An approach for an efficient execution of SPMD applications on Multi-core environments.

Ronal Muresano[a], Hugo Meyer[a,b], Dolores Rexachs[a], Emilio Luque[a]

[a]*Computer Architecture and Operating System Department (CAOS),*
University Autonoma of Barcelona (UAB), Barcelona, Spain
rmuresano@caos.uab.es, hugo.meyer@caos.uab.es, dolores.rexachs@uab.es, emilio.luque@uab.es
[b]*Computer Sciences Group,*
Barcelona Supercomputing Center (BSC-CNS), Barcelona, Spain.
hugo.meyer@bsc.es

Abstract

Executing traditional Message Passing Interface (MPI) applications on multi-core cluster balancing speed and computational efficiency is a difficult task that parallel programmers have to deal with. For this reason, communications on multi-core clusters ought to be handled carefully in order to improve performance metrics such as efficiency, speedup, execution time and scalability. In this paper we focus our attention on SPMD (Single Program Multiple Data) applications with high communication volume and synchronicity and also following characteristics such as: static, local and regular. This work proposes a method for SPMD applications, which is focused on managing the communication heterogeneity (different cache level, RAM memory, network, etc.) on homogenous multi-core computing platform in order to improve the application efficiency. In this sense, the main objective of this work is to find analytically the ideal number of cores necessary that allows us to obtain the maximum speedup, while the computational efficiency is maintained over a defined threshold (strong scalability). This method also allows us to determine how the problem size must be increased in order to maintain an execution time constant while the number of cores are expanded (weak scalability) considering the tradeoff between speed and efficiency. This methodology has been tested with different benchmarks and applications and we achieved an average improvement around 30.35% of efficiency in applications tested using different problems sizes and multi-core clusters. In addition, results show that maximum speedup with a defined efficiency is located close to the values calculated with our analytical model with an error rate lower than 5% for the applications tested.

Keywords:
Performance Improvements, Multi-core, Mapping, Scheduling, Scalability analysis, Tiling applications, SPMD.

---

1. Introduction

The increasing use of multi-core processors in High Performance Computing (HPC) is evident in the top500 [1], in which most of today's clusters are set up with multi-core architecture. However, the increase in complexity and the hierarchical communication architecture present on these multi-core clusters create significant programming challenges which have to be managed carefully if programers wish to harness the inclusion of more parallelisms inside the nodes [1], [2]. The parallel programmers have to deal with some architectural characteristics, such as: number of cores per chip, shared cache between cores, bus interconnection, memory bandwidth, communication congestion, etc. [3]. All these elements are becoming more important for programmer to consider, in case that application's scalability and efficiency want to be improved [4].

The multi-core nodes integrate a homogeneous computation architecture, which in some cases are composed of 2, 4, 6, 8, etc., cores by chip processors. However, a node can include several chip processors creating a small high speed parallel machine inside the node. Nevertheless, these nodes have to be analyzed as heterogeneous when we are

[1]TOP500: a list which provides a rank of powerful parallel machines for HPC on the world, www.top500.org

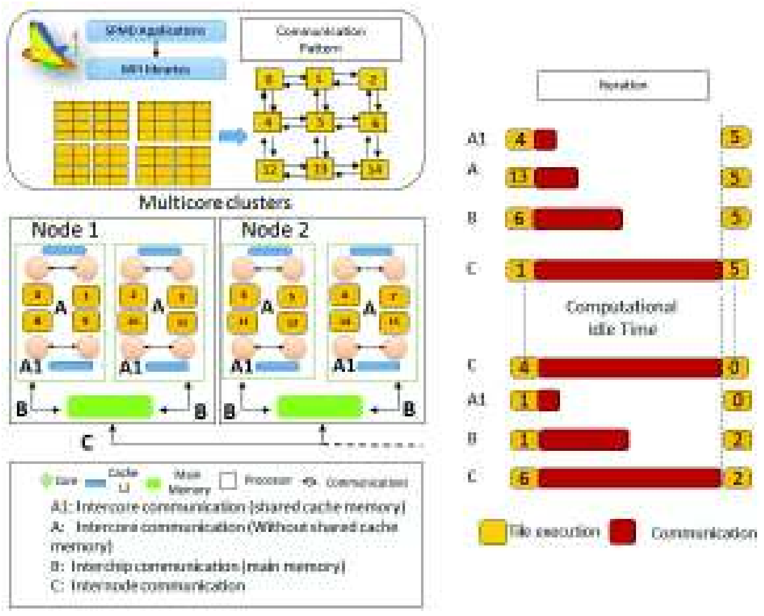


Figure 1: Mapping and execution of SPMD applications on a Multi-core cluster

working with applications that have a very high communication frequency between parallel processes. The communications between cores in these architectures use a hierarchical communication architecture that uses different paths and speeds to perform the communication processes inside the node [5], [6], [7]. For example, the parallel processes in a multi-core cluster can communicate using the cache memory or main memory for communications inside the node (Intercore and Interchip communications), or using the local area network to perform the communication with another process located in another node of the cluster (Internode communication). This communication architecture can create unbalanced issues that seriously affect the application performance, especially those applications which have a very coupled behavior (Figure 1).

Performance metrics that are commonly used to measure, such as: execution time, speedup, computational efficiency and strong and weak application scalability are all seriously affected. All these metrics are influenced in different ways due to the degradations and load balancing problems generated by the communications links [8]. Another important aspect to consider is that many MPI applications have been designed without considering the computational architecture characteristics. An example is the monocore nodes, where the communication processes were homogeneous and most of them have to be updated in order to take advantages of multi-core architecture.

A parallel paradigm which is seriously affected when executed on a hierarchical communication architecture is the SPMD (Single Program Multiple Data). This paradigm is focused on executing the same program in all processing elements but using different sets of tiles [9], [10]. However, many SPMD applications share data between parallel processes and their communications can be a very big problem, especially when we have applications very well coupled, such as: application of finite differences, fluid dynamics, weather models, econometrics models, etc., all of which have to communicate tiles between MPI processes in each iteration. Hence, a SPMD tile is computed in a similar time due to the homogeneity of the core. However, the communication processes among neighbors are performed using different communications links depending on the location of the SPMD processes on the multi-core clusters. These behaviors may cause serious delays in tile synchronization when these applications are executed on multi-core clusters.

An example of this problem is illustrated in Figure 1. The example shows us an SPMD application where each tile communicates with four tiles. As we can observe, this application needs to repeat a set of iterations but the iteration $i + 1$ depends on the results obtained in the iteration i. In this sense, the tiles are divided and assigned to each core to start the computation. Then, the computation processes have to wait until the slowest communications link finishes receiving its information to start the new iteration. These delays are due to the tile dependencies on the code. In some cases, the communication speed between MPI processes can vary in an order of magnitude for the same data packages depending on the link.

To solve these inefficiencies, we have developed a methodology which includes an analytical method that allows us to manage the communication latencies using some characteristics of each SPMD application over the parallel machine (e.g. computation and communication tile ratio). This method permits us to determine a relationship between scalability and efficiency. The objectives of this method are addressed in two analytically directions. The first one is to find the ideal number of cores needed to obtain the maximum speedup with a certain level of efficiency defined by the programmer (maximum strong scalability point). The second one is to determine how the application problem has to be increased in order to maintain an execution time constant while the number of cores are expanded (weak scalability), using similarly a defined level of computational efficiency.

An approach of our method for solving the inefficiency issues is the definition of a Supertile (ST). A ST is a grouping of tiles divided into two types: internal tiles (IT), in which communication processes are made in the same core, and edge tiles (ET), where communication processes are performed with tiles allocated to other core. This division allows us to hide the communication effects. This is done using an overlapping strategy in which the internal tiles computation is overlapped while the edge tiles communications are beging executed. This approach permits us to mask the inefficiency with more computational execution. However, defining a correct size of the ST that makes a tradeoff between efficiency and speedup on multi-core will be solved using an analytical calculation, where we will obtain the ideal number of cores to execute and the size of the ST that covers all the inefficiency generated by communication links.

All these groupings and overlapping processes are summarized in a methodology, which is composed of four phases: the characterization (which analyzes the application and environment), the tile distribution model (which determines analytically the number of tiles (ST size) and number of cores that accomplish our objective), the mapping strategy (which defines the distributions of tiles (ST assignation) over cores), and the scheduling policy (which defines the execution order of assigned ST). The success of this methodology is due to the fit between the multi-core environment and the deterministic behavior of the SPMD applications.

This method has been tested with different benchmarks and applications over different multi-core clusters composed of between 16 to 4096 cores and the results show an improvement average of around 30.35% in efficiency over applications tested with an error rate lower than 5%. The experimental evaluation makes it clear that in order to achieve a tradeoff between speedup and efficiency on SPMD applications, we have to manage the communication heterogeneities of the multi-core clusters.

This paper is organized as follows. A discussion of the related work is defined in section 2. Section 3 explains how an SPMD application behaves on multi-core cluster. Section 4 describes the methodology for efficient execution of SPMD applications on multi-core clusters. Next, section 5 shows the experimental validation, where the results obtained by testing with different applications on different multi-core clusters and the improvements obtained in the performance is illustrated. Finally, section 6 summarizes the main contributions.

2. Related Works

Improving the performance of parallel applications is a challenge. For this reason, there are different approaches that help us to understand how the multi-core environments behave when we are executing parallel applications. An approach has been developed by Kayi [11], which evaluated the memory latency for a set of multi-core architecture and how these latencies affect the parallel applications. The latency analysis is very important because memory hierarchy is part of the communication paths used by MPI, when SPMD applications are running on multi-core. As mentioned before, the inclusion of multi-core nodes has brought more parallelism within the node. In this sense, this parallelism has permitted developing a set of techniques in order to take advantage of the computational power of these multi-core clusters. A particular method has been developed by Liedbrock et al [12], where he defined a method to create a performance model for hybrid SPMD applications (MPI/OpenMP). The main focus of this work was to improve three main performance metrics: adaptability, fidelity and scalability. To achieve his objective, he applied mapping and scheduling strategies based on the multi-core characterization. Similarly, our approach has been designed using characteristics of the application and environment in order to apply the mapping and scheduling strategy. The difference of our approach is based on hiding the communication effects of the MPI-SPMD applications using the overlapping technique, which allows us to improve the performance metrics.

Another methodology that obtains a considerable improvement in the performance on multi-core cluster was presented by Vikram [13]. The main goal of this work was focused on developing a strategy that permits programmers to map tasks into reconfigurable architectures. This work uses a mapping strategy which allows us to execute and to obtain the maximum speedup under a specific configuration. However, our approach attempts to obtain the maximum speedup but by using a defined efficiency constraint. This scenario permits us to achieve an efficient and faster execution on the hierarchical communication architecture.

However, designing an efficient methodology in communication heterogeneous environments is a challenge and this has been demonstrated by Bearmount et al [14]. This work was oriented towards finding the best allocation of processes in communication heterogeneous environments. It centered on designing a method to minimize the communication overhead and it determined that obtaining efficient strategies for data allocations is an NP completeness problem when heterogeneous networks are used. In this sense, there are studies that evaluate the communications issues on multi-core clusters with the aim of improving them [15], [16]. These researches have proposed strategies to manage the communication issues presented on multi-core clusters using a multi thread implementation but the main idea of our approach is to manage the inefficiencies generated by MPI messages using two threads, one for computation and the second for performing the communication. These threads will allow us to apply an overlapping strategy and minimize the communication effects of SPMD applications on multi-core clusters [8].

In order to achieve performance improvements on multi-core clusters, our method integrates a mapping and a scheduling strategy. In this sense, we have analyzed different studies focused on distributing tiles between MPI processes. Some of them were based on improving the speedup of the parallel applications [17] [18] and others that are focused on enhancing the computational efficiency on multi-core clusters [19], [20]. However, we are including in our method a mapping technique that can combine both performance metrics (speedup and efficiency). This mapping allows us to manage the communications effects properly, considering the different communication links present on multi-cores. It will assign the amount of tiles necessary (grouped in a ST) to each core with the aim of applying an overlapping strategy to minimize the communication effects.

On the other hand, there are different scheduling strategies defined for SPMD applications. Some of them have been proposed to be used on mono-core architecture where there are not any internal communications [21],[22] and others scheduling techniques were focused on multi-core architecture [23], [24], [25]. The main idea was to find a suitable scheduling policy, which integrates parallel applications designed using pure MPI and multi-core architecture. To design our scheduling technique, we analyzed and evaluated the model defined by Panshenskov et al [23] and we chose some characteristics of this work, such as: tiles division in blocks, asynchronous communications, computation and communication overlapping. Additional to these characteristics, our scheduling process permits us to make differences between tiles using a priority assignment execution order. This process is carried out with the aim of executing the edge tiles first and then overlapping the internal tile with the edge tile communications. These characteristics, together with the tile distribution model and the mapping strategy designed, allow us to obtain a minimization in the communication overhead. The combination between the characterization, analytical model (tile distribution calculation), mapping and scheduling techniques and using overlapping strategy allows us to create a novel method which improves the efficiency, speedup and scalability of the SPMD applications on multi-core clusters.

3. Execution of SPMD applications on Multi-core clusters.

As was mentioned, the hierarchical communication architecture present on multi-core clusters can create imbalance issues between processes. In this sense, this research considers the parallel applications, which are designed to use the MPI library for communications and the SPMD as a parallel paradigm. This paradigm is one of the most used in HPC. The characteristics of SPMD applications can vary depending on the purpose of the application. In this sense, this research studied SPMD applications considering the tile dependencies synchronicity, where tiles have a specific communication pattern, which does not depend on the MPI process assignation itself. We perform a similar approach to the bulk synchronous parallel (BSP) method which was established by Valiant [26], [27]. However, we have excluded the barrier synchronizations of the BSP model which create a MPI processes synchronization. This has been excluded with the aim of minimizing the MPI communication imbalances.

Additionally, the SPMD applications used to apply the method have to accomplish the following characteristics:

- Static: Parallel application defines the communication process at the beginning of the execution and it is maintained during all the execution.
- Local: Applications that do not have collective communications
- Regular: Communications are repeated for several iterations.
- Multidimensional: Applications with N dimensions.

In addition, SPMD applications present a specific communication pattern, which can be integrated by two, four, six, or more bidirectional communications depending on the application purpose. These communication patterns are established at the beginning of the SPMD application execution and are kept until the end of the application. For this reason, we have to manage the communication imbalance of the multi-core architecture because it can create dramatic inefficiencies in the parallel execution that will be repeated throughout the application's iterations.

For example, there are a set of benchmarks and applications of different fields that accomplish these characteristics defined before. An example of these benchmarks is the NAS parallel suite in some of its programs (e.g BT, MT, SP) [28]. Moreover, there are applications where the heat transfer application, Laplace equation, wave equation model, application of fluid dynamics such as mplab to solve Lattize Bolztman problems, etc. may be mentioned. All these applications are seriously affected by the imbalance generated by different communication paths.

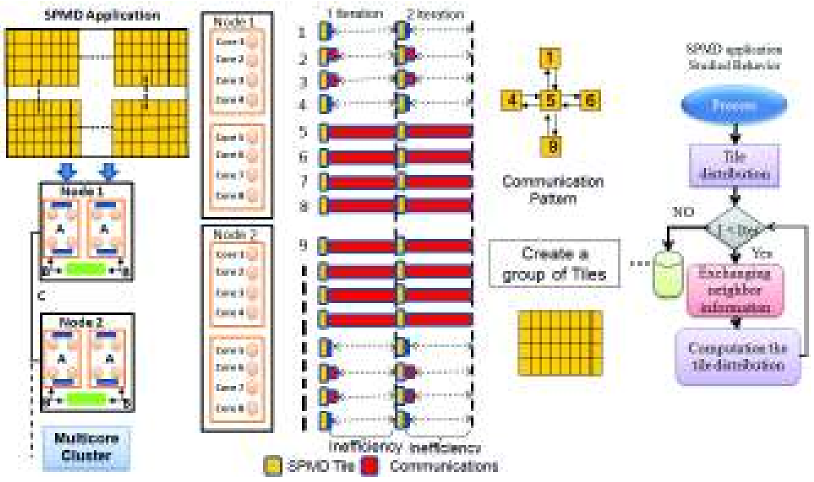


Figure 2: SPMD application behavior on multi-core cluster

An example of this inefficiency is illustrated in Figure 2, where SPMD tiles are executed in a similar time due to the homogeneity of the cores, but the communication processes are done using different communication links and each link has its own speed and bandwidth. These waiting times are translated into inefficiency for the application and these inefficiencies seriously affect the performance. Hence, the idea is to create a group of tile respecting the communication patterns and then it is to execute them into the multi-core cluster. For example, Figure 2 illustrates the behavior of a SPMD application which will be repeated a number of iterations and during this repetitive process we have to exchange information between tiles (most of them located in different cores) and then to compute the tile distribution which is the set of tiles assigned to a specific core. In our case, we measured the efficiency as $T1/(P*Tp)$, where $T1$ is the execution time on one processor and Tp is the execution time on p processors.

3.1. Efficiency and Speedup of SPMD applications on multi-core clusters

This work looks for suitable strategies in order to determine how the SPMD tiles have to be distributed and executed on the multi-core clusters. In this sense, we have proposed to develop a strategy using an overlapping technique, where the communication effects are hidden by the tile computation.

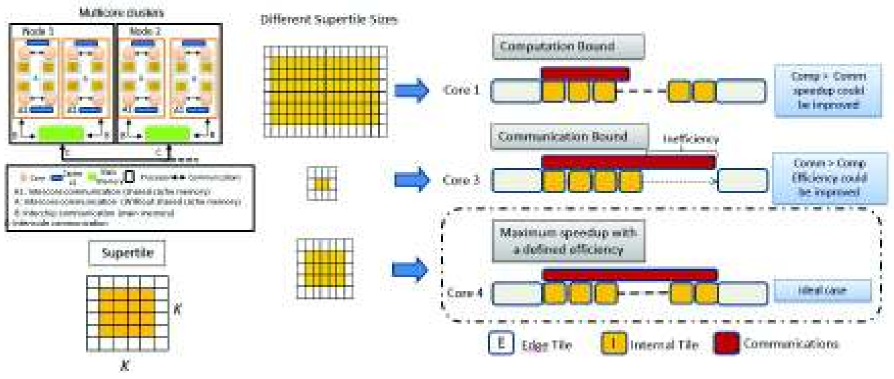


Figure 3: Supertile and relationship between efficiency and speedup

The ST is formed by two types of tiles; the internal and the edge. This division allows us to create an overlapping strategy as is shown in Figure 3, where edge tiles are computed first and then the internal tiles are executed together with the edge communications. However, we can present three possible scenarios (Figure 3). The first one assigns more internal tiles that the edge communication time required. In this case, we have an efficiency of around 100% but the speedup could be improved (computation bound scenario). The second scenario assigns fewer internal tiles than edge communication time required. In this case, the efficiency could be improved (communication bound). The last scenario is the ideal case which our method attempts to find, where the edge communications and the internal computation tile have similar performing time. In this sense, the speedup is increased because greater amounts of tiles are allocated to each core and the efficiency is enhanced due to the communication effects being hidden by the internal computation.

To find an ideal ST size, we have formulated an analytical problem, in which the ratio between computation and communication of a tile has to be found using all the communication levels integrated in the multi-core architecture tested. This relationship is found using a characterization of the machine and the environment. Then, a problem size defined by *MxM* of an SPMD application will be decomposed into *N* ST of size *KxK* tiles, where *K* represents the nth root of the ideal ST. This ST calculation will be explained in detail below with the analytical model calculation.

3.2. Efficiency and scalability of SPMD applications on multi-core clusters

Another important performance combination is related to the scalability and efficiency. However, the term scalability in HPC is divided in two different definitions; weak and strong scalability. Weak scalability is considered when the problem size and the number of *processing elements* [2] are increased with the objective of achieving a constant time-to-solution for larger problems and computational load per processor stays constant [29][30]. On the other hand, strong scalability is established when the problem size is fixed and the processing elements are increased. The goal of this scalability is to minimize the time solution [31]. Hence, scalability means that speedup is roughly proportional to the number of processing elements.

To apply both scalability concepts implies increasing the problem size (number of tiles per problem) or maintaining the problem size fixed. In both cases, if we wish to maintain a linear scalability, we should consider the ratio between computation and communication of tiles among different communication links presented on multi-core clusters. In this sense, the analytical model to calculate the ideal ST can be used in two directions with the aim of finding the maximum strong and weak scalability using a defined efficiency.

This combination of efficiency and scalability is performed by each kind of scalability (strong and weak). For example, the strong scalability is applied to determine the maximum number of cores needed to maintain the relationship between a defined efficiency and the maximum speedup. This optimal value allows programmers to determine the maximum number of cores that can be used for a specific problem size. On the other hand, the weak scalability

[2]For this work the processing elements are defined as the number of cores used to execute the SPMD application

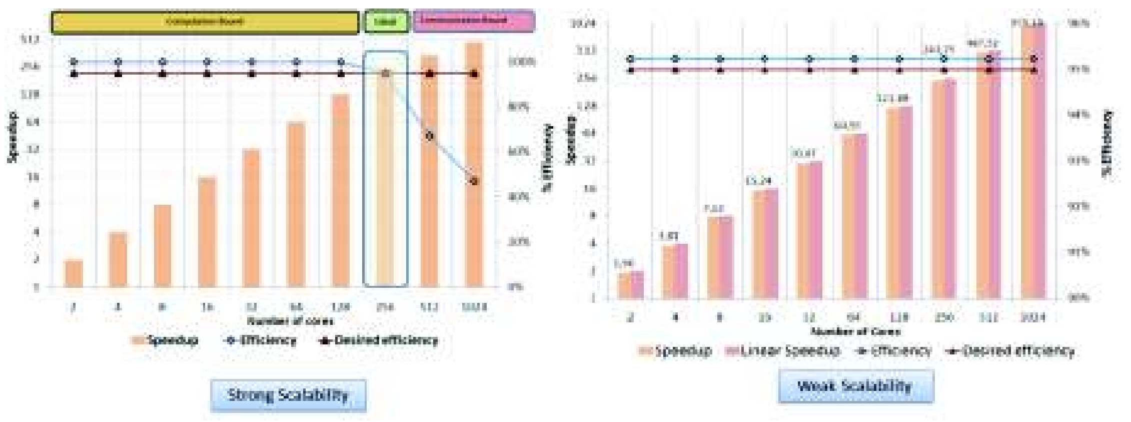


Figure 4: Strong and Weak scalability analysis using a defined threshold of efficiency

combination is performed with the aim of finding the ideal value of the ST in order to increase the application problem size using the ST size calculated as an input of the execution. In this sense, the problem size will be increased according to the number of computational resources available on the system.

An illustrative example of both scalability objectives can be detailed in Figure 4, where in the first graph (strong scalability) the behavior of the speedup is shown, when we are setting the problem size and we are increasing the computational resources. In this case, its can be detailed that speedup grows linearly, while the internal computation is bigger than the edge communication (computation bound). On the contrary, when the communication starts to be greater, the efficiency begins to drop considerably. The second part of Figure 4 illustrates the behavior of using a defined ST size on the execution time. When we start to increase the problem size in the same proportion of the ST calculated, the execution times remain constant. In this case, we can observe that efficiency value is constant and the speedup grows linearly.

4. Methodology for efficient execution of SPMD applications on multi-core clusters

This method has been developed with the aim of finding two main objectives. The first one is focused on achieving a suitable solution with the objective of obtaining the maximum speedup with a defined efficiency (strong scalability) and the second target is based on obtaining the ideal ST size in order to increase the application problem and the number of core but maintaining the execution time constant under a defined level of efficiency. In order to achieve a faster and efficient execution, the method has been divided in four phases (Figure 5).

A characterization phase, which is focused on performing an application and environment analysis with the aim of obtaining application performance parameters over a specific multi-core architecture (application and execution environment). These parameters obtained (computation of a tile, communication pattern, communication delays, etc) are inputs that will be used in the analytical model.

The second phase is integrated by an analytical model called the tile distribution model. Here, the ideal size of the ST and the number of cores needed to maintain the tradeoff between efficiency and speedup using the relationship between internal tile computation and edge tile communications are all calculated.

Next, the mapping phase is focused on assigning the ST calculated to each core, where we assume that more tiles can be added than core present on multi-core cluster. The assignation is made though an affinity process that allows us to minimize the communication effects through the slowest communication path. This mapping permits us to properly manage the workload imbalance caused by different communication links.

Finally, the scheduling phase has two functions, one of them is to assign tiles priorities with the aim of calculating first the edge tiles and then, the internal tiles. The second function is focused on controlling the overlapping process between internal tile computation and edge tile communication. At the end, these phases allow us to handle the latencies and the communication imbalances in order to improve the performance as was explained initially in [32].

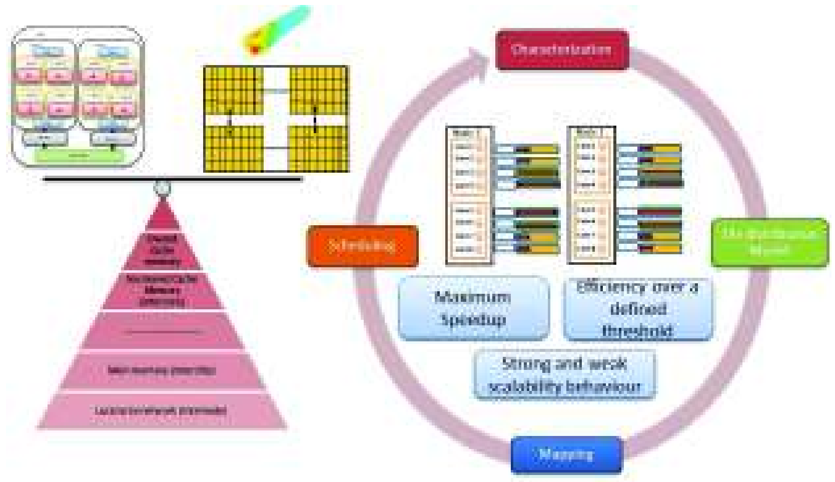

Characterization
Scheduling
Maximum Speedup
Efficiency over a defined threshold
Strong and weak scalability behaviour
Mapping

Figure 5: Methodology phases for obtaining an efficient execution of SPMD applications on Multi-core clusters.

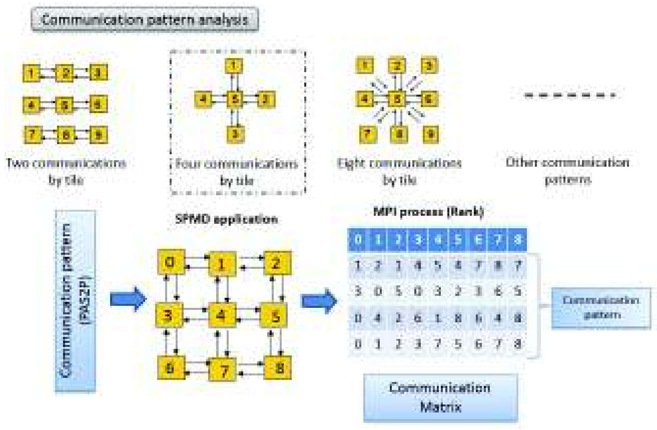
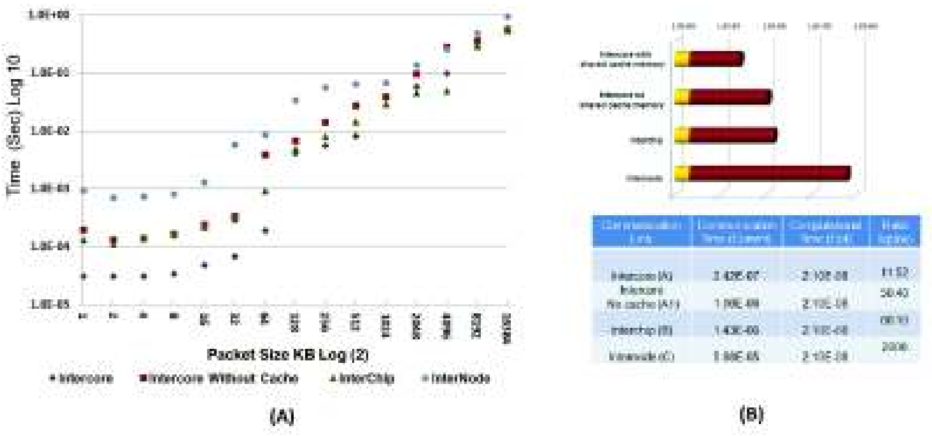

Communication pattern analysis
Two communications by tile
Four communications by tile
Eight communications by tile
Other communication patterns
Communication pattern (PAS2P)
SPMD application
MPI process (Rank)
Communication pattern
Communication Matrix

Figure 6: Communication pattern analysis with PAS2P.

4.1. Characterization phase

The main objective of this phase is to evaluate the tiles of a SPMD application into the multi-core environment. This evaluation finds the computational and communication time relationship of a tile in all the hierarchical communication architecture included on the multi-core cluster. These values will give us a ratio between computation and communication that will be used on our model. The characterization parameters are classified in three groups: the application parameters, parallel environment characteristics and the performance user needs. Then, we evaluate computation and communication behavior of SPMD application with the aim of obtaining the parameters with the nearest relationship between the machine and the application.

The application parameters offer the necessary information of the application characteristics, such as: problem size, number of tiles, iteration numbers, communication pattern, communication volume of a tile, data distribution. Moreover, these parameters allow us to determine, for example, if an SPMD communication pattern of a tile has been designed to communicate with two, three, four or more neighboring tiles. Moreover, an SPMD application can consider different distribution schemes, for example one-dimensional, two-dimensional block, column based and unconstrained. In addition, we have to identify which part of the code makes the iterative part. In some cases, SPMD applications are real kernels that are used in real applications and we can apply this method partially or totally depending on the application characteristics.



Communication Link	Communication Time (Comm)	Computational Time (Cpt)	Ratio (lambda)
Intercore (A)	2.42E-07	2.10E-08	11.52
Intercore No cache (A')	1.06E-06	2.10E-08	50.40
Interchip (B)	1.43E-06	2.10E-08	68.10
Internode (C)	5.88E-05	2.10E-08	2800

Figure 7: (A) Communication characterization and (B) Computation and communication ratio

Another important element is to find the communication pattern (source and destination of the package) and volume (size of the message sent). In this sense, we have used a tool called PAS2P (Parallel Application Signature for Performance Prediction), which allows us to know the behavior of the SPMD-MPI application making a signature of it [33] [34]. This tool gives us the communication pattern that is a key step in order to know the communication behavior of the application. The values obtained using PAS2P allow us to obtain information about the communication, which is saved into a communication matrix. This matrix has the the message order according to the MPI processes. An example is illustrated in Figure 6, where we can observe how an application with four communications patterns is analyzed.

The next analysis is related to environment characterization, which consists of evaluating the behavior of the SPMD application on a specific multi-core parallel machine. This process is done in two steps. Firstly, we analyze the hierarchical communication architecture using the pattern obtained with PAS2P in order to evaluate the communication time of a tile into each communication path. This analysis is performed using a tool that uses the same concept of the ping-pong communication tools, but in this case our tool takes into account the communication pattern in order to give us more precise results about the communication architecture (Figure 7). As can be evidenced in Figure 7 part A, the communication time in all the paths have different communication times for specific packet sizes, where in some cases we can obtain differences which are around one order of magnitude depending on the path used and the packet size sent.

This characterization is achieved using core affinity. For this reason and in order to respect communications order of the MPI processes in both steps: the characterization and the execution of the SPMD applications should use the same core affinity policy.

Then, we have to evaluate the computation time of a tile. The parameters obtained allow us to establish the communication and computational ratio time of a tile inside of the hierarchical communication architecture. These relationship values will be defined as $\lambda(p)(w)$, where p determines the link where the communication of one processing tile to another neighboring tile has been performed and w the direction of the communication pattern.

$$\lambda_{(p)(w)} = Commt_{(p)(w)}/Cpt \tag{1}$$

This ratio is calculated with equation 1, where $Commt_{(p)(w)}$ determines the time of communicating a tile for a specific link and Cpt is the time to compute one tile. An example is illustrated in Figure 7, where the different ratios have been evaluated using a double quad core cluster and a heat transfer application [3]. In this case, the multi-core architecture contains four communication levels and we can see the differences between each ratio.

Therefore, we have to calculate the ratio for each communication with the aim of finding the biggest computation and communication ratio, which will determine the slowest communication path. This characterization process has

[3]Bidimensional Heat Transfer application is an SPMD application with four communication neighbors

to be done in a controlled and monitored manner. The highest ratio value found will delimit an iteration of an SPMD application. Then, this value should be considered, when we will evaluate the analytical model to calculate the ST size and the ideal number of core necessary to meet the objectives stated.

Finally, the last set of parameters are defined by the programmer (performance requirements). In this sense, the users have to define the efficiency threshold which will be taken into consideration for the execution on the multi-core architecture. The efficiency is defined with the variable $effic$ which is an entry parameter to the model and it is considered as a constraint.

4.2. Tile distribution model

The objective of this phase is to find the ideal size of the ST which allows us to achieve the maximum speedup, while the efficiency is maintained over a defined efficiency threshold. To calculate this value, we have to start analyzing the behavior of an SPMD application using an overlapping strategy, which is summarized in equation 2. The first part of equation 2 is integrated by the edge computation time, which is represented using the $Edgecomp_{(i)}$ variable and then is executed together with the overlapping part between the internal computation ($IntComp_{(i)}$) and the edge tile communication time ($EdgeComm_{(i)}$). This process will be repeated for a set of iteration $iter$. The variables q and i are also defined, where q determines the number of specific iterations inside the SPMD application execution and the i establishes the number of a specific core inside the multi-core environment.

$$Tex_i = \sum_{q=1}^{iter}(EdgeComp_{(i)} + Max\left\{ \begin{array}{c} IntComp_{(i)} \\ Edgecomm_{(i)} \end{array} \right\}) \tag{2}$$

The values of $IntComp_{(i)}$, $EdgeComp_{(i)}$ and $EdgeComm_{(i)}$ are in function of the variable K as can be observed in equations 4, 5 and 6 respectively. This value of K represents the nth root of the optimal ST size and n determines the dimension of the problem. This value n ranges from 1 until 3, depending on the SPMD application.

$$ST = K^n \tag{3}$$

$$IntComp_{(i)} = (K-2)^n * Cpt \tag{4}$$

$$EdgeComp_{(i)} = (ST - (K-2)^n) * Cpt \tag{5}$$

$$Edgecomm_{(i)} = K^{n-1} * Max(Commt_{(p)(w)}) \tag{6}$$

To calculate the model, we have to find the maximum communications time $Max(Commt_{(p)(w)})$ using the maximum value of the $\lambda_{(p)(w)}$ ratio (Eq. 1) obtained in the characterization phase. This values defines the slowest communication path on the multi-core architecture. Once obtained, we can use this value on equation 6 with the aim of calculating the edge communication time. The sum of the computational time of the edge and the internal computation represent the total computational time of the region of K^n, which is assigned to each core. In this sense, it is important to state that the model is applied only for the communication exchanging part of the SPMD application.

In order to calculate the ST size, we need to determine the value of K. To obtain this value, we start from the overlapping strategy, between internal tile computation and the edge tile communication (Eq. 2). This part of the code is where the application lost efficiency due to the communications overhead, which we wish to improve. However, we can define a correct value of K, in which the edge communication times are completely overlapped. This can be done equalizing both equations' internal computation time (Eq. 4) and edge communication time (Eq. 6). However, the edge communication equation is in function of the communication time, while the internal computation is in function of the tile computation time. For this reason, we need to equalize both equation in function of Cpt in order to find a solution for K. This can be done using equation 1, where we can substitute the communication tile using $\lambda_{(p)(w)}$ multiplied by computational time Cpt of a tile. Then, both equations in function of Cpt are shown in the inequality 7.

$$K^{(n-1)} * Max(\lambda_{(p)(w)} * Cpt) >= ((K-2)^n * Cpt)/effic \tag{7}$$

However, in this inequality 7, we have to include the efficiency threshold, which is a constraint of the model. This inequality means that we can allow that $EdgeComm_{(i)}$has a bigger execution time than $IntComp_{(i)}$, but considering that these differences should be less than the maximum inefficiency allowed by the efficiency threshold. In this case, we have included a second restriction that defines this scenario (Eq. 8), where the efficiency value of the system must be around 100% due to the communication being overlapped by computation.

$$K^{(n-1)} * Max(\lambda_{(p)(w)} * Cpt) <= ((K-2)^n * Cpt) \tag{8}$$

Then, we have to calculate the value of K from inequality 7, which represents the conditions of our objectives. This is performed makingthis inequality 7 equal to zero, with the aim of obtaining an equation of first, second, third degree depending on the dimension of the SPMD applications (Eq. 9). Once we have solved the equation according to the degree of the application and we have obtained the value of K, we have to evaluate the possible solutions of K using the inequalities 7 and 8 with the objective of validating that the values found accomplish the constraints defined in the model.

$$((K-2)^n * Cpt) - (K^{(n-1)} * Max(\lambda_{(p)(w)} * Cpt)) = 0 \tag{9}$$

Until this point, we can calculate the ideal ST size using the equation 3. Subsequently, we need to know the ideal number of cores $Ncores$ that allows us to have maximum speedup under a defined efficiency. In this case, we divide the problem size defined as M^n and we divide between the K^n that represents the ideal size of the ST (Eq. 10). This number of cores determines the inflection point until the application has a strong scalability with a defined efficiency.

$$Ncores = M^n / K^n \tag{10}$$

In the case of weak scalability, we have to increase the problem size using the same proportion of the ST in order to maintain the execution time constant. This increment can be achieved using equation 10, where the problem size can be increased using the $Ncore$ variable and a fix K value (Eq. 11).

$$M = \sqrt[n]{Ncores} * K \tag{11}$$

$$K = \sqrt[n]{M^{(n)} / Ncores} \tag{12}$$

Finally, we can determine the theoretical behavior of the SPMD application for a lower number of cores that the optimal calculated with our model and predict its behavior. Equation 12 calculates the new values of K for a different number of core given with the objective of determining the execution time (Eq. 2) and calculating the speedup and efficiency for these values. This calculation is possible due to the deterministic behavior of SPMD applications.

4.3. Mapping phase

Once the distribution model has been analyzed, the next step is to map the ST into the multi-core architecture. For this reason, the ST assignations are made applying a core affinity, which allows us to allocate a set of tiles according to the policy of minimizing the communications latencies. This core affinity permits us to determine where the MPI processes have to be allocated and how the STs have to be assigned to each core. However, the ST assignations should maintain the initial allocation used in the characterization phase, as was explained before.

The mapping phase is divided in four key points (Figure 8). The first one determines the number of MPI processes using the number of cores obtained on the model. Next, a logical process distribution of these MPI processes, where this logical process distribution allows the application to identify the neighbor communications. This is done using a Cartesian topology of the processes that gives two coordinates in a grid distribution to each process. These two coordinates identify the cores, in which the processes have to be allocated. Moreover, we can coordinate the scheduling order with the objective of minimizing the saturation of the links. Figure 8 shows an example of distribution of MPI processes using an application with four communications patterns and two dimensions. Each ST is assigned to its respective core and we can detail that some ST may have a different number of border communications, depending on the allocation inside the logical distribution. We can also observe the communication times that depend on the location of the ST within the multi-core architecture.

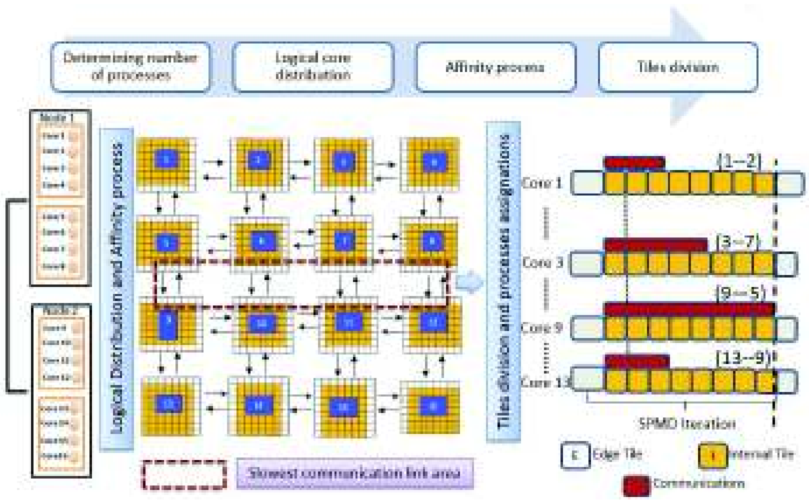

Determining number of processes
Logical core distribution
Affinity process
Tiles division
Node 1
Node 2
Logical Distribution and Affinity process
Tiles division and processes assignations
Core 1
Core 3
Core 9
Core 13
(1→2)
(3→7)
(9→5)
(13→9)
SPMD Iteration
Edge Tile
Internal Tile
Communications
Slowest communication link area

Figure 8: Mapping of SPMD applications on multi-core

The third function of the mapping phase is to apply the core affinity, where STs will be assigned to the execution cores. This process is performed with the aim of associating the logical distribution and the physical multi-core architecture. This process is beneficial in that SPMD applications are mapped in a better manner according to the machine characteristics, minimizing the communications performed using the slowest communication path.

The last part is the division and distribution of the STs into the architecture. Once the coordinates of the cores are obtained, the mapping has to divide the tiles in order to create the ST. This division is performed considering the value of K obtained by the analytical model. It is important to understand that an incorrect distribution of the tiles can generate different application behaviors, as was explained before.

4.4. Scheduling phase

This scheduling phase is responsible for determining the execution order and controlling the overlapping process. In order to achieve an efficient execution of SPMD applications, we have to determine the execution order of tiles respecting the conditions evaluated into the model. In this sense, we have to assign a execution priority to each tile that allows us to apply the overlapping strategy with the aim of minimizing the communication effects.

The execution priority assignment is a process which allows us to identify the execution tile inside the application. This process establishes the highest priorities for tiles which have communications through slower paths. These assignments have the following policies: tiles with external communications are selected with priority 1 and these edge tiles are saved in buffers with the aim of executing these tiles first (these buffers are updated in all iterations). The second assignation is made for internal tiles which are overlapped with the edge communications, these being assigned with priority 2. These internal tiles are executed while the edge communication are performed. An example can be detailed in Figure 9, where the distinction between internal and edge tiles using the priorities can be evidenced.

Then, we can apply the overlapping strategy, which allows us to hide the communication effects. In this sense, Figure 9 illustrates the overlapping strategy applied where the internal tiles computation and the edge tiles communication are performed at the same time. As can be detailed in the figure, the iteration of the SPMD application is delimited by the slowest communication links. However, the analytical model consider for the calculation the worst case. The assignation of a ST to each core allows us to balance all execution cores.

Although some cores can present different communication times, this does not mean that we have to calculate one value of K to each core. On the contrary, we have to determine only considers the worst case for the calculation. This case will limit the iteration of the application. The objective of STs is that all core finish approximately at the same time, allowing us to synchronize the execution. For the implementation of the overlapping process, we have used two threads, one of them is to perform the internal computation and the other is to manage the asynchronous

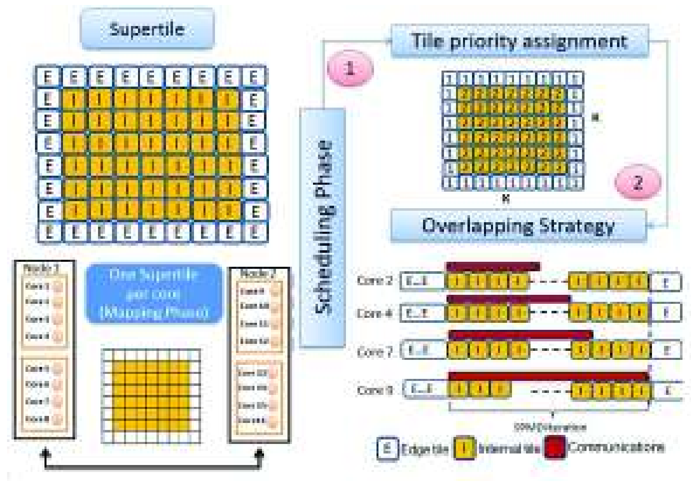


Figure 9: Scheduling of SPMD applications on multi-core

communications. These threads are opened once the edge computations have been computed and then are opened until they finish the iteration.

5. Experimental Validation

In order to evaluate the analytical model and the effectiveness of the methodology when we are executing SPMD applications on multi-core cluster, we validated using diverse multi-core environments and a set of parallel scientific applications designed using MPI. In this sense, our experiments were conducted on four different multi-core cluster configurations. A summary of these cluster's characteristics can be detailed in table 1. The first three were used to test the effectiveness of the method and the last Juropa [4] cluster has been used to evaluate the maximum weak and strong scalability under a defined efficiency.

Table 1: Multi-core Cluster Architecture

Cluster	Nodes	Cores/node	Processor Type	Memory	Network
Cluster_A	32	4	2xDual-Core 5150 Intel Xeon, 2.66GHz	4MB cache L2,12 GB RAM	GigaEthernet
Cluster_B	8	8	2xQuadCore E5430 Intel Xeon,2.66GHz	6MB shared L2 each 2 cores, 16GB RAM	GigaEthernet
Cluster_C	8	2	Dual Core Processor 3800+ 2GHz	512KB cache L2, 2 GB RAM	GigaEthernet
JUROPA	2208	8	2xQuad-Core X5570 Intel Xeon 2.93Ghz	8MB cache L2, 24 GB RAM	Infiniband

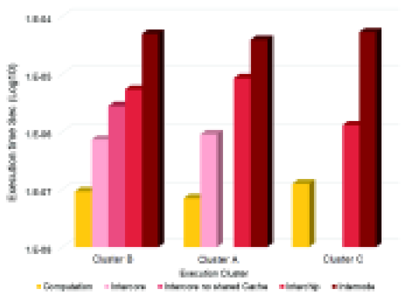
In addition, this methodology has been validated with a different set of benchmarks and applications and for this case the methodology has been validated using SPMD applications with different communication patterns. The applications used to validate the method were a Heat Transfer, Wave Equation, Laplace Equation for elliptical problems and two fluid dynamics applications, which are all integrated in the MPlab suite (LL-2D-STD-MPI and ZSC-2D-STD-MPI applications). The first three applications will be used to validate the methodology phases and the other two applications will be employed to verify the combination between scalability and efficiency. Moreover, all these applications accomplish the main characteristics defined before (local, regular and static) and they integrate a four bidirectional communication pattern.

5.1. Characterization phase evaluation

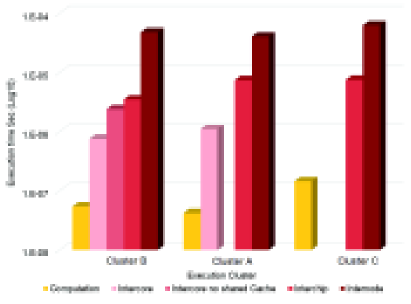
This characterization allows us to obtain real values of tiles executed on a specific multi-core architecture. In this sense, Figures 10(a) , 10(b) and 10(c) illustrate the application's analysis done in the characterization phase

[4]A large scale system located in the Julich Supercomputing Center, Germany, it was included on the top500 list position 10 in June of 2009

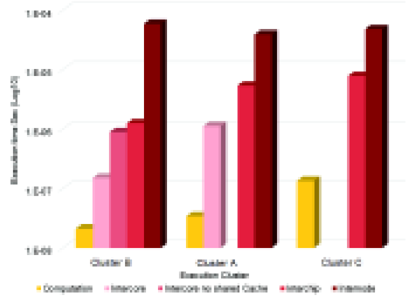
for three different applications which calculate the finite difference, heat transfer, Laplace and wave equation. The characterization is mainly focused on finding the ratio between computation and communication of a tile inside the multi-core architecture in order to apply the analytical model to determine the ideal size of the ST. Under this focus, figures 10(a), 10(b) and 10(c) show the behavior of a tile among the different communication paths and the tile computation in each cluster.



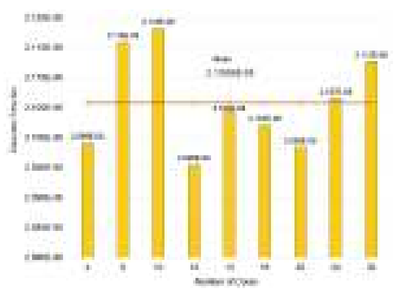
(a) Wave Equation Application



(b) Laplace Equation Application



(c) Heat Transfer Application



(d) Computation Analysis of Heat transfer app using Cluster_B

Figure 10: Computation and communication time of a tile on different multi-core clusters.

As we can observe, these times have variations depending on the clusters' composition and the application objective. These differences enable us to design strategies for allocating more tiles and permit us to eliminate the delays generated by tiles communication for improving the application efficiency. We can also find different cluster compositions that can present diverse communication architectures. For example, cluster_B has four communication links, while Cluster_C has 2 communication links. The number of paths depends on the multi-core architecture itself.

Once we have detected the communication paths and their delays, we have to find the execution time of a tile in the multi-core architecture. This value is found using a defined problem size but using different number of cores. Consequently, the number of tiles by core (ST) decrease when we are increasing the number of cores. This analysis is done in order to evaluate the impact of the cache effects on the execution time of a tile. As we can detailed in Figure 10(d), the computation time was measured for all applications using a defined problem of 9500x9500 on Cluster_B using from 2 until 32 cores. We can also observe that differences between tiles are very small with a standard deviation of around 8.5 E-11 Sec. This number is very low to affect the computation time and the precision of the model. For this reason, we have calculated the computational time using the mean of these executions and for this specific case we obtained a value for a tile of 2.105 E-8, which is the value used in the model for the calculation. It is important to remark that this characterization process is done only once, while the application has not been modified.

Table 2: Tile Distribution Model

Application	Problem Size	Desired $Effic$(%)	Cpt(Sec)	$CommT$(Sec)	λ	K	$Ncores$	Cluster
Laplace Equation	6250x6250	80%	$4.2E^{-8}$	$4.06E^{-5}$	966.6	777	64	Cluster_A
LL-2D-STD-MPI	1950x1950	95%	$2.41E^{-7}$	$6.07E^{-5}$	251.8	243	64	Cluster_A
Heat Transfer	9500x9500	85%	$2.1E^{-8}$	$5.85E^{-5}$	2785.7	2372	16	Cluster_B
Wave Equation	1100x1100	90%	$1.25E^{-7}$	$5.31E^{-5}$	424.8	386	8	Cluster_C

5.2. *Tile distribution model calculation*

A summary of the characterization process is shown in table 2, where the characterization values of computation (Cpt) and slowest communication (Commt) time of a tile, problem size, desired efficiency are illustrated. Then, we have to calculate the ideal size of the ST and the ideal number of core that allow us to achieve the maximum speedup with a defined efficiency [5]. In this sense, we can observe in table 2 the theoretical values obtained for both K and $NCores$, where the possible solution of K and Ncores represent the minimum value that maintain the efficiency over the defined threshold. This solution is evaluated in order to obtain the overlap grade between internal computation and edge communication time. Both times have to be as close as possible with the objective of obtaining an ideal. At this point we can calculate the theoretical computation time for the application and we can also start to understand how the application can grow in order to maintain the performance.

5.3. *Performance analysis: Efficiency and Speedup*

To develop our performance analysis, we executed SPMD applications but making a comparison between the theoretical value obtained with the model and the real executions for both applications with and without using our methodology. An example is illustrated in Figure 11(b), where we can observe the efficiency behavior of a heat transfer application executed with 100 iterations on Cluster_B.This Figure 11(b) illustrates a considerable improvement in terms of efficiency of around 39.72 % in the ideal point calculated. This improvement is calculated dividing the efficiency obtained using the efficiency valued obtained with the method between the application efficiency obtained without applying the method. This improvement is obtained when we use the ideal number of cores obtained by our model and applying the methodology (see table 2).

Another important aspect to highlight is the error rate. In this sense, the error rate is lower than 3% between the efficiency obtained and the theoretical efficiency defined. This value is achieved when we execute with the ideal number of cores calculated through the proposed model. Similarly, Figure 11(a) shows the behavior of efficiency and speedup of a benchmark of the mplab suite LL-2S-STD-MPI on the Cluster_A. In this case, the error rate is lower than 2% for the ideal point. Similarly, Figure 11(c) shows the behavior of a Wave application on Cluster_C. The number of cores that make a tradeoff between the efficiency and maximum speedup is the number of core calculated by the model with an error rate of 4.9%.

In addition, we can detail in Figures 11(a), 11(b) and 11(c) that performance deteriorates considerably after the point calculated by our model in the version using our method. In all Figure, we can also observe the behavior of the application using different number of core before and after of the ideal point of K. These points can be calculated using the equation 12, where we have to insert a specific number of cores and the problem size M. At the end, the model will return the K value for this specific scenario and then the execution time.

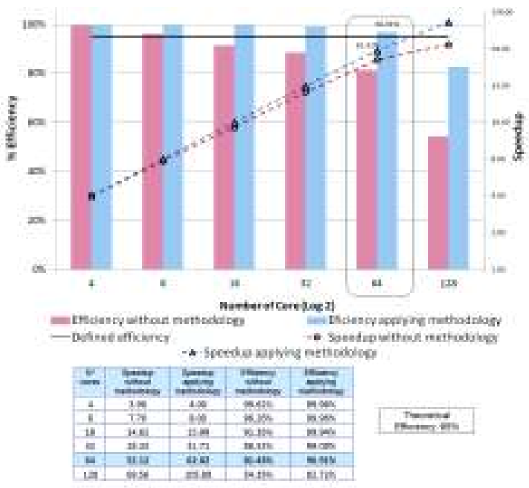
In this case, we can predict due to the fact that the application behaves as a computation bound behavior. However after the ideal point calculated with the model, the edge communication time begins to affect considerably. Consequently, we lose the determinist level inside the application due to the communication congestion. In this sense, performance after the ideal point will worsen and we cannot predict after that point. As we can observe in all cases, we have achieved our main objective of finding the maximum speedup while the efficiency is maintained over a defined efficiency threshold.

5.4. *Strong and weak scalability evaluation*

The results presented in this subsection are related to strong and weak scalability. In the first case (strong scalability), we maintain the problem size as fixed and we increase the number of cores. In this case, the speedup has

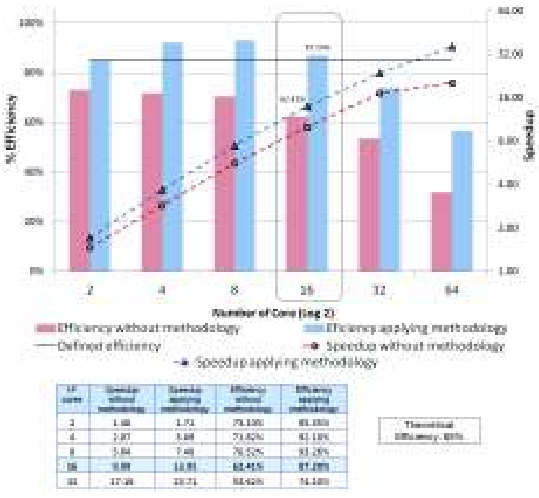
[5]The problem size used in table 2 were selected in order to have a integer value for the number of core and also considering the size of the multi-core architecture that we used to test the model

to grow roughly linear. For this test, we have used a big cluster called Juropa and two applications from the suite mplabs LL-2D-STD-MPI and ZSC-2D-STD-MPI. Both versions are two dimensional problems and they accomplish the characteristics defined as local, regular and static.



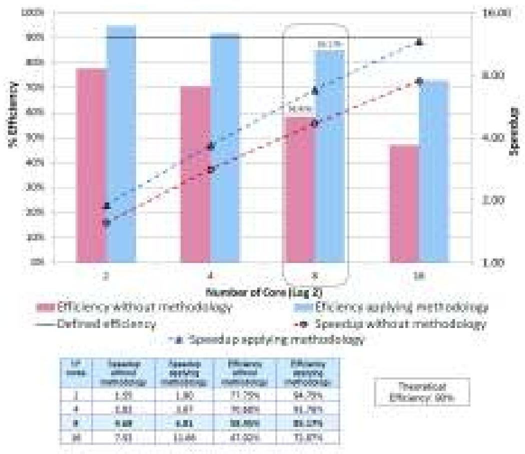
Nº cores	Speedup without methodology	Speedup applying methodology	Efficiency without methodology	Efficiency applying methodology
4	3.98	4.00	99.61%	99.98%
8	7.70	8.00	96.25%	99.98%
16	14.61	15.99	91.32%	99.94%
32	28.20	31.73	88.13%	99.08%
64	52.12	62.02	81.43%	96.91%
128	69.56	105.88	54.35%	82.72%

(a) LL-2D-STD-MPI on Cluster_A



Nº cores	Speedup without methodology	Speedup applying methodology	Efficiency without methodology	Efficiency applying methodology
2	1.46	1.71	73.10%	85.35%
4	2.87	3.69	71.82%	92.18%
8	5.64	7.46	70.52%	93.28%
16	9.89	13.91	61.81%	87.28%
32	17.18	23.71	53.62%	74.10%

(b) Heat Transfer Application on Cluster_B



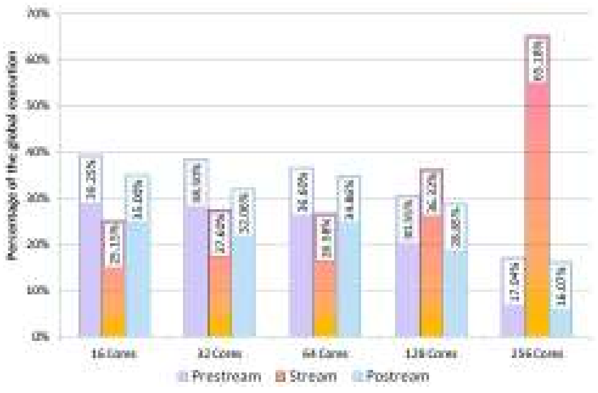
Nº cores	Speedup without methodology	Speedup applying methodology	Efficiency without methodology	Efficiency applying methodology
2	1.55	1.90	77.75%	94.75%
4	2.82	3.67	70.60%	91.78%
8	4.68	6.81	58.50%	85.17%
16	7.61	11.66	47.50%	72.87%

(c) Wave Application on Cluster_C

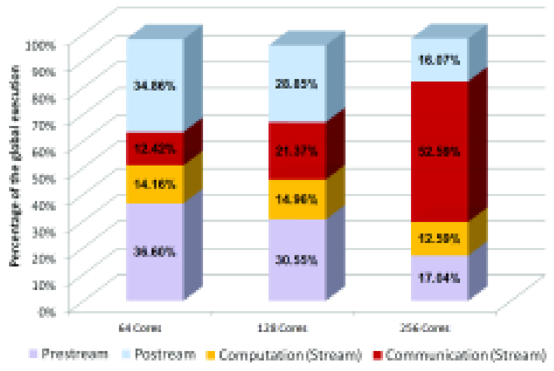
Figure 11: Performance Analysis of SPMD applications on Multi-core clusters

However, these applications have to be analyzed carefully due to the LL-2D-STD-MPI being integrated by 3 main modules: prestream, stream and poststream. However, these applications have a behavior which can be found in many real applications where the exchanging areas (where the method can be applied) could be portions of the complete application. In this case, we will only measure the impact and improvements of these computational segments.

Hence, the prestream and poststream are computation functions, where they do not have any communication. However, the stream part integrates the communication exchanging process between neighboring tiles. An example of this decomposition is illustrated in Figure 12(a), where we can observe the impact of each function and the effect over the execution time. This example shows the relationship for a defined problem size of 2000x2000 using 1000 iterations (size that will be used for both applications). As can be detailed, the functions prestream and poststream



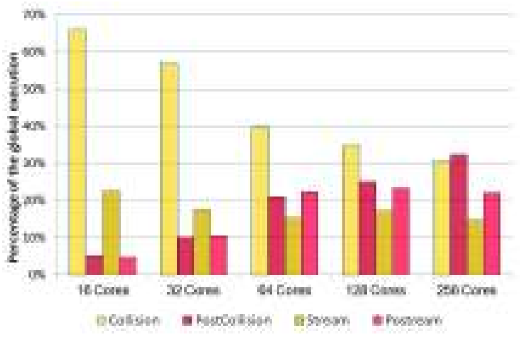
(a) Execution percentage of each function



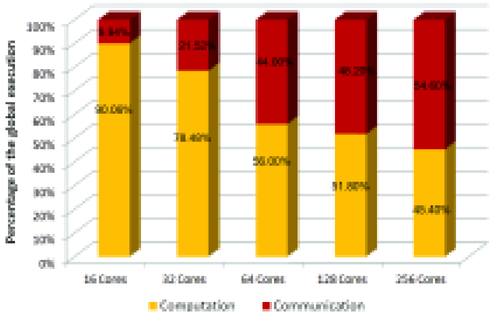
(b) Descomposition function stream

Figure 12: LL-2D-STD-MPI application characterization on Juropa cluster

maintain the same proportion of the execution time, while the stream begins to increase due to the communications effects. The communication starts to introduce a strong impact over the execution. The communication in this case is around 53% of the total execution time when we are using more than 256 cores (Figure 12(b)).



(a) Execution percentage of each function



(b) Descomposition function stream

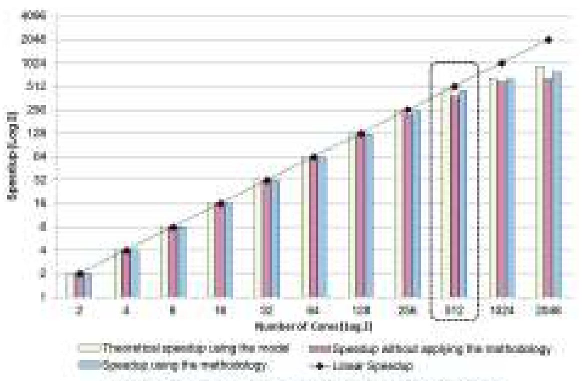
Figure 13: ZSC-2D-STD-MPI application characterization on Juropa cluster

Another example is illustrated in Figure 13, where the ZSC-2D-STD-MPI application is analyzed, which integrates two exchanging functions called collision and stream (Figure 13(a)). Both functions increase the inefficiency, while the number of cores is increased due to its communications exchanges. In this case, the impact of communication is increased while the number of cores is expanded (Figure 13(b)), where half of the time, the application is communicating when executing with 256 cores.

Table 3: Characterization and Tile Distribution Model

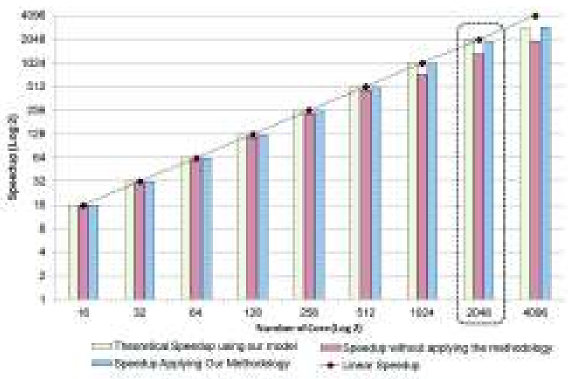
Application	Problem Size	Desired *Effic* (%)	*Cpt* (Sec)	*CommT* (Sec)	λ	K	*Ncores*	Cluster
LL-2D-STD-MPI	7200x7200	90%	$1.58E^{-7}$	$5.51E^{-5}$	348.73	318	513	Juropa
ZSC-2D-STD-MPI	7015x7015	95%	$1.24E^{-7}$	$1.97E^{-5}$	158.87	155	2048	Juropa

As we can observe, these SPMD applications have the exchanging behavior in which we can apply partially our method. For this reason, we have characterized the application and we have defined a problem size in order to evaluate



Nodes	Efficiency without applying the methodology	Efficiency applying the methodology	Speedup without applying the methodology	Speedup applying the methodology
64	95,41%	95,40%	61,10	61,265
128	94,31%	95,31%	120,71	121,997
256	85,04%	94,05%	217,69	240,771
512	75,47%	86,77%	376,16	444,249
1024	57,53%	61,40%	589,13	628,779
2048	31,26%	37,03%	640,12	757,018

(a) LL-2D-STD-MPI Application

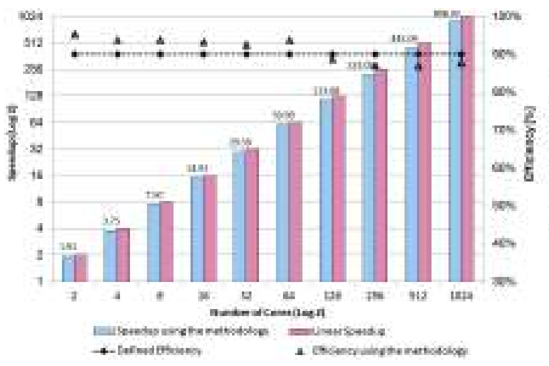


Nodes	Efficiency without applying the methodology	Efficiency applying the methodology	Speedup without applying the methodology	Speedup applying the methodology
128	91,38%	97,63%	116,30	124,869
256	80,45%	91,67%	[illegible]	[illegible]
512	86,28%	97,18%	436,71	497,590
1024	81,58%	86,60%	845,50	888,216
2048	71,61%	81,54%	1466,74	1697,357
4096	53,21%	70,54%	2179,68	2889,455

(b) ZSC-2D-STD-MPI Application

Figure 14: Strong scalability analysis

strong scalability. A summary of the characterization and the values obtained on the model can be found in table 3, where we can see the value of computational and communication time of a tile. Using these values, we have analyzed the scalability and efficiency of both applications. These have been performed evaluating the speedup and efficiency parameters using and not using our method, with the aim of evaluating the improvements obtained using our method. We also calculated the theoretical behavior of applications according to the overlapping strategy between internal computation and the edge communication defined in our method.



Nodes	Div K	Div T	Size (X)	Size (Y)	Execution Time (Sec)	Efficiency	Speedup	Error Rate
2	2	1	636	318	3.48420	95.18%	1.91	3.28%
4	2	2	636	636	3.65621	93.65%	3.75	3.85%
8	4	2	1272	636	3.66021	93.79%	7.50	3.74%
16	4	4	1272	1272	3.67621	93.31%	14.93	3.32%
32	8	4	2544	1272	3.58621	92.46%	29.58	2.46%
64	8	8	2544	2544	3.66021	93.79%	59.98	3.74%
128	16	8	5088	2544	3.65121	88.61%	113.58	3.19%
256	16	16	5088	5088	3.72284	87.13%	223.05	2.87%
512	32	16	10176	5088	3.73159	86.93%	445.08	3.07%
1024	32	32	10176	10176	3.68480	87.79%	898.92	2.27%

Figure 15: Weak scalability analysis

Then, we have selected a problem size and we have defined an efficiency threshold. Both values can be detailed in table 3 and are used to calculate the ideal number of cores and the ideal K size, which make the tradeoff between the efficiency and speedup. Two examples are shown in Figure 14(a) and 14(b), where we have defined a problem size fix and we have increased the number of cores. In this case, we can observe that the best tradeoff between efficiency and speedup are close to the point calculated through the model with an error rate lower than 4% for both applications.

Another analysis is carried out using the weak scalability concept, in which we expand the problem size and the number of cores maintaining the execution time constant. In this sense, we have considered the initial value for the

LL-2D-STD-MPI application (table 3). In this case, the ideal value of the ST is 318squared. Therefore, if we wish to obtain a constant execution time when we increase the number of cores and the problem size, we have to create the same amount of ST as execution cores in order to assign one ST per core and also, each ST has maintained the same size calculated through the model, with the aim of maintaining the overlapping strategy and thus, we can maintain the performance of the system. Then, Figure 15 shows an example of increasing the problem size and the number of cores proportionally to the ST size and the results illustrate how the execution time is approximately maintained with a small error with the worst case of 5%. This is clearly demonstrated with the values of table in Figure 15. In this case, the efficiency ranges between (+/-) 4% of the efficiency threshold defined, depending on the number of core used. It is important to remark that for this analysis we have growth in the problem size using two directions X and Y. This is because we have to create a logical MPI process mesh as was explained in the mapping step. In this sense, DivX and DivY define the number of process in dimension X and Y and the problem size has to be increased considering these two values and also the ST size.

As can be detailed in this experimental validation, our method can address the two initial targets of finding the ideal number of core that make a tradeoff between efficiency and speedup, and also, we can find the ideal ST which has to be used in order to increase the problem size and maintain the application performance. This phenomenon is motivated by the fact that the ST is created under the consideration of the overlapping strategy and the main objective of this strategy is to hide the effects of communications and therefore increase the performance of the application. It is possible to apply this method due to the deterministic behavior of the SPMD applications.

6. Conclusion and Open lines

This paper has presented a methodology for efficient execution of SPMD application on multi-core cluster. This method is based on characterization, tile distribution model, mapping strategy and scheduling policy. Our methodology is focused on determining the ideal number of tiles analytically, which have to be assigned to each ST, and it also determines the ideal number of core which maintains the execution efficiency. This is performed using an efficient way to manage the hierarchical communication architecture presented on multi-core clusters. This work also addresses how we can combine the efficiency and the strong and weak scalability in parallel applications. In this sense, using our method, we can observe how the SPMD applications with some specific characteristics can behave with a specific problem size while the number of cores is incremented. This is the main purpose of finding the maximum point that allows the SPMD application to scale linearly. On the other hand, if the problem size is increased according to the relationship of the ST, we can maintain a linear speedup when the numbers of cores are increased.

This methodology allows us to execute SPMD applications efficiently on multi-core clusters and this has been validated with the experimental evaluations. The SPMD application speedups reached the maximum level when the application efficiencies were around a defined threshold. Experimental evaluation makes it clear that in order to achieve a better performance in SPMD applications, we have to manage the communication heterogeneities. For this reason, our methodology evaluates the environment through the characterization phase and we apply our model with real values of the multi-core architecture. Then, the mapping manages the set of tiles necessary for each core according to the communication links and these tiles are divided into internal and edge tiles. Then, the scheduling allows us to design an overlapping method where internal tiles are overlapped with edge communication and using a tile execution priority method.

Finally, the examples described in the performance evaluation have shown that using the ST size and the number of core obtained through the analytical model, we can obtain the maximum strong scalability point with a small error which can vary between (+/-) 5% depending on the SPMD application tested and the multi-core system used. Moreover, the weak scalability analysis has demonstrated that if we increase the problem size according to the relationship of the ST size, we can maintain the efficiency and speedup conditions. In this sense, the weak scalability allows us to determine the ideal problem size for a specific number of cores, that is, our method seeks that the application can be adapted as well as possible to the parallel environment with the aim of obtaining a linear scalability.

The next open lines to this method are addressed in two directions, in adapting the method to execute in heterogeneous computation multi-core clusters and the second approach will be focused on the execution of SPMD applications efficiently on a Graphics processing Unit (GPU).

Acknowledgement

This research has been supported by the MINECO (MICINN) Spain under contracts TIN2011-24384 and TIN2014-53172-P

References

[1] I. M. Nielsen, C. L. Janssen, Multicore challenges and benefits for high performance scientific computing, Scientific Programming 16 (4) (2008) 277–285.
[2] M. Mccool, Scalable programming models for massively multicore processors, Proc. of the IEEE 96 (5) (2008) 816–831.
[3] S. Akhter, J. Roberts, Multi-Core Programming, Intel Press, 2006.
[4] L. Chai, Q. Gao, D. Panda, Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system, in: Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on, 2007, pp. 471–478. doi:10.1109/CCGRID.2007.119.
[5] R. Rabenseifner, G. Hager, G. Jost, Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes, in: Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on, 2009, pp. 427–436. doi:10.1109/PDP.2009.43.
[6] R. Kumar, V. Zyuban, D. Tullsen, Interconnections in multi-core architectures: understanding mechanisms, overheads and scaling, in: Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on, 2005, pp. 408–419. doi:10.1109/ISCA.2005.34.
[7] Z. Liu, J. Yu, X. Wang, B. Liu, L. Bhuyan, Revisiting the cache effect on multicore multithreaded network processors, in: Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on, 2008, pp. 317–324. doi:10.1109/DSD.2008.41.
[8] R. Muresano, D. Rexachs, E. luque., How spmd applications could be efficiently executed on multicore environment, in: In proc of IEEE International Conference on Cluster Computing and Workshops, (CLUSTER), New Orleans, LA, 2009, pp. 1–4.
[9] R. Buyya, High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice Hall , New Jersey USA, 1999.
[10] F. Darema, The spmd model: Past, present and future, Proceedings of the 8th European PVM/MPI Users (2001) pp. 1.
[11] A. Kayi, T. A. El-Ghazawi, G. B. Newby, Performance issues in emerging homogeneous multi-core architectures., Simulation Modelling Practice and Theory (2009) 1485–1499.
[12] L. M. Liebrock, S. P. Goudy, Methodology for modelling spmd hybrid parallel computation, Concurr. Comput. : Pract. Exper. 20 (8) (2008) 903–940.
[13] K. Vikram, V. Vasudevan, Mapping data-parallel tasks onto partially reconfigurable hybrid processor architectures, Trans. on Very Large Scale Integration Systems 14 (9) (2006) 1010–1023.
[14] O. Beaumont, A. Legrand, Y. Robert, Optimal algorithms for scheduling divisible workloads on heterogeneous systems, in: International Parallel and Distributed Processing Symposium (IPDPS 2003), 2003, pp. 1–14.
[15] L. C. Pinto, L. H. B. Tomazella, M. Dantas, An experimental study on how to build efficient multi-core clusters for high performance computing, 11th IEEE International Conference on Computational Science and Engineering (2008) 33–40.
[16] F. Trahay, E. Brunet, A. Denis, R. Namyst, A multithreaded communication engine for multicore architectures, Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on (2008) 1–7.
[17] C. Lee, Y.-F. Wang, T. Yang, Global optimization for mapping parallel image processing tasks on distributed memory machines, Journal of Parallel and Distributed Computing 45 (1) (1997) 29 – 45. doi:http://dx.doi.org/10.1006/jpdc.1997.1360.
[18] S. Sanyal, S. Das, Match: Mapping data-parallel tasks on a heterogeneous computing platform using the cross-entropy heuristic, 19th IEEE International Parallel and Distributed Processing Symposium (2005) 64b–72b.
[19] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, R. Namyst, hwloc: A generic framework for managing hardware affinities in hpc applications, Parallel, Distributed, and Network-Based Processing, Euromicro Conference on 0 (2010) 180–186. doi:http://doi.ieeecomputersociety.org/10.1109/PDP.2010.67.
[20] G. Mercier, J. Clet-Ortega, Towards an efficient process placement policy for mpi applications in multicore environments, EuroPVM/MPI 2009 (2009) 104–115.
[21] J. B. Weissman, Prophet: automated scheduling of spmd programs in workstation networks, Concurrency - Practice and Experience 11 (6) (1999) 301–321.
[22] J. B. Weissman, X. Zhao, Scheduling parallel applications in distributed networks, Cluster Computing 1 (1998) 109–118.
[23] M. Panshenskov, A. Vakhitov, Adaptive scheduling of parallel computations for spmd tasks, ICCSA 2007 (2007) 38–50.
[24] S. Boyd-Wickizer, R. Morris, M. F. Kaashoek, Reinventing scheduling for multicore systems, in: Proceedings of the 12th Conference on Hot Topics in Operating Systems, HotOS'09, USENIX Association, Berkeley, CA, USA, 2009, pp. 21–21.
[25] S. Mittal, P. Sharma, An optimized and efficient multi parametric scheduling approach for multi-core systems, International Journal of Computer Theory & Engineering 5 (3).
[26] L. G. Valiant, A bridging model for parallel computation, Commun. ACM 33 (8) (1990) 103–111. doi:http://doi.acm.org/10.1145/79173.79181.
[27] L. G. Valiant, A bridging model for multi-core computing, SPAA 2008, 20th ACM Symposium on Parallelism in Algorithms and Architectures Volume 5193/2008 (2008) 13–28.
[28] V. der Wijngaart, H. Jin, Nas parallel benchmarks, multi-zone versions, Tech. rep., NASA Advanced Supercomputing Division Ames Research Center, USA (2003).
[29] A. Grama, Isoefficiency: measuring the scalability of parallel algorithms and architectures, IEEE, Parallel & Distributed Technology: Systems & Applications 3 (1993) 12 – 21.
[30] L. Peng, M. Kunaseth, H. Dursun, K. ichi Nomura, W. Wang, R. K. Kalia, A. Nakano, P. Vashisht, A scalable hierarchical parallelization framework for molecular dynamics simulation on multicore clusters, Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications, USA (2009) 97–103.

[31] A. Hoisie, O. Lubeck, H. Wasserman, Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications, International Journal of High Performance Computing Applications 14 (2000) 330–346.

[32] R. Muresano, D. Rexachs, E. Luque., Methodology for efficient execution of spmd applications on multicore clusters, 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID), IEEE Computer Society (2010) 185–195.

[33] A. Wong, D. Rexachs, E. Luque, Pas2p tool, parallel application signature for performance prediction, in: K. Jasson (Ed.), Applied Parallel and Scientific Computing, Vol. 7133 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 293–302. doi:10.1007/978-3-642-28151-8$_2$9.

[34] A. Wong, D. Rexachs, E. Luque, Parallel application signature for performance analysis and prediction, Parallel and Distributed Systems, IEEE Transactions on 26 (7) (2015) 2009–2019. doi:10.1109/TPDS.2014.2329688.