Integrating Automatic Techniques in a Performance Analysis Session

Antonio Espinosa, Tomas Margalef, Emilio Luque

Computer Science Department
Universitat Autonoma of Barcelona
08193 Bellaterra, Barcelona, SPA1N
{ antonio.espinosa, tomas.margalef, emilio.luque }@uab.es

Abstract. In this paper, we describe the use of an automatic performance analysis tool for describing the behaviour of a parallel application. KappaPi tool includes a list of techniques that may help the non-expert users in finding the most important performance problems of their applications. As an example, the tool is used to tune the performance of a parallel simulation of a forest fire propagation model

1. Introduction

The main reason for designing and implementing a parallel application is to benefit from the resources of a parallel system[1]. That is to say, one of the main objectives is to get a satisfing level of performance from the application execution. The hard task of building up an application using libraries like PVM[2], MPI[3] must yield the return of a fast execution in some cases or a good scalability in others, if not a combination of both.

These requirements usually imply a final stage of performance analysis. Once the application is running correctly it is time to analyse whether it is getting all the power from the parallel system it is running on.

To get a representative value of the level of performance quality of an application is necessary to attend to many different sources of information. They range from the abstract summary values like accumulated execution or communication time, to the specific behaviour of certain primitives. Middle solutions are available using some visualization tools [4,5].

The main problem with the performance analysis is the enormous effort that is required to understand and improve the execution of a parallel program. General summary values can focus the analysis in some aspects leaving others not so apparently crucial. For example, the analysis can be focused on communication aspects when the average waiting times are high. But then, the real analysis begins.

It seems rather manageable to discover the performance flaws of a parallel application from this general sources of information. Nevertheless, there is a considerable step to take to really know the causes of the low performance values detected. A great deal of information is required at this time, like which are the

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 173-177, 2000.

processes that are creating the delay, what operations are they currently involved with, or which is the desired behaviour of the primitives used by the processes, and in what differs from the actually detected execution behaviour.

In other words, it is necessary to become an expert in the knowledge of the behaviour of the parallel language used and its consequences in the general execution.

To avoid this difficulty of becoming an expert in the performance analysis, a second generation of performance analysers have been developed. Tools like Paradyn [6], AlMS[7], Carnival[8] and P3T[9] have helped the users in this effort of the performance analysis introducing some automatic techniques that alleviate the difficulty of the analysis.

In this paper we present the use of Kappa Pi tool [10] for the automatic analysis of message-passing parallel applications. Its purpose is to deliver some hints about the performance of the application concerning the most important problems found and a possible suggestion about what can be done to improve the behaviour.

2. KappaPi Tool. Rule-Based Performance Analysis System

KappaPi is an automatic performance analysis tool for message-passing programs designed to provide some explanations to the user about the performance quality achived by the parallel program in a certain execution.

KappaPi takes a trace file as input together with the source code file of the main process (or processes) of the application. From there on, the objective of the analysis is to find the most important performance problems of the application looking at the processors' efficiency values expressed at the trace file events. Then, it tries to find any relationship between the behaviour found and any source code detail that may reveal information about the structure of the application.

The objective of KappaPi then is to help the programmers of applications to understand the behaviour of the application in execution and how it can be tuned to obtain the maximum efficiency.

First of all, the events are collected and analysed in order to build a summary of the efficiency along the execution interval studied. This summary is based on the simple accumulation of processor utilization versus idle and communication time. The tool keeps a table with those execution intervals with the lowest efficiency values.

At the end of this initial analysis we have an efficiency index for the application that gives an idea of the quality of the execution. On the other hand, we also have a final table of low efficiency intervals that allows us to start analyzing why the application does not reach better performance values.

The next stage in the KappaPi analysis is the classification of the most important inefficiencies. KappaPi tool identifies the selected inefficiency intervals with the use of a rule-based knowledge system. It takes the corresponding trace events as input and applies a set of behaviour rules deducing a new list of facts. These rules will be applied to the just deduced facts until the rules do not deduce any new fact. The higher order facts (deduced at the end of the process) allow the creation of an explanation of the behaviour found to the user. These higher order facts usually

represent the abstract structures used by the programmer in order to provide hints related to programming structures rather than low level events like communications.

The creation of this description depends very much on the nature of the problem found, but in the majority of cases there is a need of collecting more specific information to complete the analysis. In some cases, it is necessary to access the source code of the application and to look for specific primitive sequence or data reference. Therefore, the last stage of the analysis is to call some of this "quick parsers" that look for very specific source information to complete the performance analysis description.

3. Examining an Application: Forest Fire Propagation

The Forest Fire Propagation application (Xfire)[11] is a PVM message passing application that follows a master-worker paradigm where there is a single master process which generates the partition of the fireline and distributes it to the workers. These workers are in charge of the local propagation of the fire itself and have to communicate the position of the recently calculated fireline limits back to the master. The next step is to apply the general model with the new local propagation results to produce a new partition of the general fireline.

The first trace segment is analysed looking for idle or blocking intervals at the execution. A typical idle interval is the time waiting for a message to arrive when calling a blocking receive. All these intervals are identified by the id's of the processes involved and a label that describes the primitive that caused the waiting time.

Once the blocking intervals have been found, Kpi will use a rule-based system to classify the inefficiencies using a deduction process. The deduced facts express, at the beginning, a rather low level of expression like "communication between master and worker1 in machine 1 at lines 134, 30", which is deduced from the send-receive event pairs at the trace file. From that kind of facts some others are built on, like "dependency of worker1 from master" that reflects the detection of a communication and a blocking receive at process fireslave. From there, higher order level facts are deduced appling the list of rules. In the case of Xfire application, Kpi finds a master/worker collaboration. Once this collaboration has been detected, with the help of the ruled-based system, Kpi focuses on the performance details of such collaboration.

The first situation found when analysing Xfire in detail, is that all processes classified as master or worker in the deduced facts wait blocked for similar amounts of time, being the master the one that slightly accumulates more waiting time. This seems to mean that there is no much overlapping between the generation and the consumption of data messages.

On one hand, while the master generates the data, the worker waits in the reception of the next data to process. On the other hand, while the workers calculate the local propagation of the positions received, the master waits blocked to get back the new positions of the fireline boundaries. Then, the reception of messages at the master provokes a new generation of data to distribute.

Once this kind of collaboration is found, KappaPi tool tries to find which is the configuration of master-workers that could maximize the performance of the application. In this case, the key value is the adequate number of slaves to minimize the waiting times at the master.

To build a suggestion to the programmer, Kpi estimates the load of the calculation assigned to each worker (assuming that they all receive a similar amount of work). From there, Kpi calculates the possible benefits of adding new workers (considering the target processor's speed and communication latencies). This process will end when Kpi finds a maximum estimated number of workers to reduce the waiting times.

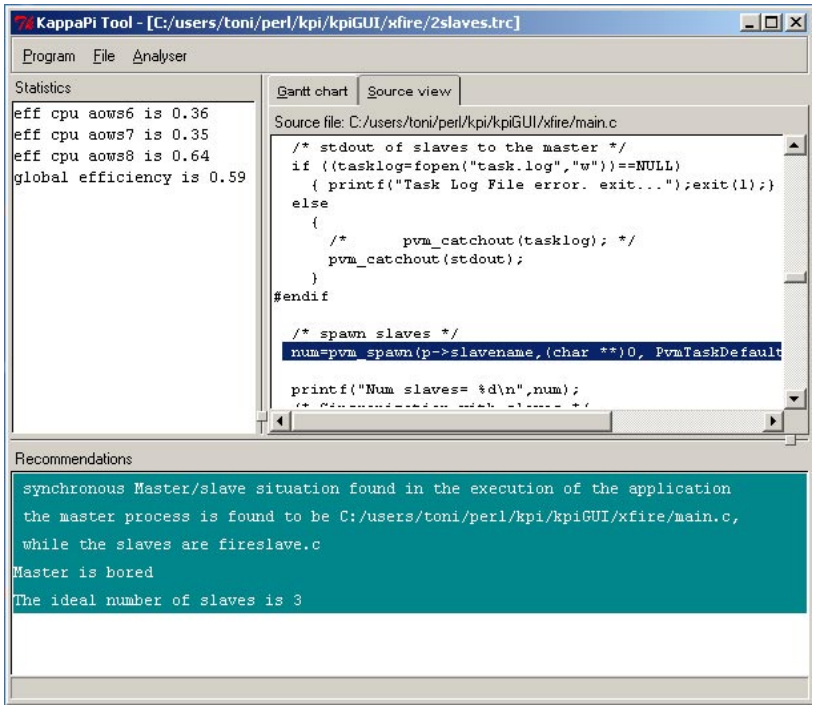

Fig. 1. Final view of the analysis of the Xfire application

Figure 1, shows the feedback given to the users of Kpi when the performance analysis is finished. The program window is split in three main areas, on the left hand side of the screen [statistics] there is a general list of efficiency values per processor. On the bottom of the screen [recommendations] the user can read the performance suggestion given by Kpi. On the right hand side of the screen [source view], the user can switch between a graphical representation of the execution (Gantt chart) and a view of the source code, with some highlighted critical lines that could be modified to improve the performance of the application. In the recommendations screen, the tool suggests to modify the number of workers in the application suggesting three as the best number of workers. Therefore, it points at the source code line where the spawn of the workers is done. This is the place to create a new worker for the application.

4. Conclusions

In conclusion, Kpi is capable of automatically detect a high level programming structure from a general PVM application with the use of its rule-based system. Furthermore, the performance of such an application will be analysed with the objective of finding which are their limits in the running machine. This process has been shown using a forest fire propagation simulator.

5. Acknowledgments

This work has been supported by the CYCIT under contract TIC98-0433

References

[1] Pancake, C. M., Simmons, M. L., Yan J. C.: „Perfonnance Evaluation Tools for Parallel and Distributed Systems“. lEEE Computer, November 1995, vol. 28, p. 16-19.

[2] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V., „PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Network Parallel Computing“. MIT Press, Cambridge, MA, 1994.

[3] Gropp W., Nitzberg B., Lusk E., Snir M.: „Mpi: The Complete Reference: The Mpi Core/the Mpi Extensions. Scientific and Engineering Computation Series“. The MIT Press. Cambridge, MA, 1998.

[4] Heath, M. T., Etheridge, J. A.: „Visualizing the performance of parallel programs“. IEEE Computer, November 1995, *vol.* 28, p. 21-28.

[5] Reed, D. A., Giles, R. C., Catlett, C. E.. „Distributed Data and Immersive Collabolation“. Communications of the ACM. November 1997. Vol. 40, No 11. p. 3948.

[6] Hollingsworth, J. K., Miller, B, P.: „Dynamic Control of Performance Monitoring on Large Scale Parallel Systems“. International Conference on Supercomputing (Tokyo, July 1993).

[7] Yan, Y. C., Sarukhai, S. R.: „Analyzing palallel program performance using normalized performance indices and trace transformation techniques“. Parallel Computing 22 (1996) 1215-1237.

[8] Crovella, M.E. and LeBlanc, T. J.: „The search for Lost Cycles: A New approach to parallel performance evaluation“. TR479. The Unhersity of Rochester, Computer Science Department, Rochester, New York, December 1994.

[9] Fahringer T.: „Automatic Performance Prediction of Parallel Programs“. Kluwer Academic Publishers. 1996.

[10] Espinosa, A., Margalef, T. and Luque, E.: „Automatic Performance Evaluation of Parallel Programs“. Proc. of the 6th EUROMICRO Workshop on Parallel and Distributed Processing, pp. 4349. IEEE CS. 1998. http://www.caos.uab.es/kpi.html

[11]Jorba, J., Margalef, T., Luque, E., Andre, J., Viegas, D. X.: "Application of Parallel Computing to the Simulation of Forest Fire Propagation". Proc. 3td International Conference in Forest Fire Propagation, Vol. 1, pp. 891-900, Luso, Nov. 1998.