






- ORIGINAL ARTICLE -

# Methodology to Define a Static Allocation Mapping based on Memory Access Patterns and the Signature of MPI Applications in HPC Systems

Metodología para Definir una Política de Asignación Estática de Procesos basada en Patrones de Acceso a Memoria y la Firma de Aplicaciones MPI en Sistemas HPC

Gerard Enrique<sup>1</sup> , Eva Bruballa<sup>1</sup> , Remo Suppi<sup>2</sup> ,  
Alvaro Wong<sup>2</sup> , Emilio Luque<sup>2</sup>  and Dolores Rexachs<sup>2</sup> .

<sup>1</sup> *Escoles. Universitaries Gimbernat, Computer Science School, Universitat Autònoma de Barcelona, Spain*  
{gerard.enrique, eva.bruballa}@eug.es

<sup>2</sup> *Dept. Arquitectura de Computadors i Sistemes operatius, Universitat Autònoma de Barcelona, Spain*  
{remo.suppi, alvaro.wong, emilio.luque, dolores.rexachs}@uab.cat

## Abstract

Tools for identifying problems and improving MPI applications performance running on an HPC system require information from both the application and the system. In this work, we will focus on defining a methodology to analyze how memory usage affects an MPI application's performance running on an HPC system. This methodology will obtain valid and comparable data based on different memory access patterns, which will allow us to define key performance values used to characterize the HPC system behaviour facing these access patterns, as well as to characterize the Application Signature behaviour. This is obtained from Parallel Application Signatures for Performance Prediction (PAS2P) tool which obtains the representative phases of the MPI application, facing these same access patterns. With this methodology, we will be able to detect memory access application problems, suggest improvements and define a mapping policy for this application in this HPC system, in order to improve its performance and to determine limits to these improvements.

**Keywords:** Cache memory, HPC performance, MPI parallel applications, Process mapping.

## Resumen

Las herramientas para identificar problemas y mejorar el rendimiento de las aplicaciones MPI que se ejecutan en un sistema HPC requieren información de la aplicación y del sistema. En este trabajo nos centraremos en definir una metodología para analizar cómo el uso de la memoria afecta el rendimiento de

una aplicación MPI que se ejecuta en este sistema HPC. Esta metodología obtendrá datos válidos y comparables basados en diferentes patrones de acceso a la memoria que permitirán definir valores clave de rendimiento utilizados para caracterizar el comportamiento de un sistema HPC frente a estos patrones de acceso y para caracterizar el comportamiento de la Firma de la Aplicación, (obtenida de la herramienta Parallel Application Signatures for Performance Prediction (PAS2P) que obtiene las fases representativas de la aplicación MPI) frente a estos mismos patrones de acceso. Con esta metodología, podremos detectar problemas de la aplicación de acceso a la memoria, sugerir mejoras y definir una política de mapeo para esta aplicación en este sistema HPC para mejorar su rendimiento y determinar los límites de estas mejoras.

**Palabras claves:** aplicaciones paralelas MPI, mapeo de procesos, Memoria caché, rendimiento HPC.

## 1. Introduction

The wide diversity of parallel architectures and the constant evolution of processors, memory, storage, and interconnection systems all require programming techniques to adapt to parallel systems and improve their performance.

The increasing complexity of these systems produces a decoupling in the development of applications from the hardware. This results in avoiding a part of system complexity and transferring the efficient use of hardware resources to the parallel execution systems. Tools to improve application performance require information about the system and the running

application. This information is obtained through instrumentation tools for defining key parameters in order to identify problems that may suggest improvements in its development and to plan the machine resources in a way that we can optimize its performance.

The characterization of a complete MPI parallel application, considering the multiple factors that may affect its performance, takes considerable time. In this work, we propose a methodology to characterize the HPC system and model the relevant phases of the application, considering that these phases represent the complete application and require much less computation.

Parallel scientific applications are typically composed of a set of phases that are repeated throughout the application. These phases were written in the application code using specific communication and computation patterns. PAS2P [1] identifies the application phases transparently and automatically generates the Application Signature (PAS2P Signature). This contains only the most representative application phases, the phases that have an impact on the application performance, and their repetition rates (weights).

It is also important to characterize the HPC system on which we are going to run this application. Many aspects of the system architecture and organization will affect its performance, so we propose to characterize the system based on performance indices obtained from running a characterization program with different functional patterns. Each case pattern analyzed allows us to know which operations have more impact on system performance and how they condition its performance when they are executed sharing the Cache memory with each case pattern. This process only needs to be carried out once for each HPC system and the patterns to be analyzed can be extended or changed in order to attain a more accurate characterization.

Once we have the Application Signature, we will run the characterization program with each memory access pattern sharing the Cache memory with each process of each phase of the Application Signature. Thus, we obtain the performance values associated with the characterization program.

By comparing performance values obtained by sharing the Cache memory with the characterization program itself as well as with each process of the Signature phases, we will be able to determine which case patterns of the characterization program are more similar to each process of each phase. With this information, together with the performance values obtained by running the characterization program with itself, we can determine what would be the effects on performance due to the interaction between

the application signature sharing the Cache memory, and this information will allow us to define the mapping in this system in order to maximize its performance (see Fig. 1).

Other studies propose methodologies which allow us to define mapping policies by analyzing other factors that also affect performance, such as message passing [2],[3]. These process allocation policies can generate conflicts amongst themselves, so it will be necessary to redefine these policies to consider all these factors and minimize their effects to obtain a better overall performance [4],[5].

In the following section, we present related works. Section 3 presents the main steps of the methodology, and the last section presents conclusions and future work.

## 2. Previous and Related Works

Analyzing parallel applications to accurately predict their performance facing changes in the system, scaling, workload, or other aspects, are all increasingly complex. In addition, the time and resources required can be very high. It is especially important in those applications that run frequently to obtain an efficient use of resources, due to considerations of cost, power consumption or pay-per-use functions.

The information obtained will be essential to allocate resources to improve application performance, but it can also be valuable for programmers and the HPC system configuration.

Analyzing the application itself usually requires an excessive workload, especially in order to predict the performance of different systems.

Therefore, we propose a tool that allows this analysis to be carried out at a reduced cost and which is applicable to different systems. Scientific applications usually have highly repetitive behavior and parallel applications are not an exception [6], [7]. PAS2P [1] is the proposed methodology, based on characterizing the dynamic behavior of parallel SPMD scientific applications that use message passing (MPI) during their execution for a specific workload and HPC system. The PAS2P methodology has 2 stages: In the first stage, there is the analysis of the application and generation of the signature, and in the second stage, a performance prediction is carried out.

The signature is associated with the behavior of a specific application. For example, if we want to predict the execution time of another parallel application or change the data set, the signature must be generated again. Therefore, the application analysis must be carried out in a short time to perform the analysis, the application is instrumented using

dynamically linked libraries, and traces are generated with the computing volume with PAPI [8], and the

MPI message passing events of each process are captured.

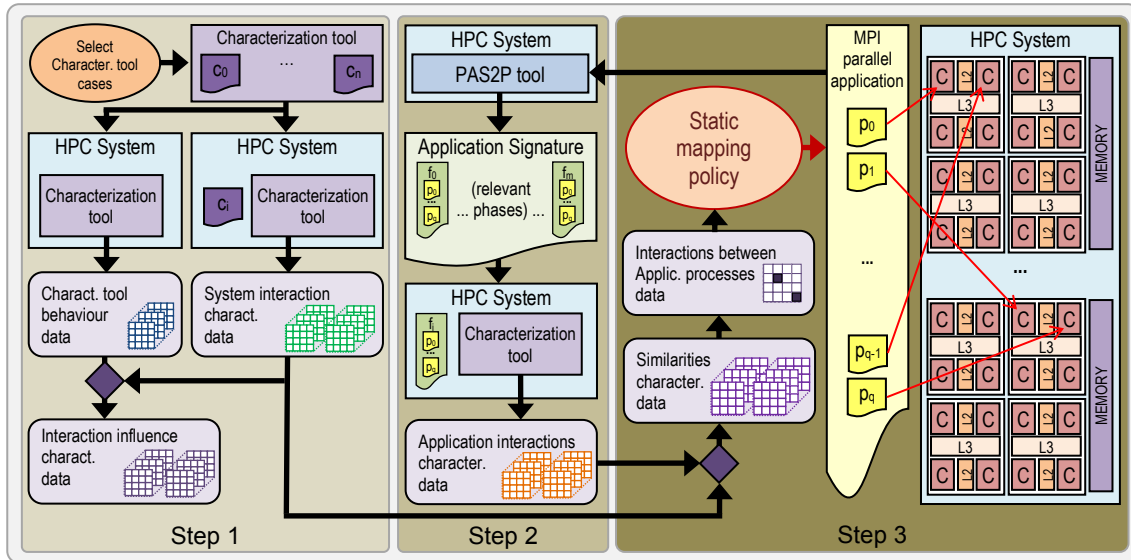


Figure 1. Methodology overview.  $C_i$  (Case  $i$  of the Characterization program),  $f_i$  (phase  $i$  of the Application signature),  $p_i$  (process  $i$  of a phase of the Application signature). As an HPC system example: C (CPU with a L1 Cache, first level Cache). L2 (second level Cache shared between 2 CPUs). L3 (third-level Cache share between 4 CPUs).

It assigns a global logical clock to the event to obtain the application model by interrelating the communication events using an algorithm inspired by Lamport and Time [9]. This ensures the correct logical ordering of events in a distributed system, where events across different nodes must be sequenced correctly relative to each other so as to maintain the causality and consistency of the system's state. Event sequence patterns with a relevant impact on performance are identified by the defined application model, which will be the phases of the application signature. A weight is assigned to each phase based on the number of times that pattern is repeated.

Once the phases that model the behaviour of the application have been identified, PAS2P generates what we call the application signature. This is formed by a set of phases from the parallel application code delimited by the events of the MPI communications and along with its weight and execution time.

In the performance prediction stage, the execution of the signature is instrumented in order to obtain the execution times of each phase which, multiplied by the weights of that phase, is defined as Predicted Execution Time (PET).

With PAS2P, we can characterize the behaviour of an MPI application based on a reduced set of phases. This allow us to focus the performance and efficiency analyses on executing the signature so that the execution time of the entire application can be predicted with a reduced computational, less than 10%, and high reliability, more than 95%, for

different HPC systems. Fig. 2 shows an overview of the methodology.

Other research has been developed on the PAS2P methodology, including a Parallel Program Scalability (P3S) methodology [10] based on the PAS2P signature. This allows us to analyze and predict the behaviour of MPI applications with strong scalability on a specific machine and estimate the execution time, using a limited execution time and a reduced set of resources.

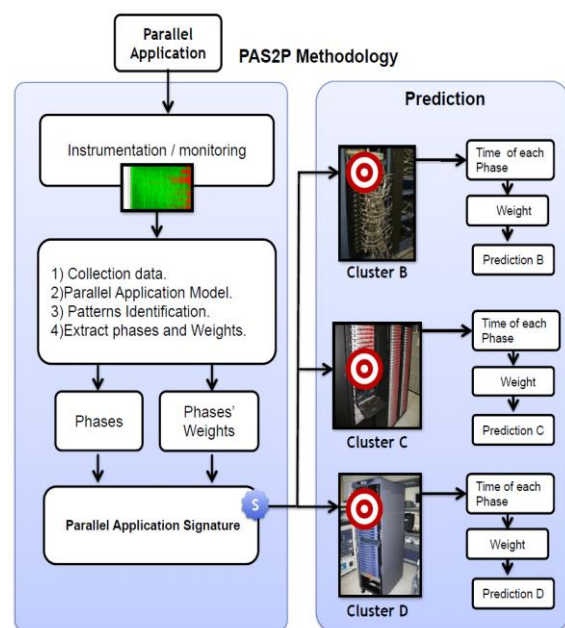


Figure 2. PAS2P methodology overview.

We analyze and characterize the communication pattern, between two communication events, and the weight of each phase of the PAS2P signature. From a set of small-scale executions, the behaviour information of the phases is obtained, in addition, how the application scales, generating the Scalable Logic Trace (SLT).

Finally, with P3S methodology, the Synthetic Signature (SS) is generated to obtain the performance of the application, using the PAS2P prediction equation, based on the performance prediction of the relevant phases.

In other research, its performance has been improved, as has the time required for the analysis and management of trace data that generates memory overload problems when scaling the application by parallelizing part of the tool [11]. The parallel approach has been developed with the message passing (MPI) standard for using the same application resources for analysis. The analysis of the traces is parallelized to extract the phases and generate the tables that will allow us to build the application signature and then be able to predict the execution time of the application, thus reducing the execution time and providing a better distribution of memory use. This parallel version allows the PAS2P methodology to be used on a large scale, achieving efficient analysis using the same resources allocated when running the parallel application.

In other work, an extension has been developed for applications with irregular behaviour [12]. During the analysis, PAS2P may reduce its efficiency due to the time used by inter-process synchronization mechanisms. This issue is more important when the application scales due to increased MPI communications.

In order to achieve this, an independent model is generated for each process, where each process has its own set of phases to minimize communications. This model fits the characteristics of SPMD applications, since all processes have similar behaviour.

The PAS2P version performs the global analysis in parallel for all processes and requires a global logical clock to order the events so as to search for similar patterns that characterize the parallel application in a single phase that represents all the processes of the application. This proposal builds an independent model for each process in order to eliminate event dependencies between processes, without the need of the global clock. The information obtained is stored by each process independently.

In the second stage, an identification of communication patterns for each process is carried out, grouping them into phases and assigning them a weight. There is an elimination of inter-process

communication events due to the identification of patterns at a global level for all processes.

Considering all this background, from the execution of the application signature obtained with PAS2P and the characterization of the machine communications, the aim is to obtain information to define a methodology that allows us to select a better mapping in a limited time, reducing application execution time, minimizing application inactivity [2].

This research aims to add to the information obtained by PAS2P the memory usage and the incidence of Cache misses on application performance in order to improve the static allocation of processes in an HPC system based on a hierarchical model using clustering methods.

### 3. Analysis of memory access behaviour.

#### 3.1. Instrumentation and data analysis

In order to analyze data, we have designed a benchmark to obtain the parameters using PAPI library instrumentation.

We analyzed the instrumentation incidence on performance and validate the parameters values that will be used for the analysis with the obtained indices. So as to avoid the effect of instrumentation and data collection between each array access, we read the PAPI counters just before and just after the array access and consider the differences between them.

The parameters analyzed are cycles, instructions, L1 Cache misses, L2 Cache misses, and number of array accesses. Based on these parameters, the performance indices defined are instruction per cycle, cycles per L1 miss, instructions per L1 miss, cycles per array access, instructions per array access, L1 misses per array access (see Fig. 3).

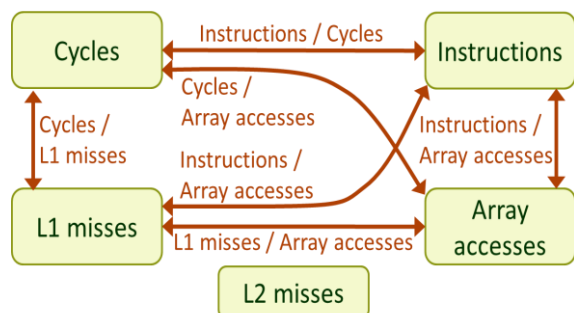


Figure 3. Parameters and performance indices analyzed.

Due to the great variability observed in the parameters analyzed in specific executions (see Fig. 4), we repeat each array access 100 times in order to consider each parameter as a variable with a Normal distribution (Central limit theorem). This allow us to

define the mean and standard deviation factors for each parameter (see Fig. 5), for cycles, instructions, array accesses and Cache misses, as well as for performance indices for different volumes of data, memory access traces and operations between memory accesses (see Table 1), thus allowing us to analyze their behaviour and make the obtained data comparable.

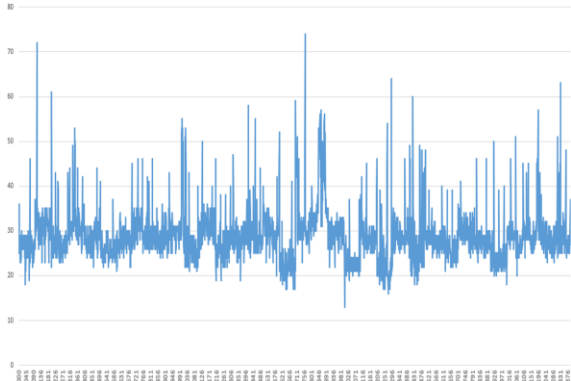


Figure 4. L1 misses reading a 4KB array using 16-byte step and an integer operation between each access to the array, values for 4000 times.

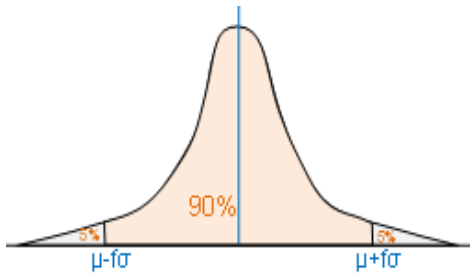


Figure 5. Factor  $f$  is applied to the deviation  $\sigma$  to have a probability greater than 90%. The mean  $\mu$  of another sample is within the interval  $[\mu - f\sigma, \mu + f\sigma]$ .

<b>Data size (v):</b>
<ul style="list-style-type: none"> <li>• Size of the array used based on the size of L1, L2, L3 Cache and RAM memory.</li> <li>• Data type used to declare and access the array: from 1 byte (char) to 8 bytes (long long).</li> </ul>
<b>Step distance (t):</b>
<ul style="list-style-type: none"> <li>• Regular steps, we always make the same step: step distance from 1-byte to distances higher than the Cache line (64 bytes), until the size L1.</li> <li>• Irregular steps, making steps with the same average as in regular steps, but with a certain variance.</li> <li>• Read and write the array.</li> </ul>
<b>Operations (o):</b>
<ul style="list-style-type: none"> <li>• Integer operations: from 1 to <math>10^6</math> operations between each two accesses to the array.</li> <li>• Floating point operations: from 1 to <math>10^6</math> operations between two accesses to the array.</li> </ul>

Table 1. Factors that affect Cache performance

### 3.2. Factors that influence performance

The representative cases can be extended in case a more precise characterization is necessary or when it is considered to introduce a new relevant factor having impact on the application performance.

Based on this test program, we define a benchmark, which we call the characterization program, to cross the most representative cases of each factor generating all cases of the memory access patterns and obtain the performance indices we will analyze.

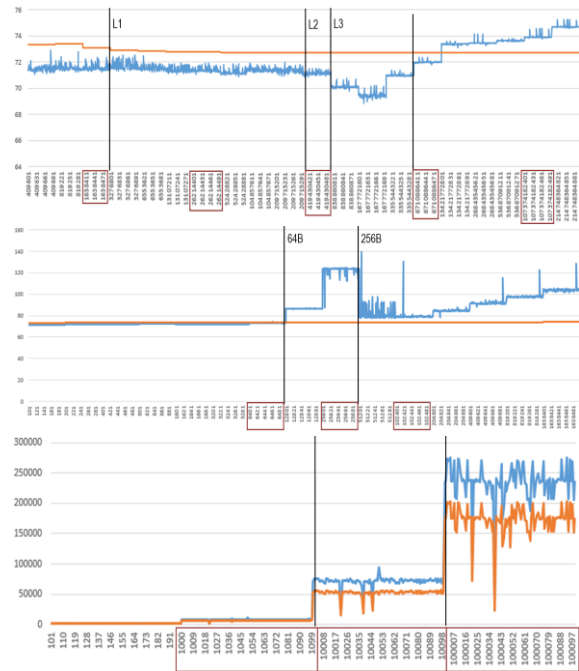


Figure 6. First chart changing array size (v), blue line cycles/memory access, orange line instructions/memory access, writing on the array. Second chart changing step (t), blue line cycles/memory access, orange line instructions/memory access, writing on the array. Third chart changing the number of operations between array access (o), blue line cycles/ L1 misses, orange line instructions/L1 misses, writing on the array.

## 4. Methodology

In this section, we explain the 3 steps of this methodology. Step1: Sections 4.1, 4.2 and 4.3, to characterize the system behaviour using known memory pattern access. Step 2: Sections 4.4, 4.5 and 4.6, the applications characterization extracting its relevant phases using the PAS2P tool. Step 3: Section 4.7, define the mapping policy.

### 4.1. System Characterization without sharing the Cache

We run the characterization program as a lone process, without sharing resources with other



programs, to obtain the most representative values of each factor  $pc(v_i, t_j, o_k)$ . It is configured for testing all the combinations, analysing  $n*m*p$  cases.

With this program, we can generate an  $m(v_x, t_y, o_z)$  matrix of means and deviations of the performance indices (see Fig. 3), with one dimension for each representative factor, where ‘n’ would be the number of representative cases for data size, ‘m’ the step distance and ‘p’ the number of operations (see Fig. 7). The data from this matrix will characterize the behaviour and performance based on each case of the memory access patterns.

This will allow us to detect which access patterns have better performance and which ones penalize the machine’s performance more. In addition, we know what the improvement that we can obtain will be when this access pattern obtains a worse performance when competing with other applications sharing the Cache.

Matrix of means and deviations of each performance index with one dimension for each factor (size, step and operations)

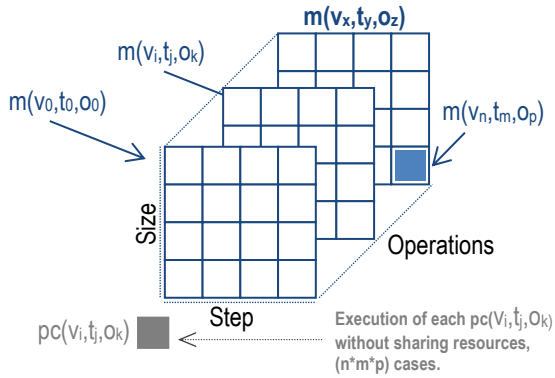


Figure 7. Matrix of means and deviations of each performance index  $m(v_x, t_y, o_z)$  produced by the characterization.

#### 4.2. System Characterization by sharing the Cache

Using the same characterization program, now running in parallel and sharing the Cache (from cache L2). An  $h_{nmp}(v_i, t_j, o_k)$  hypermatrix of means and deviations of the performance indices (see Fig. 3).

This hypermatrix is obtained from the characterization program, where each individual case of the characterization program is executed in parallel with all the cases of the characterization program sharing the Cache memory.

The data from this hypermatrix will make it possible to determine which cases generate more conflicts between them by sharing the Cache memory (see Fig. 8).

The data from this matrix will allow us to see where the most significant performance drops occur when sharing the Cache and it will generate the

$h_{nmp}(v_x, t_y, o_z)$  influence hypermatrix that specifies how it is penalized with respect to running without sharing the Cache.

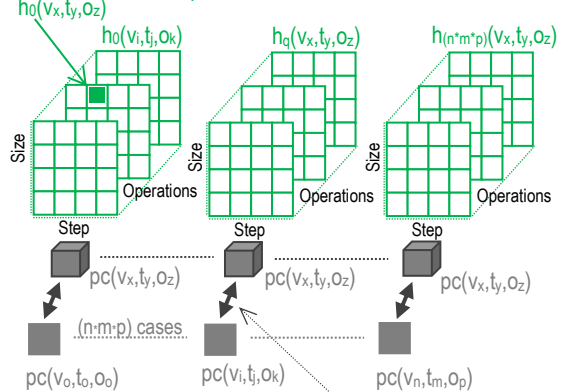
#### 4.3. Define the influence of sharing the Cache

From the differences between running the characterization program without sharing the Cache with other processes,  $m(v_x, t_y, o_z)$  matrix, and running it sharing the Cache,  $h_{nmp}(v_i, t_j, o_k)$  hypermatrix, the  $h_{nmp}(v_x, t_y, o_z)$  influence hypermatrix is generated, which will determine how it is affected, with each case being executed in parallel with another (see Fig. 9).

The data from this hypermatrix will make it possible to determine which cases most penalize performance and if they are more sensitive by sharing the Cache, in what proportion their performance decreases compared to running without sharing the Cache.

With these 3 matrices,  $m(v_x, t_y, o_z)$ ,  $h_{nmp}(v_x, t_y, o_z)$  and  $h_{nmp}(v_x, t_y, o_z)$ , we characterize the system, the behavior of an applications using known memory access patterns in this system where we want run our MPI applications. Moreover, it will not be necessary to generate them again if the characteristics of the system do not change.

Hypermatrix  $h_{nmp}(v_x, t_y, o_z)$  of means and deviations executed in parallel with other cases



Parallel execution of each  $pc(v_i, t_j, o_k)$  case against all the cases of the  $pc(v_x, t_y, o_z)$  characterization program sharing the Cache.

Figure 8. Matrix of means and deviations of each  $h_{nmp}(v_x, t_y, o_z)$  performance index produced by the characterization program sharing the Cache with other cases.

#### 4.4. Application characterization by sharing the Cache

Now, we want to characterize the behaviour of the application by analysing how this application interacts with the characterization program.

First, we generate the Application Signature with the

PAS2P tool to identify the most representative phases (the phases that have an impact on the performance of the application) and their repetition rates (weights).

Hypermatrix of influence with means and deviations comparing the execution without sharing resources and sharing the Cache.

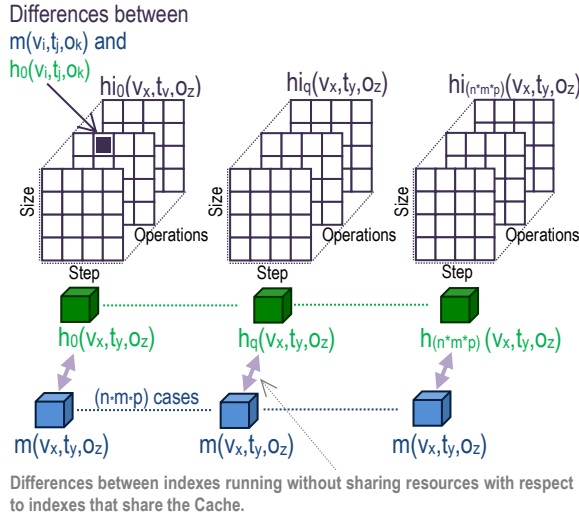


Figure 9. Hypermatrix of influence with  $h_{i_{nmp}}(v_x, t_y, o_z)$  performance indices by comparing the matrix of means and deviations of  $m(v_x, t_y, o_z)$  performance indices executed without sharing the Cache with other processes and the  $h_{nmp}(v_i, t_j, o_k)$  hypermatrix of means and deviations sharing the Cache with other cases.

Hypermatrix of means and deviations of the performance indices  $hf_x(v_i, t_j, o_k)$  executed in parallel each process of a phase with all the cases.

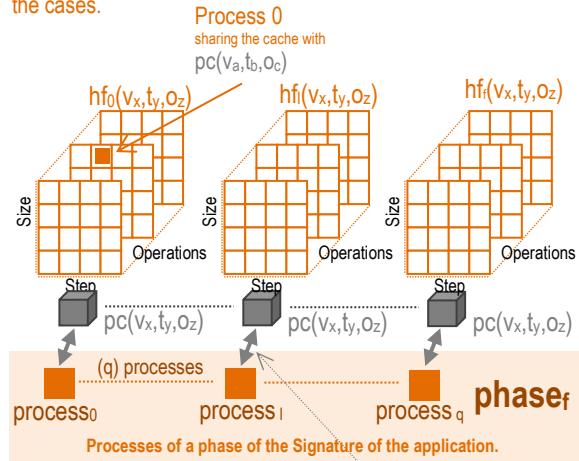


Figure 10. Hypermatrix of means and deviations of the  $hf_i(v_x, t_y, o_z)$  performance indices executing in parallel all the cases of the characterization program with each process of an application signature phase sharing the Cache.

Then, for each process of a signature phase, a hypermatrix is generated of means and deviations of the  $hf_i(v_x, t_y, o_z)$  performance indices, sharing the Cache with each case of the characterization program,

where ‘q’ would be the number of processes in which the execution of one phase of the application is split (see Fig. 10).

Comparing this hypermatrix with the  $h_{nmp}(v_i, t_j, o_k)$  hypermatrix, which shows how each case of the characterization program affects performances of all cases of the characterization program, we will find similarities between the behaviour of a phase process and the cases of the characterization program with known memory access patterns.

#### 4.5. Analyze similarities between the application behaviour and the characterization program behaviour

By comparing each submatrix of the hypermatrix of means and deviations of the performance indices obtained for each process of an  $hf_i(v_x, t_y, o_z)$  phase with the values of the hypermatrix of means and deviations of the  $h_{nmp}(v_x, t_y, o_z)$  performance indices for all the cases of the characterization program, the  $h_{snmp}(v_x, t_y, o_z)$  similarity hypermatrix is generated (see Fig. 11).

Hypermatrix of similarities of the performance indices comparing the execution sharing the Cache with all the cases and with a process of a phase.

Differences between  $hf_i(v_x, t_y, o_z)$  and  $h_0(v_x, t_y, o_z)$

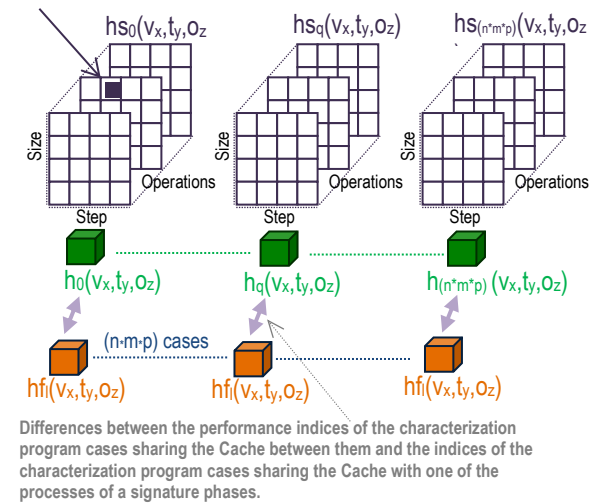


Figure 11. Similarity  $h_{snmp}(v_x, t_y, o_z)$  hypermatrix is obtained by comparing each submatrix of the hypermatrix of means and deviations of each performance index obtained for each process of a phase  $hf_i(v_x, t_y, o_z)$  with the values of the hypermatrix of means and deviations of the  $h_{nmp}(v_x, t_y, o_z)$  performance indices for all the cases of the characterization program.

The submatrices of the  $h_{nmp}(v_x, t_y, o_z)$  hypermatrix of all the cases of the characterization program that have smaller differences with the  $hf_i(v_x, t_y, o_z)$  submatrix of

each process of a phase will determine the cases that have more similarities. The mean of the case 'x' executed, sharing the Cache with process 'q', is in the range  $[\mu-f.\sigma, \mu+f.\sigma]$  of each case of the characterization program. It is executed sharing the Cache with the same case x (see Fig. 14).

The data from this hypermatrix of similarities will allow us to see the cases of the characterization program whose performance is affected in the same way, indicating which cases have a similar behaviour to that process of the phase.

A submatrix of the  $h_{fi}(v_x, t_y, O_z)$  hypermatrix indicates how each case 'x' of the characterization program is affected when it is executed sharing the Cache with the process 'q' of an application signature phase. We compare this with all the submatrices of each cases of the  $h_{nmp}(v_x, t_y, O_z)$  hypermatrix, where case 'x' has been executed in parallel sharing the cache with all the possible cases of the characterization program, as shown in Fig. 14.

To summarize, the similarity hypermatrix is generated for each process of a phase, which allows us to define a similarity factor, a mean weighted with each of the performance indices within the range  $[\mu-f\sigma, \mu+f\sigma]$ , executed in parallel with each case of the characterization program sharing the cache, in order to generate the  $ms(x,y)$  similarity matrix (see Fig. 12).

Similarity matrix between processes of a phase and the cases of the characterization program

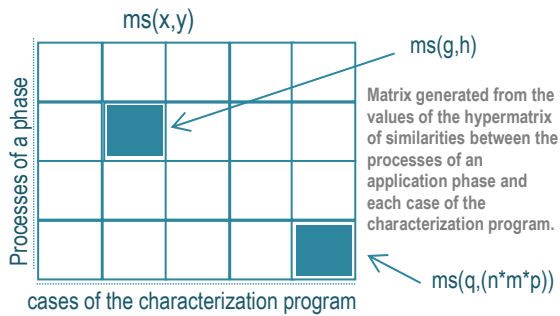


Figure 12. Similarity Matrix between processes of each phase and the cases of the characterization program.

#### 4.6. Analyse interactions between the application processes of one phase

With the  $ms(x,y)$  matrix we know the similarities between each process of a phase and the cases of the characterization program, as well as having the  $h_{nmp}(v_x, t_y, O_z)$  hypermatrix in order to know how performance is affected when two cases of the characterization program share the Cache.

With this information we can generate the  $mipf(x,y)$  interactions matrix between processes of one phase from the means and deviations of the performance

indices of the  $h_{nmp}(v_x, t_y, O_z)$  hypermatrix, where each individual case of the characterization program is executed in parallel with all the cases of the characterization program to determine their performance when sharing the Cache and the weight of each case according to the  $ms(x,y)$  similarity matrix between the processes of a phase and the cases of the characterization program (see Fig. 13).

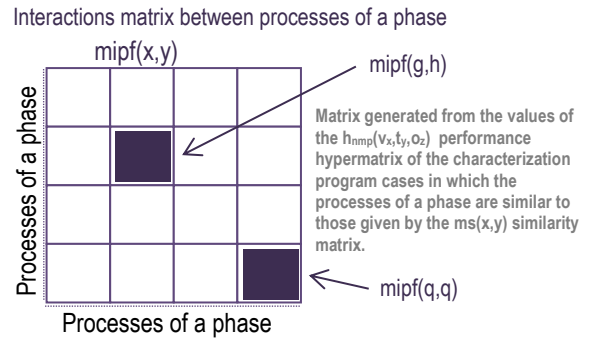


Figure 13. Similarity Matrix between processes of each phase and the cases of the characterization program.

The data from this matrix will make it possible to determine which processes in each phase can generate greater performance drops if they are executed in parallel, sharing the Cache.

#### 4.7. Define the mapping policy

Now, we are able to define the assignment of processes based on the data gathered.

We define a communication model of the system based on the transmission time grouped into 2 main categories: "Map-By-Cluster" (nodes in a cluster interconnected by a high-speed network) and "Map-By-Node" (cores in a node/socket that share the memory). We define this hierarchy to reduce the mapping complexity algorithms, and to improve application throughput. Considering the process attraction due to communications patterns (more communication volume, more proximity), and the processes repulsion due to memory patterns (more memory used, more repulsion).

In the first step, we generate groups of processes in the "Map by cluster" category using a the K-means clustering algorithm  $O(n^2)$ . This is based on the communication pattern (message size and phases weight), where 'k' is determined by the number of clusters. In each group we will join processes that have more communications between them. These communication events will be managed within a node/socket, and fewer communications between processes from other groups, this communication events are managed by the node/socket interconnection network with a lower bandwidth. In this first step, we do not consider inter-process



memory interaction hence one process in one node (a group) does not affect the performance of another process assigned to another node (another group), or the effect is negligible compared with communications due to the fact that they don't share the memory, or the memory cache.

In the second step, we assign processes to cores in the "Map by Node" category. For each group (processes assigned to a cluster) using a K-means clustering algorithm  $O(n^2)$ , based on the interactions matrix

between processes of a phase,  $mipf(x,y)$ , weighting the values of each phase with the weights obtained from the application signature shown in Fig. 15,  $k$  is determined by the cluster memory architecture, the number of NUMA nodes, and controlling the number of processes assigned to each centroid.

The static mapping policy will be defined by minimizing performance drops due to communications patterns and the interactions when sharing the Cache memory.

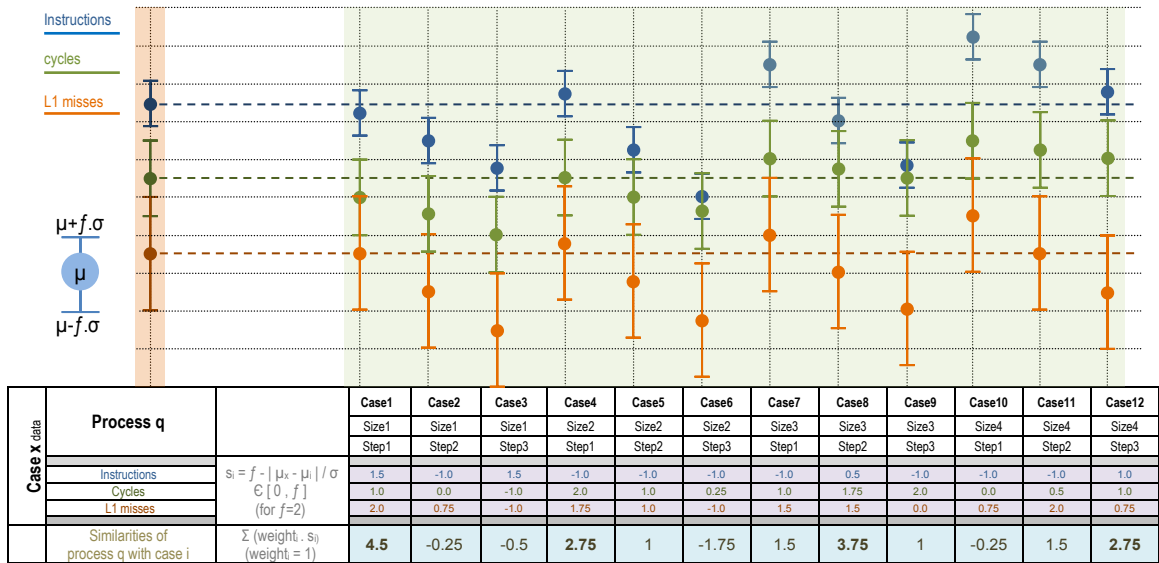


Figure 14. An example is shown for 2 dimensions (size/step) and 3 indices (Instructions, Cycles and L1 misses) to facilitate visualization. Positive values of (similarity) indicate that the index of one of the cases (case x) executed in parallel with a process (process q, left side) is similar (within the range  $[\mu-f\sigma, \mu+f\sigma]$  in each case) to that same index of each case of the characterization program executed in parallel with that same case (case x, right side). Negative values indicate that they are out of range.

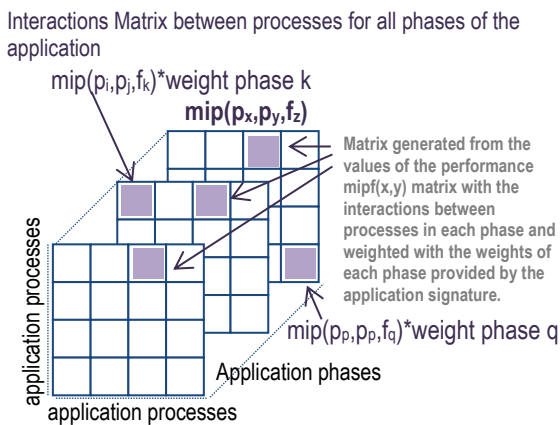


Figure 15. Interactions matrix between application processes for all phases.

### 5. Conclusions

We have proposed a methodology to analyze application behaviour and define a static mapping to

improve application performance.

It has been possible to validate that the data obtained from the instrumentation correspond to the theoretical values and are representative for identifying memory access patterns, also it has been possible to verify that working with means and deviations of the analyzed indices give more stable and comparable results, essential factors for this methodology.

The first step in this methodology is to characterize the system, this data can provide important information for software developers to know which memory access patterns achieve better performance when the application runs alone and when it shares memory with other processes.

In the second step, we characterize the application based on known memory access patterns. This information will let us know if the basic access patterns of this application obtain good performance on this system, which can allow us to improve the design of the application. It will also allow us to know

the conflicts generated by sharing the Cache and to know the room for improving performance if these conflicts are minimized. This improvement can be obtained by changing the design of the application, but also with the mapping policy to allocate processes in this system which is where this methodology is focused.

Finally, in the third step, with the data obtained in the first two steps about the interactions between application processes, we can define a static mapping policy to minimize performance drops due to communication patterns and process interactions when the Cache is shared.

Future lines will be focused on finishing implementing all the steps in this methodology based on the benchmark prototypes developed to define the methodology. Another important aspect of this methodology is that it allows changes to be made in the weighting of the indices used to give more emphasis to different aspects of the application or system performance.

### Competing interests

The authors have declared that no competing interests exist.

### Funding

This research has been supported by the Agencia Estatal de Investigacion (AEI), Spain and the Fondo Europeo de Desarrollo Regional (FEDER) UE, under contract PID2020-112496GB-I00.

### Authors' contribution

GE designed the methodology and conducted the empirical studies to refine the preliminary version of the model, developed the code, results analyzed and wrote the manuscript. EL and AW conceived the research proposal and defined the preliminary methodology. EB defined the statistical model to analyze data and validate dataset. RS and AW technical and code development support and system configuration. DR made the analysis and review the proposal. EL and DR were responsible of the computing resources. All authors reviewed and approved the final manuscript.

### References

- [1] A. Wong, D. Rexachs, and E. Luque, "Parallel Application Signature for Performance Analysis and Prediction", IEEE Transactions on Parallel and Distributed Systems, 2015, vol. 26, no. 7, pp. 2009-2019
- [2] C.R. Rangel, A. Wong, D. Rexachs and E. Luque, "Using the application signature to detect inefficiencies generated by mapping policies in parallel applications", International Conference on High Performance Computing Simulation (HPCS), 2017, pp. 534–540, doi: 10.1109/HPCS.2017.85.
- [3] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "A tool for selecting the right target machine for parallel scientific applications", Proc. Int. Conf. Comput. Sci., 2013, pp. 1824–1833.
- [4] Agung, M., Amrizal, M. A., Egawa, R., & Takizawa, H. (2020). Online MPI process mapping for coordinating locality and memory congestion on NUMA systems. Supercomputing Frontiers and Innovations, 7(1), 71-90.
- [5] I. Chung, Ch. Lee, J. Zhou, and Y. Chung, "Hierarchical mapping for HPC applications". Parallel Processing Letters, 2011, Vol. 21, No. 03, pp. 279-299.
- [6] T. Sherwood, E. Perelman, and B. Calder. "Basic block distribution analysis to find periodic behavior and simulation points in applications". In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT), 2001, pp. 3-14.
- [7] E. Perelman, M. Polito, J.-Y. Bouguet, J. Sampson, B. Calder, and C. Dulong. "Detecting phases in parallel applications on shared memory architectures". Parallel and Distributed Processing Symposium, International, 2006, pp. 0-10.
- [8] D. Terpstra, H. Jagode, H. You, and J. Dongarra. "Collecting performance data with papi-c. In Tools for High Performance Computing" Springer, 2010, pp. 157–173.
- [9] L. Lamport and C. Time, "The ordering of events in a distributed system," Commun. ACM, 1978, vol. 21, no. 7, pp. 558–565.
- [10] J. Panadero, A. Wong, D. Rexachs, E. Luque. "P3S: a methodology to analyze and predict application scalability", IEEE Trans Parallel Distrib Syst, 2017, 29 (3):642–658.
- [11] F. Tirado, A. Wong, D. Rexachs, E. Luque, "Analyzing the data behavior of parallel application for extracting performance knowledge", IEEE 21th International Conference on High Performance Computing and Communications, 2019.
- [12] F. Tirado, A. Wong, D. Rexachs, E. Luque, "Scalable performance analysis method for SPMD applications. The Journal of Supercomputing, 2022, 78, 19346–19371. <https://doi.org/10.1007/s11227-022-04588-z>

**Citation:** G. Enrique, E. Bruballa, R. Suppi, A. Wong, E. Luque and D. Rexachs. *Methodology to Define a Static Allocation Mapping based on Memory Access Patterns and the Signature of MPI Applications in HPC Systems*. Journal of Computer Science & Technology, vol. 24, no. 2, pp. 120-129, 2024.

**DOI:** 10.24215/16666038.24.e12

**Received:** November 9, 2023 **Accepted:** May 31, 2024.

**Copyright:** This article is distributed under the terms of the Creative Commons License CC-BY-NC-SA.