Benchmark based on application signature to analyze and predict their behavior.

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

[1] Universidad Católica del Maule,
Departamento de Computación e Industrias
Talca, Chili.
[2] Universidad Autónoma de Barcelona,
Computer Architecture and Operating System Department,
Barcelona, Spain.
ftirado@ucm.cl, alvaro.wong@uab.es, dolores.rexachs@uab.es,
emilio.luque@uab.es

Abstract. Currently, there are benchmark sets that measure the performance of HPC systems under specific computing and communication properties. These benchmarks represent the kernels of applications that measure specific hardware components. If the user's application is not represented by any benchmark, it is not possible to obtain an equivalent performance metric. In this work, we propose a benchmark based on the signature of an MPI application obtained by the PAS2P method. PAS2P creates the application signature in order to predict the execution time, which we believe will be very adjusted in relation to the execution time of the full application. The signature has two performance qualities: the bounded time to execute it (a benchmark property) and the quality of prediction. Therefore, we propose to extend the signature by giving the benchmark capacities such as the efficiency of the application over the HPC system. The performance metrics will be performed by the benchmark proposed. The experimentation validates our proposal with an average error of prediction close to 7%.

Keywords: High Performance Computing, MPI Application, Performance Prediction, Performance metrics.

1 Introduction

High Performance Computing (HPC) systems combine powerful hardware and software, present in clouds or clusters, used by scientists as an indispensable tool in many areas of research. The performance evaluation of these systems requires that the benchmarks subject the entire system to great stress and that they are representative of the type of workload that is executed on the machines.

In HPC, these benchmarks have followed two different approaches: The first approach consists of a set of applications and kernels, such as NAS Parallel Benchmark (NPB) [2], which aim to represent the totality of the measures of

performance through a set of relevant workloads. The second approach consists of a single application susceptible to the properties of the system that it considers most relevant for the typical workloads, such as the well-known High Performance Linpack (HPL) [4], which is used to classify the systems in the Top500 list [10].

When using a benchmark, it will be executed in such a way as to maximize performance, thus hiding the influence of certain properties and emphasizing the influence of other properties. For example, running HPL on very large problems makes the influence of the interconnection network negligible, causing this action to go unreported. This has the result of making it difficult to obtain an idea of application performance for different problem sizes.

Each application has data, memory structures and different arithmetic calculations, since each one tries to solve a different problem. This is reflected in the amount of memory that is needed to load the data and the type of instructions that the CPUs will compute and the memory access pattern. That is why systems exhibit different performance indices according to the applications that execute them, making it difficult to reflect, relate or select the appropriate benchmark that reflects the type of operation or the amount of data to be computed which is similar to that of the application.

PAS2P (Parallel Application Signature for Performance Prediction) [12] is a tool that allows us to analyze the dynamic behavior of the application, characterizing it in a set of phases that represent the performance of the application. With the phases, PAS2P constructs the application signature, which is defined by the set of phases which represent the application behavior at the performance level. To evaluate a system, the signature executes the phases to measure their execution times, which are multiplied by their weights, in order to obtain the total execution run time of the application.

We propose using the signature in order to create the benchmark that represents the application behavior, keeping the same memory, compute and communication requirements, as well the memory access pattern, reproducing the specific calculation and workload properties that the application has. The signature will allow us to obtain the performance behavior of each phase, where we can apply performance metrics such as the efficiency and the application execution time, obtaining results in a bounded time with high accuracy.

The performance evaluation proposal is presented in Fig. 1. Here we can observe that one of the problems is selecting the benchmark that has similar behavior to the parallel application in order to evaluate the suitable HPC system. On the other hand, our performance evaluation proposal is extracting the benchmark which represents the application performance that will be executed in order to evaluate the target machines.

In the following Section, the related works are presented. In Section III, we provide general information on PAS2P methodology and Section IV presents the benchmark model based on PAS2P. Section V provides the experimental results and Section VI presents the conclusions and future work.

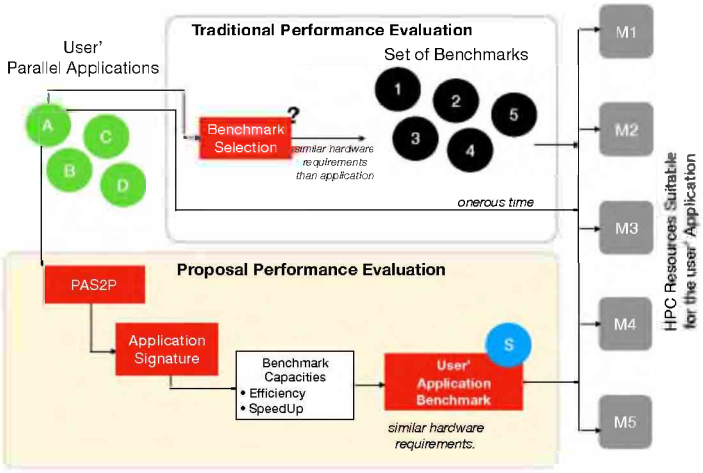

Fig. 1: Performance Evaluation using benchmarks and Proposal Performance Evaluation.

2 Related Work

In recent times, the supercomputing community has paid significant attention to three benchmarks: The HPL mentioned above, the High Performance Conjugate Gradient (HPCG) [6] and the High Performance Geometric Multigrid (HPGMG) [1]. Although HPL offers direct solutions with a computational complexity of $O(N^3)$, the two alternatives benchmarks, HPCG and HPGMG, offer iterative solutions with a linear complexity of computational calculation $O(N)$.

According to the benchmarking present in the literature, HPL and HPCG act as performance metrics and data access patterns commonly found in scientific applications, while HPGMG aims to reproduce the requirements of a specific workload class, without being clearly linked to any calculation or memory pattern, providing a balance of machine capabilities in relation to the scientific application of interest.

All the most commonly used benchmarks in HPC, in particular HPL, HPCG and HPGMG, significantly define a notion of the size of the problem, which they use as parameters to be established. But this is not enough to characterize performance, since benchmarks generally reflect the behavior of a limited set of applications, at best.

There are numerous benchmarks that represent a variety of domains. On the one hand, there is the suite of applications highlighted by the Mantevo mini-applications [5] and the Parallel NAS Benchmarks [2]. On the other hand, there is another approach consisting of a single application susceptible to the system properties that it considers most relevant for the workloads, such as HPL [4], HPCG [6]. These benchmarks are written in C / C ++ or Fortran and parallelized with MPI message passing.

Mantevo [5] presents miniapps of various kinds of scientific applications. These applications are based on the property that the performance is usually concentrated in a small subset of lines of code. This property is exploited by the miniapps, encapsulating only the most important computational operations, achieving a code smaller than the original, capturing the performance behavior of the application.

NAS Parallel Benchmarks (NPB) [2] are small application suites designed to help in the evaluation of the performance of parallel supercomputers developed by NASA. The benchmarks are based on Computational Fluid Dynamics (CFD) applications. The selection of the workload of the applications is given by five predefined classes (A, B, C, D or F). The application suite is composed of eight problems classified into five cores that mimic five numerical methods used in CFD and three simulated applications that represent a series of data calculations in complete CFD codes, which require a greater amount of resources than the cores.

HPL [4] consists of a single application composed of a single kernel. This became a point of reference in the 90s to measure the rate of execution in floating point and thus enabling the classification of supercomputers, originating the TOP500 project. HPL solved a complex system of linear equations with a complexity of $O(n^3)$. One of the main limitations of this benchmark is that it does not consider the transfer of memory or the cost in communications, which today are fundamental properties of scientific message passing applications.

On the other hand, in 2014 the benchmark HPCG [6] was developed, taking into account the limitations ofHPL. It obtains a better representation of the behavior of scientific applications, making multiplications of matrix vectors in order to strongly link the benchmark with hardware memory. In addition, it uses a simple pattern and small communication messages, which make the communication time depend mainly on the latency of the interconnection network [8].

3 PAS2P Overview.

Parallel scientific applications are typically composed of a set of phases that are repeated throughout the application. These phases are written in the application code using specific communicational and computational patterns. As shown in Fig. 2, PAS2P [12] identifies the application phases in a transparent and automatic way, and it generates the Application Signature, which contains the application phases (the phases which have an impact on the application's performance) and their repetition rates (weights). The Signature execution allows us to analyze and predict the application performance in an efficient way on target machines, covering approximately 95 per cent of the total application code in 1 per cent of the application execution time.

On the base machine, the PAS2P tool instruments each process of the application, creating a trace file. This trace, composed of hardware counters, is obtained between each MPI call. The instrumentation is performed by the MPI wrapper of the PAS2P dynamic library and the integration with the PAPI [11]

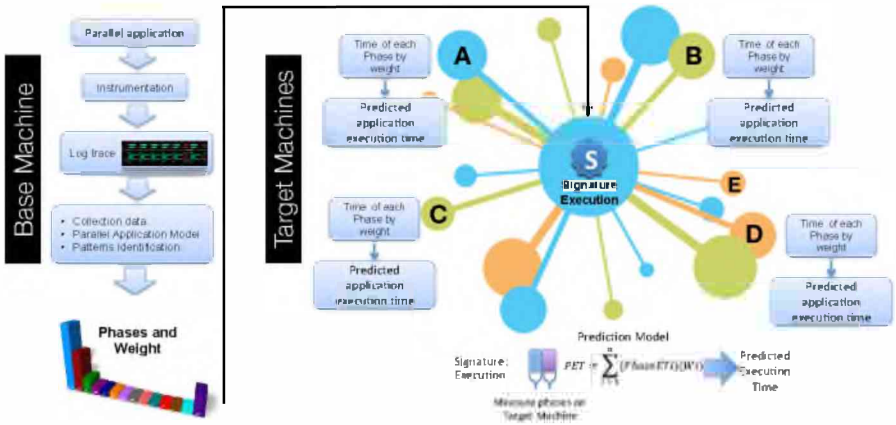

Fig. 2: PAS2P Overview.

library for the hardware counters. Finally, PAS2P defines as an event an MPI call associated with the computational data between one MPI Call and the next one.

Once the PAS2P tool intruments the application, it analyzes the data collected in order to create a machine-independent application model. To do this, it is necessary to create a logical global clock for all processes to maintain the precedence between the events. When all the events have been ordered, The PAS2P tool creates the logical trace, where the events are inserted so as to later analyze the logical trace to extract the application phases.

To construct the signature, PAS2P instruments the application (binary); to do this, PAS2P re-runs the application using a phase table to instrument and detect where the phases occur. To predict the execution time (PET) of the application on a target machine, the equation shown in Fig. 2 is used. When the signature multiplies the execution time of each phase (PhaseETi) by its weight (Wi defined as the number of phase repetitions), the signature obtains the application execution time.

The information provided by the Performance Prediction allows us to obtain a prediction of performance measures, such as application execution time and performance metrics as computational time and the efficiency of each application phase.

4 Benchmark based on the application signature.

To measure the performance of an HPC system, researchers have often used a set of application kernels as benchmarks [3, 4, 6, 7], a suite of benchmarks and mini-applications [2, 5]. However, it is not always possible to characterize the performance using only benchmarks [9], due to each application having different computing and communication behavior to solve a distinct problem. This is

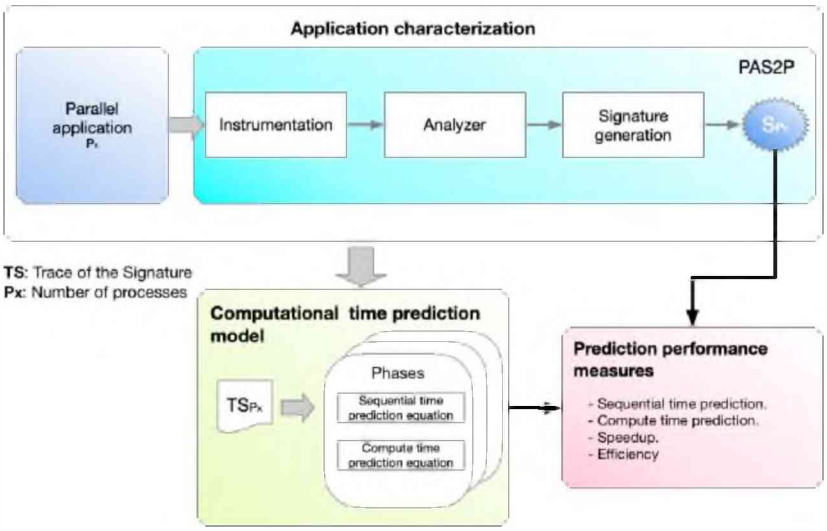

Fig. 3: Analysis and modelling the information given by the signature to obtain performance metrics.

reflected in the instructions that the CPUs will execute and the communication and memory access pattern present in the application. That is why we proposed to use the signature as a benchmark. The signature uses the application code to predict whether the performance will be the same as the application. In other words, this means same compute, same communication messages and same memory access pattern. As we have said before, a very important characteristic of the benchmarks is the bounded time in which they execute, a characteristic which we have transferred to the signature. With this advantage, we guarantee that the quality of the performance results are in a bounded time.

In order to provide this capacity to the application signature, we need to analyze how we can apply the performance metrics in each phase. Each phase represents parallel code between two MPI communications and each phase scales independently from the others. This process is called application characterization (AC), as is shown in Fig. 3.

The execution of the signature allows us to extract information about the behavior of each relevant phase from the application. This information is stored in one trace file per process, as shown in Table 1, called the Signature Physical Trace (TSP_x). The TSP_x groups the information of the application in phases with its respective degree of repeatability (weight), as well as providing information from the source or destination of each message, the type of MPI primitive, the computational time between each MPI communication, the number of instructions, cycles and cache MISSES (L1 or L2).

By using TSP_x, it will be possible to model the behavior of the phases, which provide information on the application processes such as computational time, number of instructions, number of cycles and cache misses, along with the

Table 1: Information of the phases (process 1) for application N-Body with 360,000 particles.

Source	Type of MPI primitive	Destination	Computational Time	Number of Instructions	Cycles	Cache Misses (L2)
PHASE 0 WEIGHT 289						
1	MPI_Irecv	0	3725272115	3923734167	5911572490	1107574
1	MPI_WaitAll	0	70532	62343	85415	7582
1	MPI_ISend	2	17165	412	1271	30
PHASE 1 WEIGHT 10						
1	MPI_Irecv	0	3643785450	3923734084	5778196900	1117380
1	MPI_WaitAll	0	176601	1766014	222832	3021
1	MPI_ISend	2	17687	412	1533	25

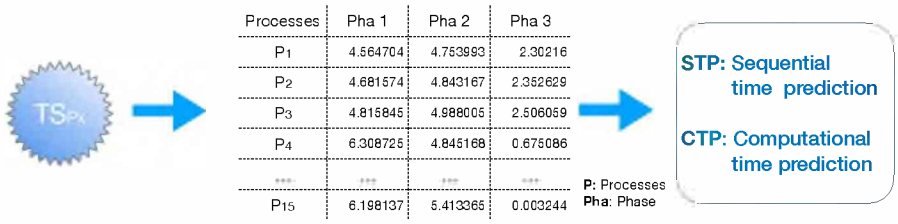

Fig. 4: Signature information: Computational Time of each phase per process.

weight of each phase W_m. With this information, we obtain the computational time prediction (CTP) of the executed phases, which we use to calculate the sequential time prediction (STP), as shown in Fig. 4. This stage is called the Computational Time Prediction Model (CTPM).

To predict sequential time (STP) on a parallel computer, we use the Eq. 1, for which we perform the sum of the computation time of each process of the phase ($pct_{i,p}$) and multiply it by its weight, W_i in all phases i, obtaining an approximation to the sequential execution time of the application. On the other hand, if we add the average computational time of each process, $\overline{x_i}$, and multiply

Table 2: Information obtained by executing the application signature.

CG Class D, 128 Processes					
Phase ID	Compute Time (sec)	Average Compute time (sec)	Weight	Total compute Time prediction (sec)	Representative Compute Time Prediction (sec)
0	0.47	0.004	5024	2361.28	20.09
1	11.15	0.087	5023	56006.45	437.00
2	10.68	0.083	200	2136.00	16.60
3	1.63	0.013	199	324,37	2.58

it by its weight, W_i, in each of the phases, we obtain an approximation of the computational time of the executed application, Eq. 2, where m is the number of phases.

$$STP = \sum_{i=0}^{m} (\sum_{p=0}^{P_x} pct_{i,p} * W_i) \quad (1)$$

$$CTP = \sum_{i=0}^{m} \overline{x_i} * W_i \quad (2)$$

We exemplify the calculation of STP and CTP, which we show in Table. 3. It shows two phases created by the signature when executing the LU application class D with 600 iterations in 128 processes. Each phase contains the compute time of each process expressed in seconds, as well as its weight W_i, the sum of computational time $\sum_{p=0}^{P_{127}} phase_i$, the compute average $\overline{x_{phase_i}}$ and the standard deviation of computational time σ. The STP value is obtained by the sum of the multiplication of the total computational time by the weight w_i for each phase. The CTP value is obtained by multiplying the compute average by the weight in each phase and then the sum of each one of the values.

In Table 2 we show the information obtained by the execution of the CG signature with 128 processes and workload class D with 200 iterations. In the same Table, we show the ID phase, the total computational time (all processes), the average computational time value and the weight of each phase. In the same way, we show the prediction of the total computational time, as well as the prediction of the representative computational time in each phase.

Table 3: Exemplification of LU application computational times with 128 processes.

Processes	Phase 0	Phase 1	
P0	0.69393	1.09571	
P1	0.71858	1.14146	
P2	0.73267	1.14258	
P3	0.72727	1.13940	
...	...	...	
P127	0.72588	0.91355	
Weight (W)	599	598	
$\sum_{p=0}^{P_{127}} phase_i$	105.904	130.353	$\sum_{i=0}^{1} (\sum_{p=0}^{P_{127}} phase_i * W_i) = 141388.350 seg. = STP$
$\overline{x_{phase_i}}$	0.837	1.018	$\sum_{i=0}^{1} \overline{x_{phase_i}} * W_i = 1104.597 seg = CTP$
σ	0,048	0,053	

The last stage, CTPM, will allow us to numerically obtain the performance measures of the application that the user executes. STP, as previously mentioned, is obtained if we sum all the phases from the prediction of the total computing time. The value of STP for the CG application represented by the phases shown in Table 2 is 60828.1 seconds. Likewise, when summing the prediction of the computational time representative of all phases, we obtain CTP. The CTP value for the CG application shown in Table 2 is 476.27 seconds. These two times will allow us to obtain performance measures such as speedup and efficiency. These measures are specific to the executed parallel application and reflect its computing and communications behavior, allowing it to have a performance index associated with the executed application.

5 Experimental results

Throughout this section, we will show the results of the measures obtained using the benchmark based on the application signature. In order to validate the prediction results, we compare the obtained performance time with the real execution time of the application.

In order to validate the experimental results, a set of scientific messages passing applications has been selected. Suites with different communication and compute patterns such as NAS parallel benchmarks, Mantevo and the Nbody application have been tested along with the application to be able to analyze their efficiency and execution time in comparison and therefore select the one that run better and validates the results. The four experimental applications are described in the Table 4.

Table 4: Application description.

Application	Description
MiniMD [5].	It is an application of the Mantevo suite on molecular dynamics (MD), it uses the spatial decomposition MD, where the processors of a cluster have subsets of the simulation problem.
LU (Lower-Upper Gauss-Seidel solver) [2].	It is an application of the NAS suite, dealing with fluid dynamics. It solves flows in a cubic domain.
CG (Conjugate Gradient) [2].	It is an application of the NAS suite that uses the inverse power method to find an estimate of the largest eigenvalue of a symmetric sparse matrix using the conjugate gradient method as a subroutine to solve the systems of linear equations.
N-Body.	It is an application that simulates the interaction of a dynamic system of particles under the influence of different physical forces, such as gravity.

Table 5: Cluster characteristics.

Cluster	Characteristics
DELL	AMD Opteron™ 6200 1.60GHz, 8 nodes (512 cores), 256 GB RAM per node (2048 GB total memory), Interconnection Infiniband QDR.

Table 6: Benchmark time results compared to the whole application.

Procs.	Application		Benchmark		Error	
	Execution Time (sec)	Computational Time (sec)	Predicted Execute Time, (PET) (sec)	Predicted Computational Time, (PCT) (sec)	PET (%)	PCT (%)
Application: miniMD						
16	1687.11	1607.21	1509.69	1426.66	-11.75	-12.66
32	839.53	810.15	751.11	726.43	-11.77	-11.52
64	532.14	489.01	463.05	423.59	-14.92	-15.44
128	279.74	245.69	251.44	245.69	-11.25	-12.16
Application: CG						
16	6825.80	6601.83	6840.98	6601.83	0.22	-0.50
32	2360.95	2257.26	2359.94	2243.03	-0.04	-0.63
64	1407.04	1222.85	1379.23	1216.74	-2.01	-0.50
128	757.07	483.41	738.57	475.438	-2.50	-1.67
Application: LU						
16	11735.83	11101.71	12037.17	11075.91	2.50	-0.23
32	5464.36	5195.83	5447.42	5175.61	-0.31	-0.39
64	2647.85	2382.46	2750.72	2361.66	3.74	-0.88
128	1315.88	1110.75	1380.00	1110.75	4.65	-0.55
Application: N-Body						
16	1849.01	1844.83	1675.35	1700.25	-10.37	-8.50
32	927.34	821.15	864.69	860.48	-7.25	-7.05
64	469.50	463.41	446.79	446.87	-5.08	-3.70
128	397.22	386.76	410.92	364.42	3.33	-6.13

For the execution environment, a DELL machine whose characteristics are described in Table 5 was used. For the experimentation set, four different executions were performed for each application in accordance to the number of processes to be executed: 16, 32, 64 and 128 processes. N-Body was executed with a workload of 360000 particles, partitioned into the number of defined processes. For the MiniMD application of the Mantevo suite, we arranged a workload of 192 x 192 x 192 with 500 iterations. The CG and LU application of the NAS suite was executed with a Class D workload with 200 and 600 iterations respectively.

The table 6 shows the results we obtained from the executions. We used 1:1 mapping (one process per core) having a maximum of 128 cores per application. Therefore, if each node of the Dell cluster has 64 cores, when we run the application with 128 cores we are actually using two nodes. The average percentage error in the execution time of the benchmarks was 4.70 %. The maximum value was 14.92 %, which is below the whole application value, obtained by the

miniMD application executed with 64 processes. On the other hand, the average percentage of error in computing time was 5.15 %, and a maximum value of 15.44 % according to the value of the whole application, obtained by the miniMD application with 64 processes. The MiniMD application obtained these results because of its distinct behaviors for different groups of processes, making the PAS2P method have a low representativeness of the application.

The table 6 shows the scalability of the applications. It can be appreciated that the CG and LU applications have a superlinear behavior when they are executed with 16 and 32 processes. This is due to the mapping we used, which favors the use of the second level cache memory. In Fig. 7, we can observe the sublinear behavior of the applications when changing the mapping of the processes.

Our proposal allows us to obtain a prediction of the sequential time of parallel message passing applications, as seen in Fig. 4, thus, allowing us to obtain performance measures that are commonly used such as speedup and efficiency. Fig. 5 and Fig. 6 show the prediction of both metrics with the mapping previously used to avoid the superlinearity of the results, with an average percentage error of 6.1% for speedup and 7.6% for efficiency. As seen in both figures, the results show a similar behavior to that of the real application.

Running the signature instead of the full application has two important benefits. For instance, the prediction of the execution time and the computing time. It also provides extensive information concerning each phase of the application to model its behavior allowing, for example, the prediction of the sequential time of the application. The sequential execution time is often impossible to achieve, due to the amount of memory that a sequential process needs to be executed and the time involved in its execution. Fig. 8 shows the bounded time in which the signature was executed versus the execution time of the entire CG application. As seen, the signature executed up to 80 times faster compared to the application, thus obtaining measurements that will allow us to evaluate the performance of the application with a low error rate.

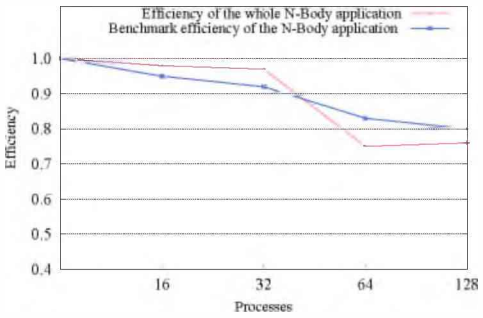

Fig. 5: Benchmark efficiency and whole application efficiency.

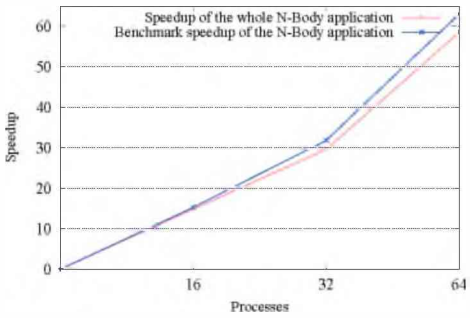

Fig. 6: Benchmark speedup and whole application speedup.

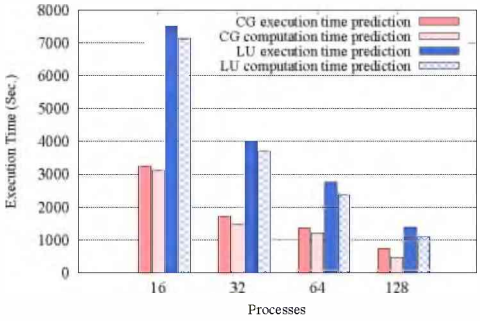

Fig. 7: Prediction of execution time with processes mapping.

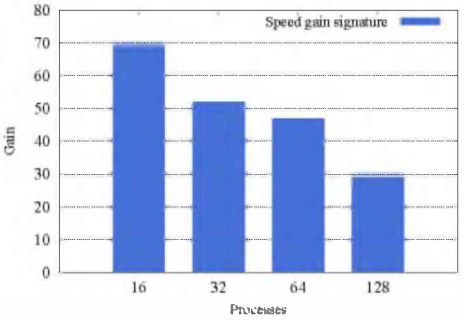

Fig. 8: Speed gain of benchmark on the CG application.

6 Conclusion and Future Work

We validated the proposed benchmark based on the application signature as follows. We gave the signature the same functionality a benchmark has in order to evaluate a system. To achieve this goal, we begin by analyzing the computational behavior on each phase to predict the sequential time that the application would have. In this way, we calculate and predict computational time, speedup and efficiency using the proposed benchmark. The performance measures obtained allowed us to detect inefficiencies without executing the application completely, since the information was obtained directly from the benchmark, thus achieving a bounded execution time with a low margin of error.

Benchmarks that are based on application signatures obtain performance measures of a specific MPI application on a specific machine, without having to depend on a suite of applications or a specific application that characterizes the overall performance. In other words, what we are proposing is a benchmark adapted to the MPI application that the user wants to execute.

Currently, the signature is built using checkpoint libraries. If the user or system administrator wants to evaluate performance in a different system, it is necessary to transport the signature to the new location. One of the disadvantages of using the checkpoint mechanism is the size it achieves. The more processes the application has, the larger the size of the checkpoints we have to save. For future work, we are analyzing how to detach the checkpoint from the signature. Being portable is another important characteristic of benchmarks, we are working on the creation of compute models that include the characterization of memory access pattern as it is an important factor in the performance impact on the execution time.

Acknowledgments

This research has been supported by the Agencia Estatal de Investigación (AEI), Spain and the Fondo Europeo de Desarrollo Regional (FEDER) UE, under con-

tract TIN2017-84875-P and partially funded by a research collaboration agreement with the Fundacion Escuelas Universitarias Gimbernat (EUG).

References

1. Mark Adams, Jed Brown, John Shalf, Brian Van Straalen, Erich Strohmaier, and Sam Williams. Hpgmg 1.0: A benchmark for ranking high performance computing systems. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2014.
2. D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. Technical report, The International Journal of Supercomputer Applications, 1991.
3. Peter N Brown, Robert D Falgout, and Jim E Jones. Semicoarsening multigrid on distributed memory machines. *SIAM Journal on Scientific Computing*, 21(5):1823–1834, 2000.
4. Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9):803–820, 2003.
5. Michael A Heroux, Douglas W Doerfler, Paul S Crozier, James M Willenbring, H Carter Edwards, Alan Williams, Mahesh Rajan, Eric R Keiter, Heidi K Thornquist, and Robert W Numrich. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, 3, 2009.
6. Michael A Heroux and Jack Dongarra. Toward a new metric for ranking high performance computing systems. *Sandia National Lab. Report, SAND2013-4744*, 2013.
7. Adolfy Hoisie, Olaf Lubeck, and Harvey Wasserman. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. *The International Journal of High Performance Computing Applications*, 14(4):330–346, 2000.
8. Vladimir Marjanović, José Gracia, and Colin W Glass. Performance modeling of the hpcg benchmark. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pages 172–192. Springer, 2014.
9. J McCalpin and CA Oakland. An industry perspective on performance characterization: Applications vs benchmarks. *Proc. Third Ann. IEEE Workshop Workload Characterization, keynote address, Sept*, 2000.
10. Hans Meuer, Erich Strohmaier, Jack Dongarra, Horst Simon, and Martin Meuer. Top 500 list, 2012.
11. Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In *Tools for High Performance Computing 2009*, pages 157–173. Springer, 2010.
12. Alvaro Wong, Dolores Rexachs, and Emilio Luque. Parallel application signature for performance analysis and prediction. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):2009–2019, 2015.