John von Neumann Institute for Computing

Automatic Phase Detection of MPI Applications

Marc Casas, Rosa M. Badia, Jesús Labarta

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr,
F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 129-136, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

Automatic Phase Detection of MPI Applications

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
E-mail: {mcasas, rosab}@ac.upc.edu, jesus@cepba.upc.es

In the recent years, strong efforts have been dedicated on the study of applications' execution phases. In this paper, we show an approach focused on the automatic detection of the phases on MPI applications' execution. This detection is based on Wavelet Analysis, which provides a meaningful and fast methodology. Information derived from phase detection is very useful to study the performance of the application. To illustrate this importance, we show a study of the scalability of several applications based on phase detection.

1 Motivation and Goal

The development and consolidation of supercomputing as a fundamental tool to do research in topics such as genomics or meteorology has prompted the release of a huge set of scientific applications (CPMD[13] , WRF-NMM[11]....). The study of the execution of those applications is fundamental in order to optimize the code and, consequently, accelerate the research in the scientific topic which the application comes from.

However, the current scientific applications are executed using thousands of processors. For that reason, huge tracefiles [a] (more than 10 GB) are generated. In that context, the traditional visualization tools (Paraver[15], Vampir[4,5], ...) become useless or, at the most, they can be used after filtering or cutting the tracefiles. Another solution is tracing again the application limiting the number of events of the tracefile. In any case, the research on new methodologies that allow the analyst to make an automatic analysis or, at least, to automatically obtain partial information based on the execution of a given application is justified.

Furthermore, the first goal of this paper is to use signal processing techniques (wavelet transform) in order to provide a very fast automatic detection of the phases of MPI applications' execution. The criterion of such signal processing techniques in order to perform the phase detection is to separate regions according to their frequency behaviour, i. e., a region with a small iteration which is repeated many times will we separated from another region with no periodic behaviour. The second goal of this work is to use the information derived from signal processing techniques to acquire remarkable conclusions about the scalability if the applications and how could be improved.

An important point of our work is that it enables the analyst to acquire some information without requiring any knowledge of the source code of the application. Several authors argue that this is a minor point because the analyst can acquire these information directly looking to the source code. Others say that if a parallel application has to be optimized, the analyst will need anyhow the source code, and knowledge about it, in order to implement

[a] time stamped sequence of events

the optimizations. These two arguments do not take into account the emerging reality of the academic and industrial supercomputing centres. These centres have to deal with many application codes and the analyst who elaborates the performance reports is not, in many cases, the same person who will optimize the code. In that context, our tool makes possible obtaining a very fast report of the general characteristics of the application's execution and, for that reason, makes easier the work of the performance analyst. Obviously, in the final stages of the performance optimization process, strong knowledge about the source code will be needed.

There are other approaches to automatically detect program phases. In[6,7] the authors perform an analysis based on Basic Bloc Vectors (BBV), which are sections of code. The frequency with which that sections of code are executed is the metric to compare different sections of the application's execution. In[8] the authors use subroutines to identify program phases. If the time spent in a subroutine is greater than a threshold, it is considered an important phase. There is another approach in[9] where the authors define a program phase as the set of instructions touched in a given interval of time. There are many differences between these approaches and our. First, in this paper we analyze the execution of the application using signal processing techniques. Second, the approaches we talked above are focused on solve problems such us reduction of simulation time or study of energy consumption. Our approach is focused on the study of the performance of the application. This paper is organized as follows: First, in Section 2 there is a description of the signal that we work with and an explanation of Wavelet Analysis. After that, in Section 3 there is a characterization of execution phases of high performance computing applications and an example. In Section 4 we show results obtained from the study of several applications. Finally, Section 5 is dedicated to conclusions.

2 Signal Processing

2.1 Signal Definition

Our approach is based on tracefile techniques, since such techniques allow a very detailed study of the variations on space (set of processes) and time that could affect notably the performance of the application. On our work, we will use OMPItrace tracing package[10] during the process of obtaining tracefiles. In order to apply signal processing techniques, we derive a signal from the data contained in the tracefile. More exactly, the signal we will work with will be the sum of durations of running bursts. This signal is generated taking into account the running states contained in the tracefile. For all of the threads, we generate a signal which contains, for each instant of time t, the duration of the running burst that is being executed in this moment t. If there is no running burst in that moment, the value assigned to t is 0. Finally, the signals generated for each of the threads are added. Therefore, we obtain the sum of durations of running bursts. In Fig. 1 there is an example in which a sum of durations of running bursts signal is generated.

2.2 Overview of Wavelet Transform

Wavelet transform is a well known signal processing tool used in a wide and heterogeneous range of topics. Specifically, this transform and the techniques based on it have

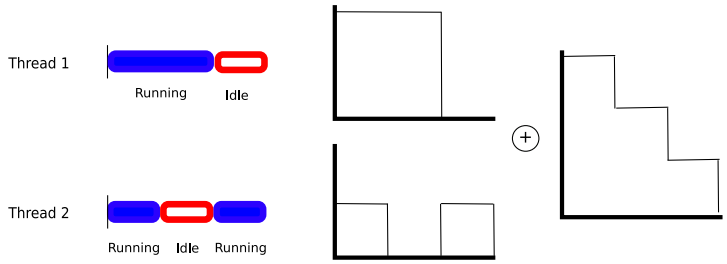

Figure 1. Generation of sum of durations of running bursts signal from a simple running burst distribution

become useful for relevant issues such as elimination of signal noises or image compression[3]. However, they have not been widely used in the study of execution of applications. One of the few examples of its utilization in this topic is in10. When we talk about signal processing, Fourier transform is the most famous technique to study the signal from the frequency domain point of view. Wavelet and Fourier transforms are related in the fact that they analyze the function in its frequency domain. However, there are two key differences between Fourier and Wavelet transforms: First, Fourier transform uses sinusoid functions to study the input signal while Wavelet transform uses a pair of analysis functions: the scaling function and the wavelet function. The first interprets low-frequency information and the second captures high-frequency information. The second difference is prompted by the fact that Fourier analysis of a signal only gives information about the frequencies while wavelet analysis, which gives information about the frequencies that appear in the signal, determines also where these frequencies are located.

The characteristics of the Wavelet transform we have explained make it specially appropriate for detect execution phases on MPI applications for several reasons: First, because the time-frequency localization is fundamental in order to overcome the non-stationary behaviour of the programs. If a given application changes its execution behaviour suddenly and starts to perform high-frequency iterations, Wavelet transform will detect this change.

Second, wavelet analysis allows one to choose the analysis function that best captures the variations shown by signals generated from tracefiles of MPI applications. We use the Haar Wavelet Function[2] because it is attuned to the discontinuities that often appear in the signals we work with.

To provide a more precise understanding of wavelet transform and a concrete explanation of the role of scaling and wavelet functions, we show the mathematical formulation of wavelet transform. First, consider a division of the temporal domain of the signal $x(t)$ in 2^N partitions:

$$a_k = \int_{-\infty}^{\infty} 2^{\frac{j_0}{2}} \phi(2^{j_0}t - k)x(t)dt \text{ where } 0 \leq k < 2^{j_0} \tag{2.1}$$

$$b_{j,k} = \int_{-\infty}^{\infty} 2^{j} \psi(2^{j}t - k)x(t)dt \text{ where } 0 \leq k < 2^{j} \text{ and } j_0 \leq j < N \tag{2.2}$$

The scaling function, $\phi(t)$ is used to obtain the coefficients a_k. We have one of these coefficients for everyone of the time regions we have divided the domain of the original

signal, $x(t)$. Their value is an average of $x(t)$ over a window of size 2^{j_0}. The value of j_0 allows us to define a maximum granularity of wavelet analysis. On the other hand, coefficients $b_{j,k}$ capture higher frequency behaviours. They are indexed by two variables: j, which corresponds to frequency, and k, which corresponds to time.

From the previous presentation of wavelet analysis, we can derive an interesting feature of it. As we increase the value of j, we obtain more coefficients, 2^j, for the same time domain, that is, the accuracy of the time resolution increases too. However, the accuracy of the frequency value decreases since the range of frequencies we are trying to detect increases as we increase j. In summary, this multi-resolution analysis provides a good time resolution at high frequencies, and good frequency resolution at low frequencies. For this reason, our analysis will be focused on the detection of phases of high frequency behaviour during the application's execution. Wavelet analysis warrants a good accuracy in this kind of detection.

Finally, from the computational point of view, the Discrete Wavelet Transform (DWT) has an implementation called FastWavelet Transform(FWT)[1]. This implementation has an algorithmic complexity of O(n). This property allows allow us to find different execution phases in a tracefile with only O(n) operations.

3 Execution Phases

Typically, high performance computing applications have severalwell defined phases. First of all, the execution has an initialization phase characterized by initial communications in order to assign values to the variables, domain decomposition, etc... This phase has not a periodic structure, that is, the impact of high frequencies is negligible on it.

Second, HPC applications have an intensive computation phase. It has typically a periodic behaviour prompted by the usual application's structure, which performs iterations on space (physical domain of the problem) and time interleavedwith communication (point to point or collectives). It is in this second phase where the most parallelizable code is being executed, since there are no input/output operations. Finally, there is a final phase phase. It consists, mainly, in output data operations. As well as the initialization, this phase has not a periodic structure.

These three phases constitute a common execution of HPC application. Wavelet transform is specially indicated to detect them because the differences between the three phases have an impact in the frequency domain. The good accuracy of Wavelet analysis detecting the places where high frequencies occur enables it to detect accurately the localization of the computation phase.

However, there are non-typical applications which do not follow the basic HPC application structure. In these cases, the Wavelet analysis will identify the ad-hoc structure if the differences between phases could be expressed from the frequency domain point of view, that is, if the phases could be characterized by occurrence of high-frequencies or low-frequencies.

3.1 Example

In this section, we discuss in detail results obtained applying wavelet analysis to a real application in order to give an overview of the information provided by this analysis. The

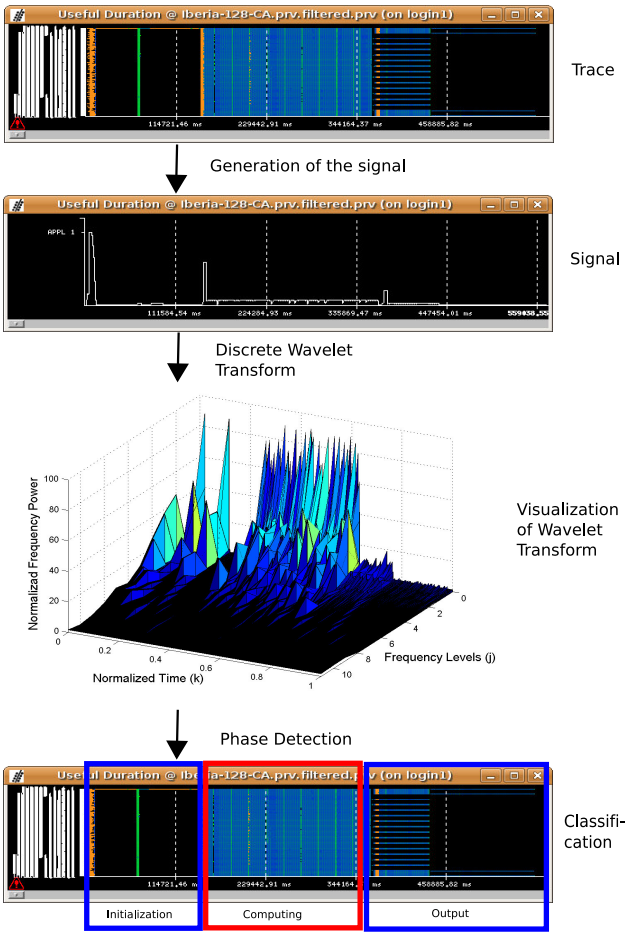

Figure 2. Example of Wavelet Analysis applied on WRF-NMM tracefile. First, running bursts distribution generated from the tracefile. After, sum of durations of running bursts signal generated from the distribution. In the third cell, numerical results of the wavelet analysis and, finally, phase detection derived from wavelet analysis.

application selected is WRF-NMM[11] over a 4 km grid of the Iberian Peninsula with 128 processors.

First of all, application has been traced. In Fig. 2 we see, on the top, a visualization of the running burst distribution contained in the tracefile. It shows clearly the typical structure derived from the execution of a HPC application: First, an initialization phase with non-periodic structure, second, a computation phase with strong periodic behaviour and, finally, an output phase. In Fig. 2, we also show the signal that has been generated from the tracefile using the methodology explained in Section 2.1. In this signal it is possible again to see the typical structure of HPC application. Wavelet analysis has been applied to the signal in order to detect execution phases. Numerical output of wavelet analysis is shown at the third picture of Fig. 2: First, in the x-axis, we draw the normalized

time of the execution of the application. In the y-axis we show the Frequency Levels. Each of these levels corresponds to a set of $b_{j,k}$ coefficients defined in 2.1 with the same j. In the Level 0 axe we show the coefficients which are sensitive to high frequencies. As the Level number increases, the sensitivity to high frequencies decreases and the sensitivity to low frequencies grows. Finally, in the Level 10 axe, we show the value of the coefficient a_k, defined in 2.2. In order to give a more intuitive visualization of the sense of the results, the values of the coefficients have been normalized by the window's size used in every level.

In the fourth cell of Fig. 2 we show the phase classification, based on Wavelet analysis results, that automatically our system has made. This classification corresponds to the division that we can do intuitively. It has been done without any knowledge of the source code and without the need of a visualization of the tracefile.

The most important thing to take into account from Fig. 2 is that wavelet analysis has been able to detect the regions of the tracefile where high frequencies occur, that is, regions where strong periodic behaviour characterizes the execution of the application.

4 Results

We have applied Wavelet Analysis to the Weather Research and Forecasting system. We have analyzed the Nonhydrostatic Mesoscale Model (NMM)[11] core and the Applied Research Weather (ARW)[12] one. Each core has been executed with 128, 256 and 512 processors. These executions have been made using, first, a 4 kilometers grid of the Iberian Peninsula and, second, a 12 kilometers grid of Europe. In Table 1 the results generated from WRF-NMM application for Iberia are shown. It is clear that the scalability of the application is severely affected by the fact that the initialization phase not only does not scale well, it also increases the time span required to be carried out. On the other hand, computation phase and output one show a better behaviour. They are far, however, to reach lineal scalability.

In Table 1 the results obtained from WRF-NMM application for Europe are shown, that is, the same code we have used to generate results from WRF-NMM-Iberia but different input data. The behaviour of the execution is similar to the Iberia case, that is, time span required to carry out the initialization does not decrease as the number of processor increases. On the other hand, computation phase and output phase show a decreasing span of time but far to lineal.

In Table 2 the results generated from WRF-ARW application for Iberia are shown, that is, the same input data used to generate Table 1 but different code. We see as WRF-ARW is faster than WRF-NMM but shows a worst scalability. This fact is explained by the poor scalability shown by the computation phase of the WRF-ARW core. This is not a trivial issue: Indicates that the resolution of the same physical problem could be solved faster by WRF-ARW if we have several hundreds of processors but WRF-NMM could be better if we can perform the execution on several thousands of processors. We highlight the fact that all this information is generated automatically without the need of knowledge of the source code.

Finally, in Table 2 the results obtained from WRF-ARW application for Europe are shown. The same behaviour as we have seen in Iberian case is detected.

Table 1. WRF-NMM. Time spans are expressed in microseconds.

Number of Processors	Initialization	Computation	Output	Elapsed Time
WRF-NMM Iberian Peninsula				
128	145338	226795	199179	571312
256	167984	143703	140392	452079
512	212785	88102	91359	392246
WRF-NMM Europe				
128	209953	203590	206924	620467
256	155502	134578	130064	420144
512	203207	90168	93760	387135

Table 2. WRF-ARW. Time spans are expressed in microseconds.

Number of Processors	Initialization	Computation	Output	Elapsed Time
WRF-ARW Iberian Peninsula				
128	140363	102442	93412	336217
256	153783	73624	63674	291081
512	189981	62973	57071	310025
WRF-ARW Europe				
128	126081	95575	86108	307764
256	141139	69705	61591	272435
512	181811	60603	54515	296929

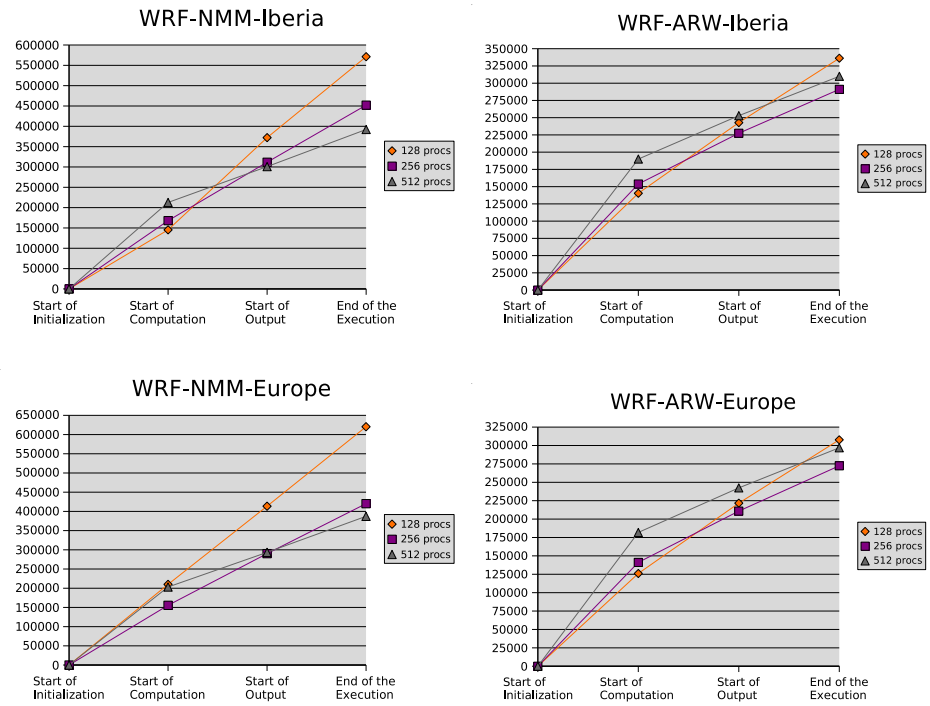

Figure 3. Representation, in microseconds, of the accumulated sum of the results shown in tables 1 and 2. In the Start of Computation axe, time span of initialization is shown. In the Start of Output axe, the sum of Initialization and Computation time spans is shown. Finally, in the End of Execution axe, the total elapsed time is shown.

5 Conclusions and Future Work

In this paper, we have shown clearly the relevance of automatic phase detection of MPI applications in order to obtain automatically non-trivial information. Specifically, to study the scalability of applications, we have shown how automatic phase detection allows the analyst to detect several execution regions and determine which of these regions are undermining the global execution scalability.

This automatic detection is performed using Wavelet analysis, which is a well known signal processing tool used in a wide and heterogeneous range of topics but very few used to study the execution of applications. A very important feature of Wavelet analysis is that it can be carried out in O(n) operations. In the future, we will use the methodologies explained in[16] to study the periodicities of the computing phase and to extract the length of the iterations. Using these methodologies we will also extract a representative set of iterations of the computing phase. After that, an analytical model of the scalability will be applied to this set of iterations. The final goal is generate automatically a complete report that guides the programmer to improve application's performance.

References

1. S. G. Mallat, *A Wavelet Tour on Signal Processing*, (Elsevier, 1999).
2. I. Daubechies, *The Wavelet Transform, time-frequency localization and signal analysis*, IEEE Transactions on Information Theory, (1990).
3. C. S. Burrus, R. A. Gopinath and H. Guo,*Introduction to Wavelets and Wavelet Transform*, (Prentice-Hall, 1998).
4. A. Knuepfer, H. Brunst and W. E. Nagel, *High Performance Event Trace Visualization*, in: Proc. PDP 2005, pp. 258–263.
5. H. Brunst, D. Kranzlmuller and W. E. Nagel, *Tools for Scalable Parallel Program Analysis - Vampir VNG and DeWiz*, DAPSYS, pp. 93–102, (2004)
6. T. Sherwood, E. Perelman, G. Hamerly and B. Calder, *Automatically Characterizing Large Scale Program Behavior*, in: Proc. 10th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 45–57, (2002).
7. T. Sherwood, E. Perelman, G. Hamerly, S. Sair and B. Calder, *Discovering and Exploiting Program Phases*, in: IEEE Micro: Micro's Top Picks from Computer Architecture Conferences pp. 84–93, (2003).
8. M. Huang, J. Renau and J. Torrellas, *Positional adaptation of processors: application to energy reduction*, in: Proc. the 30th Annual Intl. Sym. on Computer Architecture, pp. 157–168, (2003).
9. J. E. Smith and A. S. Dhodapkar *Dynamic microarchitecture adaptation via co-designed virtual machines*, in: Intl. Solid State Circuits Conference 2002, pp. 198–199, (2002).
10. OMPItrace manual `www.cepba.upc.es/paraver/docs/OMPItrace.pdf`
11. Nonhydrostatic Mesoscale Model (NMM) core of the Weather Research and Forecasting (WRF) system. `http://www.dtcenter.org/wrf-nmm/users/`
12. Advanced Research WRF (ARW) core of the Weather Research and Forecasting system. `http://www.mmm.ucar.edu/wrf/users/`
13. Plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics. `http://www.cpmd.org/`
14. Dimemas: performance prediction for message passing applications, `http://www.cepba.upc.es/Dimemas/`
15. Paraver: performance visualization and analysis, `http://www.cepba.upc.es/Paraver/`
16. M. Casas, R. M. Badia and J. Labarta,*Automatic Extraction of Structure of MPI Applications Tracefiles*, in: Proc Euro-Par 2007, pp. 3–12 (2007).