Software Architecture and Performance Comparison of MPI/Pro and MPICH

Rossen Dimitrov and Anthony Skjellum

MPI Software Technology, Inc.
101 S. Lafayette St, Suite 33,
Starkville, MS 39759 USA
{rossen,tony}@mpi-softtech.com
http://www.mpi-softtech.com

Abstract. This paper presents a comparison of two implementations of the MPI standard [1] for message passing: MPI/Pro, a commercial implementation of the MPI standard from MPI Software Technology, Inc., and MPICH, an open source, high-performance, portable MPI implementation. This paper reviews key distinguishing architectural features of the two MPI implementations and presents comparative performance results from micro benchmarks and real applications. A discussion on the impact of MPI library architecture on performance is also offered.

1 Background

MPICH was developed by Argonne National Laboratory and Mississippi State University and was the first publicly available MPI implementation [2]. MPICH has been used as a basis for a large number of open source and vendor MPI libraries. MPICH has demonstrated that a portable MPI implementation can be used to achieve high performance and scalability on a variety of parallel platforms. MPICH has played an important role in popularizing the MPI standard, which presently is the predominant model for parallel programming of multi-computers and clusters of workstations.

MPI/Pro is a high-performance, scalable implementation of the MPI 1.2 standard for clusters with Linux, Windows, and MacOS operating systems [3,5]. MPI/Pro supports communication over a variety of high-speed networks, such as Myrinet, VI Architecture, and InfiniBand, as well as over traditional TCP/IP transports, such as Fast and Gigabit Ethernet. Efficient intra-box (SMP) communication is also supported. MPI/Pro has a number of architectural features that facilitate high-performance and scalability while imposing controlled processor overhead. MPI/Pro has been commercially offered on a number of platforms and operating systems since 1997. The code base of MPI/Pro has been developed from first principles and has no legacy limitations.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 307–315, 2003.

2 Architectural Characteristics

MPI libraries implement a standard programming interface but there is a large number of architectural choices that affect the library performance and its behavior as a whole. This section presents some of the important architectural decisions in MPICH and MPI/Pro and their impact on application performance.

Message Completion Notification

Message completion notification is the mechanism that the MPI library uses to identify the completion of a communication activity and to notify the user about the completion of the requested operation. MPICH uses polling notification, which relies on continuously querying the operating system or polling a memory flag (if the underlying communication infrastructure supports user-level messaging) to identify when a communication operation is completed. Polling notification requires the involvement of the system CPU, and burns cycles, which could otherwise be used for useful computation. As opposed to this, MPI/Pro uses blocking notification, which is based on interrupts and kernel objects for synchronization. The user thread that expects message completion is put to sleep until the system is notified that the requested communication operation is completed. This reduces the use of CPU resources for communication activities, which is a major goal of any parallel system.

Blocking notification increases the message-passing latency of short messages. This increase is a result of the interrupt-based software mechanisms and kernel objects used for synchronization. The type of notification has a negligible impact on the bandwidth of medium size and long messages. In fact, experiments have shown that under similar conditions, the blocking mode can sustain higher peak bandwidth than the polling mode [4,5]. In order to address this issue, MPI/Pro implements a polling notification mode for some of its devices. Using a run-time flag, users can select the library notification mode – blocking or polling. Although experiments on real applications and computational benchmarks, such as the NAS Parallel Benchmarks, have not demonstrated any advantage of the polling mode versus the blocking mode [4], many micro-benchmarks often emphasize low latency of short messages. Having user selectable polling and blocking message completion notification is a unique feature of MPI/Pro and allows users to adjust the behavior of the library to their needs.

Message Progress

The MPI standard has defined a rule for message progress that guarantees that if the user has started a communication operation, the library should complete this operation regardless of the subsequent (call-to-the-MPI-library) behavior of the user process. This rule requires that the MPI library uses a mechanism for independent message progress. MPI/Pro uses such a scheme. Independent message progress guarantees timely and predictable delivery of messages, regardless of their size. As opposed to that, polling progress relies on a progress engine that is invoked only when the user process calls the MPI library. Thus, the message transfers, especially the ones of long messages, can be significantly affected by the behavior and execution timeline of the

user code. For example, if the code enters a long computation loop, this may directly affect the transmission of a pending long message, thus resulting in significant decrease of effective bandwidth. This effect cannot be observed and measured by micro-benchmarks such as the ping-pong test and should be studied by other means.

Low CPU Overhead

Reducing the CPU involvement in communication activities is a major goal of message-passing middleware. The internal architecture of MPI/Pro is designed so that it employs system services and mechanisms that are asynchronous in nature and thus CPU conscious. As demonstrated in the performance results section below, MPI/Pro can achieve better performance than MPICH at lower CPU utilization. Application programmers are thus able to employ various techniques for communication overhead hiding, such as overlapping of communication and computation.

Overlapping of Communication and Computation

Asynchronous completion notification, independent message progress, and the low CPU overhead for message passing allow MPI/Pro to facilitate efficient overlapping of communication and computation. Overlapping is an important programming technique for improving overall parallel application performance and can be applied with a high degree of success to a variety of algorithms and platforms. As opposed to MPI/Pro, MPICH uses polling notification and polling progress, which cause high CPU overhead for communication and does not allow for concurrent communication and computation activities. This leads to a minimal, if any, benefit of overlapping even though the application algorithm can be written in a way to take advantage of overlapping.

Multi-device Architecture

MPI/Pro has efficient multi-device architecture. Multiple devices enable the MPI library simultaneously to utilize different communication media offered by hierarchical memory/network systems. Such hierarchical systems include clusters built from networked multiprocessor nodes. Modern operating systems provide efficient interprocess communication (IPC) mechanisms between processes on one node. MPI/Pro utilizes these mechanisms through its SMP devices for Windows and Linux. The multi-device architecture of MPI/Pro allows all active devices to operate in an independent manner, which removes the performance dependency of faster devices on slower devices. MPI libraries with polling progress, such as MPICH, do not provide such isolation of their devices, which leads to a negative impact on performance and scalability [6].

Thread Safety and Thread Awareness

MPI/Pro is one of only a few MPI implementations that support multithreaded MPI applications. Thread-safety allows for a programming model with multi-level concurrency. Parallel applications can be designed so that they perform multi-threaded processing within a cluster node in order to exploit local parallelism while message-passing level parallelism is achieved through the MPI library using network communication between nodes. Thread-safety also enables MPI/Pro to operate in a hybrid-programming environment using a combination of MPI and OpenMP. OpenMP provides intra-node compiler-level parallelism. MPI/Pro not only enables multi-threaded user programs to use MPI, but also facilitates multi-threading by providing the highest-degree of thread safety as defined by the MPI-2 standard. Multiple threads can communicate efficiently as MPI/Pro ensures concurrent progress of communication activities initiated by all threads. The quality to facilitate multi-threaded programs is referred to as thread-awareness, which is viewed as a more desirable quality than basic thread-safety.

Performance Optimizations

MPI/Pro has a number of optimizations that lead to improved performance. Among these optimizations are an efficient derived data type engine, collective operations algorithms, and the message de-multiplexing scheme. In addition, MPI/Pro offers a large set of tunable parameters that can be used by the users to adjust the library performance to specific run-time environments and applications.

3 Performance Results

Performance results in this section are presented in two groups. The first group is micro-benchmarks for point-to-point bandwidth and execution time of a collective operation. The second group contains results from LINPACK used for the Supercomputer Top500 list and two real applications. All MPI/Pro and MPICH results are obtained on the same equipment, using the TCP/IP devices of the two libraries for most accurate comparison. The test platform for the tests of the first group is a cluster of eight Dell Dimension 4100 workstations with a single Intel Pentium III processor at 800 MHz with 128 MB RAM and Linux RedHat 7.2; with the 2.4.7-10 kernel. The network interconnect is 100 MB/sec Ethernet with Cisco Catalyst 2950 switch. MPICH version 1.2.4 and MPI/Pro versions 1.6.3 and 1.6.4 were used.

Point-to-Point Bandwidth

The bandwidth test is based on a ping-pong message-passing pattern that is used for measuring the round-trip time. The bandwidth for each message size is calculated as the message size is divided by the half of the round-trip time. The experiment shows that MPI/Pro's peak bandwidth is about 10% higher for message sizes greater than 256 kilobytes.

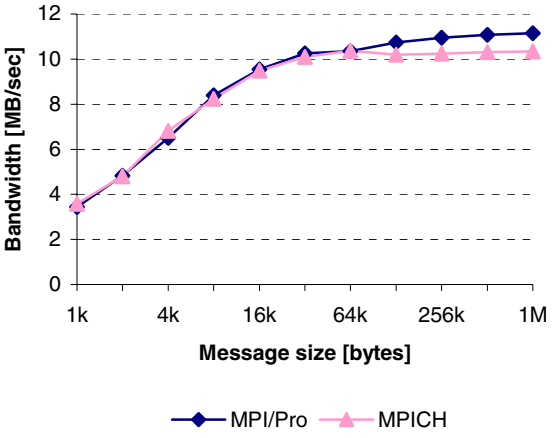

Fig. 1. Point-to-point bandwidth on a 100 Mbps FastEthernet network

The same ping-pong test was also performed on a pair of Dell PowerEdge 2650 servers with Intel Pentium 4 Xeon processors running at 2.4 GHz with 2GB RAM, interconnected with GigabitEthernet. This test was performed at Sandia National Laborary. It shows even greater advantage of MPI/Pro over MPICH, as it can be observed by the figure below – more than 40% difference in peak bandwidth in favor of MPI/Pro.

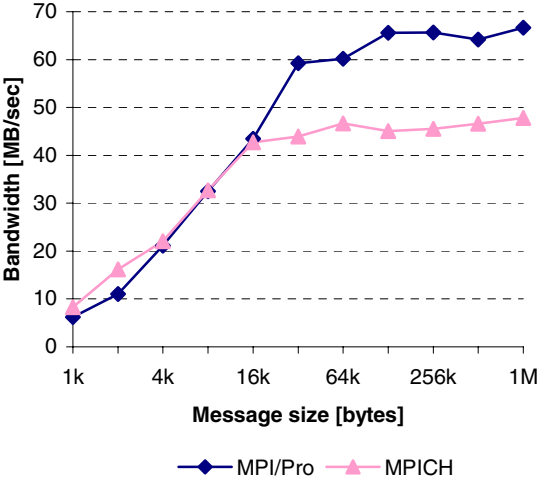

Fig. 2. Point-to-point bandwidth on a GogabitEthernet network

Collective Operations: Alltoall

This test measures the execution time of the MPI_Alltoall operation of the tested MPI library on eight processes, each executed on a single machine of the test cluster. Be-

cause of the great difference in the timing results, the graph represents the relative performance of MPI/Pro in comparison to MPICH, calculated as a percentage. MPI/Pro's performance is given as a baseline at 100%. The message size represents the message that each process sends to the other processes as part of the Alltoall transformation. The total size of the message passed to the MPI_Alltoall operations is CommSize**x**MsgSize. A message of size one Megabyte represented in the graph corresponds to a size of the message buffer passed to MPI_Alltoall operation of eight Megabytes.

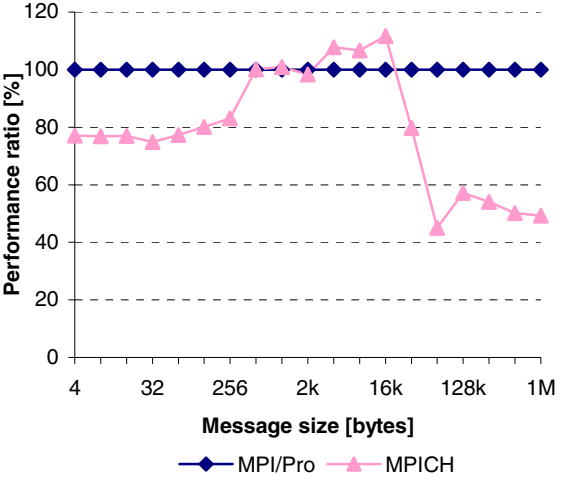

Fig. 3. Relative performance of MPI_Alltoall

During the Alltoall test, the CPU and memory utilization were observed. The results for message size of 1 Megabyte show that the average CPU utilization of MPICH was 24% while the CPU utilization of MPI/Pro was 9%. This demonstrates that MPI/Pro achieves a significantly better performance at a much lower CPU utilization, which leaves more CPU cycles for useful computation. The memory utilization shows that MPICH used more than 27 MB of memory, while MPI/Pro used less than 20 MB. The actual test memory requirements, without the MPI library, are about 17 MB. So, this shows that MPI/Pro uses only about 3 MB of system memory for internal purposes, whereas MPICH uses 10 MB, over 3 times more than MPI/Pro.

LINPACK

The results from the LINPACK benchmark were obtained in the NSF Engineering and Research Center at Mississippi State University. The test environment is as follows: 32 x IBM x330, 2 x Pentium III processors @ 1.0 GHz, 1.25 GB RAM (64 processors); Linux RedHat 7.2 with 2.4.9-13smp custom-built kernel; Cisco Catalyst 3548 FastEthernet switch. LINPACK was built with HPL 1.0 and BLAS was built with ATLAS 3.2.1, using gcc 2.95.3 compiler.

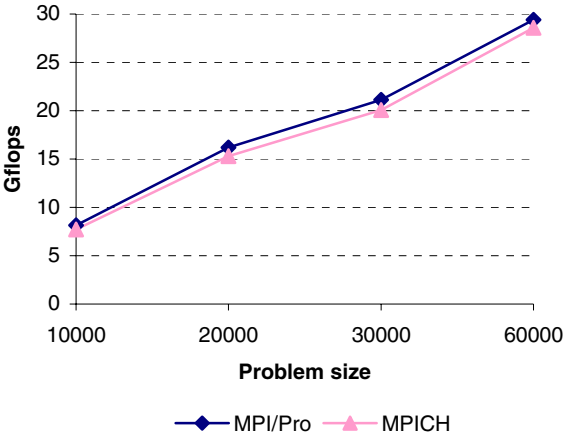

Fig. 4. LINPACK performance comparison

The LINPACK run with MPI/Pro shows between 3% and 6% higher performance than the run with MPICH.

Unstructured 3D Flow Simulation

The test results are obtained from an unstructured 3D flow solver of the incompressible Navier-Stokes equations using an implicit finite volume methodology. The code was executed at the NSF Engineering and Research Center at Mississippi State University for simulation of a Navy destroyer class combatant with and without propeller. The presented experimental results are the execution times for one iteration of the simulation. The simulation with propeller was run on 42 processors while the simulation without propeller was run on 32 processors. The test equipment used is the same as the one presented in the LINPACK benchmark above. The execution time of the simulation with propeller with MPICH was 222.76 seconds, while with MPI/Pro it was 168.93 seconds, which is an improvement of more than 31%. The simulation without propeller completed for 97.32 seconds when run with MPICH and 60.97 seconds with MPI/Pro, which is an improvement of almost 60%. So, MPI/Pro performed consistently better than MPICH on this code, showing a significant saving in time for the user.

NASA-Langley USM3D

This test was conducted at Lockheed Martin using a 64-way cluster with dual Intel Pentium III processors @ 850 MHz and 768 MB RAM, interconnected with a Foundry FastEthernet switch. The wall clock time obtained with MPICH was 3,213 seconds, while the wall clock time with MPI/Pro was 2,677 seconds, which is a performance improvement of more than 20%. Also, the user reported that occasionally random crashes were observed with MPICH while no such instabilities were seen with MPI/Pro.

4 Profiler Output

Executions of the LINPACK HPL benchmark with MPI/Pro and MPICH were compared using SeeWithin/Pro [7], a performance analysis tool for MPI applications. The table below shows a comparison of selected timings for both libraries for a test run with input problem size of 1000 on a 16-node cluster with 750 MHz Pentium III processors, 256 MB RAM, RedHat 7.3 and connected with Fast Ethernet. The profiling results reveal that the majority of MPI/Pro's communication functions incur less cumulative processing overhead than the same functions implemented in MPICH. The overall time of MPI/Pro spent on point-to-point communication is 25% less than the same time in MPICH. The significantly longer MPICH time spent in MPI_Wait could be attributed to the polling process engine of MPICH.

Table 1. Comparitive performance break-up revealed by SeeWithin/Pro analyzer

MPI Function	Total Time Spent (seconds)	
	MPI/Pro	MPICH
MPI_Irecv	65.39	10.20
MPI_Recv	1050.33	1516.77
MPI_Send	341.85	378.97
MPI_Wait	1027.09	1210.81
Total (Point To Point Communication)	*2484.66*	*3116.75*
MPI_Comm_free	0.00046	0.00066
MPI_Comm_split	1230.03	2069.03

5 Future Work

MPI Software Technology is currently developing a next generation MPI implementation under the product named ChaMPIon/Pro. ChaMPIon/Pro builds on the successes of MPI/Pro and specifically emphasizes ultra-scale parallel systems. Currently, ChaMPIon/Pro is being developed for the highly scalable DOE ASCI platforms. MPI-2 functionality is another major emphasis of ChaMPIon/Pro. At present, fully compliant support for parallel file I/O (MPI I/O) is available for NFS, GPFS, and PVSF. Also completed is one-sided communication. In development are dynamic process management and the remaining chapters of the MPI-2 standard.

6 Conclusions

MPI/Pro has a number of architectural solutions that distinguish it from MPICH and other open source public MPI implementations, among which are blocking completion notification, independent message progress, and efficient multi-device mode of operation and thread safety. These features facilitate high-performance programming mechanisms such as overlapping of communication and computation, early binding, multithreading, and exploitation of hybrid programming models such as MPI and OpenMP. MPI/Pro's architecture enables it to deliver high-performance and scalability to user applications and at the same time provide reliability and robustness. Also, MPI/Pro is well suited for environments where the timeliness and predictability of results are of critical importance.

References

1. Message Passing Interface Forum. 1994. MPI: A Message-Passing Interface Standard. *Int. J. of Supercomputer App.* 8 (3/4): 165–414.
2. Gropp, William, Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A High-performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing* 22 (6): 789–828.
3. Dimitrov, Rossen and Anthony Skjellum. 1999. An efficient MPI implementation for Virtual Interface Architecture – enabled cluster computing. In *Proceedings of the 3rd MPI developer's and user's conference*, Atlanta, Georgia, March 1999: 15–24.
4. Dimitrov, Rossen and Anthony Skjellum. 2000. Impact of latency on applications' performance. In *Proceedings of the 4th MPI developer's and user's conference,* Ithaca, New York, March 2000.
5. Dimitrov, Rossen. Overlapping of Communication and Computation and Early Binding: Fundamental Mechanisms for Improving Parallel Performance on Clusters of Workstations. *Ph.D. Dissertation*, Department of Computer Science, Mississippi State University, May 2001.
6. Protopopov, Boris and Anthony Skjellum. 2000. Shared-memory Communication Approaches for an MPI Message Passing Library. *Concurrency: Practice and Experience* 12(9): 799–820.
7. Krishna Kumar, C. R. et al. Automatic Parallel Performance Analysis and Tuning for Large Clusters. *High Performance Computing Conference,* Hyderabad, India, December 2001.