Mapping Policies based on Application Signature and Communication Clustering for HPC

Carlos Ramon Rangel, Alvaro Wong, Dolores Rexachs and Emilio Luque
Computer Architecture and Operating Systems Department (CAOS)
University Autonoma de Barcelona
Barcelona, Spain
carlosramon.rangel@uab.es, alvaro.wong@uab.es, dolores.rexachs@uab.es, and emilio.luque@uab.es

***Abstract*—Parallel applications in HPC environments, in some cases, are not being executed using the resources in an efficient way. Since there is some idle time during the execution that can be caused by synchronization due to incorrect process placement, which causes an increase in the execution time, we propose a methodology for parallel applications that attempts to reduce their execution time by selecting a better mapping policy to reduce idle times. To achieve this goal, in a bounded time, we characterize the phases of the PAS2P signature. The PAS2P signature is defined as a set of phases that represent the application's behavior used to predict the Application Execution Time in a target machine. To select a better mapping policy, we characterize the communication pattern of the phases by creating a data matrix, which will be used as an input for clustering techniques in order to generate groups of processes that have a high communication interdependence factor. We performed experiments using scientific applications where we reduced the execution time compared to the default mapping.**

Keywords: Model for efficient execution, Mapping policies, Performance Prediction, MPI applications

I. Introduction

The execution of an application can be inefficient for various reasons, one of which is the congestion in communications. This can be caused by the communication pattern of one application on the interconnection network or the congestion in the output of the compute nodes on the interconnection network. Other reasons for inefficiency could be due to an increase in the computational times originated by cache misses or bottlenecks in the internal communication buses. Modifying the source code of the application to improve efficiency is not always possible, since most of the time the source code is not available, and it may not be very useful to modify the code for a specific machine or architecture.

We propose a methodology to analyze, predict and select an appropriate mapping policy in HPC environments for MPI applications for the number of processes assigned by the user, in order to reduce the execution time or improve the application's efficiency. Some of the available tools for the instrumentation and monitoring of the behavior of parallel applications are DIMEMAS [1], Scalasca [2], and TAU [3]. Unlike these monitoring tools mentioned above, PAS2P [4] monitors and extracts relevant information about the parallel application, generating a signature of the application. As shown in Fig. 1, in the first stage we start executing the application's signature [5] with a default mapping. The signature

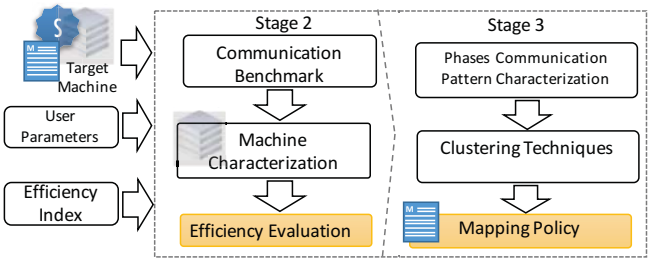

Fig. 1: Methodology overview.

allows us to predict the application's performance in a target machine and it is composed of a set of phases that define the application's behavior. The phases are defined as the parallel segments of code between two MPI communications. To evaluate efficiency in a bounded time, we created a communication benchmark to measure how much time it takes for one message to go from the source process to the destination process on an empty machine (a machine that is not running the user's processes), in order to achieve the minimum transmission times (machine link characterization). In the second stage, with the data obtained by the communicator benchmark, we created a machine characterization table with the minimum transmission time for each specific Bytes size message. Finally, we evaluated efficiency based on idle time to know if the application has idle time which needs to be reduced.

In this paper we focus on the third stage, basing our approach on the clustering method [6], [7]. The communication pattern of the phases is used as input to calculate distances between processes. Then, based on these distances, clusters of processes are detected. With these clusters, the new mapping policy is generated using the machine's architecture, which we will call "clustering mapping", and then we execute the signature with this new mapping. Our aim is to find a better mapping for the application in the given machine. As a result of this stage, we selected the mapping that is the most suitable for our objective, which is the one obtaining the lowest execution time. The applications used to validate the clustering of processes methodology were BT, CG, and SP from the NAS benchmark [8]. We compile these applications for different classes and number of processes. This paper is organized as follows: Section II presents the related work,

Section III presents the proposed methodology, Section IV the experimental validation and finally Section V deals with the conclusion and future work.

II. Related Work

Available tools such as [5] predict the performance analysis of parallel scientific applications in HPC environments. However, they do not take actions depending on whether the execution time is efficient or not. One step beyond PAS2P, which only predicts, is to detect in what way an application execution is more efficient by using the PET as one of the input parameters for the model of phase inefficiencies evaluation. As shown in Fig. 2, two different mappings can generate different idle and computational time per phases. After detecting the idle time per phases, actions that lead to an improvement in the efficiency of the application can be taken. The advantage of performing this analysis using the signature of the application is that in a bounded time, the idle time per phases can be detected. Scalasca carries out a performance and efficiency analysis of HPC applications [2], which supports the performance optimization of parallel programs and identifies potential performance bottlenecks concerning communication and synchronization. The Scalasca performance analysis toolset searches for wait states in executions of large-scale MPI applications. It measures the temporal displacement between matching communication and synchronization operations that have been recorded in event traces. The main difference between our work and theirs is that they generate a trace file of the whole application which is being analyzed, whereas we only analyze the phases that represent the whole application, making the analysis faster. To overcome these handicaps, we use the PAS2P tool [5], which has only been used so far for performance prediction, and for our methodology we use the signature of the application extracted with the PAS2P tool. Table I shows the information from the PAS2P tool when analyzing the CG from NAS class E and 500 iterations: size of the trace files generated, the application execution time, the application execution time with the overhead generated by PAS2P, the percentage from the overhead of monitoring using PAS2P, time taken for PAS2P to make the analysis, the Signature Execution Time, the Predicted Execution Time, and the error of the prediction.

The overhead of monitoring using PAS2P is shown in Table I and it will be added only in the first stage if the signature of the application is not available. It is noteworthy that this methodology is oriented to be used in scientific parallel applications which run thousands of times. So, the overhead added will be compensated by the speedup that the application will obtain using the mapping generated by the methodology. Studies such as [9] provide a performance improvement in the MPI collectives by using MPICH2 implementation to deploy the collectives into point-to-point primitives. This implementation is a machine optimization specially developed for BlueGene Systems. Our proposal will take this into account in order to model the MPI collectives to point-to-point communications when we start testing applications with

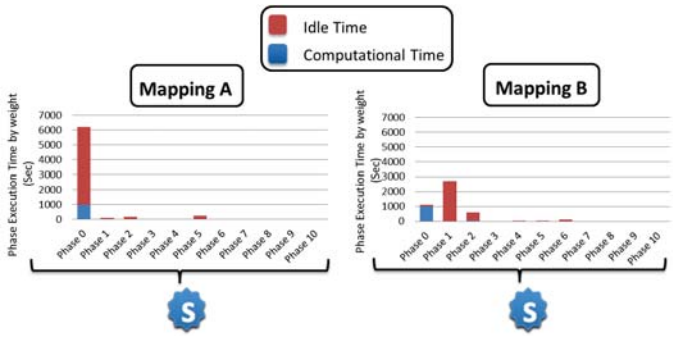

Fig. 2: Idle time's mappings analysis by phases

a high use of collectives. Predictable performance is important for understanding application performance issues [10]. To carry out a study about the interference in performance of sharing the same network links, we plan to take future action on this interference by approaching highly communicating processes and reducing idle times, which are created by this possible interference. There are several techniques and algorithms that efficiently address the issue of mapping the application, considering its communication pattern within the physical topology [11]. Using such a strategy enables us to improve the overall application execution time significantly. One of the studies addressing the issue of mapping the application is an optimization algorithm, which strives to minimize two metrics: dilation and congestion [11]. Where dilation allows a comparison of the number of times packets are transmitted over network interfaces, and congestion counts how many communication pairs use a certain link. Mapping A to H arbitrarily, minimizing either the dilation or congestion, is an NP-hard problem. Among the various factors influencing the performance, process placement plays a key role as it impacts the communication time of the application. Different techniques are shown in [11], and other algorithms and tools to perform a topology-aware process placement are presented in [12]–[14]. In all cases, the problem consists of matching what the communication pattern of the application represents to the physical topology of the architecture. In very large problems such as HPC applications, which feature hundreds of thousands of processes, mapping these processes onto the architecture requires a huge computing power [11], whilst reducing the communication load has a huge impact on energy consumption; a large part of the energy spent on executing an application is due to data movement. However, there are lack of studies about the real gain of topology-aware mapping and energy savings [11]. Our methodology differs from the studies presented in [11]–[14] because it faces the issue of topology-aware process placement, which takes a lot of time since the whole application has to be executed. We carried out our characterization using the signature. The time needed to execute the signature represents 10% of the whole application execution. Taking this into account, we made the analysis and models to obtain a better mapping for the characterization of the phases. The time spent on obtaining

TABLE I: PAS2P times overhead for NAS CG class E and 500 iterations

App. np (TFS)	AET (sec)	AET + PAS2P (sec)	PAS2P Overhead (%)	PAS2P analysis Time (sec)	SET (sec)	Predicted Time (sec)	Prediction Error (%)
128(3.4GB)	9131.76	9147.99	0.18	33.66	395.40	9166.92	0.21
256(6.8GB)	4853.63	4856.38	0.06	46.92	208.85	4847.63	0.18
512(17GB)	1860.61	1884.28	1.27	88.64	92.07	1846.32	2.01
1024(34GB)	1263.85	1300.64	2.91	128.62	70.81	1303.43	0.22
2048(80GB)	677.93	746.94	10.18	135.11	69.77	765.16	2.44
4096(161GB)	621.07	649.01	4.50	136.26	5.45	634.45	2.24

App. np: Application number of processes used and Trace file size
AET: Application Execution Time
AET + PAS2P: AET plus the time of the analysis of PAS2P
SET: Signature Execution time

the new mapping policies is onerous compared to the total execution time of the application. The problem we want to tackle is how to improve the efficiency of the applications without modifying the source code. One factor we want to change, which affects the execution time of the application with the same number of resources, is to modify the mapping (affinity) of the application processes in the machine. So far there have been studies such as [15]–[17] which modify the mapping of application processes in the machine, whether modifying the source code or executing all the application, in order to obtain a new mapping policy.

III. Proposed Methodology

In the execution of parallel applications which run in HPC environments, there are factors and system resources which affect the performance of the application and that can be modified without changing the application code. One of these factors is how the application processes (mapping) are distributed in a specific machine. We propose to modify the global mapping, bearing in mind that the modification of mapping strategies affects the application execution time, which can be reduced or augmented, due to the decrease or increase in the congestion pattern in the interconnection network. If highly communicating processes are incorrectly placed, the execution time will increase.

This paper focuses on the clustering technique used to create the new mapping based on the communication pattern, where we make use of an interdependence communication matrix ($ICommMatrix$) for each phase or a General $ICommMatrix$ for all phases. The selection of the $ICommMatrix$ will depend on the interaction of one phase with others (the execution sequence of phases). The $ICommMatrix$ is calculated after executing the phases from the signature. As the quantity of data needed to generate the $ICommMatrix$ is given by the phases, this matrix is generated in a bounded time. Once we have the $ICommMatrix$, we use it as an input for the clustering algorithm to create groups (clusters) of processes that have high communication interdependencies, in order to reduce the communication traffic between the compute nodes. All these modules are included in a short pre-execution before the application execution is carried out, in order to generate a better placement of the application processes in the target machine.

TABLE II: Characteristics of Phase 3 in process 0.

PHASE ID: 3		Weight: 14014	PhaseET: 0.1092 Secs.		
MPI Call	Dest. (MPI Rank)	Size of Buffer (count)	Size of Message (Bytes)	Comp. Time (nanosec)	Number of Instructions
MPI_Isend	1	20280	162240	308639	609
MPI_Irecv	15	20280	162240	197748	518
MPI_Wait	15	20280	162240	17491119	35780117

The methodology will be described in two sections, the first section discusses the characterization of the communication pattern; the second section discusses the clustering algorithm used, which is based on the communications pattern behavior.

A. Efficiency Evaluation Using the Signature

For the purposes of this study, inefficiencies will be defined as idle time from the cores of each processor [18], which may be time spent waiting for communications messages. Studies such as [19] show that one of the main reasons for this idle time is the delay from process communication due to

TABLE III: Idle time calculation for Phases of BT class D 256 processes and 1000 iteration and using the Gigabit Ethernet interconnection network

Phase	**W**	**Comm Time * W (s)**	**Comp Time * W (s)**	**PhaseET$_i$ * W (s)**	**Idle Time(s)**	**Phase_ MTTime P$_j$ * W**
1	14014	904.93	53.28	958.21	802.85	155.36
2	998	2846.64	275.31	3121.95	1791.39	1330.56
3	14014	1278.83	251.38	1530.22	664.60	865.61
4	14014	89.34	64.46	153.80	38.30	115.5

Signature Execution Time (SET): 54.25s
Predicted Execution Time (PET): 14323.37s
Phase_MTTimeP$_j$: Phase Minimum Transmission Time
CompTime: Computational Time
CommTime: Communication Time

imbalance. The waiting time for each process is propagated to other processes, and subsequently, delaying times increase. The total waiting time attributed to the delay in processes is defined as the sum of the waiting time it causes directly to any process and indirectly to other processes via propagation.

PAS2P makes the prediction of application execution time (PET) using Eq. 1, where $PhaseET_i$ is the execution time for each phase i ($PhaseET$) and W_i is the weight for each phase i. The execution of the signature in the target machine allows us to measure $PhaseET$. When PAS2P extrapolates $PhaseET$, using the weights, the PET is obtained.

$$PET = \sum_{i=1}^{n} PhaseET_i(W_i) \quad (1)$$

In this step we search for idle times in $PhaseET$. For each phase, we decompose the $PhaseET_i$ obtained from each MPI process by the signature execution. As shown in Eq. 2, we know that $PhaseET$ is composed of the addition of the Communicational Time ($CommTime$) and Computational Time ($CompTime$). We define $CommTime$ as the time spent on the transmission of the message plus the $Waiting\_Time$ defined as the idle time where the application processes are in an unnecessary waiting state, which is shown in Eq. 3.

Now that we have defined $CommTime$, we need to obtain the transmission time ($TransTime$) of each phase in order to clear and obtain the $WaitingTime$ variable value. To obtain $TransTime$ values, it is necessary to carry out a machine characterization on all kinds of interconnections between cores, sockets or compute nodes.

$$PhaseET_i = CommTime + CompTime \quad (2)$$

$$CommTime = TransTime + Waiting\_Time \quad (3)$$

Table II shows the communication characteristics that compose one phase for process 0. The communication characteristics are the MPI call, the size of the buffer for each message (Bytes) and the computational time (nanoseconds) between each MPI communication call. The table also shows the source for all types of MPI Receive calls (MPI Irecv, MPI Recv, etc.) and the destination for all types of MPI send calls (MPI Send, MPI Isend, etc.). Finally, the table also provides information about the weight of each phase and the PhaseET.

To obtain the $TransTime$ we developed a communication benchmark [18], which allows us to measure the transmission rate from the different links in an empty machine, and for a specific network topology. We considered this time as a Minimum Transmission Time (MTransTimes). We have divided the links into three groups. ByCore is where the message is sent between cores within the same processor chip socket. In the BySocket group, the message needs to travel from one processor chip socket to another but remaining in the same compute node. Finally, in the ByNode group, the message needs to travel between two compute nodes using the interconnection network.

From Eq. 2 and its computational time ($CompTime$), if we take the difference of ($PhaseET_i$) minus ($CompTime$), as shown in 3, we obtain the communication time ($CommTime$). It has to be mentioned that the communication time varies in different levels, so we have taken into consideration 3 levels of speed. The first of these is the communication between cores inside the same CPU socket (ByCore), the second one is the communication between cores in different sockets (BySocket), and the last one is the communication which has to use the interconnection network from computing nodes (ByNode). Finally, the $Phase\_MTTimeP_i$ plus $CompTime$ is the ideal execution time of a phase for a specific mapping. We will treat this time as a peak value. In Table III, the experimental validation is shown, where the difference between columns $Phase\_MTTimeP_i * weight$ and $PhaseET_i * weight$ can be seen.

B. Clustering based in the Phases' Communication Patterns

Once inefficiencies are detected in the application execution, using the default mapping, we proceed with the characterization of the phases' communication pattern, where we focus in the interdependency between application processes. The communication pattern per phases is used to characterize this interdependency and then it is given to a clustering technique, aiming to generate clusters of highly communicating processes.

The axis of the $ICommMatrix$ is given by the MPI processes, where each position i, j is the weight of the edge between nodes i and j. These edges are the communication relation of process i with process j and the weight increases according to the size of the message (Bytes) each time a communication is detected from process i to j. We use the $ICommMatrix$ as input for clustering techniques to generate the new mapping policy.

```
while there are elements in the ICommMatrix do
    Find Max Communication in Matrix;
    Store indexes;
    Chop matrix;
    Update dendrogram;
    Update indexes of real names;
    if node size == max processes per cluster then
        remove pair of p in matrix;
    end
end
return dendrogram;
```

Algorithm 1: clustering algorithm to group the MPI processes

Once we have the $ICommMatrix$, as shown in Fig. 3, we use a clustering algorithm to group the MPI processes following the Algorithm 1.

First, as an input of this algorithm, we transform the $ICommMatrix$ into a symmetric matrix, then we start the loop of the algorithm checking if the $ICommMatrix$ has

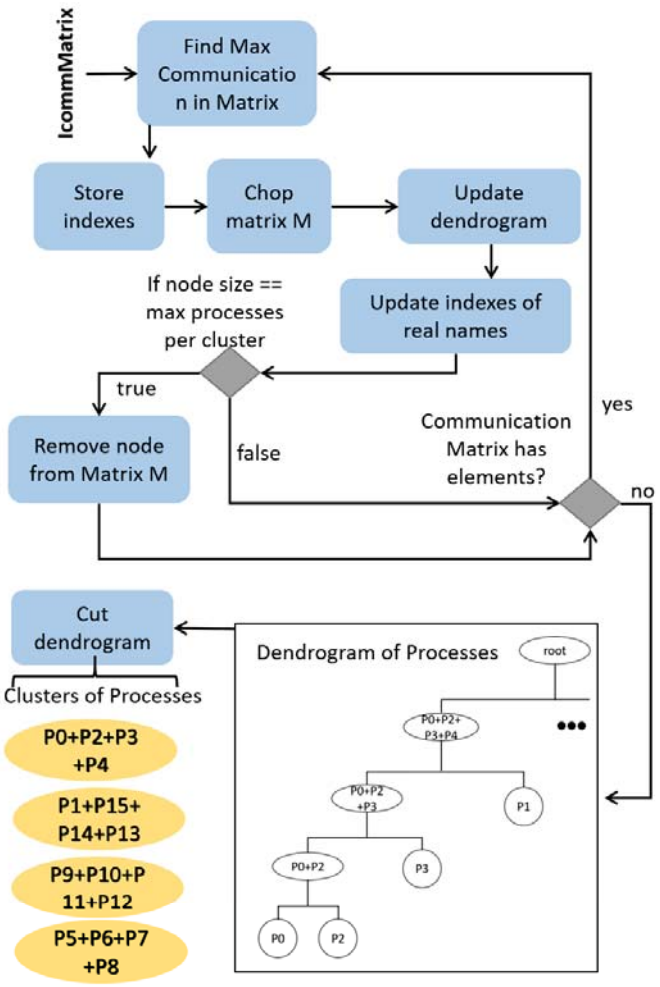

Fig. 3: Clustering of processes based on the communication pattern

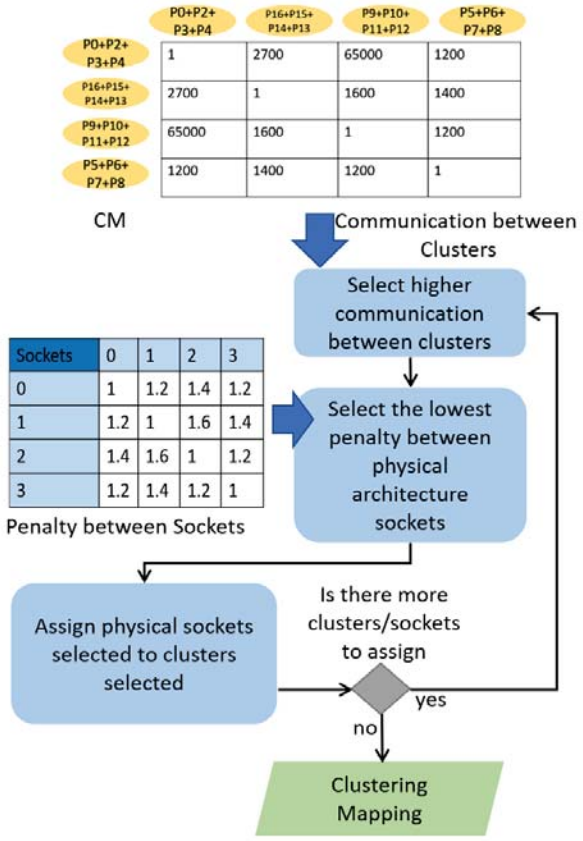

Fig. 4: Clustering of processes based in the communication pattern

elements. If there are elements, we find the maximum value of the $ICommMatrix$, if there is a tie we keep the first one found, then we store the indexes i, j of the maximum value.

Having the indexes, the matrix is reduced chopping index i both for column and row and chopping index j both for column and row as well.

Then we update the dendrogram (created in the first iteration of the loop), by adding the pair of processes i, j as a new node, and this new node has as child nodes processes i, j separated. It is worth noticing that when adding processes to a new node, an array called real names vector is used, in order to keep track of real names of processes as the matrix is chopped.

After each chop in the matrix, an update of the indexes in the real names vector is performed.

Every time a new node is created, if the node size is equal to the maximum value of processes per cluster, the node is removed from the $ICommMatrix$. It has to be mentioned that another matrix is being created here, keeping the information of clusters and distances between their further placements in the architecture.

Using the parameters of the machine characterization we cut the dendrogram in order to obtain as many clusters as the sockets that will be used.

The output of the clustering technique can be seen in Table IV from the section Experimental Validation. In this table, the clusters obtained for the BT application class E and 324 processes are shown. In the same Table IV, the clusters obtained for the CG class E for 128 processes are shown.

C. Mapping policy based on the clusters created and the machine architecture

The mapping policy is generated by using the clusters created and the data flow between them. This matrix is created every time each cluster reaches the cap of maximum processes per clusters. As shown in 4, we select the higher rate in this matrix, and keep those two clusters which has the highest data flow to place them in the fastest link.

We make use of distances between sockets given by hwloc [20], where we obtain a matrix with penalties of communicating from one socket to others.

We select the lowest penalty between physical sockets in the architecture using the matrix from hwloc, then we assign this pair of sockets to the pair of clusters selected with the highest communication flow. This is written in the affinity of processes file which is the Clustering mapping. Each time a

TABLE IV: Processes rank Distribution for the BT Class E and 324 Processes, showing the mapping for the first 2 compute nodes, and the rank distribution for the CG Class E 128 processes.

Node: Socket	BT Class E 324 ClusterId: MPI Ranks	CG Class E 128 processes ClusterId: MPI Ranks
N1:1	C1:{63-67},78,79	C1:{0-7}
	C2:{81-86},99,100	C2:{8-15}
N1:2	C3:{102-107},116,117	C3:{16-23}
	C4:{122-127},152,153	C4:{24-31}
N1:3	C5:{154-159},96,97	C5:{48-55}
	C6:{110-115},135,136	C6:{56-63}
N1:4	C7:131,148,149,150,151,166	C7:{96-103}
		C8:{104-111}
N2:1	C8:{167-172},185,186	C9:{112-119}
	C9:{187-192},204,205	C10:{120-127}
N2:2	C10:{206-211},217,218	C11:{32-39}
	C11:{219-224},236,237	C12:{40-47}
N2:3	C12:{239-244},254,2255	C13:{80-87}
	C13:{256-261},269,270	C14:{88-95}
N2:4	C14:272, 273, 274, 275, 257, 258	C15:{64-71}
		C16:{72-79}

pair of clusters is assigned to the pair of sockets, the matrices are reduced, deleting clusters/sockets selected from them.

When both matrices (communication between processes and the distances between sockets) are empty, due to the reduction when selecting clusters/sockets, the final clustering mapping is returned.

In the next section, we show the clusters of processes obtained for some NAS applications and the experimental evaluation using the default mapping policies and the mapping policy generated by our methodology.

IV. Experimental Validation

To validate our methodology, two traditional mapping policies were used. One of them is Round-Robin mapping, which is an algorithm used by process and network schedulers in computing. Generally, the processes are assigned to each compute node in equal portions and in circular order, thus handling all processes without priority. The second mapping policy used is non-binding mapping, which lets the operating system decide where and how to distribute the processes from the application. The third mapping used is the clustering mapping, which is obtained by using our methodology. The target machine used for the experiment is named CAPITA, which has 16 nodes with 64 cores AMD Opteron6262 HE for each node. The machine has 2 interconnection networks: an infiniband with a rate of 40 Gb/s for the first 8 nodes and a Gigabit Ethernet interconnection with 1 Gb/s for the other 8 nodes. Table V shows the results of applying the methodology using infiniband for CG and SP, and Gigabit-Ethernet for BT. This is performed to validate the methodology for both networks. We performed the experiments using the NAS Parallel Benchmarks that have been developed for performance

TABLE V: Speedups obtained for CG, BT and SP comparing the mapping policies

Mapp.	Network	PET (s)	SET (s)	S.U.	R%
CG class=E NP=128					
RR	Infiniband	30,158.25	406.47	1.00	-
NB	Infiniband	30,891.46	417.09	0.98	-2.43
C	Infiniband	25,752.65	355.88	1.17	14.61
BT class=C NP=256					
RR	GigabitEth	13,805.32	821.61	1.00	-
NB	GigabitEth	14,811.50	374.43	0.93	-7.29
C	GigabitEth	8,015.53	571.10	1.72	41.94
BT class=E NP=324					
RR	GigabitEth	89,451.48	1,272.37	1.00	-
NB	GigabitEth	43,238.21	623.75	2.07	51.66
C	GigabitEth	37,534.68	561.40	2.38	58.04
SP class=D NP=256					
RR	Infiniband	1,506.43	69.14	1.00	-
NB	Infiniband	1,475.82	69.01	1.02	2.03
C	Infiniband	1,056.64	41.36	1.43	29.86
LU class=E NP=256					
RR	Infiniband	5,609.25	306.94	1.00	-
NB	Infiniband	6,402.27	349.57	0.87	-14.14
C	Infiniband	5,543.39	305.27	1.01	1.17

evaluation of HPC [8]. Specifically, we used the Conjugate Gradient (CG) irregular memory access and communication class E for 128 processes and 100 iterations. We also used the Block Tri-diagonal solver (BT) class E for 324 processes and 100 iterations and class D for 256 processes and 1000 iterations. Finally, we performed another experiment using the Scalar Pentadiagonal (SP) class D for 256 processes and 1000 iterations. In the first step of our methodology, we performed the machine characterization for each of the applications. This characterization of the network transmission rate of cluster CAPITA is performed using the message size of the BT class C for 256 and class E 324 processes; the CG class E for 128 processes; and the SP class D for 256 processes. Once we have obtained the transmission characterization of the empty machine for each application, we have to calculate the idle times per phase (because each phase has its own communication pattern), which are shown in Table III, for the BT class D and 256 processes. This idle time can be reduced close to the value shown in the column of Phase_MTTimeP_j * weight. We performed the same procedure for the rest of the applications. Once the signature for each application has been executed and the characterization of communications has been performed, the $ICommMatrix$ for BT, CG, and SP are created. In each case, we created the General $ICommMatrix$ because the sequence of phases, from all the applications, is given by one phase behind the other. We generated a dendrogram for each of the applications, and then we used the clustering algorithm to create the clusters of MPI processes, which are high communicants. As a result of cutting the dendrogram, we obtained Table IV, in which we observe 14 clusters divided between the architecture of CAPITA cluster for the BT and 16 clusters for the CG. This table shows the first two nodes for the BT application. In Table V, the Prediction Execution Time (PET) is shown for each of the applications for Round-Robin, No Binding, and

Clustering mapping, as well as the signature execution time (SET). The SpeedUp reached up to $1.17x$ when using our methodology for the CG class E 128 processes and Infiniband as the interconnection network; $1.72x$ for the BT class C and 256 processes using a Gigabit Ethernet interconnection; $2.38x$ when using the BT class E and 324 processes and the same Gigabit Ethernet interconnection. Finally, the SP class D for 256 processes reached a SpeedUp of $1.43x$ when using the Infiniband interconnection network. What we have achieved are attractions of MPI processes by reducing congestion or number of ByNode communications, but we also know that by matching the mapping with the machine, processes' clusters with high communication density could cause collisions in the BySocket interconnection within the compute nodes. What we will try to achieve is to improve the matching algorithm to make a balance in the communications traffic, depending on the machine's link speeds. As seen in Table V, the mapping we produced shows significant improvements when using a Gigabit Ethernet as the interconnection network, rather than using a BySocket interconnection, because there is a great difference in orders of magnitude when comparing the speed rates. The results are compared to the default mapping, as it is the starting point. In addition, the default mapping is the one we want to change and reduce its communication inefficiencies.

V. Conclusion and Future Work

We propose a methodology that allows us to better select a mapping policy which produces a reduction in the application execution time, starting with an idle time analysis and using a clustering technique, which makes use of the communication pattern from the application and the machine characteristics. Our proposal was to improve the efficiency without needing to modify the source code, which we accomplished through the change of mapping policies, in a bounded time, without executing the whole application. We obtained better results as the amount of resources used increased because we generated mapping policies adapted to both the machine and to the application. In future work, the users will be given the option of choosing whether to go faster by sacrificing a percentage of efficiency, fixed by a lower efficiency value chosen by the same user; or just to be more efficient in the use of the system resources. In the current work, we characterize the communication pattern of the phases. In future work, we will identify other factors that may also affect the application's performance, as is suspected to be the case with cache misses in CPUs' shared memory. We will generate models based on the computational time analysis in order to take action on improving efficiency in parallel applications, considering the total runtime, energy cost, and the relationship between the runtime and the number of resources used.

VI. Acknowledgment

This research has been supported by the MICINN/MINECO Spain under contracts TIN2014-53172-P and TIN2017-84875-P.

References

[1] R. M. Badia, J. Labarta, J. Gimenez, and F. Escale, "Dimemas: Predicting mpi applications behavior in grid environments," in *Workshop on Grid Applications and Programming Tools (GGF8)*, vol. 86, 2003, pp. 52–62.

[2] M. Geimer, F. Wolf, B. J. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The scalasca performance toolset architecture," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 702–719, 2010.

[3] S. S. Shende and A. D. Malony, "The tau parallel performance system," *The International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[4] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "A tool for selecting the right target machine for parallel scientific applications," *Procedia Computer Science*, vol. 18, pp. 1824–1833, 2013.

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

[6] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[7] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[8] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The nas parallel benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[9] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng, "Optimization of mpi collective communication on bluegene/l systems," in *Proceedings of the 19th annual international conference on Supercomputing*. ACM, 2005, pp. 253–262.

[10] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: performance degradation due to nearby jobs," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 41.

[11] T. Hoefler, E. Jeannot, and G. Mercier, "An overview of process mapping techniques and algorithms in high-performance computing," 2014.

[12] C. D. Sudheer and A. Srinivasan, "Optimization of the hop-byte metric for effective topology aware mapping," in *High Performance Computing (HiPC), 2012 19th International Conference on*. IEEE, 2012, pp. 1–9.

[13] Q. Yin and T. Roscoe, "Vf2x: Fast, efficient virtual network mapping for real testbed workloads," in *International Conference on Testbeds and Research Infrastructures*. Springer, 2012, pp. 271–286.

[14] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp, "Multi-core and network aware mpi topology functions," in *European MPI Users' Group Meeting*. Springer, 2011, pp. 50–60.

[15] T. Hoefler, E. Jeannot, and G. Mercier, "An overview of process mapping techniques and algorithms in high-performance computing," 2014.

[16] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn, "Mpipp: an automatic profile-guided parallel process placement toolset for smp clusters and multiclusters," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM, 2006, pp. 353–360.

[17] E. Jeannot, G. Mercier, and F. Tessier, "Process placement in multicore clusters: Algorithmic issues and practical techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 4, pp. 993–1002, 2014.

[18] C. Rangel, A. Wong, D. Rexachs, and E. Luque, "Using the application signature to detect inefficiencies generated by mapping policies in parallel applications," pp. 1–12, 2017.

[19] D. Bohme, M. Geimer, F. Wolf, and L. Arnold, "Identifying the root causes of wait states in large-scale parallel applications," in *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE, 2010, pp. 90–100.

[20] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in hpc applications," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Feb 2010, pp. 180–186.