to better address your needs?

- Much simpler programming model
- Active Message support, better RMA support
- fault tolerance, implementation quality
- More documentation on design patterns. For example, it is really simple to implement Request/Reply and Pub/Sub patterns using ZeroMQ (http://zguide.zeromq.org/page:all) in C and much easier to understand than MPI.
- Probably not
- MPI 3 has resolved some of the problems, but we have a paper in preparation to highlight the remainder.
- Lighter weight (efficiency), simpler API, and remote function execution
- Fault Tolerance
- Task-based
- higher level API, simpler data sharing
- OpenSHMEM one-sided communication has an advantage over two-sided messaging for global communication patterns where MPI two-sided messaging overheads (matching, unexpected messages, etc) can generate high overheads. MPI RMA could be used in place of OpenSHMEM if the performance of MPI RMA were optimized to the level of SHMEM.
- Fault tolerance, fine-grained tasks
- MPI should be a compilation target. Application programmers should not see it at all.

Does the application use dependent libraries?

If yes, which libraries does it depend upon?

- numpy, scipy, astropy
- ZeroMQ and SIGAR (https://github.com/hyperic/sigar)
- CDFLIB
- Multiple
- BLAS, ScaLaPack
- HDF5

Does the application use Python?

Can your approach be combined with

Other responses:
- TBB
- Slurm
- C++, Chapel, Legion

Summary

- MPI is widely used by a large number of NERSC's 5000+ users with 32 code groups responding to our MPI Survey. We assume our advanced users responded to the survey, so the percentages do not necessarily represent a portion of the total user population, but rather those concerned with programming models
- We see that many of the advanced MPI features are being used and that a wide range of of decomposition and communication types are represented
- The majority are using MPI only or MPI + OpenMP with some users using MPI + pthreads, CUDA, OpenCL, TBB, UPC, UPC++, GASNet, SHMEM, or Global Arrays
- About 1/3 of the responders use one-sided communication (normal windows) with MPI_Win_create, MPI_Win_allocate, and MPI_Win_create_dynamic used equally
- About 1/4 use hints via the info object and 1/3 use MPI-IO with the majority using Explicit Offsets as the positioning method for IO
- Portability issues wrt different MPI implementations were cited by 10 code groups
- The three most cited issues that caused the 22 responders to the MPI Alternative Survey to consider approaches other than MPI were:
  - asynchronous execution (74%)
  - overlap of communication with computation (70 %)
  - uniform interface, not combined with separate threading model (56%)

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.