The Journal of Supercomputing (2022) 78:19346–19371
https://doi.org/10.1007/s11227-022-04588-z

Scalable performance analysis method for SPMD applications

Felipe Tirado[1,2] · Alvaro Wong[1] · Dolores Rexachs[1] · Emilio Luque[1]

Accepted: 1 May 2022 / Published online: 20 June 2022

Abstract

The analysis of parallel scientific applications allows us to understand their computational and communication behavior. One way of obtaining performance information is through performance tools. One such tool is parallel application signatures for performance prediction (PAS2P), based on parallel application repeatability, focusing on performance analysis and prediction. The same resources that execute the parallel application are used to perform its analysis, creating a machine independent model of the application and identifying its common patterns. However, the analysis is costly in terms of execution time due to the high number of synchronization communications performed by PAS2P, degrading performance as the number of processes increases. To solve this problem, we propose a model that reduces data dependency between processes, reducing the number of communications performed by PAS2P in the analysis stage and taking advantage of the characteristics of single program, multiple sata applications. Our analysis proposal allows us to decrease the analysis time by 29 times when the application scales to 256 processes, while keeping error levels below 11% in the runtime prediction. It is important to mention that the analysis time is not considerably affected by increasing the number of application processes.

Keywords Performance prediction · Application performance analysis · MPI parallel application · Application signature

✉ Felipe Tirado
felipeleonardo.tirado@autonoma.cat; ftirado@ucm.cl

Alvaro Wong
alvaro.wong@uab.es

Dolores Rexachs
dolores.rexachs@uab.es

Emilio Luque
emilio.luque@uab.es

[1] Computer Architecture and Operating System Department, Universidad Autónoma de Barcelona, Barcelona, Spain

[2] Departamento de Computación e Industrias, Universidad Católica del Maule, Talca, Chile

1 Introduction

Over the past few years, high-performance computing (HPC) systems have significantly increased the number of processing units (CPUs) [2]. Today, these systems have enormous computing power, and there is every indication that this trend will continue to increase in the coming years due to constant technological improvements. However, the gap between the theoretical maximum performance and the performance achieved in scientific applications running in parallel has widened considerably in recent years, mainly due to incorrect application programming or inefficient system resource management.

Performance Tools are available [12, 15, 25] that collect and display relevant information about application performance at a high level of abstraction, so that developers can quickly identify and determine the causes that affect the efficiency and performance of applications.

On the other hand, it is essential to determine which system is more appropriate to execute a scientific algorithm and predict its execution time. Therefore, accurate performance estimates are critical to helping a system resource scheduler efficiently schedule user jobs. For example, if system resource administrators know how many resources are requested and how long they are requested, they can make an efficient queued plan.

As presented in Fig. 1, the PAS2P Tool [13, 24] instruments and analyzes MPI parallel applications on a cluster A (base machine) to create a signature that characterizes the behavior of the application in phases. A phase is a parallel code segment delimited by MPI communications repeated (Weight) throughout the execution. The signature will be executed on a target machine (cluster B or cluster C), where each phase's time will be measured to predict the execution time on the target machine.

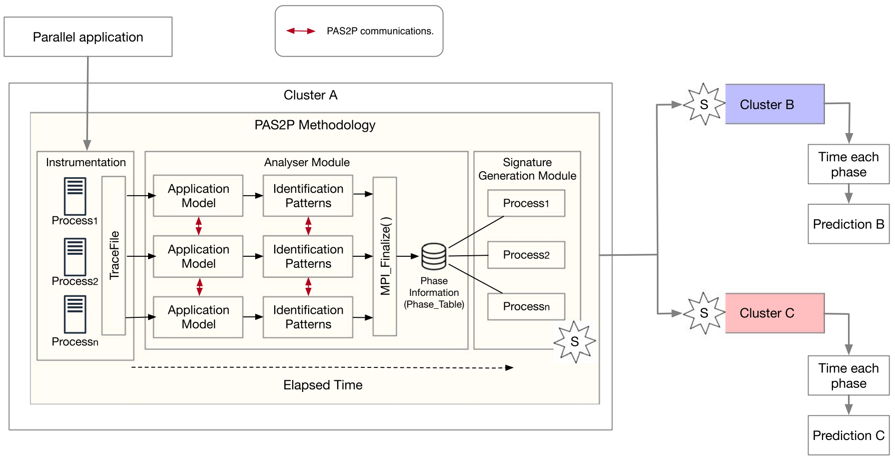


Fig. 1 Overview of parallel PAS2P. The generation of the application signature is performed on the base machine (Cluster A). Then, the signature is executed in the target machines, cluster B and cluster C, where the time of each phase is measured and multiplied by its weight, predicting the execution time of the application

It is essential to mention that the application will not be fully executed on the target machines to obtain its execution time. Only the relevant sections of the application will be measured in order to apply the prediction equation. The prediction equation multiplies the execution time of each relevant section by the number of times it is repeated. Adding all previous times, we obtain a prediction of the parallel application's execution time (PET).

Due to the PAS2P parallel analyzer [21] model, which includes several interprocess synchronization mechanisms, represented with red arrows in Fig. 1, the performance of the PAS2P analysis is inefficient because of the high time required. This performance problem occurs more significantly when the application scales due to increased MPI communications. This performance problem can cause the execution time in the PAS2P analysis stage to exceed the time of the application to be analyzed.

We propose to improve the performance of the PAS2P analysis module, considering the behavior of the SPMD application. To do so, we generate an independent model for each process, where each process has its own set of phases, thus minimizing the communications performed by PAS2P. Furthermore, this model is adjusted to the characteristics of the SPMD applications since all the processes have similar behavior.

Figure 2, exposes a comparison between the parallel analysis version and our analysis proposal. The parallel analysis approach performs a global analysis for all processes, having to communicate the events of each process to build a global logic clock that allows us to order the events, looking for similar patterns that characterize the parallel application and then create a single phase that represents all application processes. On the other hand, our proposal performs a horizontal analysis of the SPMD application. Each process builds its model independently and analyzes similarity patterns, eliminating PAS2P communications. The information obtained is stored by each process independently.

To evaluate the quality of our proposal, we performed experiments with different applications, BT, SP, LU of NPB [3] and N-Body, with different workload sizes, increasing the number of processes to verify the behavior of the analyzer and the quality of the prediction. We were able to obtain a speedup measure of the analyzer higher than 24 times compared to the parallel analyzer and an execution time prediction with an average error of less than 6%.

In the following section, we present related work. In Sect. 3, we provide an overview of the PAS2P toolkit. Section 4 presents the proposed methodology for SPMD applications. Section 5 deals with experimental results, and in Sect. 6, we present our conclusions and future work.

2 Related work

This section presents performance analysis tools that also seek to optimize their post-mortem analysis because it is essential to have a fast and simple analyzer that allows the extraction of relevant performance information to obtain the parallel application's behavior.

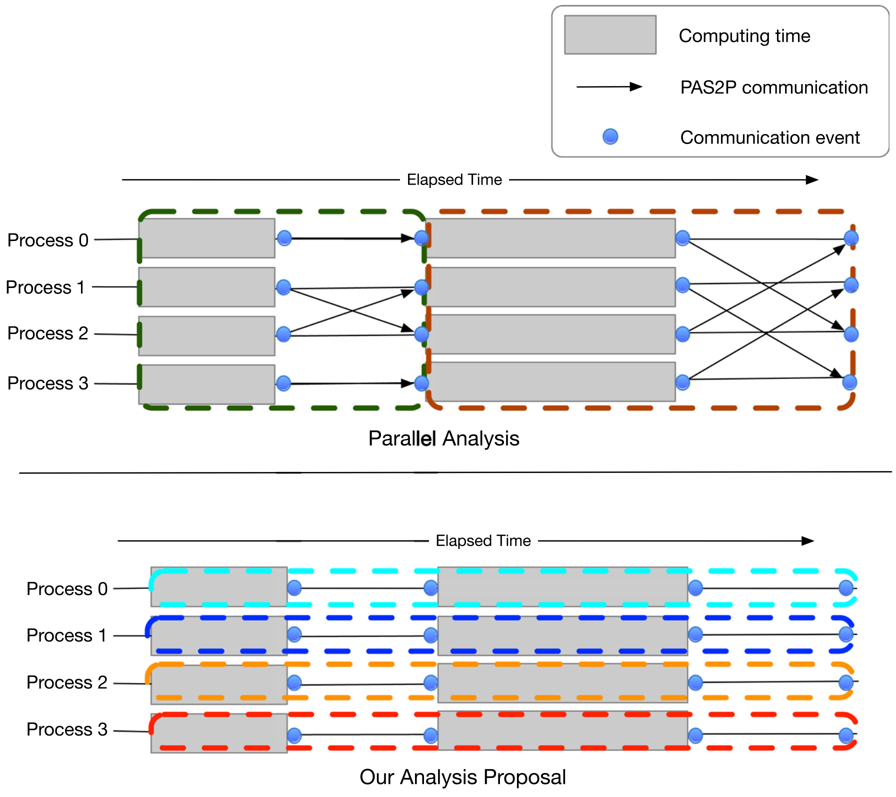


Fig. 2 Comparison of PAS2P analysis proposals. The parallel analysis proposal presents a vertical analysis model where all the application processes interact. Our proposal performs a horizontal analysis model where each process analyzes the application independently

There are parallel application analysis tools such as Scalasca [6]. Scalasca is an open-source toolkit that can analyze parallel applications' performance behavior and identify optimisation opportunities. Although Scalasca can be used on 294K cores, version 1.3.0 and the underlying versions showed scalability limitations during the collection and display of analysis reports. Scalasca v2 [25] is based on the community instrumentation and measurement infrastructure Score-P, improving the scalability limitations and adding new features.

Periscope [12] is a performance analysis tool which analyzes parallel applications to detect performance problems and their causes. This tool overcomes the scalability barrier by performing an automatically distributed online analysis on thousands of processors. The main drawback of Periscope is that the application must be executed entirely to see the optimizations applied.

Paraver [11] uses the CEPBA-tools [9] environment to scale the applicability of the Paraver trace visualization and analysis tool to systems with up to several thousand processors. Its analysis power is based on two main pillars. First, its trace format has no semantics; extending the tool to support new performance data or new programming models requires no changes to the visualizer, it just captures such data in a Paraver trace. The second pillar is that the metrics are not hardwired on the tool but programmed. To compute them, the tool offers a large set of time functions and a filter module.

MUST [7] is a framework for creating a runtime infrastructure for scalable MPI correctness checking. The main goal is to offer a full set of correctness features for 1,000 processes at a runtime overhead of less than 10%, and a restricted set of correctness features for 10,000 processes at the same runtime overhead.

The Cray performance analysis tools [5] provide an integrated infrastructure for measurement and analysis of computation, communication, *I*/*O*, and memory utilization. It is composed of a CrayPat Performance Collector for data capture, and a Cray Apprentice2 Performance analyzer to a post processing data visualization. The Cray performance analysis tools have been used on a large scale Cray XT system with more than 30,000 processors.

TAU [17] and Vampir [22] have focused their efforts on improving the analysis of data on a large scale. To efficiently achieve this analysis, TAU uses the ParaProf Parallel Performance analyzer [18], which was specifically built for the analysis of large scale data. The analysis takes place in memory for fast access and to support global aggregation and analysis views. TAU provides a compressed normalized packed data format as a container for profile data from any supported measurement tool. This makes the reading of parallel profiles significantly more efficient in ParaProf. Vampir provides efficient access to trace files. This layout allows us to distribute the data in several files, each one storing a ”frame” of the execution data. Frames can correspond to a single CPU or to a cluster of CPUs. The frames belonging to a single execution are tied together by means of an index file, thus providing better performance.

On the other hand, Jayakumar [8] developed a prediction framework for performance predictions of HPC applications using a single small-scale application execution. The framework employs a strategy of matching execution profiles of the different phases of the parallel applications to parallel reference kernels stored in a kernel database. The framework provides a suite, an RK-suite, implementations, execution profiles and performance models of reference kernels. Specifically, the RK-suite consists of a collection of these reference kernel implementations. Execution profiles, an RK-profile, including cache hits and misses, an instruction mix, etc., were all obtained using benchmarking runs of the reference kernels for a finite set of problem sizes and a number of processors. A performance model, the RK-model, can predict execution times of the kernel implementations for other problem sizes and processors.

PAS2P [24] characterize the behavior of message-passing applications on different target machines. To achieve this, it develops a method called parallel application signature for performance prediction, which describes an application in terms of its behavior. Based on the message passing activity of the application, representative phases are identified and extracted, with which a parallel application signature is created to predict the performance of the application.

Table 1 summarizes the main performance analysis tools, giving a critical view, in our opinion, of each of the papers. All the works presented seek to improve their

Table 1 Summary of papers referenced

Analysis tools	Description	Critical view
Scalasca	The Scalasca toolkit is a portable software package that supports performance optimization of parallel applications by measuring and analyzing their dynamic behavior at runtime. A distinctive feature of Scalasca is its automatic trace analysis, which identifies potential performance bottlenecks, particularly those related to communication and synchronization	Scalasca must completely analyze the application trace to identify potential performance problems, so the time required to obtain performance measurements will function the application's execution time
Periscope	The Periscope tool allows analyzing performance issues of MPI-based parallel applications and evaluating the performance of a single node. In addition, it performs an automatic search for predefined performance properties while executing the application	The analysis of MPI applications only looks for the properties of each process and does not consider the relationships of several processes
TAU	TAU (Tuning and analysis utilities) addresses performance technology issues at three levels: instrumentation, measurement, and analysis. In addition, the TAU framework supports the configuration and integration of these layers to address specific performance problem-solving needs. To this end, the TAU performance system offers support for performance analysis in a number of ways	Effective performance exploration will require careful selection from the range of methods offered by TAU to assemble meaningful performance experiments
Vampir	Vampir is an event tracking analysis tool for parallel applications. It supports the analysis of a number of important performance properties, such as function calls, hardware performance counters, communication, *I/O* behavior, and memory allocation	The analysis is resource-intensive as it does not have a redundant event recognition mechanism, loading all application events into memory, which is a great inconvenience when the application scales
RK-suite	RK-Suite is a suite for performance predictions of HPC applications using a single small-scale application execution. The suite employs a strategy of matching the execution profiles of different phases of parallel applications to parallel benchmark kernels stored in a kernel database. The benchmark kernels are standard benchmarks of various application domains	In order to predict the performance of an application, RK-Suite consults a database of kernels. The main drawback is that if the application's behavior to be predicted is not found within the stored kernels, its prediction will not be possible
PAS2P	The PAS2P methodology allows us to generate a model of a parallel application and automatically extract its most significant phases to create a signature whose execution lets us predict the application's performance on different parallel computers	Due to the large number of communication events captured in the instrumentation stage of PAS2P, trace analysis is complex and costly, even more so when the application scales to a more significant number of processes

analysis stage, some to identify performance problems and others to characterize the application better, improving prediction quality.

Previous work exists, such as the PAS2P parallel analyzer module [21], which defines a parallel analysis for the PAS2P tool. The main drawback of this analysis model is its complexity and the large number of MPI messages resulting from communication due to PAS2P event synchronisation. Given the high number of MPI messages, the performance of the analysis performed by PAS2P is low, in some cases exceeding the execution time of the application to be analyzed. This is why it is convenient to propose a less complex analysis model that reduces the MPI messages to increase performance.

3 PAS2P overview

The PAS2P tool is based on the repeatability of the parallel application, focusing on the performance analysis and prediction of the MPI application using its signature.

Figure 3 presents an overview of the PAS2P methodology. It is important to note that creating the signature of the parallel application is performed on a base machine (cluster A). This signature represents the performance characterization of the application. Next, to obtain the performance prediction on a target machine (cluster B and cluster C), the signature is executed on these machines by measuring the execution time of each sequence of more relevant events (phases). This time also includes the computation and communication time of each phase. Finally, the prediction equation is applied to obtain the predicted execution time on the target machines.

PAS2P allows the application to be instrumented when running on a parallel machine. By executing the instrumented application, a trace is obtained and

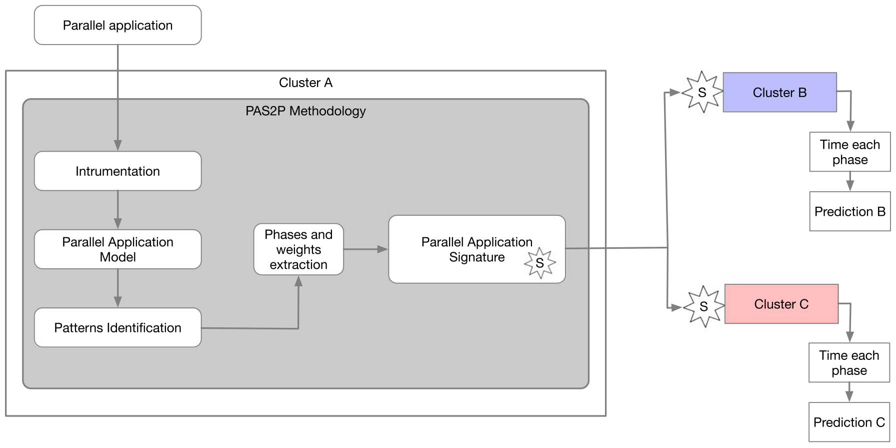


Fig. 3 Stages of the PAS2P methodology. The PAS2P methodology has two stages. The first is the generation of the parallel application signature in the base cluster (Cluster A). The second is predicting the application's execution time in a target cluster (Cluster B and Cluster C), executing the application signature

used for data collection. The collected data is used to analyze and characterize the computational and communication behavior of the application.

To obtain a machine-independent model of the application, it is necessary, once the trace has been obtained, to analyze it and assign a global logic clock according to the relationships between the communication events through an algorithm based on Lamport [10]. In this way, a unified trace is obtained by a logical clock for the whole distributed system.

Once a logical trace is obtained, the most relevant event sequences (phases) are identified and extracted, assigning them a weight defined by the number of times they occur. Subsequently, the application's signature is created, defined by a set of phases selected for their importance according to their weight (number of times they are repeated), or to the duration of the phase (its execution time). The created signature is used for executing in different clusters, which allows us to measure the execution time of each phase, thus predicting the execution time of the entire application in each of these target systems.

As shown in Fig. 4, two approaches for PAS2P have been developed, the serial analysis method and the parallel analysis method, detailed below.

- The serial analyzer approach [24] processes data from all application processes into a single collection structure to create a logical global clock and maintain precedence between communication events. When the application runs with a large number of processes, it may result in insufficient memory on the node. Thereby, having to load these data from the swap memory considerably increases the analysis execution time, or in many cases, it is not even possible to execute it due to the restrictions of the target machine.
- The parallel analyzer approach [21] has been developed using message passing to take advantage of distributed memory. This module allows us to use the PAS2P toolkit on a large scale, achieving an efficient analysis, since it divides the data analysis among all the resources and it executes using the same number of resources as the application uses for its execution.

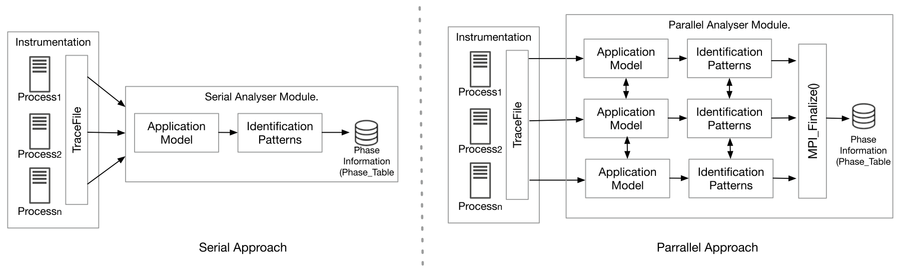


Fig. 4 Approaches to the method of analysis. On the left is the serial analysis approach. The analysis module is performed by a single process, limited to only one machine's available resources. On the right side is the parallel analysis approach. The analysis module is performed in several processes, sending and receiving messages for correct synchronization

Both versions (serial analysis approach and parallel analysis approach) have drawbacks when scaling the parallel application to a high number of processes. Performance is degraded due to communications (parallel analysis approach) or cannot guarantee a result (serial analysis approach) because of the restrictions of HPC systems.

Our proposal consists of modeling a new approach to the analyzer module, reducing the cost of communication between events of different processes due to the independent analysis mechanic for each process that runs the application, thus enabling a less complex analysis module that achieves an improvement in the performance of the PAS2P analysis module of the SPMD application.

4 Proposed methodology

Applications typically possess highly repetitive behavior, and parallel applications are no exception [16, 19]. To characterize the computational-related and communication-related behavior of parallel applications, we identify these repetitive portions of an application. We use this information to create a signature that, when executed, allows the prediction of the execution time for the machine on which the signature is run.

The signature is associated with the behavior of a specific application [4]. For example, if we want to predict another parallel application's execution time or change the data set, the signature must be generated again. Therefore, the application analysis must be carried out in a reduced time to generate the signature quickly.

Previous PAS2P approaches [21] optimize the analysis stage but have a drawback of dependencies between events, thus reducing its performance. The dependencies between events occur due to the communication effected by constructing a global clock that orders the events by precedence. In addition, the similarity algorithm needs communication synchronization steps to find a global repetitive structure for all application processes. Therefore, we propose an extension to the parallel analysis approach for solving this problem, eliminating the data dependencies between SPMD applications processes, which we defined as "Extension of Parallel Analysis to SPMD" (EPAS).

As shown in Fig. 5, we designed a proposal called EPAS that eliminates the dependence of events between processes in the SPMD applications, allowing us to reduce the communication for synchronisation reasons. Our proposal is divided into two sections: An Application model and Pattern identification.

1. The model of the application: The model is built by assigning timestamps to each process independently according to communication events' precedence relationships, using an algorithm inspired by Lamport [10]. Thus, we obtain a single logical trace for each process, eliminating the dependencies of events between processes since the application's global clock is not generated.
2. Pattern identification: Once the logical times have been assigned to the events, an identification of communication patterns is carried out, grouping them into phases and assigning them their respective repetitive frequencies (weights). This

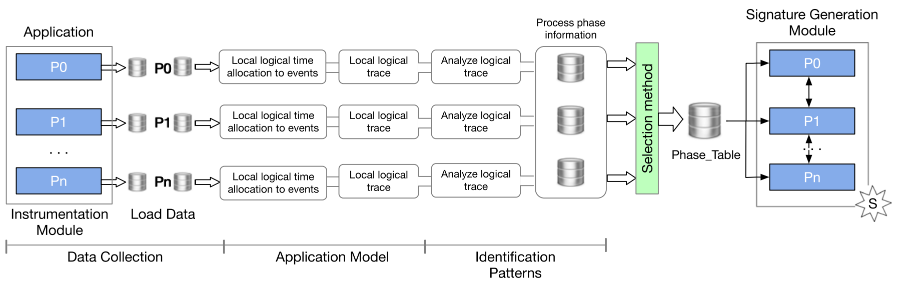


Fig. 5 Overview extension of parallel analysis to SPMD. It consists of three stages: data collection, application model, and pattern identification. Each process performs each stage independently, reducing the communication of events between processes

identification is carried out independently for each process, eliminating the communication of events between processes due to identifying patterns at the global level for all processes.

In this section, we describe each stage of our EPAS proposal. Section 4.1, namely data collection, describes the application instrumentation. Section 4.2 describes the modeling of the application, which is performed independently by each process. Finally, Sect. 4.3 describes how the phases and weights are identified for each process running the SPMD application.

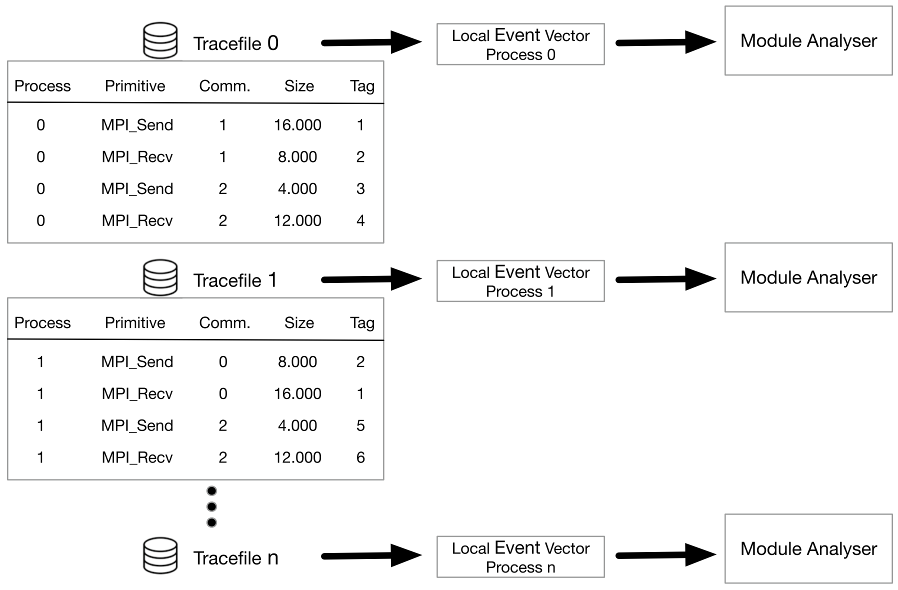
4.1 Data collection

To instrument the applications, we must collect their communication and computation times. We use the dynamic library libpas2p to produce a trace of the application. To instrument the application with libpas2p, it will be necessary to compile it with a dynamically linked library libpas2p, which intercepts MPI functions before the MPI library executes them, capturing the communication instructions performed by the parallel application in real-time.

As shown in Fig. 5, during instrumentation, each process generates information related to the communication and computational information involved in the process. In this way, the instrumentation module extracts information by using the same process number of the application.

To obtain the computational behavior of the application's processes, the PAS2P tool is integrated with the PAPI library [20] to obtain the hardware counters, such as the number of instructions and cache misses.

All the information extracted from the parallel application is stored in diverse trace files, as shown in Fig. 6. The number of trace files is related to the number of processes running the application, i.e.,the instrumentation and the other stages of PAS2P use the same processes as the parallel application.


Tracefile 0
Local Event Vector Process 0
Module Analyser
Tracefile 1
Local Event Vector Process 1
Module Analyser
Tracefile n
Local Event Vector Process n
Module Analyser

Process	Primitive	Comm.	Size	Tag
0	MPI_Send	1	16.000	1
0	MPI_Recv	1	8.000	2
0	MPI_Send	2	4.000	3
0	MPI_Recv	2	12.000	4

Process	Primitive	Comm.	Size	Tag
1	MPI_Send	0	8.000	2
1	MPI_Recv	0	16.000	1
1	MPI_Send	2	4.000	5
1	MPI_Recv	2	12.000	6

Fig. 6 Storage of application performance information. The instrumentation is performed by each process running the parallel application, capturing information from the MPI primitives, and storing it in the ”Tracefile” created by each process

4.2 Application model

The generation of an abstract model of parallel applications requires detecting the computational intervals, the communication events and the logical order that this communication must perform, independent of the machine. Therefore, a logical ordering of events is necessary that takes into account these singular aspects.

The logical ordering of the events uses an implementation based on Lamport’s [10]. Lamport defined the relationship of precedence between two events a and b, where a occurs before b as the physical clock of the process a is smaller than the physical clock of process b.

The implementation of Lamport’s algorithm is based on the assignment of Logical Time (LT) [23] to all the events of the parallel application, which uses a queue structure starting from the initial events and searching for all the related events to assign their logical time.

Algorithm 1: Logical time allocation by process.

```
Input:
Q: Event queue (POP and PUSH)
CE: Current event
FE: Next event
BE: Previous event
//All events start with an logical time of zero
//The first events are inserted in the queue $Q$
while $(Q\ != empty)$ do
    $C \leftarrow Q.POP$
    $Q.PUSH(FE)$
    if $BE = 0$ then
        $CE \leftarrow 0$
    else
        $CE.LT \leftarrow BE.LT + 1$        ▷ $TL$ is the logical time of the event
```

Our proposal uses Lamport's algorithm to assign the Logical Times (LT) to the events. Logical Time is obtained by analysing the trace file generated in the data collection stage. We take advantage of SPMD applications characteristics [1], assigning the LTs for each process independently, without considering communications due to data dependency between events due to the non-existence of a global LT for all processes. The allocation of logical times is illustrated in the Algorithm 1, executed by each process independently. For a better understanding, Fig. 7 is presented.

Our proposal differs from the implementation of the parallel analyzer [21] because we have designed a new model of the application where each process is independent from any other. This allows us to assign LT, without replicating the communication pattern of the application to send the information of the logical times of each event, as can be observed in Fig. 8, as well as using collective MPI to synchronise all processes. This new application model allows us to reduce the communication between processes, reducing the analyzer's execution time.

When all events have been assigned an LT, we use the concept of Tick as a Logical Time Unit [24]. For this purpose, a structure is created for each as a process, where the events are inserted in ascending order by LT, as can be observed in Fig. 9. The proposed ordering method is not affected by the change in the order of events due to delays in the interconnection network because the assignment of LT and orders will be carried out in each process independently.

4.3 Identification pattern

The objective of this section is to find the repetitive behavior of a parallel application. The application is analyzed by identifying similar sections in computation time, communication time and communication type. We call these relevant sections phases.

There are two similarity algorithms, the serial similarity method and the parallel similarity method [21]. In both cases, the concept is the same, the identification of phases in parallel applications, extracting directly from the logic trace. For

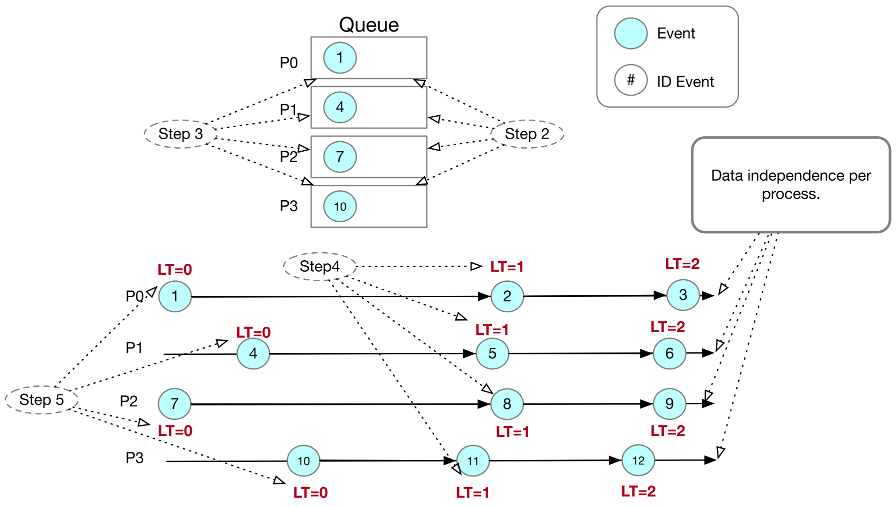


Fig. 7 Logical time insertion scheme per process independently. The following steps are presented: 1. All events start with an LT of zero. 2. A queue is created, and the first events are inserted. 3. The first event, CurrentEvent, is extracted from the queue. 4. The next consecutive event of the same process, ForwardEvent, is inserted in the queue. 5. The CurrentEvent Logical Time is the Logical Time of BackEvent plus one (BackEvent +1). 6. The procedure is completed when the queue is empty

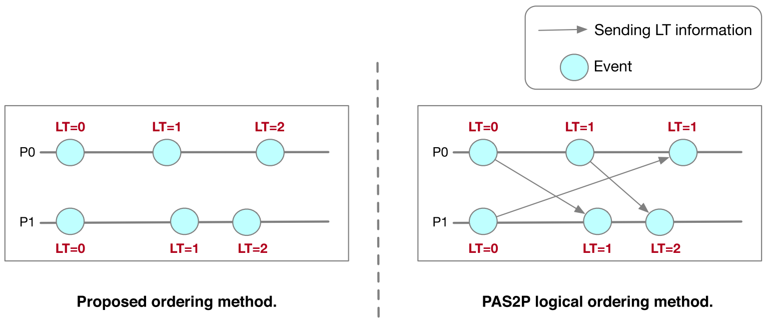


Fig. 8 Comparison of ordering methods. The right side shows the logical ordering method of PAS2P, where all the processes running the application jointly perform the ordering of PAS2P events, having to perform numerous synchronization communications. On the left side, we present our proposal, which orders the events independently by each process, eliminating the communications between PAS2P events

the serial version, the logical trace is loaded into the memory of a cluster node. Therefore, we have a global view of the logical trace that allows us to search for a pattern for all the application events, while in the parallel version, each process has its local logical trace.

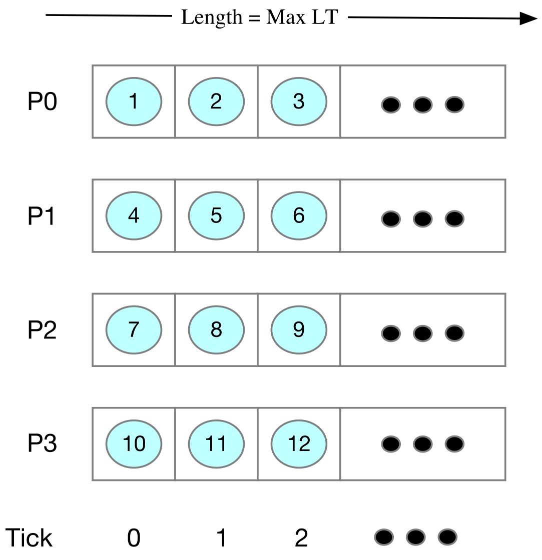


Fig. 9 Local logical trace. According to their logical time obtained, the events are inserted into an independent structure in each process

The parallel PAS2P approach, as shown in Fig. 10, performs a vertical analysis of the trace, performing for each tick calls to collective instructions, with the objective of grouping the repeatability patterns (phases) in all the processes, according to specific similarity criteria.

The parallel similarity method, illustrated in Fig. 10, requires constant synchronisation of the processes through communication collectives to know if an event is repeated, as shown in Fig. 10, which causes performance degradation due to the high number of communications when the number of processes increases.

Due to the above problem, we propose an extension of the parallel similarity method, which manages to minimise the communications caused by the synchronisation of events between processes.

Our proposal, illustrated in Fig. 11, looks for similarity of events in the logical trace of each process independently, isolating the analysis of the ticks in each process, analysing communication and computation events without the need to generate communications for the detection of similar patterns between processes.

Our approach performs local similarities in each process, thus avoiding a global similarity search, reducing the number of communications. The analysis of the similar characteristics between each event is performed tick by tick independently for each process. As shown in Fig. 11, when two events have a different communication type, it is stored in a temporary structure by assigning a phase identifier. When two events have the same type of communication, it is analyzed that they have a similar percentage of the number of instructions. This is a parameter that the user can modify. For this case, the similarity parameter must be greater than or equal to 85%. When this requirement is met, the weight of the phase is increased. Therefore we define the *Weight Vector* as the frequency with which each phase is repeated.

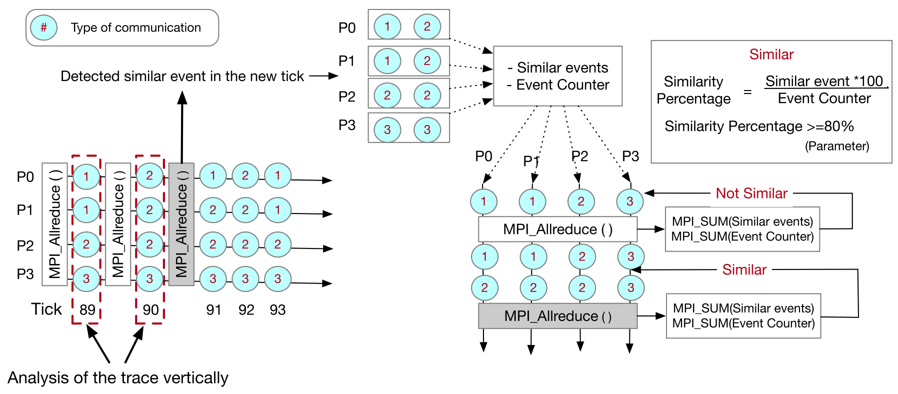
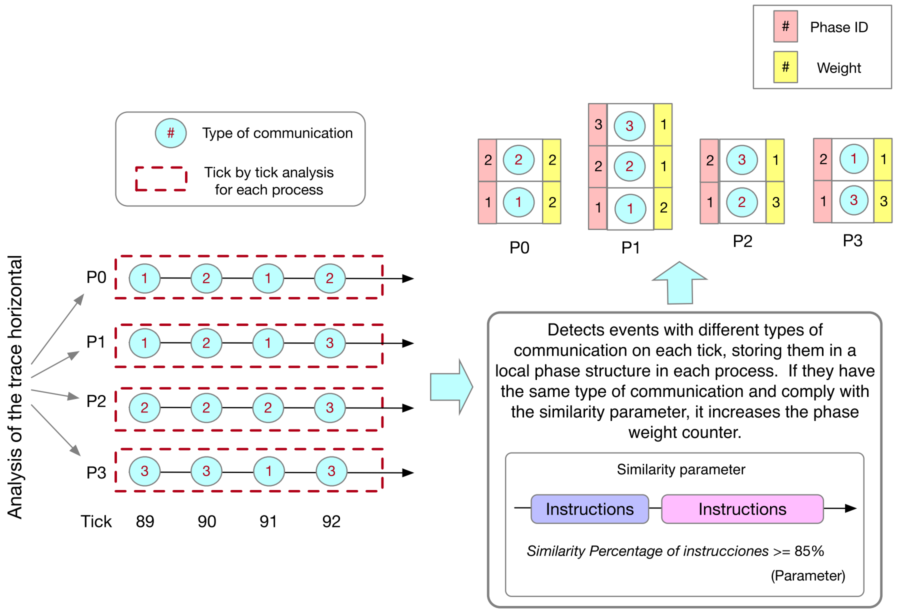


Fig. 10 Parallel similarity algorithm. The algorithm analyzes the trace vertically, i.e., all the processes perform the analysis of events simultaneously and jointly. When analyzing the events in search of a pattern, it is necessary to use some synchronization mechanism, which increases the number of communications between them



Phase ID
Weight
Type of communication
Tick by tick analysis for each process
Analysis of the trace horizontal
P0 P1 P2 P3
Tick 89 90 91 92
Detects events with different types of communication on each tick, storing them in a local phase structure in each process. If they have the same type of communication and comply with the similarity parameter, it increases the phase weight counter.
Similarity parameter
Instructions
Instructions
Similarity Percentage of instrucciones >= 85%
(Parameter)

Fig. 11 Extension parallel similarity algorithm. The trace analysis is performed horizontally; each process analyzes its trace independently in search of a repeatability pattern. The proposed analysis mechanism reduces the communication between PAS2P events

It is important to note that the procedure described above is performed using the non-dependency of SPMD application processes. We describe our proposal in the following steps:

1. A startpoint and endpoint is created with the event of the first tick of the logical trace, storing it in a phase structure.
2. Each event of the phase structure is compared with the next tick of the logical trace, verifying that the phase exists, following these criteria:

   (a) If the event does not occur with the same type of communication, the event is added to the phase structure by advancing one tick on the logical trace and returning to step 1.
   (b) If the event occurs with the same type of communication, the following criteria must be met for a phase to exist:

i. The number of instructions between the two events must be similar (85% similarity or more).

   - If it is similar, the phase weight increases and it advances one tick in the logic of the process.
   - If it is not similar, it is saved as a new phase in the phase structure and it advances one tick in the logic of the process.

3. We go back to step 1 to create a Startpoint from the tick at which the last saved phase ends

Once the temporary phase structure and the temporary weight structure have been created, the relevant phases must be defined. A phase is relevant when the weight vector multiplied by the execution time of the phase represents the execution time of the entire application. This representativeness is considered to be given if the phase represents at least 1% of the total execution time of the entire application. The user will determine this value.

Each process stores the information of its relevant phases in the time structure stored in the main memory. This stored information will allow us to predict the execution time of the application preliminarily. Equation 1 is used to predict the execution time (PET) of the application in each process. When we multiply the execution time of each significant phase ($PhaseET_i$) by its weight (W_i) (defined as the number of times significant phases which are repeated), we obtain the application's execution time.

$$\text{PET} = \sum_{i=1}^{n} (\text{PhaseET}_i) * (W_i), \text{ n is the number of phases.} \tag{1}$$

4.4 Method of selection of the representative process of the SPMD application

At the end of the pattern identification stage, we obtain relevant phases with their respective weights. With this information, we can make a preliminary prediction of the application's execution time. The prediction is made by applying Eq. 1 to each process independently.

The preliminary prediction made by each process will allow us to select the process that obtains the lowest degree of error concerning the real execution time of the application. The proposed mechanism is based on the characteristics of SPMD applications, where each process has similar computational behavior. To clarify the selection mechanism, we present Fig. 12 with the realized strategy.

When selecting the process with the lowest degree of error in the preliminary prediction, the information of the relevant phases with their respective weights is stored in a file named Phase_Table.

The Phase_Table is the result of our analysis proposal. Figure 13 shows an example of the content of the Phase_Table file obtained when executing an application with 64 processes. The content of the Phase_Table file is given by rows, representing phases, whose starting point and endpoint are represented by the first and second columns defined by the number of sends where the phase occurs. The third and fourth columns represent the phase ID. Finally, the fifth column represents the phase weight of the SPMD application.

To construct the signature, we use the *libpas*2*p* library. This library interacts with the application, as well as with the external libraries. An issue is how to detect relevant phases during the execution of the application. So, to build the signature, we re-run the application by loading the *libpas*2*p* library and Phase_Table phase file for instrumenting and measuring the occurrences of the relevant phases of the SPMD application.

Once the signature is created, we can run the signature on the target machines. This is done by measuring from the point where a phase begins to the point where it ends. Then, we repeat this method and proceed to execute all the constituent phases. Once we have the execution time of each phase and the weights of each phase, in

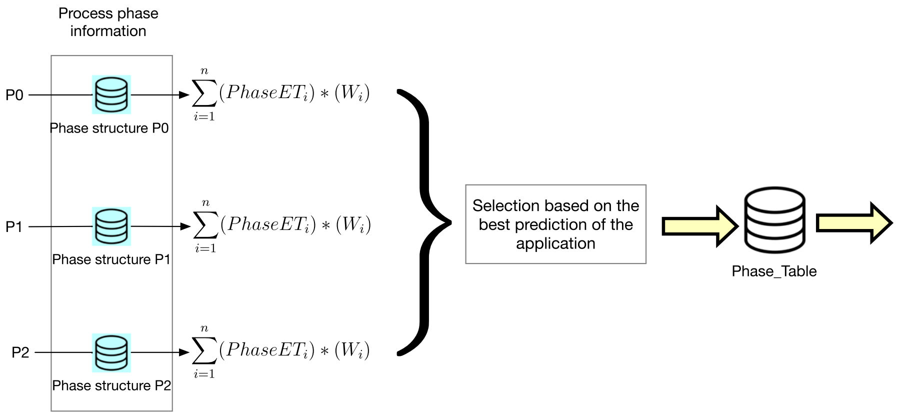
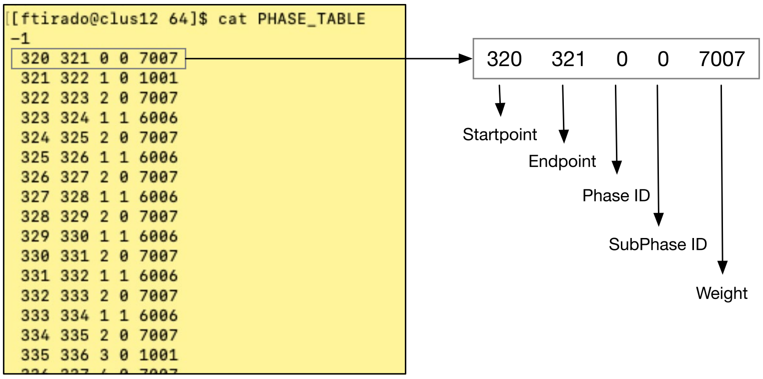

Process phase information
P0
Phase structure P0
P1
Phase structure P1
P2
Phase structure P2
$\sum_{i=1}^{n}(PhaseET_i) * (W_i)$
$\sum_{i=1}^{n}(PhaseET_i) * (W_i)$
$\sum_{i=1}^{n}(PhaseET_i) * (W_i)$
Selection based on the best prediction of the application
Phase_Table

Fig. 12 Selection of the representative process of the application. Each process performs a preliminary prediction by comparing its prediction time with the execution time of the real application. Then, the process with the lowest margin of error is selected



```
[ftirado@clus12 64]$ cat PHASE_TABLE
-1
 320 321 0 0 7007
 321 322 1 0 1001
 322 323 2 0 7007
 323 324 1 1 6006
 324 325 2 0 7007
 325 326 1 1 6006
 326 327 2 0 7007
 327 328 1 1 6006
 328 329 2 0 7007
 329 330 1 1 6006
 330 331 2 0 7007
 331 332 1 1 6006
 332 333 2 0 7007
 333 334 1 1 6006
 334 335 2 0 7007
 335 336 3 0 1001
```

Fig. 13 Contents of the phase file, Phase_Table. The selected process stores the information of its representative phases, startpoint, endpoint, phase identification, and weight in a file named Phase_Table

order to predict the execution time of the whole application, we multiply the execution time of each phase by its weight, as shown in Eq. 1.

5 Experimental results and validation

In this section, we validate our SPMD application analysis proposal, improving the analysis time against the parallel PAS2P model proposed in [21], obtaining a prediction in a bounded time (signature execution time). The experimental methodology consists of running a set of applications and increasing the number of processes to validate our analysis proposal by scaling the applications in order to compare with the parallel analysis version of PAS2P.

The set of experiments we carried out allowed us to obtain the time of our proposed extension of the parallel analysis (EPAS), the number of events obtained from the analysis stage, and the size of the trace file generated by application instrumentation. We ran the signature of each application to predict its execution time and evaluate the prediction quality of each signature.

To evaluate the prediction quality and validate the proposed methodology, we performed an experimental evaluation on the target machines described in Table 2. We present the results for the SP, BT, and LU applications of NPB [3]. For the first set of tests executed on DELL cluster, the SP application was compiled for 36 to 441 processes, using class D as the workload with 1000 iterations. The BT application was compiled for 36 to 441 processes, using class C as workload with 5000 iterations. The LU application was compiled for 32 to 256 processes, using class D as workload with 600 iterations. Finally, the NBODY application was compiled for 32 to 256 processes, using 5000 particles with 500 iterations. For the second set of tests executed on the BEM production cluster, the BT application was compiled for 256 to 900 processes, using class D as workload with 3000 iterations. The SP application was compiled for 256 to 1024 processes, using class E as the workload with 1000 iterations.

Table 2 Cluster characteristics

Cluster	Characteristics	Software
DELL	AMD Opteron^{TM} 6200 1.60GHz, 8 nodes (512 cores), 64 GB RAM per node, Interconnection Infiniband QDR	Linux versión 2.6, Open MPI 1.6.5, gcc 4.4.7, PAPI 5.4
BEM	2×12 Intel Haswell 2.30 MHz, 720 node (17280 cores), 64 GB RAM per node, Interconnection Infiniband FDR	

To obtain the results, we use the experimental methodology illustrated in Fig. 14. As shown in this figure, we run the application on a base machine and then extract its signature. Next, the signature is executed on a target machine to predict the execution time (PET). Finally, we run the entire application on the target machine to compare the predicted execution time with the effective execution time of the application, obtaining predicted execution time error (PETE).

5.1 Performance of the analysis stage

The analysis stage of the parallel approach is complex and costly due to the constant communications between the processes making the analysis time increase as the application scales. This is why we present the EPAS proposal and validate it with a set of SPMD applications, increasing the processes in each execution and comparing it with the parallel analysis of PAS2P [21], as shown in Table 3.

Table 3 presents different SPMD applications executed on the DELL cluster, with their respective execution time 'AET', presented in the third column. The trace file generation in the instrumentation stage generates a certain number of events presented in the fifth and eighth columns, which are proportionally related to the application's behavior. Some applications, such as BT, increase the number of events when scaling, whereas others do not. Finally, the 'TFAT' corresponding

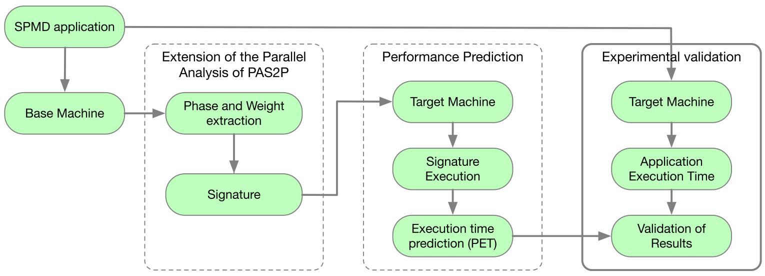


Fig. 14 Experimental methodology. First, the MPI parallel application is executed on a base machine, jointly, PAS2P instruments the application to obtain its signature. Then, the signature, which corresponds to representative segments of the application, is executed on a target machine predicting its execution time. Finally, the real execution time of the application on a target machine is compared with the prediction time, obtaining the associated prediction error

to the sixth and ninth columns shows the time required for PAS2P to perform the application analysis, i.e., create the application model and extract the phases.

On the other hand, in Table 3, it can be observed that the analysis times are proportional to the size of their trace files. With the EPAS method, the analysis times (TFAT) are considerably shorter than the parallel approach observed in the 'TFAT SpeedUp' column. The gain is due to the independent analysis model of each process, which reduces communication for event synchronization purposes. In the 'Parallel approach TFAT' column, we observe an increase in the execution time of the analysis when the application is executed with 64 processes; this is because the Dell machine has 64 cores per node and starts using the Infiniband network. The time of our proposal, observed in column 'EPAS TFAT', does not present a considerable increase in the analysis time when the application uses the network because our method reduces the PAS2P communications between processes, improving its performance.

When analyzing the results of Table 3, it is observed that the analysis time of our EPAS proposal is less in all the tests than the parallel analysis time, managing to decrease the analysis time on average by 93% and reaching a speedup of 29 for the NBODY application executed with 256 processes. Furthermore, When analyzing the variation of the analysis time when the application grows in the number of processes, we can calculate the coefficient of variation of each of the test applications, achieving for the EPAS proposal an average coefficient of variation of 42% compared to 98% of the parallel approach, this means that our proposal presents a better behavior when the application scales.

Complementing the behavior of our proposal in relation to the scalability effect, in Fig. 15 we present the execution time of the SP application's analysis executed on the DELL cluster, using the parallel approach versus our proposed Extension of the Parallel Analysis for SPMD applications. We observe a flattening of the EPAS proposal curve as the number of processes increases, compared to the parallel analysis curve, which shows an accelerated growth rate as the number of processes increases. Our proposal does not consider PAS2P communication between events, which favors its performance. However, the maximum scalability achieved was for 441 processes due to hardware limitations. To evaluate scalability on a larger scale, we can use the P3S methodology [14], which allows us to predict the scalability of message passing applications on a target system.

Table 4 presents the BT and SP application executed in the BEM production cluster, with a higher workload. The application was executed from 256 to 1024 processes. It is important to mention that several users are running applications in the cluster, which can interfere with the measurements, mainly when the application scales to a high number of processes due to the intensive use of the network. Nevertheless, in these measurements, we can observe that when increasing the number of processes, the analysis time of the EPAS proposal does not present significant alterations.

Table 3 Comparison of performance measures of parallel analysis versus EPAS proposal, executed on the DELL cluster

Appl.	Number processes	AET	EPAS trace size (GB)	EPAS number of events	EPAS TFAT(Sec)	Parallel approach trace size (GB)	Parallel approach number of events	Parallel approach TFAT (Sec)	TFAT speedup
LU	32	5440.3	2.9	979,680	15.21	2.8	979,680	45.18	2.97
	64	2483.19	6.1	979,680	25.36	5.9	979,680	68.39	2.70
	128	1167.25	12.5	979,680	25.65	12.1	979,680	193.87	7.56
	256	616.16	25.8	979,680	28.56	24.9	979,680	342.16	11.98
NBODY	32	2171.64	8.0	5,025,003	52.65	7.7	5,025,003	406.28	7.72
	64	1456.20	16.0	5,025,003	79.81	16.0	5,025,003	550.87	6.90
	128	820.97	33.0	5,025,003	92.87	32.0	5,025,003	1574.85	16.96
	256	435.30	67.0	5,025,003	103.58	64.0	5,025,003	3000.21	28.97
BT	36	3119.83	1.1	540,139	6.24	1.1	540,139	23.25	3.73
	64	1775.9	2.5	720,175	9.60	2.5	720,175	37.85	3.94
	121	998.74	6.3	990,229	14.22	6.3	990,229	168.03	11.82
	256	620.41	19.9	144,0319	25.67	19.9	1,440,319	427.50	16.65
	441	468.74	45.1	1,890,409	41.41	45.1	1,890,409	1033.80	24.96

AET, application execution time; EPAS, extension of the parallel analysis to SPMD; TFAT, tracefile analysis time

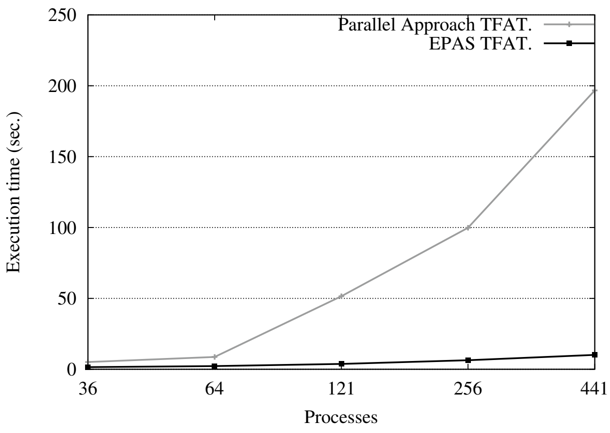


Fig. 15 Evaluation of EPAS performance in the SP application when increasing the number of processes

5.2 Evaluation of the prediction quality in target machine

To evaluate the quality of the EPAS prediction, we have obtained the application's signature on a base machine. Then, the signature is executed on a target machine to perform the execution time measurements of each phase, and we multiply it by its weights, obtaining a prediction of the SPMD application execution time. As is shown in Table 5, we run the applications by incrementing the number of processes to verify the Prediction Execution Time when SPMD applications scale.

Table 5 shows the Application Execution Time (AET), the prediction time of the EPAS and the Prediction Time of the Parallel Approach. We also present the PETE Predicted Execution Time Error obtained when executing the signature of each application. Finally, we present the SET that corresponds to the sum of the execution time of all the phases that constitute the application's signature.

The results in Table 5 show that our EPAS approach has an average prediction quality of 97.2%, compared to the parallel approach, which achieved an average prediction quality of 98.6%. In addition, we observe that the Signature Execution Time

Table 4 TFAT of our proposed EPAS executed on the BEM production cluster

Application	Number processes	AET (seg.)	EPAS TFAT (Sec)
BT	256	2590.76	17.25
	529	1304.10	25.82
	900	3344.95	34.40
SP	256	10,798.99	14.22
	512	6521.59	18.81
	1024	6224.42	35.56

AET, application execution time; EPAS, extension of the parallel analysis for SPMD; TFAT, tracefile analysis time

Table 5 AET prediction for the parallel approach and proposed EPAS implementation, using the DELL cluster

Appl.	Number processes	AET	EPAS PET (Sec)	EPAS PETE (%)	EPAS SET (Sec)	Parallel approach PET (Sec)	Parallel approach PETE(%)	Parallel approach SET (Sec)
SP	36	4252.63	4316.78	1.5	58.87	4295.97	1.0	58.71
	64	3342.22	3416.31	2.2	49.63	3402.23	1.8	49.16
	121	2381.40	2424.62	1.8	40.85	2394.34	0.5	39.57
	256	1203.04	1244.95	3.4	30.39	1236.22	2.7	28.16
	441	710.97	749.42	5.1	28.03	728.90	2.5	23.37
BT	36	3119.83	3165.05	1.4	26.11	3148.85	0.9	25.28
	64	1775.90	1825.98	2.7	21.80	1796.57	1.2	20.98
	121	998.74	1073.79	7.0	21.08	1025.28	2.6	18.82
	256	620.41	680.76	8.9	21.24	651.12	4.7	19.16
	441	468.74	525.76	10.8	20.54	517.47	9.4	18.27

AET Application execution time, *EPAS* Extension of the parallel analysis for SPMD, *PET* Prediction execution time, *PETE* Predicted execution time error, *SET* Signature execution time

(SET) is similar in both approaches, achieving a significant reduction compared to AET, reaching values lower than 1% of the Application Execution Time (AET).

Table 6 presents the prediction quality using the BEM production cluster. The BT and SP applications have a higher workload than the same applications running on the DELL cluster. The EPAS proposal has an average prediction quality of 97% for the BT application and 94% for the SP application. We can also observe that the error remains stable as the application's number of processes increases. The BT and SP applications have difficulties when scaling to a high number of processes, mainly due to the intensive use of the production cluster, but despite that, PAS2P detects it and delivers an execution time prediction with an error of 5%. Finally, the execution time of the signature (SET) does not present great alterations when the application is executed with a high number of processes.

We must point out that the construction of the signature is created only one time on the base machine. Then, to predict the performance, we migrate the signature to the target machines without analysing the application again.

6 Conclusion and future work

The PAS2P methodology allows us to generate a model of a parallel application and automatically extract its most significant phases to create a signature whose execution allows us to predict the application's performance on different parallel machines. However, the analysis performed by PAS2P to generate the application model is costly in terms of execution time due to the high communication between PAS2P events, which degrades performance as the number of processes increases.

For this reason, we propose a new model for the analysis stage of PAS2P, based on the behavior of SPMD applications. The proposal minimizes the communications between PAS2P events, reducing the execution time of the analysis stage by 84.63%, compared to the parallel approach, obtaining a prediction quality below 6% on average. In addition, we propose to use the same resources used in the execution of the application to analyze the application independently for each process.

Table 6 AET prediction for the parallel approach and proposed EPAS implementation, using the BEM production cluster

Application	Number processes	AET (seg.)	EPAS PET (seg.)	EPAS PETE (%)	EPAS SET (Sec)
BT	256	2590.76	2479.46	4	29.32
	529	1304.10	1293.16	1	22.28
	900	3344.95	3532.16	5	33.96
SP	256	10798.99	11538.61	6	133.34
	512	6521.59	7035.25	7	86.55
	1024	6224.42	6550.59	5	92.08

AET Application execution time, *EPAS* Extension of the parallel analysis for SPMD, *PET* Prediction execution time, *PETE* Predicted execution time error, *SET* Signature execution time

When executing the application signature, which consists of the relevant phases extracted from the MPI application, we measure the time of each phase on a target machine and multiply it by its weight to predict the application execution time. Our approach keeps the prediction error rates similar concerning the PAS2P parallel approach and keeps the signature execution time (PET) similar when the application scales.

As future work, we will focus on applications with irregular behavior. The independent analysis to extract the phases per process can help us characterize irregular applications' behavior, where each process can have different behavior from the rest.

Acknowledgements 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 and partially funded by the Fundacion Escuelas Universitarias Gimbernat (EUG). Appreciation to The Wroclaw Centre for Networking and Supercomputing, Poland for access to its supercomputing resources.

Funding Open Access Funding provided by Universitat Autonoma de Barcelona.

References

1. Atallah MJ, Blanton M (2009) Algorithms and theory of computation handbook, volume 2: special topics and techniques. CRC press
2. Attig N, Gibbon P, Lippert T (2011) Trends in supercomputing: the European path to exascale. Comput Phys Commun 182(9):2041–2046
3. Bailey DH, Barszcz E, Barton JT (1991) The nas parallel benchmarks. In: Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, Supercomputing, vol 91, pp 158–165
4. Canillas JM, Wong A, Rexachs D, Luque E (2011) Predicting parallel applications performance using signatures: the workload effect. In: 2011 9th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA). IEEE, pp 299–300
5. DeRose L, Homer B, Johnson D, Kaufmann S, Poxon H (2008) Cray performance analysis tools. In: Tools for high performance computing. Springer, pp 191–199
6. Geimer M, Saviankou P, Strube A, Szebenyi Z, Wolf F, Wylie B (2012) Further improving the scalability of the scalasca toolset. Appl Parallel Sci Comput 463–473
7. Hilbrich T, Schulz M, de Supinski BR, Müller MS (2010) Must: a scalable approach to runtime error detection in mpi programs. In: Tools for high performance computing 2009. Springer, pp 53–66
8. Jayakumar A, Murali P, Vadhiyar S (2015) Matching application signatures for performance predictions using a single execution. In: 2015 IEEE International Parallel and Distributed Processing Symposium. IEEE, pp 1161–1170
9. Labarta J (2010) New analysis techniques in the cepba-tools environment. In: Tools for high performance computing 2009. Springer, Berlin, pp 125–143
10. Lamport L (1978) The ordering of events in a distributed system. Commun ACM 21(7):558–565

11. Mantovani F, Calore E (2018) Multi-node advanced performance and power analysis with paraver. In: Parallel computing is everywhere (serie: advances in parallel computing), vol 32. IOS Press, pp 723–732
12. Mijaković R, Firbach M, Gerndt M (2016) An architecture for flexible auto-tuning: the periscope tuning framework 2.0. In: 2016 2nd International Conference on Green High Performance Computing (ICGHPC). IEEE, pp 1–9
13. Panadero J, Wong A, Rexachs D, Luque E (2013) A tool for selecting the right target machine for parallel scientific applications. In: ICCS, pp 1824–1833
14. Panadero J, Wong A, Rexachs D, Luque E (2017) P3s: a methodology to analyze and predict application scalability. IEEE Trans Parallel Distrib Syst 29(3):642–658
15. Parker S, Mellor-Crummey J, Ahn DH, Jagode H, Brunst H, Shende S, Malony AD, DelSignore D, Tschuter R, Castain R et al (2017) Performance analysis and debugging tools at scale. Exascale Sci Appl Scalabil Perform Portabil 17–50
16. Perelman E, Polito M, Bouguet JY, Sampson J, Calder B, Dulong C (2006)Detecting phases in parallel applications on shared memory architectures. In: Parallel and Distributed Processing Symposium, International, p 68
17. Shende S, Malony A, Allen G, Carver J, Choi S, Crick T, Crusoe M (2016) Using tau for performance evaluation of scientific software. In: Workshop on sustainable software for science: practice and experiences, p 1686
18. Shende S, Malony A, Morris A (2012) Improving the scalability of performance evaluation tools. Appl Parallel Sci Comput 441–451
19. Sherwood T, Perelman E, Calder B (2001) 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), pp 3–14
20. Terpstra D, Jagode H, You H, Dongarra J (2010) Collecting performance data with papi-c. In: Tools for high performance computing 2009. Springer, pp 157–173
21. Tirado F, Wong A, Rexachs D, Luque E (2019) Analyzing the data behavior of parallel application for extracting performance knowledge. In: 2019 International Conference on High Performance Computing and Simulation (HPCS). IEEE, pp 249–256
22. Weber M, Brendel R, Wagner M, Dietrich R, Tschüter R, Brunst H (2017) Visual comparison of trace files in vampir. In: Programming and performance visualization tools. Springer, pp 105–121
23. Wong A, Rexachs D, Luque E (2010) Parallel application signature for performance prediction. In: PDPTA 2010: Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (Las Vegas NV, July 12–15, 2010), pp 408–414
24. Wong A, Rexachs D, Luque E (2015) Parallel application signature for performance analysis and prediction. IEEE Trans Parallel Distrib Syst 26(7):2009–2019
25. Zhukov I, Feld C, Geimer M, Knobloch M, Mohr B, Saviankou P (2015) Scalasca v2: back to the future. In: Tools for high performance computing 2014. Springer, pp 1–24