Parallel Computing Environments and Methods

Ghassan Fadallah
ghassan@ele.etsmtl.ca

Michel Lavoie
mlavoie@ele.etsmtl.ca

Louis-A. Dessaint
dessaint@ele.etsmtl.ca

Groupe de Recherche en Électricque de Puissance et Commande Industrielle (GRÉPCI),
École de technologie supérieure, Department of electrical engineering,
1100 Notre Dame Ouest, Montréal, Québec, Canada, H3C 1K3,
Tel.: (514) 396-8998, Fax.: (514) 396-8684

Abstract

Recent advances in high-speed networks, rapid improvements in microprocessor design, and availability of highly performing clustering software implementations enables cost-effective high-performance parallel computing on clustered low cost workstations and PCs. Such clusters are very attractive because they rely on available off the shelf hardware and software technologies. To simplify programming in network environments and to realize component-based software architectures, many models have emerged as possible standards, namely, RPC: Remote Procedure Call, DCE: Distributed Computing Environment, DCOM: Distributed Component Object Model, CORBA: Common Object Request Broker Architecture, PVM: Parallel Virtual Machine and MPI: Message Passing Interface. The MPI approach is considered one of the most mature methods currently used in parallel programming. MPI implementations on emerging cluster interconnects are an important requirement for useful parallel processing on cost-effective computer clusters. This paper offers an overview of different parallel computing environments and models. Functionality and performance of MPI running on homogeneous and heterogeneous workstation clusters are compared. The project goals, laboratory environment, performance measurements, future work, and a conclusion are presented.

1. Introduction

Scientific applications are still the driving force behind the development of those parallel computing technologies needed to manipulate large databases, accelerate code execution, and resolve excessive time consuming problems.

The current trend in high performance computing is clustering and distributed computing. In clusters, powerful low cost workstations and/or PCs are linked through fast communication interfaces to achieve high performance parallel computing. Recent increases in communication speeds, microprocessor clocks, protocol efficiencies coupled with the availability of high performance public domain software including operating system, compiler tools, and message passing libraries, make cluster based computing appealing in terms of both high-performance computing and cost-effectiveness. It is well known that different architectures require different parallel implementations and recent publications on clustering and distributed computing have spurred research interests, amongst other things, in the implementation of software tools for efficient messaging layers and programming environments. Parallel methods and software must be easy to modify because algorithms efficiency depends on the target computer characteristics. Furthermore different parallel computers require different data-structure implementations.

Parallel computing on clustered systems is a viable and attractive proposition due to the high communication speeds of modern networks. To efficiently use more than one processor in a program, the processors must share data and co-ordinate access to and updating of the shared data. The most popular approach to this problem is to exchange data through messages between computers. The MPI (Message Passing Interface) approach is considered one of the most mature methods currently used in parallel programming mainly due to the relative simplicity of using the method by writing a set of library functions or an API (Application Program Interface) callable from C, C++ or Fortran programs. MPI was designed for high performance on both massively parallel machines and clusters. Today, MPI is considered a de facto standard for message passing in the parallel-computing paradigm.

In our project we plan to program as a single computing resource an MPI based heterogeneous cluster.
We strive to increase available computing power and obtain better performance by using numerous low cost readily available processors. This paper presents our project, a description of the environment, a preliminary performance report, an assessment of the network load, and a summary of future work.

2. Parallel computing environments

Achieving maximum performance with parallel architectures requires specific environments built from a physical-mathematical model, algorithms, programs, and a hardware configuration[18].

WEB explosive growth, PCs increasing popularity and increased access to high-speed networks have pulled distributed computing and message-passing programming in the main stream. To simplify network programming and to realise component-based software architecture, many environments or models have emerged as standards, namely, RPC (Remote Procedure Calls), DCE (Distributed Computing Environment), DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture), and Message Passing like PVM (Parallel Virtual Machine) and MPI (Message Passing Interface)[41, 20].

2.1. Distributed models

A distributed model is an application model that represents local and remote objects in a transparent manner. To simplify network programming and to realise component-based software architecture, many distributed models have emerged as standards, namely, RPC[23], DCE[20], DCOM[46], and CORBA[4].

2.2. Message passing overview

Today, message passing environments like PVM[40] and more recently MPI[3], are considered the most popular programming methods for parallel computers (including massively parallel computers and PC/workstation clusters). Currently, message passing remains the surest means to obtain high performances for the majority of distributed applications. Its flexibility permits to parallel all types of applications (client-server, data-parallel, embarked and real-time systems...). Message passing is easy to understand, portable, interoperable and effective.

The principle of message passing rests on tasks cooperation through explicit message exchanges carried out as point-to-point communication between two processes or between several processes and a unique communication task.

2.2.1. Different PVM environments. PVM is a software package that permits a heterogeneous collection of Unix and/or NT computers linked by a network to be used as a single large parallel computer. Thus large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers. PVM enables users to exploit their existing computer hardware to solve much larger problems at minimal additional cost[30].

Many PVM environments exist and improved versions were produced for different platforms, namely CUMULVS (Collaborative User Migration, User Library for Visualisation and Steering)[23], HP-PVM (High Performance Parallel Virtual Machine)[24], DMPVM (Dynamic Allocation and Migration Parallel Virtual Machine)[25], jPVM[26], WPVM (Windows Parallel Virtual Machine)[27].

2.2.2. Different MPI environments. In most MPI implementations a fixed set of processes is created at program initiation. These processes may execute different programs. From a software engineering perspective, the most important feature of MPI is its support for modular programming. A communicator allows the MPI programmer to define modules that encapsulate internal communications structures[3]. Many MPI environments exist for different platforms namely FM-MPI, WMPI, MPI/Pro and PaTENT.

FM-MPI from the Concurrent Systems Architecture Group (CSAG) is a version of MPICH built on top of Fast Messages. The FM (Fast-Message) interface is based on Berkeley Active Messages. The FM interface was originally developed on Cray T3D and a cluster of SPARCstations connected by Myrinet (a programmable network interface card capable of providing 160 Mbytes/sec links)[6, 9, 10]. Recently, a variant of FM-MPI that runs on top of WinSock 2 was released as part of the High-Performance Virtual Machines (HPVM) project being undertaken by CSAG[16, 17].

The Department of Informatics Engineering University of Coimbra, Portugal developed WMPI a full MPI implementation for Microsoft Win32 platforms. WMPI is based on MPICH and includes a P4 API and a P4 device standard that provides the communication internals and start-up mechanisms not specified in the MPI standard[11]. The WMPI package is a set of libraries for Borland C++, Microsoft Visual C++ and Microsoft Visual FORTRAN. The WMPI release provides libraries, header
files, examples and daemons for remote starting. These daemons are connected between them and with user processes through UDP and TCP sockets which provide WMPI with good scalability. WMPI can co-exist and interact with MPICH/ch_p4 in a cluster of mixed Win32 (Windows 95 and NT) and UNIX workstations over TCP/IP network.

MPI/Pro (for Windows NT) is a commercial environment released in April 1998 by MPI Software Technology, Inc. MPI/Pro is based on WinMPIch from the Engineering Research Centre at Mississippi State University. The current version of MPI/Pro is fairly radically redesigned to remove the bottlenecks and other problems that were present. The MPI/Pro developers are currently working on a new source base for MPI that does not include any MPICH code and supports the Virtual Interface (VI) Architecture[13]. The current MPI/Pro release supports both Intel and Alpha processors and was tested on Microsoft Visual C++ and Digital Visual Fortran[6]. MPI/Pro provides multi-device architecture that allows MPI applications to efficiently exploit SMP (Symmetric Multi Processors) parallelism; multithreaded design; user level thread safety; asynchronous method of synchronization and notification; optimized persistent synchronization mechanisms protects 3rd party software. MPI also efficiently manages message buffers, can efficiently program MPP (Massively Parallel Processors) and clusters, is totally portable, is formally specified, is a standard, and its synchronization mechanisms protects 3rd party software.

Recently, the University of Tennessee and Oak Ridge National Laboratory have begun investigating the feasibility of merging features of MPI and PVM. The project is called PVMPI and involves creating a programming paradigm that allows access to the virtual machine features of PVM and the message passing features of MPI[22]. PVMPI is a software system that allows independent MPI applications running under different MPI implementations and using different language bindings to communicate[21].

Today, MPI has become a de facto standard for message passing in the parallel-computing paradigm, and MPI groups are solid and efficient. Such publications pointed out that MPI is preferable to PVM for many reasons[39]. MPI offers more than one high quality implementation available, full asynchronous communication, and 3rd party profiling mechanism definition. MPI also efficiently manages message buffers, can efficiently program MPP (Massively Parallel Processors) and clusters, is totally portable, is formally specified, is a standard, and its synchronization mechanisms protects 3rd party software.

### 2.3. Parallel computing methods

Parallel programming design begins with the choice of several models or methods of parallelism can be used in parallel applications. The models of parallel computing are very different. There are many models differing in their flexibility, task interaction mechanisms, task granularities, support for locality, scalability, and modularity. In the following we present the principal models[32].

Task/channel model: is often used to describe algorithms. A task encapsulates a program and local memory and defines a set of ports that define its interface to its environment. A channel is a message queue into which a sender can place messages and from which a receiver can remove messages, “blocking” if messages are not available[31].

Message passing: is probably the most widely used parallel programming model today. Message-passing programs, like task/channel programs, create multiple tasks each encapsulating local data. Each task is identified by a unique name, and tasks interact by sending and
receiving messages to and from named tasks. In this respect, message passing is really just a minor variation on the task/channel model, differing only in the mechanism used for data transfer. For example, rather than sending a message on a "channel" we may send a message to a "task"[1]. There are two methods based on MPI and PVM[1] message-passing routines:

2.3.1. Data-passing resembles message-passing except that routines exchange data instead of messages, and they use one-sided communication, that is, one processing element (PE) can send or receive data from another PE without the knowledge of that PE[1].

2.3.2. Shared Memory provides tasks with a common asynchronous read and write shared address space which access is controlled by lock and semaphore mechanisms[32]. Two models exist: manual shared-memory directives where parallelization directives (special comment lines) are manually inserted to guide the compiler and automatic manual shared-memory directives where compilers analyse and restructure the program and generate code that splits loops processing among multiple processors[1].

2.4. Designing parallel algorithms

Four attributes are needed to achieve parallel algorithms: concurrency, scalability, locality, and modularity. Concurrency refers to the ability to perform many actions simultaneously and is essential if a program is to execute on many processors. Scalability indicates resilience to increasing processor counts and is equally important as processor counts appear likely to grow in most environments. Locality means a high ratio of local memory accesses to remote memory accesses (communication). This is the key to high performance on multicomputer architectures. Modularity (decomposition of complex entities into simpler components) is an essential aspect of software engineering in parallel computing as well as sequential computing.

Parallel algorithm design benefits largely from a methodical approach that maximises the range of options considered, provides mechanisms for evaluating alternatives, and reduces backtracking costs associated with bad choices. An algorithm design methodology structures the design process as four distinct stages: partitioning, communication, agglomeration, and mapping.

First, the problem is partitioned into many small tasks. This partitioning can be achieved by using either domain or functional decomposition techniques. The next step is to organize the required communication in order to obtain the data required for task execution. Then, the agglomeration stage consists in decreasing communication and development costs, while maintaining flexibility if possible. Finally, the tasks are mapped to processors, typically with the goal of minimising total execution time. Load balancing or task scheduling techniques can be used to improve mapping quality[1].

3. Environment description

In our project, we used MPI on heterogeneous clusters composed of workstations running MPICH under Solaris 2.6 (SunOs 5.6) and WMPI under Windows NT 4.0. In the PCs, we ran WMPI the Win32 MPI version produced by the Portuguese Instituto Superior de Engenharia de Coimbra. The test applications were compiled using Microsoft Visual C++ 6.0. Computers used are located in the advanced computing laboratory (LCA) of École de Technologie Supérieure (ÉTS). All workstations are accessible through individual 10 MHz Ethernet emulated links and the two servers communicate through individual 100 MHz Ethernet emulated links. In the laboratory are ten (10) SUN 140E workstations, one (1) SUN 200E fast server, one (1) Enterprise 6000 multiprocessor SMP server composed of 14 167 MHz CPUs, all running MPICH v1.1.2 under Solaris 2.6. In the same room and linked to the same network are ten (10) Dual 400 MHz Pentium processors running WMPI v1.3 under Windows NT4.0sp4 workstation software, and one (1) identical server running WMPI under Windows NT4.0sp4 server software. Table 1 summarises the computer characteristics.

<table>
<thead>
<tr>
<th>Computer</th>
<th>CPU</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise 6000 (14 CPU)</td>
<td>14 UltraSPARC 167 MHz</td>
<td>1.7 GB</td>
</tr>
<tr>
<td>Fast server Sun 200E</td>
<td>UltraSPARC 200 MHz</td>
<td>256 MB</td>
</tr>
<tr>
<td>Sun 140E</td>
<td>UltraSPARC 143 MHz</td>
<td>64 MB</td>
</tr>
<tr>
<td>Server NT</td>
<td>Pentium 200 MHz</td>
<td>64 MB</td>
</tr>
<tr>
<td>PC Computers</td>
<td>2 Pentium 400 MHz</td>
<td>256 MB</td>
</tr>
</tbody>
</table>

4. Performance results

This section presents the performance measurements of applications using MPI on homogeneous and heterogeneous workstations clusters composed of up to ten SUN 140E calculators up to ten dual 400 MHz Pentium calculators, and the Enterprise 6000 SMP server.
The following table presents the start-up (required time to initialise the MPI execution environment) time average variations relative to the number of CPUs used to execute the same application on both homogeneous and heterogeneous workstation clusters. The application is executed in modes allowing up to six processes to be activated on each calculator.

The column titled #CPU contains the number of CPUs used in the application. The other columns represent start-up time averages. Under homogeneous stations columns are shown the application start-up time as it is executed separately on the same number of Sun and Windows workstations. It is noticeable that in the Windows case, the CPU number is doubled. The heterogeneous columns also represent the application start-time averages as it is executed on the same number of Unix and Windows workstations added, in some cases, to the Enterprise station. In these cases the number of CPUs showed in the # CPUs column is the sum of CPUs used in the workstations. The application is executed respectively on clusters of one, four, seven and ten Windows and Unix computers.

<table>
<thead>
<tr>
<th>CPU #</th>
<th>Homogeneous stations</th>
<th></th>
<th>Heterogeneous stations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Windows and Unix master on</td>
<td>Windows, Unix and Enterprise master on</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unix Win</td>
<td>Unix Win</td>
<td>Unix Win</td>
<td>Unix Win</td>
</tr>
<tr>
<td>1</td>
<td>1.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4.2</td>
<td>6.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>6.2</td>
<td>9.0</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>66</td>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>50</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>120</td>
<td>240</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>76</td>
<td>91</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>130</td>
<td>160</td>
<td>140</td>
<td></td>
</tr>
</tbody>
</table>

### 5. Future work

Future work will consist in improving and refining the functionality, and performance characteristics. Experimenting with MPI on a large heterogeneous environment are ongoing efforts. In the longer term, improvements at the levels of communications, synchronization, and optimization are planned.

### 6. Conclusion

In massively parallel computers, processor speed and memory size are no longer the only factors to ponder. Interprocessor communication speeds and interprocess synchronization need to be taken into account. The MPI message-passing library allows us to explicitly control communications. In terms of functionality and performance WMPI appears to be the best environment for MPI applications on heterogeneous workstations.

In this paper we presented and compared processor start-up speeds in both homogeneous and heterogeneous workstations. We illustrated the cases when the master process is running on Unix (Solaris 2.6) and the slaves are running on Windows NT 4.0 and a SUN Enterprise 6000 SMP. We also illustrated the reverse when the master is running respectively on Windows NT 4.0 and the SUN Enterprise and the slaves are running on UNIX workstations and Windows NT 4.0 PCs. We presented the results of the same application executed on homogeneous or heterogeneous clusters composed of a varying number of computers. This application was executed in a mode running more than one process on each processor.

Our results suggest that the average start-up time on Windows NT 4.0 and on Solaris 2.6 is directly proportional to the number of computers activated. The average start-up time on Windows is generally shorter than on Unix. This may be due to the RAM capacity (256 MB in Windows NT and 64 MB in Solaris) and to the speed and number of processors in each computer. But in case of processes running on heterogeneous workstations, the average start-up time on Unix is generally shorter than on Windows and Enterprise. Also, in this case this average on Enterprise is shorter than on Windows.

To conclude, a fast communication interface and a powerful integrated tool for network computing are needed in order to exploit as a parallel virtual computer, emerging heterogeneous workstations linked by a fast network. Increased functionality and performance enhancements are also needed.
7. References