MPI Performance Evaluation and Characterization using a Compact Application Benchmark Code

Patrick H. Worley
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, TN 37831-6367
worleyph@ornl.gov

Abstract

In this paper the parallel benchmark code PSTSWM is used to evaluate the performance of the vendor-supplied implementations of the MPI message-passing standard on the Intel Paragon, IBM SP2, and Cray Research T3D. This study is meant to complement the performance evaluation of individual MPI commands by providing information on the practical significance of MPI performance on the execution of a communication-intensive application code. In particular, three performance questions are addressed: how important is the communication protocol in determining performance when using MPI, how does MPI performance compare with that of the native communication library, and how efficient are the collective communication routines.

1. Introduction

The MPI message-passing interface standard [7] holds great promise in providing both portability and, through a very rich model for interprocess communication, performance efficiency for application, library, and compiler developers. The active participation of numerous vendors in the standardization process and the early appearance of both portable and platform-specific implementations of the library bode well for its success. In this paper, we evaluate the current status of the vendor-supplied implementations of MPI on the Intel Paragon, the IBM SP2, and the Cray Research T3D in terms of the following performance issues:

- sensitivity of performance to choice of communication protocol
- relative performance as compared with proprietary message-passing libraries
- relative performance of collective communication routines as compared with portable implementations

In short, we are interested in determining how best to use MPI, and what, if anything, is given up by its usage.

A typical approach to evaluating interprocessor communication and communication libraries is to measure the performance of individual commands in isolation or in small kernels representing common communication functions [2], [6]. While these types of experiments are a necessary step in an evaluation, the communication protocols and the controlled measurement environment used in the experiments may not be typical of how the commands are used in practice, and it can be difficult for an application developer to interpret the results.

This study is meant to complement the "low level" studies of MPI commands by providing a qualitative evaluation based on quantitative measurements of MPI performance in a specific application code. The study is qualitative in that the results reflect the peculiarities of the application code, but the overall conclusions as to MPI performance should be more generally applicable.

In this study we use the ParkBench [6] compact application code PSTSWM [10], [11]. PSTSWM is a parallel algorithm testbed and benchmark code that solves the nonlinear shallow water equations on a rotating sphere using the spectral transform method. PSTSWM was developed to evaluate strategies for parallelizing spectral global atmospheric circulation models [3], [4], and has imbedded a large number of parallel algorithm options. Among these options are numerous choices for the communication protocols used to implement the different parallel algorithms and numerous choices of message-passing layer, including MPI.

The version of PSTSWM used here differs from that currently used in the ParkBench benchmark suite. First, this new version uses FFT and BLAS math library routines when available on a given platform. This improves computational performance, thus increasing sensitivity to communication performance. The new version also includes the "double transpose" parallel FFT, a new parallel al-

0-8186-7533-0/96 $05.00 © 1996 IEEE

Ex.1128 / IPR2025-00914 / Page 1 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

gorithm that is used in the study. Finally, the new version includes calls to the MPI collective commands MPI_ALLTOALLV and MPI_ALLREDUCE as two of the parallel algorithm implementation options and includes the command MPI_SENDRECV as one of the communication protocol options.

2. Methodology

PSTSWM is a spectral timestepping code. During each timestep of the model simulation, the state variables of the problem are transformed between the physical domain, where most of the physical forces are calculated, and the spectral domain, where the terms of the differential equations are evaluated. The physical domain is a tensor product longitude-latitude-vertical grid, and transforming from physical coordinates to spectral coordinates involves performing a fast real Fourier transform (FFT) for each line of constant latitude, followed by integration over latitude using Gaussian quadrature, approximating the Legendre transform (LT). The inverse transform involves evaluating sums of spectral coefficients ("inverse LT") and inverse real FFTs.

The parallel algorithms in PSTSWM are based on decompositions of the physical and spectral computational domains over a logical two-dimensional processor mesh of size $\texttt{PX} \times \texttt{PY}$. For the FFT and LT, there are two general families of parallel algorithms: distributed algorithms, using a fixed data decomposition and computing results where they are assigned, and transpose algorithms, remapping the domains to allow the transforms to be calculated sequentially.

In this study we examine MPI performance for two different parallel algorithms: DP, a (D)istributed FFT with a (P)ipelined distributed LT, and TN, a (T)ranspose FFT with a (N)onpipelined distributed LT. Neither of these are necessarily optimal for the underlying problem or for the given platforms, although they are reasonable, nor are they necessarily appropriate for comparing performance between platforms. (In other studies we have investigated choosing appropriate algorithms for each platform and problem size to allow for fair intermachine benchmarking [3].) Instead, these parallel algorithm were chosen because they have significantly different message-passing characteristics and exercise different aspects of message-passing performance, and because the second parallel algorithm can be implemented using MPI collective communication routines.

Communication for the distributed FFT in algorithm DP consists of $(1 + \log_2 \texttt{PX})$ swaps of equal size messages in a butterfly pattern, with some overlap of communication and computation possible at the cost of doubling the number of messages. Communication for the distributed LT in algorithm DP consists of $(\texttt{PY} - 1)$ send/receive pairs of equal size messages to/from neighbors in a logical ring, and is arranged to allow significant overlap of communication and computation (without any additional cost).

The transpose FFT used in algorithm TN is a "double transpose" in which a transpose is performed both before and after the FFT, returning the data decomposition to a similar state. (The rationale behind the double transpose is that it minimizes the load imbalance.) Communication for each transpose consists of $(\texttt{PX} - 1)$ swaps or send/receive pairs (depending on the ordering of the communication) of equal size messages in an all-to-all pattern. Each transpose is functionally equivalent to MPI_ALLTOALLV.

The distributed LT used in algorithm TN completes all local computation before invoking a parallel vector sum routine. The vector sum is implemented using a variant of the algorithm proposed in [8]. Communication for the vector sum consists of $\log_2 \texttt{PY}$ swaps of decreasing size, each half the size of the previous swap, done twice for the forward transform but with no communication during the inverse. The vector sum is equivalent to MPI_ALLREDUCE using the predefined operator MPI_SUM.

With the exception of the tuning runs mentioned in the next section, all experiments involved five day simulations of the standard benchmarking problem specified in [9], representing the calculation of solid body rotation steady state flow, using 64 bit precision. Two different problem sizes were considered: T42, using a physical grid of size $64 \times 128 \times 16$, and T85, using a physical grid of size $128 \times 256 \times 16$. Experiments were run on logical processor meshes of size 4×4, 4×8, 8×8, and 8×16, as well as on meshes of size 16×16 and 16×32 on the Paragon.

The logical mesh topology is implemented explicitly in the application code, not using the MPI topology routines. With the exception of the collective communication experiments described in the next section, a row major $((i, j) \rightarrow i + j \cdot \texttt{PX})$ or column major $((i, j) \rightarrow j + i \cdot \texttt{PY})$ linear encoding was used to map the logical mesh to the physical processors, and the same topology and processor ordering were used for both MPI and native implementations. On the Paragon and T3D, the logical meshes were "mapped" onto physical processor meshes with the same aspect ratio, treating the second and third mesh dimensions on the T3D as a single dimension. No information was available on the layout of the physical processors used in the SP2 experiments, and the row major encoding was used in all experiments.

FFT and BLAS routines from the KAI and ESSL libraries were used on the Paragon and SP2, respectively. The corresponding routines are missing from the T3D math libraries, and T3D results reported here use PSTSWM-specific Fortran subroutines for these functions.

2.1. Sensitivity and Robustness

All interprocessor communications in PSTSWM are based on the SWAP or SENDRECV functions. These can

Ex.1128 / IPR2025-00914 / Page 2 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

be implemented using MPI_SENDRECV, or they can be implemented "by hand" using combinations of the many other MPI point-to-point commands. Moreover, the different stages of the hand-implemented SWAP and SENDRECV functions can be separated, allowing, for example, receive-ahead algorithms in which nonblocking receive requests for multiple SWAPs are posted together, or overlap algorithms in which computation is interleaved with the send and receive requests [11].

In order to evaluate the performance sensitivity of MPI to these types of optimizations, we first conducted thousands of short "tuning" experiments to identify a subset of good communication protocols. We then compared PSTSWM performance when using the optimal MPI communication protocol (determined over the subset) with performance when using the "default" MPI_SENDRECV protocol. Note that the tuning experiments were also a rigorous test of the robustness of the MPI point-to-point communication routines.

2.2. Comparison with Native Libraries

Vendor-supplied MPI is still relatively new on most platforms, and often there exist alternative proprietary ("native") communication libraries. In time, we expect MPI to be the high performance communication layer on most MPPs, but currently there may be a discrepancy in performance between the native libraries and MPI that can be exploited for applications with critical performance requirements.

In order to examine this possibility, we repeated the tuning exercises described above to identify optimal communication protocols for the native libraries and compared PSTSWM performance between the "optimal" MPI implementations and the "optimal" native implementations. We also examined whether the optimal MPI protocols are system-specific (reflecting the optimal protocols of the native library), MPI-specific (common across the different platforms), or implementation-specific (different for each MPI on each platform).

On the Paragon, the native library is NX. On the SP2, the native library is MPL. On the T3D, we implemented an application-specific message-passing layer on top of SHMEM, the low level library supporting direct read and write from/to the memory of other processors. The SHMEM implementation takes advantage of the fact that, in PSTSWM, tagged message passing is not needed. Selection by source is sufficient. This allows us to implement a very efficient SWAP and SENDRECV. This is a fair comparison in that, while the native library here is not a traditional message-passing library, using MPI for SWAP and SENDRECV may show reduced performance over an application-specific implementation. The new MPI-2 standard will include one-way reads and writes (like SHMEM), When this is introduced, one-way reads and writes can be incorporated into the MPI communication protocol sensitivity studies.

2.3. Collective Communication

The collective communication routines (should) represent highly optimized implementations comparable to math library functions. In consequence, we compared the performance of parallel algorithm TN implemented using MPI collective routines with that of the best (native or MPI) implementation using the equivalent PSTSWM Fortran routines. Timings using the MPI collective routines were made both with the "default" mapping and with the process numbering resulting from a call to MPI_CART_CREATE with the reorder option specified, allowing the system to pick a better mapping if possible. The timings for the Fortran implementations were all made using the default mapping.

We also present data from instrumented runs to identify how much time is spent in the transpose and reduction routines. While much care was taken to ensure the accuracy of these measurements, they are only approximate in that load imbalances show up as communication cost and contaminate the comparison. Fortunately, TN has the best load balance of all the parallel algorithm options.

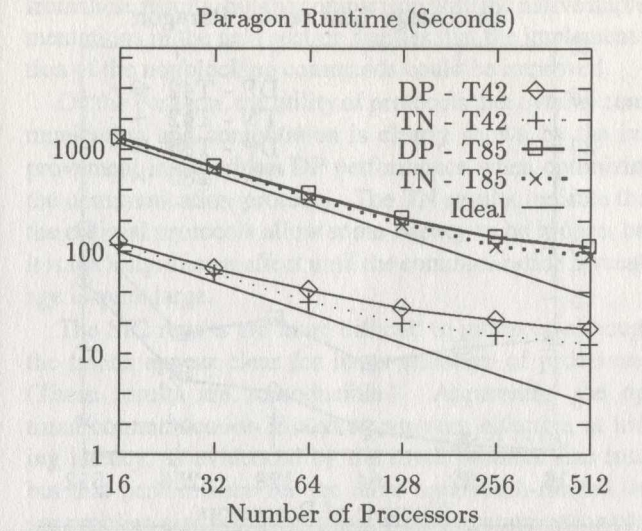
3. Results

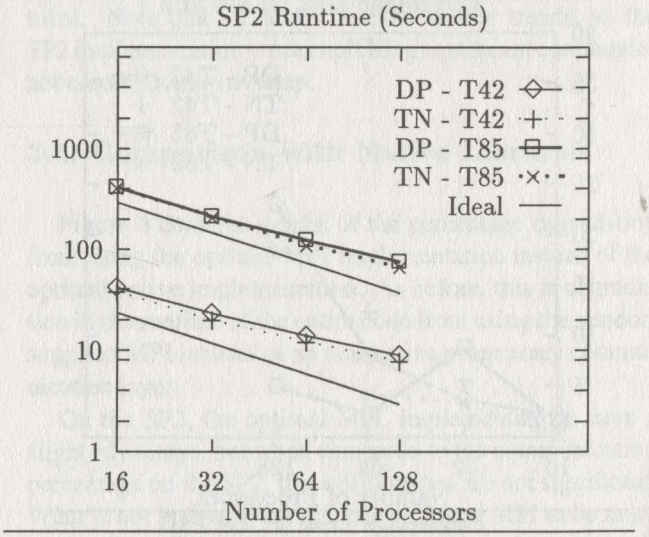
The following data was collected in March and April of 1996 on the 1024 processors Paragon XP/S 150 MP at Oak Ridge National Laboratory, on the 160 processor SP2 at NASA Ames Research Center, and on a 128 processor T3D at Cray Research in Eagen, MN. The Paragon was running OSF/1 Paragon Release 1.4, which includes a version of both MPI and NX communication libraries. The SP2 was running version 4.1.3.3 of the AIX operating system and version 2.1.0.8 of the parallel operating environment (poe) and parallel and switch software (pssp). The T3D was running version IS-9.0 of the operating system and version 1.4a of the CRI/EPCC MPI communication library. Note that the SP2 at NASA Ames is a homogeneous system in which timesharing of individual processors is not allowed, and thus is a good site for performance measurements.

3.1. Application Characterization

We begin by presenting a characterization of the communication cost in PSTSWM for all three platforms. Figure 1 contains graphs of the runtime of the fastest implementations as a function of the number of processors. The "Ideal" curves were generated by subtracting the measured communication time from the fastest 16 processor run to get an estimate of the computational time, then plotting this compu-

Ex.1128 / IPR2025-00914 / Page 3 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.





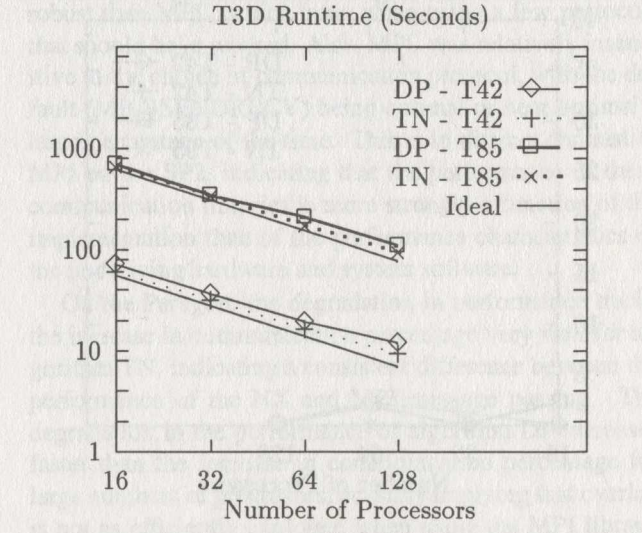
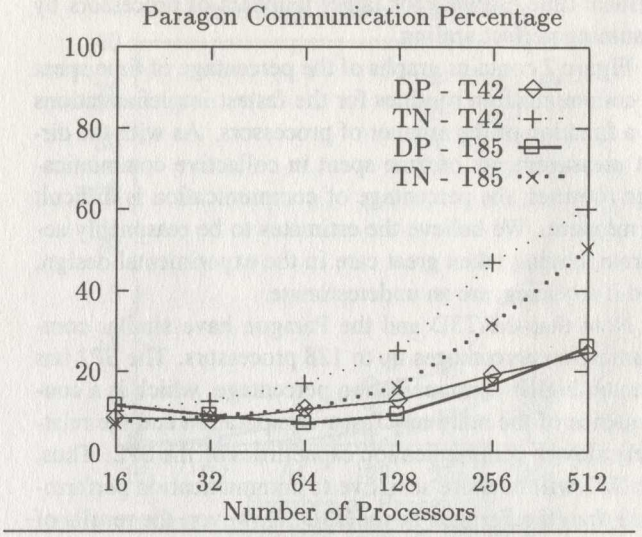
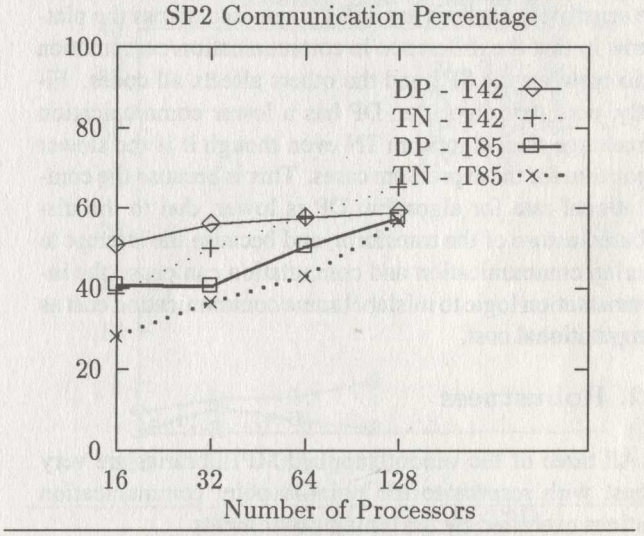


Figure 1. Runtimes for fastest implementations of PSTSWM.





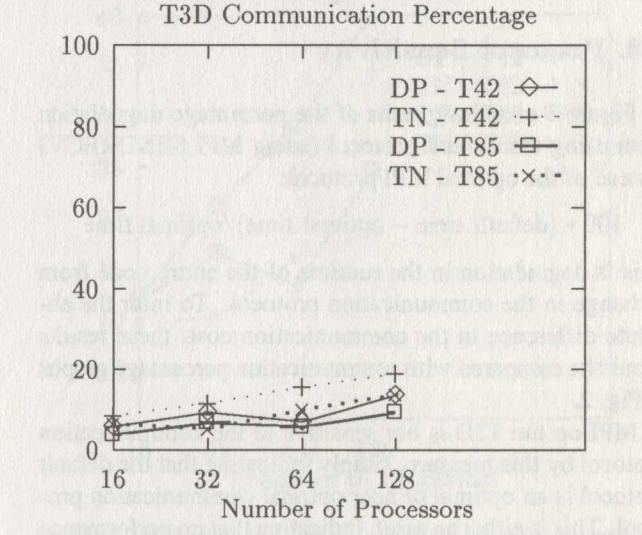


Figure 2. Communication percentages for fastest implementations

Ex.1128 / IPR2025-00914 / Page 4 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

tational time estimate for larger numbers of processors by assuming perfect scaling.

Figure 2 contains graphs of the percentage of time spent in communication routines for the fastest implementations as a function of the number of processors. As with the direct measurements of time spent in collective communication routines, the percentage of communication is difficult to measure. We believe the estimates to be reasonably accurate, having taken great care in the experimental design, and if anything, are an underestimate.

Note that the T3D and the Paragon have similar communication percentages up to 128 processors. The SP2 has a much higher communication percentage, which is a consequence of the relatively faster computation and the relatively slower communication capabilities of the SP2. Thus, the SP2 will be more sensitive to communication performance than the Paragon or the T3D. However, the results of the sensitivity analysis are still comparable across the platforms in that the difference in communication/computation ratio between the SP2 and the others affects all codes. Finally, note that algorithm DP has a lower communication percentage than algorithm TN even though it is the slower algorithm for these problem cases. This is because the computational rate for algorithm DP is lower, due to the distributed nature of the transform, and because the attempt to overlap communication and computation can cause the instrumentation logic to mislabel some communication cost as computational cost.

3.2. Robustness

All three of the vendor-supplied MPI libraries are very robust with respect to the point-to-point communication routines exercised by the tuning experiments.

3.3. Protocol Sensitivity

Figure 3 contains graphs of the percentage degradation from using the default protocol (using MPI_SENDRECV) instead of the optimal MPI protocol:

$$100 * (\text{default_time} - \text{optimal_time})/\text{optimal_time}$$

This is degradation in the runtime of the entire code from a change in the communication protocol. To infer the absolute difference in the communication cost, these results should be compared with communication percentage graphs in Fig. 2.

MPI on the T3D is not sensitive to the communication protocol by this measure, simply indicating that the default protocol is an optimal or near optimal communication protocol. This is either an asset, indicating that no performance is being lost by using the generic command, or an indication that the other MPI point-to-point commands are not implemented efficiently. It is not possible to determine which

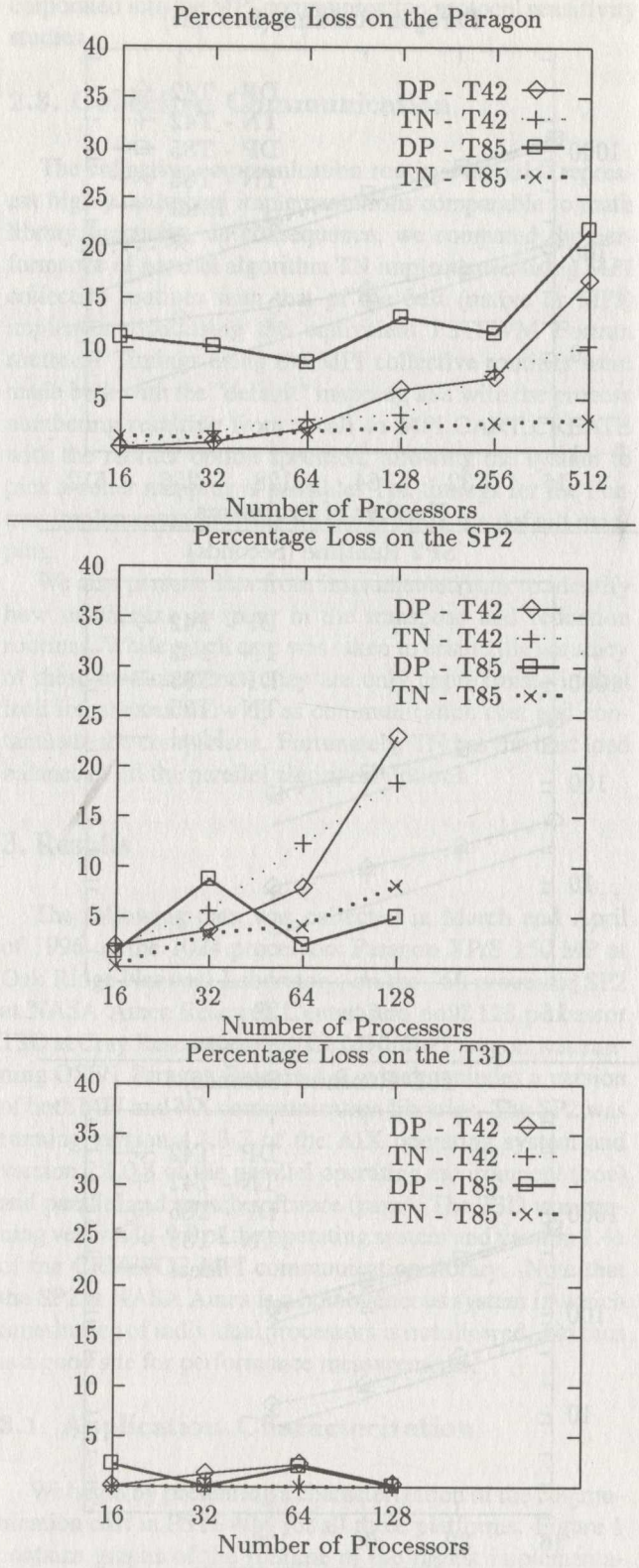


Figure 3. Performance Degradation from Using Default Protocol

Ex.1128 / IPR2025-00914 / Page 5 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

from these results, but the comparison with the native implementations in the next section implies that the implementation of the nonblocking commands could be improved.

On the Paragon, the utility of protocols that overlap communication and computation is clearly shown by the improvement in algorithm DP performance when optimizing the communication protocol. The TN results indicate that the optimal protocols allow some latency to be hidden, but it is not a significant effect until the communication percentage is quite large.

The SP2 results are more difficult to interpret, although the trends appear clear for larger numbers of processors. (These results are reproducible.) Apparently, the optimal communication protocols are very effective at hiding latency, as evidenced by the small problem size runs, but that performance for the more bandwidth-limited larger problem size is less sensitive to the communication protocol. Note that DP and TN show similar trends, so the SP2 implementation is not exploiting significant communication/computation overlap.

3.4. Comparison with Native Libraries

Figure 4 contains graphs of the percentage degradation from using the optimal MPI implementation instead of the optimal native implementation. As before, this is degradation in the runtime of the entire code from using the vendor-supplied MPI instead of an alternative proprietary communication layer.

On the SP2, the optimal MPL implementations have a slight advantage, but when compared to the communication percentage on the SP2, these differences are not significant. What is not indicated here is that we found MPI to be more robust than MPL, which hung when using a few protocols that should have worked. Also, MPL was relatively insensitive to the choice of communication protocol, with the default (MP_BSENDRECV) being optimal or near optimal a large percentage of the time. This is in distinct contrast to MPI on the SP2, indicating that the performance of these communication libraries is more strongly a function of the implementation than of the performance characteristics of the underlying hardware and system software.

On the Paragon, the degradation in performance tracks the increase in communication percentage very well for algorithm TN, indicating a consistent difference between the performance of the NX and MPI message passing. The degradation in the performance of algorithm DP increases faster than the increase in communication percentage for large numbers of processors, possibly implying that overlap is not as efficiently exploited when using the MPI library. Given the close match between NX and MPI semantics, it is surprising that the performance of the MPI and NX implementations is not closer, but the difference in the over-

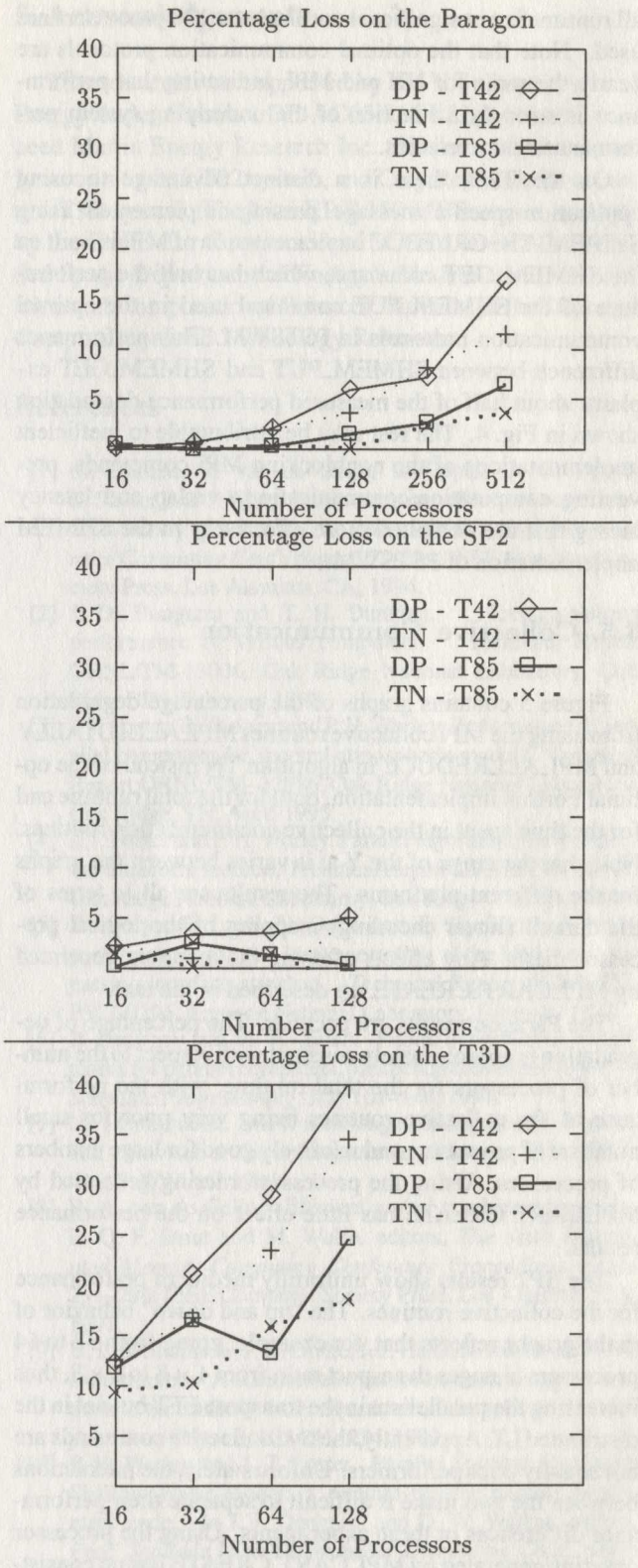


Figure 4. Performance Degradation from Using MPI Implementation

Ex.1128 / IPR2025-00914 / Page 6 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

all runtime is not significant until at least 128 processors are used. Note that the optimal communication protocols are nearly the same for NX and MPI, indicating that performance is strongly a function of the underlying system performance characteristics.

On the T3D, there is a distinct advantage to using application-specific message passing implemented using SHMEM. The CRI/EPCC implementation of MPI is built on the SHMEM_GET command, which has half the performance of the SHMEM_PUT command used in the optimal communication protocols in PSTSWM. This performance difference between SHMEM_PUT and SHMEM_GET explains about half of the measured performance degradation shown in Fig. 4. The rest may be attributable to inefficient implementations of the nonblocking MPI commands, preventing computation/communication overlap and latency hiding that are exploited more effectively in the SHMEM implementation of PSTSWM.

3.5. Collective Communication

Figure 5 contains graphs of the percentage degradation from using the MPI collective routines MPI_ALLTOTALLV and MPI_ALLREDUCE in algorithm TN instead of the optimal Fortran implementation, both for the total runtime and for the time spent in the collective communication routines. Note that the range of the Y-axis varies between the graphs for the different platforms. The results are all in terms of the default (linear encoding) mapping of the logical processor mesh. Any effects of using the mapping generated by MPI_CART_CREATE are described in the text.

The Paragon results are peculiar. The percentage of degradation is approximately constant with respect to the number of processors for the total runtime, with the performance of the collective routines being very poor for small numbers of processors and relatively good for large numbers of processors. Using the processor ordering generated by MPI_CART_CREATE has little effect on the performance results.

The SP2 results show uniformly mediocre performance for the collective routines. The "up and down" behavior of in the graphs reflects that, for example, going from 32 to 64 processors changes the aspect ratio from 4×8 to 8×8, thus increasing the parallelism in the transpose FFT but not in the distributed LT. Apparently, the two collective commands are not equally poor performers. Unfortunately, the interactions between the two make it difficult to separate their performance differences in these experiments. Using the processor ordering generated by MPI_CART_CREATE has no consistent or significant effect on the performance results.

On the T3D the MPI collective communication implementation of algorithm TN is competitive with the optimal MPI Fortran implementation for up to 64 processors,

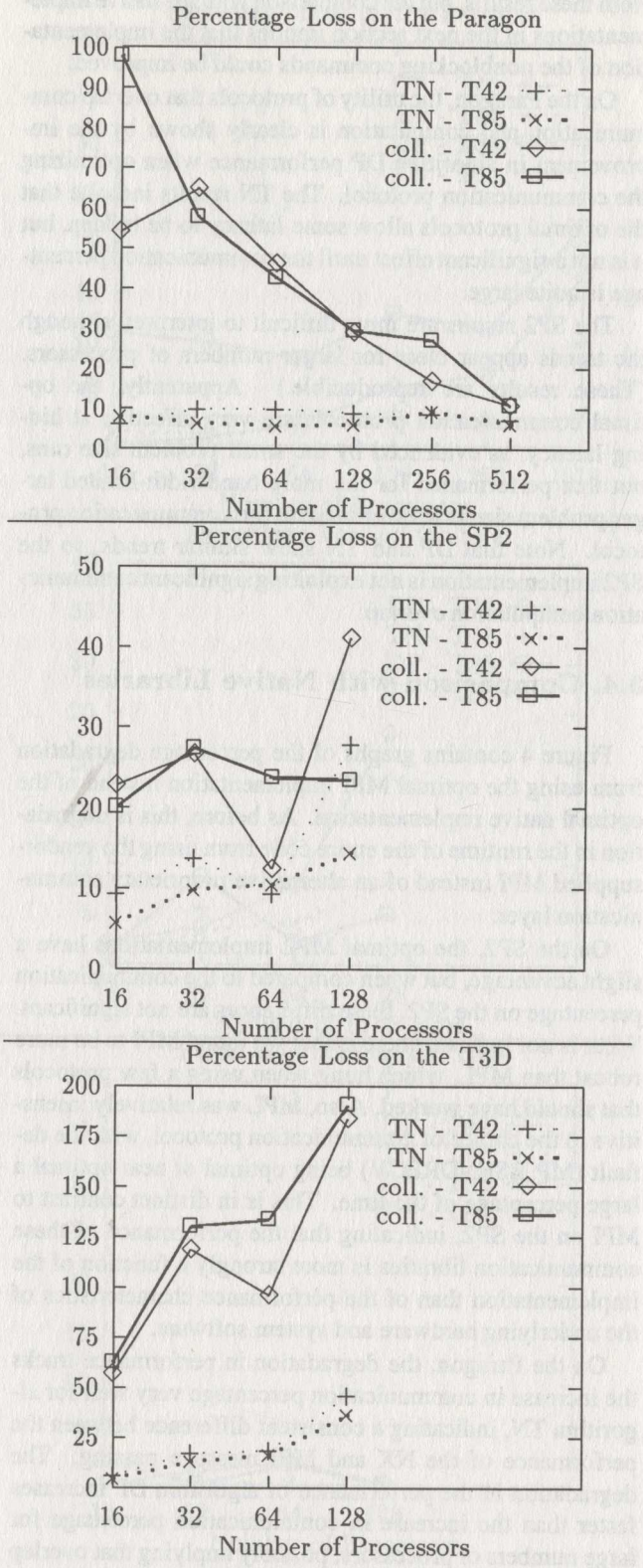


Figure 5. Performance Degradation from Using MPI Collective Communication Routines

Ex.1128 / IPR2025-00914 / Page 7 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

but is significantly worse for the 128 processor runs. In comparison with the optimal SHMEM Fortran implementation, the collective communication routines perform poorly for all numbers of processors. Looking at the profile data in more detail, it is clear that the problem is primarily in the MPI_ALLREDUCE command, which is three to five times slower than the SHMEM implementation, and scales poorly. Note that improvement is also possible in MPI_ALLTOTALLV, which is always slower than the SHMEM implementation, Using the processor ordering generated by MPI_CART_CREATE improves performance somewhat over using the default mapping, and the gain appears to increase slightly with the number of processors, but the qualitative performance behavior does not change.

4. Conclusions

As with any benchmarking exercise, performance will undoubtedly change with time, hopefully for the better. We expect the vendor-supplied implementations of MPI to continue to improve and hope that our results will be useful in accelerating this process.

In summary, we found all three vendor-supplied implementations of MPI to be robust, at least with respect to the standard point-to-point communication routines and to the two collective communication routines used in PSTSWM.

The three machines showed different sensitivities to communication protocol optimization and different optimal communication protocols. This result is unlikely to disappear in the future, and users may want to consider writing wrappers around the communication routines to make it simpler to "retune" codes when porting between platforms.

Overall, MPI performance was reasonable compared to the alternative proprietary message-passing layers, and performance should continue to improve over time. MPI on the T3D should probably be implemented on top of SHMEM_PUT for performance reasons. MPICH [5] is implemented in this fashion, but it does not work currently for this code on the T3D (due to a known problem in the implementation of MPICH) so it is unclear how difficult a SHMEM_PUT implementation will be.

The MPI collective communication routine performance was disappointing on all platforms. While we consider our codes to contain good implementations of these collective functions, we would hope that the vendor-supplied collective routines could be optimized far more than our "generic" Fortran routines. The results on the Paragon may simply indicate that the underlying iCC library [1] needs to be retuned for the ORNL Paragon.

5. Acknowledgements

This research was supported by the U.S. Department of Energy under Contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Inc. We thank NASA Ames for access to their SP2 system, and Cray Research for access to a T3D system. The Intel XP/S 150 MP Paragon operated by the Center for Computational Science at ORNL is funded by the Department of Energy's Mathematical, Information and Computational Sciences Division of the Office of Computational and Technology Research.

References

[1] M. Barnett, R. van de Geijn, S. Gupta, D. G. Payne, L. Shuler, and J. Watts. Interprocessor collective communication library (InterCom). In *Proc. Scalable High Performance Computing Conf.*, pages 357–364. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[2] J. D. Dongarra and T. H. Dunigan. Message-passing performance of various computers. Technical Report ORNL/TM-13006, Oak Ridge National Laboratory, Oak Ridge, TN, February 1996.

[3] I. T. Foster, B. Toonen, and P. H. Worley. Performance of parallel computers for spectral atmospheric models. Technical Report ORNL/TM–12986, Oak Ridge National Laboratory, Oak Ridge, TN, April 1995.

[4] I. T. Foster and P. H. Worley. Parallel algorithms for the spectral transform method. Technical Report ORNL/TM–12507, Oak Ridge National Laboratory, Oak Ridge, TN, May 1994.

[5] W. Gropp, E. Lusk, N. Doss, and T. Skjellum. A high-performance, portable implementation of the MPI message-passing interface standard. Technical Report ANL/MCS-P567-0296, Argonne National Laboratory, February 1996.

[6] R. Hockney and M. B. (Eds.). Public international benchmarks for parallel computers, parkbench committee report-1. *Scientific Programming*, 3(2):101–146, 1994.

[7] MPI Committee. MPI: a message-passing interface standard. *Internat. J. Supercomputer Applications*, 8(3/4):165–416, Fall/Winter 1994.

[8] R. A. van de Geijn. Efficient global combine operations. In Q. F. Stout and M. Wolfe, editors, *The Sixth Distributed Memory Computing Conference Proceedings*, pages 291–294. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[9] D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber. A standard test set for numerical approximations to the shallow water equations on the sphere. *J. Computational Physics*, 102:211–224, 1992.

[10] P. H. Worley and I. T. Foster. Parallel Spectral Transform Shallow Water Model: a runtime–tunable parallel benchmark code. In J. J. Dongarra and D. W. Walker, editors, *Proc. Scalable High Performance Computing Conf.*, pages 207–214. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[11] P. H. Worley and B. Toonen. A users' guide to PSTSWM. Technical Report ORNL/TM–12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.

Ex.1128 / IPR2025-00914 / Page 8 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

PROCEEDINGS

SECOND MPI DEVELOPER'S CONFERENCE

JULY 1-2, 1996
NOTRE DAME, INDIANA

IEEE COMPUTER SOCIETY
50 YEARS OF SERVICE • 1946-1996

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

Ex.1128 / IPR2025-00914 / Page 9 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

Proceedings

Second
MPI Developer's Conference

July 1 – 2, 1996

Notre Dame, Indiana

Sponsored by

IEEE Computer Society Technical Committee on Distributed Processing

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

Ex.1128 / IPR2025-00914 / Page 10 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

QA 76.642
.M67
1996

IEEE Computer Society Press
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

Copyright © 1996 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved.

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society Press, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Press Order Number PR07533
Library of Congress Number 96-76653
ISBN 0-8186-7533-0 (paper)
ISBN 0-8186-7535-7 (microfiche)

Additional copies may be ordered from:

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264
Tel: +1-714-821-8380
Fax: +1-714-821-4641
Email: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: +1-908-981-1393
Fax: +1-908-981-9667

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM
Tel: +32-2-770-2198
Fax: +32-2-770-8505

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
JAPAN
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553

Editorial production by Regina Spencer Sipple
Cover design by Joseph Daigle/Studio Productions
Printed in the United States of America by KNI, Inc.

The Institute of Electrical and Electronics Engineers, Inc.

Table of Contents

Second MPI Developer's Conference

Foreword viii
Committees ix

Implementations

MPICH on the T3D: A Case Study of High-Performance Message Passing 2
R. Brightwell and A. Skjellum

MPI on the I-WAY: A Wide-Area, Multimethod Implementation of the Message Passing Interface 10
I. Foster, J. Geisler, and S. Tuecke

Design and Implementation of MPI on Puma Portals 18
R. Brightwell and L. Shuler

MICE: A Prototype MPI Implementation in Converse Environment 26
M.A. Bhandarkar and L.V. Kalé

Implementing MPI under AP/Linux 32
D. Sitsky, P. Mackerras, A. Tridgell, and D. Walsh

Extensions

Generalized Communicators in the Message Passing Interface 42
I. Foster, C. Kesselman, and M. Snir

A Thread Taxonomy for MPI 50
A. Skjellum, B. Protopopov, and S. Hebert

Cooperative Data Sharing: A Layered Approach to an Architecture-Independent Message-Passing Interface 58
D.C. DiNucci

Language Issues

MPI as a Coordination Layer for Communicating HPF Tasks 68
I.T. Foster, D.R. Kohr Jr., R. Krishnaiyer, and A. Choudhary

Collective Communication and Communicators in mpi++ 79
D. Kafura and L. Huang

Object-Oriented MPI (OOMPI): A Class Library for the Message Passing Interface 87
B.C. McCandless, J.M. Squyres, and A. Lumsdaine

LINDA HALL LIBRARY
KANSAS CITY, MISSOURI

Ex.1128 / IPR2025-00914 / Page 12 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

Early Implementation of Para++ with MPI-2 95
O. Coulaud and E. Dillon

Applications

Parallel Molecular Dynamics Visualization using MPI with MPE Graphics 104
R.P. Kale, M.E. Fleharty, and P.M. Alsing

Message Passing in Complex Irregular Systems 111
C. Dionne, M. Nolette, and D. Gagné

Parallel Implementation of 3D FMA using MPI 119
E.J.-L. Lu and D.I. Okunbor

Parallel Solution of the Wave Equation Using Higher Order Finite Elements 125
M. Kern and S.M. Méfire

Coupling Fluids and Structures Codes on MPI 130
K. Wolf and E. Brakkee

Rendering of Numerical Flow Simulations Using MPI 138
J. Stone and M. Underwood

Cone Beam Tomography using MPI on Heterogeneous Workstation Clusters 142
D.A. Reimann, V. Chaudhary, M.J. Flynn, and I.K. Sethi

Lessons Learned from Attempts to Parallelize Linear FE Analysis Code for 2D Structures 149
R. Sahu

Debugging

Replay for Debugging MPI Parallel Programs 156
C.-E. Hong, B.-S. Lee, G.-W. On, and D.-H. Chi

Performance

Performance Evaluation of MPI Implementations and MPI-Based Parallel ELLPACK Solvers 162
S. Markus, S.B. Kim, K. Pantazopoulos, A.L. Ocken, E.N. Houstis, P. Wu, S. Weerawarana, and D. Maharry

MPI Performance Evaluation and Characterization using a Compact Application Benchmark Code 170
P.H. Worley

Posters

eMPI/eMPICH: Embedding MPI 180
T.P. McMahon and A. Skjellum

Ex.1128 / IPR2025-00914 / Page 13 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.

An Overview of a MPI Profiling Environment for the NEC Cenju-3 185
V. Govindan, Y. Park, X. Li,
S. Crear, and O. Johnson
First Class Communication in MPI 189
E. Demaine
An MPI Implementation of the BLACS 195
V. Deshpande, W. Sawyer, and D.W. Walker
MPICH Performance Characteristics and Considerations 199
R. Frost
MPI Performance on the SGI Power Challenge 203
T. Loos and R. Bramley

Author Index 207

Ex.1128 / IPR2025-00914 / Page 14 of 14
Intel Corporation v. Advanced Cluster Systems, Inc.