Parallel Programming Systems for LAN Distributed Computing

K.Zieliński M.Gajęcki G. Czajkowski
Institute of Computer Science, University of Mining & Metallurgy
Kraków, Al.Mickiewicza 30, 30-059, Poland

Abstract

The goal of this paper[1] is to give some ideas about run time efficiency of distributed computing environments. Six tools were investigated: PVM, P4, ANSA, SR, Strand, and Linda. They have been chosen because they represent different approaches to distributed programming systems construction. The experimental results of communication tests and processor farm model efficiency have been presented and discussed.

1 Introduction

The LAN distributed computing belongs to a very hot area of research. Results of many projects reached recently a mature state and a rich set of software systems and programming languages for distributed processing is available on the market and in the public domain sector. This paper is some kind of snapshot of a very rapidly changing domain.

The goal of this paper is to give some ideas about run time efficiency of distributed computing environments. Focusing only on the efficiency is much too narrow for a complete evaluation. However, it is a useful factor that can be applied objectively and performance is one of the major reasons for considering parallel computing in the first place [5].

Among more than 30 available software packages [7] six following tools representing different approaches to the distributed programming were investigated: PVM, P4, ANSA, SR, Strand, and Linda. The simplest solution such as **PVM**, offers only basic communication and synchronization functions. **P4** does not introduce any new ideas with comparison to PVM but its communication primitives are a little more mature and much more efficient. **ANSA** is a very large object oriented system that provides a whole infrastructure for an open distributed processing system development. The ANSA project seems to be a predecessor of the project CORBA or DOE, those carried on by SUN and HP [8]. **SR** is a distributed language and a run-time system designed while having a distributed processing in mind. It provides a very rich set of communication and synchronization primitives and an elegant computational model. **Strand** belongs to the family of the parallel logic programming languages. Because of simplifications and improvements it is far more efficient than many implementations of parallel logic programming languages known up to date. It was also a first available language that may be classified as a composition or a glue language. **Linda** is a simple communication library built around the tuple space concept. This leads to the declarative semantics of communication functions embedded into a pure imperative environment of C or Fortran.

The chosen software tools have been tested on different machines. This gives an idea about the hardware performance and the operating system communication layer efficiency and their influence on the features of a "network computer" created from the given type machines.

The scope of the paper is as follows. In Section 2, the scenario of evaluation experiments has been described. The hardware and software configurations have been specified in details. They have been divided into a communication test and a case study that concerns a processing farm implementation. In Section 3 the experimental results of communication test have been presented and discussed. The comparison of the investigated software in application to the processor farm implementation has been done in Section 4. The paper has been summarized in Section 5.Complete codes of the test programs are available electronically on the Internet address *galaxy.uci.agh.edu.pl* in the directory */pub/papers/cs.*

2 Scenario of evaluation experiments

The experiments described here cover two different communication scenarios. The two node tests measure conflict free communication between nodes. An

[1]Research supported by KBN grant no. 8 S503 015 06.

average time of various length data blocks transfer has been measured. These low level tests provide information about a raw performance , but are highly artificial and an extrapolation of these results to actual applications may lead to the misleading prediction. Hence, we measured the performance of the investigated software in a processor farm computational model. This very commonly used distributed processing paradigm has been chosen for the testing purpose for several reasons:

- It tests many-to-one and one-to-many communication patterns,
- An internal parallelism (asynchronism) of the provided communication operations, influences substantially the final performance. It is directly related to the overlapping of communication and computation.
- An implementation of the same, from functional point of view, piece of code in six different software environments made it possible to evaluate a few other issues, such as ease to use, debugging, compactness of programming and its elegance.

All of these tests have been performed on several types of workstations such as : SUN SLC, SPARCstation2 and IBM RS6000-320 connected through 10Mbps Ethernet LAN. This fact makes it possible to evaluate not only the programming systems under study but also the hardware and network software platforms they are implemented over. It addresses also a very important question about a balance between computational power and communication subsystem throughput, what is crucial for distributed computing efficiency and scalability.

The implementation of the processor farm computational model is very simple with the LAN based distributed computing. The nature of LAN communication systems ensure that no additional effort is necessary to provide direct communication of the one station with another. Hence no complicated routing algorithms have to be applied.

Two structures of the processor farm basic model shown in Fig. 1 have been studied. Model A is more efficient than B, if a number of workstations is not very big so the farmer process is not very busy and does not create the system bottleneck. The situation changes when a number of workstations increases and an additional worker task starts in the same node in which the farmer process is performed to disturb the farmer activity. The critical number of nodes to swap from Model A to B depends on task granularity, and

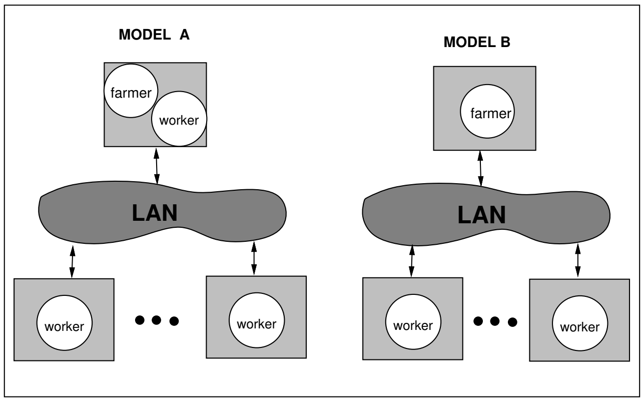

Figure 1: Two structures of a processor farm

workstation type and their communication to the computation power ratio.

The bottleneck in the LAN based processor farm has two main reasons. First, the farmer functions are centralized, so when a number of worker processes increases a workstation running the farmer process gets overloaded. Second, a communication medium is a common resource that may be used only sequentially. When a number of workstations increases, a communication contention seems to a problem.

The first problem may be solved using the hierarchical farm structure but this needs a load sharing strategy between the farmer processes to be implemented. The second problem may be partly overcome by a rebuilding of a communication subsystem using bridges. A communication cable should be divided into sections connected by bridges. A group of processes that create a subfarm should be allocated to the workstations connected to the same section of the cable. It makes it possible to perform communication in parallel in the separate subfarms.

The LAN communication system is rather slow with comparison to a shared memory or a bus data transfer and introduces a delay measured in milliseconds. To improve the communication efficiency, a double buffering mechanism of tasks allocation for the worker processor should be implemented. The implementation of the double buffering needs an asynchronous send and receive operation and a multithreading environment. Unfortunately, these features are only provided by a little more advanced software for the distributed processing such as ANSA, SR, Strand and Linda. For any software environment this mechanism of data buffering may be simulated via allocation more than one worker process to every workstation. In such a situation the probability that two processes finish a computation at the same moment of time, send result back to the farmer and, next, stay idle waiting for new data is rather low. This method has been used for PVM and P4.

There are many ways to measure the performance of a parallel algorithm running on a multiprocessor system [3, 5, 6]. The most commonly used indices are elapsed time, speed-up and efficiency.

Evaluation of the processor farm software covers the following types of experiments:

- The time of a sequential algorithm computation on various workstations has been measured.

- Influence of tasks granularity on the computation performance has been evaluated. In experiments only one worker process per workstation was allocated. Measurements were repeated for various numbers of workstations.

- For the optimal granularity the relation between a number of stations and the processing time was identified. Based on these experiments a speed-up and an efficiency were calculated.

As a case study, problems of two computational intensive tasks with geometric parallelism i.e. generation of a fractal picture and ray tracing were chosen. As a ray tracing software a parallelized version [6] of the DBW-Render package by Wiliam T.Baldridge has been used.

3 Communication performance study

The experiments reported in this paper concern the workstation types running under the operating system and the software package versions summarized in Table 1. The workstations were connected through 10Mbps Ethernet LAN. The tests were designed to measure the communication time between two nodes on a local area network. We wanted to eliminate any side effects due to network activity or competing processes running on the workstations. Therefore we were using isolated Ethernet.

The timings were obtained by measuring the round trip communication time in a so-called *ping/pong* program: a message was sent from one node to the other and then back again.

Since we were interested in elapsed time and not CPU time, every program accessed the system clock by calling the standard UNIX function *gettimeofday()* for timings

Workstation Type	Operating System Version
SUN SLC	SUN OS 4.1.3
SPARCstation2	SUN OS 4.1.1
RS 6000-320	AIX 3.1
Software Package	Version
PVM	2.4.0
P4	1.2
ANSA	4.0
SR	2.0
Linda	2.4.6
Strand 88	Buckingham Release June 1990

Table 1: Hardware and software under experiment

For Strand and Linda, that do not provide typical of message passing operations, their equivalents have been created. Each programming environment was tested by considering 5000 separately timed iterations for the round trip communication. The reported results represent average communication times.

The comparison of the investigated package efficiency has been given in Table 2 for the data length 16Byte, 1KByte, and 64KByte

The comparison of software tools leads to the following conclusions:

- P4 has the most efficient communication operations in the whole range of data block length under study.

- For small data blocks ANSA has also a very good efficiency. The ordering, according to decreasing efficiency, is as follows: P4, ANSA, Linda, SR, PVM, and Strand. It is illustrated more precisely in Fig.2 for the SPARCstation2 to SPARCstation2 communication.

- Strand efficiency is very bad for short messages.

- For large data blocks, Strand is doing even better than ANSA. The ordering according to decreasing efficiency is as follows: P4, Linda, Strand, SR, PVM ANSA. It is shown in Fig.3. Unfortunately, this order is dependent on hardware platforms.

The comparison of the results leads to the following conclusions:

- The communication efficiency between two SPARCstation2 was approximately 10% better than between the SUN SLC and SPARCstation2, and 20% better than between two SUN SLC.

- The most surprising results concern RS6000. It is the most powerful machine under tests. Unfortunately, the communication efficiency obtained

Software Package	16 Byte				1 KByte				64 KByte			
	S2	S2-SLC	SLC	RS	S2	S2-SLC	SLC	RS	S2	S2-SLC	SLC	RS
Strand	30.0	30.1	30.1		30.0	30.1	30.2		124.8	163.6	146.2	
PVM	3.6	5.6	6.6	8.9	5.5	8.2	9.5	10.7	120.2	173.8	203.9	190.8
SR	5.1	9.8	6.2		5.0	10.0	10.2		98.8	203.6	150.7	
Linda	2.6	3.4	5.1		4.4	5.4	6.5		99.1	114.9	135.4	
ANSA	1.6	2.9	3.1		3.3	4.8	6.1		213.1	224.3	236.5	
P4	1.2	2.0	2.4	3.7	2.2	3.1	3.3	4.3	71.9	100.8	107.9	122.2

Table 2: Summary of communication efficiency study (elapsed time in millisecond).

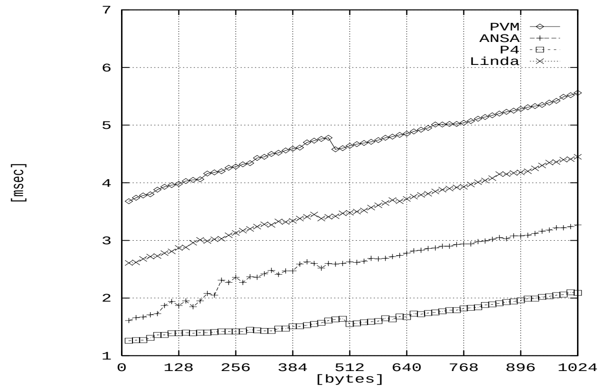

Figure 2: Data transfer time between two SPARCstation2 using different software tools vs. small data block length

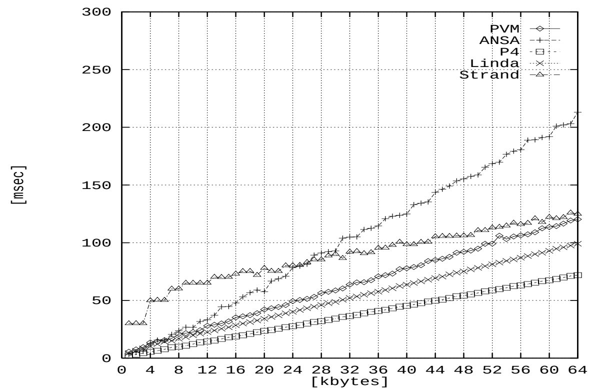

Figure 3: Data transfer time between two SPARCstation2 using different software tools vs. large data block length

for the PVM and P4 software was very disappointing. The system AIX and communication software seem to be very inefficient with the data transfer over Ethernet. It causes the communication bottleneck in the cluster of RS6000 machines built around Ethernet.

- The packages PVM, P4, ANSA, and Linda have almost a linear characteristic of the transfer time vs. data length.
- SR and Strand have some anomalies of the data transfer efficiency. The Strand data transfer time is approximately independent on the data block length up to 3 KByte and is equal to 30 milliseconds. For large data block the Strand efficiency increases and the investigated characteristic is almost linear. The SR has also anomalies of the data transfer efficiency in the small data length range. These anomalies are also observed for the large blocks for the mixed SUN SLC SPARCstation2 configuration. The reported results are an average of the results obtained during many repetitions of the experiment so the discussed anomalies are not a random effect.

4 Processor farm model study

Fractal computation results.

In the reported experiments the size of the scene was equal to 800 x 600 pixels and the maximum number of iterations per pixel was 256. There was no foreign traffic on the network. Each experiment was repeated five times. The time of the computation has been measured from the moment when the first task has been sent to the worker process until the last result has been collected. So the time of the given distributed environment set-up has not been measured.

To show that a fractal computational time is considerable and to compare a relative performance of the available processors this task was computed sequentially on various architectures. The obtained results have been summarized in Table 3. This information is especially important when the trade-off between communication and processor power is investigated. It is also necessary to point out that for the task under study modern RISC processors are approximately 10 times faster than the transputer T800.

Processor Type	Compiler	Time [sec]	Relative performance
T800/20	3LC v2.1	225.92	1.00
SUN SLC	UNIX C	174.00	1.30
Intel 486/25	UNIX C	112.50	2.01
SUN IPC	UNIX C	95.00	2.38
SPARC 2	UNIX C	44.00	5.13
I860/40	GNU	21.70	10.41
RS-6000-320	XLC	20.80	10.86
HP 9000/720	UNIX C	17.21	13.13

Table 3: Fractal computation sequential time on different processors

The influence of task granularity on the computational time has been shown in Fig.4. This experiment has been repeated for various software tools and hardware platforms. The obtained results lead to the following conclusions:

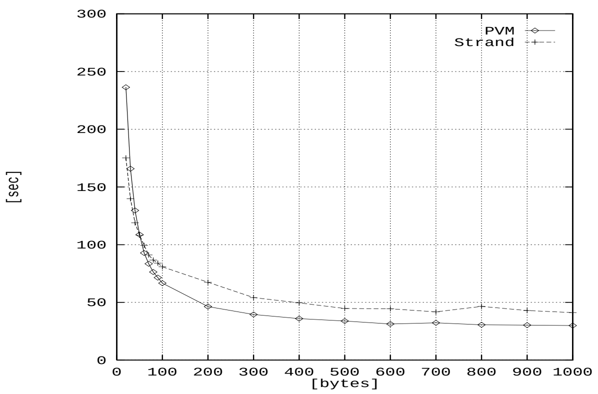

Figure 4: The influence of the task granularity on fractal computation time on 8 SLC using Strand and PVM respectively

- The small task granularity degrades the performance substantially.
- The optimal granularity for the investigated fractal lays in the range 800 to 2000 pixels per task and is independent from hardware and software platform. For the further study the 800 pixels per task granularity has been chosen.

The communication in Strand, as it was shown in the previous section, is very inefficient for small messages. PVM is doing a lot better in this range. The Strand processor farm is internally written in a much more parallel way than the PVM farm and automatically exploits such a mechanism as double buffering. The results are very surprising. Strand is doing better than PVM for small task granularity. This is also a very good example how sophisticated problem may be used for the comparison study of different software tools.

The comparison of these two software tools in terms of speed-up is shown in Fig.5. These computations were performed for the task granularity 800 pixel per task. The double buffering scheme may be very easy

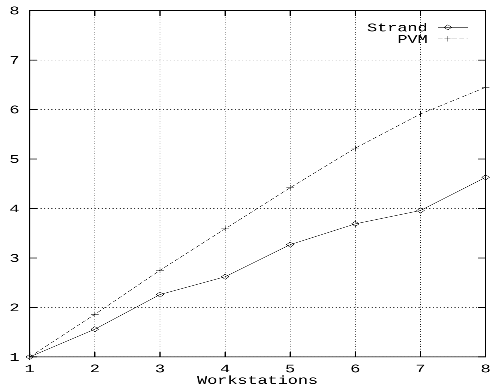

Figure 5: Speedup of fractal computation using PVM and Strand on SLC workstations

implemented using SR. The influence of this mechanism on the system performance is shown in Fig.6. This experiment has been performed for the SPARCstation2 as a farmer node and SLC as workers.

The double buffering improves efficiency for a greater number of nodes. It is justified by the fact that the double buffering helps to overcome a communication bottleneck.

So far the model A of the processor farm was investigated only. From the communication performance study a very low efficiency of the RS6000 system is evident . How it influences the speedup and efficiency of the fractal computation is shown in Table 4 for the PVM platform. There are some possibilities to improve a processor farm computation when the model B of farm instead A is used for a greater number of processors. In this way with five RS6000 it is possible to improve a performance. The same is true for the SPARCstation2 configuration but with a greater number of processors. Unfortunately, only five SPARCstation2 were available for the study hence this effect has

SPARC - 2				
No. of Processors	No. of Workers	Time [sec]	Speedup	Efficiency
1	1	43.95	1.00	1.00
2	2	26.47	1.66	0.83
3	3	20.10	2.19	0.73
4	4	15.23	2.89	0.72
5	5	12.90	3.41	0.68
5	4	14.18	3.10	0.62
RS6000				
No. of Processors	No. of Workers	Time [sec]	Speedup	Efficiency
1	1	20.81	1.00	1.00
2	2	18.91	1.10	0.55
3	3	14.77	1.41	0.47
4	4	13.02	1.60	0.40
5	5	12.10	1.72	0.34
5	4	10.77	1.93	0.39

Table 4: Fractal computation comparision on SUN SPARCstation 2 and RS6000 platforms using PVM

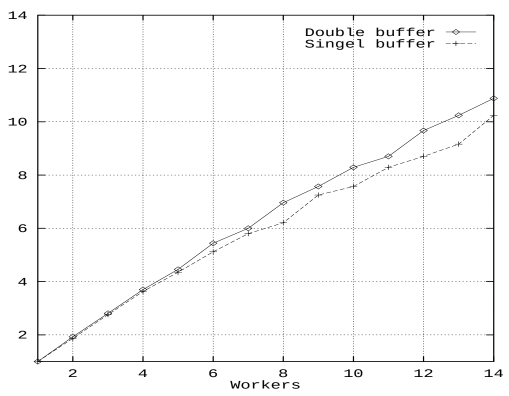

Figure 6: Influence of double buffering on SR fractal computation speedup

been shown in Fig.7 for SLC stations. The crossing of the curves A and B is also coursed in that case by the fact that the SLC processor is rather slow.

The SPARCstation2 processor is two times slower than RS6000 with the fractal sequential computation but with 5 nodes the computational time is almost equal for both architectures. It leads to a general conclusion that Ethernet is far too slow for the RS6000. The fractal distributed computations in such a system are very inefficient.

The results of the influence of the model choice on various software running on SLC workstations are depicted in Fig.7. The same experiments have been per-

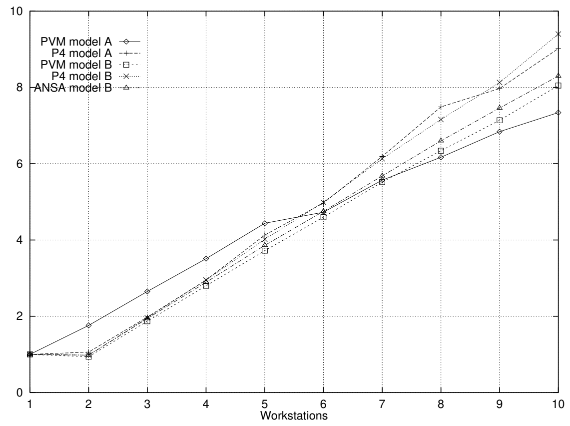

Figure 7: Fractal computation speedup over different software platforms using models A and B on SLC workstations

formed on SUN SPARCstation2 workstations. The presented results lead to the following conclusions:

- The obtained efficiency depends on a number of nodes and it is difficult to give one answer which software platform is the best one,
- Model A is very inefficient with a small number of nodes for the P4 platform. The local communication, i.e. communication between the farmer and the worker on the same node is time consuming and degrades the whole performance. Hence, P4 is doing worse than ANSA and PVM for small number of nodes. This effect is even more evident on the SUN SPARCstation2 than on SLC.
- P4 fast communication ability gives to it some advantages with greater numbers of nodes using the model A. So on 10 SLC, P4 is doing better than another software under study.
- The model B has not shown such anomalies as the model A. P4 is the most efficient platform in the whole range under study. A little worse is ANSA.
- The differences between the software tools under study are greater on SLC than on the SUN SPARCstation2.
- A very characteristic is the drop of the efficiency for a small number of the workstation using the model B. It is justified by the fact that the farmer node is not involved in task computations.

Ray tracing computation results

The ray tracing computations are much more demanding in a comparison to the fractal. The relative performance of various workstations under study has been shown in Table 5. It is interesting that very fast machines such as HP9000/720 and RS-6000 observed a very substantial degradation (50%) of relative performance in a comparison with that obtained for the fractal (See Table 3). The slow workstations, such as SLC and Intel 486 obtain a similar relative performance as for the fractal. The general conclusion is that for the ray tracing computation the gap between communication efficiency and computation power is narrower.

Processor Type	Compiler	Time [sec]	Relative performance
T800/20	3LC v2.1	1788	1.00
SUN SLC	UNIX C	1372	1.30
SUN IPC	UNIX C	958	1.87
Intel 486/25	UNIX C	914	1.96
Intel 486/33/256	UNIX C	646	2.77
SPARC 2	UNIX C	388	4.61
CONVEX C220	UNIX C	341	5.24
RS-6000-320	XLC	323	5.54
I860/40	GNU	298	6.00
HP 9000/720	UNIX C	212	8.43

Table 5: Ray tracing computation sequential time on different processors

This conclusion is confirmed by the results presented in Fig.8. These figures concern the PVM implementation of the model A processor farm. For the ray tracing computations five RS-6000 are doing better than seven SPARCstation2. This result is opposite to that reported in Table 4. In general, it was possible to speedup the ray tracing computation by 51 times via distributed computing in the heterogeneous environment built of the seven SPARCstation2, five RS-6000 and one HP9000/720 . It is a very impressive example of the efficiency of the LAN distributed computing.

In general the obtained results for the ray tracing computation are very similar to those obtained for the fractal. The only difference comes from the different computational complexity.

The comparison of the ray tracing and the fractal computations in term of the obtained speedup is depicted in Fig.9.

In general, the LAN distributed computations are more efficient when a computing is a dominant factor over communication.

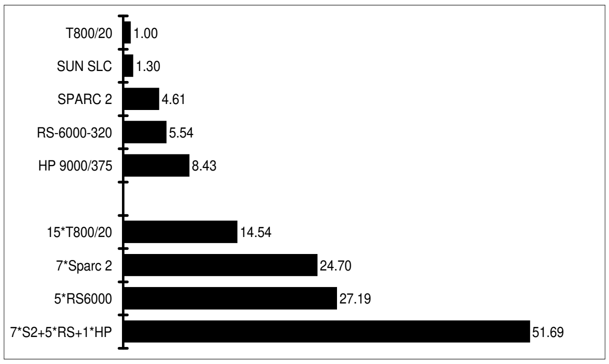

Figure 8: The maximum performance of ray tracing computation over PVM platform on various architectures

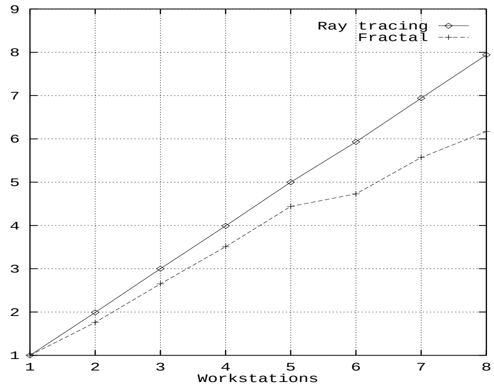

Figure 9: Speedup of fractal and ray tracing computations over PVM on SLC workstations vs. number of nodes

5 Conclusions

The results of the presented experimental study and the analysis of the investigated software packages lead to the following general conclusions :

- The packages for the LAN distributed computing provide a different degree of an internal parallelism. Hence, for the same algorithms a potential parallelism can not be fully exploited. The more advanced software tools like Strand, ANSA, and SR provide mechanisms that are better suited to parallel computation demands.
- There exists no relation between the communication efficiency and the complexity of the software package. A very complex package such as

ANSA provides very efficient communication operations. Unfortunately, the communication efficiency of some tools depends on the data length. The best examples of such behaviour are Strand and SR which are better suited for the long data blocks transfer.

- The internal parallelism of the computational model is a factor that could outperform the pure communication efficiency. Strand is the best example of that . Hence, the comparison of communication efficiency only is not a good measure of the whole package evaluation.

- The speedup of LAN distributed computations is very difficult to predict. It is, of course, a much more difficult problem than for sequential computations. There has to be reached the same kind of equilibrium between a computational power and a communication efficiency. The point of this equilibrium moves because the computational power represented by nodes of the system changes for various computational problems.

- A very important factor of the LAN distributed computation is a task decomposition. In the case of computational problems with geometrical parallelism this corresponds to the task granularity choice. A wrong decision in this choice may degrade the whole system performance. The problems under study performed much better for the small than for the large granularity. Unfortunately, the meaning of a small or large granularity changes with a problem. From the presented results it is evident that the number of tasks a problem is divided into, should be at last ten times greater than the number of the LAN computer nodes.

- The complexity of the distributed program depends substantially on the computational model. Linda and Strand provide the most elegant and compact solution of the processor farm implementation . PVM and P4 are effective tools but need more effort to organize the distributed computations. The programming with these last two mentioned packages presents more complex communication connections between processes and an error prone task. The longer time of software testing and debugging time, however, could not be justified by the final gains from a slightly better performance.

- A great attention should be paid to the characteristic features of various packages such as, for instance poor efficiency of communication of Strand using small data blocks and very inefficient local communication in P4,

- In regard to the hardware platform it is evident that a degradation of efficiency is observed for a small number of more powerful workstations such as RS-6000 and SPARCstation2 than for slower machines such as SLC. This behaviour is caused by the Ethernet communication overhead. It leads to the hypothesis that the maximum computational power that could be obtained from the LAN computer is independent of the types of the system that is built of and converge to the same value. It ought to be proved that many more experiments with greater systems than the ones that were available for the reported study should be done.

References

[1] G.R. Andrews and R.A. Olsson. *The SR Programming Language: Concurrency in practice.* Benjamin/Cummings Publishing Company, 1992.

[2] A. Beguelin et. al., A Users' Guide to PVM Parallel Virtual Machine, TR, Oak Ridge Natinal Labolatory, 1991.

[3] R. Butler, E Lusk, User's Guide to the p4 Programming System, Argone National Labolatory, 1992, TR ANL-92/17.

[4] C.H. Cap, V. Strumpen, High Performance Platform for Parallel Computing in a Distributed Workstation Environment, Institut für Informatik, Univ. Zurich, 1992, TR IFI-92-07.

[5] G.C. Douglas, T. G. Mattson, M.H. Schultz, Parallel Programming Systems for Workstation Clusters, Yale Univ. Dept. of Comp. SC. RR YALEU/DCS/TR-975, August, 1993.

[6] M. Gajęcki, K. Jędrzejek, K.Zieliński, *Parallelization of the DBW Render Package on a Transputer Farm* Machine GRAPHICS & VISION, Vol.1, No.4, 1993.

[7] Strand Software Technologies Limited: STRAND'88 User Manual, June 1990.

[8] Solaris Strategy, SunSoft TR 6/21/12