Scalability of Parallel Applications: An approach to predict the computational behavior

Javier Panadero[1], **Alvaro Wong**[1], **Dolores Rexachs**[1] **and Emilio Luque**[1]
[1]Department of Computer Architecture and Operating System (CAOS),
University Autonoma of Barcelona, Spain
javier.panadero@caos.uab.es, alvaro.wong@caos.uab.es, dolores.rexachs@uab.es, emilio.luque@uab.es

Abstract— *When a message-passing application is executed many times over a long period of time, using an elevated number of resources, it is critical to predict its behavior before executing it. We propose a methodology to predict the strong scalability behavior for message-passing applications in specific systems. It is focused on characterizing and analyzing the communication and computational application patterns, from a set of executions in small scale, to project their behavior when the number of processes increases. The methodology strives to use a reduced number of resources. This paper presents the general methodology, focusing on validating the computational time model, which is a regression based approach. This model allows us to predict the computation time with high accuracy for a large number of processes. We executed from 16 to 256 processes and we predicted the computation time up until 4,096 processes. For the applications tested, we obtained an error of less than 9%.*

Keywords: Performance Prediction, Scalability, MPI Applications, Computation time prediction

1. Introduction

In recent years, High Performance Computing clusters have increased the number of cores significantly [1]. As a consequence, the users of these systems want to get the maximum benefit from this large number of cores, scaling their applications [2] .

Due to the complex interaction between message-passing applications and the HPC system, many applications may suffer performance inefficiencies, when they scale to a large number of processes. In order to achieve an efficient use of the system, it is critical to know the application behavior in the system before executing it, using an elevated number of resources.

We propose a methodology to analyze and predict the strong scalability [2] behavior for message-passing applications on a given system, by running representative phases of the application, signatures, in small scale. Moreover, the methodology could also be useful for scheduling and code optimization.

Message-passing applications are composed of a set of phases that are repeated throughout the application [3]. These phases were written in the application using specific communication and computational patterns, which follow behavior rules. To obtain these phases automatically, we use the PAS2P tool [4]. PAS2P allows us to generate the PAS2P signature, which contains only the relevant application phases and their repetition rates (weights).

The methodology focuses on characterizing and analyzing the communication and computational patterns of each phase in a transparent way, from a set of signature executions in small scale. By executing this set of signatures, we can obtain quick information about the phases's behavior, as the application scales, to model the general behavior rules of each phase. These rules specified the phase behavior and they allow us to predict their behavior as the number of processes increases. From these rules, the logical application trace is generated for a specific number of processes. This Logical trace has to be complemented with the communication and computational time, to predict application performance.

To predict the computational time, we use a regression-based model by phase, which uses as input data the computation time of each phase of the initial signatures. In many cases, the regression models are limited by the scope of prediction, obtaining a high prediction error when a distant point of the points used to generate the model is predicted.

In order to improve the prediction quality of the model, allowing us to predict distant points with high accuracy, we carry out a change of workload domain, using a workload much less than the original, to emulate the computation time for the original workload with a large number of processes. In this way, we are able to measure a distant point without executing for that large number of processes, to fit the model.

Once the computation time has been predicted for all the application phases, the physical trace is generated, which will be used to predict the communication time and obtain the performance prediction of the application.

In order to validate the method to predict the computational time, we executed from 16 to 256 processes and we predict the computational time 4,096 processes. For all the applications tested, the prediction error is less than 9%.

There are similar works which are related to predicting the computation time based on regression models, from executions for a small number of processes. Barnes et al [5] [6] propose studying the scalability using linear and logarithmic regression functions, isolating computation and communication to predict the application performance. They use black models, where the internal application behavior is unknown. Calotoiu et al [7] present a tool to find scalability bugs. This tool automatically generates asymptotic scaling models for each part (kernel) of the application. The model is based on regression models, to fit the performance data from a set of small-scale performance experiments. Another similar work, presented by L. Carrington et at [8], offers a methodology for extrapolating the computational time of large scale applications by capturing the details of computational behavior at a series of smaller core counts. Unlike these works, our method proposes measuring a distant point using a reduced set of resources, in order to fit the computation regression model, improving the quality of prediction for far points.

This paper is organized as follows: Section II presents the proposed methodology to predict the scalability behavior, Section III presents the computation time prediction model, Section IV presents the experimental validation and finally Section V, the conclusions and future work.

2. Proposed methodology

The main goal of the methodology is to model the parallel application to analyze and predict the strong scalability behavior on a given system, by executing a limited set of application signatures in small scale, as is shown in Fig. 1.

Parallel applications are typically composed of patterns of computation and communication that are repeated throughout the application. These patterns are grouped in phases, which compose the application signature. If we execute the signature for different number of processes, we can observe that the number of phases remains constant, but their patterns change their behavior following behavior rules.

Analyzing the behavior of the phases, we know that the communications, the communication volume, the number of instructions and the computational and communication time can change, modifying their behavior, but the work to be carried out will still be the same, distributed among more processes, because we are working with strong scalability. In order to model the general behavior rules of communication, computation and weight of each phase, to project their behaviors as the number of processes increases, the phases of the signature for a different number of processes will be related by functional similarity.

Once these general rules have been modeled, we can generate the logical trace for any number of processes. This trace is composed of the communication events, the number of instructions and the weight of each phase. The trace is generated per process instead of a global trace with the objective being to model each process indepently.

To predict the application performance, the logical trace has to be complemented with the communication and computational time. To predict the computation time, we use

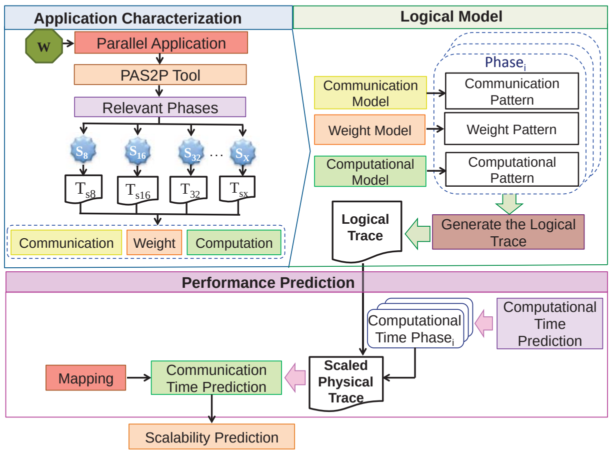


Fig. 1: Proposed Methodology

a regression-based model by phase, which uses the computation time of each phase of the initial set of signatures as input data. In order to improve the prediction quality of the regression computation model, allowing us to predict points for a large number of processes with a high accuracy, we use a method based on a change of workload. This method allows us to measure the phase computation time for an elevated number of processes (distant point), executing the signature for a reduced number processes and a small workload. In this way, we introduce a new distant point in the model, which allows us to fit the initial computation regression model, improving its quality of prediction.

Once the computation times have been provided to the logical trace, the physical trace is generated, which will be executed by pieces in a small number of cores, in a iterative way, until all the process has been measured, to predict the application performance.

In the next subsections, the steps of the methodology are presented.

2.1 Application Characterization

This step consists of characterizing the application behavior (communication and computation) to obtain information to build its logical trace. In order to do that, we carry out a set of signature executions for a small and different number of processes, which will be analyzed to extract information from each phase. The application signature extracts information of the application phases, which will be saved in a trace file per process. Fig. 2 shows an example of trace file obtained with the signature. The trace provides information

#Process	Phase	Type of primitive	Source	Dest.	Comm. Volume	#Inst.	Comm. Time (ns)	Comp. Time (ns)
4	1	MPI_Irecv	0	1	4000	756	4000	326
4	1	MPI_Send	0	1	4000	456	2345	134
4	1	MPI_Wait	0	1	4000	456746733	83593535	7654
4	1	MPI_Irecv	0	2	2000	975	7533	454
4	1	MPI_Send	0	2	2000	875	5366	332
4	1	MPI_Wait	0	2	2000	357876543	45326854	3466

Fig. 2: Trace file of the phase 1 for the process 4.

about the phase id, the type of MPI primitive, the source and destination of the communication, the communication volume in bytes, the communication time in nanoseconds and the computational time in nanoseconds.

It is noteworthy that the signature execution time is about 1% of the application execution time, covering approximately 95% of the whole application.

2.2 Logical Model

Once the phases have been characterized, the communication and computation patterns and the phase weight have to be analyzed and modeled to generate the general behavior rules of each phase. These rules will be used to generate the logical application trace for a greater number of processes.

2.2.1 Communication pattern modeling

The communication pattern 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 size of the message. To model the behavior of each phase, all the phases of the signature for a different number of processes will be related by functional similarity. 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, only in the way in which the application was developed. Once the phases have been related, the predicted data volume equation 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 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. Fig. 3 shows an overview of the procedure. This algorithm is based on obtaining the communication equations (eq.$_{processes}$) for each phase (local equations), which represent its communication pattern. From these equations, the general equation is modeled by each phase, which allows us to predict the evolution of the communication pattern of the phase for a greater number of processes.

2.2.2 Weight modeling

In order to model the weight behavior, regression models are used. Due to the deterministic way of the weight behavior as the application scales, there is a linear dependence between the number of processes and the weight of the phase. For this reason, linear regressions are initially more appropriate to fit the weight, because it allows us to obtain a prediction equation such as $y = a + bx_0$, using as an independent variable the number of processes to execute the application, which represents exactly the phase weight behavior, obtaining a $R - square = 1$.

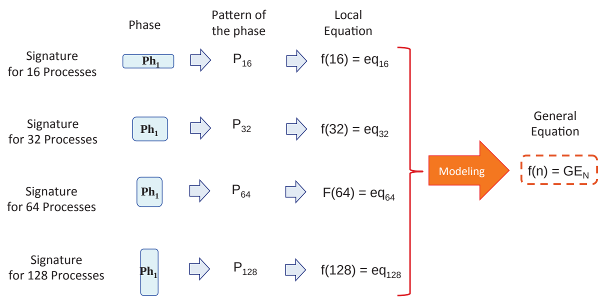
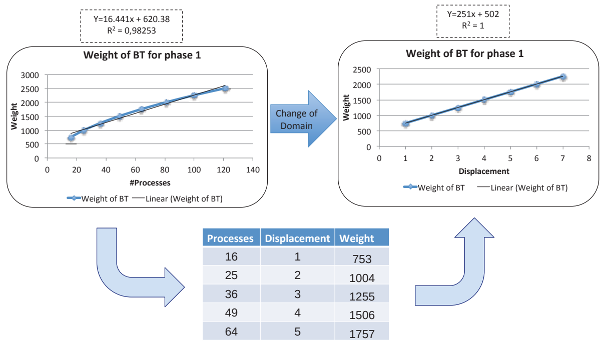


Fig. 3: Modeling the general equation of the communication pattern from a set of application signatures

Scientific applications cannot be executed for any number of processes, but they also follow execution 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-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. In fig. 4 we show an example, where through the limitations of the application (the users can only execute the application using a square number of processes), we use this to generate the model for the input points: 16, 25, 36, 49 and 64 processes.

As we can see in the figure, the distance of the input points used to model the regression is non-uniform, so, if we fit the points by a linear regression, we obtain a $R - square = 0.98253$. Through the theory of statistical regression models, we know that if we use an equation with this correlation index, the prediction error will be considerable and it will be higher as we move away from the executed points. In order to use a linear regression with an $R - square = 1$, we make a linealization process based on a change of domain, where the objective is to obtain an uniform distance among all the points. As we can see in the figure, we change the number of processes by a sequential index (displacement), making a distance of 1 for all the points. In this way, we obtain a $R - square = 1$ using a linear regression.



Processes	Displacement	Weight
16	1	753
25	2	1004
36	3	1255
49	4	1506
64	5	1757

Fig. 4: Modeling the weight of the phase

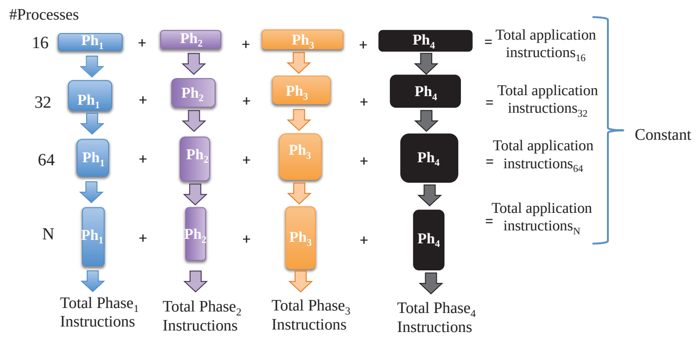

#Processes
16
32
64
N
Ph1
Ph2
Ph3
Ph4
Total application instructions16
Total application instructions32
Total application instructions64
Total application instructionsN
Constant
Total Phase1 Instructions
Total Phase2 Instructions
Total Phase3 Instructions
Total Phase4 Instructions

Fig. 5: Behavior of the strong scalability by phase.

2.2.3 Computational Pattern Modeling

In strong scalability, the application workload 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, being the total number of instructions practically constant. We can extrapolate this concept to the application phases, maintaining its total number of instructions constant, as the application scales, as is shown in fig. 5.

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. To model these equations, the processes with a similar behavior in computation, that is a similar number of instructions (95% similarity), are grouped in Instructions Groups (IG_i). The total number of instructions of each Instruction Group 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 is the total number of instructions of the phase, as is shown in Eq. 1, where x is the total number of groups.

Fig. 6 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,

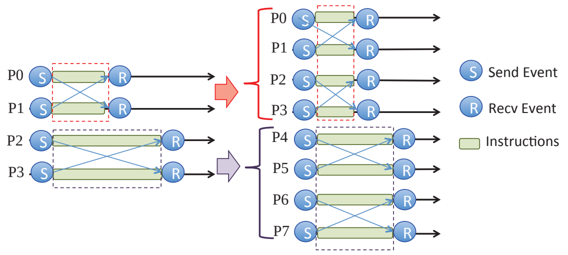


Fig. 6: Distribution of instructions as the number of processes increases.

for this example, we have 2 different groups, IG1 and IG2, where each group distributes its number of instructions in a specific way, following a behavior rule of distribution. As we show in the Eq. 2, which calculates the total number of instructions of the set of processes of a group *(n)*, the total instructions of the group is the sum of the instructions of each process *(Pi)* involved in the group, multiplied by the weight of the phase.

If we focus on Eq. 2, the aim is to predict the term "P_i" for a greater number of processes. We know that the term $TotalIG$ is constant, the weight of the phase is predicted by linear regression methods, and the number of processes between the instructions distributed in the group have been modeled. Then, we can predict the instructions of each process, isolating the term "Pi", as is shown in Eq 3.

$$TotalPhaseInstructions = \sum_{i=1}^{x} TotalIG_i * weight \quad (1)$$

$$TotalIG = \sum_{i=0}^{n} (P_i) * weight \quad (2)$$

$$P_i = \frac{\frac{TotalIG}{n}}{weight} \quad (3)$$

2.3 Performance prediction

The logical trace has to be complemented with the computational time, in order to generate the physical trace. To predict the computation time, we use a regression-based model by phase. As this is the main point of this work, this procedure will be explained in detail in the next section.

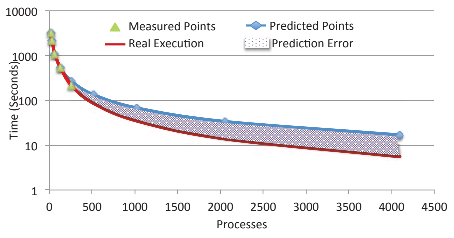
Once the physical trace has been generated, the communication time is predicted. To predict the communication time, the physical trace will be executed by range of processes in a reduced number of cores, in an iterative way, until all the processes have been executed. Once the communication time has been predicted, we will have the predicted execution times of each phase and their weight. Then, we apply eq. 4 to obtain the application performance, where PET is the Predicted application Execution Time, *m* is the number of phases, *TEPhasei* is the Phase i Execution Time and *Wi* is the weight of the phase i.

$$PET = \sum_{i=1}^{m} (TEPhasei)(Wi) \quad (4)$$

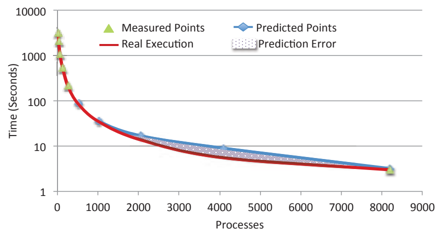
3. Computational time prediction

To predict the computational time of each phase, we use a regression-based approach named Computational Regression Model (CRM), which uses as input data the computational time of each phase for the initial set of signatures.

Despite predicting the computational time by phase instead of the whole application, the prediction error improves



(a) Computation Regression Model (CRM)



(b) Computation Regression Model (CRM) with distant point

Fig. 7: Regression models used to predict the computation time

considerably, because each phase has a different computation behavior which has to be approximated by a specific regression function, we know that the regression models are limited by the scope of the prediction. If we predicted a distant point from the real points used to generate the model, we would obtain an elevated prediction error as we move away from the measured points, as we can see in fig. 7a.

In order to avoid this problem, and therefore to improve the quality of prediction for a large number of processes (distant points) of the model, we propose a method which consists of measuring a far point, without using a large number of application processes and resources, to fit the initial Computational Regression Model (CRM) with this new point, as we can see in fig. 7b. Using this method, we manage to improve the accuracy of the model, predicting far points with a high level of accuracy.

The proposed method to measure this new point is based on doing a change of workload domain, using a workload much less than the original, to emulate the application computation 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 and cache misses by each process, rather than the original workload executed over a large number of processes, to emulate the computational time by process of the phase. In fig. 8 we show an example. We have an application, which is executed for 64 processes with a workload *W*. This workload is distributed between the application processes in a uniform way, with each process receiving a work *w'* . If we executed the same application for 4 processes with a new workload *M*, which is *w'* x 4, the processes are carrying out the same work (same instruction number and cache misses) as when executing the application for 64 processes with a workload *W* .

In fig. 9 we show a flowchart of the method used to predict the computation time of a phase:

1) From the logical trace obtained in the modeling step, we generate a new table, named Instruction Prediction Table (IPT), which contains a computational global vision of each phase. 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. Moreover, contains two different parts, a measured part and a predicted part. The measured part is generated from the information obtained during the execution of the initial set of signatures, while the predicted part is generated from the equations of computation generated in the logical trace. The measured part allows us to validate the accuracy of the model.

In order to obtain more than one Instruction Group in the computation pattern modeling, we generate as 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.

2) At the same time, to generate the IPT table, we use the computation time of each phase of the initial set of signatures, in order to generate the initial Computation Regression Model (CRM), which will be used to predict the computation time of each phase.

3) From the IPT, 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 obtain a distant point to improve the initial CRM model. In this case, we use the regression equation obtained in the CRM model, generated in the last step, to predict the computation time. Otherwise, we follow on to the next step to obtain a distant point.

4) In order to measure this new point, we select a small workload and we execute the signature with this workload for a small number of processes. We start executing the

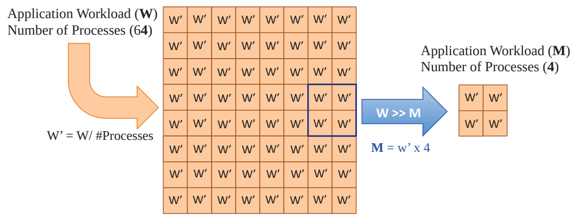


Fig. 8: Emulating the computation time of a process changing the application workload

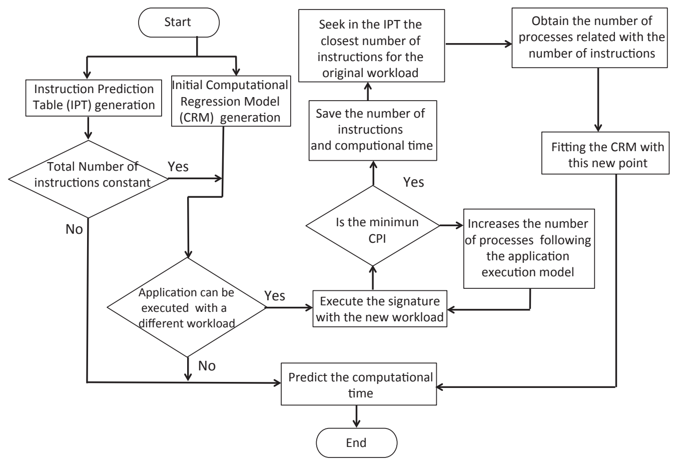

Start
Instruction Prediction Table (IPT) generation
Initial Computational Regression Model (CRM) generation
Total Number of instructions constant
Yes
No
Application can be executed with a different workload
Yes
No
Execute the signature with the new workload
Is the minimun CPI
Yes
Increases the number of processes following the application execution model
Save the number of instructions and compuational time
Seek in the IPT the closest number of instructions for the original workload
Obtain the number of processes related with the number of instructions
Fitting the CRM with this new point
Predict the computational time
End

Fig. 9: Flowchart of the method used to predict the computation time

signature for 16 processes, and we increase the number of processes following the application execution model, until we find the minimum CPI of the phase. We select the minimum CPI because of two main goals, the first being that by using the minimum CPI we are sure that the workload is small enough to obtain a sufficiently distant point to fit the regression. The second reason is to avoid the effect of cache misses, since if we predict the computation time between the points of signatures used to generate the CRM model and the distant point, the model considers the cache effects. In addition, if we predict the computation time from the distant point, the cache effects practically do not have any influence in the model. Once we have executed the signature, we save this number of instructions and the computational time. We know that in some cases, because of 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.

5) Then, we relate the number of instructions obtained in the previous step, with the number of processes for the original workload. In order to obtain this number of processes, we used the IPT generated in step 1. In the table, we seek the number of instructions closest to the number of instructions obtained by the small workload. Then, we will obtain the number of processes for this number of instructions.

6) Once we have the number of processes for the original workload, and the computational time, measured in step 4, we incorporate this new point to the the CRM model, to generate a more accurate new regression function.

7) We use this new regression function to predict the computation time.

4. Experimental Validation

In this section, the method to predict the computation time has been validated. We used different benchmarks such as: BT, CG, SP and LU from the NPB NAS [10] suite, using as input class D. Moreover, we used two applications: QCM [11] and N-Body. We predicted the computation time of each phase for BT and SP for 1024 processes, CG, LU and N-body for 4096 processes and QDIM for 2048 processes.

Table 1: Instruction prediction table for phase 1 of BT

Number of Instructions	Number of Processes	Weight	Weight Displacement	Total inst. Number
Measured Values (Initial Set of Signatures)				
1539358893	16	1255	3	30910326571440
820993294	25	1506	4	30910397519100
488835328	36	1757	5	30919812166656
314154733	49	2008	6	30910312489336
213799791	64	2259	7	30910318583616
152035493	81	2510	8	30910336081830
111953973	100	2761	9	30910491945300
Predicted Values (Instruction Prediction Table)				
1539358780	16	1255	3	30910324302400
.....	.....	.....	.....	.....
152035395	81	2510	8	30910316157450
111953349	100	2761	9	30910319658900
84813133	121	3012	10	30910315829644
65784545	144	3263	11	30910315829644
.....	.....	.....	.....	.....
1877266	1600	10291	39	30910315829644
1744266	1681	10542	40	30910315829644
1623539	1764	10793	41	30910315829644
1513701	1849	11044	42	30910315829644

For the BT, we predicted the computational time of each phase for 1,024 processes. We executed the signatures for 16, 36, 64, 81 and 100 processes, using the workload D, to generate the initial Computation Regression Model (CRM). We obtained 6 phases for this application.

For the case of phase 1, first of all, we modeled the computation pattern and the weight pattern in the logical model step. Through the modeling of the computation pattern, we obtain that all the processes have the same number of instructions, so we have only one Instruction Group, therefore, one IPT. Regarding the weight modeling, we used the linear regression equation $y = 251 * x + 502$, where "y" is the predicted weight and "x" is the displacement, as we increase the number of processes. This regression equation has a $R - square = 1$. From this information, we generate the IPT table, which is shown in Table 1. As we can check on the top of the table, the total number of instructions for the phase is practically constant, as the number of processes increases. For this reason, we look for a far point in order to be provided to the CRM.

To obtain a far point, we executed the signature of the BT using workload B (much less than workload D) for a reduced number of processes, until we found the minimum CPI. In table 2, we show the information of the different signature executions for phase 1 for this workload. We

Table 2: Signature executions for BT using the CLASS B

Number of Processes	Number of. Instructions	Number of Cycles	LLC Misses	CPI	Computation time (nsec.)
16	34356426	27621512	20241	0.803	17263445
25	17836224	14324977	10275	0.803	8953111
36	9915863	7861580	5479	0.792	4913488
64	4027761	3104163	1467	0.770	1940102
81	2392021	1721033	865	0.719	1075646
100	**1759558**	**1265527**	**695**	**0,719**	**790954**

show the number of processes for which the signature was executed, the number of instructions by process, the cycles, the misses of Last Level of Cache (L2), the CPI and the computation time in nanoseconds. As we can see in the table, from 81 processes we obtain the minimum CPI (0.719). We know what the minimum CPI is because for the next execution (100 processes), we obtain the same CPI and the number of misses is insignificant. Then, we select the number of instructions for the last execution of 100 processes (1759558) and its computation time (790954 ns).

The next step is to relate the instructions obtained for workload B with the number of processes of workload D. In order to do that, we seek the number of instructions obtained (1759558) in the Instruction Prediction Table for this phase.

As we can see in table 1, the closest number of instructions is 1744266, which has a difference of 0.87% with the number of instructions of workload B (1759558). This number of instructions corresponds to the execution of class D with 1681 processes. Thus, we select this number of processes to improve our model.

We introduce this new point in our model (number of processes and execution time), obtaining the regression equation $y = 9 * 10^9 * x^{-1.547}$, where the variable y is the computation time and the variable x is the number of processes to be predicted. We used this equation to predict the computation time for the phase for 1024 processes. As we can see in table 3, we obtain a prediction error of 2.11%.

In the same table, we show both the prediction error for the other phases of BT and the other applications tested, which were predicted to carry on with the same procedure. For the CG, LU, N-body and QDIM, we execute the signatures from 16 to 256 processes, while for SP, we execute the same number of signatures as for BT. All the phases of the application fulfilled the condition that the total number of instructions is constant. Therefore, the method to find the distant point was applied. For all the phases of the applications, the prediction error is below 9%.

Table 3: Summary of error prediction for the application phases

Summary of phases of BT (prediction for 1024 processes)				
Phase Num.	Real time(ns)	Predicted Time(ns)	Prediction Error (%)	Regression Equation
Phase 1	1941784	1982934	2.11%	$y = 9 * 10^9 * x^{-1.547}$
Phase 2	35960361	39114740	8.77%	$y = 1 * 10^{11} * x^{-1.132}$
Phase 3	165862020	160857620	3.01%	$y = 2 * 10^{11} * x^{-1.028}$
Phase 4	451140	480522	6.51%	$y = 3 * 10^{10} * x^{-1,593}$
Phase 5	2214673	2096107	5.35%	$y = 8 * 10^{10} * x^{-1,522}$
Phase 6	36062311	39114740	8.46 %	$y = 1 * 10^{11} * x^{-1.132}$
Summary of phases of CG (prediction for 4096 processes)				
Phase 1	2698523	2796674	3.63%	$y = 2 * 10^{10} * x^{-1.067}$
Phase 2	137928	149985	8.74%	$y = 3 * 10^7 * x^{-0.637}$
Phase 3	344297	375251	8.99%	$y = 2 * 10^7 * x^{-0.478}$
Phase 4	601238	562501	6.44%	$y = 8 * 10^7 * x^{-0.596}$
Summary of phases of LU (prediction for 4096 processes)				
Phase 1	51549	49647	3.83%	$y = 2 * 10^8 * x^{-0.998}$
Phase 2	34645	32573	6.36%	$y = 4 * 10^8 * x^{-1.132}$
Summary of phases of SP (prediction for 1024 processes)				
Phase 1	165110327	163103109	1.21%	$y = 4 * 10^{11} * x^{-0.926}$
Phase 2	561715	598590	6.56%	$y = 4 * 10^{10} * x^{-1.635}$
Phase 3	240359	260203	8.25 %	$y = 2 * 10^{10} * x^{-1.623}$
Summary of phases of N-BODY (prediction for 4096 processes)				
Phase 1	449150	430921	4.23%	$y = 2 * 10^{10} * x^{-0.975}$
Summary of phases of QDIM (prediction for 2048 processes)				
Phase 1	16428976	17724522	7.88%	$y = 3 * 10^{13} * x^{-2.039}$
Phase 2	26478907	25170124	4.94%	$y = 5 * 10^{10} * x^{-0.996}$

5. Conclusions and future work

In this paper, we have presented a methodology that allows us to analyze and predict strong scalability for message-passing applications on a given system, by executing a limited set of application signatures in small scale. The methodology has been explained focusing on validating the method to predict the computational time of each application phase. The proposed method allows us to predict the computation time for a large number of processes with a high accuracy using a reduced number of processes. For all the applications tested, the prediction error is less than 9%.

As future work, we are working on extending the computation model to measure far points of phases which do not have a similar number of instructions, as the number of processes increases.

Acknowledgment

This research has been supported by the MINECO (MICINN) Spain under contract TIN2011-24384

References

[1] N. Attig, P. Gibbon, and T. Lippert, "Trends in supercomputing: The european path to exascale," *Computer Physics Communications*, vol. 182, no. 9, pp. 2041 – 2046, 2011.

[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] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature for performance analysis and prediction," in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2014 (Acepted).

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

[5] 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.

[6] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ser. ICS '08, 2008, pp. 368–377.

[7] 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.

[8] L. Carrington, M. Laurenzano, and A. Tiwari, "Characterizing large-scale hpc applications through trace extrapolation," *Parallel Processing Letters*, vol. 23, no. 4, 2013.

[9] J. Dongarra, A. D. Malony, S. Moore, P. Mucci, and S. Shende, "Performance instrumentation and measurement for terascale systems," in *European Center for Parallelism of Barcelona*, 2003, pp. 53–62.

[10] 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.

[11] S. Hioki, "QCDMPI—pure QCD monte carlo simulation code with mpi," *Nuclear Physics B-Proceedings Supplements*, vol. 63, pp. 1000–1002, Apr. 1998.