The Effect of the 2^{nd} Generation Clusters: Changes in the Parallel Programming Paradigms

Jari Porras, Pentti Huttunen, and Jouni Ikonen

Lappeenranta University of Technology, Lappeenranta, FIN-53100, Finland
{Jari.Porras,Pentti.Huttunen,Jouni.Ikonen}@lut.fi

Abstract. Programming paradigms for networks of symmetric multiprocessor (SMP) workstation (2^{nd} generation of clusters) are discussed and a new paradigm is introduced. The SMP cluster environments are explored in regard to their advantages and drawbacks with a special focus on memory architectures and communication. The new programming paradigm provides a solution to write efficient parallel applications for the 2^{nd} generation of clusters. The paradigm aims at improving the overlap of computation and communication and the locality of communication operations. The preliminary results with large message sizes indicate improvements in excess of 30% over traditional MPI implementations.

1 Introduction

The main goal of utilizing a parallel computing environment is, most likely, the need to increase the performance of time-critical applications. There are numerous factors that impact the overall performance of a parallel application such as the number of processors, the memory architecture, and communication between processors to name a few. In order to maximize the performance of a parallel system all factors have to be considered and an optimal balance must be found.

This paper discusses the latest trends in parallel computing environments. The study shows that a new environment has emerged. This new environment is a successor of a network of workstations where each workstation consists of one processor. The 2^{nd} generation of clusters is composed of workstations that have two or more processors. Furthermore, such systems generally pose a more heterogeneous execution environment due to various hardware and software components found in them. The new parallel environment requires changes in traditional programming paradigm or entirely new programming paradigms to efficiently utilize the available resources. This paper discusses requirements for programming paradigms for 2^{nd} generation clusters, and describes an implementation of a communication library (MPIT) that supports the special features of SMP clusters, such as multilevel memory architecture.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 10–17, 2004.

2 Developments in Parallel Environments

Parallel environments have developed significantly during the past few years as processor and network technologies have evolved. Presently, the environments can be divided into the following categories:

2.1 Symmetric Multiprocessors (SMP)

In SMP machines the processors are connected through an internal network in such a way that all processors have equal access to the local (global) memory. Communication between processors occurs through this memory. Therefore, the communication is expedient, but can suffer from congestion on the network. Current multiprocessor PCs represent this category of parallel systems.

2.2 Clusters or Networks of Workstations (NOW)

Clusters are environments where processors are physically separated into different workstations and the workstations are connected through a network. The popularity of these environments is often contributed to the significant improvement in the performance-to-cost ratio compared to proprietary parallel systems. However, these environments introduce additional complexity to programming and running of parallel applications. This is a result of the distributed memory architecture and the external communication network. Together, these two characteristics introduce latencies to memory access and to communication between processors. For the purpose of this paper, the networks of single processor workstations (such as Beowulf clusters) are denoted as 1^{st} generation of the clusters.

2.3 Clusters of SMP Machines

Due to the low cost of multiprocessor workstations, clusters of SMP workstations have seen a considerably increased in number [3]. The workstations are connected in a similar way than in the traditional cluster environments but each node is a symmetric multiprocessor workstation with two or more processors. Therefore, the communication can occur internally or externally depending on the location of source and destination processors. An SMP cluster is denoted as a 2^{nd} generation cluster.

3 Changes in the Parallel Programming Paradigms

Programming paradigms for parallel environments commonly consists of a communication library to allow processors to exchange messages. In fact, most popular programming paradigms, such as MPI [10] and PVM [5], are merely message passing libraries. The communication introduces overhead to parallel applications, and thus must be carefully implemented to minimize its contribution.

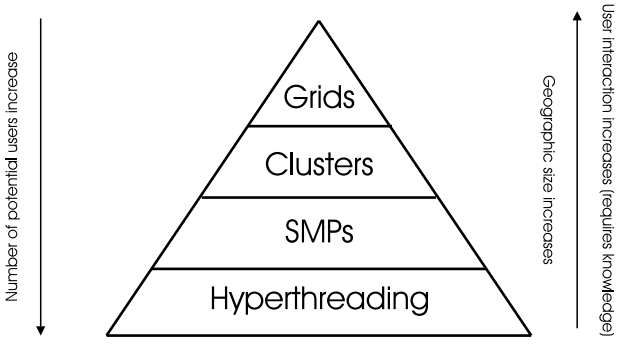

Fig. 1. The four levels of programmability in parallel systems.

There are various programming paradigms to write parallel applications based on the environment in which the applications are run. Fig. 1 depicts the paradigms and their main characteristics.

At the lowest level, parallelism occurs within a processor through the Hyperthreading technology [9]. Hyperthreading is the simplest approach from the programmer's point of view as the parallelization is automatically performed by the processors. The communication is handled implicitly through the shared memory (or even registers). The utilization of user-level threads in an SMP workstation represents the second level in Fig. 1. The programmer must be familiar with the thread-programming paradigm to efficiently implement a parallel application. Since the processors are still within a single workstation in an SMP environment, the communication between the processors occurs via the shared memory. In the 1^{st} generation of clusters the memory architecture is distributed. Therefore, the environment requires messages to be transferred from one workstation to another with a help of a communication library such as MPI [10] or PVM [5]. The 2^{nd} generation of clusters combine the SMP and traditional cluster environments by incorporating shared and distributed memory environments. Thus, a new paradigm is required in order to fully utilize the available resources and to obtain maximum performance. Finally, the top level in Fig. 1 illustrates an environment that integrates (SMP) clusters to each other to form grids. The grids add another level of message passing as communication between clusters is necessitated.

4 New 2-Level Communication Paradigm

The 2-level programming paradigm introduced in this section is not novel per se, since several studies have been conducted on 2-level communication paradigms:

1. Multithreaded and thread-safe MPI implementations have been proposed in [1], [2], [11], [12], [14], [15]
2. Hybrid models with MPI and OpenMP have been studied in [4], [6], [7], [8], [13]

The programming paradigm, MPIT, developed by the authors is a communication library built on top of MPI. The library provides functions to handle MPI operations in a thread-safe manner. MPIT considers both intra and inter workstation communication by using shared memory and message passing according to the resources available. The POSIX threads are used to run instances of the code on processors in a workstation. The intra workstation communication among threads is handled through the shared memory. The message passing is based on the MPI library and used only in inter workstation communication. The determination of whether shared memory or message passing is utilized is done automatically within the library.

An MPIT application operates in a single process on a workstation regardless of the number of processors. This process is created when the MPI environment is initialized through a 3rd party software (such as mpirun). The MPI environment must be initialized prior to calling the MPIT initialization function. The initialization of the MPIT library consists of the creation of the communication and the worker threads and local message buffers. The communication thread is responsible for handling all incoming and outgoing messages from a process (i.e. workstation), whereas worker threads execute the actual code.

In addition to the initialization and termination functions, the MPIT library provides a set of communication routines. The routines are very similar to the standard MPI communication operations. As a matter of fact, the only difference is that the MPIT routines have an additional argument, the target thread id. The target thread identifier allows for messages to be sent to a specific thread on a workstation rather than merely to the workstation. However, if the thread identifier is not defined, the message is sent to the workstation and processed by any of the worker threads on that workstation. Analogous to MPI, there are blocking and non-blocking send and receive operations available.

Also, MPIT provides various thread manipulation routines. These routines allow the programmer to control the number of worker threads on a workstation and the execution of the threads. With the help of the thread manipulation routines, the programmer is capable of adjusting the load imposed by the MPIT application on the workstation; more threads can be created, if the load on the workstation is low, whereas existing threads can be terminated to lower the impact of the MPIT application on the workstation's resources. The remaining thread manipulation routines are related to synchronization. There are barrier synchronization routines as well as routines to control the execution of a single thread.

Aspects like scheduling and load balancing can be addressed by this approach. The communication thread can be instructed to perform scheduling and load balancing operations to dynamically adjust loads on workstations or to retrieve new work for the workstation. This feature provides automatic scheduling functionality that occurs simultaneously with the execution of the worker threads. Therefore, the proposed programming paradigm supports not only the overlap of communication and computation but also the overlap of scheduling/load balancing and computation. The support for the scheduling/load balancing and

computation overlap is very important in heterogeneous and non-dedicated 2^{nd} generation clusters due to the necessity of frequent scheduling and load balancing operations.

The MPIT programming paradigm also provides a mechanism to assign priorities to the threads executed in a workstation. With this priority scheme a preference may be given either to the communication thread or the worker threads. Thus, it provides the means for the paradigm to adapt to the needs of the application and to the requirements set by the underlying network. For example, in an application where a large number of messages need to be exchange, the communication thread should be assigned with a higher priority than the worker threads. This allows for the communication operations to occur practically instantly minimizing the overhead introduced by the communication. A reversed priority scheme is applicable when there is no need for constant communication.

5 Performance

The MPIT library was tested in a cluster of 4 dual processor workstations. Table 1 shows the configuration of the workstations.

Table 1. The configuration of dual processor workstations used in the performance analysis

CPUs	Memory	Network	OS	MPI
2 x Pentium III 800MHz	1GB	Myrinet 2Gb/s	Red Hat 2.4.24	MPICH-GM 1.2.5..10

In order to estimate the performance of the MPIT library, a new benchmark was implemented. This benchmark makes it possible to evaluate the MPIT performance with various processor topologies, messaging schemes, and message types and sizes. The benchmark configuration for the preliminary results shown in this paper was as follows: the benchmark had 2 phases, computation and communication. During the computation phase each processor spent 1 second in a busy-wait loop, after which the process sent and received a message. The processor topology was a ring, in which each processor send a message to its right neighbor and received a message from its left neighbor.

The performance of MPI and MPIT were compared with 2 cluster configurations. The results shown in Fig. 2 illustrate three different cases of where one processors of all 4 workstations were used. In the dedicated system no other user processes were run, whereas in two other scenarioes 1 and 3 CPUs on different workstations were executing other user processes. As all the processes were distributed among the workstations all communication operations involved the Myrinet network.

The results clearly indicate that a simple MPI application is faster when the message size is below 16KB. However, the MPIT application provides up to 30%

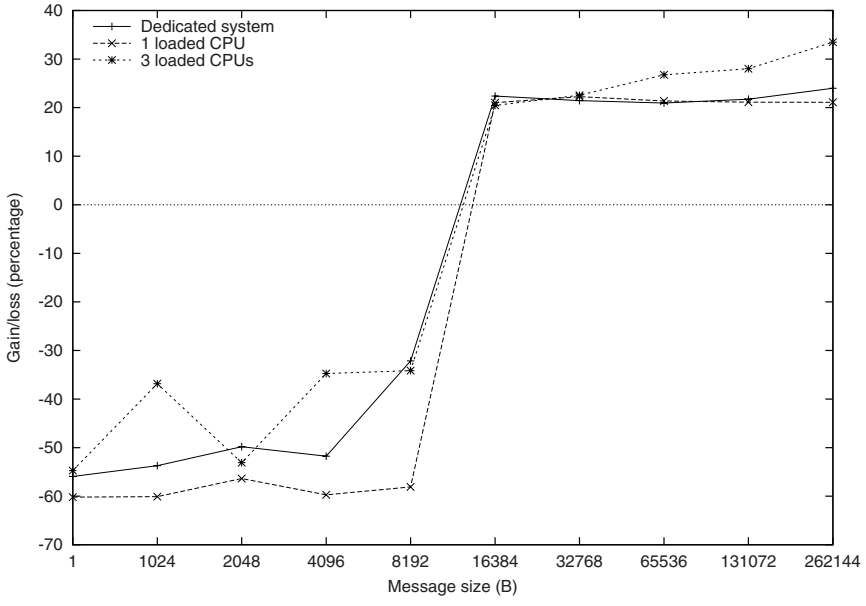

Fig. 2. Performance gain/loss of the MPIT library with 4 processes (1 process per workstation).

increase in the performance with message sizes greater than 16KB. The significant diffences with small message sizes is explained by the short message transfer times, whereas their impact becomes more apparent with the large message sizes. In addition, the current implementation of the MPIT library incurs unnecessary overhead when a message is sent synchronously to a remote process. This is due to a fact the communication thread is involved in the transmission even though it is not mandatory. This issue will be solved in the next release of the software. The performance improvement over the MPI application can be contributed to the overlap of computation and communication. The overlap allows processors to continue their execution immediately after initiating a communication operation. This also leads to a more synchronized execution of the processes which minimizes wait times during the communication phases.

The second test included running the benchmark on all available 8 processors on the test cluster. The test was run only in the dedicated mode where no other user processes were running. The results are shown in Fig. 3. The figure also entails the results from the dedicated 4 processor test for comparison purposes. Again, the MPI application has the advantage over the MPIT library with message sizes smaller than 16KB. With larger message sizes, the MPIT library has significant better performance than MPI. Furthermore, the MPIT performance has improved slightly compared to the 4 processor test.

These preliminary results indicate that MPIT has the potential to provide superior performance than MPI. However, at this point, MPI outperforms MPIT

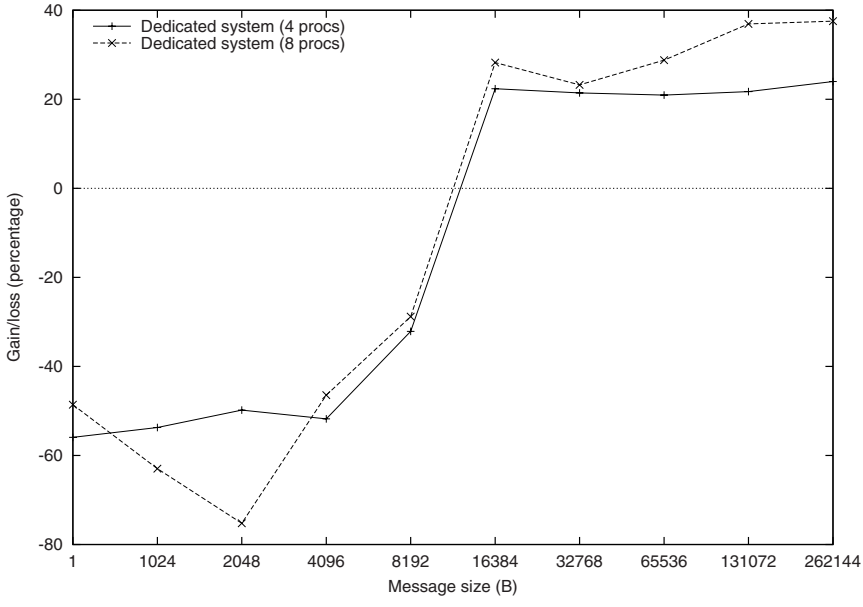

Fig. 3. Performance gain/loss of the MPIT library with 4 and 8 processes (1 process per workstation) in dedicated systems.

with small message sizes. As earlier discusses, the next release of the MPIT code will include an improved transmission mechanism for small messages, which should allow MPIT to excel regardless of the message size.

6 Conclusions

In this paper programming paradigms for cluster of SMP workstations were discussed. Also, a new programming paradigm, implemented as a thread-safe communication library on top of MPI, was introduced. The library, MPIT, optimizes the communication within and between the workstations of the cluster. It is important to realize that MPIT does not require any changes to the actual MPI implementation.

The performance results obtained show that an implementation like MPIT is required order to take full advantage of the 2^{nd} generation clusters. Although, the performance of MPIT was inferior to MPI with small message sizes, it outpermed MPI with large message sizes (16KB and up). The new release of the MPIT library will include a new design for the transmission of small messages. Overall, the preliminary results are very encouraging as nearly 40% improvements over MPI were observed on a cluster of dual processor workstations.

In addition to the new communication scheme, MPIT includes features to perform automatic scheduling and load balancing, and to prioritize execution of

communication and worker threads. In fact, as MPIT encompasses all these feature it could be extended for use in GRID environments where communication, load balancing, and scheduling are even further more complex issues than in 2^{nd} generation clusters.

References

1. Bader, D.A., Jájá, J.: SIMPLE: A Methodology for Programming High Performance Algorithms on Clusters of Symmetric Multiprocessors (SMPs). Journal of Parallel and Distributed Computing **58** (1999) 92–108
2. Chowdappa, A.K., Skjellum, A., Doss, N.E.: Thread-Safe Message Passing with P4 and MPI. Technical Report TR-CS-941025, Computer Science Department and NSF Engineering Research Center, Mississippi State University (1994)
3. Clusters@TOP500 list. http://clusters.top500.org
4. Djomehri, M.J., Jin, H.H.: Hybrid MPI+OpenMP Programming of a Overset CFD Solver and Performance Investigations. NAS Technical Report NAS-02-002 (2002)
5. Geist A. et. al.: PVM: Parallel Virtual Machine. MIT press, 1994.
6. He, Y., Ding, C.H.Q.: MPI and OpenMP Paradigms on Clusters of SMP Architectures: the Vacancy Tracking Algorithm for Multi-Dimensional Array.
7. Hu, Y., Lu, H., Cox, A. and Zwaenepoel, W.: OpenMP for Networks of SMPs. Journal of Parallel and Distributed Computing **60** (2000) 1512–1530
8. Kee, Y-S., Kim, J-S., Ha, S.: ParADE: An OpenMP Programming Environment for SMP Cluster Systems. Proceedings of Supercomputing 2003 (2003)
9. Leng, T., Ali, R., Hsieh, J., Mashayekhi, V., Rooholamini, R.: An Empirical Study of Hyper-Threading in High Performance Computing Clusters. Proceedings of LCI International Conference on Linux Clusters: The HPC Revolution 2002 (2002)
10. Pancheco, P.: Parallel Programming with MPI. Morgan Kaufmann (1997).
11. Protopopov, B. Skjellum A.: A multithreaded message passing interface (MPI) Architecture: Performance and Program Issues. Journal of Parallel and Distributed Computing **61** (2001) 449–466
12. Rauber, T., Runger, G., Trautmann, S.: A Distributed Hierarchical Programming Model for Heterogeneous Cluster of SMPs. Proceedings of the International Parallel and Distributed Processing Symposium (2003) 381–392
13. Smith, L., Bull, M.: Development of mixed mode MPI / OpenMP Applications. Scientific Programming **9** (2001) 83–98
14. Tang, H., Yang, T.: Optimizing Threaded MPI Execution on SMP Clusters. Proceedings of Supercomputing 2001 (2001) 381–392
15. Tang, H., Shen, K., Yang, T.: Compile/Run-time Support for Threaded MPI Execution on Multiprogrammed Shared Memory Machines. Proceedings of Programming Principles of Parallel Processing (1999) 107–118