The Influence of Efficient Message Passing Mechanisms on High Performance Distributed Scientific Computing

Seyedeh Leili Mirtaheri
Computer Engineering Department
Iran University of Science & Technology
mirtaheri@comp.iust.ac.ir

Ehsan Mousavi Khaneghah
Computer Engineering Department
Iran University of Science & Technology
emousavi@comp.iust.ac.ir

Mohsen Sharifi
Computer Engineering Department
Iran University of Science & Technology
msharifi@iust.ac.ir

Mohammad Abdollahi Azgomi
Computer Engineering Department
Iran University of Science & Technology
azgomi@iust.ac.ir

Abstract

Parallel programming and distributed programming are two solutions for scientific applications to provide high performance and fast response time in parallel systems and distributed systems. Parallel and distributed systems must provide inter process communication (IPC) mechanisms like message passing mechanism as underlying platforms to enable communication between local and especially geographically dispersed and physically distributed processes. Communication overhead is the major problem in these systems and there are a lot of efforts to develop more efficient message passing mechanisms or to improve the network communication speed. This paper provides hard evidence that an efficient implementation of message passing mechanism on multi-computers reduces the execution time of a molecular dynamics code. A well-known program for macromolecular dynamics and mechanics called CHARMm is executed on a networked cluster. The performance of CHARMm is measured with two distributed implementations of message passing, namely a kernel-level implementation called DIPC2006 and a renowned library level implementation called MPI. It is shown that the performance of CHARMm on a DIPC2006 configured cluster is by far better than its performance on an optimized MPI configured similar cluster. Even ignoring the favorable points of kernel-level implementations, like safety, privilege, reliability, and primitiveness, the insight is twofold. Scientists are nowadays faced with more computational complexity and look for more efficient systems and mechanisms. Efficient distributed IPC mechanisms have direct effect on running scientists' simulations faster, and computer engineers may try harder to develop more efficient distributed implementations of IPC.

Keywords: High Performance Scientific Computing, Distributed Inter Process Communication, Distributed Systems, Operating System Kernel, Molecular Dynamics.

1. Introduction

Distributed computing systems aim at connecting processes to each other to obtain more computing resources in a distributed environment using a distributed inter process communication (IPC) mechanism. Primary distributed inter process communications were performed by network message passing that deal with routing mechanisms and machine addresses. But networks are unreliable and writing large scale distributed applications using network message passing protocols proved difficult and complex [1].

To ease distributed programming, especially for scientific application developers who are uninterested in the intricacies of networks, inter process communication mechanisms had been provided at higher levels as program libraries, as middleware and as operating system kernel level primitives. These mechanisms present some primitives to programmers for sending and receiving messages, while low level operations and complexities are handled by the mechanisms in a way transparent to programmers and users.

978-0-7695-3471-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ISPA.2008.131

IEEE computer society

Generally speaking, higher level implementations of IPC are more flexible but less efficient, while lower level ones are in contrast less flexible but more efficient. It is thus up to the programmers and users to choose an appropriate level depending on whether high performance or flexibility is paramount. In this paper we try to back up this belief that high performance is more critical to scientific applications that in turn require the most efficient IPC implementations, i.e. kernel level implementations.

Programs need to be parallelized either manually by programmers or automatically by computer software, using distributed inter process communications like MPI [2], PVM [3] and DIPC 2006 [4], in order to be able to run with higher performance on clusters or supercomputers. The parallelization is however quite difficult and requires a good understanding of the application logic in order to reduce the communication time of running the application in a distributed style on a cluster.

As an example, a non-hydrostatic version of the macromolecular dynamics and mechanics, called CHARMm [5], is selected as a scientific application with intensive computation and large data size. In addition, CHARMm has a noticeable amount of computation and communication which makes it a good candidate for testing communication and computation primitives in inter process communication mechanisms that are used in scientific clusters.

Two inter process communication mechanisms, namely, DIPC2006 and LAM/MPI [16]. DIPC2006 is implemented at the operating system kernel level, while MPI is implemented as program library. The CHARMm program was manually parallelized in two ways, once by using DIPC2006 primitives for distributed execution, and once by using LAM/MPI program library routines. The DIPC2006 version was run on a DIPC2006 configured cluster, and the MPI version was run on an optimized MPI configured cluster. Both clusters had the same hardware in type and numbers. As it will be reported in the next sections in this paper, the measured performances of program executions provides hard evidence in favor of kernel level implementation of distributed inter process communication mechanisms.

The rest of paper is organized as follows. Section 2 presents general approaches to reducing communication overhead in distributed systems. Section 3 gives a brief comparative study between DIPC2006 and MPI. Section 4 presents the parallel implementations of CHARMm on DIPC configured and MPI configured clusters, and quantitatively compares their performances. Given the experimental results and analyses and further discusses the reasons behind the superiority of DIPC2006 over MPI. Section 5 concludes the paper.

2. Related Work

There have been two general approaches to reducing communication overhead in distributed systems. Some have tried to optimize network operations and communication mechanisms at the lower layers of networks. Others have presented more and more efficient distributed inter process communication mechanisms at higher levels of networks as middleware, libraries, kernel level primitives.

T. Matthey et als. [8] have evaluated the possibilities of one-sided communication, a new feature of the MPI-2 standard, on the Origin2000 for relatively short-range molecular dynamics (MD) simulations. They concluded that the use of MPI's one-sided communication mechanism on the Origin2000 is not only feasible but it improves performance as well.

K. J. Bowers et als. [9] have presented several new algorithms and implementation techniques that significantly accelerate parallel MD simulations compared to current state-of-the-art codes. These include a novel parallel decomposition method and message-passing technique that reduces communication requirements, as well as novel communication primitives that further reduce communication time.

There are a lot of researches too that have tried to improve distributed inter process communication mechanisms at higher levels than the network layer. One of the popular distributed inter process communication mechanisms is MPI. MPI is a standard for inter process communication that has been implemented by many vendors, and groups of researchers [5, 10, 13, 14, and 15] who have tried to improve its performance by reducing its communication overhead during distributed execution of large size scientific application programs or popular benchmarks.

For example, M. Jiayin et als. [12] have improved the performance of MPI using a “multithreaded model to implement MPI point-to-point operations in order to overlap communication and computation. They presented theoretical and experimental results and compared the performance of both multithreaded and single threaded implementations.

P. Werstein et als. [11] have compared the performance of three kinds of distributed IPC mechanisms, namely TreadMarks DSM, MPI and PVM, in three kinds of parallel applications namely,

merge, sort and Mandelbrot set generator. They used a back propagation neural network and tried to show the behavior of the mechanisms in these parallel applications. They conclude that IPC performance is greatly influenced by application properties. That is to say that IPC mechanisms demonstrate different performances in applications that are loosely synchronous, embarrassingly parallel, or synchronous, and also when the number of nodes increase.

We believe that the resulting improvements in the performance directly relate to the efficiency of implementation of distributed inter process communication mechanisms. For higher level implementations and optimizations of distributed inter process communication mechanisms, namely those at application levels as binaries or operating system interface levels, performance improvements were lower than when these mechanisms were implemented at the operating system kernel level. In other words, the communication overhead in none kernel level implementations are still high for scientists. This paper backs this belief in a practical comparative implementation setting using a kernel level implementation of inter process communication, namely DIPC2006.

3. A Comparative Study

Some notable intricacies and characteristics of DIPC2006 and MPI that we have experienced in investigating and running CHARMm on DIPC2006 configured and MPI configured clusters are reported in this section.

To run programs under LAM/MPI, MPI demons must be explicitly booted in all machines in the cluster. But since DIPC2006 is implemented in the kernel of operating system, it is booted automatically when the operating system is booted.

DIPC2006 only adds distribution support to three SVIPC mechanisms, namely, semaphore, shared memory and message passing, and it does not change or modify system calls to these mechanisms. Programmers use these system calls as usual and do not require learning new functions to write share memory or message based distributed programs. MPI, on the other hand, presents new message passing primitives to programmers and programmers must learn them to write message based distributed programs. Furthermore, DIPC2006 programs can be debugged locally as any other SVIPC based program, but run distributed, while MPI programs can be debugged only in a distributed debugging environment.

DIPC2006 is limited by the size of message queue and message size in the operating system kernel, while MPI has no limit on message size as it opens sockets to transfer messages using the underling network.

DIPC2006 message features, like message size and the number of messages, are dependent on SVIPC message features. If SVIPC parameters change, DIPC2006 parameters also change. MPI parameters, on the other hand, can be changed or defined by programmers in their programs, or are set to default values by MPI pre compilers.

MPI compilers must be up and running when MPI programs are executing, but DIPC2006 programs are once compiled with ANSI compilers like *gcc* and can run independently without requiring the presence of any compilers at run time. .

DIPC2006 has been implemented on Linux 2.2.x kernels, and it can run on Intel i386, Motorola 680x0, ALPHA, PowerPC, SPARC and MIPS processors. MPI has different implementations too, like LAM/MPI and MPICH, each of them supporting different hardware architectures. Different implementations of MPI cannot interoperate and communicate with each other; i.e. each implementation can communicate only with its own kind of implementation. For example, LAM/MPI will work between just about any flavors of POSIX (with a few restrictions). That is to say on two completely different machines (e.g., a Sun machine and an Intel-based machine), LAM will run on both of them. More importantly, one can run a single parallel job that spans both of them, while DIPC2006 allows interoperability on different hardware.

Although LAM/MPI transparently converts data as required when data is transferred from one machine to a different machine, it does not support data types of different sizes. For example, if an integer is 64 bits on one machine and 32 bits on another, the LAM/MPI behavior is undefined. It also requires that floating point formats are the same on all machines [16].

4. Performance Analysis of CHARMm

Based on our experiences, some of the important ones being noted in the previous section, we selected a popular research tool for computational biology in molecular dynamics, called CHARMm, to run on DIPC2006 and LAM/MPI clusters. CHARMm (Chemistry at HARvard Macromolecular Mechanics) is a program for simulating biologically relevant macromolecules (proteins, DNA, RNA) and complexes thereof [5]. It allows investigating the structure and dynamics of large molecules (solute) in

the condensed phase (solventor crystal) [5, 6]. CHARMm has been modified to allow computationally intensive simulations to be run on multi-machines using a replicated data model [5]. This version, though employing a full communication scheme, uses an efficient divide-and-conquer algorithm for global sums and broadcasts.

We built the physics of a Linux based cluster by using 16 dual AMD CPUs and a Giga-bit network. The CPU type was Athlon 2500+ and we used a 1 GB/s switch and 1 GB of memory. In fact we constructed two clusters with the same hardware specification, one based on DIPC 2006 and the other based on MPI. We modified CHARMm to use DIPC2006 resulting in what we call CHARMm-DIPC. Similarly, we modified CHARMm to use MPI resulting in what we call CHARMm-MPI. We then ran CHARMm-DIPC and CHARMm-MPI on their cluster and measured the execution times using different number of processors.

As it is shown in Figure 1, the execution time of CHARMm-DIPC 2006 is less than that of CHARMm-MPI.

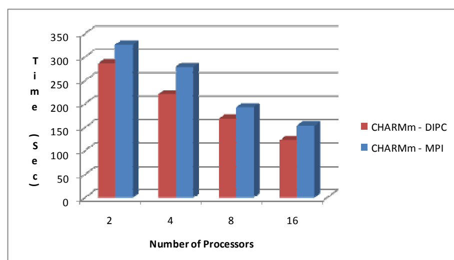

Figure 1. Execution times of CHARMm-DIPC and CHARMm-MPI

We also measured and evaluated the message passing communication overheads of running the parallel version of CHARMm on DIPC2006 and LAM/MPI configured clusters, conducting various executions with different number of processors, and calculated the execution times of send and receive messages in in both clusters. But we generally know that execution time is equal to communication time plus computation time, and any possible differences in execution times of send and receive messages in CHARMm-DIPC2006 and CHARMm-MPI are only attributable to communication time; the computation times can be fairly considered equal in both clusters.

We can consider the communication time (C) of a send/receive message in both clusters as follows:

*C = (time spent for sending/receiving an empty message) + n * (time spent for transmission of a unit of data worldwide)*

where n is the number of data to be transmitted over the network. The communication time for q messages is therefore qC.

Since we have only replaced the MPI calls (MPI_send or MPI_recv) in the CHARMm-MPI program with equivalent calls (Msg_send or Msg_recv) in DIPC2006 to get the CHARMm-DIPC2006 program, we may safely assume that the number of messages and the amount of data in each message had been the same when these programs were run on their respective clusters. This is to say that the difference in the execution times of CHARMm-MPI program and CHARMm-DIPC2006 program were not due to the number of transferred messages or the amount of data transferred, and therefore the second part of the above formula is the same for both cases. This is to say that the execution times of CHARMm-MPI and CHARMm-DIPC2006 were only affected by the first element of C in the above formula, i.e. only by the *time spent for sending/receiving an empty message.* Therefore we only measured the message send time in CHARMm-MPI and CHARMm-DIPC 2006 that are shown in Table 1.

Table 1: Send times for CHARMm-DIPC 2006 and CHARMm-MPI

Number of Processors	CHARMm-DIPC Send Time	CHARMm-MPI Send Time	Percentage of Difference
2	85.698	117.504	32%
4	66.06	100.224	34%
8	50.4	89.12	38%
16	36.72	77.296	40%

As it is shown in Table 1, increases in the number of processors increases the send time in both CHARMm-DIPC2006 and CHARMm-MPI because the amount of communication increases. The send time in CHARMm-MPI is comparatively higher than that in the CHARMm-DIPC2006 and the amount of difference improvement percentage has increased with increases in the number of processors.

Table 2 shows the receive times in both CHARMm-DIPC2006 and CHARMm-MPI receive times in MPI based program is more than in DIPC2006 based ones. The MPI overhead has caused these results.

The time spent for sending/receiving an empty message is in turn affected by the IPC mechanisms used as well as the underlying network properties. Given that the network had been chosen exactly identical in our experimentation, the differences in the execution times of CHARMm-MPI and CHARMm-DIPC2006 are only attributed to the implementation of IPC mechanisms used in these two

programs. Given that experimental results showed lower execution times for CHARMm-DIPC2006 relative to execution times of CHARMm-MPI, we rightly concluded that the implementation of IPC is more efficient in DIPC2006 than in MPI. This also describes why the execution time of CHARMm-MPI got comparably worse than CHARMm-DIPC2006 when the number of processors, and consequently the number of messages, were increased.

Table 2: Receive times for CHARMm-DIPC 2006 and CHARMm-MPI

Number of Processors	CHARMm-DIPC Receive Time	CHARMm-MPI Receive Time	Percentage of Difference
2	99.981	153.408	53%
4	77.07	131.848	54%
8	54.8	110.24	55%
16	31.84	89.192	57%

We can strengthen our argument by considering that MPI is implemented at the application level. All queues that are needed for sending and receiving of messages reside in the user space (Figure 2). Upon each communication request, a lot of data should be copied from user space to kernel space and vice versa. These in and out copying increase the communication overhead and thus execution time of programs.

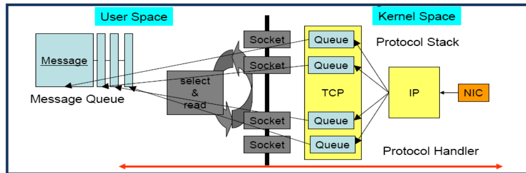

Figure 2. MPI communication architecture [13]

In contrast, DIPC2006 is implemented at the kernel level and all queues reside in the kernel. There is no need to copy between user space and kernel space and bypass socket APIs (Figure 3). That is why DIPC2006 programs have lower communication overhead and execution time.

A call to an MPI communication function involves some preparatory operations like searching and matching that are done by a process at the application level in the user space with limited privileges. The process then calls suitable socket APIs for transmission. These APIs are converted to related socket system calls in the kernel. The call to kernel switches the context from the user level to the kernel level and the kernel decides which process to run next that it may well not be the process just context switched. This overhead is in addition to the conversion and preparation overheads. On the contrary, in DIPC2006, all these operations are in fact calls to the kernel that are run in the kernel space with high privilege. These system calls use socket system calls locally. The process doing the call immediately continues to perform the call without any context switch. Overall, these are other reasons for MPI communication mechanisms to entail more overhead than those in DIPC2006.

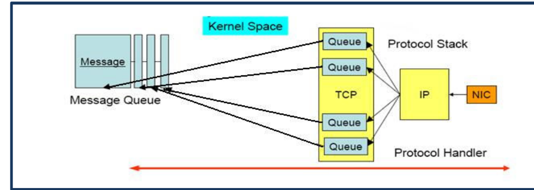

Figure 3. DIPC2006 communication architecture

Given all the noted differences of MPI and DIPC2006 in communication overheads, it becomes clear why communication overhead and consequently execution time showed up higher in case of CHARMm-MPI, compared to CHARMm-DIPC2006.

The general recommendation to distributed application and system developers is thus to program applications in a way to reduce execution time by lowering the communication time and spending most of the time on application level computation. This can be achieved by using as few communication mechanisms as possible, and at the same time by using more efficiently implemented communication mechanisms like DIPC2006.

Two points are noticeable from the study of experimental results of running CHARMm. First, the performance got better as the number of processors increased. This is however not always the case and it depends on many factors including the parallelism degree of application, which in this case is the parallelism degree of CHARMm. It is quite probable that the performance of CHARMm gets worse after the number of processors reaches a certain number, which in our case is higher than 16. This is primarily because when the number of processors gets higher than the parallelism degree of the running application, the communication costs of running processes cannot be compensated by computation speed up. The second point that is somehow implicated by the first point is that the performance of CHARMm-DIPC2006 is better than the performance of CHARMm-MPI. This is due to less communication overheads in the CHARMm-DIPC2006 than in the CHARMm-MPI. In other words, it is because the communication mechanism used in CHARMm-DIPC2006 is more efficient than its counterpart used in CHARMm-MPI.

5. Conclusion

This paper took a close look at the communication overhead of two different levels of distributed IPC mechanism at kernel and library levels, namely DIPC2006 and MPI. Many parallel and distributed scientific codes use a layered software system to facilitate the programming of inter process communication in order to get better portability and performance. This paper argued in favor of kernel level implementations of communication mechanisms, in contrast to library level implementations, in the scope of distributed systems. It selected the parallel molecular dynamics program CHARMm as an exemplar. It showed that this exemplar scientific application was much better off to run distributed on a high performance cluster that is configured with DIPC2006, which is a kernel level distributed implementation of standard IPC, than on an MPI based cluster. Although the paper only studied the execution times on these two clusters and argued that one is more efficient than the other for distributed execution of our selected scientific application, it also demonstrated that the increased cost of communication and the loss of performance is strongly correlated to the amount of latency and overhead of IPC implementation layers. Such scientific applications can well benefit from security, reliability, openness, and availability of kernel level communication mechanisms too.

6. References

[1] A. Tanenbaum, *Distributed Operating Systems*, Prentice Hall, 2005.

[2] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", Supported in Part by ARPA and NSF under Grant ASC-9310330, the National Science Foundation Science and Technology Center Cooperative, 2003.

[3] A. Geist, A. Beguelin, J, Dongarra, W, Jiang, R. Manchek and V, Sunderam "PVM: Parallel Virtual Machine", A User's Guide and Tutorial for Networked Parallel Computing, MIT Press. Available at: "http://www.netlib.org/", Last Accessed on January 2008.

[4] M. Sharifi, et als., "DIPC: A System Software Solution for Distributed Programming", International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 97, Georgia University, U.S.A., 1997.

[5] M. Taufer, E. Perathoner, A. Cavalli, A. Caflish and T. Stricker, "Performance Characterization of a Molecular Dynamics Code on PC Clusters, Is there any easy parallelism in CHARMm?", IPDPS 2002, IEEE/ACM International Parallel and Distributed Processing Symposium, Florida, USA, 2002.

[6] E. M. Boczko and C. L. Brooks III, First-Principles Calculation of the Folding Free Energy of a Three-He. *Science, 269:393*–396, 1995.

[7] M. Iwasaki, H. Chiba, N. Utsunomiya, K. Sonoda, S. Yoshizawa and M. Yamauchi, "Method for Inter Processor Communication", US Patent 5867656, February 1999.

[8] T. Matthey1 and J. P. Hansen, "Evaluation of MPI's One-Sided Communication Mechanism for Short-Range Molecular Dynamics on the Origin2000", PARA 2000, LNCS 1947, pp. 356-365, 2001.

[9] K.J. Bowers, E. Chow, H. Xu, R.O. Dror, M.P. Eastwood, B.A. Gregersen, J.L. Klepeis, I. Kolossvary, M.A. Moraes, F.D. Sacerdoti, J.K. Salmon, Y. Shan and D.E. Shaw, "Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters", SC2006, Tampa, Florida, USA, 2006.

[10] P. Husbands, J. C. Hoe, "MPIStarT: Delivering Network Performance to Numerical Applications", SC98. IEEE/ACM Conference, USA, 1998.

[11] P. Werstein, M. Pethick and Z. Huang, "A Performance Comparison of DSM, PVM, and MPI", The 4th International Conference on Parallel and Distributed Computing, Applications and Technologies, China, 2003.

[12] M. Jiayin, S. Bo, W. Yongwei, and Y. Guangwen, "Overlapping Communication and Computation in MPI by Multithreading", PDPTA 2006: 52-57, Las Vegas, USA, 2006.

[13] M. Matsuda, T. Kudoh, H, Tazuka and Y. Ishikawa "The Design and Implementation of an Asynchronous Communication Mechanism for the MPI Communication Model", International Journal of IPSJ Transactions on Computer Vision and Applications, Vol.45, PP: 14-23, 2004.

[14] William Gropp. "MPICH2: A New Start for MPI Implementations", In Dieter Kranzlm¨uller, Peter Kacsuk, Jack Dongarra, and Jens Volkert, editors, Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, Number LNCS 2474, 2002.

[15] M. Sharifi, L. Mirtaheri and E. Mousavi Khanegah, "A Case for Kernel Level Implementation of Inter Process Communication Mechanisms", IEEE Third International Conference on Information & Communication Technologies: from Theory to Applications (ICTTA2008), Syria, Damascus, 2008.

[16] LAM/MPI Team, "LAM/MPI User's Guide Version 7.1.2", Open Systems Lab, March 2006, Available at: http://www.lam-mpi.org/, Last Accessed on 2008.