# Accelerating MPI Reductions on Intel Xeon Phi

Nathan T. Weeks
Department of Computer Science
Department of Mathematics
Iowa State University
Ames, IA, USA
weeks@iastate.edu

Meiyue Shao
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
myshao@lbl.gov

Brandon Cook
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
bgcook@lbl.gov

Marcus Wagner
Cray, Inc.
St. Paul, MN, USA
marcus@cray.com

Glenn R. Luecke
Department of Mathematics
Iowa State University
Ames, IA, USA
grl@iastate.edu

Pieter Maris
James P. Vary
Department of Physics
Iowa State University
Ames, IA, USA
{pmaris,jvary}@iastate.edu

## ABSTRACT

MPI global reduction operations are widely-used in MPI applications, and significant efforts have been made to minimize their communication cost. However, their computational cost can be substantial for large arrays on many-core architectures such as the Intel Xeon Phi "Knights Landing" (KNL) due to lower single-thread performance. We demonstrate that optimized user-defined reduction operations leveraging vectorization and OpenMP multi-threading can substantially impact MPI sum reduction performance on KNL—up to a 4X speedup vs. the predefined MPI_SUM operation in our microbenchmark. We also demonstrate how OpenMP taskloops can in principle facilitate load balancing in multi-threaded MPI reduction operations that overlap other multi-threaded computation.

## KEYWORDS

MPI, OpenMP, Intel Xeon Phi

**ACM Reference format:**
Nathan T. Weeks, Meiyue Shao, Brandon Cook, Marcus Wagner, Glenn R. Luecke, Pieter Maris, and James P. Vary. 2017. Accelerating MPI Reductions on Intel Xeon Phi. In *Proceedings of The 23rd European MPI Users' Group Meeting, Chicago, IL USA, September 2017 (EuroMPI'17),* 2 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Global reduction operations have been a fundamental component of MPI since its inception, and there has been much research on improving their communication performance[1]. However, since the inaugural MPI 1.0 standard, supercomputers have steadily evolved larger per-node memory capacities. As a result, MPI reductions can be performed on increasingly-larger arrays. Recent years have also seen the advent of many-core processors such as the Intel Xeon Phi "Knights Landing" (KNL). KNL presents opportunities for increased aggregate performance vs. current multi-core processors by exposing additional parallelism to applications (via both multiple processor cores/threads and vectorization) at the expense of lower single-threaded performance. As a result, MPI reduction operations may have different performance characteristics than on multi-core processors with faster single-threaded performance.

This work gauges the performance of MPI sum reductions on KNL, and characterizes the effect of compile-time optimizations and multi-threading.

## 2 METHODOLOGY

An MPI sum reduction microbenchmark[1] was run on the NERSC Cori supercomputer. This microbenchmark comprises three tests, each of which performs an MPI sum reduction on $2^{30}$ REALs (single-precision floating-point values) per MPI process using: (1) the predefined MPI_SUM operation implemented in cray-mpich/7.5.5, (2) an equivalent user-defined reduction (Listing 1) that is compiled using the same Intel Fortran compiler (version 17.0.3) and compiler options targeting the KNL (defined by the `craype-mic-knl` environment module) as the rest of the program, and (3) the aforementioned user-defined reduction with OpenMP enabled.

Each test was run using 2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024 nodes, each with 1 MPI process per node. The OpenMP version used 64 threads. The Cori KNL nodes, each equipped with a 68-core Intel Xeon Phi Processor 7250, were utilized in the *cache* memory mode and *quadrant* clustering mode, as this general-purpose configuration is the default on Cori, and avoids time-consuming node reboots. Each test was run 11 times at each MPI process count. The first ("warm-up") time for each test was discarded.

## 3 RESULTS

Benchmark results are illustrated in Figure 1. Replacing the predefined MPI_SUM operation with the user-defined equivalent listed in Listing 1 (without OpenMP enabled) resulted in a speedup of

[1]Source code and results available at https://doi.org/10.5281/zenodo.827377


*EuroMPI'17, September 2017, Chicago, IL USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

**Listing 1: User-defined MPI sum reduction. If load balancing with other OpenMP tasks is not required, an OpenMP parallel do could be substituted.**

```
subroutine openmp_sum(invec,inoutvec,len,datatype)
   integer, intent(in)    :: len, datatype
   real,    intent(in)    :: invec(len)
   real,    intent(inout) :: inoutvec(len)
   integer                :: i
!$omp taskloop simd shared(invec, inoutvec, len)
   do i = 1, len
      inoutvec(i) = inoutvec(i) + invec(i)
   end do
end subroutine openmp_sum
```

**Listing 2: Conceptual application of an OpenMP taskloop within a user-defined MPI reduction, allowing load balancing with concurrent tasks generated by a taskloop construct outside of the user-defined reduction. Tasks generated by the reduction taskloop are prioritized.**

```
subroutine openmp_sum(invec,inoutvec,len,datatype)
   integer, intent(in)    :: len, datatype
   real,    intent(in)    :: invec(len)
   real,    intent(inout) :: inoutvec(len)
   integer                :: i
!$omp taskloop simd default(shared) priority(1)
   do i = 1, len
      inoutvec(i) = inoutvec(i) + invec(i)
   end do
end subroutine openmp_sum
...
program main
...
call MPI_Op_create(user_sum,.true.,op,ierror)
...
!$omp parallel
!$omp master
!$omp task priority(1)
call MPI_Reduce(sendbuf,recvbuf,count,MPI_REAL,op,&
                comm,root,ierr)
!$omp end task

!$omp taskloop priority(0)
do ...
   ... other concurrent tasks overlapping reduction
end do

! ensure MPI_Reduce task done before reusing buffers
!$omp taskwait
...
```

approximately 1.6X to 1.8X. Enabling OpenMP at compile time resulted in a further speedup of 1.3X - 2.2X.

## 4 FUTURE WORK

A goal of this work is to implement performant multi-threaded MPI sum reductions that can be executed concurrently with other

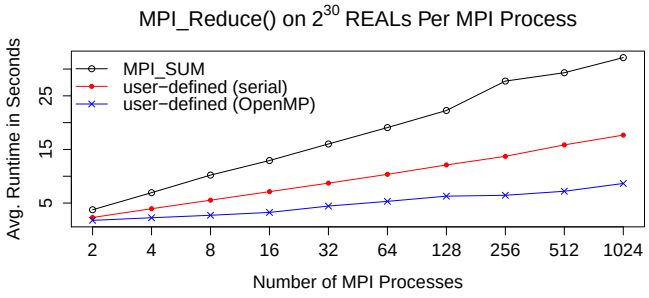


**Figure 1: Timings for predefined (MPI_SUM) and user-defined (serial and OpenMP) reductions on $2^{30}$ REALs (1 MPI process per KNL node; 64 threads for the OpenMP version). Each reported timing is the average of 10 calls to MPI_Reduce().**

OpenMP loops. One option is to utilize nested OpenMP parallelism, with separate teams of (fixed numbers of) threads executing OpenMP loops for the user-defined reduction and concurrent computation. Another option is to use OpenMP taskloops to allow loop iterations from both the user-defined MPI sum reduction and other computation to be executed by same team of threads. The latter option, exemplified in Listing 2, has the advantages of both implementation simplicity and the potential for better load balancing, as threads that are done executing tasks generated by one taskloop can be employed in the execution of other taskloops. Task priorities can be used to prioritize MPI sum reduction tasks over other computational tasks to hasten the completion of the reduction computation (and indirectly the associated communication).

While conceptually achievable, additional Intel OpenMP runtime optimizations appear to be needed to make the aforementioned concurrent-taskloop approach performant.

## 5 CONCLUSION

MPI implementations should continue to strive to reduce communication overhead in MPI global reduction operations. However, compile-time optimizations for the predefined reduction operations should not be overlooked—especially on architectures where vectorization and multi-threading are of utmost importance, such as KNL.

## ACKNOWLEDGMENTS

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.

This work was supported in part by the DOE under grant No. DESC0008485 (SciDAC-3/NUCLEI).

## REFERENCES

[1] Khalid Hasanov and Alexey Lastovetsky. 2017. Hierarchical redesign of classic MPI reduction algorithms. *The Journal of Supercomputing* 73, 2 (01 Feb 2017), 713–725. https://doi.org/10.1007/s11227-016-1779-7