Automatic structure extraction from MPI applications tracefiles*

Marc Casas, Rosa M. Badia, and Jesús Labarta

Barcelona Supercomputing Center (BSC),
Technical University of Catalonia (UPC),
Campus Nord, Modul C6, Jordi Girona, 1-3, 08034 Barcelona

Abstract. The process of obtaining useful message passing applications tracefiles for performance analysis in supercomputers is a large and tedious task. When using hundreds or thousands of processors, the tracefile size can grow up to 10 or 20 GB. It is clear that analyzing or even storing these large traces is a problem. The methodology we have developed and implemented performs an automatic analysis that can be applied to huge tracefiles, which obtains its internal structure and selects meaningful parts of the tracefile. The paper presents the methodology and results we have obtained from real applications.

1 Motivation and Goal

In the recent years, parallel platforms have amazingly increased in performance and in number of nodes and processors. Thus, the study of the execution of applications in these platforms has become a hard and tedious work. A complete timestamped sequence of events of an application, that is, a tracefile of the whole application, results in a huge file (10-20 GB). It is impossible to handle this amount of data with tools like Paraver [1]. Also, often, some parts of the trace are perturbed, and the analysis of these parts can be misleading. A third problem is the identification of the most representative regions of the tracefile.

To reduce tracefiles sizes, the process of application tracing must be carefully controlled, enabling the tracing in the interesting parts of the application and disabling otherwise. The number of events of the tracefile (hardware counters, instrumented routines, etc...) must be limited. This process is tedious and large and requires knowledge on the source code of the application.

For these reasons, several authors [8, 9] believe that the development and utilization of trace based techniques is not useful. However, techniques based on tracefiles allow a very detailed study of the variations on space (set of processes) and time that could affect notably the performance of the application. Therefore, there is a need for developing techniques that allow to handle large event traces.

The goal of our approach is to start from very large tracefiles of the whole application, allowing simple tracing methodologies, and then analyzing them automatically. The underlying philosophy is to use resources that are generally

* Funded by project TIN2004-07739-CO2-01 and by a FPI grant from spanish gov.

available (Disk, CPU, ...) in order to avoid spending an expensive resource: analyst time. The tool we have implemented will, first, warn the analyst about those parts of the trace perturbed by an external factor not related to the application or to the machine itself. Second, the tool will give a description of the internal structure of the application and will identify and extract the most relevant parts of the trace.

There are other approaches to either avoid or handle large event traces. KOJAK [2] is a tool for automatic detection of performance bottlenecks and inefficient behavior. Our methodology could be applied before KOJAK to reduce the size of the tracefile. VAMPIR Next Generation tool (VNG) [3, 4] consists of two major components: A parallel analysis server and a visualization client, each of them executed on a different platform. An important VNG feature is the utilization of the data structure Complete Call Graph (CCG). It holds the full event stream including time information in a tree. It is also possible to compress the CCG into a compressed Call Graph (cCCG) in order to achieve a compressed representation of trace data. Compression errors can be maintained in a given range. Finally, the main goal of VNG is to make huge event traces accessible for interactive program analysis. Therefore, the VNG approach is different from ours since it does not perform an automatic analysis of the internal structure of the event traces. Related to VNG, there is another tool called DeWiz [5]. It is based, as VNG, on the event graph model. Two important characteristic of DeWiz are modularity, which enables it to be executed in distributed computing infrastructures, and automatic analysis, which enables it to detect significative information on the event graph. However, DeWiz is unable to find event based structure since it works using graph based methods. In that context, our work satisfies the need for an automatic performance analysis based on structural properties of the application. Furthermore, these structural properties of the application are event based and, for that reason, they have clear physical meaning. Other previous work in [6] presents a proposal for dynamic periodicity detection of iterations in parallel applications.

The paper is organized as follows: First, an explanation of the methodology we have developed and implemented is presented in Section 2. Next, a presentation of the results we have obtained using this methodology is described in section 3. Finally, conclusions and future work are shown in Section 4.

2 Methodology

The starting point is a Paraver tracefile generated with OMPItrace package [11]. This tracefile consists of a complete timestamped sequence of events of the whole execution of an application. The first problem we consider is the detection of perturbed regions of the tracefile. The second phase consists in a search for the internal structure of the trace, based on periodicities. In those two phases, we will use signals to characterize properties of the tracefile. Both phases will be handled using techniques of signal processing. Mainly, we will use non-linear filtering in both phases and spectral analysis in the second one.

2.1 Clean-up

In this phase, our tool performs an analysis oriented to the identification of the perturbed regions of the tracefile. By perturbed regions we mean those regions with distorted relative timing behavior of the application being analyzed. In [7] is shown a detailed discussion about these distortions. Furthermore, all these perturbations have a common characteristic: They are neither caused by the application nor the architecture. They are caused by external factors such as tracing packages, unknown system activity, etc. Different phenomena or metrics can be identified as being the cause of a significant perturbation of the program behavior. An example of those phenomena is flushing, which is caused by the fact that tracing packages keep individual records in a buffer in memory during the tracing process. The problem is that when the buffer is full, these records will have to be flushed to disk. This flushing will take a significative time, will affect the execution and the statistics derived from the tracefile. Also, the flushing does not appear simultaneously in all the processes although it is typical that the flush of the different processes occur in bursts.

Identifying perturbed regions The flushing phenomenon will be characterized by a signal indicating for each instant of time the number of processors flushing to disk. We derive that signal from a Paraver tracefile that contains flushing events, indicating when each process starts and finish the flushing to disk. Figure 1 shows an example of a flushing signal. In this kind of signals we frequently observe interleaved small bursts with flushing peaks and periods without flushing. The tracefile is perturbed not only during flushing but also in instants right after flushing peaks. Therefore, we want to consider the bursts of flushing as a single perturbed region.

With this objective, we will use a set of morphological filters, defined in the context of Mathematical Morphology. These filters are non-linear and are based on the minimum and maximum operations, aiming at the study of structural properties of the signal. The two basic morphological filters are **Erosion** and **Dilation**. The first has the property of eroding those regions of the signal with values different to zero. The second filter has the property of dilating the regions of the signal different to zero. Both operators have associated a width that has to be specified before the filter is applied. If we combine the two operators doing a Dilation followed by an Erosion we obtain an interesting result: First, the Dilation will merge the small regions with their larger or nearby neighbors. After that, the Erosion will allow us to return towards the initial signal, except in the cases that two different regions have been merged by the Dilation. With this combination, we obtain a new morphological operator called **Closing**. figure 2 shows the result of performing a Closing to the signal represented in figure 1. Note that the small regions that appear in figure 1 have been merged into larger region. The fundamental concepts and the formal definitions of Mathematical Morphology are described in [15].

In summary, following the methodology described in this section, we do three steps: First, we generate from the initial tracefile a signal, for example the num-

ber of processes flushing to disk. Second, we apply a Closing in order to merge the pulses of the signal that are too close in an unified burst. The width associated to the Erosion and the Dilation are the same. Its choice is based on the minimum span of time we consider useful for the analysis. The pulses of the resulting signal indicate which are the perturbed (thus, useless) regions of the tracefile and the areas of the signal that equals zero indicate which are the non-perturbed regions. Finally, the process of identification of structure explained in the next section is applied to the non-perturbed regions of the trace.

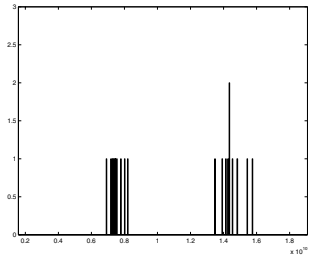

Fig. 1. Signal before closing

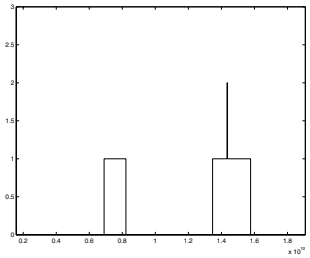

Fig. 2. Signal after closing

2.2 Structure

There are two main characteristics about the structure that we will look for in a trace. First, the periodic structure is based on the identification of several different trace regions that are very similar. Furthermore, we will say that two trace regions are very similar if the signal that represents a metric is very similar in the two regions. Another approach based in this type of structure is presented in [17]. The second main characteristic we look for is the hierarchy of the periodicities. The periodic structure will be expressed in different levels: The first level is the structure within the original trace, the second level of periodicities is the structure within one period of the first level, and so on. In order to obtain that hierarchical structure, our algorithm will be recursive, i. e., when the internal structure of one level is detected, we will apply again the algorithm within one period of that level.

The information derived from this hierarchical structure based on periodicities is useful in, at least, two aspects. First, our tool will show the structure to the analyst as a first approach to the execution of the application under analysis. Second, the tool will provide the user with chunks of traces which are cut from the original trace. These small traces are representative parts of the original trace at different levels of the structure.

These chops of the original tracefile will allow an accurate analysis, but the tool is also reporting several metrics (percentage of time in MPI [10], ...) for each of the regions to give a first approach to the performance obtained by the application.

Metrics There are many metrics that can be used to identify periodicities. Indeed, any user function correctly defined can be considered to generate a signal that describes several aspects of the execution. The limitation is that there are metrics that need some information not always included in the tracefile. The instantaneous MFLOPs ratio might be representative of the program structure, but needs hardware counters. If these counters are not included in the original tracefile, it will be impossible to generate the signal. Several examples of metrics that can be used without hardware counters and that can capture the global structure of the application are the following:

Number of MPI Point to Point Calls. This metric is represented by a signal which indicates how many MPI Point to Point Calls are being executed in a given moment.

Number of MPI Collective Calls. Very similar to the above metric, but considering MPI Collective Calls.

Specific MPI Call. This metric is represented by a signal which says how many calls to a specific MPI function are being executed in a given moment.

Autocorrelation To find the internal structure of the application we apply the Autocorrelation function [14], to the signal generated from the tracefile:

$$A(k) = \sum_{i=0}^{N-1} (x_i - \mu)(x_{i+k} - \mu) \tag{1}$$

where μ is the arithmetic mean of the set $\{x_i\}$. This set is generated sampling the signal obtained from the tracefile. The higher values of the function $A(k)$ will be reached when k is equal to one of the main periods of $\{x_i\}$. However, for accuracy reasons [14], the numerical values of $A(k)$ are not obtained following (1). It is possible to obtain the value of $A(k)$ function performing, first, a Discrete Fourier Transform (DFT) and, after that, an Inverse Discrete Fourier Transform (IDFT) taking the square of the modulus of each spectral coefficient obtained with the DFT [14]. This method can we implemented using a FFT library. An important feature of FFT is its computational complexity, O($n\ log(n)$), which allows us to calculate the values of $A(k)$ in reasonable time.

Periodicities Once we have the values of the Autocorrelation function, the principal periodicities are selected [18]. We will select the maximum of the relatives maximums. In other words, the period we select, T, will satisfy the following:

$$A(T) = Max\{A(k)|A(k-1) < A(k) > A(k+1), k > 0\} \tag{2}$$

A remarkable point here is that it is possible that the signal does not have meaningful periods or that has 2 or 3 significant periods. Therefore, there is a need for a method to estimate the correctness of the period obtained. The

approach taken is the following: assuming that T is the period identified and that M is the set of those k where $A(k)$ has a relative maximum.

$$\forall k, (k \in M \wedge k \neq T) \Rightarrow 0.9 > \frac{A(k)}{A(T)} \tag{3}$$

If the above formula holds, the 90 % of the value of $A(T)$ is higher than all the values in the rest of the maximum values. In that case, we will assume that T is a good approximation to the main period. We will check the logical formula (3) every time we perform an autocorrelation. If the formula is true, we will assume the correctness of the results. If not, we will perform a closing to the original signal in order to filter the small oscillations that can perturb the results and repeat the process. We use the closing filter in order to obtain a coarse-grained description of the signal. This replacement of a fine-grained description with a lower-resolution coarse-grained model will outline the global signal behavior.

Once a "good" period is identified, we select a region of the signal containing an iteration of this period and apply the methodology again to look for inner structure. At the same time, we cut the original tracefile in order to provide to the analyst one period on every periodic zone we found.

Finally, this methodology needs the execution of intense processes. In order to perform these executions and take advantage of several concurrent processes, we have implemented the methodology with GRID Superscalar [12], a grid programming environment developed at BSC.

3 Results

We have applied the methodology explained above to four real applications: Liso [19] with 74 processors, Idris [20] with 200 processors, Gadget [21] with 256 processors and Linpack[22] with 2048 processors. These have been executed and traced in MareNostrum. The structure found in these applications is based on the first of the metrics explained in section 2.2, the MPI Point to Point calls.

In figure 3 we show graphically a part of the structure of the Liso tracefile. This structure is shown with a Paraver visualization. In that visualization, the horizontal axis represents the time and the vertical axis the different processes. Black color means that a given process in a given instant of time is not executing any MPI Call. On the other hand, a light colored point (green when printing or visualizing in color) represents that an MPI call is being executed by the process. The picture first shows a visualization of the whole tracefile. The flushing regions are also outlined. In the second part of figure 3 we show the structure of the first region without flushing. In that case, we show, first, a region with a non-periodic structure that corresponds with the initialization phase of the application. The span of the initialization phase is 18029 ms. After that, there is periodic region with 5 iterations. The span is 47306 ms and the period shown is 9010 ms. Finally, in figure 3 we show one of the iterations of the periodic zone.

Table 5 shows that the automatic system has been able to detect the structure shown in figure 3. Furthermore, we can see the results of the automatic analysis

for the whole Liso application. The first five rows correspond to the structure represented in figure 3.

Mainly, in table 5 we show two characteristics of the execution: First, from left to right we show the hierarchy. Second, from top to bottom we show the temporal sequence.

In the first column, we show the duration of the whole execution in milliseconds. Next, in the second column of table 5 there is a decomposition of the total elapsed time of execution. In this second column, we show a set of numbers in each cell. The first number is the total time span of the region, the second is the number of periods found in that region and, finally, the third is the duration of each period. In the regions where no periodicity has been found a dash line is written. This second column refers to the first level of hierarchical structure, i. e., is the structure over the original trace. For example, the MPI Point to Point calls distribution of the first (18020 ms) and second (47306 ms) regions of the tracefile shown in figure 3 are represented in the first and second cells of the second column.

The third column contains the second level of the structure, i.e, the structure that can be found in one of the periods of the first level. For example, the second, third, fourth and fifth cells of the third column are the decomposition of one of the periods of level 1. The MPI Point to Point distribution is shown on figure 3. Here it can be identified the second level of structure, with 6 periods (of 545 ms) of communication, a computation period (of 1005 ms), 3 periods more of communication (each of 550 ms) and a final computation period (of 3180 ms).

Finally, the fourth column shows the existence of flushing events in a given region of the tracefile.

The output of the tool is basically the information contained in this table plus the names of the files where can be found the chops of the original tracefile.

In table 1 we show the structure detected in Idris application. In table 2 we show another possible representation of the same information. Finally, in tables 3 and 4 the structure found in Gadget and Linpack applications is shown.

Table 6 shows the average size of the chops of the original trace. As we have said in section 2.2, every time the system finds a periodic region it selects one of the periods of that region and cuts the tracefile to provide the analyst with representative chops of the original tracefile. The sizes shown in table 6 are, first, the size of the total tracefile and, second, the average size of the periods of the first level. For example, Liso tracefile has 6 periodic regions, one of these regions in every non-flushing zones. If we take one period of every periodic region and then we cut the tracefile, we will obtain 6 small tracefiles. The average of its sizes is the value we show. Finally, the third column is the average size of the second level periods. Note the large reduction in the amount of data to study.

Finally, in figure 4 we represent, first, the size of the whole trace. Next, we show the total sum of the sizes of the first and second level chops. Obviously, if there is only one periodic region, the value shown in figure 4 is the same as the value represented in table 6. The first level of Idris application is an example. The most important thing, however, is that the global behavior of the applications,

with the exception of the flushing and initialization regions, is contained in Level 1 tracefiles. We have reduced notably the amount of data to be analyzed in order to study the performance of the applications.

Table 1. Idris. Table Representation

Span/#it/T			Flushing
Level 0	Level 1	Level 2	
1091695/1/-	205740/1		
	885955/9/102870	550/5/110	
		590/1/-	
		540/7/80	
		101190/1/-	

Table 2. Idris. Tree Representation

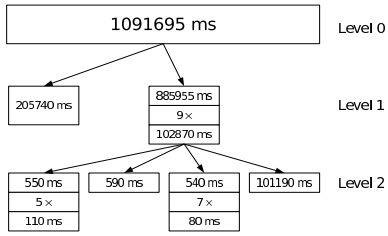

Table 3. Gadget

Span/#it/T		Flushing
Level 0	Level 1	
1097460/1/-	55000/1/-	
	811126/23/35095	
	286334/1/-	X

Table 4. Linpack

Span/#it/T		Flushing
Level 0	Level 1	
223168/1/-	82850/1/-	
	140318/130/1130	

4 Conclusions and Future work

In this paper we have analyzed the possibility of automatically deriving the internal structure of a tracefile. This structure has two main properties: First, it is based on periodicities and, second, is hierarchical. We have shown that is possible to, first, detect the perturbed regions of the tracefile and, second, derive the internal structure of non-perturbed regions. It is useful in many aspects: It makes easier the process of tracing the application, it avoids the spending of time studying perturbed zones, it gives the internal structure of the tracefile and, finally, gives the most representative regions of it. In conclusion, we have reduced the problem of analyzing a huge tracefile (10 or 20 Gb) to the study of several hundreds of Mb.

In the future, this tool will perform an analysis of other parallelization problems such as Load Imbalance, efficiency, overhead, etc... Our objective is to automatize all the process of analysis and visualization of Paraver tracefiles with the intention to reduce the time required to analyze a tracefile. Finally, this tool will incorporate an expert system. It will be able to detect new problems and learn about it. What is more, the potential of tools such as Dimemas [13] will be used with the objective of predicting and automatically detecting the performance of message passing applications in hypothetic architectures.

Fig. 3. On top, visualization of the whole Liso tracefile and the flushing zones . In the middle, the first region without flushing is shown. We highlight a region without periodic structure and a region with 5 iterations. At the bottom, we show one of these iterations.

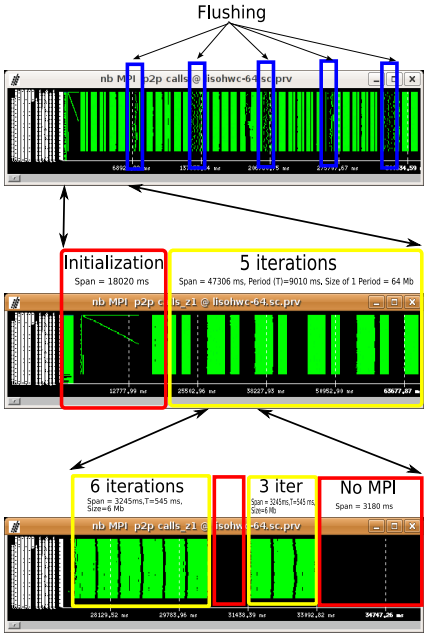

Fig. 4. Sum of the sizes of all the representative traces of each level.

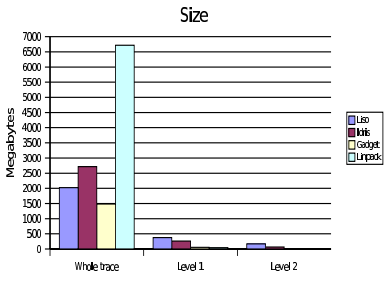

Table 5. Liso application structure detected by our system. The time units are milliseconds. The first three columns show the hierarchical levels of periodicity. Level 0 column shows the total elapsed time, Level 1 column shows the different phases detected by the automatic system and Level 2 shows the internal structure of one of the periods of Level 1. In the first three columns, each cell contains three numbers: The total span of the region, the number of the periods found in that region and the duration of each period.

Span/#it/T			Flushing
Level 0	Level 1	Level 2	
356352/1/-	18020/1/-		
	47306/5/9010	3245/6/545	
		970/1/-	
		1615/3/550	
		3180/1/-	
	10273/1/-		X
	49166/5/9105	1490/3/545	
		1140/1/-	
		1585/3/550	
		3030/1/-	
		1860/3/535	
	11880/1/-		X
	60773/7/9100	3215/6/540	
		1004/1/-	
		1615/3/550	
		3266/1/-	
	11576/1/-		X
	49253/5/9145	1650/3/550	
		2825/1/-	
		3400/6/550	
		1270/1/-	
	12269/1/-		X
	48347/5/9045	3185/6/550	
		1015/1/-	
		1625/3/560	
		3220/1/-	
	13312/1/-		X
	24177/3/8905	1875/1/-	
		3425/6/550	
		1010/1/-	
		1635/3/555	
		960/1/-	

Table 6. Sizes of all the representative traces of each level.

Application	Total Trace Size	Level 1 Size	Level 2 Size
Liso	2.02 Gb	64 Mb	6 Mb
Idris	2.7 Gb	250 Mb	25 Mb
Gadget	2.7 Gb	53 Mb	
Linpack	6.7 Gb	46 Mb	

References

1. Paraver: performance visualization and analysis, http://www.cepba.upc.es/Paraver/
2. KOJAK: Kit for Objective Judgment and Knowledge-based Detection of Performance Bottlenecks, http://www.fz-juelich.de/zam/kojak/
3. Knuepfer, A.; Brunst, H.; Nagel, W. E.; "High Performance Event Trace Visualization", Proc. PDP 2005, 258-263 (2005)
4. Brunst, H.; Kranzlmuller, D.; Nagel, W. E.; "Tools for Scalable Parallel Program Analysis - Vampir VNG and DeWiz", DAPSYS 2004, 93-102
5. Kranzlmuller, D.; Scarpa, M.; Volkert, J.; "DeWiz - A Modular tool Architecture for Parallel Program Analysis", Proc. Euro-Par 2003, 74-80
6. Freitag, F.; Corbalán, J.; Labarta, J.; "A Dynamic Periodicity Detector: Application to Speedup Computation". IPDPS 2001
7. Mohr, B.; Traff, J. L.; "Initial Design of a Test Suite for Automatic Performance Analysis Tools". IPDPS 2003
8. Nataraj, A.; Malony A.; Shende, S.; Morris, A.; "Kernel-Level Measurement for Integrated Parallel Performance Views: the KTAU Project", IEEE International Conference on Cluster Computing 2006
9. Vetter, J.S.; Worley, P.H.; "Asserting Performance Expectations", Supercomputing, ACM/IEEE 2002 Conference
10. The Message Passing Interface (MPI) standard, http://www-unix.mcs.anl.gov/mpi/
11. OMPItrace manual, www.cepba.upc.es/paraver/docs/OMPItrace.pdf
12. Badia, R. M.; Labarta, J.; Sirvent, R.; Perez, J. M.; Cela, J. M.; Grima, R.; "Programming grid applications with GRID Superscalar", Journal of Grid Computing, Volume 1, Issue 2, 2003
13. Dimemas: performance prediction for message passing applications, http://www.cepba.upc.es/Dimemas/ 3rd ed. New York: McGraw-Hill, pp. 40-45, 1999.
14. Press, W. H.; Flannery, B. P.; Teukolsky, S. A.; and Vetterling, W. T. "Correlation and Autocorrelation Using the FFT." 13.2 in Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed. Cambridge, England: Cambridge University Press, pp. 538-539, 1992.
15. Serra, J. "Image Analysis and Mathematical Morphology", Academic Press, 1982
16. Simon, B.; Odom, J.; DeRose, L.; Ekanadham, K.; Hollingsworth, J. K.; Sbaraglia, S.; "Using Dynamic Tracing Sampling to Measure Long Running Programs", Proceedings of the 2005 ACM/IEEE conference on Supercomputing, 2005.
17. Sherwood, T.; Perelman, E.; Hamerly, G.; Calder, B.; "Automatically Characterizing Large Scale Program Behavior" 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2002.
18. De Chevigne, A.; Kawahara, H.; "YIN, a fundamental frequency estimator for speech and music", Journal of Acoustical Society of America, 2002.
19. Hoyas, S.; Jimeńez, J. "Scaling of velocity fluctuations in turbulent channels up to Re=2003", Physics of fluids, 2006.
20. Teysser, R.; "Cosmological hydrodynamics with adaptive mesh refinement - A new high resolution code called RAMSES", Astronomy & Astrophysics, 2002
21. Springel V., Yoshida N., White S. D. M., "Gadget: a code for collisionless and gasdynamical cosmological simulations" New Astronomy, 6. 2001.
22. Linpack benchmark: http://www.netlib.org/linpack/