A Comparison of the Iserver-Occam, Parix, Express, and PVM Programming Environments on a Parsytec GCel

P.M.A. Sloot, A.G. Hoekstra, and L.O. Hertzberger

Parallel Scientific Computing & Simulation Group, Department of Computer Systems, Faculty of Mathematics and Computer Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, the Netherlands, tel.:3120-5257463, fax.: 3120-5257490, email: peterslo@fwi.uva.nl

Published in

High Performance Computing and Networking, eds W. Gentzsch and U. Harms, Lecture Notes in Computer Science 797, Springer Verlag, pages 253 - 259, 1994.

Abstract

We compare the Iserver-Occam, Parix, PVM, and Express parallel programming environments on a Parsytec GCel with 512 T805 transputers. The comparison will be made by a detailed analysis of the performance of one particular application. In our approach we start with this application and isolate the basic (environment dependent) building blocks. These basic building blocks, which depend on floating point performance and communication capabilities of the environments, are analysed independently. We have measured point to point communication times, global communication times and floating point performance. All information will be combined into a time complexity analysis, allowing us to compare the environments on all relevant degrees of functionality. Together with demands for portability of the code, and development time (i.e. programmability), an overall judgement of the environments can be made.

1 Introduction

Real success of Massively Parallel Processing critically depends on programmability of the parallel computers and on portability of parallel programs. We are made to believe that “parallel computing has come to age”. Although it is safe to say that parallel hardware has reached a convincing stage of maturity, both programmability of the parallel hardware and portability of parallel programs still pose serious problems to developers of parallel applications.

Today, an application programmer is usually faced with a situation as drawn in figure 1. A parallel computing platform supports native environments, which allow very low level programming, or allow a more abstract view of the hardware. Furthermore, generic environments, also available on other platforms, can be used. These environments can be grouped in order of decreasing hardware visibility and increasing portability. Of course, one expects that the price to be paid for portability is a decrease of control of the hardware and associated degradation of performance.

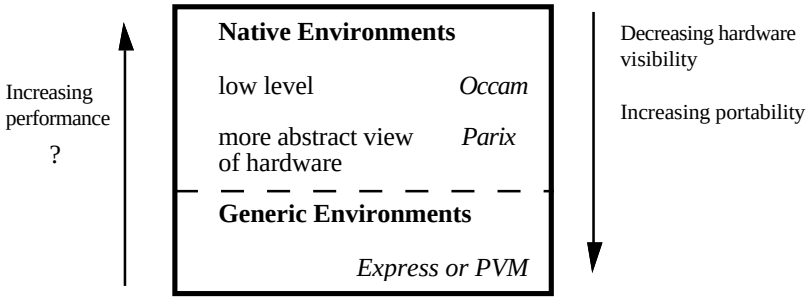

Fig. 1. a typical situation encountered by application programmers of parallel systems.

In this paper we will compare two native environments, Iserver-Occam [1] and Parix [2], with two generic environments, Express [3] and PVM [4]. All experiments are executed on the Parsytec GCel. As part of a contract with Genias, Sloot et al. ported PVM to the GCel [5].

Our experiments allows us to judge the trade-off which clearly exists between native environments, usually offering a better performance at the price of extensive programming effort, and generic environments which allow to develop more portable programs. In this paper we will briefly describe our methodology and summarise the most important results. Reference [6], chapter 6 contains further details.

As a representative case study we have implemented, in the four environments, an application from Physics: Elastic Light Scattering simulations using the Coupled Dipole method [6, 7, 8]. This application is a real application (in the sense that it is used for actual production), the time complexity of the program is predictable, it contains global communication routines, and it does not exhibit (severe) load imbalance.

The run time of the parallel Coupled Dipole implementation is determined by the floating point performance τ_{calc}, i.e. the time to perform a floating point operation, and global communication times (specifically a vector gather operation). In the Iserver-Occam and Parix implementation the global routines are explicitly implemented using (nearest neighbour) point to point communications. In Express and PVM we rely on the available global communication routines. In order to compare the environments we have measured floating point performance, basic communication routines, global communication routines and finally the execution time of the parallel Coupled Dipole method.

2 Results

Measurement of the floating point performance of the Coupled Dipole simulation results in: for Iserver-Occam τ_{calc} = 2.63 μs/flop; for Parix τ_{calc} = 1.28 μs/flop; for Express τ_{calc} = 1.72 μs/flop; and for PVM τ_{calc} = 1.3 μs/flop. Analysis of the communication performance reveals that:

1] Both the point to point - and the global communication routines can be fitted to a straight line, $T_{comm} = \tau_{setup} + n\,\tau_{send}$, with n the number of transferred bytes;

2] The Iserver-Occam environment is the most efficient for communication, followed by Parix, Express and PVM. The send time for the point-to-point communication is ±0.85 μs/byte, ±.90 μs/byte, 0.90 - 1.54 μs/byte, and 0.8 - 1.8 μs/byte for Iserver-OCCAM, Parix, Express and PVM respectively. The setup times are (in the same order), ±3.8 μs, 67 μs, 50 - 300 μs, and 2000 - 3000 μs.

3] On large processor domains both Express and PVM have remarkably high setup times for the global communication routines, in the order of seconds;

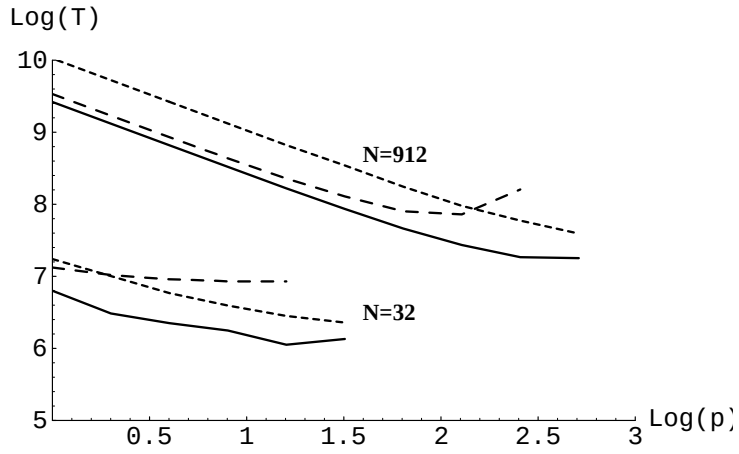

Fig. 2. The execution time of a total Coupled Dipole simulation (including I/O), as a function of the number of processor, for N = 32 and N = 912; the solid line is for Parix, the short-dashed line for Iserver-Occam, and the long-dashed line for Express.

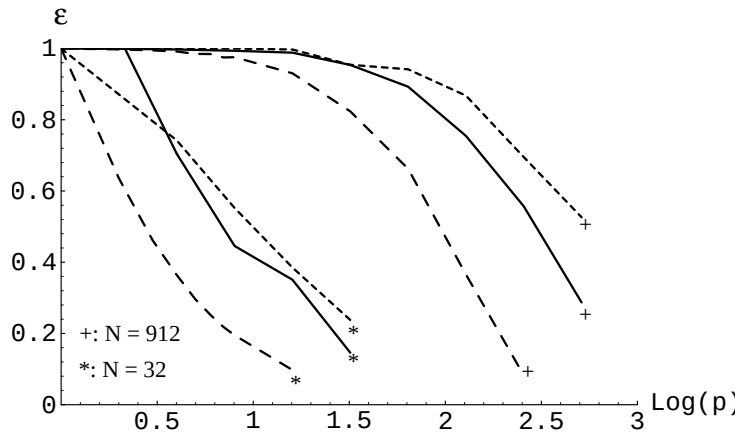

Fig. 3. The efficiency of a total Coupled Dipole simulation (including I/O), as a function of the number of processor, for N = 32 and N = 912; the solid line is for Parix, the short-dashed line for Iserver-Occam, and the long-dashed line for Express.

Finally we measured the execution time of the parallel Coupled Dipole simulation as a function of the number of processors, for a set of relatively small simulations containing 32 up to 2200 dipoles. Unfortunately, in this case PVM results are not yet available. We are currently performing these experiments, and the results will be published elsewhere. From the execution times the parallel efficiencies are calculated. Typical results for the execution time and efficiencies are shown in figures 2 and 3 respectively.

Knowledge of the floating point performance and the global communication capabilities of the environments allows us to interpret and understand the execution time and efficiency of the application. Next we will investigate if we can *predict* the execution time of the Coupled Dipole application, using the results of the basic performance measurements. We will only present results for the execution time of one Conjugate Gradient iteration, which is the computational kernel of the Coupled Dipole simulation. Typical results are shown in figures 4 and 5.

In all cases we can accurately predict the execution time of the Parix, and Iserver-Occam implementation. For small partitions we can also accurately predict the execution time of the Express implementation. However for larger partitions errors between theory and experiment as large as 30 % are observed. These errors can be traced back to uncertainties in the fit parameters of the global communication routine in Express.

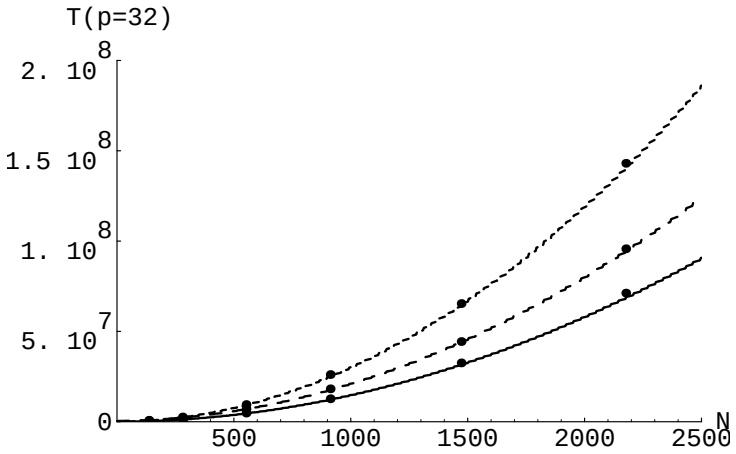

Fig. 4. The execution time (in μs) on 32 processors, for one Conjugate Gradient iteration, as a function of the number of dipoles N ; the dots are the measurements, the lines are the theoretical predictions; solid line for Parix, short-dashed line for Iserver-Occam, and long-dashed line for Express.

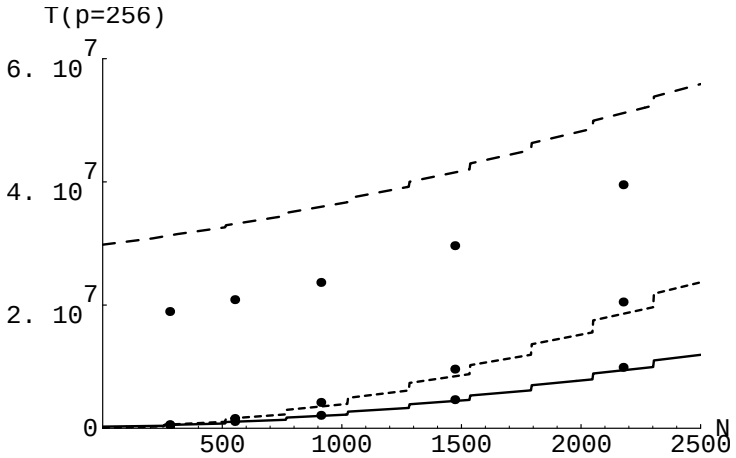

Fig. 5. Same as figure 4, but now for 256 processors.

4 Discussion and Conclusions

We can predict the execution time of our specific application in the different environments using basic performance measurements in these environments. Therefore, these basic performance measurements, combined with a time complexity

model and information on programmability and portability, allows us to compare the environments. Table 1 compares the four environments on a scale from good (++) to poor (--). The environments are judged on programmability (Prog.), on source level portability between hardware platforms (Port.), on the availability of tools, such as a performance analyser, debuggers etc. These first three characteristics are independent of the particular application. The rest of the columns are devoted to application dependent characteristics. These characteristics are the floating point performance (F.P.), the communication performance (Comm.), and the scalability, or efficiency, of the resulting parallel program (Scal.). Depending on the importance of e.g. portability versus scalability or floating point performance an overall judgement of the environments can be made.

	Prog.	Port.	Tools	F.P.	Comm.	Scal.
Iserver	- -	- -	- -	-	++	++
Parix	+-	-	-	++	+	+
PVM	- +	++	-	++	- -	-
Express	+	++	+	+	- -	-

Table 1. A comparison of the Iserver-Occam, the Parix, and the Express environment on programmability (Prog.), on source level portability (Port.), on the availability of tools (Tools), on floating point performance (F.P.), on communication performance (Comm.), and on scalability, or efficiency, of the resulting parallel program (Scal.).

In conclusion, we have compared the Iserver-Occam, Parix, Express, and PVM parallel programming environments on a Parsytec GCel, by a detailed analysis of the performance of a particular application. Our approach allows us to compare the environments on all relevant degrees of functionality. Together with demands for portability of the code, and development time (i.e. programmability), an overall judgement of the environments can be made.

5 Acknowledgements

We wish to thank J.J.J. Vesseur, F. van der Linden, M. van Muiswinkel, and P.A. Trenning of the faculty of Mathematics and Computer Science of the University of Amsterdam for their skilful porting of the original code, and subsequent performance measurements.

6 References

1] Inmos Ltd, *OCCAM® 2 Reference Manual* (Prentice Hall, 1988).

2] Parsytec, "The PARIX programming environment," In *Transputer Systems - ongoing research* (IOS Press, A. Allen Ed., pp. 218-230, Amsterdam, Oxford, Washington, Tokyo, 1992).

3] J. Flower and A. Kolawa, "A packet history of message passing systems," Physics Reports **207**, 291-304 (1991).

4] J. Dongarra, G.A. Geist, R. Manchek, and V.S. Sundaram, "Integrated PVM Framework Supports Heterogeneous Network Computing," Computers in Physics **7**, 166-175 (1993).

5] P.M.A. Sloot, for more information, please send email to peterslo@fwi.uva.nl.

6] A.G. Hoekstra, "Computer Simulation of Elastic Light Scattering: Implementation and Applications", Ph.D. thesis, University of Amsterdam, the Netherlands, 1994.

7] A.G. Hoekstra and P.M.A. Sloot, "New Computational Techniques to Simulate Light Scattering from Arbitrary Particles," in *Proceedings of the 3rd International Congress on Optical Particle Sizing '93 - Yokohama*, Ed. M. Maeda, 1993, pp. 167-172.

8] A.G. Hoekstra and P.M.A. Sloot, "Simulating Elastic Light Scattering Using High Performance Computing Techniques," in *European Simulation Symposium 1993*, Ed. A. Verbraeck and E.J.H. Kerckhoffs, Society for Computer Simulation International, 1993, pp. 462-470.