Message-Passing Performance of Parallel Computers

Vladimir Getov[1,2], Emilio Hernández[3], Tony Hey[2]

[1] School of Computer Science
University of Westminster, Harrow Campus, London, UK
[2] Department of Electronics and Computer Science,
University of Southampton, Southampton, UK
[3] Departamento de Computacion,
Universidad Simon Bolivar, Apartado 89000, Caracas, Venezuela

Abstract. In this paper we investigate some of the important factors which affect the message-passing performance on parallel computers. Variations of low-level communication benchmarks that approximate realistic cases are tested and compared with the available Parkbench codes and benchmarking techniques. The results demonstrate a substantial divergence between message-passing performance measured by low-level benchmarks and message-passing performance achieved by real programs. In particular, the influence of different data types, the memory access patterns, and the communication structures of the NAS Parallel Benchmarks are analysed in detail, using performance measurements on the Cray-T3D in Edinburgh and the IBM-SP2 in Southampton.

1 Introduction

Generally speaking, the time taken for communications represents a substantial part of the performance penalty that one has to pay when using message-passing parallel computers. In most cases, the communication subsystems on such machines work in a *pipeline* fashion, and therefore, the communication time can be modelled as a simple linear function of the message length [9]:

$$t_{com} = t_0 + t_{trans} n \tag{1}$$

where t_0 is the message startup time, t_{trans} is the transmission time per byte, and n is the message length in bytes.

Perhaps the simplest measure of message-passing performance is the *ping-pong* or *echo* benchmark between a pair of nodes, which has been described by several authors [9,5,2]. In this test one node sends a message to the other, which in turn, receives the message and sends it back immediately to the first node. Half the time for this test is recorded as the time to send a message of the given length. The message-passing performance can be characterised by extending Hockney's performance description for vector pipelined processors over the above model

using the asymptotic performance, r_∞, and the half-performance length, $n_{\frac{1}{2}}$, parameters [5].

$$t_{com} = \frac{n_{\frac{1}{2}}^c + n}{r_\infty^c} \tag{2}$$

Although the communication subsystems of tightly coupled parallel computers have been improving very quickly over the last few years, this same simple model is also valid for multistage interconnection networks based on high-speed communication switches [10].

Different sets of low-level benchmarks may be defined, depending on the aims of the particular evaluation exercise. If the benchmarking goal is *to compare* the performance of different parallel computers, the low-level benchmarks are necessary to mark off the bounds of calculation and message-passing performance. The communication patterns which outline the message-passing performance bounds may be grouped into four categories [4]: single point-to-point messages; single broadcasts; point-to-point transmissions from all nodes; broadcasts from all nodes (multibroadcasting). If, however, the benchmarking goal is *to estimate* the application performance, the low-level benchmarks must provide much more accurate and sometimes application-specific time measurements. There have been attempts to relate message-passing performance of real applications to the results of the existing low-level communication Parkbenchmarks [8]. However, it is difficult to predict the performance of a particular communication pattern, for instance, a circular shift, by the currently available low-level Parkbenchmarks, as they have been designed for performance comparisons and not for performance predictions. One of the main reasons for this is the fact that the communication time depends not only on the message length, but also on the current workload of the interconnection network [2], because the network itself cannot be considered as a single resource. It has input and output ports for each node in the distributed memory system, all of which may operate simultaneously, and it is therefore fallacious to draw conclusions about the aggregate message-passing performance from simple ping-pong measurements. Such tests highlight only the case of a single message in isolation and could not be useful as a basis for performance predictions of higher level benchmarks and application codes.

In order to tackle this problem, a list of performance models for frequently used communication patterns can be employed to estimate the message-passing performance of an application, but such a list may be very large. A more pragmatic alternative is to extract the communication structure from the application kernel and to evaluate the asymptotic communication parameters by varying the message length for different numbers of processors. In this article we follow the latter approach in order to demonstrate the substantial divergence between message-passing performance measured by the existing low-level benchmarks and message-passing performance achieved by real programs. The next section shows results related to transmitting data elements other than bytes, which is a common requirement in scientific computing. Section 3 focuses on the performance effects of data movements associated with message-passing, while section 4 presents results related to real communication structures, extracted from

the NAS parallel benchmarks [1]. Finally, section 5 discusses the conclusions to the study.

2 Alternative data types

Modern processors can load/store a 64-bit word with a single instruction. The memory access speed depends on the data type and is usually optimised for double precision (64-bit) words. In order to investigate how the choice of data type affects the message-passing performance, the COMMS1 low-level benchmark was modified to send contiguous and non-contiguous double precision elements, rather than bytes. Such performance figures are closer to those a real scientific application would yield. Table 1 shows measurements made with the original COMMS1 and a version of COMMS1 modified to send double precision data type elements. The experiments were conducted on a Cray T3D and an IBM SP2. The T3D used for the experiments has 150MHz 21064 Alpha processors, each with 64MB local memory, 8KB instruction cache and 8KB data cache, while the operating system was UNICOS 8.0.4 with a cf77 6.2.1 Fortran compiler. The compiler directives used were "-dp -Oscalar3". Meanwhile, the IBM SP2 nodes used were "thin 1" type nodes, each with a 66MHz Power2 processor, 128MB RAM (64bit memory bus), 32KB instruction cache and 64KB data cache, with an AIX 4.1.4 operating system and a xlf 3.2.5 Fortran compiler. The codes were compiled with "-O3 -qstrict" compiler directives.

	r_∞^c	r_∞^c (MB/s)	$n_{1/2}^c$	$n_{1/2}^c$ (Bytes)	Startup Time
Orig. (SP2)	28.592 MB/s	28.592	2844.935 B	2844.935	99.501 μsec
Modif. (SP2)	4.188 Mdp/s	33.504	897.039 dp	7176.312	214.174 μsec
Orig. (T3D)	28.466 MB/s	28.466	1966.929 B	1966.929	69.098 μsec
Modif. (T3D)	3.709 Mdp/s	29.672	387.048 dp	3096.384	104.359 μsec

Table 1. Bandwidth, $n_{1/2}^c$, and startup time reported by COMMS1, using bytes (B) and double precision elements (dp). The modified version transmits double precision elements.

Even though the bandwidth measured when transmitting bytes is similar on the T3D and the SP2, there is a key difference in message-passing performance between these two machines when the ping-pong benchmark uses double precision elements. Normalised measures of both r_∞^c and $n_{1/2}^c$, in Mbyte/s and bytes respectively, change when 64-bit words are transmitted.

3 Memory Access Patterns

Data movements related to message-passing are very common in distributed-memory parallel applications. For instance, data are frequently moved from non-contiguous locations to the communication buffer when a row of a matrix stored

by columns is transmitted, and vice versa. In order to test the communication bandwidth when the data are non-contiguous, several variants of the COMMS1 benchmark were created. These variants use either DO loops to explicitly transfer data from the original non-contiguous location to the message buffer, or the proper message-passing library support in order to make such data transfers. In this way, we are not only testing the impact of the data movements themselves, but also the ability of different message-passing library implementations to perform such operations efficiently. The stride used in all examples was 16 double precision elements – a stride big enough to load a different cache line into cache for every element copy, in the architectures under consideration.

The Message-Passing Interface (MPI) provides a mechanism to specify non-contiguous data on the basis of *derived data types* [7] and the user can declare the shape and size of the object to be transferred using *data type constructors*. A *data type descriptor* of the object must have been previously created, so that when an MPI_SEND or an MPI_RECV is executed, the data is taken from the specified location according to the data type descriptor information. The software/hardware implementation may have to copy the original object into a buffer before sending it out, relying on the information provided in the data type descriptor. The variants of the MPI version of COMMS1 used to conduct the experiments are sketched in the following boxes:

(1) Non-contiguous data in master and slave (MPI_Datatype solution)

```
/* Master */                    /* Slave */
MPI_TYPE_VECTOR(stride=16)      MPI_TYPE_VECTOR(stride=16)
...                             ...
MPI_SEND(BUFFER)                MPI_RECV(BUFFER)
MPI_RECV(BUFFER)                MPI_SEND(BUFFER)
```

(2) Non-contiguous data in master and slave (DO-loop solution)

```
/* Master */                    /* Slave */
DO I = 1,N                      MPI_RECV(BUFFER)
  BUFFER(I) = MATRIX(16,I)      DO I = 1,N
ENDDO                             MATRIX(16,I) = BUFFER(I)
MPI_SEND(BUFFER)                ENDDO
MPI_RECV(BUFFER)                DO I = 1,N
DO I = 1,N                        BUFFER(I) = MATRIX(16,I)
  MATRIX(16,I) = BUFFER(I)      ENDDO
ENDDO                           MPI_SEND(BUFFER)
```

(3) Non-contiguous data in master only (MPI_Datatype solution)

```
/* Master */                    /* Slave */
MPI_TYPE_VECTOR(stride=16)
...
MPI_SEND(BUFFER)                MPI_RECV(BUFFER)
MPI_RECV(BUFFER)                MPI_SEND(BUFFER)
```

(4) Non-contiguous data in master only (DO-loop solution)

```
/* Master */                    /* Slave */
DO I = 1,N
  BUFFER(I) = MATRIX(16,I)
ENDDO
MPI_SEND(BUFFER)                MPI_RECV(BUFFER)
MPI_RECV(BUFFER)                MPI_SEND(BUFFER)
DO I = 1,N
  MATRIX(16,I) = BUFFER(I)
ENDDO
```

(5) Contiguous data in master and slave

```
/* Master */                    /* Slave */
MPI_SEND(BUFFER)                MPI_RECV(BUFFER)
MPI_RECV(BUFFER)                MPI_SEND(BUFFER)
```

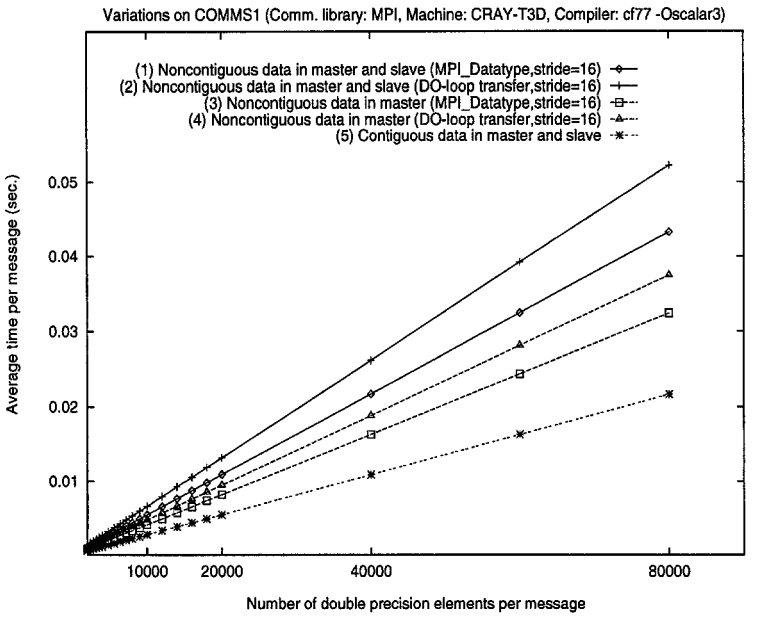

Fig. 1. COMMS1 on CRAY-T3D

The standard COMMS1 benchmark with MPI calls as provided in the Parkbench suite follows the implementation scheme of variant 5. The results obtained from all of the MPI variants on the Cray T3D and the IBM-SP2 are shown in Figures 1 and 2, respectively. The general conclusion that can be drawn from these results is that there is a very significant difference between the raw communication bandwidth and the communication bandwidth with associated data movements. On the SP2, the ratio between the raw communication bandwidth and the slowest version which performs data movements is 5.43, while on the T3D, the same ratio is 2.42. Meanwhile, the ratio between the best implementation of the non-contiguous case and the best implementation of the contiguous case is 4.50 on the SP2 and 2.00 on the T3D. It is interesting that on the SP2, the best option for transmitting non-contiguous data is by means of a DO loop (a performance ratio equal to 1.20), while on the T3D the fastest COMMS1 implementation was the one which uses library calls (here the same performance ratio was 0.83). This demonstrates the importance of the quality of the communication library in terms of performance.

4 Communication Structures

We have used the NAS parallel benchmarks (NPB), version 2.0 [1], in order to verify the validity of the ping-pong benchmark for predicting which communication pattern will execute faster on different machines. The NPB suite consists of two kernels (FT and MG) and three compact applications (LU, BT ans SP).

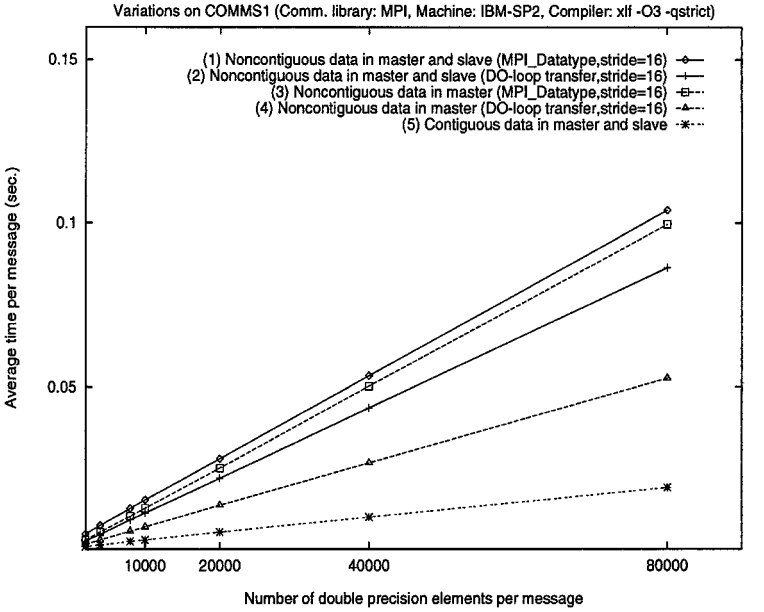

Fig. 2. COMMS1 on IBM-SP2

Our experiments were based on the compact applications, in which the communication skeletons were extracted by commenting out the computation codes of these benchmarks, leaving only the communication functions and the data movements associated with message-passing.

The LU benchmark is a compact application which finds a finite difference solution of the 3-D compressible Navier-Stokes equations. The implementation utilises a block-lower-triangular block-upper-triangular approximate factorisation of the original scheme. A particular characteristic of the algorithm is that it sends a relatively large number of small messages (40 bytes each). Consequently, this benchmark would penalise machines whose interconnect subsystem has a high startup time. The BT benchmark solves three sets of uncoupled systems of equations which are block tridiagonal with 5x5 blocks, while the SP benchmark is similar to BT, but solves a scalar pentadiagonal system. The code structures of the BT and the SP benchmarks are so similar because the systems of equations are solved using Gaussian elimination, without pivoting, in both cases. In contrast to LU, these programs have a coarse-grained communication scheme. In other words, the LU communication kernel should benefit from low latency interconnect subsystems, while the BT and SP communication kernels would execute faster on networks with a greater bandwidth.

Several experiments concerning the communication structure of the NPB kernels were performed on the Cray T3D and the IBM SP2. Figure 3 compares the communication overhead on the T3D and the SP2 for LU, BT and SP by dividing the communication times shown into the "pure" communication time and time spent by explicit data movements related to message-passing. The

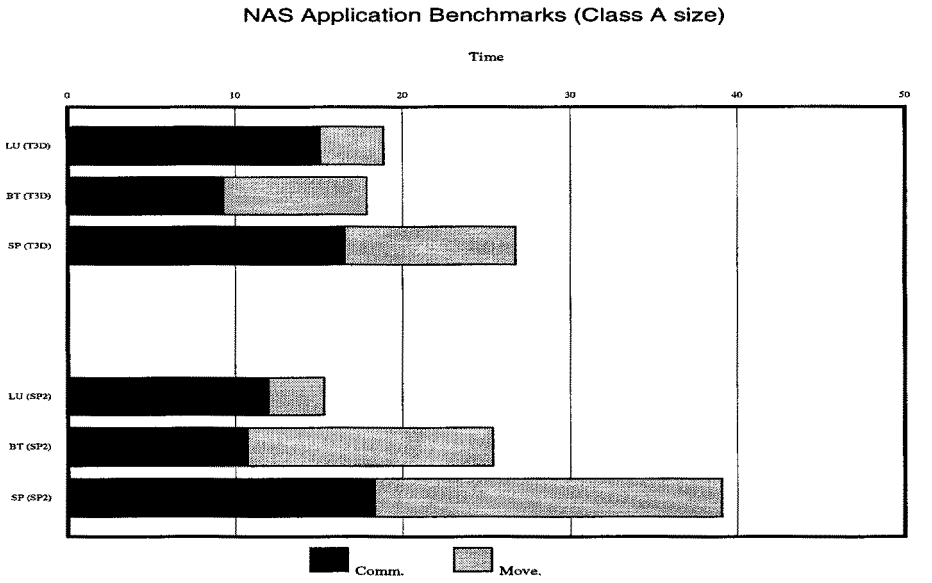

Fig. 3. Comparison of communications in T3D and SP2 (16 processors).

communication skeleton of LU runs marginally faster on the SP2 than on the T3D, while the SP and BT communication skeletons execute faster on the T3D.

The measurements shown in Table 1 were obtained on the same platforms and these COMMS1 results seem to contradict the observed behaviour of the LU, BT and SP communication skeletons on the T3D and the SP2. Apart from the bandwidth and startup time, many other factors not measured by COMMS1, play an important role in message-passing performance. These include the network contention, the presence of collective communication functions, the fact that messages are sent from different memory locations, etc. The execution of the communication skeletons indicates that there is not a substantial difference in message-passing performance between the SP2 and the T3D. The main feature in favour of the T3D, however, is the shorter time spent in data movements associated with message-passing, rather than the communication time itself.

5 Conclusions

The performance analysis of communication subsystems, based on the performance reported by ping-pong benchmarks, has been of great value for comparisons of the message-passing performance across different machines. However, predicting communication performance in terms of performance parameters provided by existing Parkbench codes is not just a function of low-level parameters, measured by simple communication benchmarks. Previous studies have shown that the analytic expression (2) fits well into parallel performance data [6,3]. Unfortunately, the fitted values of parameters turn out to be very different from

those determined by the low-level benchmarks. Therefore, in most cases the existing low-level communication benchmarks are not suitable for performance estimation.

Clearly, the low-level message-passing parameters (r_∞^c, t_0^c) do not represent only the communication bandwidth and latency of the message-passing channels, but as our analysis and measurement results show, they are also aggregate parameters for memory-to-memory transfers between different nodes of a distributed memory computer. The message-passing performance, therefore, depends upon such factors as the data type of the message elements and the memory access patterns, as well as the storage location of the data to be transferred within the memory hierarchy, in addition to the hardware parameters of the communication subsystem. Hence, the trade-off between efficiency and portability, as well as the implementation level of the message-passing paradigm are also of significant importance. A broader spectrum of low-level benchmarks and parameters has to be considered if more accurate performance predictions are to be made.

6 Acknowledgments

Special thanks go to the staff of the Edinburgh Parallel Computer Centre and Computing Services at the University of Southampton, for their assistance in operating the Cray-T3D and the IBM-SP2 installations.

References

1. D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. TR-NAS-95-020, NASA Ames RC, Dec. 1995.
2. T. Dunigan. Performance of the Intel iPSC/860 and Ncube 6400 Hypercubes. *Parallel Computing*, 17:1285–1302, 1991.
3. V. Getov. 1-Dimensional Parallel FFT Benchmark on SUPRENUM. *Lecture Notes in Computer Science*, 605:163–174, 1992.
4. V. Getov and C. Jesshope. Simulation Facility of Distributed Memory System with 'Mad Postman' Communication Network. *Lecture Notes in Computer Science*, 487:224–233, 1991.
5. R. Hockney. Performance Parameters and Benchmarking of Supercomputers. *Parallel Computing*, 17(10/11):1111–1130, 1991.
6. R. Hockney. Computational Similarity. *Concurrency: Practice and Experience*, 7(2):147–166, 1995.
7. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *International J. Supercomputer Applications*, 8(3/4):169–414, 1994.
8. Parkbench Committee. Report - 1: Public International Benchmarks for Parallel Computers. *Scientific Programming*, 3(2):101–146, 1994.
9. D. Reed and D. Grunwald. The Performance of Multicomputer Interconnection Networks. *IEEE Computer*, 20:63–73, June 1987.
10. Z. Xu and K. Hwang. Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2. *IEEE Parallel and Distributed Technology*, 4(1):9–23, 1996.