P3S: A methodology to Analyze and Predict Application Scalability

Javier Panadero, Alvaro Wong, Dolores Rexachs and Emilio Luque

Abstract—Executing message-passing parallel applications on a large number of resources in an efficient way is not a trivial task. Due to the complex interaction between the parallel applications and the HPC system, many applications may suffer performance inefficiencies when they scale. To achieve an efficient use of these large-scale systems using thousands of cores, a point to consider before executing an application is to know its behavior in the system. In this work, we propose a novel methodology called P3S (Prediction of Parallel Program Scalability), which allows us to analyze and predict the scalability of message-passing applications on a given system. The methodology strives to use a bounded analysis time, and a reduced set of resources to predict the application behavior for large-scale. The experimental validation proves that the P3S is able to predict the application scalability with an average accuracy greater than 95 percent using a reduced set of resources.

Index Terms—HPC Systems, Scalability Prediction, MPI Application, Prediction of an application scalability.

1 Introduction

With the advent of multi-core processors, High Performance Computing clusters have increased the number of cores significantly [1]. The users of these systems want to get the maximum benefit from this large number of cores scaling their applications, either by reducing the execution times (Strong Scalability [2]) or increasing their workload (Weak Scalability [2]).

Executing MPI applications on a large number of resources in an efficient way is not a trivial task. Due to the complex interaction between the parallel applications and the HPC system, many applications may suffer performance inefficiencies when they scale. This problem is particularly serious when the application is executed many times over a long period of time. To achieve an efficient use of these systems using a large number of cores, a point to consider before executing an application is to know its performance behavior in the system. It is important to know this information since the ideal number of processes and resources required to run the application may vary from one system to another, due to hardware architecture differences. Moreover, it is known that using more resources does not always imply a higher performance [3]. The lack of this information may produce an inefficient use of the cores, causing problems such as not achieving the expected speedup, and increased energy and economic costs.

With the purpose of avoiding these problems and making an efficient use of the system, users and system administrators take advantage of predictive performance models.

In previous papers, we have proposed the PAS2P tool [4], which is based on the PAS2P methodology [5], which allows to predict the performance of MPI applications for a specific number of processes on target machines.

• *The authors are with the Computer Architecture and Operating System Department, Universitat Autonoma de Barcelona, Barcelona, Spain.*
E-mails: javier.panadero, alvaro.wong@caos.uab.es, dolores.rexachs, emilio.luque@uab.es

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 communicational and computational patterns, which follow specific rules. PAS2P [4] identifies the application phases in a transparent and automatic way, and it generates the Application Signature (PAS2P Signature), which contains only the relevant application phases (the phases which have an impact on the application's performance) and their repetition rates (weights). The PAS2P Signature execution allows us to analyze and predict the application performance in a efficient way on target machines, covering approximately 95% of the total application code in 1% of the application execution time.

To predict the application performance, the PAS2P Signature has to be executed using the same number of resources required to execute the application. In this contribution, we go one step further, proposing the Parallel Program Scalability (P3S) methodology. This methodology is based on the concept of the PAS2P Signature, and it allows us to analyze and predict the strong scalability behavior for MPI applications on a given target machine, using a bounded execution time and a reduced set of resources. Moreover, the methodology could help the system administrators, who are concerned with the overall resource utilization, by knowing the execution time of the application. A job schedule can maximize the system throughput, especially in high-throughput systems.

The P3S methodology is composed of three steps. The first step constists of characterizing and analyzing the communication pattern, the computational pattern (number of instructions between two communication events), and the weight of each relevant phase in a transparent way (without the application source code), from a set of executions of PAS2P signatures in small-scale. By executing this set of PAS2P Signatures, we can obtain information about the phases's behavior, as the application scales.

This information will be used in the second step to model the general behavior rules of each phase. These rules specify the phase behavior and they allow to predict its behavior as the number of processes increases. The P3S methodology is limited to applications which follow regular communicational and computational behaviour rules, as the application increases the number of processes.

From these rules, the machine-independent Scalable Logical Trace (SLT) of the application is generated, which only depends on the way in which the application was developed. The SLT can be parameterized for the number of processes for which it is desired to predict the application performance. In order to predict the performance, the SLT has to be complemented with the communicational and computational time.

To predict the computational time, we used measurable points that will be executed using reduced resources. It combines the computational time of each phase of the initial set of signatures and the computational behavior rules (computational pattern). Then, the computational time of each phase is provided to the SLT, to generate the Scaled Trace for N Processes (ST4NP), which contains all the information of the SLT, plus the computational time of each phase.

Finally, the last step of the methodology before obtaining the performance prediction, is to process the ST4NP on the target system, using a reduced number of resources, in order to obtain the communicational time of each phase in that system. To process the ST4NP, we developed a tool called Synthetic Signature (SS). The SS receives as input the ST4NP and it processes the communication and compute events of each relevant phase in the parallel system, in order to obtain the performance of the application. The aim is to measure the time of the communication events of each relevant phase in order to obtain the performance prediction of each phase. Then, we estimate the entire application run time in the system, using the PAS2P prediction equation, which aggregates all relevant phase execution times (PhaseET).

To evaluate the quality of the predicted performance, using a reduced number of resources, we conducted a set of experiments using different applications, including CG, BT, LU and SP from NPB [6], Sweep3D [7], and N-BODY. We were able to predict the execution time with an average accuracy of more than 95%.

In the following section we describe related work. In Section 3, we describe the proposed methodology. In Section 4, we present our performance prediction model. Section 5 provides the experimental results and discusses the obtained prediction results, and Section 6 presents the P3S performance, that is the time required to predict the application scalability. The last section concludes our work and future work.

2 Related works

There are some tools and methodologies focusing on predicting the scalability of HPC applications based on observations made from smaller process executions. Wenguang et al [8] propose the PHANTOM tool, which can be used for predicting scalability performance based on execution in small scale, assuming that only one node is available. Instead of executing the whole application, PHANTOM only executes a set of representative processes. Finally, a simulation of the communications is carried out to predict the application scalability. In same scope there are also studies based on the use of trace-driven simulators [19] [20] [21] to predict the application performance. These types of simulators are limited in that they do not implement the hardware characteristics of all HPC systems. Our work proposes to use the real HPC system in order to measure the interaction between the application and the hardware characteristics using the SS.

Another interesting framework is SWAPP [9], which projects the performance of MPI applications. The framework assumes that the system is not available for running the application, only benchmarks are available. The projections are developed using the performance profiles of the benchmarks. SWAPP projects the performances of compute and communication components separately and then combines the two projections to get the full application projection. Calotoiu et al [10] propose a tool to automatically generates asymptotic scaling models for each part (kernel) of the application. The model is based on fitting the performance data from a set of manageable number of small-scale performance experiments. Our methodology only focuses on modeling the application phases, which will be used to predict the application performance, instead of using the whole application.

Xu et al [11] propose a method to generate an application skeleton with which to predict the performance application, while in subsequent work [12] [13] they propose generating a communication benchmark to predict performance. Both methods are based on the application trace. Similar approaches [14] [15] are based on reducing the application to a minimal form that preserves key properties of the original one. Our methodology generates a physical application trace for a larger number of processes from small scale executions. Those will be processed in the Synthetic Signature program using a reduced number of resources to predict its performance.

There are other works based on analytical regression and machine learning methods from executions for a small number of processes. Barnes et al [16] propose studying the scalability using linear and logarithmic regression functions, isolating computation and communication to predict the application performance. Lee et al [17] present two techniques to model applications as a function of its input parameters to predict performance. The first technique is based on piecewise polynomial regression and the second technique is based on articial neural networks. Ipek et al [18] present a diferent approach based on multilayer neural networks. From a training set of the application executions, the application model is created automatically. This approach is interesting for its ease of use and its obliviousness to details of application internals. These works are based on the input parameter space to obtain the regression models and extrapolate its behavior. As we move away from the points used to generate the regression model, the prediction error is higher. By this reason, we propose in our methodology to use measurable points to reduce the error and fit the regression model.

Characterizing the communication behavior is a key goal to predict the application performance. Noeth et al [22] propose ScalaTrace, a tracing framework which introduces intra- and inter-node compression techniques, based on identifying regular communication patterns. The obtained traces are orders of magnitude smaller than traditional traces, preserving all structural and temporal-order information. The objective is similar to the PAS2P tool which reduces the application behavior in phases.

There is an extension of ScalaTrace, called ScalaExtrap tool [23], which extrapolates communication patterns of SPMD applications. The time needed to predict (Replay time) defined by ScalaExtrap is equal to the application execution time. To obtain the application prediction time is necessary to use the same number of resources required to execute the application. The P3S methodology provides an alternative to avoid using all the resources required by the application to predict its execution time. Our proposal focus on generating a SS, which can be executed in the system using a reduced set of resources.

In this paper, we present the whole P3S methodology, all of whose steps have been refined. We have improved the algorithm to generate the general equations automatically, making communication equations transparent to the user. Moreover, we have extended the SS program to process physical traces for a large number of processes using a reduced number of resources. Unlike previous published works, we have developed an extensive experimental validation, where we show the application scalability of scientific applications using small-scale signatures executions.

3 P3S (Prediction of Parallel Program Scalability) Methodology

Parallel message-passing applications are typically composed of repetitive patterns of computation and communication that are grouped in phases. These phases compose the application Signature. It is important to mention that the number of phases remains constant when the number of processes increases, but their patterns change their behavior following behavior rules. Based on the information obtained from the execution of a set of small-scaled Signatures, our objectives are to model the general behavior rules of each phase as well as to project the behavior of the communication and computational patterns to predict the application performance when the number of processes increases.

The main goal of the P3S methodology is to analyze and predict the strong scalability [3] behavior for message-passing applications on a given system using a limited set of small-scale signatures as input, as is shown in Fig. 1. The P3S methodology strives to use a bounded analysis time, and a reduced set of resources to predict the application behavior for large-scale.

The P3S methodology is composed of three steps and it is based on analyzing the repetitive behavior of parallel message-passing applications. In the first step the behavior (communication and compute) of each relevant phase is characterized. To do that, we execute a set of small-scale PAS2P Signatures. The Signatures are obtained with the PAS2P tool, and they contain only the relevant application phases (the phases which have impact on the application performance) and their repetition rates (weights). By executing this set of PAS2P Signatures, we obtain quick information about the phases's behavior, which will be used in the next methodology step to model their scalability behavior.

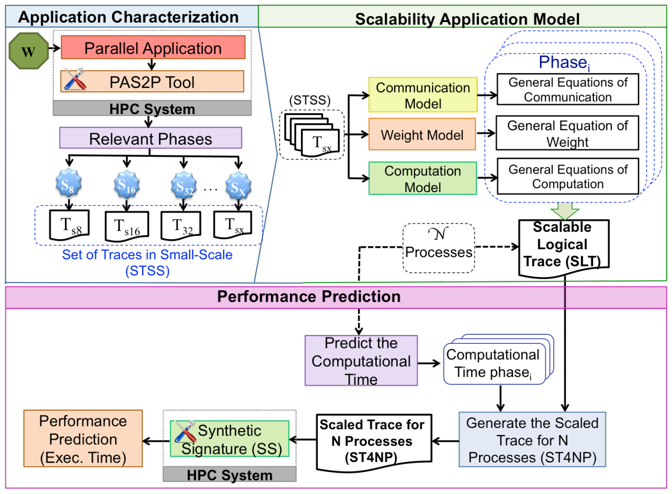

Application Characterization
Scalability Application Model
W
Parallel Application
PAS2P Tool
HPC System
Relevant Phases
S_8
S_{16}
S_{32}
...
S_X
T_{s8}
T_{s16}
T_{32}
T_{sx}
Set of Traces in Small-Scale (STSS)
(STSS)
T_{sx}
Communication Model
Weight Model
Computation Model
$Phase_i$
General Equations of Communication
General Equation of Weight
General Equations of Computation
$\mathcal{N}$ Processes
Scalable Logical Trace (SLT)
Performance Prediction
Predict the Computational Time
Computational Time $phase_i$
Generate the Scaled Trace for N Processes (ST4NP)
Scaled Trace for N Processes (ST4NP)
Synthetic Signature (SS)
HPC System
Performance Prediction (Exec. Time)

Fig. 1. Overview of P3S Methodology

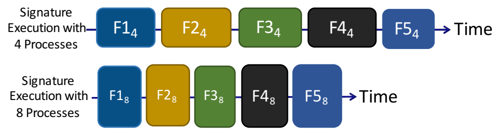
In the second step of P3S methodology, the information obtained in the previous step will be analyzed to generate the scalability application model of each relevant phase. In order to generate these models, the information obtained in the characterization step will be used to model the general behavior rules of each phase. These general rules specified the phase behavior and they allow us to predict its behavior as the number of processes increases. From these general rules the Scalable Logical Trace (SLT) of the application is generated. This is machine-independent as it only depends on the way the application was developed. The SLT can be parameterized for the number of processes for which it is desired to predict the application performance.

Finally, the last stage of the methodology consists of predicting the application performance for a large-scale of processes. In order to do that, we have developed the Syntethic Signature (SS) program. The SS receives the SLT as input, which will be specified for the number of processes to predict the application performance with the objective of predicting the computational and communication time of each relevant phase for this number of processes, to obtain the performance prediction of the application.

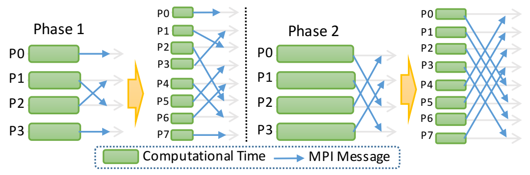
3.1 Application characterization

To generate the SLT, the behavior of each application phase (communication and compute) has to be characterized as the application scales. To do that we carry out a set of PAS2P Signature executions in small-scale. The traces generated by the Signatures will be analyzed to obtain information from each phase. When the PAS2P signature is executed in the system it generates a trace file per process, which contains information from each phase. The trace provides information about the phase id, the type of MPI primitive, the source and destination of the communication, the communication volume in bytes, the computational time in nanoseconds, and finally, the number of instructions for the computational time. This information will be used by the P3S methodology, to generate both the communication model as well as the computational model.

When we analyze the behavior of the phases, we know that as the application scales, increasing its number of processes, the communications (number of messages and des-


Signature Execution with 4 Processes
F1₄ F2₄ F3₄ F4₄ F5₄
Time
Signature Execution with 8 Processes
F1₈ F2₈ F3₈ F4₈ F5₈
Time

(a) Example of logical sequence of the application phases during the execution time for 4 and 8 processes.



(b) Behavior transition of two phases from 4 processes to 8 processes.

Fig. 2. The functional similarity relates the phases for different number of processes

tinations), the communication volume, the computational time and the number of compute instructions of a phase can all change, but the work to be carried out will still be the same, distributed among more processes.

When the number of processes of the application increases, the application patterns change their behavior, whereby it is difficult to relate them to analyze and model their evolution as the application scales. To solve this problem, it is necessary to recognize and relate the phases of the small-scale PAS2P signatures.

To relate the phases for a different number of processes, we use functional similarity. Two phases will have functional similarity, when the computation work to be carried out for both phases is the same. This happens when is the same code segment distributed between a different number of processes, changing only the structure of the communication pattern.

To relate the phases, we use a method which is based on how the sequence of phases occurs, since it does not depend on the number of processes but only the way in which the application was developed. As we can see in the Fig. 2a, the number of phases remains constant as the application increases from 4 processes to 8 processes. In Fig. 2b, phases 1 and 2 are showed in detail, we see that the behavior is different from 4 processes to 8 because the phases have different communication patterns and different computational times between them. However, the work carried out by the phases is the same, distributed between a different number of processes, because they are in the same logical position in the application.

Once the phases have been related, their communication pattern, the computational pattern and the weight of each phase are analyzed and modeled to generate the general behavior rules, to construct the SLT for a N number of processes. The input parameters of the general behavior rules will project the SLT for a specific number of processes. The SLT is composed of the intrinsic parameters of each phase needed to model the scalability of the parallel application, which are: the phase ID, communication pattern (spatial and volume parameters), number of instructions of the computational time and phase weight. The SLT is generated per process instead of a global trace, with the objective of modeling each process independently.

3.2 Scalability Communication Model

The scalability communication model comprises the general behavior equations and the data volume equations for each communication of each phase. The general behavior equations calculate the message destination from the source, while the data volume equations calculate the message size.

Once the phases have been related, the predicted data volume of each communication will be obtained by mathematical regression models, while for obtaining the general communication rules (Source-Destination), an algorithm has been proposed. This algorithm is based on obtaining the communication equations (eq.$_{processes.phase.comm}$) for each phase (Local Equations) of the set of small-scale signatures executed, which identifies the communication pattern for each phase. From these Local Equations, as is shown in Fig. 3 for phase 1, the General Equations (GE_{Ph_i}) are modeled for each phase, and they are used to predict the communication pattern for a N number of processes.

3.2.1 *Generating the Local Equations*

This stage is composed of two steps. A first step of analysis, in which the information obtained in the characterization stage is analyzed to obtain information about the communication pattern of each phase. And a second step of modeling, where the Local Equations for each phase are generated. The Local Equations are an abstract representation of the communication pattern of a phase for a specific number of processes. During the analysis step, the dependencies between processes, the pattern type: Static (Mesh, Ring, etc.) or Dynamic (Exchange, Permutation, etc.), and the distance matrix between processes are obtained for each phase. All this information is provided to the second step to generate the Local Equations. In this second step, the Local Equation of each communication of the phase is obtained using an algorithm of identification. This algorithm is based on the fact that the application is well developed, and it executes a deterministic communication pattern for all the processes, without non-predictive conditional sentences as the number of processes increases. The algorithm compares the source-destination of each send primitive for all the processes of the phase, in order to identify the specific rule to obtain the destination from the source for that number of processes. Moreover, the repeatability of a set of communications is sought to generate easier equations and simplify the analysis and modeling of the General Equations.

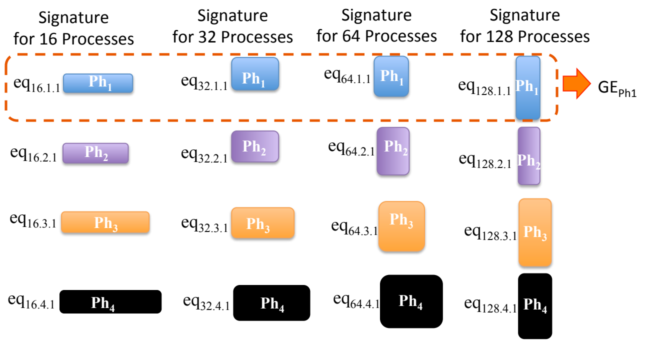
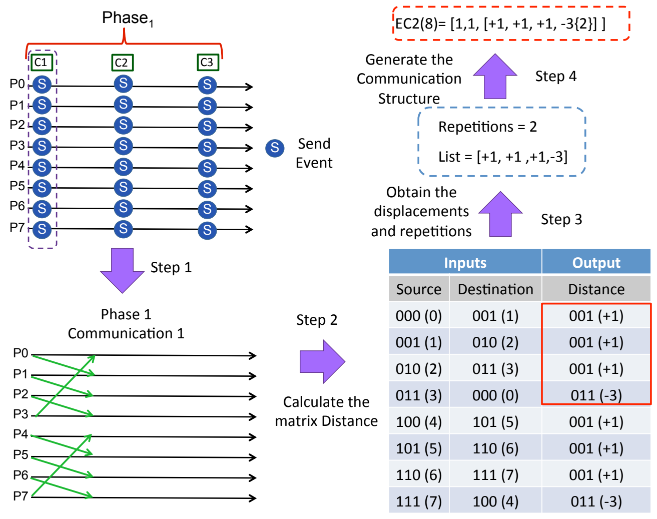


Fig. 3. P3S obtains the General Eq. from Local Equations


Phase$_1$
C1
C2
C3
P0
P1
P2
P3
P4
P5
P6
P7
S
Send Event
Step 1
Phase 1
Communication 1
Step 2
Calculate the matrix Distance
Obtain the displacements and repetitions
Step 3
Repetitions = 2
List = [+1, +1 ,+1,-3]
Generate the Communication Structure
Step 4
EC2(8)= [1,1, [+1, +1, +1, -3{2}]]

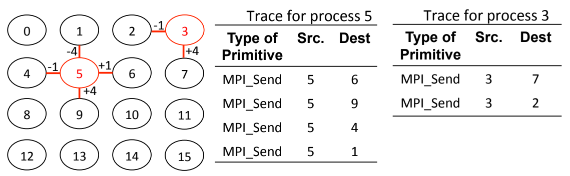
Inputs		Output
Source	Destination	Distance
000 (0)	001 (1)	001 (+1)
001 (1)	010 (2)	001 (+1)
010 (2)	011 (3)	001 (+1)
011 (3)	000 (0)	011 (-3)
100 (4)	101 (5)	001 (+1)
101 (5)	110 (6)	001 (+1)
110 (6)	111 (7)	001 (+1)
111 (7)	100 (4)	011 (-3)

Fig. 4. Example how P3S generates Local Equations.

We use two different structures to generate the Local Equations because the way to predict the communication pattern is different, depending on the pattern type. If the pattern is dynamic, the way to obtain the destination processes is based on the exchange of certain numbers of source bits, which are called *bits involved*. For this pattern, the EC1 structure is used. In case of static patterns, to obtain the destination processes, the distance between the processes and the repeatability of the communications are identified. For this pattern the EC2 structure is used. The EC1 structure has as parameters the phase number (#Phase), the number of communication in the phase (#Comm), the algorithm type (Exchange, Permutation) and the list of *bits involved*, (*EC1(p)* = {*#Phase, #Comm, Type* , *List of bits involved* }). The EC2 structure has the number of phase, the number of communication in the phase and the list of communication distances and its number of repetitions (*EC2(p)* = {*#Phase, #Comm, list[communication distances*{*#repetition*} *]* }).

Fig. 4 shows a brief example of the procedure. We have a phase with 8 processes (p=8) and three communications. These three communications compose the communication pattern of the phase, which is static because it is a 4x2 mesh, identified in the analysis phase. Then, the EC2 structure will be used. If we focus on the first communication of the pattern (Step 1), we generate the matrix distance between the source and the destination (Step 2). Then, we search for repetitions, in this case, the sequence {+1,+1,+1,-3} is repeated two times (Step 3), once for processes 0 to 3 and another for processes 4 to 7. Once we have the sequences and repeatability, we create the Local Equation with the structure of communication EC2 (Step 4).

Depending on the communication algorithm of the application, it can be possible that the application trace does not have the same number of communications for all the processes. Fig. 5 shows an example of an SPMD (Single Program Multiple Data) application, where process 5 has four communications and process 3 has two communications, due to be a border process. With the aim of obtaining the correct local equations, the traces must have the same number of communications for all the processes. Otherwise the algorithm could not relate the communications properly.



Trace for process 5

Type of Primitive	Src.	Dest
MPI_Send	5	6
MPI_Send	5	9
MPI_Send	5	4
MPI_Send	5	1

Trace for process 3

Type of Primitive	Src.	Dest
MPI_Send	3	7
MPI_Send	3	2

Fig. 5. Communication pattern of an SPMD Application.

To solve this problem, the algorithm selects the process with the maximum number of communications, and it completes the trace for the other processes with null communications, until all the processes have the same number. To complete the traces with null communications, the algorithm is based on the fact that the application is written in a deterministic way and the sends follow a logical order with a specific behavior. We show an example in Fig. 5. Process 5 has 4 communications, with a vector of distance [+1,+4,-1,-4], while process 3 has 2 communications with a vector of distance [+4,-1]. Comparing the vectors, the communications that do not appear in the trace for process 3 are the first and the last (+1,-4). Then, the algorithm completes the trace with null communications, which are marked with an 'X' in the destination of the message.

3.2.2 *Modeling the General Equations*

From the Local Equations, the General Equations of each phase are modeled. They will be used to predict the communication pattern for a *N* number of processes. To generate the General Equations of each phase, the Local Equations are analyzed in order to model the evolution of the communication pattern. The method consists of comparing the Local Equations to model by a function, as the parameters change their values as the number of processes increases, as is shown in Fig. 6. The General Equations have the number of processes to predict as input. The structure of these equations is the same as the one for the Local Equations, the difference being that the parameters have been modeled as a function.

In some applications, when the number of processes increases, the communication pattern expands communicating with new processes and new communications appear. To predict these communications, the algorithm models the behavior of how these new communications will appear (number of communications and their destination) for N processes. Fig. 7 shows this procedure. The signature traces of processes 2, 4, 8 and 16 were obtained by the signature executions, and we want to predict the communication pattern for 64 processes. As we can observe, when the number of processes is increased per two, a new communication appears. To predict the communication pattern for 64 processes, first of all, the algorithm models a function to predict the number of communications of the phase as the application scales. The function has as input parameter

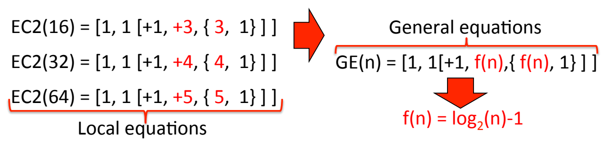

EC2(16) = [1, 1 [+1, +3, { 3, 1}]]
EC2(32) = [1, 1 [+1, +4, { 4, 1}]]
EC2(64) = [1, 1 [+1, +5, { 5, 1}]]
Local equations
General equations
GE(n) = [1, 1[+1, f(n),{ f(n), 1}]]
f(n) = log$_2$(n)-1

Fig. 6. P3S Models General Equations from Local Equations

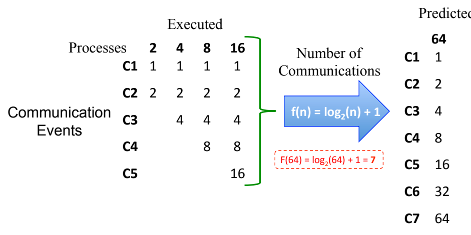


Fig. 7. Generation of new communications

the number of processes to predict the number of communications. When we the number of communications of the phase has been obtained, we apply the General Equations to predict the destination.

Once we have modeled the General Equations and the communication volume equations, which are obtained by regression models, we have the evolution of the communication pattern (spatial and volume) for each phase of the application. These equations will be used to generate the SLT. The SLT has to be complemented with the General Equations of the weight and the compute, which will be modeled in the next section.

In order to follow the methodology, we present an example using the BT from the NPB NAS benchmarks [6]. In such an example, we use a set of BT small-scale PAS2P signatures to predict the General Equations and the Communication Volume Equations for each one of its phases.

Table 1 shows the predicted equations for each phase of the BT application. Each MPI primitive of each phase is modeled with the objective to predict their the communication pattern evolution.

These equations have been obtained by the execution of 3 small-scale signatures (256, 324 and 484), using class E with 1000 iterations as input. Due to the type of pattern (static), the EC2 type structure has been used to model the General Equations. As we mention in this section, these equations have the number of processes for which it is desired to predict the communication pattern as input parameters. For this particular example we have used 4096 processes as input parameters, considering that it is far enough from the number of processes used in the executed signatures. For readability, with the objective of showing the outputs of these equations for 4096 processes, we show the predicted values for process 0.

3.3 Computational Model

In this section we present the computational model. This model allows us to predict the computational time with high accuracy for a large number of processes. To predict the computational time, we use a set of measurable points per phase, using as input data phase the computational time of the initial small-scaled signatures obtained in the previous phase.

In previous section (Section 3.2) we show how P3S obtains the communicational model for N number of processes creating the Scaled Trace for N Processes (ST4NP). If we want to predict the scalability of the application, we need to predict the evolution of the compute of each phase for N number of processes. That's the reason why we have to insert the computational time between the communication events into the SLT.

TABLE 1
Communication Model of BT application obtained from the execution of small-scale signatures

Communication Pattern Model		Predicted Communication Pattern for 4096 processes		
Global Equations (Dest)	Comm. Volume Equations (Bytes)	MPI Primitive	Src. (P0) - Dest.	Comm. Volume
Phase 1				
1) $f(n) = 1$	$f(n) = 3E + 08n^{(-1.015)}$	ISend	1	64651
2) $f(n) = \sqrt{n} - 1$	$f(n) = 3E + 08n^{(-1.015)}$	Irecv	63	64651
3) $f(n) = 0$	$f(n) = 0$	Wait	0	0
4) $f(n) = 0$	$f(n) = 0$	Wait	0	0
Phase 2				
1) $f(n) = \sqrt{n} - 1$	$f(n) = 4E + 07n^{(-1)}$	ISend	63	9765
2)$f(n) = 1$	$f(n) = 4E + 07n^{(-1)}$	Irecv	1	9765
3)$f(n) = 0$	$f(n) = 0$	Wait	0	0
4)$f(n) = 0$	$f(n) = 0$	Wait	0	0
Phase 3				
1) $f(n) = n - \sqrt{n}$	$f(n) = 3E + 08n^{(-1.015)}$	ISend	4032	64651
2) $f(n) = \sqrt{n}$	$f(n) = 3E + 08n^{(-1.015)}$	Irecv	64	64651
3) $f(n) = 0$	$f(n) = 0$	Wait	0	0
4) $f(n) = 0$	$f(n) = 0$	Wait	0	0
Phase 4				
1) $f(n) = 2 * \sqrt{n} - 1$	$f(n) = 4E + 07n^{(-1)}$	ISend	127	9765
2) $f(n) = n * -\sqrt{n} + 1$	$f(n) = 4E + 07n^{(-1)}$	Irecv	4031	9765
3) $f(n) = 0$	$f(n) = 0$	Wait	0	0
4) $f(n) = 0$	$f(n) = 0$	Wait	0	0
Phase 5				
1) $f(n) = n * -\sqrt{n} + 1$	$f(n) = 3E + 08n^{(-1.015)}$	ISend	4031	64651
2) $f(n) = 2 * \sqrt{n} - 1$	$f(n) = 3E + 08n^{(-1.015)}$	Irecv	126	64651
3) $f(n) = 0$	$f(n) = 0$	Wait	0	0
4) $f(n) = 0$	$f(n) = 0$	Wait	0	0
Phase 6				
1) $f(n) = 1$	$f(n) = 7E + 07n^{(-0.482)}$	ISend	1	1270400
2) $f(n) = \sqrt{n} - 1$	$f(n) = 7E + 07n^{(-0.482)}$	ISend	63	1270400
3) $f(n) = \sqrt{n} + 1$	$f(n) = 7E + 07n^{(-0.482)}$	ISend	64	1270400
4) $f(n) = n - \sqrt{n}$	$f(n) = 7E + 07n^{(-0.482)}$	ISend	4032	1270400
5) $f(n) = 2 * \sqrt{n} - 1$	$f(n) = 7E + 07n^{(-0.482)}$	ISend	127	1270400
6) $f(n) = n - \sqrt{n} + 1$	$f(n) = 7E + 07n^{(-0.482)}$	ISend	4033	1270400
7) $f(n) = 1$	$f(n) = 7E + 07n^{(-0.482)}$	Irecv	1	1270400
8) $f(n) = \sqrt{n} - 1$	$f(n) = 7E + 07n^{(-0.482)}$	Irecv	63	1270400
9) $f(n) = \sqrt{n} + 1$	$f(n) = 7E + 07n^{(-0.482)}$	Irecv	64	1270400
10) $f(n) = n - \sqrt{n}$	$f(n) = 7E + 07n^{(-0.482)}$	Irecv	4032	1270400
11) $f(n) = 2 * \sqrt{n} - 1$	$f(n) = 7E + 07n^{(-0.482)}$	Irecv	127	1270400
12) f$(n) = n - \sqrt{n} + 1$	$f(n) = 7E + 07n^{(-0.482)}$	Irecv	4033	1270400
13-24) $f(n) = 0$	$f(n) = 0$	Wait	0	1270400

In strong scalability the application workload (Application Input Data) remains constant as the application scales. The workload is distributed among all the processes, and the instructions executed by each process decrease as the number of processes increases. The Application Instructions number (AppInstr) remains practically constant. We use Eq. 1 to obtain the total number of instructions, where $InstrP_i$ is the Instructions per process and NP is the Number of Processes. We can extrapolate this concept into the application phases, maintaining its total number of instructions constant as the application scales, as it is shown in Fig. 8.

$$AppInstr = (\sum_{i=1}^{x} InstrP_i) * NP \tag{1}$$

To predict computational time with high accuracy we use a method based on a change of workload as shown in Fig. 9. The main idea is to find with Workload A similar

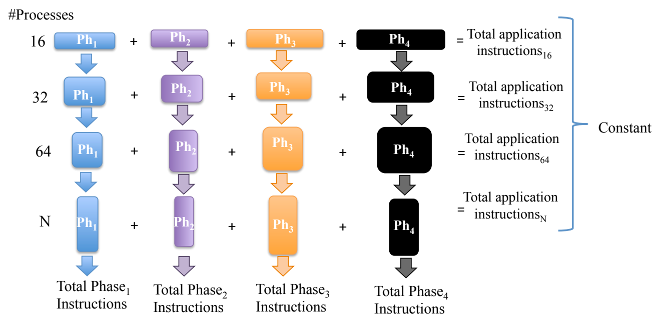

#Processes
16
32
64
N
Ph1
Ph2
Ph3
Ph4
Total application instructions16
Total application instructions32
Total application instructions64
Total application instructionsN
Constant
Total Phase1 Instructions
Total Phase2 Instructions
Total Phase3 Instructions
Total Phase4 Instructions

Fig. 8. Behavior of the strong scalability per phase (Ph_i).

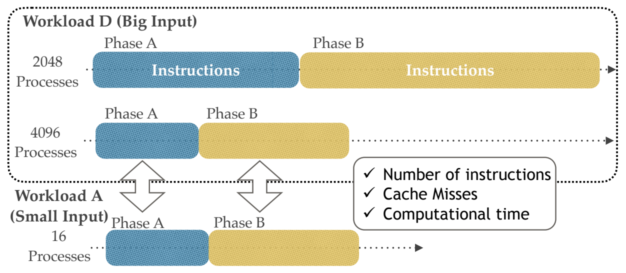


Fig. 9. Workload A allow us to measure the computational time using small-scale of processes.

number of instructions (small-scale of processes) that workload D would have with great number of processes. If we find similar number of instructions, we execute Workload A with small-scale processes to measure the computational time.

This method allows us to measure the computational time for the phase for a large number of processes, executing small-scale signatures with small workloads. In this way, we introduce the measurable points into the model and find the regression function that fit between the small-scales signatures and the measurable points.

To predict the number of instructions of each process by phase we model as the instructions are distributed in the phase when the number of processes increases. We start modeling the processes with a similar computational behavior, that is a similar number of instructions (95% similarity), which are grouped in Instructions Groups (IG_i). The total number of instructions of each Instruction Group ($TotalInstG_i$) remains constant. Then, each Instruction Group is modeled as the instructions are distributed as the number of processes increases. The sum of instructions of each group multiplied by the weight of the phase will be the Total Number of Instructions Predicted ($PTInstPhase$), as is shown in Eq. 2, where j is the total number of groups.

$$PTInstPhase_i = (\sum_{j=1}^{n} TotalInstG_j * weight) \tag{2}$$

Fig. 10 shows an example of a phase with 4 processes with a different number of instructions. Processes 0 and 1 have a similar number of instructions, and processes 2 and 3 another. Scaling the application for 8 processes, processes 0 and 1 distribute their instructions between processes 0 to 3, while processes 2 and 3 distribute their instructions between processes 4 to 7, following their computation rules. Then, for this example, we have 2 different groups, IG_1 and IG_2, where each group distributes its number of instructions in

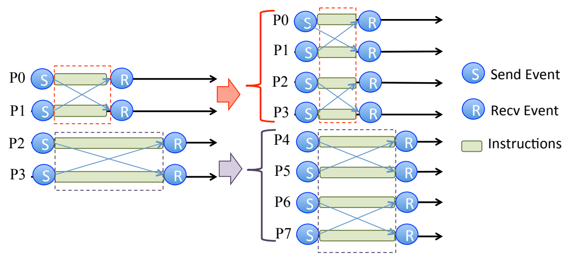


Fig. 10. Distribution of instructions as the number of processes increases.

TABLE 2
Information obtained of process 0 by the execution of BT small-scale signatures.

Phase ID	PhaseInst	NP	Weight	WD	PTInstPhase
-	-	4	-	2	-
-	-	9	-	3	-
-	-	16	-	4	-
Small-Scale Signature Execution for 256 processes					
		256		16	
0	506722396		15015		1947759814640640
1	31000057		15015		119159259098880
2	503203238		15013		1933975094296060
3	29545085		15015		113566579526400
4	506946549		15015		1948621422908160
5	30349477		15015		116658533671680
6	2990984267		999		764926280379648
Total Number of Instructions Predicted					6944666984521470
Small-Scale Signature Execution for 324 processes					
		324		18	
0	354379986		17016		1953762068735420
1	21332142		17016		117608023960128
2	357616263		17017		1971720126980600
3	21786096		17017		120117814584768
4	357583752		17017		1971540877322020
5	20764767		17017		114486708972636
6	2350571435		999		760823559795060
Total Number of Instructions Predicted					7010059180350640
Small-Scale Signature Execution for 484 processes					
		484		22	
0	195756794		21021		1991661726270220
1	12020277		21021		122296269523428
2	194144876		21019		1975073875943700
3	11367995		21021		115659845481180
4	195782158		21021		1991919783765910
5	11587976		21021		117897968252064
6	1548555330		999		748751278940280
Total Number of Instructions Predicted					7063260748176780
PhaseInst=Number of Instructions per phase NP= Number of Processes WD= Weight Displacement					

a specific way, following a behavior rule of distribution. As shown in Eq. 3, we calculate the total number of instructions of the set of processes of a group *(n)*, the Total Instructions of the Group ($TotalInstG$) is the sum of the instructions of each process ($PhaseProcessInstr_i$) involved in the group for each phase.

$$TotalInstG_j = \sum_{i=0}^{n} (PhaseProcessInstr_i) \tag{3}$$

As we did with the communication pattern section, in order to follow the methodology, we are going to use the BT small-scale signatures to predict the number of instructions of each phase for 1024, 2025 and 4096 processes. Then, we will assign the computational time executing small-scale signatures with small workloads (measurable points) that have a similar number of instructions.

In Table 2 we show the information obtained by the execution of 3 small-scale signatures (256, 324 and 484 processes) using class E with 1000 iterations as input. For BT application we have one Instruction Group $IG = 1$. That is, all processes have the same number of instructions and Table 2 focuses on the information in process 0.

As we said before, the signatures give us the number of instructions per phase. We apply Eq. 2 in order to obtain the Predicted Total number of instruction per phase (PTIn-

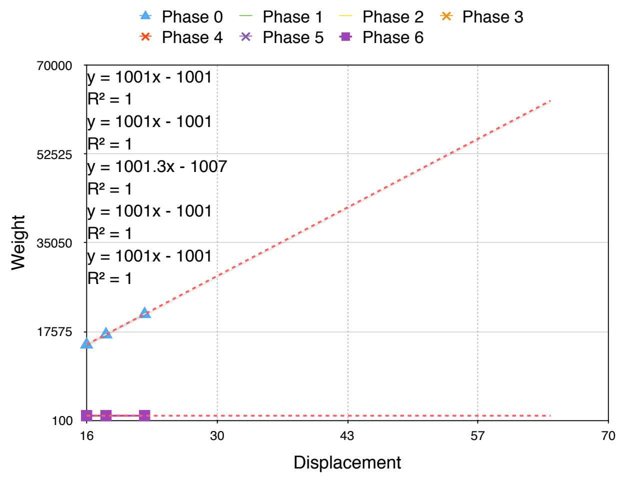


Fig. 11. Modeling the weight of the BT phases.

stPhase). The sum of all PTInstPhase will give us the Total Number of Instructions Predicted, which will be practically constant when we execute the small-scale signatures using a different number of processes.

The main idea in this section is to create a similar Table to 2 for N number of processes that will be defined as Instruction Prediction Table (IPT). The IPT will give us information about the weight and number of instructions per phase and process for a specific number of processes.

$$PhaseProcessInstr_i = \frac{\frac{TotalInstG_i}{n}}{weight} \quad (4)$$

The goal is to predict the term $"PhaseProcessInstr_i"$ for a greater number of processes. Given that the sum of $TotalInstG * weight$ is practically constant, the weight of each phase was predicted by the regression methods describe in section 3.3.1, and the Number of Processes (n) between the instructions are distributed in the group is known, we can predict the instructions of each process, isolating the term $"PhaseProcessInstr_i"$, as is shown in Eq 4.

3.3.1 Weight Model

Regression models are used to model the weight behavior. Due to the deterministic way of the weight behavior as the application scales, there is a dependence between the number of processes and the weight of the phase. For this reason, linear regression is most appropriate to fit the weight, as it allows the creation of a prediction equation such as $y = a + bx_0$ by using the number of processes to execute the application as an independent variable representing the weight behavior of each phase, obtaining a $R\text{-}square = 1$.

Scientific applications cannot be executed for any number of processes, but they also follow data distrubution rules. Depending on the number of processes required to execute the application, it can be possible that the linear regression does not fit properly obtaining a correlation index $R\text{-}square$ distant to 1. In this case another kind of regression could be more appropriate. This happens by the distance between the input samples (Number of processes to execute the application) used to fit the regression.

TABLE 3
Weight Prediction Table (WPT) for BT Application

Phase ID	Weights obtained with the Small-Scaled Signatures NP (Displacement)			Weights Predicted for N Processes NP (Displacement)		
	256 (16)	324 (18)	484 (22)	1024 (32)	2025 (45)	4096 (64)
0	15015	17017	21021	31031	44044	63063
1	15015	17017	21021	31031	44044	63063
2	15013	17017	21021	31031	44044	63063
3	15015	17017	21021	31031	44044	63063
4	15015	17017	21021	31031	44044	63063
5	15015	17017	21021	31031	44044	63063
6	999	999	999	999	999	999
NP= Number of processes						

In Fig. 11 we show the weights of each phase obtained with the small-scaled signatures (256, 324 and 484 processes). Note that due the limitations of the application, users can only execute the application using a square number of processes.

In Table 3 and Fig. 11 the distance of the input points used (256, 324 and 484 processes) to model the regression is non-uniform, so if we fit the points by a linear regression we obtain a $R\text{-}square = 0.98253$. We know that if we use an equation with this correlation index, the prediction error will be relevant and it will be higher as we move away from the executed points. In order to use a linear regression with an $R\text{-}square = 1$, we make a linearization process based on a change of domain (Weight Displacement (WD)), where the objective is to obtain a uniform distance among all the points. In same Figure 11 we changed the number of processes by a sequential index (WD), making distance equal 1 for all the points. In this way we obtain a $R\text{-}square = 1$ using a linear regression. Table 3 shows the Predicted Weights obtained using the regression Equation for each phase.

3.3.2 Computational Time Model

The objective of this section is to assign a computational time to each phase. Predicting the computational time by phase, instead of the whole application, considerably improves the prediction error. This happens because each phase has a different computation behavior which has to be approximated by a specific regression function. Despite this improvement, regression models are limited by the scope of the prediction.

In previous work [24] we based the prediction on regression functions using the small-scale signatures as the first points of the regression and a distant point to fit the regression function. This method only allows to predict some regions of the curve. This is because when the application increases the number of processes the ratio of cache misses is not constant.

Another problem is to know which regression function has the best approximation to the computation behavior using small-scale signatures. In Fig. 12a we show the first 3 points executed with BT signatures using Class E with 1000 iterations as input. We selected the regression function (potens or exponential regressions) to predict the behavior. On the other hand, if we predicted a distant point from the real points used to generate the model, we would obtain an initial prediction error as we move away from the first points.

To avoid this problem and therefore to improve the quality of prediction for a large number of processes, as we show in Fig. 12b, we propose a method which consists of measuring far points (measurable points) without using a large number of processes and system resources. Using this method we can assign the computational time for each phase, afterward we select the regression function that fit better between the points.

The proposed method to measure these points are based on doing a change of workload domain (application input) using a workload much smaller than the original, to emulate the application computational time of each process for the original workload with a large number of processes.

A phase is a reduced segment of code which executes a specific function. We can select a new workload for the phase, smaller than the original, which will be executed over a small number of processes. The objective is to achieve a similar number of instructions that will result in similar cache misses per process, rather than the original workload executed over a large number of processes, to emulate the computational time by process and phase.

We generate a new table, named Instruction Prediction Table (IPT), which contains a computational global vision of each phase. The IPT contains the information about the number of instructions by process, the number of processes, the weight of the phase and its displacement, and finally the total number of instructions as the application scales.

In order to obtain the number of instructions per phase and process ($PhaseProcessInstr$), we start filling the column Weight of Table 4 using the information obtained in Table 3. To obtain the $PhaseProcessInstr_0$ for phase 0, we use Equation 4 where we take for BT application the $TotalInstG$ as 1.9916E+15, the n being the number of processes that we want to predict (1024, 2025, 4096) and the weight predicted. Finally we obtain the $PhaseProcessInstr_i$ that is showed in the second column of the same Table 4 .

To obtain more than one Instruction Group in the computation pattern modeling, we generate many Instruction Prediction Tables as Instruction Groups. The total number of instructions of each Instruction Prediction Table will be the total number of instructions of the phase. Going forward from the IPT table, we check if the total number of instructions is practically constant as the number of processes increases. If this assumption is not met, the method is not applicable and we cannot construct the IPT table and we can not calculate the computational time of the phases.

Workload E is distributed among the application processes in a uniform way, with each process receiving a work E'. If we executed the signature for 64 processes with a small workload X, the processes would be carrying out the same work (same number of instructions and cache misses) as when executing the application for 4096 processes with a workload equal to E.

In Table 5 we show the predicted number of instructions of phase 6 with CLASS E extracted from the IPT Table for 1024, 2025, 4096 and 4900 processes. We start generating a signature with 25 processes adjusting the problem size (workload) for CLASS X, to generate similar number of instructions to CLASS E with 1024 processes with the goal of measuring its computational time. Once we found the relation for 1024 processes, we increase the number of processes (49, 64 and 81) measuring the computational time for each point.

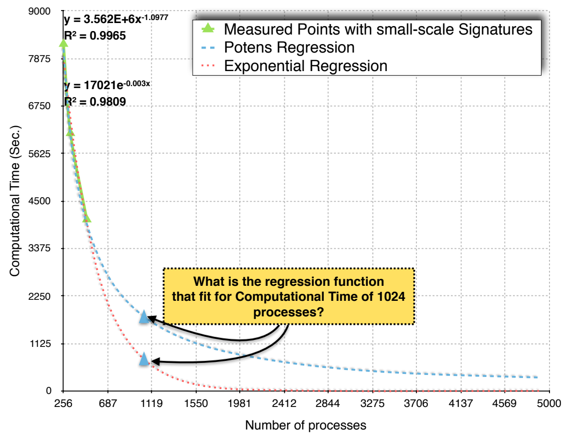
TABLE 4
Instruction Prediction Table (IPT) for BT application.

Phase ID	PPhaseInst	NP	Weight	WD	PTInstPhase
Predicted Phase Instructions (PPhaseInst) for 1024 processes					
		1024		32	
0	62678681		31031		1991661726270220
1	3848730		31031		122296269523428
2	62156652		31031		1975073875943700
3	3639878		31031		115659845481180
4	62686802		31031		1991919783765910
5	3710313		31031		117897968252064
6	731934355		999		748751278940280
Total Number of Instructions Predicted					7063260748176780
Predicted Phase Instructions (PPhaseInst) for 2025 processes					
		2025		45	
0	22330775		44044		1991661726270220
1	1371202		44044		122296269523428
2	22144790		44044		1975073875943700
3	1296794		44044		115659845481180
4	22333668		44044		1991919783765910
5	1321888		44044		117897968252064
6	370123842		999		748751278940280
Total Number of Instructions Predicted					7063260748176780
Predicted Phase Instructions (PPhaseInst) for 4096 processes					
		4096		64	
0	7710473		63063		1991661726270220
1	473455		63063		122296269523428
2	7646255		63063		1975073875943700
3	447763		63063		115659845481180
4	7711472		63063		1991919783765910
5	456427		63063		117897968252064
6	182983589		999		748751278940280
Total Number of Instructions Predicted					7063260748176780

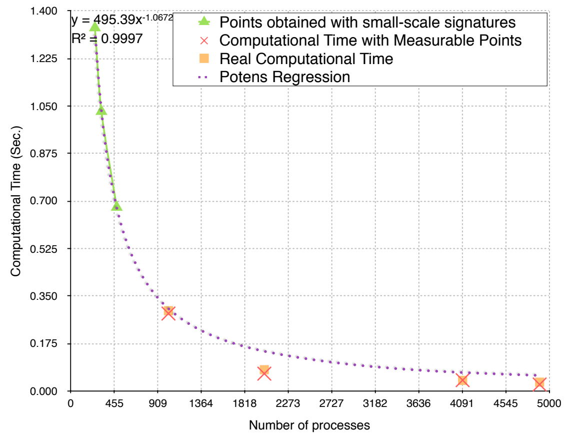
PPhaseInst=Predicted Instructions ($PhaseProcessInstr$)
NP= Number of Processes
PTInstPhase=Predicted Total Number of Instructions per Phase (PPhaseInst*Weight)*(NP)
WD= Weight Displacement

The time required (AET and SET) to obtain the computational time for all application phases is shown in Table 5. For workload CLASS X, we execute the application to obtain the signature for each point. Finally, the last column of same table shows the number of resources used.

We show the computational information of phase 6 (BT application) obtained for the measurable points in Table 6. We divide the Table in two sections. In the first section we show information obtained by the small-scale signatures and the second part shows the predicted information using the measurable points. All information its related to the computational behavior of the phase, such as the number of instructions and computational times, the L1 misses and cycles counter, the cycles by number of instructions executed and the ratio between them. Importantly, there is a significant change going from 1024-2025 processes, especially in the column Cycle/Number of Instructions, where the number of instructions executed per cycle changes from 1 instruction per 1 cycle (1024 processes) to 2 instruction per cycle (2025 processes). This behavior happens because the problem size (workload) is decreasing and L2 misses do not occur any longer. For the BT application, that's the reason why we can't predict the region of 1024-2025 processes using only the regression models.



(a) Regression models used to predict the computational time for BT application



(b) Prediction of the computational time using measurable points

Fig. 12. Prediction of computational time for phase 6 (BT application)

TABLE 5
Measures of computational times, (Phase 6 BT application), using measurable points.

Prediction for N Processes	Phase ID	Predicted Number of Instructions with CLASS E	Number of Instructions with CLASS X	DIF (%)	Computational Time with CLASS X (Sec.)	Time to obtain the computational time with CLASS X AET (Sec.)	SET (Sec.)	Resource used to obtain the computational time (cores)
1024	6	731,934,355	731,834,575	0.01	0.285265	163.13	35.92	25
2025	6	370,123,842	343,571,970	7.17	0.063803	75.87	33.48	49
4096	6	182,983,589	183,364,223	0.21	0.039758	62.70	31.68	64
4900	6	152,959,343	154,713,060	1.15	0.023913	41.21	30.75	81

DIF: Difference between number of instructions of CLASS E and CLASS X.
AET: Application Execution Time
SET: Signature Execution Time

TABLE 6
Computational data obtained by the execution of measurable points.

NP	Cycles	L2 Cache Misses	Comp. Time (Sec.)	#Instr	Cycles/ #Instr	Ratio Miss/ #Instr
Information obtained by small-scale signatures						
256	3.03E+09	8.99E+07	1.3381	2.99E+09	1.0142	0.030
324	2.34E+09	7.01E+07	1.0300	2.35E+09	0.9937	0.029
484	1.53E+09	4.68E+07	0.6770	1.55E+09	0.9904	0.030
Information Obtained by Measurable Points						
1024	6.73E+08	2.20E+07	0.2852	7.32E+08	0.9194	0.030
2025	1.81E+08	1.15E+07	0.0638	3.44E+08	0.5272	0.033
4096	8.84E+07	6.25E+06	0.0397	1.83E+08	0.4823	0.034
4900	7.37E+07	5.24E+06	0.0239	1.55E+08	0.4763	0.033

NP: Number of processes
Comp. Time: Computational Time
#Instr: Number of Instructions
Ratio Miss/#Instr: Ratio L2 Misses / Number of instructions

Finally, after having the number of processes for the original workload and the computational time measured, we incorporate the computational time measured with the measurable points with CLASS X to picture how the computational times of phase 6 scale as shown in Fig. 12b.

Once we have predicted the computational times, we create the ST4NP, which will be used to obtain the communication times to predict the application performance using a limit set of system resources.

4 P3S: Synthetic Signature for Scalability Prediction

In this section we propose a method to predict the performance of MPI parallel applications with more processes than the number of resources used to predict the application scalability. The method is focused on generating the Synthetic Signature (SS), which will use the ST4NP as input. Once processed, this will allow to predict the application performance using the number of processes that the ST4NP was created for.

To achieve our goal we focus on how to execute the ST4NP generated by the P3S tool in a limited number of system resources. We designed the SS program, which allows to execute the information of the ST4NP by loading ranges of processes. As shown in Fig. 13, the SS program loads range A that corresponds to the Scaled Traces of process 0 to 63 and range B that corresponds to the processes 64 to 127 (first iteration).

The goal of the SS program is to execute the communication events at an appropriate frequency to measure the communication time of the phases to predict the application execution time. To do this, from the information of the ST4NP, the SS generates the same type of communications (MPI_Datatype), the source and destination of the message and the communication volume which would be generated by the application in its execution. In addition, to emulate the computational time given by the Scaled Trace, the SS calls to *nanosleep()*, a system call, per each event executed.

Due to the limited resources, we define a system configuration to execute the SS program. As shown in Fig. 13, SS uses Nodes 1 and 2 in the system which we want to predict to measure the communications, and Nodes 3 and 4 to solve the communications that depend on the ranges of ST4NP. Once the communications are measured, SS changes the

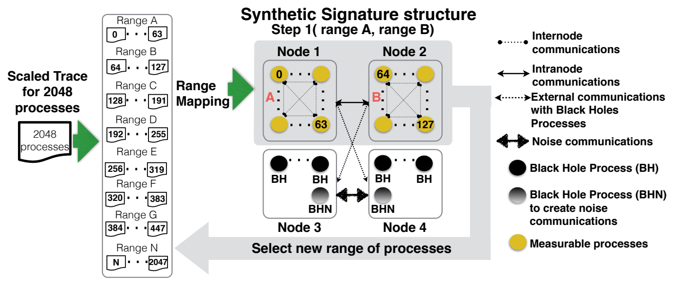


Fig. 13. Synthetic Signature structure.

ranges of the processes. In order to explain how SS works we will use the following definitions:

Range of Scaled Traces processes (RangesST): Range of processes that will measure their communication events and will be loaded in Nodes 1 and 2.

Measurable Process (MeasurableP): A process that loads a Scaled Trace of a process in order to measure their communications. Some MeasurablePs will communicate with the Black Hole processes when their communications are out of the RangesST processes.

Black Hole process(BH): A process that sends or receives messages from MeasurablePs, the SS creates a number of BH processes depending on the number of MeasurableP that communicate out of the RangesST processes.

Black Hole Noise processes (BHN): Pair of processes that generate the same number of messages in transit on the interconnection network per phase.

The following definitions correspond to the MeasurableP:

Internode communications: Communications between two processes of the same system node (Node 1 or Node 2). These messages will be measured using timers of the operative system. Fig. 14 shows an example consisting of Process 0 sending a message to process 63.

Intranode communications: Communications between two processes using the interconnection network from Node 1 to Node 2 or vice versa. Fig. 14 shows an example consisting of Process 0 sending a message to process 64.

In Fig. 14 we show the next type of messages that correspond to the BH processes, emphasizing that the following communications will not be measured:

Control Message: There are two types of Control Messages. The first type, as seen in point A in Fig. 14, is when a message is received by the BH process from MeasurableP, which means that the MeasurableP is requesting a message with some specific characteristics. This happens when a

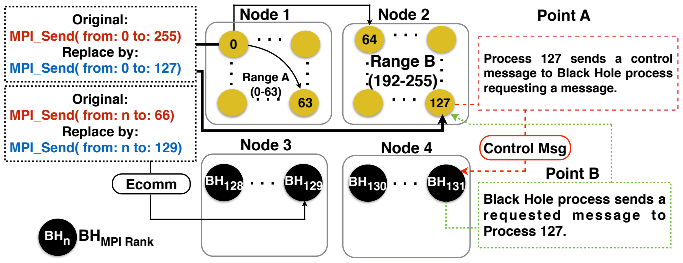

Original:
MPI_Send(from: 0 to: 255)
Replace by:
MPI_Send(from: 0 to: 127)
Original:
MPI_Send(from: n to: 66)
Replace by:
MPI_Send(from: n to: 129)
Node 1
Node 2
Point A
0
63
Range A (0-63)
64
127
Range B (192-255)
Process 127 sends a control message to Black Hole process requesting a message.
Control Msg
Node 3
Node 4
Point B
Ecomm
BH128
BH129
BH130
BH131
Black Hole process sends a requested message to Process 127.
BHn BHMPI Rank

Fig. 14. Communications with the BH processes of the Synthetic Signature.

MeasurableP needs to receive a message from a process which is not within the established RangesST processes. For this reason, the BH process generates a message with specific buffer size, MPI datatype, and MPI_TAG for the MeasurableP that requests it, as shown in point B of Fig. 14. The second type of Control Message occurs when any BH process sends a message to the BHN in the same Node in order to enable the noise communications when a phase starts.

External Communications (EComms): The communication is executed between the MeasurableP and the BH process, in order to solve the communication dependencies of the phase.

Noise Communications: Messages between two BHN processes. The SS generates the same number of communications in transit, as does the application.

Next subsections describe the procedure of SS to predict the application execution time.

4.1 Setup the RangesST and Black Holes Processes.

Once we have defined the structure of the SS we have to execute SS to predict the application performance. We need to apply the same mapping policies that the application will use to execute. To do this, these mapping policies have to be provided to the RangesST processes to execute in each iteration and SS has to make n as many iterations as the number of RangesST processes.

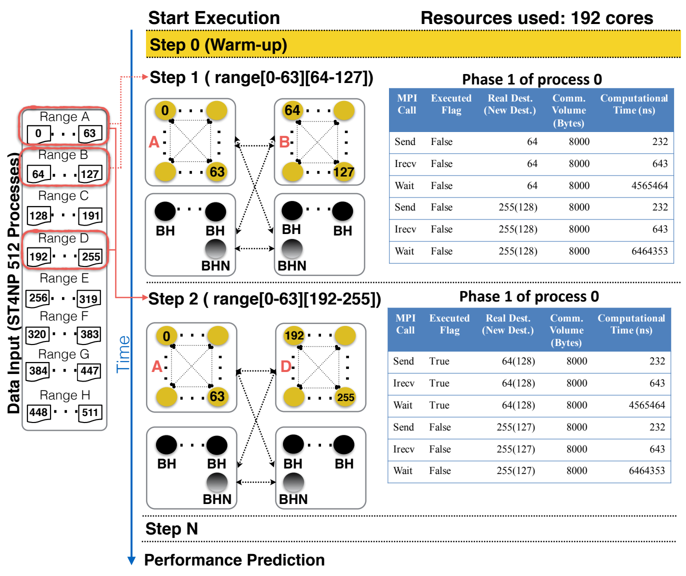
For every iteration of SS, we need to load different RangesST processes in the MeasurablePs. As an example, for a total of 256 processes, if we will to execute the SS in a system with 64 core nodes, the SS will use two nodes of 64 cores for the MeasurablePs and two more nodes for the BHs processes. For each iteration SS will make 4 RangesST processes, whose range values are (startProcess - EndProcess): 0-63, 64-127, 128-191 and 192-256.

$$NumBHProcesses = \sum_{i=1}^{m} (MeasurablePwithEComms_i) \tag{5}$$

To update the RangesST processes after measuring, the MeasurablePs will load different Scaled Traces of processes and SS will modify the message destination which will depend on the loaded RangesST processes. For example, if we launch SS using 2 nodes of 64 cores, the MPI world contains 128 processes (from 0 to 127 process). Now, if we select the RangesST processes from 0 to 63 it will be executed in Node 1 and the other RangesST processes from 192 to 256 will be executed in Node 2. In order to communicate the processes from both RangesST, the messages' destinations will be labeled with processes belonging to the MPI world using the MPI_Comm_Rank of the SS, as shown in Fig. 14.

Once we have selected the range of processes to execute in the current iteration, the Scaled Traces of each process selected have to be analyzed in order to know the total number of the EComms (out of the RangesST processes) and the total number of BH processes.

To select the number of BH processes to be created we use Eq. 5. We create one BH process for each MeasurableP that has an EComm. The destination of these messages will be changed to the rank of the BH process, as shown in Fig.



Phase 1 of process 0

MPI Call	Executed Flag	Real Dest. (New Dest.)	Comm. Volume (Bytes)	Computational Time (ns)
Send	False	64	8000	232
Irecv	False	64	8000	643
Wait	False	64	8000	4565464
Send	False	255(128)	8000	232
Irecv	False	255(128)	8000	643
Wait	False	255(128)	8000	6464353

Phase 1 of process 0

MPI Call	Executed Flag	Real Dest. (New Dest.)	Comm. Volume (Bytes)	Computational Time (ns)
Send	True	64(128)	8000	232
Irecv	True	64(128)	8000	643
Wait	True	64(128)	8000	4565464
Send	False	255(127)	8000	232
Irecv	False	255(127)	8000	643
Wait	False	255(127)	8000	6464353

Fig. 15. Execution of the Synthetic Signature

14. In the worst case we would have to create the same number of MeasurableP for each BH process.

4.2 Execution of the SS program.

In this section we describe how SS is executed with the selected resources in the previous section to obtain the execution time of each phase in a bounded time. After this, we used the PAS2P Equation 2 to predict the application performance for the number of processes for which the Scale Trace was generated.

We illustrate the procedure followed in order to execute SS with the example of Fig. 15. We used a system with 64 core nodes and a Scaled Trace generated for 512 processes, thus, we have 8 RangesST processes. We have executed SS with 192 processes using 4 nodes of 64 cores (limited resources), 128 MeasurableP and 64 BH processes (32 BH processes in Node 3 and Node 4). As said before, the number of the BH processes has been selected in function of the Ecomms of the RangesST processes.

It is worth noting that before we start the measures, SS executes one first iteration (step 0) to warm-up all the machine components. In this example, we focus on phase 1 of process 0. As we can see in Figure 15, for the first iteration (step 1), we have selected the RangesSTs processes 0-63 for Node 1, and 64-127 for Node 2. We have selected the same mapping processes, in function of the application mapping used, to execute the application with all the resources (512 processes) and seeking to maintain the physical distances of the interconnection network. To measure the communication events, we start when the communication event occurs in one MeasurableP, and we stop when the same communication event ends.

The step 1, as seen in the table contained in Fig. 15, all events start with the flag ”Executed” as false. The same Fig. shows that the events with the ID 3, 4, and 5 exchange information with process 255 (Ecomms). Since process 255 is not involved in the RangesST processes of the current iteration, we changed the destination of these messages to the BH process, which in this case (BH process 128) is allocated in Node 3. The objective is to execute Ecomms but not measure them, to create the same number of messages in transit. If we need more messages, BHNs create the needed messages to reflect the same number of messages on the application.

The communication events with the ID 0, 1 and 2 exchange messages with process 64 (intracommunications). Therefore, we execute these communications to measure their execution time and it is saved in the event structure, and we change the flag ”Executed” to true, as shown in the Executed column of the table contained in Fig. 15 (see step 2). Additionally, whenever an event is executed, we emulate the computational time of the application (see Compute Time column in table contained in Fig. 15) using the nanosleep() system call of the Operaty System. As shown in Fig. 16, the SS measures the Phase Execution Time per process, adding the communication times and the computational time that is emulated as we say before.

When all the communications events have been executed, SS checks if all the communication events have been measured. Since there are 3 events which have not been measured in the iteration, we need to carry out a new iteration. As we can see in Fig. 15, for this iteration (step 2), we selected new RangesST processes (192 to 255) for Node 2. We have to identify the external communications and assign them a BH process. In this second iteration, since the first 3 communications have already been measured in the previous iteration (step 1), they are executed but not measured. In this iteration, these communications are Ecomms, thus, they exchange information with the BH process.

Once all the communications events have been measured, in order to obtain the predicted execution time, we follow the same procedure until all the communication events have been measured. Then, we select the physical process of SS with the highest execution time in order to apply the PAS2P equation 6, to obtain the Predicted Execution Time, where PET is the Predicted application Execution Time, m is the number of phases, $PhaseETi$ is the Phase i Execution Time, and Wi is the weight of phase i.

$$PET = \sum_{i=1}^{m} (PhaseET_i)(W_i) \tag{6}$$

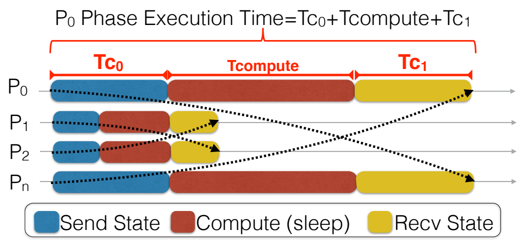


Fig. 16. Measure the Phase Execution Time per process

5 Experimental Results

In this section we present the experimental validation of the P3S methodology. To evaluate the prediction quality, we selected a wide range of scientific message-passing applications with different communicational and computational patterns. The selected applications can be executed with a high number of processes. Keeping in mind all these considerations we have selected the following benchmarks and applications: BT, CG, SP and LU from the NPB NAS [6] benchmark suite, Sweep3D [7] and N-Body.

Table 7 shows in detail the workload used in each application, the set of small-scale signatures used to carry out the prediction, and the predicted points. As we show in this table, for all the applications we have used 3 small-scale signatures to predict the far points. The number of processes of the small-scale signatures vary from 64 to 512, depending on how the application is executed. Concerning to the predicted points from this set of small-scale signatures, we predicted the points from 529 to 4096 processes.

As an experimental environment we used the BEM cluster, whose architecture is shown in Table 8. This cluster has 205 nodes with 24 cores per node, making a total of 4920 cores. Moreover, it has 60GB of RAM by node and it uses an Infiniband network.

The experimental validation was divided in two stages. During the first stage, from a small-scale signatures a series of points are predicted for a greater number of processes using SS. Thus, we can evaluate the prediction quality as we move away from the small-scale signatures executed. In the second validation stage we use both the predicted points using SS and the small-scale points obtained when we executed the signatures, to predict the application scalability.

5.1 Performance prediction evaluation

Table 9 shows the results obtained for the different applications. Regarding the execution of the small-scale signatures, we have used a 1:1 mapping (one process per core) using a maximum of 512 cores for the NBody Application (NBody-512 signature). The average predicted error (PETE) is about 1.45%, with the maximum PETE being about 4.2% for the signature SP-484. The average of the signature execution time (SET) is about 4% of the whole application execution time (AET), the maximum SET being about 8,26% of the AET for the Sweep-256 signature. Considering the benefits of the signature, it is used instead of the application as it has two major benefits in this work: it allows to predict the application execution time in a bounded time, this predicted time will be useful to generate the predicted application scalability. In addition, it provides accurate information about each application phase, to model its behavior as the number of processes increases and it generates the SS's for different numbers of processes.

Related to the execution time of the SS (SynET), coming back to Table 9, the average SynET is about 5.8% of the AET, with the minimum SynET being about 1% for the NBody SS's, and the maximum SynET about 13,8% for the CG-4096 SS. As the number of processes increases the SynET decreases for all the tested applications. This is due to the fact that although the number of communications of each phase increases as the number of processes increases, the computational time of each phase decreases. The average PETE is about 3% showing a maximum PETE for the SS

TABLE 7
Applications used for experimental evaluation

Program	Workload	Processes of small-scale signatures	Predict scalability points
CG	Class E 500 iterations	128, 256, 512	1024, 2048, 4096
BT	Class E 1000 iterations	256, 324, 484	1024, 2025, 4096, 4900
SP	Class E 1000 iterations	256, 324, 484	1024, 2025, 4096, 4900
Sweep3D	2000 13 iterations	64, 121, 256	529, 1024, 2048, 4096
Nbody	6,000,000 atoms	128, 256, 512	1024, 2048, 4096

TABLE 8
Clusters characteristics.

Cluster	Characteristics
Processor:	2x12 Intel Haswell 2.30MHz.
Memory:	L2 256KB, L3 15MB,RAM 60 GB.
Interconnection Network:	Infiniband QDR

TABLE 9
Performance prediction results using the SS.

Procs	AET (Sec.)	SET (Sec.)	SynET (Sec.)	PET (Sec.)	PETE (%)	Cores Used
SP Prediction using the small-scale signatures						
256	6035.67	271.34		6016.84	0.31	256
324	3965.57	61.33		4019.01	1.35	324
484	2896.99	58.23		2775.58	4.19	484
SP Prediction using the Synthetic Signatures						
1024	1176.86		55.58	1126.15	4.31	96
2025	525.08		25.70	523.29	0.34	96
4096	359.37		8.33	350.84	2.37	96
4900	265.27		7.80	278.72	5.07	96
CG Prediction using the small-scale signatures						
128	9131.76	393.23		9166.93	0.39	128
256	4853.63	177.43		4847.64	0.12	256
512	1860.61	60.05		1846.33	0.77	512
CG Prediction using the Synthetic Signatures						
1024	1263.85		37.72	1275.32	0.91	96
2048	677.93		28.08	671.96	0.88	96
4096	621.07		8.15	623.39	0.37	96
NBody Prediction using the small-scale signatures						
128	26131.87	325.77		25531.06	2.30	128
256	11427.99	165.58		11326.29	0.89	256
512	6535.28	73.45		6502.68	0.50	512
NBody Prediction using the Synthetic Signatures						
1024	3269.23		15.16	3259.98	0.28	96
2048	1642.87		13.53	1632.14	0.65	96
4096	830.50		10.64	818.06	1.50	96
Sweep3D Prediction using the small-scale signatures						
64	4514.79	261.35		4486.52	0.63	64
121	2398.64	140.28		2481.18	3.44	121
256	1147.35	94.80		1186.45	3.41	256
Sweep3D Prediction using the Synthetic Signatures						
529	569.71		28.71	606.16	6.40	96
1024	317.94		20.49	343.97	8.18	96
2025	165.91		15.21	161.46	2.68	96
4096	93.49		11.73	106.70	14.13	96
BT Prediction using the small-scale signatures						
256	8379.85	367.72		8401.45	0.26	256
324	6544.89	284.19		6525.11	0.30	324
484	4408.87	207.21		4403.46	0.12	484
BT Prediction using the Synthetic Signatures						
1024	2223.33		91.75	2146.77	3.44	96
2025	987.14		44.03	980.94	0.63	96
4096	549.33		23.33	547.00	0.42	96
4900	480.16		9.77	470.00	2.12	96
Procs: MPI Processes						
AET: Application Execution Time						
SET: Signature Execution Time						
SynET: Synthetic Execution Time						
PET: Predicted Execution Time						
PETE: Predicted Execution Time Error						

sweep-4096. Note that both the average PETE for the smalls-scale signatures and for the SS is under 5%.

5.2 Evaluation of P3S scalability prediction

This section is devoted to evaluate the final objective of P3S methodology, which is to obtain the application scalability using a bounded time and a reduced number of resources. To predict the application scalability all the predicted points for the different number of processes are used. Taking account that besides of the points obtained with the SS, we use the predicted points obtained by executing the small-scale signatures to generate the application scalability. Figures 17 and 18 show the predicted scalability for the tested applications, which are compared with the real scalability, obtained through executing the application with a mapping 1:1. Moreover, we show the linear (Theoretical) Speedup, where the application efficiency is 100%. In this way, users can select the most efficient way to execute their applications. In Fig. 18 we can observe the best way to execute the CG application with the workload used would be to use 2048 processes, obtaining efficiency of about 84%. From this point the application stops to scale, and the scalability curve become linear, decreasing the efficiency as the number of processes increases. If we focus on the next valid point to execute the application (4096 processes), the efficiency has decreased until about 45%. If we selected this number of processes to execute the application, we would be using the system in an inefficient way. Another interesting point in the figure is that when the application stops to scale, SynET takes about the same time as the number of processes increases. That is because the computational time of the phases is irrelevant compared to the communication time.

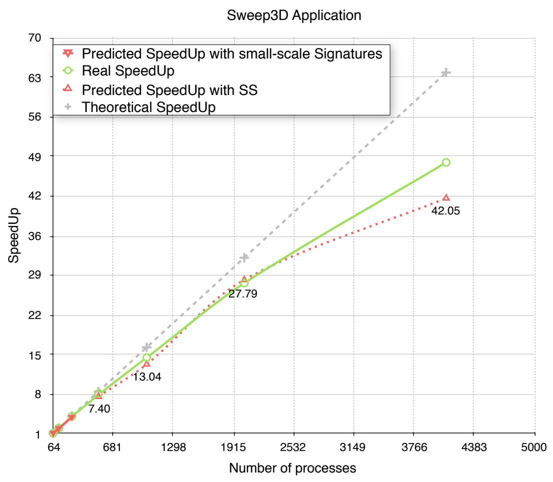
There are cases when is not easy to choose the best number of processes. Focusing now on Fig. 18 which presents the scalability of the Sweep3D application in the same system, from the point of 2025 processes, the real scalability starts far from the ideal speedup, introducing inefficiency into the system. The application efficiency moves from about 86% using 2025 processes to about 75% using 4096 processes. As the effiency in the last point is still high enough, the user should consider the trade-off between execution time and the number of resources to be used.

On the other hand, as is shown in Fig. 18, the SP application presents superscalability, which means that the application scalability is over the ideal scalability.

Another interesting case is the behavior of BT, which is shown in Fig. 17. This application has an ideal speedup from 64 processes to about 3300 processes. From 3300 processes, the real scalability starts to far from the ideal speedup, beginning to introduce inefficiency into the system. The application moves from an efficiency of 100% to an efficiency of 95% for 4090 processes and 91% for 4900 processes.

Finally, the NBODY application, whose scalability is shown in Fig. 18, presents a perfect theoretical linear speedup, which remains as the number of processes increases, and there is an application efficiency from 64 to 4096 processes of about 100%.

Concerning to the execution of the SS, we have used 4 full compute nodes to execute all the SS of the tested applications, making a total of 96 cores. As we ran the signatures to execute the SS we used a 1:1 mapping. Fig. 19 compares



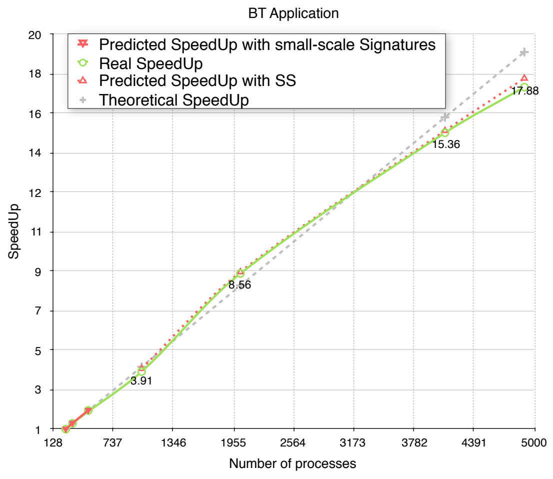


Fig. 17. Predicted Scalability.

the resources used to execute the application against the resources used to execute the small-scale signatures and the SS's.

In same Fig. 19, to execute the small-scale signatures we use the same number of cores needed to run the application. Meanwhile, to execute the SS, we use a reduced number of resources. Focusing on Fig. 19 (the application with the least SS executed), to predict the performance for 529 processes we only use 18% of the total resources needed to execute the application. Following the same figure, the saving is higher as the number of processes increases. This is because we are using 96 cores in all the cases to execute all the SS, independently to the number of the application processes. As we can see, for the SS of 4096 processes, we are using only 2% of the resources needed to execute the application.

Regarding to the overhead of applying P3S to generate the application scalability curve, the Eq. 7 is used to calculate it. As is shown in this Eq., the overhead is computed as the sumatory of the execution times of all the points used to generate the scalability curve (SET and SynET), multiplied by the number of cores (#C) used in each point.

All the data to calculate the overhead of applying P3S is provided in Table 9. The same table also provides the data to calculate the overhead to generate the scalability curve using just the application (AET). Taking as example the BT application, if we apply the equation 7, we obtain

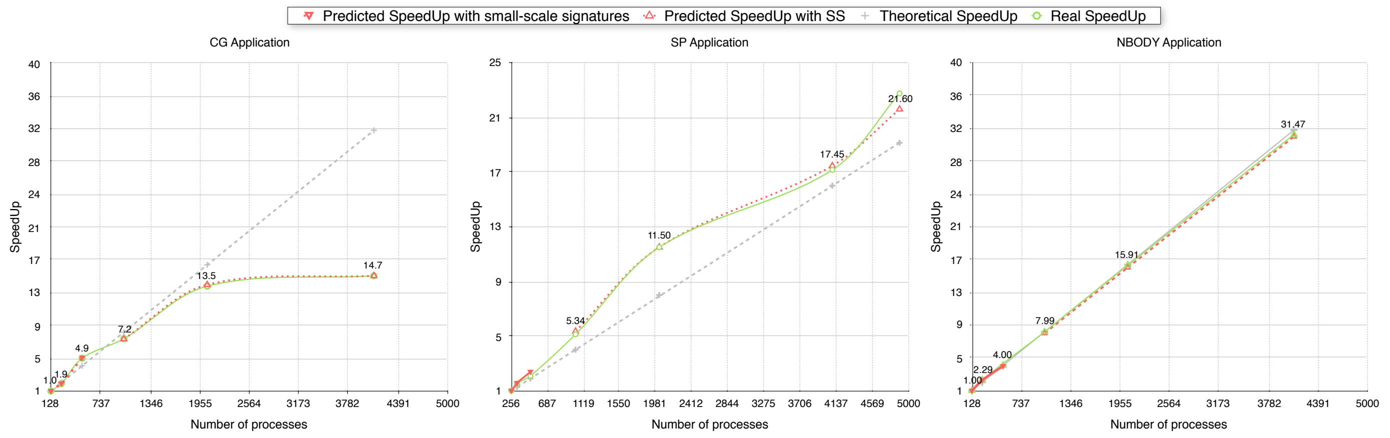


Fig. 18. Predicted Scalability.

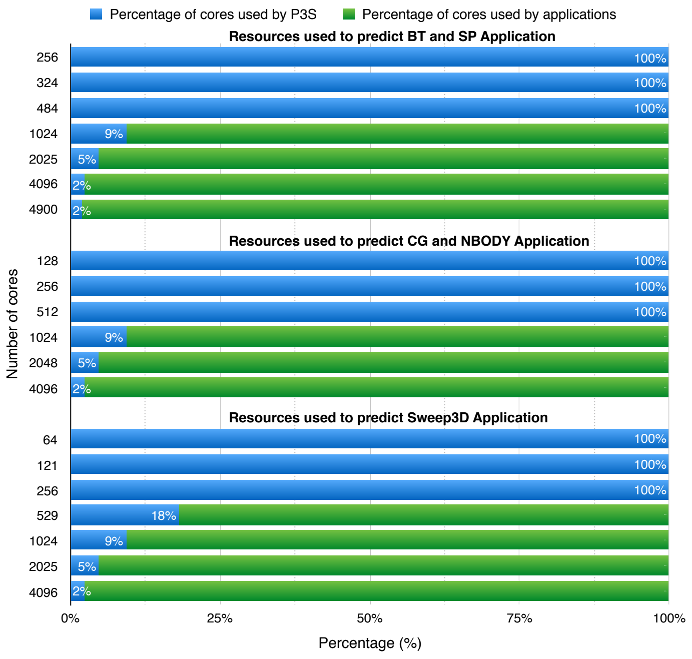


Fig. 19. Number of cores used to predict the applications scalability.

an overhead applying the P3S of 84.08 Cores Hours, against 4,243.93 Cores Hours of using the application. As average, P3S achieves to reduce the overhead of generating the scalability curve using the application above the 95%.

$$Overhead_{P3S} = \sum_{i=1}^{N}(SET_i * \#C) + \sum_{j=1}^{M}(SynET_i * \#C) \quad (7)$$

As we have shown, by using the P3S we achieve predicting the application scalability in a bounded time and using a reduced number of cores. Analyzing the predicted scalability, users can choose the best way to execute their applications, since they can detect where the application starts to have an inefficient behaviour in the system.

6 Conclusion and Future Work

In this work we have presented and validated the P3S methodology, which allows us to analyze and predict the strong scalability behavior for MPI applications on a given system. As we have shown, the methodology strives to use a bounded analysis time and a reduced set of resources to predict the application behavior for large-scale. We validated our methodology on a wide set of scientific applications, with different communication and computational patterns, obtaining on average a prediction error less than 5%.

In addition to predict the application scalability, the P3S methodology provides information to users or developers about each application phase. This information can be used to analyze their behavior to detect possible inefficient phases, which become potential bottlenecks at the application level.

The P3S methodology also enables the system administrators to plan policy scheduling to make an efficient use of the system. Using the methodology, system administrators can quickly predict the set of applications to be executed on the system, with the goal of designing the best policy planning for the queues of the system.

The P3S methodoloy has some limitations that could be considered as future work. Currently, the methodology is limited to scientific message-passing applications which have been developed using communication and computational rules of behaviour, as the application increases the number of processes. Although this is the common way to develop scientific applications and we have not found any application which does not follow this behavior, this does not imply that applications with irregular communicational and computational patterns cannot exist.

Once we have executed the signature we save this number of instructions and the computational time. We know that in some cases, due to the limitations of the parallel application, it is not feasible to generate a different workload from the original. In these cases it is not possible to obtain the distant point.

The SS is dependent on the system in which it has been generated. It could be useful for future research to generate an SS independent of the system. Thus, we would carry out cross-architectural predictions and choose a better system for executing the application.

In this work we have focused on the study of the strong scalability. Due to the weak scalability is an important and complementary study to be treated, we not discard as future work to extend the methodology to study this scalability.

ACKNOWLEDGMENTS

This research has been supported by the MINECO Spain, under contract TIN2014-53172-P. Experimental results have been carried out using resources providing by Wroclaw Centre for Networking and Supercomputing (http://wcss.pl), grant No 266.

REFERENCES

[1] J. Dongarra, S. Tomov, P. Luszczek, J. Kurzak, M. Gates, I. Yamazaki, H. Anzt, A. Haidar, and A. Abdelfattah, "With extreme computing, the rules have changed," *Computing in Science Engineering*, vol. 19, no. 3, pp. 52–62, 2017.

[2] R. Nishtala, P. Hargrove, D. Bonachea, and K. Yelick, "Scaling communication-intensive applications on bluegene/p using one-sided communication and overlap," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–12.

[3] B. Barney *et al.*, "Introduction to parallel computing," *Lawrence Livermore National Laboratory*, vol. 6, no. 13, pp. 1–10, 2010.

[4] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "A tool for selecting the right target machine for parallel scientific applications," in *Proceedings of the International Conference on Computational Science, ICCS*, 2013, pp. 1824–1833.

[5] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature for performance analysis and prediction," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 7, pp. 2009–2019, 2015.

[6] D. Bailey, E. Barszcz, J. Barton, and D. Browning, "The NAS Parallel Benchmarks," *International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 66–73, Jan 1991.

[7] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional," *Journal of High Performance Computing Applications*, vol. 14, pp. 330–346, Jan 2000.

[8] J. Zhai, W. Chen, and W. Zheng, "Phantom: Predicting performance of parallel applications on large-scale parallel machines using a single node," in *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '10, 2010, pp. 305–314.

[9] S. Sharkawi, D. DeSota, R. Panda, S. Stevens, V. E. Taylor, and X. Wu, "SWAPP: A framework for performance projections of HPC applications using benchmarks," in *IPDPS*, 2012, pp. 1723–1731.

[10] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13, 2013, pp. 45:1–45:12.

[11] Q. Xu and J. Subhlok, "Construction and elevation of coordinated performance skeleton," in *International Conference on High Performance Computing*, 2008, pp. 73–86.

[12] Q. Xu, J. Subhlok, R. Zheng, and S. Voss, "Logicalization of communication traces from parallel execution," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, 2009, pp. 34–43.

[13] Q. Xu and J. Subhlok, "Construction and elevation of coordinated performance skeleton," in *International Conference on High Performance Computing*, 2008, pp. 73–86.

[14] L. Van Ertvelde and L. Eeckhout, "Dispersing proprietary applications as benchmarks through code mutation," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 2, pp. 201–210, 2008.

[15] J. Zhai, T. Sheng, J. He, W. Chen, and W. Zheng, "Fact: fast communication trace collection for parallel applications through program slicing," in *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, 2009, pp. 1–12.

[16] B. J. Barnes, J. Garren, D. K. Lowenthal, J. Reeves, B. R. de Supinski, M. Schulz, and B. Rountree, "Using focused regression for accurate time-constrained scaling of scientific applications," in *IPDPS*, 2010, pp. 1–12.

[17] B. C. Lee and D. M. Brooks, "Methods of inference and learning for performance modeling of parallel applications," in *in Proc. of the International Symposium on Principles and Practices of Parallel Programming*, 2007, pp. 249–258.

[18] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *Proceedings of the 11th International Euro-Par Conference on Parallel Processing*, ser. Euro-Par'05, 2005, pp. 196–205.

[19] S. Prakash and R. L. Bagrodia, "Mpi-sim: using parallel simulation to evaluate mpi programs," *Proceedings of the 30th conference on Winter simulation*, pp. 467–474, 1998.

[20] M. M. Tikir, M. Laurenzano, L. Carrington, and A. Snavely, "PSINS: an open source event tracer and execution simulator for MPI applications," in *Euro-Par 2009 Parallel Processing, 15th International Euro-Par Conference, Delft, The Netherlands, August 25-28, 2009. Proceedings*, 2009, pp. 135–148.

[21] F. J. R. Perez and J. Miguel-Alonso, "INSEE: an interconnection network simulation and evaluation environment," in *Euro-Par 2005, Parallel Processing, 11th International Euro-Par Conference, Lisbon, Portugal, August 30 - September 2, 2005, Proceedings*, 2005, pp. 1014–1023.

[22] M. Noeth, P. Ratn, F. Mueller, M. Schulz, and B. R. de Supinski, "Scalatrace: Scalable compression and replay of communication traces for high-performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 8, pp. 696 – 710, 2009.

[23] X. Wu and F. Mueller, "Scalaextrap: Trace-based communication extrapolation for spmd programs," in *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '11, 2011, pp. 113–122.

[24] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "Scalability of parallel applications: An approach to predict the computational behavior," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2015, pp. 398–403.

Javier Panadero is a PostDoc researcher at the Computer Science, Multimedia and Telecommunication Department at Open University of Catalonia (UOC). His major research areas are: performance prediction of HPC applications, Modeling and analysis of parallel applications and Simulation of parallel and distributed system. He has co-authored a total of 11 full-reviewed technical papers in journals and conference procedings.

Alvaro Wong is an Associate Researcher at the Computer Architecture and Operating System Department at University Autonoma of Barcelona, Spain. He has worked in performance prediction of HPC applications in the ITEA 2 European Project No 09011, research centers & industries. He has co-authored a total of 11 full-reviewed technical papers in journals and conference proceedings.

Dolores Rexachs is an Associate Professor at the Computer Architecture and Operating System Department at University Autonoma of Barcelona (UAB), Spain. She has been the supervisor of 9 PhD thesis and has been invited lecturer in Universities of Argentina, Brazil, Chile and Paraguay. The research interests include parallel computer architecture, parallel I/O subsystem, fault tolerance in parallel computers, tools to evaluate, predict, and improve the performance in parallel computers. She has co-authored more than 70 full-reviewed technical papers in journals and conference proceedings.

Emilio Luque is an Emeritus Professor at the Computer Architecture and Operating System Department at University Autonoma of Barcelona, Spain. Invited lecturer at universities in the USA, South America, Europe and Asia, key note speaker in several conferences and leader in several research projects funded by the European Union (EU), Spanish government and different industries. His major research areas are: parallel and distributed simulation, performance prediction and efficient management of multicluster-multicore systems and fault tolerance in parallel computers. He has supervised 19 PhD thesis and co-authored more than 230 technical papers in journals and conference proceedings.