Comparing the Parix and PVM parallel programming environments

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

Parallel Scientific Computing & Simulation Group, Computer Systems Department, Faculty of Mathematics and Computer Science, University of Amsterdam, Kruislaan 403, 1098 SJ, Amsterdam, the Netherlands, telephone: +31 20 5257463, email: alfons@fwi.uva.nl, http://www.fwi.uva.nl/fwi/research/vg4/pwrs/

Summary

Genericity of parallel programming environments, enabling development of portable parallel programs, is expected to result in performance penalties. Furthermore, programmability and tool support of programming environments are important issues if a choice between programming environments has to be made. We propose a methodology to compare native and generic parallel programming environments, taking into account such competing issues as portability and performance. As a case study, this paper compares the Parix and PVM parallel programming environments on a 512 node Parsytec GCel. Furthermore, we apply our methodology to compare Parix and PVM on a new architecture, a 32 node Parsytec PowerXplorer, which is based on the PowerPC chip. In our approach we start with a representative 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 is combined into a time complexity analysis, allowing the comparison of the environments on different degrees of functionality.

1. Introduction

Real success of Massively Parallel Processing critically depends on programmability of 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.

In this paper we will address the question how to compare (native and generic) parallel programming environments, taking into account issues such as performance, portability, and availability of tools. We propose a methodology and apply it to a case study of native and generic environments on a transputer platform and on a PowerPC platform.

Many research groups have compared parallel programming environments [e.g. 1, 2, 3, 4, 5]. The majority of such comparisons however

concentrate around clusters of workstations, and have not analysed the behaviour of programming environments such as PVM on true massively parallel machines.

The goal of this work is to propose a strategy to compare parallel programming environments, and to apply this to native and generic programming environments running on a large massively parallel system. We will compare a native parallel programming environment, i.e. Parix, with a generic environment, i.e. PVM, by examining the behaviour of a representative parallel application implemented in these environments. These experiments are executed on a 512 node Parsytec GCel. As a case study we have implemented an application from Physics, i.e. Elastic Light Scattering simulations using the Coupled Dipole method [6, 7, 8] on the Parsytec GCel.

We will analyse the behaviour of the parallel Coupled Dipole method in both environments by analysing basic and global communication routines, floating point performance, and actual execution times of the parallel program as a function of problem size and number of processors. We will investigate if the basic measurements can predict the runtime of the application, and if such basic measurements can be used as a heuristic to assess the merits of a programming environment. In this way we can judge the trade-off which 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.

Finally, we will apply our methodology to assess the merits of Parix and PVM on a very recent PowerPC based architecture, the Parsytec PowerXplorer.

2. Results

2.1 Methodology

In order to compare the environments we have measured floating point performance, basic communication routines, a global communication routine (a vectorgather operation) and finally the execution time of the parallel Coupled Dipole method on the Parsytec GCel. Subsequently we have compared PVM and Parix on the PowerXplorer system by measuring floating point performance and relevant communication routines. Here we will briefly present the results for the communication measurements, the complete set of experiments will be provided in the full paper.

2.2 Result for the Parsytec GCel

We assume that point to point communication can be described by a linear two parameter model. The point to point communication time t_{pp} is

$$t_{pp} = \tau_{setup} + n\,\tau_{send}, \qquad [1]$$

with n the number of bytes sent, τ_{setup} a setup time to initialise and start the communication, and τ_{send} the send time to transfer 1 byte. Here we have neglected effects due to buffering.

Parix's virtual links allow point to point communication between any node, the kernel of Parix routes the messages through the hardware. However, in the Coupled Dipole implementation the only point to point communication consists of synchronous send/receive pairs between adjacent processors. Therefore we have only measured synchronous point to point communication between neighbouring processors. The results are $\tau_{send} = 0.92 \pm 0.02$ μs/byte and $\tau_{setup} = 67 \pm 2$ μs.

The analysis of point to point communication in PVM is complicated by the fact that the localisation of the parallel processes is not known, and therefore we do not know if a communication is between physical neighbouring processors. In order to get a picture of the basic communication performance of PVM we measured all possible point to point communications between node zero and all other nodes in a PVM partition, and analysed the distribution in the resulting setup - and send times. We have fitted all experiments and generated histograms of τ_{send} and τ_{setup}. The histograms are drawn in figures 1 and 2.

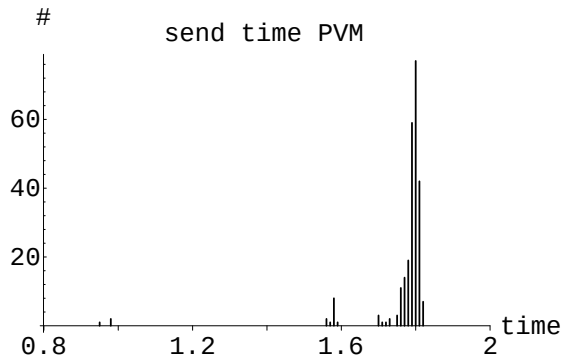

Figure 1: The histogram of the occurrences of send times (in μs/byte) in a 256 node partition in the GCel; the step size is 0.01 μs/byte.

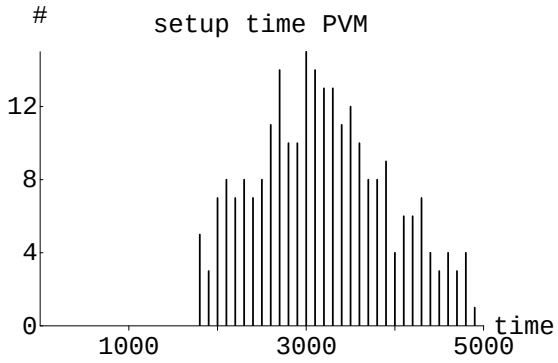

Figure 2: The histogram of the occurrences of setup times (in μs) in a 256 node partition in the GCel; the step size is 100 μs.

We observe two fast connections having a τ_{send} of approximately 0.9 μs/byte and 1.0 μs/byte. These numbers are comparable to the τ_{send} of Parix. The rest of the connections of PVM cluster around 1.6 and 1.8 μs/byte. PVM shows a broad distribution of τ_{setup} around approximately 3000 μs. Here we clearly observe a difference with Parix. PVM has much higher setup times for point to point communication.

We have measured the time for the vector gather operation as a function of the number of processors and as a function of the total vector length n in bytes. For each value of p we measured communication time as a function of n. We have fitted the measurements to $t_{vg}=\tau_a n+\tau_b$.

We observe two fundamental differences between PVM and Parix. First, in PVM the τ_a depends on p, and increases sub linear in p. Furthermore, the initialisation τ_b of PVM increases faster than linear with p, which results for large p in (unacceptably) high initialisation times. This result is a very good example of the trade-off of programmability against performance. The PVM vector gather operation consist of just one call to a global communication routine, and a trivial buffer compacting. However, the price to be paid is a bad

scaling behaviour, as compared to Parix.

2.3 Comparison of Parix and PVM on the Parsytec PowerXplorer

Figures 3 and 4 shows the histograms for τ_{send} and τ_{setup} (as introduced in section 4.3), in the 32 node partion of the PowerXplorer system for Parix and PowerPVM.

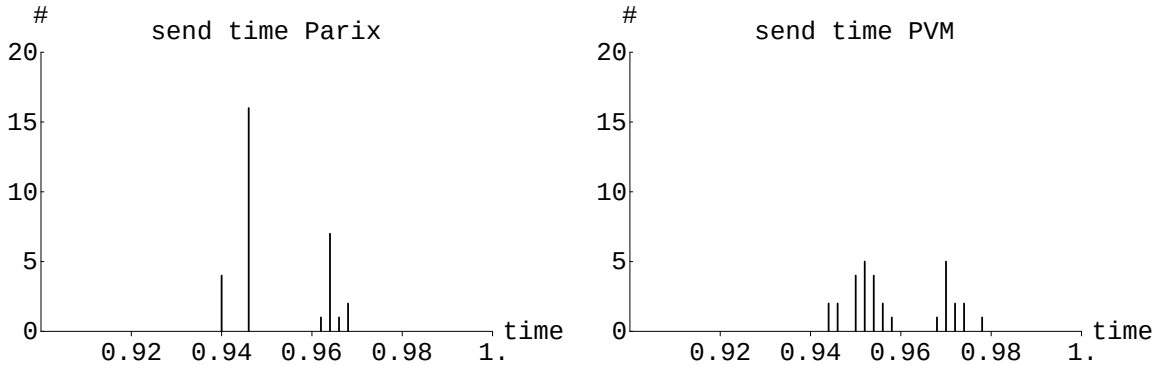

Figure 3: The histogram of the occurrences of send times (in μs/byte) in the 32 node partition in the PowerXplorer; the step size is 0.01 μs/byte.

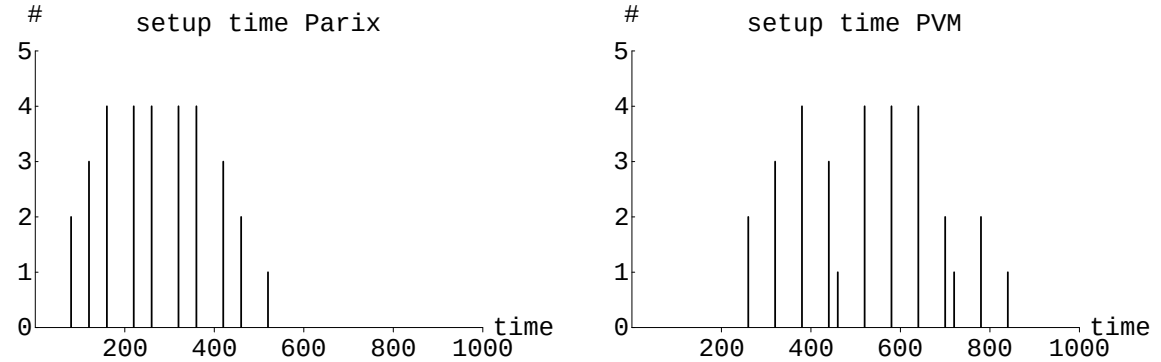

Figure 4: The histogram of the occurrences of setup times (in μs) in the 32 node partition in the PowerXplorer; the step size is 10 μs.

3. Concluding Remarks

We have compared the Parix and PVM parallel programming environments on a Parsytec GCel, by a detailed analysis of the performance of a particular application. Our approach, in which we start with an application, isolate the basic (environment) dependent building blocks which are analysed independently, and combining all information in a time complexity analysis, allowed 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. Our approach has similarities with the much more elaborate Parkbench benchmark suite [9]. However, by placing the basic measurements in the context of a specific application, as we do, the interpretation of the measurements can be more accurate.

In general we observe that increasing portability and programmability, in

going from Parix to PVM, results in a degradation of especially the communication capabilities. The global communication routine of PVM that we tested has a very bad scaling behaviour which clearly shows up in the larger partitions. This results in poor scalability of the PVM implementation. Fortunately, in real production situations, with large problem sizes, the application has an efficiency very close to one, and the run time is mainly determined by the floating point performance.

Application of our heuristic to compare PVM with Parix on a Parsytec PowerXplorer shows that point to point communication in PVM is as good as Parix. However, the global communication in PVM is still far from optimal. Future implementations of PVM should strive to more optimized global routines.

Global communication routines strongly increase the programmability of parallel computers. However, in our case studies, we have observed a poor scaling behaviour of these routines, as compared to handcoded Parix routines that exploit data locality and point to point communication on virtual topologies.

Currently the Message Passing Interface (MPI) standard has been defined, and the first implementations of MPI have been reported [10]. Our group is working on a MPI implementation on top of Parix [11]. We will test the MPI implementation using the methods as described in this paper. Furthermore, the global communication routines of MPI, such as the *MPI_BCAST*, will be implemented such that the reported scaling behaviour of comparable Express and PVM routines can be improved.

4. References

1. M. Weber, 'Workstation clusters: one way to parallel computing', *International J. Mod. Phys. C* **4**, 1307-1314 (1993).
2. A.R. Larrabee, 'The P4 Parallel Programming System, the Linda Environment, and Some Experiences with Parallel Computing', *Scientific Programming* **2**, 23-35 (1993).
3. A. Matrone, P. Schiano, and V. Puotti, 'LINDA and PVM: A comparison between two environments for parallel programming', *Parallel Computing* **19**, 949-957 (1993).
4. T.G. Mattson, C.C. Douglas, and M.H. Schultz, 'A comparison of CPS, Linda, P4, Posbyl, PVM, and TCGMSG: two node communication times', Tech. Rept. YALEU/DCS/TR-975 Dept. of Computer Science, Yale University, 1993.
5. C.C. Douglas, T.G. Mattson, and M.H. Schultz, 'Parallel Programming systems for workstation clusters', Tech. Rept. YALEU/DCS/TR-975 Dept. Computer Science, Yale University, 1993.
6. A.G. Hoekstra, 'Computer Simulations of Elastic Light Scattering:Implementation and Applications', Ph.D. Thesis, University of Amsterdam, 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*, pp. 167-172, M. Maeda (Eds.), 1993.
8. A.G. Hoekstra and P.M.A. Sloot, 'Simulating Elastic Light Scattering Using High Performance Computing Techniques', In *European Simulation Symposium 1993*, pp. 462-470, A. Verbraeck and E.J.H. Kerckhoffs (Eds.), Society for Computer Simulation International, 1993.
9. R. Hockney and M. Berry, 'Public International Benchmarks for Parallel Computers: Parkbench Committee report 1.', *Scientific Computing* **3**, 101-146 (1994).
10. D.W. Walker, 'The design of a standard message passing interface for distributed memory concurrent computers', *Parallel Computing* **20**, 657-673 (1994).

11. P.M.A. Sloot, private communication, for more information you can send email to peterslo@fwi.uva.nl.