Evaluation of the communication performance on a parallel processing system

Luís Paulo Santos   Vítor Castro   Alberto Proença

Dep. Informática, Universidade do Minho,
4719 Braga Codex, PORTUGAL
e-mail: psantos@di.uminho.pt

Abstract. This article presents an evaluation study of point-to-point and collective communication performance on a parallel processing system, a 16 node Parsytec PowerXplorer, using three different communication environments: PARIX, PVM and MPI.

1 Introduction

Most current massively parallel processing systems (MPP) are distributed memory message passing computers. Applications developed to run on these machines commonly use three types of communication: point-to-point, collective communication and collective computation [1, 2]. These communication operations incur costs which include software overheads (communication and synchronization protocols), hardware latencies and message delays (network and memory contention).

This article presents a systematic study of communication costs on a 16 node Parsytec PowerXplorer with three different communication environments: PARIX [3], PVM [4] and MPI [2, 5]. The tests were performed assuming the user view of the whole computer system through a high-level language, i.e., a particular software interface to a given computer architecture [6, 7].

This work used the methodology presented by Hwang and Xu [1] for collective operations, which is a generalization of Hockney's model [8]. Most of the recommendations of the PARKBENCH Committee on Parallel Benchmarks [6, 7] are also considered.

Quantifying the costs of communication operations has several advantages. As the understanding of these operations increases, informed decision making during the design and/or execution of parallel applications becomes feasible, and it is easier to identify weaknesses on communication libraries and/or on the workload distribution strategies.

2 Scientific foundations

Roger Hockney proposed the following model for communication times in point-to-point operations [7, 8]:

$$t(n) = t_0 + \frac{n}{r_\infty} \tag{1}$$

where n is the length in bytes of the user data field in the message, t_0 is the startup time (or latency) and r_∞ is the asymptotic bandwidth which is the maximum achievable bandwidth when the message length approaches infinity.

The message length required to achieve a performance of $r_\infty/2$ is given by

$$n_{\frac{1}{2}} = t_0 * r_\infty \tag{2}$$

and is known as the half performance length. $n_{\frac{1}{2}}$ can also be seen as the length of the message that could be sent during the startup time. In practice, $n_{\frac{1}{2}}$ is a good measure of the message length needed to approach r_∞.

The pair of $(r_\infty, n_{\frac{1}{2}})$ parameters completely characterizes the performance of a given operation. For long messages ($n \gg n_{\frac{1}{2}}$) the startup time may be neglected and only r_∞ is needed, while for short messages ($n \ll n_{\frac{1}{2}}$) only the startup time t_0 is necessary, although usually the specific performance ($\pi_0 = t_0^{-1}$) is used instead.

Xu and Hwang [1] generalized this model for collective operations involving p nodes. The time to complete a communication is given by

$$t(n,p) = t_0(p) + \frac{n}{r_\infty(p)} \tag{3}$$

where t is still a linear function of message length n, but the startup time $t_0(p)$ and the asymptotic bandwidth $r_\infty(p)$ are both functions of the number of nodes p involved in the communication. The half-performance length is given by

$$n_{\frac{1}{2}}(p) = t_0(p) * r_\infty(p) \tag{4}$$

An additional metric, the aggregated asymptotic bandwidth R_∞ was derived. R_∞ is the ratio of the total number of data bytes transmitted by all nodes to the total time needed to execute the operation, as n approaches infinity. For a point-to-point communication $R_\infty = r_\infty$; for broadcast, gather, scatter, reduction and scan $R_\infty = p * r_\infty$; for the total exchange $R_\infty = p^2 * r_\infty$.

3 The environment

All experiments were performed on a 16 node PowerXplorer from Parsytec. Each PowerXplorer node contains a Motorola PowerPC 601 80 MHz RISC microprocessor for computation and an Inmos T805 30 MHz Transputer for communication. Both processors are closely coupled via shared memory. The nodes are interconnected via a 2-dimensional grid using the Transputer links (a 4*4 grid in this particular system).

PARIX 1.3.1 (PARallel extensions to unIX) is the operating system actually being supplied with the PowerXplorer [3]. PARIX provides, among other services, message routing and multi-user partitioning of the 2D grid of processors. A partition is a set of processors that are exclusively allocated to one user. PARIX ensures that simultaneous users of different partitions will not conflict with each others.

Two synchronous point-to-point communication modes offered by PARIX were evaluated: **virtual links** which require that the communicating processes are connected via a pre-established virtual link, and **random communication** which does not require the definition of virtual links but, for messages longer than 1024 bytes, one virtual link is internally established due to performance reasons. PARIX does not offer direct support for collective operations.

PowerPVM 1.1 [4] is a dedicated implementation of PVM 3.2.6 on PARIX systems from Parsytec. The following communication operations were evaluated: **asynchronous point-to-point** (pvm_send /pvm_recv), **broadcast** and **barrier synchronization**. Broadcast is a collective communication; the broadcast message is not sent back to the sender, so $R_\infty = (p-1) * r_\infty$. The barrier is a collective computation; defining the message length makes no sense, its time to completion is given by the startup time. The time for buffers initialization and message packing was not considered for any of the PVM operations.

PowerMPI 1.1 [5] is a dedicated implementation of the MPI 1.0 standard for Parsytec systems running PARIX. The following communication operations were evaluated: **point-to-point standard mode** (MPI_Send/ MPI_Recv), **broadcast**, **gathering** and **scattering**, **all-to-all exchange** (MPI_Allgather), **barrier synchronization**, data **reduction** and **scan** (prefix-reduction). The last three operations are collective computations; only small message lengths are considered (4 bytes for single-precision floating point), consequently, t is given by the startup time $t_0(p)$.

4 Testing methodology

The programs were written in standard C and compiled with the C Solaris compiler for PowerPC from Motorola, version 1.5.1, with optimization level -O5. The PowerXplorer partitions were fully dedicated, minimizing interferences from the OS and avoiding conflicts with other users.

The experimental procedure followed several rules:

- only one user process was assigned to each processor in all experiments; all processes were engaged in all collective operations; the number of processes (nodes) used were 2, 4, 8, 12 and 16; the message sizes, where appliable, are powers of 2, from 2^2 to 2^{16};
- to gather data for each experiment the minimum measured time value was selected out of 5 runs, minimizing interference from the OS [1];
- for collective operations, each processor measured its local time to complete the communication; the maximum time across all processors is considered to be the time to complete the collective operation;
- each experiment repeats the communication operation 20 times to filter out any eventual interference from the OS; the average time is considered to be the one which most accurately reflects the typical time to completion;
- for asynchronous collective operations a barrier synchronization is performed between any two successive communications to ensure that no operation begins before the total completion of the previous one.

According to the PARKBENCH Committee recommendations the fundamental measurement in any benchmarking is the elapsed wall-clock time. The Committee proposes two benchmarks, TICK1 and TICK2, to measure the resolution and check the absolute value of the computer clock [6, 7].

On the PowerXplorer TICK1 showed that the clock ticks at least once between successive timer calls, suggesting that there is no need for a repeat loop on low-level benchmark measurements. The number of clock ticks between two successive timer calls is a time measurement overhead and must be subtracted from the values obtained on further measurements.

TICK2 confirms that the absolute values returned by the computer clock are correct, by comparing its measurement of a given time interval with that of an external wall-clock, like the benchmarker's wristwatch. The PARIX timer subroutine was verified to accurately measure wall-clock time.

Before any time measurements 5 iterations of the communication operation are performed to avoid measurement errors due to warmup overheads of the message-passing system. These overheads include, among others, code loading into memory and buffers and cache initialization.

The time between two consecutive calls to the timer subroutine is measured and later deduced from all time measurements.

The ping-pong method is used to measure the point-to-point communication time; this is similar to the COMMS1 benchmark proposed by the PARKBENCH Committee; this approach correctly handles both synchronous and asynchronous message-passing modes; in our experiments the distance between nodes varies between 1 and 6 hops;

The code is organized as follows:

```
get time1
get time2
time overhead = time2 - time1
local time = 0;
for (i=0 ; i<20 ; i++)
      barrier synchronization
      get start time
      perform collective operation
      get end time
      local time += end time - start time - time overhead
local time = local time / 20
communication time = maximum reduce (local time)
```

5 Results

The measured time values fill 13 tables (available on [9]), covering the several types of communication and operating environments. To summarize, and to extract useful information, the results are presented in tables 1 and 2 using a least-squares fitting technique. $t(n, p)$ can be obtained applying equation (3) to the expressions presented for $t_0(p)$ and $r_\infty(p)$. The results obtained with the curve fitting technique closely match the experimentally measured time values.

Table 1 shows the curve fitting results for point-to-point operations. p represents the number of processors in the path of the message, i. e., the number of hops plus 1.

	$t_0(\mu s)$	$r_\infty(MB/s)$
Virtual Links (PARIX)	$41 + 42p$	1.05
Random (PARIX) $(n \leq 1024)$	$153 + 141p$	$0.63 - 0.26\ln(p)$
Random (PARIX) $(n > 1024)$	$413 + 273p$	1.05
PVM	$169 + 146\ln(p)$	1.05
MPI	$231 + 126\ln(p)$	1.05

Table 1. Curve fitting results for point-to-point communication

PARIX uses two different mechanisms for random communication depending on the message length. If it is greater than 1024 bytes a virtual link is internally created between the two communicating processes. Thus two different expressions for $t(n,p)$ are obtained, depending on the message length.

Table 2 shows the results for collective operations.

		$t_0(\mu s)$	$r_\infty(MB/s)$
PVM	Broadcast	$256 + 101p^{1.22}$	$0.019 + \frac{2.31}{p^{1.38}}$
	Barrier	$118 + 144p + 3.19p^2 - 0.004p^3$	—
MPI	Broadcast	$35.7 + 451\ln(p)$	$0.2 + \frac{2.16}{p^{1.36}}$
	Gather	$118 + 135p$	$0.017 + \frac{2.08}{p^{1.34}}$
	Scatter	$269 + 137p$	$0.018 + \frac{2.2}{p^{1.36}}$
	AllGather	$-704 + 775p - 38p^2 + 1.94 * p^3$	$-0.05 + \frac{0.18}{\ln(p)}$
	Barrier	$-329 + 409p - 52p^2 + 2.52p^3$	—
	Reduction	$-110 + 277p - 20.4p^2 + 0.64p^3$	—
	Scan	$-902 + 712p - 56.6p^2 + 2.16p^3$	—

Table 2. Curve fitting results for collective operations

For MPI_Allgather the measured time values for larger values of n and p are several orders of magnitude larger than those for small values of n and p. The fitting method tends to be slightly dominated by these larger values, leading to poor approximations in the region of the $n*p$ space where n is large and p small.

MPI_Barrier presents a maximum value for $p = 12$ ($1478\mu s$) and then a smaller value for $p = 16$ ($1286\mu s$). This suggest that a different mechanism is used for $p > 12$. As this is an undocumented feature the results were approximated for $p \in [2, 12]$.

6 Analysis of results

Analyzing point-to-point communications, table 1 shows that PARIX virtual links offer better performance than random communications, independently of message size. For the former communication mode the obtained values are very similar to those presented by Parsytec [3]. The best achieved bandwidth was 1.05 MB/s and Parsytec claims 1.06 MB/s. For message lengths larger than 1024 bytes random communications present longer startup times than virtual links due to the added cost of establishing a virtual link for every communication. These longer startup times are reflected on $n_{\frac{1}{2}}$ (equation 2), which indicates that messages should be larger in order to achieve a satisfying bandwidth (4 times $n_{\frac{1}{2}}$ to achieve 80% of r_∞). It is interesting to note that the three environments, PARIX, PVM and MPI, have $r_\infty = 1.05 MB/s$, independently of the distance between the communicating processes. Also interesting is that MPI presents a somewhat higher latency than PVM, while it exhibits better performance on broadcasts and barriers.

Looking at collective communications, MPI's broadcast presents better performance than PVM's broadcast, due to a smoothly latency degradation with p (table 3 and figure 1); r_∞ has the same behavior for both environments; MPI_Scatter performs worst than MPI_Gather; this is not expected as both operations should present a similar communication pattern; notice from table 2 and figure 2 that the larger cost of scatter is due to latency; this difference in performance fades away as n increases due to r_∞.

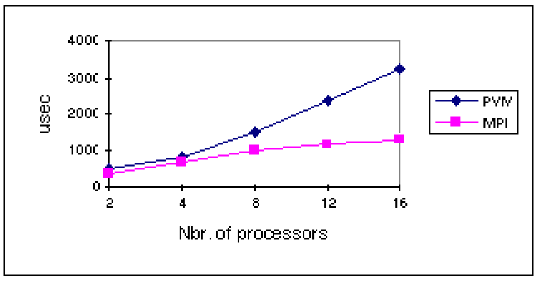

Fig. 1. Latency ($t_0(p)$) in μs for PVM and MPI broadcast

Analyzing collective computations, MPI's barrier performs better than PVM. As stated before the expression presented for MPI_Barrier's latency only holds for $p \in [2, 12]$. The reduction and scan operations can not be compared because PVM 3.2.6 does not support them (pvm_reduce is included in PVM 3.3).

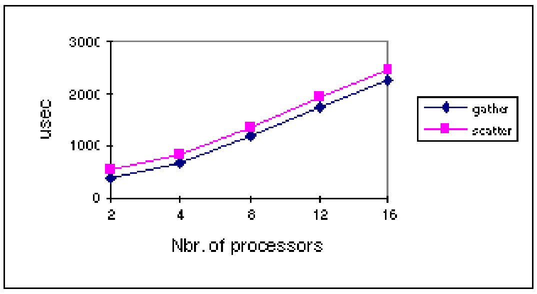

Fig. 2. Latency ($t_0(p)$) in μs for MPI gather and scatter

6.1 Identifying weaknesses

Operation	PARIX		PVM		MPI	
	t_0	r_∞	t_0	r_∞	t_0	r_∞
point-to-point	$\propto (p)$	$\propto (1)$	$\propto (\ln(p))$	$\propto (1)$	$\propto (\ln(p))$	$\propto (1)$
broadcast	—	—	$\propto (p^{1.22})$	$\propto (p^{-1.38})$	$\propto (\ln(p))$	$\propto (p^{-1.36})$
gather	—	—	—	—	$\propto (p)$	$\propto (p^{-1.34})$
scatter	—	—	—	—	$\propto (p)$	$\propto (p^{-1.36})$
allgather	—	—	—	—	$\propto (p^3)$	$\propto ((\ln(p))^{-1})$
barrier	—	—	$\propto (p^2)$	—	$\propto (p^3)$	—
reduction	—	—	—	—	$\propto (p)$	—
scan	—	—	—	—	$\propto (p^3)$	—

Table 3. Proportionality of $t(n, p)$ with respect to p

Table 3 summarizes all results, showing how $t(n, p)$ evolves with respect to p, $p \in [2, 16]$ (the symbol $\propto$ means 'proportional to').

Although PARIX virtual link point-to-point communications are the most efficient ones, one would expect their latency to be $\propto (\ln(p))$ as MPI and PVM. PVM's broadcast latency is $\propto (p^{1.22})$ which is quite innefficient when compared with MPI. MPI_Scatter and MPI_Gather latency are both $\propto (p)$. However, MPI_Scatter presents worst completion times due to added constants. In practice, $n_{\frac{1}{2}}$ is larger for MPI_Scatter. pvm_barrier is $\propto (p^2)$, while MPI_Barrier is $\propto (p^3)$, which suggests that the later's scalability could be improved. MPI reduction presents a latency with the same behavior as scatter and gather. This is an expected result, as the communication pattern is identical for all these operations.

Xu and Hwang [1] present results for MPI on a IBM SP2. The asymptotic bandwidth found is larger than for the PowerXplorer, since the SP2 interconnection network – High Performance Switch (HPS) – provides a peak bandwidth of

$40MB/s$ whereas a Transputer link has a peak bandwidth of $1.8MB/s$. However the degrees of proportionality they present for the asymptotic bandwidth of broadcast, scatter and gather are better for all communication operations than the ones found for the PowerXplorer, which may be due to two reasons: the MPI implementation used on the IBM SP2 is more efficient and/or the architecture of the HPS – a multi-stage Omega network with wormhole routing, allowing direct connections between any pair of nodes – is better suited for collective and point-to-point operations.

7 Concluding remarks

The results of this work help to understand the performance of some communication primitives on a 16 node parallel system, running 3 communication environments (PARIX, PVM and MPI). Quantifying communication costs is essential to support informed decision making when designing operating environments and parallel applications. The obtained results show that PowerMPI 1.1 communication primitives only achieve better performance than PowerPVM 1.1 on collective operations.

The current poor asymptotic bandwidth is essentially due to the peak bandwidth of the Transputer links. Current Parsytec MPP systems support a High-Speed link with a peak performance of 0.6 Gb/s.

Future work will include evaluation of a larger machine ($\gg p$), different MPP architectures and the dependence of computer performance on communication bottleneck (POLY3 benchmark from PARKBENCH).

References

1. Zhiwei Xu and Kai Hwang. Modeling communication overhead: MPI and MPL performance on the IBM SP2. *IEEE Parallel & Distributed Technology: Systems & Applications*, pages 9–23, 1996.
2. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT Press, 1995. ISBN 0-262-57104-8.
3. Parsytec. *PARIX 1.3.1.* Parsytec Eastern Europe GmbH, May 1995.
4. GENIAS Software GmbH. *PowerPVM for Parsytec Computers*, 1994.
5. GENIAS Software GmbH. *PowerMPI for Parsytec PowerPC Systems*, 1995.
6. R. Hockney and M. Berry. Public international benchmarks for parallel computers. Technical report, PARKBENCH Committee, February 1994.
7. R. Hockney. *The Science of Computer Benchmarking.* Society for Industrial and Applied Mathematics, 1996. ISBN 0-89871-363-3.
8. R. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Computing*, 20:389–398, 1994.
9. Luís Paulo Santos, Vítor Castro, and Alberto Proença. Evaluation of the communication performance on a parallel processing system - time tables. http://www.di.uminho.pt/ psantos/RESEARCH/PUBL/pvmpi97/tempos.html, May 1997.

This article was processed using the LaTeX macro package with LLNCS style