Improving the Performance of Scientific Parallel Applications in a Cluster of Workstations*

A. Flores and J.M. García

Dept. de Ingeniería y Tecnología de Computadores, University of Murcia,
Campus de Espinardo s/n,
30080 Spain
{aflores, jmgarcia}@dif.um.es

Abstract. Recent improvements in LANs make network of workstations a good alternative to traditional parallel computers in some applications. However, in this platform the communication performance is over two orders of magnitude inferior to state-of-art multiprocessors. Currently networking technologies have put the pressure in the software overhead. Because of this, applications could not take advantage of this communication performance potential. In this paper, we present an implementation of Virtual Circuit Caching that reduces the software overhead by allocating communication resource once and re-using them for multiple messages to the same destinations. With this approach, the communication overhead is reduced by approximately a 35% for long messages; this reduction should enable the extensive use of networks of workstations for scientific parallel applications.

keywords: Latency Hiding, Fast Network Interface, Cluster of Workstations, Network Protocol.

1 Introduction

Research in parallel computing has traditionally focused on multicomputers and shared memory multiprocessors. Currently, networks of workstations (NOWs) are being considered as a good alternative to parallel computers. That is due to there are high performance workstations with microprocessors that challenge custom-made architectures. This class of workstations is widely available at relatively low cost. Furthermore, these networks provide the wiring flexibility, scalability and incremental expansion capability required in this environment.

Communication performance at the application level depends on the collaboration of all components in the communication system, especially the network interface hardware and the low-level communication software that bridges the hardware and the application. In principle, scientific parallel applications should be coded in a portable message-passing library, as MPI [7] or PVM [6]. Usually, these libraries are based in the TCP/IP layer. Unfortunately, TCP/IP layer

* This work was supported in part by the Spanish CICYT under Grant TIC97-0897-C04-03

imposes large software overhead [11](several hundred to several thousand instructions per message). As a result, only embarrassingly parallel applications (that is, applications that almost never communicate) can make use of workstation clusters. To solve this problem, we can design better layers or we can improve the TCP/IP layer. In both cases, the final objective is the same, that is, improving the network interface rather than the network switches and links [3].

Currently, in the first approach there are new layers with reduced overheads (that is, on the scale of tens of instructions per message). Active Messages [5] is one of this. This layer offers simple, general-purpose communication primitives as a minimal layer over the raw hardware. However, it is not intended for direct use by application programmers and it is not portable. In fact, Active Messages provide low level services from which communication libraries can be built. The current prototype of Active Messages is the GAMMA project [2]. GAMMA exploits a more general Active Message-like model inspired to Thinking Machines' CMAML for the CM-5, where each process has a number of *communication ports* and may attach both a receiver handler and a data structure to a single communication port in order to handle all messages incoming through that port. This technique is suitable for MIMD as well as SPMD programming, as it does not require a common address space among cooperating processes.

The second approach is more interesting form point of view of portability. The main objective in this approach is to improve the network performance via to reduce the latency of the network. Traditionally, latency and throughput are the two parameters used to indicate the performance of an interconnection network. Although these parameters are very related, it is easier to increase the peak throughput -achievable for long messages-, but it is harder to reduce the latency -because the software overhead-. Recently, several works have been developed in this way as Pupa [12] or Beowulf [1]. These approaches have been implemented over off-the-shelf network hardware and they co-exist with legacy communication software such as TCP/IP.

Our research line is in the later approach. Nowadays, it does make sense from a technological as well as financial point of view to use off-the-shelf network hardware with existing software, usually based on TCP/IP. So, it is very important to study this protocol and try to reduce the large software overhead that it imposes. In this paper, we focus on analyzing the TCP/IP layer to detect the main software overhead points, and to solve this problem via the Virtual Circuit Caching technique. In this way, we can significantly reduce the communication latency in the network.

The rest of the paper is structured as follows. In the next section we analyze sources of software overhead in the communication system. In section 3, we describe the techniques used to reduce software communication overhead. The results of running test programs are analyzed in section 4. Finally, some conclusions and ways of future work are drawn.

2 Analysis Software Overhead Sources

As Local Area Networks improve the performance, their interfaces, resources, organization, and integration into their host computer become increasingly important. Recent papers have pointed out that announced performance for such as LANs could suffer a severe degradation due to overhead in communication software. Currently, with the most widespread use of ATM [8] in LANs and the announcement of Ethernet Gigabit standard [9], bottleneck studies and alternative solutions become essential.

To make an exhaustive study of software overhead, we must evaluate the communication channel performance at several layers in the protocol stack in order to identify bottlenecks. Using this technique we can measure the added overhead of each layer, the total software overhead and the network congestion.

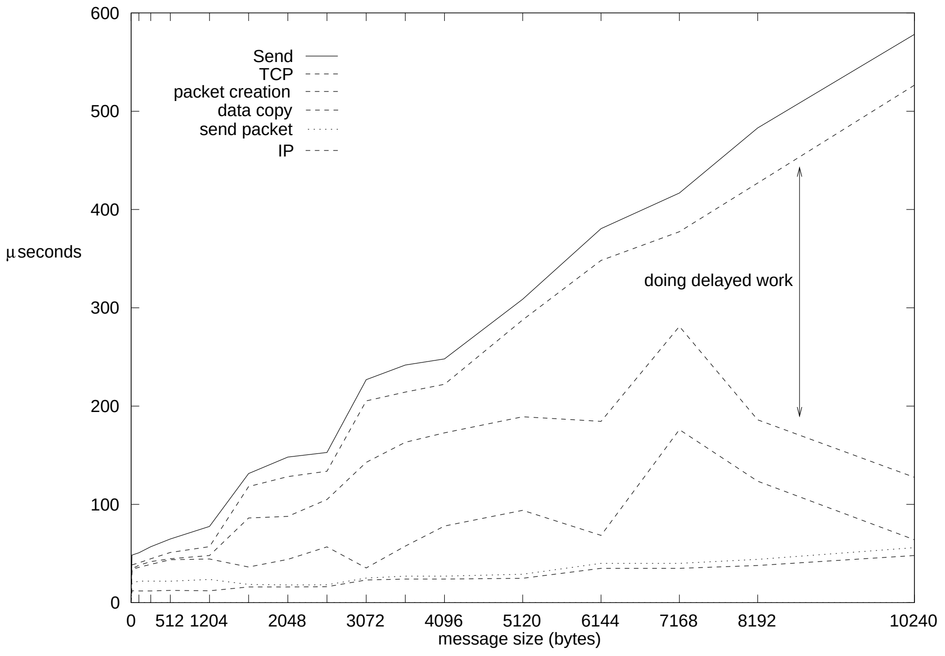

Fig. 1. Detailed study of software overhead in TCP/IP layer.

Figure 1 show software overhead breakdown in a TCP send call. Data was obtained take advantage of TCP/IP source code available in LINUX operating system. By introducing measurement instructions in appropriate points of TCP/IP implementation, we could obtain real overhead of critical sections of TCP/IP protocols. To obtain an accurate measurement we measure a portion of TCP/IP source code at once. The measurement overhead was estimate in

1μsecs. Measurement results were retrieve via an additional system call that we introduce for this purpose.

As it can be seen in the figure, the send latency is composed by various factors. This figure shows two important facts. Firstly, packet creation and data copy are the main causes of the send software overhead as expected. The second thing is that exists a significant difference between TCP layer overall overhead and the breakdown of this overhead. That is, the overhead obtained by adding the measured overhead of critical sections of TCP/IP communication layer. Although there are lines of code that are not taken into account, their impact in the overhead is very small. The real reason is the behavior of LINUX TCP/IP protocols with high workloads. In this case, most of the work is delayed in order to avoid saturation in the receiver side. This delayed work will be done as soon as possible, interrupting the normal program execution and introducing a software overhead that have been labeled as "*doing delayed work*" in figure 1.

Above, we show packet creation and data copy were the main causes of software overhead in a send system call. So, it should be possible to improve the network performance by reducing these two overhead sources. In the next section, we show an approach that minimizes these two overhead causes.

3 Latency Tolerance and Reduction Techniques

In previous section, we identified two main sources of software overhead in communication events. Namely, overhead due to allocation of resources and overhead due to data copy from/to user memory space.

The first factor can be greatly alleviated by implementing software techniques of resource reutilization. In [4], authors propose a new technique named Virtual Circuit Caching (VCC) that enable reduction in communication overheads by allocating communication resources once and re-using them form multiple messages to the same destination.

In the MPI/RT Standard Draft [10], is proposed a similar technique in buffer management, allowing to the operating system buffer re-use. This technique also allows zero copy data transfer at the expense of limited buffer length.

We propose a transparent software implementation that allows buffer reutilization at top level of protocol stack. Figure 2 show the basic idea of resource utilization. In this case, the resource to re-use is packet structure used by operating system.

The idea is the following. When a packet is sent, the packet structure (protocols headers and data space) is not freed but insert in a pool of free packets. In this way, they can be reused to hold new data, avoiding the overhead of handling the creation and deallocation of messages.

The second factor (software overhead due to data copy from/to user memory space) is more difficult to hide because the overhead can not be distributed among several messages. However, a precommunication technique can be applied in some cases obtaining good results. Next, we show an example of communication pattern when this technique can be applied.

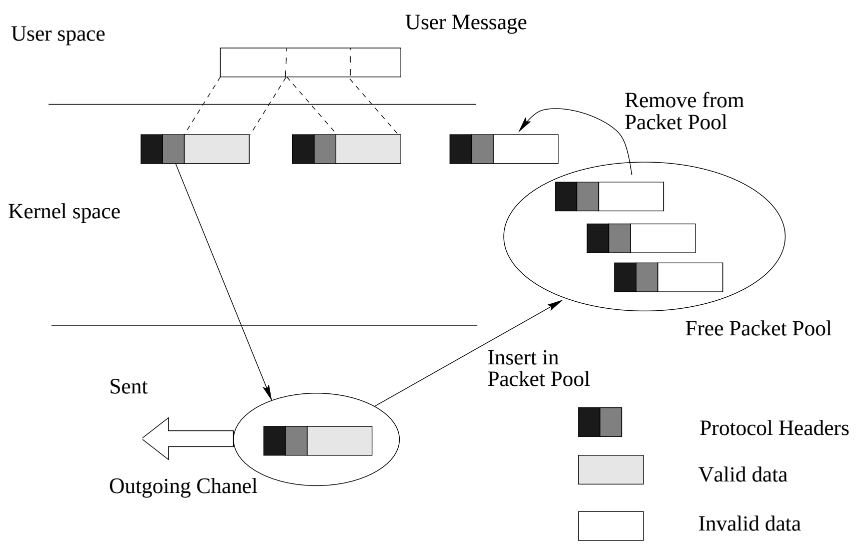

Fig. 2. Overview of VCC software implementation (Opaque to users).

Example of no dependence pattern

```
compute message A;
recv(token,length(token));  /* Waiting for permission.     */
send(dest,A,length(A));     /* Permission granted, send A. */
```

We have observed this communication pattern in programs with global communication events, such as barriers, where all processes wait in the barrier until last process arrive. We can use this idle time in order to process subsequent send events in the assumption of no data dependence. In order to deal with this communication pattern, we have include a new system call, named *indication*, that allows an earlier processing of the *send* system call. Then, the modified code with the precommunication technique is seen as follows:

Precommunication of message A

```
compute message A;
indication(dest,A,length(A)); /* There is a send with no     */
                              /* dependence.                 */
recv(token,length(token));    /* Waiting for permission;     */
                              /* begin to process A sending. */
send(dest,A,length(A));       /* Permission granted, send A. */
```

4 Result Evaluation

4.1 Hardware and Software platforms

We have performed our tests on a cluster of workstations with Intel Pentium 200Mhz processor, 32 Mbytes main memory, 256KBytes cache memory, 1GByte IDE hard disk and Fast Ethernet SMC EtherPower 10/100 adapter. The operating system used is Linux 2.0.32.

4.2 The test program

To evaluate performance parameters, we make use of Ping-Pong test. This test program is written in standard C. The best compiler option is always applied, unless otherwise noted. The test Ping-Pong is a simple echo between two adjacent nodes. A receiving node simply echoes back whatever it is sent, and the sending node measures round-trip time. When precommunication technique is used, we suppose no dependence between send and receive messages, although program order is maintain (a message is not sent until the process receives the matching message). Times are collected for some number of iterations (we have used 10000) over various messages sizes and after a transient period of 500 iterations. The minimal time, the maximal time and the mean time from all processes are collected. To interpret the results, we focus on the average time, because it is more representative of the performance the user can obtain from the machine.

4.3 Results

Figure 3 show software overhead for messages up to 5K. Results are obtain by subtract delay due to physical transmission from round-trip time. In the ideal case, hiding of almost all software overhead, results will be near to zero. Results are shown for three cases: original TCP/IP protocols, TCP/IP with VCC (Virtual Circuit Caching), and TCP/IP with precommunication and packet VCC.

From figure 3 several interesting conclusions follow. First of all, using a transparent technique such as packet reuse mechanism we reduce software overhead in a range that vary from 5-10% for short messages up to 15-35% for long ones.

Another remarkable fact is the software overhead reduction when using precommunication. For messages up to 1K, utilization of both techniques: precommunication and VCC packet show worse performance that packet reuse technique. This is due to new software overhead sources. Namely, *indication* system call and modifications introduced in *recv* system call to deal with precommunication. For longer messages, this overhead is overwhelmed with benefits of an earlier buffer copy.

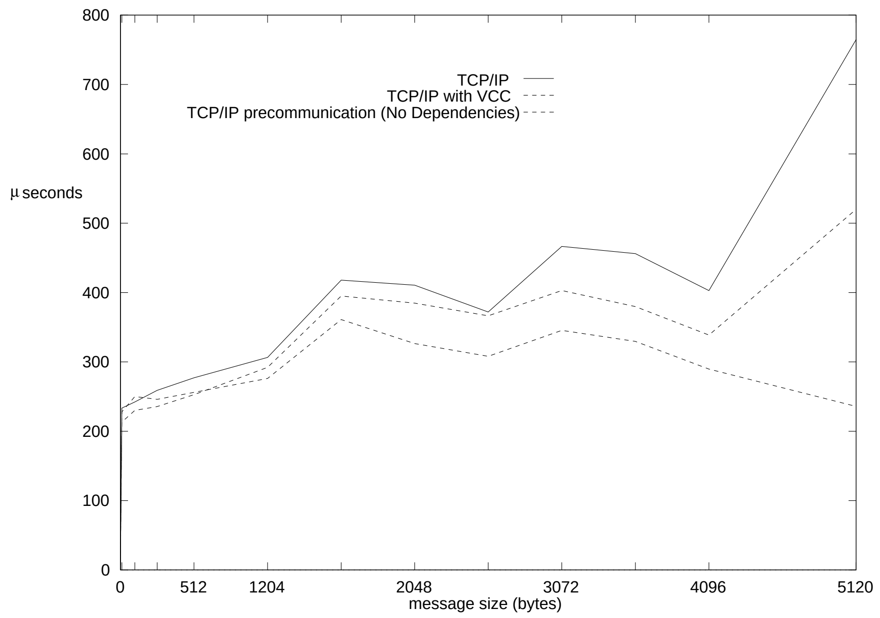

Fig. 3. Software overhead in TCP/IP protocol.

5 Conclusions and future work

The emergence of high-bandwidth, low-latency networks makes the use of workstation clusters attractive for parallel computing applications. Up to date, the main problem in this environment is the high latency compared to integrated custom multiprocessors.

In this paper, we have presented a software implementation of Virtual Circuit Caching (VCC) that enables reductions in communication overheads by allocating communication resource once and re-using them for multiple messages to the same destinations. The software overhead is reduced in a range that varies from 5-10% for short messages up to 15-35% for long ones. In same cases, further reduction in communication overheads can be achieved using stalls in process execution to process messages ahead in program order.

Our future work is focused on testing these techniques with usual scientific applications (FFT, LU, etc). Furthermore, we are currently examining the latest generation of ATM network interfaces and the Gigabit Ethernet standard to study the application of our techniques to these new interfaces.

References

1. D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawake, C. Packer: Beowulf: A Parallel Workstation for Scientific Computation. Procc. of Int. Conference on Parallel Processing, 1995.

2. G. Ciaccio: Optimal Communicationn Performance on Fast Ethernet with GAMMA. Procc. of the Workshop on Personal Computer Based Networks of Workstations, IPPS/SPDP, 1998.
3. D.E. Culler, L.T. Liu, R.P. Martin and C.O. Yoshikawa: Assessing Fast Network Interfaces. IEEE Micro, Vol. 16, No. 1, pp. 35-43, Feb. 1996.
4. B.V. Dao, S. Yalamanchili, and J. Duato: Architectural Support for Reducing Communication Overhead in Multiprocessor Interconnection Networks. Procc. of the Third Int. Symp. on High Performance Computer Architecture, February 1997.
5. T. Von Eicken *et al*: Active Messages: A Mechanism for Integrated Communication and Computation. Procc. of the 19th ISCA, pp. 256-266, May 1992.
6. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam: PVM: Parallel Virtual Machine. The MIT Press, 1994.
7. W. Gropp, E. Lusk and A. Skjellum: Using MPI. The MIT Press, 1996.
8. R. Handel, M.N. Huber and S. Schroder: ATM Networks, Concepts, Protocols, Applications. Addison-Wesley 1994, 2/e.
9. Gigabit Ethernet Alliance. DRAFT Document for IEEE 802.3z Gigabit Ethernet Standard. *http://www.gigabit-ethernet.org*.
10. Real-time Message Passing Interface (MPI/RT) Forum: DRAFT Document for the Real-time Message Passing Interface (MPI/RT) Standard. *http://www.mpirt.org*, May 1998.
11. J. Piernas, A. Flores, J. M. García: Analyzing the Performance of MPI in a Cluster of Workstations base on Fast Ethernet. Procc. of 4th European PVM/MPI Users' Group Meeting, pp. 17-24, November 1997.
12. M. Verma, T. Chiueh: Pupa: A low Latency Communication System for Fast Ethernet. Procc. of the Workshop on Personal Computer Based Networks of Workstations, IPPS/SPDP, 1998.