Automatic detection of parallel applications computation phases*

Juan Gonzalez, Judit Gimenez and Jesus Labarta

Barcelona Supercomputing Center - Universistat Politècnica de Catalunya
Barcelona, Spain

{juan.gonzalez, judit.gimenez, jesus.labarta}@bsc.es

Abstract. Analyzing parallel programs has become increasingly difficult due to the immense amount of information collected on large systems. The use of clustering techniques has been proposed to analyze applications. However, while the objective of previous works is focused on identifying groups of processes with similar characteristics, we target a much finer granularity in the application behavior.
In this paper, we present a tool that automatically characterizes the different computation regions between communication primitives in message-passing applications. This study shows how some of the clustering algorithms which may be applicable at a coarse grain are no longer adequate at this level. Density-based clustering algorithms applied to the performance counters offered by modern processors are more appropriate in this context. This tool automatically generates accurate displays of the structure of the application as well as detailed reports on a broad range of metrics for each individual region detected.

1 Introduction

Nowadays, parallel applications are able to run with ever-growing number of processes. In terms of analysis, bigger applications demand ever more complexity of the analysis task. This difficulty comes from the huge amount of data to evaluate: high number of processes, high numbers performance metrics, etc. An interesting way to identify the most important information is to use data mining techniques. We have chosen clustering because it is a simple but powerful technique which summarizes the different information available to analyze the performance of these applications.

In previous studies, [14,12,2,7], clustering has been used in coarse grain scenarios. Basically, in these works, the authors try to summarize global information extracted during the execution of parallel applications, in order to group those processors which demonstrate similar behavior. The work done in [15] is totally different: the authors use clustering algorithms to detect different phases of applications, using lowest-level statistics regarding the use of basic blocks as input data.

* This work has been supported by the Ministry of Education of Spain under grant BES-2005-7919 and the IBM/BSC Mare Incognito project

In this paper, we present an intermediate approach. Our goal is to use clustering techniques in order to outline the different trends of the CPU computation areas of parallel applications. In a message-passing parallel application, we consider a CPU computation area (or CPU burst) to be the region between two consecutive communications, for any given task. Each CPU burst is characterized by its duration and a set of performance counters. We consider that this is a much finer characterization of the applications than the ones used in [14,12,2,7] but coarser than the characterization done in [15] and it is a valuable aid to the developer during the performance analysis stage.

Summarizing, the main contribution of this paper is **the application of clustering techniques to automatically detect parallel applications structure**. Applying clustering algorithms over performance hardware counters allow us to detect the different sections or phases that characterize a message-passing parallel application. This is a useful method to provide the developer a space-temporal structural description of his application with statistical summaries of performance metrics for the different regions identified. With these summaries the developer can quickly distinguish the regions which suffer bad sequential performance.

The rest of the paper is organized as follows: Section 2 presents the environment where this study was carried out. Section 3 shows the details about the data used in this study, as well as the pre-processing techniques applied to this data. In Section 4 the clustering algorithms used to detect the application structure is described. Finally, Section 5 shows the experimental validation of the proposed techniques.

1.1 Related work

Nickolayev et al. [12], based on [14], propose the application of clustering techniques in real-time analysis. In their paper the classic K-means algorithm is used to aggregate the analysis data extracted on each processor during the program execution. Ahn and Vetter [2] propose the use of hierarchical clustering and K-means clustering. Principal Components Analysis (PCA) and F-Ratio analysis is also used in this study to reduce the dimensionality of the collected data. In both works, the summarized data always tries to differentiate the behavior of the processes of a parallel application, i.e. separate master and worker threads/tasks. The main difference between them is the collected data itself: in [12] high level metrics such as processor idle or running times are used and in [2] the metrics used are the processor performance counters. Another important difference between these studies is data acquisition: while in [12] the metrics are read every a certain time t, in [2] source code is manually bracketed with directives to perform the performance counters read.

In [7], Huck et al. describe a framework called PerfExplorer integrated with the performance analysis database PerfDMF. The aim of this framework is to develop data mining algorithms in order to extract useful information from the large amount of performance data. This data is stored in PerfDMF, and includes a huge range of metrics from high level (such as idle or running times) to low level (such as performance counters metrics), extracted for every application

subroutine. In addition, the database can handle different experiments for a given application (for example, executions with different number of processors). The algorithms evaluated to show the potential of PerfExplorer are essentially the same as those used in the other works, namely: K-means, hierarchical clustering and PCA analysis.

Sherwood et al. [15] developed an approach similar to the work presented in this paper but in a much finer level of granularity. In this study, the author applies clustering (concretely, K-means) to capture the similarity between Basic Block Vectors (BBV). The BBVs contain the count of how many times a basic block of the whole program has been executed during an interval of 100 million instructions. The similarity measure is based on the distance (Manhattan or Euclidean) between BBVs. This study demonstrates how the K-means clustering algorithm is able to detect different phases in sequential applications, such as those present in the SPEC2000 benchmark.

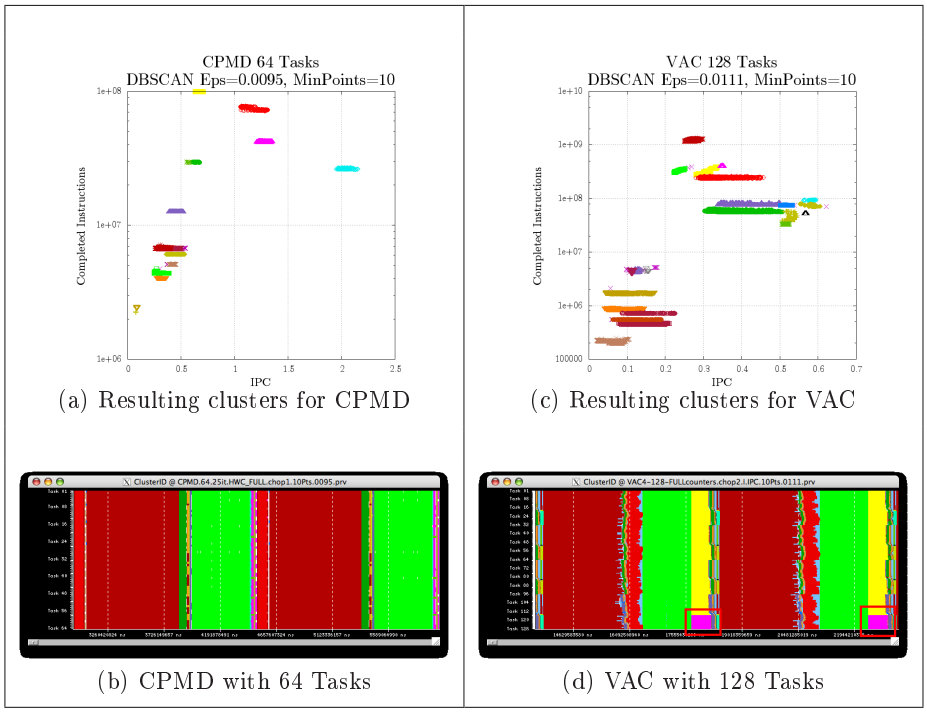
2 Clustering framework

The present study was carried out using the CEPBA-Tools [1] suite, a set of trace-based tools which focus on the analysis of parallel applications. The most important tools in this suite are the Paraver trace analyzer and the Dimemas trace-driven simulator. Although both tools use different traces, they can easily inter-operate, generating input traces for each other. The MPItrace library is used to obtain the application trace. This tracing library is able to intercept MPI calls. In addition, the library can read the values of the selected performance counters and shift between different counters groups using the PAPI library [1].

As a result of this study, we developed a tool that performs the computation structure detection. This tool has two inputs, the Dimemas trace with the application information to be analyzed and a XML document where the clustering process and the data pre-processing are defined. As can be seen in the Figure, the tool produces three different outputs: 1) the Dimemas trace where every CPU burst is labeled with the cluster to whom it belongs to, 2) a GNUPlot script that shows a scatter plot with two of the metrics used in the clustering process; 3) a report file with relevant cluster metrics and a counter extrapolation (not explained in this paper) per cluster, when the trace contains the required counter information.

Clustering can be applied in other scenarios, as in [12] where it is applied during program execution, but we have selected a trace-based scenario because it provides a mechanism to check the suitability and the results obtained applying different clustering algorithms: the trace contains the whole information needed by the algorithms to compute the clusters; is easy to add the cluster information back to the original trace; and, last but not least, the results can be graphically displayed over the time-line for a qualitative evaluation.

[1] `http://icl.cs.utk.edu/papi/`



(a) Resulting clusters for CPMD

(c) Resulting clusters for VAC

(b) CPMD with 64 Tasks

(d) VAC with 128 Tasks

Fig. 1. Results of structure detection using DBSCAN clustering algorithm for the applications CPMD, (a) and (b), and VAC, (b) and (d). The algorithm was applied to Completed Instructions and IPC. The clustering algorithm has been able to correctly detect the SPMD structure of the applications. The section inside the red box in (b) shows the ability of clustering to also detect an unbalanced region.

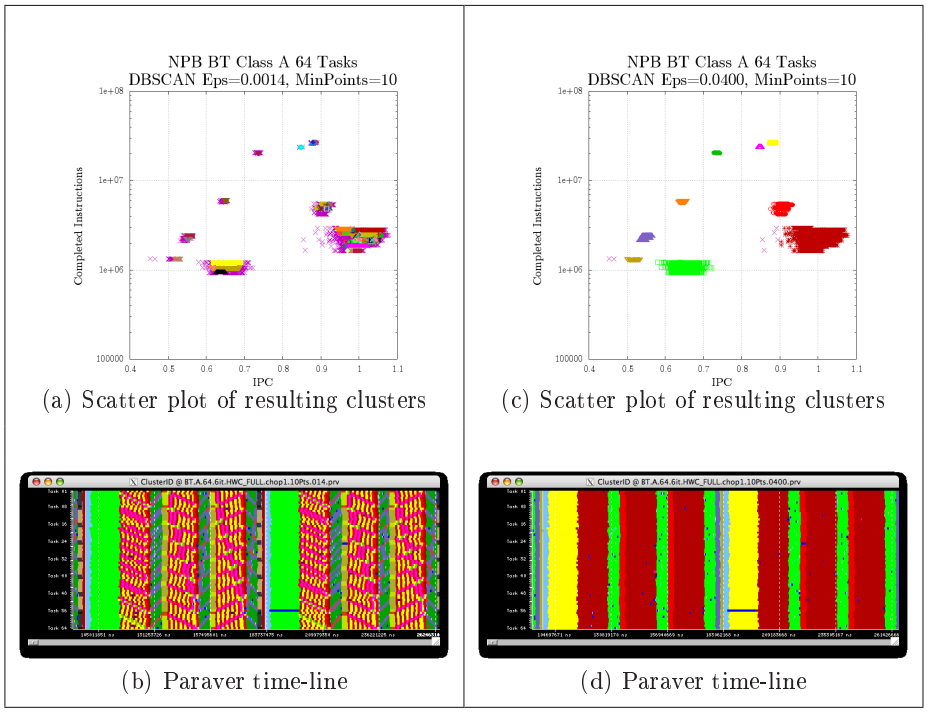
2.1 Analyzed applications

The message-passing parallel applications used in our experiments and presented in this paper are:

Car-Parinello Molecular Dynamics (CPMD) [8]. The CPMD code is a parallelized plane wave / pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics.

Versatile Advection Code (VAC) [18]. The Versatile Advection Code is a general tool for solving hydrodynamical and magnetohydrodynamical problems arising in astrophysics.

NAS Parallel Benchmarks (NPB) BT [4]. One of the well-known NAS Parallel Benchmarks that computes a finite difference solution to the 3D compressible Navier-Stokes equations. In the experiments, we used the class A variant of this benchmark.



(a) Scatter plot of resulting clusters

(c) Scatter plot of resulting clusters

(b) Paraver time-line

(d) Paraver time-line

Fig. 2. Different clusterings of NPB BT class A Benchmark using DBSCAN clustering algorithm over Completed Instructions and IPC. Figures (a) and (b) show a finer detail of application structure due to the lower value of Eps. In Figures (c) and (d) the higher value of Eps show a coarser phase detection.

Weather Research Forecast (WRF) Model [11]. WRF is a mesoscale numerical weather prediction system designed to serve both operational forecasting and atmospheric research needs. In the experiments, we used the non-hydrostatic mesoscale model (WRF-NMM) [10] dynamical core.

Spanish Initiative for Electronic Simulations with Thousands of Atoms (SIESTA) [16]. SIESTA is both a method and its computer program implementation, to perform electronic structure calculations and ab initio molecular dynamics simulations of molecules and solids.

3 Input data

The data used to characterize the application is the hardware counters provided by modern processors. The performance counters are read when each CPU burst finishes, in other words, just before a communication is executed. In our tests we used the IBM PowerPC® 970MP processor. In this processor up to 8 different counters can be read simultaneously. However, the processor design [9]

fixes the combinations of the counters that can be read at the same time. Unless otherwise stated, the counters used in the experiments were: Processor Cycles, Completed Instructions, Dispatched Instructions, Data Loaded From Memory, L1 Data Cache Load Miss, Global Completion Table Empty Cycles, Store Instructions and Load Instructions.

3.1 Data pre-processing

Data pre-processing is the first step to reduce the volume of the clustering algorithm input data. We applied two different techniques: first, filtering CPU bursts with small durations, and, second, data normalization.

Filtering. Filtering simply consists of discarding those bursts whose duration is negligible in the application execution. In this way, a vast amount of irrelevant data is directly discarded when applying the clustering algorithm. In the applications tested, the filtering process has been able to discard up to 80% of processed bursts maintaining the 99% of application time.

Normalization. Normalization is applied in order to ensure that when using counters with different data ranges, none of them bias the clustering results. Two different normalizations methods are used. First, logarithmic normalization is used when the dynamic range of the performance metric is large. Reducing this dynamic range guarantees that the results of clustering are not displaced to the higher values of a counter. Additionally, range normalization, simply scaling the data to $[0, 1]$ range ($\forall a_i \in A, a_i \leftarrow (a_i - min(A))/(max(A) - min(A))$), ensures that all factors have a similar weight in the multi-dimensional clustering

3.2 Dimensionality reduction

A common problem when applying clustering algorithms is related to the dimensionality of the data. With the performance counters data we have 8 different counters for each CPU burst. Our proposal to address this problem is to reduce the dimensionality by selecting counters or derived metrics with "physical" meaning to the analyst. In our experiments, two different groups of the available metrics were used:

- Processor Cycles combined with IPC. This combination focuses the clustering on the "performance view" of the application.
- Completed Instructions , L1 and L2 cache misses. This combination reflects the impact of the architecture on the application structure, via the cache misses counters.

These two combinations are useful to detect regions with different computational complexity (Instructions Completed), and at the same time to differentiate between regions with the same complexity but different performance.

In [2] and [7], Principal Components Analysis (PCA) is used to reduce the dimensionality of the data. Using PCA, the dimensions are reduced by creating a

new space with a lower number of dimensions, that are the principal components of the original data. Then each point is projected to this new space, and the clustering algorithm is applied to the transformed set of points. This technique succeeds in reducing the dimensionality, as an aid to the clustering algorithm. In our experiments, we tested this technique obtaining similar results to our manual selection of attributes. We conclude that using PCA is not strictly necessary in this scenario because it adds no significant benefits in the clustering results.

4 Density-based algorithms and DBSCAN

From the wide variety of algorithms ([5,19]), we initially looked at X-means [13] (a simple improvement over K-means that approximates the value of k). K-means-like algorithms always suppose a Gaussian model of the data. However, the performance hardware counters data is not distributed following this Gaussian model so we had to use another kind of clustering algorithms. We choose density-based clustering as the best way to reach our goal. The main point of these clustering algorithms is that they do not make any assumption about the data structure or model. They try to group points into the space where their density is big enough to be considered as a real cluster. In addition, these algorithms are robust against outliers and noise.

The selected algorithm was DBSCAN [6]. It is a simple approach to density-based clustering and easily incorporates the benefits of this family of clustering algorithms. The input of DBSCAN are two parameters, the radius Epsilon (Eps) and minimum number of points ($MinPoints$) plus the data itself. The resulting clusters obtained are those subsets C_i of the data that fulfill the following[2]:

1. For any given pair of points $p \in C_i$ and $q \in C_i$ it is possible to find a set of points $a_1, a_2, ..., a_{n-1}, a_n \in C_i$, being $p = a_1$ and $q = a_n$, where the Euclidean distance for each pair a_i, a_{i+1} is less or equal to Eps. This property is called *density reachability*.
2. $|C_i| \geq MinPoints$. This is the minimum density condition to consider C_i as a cluster.

The technique applied for parameter selection is also described in [6] and it consists of generating a histogram with the sorted k-neighbor distance, being k the desired value of MinPoints. Then this distance is sorted (descending) and plotted. The histogram will show a descending curve. In [6], the authors suggest that the optimum value of Eps is the distance where the curve makes its first inflexion (or "valley"). The points located on the left of this "valley" will be noise in the resulting partition and the rest will be present on one cluster. In [6], the authors ensure that choosing 4 as the default value of $MinPoints$ produces the best results in 2-dimensional clusterings. In our experiments higher values, usually 10, obtained a better characterization of applications structure.

[2] These definitions are not exactly the same as those found in [6], but easily show the algorithm basis

4.1 DBSCAN suitability

Figure 3 shows the result of applying X-means and DBSCAN to the same trace, a section of CPMD with 128 tasks, using L1 vs. L2 cache misses. The scatter plots show some clouds of points that have a spherical shape, but others can be elliptical with different principal component directions. In this case, X-means tends to partition the ellipses. The red box marks a region with a strong vertical component where X-means, 3(a), detected two clusters, but DBSCAN, 3(b), detected only one cluster. The blue box shows another equivalent region where X-means also detected two clusters and DBSCAN only one, in this case having a strong horizontal component. We consider the homogeneous shape detection done by DBSCAN to be better than the X-means cluster assignment. As can be seen in next point, these isolated groups correspond to fixed regions that define the structure of the application.

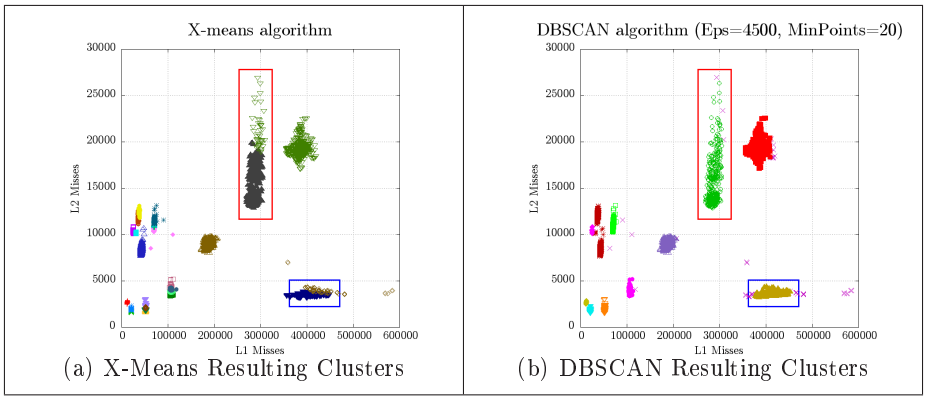
4.2 Clustering results

Figures 1 and 2 present the results of four different clusterizations obtained for the applications CPMD and VAC and the NPB BT benchmark. These Figures serve as example of how DBSCAN is able to detect the application structure in different ways.

In Figure 1, the algorithm was applied using Completed Instructions and IPC to CPMD and VAC. The scatter plots, 1(a) and 1(c), in both cases show a good detection of the different groups. Figures 1(b) and 1(d) are the time-line reconstruction of each application. In these time-lines the y axis represent the parallel tasks (or processors) and the time evolution on the x axis. The color represent which cluster is executed by each task in a given point of time. These Figures confirm that the repetitive structure of this SPMD applications were detected correctly. In Figure 1(d), the red box marks a region where some tasks were assigned to a different cluster than the general trend. This example shows the ability of DBSCAN to detect potential unbalanced regions in terms of the clustering dimensions.

Figure 2 shows two different clusterings of NPB BT class A benchmark, with 64 tasks. Figures 2(a) and 2(b) present the results using a small value of *Epsilon* (0.0014). In this case the number of obtained clusters is high, as can be seen in 2(a). Figure 2(b) shows how the application structure is detected in fine detail: for example, we can observe a staggered computation pattern (similar to a pipeline) between different tasks.

The clustering shown in Figures 2(c) and 2(d) presents a different approach. Using a higher *Epsilon* value (0.0400) we obtained a small number of clusters. In this case, the time-line, 2(d), shows the detected application structure at a coarser granularity, and the typical SPMD structure appears. As a whole, Figure 2 shows how different levels of detail can be obtained. This hierarchical characterization of a program is highly interesting from the performance analyst point of view.



(a) X-Means Resulting Clusters (b) DBSCAN Resulting Clusters

Fig. 3. X-means and DBSCAN algorithms comparison. Red and blue boxes remark clouds of points with strong components, vertical and horizontal, where X-means has divided the isolated group.

4.3 Clustering quality evaluation

In previous studies in this area, the output of the clustering algorithm applied is assumed to be correct, and no other analysis of the clusters is done. The quality of the clustering is assessed informally by checking that it displays some of the expected differences between processes. An exception is found in [15]. where the authors use the Bayesian Information Criterion (BIC), as is defined in the X-means algorithm [13], to evaluate the quality of the clustering results. This measure is closely related to the assumption of a Gaussian model of the clusters.

Due to the no data structure assumption of density-based clustering algorithms, the most common method to evaluate the clustering results is the "expert criterion" [3] (also known as *gold standard* [17]). This method simply consists in comparing the partition made by the clustering algorithm with a decomposition of the data made by an expert. We provide three complementary methods to perform the "expert validation" to check the quality of the resulting clusters:

1. Using the GNUPlot scatter plot to examine the cluster assigned to each point as a result of a good clustering, the scatter plot would show that isolated groups of points are detected as different clusters.
2. If the application is purely SPMD, we would expect the Paraver time-line to show that all processes execute the same cluster at the same time. In addition, we expect to find a repetitive pattern among clusters in the time-line, due to the common iterative structure of parallel codes.
3. If the original trace includes code linkage (user added events or back-traced caller events on MPI calls), a good clustering should show that every cluster region corresponds to a fixed sections of code.

	WRF	SIESTA	CPMD	VAC	NPB BT
Input Trace Size	118MB	67MB	91MB	53MB	69MB
Clustering Running Time	30.237s	1m 30.31s	51m 7.79s	1m 38.00s	12m 26.16s
Processed Bursts	13,917	11,234	53,454	25,733	24,850
Filter Threshold	$1000\mu s$	$1000\mu s$	$5000\mu s$	$1000\mu s$	$500\mu s$
% Bursts Discarded	88.70%	83.74%	34.03%	50.24%	64.26%
% Application Time Discarded	0.44%	0.33%	7.28%	0.08%	1.10%
# Clusters found	23	23	16	22	9
# Clusters with >10% of total time	3	3	2	2	3

Table 1. Clustering tool statistics for the applications used in this paper. The values regarding the NPB BT benchmark correspond to the execution with higher value of *Epsilon*.

5 Performance analysis experiments

Complementing the clustering results presented in Section 4.2, in this Section we present a deeper analysis of the clustering results for the applications WRF and SIESTA. These results are detailed in two different parts: **Results validation**, where we answer the question "is this clustering correct?"; and **Analysis**, where we answer the question "which information does the clustering provide?".

Table 1 shows an overview with some statistics of the clustering tool execution for the applications analyzed in this paper. The first fact to mention about this table is the high running time variability, from seconds to nearly an hour. This is caused by the sensitivity of DBSCAN to the data distribution as well as the input parameters. Another interesting fact presented on the table is the benefit of filtering stage, especially in WRF and SIESTA, where applying a $1000\mu s$ filter threshold we discarded up to the 88% of bursts in the trace, retaining 99% of the application total time.

We want to note that in the reports obtained by collecting the results of clustering application, Figures 4 and 5, only clusters that are interesting in the analysis process are shown, usually those which take up the biggest amount of application time.

5.1 WRF

The trace used in this analysis was extracted in an execution with 64 MPI tasks. The clustering report shown in Figure 4 was obtained after applying DBSCAN using Completed Instructions, L1 data cache misses and L2 data cache misses as clustering dimensions, with the parameters $Eps = 0.0083$ and $MinPoints = 10$.

Results validation. The time-line of four WRF internal time-steps, 4(a), shows clusters uniformly distributed along the program execution, as well as, the repetitive pattern of the application. In the 2D scatter plot of Completed Instructions vs. L1 data cache misses, 4(b), different groups were correctly as-

signed to different clusters. The actual separation between Clusters 1, 2 and 3 comes from the third variable used in the clustering, L2 data cache misses.

Table 4(c) shows an interesting result of this clustering: the correspondence of the four major clusters and the source code of the application, obtained with the back-trace information on each MPI library entry point. This table reflects that each cluster corresponds to 1 or 2 different regions of code, as we expected. In this experiment we found a good example regarding to the ability of clustering to detect the application structure. In Figure 4(a), vertical dotted lines are grouping Clusters 5 and 6, which only appear every two time-steps of the execution. Via the code correlation, Table 4(c), we checked that, effectively, the clusters correspond to two subroutines only executed on even time-steps (in detail, Cluster 5 corresponds to vertical passive advections and Cluster 6 to horizontal passive advections).

Analysis. Using the cluster statistics table, 4(d), the analyst could be focused on Clusters 1 and 2. These clusters take up the 55% of application execution time and their IPC is 0.53 and 0.50. In both cases, the values of L1 cache misses per 1000 instructions are high, 22.72 in Cluster 1 and 32.63 in Cluster 2. With this information, the data access pattern seems to be the main factor causing the degradation of the performance of these clusters. Comparing Clusters 1 to 4 we can see how similar L2 miss rates obtain different IPC depending on L1 miss ratio.

5.2 SIESTA

For this experiment, the SIESTA trace used was also obtained from an execution with 64 tasks. Figure 5 summarizes the results. With the collaboration of the application developers, we marked the source code with entry and exit events of main subroutines. In this experiment we used Completed Instructions and IPC metrics as DBSCAN input. The parameters used were $Eps = 0.0151$ and $MinPoints = 10$.

Results validation. The scatter plot, 5(b), shows a correct detection of isolated clouds as different clusters. Furthermore, it was able to even detect different groups having different components, as Cluster 3, which has a strong horizontal component (steady number of Completed Instructions and variable IPC) and Cluster 4, which has a strong vertical component (steady IPC but variable number of Completed Instructions.

In Figure 5(a) Window 1 shows the cluster information and Window 2 shows the subroutine information obtained using the manually added events. This Figure demonstrates how clustering detected exactly the most important routines in SIESTA code. Subroutines `rhoofd`, `vmat` and `cellxc` were detected as Cluster 2, Cluster 3 and Cluster 5, respectively. In `compute_dm` subroutine, the clustering detected two phases: an initial corresponding to Cluster 1 and a small region at the end of the subroutine, corresponding to Cluster 4. This detection was confirmed by the developers: `compute_dm` has an initial region of computation and finishes with a communication period where different tasks share the data.

Analysis. This application shows a slightly better results than WRF, in terms of IPC. Table 5(c) points out that Cluster 1 has the lower IPC (0.65).

This score As in the previous example, this low IPC shows to be related to the memory: Cluster 1has the highest values of L1 misses and Memory Bandwidth. In addition, it has the the second higher value of MFLOPS. The subroutine `compute_dm`, which corresponds to Cluster 1, solves an eigenvalues equation and intensively uses a sparse matrix, with indirect access to values, causing the detected memory behavior.

6 Conclusions and future work

This paper presented the suitability of DBSCAN [6] based on performance counters to characterize the internal structure of message-passing applications. Furthermore, we demonstrated the inability of other clustering algorithms such as K-means or AHT used in previous works [12,2,7].

The experiments performed with the applications NPB BT, CPMD, VAC, WRF-NMM and SIESTA, validate how our clustering technique is able to correctly detect different algorithm phases as well as regions of different subroutines with similar behavior. In addition, the experiments show the potential of the clustering, combined with other analysis tools such as Paraver, to help the analyst/developer in their work, focusing on a better understanding of the computation regions.

The first application of the structure detection we did was counter extrapolation. Due to page limitation, this technique is not shown in this paper. It simply consists on read different counters groups during the execution. Having these groups a common subset of counters, we apply the clustering algorithm to the common subset and extrapolate the value for the rest of counters for each cluster.

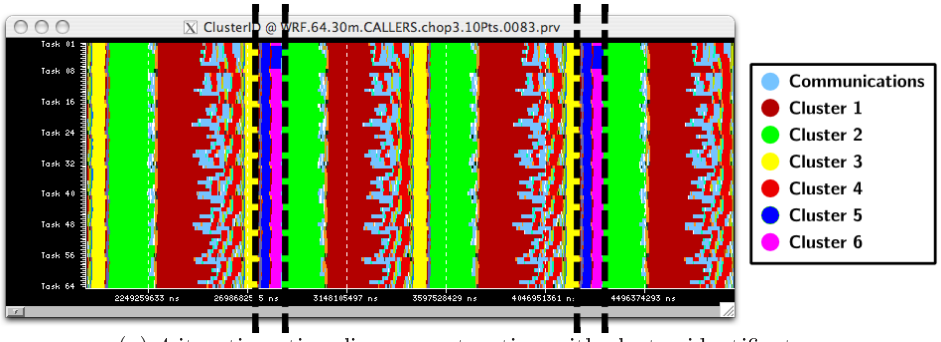
We are currently using clustering to automatically identify the most representative regions to be traced and simulated at instruction level. This simulation is done in order to evaluate the micro-architectural impact on the whole parallel application. Also related to performance prediction, our research is now oriented to applying the information provided by DBSCAN so as to make more precise performance predictions. The memory characterization provided by clustering (cache misses and bandwidth consumptions) will be used in Dimemas [1] to accurately simulate CPU bursts.

Furthermore, we are going to apply this automatic structure detection during the application execution, in order to obtain different "on-line" characterizations.

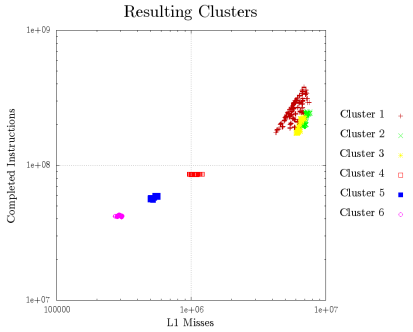
7 References

1. *CEPBA-Tools Team@BSC Home*, `http://www.bsc.es/plantillaF.php?cat_id=52`.
2. D. H. Ahn and J. S. Vetter, *Scalable analysis techniques for microprocessor performance counter metrics*, SC '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing (Los Alamitos, CA, USA), IEEE Computer Society Press, 2002, pp. 1–16.
3. Nicolas Anquetil, Cédric Fourrier, and Timothy C. Lethbridge, *Experiments with clustering as a software remodularization method*, WCRE '99: Proceedings of the

Sixth Working Conference on Reverse Engineering (Washington, DC, USA), IEEE Computer Society, 1999, p. 235.
4. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R.Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, *The NAS Parallel Benchmarks*, Tech. Report RNR-94-007, NASA Advanced Supercomputing (NAS) Division, 1994.
5. P. Berkhin, *Survey Of Clustering Data Mining Techniques*, Tech. report, Accrue Software, San Jose, CA, 2002.
6. M. Ester, Hans P. Kriegel, J. Sander, and X. Xu, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, Second International Conference on Knowledge Discovery and Data Mining (Portland, Oregon) (Evangelos Simoudis, Jiawei Han, and Usama Fayyad, eds.), AAAI Press, 1996, pp. 226–231.
7. K. A. Huck and A. D. Malony, *PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing*, SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing (Washington, DC, USA), IEEE Computer Society, 2005, p. 41.
8. J. Hutter and A. Curioni, *Car-Parrinello Molecular Dynamics on Massively Parallel Computers*, ChemPhysChem **6** (2005), 1788–1793.
9. IBM Systems and Technology Group, *IBM PowerPC 970MP RISC Microprocessor. User's Manual*, March 2007, Version 2.2.
10. Z. I. Janjic, Jr. J. P. Gerrity, and S. Nickovic, *An Alternative Approach to Nonhydrostatic Modeling*, Monthly Weather Review **129** (2001), 1164–1178.
11. J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, *The Weather Reseach and Forecast Model: Software Architecture and Performance*, Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology (Reading, UK), 25 - 29 October 2004 2004.
12. O. Y. Nickolayev, P. C. Roth, and D. A. Reed, *Real-Time Statistical Clustering for Event Trace Reduction*, The International Journal of Supercomputer Applications and High Performance Computing **11** (1997), no. 2, 144–159.
13. D. Pelleg and A. W. Moore, *X-means: Extending K-means with Efficient Estimation of the Number of Clusters*, ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 2000, pp. 727–734.
14. P. C. Roth, *ETRUSCA: Event Trace Reduction Using Statistical Data Clustering Analysis*, Master's thesis, University of Illinois at Urbana-Champaign, 1996.
15. T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, *Automatically characterizing large scale program behavior*, ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems (New York, NY, USA), ACM Press, 2002, pp. 45–57.
16. José M. Soler, Emilio Artacho, Julian D. Gale, Alberto García, Javier Junquera, Pablo Ordejón, and Daniel Sánchez-Portal, *The SIESTA method for ab initio order-N materials simulation*, Journal of Physics: Condensed Matter **14** (2002), 2745.
17. P. Tonella, F. Ricca, E. Pianta, and C. Girardi, *Evaluation methods for web application clustering*, Fifth IEEE International Workshop on Web Site Evolution, 2003, 2003, pp. 33–40.
18. Gábor Tóth, *Versatile advection code*, HPCN Europe '97: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking (London, UK), Springer-Verlag, 1997, pp. 253–262.
19. R. Xu and D. Wunsch II, *Survey of clustering algorithms*, IEEE Transactions on Neural Networks **16** (2005), 645–678.



(a) 4 iterations time-line reconstruction with cluster identificators



(b) Scatter plot of resulting clusters

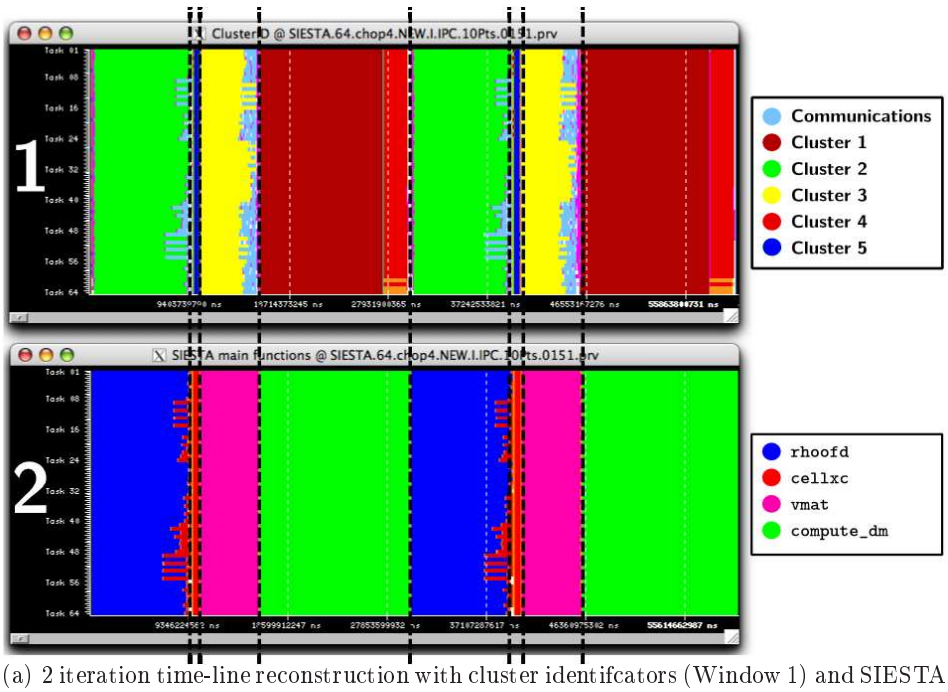
Cluster	Code Section
1	solve_nmm.f:[2037 - 2310]
2	solve_nmm.f:[1478 - 1782]
	solve_nmm.f:[2030 - 1782]
3	solve_nmm.f:[1241 - 1345]
4	solve_nmm.f:[2771 - 2865]
	solve_nmm.f:[2388 - 2489]
5	solve_nmm.f:[1478 - 1569]
6	solve_nmm.f:[1607 - 1633]

(c) Application code regions for each cluster

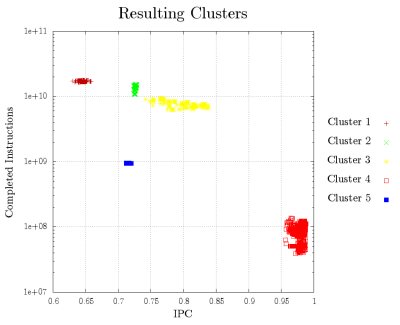
Cluster	1	2	3	4	5	6
% Time	36.29	29.52	10.13	9.68	3.73	1.71
Avg. Burst Dur. (ms)	220.46	177.70	60.81	29.09	38.71	44.83
IPC	0.53	0.50	0.62	0.77	0.66	0.59
MIPS	1210.07	1164.36	1403.19	1743.32	1499.47	1338.24
L1M/KInstr	22.72	32.63	12.65	8.39	16.12	6.86
L2M/KInstr	0.59	1.23	1.08	0.61	1.23	1.73
Mem.BW (MB/s)	90.77	182.65	193.32	136.33	236.15	295.71

(d) Individual cluster statistics

Fig. 4. Clustering results. WRF with 64 tasks (DBSCAN $Eps = 0.0083$, $MinPoints = 10$)



(a) 2 iteration time-line reconstruction with cluster identifcators (Window 1) and SIESTA main subroutines (Window 2)



(b) Scatter plot of resulting clusters

Cluster	1	2	3	4	5
%Time	43.37	32.07	15.81	4.92	2.20
Avg. Burst Dur. (ms)	11,794,11	8,524.79	4,201.32	39.22	584.35
IPC	0.65	0.73	0.80	0.98	0.72
MIPS	1460.45	1646.31	1805.65	2221.26	1621.42
MFLOPS	307.45	80.96	113.20	160.32	328.51
L1M/Kinstr	11.23	1.07	1.38	3.00	8.77
L2M/Kinstr	0.220	0.010	0.033	0.050	0.001
Mem.BW (MB/s)	41.21	2.03	7.68	14.22	0.29

(c) Individual cluster statistics

Fig. 5. Clustering results. SIESTA with 64 tasks (DBSCAN $Eps = 0.0151$, $MinPoints = 10$**)**