Performance Analysis of Parallel Applications under Message Passing Environments

Marian Bubak[1], Włodzimierz Funika[1], Jacek Mościński[1,2]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059, Kraków, Poland
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
email: {bubak,funika,jmosc}@uci.agh.edu.pl

Abstract. We present a view on building a performance analysis tools for parallel applications running on networked workstations under message passing environments (PVM and MPI). Instrumenting a parallel program's components is carried out with a graphical tool. An execution trace is processed using a special trace handling language. A series of metrics and visualizations help infer about application behavior. We show the results of applying the environment facilities to performance analysis of a lattice gas simulation parallel program.

1 Introduction

Parallel applications behavior is usually hard to understand and tune since it is necessary to take into consideration a number of factors that can significantly affect an application's performance. Performance understanding and tuning may be divided into performance observation and modelling. Usually [1], performance observation comprises measurement, e.g. monitoring, data storage and analysis, the latter involves processing and presentation of trace data. Performance modelling aims at expressing performance trends in terms of internal and external factors of an application's execution, and the output of performance observation is used for calibration and validation of models.

One of the most comprehensive source of measurement data is tracing. There are three main approaches to capturing trace data: hardware, hybrid and software as well as a number of problems with them: compensation of perturbation of an application's execution, establishment of global time and portability. The first two approaches manage to minimize perturbation and setup global time, nevertheless the problem of portability still remains unsolved [2]. The software approach offers a larger portability that is of great importance for parallel applications on networked workstations, however, performance is affected by perturbations, and it is difficult to establish global time.

Due to indeterminism the behavior and performance of a parallel program running on networked workstations under message passing environments can hard be modelled and predicted. Therefore, observation remains the main tool of getting insight into a program's behavior and inferring. The main questions, which are to be answered by observation, are *when*, *where* and *why*, or in other words, *what* affects an appplication's performance. Answering questions *when*

and *where* is automated and presented in tabular form or visualized like performance metrics, indices, profiles, diagrams. Answers obtained can be used for inferring on the causes of poor performance, thus for answering the *why* question.

Our approach to building a performance analysis environment is a trial to integrate some important features offered by a number of recently developed tools and techniques [11]. These features are extensibility, flexibility, portability and ease of use.

2 Available monitoring and analysis tools

Users who develop parallel distributed applications for networked workstations have at their disposal a number of monitoring tools. The essential features of these tools are listed in Tab. 1.

Among them, ParaGraph [3], the first public domain tool, offers a lot of graphs and performance data, but it is difficult to extend and customize. XPVM [4], a simple in use monitoring, tool enables tracing PVM application. Its main disadvantage is that traces it generates represent potentially perturbed applications due to on-line event message routing by XPVM. The Pablo analysis environment [5] is an example of flexible and powerful toolkit comprised of generic analysis and visualization modules. Unfortunately, its displays have mainly statistic character. ParaDyn [6] is an intelligent tool of next generation exploiting dynamic instrumentation of applications, but its portability is insufficient to be applied on a wide variety of workstation architectures. Medea [7], one of the recent tools, allows the evaluation of the performance of parallel programs by processing trace files produced by monitoring tools and applying various types of statistical techniques along with visualization facilities. It enables filtering, cluster and fitting operations on the data in a number of formats. However, it lacks space-time and similar dynamic diagrams. VAMPIR [8] is a newly developed commercial tool for post-mortem analysis of traces generated by MPI, PVM and PARMACS applications.

3 Design of the performance analysis tool

3.1 Sources of the ideas underlying the design

A number of techniques used in some existing tools served the lines of investigation we followed. The monitoring facility is based on the monitoring tool TAPE/PVM [10]. Trace data is expressed in the SDDF meta-format [9] as a flexible way to store and access trace data. The Pablo environment is used as a ground for building a visualization environment, being, on one hand, an acting toolkit, on the other hand a contributor of ideas about what and how should be processed and presented while analyzing trace data.

Table 1. Monitoring and performance analysis tools for distributed programs

Name	type	trace generation	mode	visualization	instrum.
for PVM and PVM based programs					
HeNCE	env,d,pt	au	on,off	g,an,sm,p	au
XPVM	m,d,pt	au	on,off	g,an,sm,p	au
EASYPVM	ctl	au	off	PG	au
environment independent					
Pablo	pt	u	off	g,an	g, i-a
ParaGraph	pt	PICL	off	g,an,sm,p	src
IPS-2	pt	au, u	off	sm,g,p	au
JEWEL	pt	u	on,off	sm,g,p,an	src
AIMS	pt	au,tpt	off	g	g, i-a, src
SCOPE	pt	au,tpt	off	g	src
ParaDyn	pt	au,dt	on	g,sm,p	exe
VAMPIR	pt,d	au,tpt	on,off	g	g, i-a, src
Medea	pt	au,tpt	off	g	src
environment dependent					
PICL	ctl	au,u	off	ParaGraph	au,src
Express-tools	pt	au,u	off	g,sm,p	au,src
TOPSYS-tools	d,pt	au	on,off	g,an,sm,p	au
Linda-tools	d,pt	au	on,off	g,an,sm,p	au

an	animated	i-a	interactive
au	automatically generated	m	monitoring program
ctl	comm. & tracing library	off,on	off-line, on-line
d	debugging tool	p	performance
di	dynamic instrumentation	pt	performance tool
dt	dynamic tool	sm	summaries
env	environment	src	source
exe	executable	tpt	trace processing tool
g	graphical	u	user-defined

3.2 Trace handling language

In order to facilitate processing trace data processing a trace handling language (THL) has been designed. THL make it possible to access a trace in SDDF format, extract any data, process and combine it for output with a few-lines long program. The language has to be simple in use. The user does not need to know the SDDF handling functions. THL is used to filter and handle trace data with simple means without knowing SDDF functions. THL provides the features

common with a normal algorithmic language like C as well as some features specific to processing trace data. There is a number of built-in functions which enable to reconstruct the directed acyclic graph of execution of an application.

3.3 The structure of the environment

The whole process of applying the environment facilities to analysis of an application is divided into five phases: instrumentation, compilation and linkage, execution and measurement experiment, analysis, and visualization. In line with this layout the environment comprises the following parts: instrumentation facility, trace capture library, monitoring program (spawned by intrumented application), trace data access modules, performance analysis modules and visualization modules. In the course of a monitoring and analysis session static and dynamic information is generated and used later in the analysis and visualization phases. The structure of the environment is shown in Fig. 1.

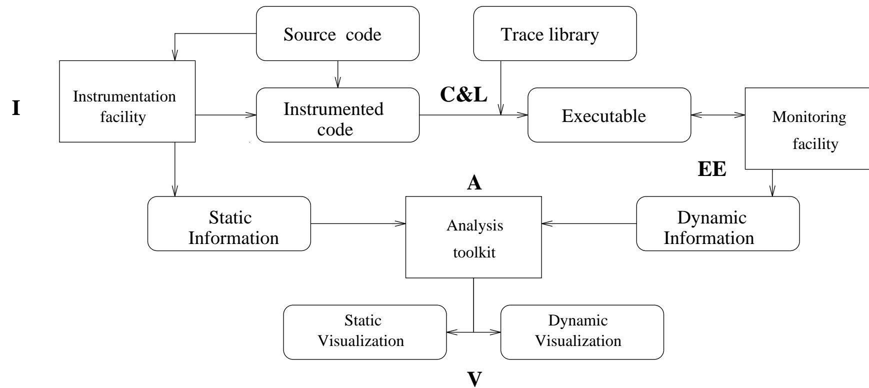

Fig. 1. The structure of the environment: **I**- instrumentation, **C&L** - compilation and linkage, **EE** - execution experiment, **A** - analysis, **V** - visualization.

3.4 Instrumentation

One of the most arduous tasks while preparing an application for monitoring is instrumentation. Programmer marks usually points of interest manually and do this mainly for timing purposes. The instrumentor developed enables instrumenting an application authomatically by means of a special window editor. The objects of instrumentation may be application function calls (entry and exit), communication library functions calls and loops. Instrumenting variables and source code fragments can be performed made manually with a number of primitives so far. The instrumented objects are supplied with the number of line and the source code identifier. The data relating to the instrumented source code

Fig. 2. A sample of instrumention session.

in the database are updated. A sample of instrumentation session is shown in Fig.2.

3.5 Compilation and execution

After a source code is instrumented it is compiled and linked with the trace library. The monitoring facility comprises tracing library primitives and a monitoring daemon. Primitives provide tracing the objects instrumented. The daemon monitors the most important events of an application's execution like spawning, exiting and killing tasks. The daemon obtains from the master process all the data indispensible to identify an experiment: program name, experiment date, architecture of the nodes, options of the experiment. These data are used to form a unique name of trace file. The name is attributed to a file where trace data from all nodes involved are collected. The trace file is placed in an individual directory of an experiment.

3.6 Trace file processing

The first action on a trace file obtained in the course of monitoring is perturbation compensation. Afterward trace data are converted into the SDDF metaformat. Events in trace records are supplied with additional data, e.g. time spent in individual states. Process numbers are converted into a range of numbers starting from 0. After that trace data are prepared to analysis phase.

3.7 Performance analysis

Performance analysis resembles a "cause-effect" inference process which enables to get insight into an application's behavior. Analysis is aimed at correlating implementation decision to an application performance. Analysis starts with global performance metrics such as speedup and efficiency and proceeds with increasingly deeper insights. Observations based on measurement can be roughly divided into summary statistics, tracing application events and generating abstract events. By providing an average evaluation of an application's preformance as a whole, summary statistics are sufficient for understanding performance trends sometimes. These can be presented as static views. Tracing application events enables observing an application's behavior at a more detailed level. The most usable visualization presenting application events is space-time diagram. This can be presented as dynamic view developing with time. Abstract events are synthetic values which represent the occurence of some performance phenomena. They comprise performance metrics, indices and categorization of performance phenomena. By the moment we have developed a number of performance metrics common with parallel programs and their distributions, e.g. utilization, communication activity, parallelism.

3.8 Static and dynamic visualization.

As it was mentioned above an indispensible part of performance analysis are statistic visualizations. A number of profiles in the form of time-usage diagrams are developed: time usage vs. nodes, time usage vs. nodes and functions, time usage vs. functions.

For dynamic visualization, a communication activity and application events time-lined diagrams were developed. Now we are working on improving the informative features of these diagrams in order to be able to correlate performance phenomena of an application to a source code.

4 Sample of performance analysis session

As an example of performance analysis with our environment we have chosen a typical application running on networked workstations under PVM - namely lattice gas automata simulation (LGA) for 480×2880 sites.

The experiments were carried out on a heterogenous network of workstations comprised of HP9000/712 (HP712), IBM RS/6000-520 (RS520), IBM RS/6000-320 (RS320) and SUN IPX (IPX) as well as on a homogenous network that consisted of IBM RS/6000-320 (RS320) workstations.

For parallel lattice gas automata (LGA) simulation lattice is divided into domains along the y axis [12]. Master sends to workers the geometrical description of a system simulated. Workers generate initial states of domains and after the computation phase send the averaged values to master. Communication among neighbors can be optionally asynchronous that results in reduced communication overhead. The communication topology is ring.

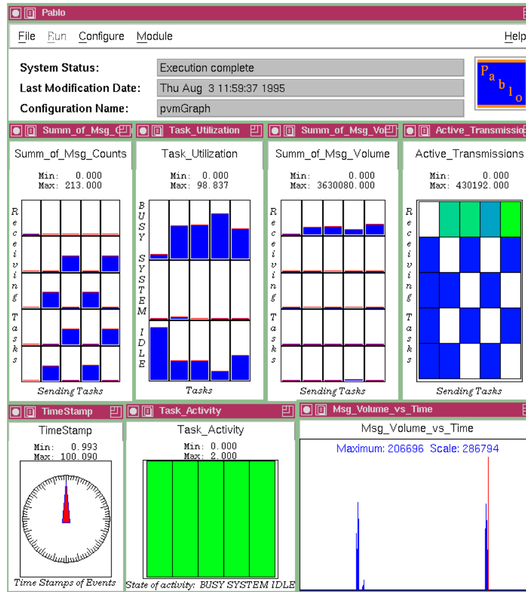

Fig. 3. Sample of monitoring session of LGA on a heterogenous network.

In Fig. 3 we present a sample of monitoring session with Pablo based displays for a run on 4 workstations (HP712, RS520, RS320 and IPX). The following performance data are presented: synchronized time of every event in a parallel program (Time_Stamp), matrix of current connections with the trace of recent message volume (Active_Transmissions), current computation time spent by every task between two consecutive barriers (Barrier_Mean_Time), cumulative time spent in each state by every task (Task_Utilization), cumulative number of messages sent between tasks (Summ_of_Msg_Counts), cumulative volume of messages sent between tasks (Summ_of_Msg_Volume), current total volume of messages in a program as a function of time (Msg_Volume_vs_Time), state (busy, system, idle) of every task (Task_Activity). We can see that most time is spent in computing except the first task which is the master.

We studied communication times, variability of the utilization metric across various configurations and the trends of individual routines' fractions in the execution time. Messages are short during evolution while the averaging (< 120 bytes) and long in load-balancing phases (cca. 100 - 200 KB). In Fig. 4 we show the distribution of send times accross processors on the heterogenous network of 4 machines. Send times vary on different machines for the same message lengths, e.g. for the length bytes from cca. 0.1 ms on HP712 to cca. 15 ms on RS320. On HP720 most send times are contained within 3 ms, on RS320 within 4.0 - 15 ms, on RS520 within 4.5 - 15 ms, on IPX within 0.5 to 7 ms. The most scattered times are on RS520 and RS320.

In Fig. 6 the time usage of LGA on a heterogenous network for the same as above configuration is shown. The times across nodes are almost equal, the send times being much lower than receive blocking times. But one can see that

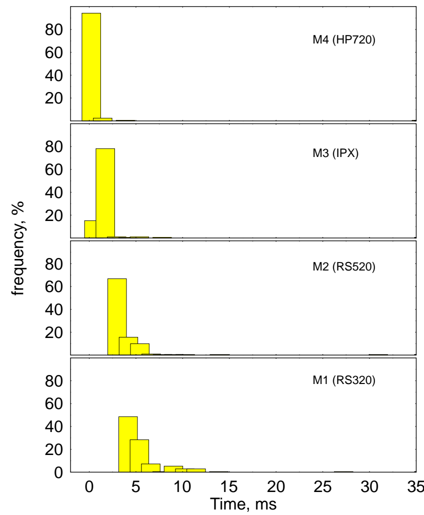

Fig. 4. Distribution of send time for LGA on 4 workstations.

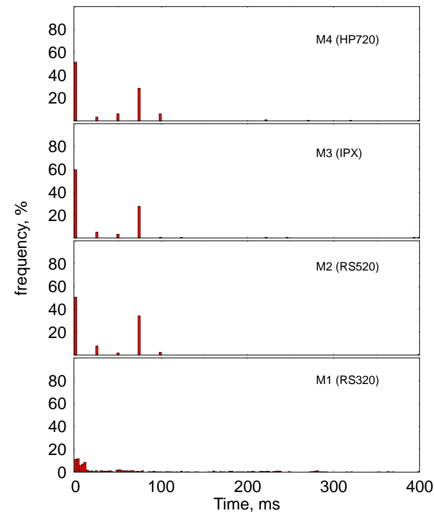

Fig. 5. Distribution of receive time for LGA on 4 workstations.

the structure of these timings varies across nodes. The worst usage is on HP712. It may seem that something is wrong with load-balancing. The node program comprises *main* and the principal unit *stepon*. As shown in Fig. 7 it can be seen that *stepon* procedure is rather good balanced as for its duration as well as *main*, but Fig. 8 suggests that percentage of the time spent in *stepon* decreases in favour of *main*.

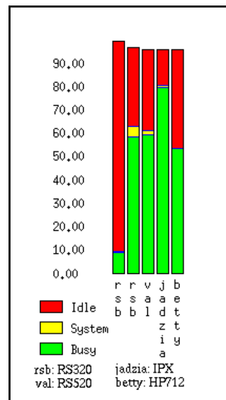

Fig. 6. Time usage by nodes for LGA on on four nodes.

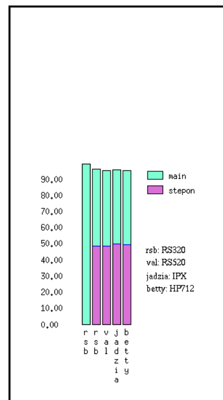

Fig. 7. Time usage by routines for LGA on four nodes.

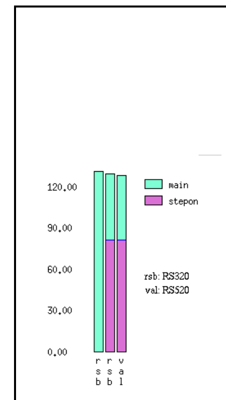

Fig. 8. Time usage of nodes by routines for LGA on two nodes.

Another view in Fig. 9 enables insight into the time usage of procedures. Here, it can be seen that the receive and send time percentages on HP712 are due to *main* procedure. Now one can conclude that a source of non-productive time is to be looked for in *main* procedure. It is obvious that inference process about

the causes of poor performance makes it indispensible that the performance analysis enviroment facilities could enable wandering about the information of different level of abstraction starting with speed-up and efficiency and correlating performance phenomena to implementation decisions, e.g. a source code.

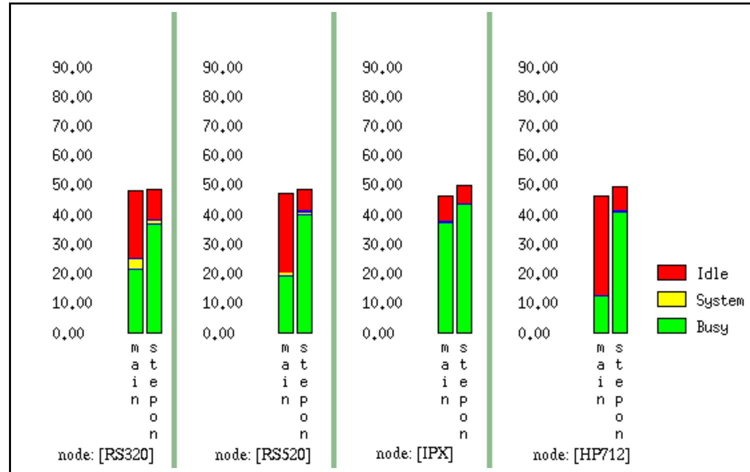

Fig. 9. Time usage of routines across nodes for LGA on four nodes.

5 Summary and perspectives

We have presented the approach to the development of the performance analysis environment for parallel applications running on networked workstations under message passing environments. Instrumentation is carried out with the graphical tool. The trace data are processed with the trace handling language. Trace analysis enables to get insight into performance trends. As an example the results of applying the environment to performance analysis of the lattice gas automata simulation parallel program were presented.

The environment will be extended to monitor MPI applications. Despite the difficulties we mentioned in the introduction we will address the issue of performance modelling. Performance losses due to communication, load imbalance, synchronization, insufficient parallelism as well as the impact of application and execution environment parameters on application performance will be the subject of further study.

Acknowledgements.
We are very grateful to Mr Marek Pogoda, Ms Renata Słota and Dr Éric Maillet for their valuable help. The contribution of Gregorz Kazior and Radosław Gembarowski is gratefully acknowledged.

This research was partially supported by the AGH grant.

References

1. Jain, R.: *The art of computer systems performance analysis*, John Wiley & Sons, 1991.
2. Hollingsworth, J.K., Lumpp, J.E., and Miller, B.P.: Techniques for performance measurement of parallel programs, *Parallel Computers: Theory and Practice (IEEE Press)*, 1995, pp. 225-240.
3. Heath, M.T. Etheridge, J.A.: Visualizing the performance of parallel programs. *IEEE Software* September 1991, pp. 29-39.
4. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V,: PVM: Parallel Virtual Machine. MIT Press, 1994.
5. Reed, D.A. Aydt, R., Madhyastha, T.M., Noe, R.J., Shields, K.A., and Schwartz, B.W.: The Pablo Performance Analysis Environment. *Technical Report*, Dept. of Comp. Sci., University of Illinois, 1992.
6. Miller, B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., Karavanic, K.L., Kunchithapadam, K., and Newhall, T.: The Paradyn Parallel Performance Measurement Tool, *IEEE Computer*, vol. 28, No. 11, November 1995, pp. 37-46.
7. Calzarossa, M., Massari, L., Merlo, J., Tessera, D.: Parallel Performance Evaluation: The Medea Tool, in: *Liddell, H., Colbrook, A., Hertzberger, B., Sloot, P., (eds.), Proc. Int. Conf. High Performance Computing and Networking, Brussels, Belgium, April 1996*, Lecture Notes in Computer Science **1067**, 522-529, Springer-Verlag, 1996.
8. http://www.pallas.de/pages/vampir.htm
9. Aydt, R. A.: The Pablo Self-Describing Data Format. *Technical Report*, Dept. of Comp. Sci., University of Illinois, 1994.
10. Maillet, E.: TAPE/PVM an efficient performance monitor for PVM applications - User guide. *IMAG, Institut National Polytechnique de Grenoble*, 1995.
11. Bubak M., Funika W., Mościński J., Tasak D.: Pablo based monitoring tool for PVM applications, in: Jack Dongarra, Kaj Madsen, Jerzy Waśniewski (Eds): *PARA95 - Applied Parallel Computing*, Springer-Verlag Lecture Notes in Computer Science **1041** (1996) 69-78.
12. Bubak, M., Mościński, J., Słota, R., Implementation of parallel lattice gas program on workstations under PVM, in: Dongarra, J.J., Hansen, P.C, and Waśniewski, J., *Proc. PARA'94 – Workshop on Parallel Scientific Computing*, Lyngby, Denmark, June 1994, *Report UNIC-94-05*; and in: Dongarra, J., Waśniewski, J. (Eds.): "Parallel Scientific Computing", Lecture Notes in Computer Science **879**, Springer-Verlag, 1994, pp. 13 6-146.

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