Universitat Autònoma de Barcelona
Escola d'Enginyeria
Departament d'Arquitectura de Computadors i Sistemes Operatius

Metodología para la ejecución eficiente de aplicaciones SPMD en clústeres con procesadores multicore

Memoria presentada por Ronal Roberto Muresano Cáceres para optar al grado de Doctor por la Universidad Autónoma de Barcelona. Este trabajo ha sido desarrollado en el departamento de Arquitectura de Computadores y Sistemas Operativos bajo la dirección del Dr. Emilio Luque Fadón

Barcelona; Junio, 2010

Metodología para la ejecución eficiente de aplicaciones SPMD en clústeres multicore

Tesis Doctoral presentada por Ronal Roberto Muresano Cáceres para optar al grado de Doctor por la Universidad Autónoma de Barcelona. Este trabajo ha sido desarrollado en el departamento de Arquitectura de Computadores y Sistemas Operativos de la Escuela de Ingeniería de la Universidad Autónoma de Barcelona, dentro del Programa de Doctorado en Computación de Altas Prestaciones bajo la dirección del Dr. Emilio Luque Fadón.

Barcelona; Junio 2011

Director de Tesis

Dr. Emilio Luque Fadón.

A Juan Diego y Carolina
por todo su apoyo

.

Agradecimientos

Una etapa de mi vida profesional culmina con este trabajo de investigación, y hay muchas personas que me han apoyado para la realización y culminación del mismo. Por eso comienzo agradeciendo a Dios que siempre me guía a tomar las mejores decisiones y que me ha dado la fortaleza para seguir adelante en los momentos difíciles durante la realización de este trabajo.

A Carolina y a Juan Diego por ser siempre un apoyo incondicional y por la comprensión que en estos meses finales han tenido, les agradezco por todos esos momentos que hemos tenido que sacrificar para trabajar con la tesis, pero esos momentos se ven recompensados con el alcancel de esta meta. Este logro es realmente de los tres, gracias por ser la familia excepcional que son.

A mis padres Irene y Emilio, ambos son un ejemplo de admiración, muchas gracias por soportar todas las dificultades que han tenido que pasar en estos últimos años y que por ese amor incondicional de padres lo han soportado para que en ningún momento flaqueara y pudiera terminar este sueño.

Lola muchas gracias por el apoyo que me has dado en todo momento, he aprendido mucho de tu parte, no solo en lo profesional sino también en lo afectivo. Haz sido uno de los pilares fundamentales de este trabajo. Muchas gracias por enseñarme a levantarme en los momentos más difíciles. Eres un ejemplo…

Quiero agradecer a mi director Emilio Luque por ser siempre un guía en este proyecto, siempre dando los mejores consejos. Mil gracias por todo tu apoyo. Me has enseñado que no hay nada imposible en el mundo científico. Gracias por todos tus consejos.

A los Titos (Fernando y Yorley) por ser los amigos incondicionales que en todo momento han estado pendiente de mi familia y de mí, gracias por ser como unos hermanos.

A mis amigos Guna, Genaro, Javier, Marcela, Carlos Núñez, Hugo, Paula, Alexander, Alvaro Wong, Gonzalo, Leonardo, Alvaro Chalar, Kerstin, Andrés, Joao y todos aquellos que me han apoyado en el desarrollo de este trabajo.

Gracias a las personas que hacen posible que todas nuestras ideas puedan ser probadas. A Dani y Xavi por tener la paciencia suficiente a la hora de hacer las peticiones. Estoy muy agradecido por todos los soportes ofrecidos.

Finalmente quiero agradecer a todos aquellos que por espacio no he nombrado y que han sido partícipes del desarrollo de esta investigación.

Resumen

La necesidad de ejecutar aplicaciones en ambientes heterogéneos es un obstáculo que la programación paralela quiere superar. Por esta razón, hemos diseñado una metodología de ejecución eficiente que permita gestionar las heterogeneidades de comunicación y cómputo en sistemas jerárquico de comunicaciones como el presentado en un clústeres con procesadores multicore. Por lo tanto, se busca mejorar la eficiencia del sistema y determinar el máximo *speedup* (escalabilidad de la aplicación en el sistema) con una eficiencia definida.

La metodología está compuesta por cuatro fases: la de caracterización, un modelo de distribución de tareas, una estrategia de mapping y la política de Scheduling. Nuestra metodología se enfoca en las aplicaciones Single Program Multiple Data (SPMD) que están diseñadas con librería de paso de mensajes para realizar el proceso de comunicación.

Asimismo, fueron seleccionadas debido a que son aplicaciones con alto volumen de comunicación y sincronismo, puntos que genera desafíos a los programadores cuando éstos desean ejecutarlas en entornos de comunicaciones heterogéneas de manera rápida y eficiente. La principal contribución de ésta metodología es determinar el número aproximado de cores necesarios y el número de *tiles* que deben ser asignadas para cumplir nuestro objetivo, de obtener el máximo *speedup* mientras la eficiencia es mantenida por encima de un umbral definido por el usuario.

La metodología ha sido evaluada con un conjunto de aplicaciones científicas y lo resultados muestran en el mejor de los casos mejoras que alcanzan hasta 39% en la métrica de la eficiencia.

Abstract

The need to efficiently execute applications in heterogeneous environments is an obstacle parallel computing wants to overcome. The communication heterogeneities present on Multicore clusters ought to be handled carefully for improving efficiency and speedup. For this reason, executions on Multicore environments become a challenge that parallel application programmers must be prepared to deal with.

This work proposes an execution methodology centered on controlling communication heterogeneities and improving system efficiency on Multicore clústers. Such methodology is composed by four phases: characterization, a tile distribution model, mapping strategy, and scheduling policy. This method has been developed for Single Program Multiple Data (SPMD) applications with high communication synchronism. We will focus on SPMD applications which are designed through a message-passing library for communication, and selected according to their synchronicity and communications volume.

The main contribution of this methodology is to determine the approximate number of cores necessary to achieve a suitable solution in obtaining a good execution time, while the efficiency level is maintained over a threshold defined by users and.

Our results show an improvement around 39% in the best case of efficiency on SPMD applications tested when our methodology is applied.

Resum

La necessitat d'executar aplicacions en ambients heterogenis és un obstacle que la programació paral·lela vol superar. Per aquesta raó, hem dissenyat una metodologia d'execució eficient que permeti gestionar les heterogeneïtats de comunicació i còmput en sistemes jeràrquics de comunicacions com el presentat en un Cluster Multicore. Per tant, es busca millorar l'eficiència del sistema i determinar el màxim *speedup* (escalabilidad de l'aplicació en el sistema) amb una eficiència definida.

La metodologia està composta per quatre fases: la de caracterització, un model de distribució de tasques, una estratègia de *mapping* i la política de *scheduling*. La nostra metodologia s'enfoca en les aplicacions Single Program Multiple Data (SPMD) que estan dissenyades amb llibreria de pas de missatges per realitzar el procés de comunicació.

Així mateix, van ser seleccionades a causa que són aplicacions amb alt volum de comunicació i sincronisme, punts que genera desafiaments als programadors quan aquests desitgen executar-les en entorns de comunicacions heterogènies de manera ràpida i eficient. La principal contribució d'aquesta metodologia és determinar el nombre aproximat de cores necessaris i el nombre de *tiles* que han de ser assignades per complir el nostre objectiu, d'obtenir el màxim *speedup* mentre l'eficiència és manté per sobre d'un llindar definit per l'usuari.

La metodologia ha estat avaluada amb un conjunt d'aplicacions científiques i el resultats mostren en el millor dels casos millores que aconsegueixen fins a 39% en la mètrica de l'eficiència.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Ecuaciones

European Doctor Mention (Thesis Resume, Introduction and Conclusions)

1. Introduction

The increasing use of multicore in High Performance Computing (HPC) can be evidenced in the top500[1] list, in which most of nodes of the today clusters are set up with multicore processors and these nodes use a hierarchical communication architecture that can affect the communication performance of the application. In this sense, this architecture has to be handled carefully if programmers want to study the impact of multicore technology on computational and communication speed with the aim of improving the parallel application performance metrics [3]. Also, the insertions of multicore nodes in HPC have allowed having more parallelism within the nodes, however, this parallelism have to be managed properly when the performance metrics wish to be enhanced. The most relevant problems found on multicore clusters are related to number of cores per chip, number of cores per nodes, data locality, shared cache, bus interconnection, memory bandwidth, communication congestion etc. [42], and these problems are becoming more important to be considered.

Therefore, the need of improving performance metrics in these hierarchical communication environments is an obstacle that parallel computing is striving to overcome. Metrics such as execution time, *speedup*, efficiency and application scalability are influenced in a different manner, when parallel applications are executed in these heterogeneous environments.

In this sense, we consider the multicore clusters as heterogeneous due to their different communication paths, which present different speeds and bandwidths[4] and these differences may cause degradations in the application performance [20][31]. For example, on multicore clusters some communications are realized through network links such as local area networks (LAN), and other communications are established by internal processor buses, e.g., an *intercore* communication travels through cache memory, while an interchip communication is delivered by the main memory.

[1] TOP500 is a list which provides a rank of the parallel machines used for high performance computing www.top500.org

Another issue that parallel programmers have to consider is that many traditional MPI (Message Passing Interface) parallel applications were designed to be executed in clusters comprised of single core nodes, in which communication within the processor is not present. However, when these applications are executed on multicore clusters, the processes have to exchange their information with other processes that are located in the same chip processor, same node or in different nodes. These communication exchanging processes can generate communications imbalances that can create delays in the execution.

An additional aspect to be contemplated for an efficient execution is the parallel paradigm. The master/worker, Single Program Multiple Data SPMD (SPMD), pipeline, divide and conquer, etc. are examples of parallel paradigms and each of these paradigms has a different behavior and communication pattern that have to be managed properly.

Then, this thesis is mainly centered on studying applications using message-passing libraries with high synchronicity through tile dependencies and communication volumes such as SPMD applications on multicore clusters. This SPMD paradigm has been chosen because is one of the most used parallel paradigm in HPC [43] . Also, it was selected due to its behavior, which is to execute the same program in all processes but with a different set of *tiles* [7]. An SPMD tile is executed in a similar computational time but the communication process among neighbor are performed by different links depending on the location of SPMD processes. These communication links can vary their time in an order of magnitude according to the links and these variations are a limiting factor to improve the performance.

Despite of these communications issues and the behavior of MPI applications, we can take advantage of the computational power that these clusters offer with the aim of running applications faster and efficiently [34][36]**.** The applications selected have to meet three characteristics: static, where the communication process is maintained during all the execution, local, because do not have collective communications and regular, due to communications are repeated for several iterations. Also, the SPMD

applications used have been 2D grid problems with high communication volume.

The computation processes have to wait until the slowest communications link finishes of receiving its information to start the new iteration. These waiting times are translated into system inefficiency. The main objective of this extended abstract is focused on describing an efficient execution methodology for multicore clusters, which is based on achieving a suitable application execution with a maximum *speedup* achievable while the efficiency is maintained over a defined threshold. Also, this methodology is centered on calculating the maximum number of cores that maintain the strong and weak application scalability under a desired efficiency for SPMD applications. The objective of strong application scalability is to maintain the problem size constant while the number of processors increases [2][26] and the weak, where the problem size and the multiprocessor size expand, and the goal is to achieve constant time-to-solution for larger problems [48].

This methodology sets the SPMD *tiles* of the application in groups, called *supertile* (ST), assigning each one of them to one core. The *tiles* of these ST belong to one of two types: internal *tiles*, which their communications are made in the same core and edge *tiles*, where their communications are performed with *tiles* allocated in other core. This division allows us to apply an overlapping method that permits us to execute the internal *tiles* while the edge communications are performing.

Also, the methodology has been designed in four phases as follow: the characterization (application and environment), a tile distribution (we determine the number of tile and number of core necessary to execute efficiently and with the maximum *speedup*), a mapping strategy (distribution of SPMD *tiles* over cores), and a scheduling policy (define the execution order of SPMD *tiles* assigned). This efficient execution methodology has been tested with different scientific applications and we have reached an improvement around 42% in efficiency when we applied our method.

2. SPMD applications on multicore Clusters

The communication processes of SPMD applications on multicore clusters have to be performed through different communication links, and these links have different speed and bandwidths. These differences can create waiting time and these times are mainly created due to the SPMD application behavior, which is to execute the same program in all processes but with a different set of *tiles*. These processes have to exchange their information related to the *tiles* due to the tile dependency each iteration, in order to calculate the values for next iteration. It's important to define the kind of SPMD applications that our methodology attempt to improve. In this sense, the SPMD applications selected present a synchronicity through task or tile dependencies and they do not include a process synchronicity as was established by Valiant [57] in the BSP model.

The different communication patterns can vary according to the objective of the parallel applications. In some cases, the communications can be executed in two, four, six, etc bidirectional communications. These communications pattern are established at the beginning of the SPMD application and these patterns are kept until the application finishes. For this reason, we have to manage the communication imbalance of the multicore architecture because it can create a huge inefficiency in the parallel execution that can be repeated during all the application's iterations.

As mentioned before, the applications used to apply our methodology have to be designed under the characteristics of static, local and regular. A large number of benchmarks and applications of diverse fields accomplish all these characteristics. One example of benchmark can be detailed the NAS parallel benchmark in the CG, BT, MG, SP app. [58]**,** all these benchmark have been designed for 2D and in some case for 3D grid problem size. Also, there are examples of real applications such as: linear algebra problems, algorithms for solving partial differential equations and finite differences, etc [8]. Some examples of these applications are the heat transfer [64], wave simulation [L13] , Laplace equation [23] , fluid dynamics

(mpbl suite) [54][66], etc. All these applications efficiency are mainly affected by the communications imbalances of these multicore environments.

An example of the inefficiency is illustrated in Figure 1, where the SPMD *tiles* are executed in similar time due to the homogeneity of the cores, but the communication process have different communication times depending on the communication links that are used for these processes. Also, the figure illustrates the idle time generated by slower communication links, for example, The core identify as 5 is communicating from node 1 with core 9 of node 2 through the inter-node link identified in figure as C. These communications have the bigger delay than communication perform of core 5 with core 6 of the same node using an interchip communication (B). The slowest communication is a limiting factor because core 5 cannot begin to calculate the next iteration due to all the information is not complete transferred. These issues are translated into inefficiencies that are repeated until the application ends.

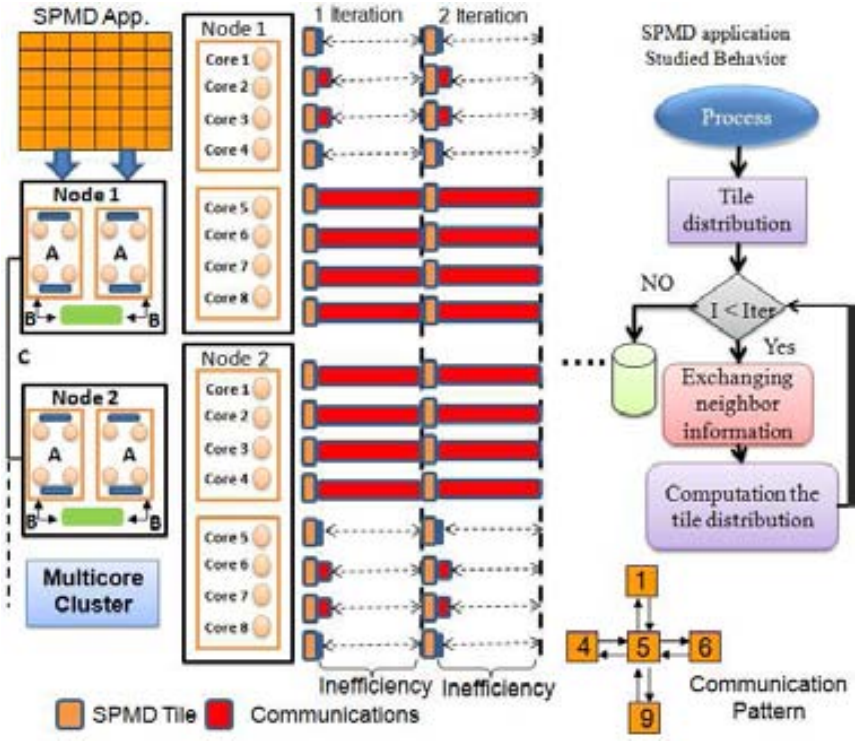


Figure 1 Computation and communication behavior of SPMD application on multicore cluster

However, these idle times allows us to establish suitable strategies in order to organize how SPMD *tiles* could be distributed on multicore cluster with the aim of managing these communications in an efficient manner. It's important to understand that the latency of the slower link will determine an SPMD iteration as is observed in the Figure 1. For this reason, the communication inefficiencies have to be managed if we wish to executed the SPMD applications faster, efficiently and scalable. In order to solve these inefficiency problems, we use the problem size of the SPMD applications that is composed by a number of *tiles* and we create the *supertile* (ST) which is a group of data that will be assigned to each core.

The problem of finding the optimal ST size is formulated as an analytical problem, in which the ratio between computation and communication of the tile has to be founded with the objective of searching the relationship between efficiency and *speedup*. The ST is calculated maintaining the focus of obtaining the maximum *speedup* while the efficiency is maintained over a defined threshold. This optimal ST is decomposed by two different types of tiles: internal and edge. This tile's division allows us to apply an overlapping strategy which consists of executing first the edge tiles and then, we overlap the edge communications and internal tiles as is shown in Figure 2.

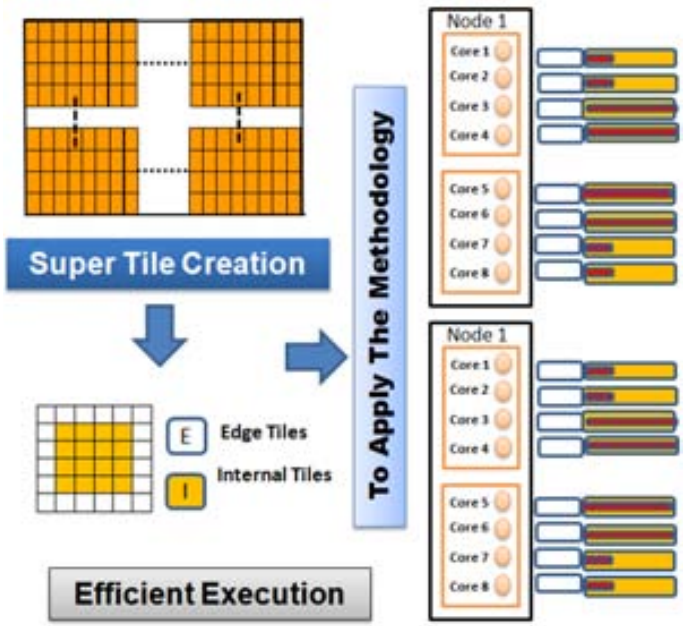


Figure 2 SuperTile (ST) creations for improving the efficiency.

Finally, we can use a set of *tiles* to form a ST of (K^n), where ***K*** is the square root of the number of *tiles* that have to be assigned to each core in order to maintain the relationship between efficiency and *speedup* and ***n*** is the application dimension . This ideal scenario is found when we apply our methodology to maintain the efficiency. The main target of our method is to find the ideal number of *tiles* of the *supertile* and cores that allows us to achieve the maximum *speedup* with a desired efficiency.

3. Methodology for efficient execution of SPMD applications on multicore clusters

This methodology is focused on managing the communications heterogeneities presents on multicore clusters with the objective of improving both efficiency and *speedup*. This improvement process is realized through four phases (Figure 3): a characterization phase, which is focused on performing an application and environment analysis with the aim of obtaining the system parameters (application and execution environment) which are used in the second phase. The second phase is integrated by an analytical model called tile distribution model, where the ideal size is calculated. This phase determines the association between the ideal number of *tiles* and cores necessary with the objective of maintaining a relationship between internal computation and edge communications *tiles*.

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

Finally, the scheduling phase has two functions, one of them is to assign *tiles* priorities with the aim of calculating first the *tiles* with edge communication and then, the *tiles* with internal communication, and the other function is to control the overlapping process between internal computation and edge communication. These phase allow us to handle the latencies and the communication imbalances due to the different communication paths [38].

Figure 3 Methodology for efficient execution steps

- ***Characterization Phase***

When an SPMD application has tile dependency with neighboring *tiles*, we need to evaluate, how the SPMD application behave on a specific multicore clusters. This evaluation allows us to obtain the inputs necessary to set up the number of *tiles* to each core with the aim of finding the relationship between efficiency and *speedup*. Then, the main objective of this phase is to gather the necessary parameters of SPMD applications and the execution environment in order to calculate the tile distribution model. The characterization parameters are classified in three groups: the application parameters, parallel environment characteristics and the performance user needs. To develop this phase, we evaluate computation and communication behavior of SPMD applications with the aim of obtaining the parameters with the nearest relationship between the machine and the application.

The application parameters offer the information necessary of the application characteristics such as: problem size, number of tile, iteration number, communication pattern, communication volume of a tile, data distribution. Also, 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, and so on, neighboring tile. Moreover, an SPMD application can consider different distribution schemes, for example, one-dimensional, two-dimensional block, column based and unconstrained [44]**.**

The environment characterization consists in to evaluate the behavior of the SPMD application on a specific multicore parallel machine. The parameters determined allow us to establish the communication and computational ratio time of a tile inside of the hierarchical communication

architecture. This relationship values will be defined as ***λ(p)(w),*** where **p** determine the link where the communication of one processing tile to another neighboring tile has been performed.

The value of determines the communication and computation ratio of a tile and **w** describes the direction of the communication processes (e.g. Up, right, left or down in a four communications pattern). This ratio is calculated with equation Eq. 1, where ***Commt(p)(w***) determines the time of communicating a tile for a specific p link and the C***pt*** is the value of computing one tile on a core. This characterization process has to be done in a controlled and monitored manner.

$$\lambda(p)(w) = \frac{\mathrm{Commt(p)(w)}}{Cpt} \qquad \text{(Eq. 1)}$$

Finally, once all parameters are found through the characterization phase, we have to include the efficiency value in the model. The efficiency value is defined by the variable **effic** and it defines the threshold for the execution.

- ***Tile distribution model Phase***

The main objective of this phase is to determine the optimal value of the ST. The first step is to determine the behavior of the execution time of these kinds of SPMD applications. To calculate this value, we use the equation (Eq 2). This equation represents the behavior of SPMD application using an overlapping strategy with the characteristic explained before. As can be detailed in equation (Eq. 2), the first part calculated is the edge tile computation time ***EdgeComp(i)*** and then we add the maximum value between internal tile computation ***IntComp(i)*** and edge tile communication ***EdgeComm(i).*** This process will be repeated for a set of iteration ***iter***. In equation Eq 2, the ***n*** value determines the number of an individual iteration and ***i*** represents the number of a specific core inside the multicore cluster. This model is only done for the communication exchanging part of the SPMD application.

$$Texe_i = \sum_{q=1}^{iter}(\mathrm{EdgeComp}_i + Max\begin{Bmatrix} \mathrm{IntComp_i} \\ \mathrm{EdgeComm_i} \end{Bmatrix}) \quad \text{(Eq. 2)}$$

The values of ***IntComp(i), EdgeComp(i)*** and ***EdgeComm(i)*** are in function of the variable ***K*** as can be observed in equations Eq 3, Eq 4 and Eq 5 respectively. This value of ***K*** represents the square root of the optimal size of the ST and ***n*** determines the dimension of the problem and this value ranges from 1 to 3, according to the SPMD application dimension.

$$ST = K^n \quad \text{(Eq. 3)}$$

$$\mathrm{IntComp}_i = (K-2)^n * Cpt \quad \text{(Eq. 4)}$$

$$\mathrm{EdgeComp}_i = (ST - (K-2)^n) * Cpt \quad \text{(Eq. 5)}$$

$$\mathrm{EdgeComm}_i = K^{n-1} * \mathrm{Max}\ (\mathrm{Commt(p)(w)}) \quad \text{(Eq. 6)}$$

In addition, the edge communication (Eq. 6) has to be for the worst communication case. This means that we have to use the slowest communication time to estimate the number of *tiles* necessary for maintaining the efficiency. We chose to apply our method using the worst case because SPMD iterations are bounded by the slowest communication as was explained before. These communications may cause that other cores have to wait until the slowest communications finishes due to the data synchronization. Therefore, we have to find the maximum communications time using the maximum value of the ***λ(p)(w)*** ratio (Eq. 1) On the other hand, the sum of the computational time of the edge and the internal computation represent the total computational time of the region of $\mathbf{K^n}$, that is assigned to each core.

From these initial part of the model, our next step is to determine the ideal value of K which represents the conditions of our objective of finding the maximum *speedup* while the efficiency **Effic** is maintained over a defined threshold by user. We start from the overlapping strategy, where internal tile computation and the edge tile communication are overlapped. The equation (Eq. 7) represents the ideal overlapping that allows us to obtain the maximum *speedup* while the efficiency is maintained over a defined threshold.

$$K^{n-1} * \text{Max}\,(\text{Commt(p)(w)}) \geq \frac{(K-2)^n * Cpt}{effic} \qquad \text{(Eq. 7)}$$

However, this equation (Eq. 7) has to consider a constraint defined in equation (Eq. 8) In this sense, the model can allow us that ***EdgeComp(i)*** can be bigger than ***IntComp(i)*** over the defined efficiency (Equ. 7) but the ***EdgeComm(i)*** have to be slower than the ***IntComp(i)*** without any efficiency definition. In this last case, when the edge communication and the internal communication are equal the efficiency is around the maximum value.

$$K^{n-1} * \text{Max}\,(\text{Commt(p)(w)}) \leq (K-2)^n * Cpt \qquad \text{(Eq. 8)}$$

Next step is to find the value of ***K*** from equation (Eq. 7) which calculates the ideal value of ***K*** which represents the conditions of our objective. We start from de ***λ(p)(w)*** (Eq. 1) and solve the $\boldsymbol{Commt(p)(w)}$ value which can be calculated with respect to ***λ(p)(w)*** multiplied by computational time \$Cpt\$ of a tile. This process is performed with the aim of equalizing both internal computation and edge communication equations in function of ***Cpt***. This new value in function of ***Cpt*** is replaced in equation (Eq 7) and we obtain the equation (Eq. 9)

$$K^{n-1} * \text{Max}\,(\lambda\text{(p)(w)} * Cpt) * effic \geq (K-2)^n * Cpt \qquad \text{(Eq. 9)}$$

The next step is to find the value of ***K*** and the number ideal of cores (Eq. 10) which are needs to execute the application with the maximum *speedup* and with the efficiency over a defined threshold. To do this, we start of the initial consideration that establishes that one ST will be assigned to each core. For this reason, we divided the problem size defined as $\boldsymbol{M^n}$ and we divided between the $\boldsymbol{K^n}$ that represent the ideal size of the ST. With this equation (Eq. 10) we can obtain the ideal number of core that allows us to obtain the objective. Also, this number of cores determines the inflection point until the application has a strong scalability.

$$Ncores = \frac{M^n}{K^n} \quad \text{(Eq. 10)}$$

Finally, we can determine the theoretical behavior of the SPMD application for a lower number of cores that the optimal calculated and predict its behavior. Equation (Eq. 11) calculates the new values of K for a number of core given with the objective of determining the execution time with equation (Eq. 2) and calculating the *speedup* and efficiency for these values. This calculation is possible due to the deterministic behavior of SPMD applications used.

$$K = \sqrt[n]{\frac{M^n}{Ncores}} \quad \text{(Eq. 11)}$$

- ***Mapping* Phase**

A set of difficulties arise when we allocate SPMD tile into distinct cores and these cores have to communicate through different links. Under this focus, the main objective of this phase is to design a strategy of allocating the ST into each core with the aim of minimizing the communication effects. The ST assignations are made applying a core affinity which allows us to allocate the set of *tiles* according to the policy of minimizing the communications latencies [32]**.** This core affinity permits us to identify where the processes have to be allocated and how the ST can be assigned to each core.

The next step in this phase is to create a logical processes distribution that allows the application to identify the neighbor communications. This is done using a Cartesian topology of the processes that give to each process two coordinate in the grid distribution. These two coordinates identify the cores, in which the processes have to be allocated. Also, we can coordinate the communication order with the objective of minimizing the saturation of the links.

The last step is to create the ST with the values obtained with the model. It's important to understand that an incorrect distribution of the *tiles* can generate different application behaviors. One example is when the computational time of the *tiles* assigned is bigger than the slower communication time. In this case, the SPMD application has a computation

bound behavior and it determines that application could improve the *speedup* and its efficiency is around the maximum value. This case allows us to add more cores to the execution. Another example is determined, when communication time is bigger than computation time in this case has a communication bound behavior and we can add more *tiles* to each core in order to balance the execution. Then, our analytical model attempts to found the ideal overlap, where all the cores ends their execution in a similar time.

- ***Scheduling* Phase**

The scheduling phase is divided into two main parts: the first one is to develop an execution priority which determines how the *tiles* have to be executed inside the core and the second part of the scheduling phase which is focused on applying an overlapping strategy between internal computation and edge communication *tiles*.

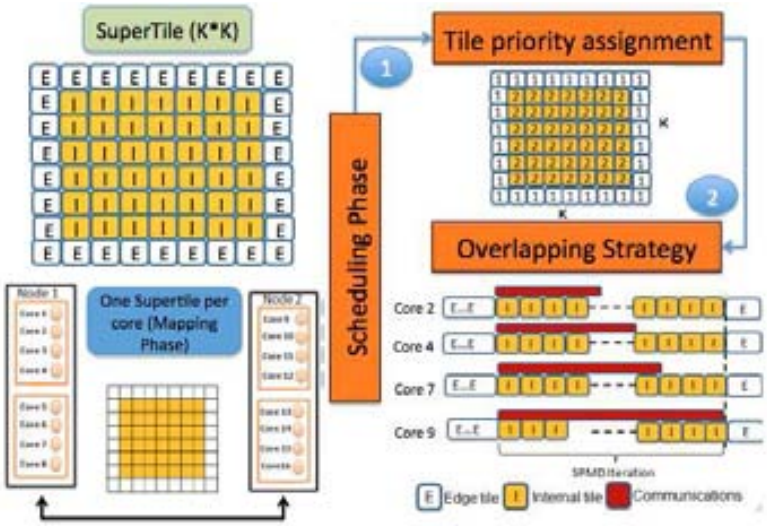


Figure 4 *Scheduling* steps

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. These edge *tiles* are saved in buffers with the aim of executing these *tiles* first. These buffers are updated all iterations. The

second assignation is made for internal *tiles* which are overlapped with the edge communications, which are assigned with the priority 2. The overlapping process uses two threads, one of them is to perform the internal computation and the other is to manage the asynchronous communications. These communications enable us to perform the internal computation and the edge communication together (Figure 4).

4. Scalability and efficiency of SPMD applications on multicore clusters

Our methodology attempts to find the number of core that achieves the maximum strong scalability with a defined efficiency. However, there are two distinct definition of scalability in HPC. One definition is the weak scalability that is considered when the problem size and the number of processing elements are expanded. The main goal of this scalability is to achieve constant time-to-solution for larger problems and the computational load per processor stays constant [48]. The second definition is the strong scalability in which the problem size is fixed and the number of processing elements is increased [26]. The goal in this scalability is to minimize the time solution. Hence, the scalability means that *speedup* is roughly proportional to the number of processing elements.

Under these two scalability definition, our methodology searches a combination between scalability and efficiency. This combination means that our analytical model has to determine the number of cores that allow us to obtain the ideal relationship between *speedup* and the defined efficiency.

This number of cores can be calculated using the model proposed and this number determines the maximum systems capacity growth. Also, we can determine the theoretical behavior of the application (Eq.11). This equation allows us to find the ***K*** value size that has to be assigned to each core. The model only finds one ideal value to maintain the ideal overlapping. However, we can calculate values for another number of cores with the aim of evaluating the performance [39][40].

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 core which maintains the relationship between a defined efficiency and *speedup*. This optimal value allows the programmer to determine the maximum number of cores that are needs when is executed a specific problem size.

On the other hand, the weak scalability combination is performed with the aim of finding the ideal value for a specific problem size. In this sense, the execution can present an approximately linear *speedup* when the problem size and the number of cores are increased according to the *supertile* size. This *supertile* is calculated through the analytical model. The result allows the programmer to visualize, how the application can be adapted in order to improve the performance for a specific number of cores.

- **A theoretical Example**

In order to understand how this methodology works, this numerical example illustrates how efficiency and the strong scalability concept can be combined. Suppose the following application characteristics: a defined problem of ***M***=1585, a defined efficiency ***effic*** of 95% and a four communication neighbors pattern, with three different communication links (quad core architecture). Then, we have to determine the ***λ(p)(w)*** using equation (Eq. 1). This ratio has to be calculated for each link and we use the maximum value obtained. In this sense, we assume that the computational time of a tile is equal to a one unit of time and the maximum communication time for the slowest communication link is equal to 100 time units.

Afterward, we apply our analytical model with the aim of finding the ideal number of cores and the ST size that allow us to achieve the maximum speedup while the efficiency is maintained over a defined threshold. Equation (Eq. 3) determines the ideal ST and equation (Eq. 10) discovers the ideal number of cores. The number of cores calculated represents the maximum combining strong scalability and efficiency for this example. Once the analytical model has been applied, the ideal value of K is equal to 98.95 (Equation Eq. 11) and this value is rounded to the nearest value (K =99) and the number of cores for this execution is equal to 256(Eq. 10). The next step

is to obtain the speedup and the efficiency for this point. To obtain both values, we have to determine the serial execution time of the application. This theoretical time is estimated using the multiplication of the problem size with the computational time of one tile. For this specific problem size, this example has a serial time of 2.512.225 time units; this value is for one iteration.

The analytical model results are shown in Table 1 where we can observe the ideal case calculated and also shows the result obtained for a different distribution of cores. The number of core has been increased in a logarithmical manner (Log_2) with the aim of visualizing the efficiency and speedup curve for this example. Using the equation (Eq.11), we can calculate ***K*** for a specific number of cores.

Table 1 Efficiency, speedup and scalability analysis for an SPMD app.

Cores	K	Edge computation	Internal Computation	Edge Communication	Execution Time	Speedup	Effic
2	1120	4476	1249924	112000	1254400	2	100%
4	792	3164	624100	79200	627264	4	100%
8	560	2236	311364	56000	313600	8	100%
16	396	1580	155236	39600	156816	16	100%
32	280	1116	77284	28000	78400	32	100%
64	198	788	38416	19800	39204	64	100%
128	140	556	19044	14000	19600	128	100%
256	**99**	**392**	**9409**	**9900**	**10292**	**244**	**95%**
512	70	276	4624	7000	7276	345	67%
1024	50	196	2304	5000	5196	483	47%

The last step is to determine the parallel execution time of the application for a specific number of cores with the aim of calculating the speedup and efficiency for these executions. Figure 5 illustrates the performance behavior for different numbers of cores. As can be detailed, the ideal number of cores calculated with the model has an efficiency around the optimal value defined and the speedup up to this point has a roughly linear growth. In this sense, this point is the maximum strong application scalability under a desired efficiency.

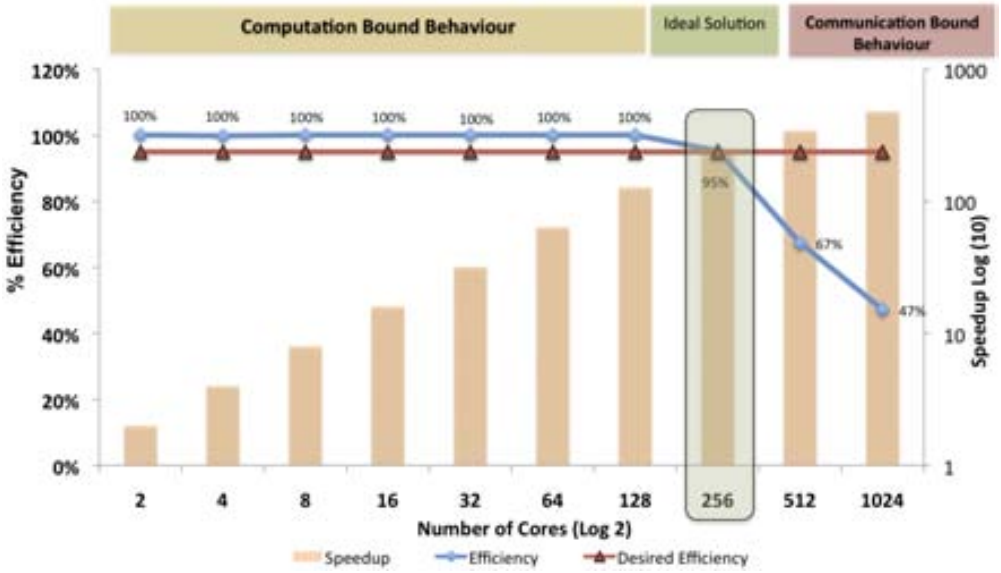


Figure 5 Performance evaluation theoretical example

After this ideal point, we can observe that speedup increases but no proportionally to the number of core. Also, we can see that the efficiency begins to decrease considerably. This decrease in execution efficiency is motivated by the communication bound behavior. This means that after this point the edge communication is bigger that internal communication (Table 1). In addition, we can observe in Figure 5 that the behavior of efficiency and speedup before the ideal point, and in this specific case the efficiency is around the maximum values. This example is a representation of an SPMD application with four neighbor but with computational and communication values adapted for this example.

5. Performance Evaluations

This performance evaluation analyses the behavior of SPMD application on multicore clusters when our methodology is applied. The evaluation is focused on describing the results obtained and the relationship with the *speedup* and efficiency defined in our goal. The methodology experiments have been tested in different multicore clusters. However, for this extended abstract, we show an example executed on an DELL cluster with 8 nodes with 2 processors Quad-Core Intel Xeon E5430 of 2.66 Ghz, 6 MB of cache

L2 shared by each two core and 12GB of RAM memory and a gigabit Ethernet network.

Also, to test our methodology, we have applied the methodology to different scientific applications, however, to show the effectiveness of the execution methodology. We have selected an example of SPMD applications which is programmed using message passing library and also, it has a local, regular and statics behaviors which are the conditions necessary to apply the method. This application is the heat transfer application with a problem size of 9500 tiles squared.

- ***Characterization evaluation***

The first step in order to apply the efficient execution methodology to a heat transfer application is to execute the characterization phase. In this sense, the Figure 6 illustrates the diverse communication paths presents on the cluster DELL and the differences between computation and communication, when we execute a tile. These communication paths present differences which in some cases could be around one and a half orders of magnitude when the same packet size is sent through these communication links. Also, the figure shows the relationship between computation and communication of a task. The behavior shown in the figure allows us to visualize differences in the tile behavior of each communication link. 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.

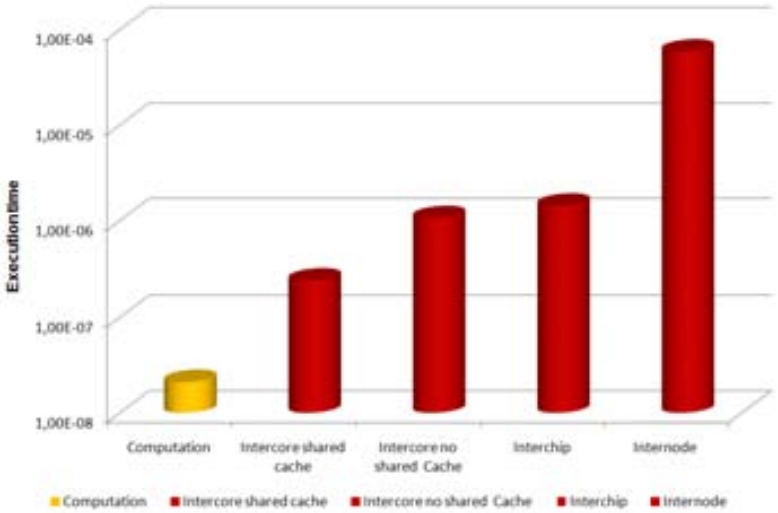


Figure 6 Characterization values of Heat transfer application

A summary of the characterization process is shown in Figure 6, where the characterization values of computation ***Cpt*** and ***Commt(p)(w)*** of slowest communication of a tile, problem size, desired efficiency are illustrated. Once these values are established, the next step is to calculate the ideal size of the supertile and the ideal number of core that allow us to achieve the maximum *speedup* with a defined efficiency.

Table 2 Characterization values on cluster DELL

Application	Tile computation (Sec)	Tile Communication (Sec)	Problem Size (M)	Desired Efficiency (Effic)	Ratio Comm-Comp (λ)	Cluster
Heat Transfer	2.10E-8	5.88E-5	9500	85%	2800.0	DELL

Table 3 illustrates the theoretical values of number of cores, edge and internal computation, the edge communication, the number of tiles ***K*** and execution time are defined for one iteration.

Table 3 Tile distribution model evaluation for one iteration

Application	N^a cores	K	Edge Computation (Sec)	Internal Computation (Sec)	Edge communication (Sec)	Execution time (Sec)
Heat Transfer	16	2383	1.99E-04	1.18E-01	1.40E-01	0.139849

These values are obtained for the best case, however when we apply the model we can predict the execution of the application with different set of cores.

- ***Speedup and efficiency improvement evaluation***

To develop our performance analysis, we executed SPMD applications but making a comparison between the theoretical value, the application without using and the application using our methodology. In this sense, the Figure 7 shows the efficiency behavior of heat transfer application executed with 100 iterations. This figure illustrates a considerable improvement in efficiency of around 39% when we execute with the number of cores determined by our model. The error rate is around 3% between the efficiency obtained and the defined efficiency. This value is achieved when we execute with the ideal number of core calculated through the proposed model.

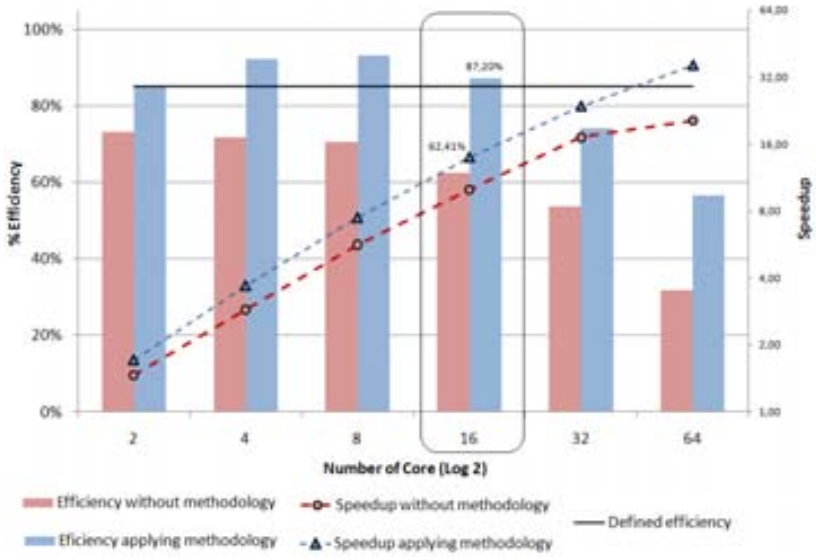


Figure 7 Efficiency and *speedup* improvements

Also, the Figure 7 shows how the *speedup* increases when we add more cores. However, this *speedup* does not scale linearly after the maximum number of core determined with our model.

Another evaluation is related to the combination of both *speedup* and efficiency performance metrics. The values obtained are shown in Table 4, where we can observe that both version using and not using the methodology have an improvement in *speedup*. However, when we compare

the efficiency, we can detect that the application without using the methodology has achieved efficiency around 62%. On the contrary, the efficiency applying the methodology for the ideal number of core is near to 87,20%.

Table 4 Combination of efficiency and *speedup*

N cores	*Speedup* Without applying the method	*Speedup* applying the method	Efficiency without applying the method	Efficiency applying the method
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%	93.26%
16	**9.99**	**13.95**	**62.41%**	**87.20%**
32	17.16	23.71	53.62%	74.10%

This value confirms that when we use our methodology we can combine both metrics in order to find an efficient and fast execution. This is one of the examples that can be found in this research work.

6. Conclusion and open lines

This thesis has presented a novel methodology for efficient execution of SPMD application on multicore cluster [35]. This method is based on characterization, tile distribution model, mapping strategy and scheduling policy [38]. Our methodology is focused on determine the ideal number of *tiles* which have to be assigned to each *supertile*, and also, it determines the ideal number of core which maintains the execution efficiency. This is performed using an efficient manner to manage the hierarchical communication architecture presents on multicore clusters.

Also, this works addresses how we can combine the efficiency and the strong and weak scalability in parallel applications [40]. 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 is incremented the number of cores. 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 *supertile*, we can maintain a linear *speedup* when the numbers of cores are increased [41].

This methodology allows us to execute efficiently SPMD applications on multicore clusters and this has been demonstrated with the performance evaluations. The SPMD application *speedup*s reached the maximum level when the applications efficiencies were around of a defined threshold.

Experimental evaluation makes clear that 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 multicore architecture. Then, the mapping manages the set of *tiles* necessary for each core according to the communication links and these *tiles* are divided in internal and edge *tiles*. Then, the scheduling allows us to design overlapping method where internal *tiles* are overlapped with edge communication and using a tile execution priority method.

The open lines are addressed in two directions, the first one is to include into the model heterogeneous multicore computation environment with the aim of executing efficiently using multicore and GPU architecture together. The second direction is related to apply our methodology to a multicluster architecture where some different communications levels are added to the execution.

Capítulo 1

Introducción

Resumen

Un gran desafío para los desarrolladores de aplicaciones paralelas es ejecutar eficientemente aplicaciones de paso de mensajes. Inicialmente, estas aplicaciones fueron diseñadas para ser ejecutadas en clústeres cuyos procesadores eran de un sólo núcleo, pero actualmente nos encontramos con clústeres compuestos por procesadores de múltiples núcleos (multicore). Los clústeres con procesadores multicore incluyen un sistema jerárquico de comunicaciones, que debe ser gestionado cuidadosamente si se quiere mejorar las métricas de rendimiento de la aplicación paralela como el tiempo de ejecución, el speedup, la eficiencia, y la escalabilidad. Por lo tanto, éste capítulo describe los retos que emergen cuando se utiliza un sistema de cómputo que incluye un sistemas jerárquico de comunicaciones. Estos problemas están relacionados con el desbalanceo que provocan las comunicaciones, y cómo éstas afectan la ejecución de las aplicaciones *Single Program Multiple Data* (SPMD) en los clústeres multicore. Asimismo, se describe el marco de trabajo y los objetivos de la investigación que permiten el desarrollo de una metodología para la ejecución eficiente en clústeres con procesadores multicore. Finalmente, se detalla la organización global del presente documento.

1.1. Motivación

El constante desarrollo que tienen las aplicaciones que requieren cómputo de alto rendimiento y los avances científicos, originan que las mismas sean diseñadas con mayores niveles de complejidad y precisión. Sin embargo, estas precisiones necesitan de mayor poder computacional para ser ejecutadas de una forma considerablemente más rápida [15]. Por este motivo, el cómputo de altas prestaciones (*High Performance Computing* (HPC)) evoluciona constantemente para adaptarse a los requerimientos de cómputo de ciertas aplicaciones con el objetivo de reducir el tiempo de ejecución[51]. Uno de los beneficios de esta constante evolución es la integración en los clúster de nodos con procesadores multicore [21][50].

Esta integración en la computación de altas prestaciones puede observarse en la lista del top500[2], donde la mayoría de los sistemas actuales están diseñados con nodos con procesadores multicore. Estos nodos de cómputo más potentes plantean nuevos retos, al incluir una arquitectura jerárquica de comunicaciones que debe ser gestionada cuidadosamente si los programadores de aplicaciones paralelas requieren mejorar las métricas de rendimiento de los sistemas paralelos [34].

Los nodos multicore en HPC pueden ser considerados heterogéneos debido a que la comunicación entre los cores puede realizarse a través de componentes del mismo procesador, del mismo nodo o externos al nodo. Cada uno de estos componentes presenta un impacto diferente desde el punto de vista de las prestaciones. Por lo tanto, el manejo de la heterogeneidad de comunicaciones en los clústeres con procesadores multicore es un gran desafío que la computación de altas prestaciones debe estar preparada para afrontar, si se desea mejorar las métricas de rendimiento tales como: el tiempo de ejecución, el *speedup*, la eficiencia, y la escalabilidad en estos entornos jerárquicos de comunicaciones.

[2] Top500: lista que determina el ranking de las 500 supercomputadoras con mayor capacidad de cómputo del mundo www.top500.org

Asimismo, la integración de los nodos con procesadores multicore en HPC ha permitido la inclusión de más paralelismo dentro del nodo [42]. No obstante, este paralelismo debe hacer frente a la diversidad de configuración en los ambientes multicores, motivado por el número de cores por procesador (2, 4, 8,… ***n***), la memoria compartida (Cache L1, L2, L3, etc.), el bus de interconexión, el ancho de banda de la memoria, la latencia de los canales de comunicación, entre otros. Todos estos elementos, deben ser considerados cuando se requiere gestionar y/o sintonizar con el fin de lograr una ejecución paralela de forma eficiente.

Es por esto, que la necesidad de mejorar las métricas de rendimiento de éstos ambientes jerárquicos de comunicación, se está convirtiendo día a día en un elemento fundamental a ser gestionado, por lo que deben existir modelos que faciliten su gestión en estos entornos de la computación paralela.

Las métricas de rendimiento paralelo (el tiempo de ejecución, el *speedup*, la eficiencia y la escalabilidad) se ven afectadas por los diferentes enlaces de comunicaciones que incluyen los clústeres con procesadores multicore y más aún cuando las aplicaciones paralelas que se desean ejecutar han sido diseñadas usando una librería de paso de mensaje (tal como MPI, *Message Passing Interface*, que se está convirtiendo en un estándar) para la comunicación entre procesos.

Las diferentes rutas de comunicaciones entre los cores hacen que los clústeres con procesadores multicore se consideren heterogéneos, debido a que los enlaces de comunicación presentan distintas velocidades de trasmisión y anchos de bandas [1]. Estas diferencias de velocidad pueden crear degradación en el rendimiento de la aplicación paralela afectando principalmente al tiempo de ejecución y por consiguiente al *speedup* y a la eficiencia [1] [16][31].

Por esta razón, la gestión de estos niveles de comunicaciones se convierte en un reto cuando se ejecutan aplicaciones que tienen dependencias de la información localizadas en otros nodos; como las aplicaciones desarrolladas bajo el enfoque del paradigma SPMD (*Single*

Program Multiple Data). Por ejemplo, si una aplicación SPMD se ejecuta en un clúster con procesadores multicore, algunas comunicaciones pueden viajar a través de los enlaces de red (LAN, Local Area Network) mientras que simultáneamente se pueden realizar comunicaciones dentro de la arquitectura del nodo (comunicaciones entre procesadores a través de buses internos de la memoria, y comunicación entre cores dentro del chip ejecutadas usando los niveles de memoria cache existentes).

La Figura 8 muestra el problema de comunicaciones para un mensaje de 128Kbytes en los diferentes canales de comunicación de un clúster multicore. Para este ejemplo, se utiliza un entorno multicore con dos *Quad-core* por nodo, con una memoria cache L2 de 6 MB compartida por dos cores, una memoria RAM de 12 GB y una red Giga-Ethernet. En la figura, se visualizan las diferencias entre los distintos niveles de comunicación donde en este caso alcanza un orden y medio de magnitud entre el enlace más lento y el más rápido para el mismo tamaño de paquete.

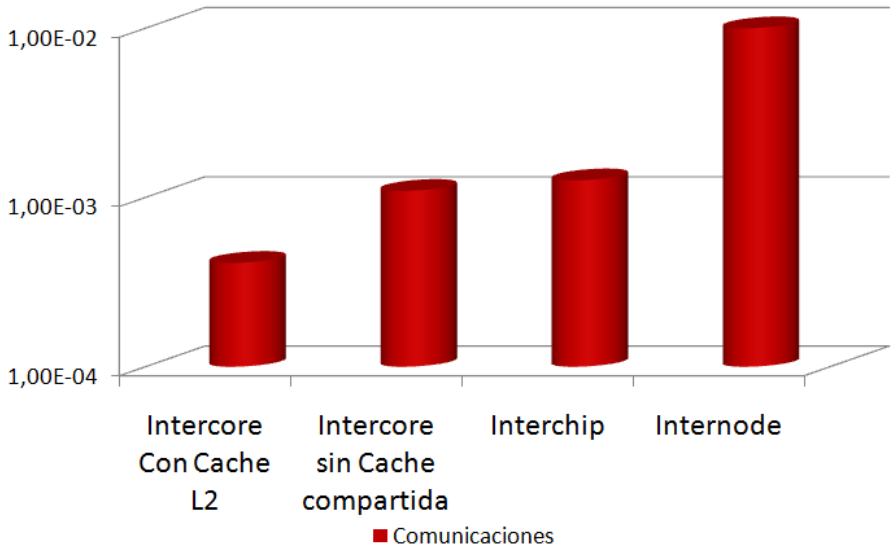


Figura 8: Tiempo de comunicación en segundo para un mensaje de 128 KByte

Igualmente, con el ejemplo queda demostrada la heterogeneidad del sistema de comunicaciones que presenta este tipo de clúster con procesadores multicore. La gran diferencia en los tiempos de latencia de cada enlace puede ocasionar tiempos de espera, específicamente en

aplicaciones sincronizadas. Este problema de comunicaciones debe ser trabajado cuidadosamente, debido a que puede generar grandes ineficiencias para la aplicación.

Otro elemento importante que los programadores paralelos deben considerar al utilizar los clúster con procesadores multicore, es que muchas de las aplicaciones tradicionales bajo paso de mensaje han sido diseñadas para ser ejecutadas en clúster compuestos por nodos con un procesador monocore, donde la comunicación interna no estaba presente. Sin embargo, cuando estas mismas aplicaciones son ejecutadas en clústeres con procesadores multicore, los procesos MPI deben intercambiar información con otros procesos que pueden o no estar localizados en el mismo nodo. Este intercambio de información entre procesos, puede ocasionar un gran desbalanceo por las distintas velocidades y latencias de los enlaces de comunicación, que al final crea retardos en la ejecución paralela [16][34][49].

Adicionalmente, para ejecutar una aplicación eficientemente en estos entornos jerárquicos de comunicación, se debe considerar el paradigma de programación paralelo con el que ha sido diseñada la aplicación. Buyya [7] precisa algunos ejemplos de paradigmas paralelos como el *master/worker (M/W)*, el SPMD, el *pipeline*, el divide y vencerás, etc. Cada uno de estos paradigmas se diferencia entre sí debido a que tienen un comportamiento y un patrón de comunicación definido. Estos patrones dentro de la aplicación paralela deben ser caracterizados y evaluados con el fin de ajustar la aplicación lo mejor posible al entorno de ejecución paralelo a utilizar.

Hemos centrado el trabajo en aplicaciones paralelas que trabajan con el paradigma de programación paralelo SPMD con el objetivo de lograr una ejecución eficiente en un ambiente jerárquico de comunicaciones. Específicamente nos concentramos en aplicaciones diseñadas con librerías de paso de mensaje, con alto nivel de sincronismo a través de las dependencias de los *tiles*[3] y con grandes volúmenes de comunicación entre los mismos.

[3] Tiles: Es una unidad de trabajo que tiene dependencia de cálculo con elementos vecinos.

El paradigma SPMD se caracteriza por ejecutar el mismo programa para un conjunto diferente de datos (*tiles*) [7][16][17][24]. Este comportamiento origina que al ejecutar una aplicación en un ambiente multicore, la misma pueda presentar retrasos por el sincronismo propio de los *tiles*. Estos *tiles* pueden verse afectados seriamente por los diferentes niveles de comunicaciones que introducen los procesadores multicore, debido a que los mismos incluyen diferencias con respecto a las velocidades y las latencias de comunicación. Las diferencias ocasionan que se deba esperar hasta que el enlace más lento comunique su información para comenzar la siguiente iteración de la ejecución.

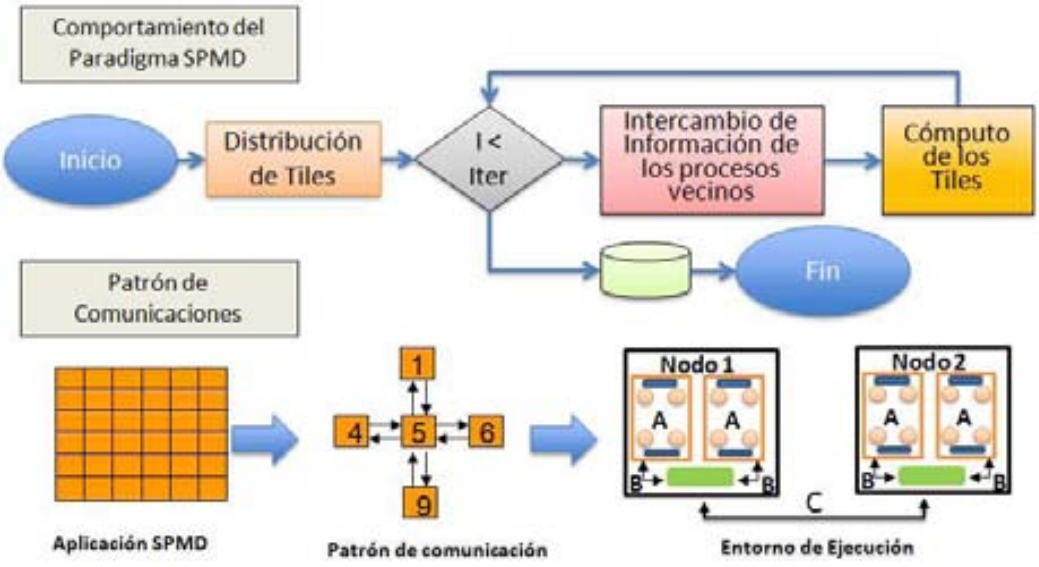


Figura 9: Comportamiento del paradigma Single Program Multiple Data (SPMD)

En la Figura 9 se detalla el comportamiento del paradigma SPMD. Inicialmente en las aplicaciones SPMD existe una distribución de los *tiles* o tareas de la aplicación y luego un conjunto de instrucciones que se ejecutan un número determinado de veces. Para cada iteración existirá un intercambio de información necesaria dentro de los diversos procesos MPI [7]. Este intercambio de información (a través de mensajes de comunicación entre procesos MPI) permite entregar la información necesaria para realizar el proceso de cómputo de la siguiente iteración de la aplicación. Esta comunicación se realiza de acuerdo al patrón de comunicación de la

aplicación durante toda la ejecución de la misma (un ejemplo para un patrón determinado se muestra en la Figura 9).

Debido a este intercambio de información, tal y como se ha visto se debe tomar en consideración que la aplicación será ejecutada en un entorno donde existen diversos niveles de comunicación. Por lo tanto, el patrón de comunicaciones entre procesos y *tiles* debe ser considerado dentro de la arquitectura de ejecución. No obstante, al ejecutar una aplicación SPMD en un entorno jerárquico de comunicaciones, los mensajes dentro de la aplicación tendrán un tamaño similar, pero el tiempo de realizar el envío será variable dependiendo del enlace de comunicación utilizado para la comunicación [11][14][53].

En este sentido, los programadores de aplicaciones SPMD deben considerar las heterogeneidades de las comunicaciones de los nuevos entornos de cómputo en HPC, ya que este proceso se puede realizar por distintos enlaces. Las diferencias entre los enlaces crean tiempos de espera, principalmente generados por el comportamiento de sincronismo de este tipo de aplicaciones.

Por lo tanto, si una aplicación SPMD incluye estos tiempos de espera dentro de la ejecución, los mismos serán repetidos en cada iteración de la aplicación, ocasionando que las métricas de rendimiento tales como el tiempo de ejecución, el *speedup*, la eficiencia y la escalabilidad se vean seriamente afectadas por el manejo inadecuado de la heterogeneidad de las comunicaciones.

Sin embargo, estos problemas de comunicación creados por el sistema jerárquico de comunicaciones, pueden ser gestionados de forma eficiente usando estrategias de balanceo de carga, planificación espacial (*mapping*) y planificación temporal (*scheduling*) con el objetivo de mejorar las métricas de la ejecución paralela [37]. Bajo este enfoque, el presente trabajo de investigación plantea dar una metodología que busca para cada aplicación SPMD la solución adecuada en entornos jerárquicos de comunicaciones, que permita mejorar la ejecución considerando los parámetros de

rendimiento tales como el tiempo de ejecución, el *speedup*, la eficiencia y la escalabilidad.

El tipo de aplicaciones SPMD seleccionadas para el desarrollo de esta investigación deben cumplir con ciertas características. En primer lugar, deben ser estáticas para determinar hacia qué procesos MPI se comunicarán los *tiles* y este patrón debe mantenerse durante toda la ejecución. Asimismo, las aplicaciones deben ser locales, debido a que solamente se han considerado aplicaciones cuya comunicación es punto a punto. Además, el patrón de comportamiento de las aplicaciones SPMD debe ser regular; es decir, se debe intercambiar el mismo volumen de información por iteración. Al mismo tiempo, las aplicaciones utilizadas deben estar compuestas por un conjunto de *tiles* o bloques de cómputo con distribuciones de una, dos ó tres dimensiones.

Las aplicaciones seleccionadas presentan un sincronismo a nivel de la dependencia de información de los *tiles*, por lo que no incluimos sincronismo a nivel de proceso como ha sido establecido en otros modelos como el BSP (*bulk-synchronous parallel*) definido por Valiant en los años 1993, 2011 respectivamente [56][57]. Estas características conceden a la aplicación cierto nivel de determinismo que permite aplicar un conjunto de estrategias para mejorar las métricas de rendimiento.

En este sentido, existe un conjunto de *benchmarks* y aplicaciones que cumplen con las características definidas anteriormente. Un ejemplo de estos *benchmarks* son los NAS *parallel benchmark* en algunas de sus diferentes versiones, CG, BT, MG, SP, [58]. Entre las aplicaciones, se pueden mencionar aplicaciones reales como: la transferencia de calor [64], la ecuación de Laplace, las aplicaciones para el cálculo de ondas [L13], las aplicaciones de dinámicas de fluidos como la suite Mpbl para resolver problemas de Lattize Bolztman [54][66], entre otras. Todas las aplicaciones mencionadas están seriamente afectadas por el desequilibrio que generan los enlaces de comunicación dentro de los clústeres con procesadores multicore.

Una vez presentada la problemática y las características principales de las aplicaciones SPMD utilizadas, la Figura 10 muestra un ejemplo de cómo se puede visualizar el problema de ejecutar una aplicación SPMD en un clúster con procesadores multicore. La figura ilustra las diferencias de cada uno de los enlaces de comunicación y permite observar los momentos de ineficiencia, que ocasionan que la aplicación SPMD empeore sus métricas de rendimiento. No obstante, este tiempo de espera puede ser gestionado aplicando estrategias que permitan mantener a los cores con suficiente carga de trabajo mientras se efectúa el proceso de comunicación entre los *tiles* vecinos. Una de estas estrategias está enfocada en un proceso de solapamiento entre el cómputo y la comunicación, que permita ocultar los efectos de las comunicaciones y por ende mejorar las métricas de rendimiento.

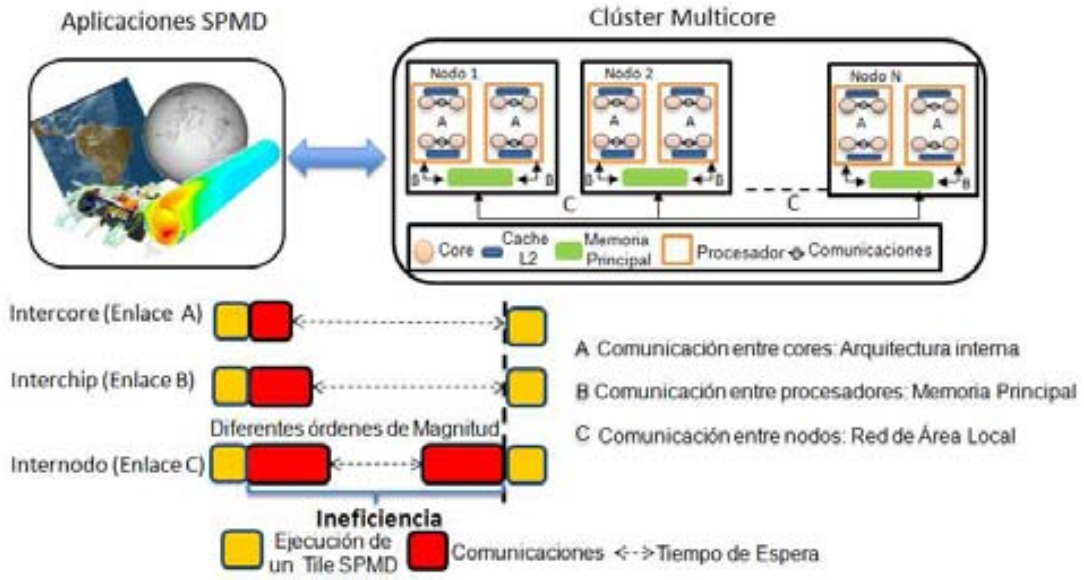


Figura 10: Aplicación SPMD ejecutada en un clúster multicore.

Como puede visualizarse en la Figura 10, cada *tile* se ejecuta en un tiempo de cómputo similar debido a la homogeneidad de los cores, pero el tiempo de comunicación de los *tiles* es diferente aunque su volumen de comunicación sea el mismo; este fenómeno es motivado a los diferentes enlaces presentes en la estructura jerárquica de comunicaciones y por la localización de los procesos MPI en el clúster. Además, se puede observar en la Figura 10 los tiempos de ineficiencia creados por los diferentes enlaces

de comunicaciones que en algunos casos puede alcanzar diferencias que oscilan alrededor de un orden y medio de magnitud para el mismo volumen de datos, si utilizan diferentes enlaces de comunicación.

Por lo tanto, considerando el problema de la heterogeneidad de comunicaciones que incluyen los clústeres multicore al ejecutar las aplicaciones SPMD, la presente tesis está orientada en desarrollar una metodología que permita detectar y gestionar las ineficiencias de las aplicaciones SPMD en entornos jerárquicos de comunicaciones (clúster con procesadores multicore) usando estrategias de *mapping* y de *scheduling*, con el objetivo de encontrar el máximo *speedup* mientras la eficiencia se mantiene por encima de un umbral definido por el usuario.

Este trabajo de investigación se enfoca en encontrar el número ideal de *tiles* que deben asignarse a cada core de manera analítica y además, en determinar el número ideal de cores que permite lograr una adaptación de la aplicación al entorno (escalabilidad) con el fin de alcanzar el máximo *speedup* con una eficiencia definida.

1.2. Objetivos

El objetivo principal de esta investigación es crear una metodología para la ejecución eficiente de aplicaciones SPMD en clústeres con procesadores multicore, que permita alcanzar el máximo *speedup* mientras la eficiencia se mantiene por encima de un umbral definido. Para lograr una ejecución eficiente la metodología puede ser utilizada para los siguientes propósitos:

a) Determinar el tamaño ideal del bloque de datos que se deben asignar a cada core, para mantener la relación de máxima aceleración y eficiencia (relación *speedup* y eficiencia).
b) Calcular el número de cores con el que se alcanza el máximo *speedup* mientras la eficiencia se mantiene por encima de un umbral definido.
c) Determinar el máximo número de cores que mantienen la relación de eficiencia y escalabilidad dado un tamaño de problema SPMD fijo aumentando el número de cores (escalabilidad fuerte).

d) Determinar cómo una aplicación se puede adaptar lo mejor posible a los recursos de cómputo disponibles.

Con el fin de que la metodología considere la gestión eficiente de los enlaces de comunicaciones del entorno multicore, se plantea un conjunto de objetivos específicos:

- Diseñar una estrategia de caracterización que permita analizar y obtener los parámetros que definen o describen el comportamiento de la aplicación en un entorno jerárquico de comunicaciones.
- Evaluar y determinar los parámetros necesarios de la aplicación paralela SPMD con el objetivo de acoplarla lo mejor posible al entorno de ejecución.
- Desarrollar un modelo analítico basado en los parámetros de caracterización para determinar el número ideal de cores y el tamaño del conjunto de *tiles* que deben ser asignados a cada core, con el objetivo de mantener una estrecha relación entre el máximo *speedup*, la máxima escalabilidad y la eficiencia definida.
- Diseñar una estrategia de planificación espacial (*mapping*) que permita minimizar los efectos de las latencias de los diferentes enlaces de comunicaciones.
- Desarrollar una política de solapamiento que mejore las prestaciones de la aplicación paralela.
- Planificar el orden de ejecución de los *tiles* de la aplicación SPMD en un ambiente jerárquico de comunicaciones.
- Evaluar el rendimiento de la aplicación paralela usando la metodología, para verificar el funcionamiento de la misma.

Estos objetivos específicos están integrados en el desarrollo de una nueva metodología. Las distintas fases de la metodología buscan gestionar eficientemente las comunicaciones y por ende mejoran las métricas de rendimiento como el tiempo de ejecución, el *speedup*, la escalabilidad, respetando un nivel de eficiencia del sistema.

1.3. Marco de trabajo

La presente investigación está orientada al desarrollo de una nueva metodología que permita la ejecución eficiente de aplicaciones SPMD en entornos jerárquicos de comunicaciones. Esta metodología permitirá que las aplicaciones SPMD desarrolladas solamente con primitivas de comunicación de paso de mensajes (MPI), puedan adaptarse considerando los diferentes niveles de comunicaciones de tal modo que se pueda realizar una ejecución de la aplicación de manera rápida y eficiente.

El principal enfoque de ésta metodología consiste en alcanzar el máximo *speedup* mientras la eficiencia se mantiene por encima de un umbral definido por el usuario. Para lograr esta meta, la metodología desarrolla un conjunto de pasos que se deben seguir, considerando tanto el entorno de ejecución como la aplicación SPMD que quiere ser ejecutada.

En este sentido, la metodología busca obtener dos elementos claves para lograr la ejecución eficiente. En primer lugar, se desarrolla una estrategia de agrupación de *tiles* que permita solapar el cómputo y las comunicaciones, con el fin de aprovechar el tiempo de espera generado por las comunicaciones. El segundo elemento es determinar el número ideal de cores que son necesarios para mantener una relación entre la eficiencia y el *speedup*.

La Figura 11 muestra una idea general de cómo una aplicación SPMD está compuesta por un conjunto de *tiles*, y cómo estos deben ser agrupados con el objetivo de aplicar una estrategia para minimizar los efectos de las comunicaciones. Este agrupamiento permite aplicar una estrategia de solapamiento donde se pueden ejecutar los *tiles* de la siguiente manera: primero se ejecutan los *tiles* de bordes, para permitir el solapamiento entre las comunicaciones de los *tiles* de bordes y el cómputo de los *tiles* internos. Esta estrategia de solapamiento permite esconder los efectos de las comunicaciones más lentas, aprovechando la capacidad de cómputo de los cores que pueden ejecutar otros *tiles* mientras las comunicaciones de los tiles de borde se están realizando.

Para obtener una relación aceptable entre el *speedup*, la escalabilidad y la eficiencia, se desarrolla un modelo analítico que permite determinar el número de *tiles* necesarios, los cuales serán asignados a cada core, con el objetivo de mantener la relación del *speedup* y la eficiencia. Este modelo determina el número de cores necesarios que marcan la máxima escalabilidad bajo una eficiencia definida.

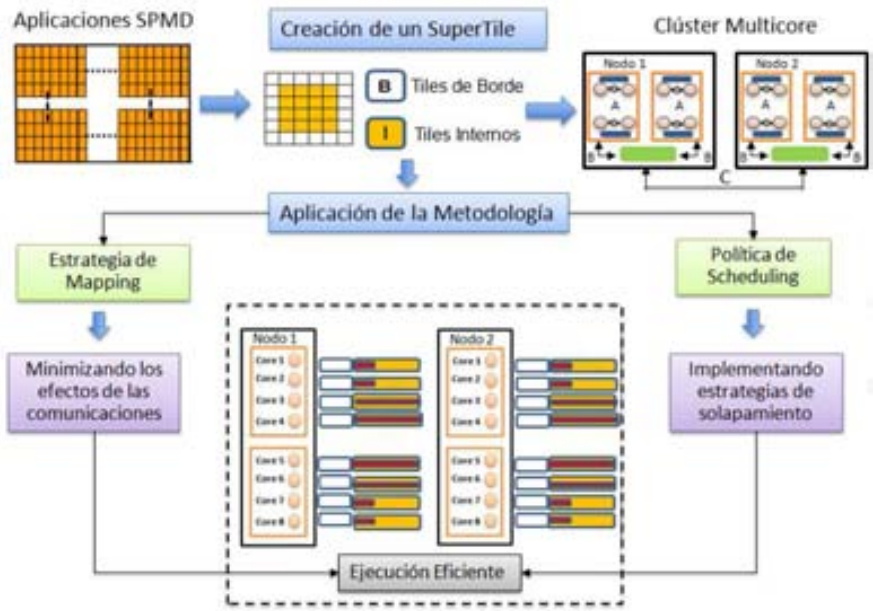


Figura 11: Características de la metodología para ejecución eficiente en entornos jerárquicos de comunicación

Como se detalla en la Figura 12, la metodología agrupa un conjunto de *tiles* de la aplicación en grupos llamados *supertile.* Consideramos un *supertile* a la agrupación de *tiles* que será asignado a cada uno de los elementos de procesamiento (en este caso un Core). Estos *supertiles* dividen los *tiles* en dos tipos: *tiles* de borde y *tiles* internos. En los *tiles* internos el proceso de comunicación se realiza dentro del mismo core y en el caso de los *tiles* de borde deben comunicar con otros *tiles* que están ubicados en otros cores. Esta división de *tiles* nos permite aplicar la estrategia de solapamiento, que busca ejecutar los *tiles* internos mientras se ejecutan las comunicaciones de los *tiles* de bordes.

Para minimizar los efectos de las diferentes velocidades de los enlaces de comunicación y para diseñar ésta estrategia de solapamiento, la metodología se compone de cuatro fases: caracterización, modelo de distribución de *tiles*, *mapping* y *scheduling* (Ver Figura 12).

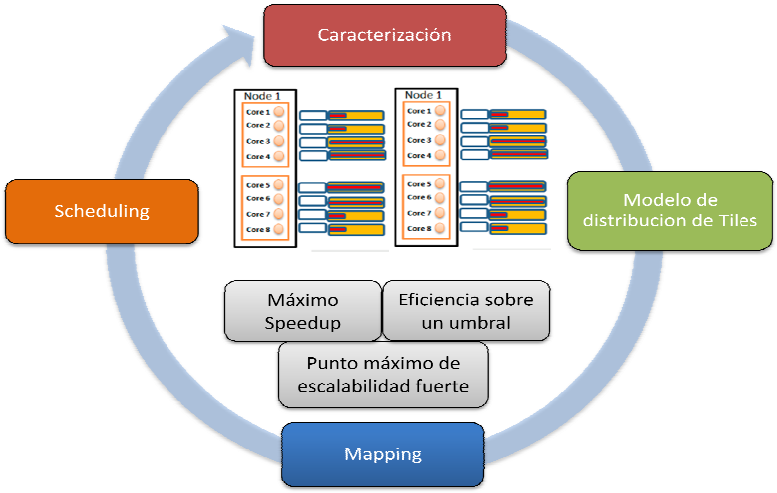


Figura 12: Descripción general de la metodología

En la fase de caracterización se realiza un análisis de la aplicación y del entorno de ejecución con el objetivo de encontrar los parámetros de cómputo y comunicación necesarios para el cálculo del modelo analítico de la siguiente fase. El análisis de esta fase está compuesta por un estudio del comportamiento de la aplicación SPMD, donde se pueden encontrar parámetros relacionados con: el tamaño del problema, el volumen de comunicación por tile, el número de iteraciones, entre otros. Adicionalmente, se debe caracterizar el entorno paralelo, desde donde se extraen los valores de los parámetros para determinar el tiempo de cómputo, el tiempo de comunicación, el número de cores por nodo, los niveles de comunicación, etc, y finalmente los valores definidos por el usuario, donde se define la eficiencia deseada.

Con los parámetros obtenidos en la fase de caracterización, se calcula el modelo de distribución de *tiles*, el cual permite determinar el número de *tiles* que deben ser asignados a cada core (*supertile*) y el número de cores que mantienen una estrecha relación entre la eficiencia definida y el máximo

speedup. Una vez obtenido los valores ideales, el siguiente paso es la aplicación del *mapping* donde los *tiles* son ubicados entre los cores de ejecución. Este paso se realiza utilizando un método de afinidad de procesos MPI a los cores y aplicando un modelo lógico de distribución, que permite minimizar los efectos de las comunicaciones en los entornos con procesadores multicore.

La última fase, tiene dos funciones principales, una de ellas es determinar el orden de ejecución, el cual establece cómo se ejecutan los *tiles* dentro del core y la segunda es el control del proceso de solapamiento. La integración de todas estas fases demuestra que la metodología mejora el rendimiento de las aplicaciones SPMD como se mostrará en la validación experimental realizada y queda reflejado en el capítulo de experimentación.

1.4. Organización de la tesis

En este capítulo se presentó el enfoque general de la investigación, describiendo la problemática que se presenta en el rendimiento al ejecutar aplicaciones SPMD en un ambiente jerárquico de comunicaciones específicamente en un entorno de clúster con procesadores multicore. Asimismo, se describió brevemente la metodología propuesta. Para la explicación del desarrollo de la metodología, este trabajo estará organizado de la siguiente manera:

En el capítulo 2 se describen las bases teóricas que fundamentan el desarrollo de la investigación, así como los trabajos relacionados. Se describen los entornos con procesadores multicore, seguido de los paradigmas de programación paralela, haciendo énfasis en el paradigma SPMD. Luego se describen los principales problemas de las aplicaciones SPMD cuando son ejecutadas en clústeres con procesadores multicore. Dentro de estos problemas se analizan los desequilibrios tanto a nivel de cómputo como a nivel de comunicaciones, ocasionados por el sistema jerárquico de comunicaciones. Seguidamente, se explica cómo a partir de la problemática presente se pueden generar oportunidades que permitan realizar mejoras en las métricas de rendimiento de la aplicación paralela.

En el Capítulo 3 se presenta la metodología de ejecución eficiente para ambientes jerárquicos de comunicaciones, realizando una descripción detallada de cada fase de la metodología, con el objetivo de explicar cómo se puede obtener el máximo *speedup* mientras la eficiencia está por encima del umbral definido.

Por otra parte, la aplicabilidad de la metodología se muestra en el capítulo 4, el cual describe el método utilizado para aplicar la metodología. Para esto, se han utilizado varias aplicaciones y diferentes entornos con procesadores multicore, con el fin de mejorar su rendimiento paralelo (en algunos casos pueden llegar hasta un 39% en eficiencia al aplicar la metodología). Las mejoras presentadas en éste capítulo, se han diseñado de acuerdo a las fases de la metodología presentadas en el capítulo 3.

En el capítulo 5, se estudia cómo se puede aplicar la metodología para hacer una combinación de dos métricas de rendimiento como lo son la escalabilidad de la aplicación y la eficiencia. En este sentido, se ha utilizado por separado el concepto de escalabilidad fuerte y débil con la finalidad de encontrar una relación ideal entre el máximo punto de escalabilidad de la aplicación con la eficiencia definida. Asimismo, se describe con ejemplos teóricos y prácticos, cómo ambas métricas pueden ser combinadas para alcanzar el máximo punto de escalabilidad fuerte de las aplicaciones utilizadas.

Finalmente en el capítulo 6, se explican las conclusiones generales y se exponen las principales contribuciones de esta investigación para la comunidad científica. Además, se presentan cuáles son los trabajos futuros y las líneas abiertas que pueden ser direccionadas para mejorar el rendimiento en clústeres con procesadores multicore heterogéneos en cómputo, así como en otros sistemas de comunicaciones jerárquicas como los entornos multicluster.

Capítulo 2

Prestaciones de aplicaciones SPMD en entornos multicore. Conceptos y trabajos relacionados

Resumen

Este capítulo se centra en presentar las bases teóricas necesarias para el desarrollo de la metodología de ejecución eficiente en entornos jerárquicos de comunicaciones. Inicialmente, se realiza una introducción general de los entornos jerárquicos de comunicaciones específicamente en los clúster compuestos por nodos de múltiples núcleos (multicore). Asimismo, se describen los conceptos claves de los paradigmas de programación paralelos, y específicamente se profundizará en el paradigma Single Program Multiple Data (SPMD). A continuación se enfatizará en el estudio de las métricas de rendimiento y cómo las mismas pueden ser afectadas cuando se combina la ejecución de una aplicación SPMD en clústeres multicore. Al final de este capítulo se comparan algunos trabajos relacionados en el ámbito de estudio.

2.1 Introducción

El tiempo de cómputo de una aplicación está relacionado al tiempo que el computador requiere en ejecutar un conjunto de instrucciones básicas y al número de instrucciones concurrentes que el sistema de procesamiento puede realizar [33]. Es por ello que la eficiencia y el tiempo de ejecución de las aplicaciones se ven afectadas cuando existen nuevos modelos a nivel de la arquitectura de ejecución, debido a que la aplicación no ha sido inicialmente diseñada para manejar las mejores prestaciones de ese nuevo sistema de cómputo.

En el ámbito de sistemas de cómputo paralelo, la principal estrategia para la ejecución de aplicaciones de forma rápida y eficiente, consiste en incluir un conjunto de técnicas para proporcionar simultaneidad al procesamiento de información, con el fin de aprovechar la velocidad de cómputo del sistema. Foster define el procesamiento paralelo como "…el conjunto de procesadores que están activos para trabajar cooperativamente para solventar un problema computacional… [pág. 3]"[22]. Esta definición establecer que las aplicaciones que utilizan el procesamiento paralelo pueden tener concurrencia, por lo que se divide la carga de trabajo a cada nodo de cómputo y ésta carga de trabajo debe ser procesada simultáneamente para conseguir un menor tiempo de ejecución de la aplicación.

Para aprovechar al máximo el poder de procesamiento de un computador con arquitectura paralela, es necesario utilizar técnicas de programación paralela, que permitan definir cómo los procesos se ejecutarán en forma concurrente. Las técnicas de programación paralela permiten que los problemas de las aplicaciones puedan ser divididos en eventos más pequeños, generando así una mayor cantidad de eventos que pueden ser ejecutados en forma paralela.

El principal beneficio que establece el procesamiento paralelo es la reducción del tiempo computacional de la aplicación. Sin embargo, las métricas de rendimiento como la escalabilidad, *speedup*, eficiencia, entre otros, se pueden ver afectadas, debido a la disputa de los recursos que son

compartidos (red, memoria compartida, dispositivos de almacenamiento, entre otros). Este problema se incrementa cuando la aplicación paralela ha sido diseñada para ser ejecutada inicialmente en un clúster con procesadores *monocore* y luego por la evolución de los sistemas de cómputo paralelo, la mismas son ejecutada en un sistema jerárquico de comunicaciones como en el caso de los clústeres con procesadores multicore, donde la comunicación entre procesos de paso de mensaje (MPI) se realizan por diferentes rutas de comunicación (red, memoria principal, memorias cache, etc.) y originalmente estas aplicaciones paralelas no han sido pensadas para tener comunicaciones dentro del nodo.

Por esta razón, la ejecución eficiente de aplicaciones desarrolladas utilizando sólo librerías de paso de mensaje en estos entornos de comunicaciones heterogéneas es un gran desafío; más aún, cuando se utilizan aplicaciones paralelas que tienen gran intercambio de comunicación dentro de su proceso de ejecución como es el caso de las aplicaciones SPMD. Es por esto, que al mapear una aplicación SPMD en estos entornos se deben considerar las variaciones de ancho de banda, latencia y velocidad que presentan los diversos enlaces de comunicaciones, con el objetivo de disminuir los efectos de degradación que éstos pueden incluir en el proceso de comunicación de la aplicación y los cúales afectan a las métricas de rendimiento.

En este sentido, el presente capítulo describe las bases teóricas utilizadas para el desarrollo de la metodología de ejecución eficiente de aplicaciones SPMD en sistemas jerárquicos de comunicaciones, como es el presentado en los clústeres con procesadores multicore.

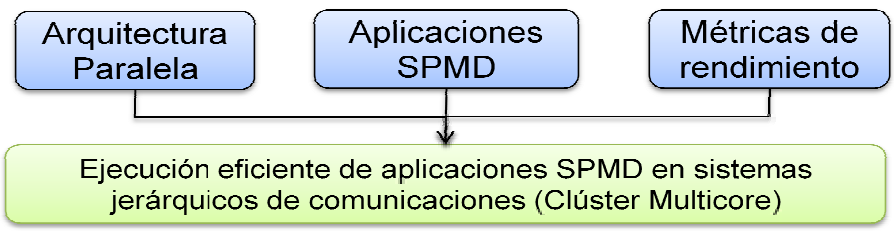


Figura 13 Descripción de desarrollo del estado del arte

La Figura 13 muestra los tres principales tópicos a considerar en este capítulo. Inicialmente se describirá la arquitectura paralela, profundizando en los sistemas jerárquicos de comunicaciones, específicamente en los clúster multicore y se visualizarán los problemas que generan las diversas rutas de comunicación tanto a nivel de cómputo como a nivel de comunicaciones. Luego, se describirán los paradigmas de programación paralela haciendo énfasis en el paradigma de estudio de este trabajo (paradigma SPMD). Seguido, se describe el comportamiento de las aplicaciones SPMD de paso de mensajes ejecutados en clústeres con procesadores multicore. En esta parte se describen los problemas del desequilibrio de comunicaciones y las influencias en las métricas de rendimiento al ejecutar una aplicación SPMD en un clúster con procesadores multicore.

Finalmente, en este capítulo se describen las investigaciones en el área de estudio y los aportes que las mismas ofrecen al desarrollo de la metodología.

2.2 Arquitectura paralelas

Las arquitecturas paralelas han emergido de las arquitecturas de cómputo secuenciales tradicionales. Este modelo consiste en una unidad central de procesamiento y una memoria. Las máquinas de este modelo ejecutan una única secuencia de instrucciones, por lo que su velocidad de ejecución está limitada por la capacidad de ejecución de instrucciones y por la capacidad de intercambio entre la memoria y el procesador [33] [59].

Sin embargo, las aplicaciones científicas actuales requieren gran capacidad para hacer sus cómputos, lo que conlleva a un mayor tiempo de ejecución en un entorno secuencial. Por esta razón, no es viable utilizar el modelo de Von Neumann y es necesario aplicar otras estrategias tecnológicas que permitan minimizar los tiempos de ejecución.

Una de estas soluciones paralelas es emplear un sistema denominado supercomputador, que consiste en un sistema con gran capacidad de cómputo a alta velocidad. Sin embargo, es una solución que resulta muy cara y difícil de amortizar. Otra alternativa son los clústeres de cómputo.

Según Buyya [7], un clúster de cómputo, es un tipo de sistema de procesamiento paralelo o distribuido, compuesto por un conjunto de nodos que ejecutan una o varias aplicaciones de forma simultánea, con el objetivo de ejecutarlas en menores tiempos de cómputo.

Un clúster permite ejecutar una serie de aplicaciones aumentando el rendimiento y las prestaciones de las aplicaciones paralelas. Los nodos que integran un clúster, pueden ser configurados con arquitecturas monoprocesador (un sólo procesador por nodo); multiprocesador (integra más de un procesador por nodo); multicore (integran varios núcleos por procesador) o multiprocesador multicore (es la combinación de varios procesadores con múltiples núcleos en un mismo nodo). Un ejemplo de estas arquitecturas en mostrado en la Figura 14.

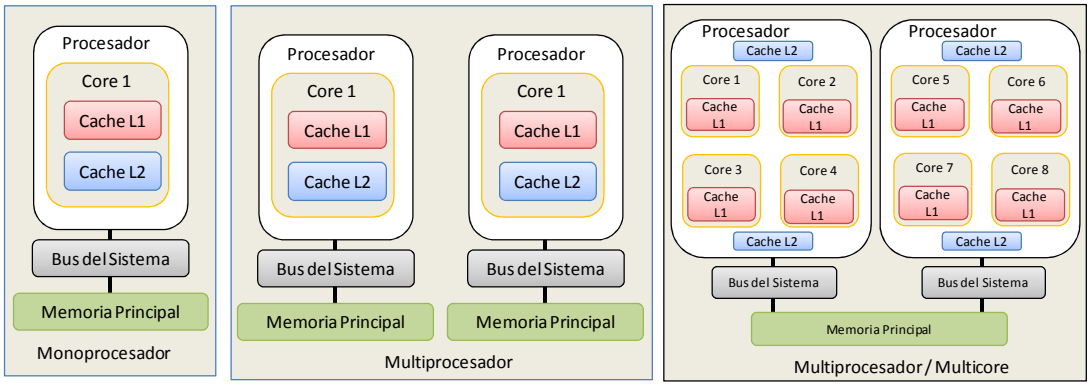


Figura 14 Tres configuraciones de arquitecturas

Actualmente, la mayoría de los sistemas de clúster de cómputo están configurados con arquitecturas multiprocesador, multicore o multiprocesador multicore. Esto es evidenciado en la lista del top500, donde la gran mayoría de los sistemas de cómputo están compuestos por nodos multiprocesadores multicore [L12] .

2.2.1 Arquitectura Multiprocesador

Una arquitectura multiprocesador como se ha visto consiste en la unión de un conjunto de procesadores dentro de un mismo sistema de forma paralela para obtener una mayor capacidad de cálculo [18]. Esto, permite al sistema ejecutar mayor cantidad de trabajo en el mismo tiempo, mejorando

las métricas de rendimiento de las aplicaciones. En este tipo de arquitectura se debe gestionar la distribución de la memoria así como la carga de trabajo entre los procesadores que integran la arquitectura.

Las principales ventajas del uso de un sistema multiprocesador es la reducción de los costos, debido a que los procesadores comparten recursos tales como la memoria, el sistema de energía, la tarjeta madre, etc,. Por otra parte, estos sistemas permiten mejorar la disponibilidad debido a que se dispone de una redundancia intrínseca de componentes, por lo que si un procesador falla el otro puede continuar ejecutando la aplicación, aunque con menores prestaciones.

Los sistemas de multiprocesamiento han evolucionado a tal nivel que los sistemas actuales incluyen varios procesadores por nodo y cada procesador contiene un conjunto de cores. Esta integración permite que las aplicaciones paralelas puedan incluir mayor nivel de paralelismo aunque se presente problemas entre las comunicaciones de los procesos.

Para este trabajo consideramos la última arquitectura de la Figura 14 la cual está basada en los sistemas multiprocesadores con múltiple núcleos. Esta arquitectura se explicará a continuación con mayor detalle.

2.2.2 Arquitectura multicore

La arquitectura multicore se puede definir como la integración de dos o más recursos computacionales (cores) residiendo dentro de una interface o procesador [11] [18] . Esta integración fue principalmente conceptualizada para incluir dentro del procesador la capacidad de hacer paralelismo. La principal ventaja de este concepto es mejorar las prestaciones de las aplicaciones [45][49]. El uso de la arquitectura multicore permite tener ejecuciones más rápidas de las aplicaciones por lo que las métricas de rendimiento se ven mejoradas considerablemente. No obstante, para obtener este rendimiento, se debe tener en cuenta ciertas consideraciones en el momento de diseñar aplicaciones paralelas.

Algunas de estos aspectos están relacionados con: el número de cores por procesador, la localidad de los datos, diseño del procesador, manejo de

la memoria compartida, la cantidad de buses de interconexión y una de las más importante es el sistema jerárquico de comunicaciones (afecta seriamente las aplicaciones paralelas con alto volumen de comunicación como se presenta en las aplicaciones SPMD). Estas dificultades deben ser gestionadas para hacer un uso eficiente de la arquitectura.

Existen varias alternativas a nivel de diseño de los nodos con procesadores multicore, donde puede variar la organización de la jerarquía de las memorias caches y la forma de acceso a la memoria. Un ejemplo relacionado a la organización de la jerarquía de las memoria caches se ilustra en la Figura 15, en este ejemplo se puede detallar que existen arquitecturas que tienen cache L2 individual por cada core y por otro lado se tiene la misma cantidad de cores pero usando a su vez una cache L2 compartida.

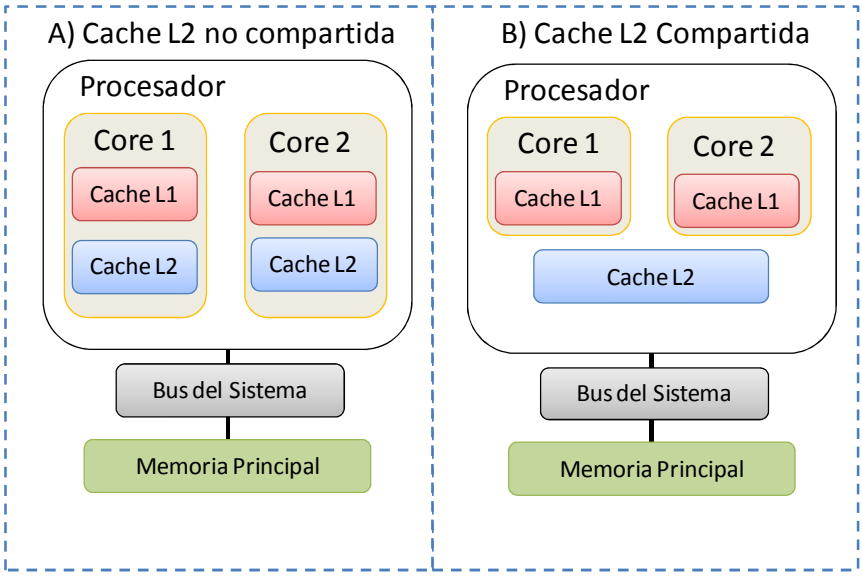


Figura 15 A) Procesador Dual Core con memoria cache individual, B) Procesador Dual Core con memoria cache compartida

Esta diferencia en el diseño origina que el conjunto de aplicaciones paralelas desarrolladas con librerías de pasos de mensajes como: MPI [L9], PVM [L11], entre otros, y que a su vez tengan un elevado nivel de sincronismo y comunicaciones, se vean afectadas por los diferentes tiempos de latencias que generan las diversas rutas de comunicaciones del entorno

(comunicación a través de la Memoria Cache, Memoria Principal, o por la red en el caso de usar varios nodos) [31].

Otra característica que afecta el sistema de jerarquía de comunicaciones entre los clúster con procesadores multicore, es el modelo de acceso a la memoria de los nodos. En este sentido, se mencionan dos enfoques: los procesadores que están diseñados utilizando una arquitectura NUMA (*Non-Uniform Memory Access*) [11][27] y los que utilizan la arquitectura UMA (*Uniform Memory Access*)[12]. En el caso de una arquitectura NUMA (Figura 16) el sistema de memoria es individual por cada uno de los procesadores que integran la arquitectura, pero a su vez tienen acceso a los espacios de memorias del otro procesador con un tiempo de acceso superior.

Este esquema es usado en sistemas con grandes cantidades de procesadores por nodos, donde debe existir un control sobre los datos de la memoria principal [6]. En la Figura 16, se puede visualizar cómo los cuatro cores dentro de cada chip comparten su controlador de memoria y el espacio local dentro de la memoria. Además, se puede detallar que los procesadores tienen acceso a la memoria remota del otro procesador, en este caso el acceso es más penalizado con latencia que el propio acceso a la memoria local.

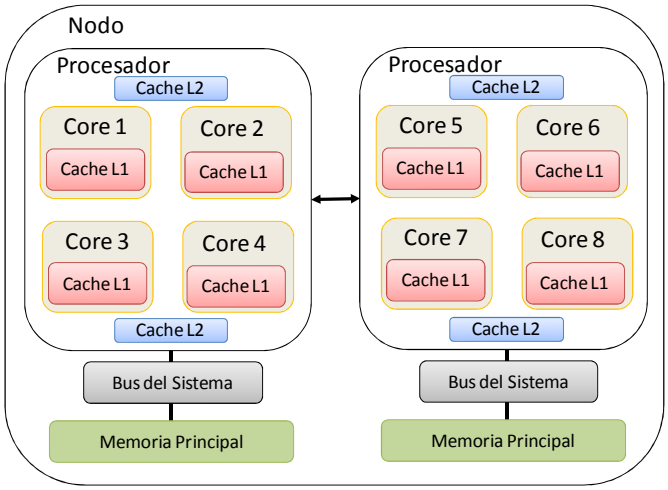


Figura 16 Nodo con doble Quad core con arquitectura NUMA

Por otra parte, en la arquitectura UMA [25], todos los cores comparten de forma uniforme el acceso a la memoria principal. Por lo tanto el tiempo de

acceso es igual para cada uno de los cores que integran la arquitectura. Un ejemplo es visualizado en la Figura 17, en la que se ilustra que la memoria principal es compartida entre todos los cores.

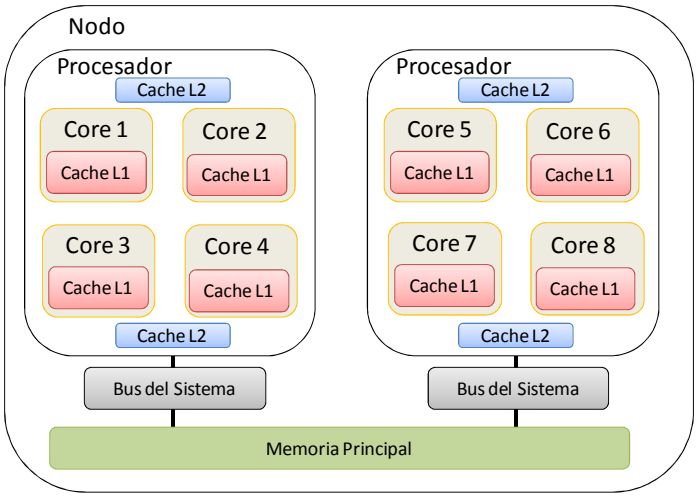


Figura 17 Nodo con doble Quad core con arquitectura UMA

El uso de nodos con procesadores multicore en la computación de altas prestaciones cada día es más frecuente, debido a la gran efectividad que tienen estos sistemas tanto a nivel de ahorro de energía, como en el caso de las mejoras de las prestaciones de las aplicaciones. Un ejemplo de un esquema clúster multicore es mostrado en la Figura 18, donde se tiene un clúster con una cantidad de nodos compuestos por procesadores multicore específicamente utilizando procesadores *quadcore*.

En este tipo de clúster la comunicación entre los cores (*intercore*) es ejecutada a través de la arquitectura interna del procesador, esta comunicación se puede dividir en dos tipos, cuando comparten la memoria cache (ejemplo comunicación entre el core 1 con el core 2) y cuando no comparten la memoria cache (ejemplo, comunicación entre los cores 1 y core 3). En este último caso la comunicación se realiza a través de la memoria principal (Figura 18). En el caso de la comunicación entre los procesadores dentro del mismo nodo, la misma es realizada utilizando la memoria común (*interchip*) y la comunicación entre dos procesadores ubicados en distintos nodos es efectuada utilizando la red de comunicaciones o red de área local (Internode).

Este sistema de jerarquía de comunicaciones impone un gran desafío a la hora de hacer diseño de aplicaciones que usen eficientemente estos clústeres multicore. Bajo este enfoque, se puede hacer una jerarquía de las comunicaciones utilizando un esquema de niveles.

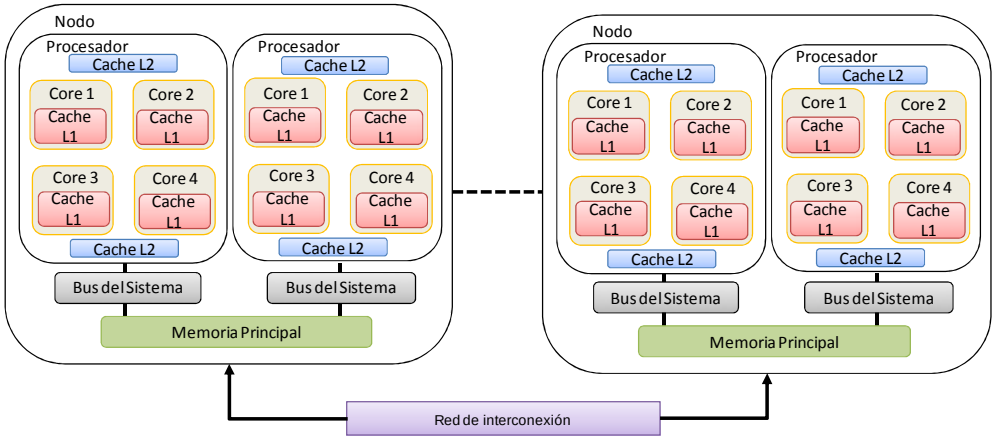


Figura 18 Clúster compuesto por varios nodos multicore

Por lo tanto, entre más cercano al procesador esté el nivel de comunicaciones, éste será más rápido (Figura 19), por ello, cuando se tiene diversas comunicaciones con velocidades y latencias distintas, las mismas deben ser identificadas para obtener las mejores prestaciones de la arquitectura de ejecución. Además con este enfoque se pueden desarrollar estrategias que permitan manejar adecuadamente las ineficiencias que generan los niveles de comunicaciones a la aplicación.

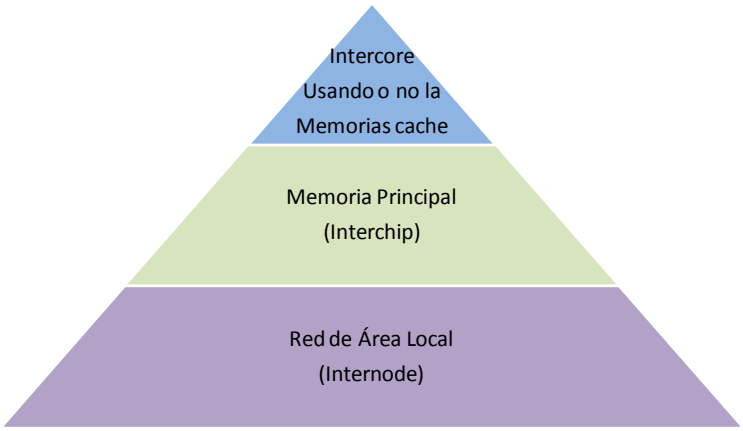


Figura 19 Niveles de comunicaciones en un clúster multicore

Aunque un clúster multicore sea homogéneo a nivel de cómputo, los diferentes niveles de comunicaciones dan un grado de heterogeneidad al

sistema, que en algunos casos afectará en mayor o menor medida dependiendo del tipo de aplicación paralela que se ejecute en el clúster.

2.3 Modelo de Programación Paralela

Los modelos de programación paralela se definen como una abstracción de la arquitectura entre el uso de hardware y la memoria [8]. Dentro de estos modelos se encuentran los dos más usados para el desarrollo de aplicaciones en el mundo paralelo, el modelo de memoria compartida, y el de paso de mensajes. En el caso de las aplicaciones basadas en memoria compartida, los procesos pueden utilizar las mismas secciones de memoria simultáneamente entre varios procesos, mientras que en el sistema de paso de mensaje la información viaja a través de paquetes de comunicación utilizando librerías específicas de paso de mensajes.

2.3.1 Modelo de memoria compartida

En el sistema de programación de memoria compartida, varios procesos pueden ejecutarse simultáneamente, utilizando un mismo espacio de área compartida de la memoria [7]. Foster [22], establece que en los sistemas con memoria central o compartida, cada proceso puede acceder a los espacios reservados, para utilizarlos en la aplicación de comandos de lectura y escritura. Los procesos se pueden comunicar a través de esta área reservada siempre y cuando se haya establecido un sector de la memoria como compartido.

Las modificaciones que se realicen en los espacios compartidos, afectan a todos los procesos que tengan acceso al mismo, por lo que este espacio de memoria debe ser declarado previamente. Al hacer una creación o subdivisión de procesos o tareas, el sistema operativo se encarga de la comunicación entre ellos, a través de entornos de variables. En este caso, la comunicación resulta sencilla; no obstante, se deben controlar ciertos aspectos relacionados con el sincronismo de acceso y comunicación de datos.

Las ventajas de uso de memoria compartida, es que permite generar espacios globales de direcciones, además de permitir compartir los datos entre los procesos, lo que origina que las tareas sean ejecutadas más rápidas y uniforme debido a la proximidad de las memorias a las unidades de procesamiento.

2.3.2 Modelo de paso de mensajes

Este modelo permite desarrollar programas paralelos eficientes para ser ejecutados en ambientes de memoria distribuida [7][61]. La principal ventaja de usar este modelo es que se encuentra implementado en la mayoría de las plataformas de cómputo paralelo. Este modelo, permite hacer un control del intercambio de información entre los procesos que ejecutan la aplicación paralela, este intercambio es realizado a través de mensajes entre los procesos. Los programas escritos en librerías de paso de mensajes pueden ser ejecutados sobre sistemas de multiprocesamiento de memoria distribuida o memoria compartida.

Las librerías más utilizadas de paso de mensaje son el MPI y el PVM; en ambas, al realizar el intercambio de información, el proceso fuente inicia la comunicación enviando un mensaje de datos utilizando una interfaz de programación de paso de mensajes, y el proceso receptor recibe la información la cual está etiquetada con el identificador del proceso que recibe la información. Los mensajes dentro de las aplicaciones pueden ser establecidos de acuerdo al tipo de sincronismo que se desee incluir al desarrollo paralelo, los mensajes pueden ser sincrónicos, asíncronos, bloqueantes, no bloqueantes, en el caso de ser un mensaje sincrónico, se necesita mantener un acuerdo entre el proceso fuente y destino.

2.4 Paradigmas de programación paralela

Los paradigmas de programación paralela son patrones que permiten desarrollar programas para entornos paralelos [7]. Cada uno de estos paradigmas define un modelo para el diseño del algoritmo que permite producir aplicaciones con ciertas directrices específicas. Los paradigmas de

programación se caracterizan por incluir una estructura para la distribución de datos y el modelo de comunicación. Estos paradigmas pueden tener diferentes comportamientos dependiendo de la arquitectura de ejecución que se utilice.

La selección del paradigma de programación se hace de acuerdo al comportamiento que tenga la aplicación y la disponibilidad de recursos de computación paralela. Dependiendo del tipo de aplicación, se debe seleccionar el paradigma acorde a las características de resolución de la misma. En la actualidad, casi todas las aplicaciones están identificadas a un paradigma de programación paralela. Algunos ejemplos de paradigmas paralelos son: Divide y Vencerás, Pipeline, Master / Worker y el SPMD (Single Program Multiple Data) [7].

2.4.1 Paradigma Divide y Vencerás

En este paradigma, la aplicación es sub-dividida en varias particiones, lo que permite que puedan ser ejecutadas independientemente; antes de finalizar la ejecución se integran todos los resultados para dar el resultado final de la aplicación. El proceso de partición y de recombinación hace uso de paralelismo, por lo que estas operaciones requieren de comunicaciones. Sin embargo, los sub-problemas son independientes, por lo que no es necesaria la comunicación entre los procesos de la aplicación. En este paradigma se realizan tres operaciones, dividir, computar y unir, por lo que la estructura del programa es un árbol, siendo los sub-problemas las hojas que ejecutarán cada uno de los procesos

2.4.2 Paradigma Pipeline.

El paralelismo en este paradigma está basado en una descomposición funcional de la aplicación, por lo que se deben identificar las tareas del algoritmo que son capaces de ser ejecutadas paralelamente, lo que origina que cada proceso ejecute una porción del algoritmo general. Esta técnica, permite aumentar la capacidad de instrucciones por procesador, debido a

que las porciones o actividades de programa, pueden ejecutarse de forma solapada [33].

Además, cada proceso ejecuta una etapa del pipeline y es responsable de una tarea individual. Los patrones de comunicación son simples y pueden hacerse de forma tanto sincrónica como asíncrona, la eficiencia de ejecución de éste paradigma es dependiente del balanceo de carga a lo largo de las etapas de pipeline.

2.4.3 Paradigma *Master/Worker*

Este paradigma está compuesto por dos entidades el Máster y los *workers*. El Máster, es el encargado de dividir y distribuir las tareas de la aplicación, y los *workers* son entidades de trabajo que reciben la información, la procesan y envían los resultados al Máster. En el caso de los *workers*, reciben un conjunto de tareas a través de mensajes del Máster, por lo que la comunicación es solo a través de estas 2 entidades. Este paradigma puede utilizar un esquema de balanceo de carga estático o dinámico [22] [61].

2.4.4 Paradigma SPMD (Single Program Multiple Data).

Este paradigma está compuesto por dos características principales, ejecuta el mismo programa en todos los procesos MPI y divide un conjunto de *tiles* que tienen dependencia para el cálculo y son asignados a cada proceso [43][55].

Estas aplicaciones tienen un comportamiento determinístico donde existe un proceso de reparto de los *tiles* y luego se ejecuta un proceso iterativo de intercambio de información. Este proceso iterativo es repetitivo y se realiza por un conjunto de iteraciones hasta que se termina la ejecución [7][22]. La Figura 20, muestra un diagrama del comportamiento de estas aplicaciones SPMD, donde se puede visualizar el comportamiento de intercambio de información entre procesos.

Este comportamiento puede variar dependiendo del tipo de aplicaciones SPMD; no obstante, este tipo de aplicaciones define un patrón de

comunicaciones, el cual es mantenido durante toda la ejecución. Los *tiles* que componen este tipo de paradigmas tienen una dependencia entre los *tiles* vecinos, por lo que se debe comunicar un conjunto de informaciones entre los *tiles* en cada iteración

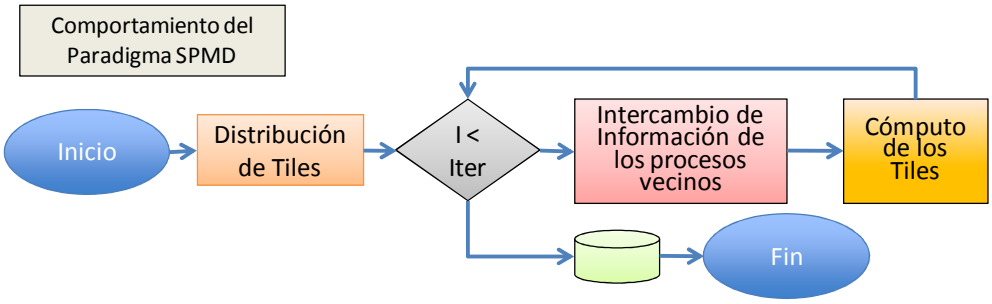


Figura 20 Comportamiento de las aplicaciones SPMD

Estas comunicaciones generan gran overhead si se utiliza un sistema jerárquico de comunicaciones. Un ejemplo se puede detallar en la Figura 21, donde una aplicación está compuesta por un conjunto de *tiles* que pueden estar distribuidos en una, dos o tres dimensiones dependiendo del propósito de la aplicación paralela.

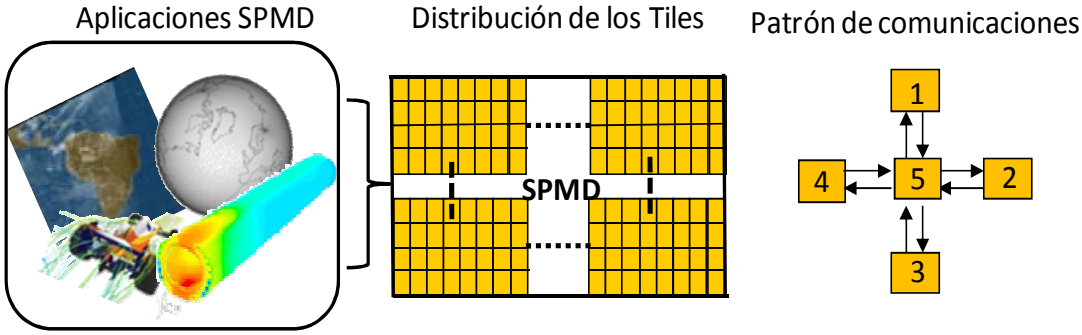


Figura 21 Tamaño del problema (*tiles*) y el patrón de comunicaciones

Esta distribución de la aplicación debe mantener un patrón de comunicaciones, el cual permite el intercambio de información entre los *tiles* que será utilizada para el cómputo de la siguiente iteración de la aplicación. Un retraso en la transferencia del contenido de los *tiles*, genera que la aplicación degrade las métricas de rendimiento; lo anterior es motivado al sincronismo de datos que existe en cada iteración [36].

2.5 Aplicaciones SPMD en clústeres con procesadores multicore

Como se observó anteriormente, los clústeres con procesadores multicore presentan un conjunto de niveles de comunicaciones, cada uno con diversos tiempos de comunicación y latencia [37]. Estas latencias generan gran impacto en las aplicaciones por lo que se deben considerar al ejecutar una aplicación paralela en este tipo de entorno [12]. Este impacto es mayor, cuando se utilizan aplicaciones que han sido diseñadas para usar librerías de paso de mensajes para su proceso de comunicación y han sido desarrolladas bajo el enfoque del paradigma de programación SPMD.

Las aplicaciones diseñadas bajo el paradigma SPMD tienen un proceso de comunicación que es iterativo donde se debe realizar el intercambio de información. Cuando una aplicación SPMD es mapeada para ser ejecutada en un entorno multicore, se deben considerar las latencias de comunicaciones, debido a que las mismas generan desequilibrios en las iteraciones de ejecución. Un ejemplo se puede detallar en la Figura 22, donde una aplicación ha sido mapeada en un conjunto de nodos compuesto por un sistema multicore y las comunicaciones del enlace más lento determinan el final de una iteración de intercambio.

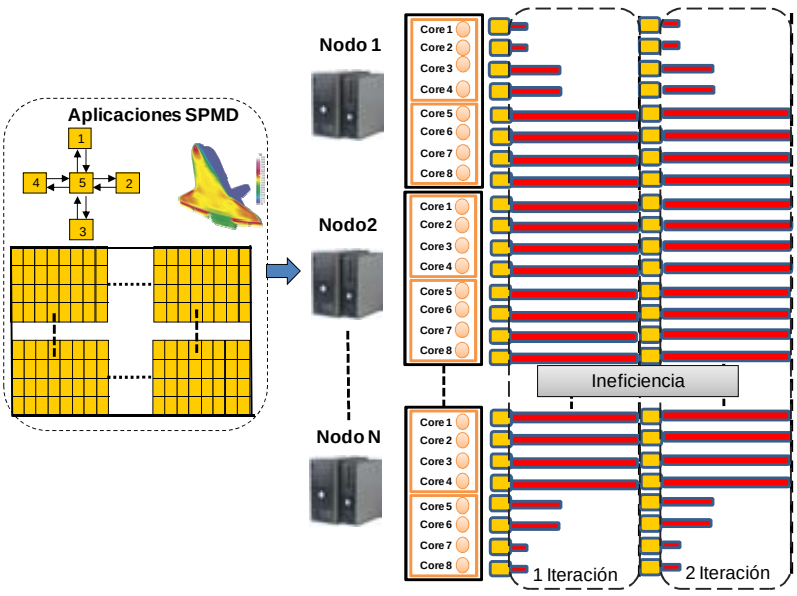


Figura 22 Aplicación SPMD mapeada en un clúster multicore

En la imagen se detalla que pueden existir *tiles* que tienen comunicaciones con *tiles* que están dentro del mismo core por ejemplo en el caso de los cores 1 y 2 del nodo 1. En este caso en particular la comunicación es realizada por la arquitectura interna del procesador. Otro ejemplo es la comunicación entre los cores 5 y 8 del nodo 1 y los cores 1 y 4 del nodo 2 que realizan su proceso de comunicación por la red. En este caso si se asigna la misma cantidad de *tiles* por cores, el sistema se desbalancea por la heterogeneidad de las comunicaciones. La diferencia de tiempos crea tiempos de espera debido a que no se puede empezar la siguiente iteración hasta que culmine la comunicación del enlace más lento.

Aunque el tiempo de cómputo de los *tiles* es similar por la homogeneidad de los cores, los mismos se deben comunicar a *tiles* que pueden ubicarse en el mismo nodo o en otro nodo. Esto genera comunicaciones entre diferentes enlaces que en algunos casos alcanza hasta un orden y medio de magnitud en diferencia de latencia para el mismo volumen de transmisión.

Estas ineficiencias afectan seriamente al tiempo de ejecución de la aplicación, a la eficiencia y al *speedup* por lo que se debe buscar un modelo que permita gestionar estas comunicaciones de forma eficiente. Una solución al problema de gestionar la comunicaciones de los clúster con procesadores multicore es utilizando estrategias de mapping y *scheduling* que permitan minimizar los efectos de las comunicaciones para aumentar la eficiencia.

La estrategia de *mapping* se aplica para dar una planificación espacial de los *tiles* o mapeo de datos, y la estrategia de *scheduling* se utiliza para realizar una planificación temporal que permita determinar el orden de ejecución de los *tiles* para minimizar el efecto de las comunicaciones [37].

2.6 Métricas de rendimiento paralelo

Las métricas de rendimiento de las aplicaciones SPMD en ambientes con comunicaciones heterogéneas como el presentado en los clústeres con procesadores multicore, se ven seriamente afectadas por las diferentes latencias de cada enlace, lo que origina que se pierda eficiencia y

aceleración computacional. Es por ello, que al ejecutar aplicaciones con alto nivel de comunicaciones en estos clústeres se deben considerar algunos parámetros de rendimiento que son fuertemente influenciados por la ejecución paralela.

A continuación se detallan las métricas de rendimiento que se consideran en la presente investigación.

2.6.1 Tiempo de ejecución

La métrica más afectada por el problema del desequilibrio que generan las comunicaciones en los entornos multicore es el tiempo de ejecución de la aplicación [20][31]. Esto motivado a que la iteración de las aplicaciones SPMD está delimitada por el tiempo de comunicación del enlace más lento. Si ejecutamos una aplicación con alta dependencia en la comunicación de los *tiles*, los *tiles* con menor tiempo de comunicación deben esperar la comunicación de los vecinos. Es por esto, que al buscar una estrategia que disminuya el efecto de las comunicaciones, la misma permitirá que esta métrica de rendimiento se minimice y por ende se mejoren las métricas que dependen de este índice como la eficiencia y el *speedup*.

2.6.2 *Speedup*

La otra métrica que es afectada directamente por el tiempo de ejecución paralelo es el *speedup*. Este *speedup* determina qué tan rápido un entorno multiprocesador resuelve una aplicación. Para el cálculo del *speedup* se considera la mejor ejecución serial y es comparado contra el tiempo de ejecución paralelo [61]. Este valor es definido como factor de *speedup*, y el valor ideal es conseguir un *speedup* lineal con respecto al número de elementos de procesamiento (cores) que ejecutan la aplicación.

2.6.3 Eficiencia

La buena administración de los recursos de cómputo es un aspecto fundamental a la hora de escribir aplicaciones para entornos paralelos. Por

lo tanto, se debe buscar una relación entre la aceleración computacional y el porcentaje de tiempo empleado por un recurso efectivo, esto determina si la ejecución ha sido eficiente o no. Los tiempos de espera que generan las comunicaciones son traducidos como ineficiencia, por lo que la eficiencia global de la aplicación se ve afectada.

El índice de eficiencia se define como el tiempo que los procesadores están siendo usados por el cómputo [61]. Esta eficiencia es calculada dividiendo el tiempo secuencial de la aplicación entre el tiempo paralelo multiplicado por el número de elementos de procesamientos de ejecución. Al minimizar el tiempo de ejecución paralelo, la eficiencia se verá incrementada. Bajo este enfoque, el objetivo de esta investigación es ejecutar aplicaciones SPMD en clústeres con procesadores multicore con un nivel de eficiencia definido por un umbral. Esta eficiencia definida permite establecer estrategias para gestionar los puntos que crean mayor ineficiencia en las aplicaciones paralelas.

2.6.4 Escalabilidad

El rendimiento de la aplicación paralela depende del tamaño del problema y del número de procesadores que integran el sistema o entorno en que se está ejecutando la aplicación. La escalabilidad de la aplicación mide el nivel de adaptación que tiene el sistema de ejecución paralelo y la aplicación. En este sentido, existen dos tipos de escalabilidades. La primera es definida como escalabilidad débil, la cual es considerada como el incremento del tamaño del problema y el número de elementos de procesamiento simultáneamente [46]. El principal objetivo de esta escalabilidad es encontrar una solución constante a este incremento.

Por otra parte, la escalabilidad fuerte se calcula cuando se fija el tamaño del problema y el número de elementos de procesamientos se incrementan [1][26]. Esta escalabilidad busca encontrar el número de elementos de procesamiento que minimizan el tiempo de solución de la aplicación. El término de escalabilidad significa el crecimiento proporcional del *speedup* con respecto al número de cores [39].

2.7 Estado del Arte

La presente investigación busca desarrollar una metodología de ejecución eficiente de aplicaciones SPMD en clústeres multicore. Esta metodología está enfocada en encontrar el máximo *speedup* mientras la eficiencia se mantiene sobre un umbral definido. Bajo el enfoque de la presente investigación, existen trabajos que desarrollan metodologías para mejorar algunas métricas de rendimiento de aplicaciones SPMD en diferentes entornos.

Un método de particular interés ha sido desarrollado por Liedbrock [29], este trabajo define y analiza una metodología para derivar un modelo de rendimiento para aplicaciones híbridas SPMD (MPI- OpenMP). El foco principal de la investigación es la mejora de tres métricas de rendimiento: adaptabilidad, fidelidad y escalabilidad. Para alcanzar su objetivo, aplicó estrategias de *mapping* basadas en características del entorno. A diferencia de Liedbrock, nuestra metodología se centra en la integración de estrategias de *mapping* y *scheduling*, minimizando los efectos de las comunicaciones. Estas estrategias buscan una mejora entre el *speedup* y la eficiencia para los clústeres con procesadores multicore.

Otro método que permite la mejora de métricas de rendimiento fue presentado por Vikram [60]. La propuesta de este trabajo fue el desarrollo de una estrategia para mapear tareas en arquitecturas reconfigurables. El objetivo principal de este *mapping* es encontrar el máximo *speedup* posible para una configuración específica, una velocidad de bus y un tiempo de cómputo. Sin embargo, nuestra propuesta busca el máximo *speedup* pero se ha colocado la restricción de la eficiencia por lo que se debe desarrollar un modelo que maximice el índice del *speedup* con la restricción de la eficiencia sobre un umbral.

Al enfocar una solución en una estrategia de planificación espacial o *mapping*, existen trabajos que permiten la gestión de la distribución de los *tiles* entre los procesos, pero muchos de éstos no consideran la eficiencia del entorno [28][52]. No obstante, el *mapping* aplicado a nuestra metodología busca minimizar los efectos de las comunicaciones con el fin de lograr una

ejecución eficiente a través de la configuración de un conjunto de *tiles* a cada core de ejecución. Este *mapping* hace posible asignar la cantidad necesaria de *tiles* que permite desarrollar una estrategia de solapamiento, la cual está incluida en la fase de *scheduling*.

Por otra parte, en relación a la planificación temporal o *scheduling*, hay investigaciones que se han enfocado en diseñar estrategias de *scheduling* para aplicaciones SPMD en entornos mono procesamiento [8][46][61][62] y otras orientadas a entornos multicore [1][19]. Estos métodos fueron desarrollados para minimizar el tiempo de ejecución de las aplicaciones SPMD en cada uno de los entornos de ejecución, el cual es parte del objetivo de la metodología propuesta.

Específicamente, se evaluó un método de *scheduling* desarrollado por Panshenskov [46] y de forma similar, la metodología propuesta considera ciertas características para la aplicación del método de solapamiento. Dentro de estas características, se mencionan: la división de los *tiles* por bloques, el uso de comunicaciones asíncronas y la minimización de la sobrecarga generada por los enlaces de comunicaciones. Además de estas características, el proceso de *scheduling* integrado en nuestro método permite diferenciar los *tiles* por orden de ejecución usando un modelo de asignación de prioridades de ejecución, con el fin de ejecutar los *tiles* de bordes primero y solapar el cómputo interno de los *tiles* con la comunicación de los bordes. Este proceso de solapamiento y asignación será detallado en profundidad cuando se describa la metodología.

El desarrollar una metodología de ejecución eficiente en entornos de comunicaciones heterogéneas es un desafío como fue demostrado por Oliver [44]. En ese sentido, ese trabajo estaba orientado en encontrar la mejor ubicación de procesos dentro de los cores de ejecución, con el objetivo de gestionar las comunicaciones. Además, este trabajo se centró en diseñar un método para minimizar el overhead de comunicaciones, considerando que diseñar estrategias de distribución y planificación de datos eficiente es un problema NP complejo cuando se utilizan entornos de comunicaciones variables [44].

Otras investigaciones que evalúan las comunicaciones en los entornos multicore, se basan en proponer soluciones para gestionar las comunicaciones internas dentro del multicore [4][49]. En este sentido, nuestra metodología realiza una evaluación de las comunicaciones considerando el patrón real de la aplicación con el objetivo de tener las diferencias reales que se obtienen al usar la estructura jerárquica de comunicaciones del multicore [37].

Considerando los trabajos descritos anteriormente, el siguiente capítulo presenta una novedosa metodología, enfocada en minimizar el *overhead* de las comunicaciones y gestionar los recursos de forma eficiente.

2.8 Conclusiones

Este capítulo ha presentado las bases teóricas que sustentan el desarrollo de la presente investigación. En primer punto se describió las arquitecturas paralelas, enfocándonos principalmente en las arquitecturas de multiprocesamiento con multicore. Luego, se describió los modelos de programación paralela, considerando que la investigación se desarrollará para aplicaciones basadas en el modelo de paso de mensaje. Seguido, se describieron los paradigmas de programación paralelos, profundizando el comportamiento del paradigma SPMD.

Al exponer ambos tópicos a nivel de arquitectura (multicore) y a nivel de programación (Paradigma SPMD), se evalúan los problemas que se presentan al ejecutar aplicaciones SPMD en clústeres formados por nodos con procesadores multicore. Dentro de los principales problemas se encuentran los retrasos que se pueden generar por el sistema jerárquico de comunicaciones de los entornos multicore. Estos problemas afectan seriamente las métricas de rendimiento como los son el tiempo de ejecución, la eficiencia y el *speedup*. Es por ello, que la presente investigación se enfoca en el desarrollo de una metodología de ejecución eficiente para aplicaciones SPMD en clústeres con procesadores multicore.

Para finalizar, en este capítulo se describieron los trabajos más importantes en el área de estudio y las diferencias que existen entre esas propuestas y el desarrollo de la metodología.

Capítulo 3

Metodología de ejecución eficiente para aplicaciones SPMD en clúster con procesadores multicore

Resumen

Ejecutar aplicaciones paralelas de forma eficiente se convierte cada día en uno de los principales retos de los programadores de HPC y más aún cuando se utiliza un entorno de cómputo que incluye cierta heterogeneidad. En el caso de los clústeres con procesadores multicore, la heterogeneidad está integrada por los diferentes niveles de comunicación, los cuales generan gran desbalanceo a la hora de ejecutar aplicaciones con alto nivel de comunicaciones, como las desarrolladas bajo el paradigma SPMD. Este capítulo presenta una metodología de ejecución eficiente para aplicaciones SPMD enfocada en el manejo de las heterogeneidades de comunicaciones para mejorar la eficiencia de la aplicación en los clústeres con procesadores multicore. Esta metodología, está integrada por 4 fases: caracterización, modelo de distribución de *tiles*, *mapping* y *scheduling* y su principal objetivo es determinar el número ideal de tiles que deben ser asignados a cada core, así como el número de cores que permiten alcanzar el máximo speedup mientras la eficiencia de la aplicación se ubica por encima de un umbral definido por el usuario.

3.1 Introducción

En este capítulo se presenta la metodología propuesta que permite ejecutar eficientemente aplicaciones desarrolladas bajo el paradigma SPMD en un clúster con procesadores multicore, donde es importante es analizar la heterogeneidad de los enlaces de comunicación utilizados para intercambiar datos entre procesos. Al aplicar la metodología, se consigue una solución adecuada sobre la asignación del los *tiles* y el número de cores que permita cumplir con el principal objetivo de obtener el máximo *speedup* de la aplicación mientras se mantiene la eficiencia sobre un umbral definido. Para obtener una ejecución eficiente y cumplir con los objetivos planteados (descritos en el capítulo 1), la metodología se divide en cuatro fases principales llamadas: caracterización, modelo de distribución de *tiles*, *mapping* y *scheduling*. Estas fases proponen cómo manejar de forma adecuada los desequilibrios generados por las diferentes rutas de comunicación presentes en el multicore. Cuando no se consideran estos desequilibrios se crean ineficiencias, que deben ser controladas si se desean mejorar las métricas de rendimiento.

Las ineficiencias creadas por los tiempos de espera (idle) del procesador en las aplicaciones SPMD, se pueden originar por diversas razones tales como: la espera por las comunicaciones entre *tiles* ubicados en diferentes cores, procesadores o nodos (estos tiempos de espera pueden aumentar cuando se producen cuellos de botella en los enlaces de comunicaciones), la sincronización de los *tiles*, la adaptación de aplicaciones de paso de mensaje (MPI) diseñadas para ejecutarse en ambientes monocore y que por la evolución del HPC son ejecutadas en clúster compuestos de nodos con procesadores multicore, entre otros.

Estos tiempos de espera afectan a las métricas de rendimiento, especialmente el *speedup* y la eficiencia conseguida. Por lo tanto, la metodología busca una estrategia que permita gestionar los efectos de la comunicación y manejarlo para lograr una ejecución eficiente, con el objetivo de mejorar las métricas de rendimiento de la aplicación paralela.

En la Figura 23, se muestra cada una de las fases de la metodología y cómo las mismas permiten buscar una ejecución eficiente de la aplicación paralela. La primera fase a realizar es la caracterización, cuyo principal objetivo es recolectar la información necesaria de la arquitectura paralela (en este caso en clúster con procesadores multicore) y de la aplicación SPMD, con el objetivo de calcular el modelo de la siguiente fase de la metodología. Este conjunto de parámetros considera la estrecha relación que existe entre la máquina y la aplicación paralela, para obtener mejoras en las prestaciones de la aplicación.

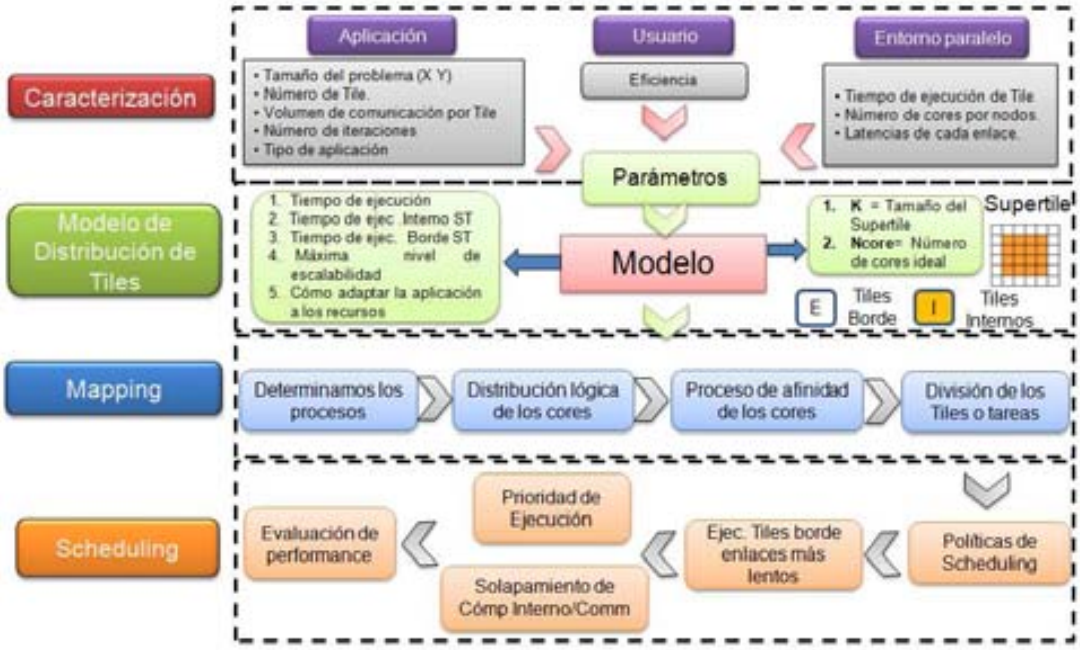


Figura 23 Metodología de ejecución eficiente de aplicaciones SPMD en clúster con procesadores multicore.

Una vez obtenidos los parámetros en la fase de caracterización, el siguiente paso es calcular de forma analítica el número de tiles que componen el supertile (ST) que mantiene el máximo speedup con una eficiencia definida, así como determinar el número de cores que obtienen una ejecución con el máximo punto de escalabilidad fuerte considerando la eficiencia sobre un umbral definido. Para hacer estos cálculos partimos de la clasificación de los tiles del supertile en borde e internos, donde los tiles de bordes se definen como las tareas que ejecutan su proceso de comunicación con tiles ubicados en otros cores, y los tiles internos son aquellos que su

comunicación es realizada dentro del mismo core. Por lo tanto, el modelo parte de que se puedan ejecutar primero los tiles de borde seguido de una estrategia de solapamiento entre las comunicaciones de los tiles de borde y el cómputo de los tiles internos.

La siguiente fase de la metodología consiste en aplicar la estrategia de *mapping*, con el principal objetivo de minimizar los canales de comunicación por los enlaces más lentos. Esta estrategia de *mapping* busca la forma de distribuir los procesos MPI y los *supertiles* en cada uno de los cores, minimizando las latencias de los diferentes enlaces de comunicación.

La última fase es la del *scheduling,* donde se cumplen dos funciones principales. La primera es mantener un orden de ejecución de los *tiles*, que permite ejecutar primero los *tiles* de bordes, seguido por la ejecución de los *tiles* internos. Este orden de ejecución permite aplicar la segunda función que consiste en la aplicación de la estrategia de solapamiento entre las comunicaciones de los bordes y el cómputo interno. El solapamiento permite ocultar los efectos de las comunicaciones motivado a que los cores computan mientras comunican. Las comunicaciones de los bordes por el enlace más lento serán un elemento limitante para mejorar el *speedup*, y la mayor cantidad de trabajo permitirá mejorar la eficiencia. Es por esto que la metodología busca la mejor relación entre la máxima aceleración de la aplicación, sin sacrificar la eficiencia.

A continuación se explicará detalladamente cada una de las fases de la metodología y conjuntamente cómo se pueden aplicar a un ejemplo real de una aplicación SPMD en un clúster con procesadores multicore, específicamente se utilizará la aplicación de la transferencia de calor[4] ejecutada en un entorno multicore[5].

3.2 Fase de Caracterización

El objetivo de esta fase es evaluar el entorno de ejecución tanto a nivel de cómputo como a nivel de comunicaciones, y hacer un análisis de la

[4] Aplicación de diferencias finitas con cuatro comunicaciones por tiles
[5] Entorno multicore con dos Quad core por nodo, con una memoria cache L2 de 6 MB compartida por dos cores, una memoria RAM de 12 GB y una red Giga-Ethernet

aplicación paralela SPMD a ejecutar. Este proceso de evaluación y análisis permite generar un conjunto de parámetros que serán claves para el cálculo de la siguiente fase de la metodología (Modelo de distribución de *tiles*). Estos parámetros están divididos en tres sub grupos y son definidos de la siguiente manera: parámetros de la aplicación, parámetros del entorno de ejecución y parámetros de performances deseados para la ejecución de la aplicación en el entorno.

3.2.1 Caracterización de los parámetros de la aplicación

El objetivo de esta caracterización es descubrir un conjunto de parámetros y el comportamiento de la aplicación SPMD en el entorno de ejecución. Algunos de estos parámetros se deben buscar dentro de la aplicación SPMD y están relacionados con el tamaño del problema, el número de iteraciones o tipo de distribución. Además, se debe identificar la parte de la aplicación que hace el intercambio iterativo de la aplicación SPMD para determinar el tiempo de cómputo y el volumen de comunicación de un *tile*. Ambos valores serán utilizados para calcular la ratio entre el cómputo y la comunicación (este proceso de cálculo se define en la caracterización del entorno).

El siguiente paso es evaluar el patrón de comunicaciones de la aplicación. La Figura 24, muestra diversos patrones de comunicaciones que se pueden presentar en este tipo de aplicaciones. Para tener un conocimiento del patrón de comunicación entre los procesos MPI se utiliza una herramienta de predicción llamada PAS2P (*Parallel application signature for performance prediction*) [65].

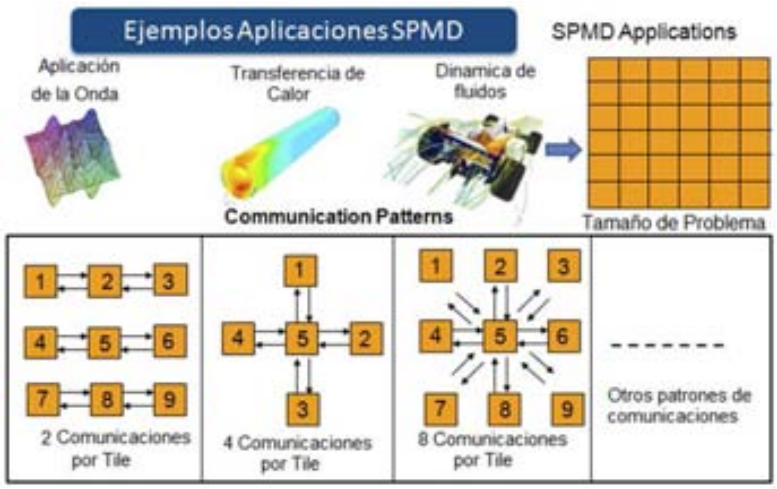
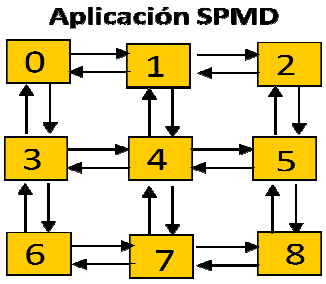


Figura 24 Ejemplo de aplicaciones SPMD y algunos patrones de comunicación

La función principal de esta herramienta es extraer las fases de la aplicación a través de la recolección y el análisis de eventos de cómputo y comunicación. La información de comunicación recolectada es agrupada y leída como parámetros de entrada para el módulo de caracterización de comunicaciones del entorno.

Como se muestra en la Figura 25, los valores obtenidos a través de la herramienta PAS2P para una fase, ofrecen una información detallada del patrón de comunicaciones de la aplicación, lo que permite tener un conocimiento del comportamiento de la aplicación a nivel de las comunicaciones. Esta información es grabada en el vector de entrada que simula el comportamiento de los mensajes MPI para la herramienta de caracterización; los espacios donde no existan comunicaciones son rellenados con el propio Rank del proceso MPI.

Esto es realizado para identificar que esta comunicación no debe ser realizada en la caracterización de las comunicaciones (Figura 25). Además, en la figura se puede observar el vector de comunicaciones para cada proceso, donde cada columna es la información de cada proceso MPI y en la fila identifica hacia qué proceso MPI se debe realizar la comunicación. El ejemplo de la Figura 25 muestra el patrón de comunicaciones de la aplicación de la Transferencia de Calor, la cual está compuesta por un patrón de comunicaciones a cuatro vecinos.



Patrón de Comunicaciones (PAS2P)
Rank de Procesos MPI

0	1	2	3	4	5	6	7	8
1	2	1	4	5	4	7	8	7
3	0	5	0	3	2	3	6	5
0	4	2	3	1	8	6	4	8
0	1	2	3	7	5	6	7	8

Figura 25 Patrón de comunicaciones aplicación de la trasferencia de calor

Finalmente, una vez obtenido todos los parámetros de caracterización de la aplicación SPMD, se procede a realizar el análisis del entorno de ejecución con elementos propios de la aplicación para evaluar la estructura jerárquica de comunicaciones y el proceso de cómputo de la aplicación.

3.2.2 Caracterización del entorno de ejecución

El proceso de caracterización del entorno paralelo se divide en dos partes. Inicialmente se realiza un análisis de la estructura jerárquica de comunicaciones, basándose en el patrón de comunicaciones obtenido en la caracterización de la aplicación. Luego se evalúa el proceso de cómputo que realizan los *tiles* de la aplicación paralela en el entorno de ejecución. Ambos análisis se ejecutan para obtener la ratio de cómputo-comunicación de los *tiles* en cada uno de los niveles de comunicación que integran el clúster con procesadores multicore.

- **Caracterización de la estructura jerárquica de comunicaciones**

Para el estudio del sistema jerárquico de comunicaciones en el clúster con procesadores multicore, se debe tener cierto conocimiento de la arquitectura del clúster donde se desea realizar la ejecución, por ejemplo un clúster puede estar compuesto por nodos con procesadores comerciales con diseños como: *Intel Quad-core* [L5], *Dual-core* [L3] *Xeon*, *AMD Quad-core* [L2], *Dual-core* [L1], *IBM Cell* [L3], entre otros; debido a que dependiendo de la arquitectura del procesador, estos pueden variar el número de cores

dentro del procesador, el número de procesadores en un nodo, los niveles de comunicaciones. En la Figura 26 se muestran los diversos niveles de comunicaciones de una arquitectura de un clúster con nodos con procesadores Intel *Quad-core*. Las comunicaciones entre los cores (A), se realizan dentro del procesador a través de las memorias cache; la comunicación entre los procesadores (B), se hace utilizando como canal de comunicación la memoria principal y finalmente la comunicación entre los nodos (C) es realizada a través de la red de área local.

El siguiente paso para el análisis de las comunicaciones consiste en utilizar una herramienta que permita testear los tiempos de latencia de comunicación de un mensaje por los diversos enlaces de la arquitectura multicore. Para hacer estas medidas existen distintas herramientas en la literatura de las que se puede mencionar, *Netpipe*[L10], *Stream*[L7], entre otros. Estas herramientas ejecutan su proceso de recolección de medidas de latencia y ancho de banda utilizando una estrategia de comunicaciones punto a punto con dos procesos utilizando técnicas de Ping Pong.

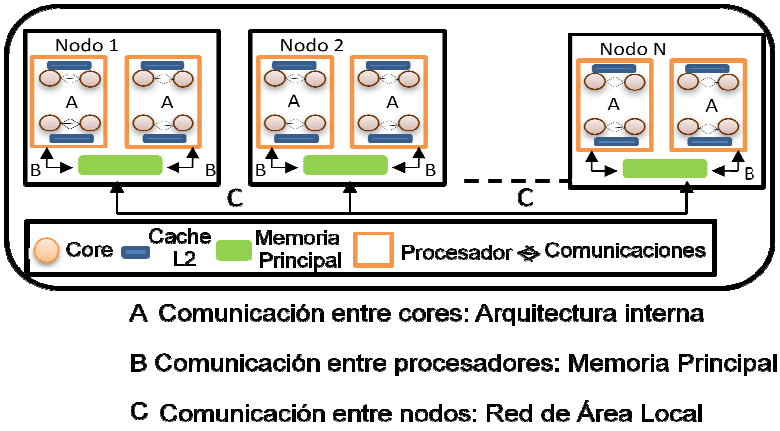


Figura 26 Enlaces de comunicación clúster con procesadores multicore con nodos QuadCore

Sin embargo, este modelo de comunicación no se ajusta al patrón real de las aplicaciones SPMD, debido a que las mismas están diseñadas para comunicar con dos, cuatro, ocho,… n vecinos de forma simultánea. Esto conlleva a que exista mayor probabilidad de colisiones y niveles de latencia cuando ejecutamos utilizando todos los cores del nodo.

Para resolver este proEnvironmenteblema hemos diseñado una herramienta de caracterización, que permite medir los tiempos de latencia de los enlaces utilizando de forma similar estrategias de ping pong, pero con la ventaja adicional que se puede medir utilizando el patrón de comunicaciones de la aplicación y se puede especificar la cantidad de procesos con los que se desea medir. Esta herramienta incluye medidas para comunicaciones simultáneas entre todos los procesos que se ejecutan dentro del clúster con procesadores multicore, para simular el comportamiento real de la aplicación SPMD en cada una de sus iteraciones y tener así mayor precisión de los tiempos de comunicación.

Modulo de Caracterización para un número (i) de procesos MPI

```
1:  Distribución lógica de los procesos (Número global de procesos)
2:  Afinidad de procesos (PID del proceso MPI)
3: Creación de la topología de comunicación (Determinar topología 1D, 2D, 3D)
5: Determinar comunicaciones vecinas ()
6: Haga desde mensaje =8B  hasta 16MB //Determina el mensaje a evaluar
7:     Haga desde iteración = 1 hasta 1000 // número de iteraciones a recolectar
8:         Haga desde comunicación=1 hasta Número de comunicaciones
9:             Si ( comunicador[comunicación]   !=  Rank)
10:                 ISend(Mensaje,….., Comunicador[comunicación], ….. request)
11:                 Irecv(Mensaje,……. Comunicador[comunicación], …..   request)
12:                 Proceso de recolección de tiempos()
13:             Fin Si (10)
14:  Fin haga desde(7)
15: Fin haga desde (6)
16: Generación de reporte de comunicaciones()
```

Figura 27 Algoritmo de caracterización de comunicaciones

En la Figura 27 se describe el algoritmo de caracterización. Inicialmente hace una distribución lógica de los procesos MPI, en el que se determina un número de coordenadas X, (X,Y) o (X,Y,Z) dependiendo del tipo de aplicación (1D, 2D o 3D) con el fin de generar un identificador a cada proceso MPI. Una vez realizada la identificación se procede a la aplicación de un módulo de afinidad entre los procesos, que permite asignar los respectivos cores de ejecución y tener el control de la ubicación dentro de la

distribución lógica de procesos MPI. Este proceso de afinidad es realizado para asignar el core donde debe ser ubicado el proceso MPI, y de esta manera controlar la topología de comunicación (ejemplo 1D, 2D o 3D respectivamente).

El siguiente paso en la herramienta de caracterización, es incluir en el vector de comunicaciones, el patrón de comunicaciones vecinas entre los procesos MPI involucrados. Esta información es obtenida cuando se caracteriza la aplicación con la herramienta PAS2P, como fue explicado anteriormente y luego esa información es incluida como parámetro en la herramienta de caracterización.

Al tener configurada toda la información de comunicación, se realiza el proceso iterativo donde se recolecta la información de tiempos de latencias y ancho de bandas para diferentes tamaños de mensajes utilizando comunicaciones asíncronas. Este tipo de comunicaciones no bloqueante son utilizadas para simular el comportamiento de las comunicaciones en un entorno donde se solapará el cómputo con la comunicación. El proceso de recolección de información es repetido en 1000 iteraciones para cada tamaño de paquete.

Seguidamente se adquiere y se procesa la información generando los ficheros de caracterización, que luego son evaluados y filtrados dependiendo del tipo de comunicación que se desea analizar (*intercore, interchip o internode*).

La Figura 28 muestra un ejemplo de los datos de caracterización de comunicaciones para un clúster con procesadores *Quad-Core*. Como puede visualizarse, los niveles de comunicaciones del clúster tienen diferentes latencias que en algunos casos alcanza hasta un orden y medio de magnitud de diferencia para el mismo volumen de comunicación. Estas diferencias generan gran desequilibrio a la hora de ejecutar una aplicación con altos volúmenes de comunicación y dependencia de las comunicaciones como es el caso de las aplicaciones SPMD que están siendo objeto de estudio en este trabajo.

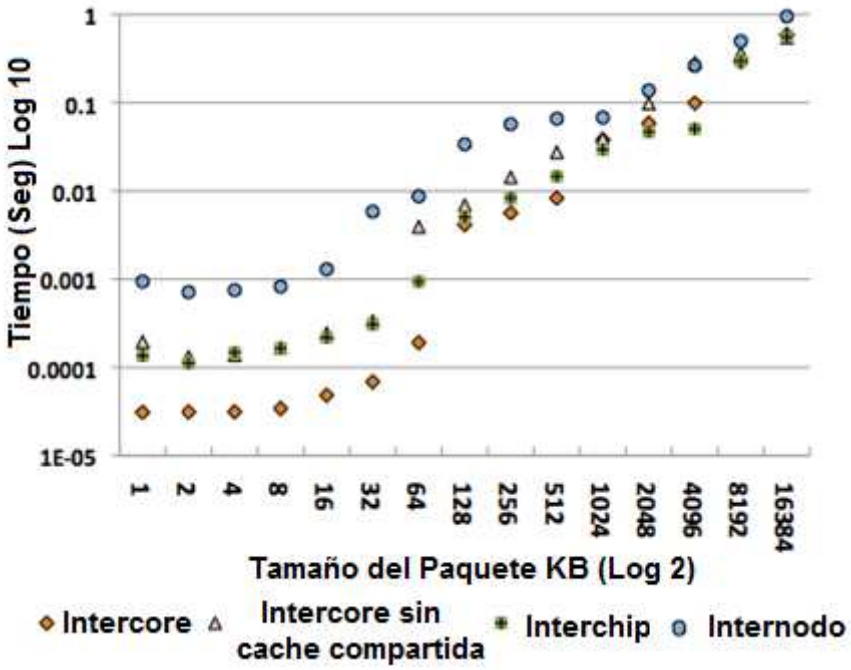


Figura 28 Tiempo de comunicación para un clúster Quad Core

Además, se puede apreciar en la Figura 28 que a medida que el tamaño del paquete va aumentando, los canales de comunicación a través de la arquitectura interna del nodo se van saturando, hasta el punto que la diferencia entre los enlaces están dentro del mismo orden de magnitud. Para finalizar esta parte de la caracterización, se determina la latencia y el ancho de banda de cada enlace de comunicación. Asimismo, se puede visualizar dos comunicación entre los cores (*intercore*), esto es debido a que existen cores que no comparten la misma memoria cache y por ende para su comunicación deben utilizar la memoria principal. Este análisis de comunicaciones es muy importante para la precisión del modelo en la siguiente fase.

- **Caracterización del proceso de cómputo de la aplicación**

Para determinar el cómputo de un tile dentro de la aplicación, se realiza una instrumentación de la aplicación SPMD, donde el primer elemento a monitorizar está relacionado con la función de cálculo que debe ejecutar cada *tile* en la arquitectura. Para realizar este proceso se asigna un conjunto

de *tiles* a cada core y se toma el tiempo de cómputo promediado del conjunto de *tiles* ejecutados.

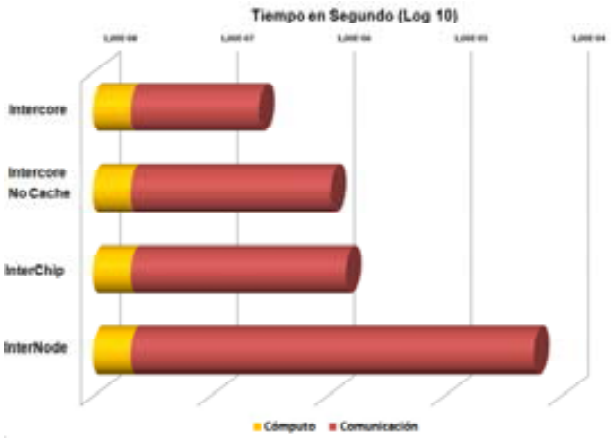
De forma similar, se debe medir el volumen de comunicación de los *tiles* para poder obtener la ratio entre el cómputo y la comunicación dentro de la arquitectura jerárquica de comunicaciones. Esta ratio será utilizada dentro del modelo analítico para determinar el tamaño ideal del *supertile* (ST) y del número de cores a utilizar.

Esta relación entre el cómputo y la comunicación es definida por la variable ***λ(p)(w),*** donde ***p*** determina el enlace de comunicación de un tile hacia otro tile vecino; la variable ***w*** describe la dirección del proceso de comunicación (por ejemplo, arriba, abajo, derecha o izquierda en un patrón de 4 comunicaciones (Figura 25)). Esta ratio es calculada utilizando la ecuación E1.

$$\lambda(p)(w) = \frac{TiempoComm(p)(w)}{TiempoCpt} \qquad \text{(E1)}$$

La ecuación E1 está representada por $\boldsymbol{TiempoComm(p)(w)}$ que determina el tiempo de comunicar un tile en un enlace de comunicación específico (***p*** **=** *intercore, interchip, Internode*) y la variable $\boldsymbol{TiempoCpt}$ precisa el tiempo de cómputo de un tile en el core de ejecución.

Un ejemplo se muestra en la Figura 29, donde se puede visualizar la ratio cómputo comunicación calculada para la aplicación de la transferencia de calor. Como se detalla en la figura, el tiempo de cómputo es similar por la homogeneidad de los cores y las comunicaciones pueden tener grandes variaciones que ocasionan la diferencia de la ratio cómputo y comunicación en cada enlace. En este caso en particular, la aplicación tiene un volumen de comunicación de 8 Bytes por tile.



Enlace de comunicación	Tiempo Comm(p)(w)	Tiempo Cpt	Ratio λ
Intercore	2.42E-07	2.10E-08	11.52
Intercore No cache	1.06E-06	2.10E-08	50.40
Interchip	1.43E-06	2.10E-08	60.10
Internode	5.88E-05	2.10E-08	2800

Figura 29 Ratio cómputo comunicación aplicación de trasferencia de calor

Un elemento importante a considerar en este proceso de caracterización, es que el mismo debe ser realizado de forma exclusiva, monitorizada y controlada con el objetivo de obtener los valores más acordes con la realidad de la ejecución de la aplicación en el entorno.

3.2.3 Caracterización de los parámetros de performance deseados

El último parámetro es definido por el programador o usuario de la aplicación. En este sentido el usuario define el umbral de eficiencia con el que desea que la aplicación sea ejecutada. Esta eficiencia está definida con la variable ***effic*** y será incluida como el umbral de eficiencia requerido como parámetro de entrada en el modelo a la hora de calcular el tamaño ideal del *supertile* y el número de cores que mantiene el máximo *speedup* y la eficiencia por encima de este umbral definido.

3.3 Modelo de distribución de tiles

Un incorrecto cálculo en la cantidad de *tiles* que se deben asignar a los cores, puede ocasionar diversos escenarios como se muestra en la Figura 30.

Analizando la Figura 30, se puede detallar un primer escenario identificado como core 1. En este caso el número de *tiles* internos es mayor que las comunicaciones de los *tiles* de bordes. La estrategia de solapamiento tiene un comportamiento identificado como limitado por el

cómputo, por lo que la eficiencia es cercana al máximo valor, debido a que el core siempre tiene carga de trabajo para su procesamiento. Sin embargo, el *speedup* puede ser mejorado debido a que el tiempo de ejecución de los *tiles* internos sobrante puede ser asignado en otro conjunto de cores. Esta nueva asignación permitiría mejorar la aceleración computacional de la aplicación.

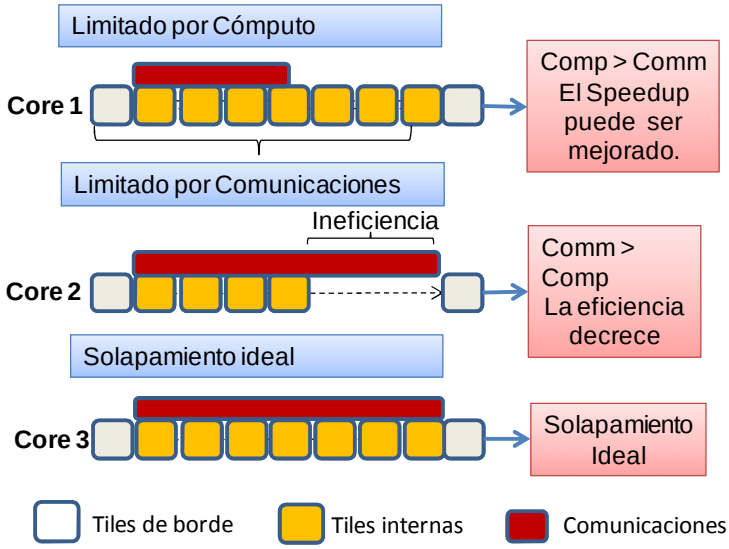


Figura 30 Comportamientos de la estrategia de solapamiento

El siguiente escenario mostrado en la Figura 30 es el presentado por el Core 2, donde el tiempo de comunicaciones de los *tiles* de borde es mayor al tiempo de cómputo de los *tiles* interno. Este comportamiento es limitado por las comunicaciones y afecta seriamente la eficiencia de la aplicación. En este caso lo ideal es asignar una mayor cantidad de *tiles* por cores para incrementar la eficiencia.

Finalmente, el tercer escenario se encuentra al realizar un solapamiento ideal (Core 3 de la Figura 30). Este comportamiento busca un solapamiento que permita terminar la ejecución del cómputo de los *tiles* interno y las comunicaciones de los *tiles* de borde en un tiempo similar. Este solapamiento ideal es uno de los elementos más importantes de la metodología, debido a que en este punto se puede alcanzar el máximo *speedup* de la aplicación con una eficiencia alrededor de los máximos valores.

Por lo tanto, para alcanzar el objetivo del máximo *speedup* bajo una eficiencia definida, esta metodología incluye un modelo analítico que permite determinar el valor ideal que debe tener el *supertile* y el número de cores con que se debe ejecutar la aplicación SPMD y fundamentándose en el tercer caso presentado en la Figura 30. Para el cálculo de ambos valores, el modelo se fundamenta en una estrategia de solapamiento entre el cómputo interno y las comunicaciones de los bordes. Este solapamiento permite gestionar el efecto de las comunicaciones y por ende mejorar las métricas de rendimiento como el *speedup* y la eficiencia.

Entonces partiendo del solapamiento ideal, la Figura 31 muestra cómo se debe gestionar el umbral de eficiencia definido por el usuario. En este sentido, la metodología permite establecer un valor de ineficiencia máximo que es definido por el programador. Esta ineficiencia es establecida con el objetivo de cumplir con el umbral de eficiencia que debe ser definido para ejecutar la aplicación SPMD. Este valor de ineficiencia permite que una menor cantidad de *tiles* sean asignados por Core y por ende mejorar el *speedup.*

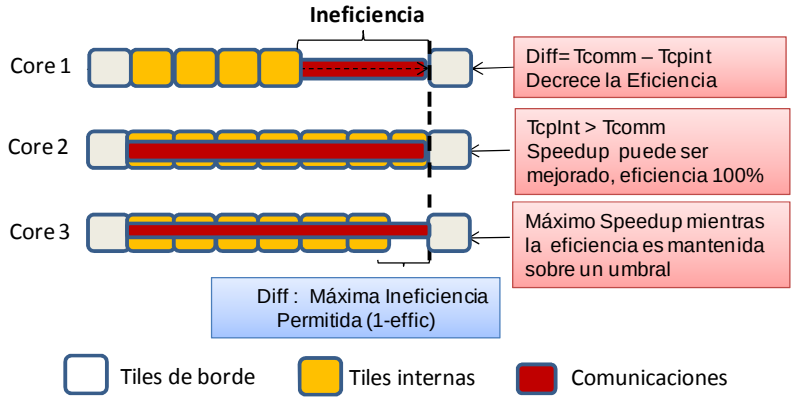


Figura 31 Máxima ineficiencia permitida por el modelo de distribución de *tiles*

No obstante, esta ineficiencia debe ser adicionada de forma controlada y definida como punto máximo. El valor de esta ineficiencia es definida por la sustracción entre el umbral de eficiencia máxima (definida por el 100%) y la eficiencia ***effic,*** que representa el umbral de eficiencia definido (Figura 31, para el caso del core 3).

3.3.4 Desarrollo analítico

Al analizar el comportamiento de las aplicaciones SPMD, se debe considerar la influencia en cada iteración de la ejecución del comportamiento de las comunicaciones por el enlace más lento. Un ejemplo de este comportamiento se muestra en la Figura 32, donde algunos cores pueden presentar menor o mayor tiempo en su proceso de comunicaciones, esto es debido a los enlaces que intervienen en el proceso de intercambio de información. Bajo este comportamiento de comunicaciones, la idea es crear un *supertile* de tamaño homogéneo debido a que se analizan clústeres con procesadores multicore con nodos y cores homogéneos. El tamaño del *supertile* se debe determinar considerando el tiempo de comunicaciones del enlace más lento, el cual marca el tiempo de una iteración de la aplicación SPMD (Figura 32).

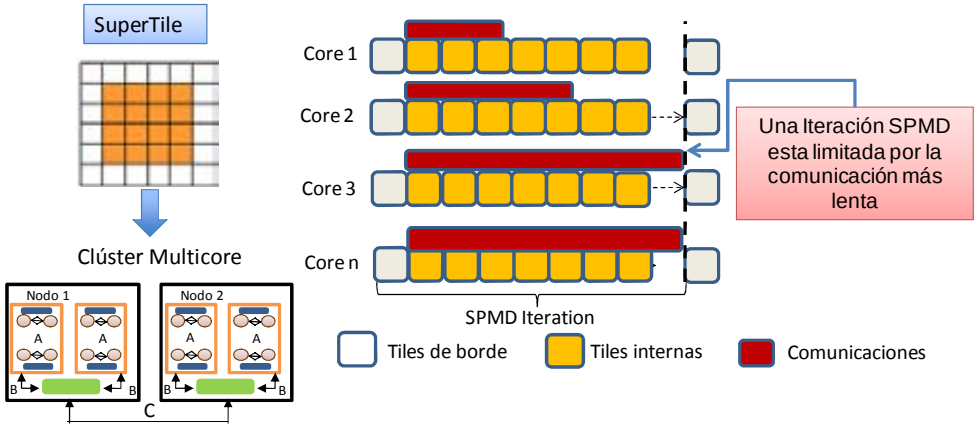


Figura 32 Modelo de ejecución de una aplicación SPMD con la estrategia de solapamiento

Para el cálculo del tamaño ideal del *supertile* (ST), se parte del comportamiento de la aplicación SPMD (utilizando un esquema de solapamiento entre el cómputo y las comunicaciones). Estas aplicaciones deben cumplir con las características de ser estáticas, locales y regulares definidas en el contexto de este trabajo. Estas características se establecen con el objetivo de dar cierto determinismo al comportamiento de la aplicación.

Para comenzar el análisis de la aplicación SPMD, se parte de la ecuación de tiempo de ejecución definida por Wilkilson [63], donde el tiempo de ejecución de una aplicación está compuesto por la suma entre el tiempo de cómputo y el tiempo de comunicaciones que tengan entre los procesos MPI. En este sentido, la ecuación E2 representa este comportamiento pero aplicando el modelo de solapamiento entre el cómputo interno y las comunicaciones de los bordes. En este caso, primero se ejecuta el cómputo de los *tiles* de borde ($\boldsymbol{TC\acute{o}mputoBorde_i}$) sumado con el máximo valor entre el cómputo interno **(**$\boldsymbol{TcompInterno_i}$) y las comunicaciones de los bordes **(**$\boldsymbol{TcomunicacBorde_i}$) que es la parte que la metodología propone solapar dentro de la aplicación. Este proceso de cálculo es repetido por un conjunto de iteraciones definido por la variable ***iter*** y los valores resultantes se van sumando hasta el final de la ejecución de la aplicación.

$$Teje_i = \sum_{q=1}^{iter} (TC\acute{o}mputoBorde_i + Max \begin{cases} TcompInterno_i \\ TcomunicacBorde_i \end{cases} \quad \text{(E2)}$$

Asimismo, en la ecuación E2 la variable ***q*** determina el número actual de la iteración y el valor de ***i*** representa el número del proceso que es asignado a un core específico en el clúster. Esta ecuación (E2) modela el comportamiento de la aplicación SPMD para la sección de intercambio de comunicación entre iteraciones.

Otro aspecto importante a considerar es que las aplicaciones SPMD están diseñadas para tamaños de problemas que pueden tener una distribución de 1, 2, 3,…,n dimensiones, por lo que el modelo varía dependiendo del tipo de distribución de la aplicación. En este sentido la variable n de las ecuaciones E3, E4 E5 y E6 representa la dimensión del problema; en el presente trabajo de investigación se han considerado casos donde esta variable (***n)*** toma valores entre 1 y 3, que son el tipo de dimensión que se incluye dentro de las aplicaciones consideradas.

Por otra parte, para el cálculo del tiempo de ejecución debemos conocer los valores de ($TcompInterno_i$), ($TC\acute{o}mputoBorde_i$) y ($TcomunicacBorde_i$),

que se calculan utilizando las ecuaciones E4, E5 y E6 respectivamente. Sin embargo, estas ecuaciones están en función de la variable K, que es la raíz enésima del tamaño ideal del *supertile* (Ecuación E3).

$$ST = K^n \quad \text{(E3)}$$

$$TcompInterno_i = (K-2)^n * TiempoCpt \quad \text{(E4)}$$

$$TCómputoBorde_i = (ST - (K-2)^n) * TiempoCpt \quad \text{(E5)}$$

$$TcomunicacBorde_i = K^{n-1} * \text{Max}\,(TiempoComm(p)(w)) \quad \text{(E6)}$$

El tiempo de comunicación a utilizar ($TcomunicacBorde_i$) para el cálculo del modelo, se calcula utilizando el canal de comunicación para el caso más desfavorable o con mayor latencia, el cual fue obtenido en la fase de caracterización de la comunicación dentro del clúster con procesadores multicore. Esta comunicación será la que limita la iteración de la aplicación SPMD como ha sido explicado anteriormente. Por lo tanto, se debe utilizar el mayor valor encontrado en la caracterización de la ratio entre cómputo–comunicación de los tiles, el cual fue obtenido con la ecuación E1. Además, la suma de $TcompInterno_i$ y $TCómputoBorde_i$ representa el tiempo de cómputo global de la región del *supertile* (ST) que fue asignado a cada core.

A partir de las consideraciones anteriores, el siguiente paso es determinar el valor de *K* considerando las condiciones iniciales del objetivo de encontrar el máximo *speedup* manteniendo la eficiencia sobre un umbral definido. Para lograr este objetivo, se parte de la estrategia de solapamiento entre el cómputo interno y las comunicaciones de los borde. La ecuación E7 muestra como se puede igualar las comunicaciones de borde con el cómputo interno con el objetivo de encontrar una solución al valor de ***K*** utilizando la estrategia de solapamiento.

Para definir un umbral de eficiencia, se ha definido como entrada un valor máximo de ineficiencia permitida. Esta ineficiencia permite considerar un margen en el cual el tiempo de comunicaciones de los *tiles* de bordes sea mayor al tiempo de cómputo de los *tiles* internos. Sin embargo, para mantener el umbral de eficiencia se debe definir una restricción (ecuación E8). Esta restricción permite que la comunicación de los tiles de bordes sean

mayor al tiempo de cómputo de los tiles internos (ecuación E7), pero a su vez este tiempo de comunicaciones no sea mayor que el tiempo de cómputo interno de los tiles sin ninguna disminución por un valor de eficiencia (ecuación E8).

$$K^{n-1} * \text{Max}\ (TiempoComm(p)(w)) \geq \frac{(K-2)^n * TiempoCpt}{effic} \qquad \text{(E7)}$$

En este sentido, el modelo permite que el tiempo de comunicación del borde sea mayor al tiempo interno considerando una eficiencia definida, pero el tiempo de comunicación de borde debe ser menor o igual que el tiempo de cómputo interno si se define como entrada una eficiencia al 100%. Este esquema permite obtener un escenario como el presentado en la Figura 31 para el core 3.

$$K^{n-1} * \text{Max}\ (TiempoComm(p)(w)) \leq\ (K-2)^n *\ TiempoCpt \qquad \text{(E8)}$$

Una vez definida la condición y la restricción del modelo, se procede a encontrar el valor ideal de K utilizando la ecuación E7. Para calcular este valor, se puede partir de la ecuación E1 que permite calcular el valor de λ(p)(w). Este valor está en función del tiempo de comunicación $TiempoComm(p)(w)$ y el tiempo de cómputo $TiempoCpt$ de un tile por cada enlace presente. Al despejar el $TiempoComm(p)(w)$ nos queda como una función que depende de λ(p)(w) multiplicado por $TiempoCpt$.

La ecuación E7 se puede escribir en función del $TiempoCpt$, haciendo un despeje en la ecuación E1 de λ(p)(w) que está en función del $TiempoComm(p)(w)$ y $TiempoCpt$. Este despeje es sustituido por el $TiempoComm(p)(w)$ en la ecuación E8, teniendo como resultado la ecuación E9.

$$K^{n-1} * \text{Max}\ (\lambda(\text{p})(\text{w}) * TiempoCpt) * effic \geq\ (K-2)^n *\ TiempoCpt \qquad \text{(E9)}$$

Las ecuaciones iniciales del modelo, se verán afectadas por la dimensión del problema dependiendo de si es de una, dos o tres dimensiones. Por lo

tanto a continuación se describe el modelo para cada uno de los casos estudiados utilizando como base las ecuaciones anteriormente descritas.

- **Aplicaciones con 1 dimensión**

Partiendo que se tiene una aplicación SPMD en línea con un tamaño de problema ***M***, y la dimensión ***n=1,*** el cálculo del número de cores a utilizar está expresado por la ecuación E10, donde se subdivide el tamaño del problema en *tiles* entre el tamaño del *supertile* ideal encontrado aplicando las estrategias de solapamiento.

$$Ncores = \frac{M}{ST} \tag{E10}$$

Para calcular el valor del *supertile* se sustituye la dimensión ***n=1*** de la aplicación en la ecuación E9, ésta permite calcular el valor ideal de ***K***. Por lo tanto, al hacer la sustitución y despejar el valor de ***K*** se genera la ecuación siguiente (ecuación E11).

$$K = \mathrm{Max}\left(\lambda(\mathrm{p})(\mathrm{w})\right) * effic - 2 \tag{E11}$$

Ambos valores de ***K*** y ***Ncores***, representan el valor ideal del *supertile* y el número de cores que mantienen la relación de máximo *speedup* con una eficiencia definida. Esta relación es para aplicaciones lineales cuyo patrón de comunicaciones es para dos vecinos.

- **Aplicaciones con 2 dimensiones**

De forma similar al caso de una dimensión, las aplicaciones de 2 dimensiones parten de un tamaño de problema M^2 y con un valor de dimensión ***n=2***, en este caso el número de cores está expresado por la ecuación E12.

$$Ncores = \frac{M^2}{ST} \quad \text{(E12)}$$

De forma similar al caso de una dimensión, el valor de ***K*** es encontrado utilizando la ecuación E9 pero igualándola a 0. Esta igualdad genera una ecuación de segundo grado que se presenta en la ecuación E13. Esta ecuación de segundo grado genera dos posibles soluciones para el valor ideal de K, las cuales pueden o no ser iguales. En este sentido, ambas soluciones deben ser substituidas en la ecuación E7 y E8 con el objetivo de validar si el valor obtenido cumple con las condiciones iniciales y la respectiva restricción.

$$K^2 - \left(4 + effic * \text{ Max}\left(\lambda(\text{p})(\text{w})\right)\right) * K + 4 = 0 \quad \text{(E13)}$$

- **Aplicaciones con 3 dimensiones**

El último caso analizado por el modelo es para aplicaciones de 3 dimensiones en este sentido tenemos un problema cubico conformado por un tamaño de M^3 y con un valor de dimensión ***n=3,*** en este caso el número de cores a utilizar esta expresado por la ecuación E14.

$$Ncores = \frac{M^3}{ST} \quad \text{(E14)}$$

Partiendo de la ecuación E9 para el cálculo de ***K*** y sustituyendo la dimensión, nos queda una ecuación de tercer grado, la cual se debe resolver para encontrar las tres posibles soluciones que dan el valor ideal de ***K***. Estas soluciones pueden ser al menos una o más reales o imaginarias, en este caso se deben utilizar las raíces reales para validarlas en el modelo con las ecuaciones E7 y E8.

$$K^3 - \left(6 + \text{Max}\left(\lambda(\text{p})(\text{w})\right) * effic\right) * K^2 + 12 * K - 8 = 0 \quad \text{(E15)}$$

Una vez obtenido el valor ideal de K para el caso que corresponde (dependiendo de la dimensión de la aplicación), se puede determinar el comportamiento de la aplicación SPMD para valores distintos del valor ideal. Para esto podemos despejar de la ecuación E10, E12 o E14 dependiendo de la dimensión, y dado un número de core podemos calcular la nueva K con el objetivo de determinar el tiempo de ejecución para ese número de cores. Este valor de ***K*** no corresponde al valor ideal que mantiene el máximo *speedup* con una eficiencia definida, pero si permite dar el comportamiento del tiempo de ejecución, *speedup* y eficiencia al aplicarlo en la ecuación E2.

3.3.5 Ejemplo de aplicación del modelo.

Para ilustrar el uso del modelo se usará un ejemplo de la aplicación de la transferencia de calor. Esta aplicación tiene un patrón de comunicaciones con cuatro vecinos y una vez realizado el proceso de caracterización, se obtuvieron los datos mostrados en la Tabla 5. Para el dato de las comunicaciones se ha considerado el peor caso presentado para el volumen de comunicación de cada *tile*.

El primer paso una vez obtenido los datos de caracterización, es aplicar el conjunto de ecuaciones que corresponda dependiendo de la dimensión de la aplicación. En este caso en particular es una aplicación de 2 dimensiones.

Tabla 5 Datos obtenidos de la caracterización aplicación de la transferencia de calor.

Aplicación	T. Cómputo $TiempoCpt$ (Seg)	T. Comunicación $TiempoComm(p)(w)$ (Seg)	Tamaño del Problema (M)	Eficiencia deseada (Effic)	Ratio Cómputo-Com (λ)
Transferencia de calor	2.10E-08	5.88E-05	9500	0.85	2800

Para calcular el valor ideal de cores y el valor de K se utiliza las ecuaciones E12 y E13 respectivamente, obteniendo así los resultados mostrados en la Tabla 6. Estos resultados muestran los valores calculados para el tiempo aproximado de una iteración (considerando que los recursos de cómputo están dedicados para la ejecución), para el tamaño ideal

calculado que mantiene el máximo *speedup* con una eficiencia sobre un umbral.

Tabla 6 Resultados obtenidos aplicando el modelo de 2 dimensiones

Ncores	K	Cómputo de Borde (Seg)	Cómputo Interno (Seg)	Comunicación de Borde (Seg)	Tiempo de Ejecución (Iter) (Seg)
16	2375	1.99E-04	1.18E-01	1.99E-04	0.139849

Al obtener el resultado del caso ideal, se procede a predecir el comportamiento de la aplicación en relación al tiempo de ejecución. Para conocer este tiempo se debe calcular el valor de ***K*** que debe ser asignado dependiendo del número de cores con que se desea estimar el comportamiento. Esto se realiza haciendo un despeje en la ecuación E10, E12 o E14 dependiendo de la dimensión de la aplicación. Una vez obtenido los resultados, se puede visualizar un comportamiento calculado de la aplicación en la sección de la aplicación donde se aplicará la metodología.

3.4 Fase de *mapping*

El propósito de esta fase es aplicar una distribución espacial de los *supertile* en los cores de ejecución con el objetivo de gestionar la heterogeneidad de comunicaciones del entorno multicore. Por lo tanto, el objetivo de esta fase es diseñar una estrategia para ubicar los procesos MPI en los cores que van a ejecutar los *supertiles*. Es importante recordar que solo un *supertile* será ejecutado por core. Esta estrategia de *mapping* se incluye para minimizar los efectos de las comunicaciones, esto es realizado aplicando un método para ponderar cada comunicación y minimizar el número de comunicaciones por los canales más lentos.

Este proceso de *mapping* debe ser capaz de seleccionar el core más adecuado y ubicarlo de acuerdo a una distribución lógica de los procesos. Este mapeo de los procesos debe realizarse de forma similar a la realizada en la fase de caracterización de las comunicaciones, donde se obtuvo el comportamiento de los niveles de comunicación de acuerdo al patrón de la aplicación obtenido con PAS2P [65]. Asimismo, este proceso de *mapping* debe ser igual al usado para la caracterización de las comunicaciones, con

el fin de no tener variaciones entre la ejecución de la caracterización y luego la de la aplicación. Esto permite tener valores más reales del comportamiento y de esta forma predecir mejor el comportamiento de la aplicación.

La fase de *mapping* se divide en tres puntos claves, el primero está asociado a la realización de una distribución lógica de los procesos MPI, seguido por un proceso de afinidad de los cores donde se asocia el proceso MPI a un core específico dentro de un nodo donde se ejecutará este proceso y por último se realiza el proceso de división de los *tiles* y asignación a cada core.

La Figura 33 muestra una distribución de procesos MPI dentro de una topología de comunicación con cuatro vecinos y para una aplicación de dos dimensiones. Cada *supertile* es asignado a su core de ejecución, donde se puede detallar que algunos cores pueden tener mayor número de comunicaciones de borde dependiendo de la ubicación dentro de la distribución lógica de procesos. Esta distribución retorna unas coordenadas que luego serán traducidas para seleccionar en cual core dentro del nodo debe ir cada proceso.

Además, para determinar la ubicación del proceso MPI en cada core se utiliza un proceso de afinidad. Es importante conocer que cuando una aplicación requiere una cantidad determinada de cores en una máquina para su ejecución, el sistema operativo puede seleccionar cualquier core dentro del nodo siempre y cuando éste esté disponible para la ejecución.

No obstante, existen métodos para seleccionar el core de ejecución dentro del nodo. Una forma de hacer esta afinidad es desde la propia aplicación, o utilizando funcionalidades de las librerías de paso de mensajes. En la metodología la planificación de la estrategia de *mapping* se fundamenta en la selección del core para disminuir los canales de comunicación como se describirá más adelante.

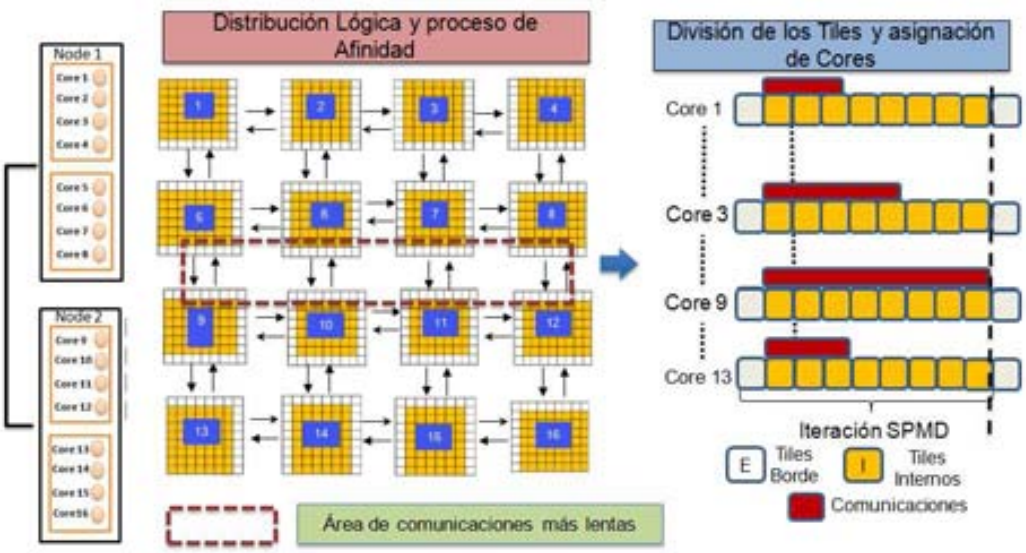


Figura 33 Distribución espacial del *supertile* y asignación del core de ejecución

Los canales de comunicaciones más lentos se han etiquetado en la fase de caracterización, esto permite variar el orden de ejecución de los mensajes en función de la distribución y de los canales, para minimizar la congestión de los mismos. Una vez identificado los cores de ejecución, el último paso es realizar el proceso de división de los *tiles* y su respectiva asignación a cada uno de los cores.

3.4.1 Distribución lógica de los cores

La función de la distribución lógica de procesos consiste en crear un identificador para cada proceso MPI, además de establecer una estrategia para minimizar el número de comunicaciones por el enlace más lento.

Dependiendo de la distribución y asignación de los *tiles* a cada core, los *supertile* pueden tener mayor cantidad de comunicaciones a través de los enlaces más lentos, ocasionando que el tiempo de comunicación de ese enlace se vea incrementado. Para realizar una correcta distribución se debe diferenciar entre la distribución física de los cores, y la distribución lógica de los *tiles*. En el primer caso, cada core tiene un identificador único dentro del nodo, que permite diferenciar a nivel físico cuál es el core que ejecutará el proceso MPI seleccionado. Por otra parte, la distribución lógica se define como un identificador lógico dentro de la aplicación que permite determinar

un conjunto de coordenadas, que hacen que se pueda identificar el *supertile* y el core que lo va a ejecutar.

Un ejemplo de lo anterior puede ser visualizado en la Figura 34, donde la distribución lógica de los *tiles* se ha realizado de forma secuencial; es decir, se van llenando los cores con su respectivo *supertile* de forma consecutiva y aplicando a su vez a nivel físico una afinidad de los cores igualmente de forma consecutiva. Este tipo de distribución puede ocasionar una sobrecarga o congestión dependiendo del esquema de comunicación de la aplicación. En un esquema de distribución secuencial el número de comunicaciones por el enlace más lento es mayor a una estrategia de agrupamientos.

La Figura 34, hace referencia a una ejecución en 3 nodos con 8 cores y donde se asigna un proceso MPI por core, estos procesos mantienen el patrón de comunicaciones a cuatro vecinos, excepto los cores que integran el borde de la distribución (ejemplo los cores identificados con 0, 8, 16, 24 entre otros) por estar en la frontera de la distribución. Si se analiza lo que ocurre en el enlace más lento (LAN), se puede detallar que los cores centrales tienen mayor número de comunicaciones (cores del 8 al 15 y del 16 al 23) que los cores de las fronteras (cores del 0 al 7 y del 24 al 31), lo que generar una mayor latencia, la cual debe ser considerada para aplicar el modelo. En este caso el número de comunicaciones globales por el enlace más lento para cada iteración de la aplicación es de 48 (Figura 34).

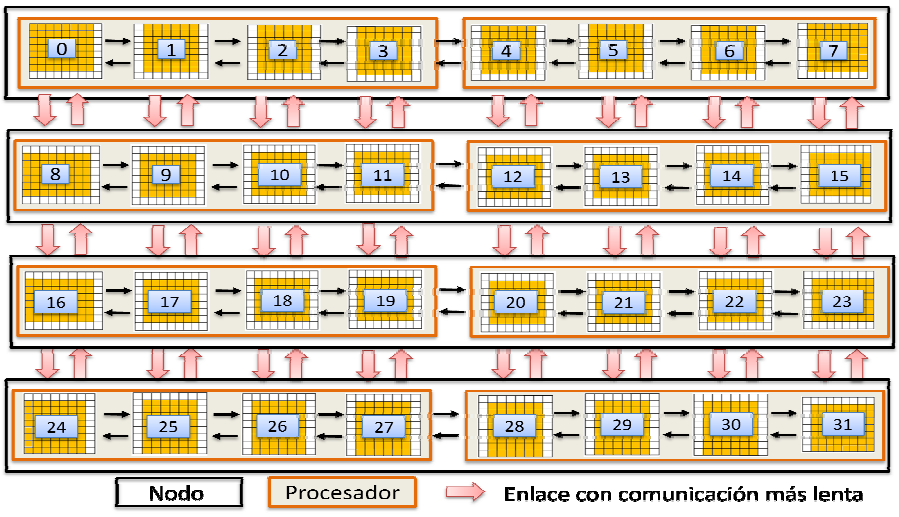


Figura 34 Distribución lógica secuencial de los *supertile* en los cores

Aunque esta distribución lógica secuencial para el cálculo del modelo no afecta los resultados (debido a que el tiempo de la ratio λ(p)(w) es calculado con los tiempos medidos de latencia para este comportamiento), se pueden proponer otras estrategias de agrupamiento que permitan que los *supertiles* se puedan ubicar de una manera más eficiente entre los procesos MPI, dependiendo de la arquitectura y del número de cores por chip.

Esto es realizado con el objetivo de minimizar el número de comunicaciones por el enlace más lento y evitar la congestión del enlace de comunicación que en caso de presentar saturación, los tiempos de latencias pueden variar considerablemente.

En este sentido, la Figura 35 muestra un ejemplo de cómo se pueden agrupar los *supertiles* considerando la arquitectura física del entorno de ejecución. En este ejemplo se puede detallar que el número de comunicaciones internas dentro del nodo incrementa pero a su vez el número de comunicaciones por el enlace más lento se disminuye. En este caso particular tenemos que el sistema tendría 24 comunicaciones por el enlace más lento, esto refleja un 50% menos comunicaciones por el enlace que delimita la iteración de la aplicación SPMD. Este porcentaje puede variar dependiendo del número de cores a utilizar, por ejemplo entre mayor sea el número de cores de ejecución este porcentaje de ahorro puede ser mayor, por lo que esta estrategia minimiza la cantidad de comunicaciones por el enlace más lento.

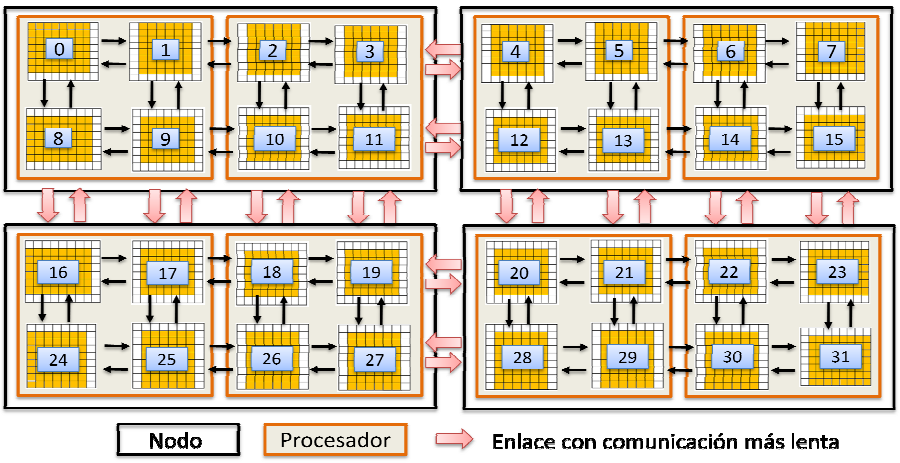


Figura 35 Distribución lógica por agrupaciones de los *supertile* en los cores

Este esquema de agrupaciones será más eficiente a medida que el número de *supertiles* incremente. Por ejemplo, supóngase que se desea ejecutar una aplicación en un entorno a mayor escala con la misma configuración del ejemplo anterior; es decir, una arquitectura integrada por nodos con 2 Quad cores pero esta vez ejecutando la aplicación en 32 nodos o 256 cores. Si se parte del modelo de distribución de *tiles* explicado anteriormente, el tamaño del problema se definía como M^n donde la n determina la dimensión de la aplicación. Por ejemplo, en el caso de una aplicación de 2 dimensiones (X,Y) la distribución de cores será de una malla bidimensional de cores distribuida por 16x16 cores, para distribuir los 256 *supertiles*. Esto significa que si se realiza una distribución secuencial tendremos cores con tres comunicaciones al enlace más lento como se puede apreciar en la Figura 36.

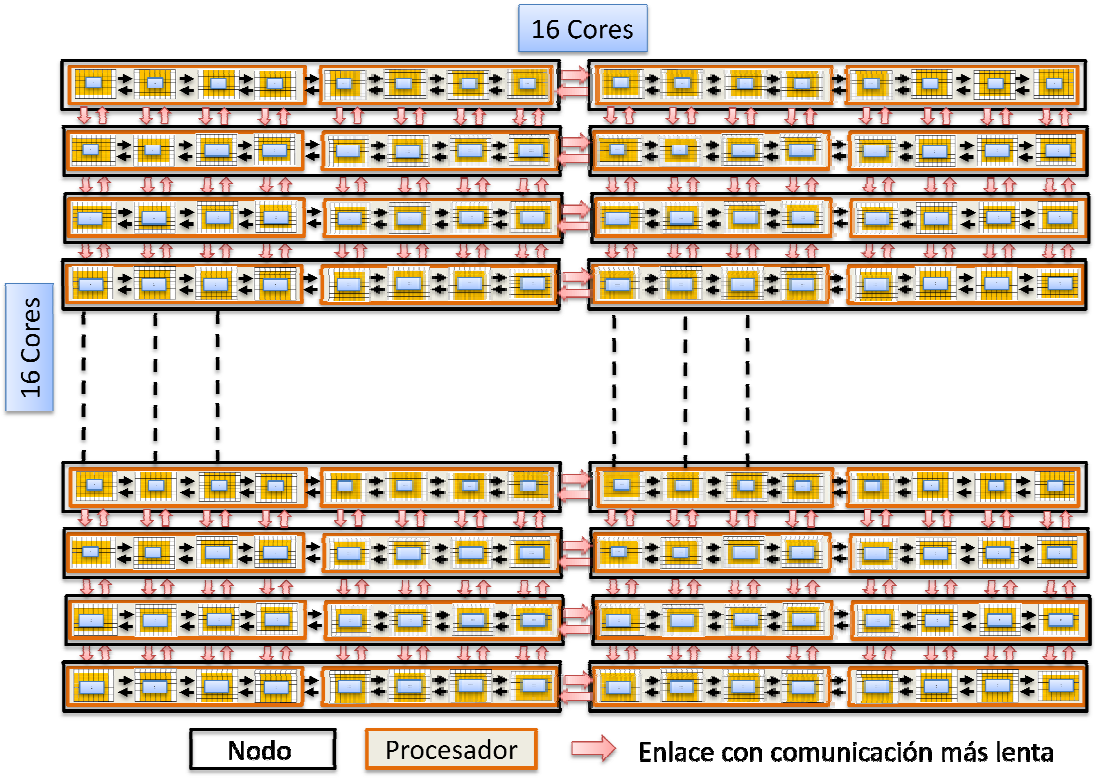


Figura 36 Comunicaciones en distribución lógica secuencial para un problema con 256 cores.

Esta distribución secuencial genera un mayor número de comunicaciones por el enlace más lento. Además, algunos cores tendrán mayor cantidad de transmisiones y agrupando estas comunicaciones generan mayor latencia al

enlace de comunicaciones. Para este caso el número de comunicaciones globales por el enlace más lento es de 512 (32 comunicaciones por 15 líneas de cores más 32 de los cores centrales) por iteración y tiene 30 cores que mantienen 3 comunicaciones (Figura 36).

No obstante, si se realiza una estrategia de agrupaciones con el fin de gestionar el nivel de comunicaciones de forma eficiente, se puede ganar en dos aspectos el primero en el número de comunicaciones globales por iteración a través del enlace más lento y el segundo porque el número de comunicaciones por core al enlace más lento es menor. Ambas mejoras pueden evidenciarse en la Figura 37, donde el número de comunicaciones globales por el enlace más lento es de un valor de 320 lo que refleja una reducción aproximada de 38% en el número de comunicaciones. Además, se puede observar en la figura que no existen cores con comunicaciones a tres cores vecinos por el enlace más lento.

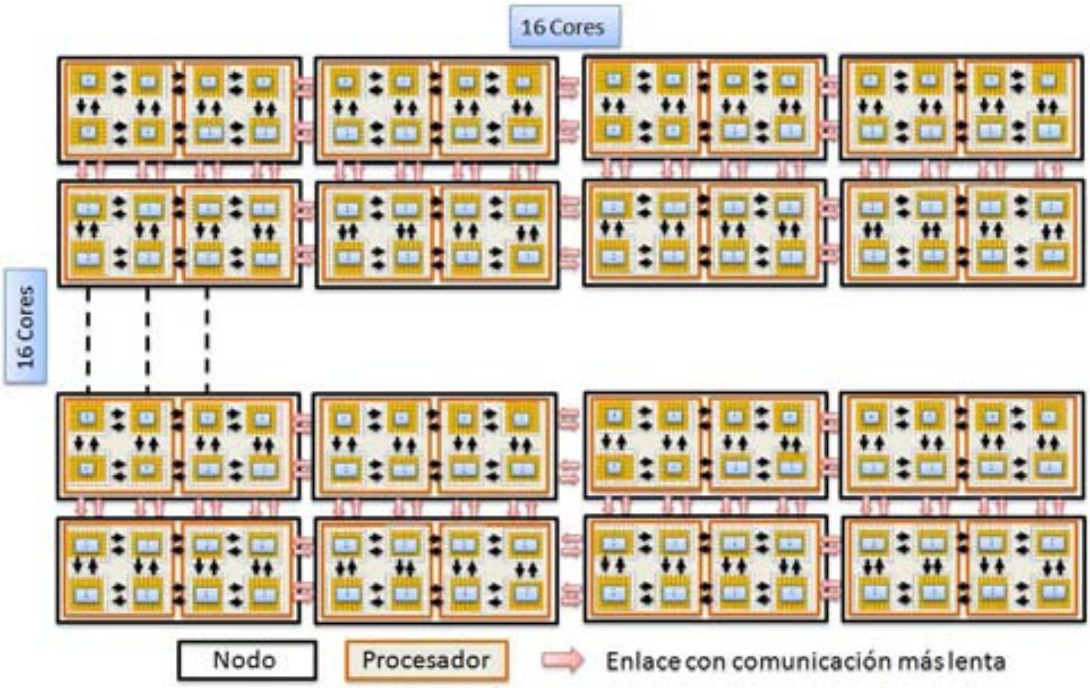


Figura 37 Comunicaciones en distribución lógica por agrupaciones para un problema con 256 cores.

Por otra parte, el incremento en el número de comunicaciones en el core usando el enlace más lento tiene un efecto motivado a la latencia creada. Esto origina que el valor de la ratio cómputo comunicación λ(p)(w) sea

mayor y por ende el número de *tiles* que se asignan a los *supertiles* es mayor.

Para el aumento del *speedup* de la aplicación, lo ideal es asignar la menor cantidad de *tiles* posibles; no obstante, cuando se desea una ejecución rápida y eficiente, este número de *tiles* debe mantener una estrecha relación entre el tiempo de la comunicación más lenta y el tiempo de cómputo interno. La Figura 38 muestra el efecto de tener cores con dos o tres comunicaciones. En este caso el tiempo de latencia de un core es mayor al de los cores con dos comunicaciones.

Esta variación en la práctica para el tipo de red que utilizamos, puede estar alrededor del 7% de ganancia en el tiempo de comunicación para el mismo volumen de datos, cuando se utiliza un esquema donde el número de comunicaciones por core se minimice. Es por esto que el modelo de distribución de *supertile* por agrupaciones es una solución que adiciona una mejora a la métrica de la eficiencia de la aplicación.

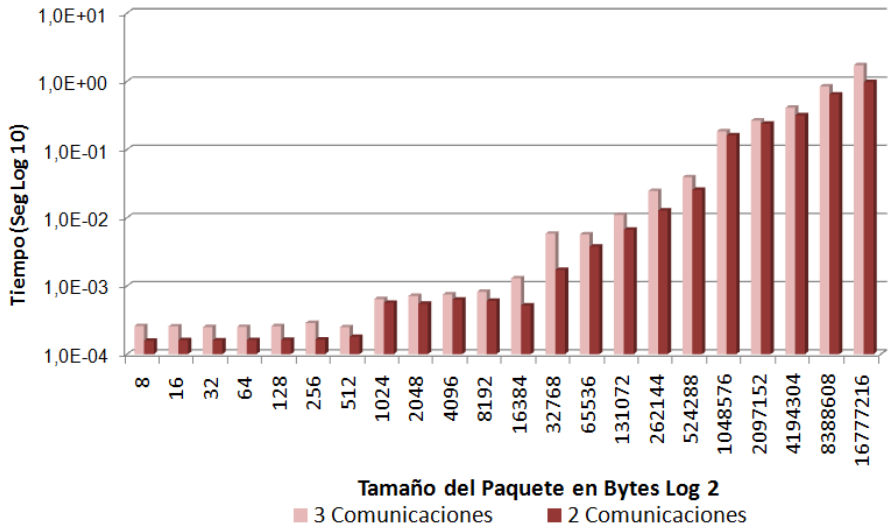


Figura 38 Comparación de cores con dos y tres comunicaciones

Una vez definido el modelo de distribución lógico de la aplicación, el siguiente paso es aplicar el proceso de afinidad de los cores que ejecutarán cada *supertile*. Esta ejecución es realizada con la información obtenida en este proceso de distribución lógica.

3.4.2 Modelo de afinidad de procesos

El objetivo de aplicar una técnica de afinidad es asignar la distribución lógica de procesos calculada a los recursos físicos (cores). Esto se realiza para obtener las mejores prestaciones de la arquitectura multicore a utilizar [32]. Desde el punto de vista práctico, el proceso de afinidad se compone de dos etapas. La primera es una asignación del conjunto de *supertiles* a los nodos de ejecución y la segunda consiste en ubicar los procesos en los cores específicos para controlar la ubicación de los *supertiles* en los mismos.

Al hacer esta planificación espacial, se puede determinar de forma sencilla cómo se realizarán las comunicaciones de los vecinos, debido a que existe un conocimiento exacto del mapeo de los procesos MPI a los cores de ejecución. Asimismo, al aplicar una técnica de afinidad basada en la gestión de comunicaciones, se logra una ejecución de la aplicación SPMD más estable debido a que las comunicaciones entre los cores están controladas por el modelo de distribución lógica [36].

Este proceso de afinidad permite asociar la distribución lógica de la aplicación con una planificación de la arquitectura de ejecución, trayendo como beneficio que la aplicación este mapeada lo mejor posible de acuerdo a las características de la máquina a utilizar [5]. La Figura 39 muestra un script de ejemplo de afinidad de procesos MPI, realizado a través de la librería de paso de mensajes y su respectiva asignación a los nodos y procesadores de ejecución. En la primera etapa, este esquema de mapeo de procesos permite realizar agrupaciones y se puede definir cuales procesos MPI de la distribución lógica serán asignados al nodo.

Una vez ubicados los procesos MPI de forma correcta en los nodos, el siguiente paso es realizar el proceso de asignación de los cores considerando la distribución lógica definida. Este mapeo es realizado con un sistema de funciones de la librería sched.h para asociar la unión entre proceso MPI y el core que va a ejecutar el *supertile*.

Afinidad de procesos MPI a través de la librería de paso de mensaje

```
1:  rank  0=Nodo1 slot=0:*
2:  rank  1=Nodo1 slot=0:*
3:  rank  2=Nodo1 slot=1:*
4:  rank  3=Nodo1 slot=1:*
5:  rank  4=Nodo2 slot=0:*
6:  rank  5=Nodo2 slot=0:*
7:  rank  6=Nodo2 slot=1:*
8:  rank  7=Nodo2 slot=1:*
9:  rank  8=Nodo1 slot=0:*
10: rank  9=Nodo1 slot=0:*
11: rank 10=Nodo1 slot=1:*
12: rank 11=Nodo1 slot=1:*
13: rank 12=Nodo2 slot=0:*
14: ……..
```

Figura 39 Afinidad de procesos MPI para asociar las agrupaciones lógicas de procesos

Esta segunda etapa de afinidad se ejecuta utilizando el pid o identificador de proceso para mover el proceso MPI y por ende el *supertile* al core específico. La Figura 40 muestra la distribución lógica y cómo la misma se puede mapear en el entorno de ejecución.

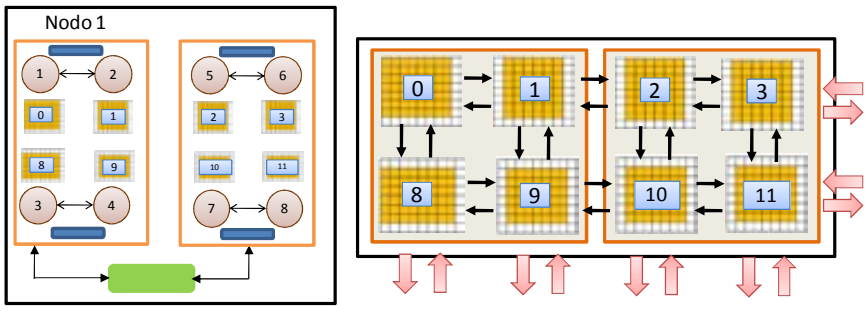


Figura 40 Afinidad de procesos a cores usando funciones del sistema

Esta fase de afinidad permite ubicar los procesos MPI y por ende los *supertile* en el core adecuado para hacer el proceso de planificación temporal o fase de *scheduling*.

3.4.3 Modelo de división y distribución de *tiles*

La distribución lógica determina las coordenadas donde estarán ubicados lógicamente los *supertiles*. Estas coordenadas son definidas como parte de la topología de distribución que se utilizará para la ejecución. La Figura 41

muestra la asignación de las dos coordenadas (X,Y) las cuales permiten asociar un proceso MPI con un *supertile*, esta asignación permite crear la división de los tiles dentro de la aplicación, con el objetivo realizar la creación de los *supertiles.* Estos cuales son creados de acuerdo al tamaño definido en la fase de modelo de distribución de *tiles*. Además, las coordenadas asignadas a cada *supertile* permiten determinar los procesos vecinos de acuerdo al patrón de comunicaciones definido en la aplicación. Estas comunicaciones se identifican para tener mayor control sobre las comunicaciones.

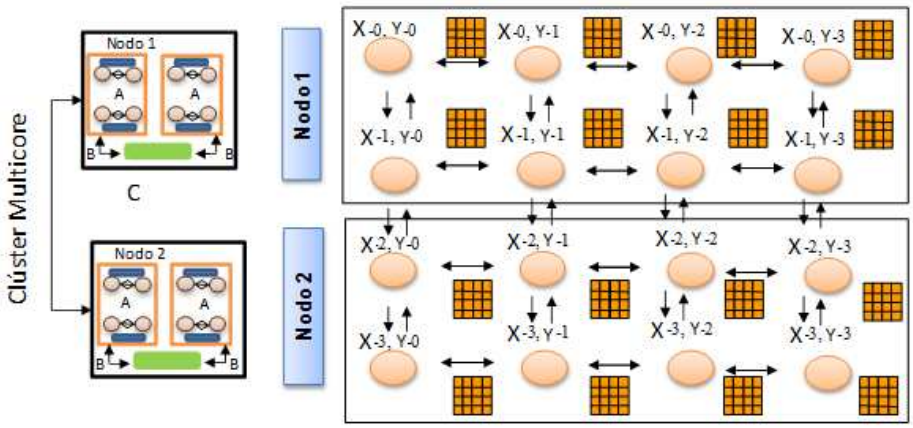


Figura 41 Modelo de división y distribución de Supertiles

Al crear los *supertiles*, se debe hacer la planificación espacial en la arquitectura de modo de hacer una integración entre la distribución lógica y la arquitectura de la máquina.

3.5 Fase de *scheduling*

La última fase de la metodología es la planificación temporal del orden de ejecución de los *supertiles*. Esta fase de *scheduling* se divide en dos partes principales, la primera está asociada a la asignación de las prioridades de ejecución y la segunda es la que permite administrar la estrategia de solapamiento entre el cómputo de los *tiles* internos y los *tiles* de borde.

3.5.1 Prioridades de ejecución

Para hacer una ejecución eficiente de la aplicación SPMD, el orden de ejecución de los *tiles* debe ser planificado a la hora de su ejecución. Por lo

tanto, a los *tiles* se le debe asignar una prioridad de ejecución que permita aplicar la estrategia de solapamiento que minimiza el efecto de las comunicaciones.

En este sentido, la asignación de prioridades se realiza con la siguiente política: los *tiles* de borde se asignan con prioridad más alta de ejecución (prioridad=1), debido a que estos *tiles* tienen comunicaciones a los elementos externos del core y pueden crear problemas de desbalanceo de comunicaciones. Estos *tiles* que llamamos de borde son guardados en buffers de ejecución con el objetivo de ejecutarlos primero. Estos búferes son actualizados en cada iteración. La otra asignación es la prioridad para los *tiles* internos, los cuales son identificados como la prioridad 2. Los *tiles* internos se ejecutarán de forma solapada mientras las comunicaciones de los *tiles* de borde se ejecutan.

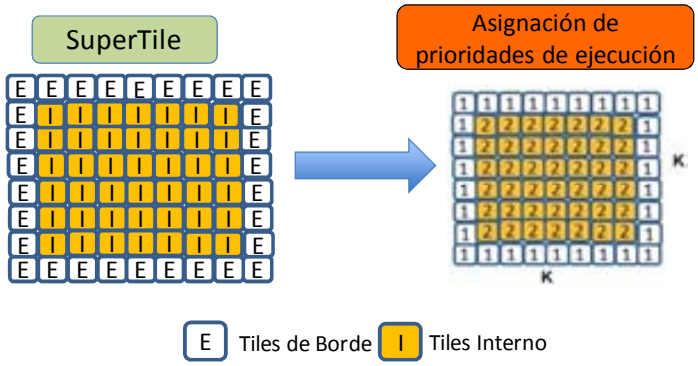


Figura 42 Asignación de prioridades de ejecución

La Figura 42 muestra un ejemplo de la asignación de prioridades entre los distintos *tiles* del *supertile*. Además de estas prioridades de ejecución en los *tiles*, esta identificación permitirá la utilización de la estrategia de solapamiento.

3.5.2 Estrategia de solapamiento

Para la metodología esta fase de *scheduling* es clave. El modelo analítico se fundamenta en una estrategia de solapamiento para minimizar el efecto de los diversos enlaces de comunicación del clúster con procesadores multicore. En este sentido, la Figura 43 ilustra la estrategia de solapamiento de los *tiles* internos y las comunicaciones de los *tiles* de borde.

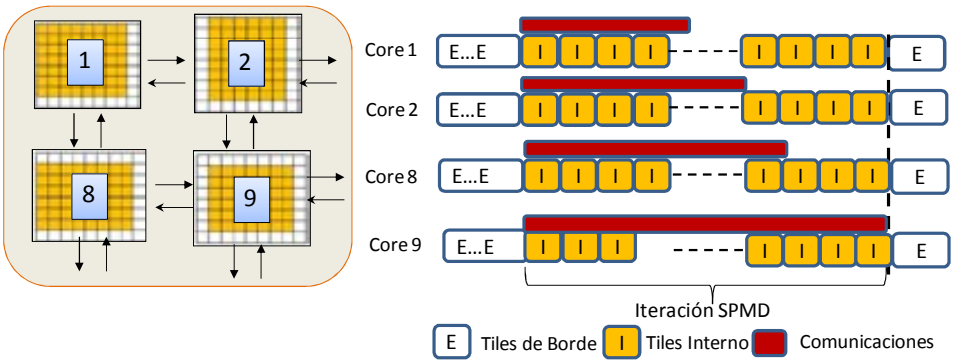


Figura 43 Estrategia de solapamiento

Como se puede apreciar en la imagen la iteración de la aplicación SPMD está delimitada por la comunicación más lenta, por lo que a los demás cores de ejecución se les debe asignar igual cantidad de *tiles* de ejecución para mantener el equilibrio entre todos los cores y así mejorar la eficiencia de la aplicación.

Aunque algunos cores con este esquema tengan comunicaciones más cortas, esto no significa que se deba calcular un valor de ***K*** para cada core; por el contrario se propone calcular un único valor para todo el sistema, considerando la mayor ratio entre el cómputo y la comunicación. Esto permite que todos los *supertiles* duren aproximadamente el mismo tiempo de ejecución y por ende sincronice el sistema de cómputo.

Este proceso de solapamiento es realizado utilizando estrategias de comunicación asíncronas, para que permita hacer el solapamiento entre el cómputo y la comunicación. Ambas funciones pueden realizarse de forma simultánea debido a que se utiliza dos *threads*, uno que controla las comunicaciones y otro que controla el cómputo. El proceso de solapamiento debe controlar que tanto el cómputo de los *tiles* internos como el de las comunicaciones de borde terminen en tiempos similares para poder mantener el éxito del modelo analítico.

Este proceso de solapamiento permite minimizar los efectos de las comunicaciones; además, beneficia a la eficiencia al hacer un manejo adecuado de las ineficiencias provocadas por los enlaces de comunicaciones.

3.6 Ejemplo de evaluación

Para verificar los resultados de la metodología, continuaremos con el ejemplo que se ha descrito a lo largo de este capítulo. Una vez aplicado el modelo de distribución de *tiles,* y los *supertiles* han sido asignado a cada core de ejecución (aplicando los resultados obtenidos en la Tabla 6), el siguiente paso es evaluar el comportamiento de la estrategia de solapamiento. En este sentido, la Figura 44 muestra la traza de ejecución generada con la herramienta *Jumpshot* [67] de la aplicación de transferencia de calor sobre el clúster integrado por nodos Quad core. Como se puede apreciar en la traza las comunicaciones están totalmente solapadas con el cómputo interno. Esto permite que el efecto de la ineficiencia de las comunicaciones tenga poco impacto en el tiempo de ejecución de la aplicación.

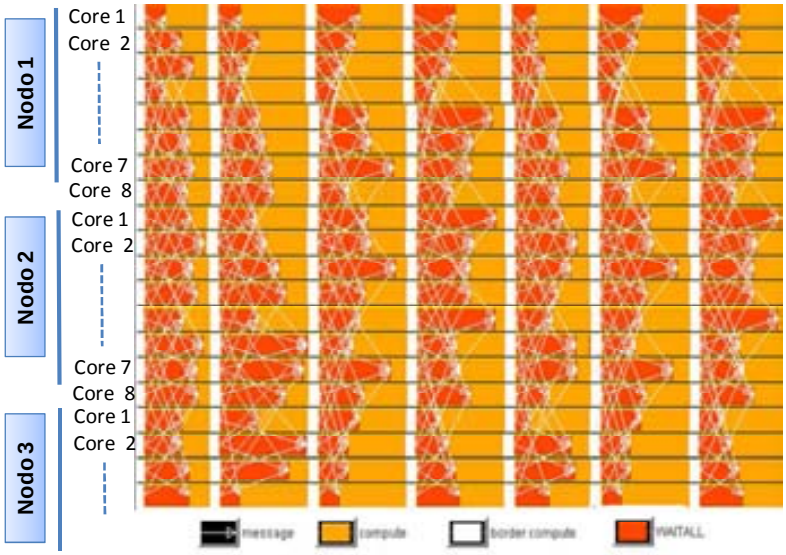


Figura 44 Traza de ejecución aplicación de Transferencia de Calor

Para detallar el impacto de la metodología en la eficiencia para esta aplicación y tamaño de problema, se hace un análisis del comportamiento de la eficiencia. Para este tamaño de *tiles*, el valor teórico obtenido del número de cores ideal para la ejecución es de 16, por lo tanto se realiza la ejecución con ese número de cores y para comparar se hacen dos ejecuciones, tanto para la aplicación sin utilizar la metodología como aplicando la metodología. Los resultados de la eficiencia mostrados en la Figura 45 muestran que la

versión usando la metodología ofrece mejoras a nivel de eficiencia. Además, el margen de error para este caso particular es inferior al 4%.

Asimismo se puede apreciar qué ocurre con la eficiencia cuando se varía el número de cores, la eficiencia siempre mejora cuando se aplica la metodología, y se respeta la eficiencia solicitada por el usuario cuando se utiliza el número de cores calculados. Evaluando en diferentes números de cores las ganancias en eficiencia son considerables.

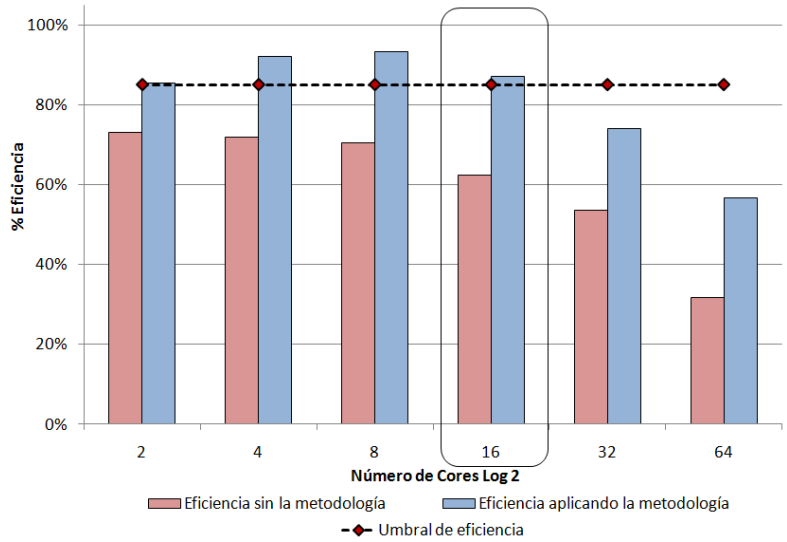


Figura 45 Análisis de eficiencia aplicación de Transferencia de Calor

Este mismo análisis se debe realizar para la métrica que se busca maximizar, en este caso es el *speedup*. La Figura 46 ilustra el comportamiento del *speedup,* igualmente se muestran los valores obtenidos para la versión original de la aplicación y la versión aplicando la metodología. En este caso se observa un crecimiento aproximadamente lineal con respecto al *speedup* teórico para la versión que aplica la metodología. Este crecimiento se mantiene hasta que se ejecuta con el mismo número de cores que se determinó con el método de ejecución. Si se observa la figura el *speedup* sigue creciendo por lo que se puede conseguir una aceleración mayor. No obstante, considerando la eficiencia, vemos que la misma funciona como una restricción de sistema.

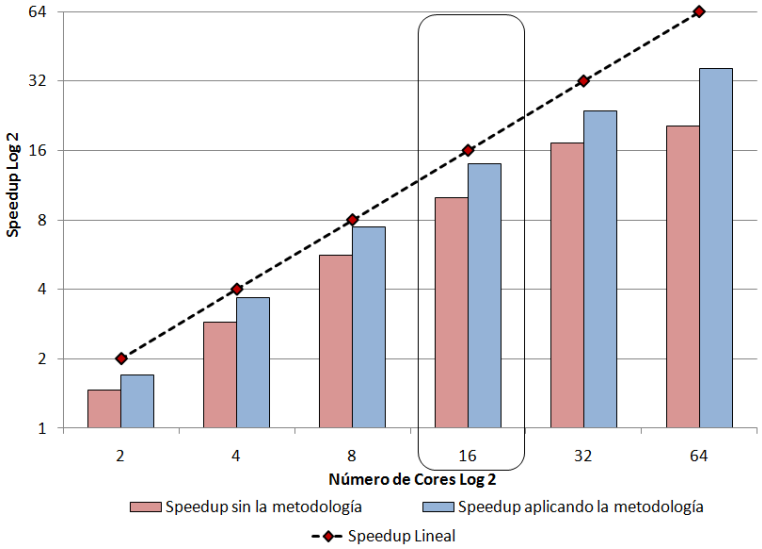


Figura 46 Análisis de *speedup* aplicación de transferencia de Calor

Como se puede evidenciar en este ejemplo, la metodología puede generar grandes ganancias tanto en *speedup* como en eficiencia como se verá con más detalle en el capítulo de validación experimental.

3.7 Conclusiones

Este capítulo ha descrito la metodología propuesta para lograr un ejecución eficiente de aplicaciones SPMD en clústeres con procesadores multicore. La misma se basa en cuatro fases: caracterización, modelo de distribución de *tiles*, *mapping* y *scheduling*. Esta metodología está enfocada en encontrar el número ideal de cores y del tamaño ideal del *supertile* que mantiene la máxima aceleración computacional mientras la eficiencia está por encima de un umbral definido. El éxito de la metodología esta fundamentalmente definido por el determinismo de las aplicaciones SPMD que permiten que se pueda hacer una gestión eficiente de las comunicaciones en los clústeres con procesadores multicore.

Capítulo 4

Validación experimental de la Metodología

Resumen

La ejecución eficiente de aplicaciones SPMD de paso de mensajes en clústeres con procesadores multicore de forma eficiente se convierte en un punto fundamental que la metodología busca resolver. En este sentido, el presente capítulo describe la validación experimental de la metodología con diferentes clústeres con procesadores multicore y con varias aplicaciones científicas. Al evaluar los resultados obtenidos y las mejoras tanto en la eficiencia como en el speedup de las aplicaciones testeadas, la experimentación ha demostrado, cómo se pueden mejorar las métricas de rendimiento aplicando la metodología de ejecución eficiente. Este capítulo de validación experimental, se divide en tres partes principales: una evaluación de caracterización de cómputo y comunicación, la cual permite determinar el comportamiento del entorno de ejecución y de la aplicación; la aplicación del modelo de distribución de tiles dando una solución ideal de forma analítica relacionada con el valor ideal de tiles y cores necesarios para mantener la relación de máximo speedup con la eficiencia mantenida sobre un umbral definido y finalmente se ilustran las mejoras de las métricas de rendimiento obtenidas utilizando la metodología. Dentro de los resultados se pueden destacar mejoras en las métricas de rendimiento de hasta el 39% en la eficiencia para el mejor de los casos de las aplicaciones validadas.

4.1 Introducción

Al aplicar la metodología, se obtiene de una forma guiada, cómo una aplicación desarrollada inicialmente para ser ejecutada en un entorno monocore, se puede adaptar a un entorno multicore de forma eficiente. Esta adaptación permite que la aplicación pueda hacer un mejor uso de los recursos con el fin de mejorar las métricas de rendimiento como el tiempo de ejecución, el *speedup* y la eficiencia.

Hemos visto que para aplicar la metodología es importante obtener las características principales de la aplicación y del entorno, las cuales permiten aplicar un modelo analítico para encontrar el tamaño ideal del *supertile.* El *supertile* mantiene una estrecha relación entre el cómputo interno y las comunicaciones de los bordes, gestionando las comunicaciones de forma eficiente y además, minimizando las comunicaciones por los enlaces que comprometen la ejecución de la aplicación.

A través del modelo analítico incluido en la metodología, se pueden predecir ciertas características de la aplicación como el tiempo de ejecución, el *speedup* y la eficiencia de la aplicación para un entorno con procesadores multicore caracterizado. En este sentido, la primera parte de este capítulo describe cómo se seleccionaron diversos entornos con procesadores multicore y un conjunto de aplicaciones científicas para validar el modelo analítico.

Entre las aplicaciones utilizadas se encuentran la de Transferencia de Calor [64], la Ecuación de la Onda, la Ecuación de Laplace utilizando métodos de diferencias finitas para la resolución de problemas elípticos [23] y dos aplicaciones integradas en la suite de métodos de multi-fase de dinámicas de fluidos Mplabs [L7]. Las primeras tres aplicaciones serán utilizadas para validar la metodología en diferentes arquitecturas multicore con el fin de comprobar el alcance del objetivo planteado por la metodología de obtener el máximo *speedup* con una eficiencia definida; las otras dos aplicaciones de la suite MPlabs serán utilizadas para evaluar la combinación de la escalabilidad y la eficiencia de aplicaciones SPMD en clústeres con

procesadores multicore de gran escala, por lo que se realizará una descripción en profundidad en el siguiente capítulo.

El presente capítulo se ha organizado en tres partes. La primera es la descripción de cada una de las aplicaciones, así como la de cada entorno de ejecución multicore utilizado. Además, se describirá el proceso de caracterización que permite determinar el comportamiento del entorno de ejecución y de la aplicación.

Luego se hace la evaluación analítica de cada aplicación, para ello se analiza un conjunto de tamaños definidos, donde se describe la evaluación de las aplicaciones utilizando el modelo analítico. Este modelo permite predecir de forma analítica el tiempo de ejecución de la aplicación o de la porción de la aplicación donde se ha aplicado la metodología. Esta predicción conlleva a un pequeño margen de error que en los peores casos es menor a +/- 5% sobre el tiempo de ejecución real de la aplicación utilizando la metodología.

Finalmente se realiza la validación de los resultados obtenidos en ambas formas de ejecución (antes y después de la aplicación de la metodología). Esta validación es realizada para verificar las mejoras en las métricas de eficiencia y *speedup*, con el objetivo de mostrar los resultados experimentales para alcanzar el máximo *speedup* con una eficiencia definida.

4.2 Entornos de ejecución

Para la validación de la metodología, se han utilizado diferentes arquitecturas multicore con el fin de visualizar y validar la efectividad de la misma a la hora de buscar la relación entre el máximo *speedup* y la eficiencia definida. Específicamente, se han utilizado tres clústeres cuyas descripciones se aprecian en la Tabla 7.

Tabla 7 Características de los clústeres utilizados para la validación experimental

Cluster	Hardware	Software
DELL	8 Nodos, con pocesador 2 x Quad-Core Intel(R) Xeon(R) E5430 de 2.66GHz, 2x6MB de memoria cache L2, 16 GB de Memoria RAM Fully Buffered DIMMs (FBD) 667MHz, SAS6ir (H/W based) RAID 1 120GB 2.5" SAS / Plain 2.5" SATA (7.2k rpm): 80GB, una red de interconexión Gigabit Ethernet	Sistema operativo Linux Red Hat Enterprise Linux Server release 5.5, Librería de paso de Mensajes OpenMPI 1.4.1, gcc (GCC) y gfortran 4.1.2,
IBM	32 nodos, con procesador 2 x Dual-Core Intel(R) Xeon(R) 5160 3.00GHz 4MB L2 (2x2), 12 GB de Memoría RAM Fully Buffered DIMM 667 MHz, Hot-swap SAS Controller 160GB SATA Disk y una red de interconexión Gigabit Ethernet.	Sistema operativo Linux, Librería de paso de Mensajes OpenMPI 1.4.1, gcc y gfortran
AMD	8 Nodos con procesador AMD Athlon(tm) 64 X2 Dual Core Processor 3800+, 2GHz, memoria cache de 512KB L2 cache, 2Gb de memoria RAM, y red de interconexión Gigabit Ethernet	Sistema operativo Linux Ubuntu 9.10 Librería de paso de Mensajes OpenMPI 1.4.1

Como se puede apreciar en la Tabla 7, los clústeres tienen características diferentes, tanto a nivel de la arquitectura como en el número de cores por nodo. Esta variación permite que la metodología sea verificada con diversos niveles de comunicaciones, los cuales estarán asociados a la arquitectura de ejecución utilizada.

4.3 Aplicaciones científicas

Una vez descritos los entornos de ejecución para la validación de la metodología, se deben definir las aplicaciones SPMD que serán utilizadas. Estas aplicaciones deben cumplir con las características definidas para poder aplicar la metodología. En este sentido, las aplicaciones utilizadas

deben ser locales, regulares, estáticas, y de dos o tres dimensiones. Además, las aplicaciones deben ser desarrolladas bajo librería de paso de mensajes para ejecutar el proceso de comunicación, como se definió en el apartado de la metodología.

A continuación se hace una descripción general de las aplicaciones utilizadas:

- Transferencia de Calor [64]: Esta aplicación se encarga de calcular la difusión de calor en un cuerpo. Para realizar esto, utiliza un método de diferencias finitas que permite adaptar el valor para la siguiente iteración de la aplicación. Es una aplicación que está integrada por una malla bidimensional y que ejecuta un patrón de cuatro comunicaciones entre los *tiles* vecinos.
- Ecuación de la Onda [L13]: Es una aplicación que determina el comportamiento de una onda en un sistema de dos dimensiones utilizando métodos de diferencias finitas de aproximación. Al igual que la aplicación anterior mantiene un patrón de comunicaciones de cuatro vecinos.
- Ecuación de Laplace [23]: Es una aplicación que resuelve problemas elípticos de dos dimensiones utilizando la ecuación de Laplace con diferencias finitas. Esta aplicación está compuesta por un conjunto de *tiles* de manera uniforme, integrados en una malla de dos dimensiones. El patrón de comunicaciones entre los *tiles* es de cuatro vecinos.
- LL-2D-STD-MPI [L7]: Es una aplicación de dinámicas de fluidos que calcula la formulación de Lee& Lin, simulando el flujo de dos fases con densidad arbitraria y radios de viscosidad. Está principalmente integrada por tres fases. Las primeras dos fases (preestream, poststream) son de cómputo intensivo y la última es la fase de intercambio (hydrodinamic, stream), donde se realiza el proceso de comunicación y a la cual se le aplica la metodología. Esta aplicación, tiene un patrón de comunicaciones de 8 vecinos, por lo que cada *tile* necesita cierta información de todos los *tiles* vecinos. Una explicación en profundidad de esta aplicación se detallará en el siguiente capítulo.

4.3 Resultados experimentales de la fase de caracterización

La fase de caracterización permite obtener valores reales de los *tiles* entre los diferentes enlaces de comunicación de la arquitectura tanto a nivel de cómputo como a nivel de comunicaciones.

Es importante identificar de forma correcta los niveles de comunicación no sólo por la arquitectura sino también por el número de comunicaciones que se realizan a través del enlace identificado como el más lento. El determinar el número de comunicaciones por core para cada enlace, permite identificar el tiempo de latencia de cada enlace, para luego seleccionar el que posee mayor latencia de acuerdo a los resultados de caracterización.

Esta caracterización debe ser realizada con un clúster dedicado, controlado y monitorizado. Los valores obtenidos son el promedio de mil repeticiones por cada volumen de mensaje a caracterizar. Además, se debe utilizar el conjunto de cores disponibles en el clúster con el objetivo de visualizar los tiempos de latencias y los puntos de saturación de cada uno de los enlaces caracterizados, para luego obtener los valores más precisos de la arquitectura de comunicaciones que usará la aplicación.

La Figura 47 muestra los tres niveles de rutas de comunicaciones que componen el clúster IBM. Dos de estos niveles se ejecutan a través de la arquitectura interna del nodo y el otro a través de la red de comunicaciones. Esta caracterización se ha realizado manteniendo el patrón de comunicaciones de la aplicación de la Onda que se ha obtenido utilizando la herramienta PAS2P [65].

El patrón de comunicaciones es utilizado para simular en forma precisa el comportamiento de la aplicación en el entorno de ejecución. En la Figura 47 se puede apreciar la diferencia entre el enlace más rápido y el más lento, que es de por lo menos medio orden de magnitud para el mismo volumen de datos trasmitido.

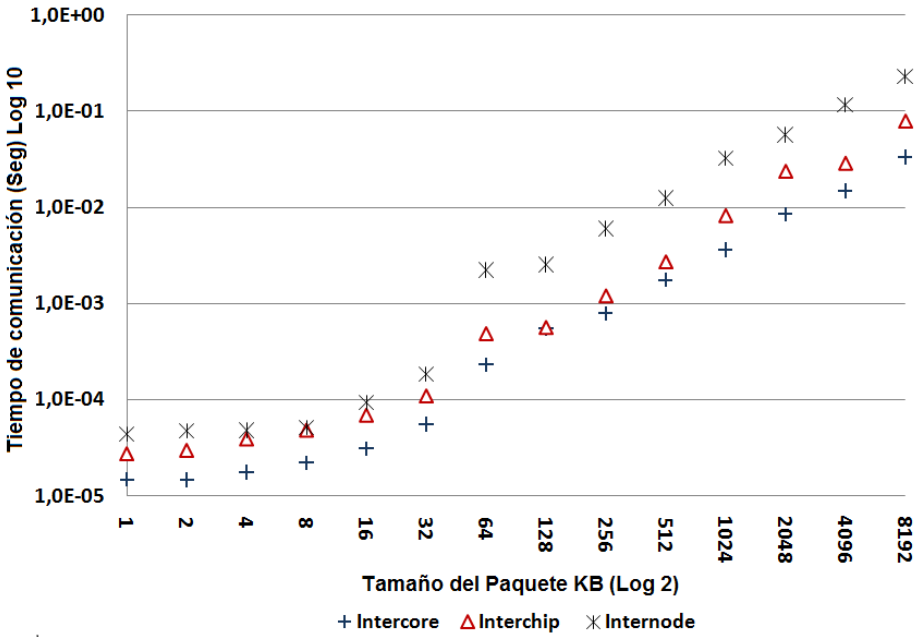


Figura 47 Caracterización de comunicaciones clúster IBM

De forma similar la Figura 48 muestra la caracterización realizada utilizando el patrón de comunicaciones de la aplicación de la trasferencia de calor, en el clúster DELL. Este patrón de comunicación es con 4 vecinos de forma simultánea. En este sistema se puede observar que se utilizan cuatro niveles de jerarquía de comunicación, dos para las comunicaciones entre cores dentro de la pastilla del procesador (*intercore*)[6], uno entre procesadores en el mismo nodo *(interchip)* y el último para la comunicación entre nodos *(internode)*. Las comunicaciones de paso de mensaje en el nivel de la arquitectura interna se diferencian en que un conjunto de cores comparten la memoria cache y otro conjunto de cores no comparte la memoria cache, la comunicación se realiza por medio de la memoria principal del nodo.

Los niveles de comunicaciones son necesarios caracterizarlos porque varían dependiendo de la arquitectura del sistema con procesadores multicore que se esté utilizando para la ejecución.

[6] Estas comunicaciones son divididas en Intercore con o sin memoria cache, debido a que los cores en este sistema puede comunicar compartiendo o no la memoria cache.

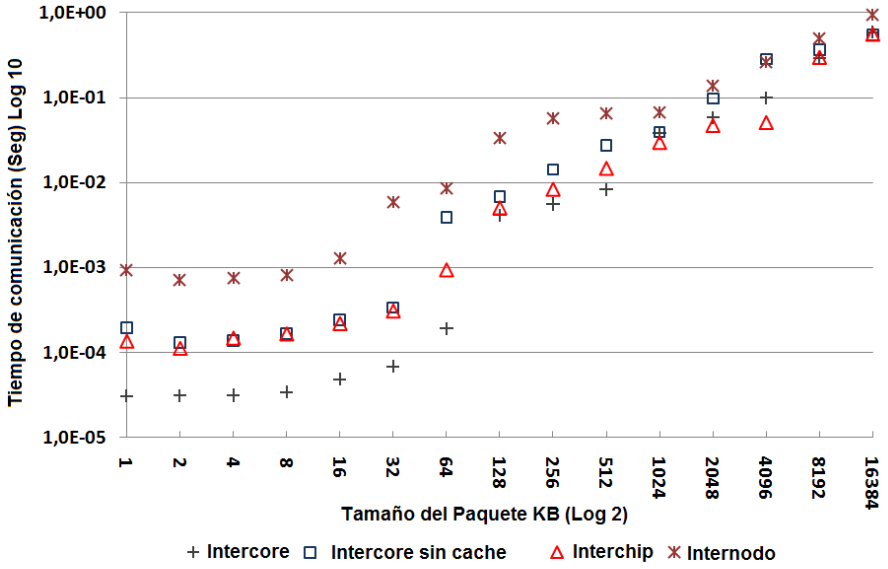


Figura 48 Caracterización de comunicaciones clúster DELL

Por otra parte, al caracterizar las comunicaciones del clúster AMD se logran identificar tres niveles de comunicación (Figura 49). Un nivel que está dentro de la arquitectura y dos que dependen de la distribución que se realice a nivel de la red. Este clúster por tener dos cores por nodo (Tabla 7), origina que las comunicaciones a través de la red (*internodo*) tengan diferencias, al no poder aplicar una estrategia para minimizar el número de comunicaciones que utilizan la red.

Esta caracterización se realizó utilizando el patrón de comunicaciones de la aplicación de Laplace, la cual tiene comunicación con cuatro vecinos. La Figura 49 muestra los diferentes niveles de comunicaciones, donde se puede ver que por la cantidad de cores por nodo, el número de comunicaciones por core a través de la red se incrementa y por ende se tiene que considerar la latencia dependiendo del número de comunicaciones por la red. Asimismo en la Figura 49, se puede apreciar la comunicación a través de la red cuando se utilizan dos y tres comunicaciones.

En el caso donde los cores tienen dos comunicaciones para un tamaño menor a 512KBytes, el tiempo de comunicaciones es considerablemente inferior con respecto a los cores con tres comunicaciones a la red. No

obstante, si el volumen de transmisión es mayor a 512 KBytes ambas comunicaciones tienen un valor similar.

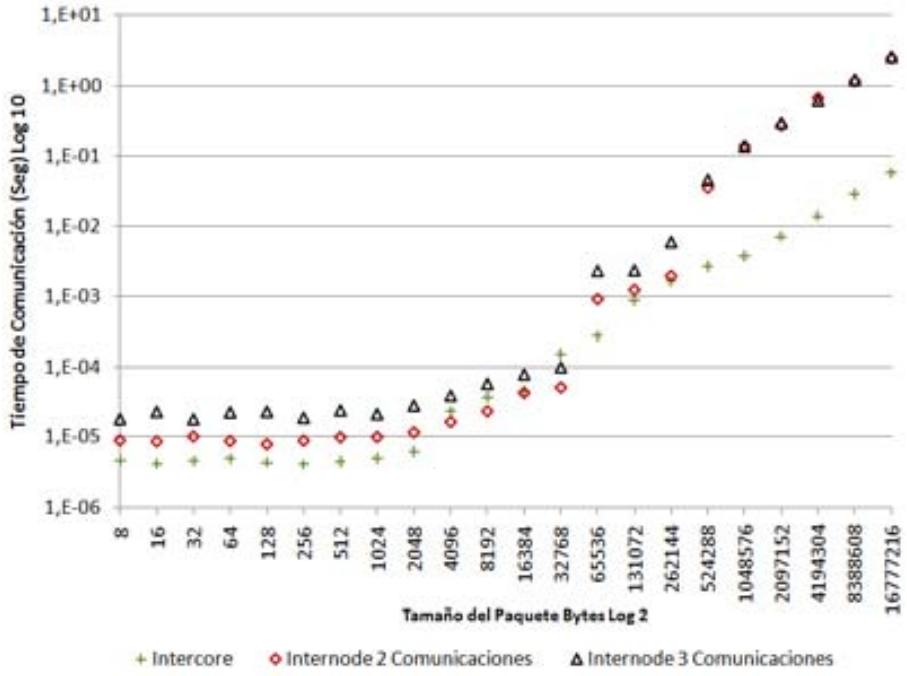


Figura 49 Caracterización de comunicaciones clúster AMD

Otra caracterización necesaria para el cálculo del modelo, está dada por la relación cómputo/comunicación del tile en la arquitectura. Esta caracterización se realiza con el fin de calcular la ratio entre los mismos, para cada aplicación analizada y en cada uno de los tres clústeres multicore utilizados.

El primer análisis es para la aplicación de Transferencia de Calor. La Figura 50 muestra la diferencia entre el cómputo y cada uno de los niveles de comunicación para cada uno de los diferentes clústeres utilizados.

Previamente, para cada clúster se identificaron los diferentes niveles de comunicaciones: cuatro niveles, en otros tres y en el caso del clúster AMD se tiene un nivel adicional que es generado al hacer tres comunicaciones por los cores a través de la red. Esta comunicación adicional es motivada porque no se pueden hacer agrupaciones de comunicaciones a la hora de hacer el mapeo de los procesos MPI en los cores de ejecución como se explicó en el capítulo anterior.

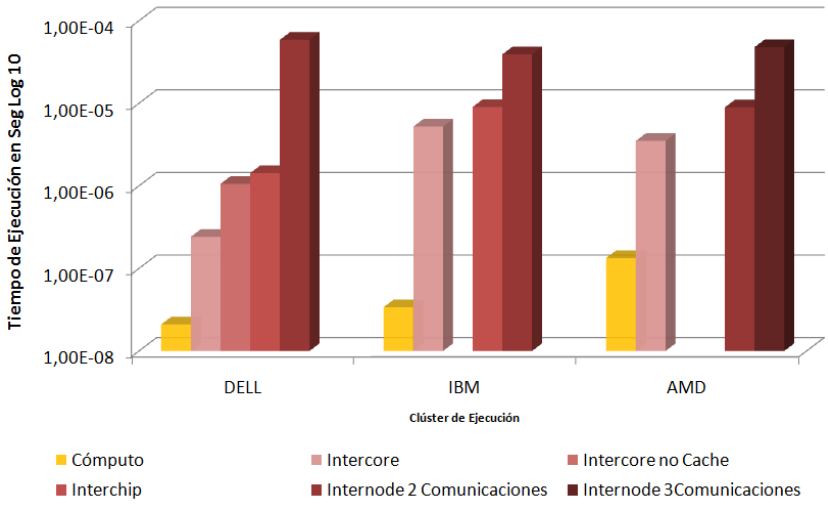


Figura 50 Caracterización de cómputo/comunicación para la aplicación de transferencia de calor

Existen diferencias de hasta un orden y medio de magnitud entre la comunicación más lenta y la más rápida de un clúster con procesadores multicore. El tiempo de cómputo tiene variaciones dependiendo de la arquitectura de ejecución (Figura 50).

El mismo caso se presenta en la Figura 51, donde se analiza el comportamiento de la aplicación de la Ecuación de Laplace. De forma similar existe una gran diferencia entre los tiempos de comunicación por *tile* y el tiempo de cómputo. Esta diferencia permite implementar estrategias para minimizar el *overhead* generado por las comunicaciones y aplicar métodos para hacer la planificación espacial con el fin de mejorar las métricas de rendimiento.

Al ejecutar la aplicación de Laplace en el clúster IBM, la diferencia entre los diversos niveles de comunicación es menor de un orden de magnitud. Este tiempo de comunicación puede variar dependiendo de la cantidad de buses de acceso a la memoria entre los cores y la memoria, así como la velocidad de transmisión de la red o el número de redes existentes para el proceso de comunicación entre los nodos.

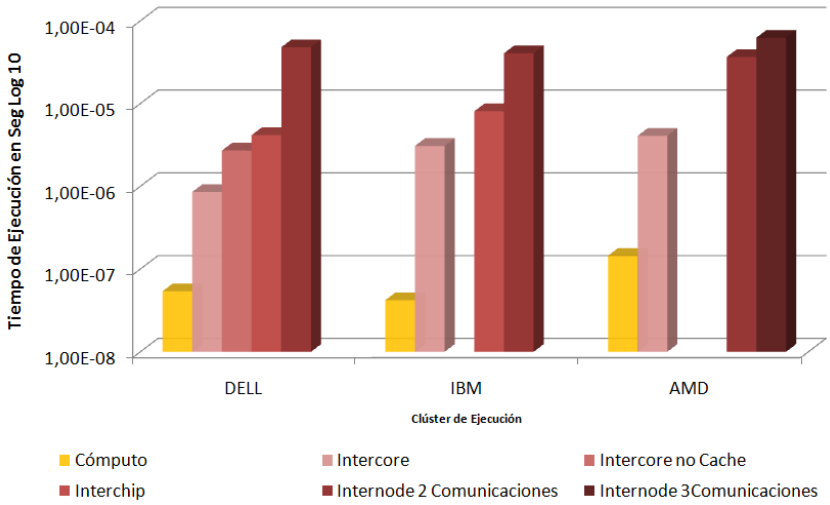


Figura 51 Caracterización de cómputo/comunicación para la aplicación de Laplace

El siguiente análisis es para la aplicación de la Ecuación de la Onda (Figura 52). Esta aplicación tiene un comportamiento similar a las dos aplicaciones anteriores, donde se pueden apreciar los diferentes niveles de comunicación y el tiempo de cómputo de cada tile en cada uno de los clústeres.

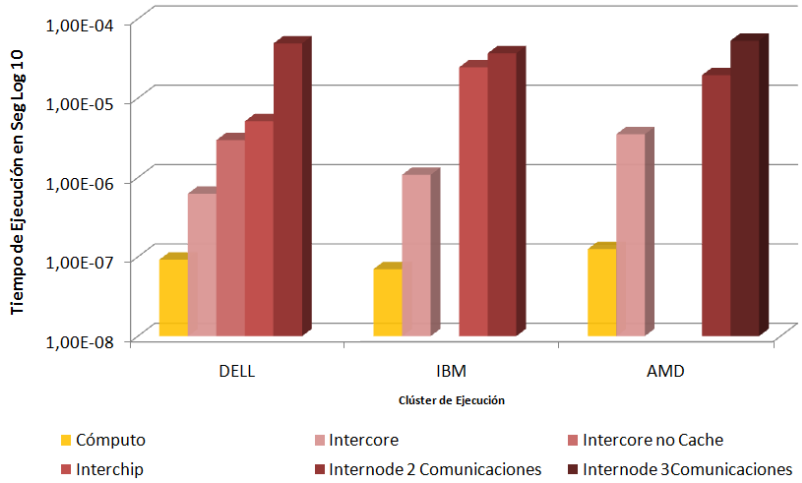


Figura 52 Caracterización de cómputo/comunicación de la aplicación de la Onda

En la Figura 53 se detalla el comportamiento de la aplicación LL-2D-STD-MPI de dinámica de fluidos en el clúster DELL. En esta figura se puede ver las diferencias entre el cómputo del *tile* y las comunicaciones. Estas diferencias nos permiten aplicar las estrategias de agrupaciones que luego

son utilizadas para la estrategia de solapamiento entre el cómputo y las comunicaciones.

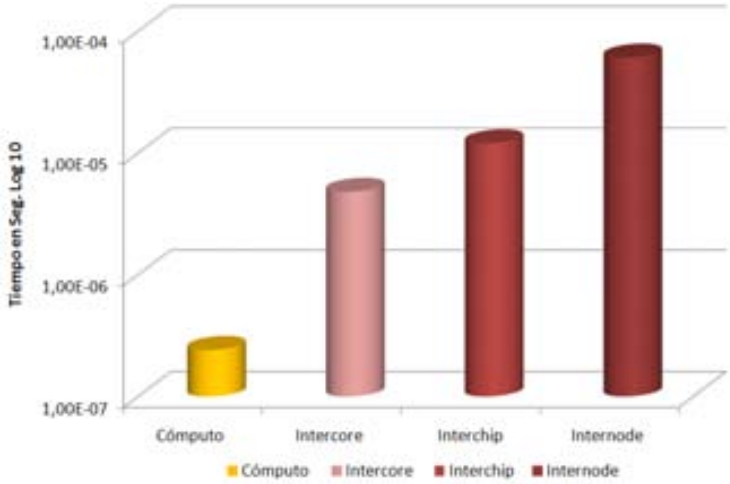


Figura 53 Caracterización de cómputo y comunicación clúster IBM Aplicación LL-2D-STD-MPI

Una vez obtenidos los valores de cómputo de los tiles y los de comunicación entre los *tiles* en cada uno de los enlaces de comunicación, se debe realizar el cálculo del modelo de distribución, que permitirá determinar el tamaño ideal del *supertile* y del número de cores que son necesarios para ejecutar la aplicación con el máximo *speedup* y una eficiencia definida sobre un umbral. Para el cálculo del modelo se utiliza la mayor relación entre el cómputo y la comunicación, debido a que la misma será la que delimitará la iteración de la aplicación SPMD como se explicó en el capítulo anterior.

4.4 Aplicación del modelo de distribución de *tiles*

Al obtener la ratio de cómputo y comunicación, se puede calcular el modelo de distribución de tiles. La Tabla 8 muestra los tiempos de cómputo y los tiempos de comunicación de un tile para el enlace más lento.

Además del tiempo de cómputo y de comunicación, es necesario definir ciertos parámetros para poder calcular el tamaño ideal del supertile, como el tamaño total de tiles con que se desea ejecutar la aplicación (tamaño M^n del problema), el umbral de eficiencia y la ratio λ(p)(w) calculada para cada

enlace utilizando la ecuación E1. Todos estos parámetros de entrada para aplicar el cálculo del modelo se muestran en la Tabla 8.

Tabla 8 Resultados de caracterización y definición de parámetros de ejecución

Aplicación	T. Cómputo $TiempoCpt$ (Seg)	T. Comunicación $TiempoComm(p)$(1 (Seg)	Tamaño del Problema (M)	Eficiencia deseada (Effic)	Ratio Cómputo-Com (λ)	Clúster
Transferencia de Calor	2.10E-8	5.88E-5	9500	85%	2800.0	DELL
Ecuación de Laplace	5.4E-8	4.8E-5	4550	90%	888.8	DELL
Ecuación de la Onda	9.2E-8	4.9E-5	1950	90%	532.6	DELL
Transferencia de Calor	3.40E-8	3.92E-5	7880	85%	1152.9	IBM
Ecuación de Laplace	4.20E-8	4.06E-5	6250	80%	966.6	IBM
Ecuación de la Onda	7.00E-8	3.68E-5	1800	85%	526.0	IBM
LL-2D-STD-MPI	2.41E-7	6.07E-5	1950	95%	251.8	IBM
Transferencia de Calor	1.34E-7	4.79E-5	1080	85%	357.2	AMD
Ecuación de Laplace	1.45E-7	6.27E-5	1100	90%	432.0	AMD
Ecuación de la Onda	1.25E-7	5.31E-5	1370	85%	424.8	AMD

Obtenida la ratio el siguiente paso es calcular el tamaño ideal de cada supertile y el número de cores que mantienen la eficiencia sobre un umbral. Este valor es definido por K y calculado utilizando la ecuación E9. La Tabla 9 muestra los valores obtenidos una vez aplicado el modelo analítico para las aplicaciones que serán ejecutadas en el clúster DELL. Para todos los casos consideramos un umbral de error de +/- 5% del tiempo de ejecución de la aplicación. Estos valores se determinan para la versión que utilizará la estrategia de solapamiento integrada con la metodología.

En la Tabla 9 se pueden apreciar los tiempos de cómputo determinados de forma analítica, tanto para el cómputo interno como para el cómputo de borde (ecuaciones E4 y E5 respectivamente). También, se puede observar el tiempo de comunicación para el enlace más lento (ecuación E6) y el tiempo de ejecución de la aplicación para una iteración (ecuación E2).

Tabla 9 Valores analíticos obtenidos para las aplicaciones ejecutadas en el clúster DELL para una iteración

Aplicación	Nº cores	K	Cómputo de Borde (Seg)	Cómputo Interno (Seg)	Comunicación de Borde (Seg)	Tiempo de Ejecución (Seg)
Transferencia de Calor	16	2383	1.99E-04	1.18E-01	1.40E-01	0.139849
Ecuación de Laplace	32	803	1.73E-04	3.47E-02	3.86E-02	0.038765
Ecuación de la Onda	16	483	1.79E-04	2.17E-02	2.39E-02	0.024091

Los valores obtenidos dan una referencia del valor ideal de cores de ejecución para la aplicación seleccionada en el clúster DELL. Este valor es encontrado utilizando las ecuaciones E10, E12 y E14 dependiendo de la dimensión de la aplicación.

La precisión de estos valores serán analizados en el apartado de evaluación de rendimiento de este capítulo, cuando se realice el análisis de eficiencia y el *speedup* tanto para la versión sin aplicar la metodología, como para la versión que aplica la metodología. Es importante destacar que los tamaños de los problemas a ejecutar han sido adaptados para que el número de cores ideales obtenidos por el modelo, sea un valor dentro de una escala logarítmica base 2.

Tabla 10 Valores analíticos obtenidos para las aplicaciones ejecutadas en el clúster IBM para una iteración

Aplicación	Nº cores	K	Cómputo de Borde (Seg)	Cómputo Interno (Seg)	Comunicación de Borde (Seg)	Tiempo de Ejecución (Seg)
Transferencia de Calor	64	983	1.34E-04	3.29E-02	3.86E-02	0.038746
Ecuación de Laplace	64	777	1.31E-04	2.55E-02	3.17E-02	0.03184
Ecuación de la Onda	16	451	1.26E-04	1.40E-02	1.66E-02	0.016696
LL-2D-STD-MPI	64	243	2.34E-04	1.41E-02	1.48E-02	0.015045

Por otra parte, la Tabla 10 muestra los valores obtenidos para los mismos parámetros de caracterización establecidos en la Tabla 8. En la Tabla 10, se

muestran los tiempos de ejecución teóricos y el tamaño ideal de cores y *supertile* al ejecutar la aplicación en el clúster IBM.

Finalmente la Tabla 11 muestra los resultados obtenidos por el modelo para todas las aplicaciones evaluadas en el clúster AMD. De forma similar el modelo permite obtener un perfil de la ejecución de la aplicación para una iteración. No obstante, al conocer el número de iteraciones se puede predecir el tiempo de ejecución global de la aplicación para el entorno.

Tabla 11 Valores analíticos obtenidos para las aplicaciones ejecutadas en el clúster AMD para una iteración

Aplicación	Nº cores	K	Cómputo de Borde (Seg)	Cómputo Interno (Seg)	Comunicación de Borde (Seg)	Tiempo de Ejecución (Seg)
Transferencia de Calor	12	307	1.67E-04	1.29E-02	1.49E-02	0.015102
Ecuación de Laplace	8	393	2.25E-04	2.17E-02	2.44E-02	0.024596
Ecuación de la Onda	14	365	1.83E-04	1.66E-02	1.94E-02	0.019617

En el caso de utilizar aplicaciones SPMD donde la metodología se puede implementar sólo a una porción de las mismas, los tiempos de ejecución se deben medir solamente para esa porción donde se ha aplicado el método. Además, los tiempos de ejecución de las aplicaciones evaluadas es considerando que el sistema de entrada/salida de la aplicación se ha deshabilitado. Por lo que los tiempos de ejecución reflejan solamente la fase de intercambio de comunicaciones sin tener en cuenta la fase de repartición inicial, así como el proceso de entrada-salida y el proceso de recolección de la aplicación.

4.5 Evaluación de rendimiento

Para la evaluación de la metodología se han considerado cuatro ejemplos de los definidos en la Tabla 8. Estos ejemplos se han seleccionado considerando cada aplicación y su ejecución en cada uno de los clúster definidos.

El primer caso mostrado en la Figura 54 representa al comportamiento de la aplicación de Transferencia de Calor en el clúster DELL. Este experimento

muestra el comportamiento de la eficiencia y del speedup, tanto para la versión sin aplicar la metodología como para la versión una vez aplicada la metodología. Como se pudo detallar en la Tabla 9, el valor ideal de cores para este tamaño de problema (9500x9500) y clúster es de 16 cores.

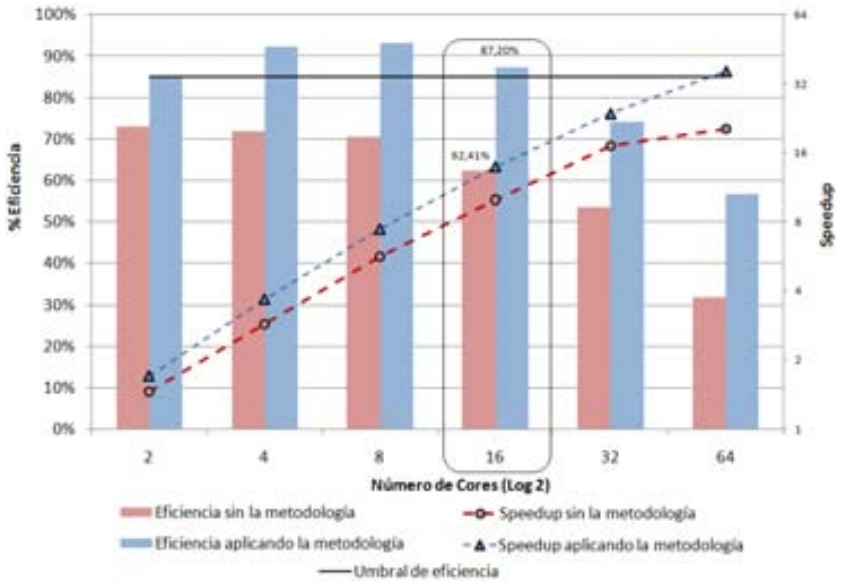


Figura 54 Evaluación de eficiencia y *speedup* aplicación de transferencia de calor en clúster DELL (Tamaño 9500x9500).

Este valor es obtenido a través de la aplicación del modelo, por lo cual al visualizar el comportamiento de ambas versiones se puede constatar la mejora al aplicarse la metodología. Asimismo, se puede visualizar que la eficiencia para ese número de cores está muy cercana al umbral definido, con sólo un 3% de error sobre el tiempo de ejecución. Esta aplicación se ha ejecutado considerando un número de 100 iteraciones y cada resultados es el promedio de 50 ejecuciones por cada número de cores definidos en el eje X.

El comportamiento creciente de la aplicación con relación al máximo *speedup* sobre una eficiencia definida se muestra en la Tabla 12 tanto aplicando la metodología como sin la metodología. No obstante, cuando se realiza la comparación con la eficiencia se puede visualizar que la versión sin aplicar la metodología alcanza una eficiencia alrededor del 62% para el punto ideal calculado a través del modelo, mientras que la versión que aplica la metodología, donde la eficiencia alcanza un valor de 87,20%.

Estos valores, confirman que al combinar ambas métricas se puede conseguir una ejecución rápida y eficiente. Además la versión aplicando la metodología tiene una ganancia en este punto de aproximadamente 39% de eficiencia. Este valor es calculado dividiendo la eficiencia de la versión aplicando la metodología entre la eficiencia de la versión sin aplicar la metodología.

Tabla 12 Combinación de eficiencia y *speedup* en la aplicación de transferencia de calor en clúster DELL (Tamaño 9500x9500).

N cores	*Speedup* sin aplicar la metodología	*Speedup* aplicando la metodología	Eficiencia sin aplicar la metodología	Eficiencia aplicando la metodología
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%	93.26%
16	**9.99**	**13.95**	**62.41%**	**87.20%**
32	17.16	23.71	53.62%	74.10%

Por otra parte, se puede observar que al aumentar el número de cores, la eficiencia comienza a decrecer considerablemente. Esto se debe a que la aplicación tiene un comportamiento limitado por las comunicaciones que es lo que trata de gestionar la metodología.

Otro ejemplo mostrado es para la aplicación LL-2D-STD-MPI (Figura 55). Se ha ejecutado con tamaños diferentes de tiles, para un número de 1000 iteraciones utilizando el clúster IBM. De forma similar al caso anterior, se puede detallar que el valor obtenido por el modelo, se acerca considerablemente al umbral definido de eficiencia. El comportamiento para el conjunto de cores inferior al valor ideal calculado por el modelo (Tabla 9) para ambas versiones es similar, esto es motivado a que la aplicación está dividida en tres fases, dos de ellas de cómputo intensivo y la tercera de intercambio por lo cual la metodología se ha aplicado solamente a esta última fase.

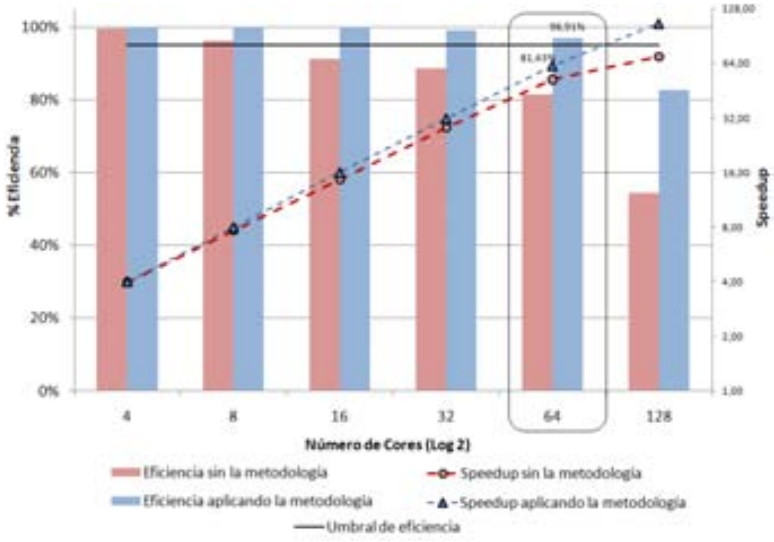


Figura 55 Evaluación de eficiencia y *speedup* aplicación de LL-2D-STD-MPI en clúster IBM (Tamaño 1950x1950).

Además se puede apreciar que la eficiencia ronda el umbral con un error aproximado del 2%. En este caso se puede ver en la Tabla 13 que la eficiencia para la versión con la metodología comienza a decrecer en el punto ideal de cores. A partir de ese punto la aplicación tiene un comportamiento limitado por las comunicaciones y por ende la aplicación comienza a perder eficiencia.

Tabla 13 Combinación de eficiencia y *speedup* aplicación de LL-2D-STD-MPI en clúster IBM (Tamaño 1950x1950)

N cores	*Speedup* sin aplicar la metodología	*Speedup* aplicando la metodología	Eficiencia sin aplicar la metodología	Eficiencia aplicando la metodología
4	3.98	4.00	99.62%	99.98%
8	7.70	8.00	96.26%	99.96%
16	14.62	15.99	91.35%	99.94%
32	28.33	31.71	88.52%	99.09%
64	**52.12**	**62.02**	**81.43%**	**96.91%**
128	69.56	105.88	54.35%	82.72%

En relación a la combinación del *speedup* y la eficiencia definida, se muestra en la Tabla 13, el comportamiento del *speedup*, el cual para la versión que aplica la metodología tiene un comportamiento similar con respecto al *speedup* teórico y este crecimiento comienza a disminuir a partir del punto ideal obtenido por el modelo. Caso contrario ocurre con la versión

sin la metodología, debido a que las comunicaciones no son gestionadas y por ende el efecto de las comunicaciones comienza a tener gran influencia a partir de los 16 cores. En ambos casos el *speedup* tiene un crecimiento por lo que el máximo valor se podría definir en un conjunto de cores mayor. Sin embargo, la metodología evalúa la restricción de la eficiencia, lo que conlleva a que ese número sea menor como se ha presentado en los datos obtenidos (Tabla 13).

De forma similar a la aplicación anteriormente estudiada, la aplicación de la Ecuación de la Onda en el clúster IBM tiene un comportamiento semejante hasta el valor de cores calculado por el método. En este caso, el valor ideal es de 16 cores para el tamaño de problema definido de 1800x1800. En este punto la ganancia de una versión con respecto a la otra en eficiencia alcanza un valor de 13%.

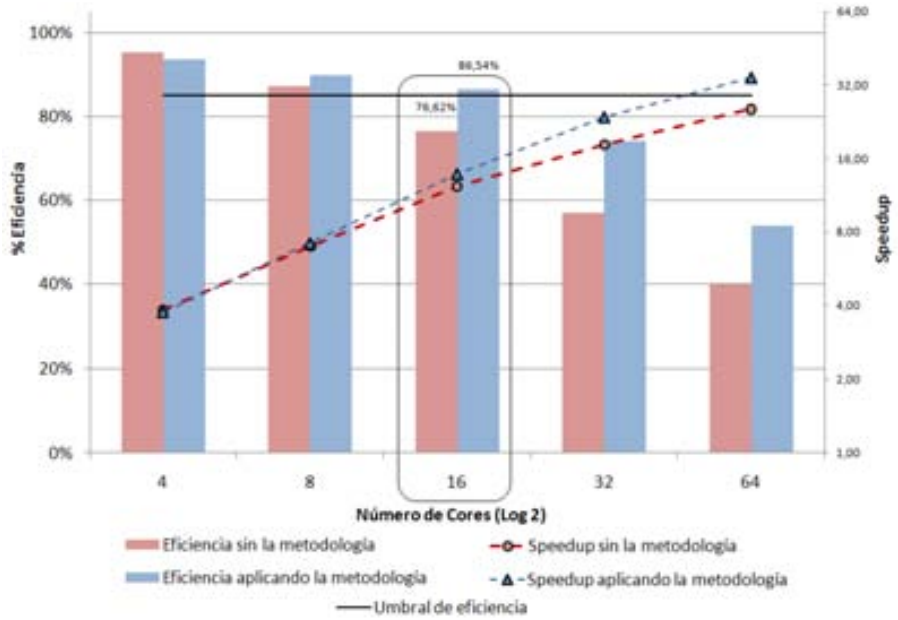


Figura 56 Evaluación de eficiencia y *speedup* aplicación de la Onda en clúster IBM (Tamaño 1800x1800).

Al comprobar la obtención del máximo valor de speedup con la eficiencia definida, se puede ver en la Tabla 14 que esta aplicación tiene un comportamiento similar a los casos anteriormente explicados, donde el valor de cores obtenido por el método alcanza un speedup máximo con la restricción de la eficiencia al 85% (Tabla 8). En este caso el error obtenido

por la aplicación con la metodología es de aproximadamente 1.5% sobre el umbral de eficiencia definido.

Tabla 14 Combinación de eficiencia y *speedup* aplicación de Onda en clúster IBM (Tamaño 1800x1800).

Ncores	*Speedup* sin aplicar la metodología	*Speedup* aplicando la metodología	Eficiencia sin aplicar la metodología	Eficiencia aplicando la metodología
4	3.81	3.75	95.29%	93.68%
8	6.98	7.19	87.23%	89.83%
16	**12.26**	**13.85**	**76.62%**	**86.54%**
32	18.22	23.72	56.93%	74.14%
64	25.48	34.47	39.82%	53.86%

El último caso utilizado para evaluar la metodología es presentado en la Figura 57, donde se evalúa el comportamiento tanto del *speedup* como de la eficiencia para la aplicación de la Ecuación de Laplace en el clúster AMD. En la gráfica se puede apreciar que la eficiencia para la versión que aplica la metodología está por debajo del umbral definido para el número de cores determinados a través del modelo (8 cores). Sin embargo, este valor tiene un error menor del 4.8% para un tamaño de problema de 1100x1100.

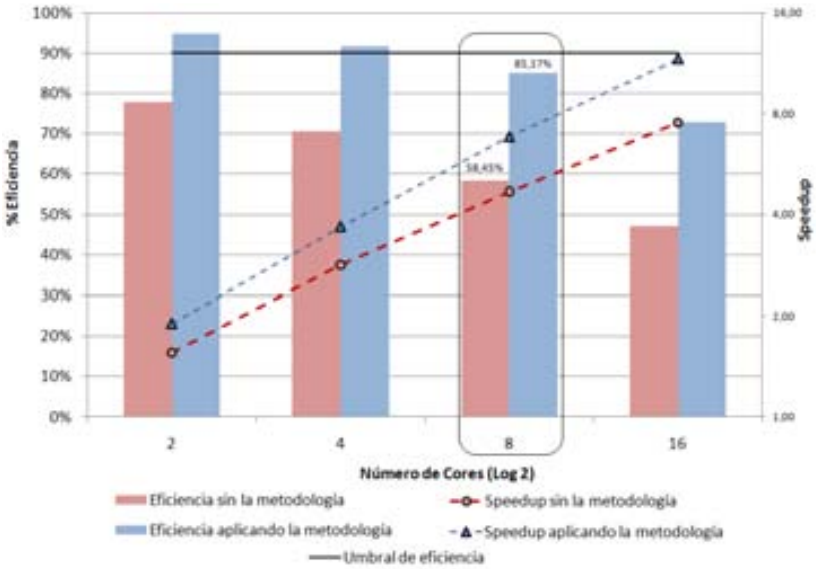


Figura 57 Evaluación de eficiencia y *speedup* aplicación de Laplace en clúster AMD (Tamaño 1100x1100).

En este caso en particular se puede detallar que usando cuatro cores la eficiencia es más cercana al valor obtenido con la metodología. No obstante,

si se evalúa el valor de *speedup* se consigue que para ocho cores el *speedup* es considerablemente mayor.

Lo anterior, puede ser constado en la Tabla 15, donde el *speedup* tiene un incremento considerable con respecto a la versión sin aplicar la metodología. En este caso en particular, la eficiencia obtenida con la metodología presenta un error aproximado del 5% con respecto al umbral definido.

Tabla 15 Combinación de eficiencia y *speedup* aplicación de Laplace en clúster AMD (Tamaño 1100x1100).

Ncores	*Speedup* sin aplicar la metodología	*Speedup* aplicando la metodología	Eficiencia sin aplicar la metodología	Eficiencia aplicando la metodología
2	1.55	1.90	77.75%	94.75%
4	2.82	3.67	70.60%	91.76%
8	**4.68**	**6.81**	**58.45%**	**85.17%**
16	7.52	11.66	47.02%	72.87%

De forma similar a los casos anteriores, el valor obtenido como ideal cumple las condiciones de máximo *speedup* con la eficiencia definida sobre el umbral.

4.6 Conclusiones

Como se ha logrado constatar, la metodología consigue obtener un valor ideal del *supertile,* que luego es utilizado para determinar el número de cores necesarios para la ejecución de la aplicación SPMD en el clúster multicore que se está evaluando. La precisión del modelo es debido a dos aspectos claves, el primero está relacionado con el comportamiento determinístico de las aplicaciones SPMD y el segundo a los valores obtenidos de la caracterización de la aplicación en el entorno de ejecución.

Ambos elementos permiten que se puedan aplicar las estrategias para minimizar los efectos de las comunicaciones y por ende mejorar las prestaciones de la aplicación en una arquitectura multicore específica. Esto se pudo comprobar en los experimentos realizados, donde en el mejor de los casos se ha logrado una mejora de hasta 39% para la versión de la aplicación con la metodología con respecto a la versión original. Además, el

margen de error obtenido para todas las pruebas testeadas es menor del 5%, lo que hace que la metodología sea usable en las aplicaciones SPMD que siguen características definidas al comienzo de éste capítulo.

Capítulo 5

Evaluación de escalabilidad y eficiencia de las aplicaciones SPMD en clúster con procesadores multicore

Resumen

Un problema que tienen las aplicaciones SPMD de paso de mensajes a la hora de utilizar entornos con comunicaciones heterogéneas está relacionado con la escalabilidad. Este problema incrementa cuando se define un tamaño de problema fijo y se aumenta el número de cores a ejecutar (Escalabilidad fuerte). En este capítulo se explica cómo determinar el máximo punto de escalabilidad fuerte bajo una eficiencia definida. Además, se evalúa la escalabilidad débil que consiste en el incremento del tamaño del problema y del número de cores. Para esto se evalúa el tamaño del supertile y se estudia el comportamiento de la aplicación SPMD mientras el tamaño del supertile se mantiene aunque se incremente el tamaño del problema. Este incremento se realiza basándose en la relación de cómputo y comunicación que se definió en el capítulo anterior. Finalmente, este capítulo muestra algunos resultados para ambas escalabilidades y cómo a través de la metodología de ejecución eficiente se puede determinar si el tamaño del problema se adapta a la cantidad de recursos disponibles o no.

5.1 Introducción

La eficiencia y la escalabilidad de las aplicaciones paralelas son conceptos claves a la hora de analizar las prestaciones de sistemas paralelos [47]. Actualmente, estos sistemas incluyen elementos que generan cierto nivel de heterogeneidad, tanto a nivel de cómputo como a nivel de comunicaciones, ocasionando grandes desafíos cuando se desea combinar ambas métricas paralelas. En tal sentido, en éste capítulo se muestra que aplicando las estrategias definidas en la metodología, se pueden combinar las métricas de eficiencia y escalabilidad para determinar el número ideal de cores necesarios para ejecutar en el entorno multicore.

El término escalabilidad en HPC se puede dividir en dos definiciones, escalabilidad fuerte, donde el tamaño del problema se fija mientras el número de elementos de procesamiento se incrementa, con el objetivo de minimizar el tiempo de ejecución de la aplicación[26][39][47]; y la escalabilidad débil, considerada cuando el tamaño del problema y el número de cores se incrementan. El principal objetivo de ésta escalabilidad es alcanzar un tiempo de ejecución constante para grandes problemas, así como conservar la misma carga de trabajo por procesador para mantenerla contaste [26]. La escalabilidad débil tiene la habilidad de mantener un tiempo de ejecución fijo para problemas grandes en sistemas con gran número de cores, esto significa un *speedup* aproximadamente proporcional al número de cores [2] [39].

Aplicar estos conceptos de escalabilidad a las aplicaciones SPMD, implica hacer crecer el número de *tiles* en la aplicación o mantener el tamaño de *tiles* fijo. En ambos casos para obtener una escalabilidad lineal, se debe mantener la ratio cómputo comunicación de la aplicación considerando que el entorno de ejecución tiene diversos canales de comunicaciones. Por esta razón, el modelo analítico definido en la metodología se puede utilizar para calcular el máximo número de cores que mantienen la eficiencia sobre un umbral para un tamaño de *tiles* fijos (Escalabilidad fuerte) y además, analizar cómo debe crecer la aplicación con respecto al número de cores. Esto se realiza manteniendo el mismo tamaño del *supertile* calculado con el modelo

analítico y asignado a un número de cores específico. Si se mantiene el tamaño del *supertile* se puede lograr un crecimiento lineal cuando se busca el límite de la escalabilidad débil.

Bajo las consideraciones de escalabilidad, en este capítulo se detalla cómo determinar el máximo número de cores que mantienen la máxima escalabilidad fuerte bajo una eficiencia definida. Es decir, se demuestra cómo el entorno multicore utilizado para ejecutar la aplicación, tiene la capacidad para mantener un *speedup* lineal mientras el tamaño del problema se fija a un valor. En este sentido, se puede encontrar el máximo punto donde se consigue el máximo *speedup* bajo una eficiencia definida usando el modelo analítico de la metodología.

Por otra parte, este capítulo describe cómo manteniendo el tamaño del *supertile* en la aplicación, se puede utilizar el modelo para adaptar la aplicación al entorno de ejecución y así obtener una escalabilidad lineal respecto al número de cores utilizado. Para obtener este resultado, se debe calcular el tamaño ideal del *supertile* que mantenga una estrecha relación entre el cómputo interno y las comunicaciones de los bordes.

Seguidamente se explican las limitaciones que tienen los entornos de comunicaciones heterogéneas con relación a la escalabilidad. Para esto se evalúan ambas escalabilidades y se describe con un ejemplo teórico la aplicación de la metodología para combinar ambos términos eficiencia y escalabilidad. Al final, se realiza una evaluación de rendimiento para validar la combinación de ambas métricas paralelas.

5.2 Limitaciones de escalabilidad y aplicaciones SPMD

Para gestionar las ineficiencias de las comunicaciones a través de la metodología, se utiliza una estrategia de *mapping* y *scheduling* que se fundamenta en la minimización de los efectos de comunicaciones. Como hemos visto, la metodología crea un *supertile* que mantiene una relación entre el tiempo que se tarda en realizar el cómputo interno y el tiempo que requieren las comunicaciones de los tiles de borde. Un ejemplo de cómo se

puede aplicar la metodología para combinar la escalabilidad y la eficiencia es mostrado en la Figura 58.

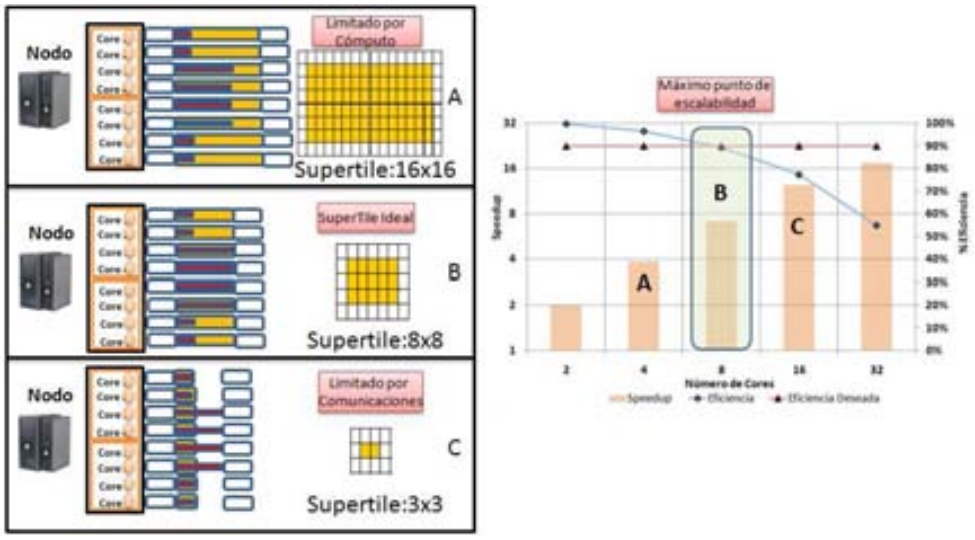


Figura 58 Equilibrio escalabilidad-eficiencia deseada.

La figura ilustra tres tamaños diferentes de *supertile*[7], cada uno presenta un comportamiento distinto, el caso identificado con la letra “A” representa un *supertile* cuyo comportamiento está limitado por el cómputo. La letra “B” representa el caso del solapamiento ideal del tiempo de cómputo y tiempo de comunicaciones y finalmente el identificado con la letra “C” es el caso donde la iteración está limitada por el tiempo que se tarda en realizar las comunicaciones.

Como se evidencia en el último caso las comunicaciones son las que limitan el tiempo de ejecución de la iteración, por lo que si se agrega mayor cantidad de cores para ejecutar el mismo tamaño de problema, el número de comunicaciones a través del enlace más lento seguirá creciendo en proporción al tiempo de ejecución, y el tamaño del *supertile* por core disminuirá, lo que origina que la eficiencia de la aplicación se vea afectada considerablemente y que el crecimiento del *speedup* no sea lineal con respecto al número de cores (Figura 58). Aunque en este punto el *speedup* sigue creciendo la eficiencia comienza a decrecer considerablemente con respecto a las condiciones deseadas.

[7] Los tamaños definidos en la figura para el supertile son de carácter ilustrativo.

En el caso del *supertile* identificado con la letra “A”, el tiempo de cómputo interno es mayor que el tiempo de comunicación de los bordes por lo que al estar solapadas estas comunicaciones no afectan la métrica de eficiencia. En este punto, el *speedup* es lineal y la eficiencia está al máximo valor. Sin embargo, lo ideal es encontrar de forma analítica el máximo número de cores que mantiene la estrecha relación entre el máximo *speedup* con una eficiencia definida, el cual permitirá obtener un escenario como el presentado en el caso del *supertile* identificado con “B” (Figura 58). A partir de ese punto las comunicaciones tienen mayor impacto sobre la ejecución, debido a que al aumentar el número de cores, se disminuye el tamaño del *supertile*, y por ende las comunicaciones de borde de los tiles no estarán solapadas con el cómputo de los tiles internos, por lo que empiezan a afectar a la eficiencia y al *speedup.* Esto conlleva a que la escalabilidad de la aplicación no tenga un crecimiento proporcional al número de cores.

Este análisis es necesario realizarlo principalmente cuando se está evaluando el término de la escalabilidad fuerte; es decir, tenemos un tamaño del problema ***M*** constante y se quiere aumentar el número de cores. Para analizar el tamaño de ***K*** para un número de cores específicos*,* se despeja el valor de las ecuaciones E10, E12 y E14, dependiendo del caos y la dimensión del problema. Sin embargo, si se mantiene una relación del tamaño del *supertile* con respecto al incremento del número de cores a ejecutar, entonces se puede lograr obtener una evaluación de la escalabilidad débil de la aplicación y detallar cómo a medida que se incrementa el tamaño del problema y el número de cores, la relación de eficiencia y escalabilidad se mantiene, esto se logra despejando el valor de ***M*** en las ecuaciones E10, E12 y E14 dependiendo de la dimensión del problema.

5.3 Aplicabilidad de la metodología para el estudio de la escalabilidad fuerte

La aplicabilidad del modelo analítico con relación a la combinación de la eficiencia y la escalabilidad fuerte se explicará a través de un ejemplo teórico, donde se muestra que a través de la metodología de ejecución

eficiente, se puede obtener analíticamente el máximo punto de escalabilidad fuerte bajo las condiciones de eficiencia definidas. En este sentido, se tiene una aplicación SPMD compuesta por las siguientes características: un tamaño de problema M, con 1585 *tiles* cuadrados, una eficiencia definida del 95%, una aplicación bidimensional, un patrón de comunicaciones a cuatro vecinos y un entorno con procesadores multicore que integra tres diferentes niveles de comunicaciones (*intercore, interchip e internodo*).

El primer paso es realizar la evaluación de la arquitectura y de las características de la aplicación en el entorno de ejecución. Al realizar la caracterización se obtiene la información necesaria para calcular la ratio de cómputo y comunicación **λ(p)(w)** de cada enlace de comunicación utilizando la ecuación E1. Una vez determinada la ratio cómputo-comunicación de los *tiles*, se determina la mayor ratio, la cual se utiliza para calcular el tamaño del *supertile* y el comportamiento con diferente número de cores necesarios para mantener la estrategia de solapamiento entre el cómputo interno y la comunicación de borde.

La Tabla 16 muestra un ejemplo de caracterización, donde se han establecido valores de carácter ilustrativo, expresados en unidades de tiempo para entender la aplicabilidad de la metodología. Se puede observar que el tiempo de comunicación por el enlace *intercore* es 10 veces mayor que el tiempo de cómputo del *tile* (Tabla 16).

Tabla 16 Valores de la ratio cómputo-comunicación para calcular el ejemplo de escalabilidad fuerte

Enlace de Comunicación	Tiempo de Cómputo (*tile*)	Tiempo de Comunicación (*tile*)	*λ(p)(w)*
Intercore	1	10	10
Interchip	1	50	50
Internodo	1	100	100

Una vez encontradas las ratios de cada enlace, se selecciona la mayor ratio de cómputo-comunicación, la cual determina la comunicación más lenta dentro del entorno de ejecución. En este ejemplo particular la mayor ratio es la del enlace *internodo* con un valor de ***λ(p)(w)*** = 100 (Ver Tabla 16).

Al obtener el valor máximo de la ratio, se aplica el modelo analítico de la fase de distribución de *tiles* explicado en el capítulo anterior. El objetivo al aplicar el modelo es encontrar el valor ideal de cores y el tamaño del *supertile* que permite alcanzar el máximo *speedup* con una eficiencia definida. Al ser una aplicación bidimensional, se debe utilizar para el cálculo las ecuaciones E12 y E13 respectivamente. La ecuación E13 determina el tamaño ideal de ***K*** para calcular el *supertile* y la ecuación E12 calcula el número ideal de cores para la ejecución con las características definidas.

Este valor de cores representa el máximo punto de la escalabilidad fuerte con una eficiencia definida. Para este ejemplo, el valor ideal del K es aproximadamente 99 *tiles* por lo que el tamaño del *supertile* es calculado con la ecuación E3. Con este valor del *supertile*, el número de cores es de 256. Al conseguir ambos valores, se debe calcular el *speedup* y la eficiencia teórica: Para realizar estos cálculos se debe determinar el tiempo de ejecución serie de la aplicación. Este valor se puede estimar usando el tamaño del problema elevado a la dimensión (conjunto de *tiles* que integra la aplicación) multiplicado por el tiempo de cómputo del *tile* (Ecuación E16), donde ***n*** representa la dimensión del problema, ***M*** el tamaño del problema y $\boldsymbol{TiempoCpt}$ es el tiempo en computar un *tile* en la arquitectura. El tiempo serie para este ejemplo es de 2.509.056 unidades de tiempo para una iteración.

$$Tiempo\ Serie = M^n * TiempoCpt \quad \text{(E16)}$$

Una vez encontrado el tiempo serie, el siguiente paso es obtener los valores de *speedup* y eficiencia para el número de cores y el tamaño ideal del *supertile*. Para lograr esto, se utilizan las ecuaciones definidas en [63] e identificadas como las ecuaciones E17 y E18 respectivamente.

$$Sp = \frac{Tiempo\ Serie}{Tiempo\ Paralelo} \quad \text{(E17)}$$

$$Eficiencia\ = \frac{Tiempo\ Serie}{(Tiempo\ Paralelo * Ncores)} \quad \text{(E18)}$$

Es importante recordar que la metodología se puede aplicar, o bien a toda la aplicación, o sólo a las fases con un patrón regular de intercambio de información de la aplicación. En este sentido se debe determinar el tiempo de ejecución paralelo aplicando la ecuación E2 del modelo y utilizando los valores obtenidos para K y N cores.

Los resultados pueden ser detallados en la Tabla 17, donde se visualizan los valores ideales obtenidos a través del modelo, para cada distribución de cores. Asimismo, se puede visualizar el comportamiento del valor ideal, en este punto la eficiencia cumple con el requisito de estar cerca del umbral especificado y el speedup está en su valor máximo bajo la eficiencia definida.

Tabla 17 Análisis de eficiencia, *speedup* y escalabilidad para el ejemplo teórico de una aplicación SPMD.

Cores	K	Tiempo de Cómputo de Borde	Tiempo de Cómputo Interno	Tiempo de Comunicación. de Borde	Tiempo de Ejecución (Iter)	Sp	Effic
2	1120	4476	1249924	112000	1254400	2	100%
4	792	3164	624100	79200	627264	4	100%
8	560	2236	311364	56000	313600	8	100%
16	396	1580	155236	39600	156816	16	100%
32	280	1116	77284	28000	78400	32	100%
64	198	788	38416	19800	39204	64	100%
128	140	556	19044	14000	19600	128	100%
256	**99**	**392**	**9409**	**9900**	**10292**	**244**	**95%**
512	70	276	4624	7000	7276	345	67%
1024	50	196	2304	5000	5196	483	47%

Al conocer el valor del máximo del número de cores que mantiene la combinación entre la escalabilidad y la eficiencia, se procede a evaluar el comportamiento teórico de la aplicación calculando el valor de K que debe ser asignado con el mismo tamaño del problema, pero realizando variaciones en el número de cores. Esta variación para este ejemplo se realiza en forma logarítmica base 2 con el objetivo de visualizar el comportamiento de las curvas de la eficiencia y del *speedup*.

En este sentido, la Tabla 17 muestra los valores obtenidos para un conjunto de cores específicos, variando el número de cores desde 2 hasta

1024. Para cada número de cores se puede calcular el tiempo de cómputo interno, cómputo de borde y las comunicaciones. Con estos valores se puede determinar el tiempo de ejecución paralelo teórico para luego estimar el *speedup* y la eficiencia para cada caso.

Entonces, para determinar el valor de K para un número de cores específico se hace un despeje de la ecuación E19, quedando como resultado la ecuación E20, la cual permite obtener un valor de K para un número específico de cores.

$$Ncores = \frac{M^n}{K^n} \tag{E19}$$

$$K = \sqrt[n]{\frac{M^n}{Ncores}} \tag{E20}$$

Como puede ser visualizado en la Figura 59, el valor ideal del número de cores que permite alcanzar la eficiencia definida, y el *speedup* hasta este conjunto de cores muestra un crecimiento aproximadamente lineal, motivado a que el tamaño del *supertile* mantiene un comportamiento limitado por el cómputo, lo cual hace que la eficiencia de la aplicación no se vea comprometida por las comunicaciones.

Luego del punto ideal de cores, se observa que el *speedup* continúa incrementándose aunque no proporcional al número de cores que ejecutan la aplicación. Además, se puede detallar que la eficiencia a partir de ese punto ideal comienza a decrecer considerablemente motivado al comportamiento, que queda limitado por comunicaciones de la aplicación. Esto significa que el tiempo de comunicaciones de borde es mayor que el tiempo de cómputo interno (Tabla 17).

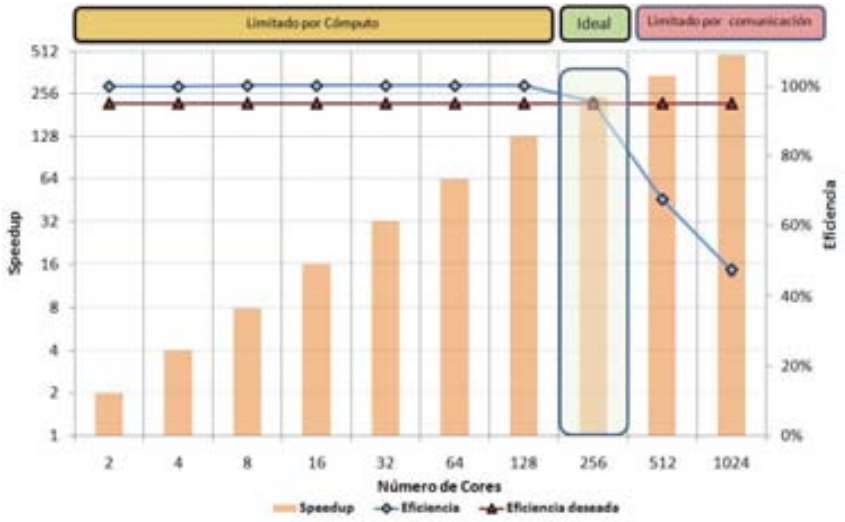


Figura 59 Evaluación de rendimiento de ejemplo teórico.

Por lo tanto, es importante encontrar el valor ideal del *supertile* que permita cumplir las condiciones especificadas en este trabajo de investigación, las cuales están orientadas a mejorar y gestionar las comunicaciones de los entornos multicore, con el objetivo de mejorar las prestaciones de la aplicación. Este ejemplo ilustrativo ha permitido visualizar cómo puede ser aplicada la metodología para encontrar el valor ideal de cores para un tamaño de problema fijo y cómo se comportaría la aplicación con respecto a la escalabilidad fuerte.

5.4 Aplicabilidad de la metodología para el estudio de escalabilidad débil

El objetivo de aplicar la metodología con respecto al análisis del comportamiento considerando la escalabilidad débil, es determinar el comportamiento de la aplicación cuando se incrementa el tamaño del problema y el número de cores, manteniendo la proporción del *supertile* calculado. Para la explicación de esta evaluación, se utilizarán los mismos valores definidos para el estudio expuesto anteriormente relacionado con la escalabilidad fuerte.

Al aplicar las estrategias de planificación espacial y temporal a la aplicación y mantener la relación de similitud entre el tiempo de cómputo interno y las comunicaciones de bordes del *supertile*, se puede mantener la

escalabilidad de la aplicación SPMD. Esto significa que si el problema se incrementa, buscando la misma proporción para escalar el tamaño del *supertile* se puede lograr una escalabilidad lineal ideal de la aplicación. Lo anterior es mostrado con los valores de la Tabla 18, donde el tamaño del problema (***M)*** es incrementado proporcionalmente al tamaño del *supertile* ***(K)*** y al número de cores que se desea incrementar (en el caso de ***M*** y ***K*** serán elevados a la ***n***, donde ***n*** representa la dimensión del problema). En este sentido al mantener la misma relación del tamaño del *supertile* que debe procesar un core, el tiempo de ejecución, el *speedup* y la eficiencia se mantienen sobre los valores calculados analíticamente.

Tabla 18 Incremento del tamaño del problema manteniendo la relación del *supertile*.

Cores	Tamaño del problema (M)	*supertile* (K)	Eficiencia Deseada	Eficiencia Calculada	*speedup* Obtenido
2	140	99	95%	95.22%	1.90
4	198	99	95%	95.23%	3.80
8	280	99	95%	95.22%	7.61
16	396	99	95%	95.23%	15.23
32	560	99	95%	95.22%	30.47
64	792	99	95%	95.23%	60.94
128	1120	99	95%	95.22%	121.88
256	1584	99	95%	95.23%	243.78
512	2240	99	95%	95.22%	487.52
1024	3168	99	95%	95.23%	975.14

Cuando se realiza una variación del tamaño de la aplicación se debe recalcular con los nuevos valores utilizando el modelo analítico de la metodología para encontrar el valor ideal del *supertile*. No obstante, si el tamaño de la aplicación SPMD es incrementado manteniendo el tamaño del *supertile*, entonces se obtendrá una ejecución para ese número de cores con un *speedup* aproximadamente lineal con respecto al número de cores a utilizar.

Este estudio de escalabilidad se realizó para visualizar cómo se adaptaría la aplicación en caso de no tener la cantidad de cores que determina el modelo, y así poder determinar el tamaño ideal de *tiles* que la aplicación

necesita para un número de cores, con el fin de lograr una eficiencia sobre el valor deseado y con una máxima aceleración.

La Figura 60 detalla los valores teóricos de la aplicación de ejemplo, donde el *speedup* tiene un crecimiento aproximadamente lineal con respecto al número de cores con los que se desea ejecutar la aplicación. Asimismo, la eficiencia se mantiene similar para todos los casos evaluados debido a que el tamaño del problema ha sido ajustado de acuerdo a la proporción de la carga de trabajo de un core (un *supertile*).

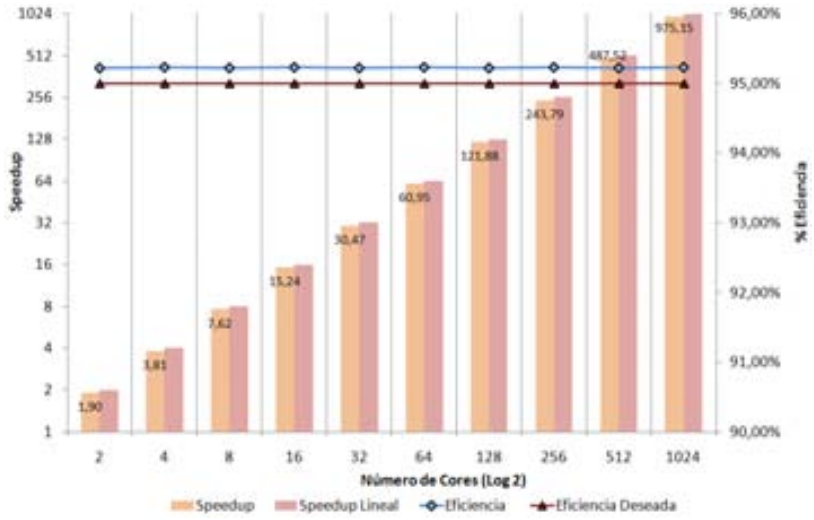


Figura 60 Análisis de la escalabilidad débil manteniendo la relación de la cantidad de trabajo asignada a un core (tamaño del *supertile*)

Este análisis permite al programador evaluar el comportamiento de la aplicación cuando la cantidad de recursos computacionales es menor al calculado, por lo que se puede adaptar el tamaño del problema de la aplicación a la cantidad específica de cores que se posee para la ejecución. Esto se debe realizar considerando el tamaño ideal del *supertile* que mantiene la relación de solapamiento entre el cómputo interno y las comunicaciones de borde, para así lograr la máxima aceleración de la aplicación con una eficiencia definida.

5.5 Evaluación de rendimiento de escalabilidad y eficiencia

Para demostrar la aplicabilidad de la metodología, lo ideal es utilizar aplicaciones reales y un entorno de gran escala para comprobar la

efectividad de la combinación de las métricas de eficiencia y escalabilidad. Es por esto que a continuación se describirán las aplicaciones SPMD utilizadas para validar la combinación de ambas métricas.

5.5.1 Entorno de ejecución de gran escala

Para la comprobación experimental de la aplicación de la metodología en un entorno multicore de gran escala, se ha utilizado el clúster JUROPA [L6] del Forschungszentrum Juelich. Este clúster está compuesto por las características mostradas en la Tabla 19.

Tabla 19 Características clúster Juropa [L6]

Clúster Juropa	Características	Software instalado
Nodos	2208	• SUSE Linux Enterprise Server 11 (x86_64) • Intel Fortran, C/C++ Compiler • Intel Cluster Tools • ParTec MPI Message Passing Interface .
Procesador	2 Intel Xeon X5570 (Nehalem-EP) quad-core processors	
Memoria RAM	24 GB memoria (DDR3, 1066 MHz)	
Memoria Cache	8 MB cache	
Red de Interconexión	Infiniband	
Elementos de Cómputo Total	17664 cores	

Este entorno multicore posee 8 cores por nodo por lo que al evaluar el entorno, el mismo tiene varios niveles de comunicaciones, lo que permite validar la metodología. Asimismo, este clúster es utilizado para comprobar que la metodología con un margen de error menor al 5% puede realizar una planificación tanto espacial (*mapping*), como temporal (scheduling) de la aplicación con el fin de mejorar la eficiencia de la aplicación.

5.5.2 Aplicaciones científicas para la evaluación de la escalabilidad y eficiencia

Como se mencionó en el capítulo anterior, las aplicaciones SPMD utilizadas deben cumplir con un conjunto de características específicas para poder aplicar el modelo planteado en el presente trabajo de investigación [39]. En este sentido, para evaluar la combinación de las métricas de eficiencia y escalabilidad se han utilizado dos aplicaciones, las cuales están

integradas dentro de la suite de simulación para flujos multi-fases basados en el método de energía libre de Lattice Boltzman (LBM), MPlabs [L7]. Ambas aplicaciones corresponden a problemas de dos dimensiones con versiones series y paralelas desarrolladas en el lenguaje de programación fortran y que permiten la simulación del método multiflujo de Lee-Lin [54], así como el método de Zheng-hu-chew [66].

La aplicación que simula la formulación de Lee-Lin permite el cálculo de flujo de dos fases utilizando densidad arbitraria y relaciones de viscosidad. El proceso de integración lo hace mediante una regla trapezoidal que requiere tres pasos: *PreStream, Stream (hydrodynamics)* y *Poststream*. De estas tres funciones principales, el *Pre-Stream* y el *Poststream* son funciones de cómputo por lo que no existe ningún intercambio de información, mientras que la función *Stream* es la que integra todo el proceso de intercambio y cómputo. En este sentido, la metodología es aplicable solamente a la fase de intercambio o la función *Stream*.

La Figura 61 muestra el comportamiento de la aplicación en relación a las tres funciones principales para un tamaño de problema de 2000x2000 *tiles* y para un número de 1000 iteraciones.

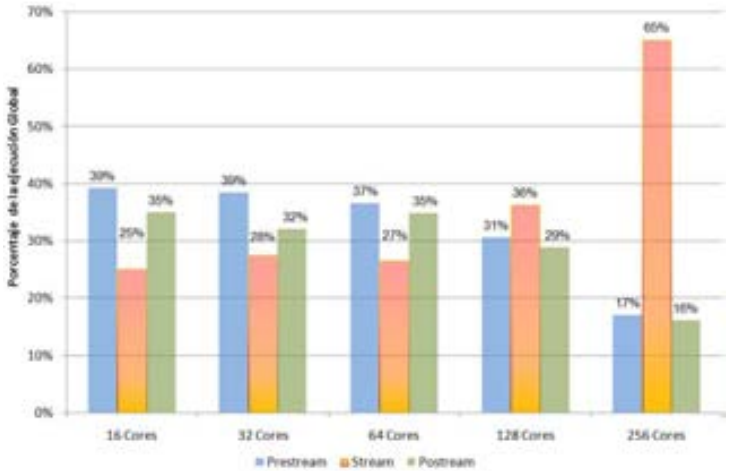


Figura 61 Porcentaje de ejecución de las funciones aplicación LL-2D-STD-MPI

Para este caso en particular, se fija un tamaño de problema y se valida el comportamiento de cada función con respecto al tiempo de ejecución global de la aplicación, Como se puede apreciar, el comportamiento de las funciones *PreStream* y *PostStream* es constante, manteniéndose

aproximadamente entre 39% y 35% respectivamente del tiempo de ejecución global de la aplicación.

El tiempo restante es para la función *Stream* que es la encargada de realizar el intercambio, por esto a medida que se incrementa el número de cores esta función varia los porcentajes constantes de las otras funciones. Por ejemplo, en el caso de la ejecución con 128 y 256 cores el comportamiento de las funciones de la aplicación en relación al tiempo de ejecución no mantienen un comportamiento constante, motivado a los efectos de las comunicaciones incluidas en la función *Stream*. Esto es reflejado en la Figura 62, donde se analiza la función *Stream* tanto a nivel de cómputo como a nivel de comunicaciones.

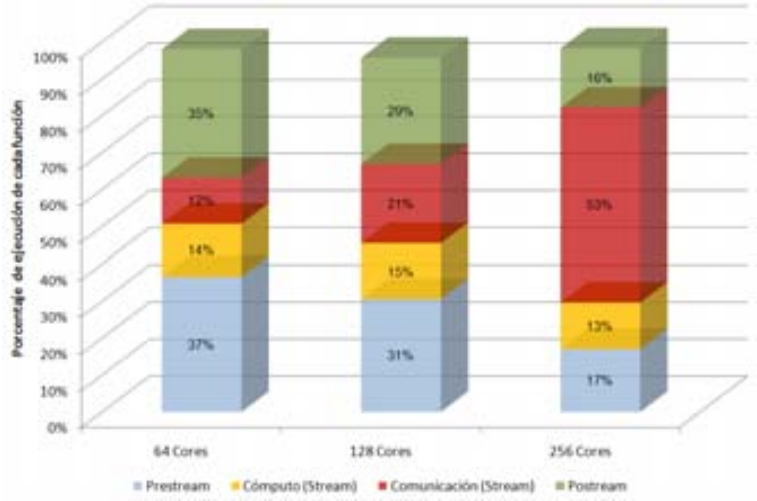


Figura 62 Descomposición de la función Stream (cómputo y comunicación) LL-2D-STD-MPI

Como se puede ver en la figura, a medida que el número de cores se incrementa la función *Stream* tiene mayor impacto en el porcentaje del tiempo total de la ejecución de la aplicación. No obstante, este tiempo es incrementado por las variaciones de las comunicaciones y por el poco cómputo asignado a los cores. Asimismo, se puede evidenciar que el porcentaje del tiempo de cómputo de la función *Stream* se mantiene constante aunque el número de cores se incremente. Al visualizar el comportamiento de la aplicación, se puede deducir que las comunicaciones afectan a la eficiencia y la escalabilidad de la aplicación. En este sentido, la metodología busca minimizar los efectos que las comunicaciones le

adicionan al tiempo de ejecución de la aplicación para mejorar las métricas de rendimiento.

De forma similar, la aplicación del método de Zheng-hu-chew cuenta con 4 funciones principales. No obstante, esta aplicación está compuesta por 2 fases de intercambio de comunicaciones. Las funciones que integran esta aplicación son: Collision, Postcollision, *Stream*, Poststream. Esta aplicación ha sido evaluada de forma similar con un tamaño de 2000x2000 *tiles* y con un número de 1000 iteraciones.

Collison y *Stream* son funciones de cómputo y las otras dos funciones se encargan de hacer el proceso de intercambio de información. Bajo este enfoque, se debe aplicar la metodología a cada una de las fases de intercambio. En esta aplicación no existe ninguna función con sólo cómputo como fue en el caso de la aplicación del método de Lee-Lin, por lo tanto, el porcentaje del tiempo de ejecución de ninguna de las funciones es constante con respecto al tiempo de ejecución de la aplicación.

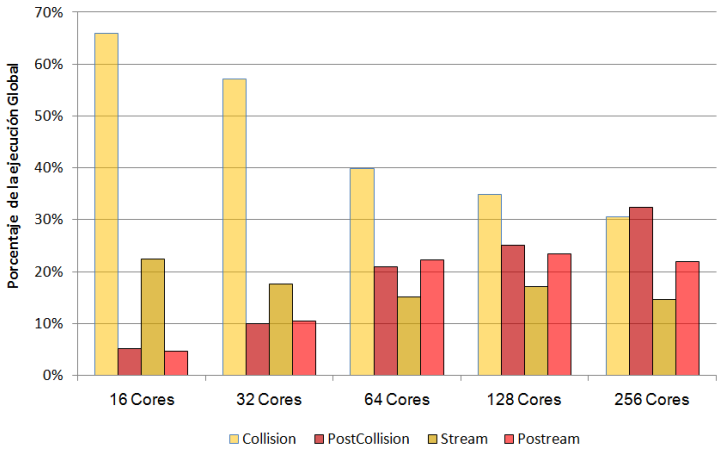


Figura 63 Porcentaje de ejecución de las funciones aplicación ZSC-2D-STD-MPI

Esto es demostrado en la Figura 63, donde la función Collision y *Stream* decrecen con respecto al tiempo de ejecución global. Este comportamiento es debido a que el tamaño del problema (*tiles*) se mantiene y el número de cores se incrementa, haciendo que cada vez la partición asignada a cada core sea más pequeña y por ende el tiempo de cómputo sea menor.

Mientras el porcentaje de tiempos de ejecución de las funciones de comunicaciones PostCollision y Poststream se ven incrementados motivados al efecto de las comunicaciones.

Ante este comportamiento, se puede concluir que a medida que el número de cores se va incrementando, las comunicaciones tiene gran efecto en el tiempo de ejecución de la aplicación. Al analizar, el tiempo de cómputo y comunicación de la aplicación del método de Zheng-hu-chew, se han obtenido los resultados que son ilustrado en la Figura 64, donde se comprueba que a medida que el número de cores aumenta, la proporción del tiempo dedicado a las comunicaciones es mayor, hasta que para un número de 256 cores el tiempo de comunicaciones es más de la mitad del tiempo de ejecución de la aplicación. Por lo tanto, las comunicaciones en esta aplicación afectan seriamente la escalabilidad y la eficiencia de la aplicación.

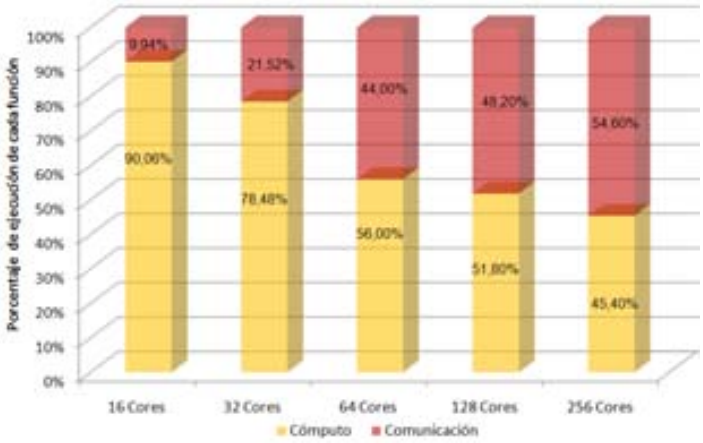


Figura 64 Descomposición de tiempo de cómputo y comunicación Aplicación ZSC-2D-STD-MPI

Una vez descritas las aplicaciones a ser utilizadas en este apartado para comprobar el funcionamiento de la metodología, se evaluará la aplicación de la misma por fases con el objetivo de encontrar el máximo punto de escalabilidad fuerte con la eficiencia definida. Además, se evaluará el comportamiento de la aplicación a nivel de la escalabilidad débil.

5.5.3 Aplicación de la metodología para combinar la escalabilidad y la eficiencia.

- **Caracterización del entorno y las aplicaciones**

Para iniciar la validación de la metodología se realizó el proceso de caracterización, considerando los parámetros definidos para la ejecución de la metodología. En este sentido, la Figura 65 muestra la caracterización de comunicaciones del clúster Juropa. En este clúster se identificaron cuatro niveles de comunicaciones, dos dentro de la arquitectura interna y dos a través de otros enlaces externos al procesador. Se han utilizado 8192 cores para el proceso de caracterización.

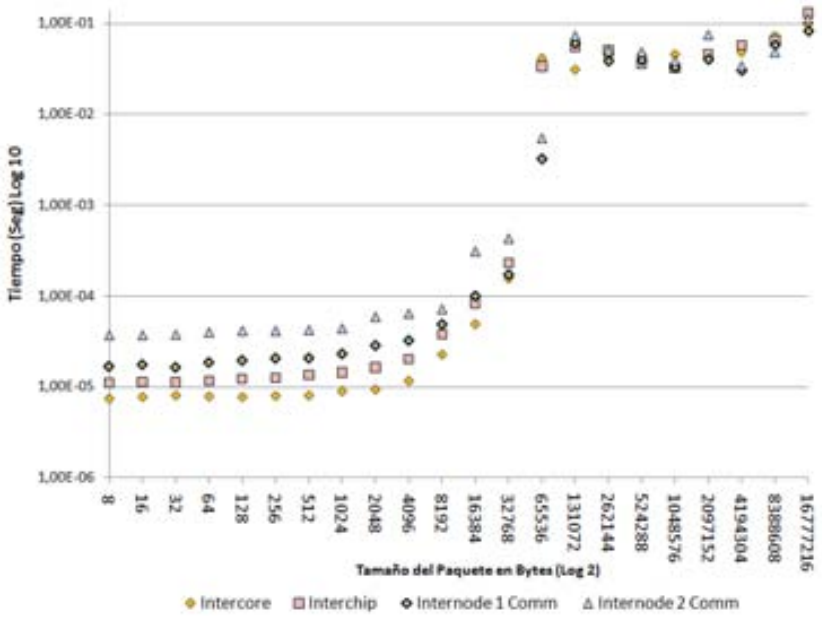


Figura 65 Caracterización de comunicaciones clúster Juropa.

Para evaluar el tiempo de comunicación del sistema, se ha utilizado el patrón de comunicaciones obtenido a través de la herramienta del PAS2P [65]. Tanto la aplicación LL-2D-STD-MPI, como ZSC-2D-STD-MPI tienen un patrón de comunicaciones a ocho vecinos, por lo que la simultaneidad de las comunicaciones debe ser evaluada para simular el comportamiento de la aplicación en el entorno de ejecución.

Una vez encontrado el patrón de comunicaciones se evaluó la ratio cómputo comunicación de los *tiles,* en este sentido la Figura 66 ilustra el comportamiento de esta relación para la aplicación LL-2D-STD-MPI.

Enlace de comunicaciones	Tiempo de Comunicaciones	Tiempo de Cómputo (Cpt)	Ratio λ(p)(w)
Intercore	2.43E-06	1.58E-07	15.38
Interchip	3.54E-06	1.58E-07	22.41
Internode (1C)	1.87E-05	1.58E-07	118.35
Internode (2C)	5.51E-05	1.58E-07	348.73

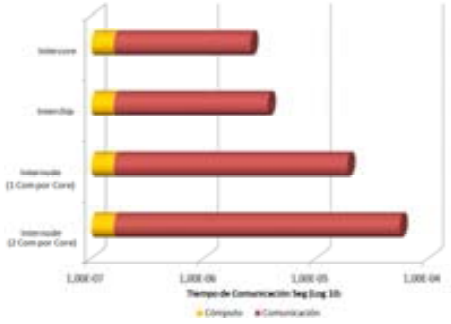


Figura 66 Ratio cómputo y comunicación aplicación LL-2D-STD-MPI

De forma similar, la Figura 67 muestra la ratio de cómputo y comunicación para la aplicación ZSC-2D-STD-MPI. En ambos casos, cada nivel de comunicaciones tiene distintos valores de latencias, por lo que la ratio utilizada para calcular el modelo es la que posea el mayor valor (esto siguiendo los pasos de la metodología).

Enlace de comunicaciones	Tiempo de Comunicaciones	Tiempo de Cómputo (Cpt)	Ratio λ(p)(w)
Intercore	2.42E-06	1.24E-07	19.52
Interchip	3.19E-06	1.24E-07	25.73
Internode (1C)	1.67E-05	1.24E-07	1.34.68
Internode (2C)	1.97 E-05	1.24E-07	158.87

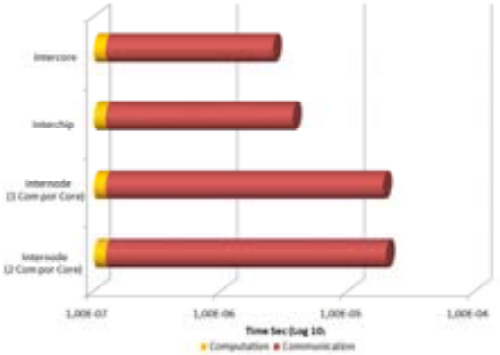


Figura 67 Ratio cómputo y comunicación aplicación ZSC-2D-STD-MPI

- **Modelo de distribución de *tiles***

Para estudiar el apartado de la escalabilidad se ha considerado un tamaño de problema, una eficiencia definida y los valores de caracterización, los cuales se especifican en la Tabla 20 para ambas aplicaciones.

Tabla 20 Datos obtenidos de la caracterización aplicaciones

Aplicación	T. Cómputo $TiempoCpt$ (Seg)	T. Comunicación $TiempoComm(p)(w)$ (Seg)	Tamaño del Problema (M)	Eficiencia deseada (Effic)	Ratio Cómputo-Com (λ)	Máximo error aceptado
LL-2D-STD-MPI	1,58E-07	5,53E-05	7200	90%	349	+/- 5%
ZSC-2D-STD-MPI	1,24E-07	1.97E-05	7015	95%	158	+/- 5%

Para calcular el número de cores y el tamaño ideal del *supertile* que mantiene la relación entre el máximo *speedup* y la eficiencia sobre un umbral definido, se aplican las ecuaciones E12 y E13 respectivamente por ser ambas aplicaciones de 2 dimensiones.

Tabla 21 Resultados obtenidos aplicando el modelo de 2 dimensiones (escalabilidad y eficiencia)

Aplicación	Ncores	K	Cómputo de Borde (Seg)	Cómputo Interno (Seg)	Comunicación de Borde (Seg)	Tiempo de Ejecución (Seg)
LL-2D-STD-MPI	513	318	2,00E-04	1,58E-02	1,75E-02	1,77E-02
ZSC-2D-STD-MPI	2048	155	7,66E-05	2,91E-03	3,06E-03	3,13E-03

Los datos mostrados en la Tabla 21 representan los valores teóricos obtenidos utilizando el modelo para ambas aplicaciones en el entorno de ejecución. Luego, estos resultados se han validado con la ejecución real de la aplicación sin la metodología y aplicando la metodología en el entorno de ejecución.

- **Evaluación de escalabilidad fuerte.**

Para la evaluación de la aplicabilidad de la metodología en ambas aplicaciones en un clúster de gran escala, se utilizaron los valores mostrados en la Tabla 20 y Tabla 21. Para realizar esta evaluación, se analizó la eficiencia y el *speedup* tanto para la aplicación SPMD utilizando y sin utilizar la metodología, con el objetivo de evaluar las mejoras obtenidas entre ambas versiones.

Asimismo, se evaluó el comportamiento teórico de la aplicación de acuerdo a la estrategia de solapamiento entre el cómputo interno y las comunicaciones de bordes.

En este sentido, la Figura 68 muestra el comportamiento del *speedup* de la aplicación LL-2D-STD-MPI, considerando las tres funciones principales (*prestream, stream, poststream*). Como se puede apreciar en la figura, el comportamiento teórico de la aplicación a través del modelo, tiene un crecimiento similar hasta el valor ideal de cores calculado para el tamaño del

problema; de igual forma, la ejecución con la metodología tiene un comportamiento similar al modelo con un error aproximado menor al 5% hasta el valor ideal de cores, a partir de ese punto este error comienza a incrementar por las congestiones de las comunicaciones que no están actualmente validadas en el modelo.

Este valor puede alcanzar hasta un 17% de variación para los casos validados en este entorno. Asimismo, se puede evidenciar la mejora a nivel de aceleración que se tiene entre la aplicación sin aplicar la metodología y aplicando la metodología.

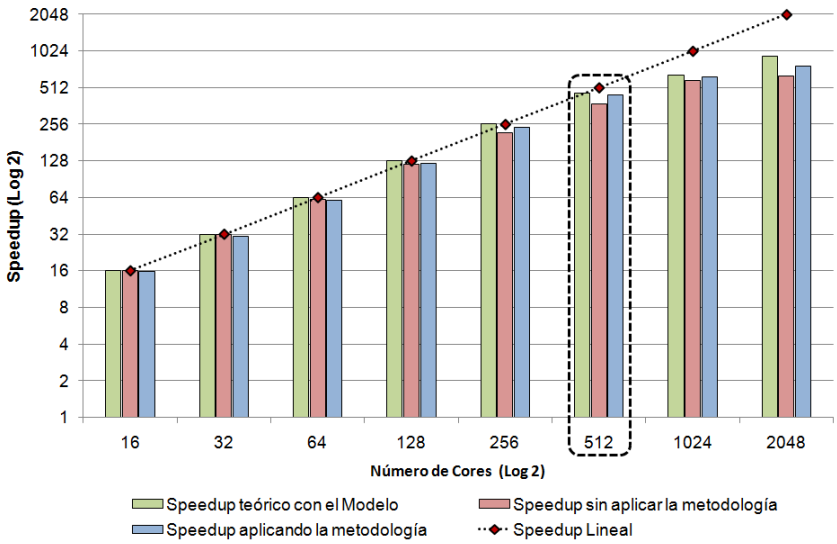


Figura 68 Evaluación de *speedup* aplicación LL-2D-STD-MPI

Luego, la Figura 69 ilustra el comportamiento de la eficiencia donde el margen de error para el valor obtenido de la metodología está alrededor de un 4%, lo que hace que la aplicación de la metodología permita determinar el tamaño ideal y el número de cores con los que se debe ejecutar la aplicación para la mejora de la eficiencia.

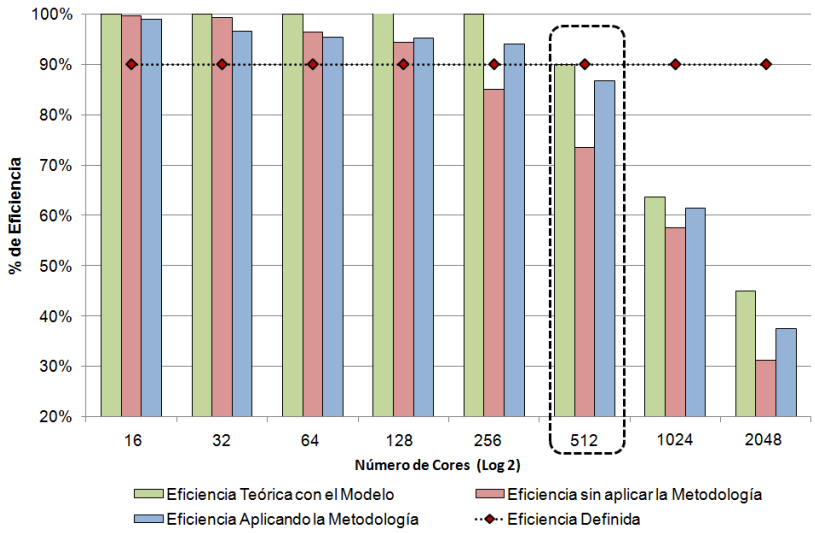


Figura 69 Evaluación de eficiencia aplicación LL-2D-STD-MPI

Como puede ser evidenciado en la Figura 69, la eficiencia antes del valor obtenido está por encima del umbral de eficiencia definido para la aplicación usando la metodología, y en el caso de la aplicación sin la metodología a partir de 256 cores, esta métrica comienza a decrecer considerablemente. El anterior comportamiento es motivado por el efecto de las comunicaciones en la función *stream* y el alto nivel de congestión de los enlaces de comunicación.

Tabla 22 Análisis de eficiencia y *speedup* aplicación LL-2D-STD-MPI

Ncores	Eficiencia sin aplicar la metodología	Eficiencia aplicando la metodología	*Speedup* sin aplicar la metodología	*Speedup* aplicando la metodología
64	96,41%	95,40%	61,70	61,055
128	94,31%	95,31%	120,71	121,997
256	85,04%	94,05%	217,69	240,772
512	73,47%	86,77%	376,16	444,249
1024	57,53%	61,40%	589,13	628,779
2048	31,26%	37,45%	640,22	767,018

Al evaluar los resultados mostrados en la Tabla 22 para valorar la combinación de ambas métricas, se puede observar que el *speedup* continua con un crecimiento para un número mayor de cores; sin embargo, la eficiencia para estos casos es considerablemente inferior a la eficiencia definida por el umbral del 90% que fue mostrado en Tabla 20. Además, al evaluar la mejora en los niveles de eficiencia entre ambas versiones se logró

una mejora alrededor del 18% en eficiencia para la versión que utiliza la metodología con respecto a la versión sin la metodología. Este valor se encuentra al dividir la eficiencia obtenida con la metodología entre la eficiencia encontrada sin aplicar la metodología.

De forma similar al evaluar la aplicación ZSC-2D-STD-MPI con los valores obtenidos de caracterización y del modelo (Tabla 21), se puede detallar que de forma análoga a la aplicación anterior, el modelo analítico tiene un comportamiento considerablemente lineal al valor ideal de *speedup* (Figura 70).

La Figura 70, muestra que el *speedup* de la aplicación utilizando la metodología tiene un comportamiento similar al modelo teórico, con un error menor al 4% hasta el valor ideal de cores calculado. Además, se puede evidenciar la mejora del *speedup* de un 29% con respecto a la versión original, para el valor ideal de cores.

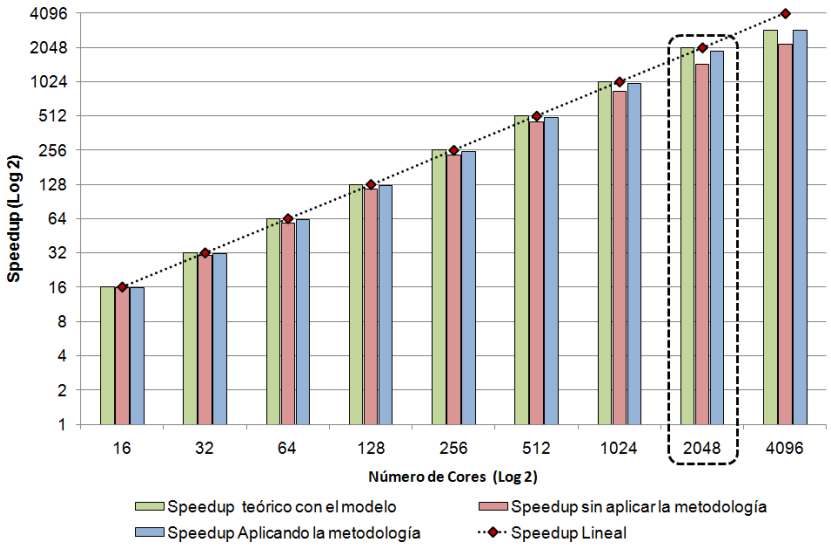


Figura 70 Evaluación de *speedup* aplicación ZSC-2D-STD-MPI

Se puede detallar en la Figura 71, que al evaluar la eficiencia de la aplicación utilizando la metodología, la misma está por encima del umbral definido, mientras que la versión sin aplicar la metodología tiene un valor inferior al umbral a partir de los 64 cores de ejecución. En este caso el margen de error entre el valor calculado por el modelo y el obtenido por la metodología está alrededor del 3,5%. Adicionalmente, se visualiza que la

eficiencia en la versión sin la metodología comienza a decrecer considerablemente a partir de 1024 cores, mientras que en la versión que usa la metodología esta métrica decrece a partir de 4096.

Al evaluar la combinación de ambas métricas la eficiencia y el speedup podemos evidenciar en los resultados de la Tabla 23, que el máximo speedup con la eficiencia cercana al umbral está entre los valores de 1024 y de 2048 cores, debido a que ambos valores tienen una eficiencia cercana al umbral. En el modelo analítico, el valor obtenido para el número ideal de cores es de 2048 con un tamaño cuadrático de supertile de 158 (Tabla 21).

En ambos casos, el error es aproximadamente menor del 3%, no obstante la aceleración computacional sigue incrementando considerablemente cuando se utilizan 2048 cores.

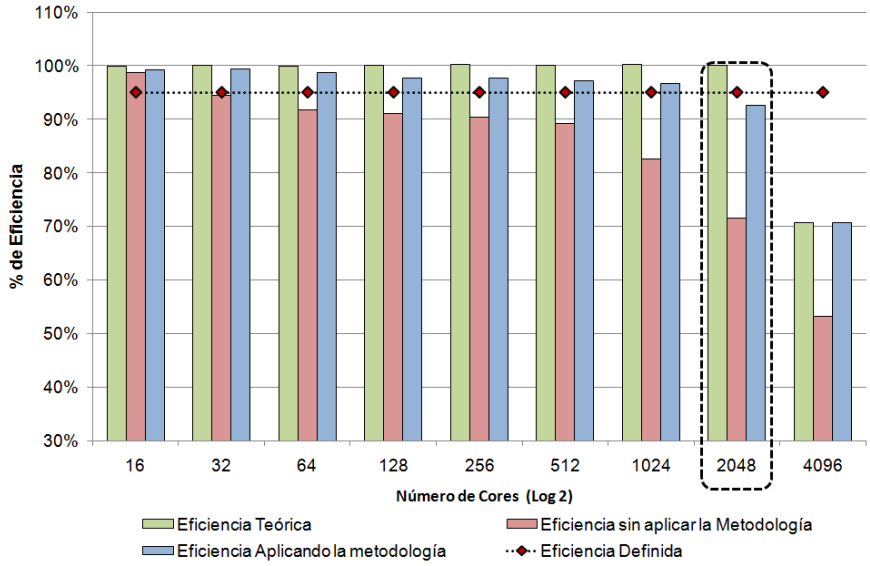


Figura 71 Evaluación de eficiencia aplicación ZSC-2D-STD-MPI

En este caso ambos valores pueden ser considerados válidos; no obstante, al considerar el valor obtenido por el modelo propuesto, el margen de error es muy pequeño y el incremento de la aceleración computacional es considerable. Es por esto, que el valor obtenido por la metodología es válido considerando que se busca el máximo *speedup* y por ende una minimización en el tiempo de ejecución.

Tabla 23 Análisis de eficiencia y *speedup* aplicación ZSC-2D-STD-MPI

Ncores	Eficiencia sin aplicar la metodología	Eficiencia aplicando la metodología	*Speedup* sin aplicar la metodología	*Speedup* aplicando la metodología
128	91,08%	97,63%	116,58	124,969
256	90,45%	97,67%	231,56	250,035
512	89,20%	97,19%	456,72	497,590
1024	82,58%	96,60%	845,58	989,226
2048	71,62%	92,64%	1466,74	1897,367
4096	53,21%	70,64%	2179,68	2893,485

En ambas aplicaciones se ha fijado un tamaño de problema y se logró a través de la metodología encontrar el valor ideal de cores, este valor se ha obtenido con un pequeño margen de error que en el peor de los casos es menor al 5%. El valor ideal de cores encontrado permite mantener una combinación entre la eficiencia y la escalabilidad fuerte como se ha evidenciado [40].

- **Evaluación de escalabilidad débil**

El siguiente análisis hace referencia a la evaluación de la escalabilidad débil de la aplicación. En este sentido se realiza un incremento en el tamaño del problema y en el número de cores para la aplicación LL-2D-STD-MPI. Para hacer esta validación se han considerado los valores iniciales definidos en las Tabla 20 y Tabla 21 (valores de la caracterización y del cálculo del modelo respectivamente). Cuando se realiza un incremento del tamaño en la misma proporción al tamaño de los *supertiles*, se puede mantener la relación entre el máximo *speedup* y una eficiencia establecida sobre un umbral.

Bajo este enfoque la Tabla 24 muestra el incremento del tamaño del problema de la aplicación (conjunto de *tiles*) considerando el valor del tamaño del *supertile*. El valor ideal del *supertile* es de 318^2 (Valor de ***K*** elevado a la dimensión, Tabla 21), es decir, que para el incremento del tamaño del problema se debe mantener esta relación si se desean obtener las prestaciones deseadas.

Tabla 24 Incremento del tamaño del problema y evaluación de rendimiento por iteración

Cores	Div X	Div Y	Tamaño X	Tamaño Y	Tiempo Ejecución (Seg)	Eficiencia	*Speedup*
2	2	1	636	318	3.40420	95.28%	1.91
4	2	2	636	636	3.45621	93.85%	3.75
8	4	2	1272	636	3.46022	93.74%	7.50
16	4	4	1272	1272	3.47622	93.31%	14.93
32	8	4	2544	1272	3.50822	92.46%	29.59
64	8	8	2544	2544	3.46022	93.74%	59.99
128	16	8	5088	2544	3.65223	88.81%	113.68
256	16	16	5088	5088	3.72284	87.13%	223.05
512	32	16	10176	5088	3.73119	86.93%	445.09
1024	32	32	10176	10176	3.69492	87.79%	898.92

Para hacer el incremento del tamaño del problema en proporción al valor del *supertile*, se consideró la distribución lógica asignada a los procesos en la fase del *mapping* dentro de la metodología. En la Tabla 24 se muestra la distribución de los procesos en el mapa cartesiano (DivX, DivY) y el incremento del tamaño del problema que mantiene el tamaño ideal del supertile.

Luego se evaluaron las prestaciones tanto del *speedup* como de la eficiencia cuando se comienza hacer el incremento del número de cores y el tamaño del problema. En la Tabla 24 se puede ver que el tiempo de ejecución en cada una de las ejecuciones se mantiene similar, cuando se incrementa el tamaño del problema y se mantiene la relación del tamaño del *supertile* que se calculó con el modelo, ya que en el peor de los casos tiene una variación alrededor del 5% con respecto a la media de todos los valores. Esto claramente verifica que al mantener la relación del *supertile*, e incrementar el tamaño del problema, el tiempo de ejecución se mantiene similar.

Al evaluar el *speedup* y la eficiencia, se puede detallar en la Figura 72, que el *speedup* tiene un crecimiento similar con respecto al valor del *speedup* teórico. De igual forma la eficiencia se mantiene cercana al valor del umbral definido. En este caso la eficiencia oscila con una diferencia real

de (+/-4%) de error, dependiendo del número de cores con el que se esté ejecutando la aplicación.

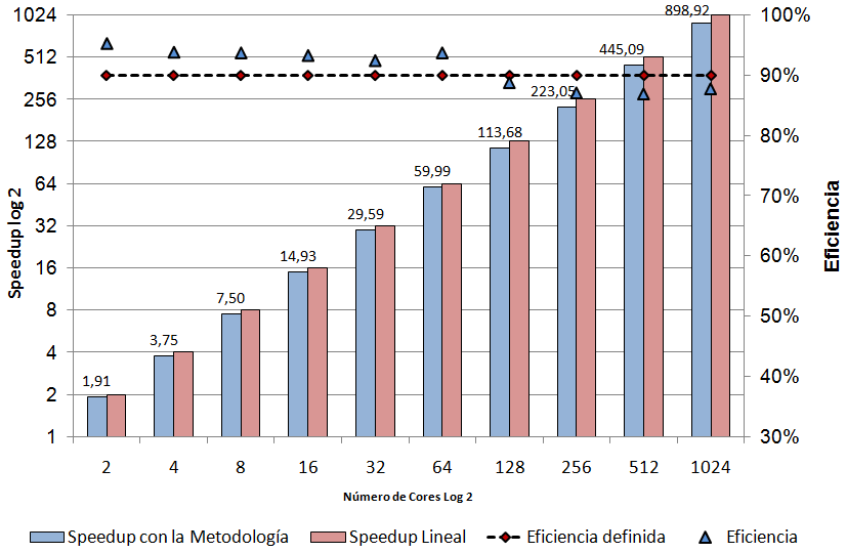


Figura 72 Escalabilidad débil manteniendo la proporción del *supertile* aplicación LL-2D-STD-MPI

Este análisis permite validar que manteniendo la proporción del tamaño del *supertile* las prestaciones ideales del máximo *speedup* y la eficiencia por encima del umbral definido se mantienen, por lo que se puede concluir que el tamaño de la aplicación se puede adaptar a la cantidad de cores disponible, siempre y cuando se mantenga el tamaño del *supertile.*

5.6 Conclusiones

En este capítulo se describió la forma para combinar las métricas de eficiencia y de escalabilidad, haciendo uso de la metodología de ejecución eficiente utilizando las cuatro fases definidas. Estas fases permitieron encontrar el valor ideal del *supertile* y el número de cores que mantiene una escalabilidad lineal de la aplicación. Esto se realizó utilizando el modelo analítico definido en la metodología. Los ejemplos descritos y la experimentación muestran que el valor ideal encontrado cumple con la condición de encontrar el máximo *speedup* con una eficiencia definida. Esto se logra por la forma efectiva de gestionar los diversos enlaces de comunicación de los clústeres con procesadores multicore.

Con respecto a la escalabilidad fuerte, la evaluación de rendimiento presentada mostró que usando la metodología de ejecución eficiente de aplicaciones SPMD se puede encontrar el máximo valor de escalabilidad con una eficiencia definida. Esto se logra al definir un tamaño de problema fijo e incrementando el número de cores para la ejecución. Esto ha sido confirmado con los ejemplos teóricos utilizados y con las aplicaciones utilizadas para la validación ejecutadas en un clúster con procesadores multicore de gran escala.

Asimismo, en este capítulo se pudo detallar cómo las aplicaciones SPMD con las características definidas se comportan de forma similar a los valores aportados por el modelo analítico. El principal propósito de encontrar el máximo punto de escalabilidad fuerte, es para determinar el número de cores que logra alcanzar el máximo punto de escalabilidad lineal con respecto al número de cores. Como se pudo observar, a partir del punto encontrado con el modelo analítico, la eficiencia y la escalabilidad quedan fuertemente afectadas por el comportamiento limitado de comunicaciones que tiene la aplicación.

Finalmente, el estudio de la escalabilidad débil permite determinar cuál es el tamaño ideal de la aplicación para ser ejecutada con un número de cores específicos; es decir, cómo la aplicación primero se debe adaptar para mantener el máximo *speedup* con una eficiencia definida sobre un umbral. Al lograr incrementar el tamaño del problema de la aplicación manteniendo la proporción con el *supertile* que mantener un solapamiento ideal, se logra que las métricas de rendimiento tengan un crecimiento lineal y con un error menor del 4% como pudo ser evidenciado en la evaluación experimental. Al mantener la proporción del *supertile* y el tamaño del problema, nos permite escalar el problema para ejecutarlo con una mayor cantidad de cores.

Este apartado ha mostrado cómo haciendo uso de la metodología se pueden mejorar las métricas de rendimiento por el eficiente manejo de la comunicaciones dentro de la aplicación.

Capítulo 6

Conclusiones, principales aportaciones y líneas abiertas

Resumen

Este capítulo presenta las conclusiones alcanzadas relacionadas al desarrollo y aplicación de la metodología para la ejecución eficiente de aplicaciones SPMD en clústeres con procesadores multicore. Además, se muestran las principales aportaciones de este trabajo de investigación. Este capítulo culmina describiendo las líneas abiertas, con el objetivo de dar continuidad al trabajo de investigación presentado.

6.1 Conclusiones

En esta tesis se presentó una metodología para lograr la ejecución eficiente de aplicaciones SPMD en clústeres con procesadores multicore. El principal objetivo de la metodología es encontrar el máximo *speedup* mientras la eficiencia se mantiene por encima de un umbral definido. La metodología integra cuatro fases principales (caracterización, modelo de distribución de *tiles*, *mapping* y *scheduling*) que permiten gestionar y minimizar las comunicaciones en los entornos jerárquicos de comunicaciones, como los incluidos en los entornos con procesadores multicore.

Esto se obtiene, considerando los valores de caracterización de la máquina, que luego servirán como entradas al modelo de distribución de *tiles*. Este modelo permitió calcular las agrupaciones de *tiles* en *supertile*. Como un *supertile* es una unidad que se compone de *tiles* internos y externos, permite implementar una estrategia de solapamiento que minimiza los efectos de las comunicaciones entre los cores de ejecución.

Los resultados obtenidos para la validación de la metodología muestra que el modelo con un error menor al 5%, puede determinar el tamaño ideal del *supertile* que luego será asignado a cada core de ejecución.

Una vez calculado el tamaño de *supertile* y dado un tamaño de problema en tiles, se puede calcular el número ideal de cores necesario para mantener una estrecha relación entre el máximo *speedup* y la eficiencia definida.

Al analizar los resultados obtenidos al aplicar la metodología, se pudo comprobar que se logran mejoras considerables con respecto a la versión sin aplicar la metodología. En el mejor de los casos se alcanzó hasta un 39% de eficiencia para los casos presentados en este documento y en el peor de los casos tuvo una mejora superior al 12% para el valor ideal de cores obtenido por el modelo. Estas mejoras son logradas por la adecuada adaptación de la aplicación al entorno de ejecución.

Luego, se ha propuesto una estrategia de planificación espacial (*mapping*) donde se busca minimizar el número de comunicaciones en el entorno. Finalmente se propuso una política para hacer la planificación temporal

(*scheduling*), la cual permitió gestionar la estrategia de solapamiento entre el cómputo interno del *supertile* y las comunicaciones de los bordes.

La metodología fue evaluada en diversas arquitecturas con procesadores multicore y para un conjunto de aplicaciones provenientes de diversos campos. Los resultados obtenidos permiten concluir que la metodología logra que la aplicación se adapte de una forma más adecuada al entorno de ejecución, mejorando así las métricas de rendimiento como el *speedup* y la eficiencia.

Una de las limitaciones que tiene actualmente la metodología es que las aplicaciones SPMD tienen que cumplir un conjunto de características (tales como ser regular, estática y local). Esto origina que se incluya cierto determinismo que luego es traducido en uno de los puntos importantes para la precisión del modelo.

Una vez desarrollada y validada la metodología, se realizó un estudio sobre la aplicabilidad de la misma para analizar la mejora de la escalabilidad fuerte y débil de la aplicación, al considerar una eficiencia definida. En este sentido, se realizó un estudio de escalabilidad utilizando un clúster de gran escala compuesto por 24 mil cores de ejecución.

Los resultados obtenidos, mostraron que utilizando la metodología se puede alcanzar el máximo punto de escalabilidad fuerte cuando se define un tamaño de problema y se comienza a incrementar el número de cores. Este valor alcanza un crecimiento similar al del *speedup* teórico mientras la aplicación tiene un comportamiento limitado por cómputo. A partir del punto ideal las prestaciones comienzan a decrecer. Por esta razón, al determinar este punto se puede ejecutar la aplicación con el número calculado de cores con que se consigue el máximo *speedup* con la eficiencia cercana al umbral definido.

De igual forma, se evaluó la aplicabilidad de la metodología para mantener la escalabilidad débil de la aplicación y se logró concluir que manteniendo la relación del tamaño del *supertile*, se puede lograr que la aplicación se adapte al número de cores con que se desea ejecutar. Es decir, si el crecimiento del tamaño del problema aumenta de forma

proporcional al valor del tamaño del *supertile* y el número de cores, entonces se puede lograr un máximo *speedup* con la eficiencia sobre el umbral, para ese tamaño de problema y con ese número de cores específico, lo cual fue comprobado experimentalmente.

Como fue mostrado para ambos tipos de escalabilidad se puede lograr determinar el número de cores ideales y el tamaño del *supertile,* con el objetivo de tener el máximo punto de escalabilidad bajo una eficiencia definida.

Finalmente, al aplicar la metodología de ejecución eficiente se logró hacer un mejor uso de los recursos por parte de las aplicaciones SPMD evaluadas. Los resultados mostraron que las versiones con la metodología se adaptaban de mejor manera a los diversos entornos jerárquicos de comunicaciones utilizados.

6.2 Principales contribuciones

Las principales aportaciones de esta tesis están resumidas en las siguientes publicaciones realizadas:

> *Muresano R, Rexachs D. y Luque E., "Administración Eficiente de Recursos de Cómputo Paralelo en un Entorno Heterogéneo," publicado en XX jornadas de paralelismo, A Coruña España, pp.427-432, 2009.*

Esta publicación describe la problemática de los niveles de comunicación y cómo las mismas afectan el rendimiento de las aplicaciones SPMD. Se plantea una solución aplicando estrategias de solapamiento entre el cómputo y comunicación, con el fin de mejorar las métricas de rendimiento.

> *Muresano R, Rexachs D. y Luque E. , "How SPMD applications could be efficiently executed on multicore environments?," In prooceeding of International Conference on Cluster Computing (CLUSTER), New Orleans, USA, 2009, pp. 1-4.*

Se desarrolla una primera aproximación de la metodología, donde se definían las fases de caracterización, *mapping* y *scheduling*. La principal

aportación de este trabajo fue el análisis y el desarrollo de un método para la mejoras de las prestaciones en entornos jerárquicos de comunicaciones.

Por otra parte, para lograr una ejecución eficiente se deben gestionar de forma adecuada las diferencias de los enlaces de comunicación. Para esto se diseñó un método de *mapping* que aplicando estrategias de afinidad de cores permite minimizar el volumen de comunicaciones por los enlaces de comunicación, este método fue validado en la siguiente publicación.

> *Muresano R, Rexachs D. y Luque E. , "Could be improved the efficiency of SPMD applications on heterogeneous environments?. ," XV Congreso Argentino de Ciencias de la Computación (CACIC), Jujuy , Argentina 2009.*

La integración de las fases de la metodología y la primera aproximación del modelo analítico fue expuesta y publicada en la siguiente publicación.

> *Muresano R, Rexachs D. y Luque E. , "Methodology for efficient execution of SPMD applications on multicore Clusters", In Proceedings of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID), Melbourne, Australia, pp 185-195, 2010.*

Los resultados mostraron cómo las aplicaciones SPMD aplicando estrategias de planificación espacial y temporal se adaptaban a la arquitectura multicore mejorando las métricas de rendimiento. Las ganancias obtenidas en eficiencia y *speedup* en la versión con la metodología fueron considerables en relación a la versión sin la metodología.

En la siguiente publicación, se demostró la aplicabilidad de la metodología en diferentes arquitecturas con procesadores multicore y con un conjunto de aplicaciones SPMD. En este trabajo se presentaron los pasos a seguir y el diagrama funcional para utilizar la metodología en una aplicación SPMD. Los resultados mostraron que las aplicaciones SPMD evaluadas se adaptan de forma adecuada a las arquitecturas con procesadores multicore testeadas logrando obtener el máximo speedup manteniendo la eficiencia sobre el umbral definido.

Muresano R, Rexachs D. y Luque E. , "A tool for efficient execution of SPMD applications on multicore clusters," In Proceedings of 10 International Conference in Computational Sciences (ICCS), Amsterdam Netherland, pp 2599-2608, 2010.

La siguiente publicación demuestra cómo creando supertiles se puede encontrar el máximo punto de escalabilidad de la aplicación SPMD. La aplicación de la metodología muestra que con un pequeño margen de error que en el peor de los casos es menor al 5%, se logra obtener el número ideal de cores que permiten encontrar el máximo speedup con una eficiencia definida.

Muresano R, Rexachs D. y Luque E. "Combining scalability and Efficiency for SPMD Applications on multicore," accepted to be presented and published on the International conference on Parallel and Distributed Processing techniques and applications (PDPTA), Las Vegas, USA 2011, to be appear.

Se hizo una adaptación del modelo analítico final, con ejemplos teóricos y prácticos se muestra la aplicabilidad del modelo para encontrar tanto el tamaño ideal del *supertile* como el número de cores ideales que permite alcanzar el máximo speedup con una eficiencia definida. Esto fue publicado en el siguiente capítulo de libro.

Muresano R, Rexachs D. y Luque E. "An Approach for Efficient Execution of SPMD Applications on Multi-core Clusters." In S. Pllana, F. Xhafa (Eds) Programming Multi-core and Many-core Computing Systems (pp. to be appear), Ed. Wiley Series on Parallel and Distributed Computing, 6079, 2011.

Todos estos trabajos validan la aplicabilidad de la metodología al usar aplicaciones SPMD en un entorno jerárquico de comunicaciones, como es el caso de clúster con procesadores multicore.

6.3 Líneas Abiertas

Este trabajo plantea nuevas líneas de investigación derivadas a la aplicabilidad de la metodología para la ejecución eficiente de aplicaciones SPMD en un sistema jerárquico de comunicaciones.

En primer lugar se plantea la adaptación de la metodología para incluir entornos con procesadores heterogéneos. Este planteamiento es principalmente por la nueva tendencia de la integración de GPU más los entornos multicore. Asimismo, se puede encontrar clústeres compuestos por diferentes arquitecturas multicore, motivado a la actualización de los mismos. Estos entornos heterogéneos tanto en cómputo como en comunicación hacen un reto que puede ser planteado a través de la metodología de ejecución eficiente.

Una vez resuelto ese caso se puede aplicar la metodología en un entorno con mayores niveles de comunicaciones como es el caso de los entornos multiclusters con procesadores multicore. En este tipo de entornos, se adiciona uno o más niveles de comunicaciones por lo que se debe igualmente considerar para la aplicabilidad de la metodología el nivel de comunicaciones más lento.

No obstante, este tipo de entornos tiene gran variabilidad a nivel de las comunicaciones entre clústeres, y a nivel de cómputo por la heterogeneidad de los clústeres. Por lo tanto, se deben incluir en la metodología estrategias para gestionar el desbalanceo tanto para el cómputo como para la comunicación de modo de ejecutar aplicaciones SPMD eficientemente en un sistema multicluster.

Referencias Bibliográficas.

[1] Anderson J. and Calandrino J. “Parallel task scheduling on multicore platforms,” Special Interest Group on Embedded Systems (SIGBED) ACM Review - Special issue: The work-in-progress (WIP), Vol. 3, Issue 1, pp.1-6, 2006.

[2] Bekas C., Curioni A., Arbenz A, Flaig C, Van Lenthe G., M¨uller R., and Wirth A., “Extreme scalability challenges in micro-finite element simulations of human bone,” Journal Concurrency and Computation: Practice and Experience, Vol. 22, Issue 16, pp. 2282–2296, 2010.

[3] Brehm P., Patrick H., and Mahukar M., “Performance modeling for SPMD message-passing programs,” Journal Concurrency and Computation: Practice and Experience, Vol. 10, Issue 5, pp. 333-357, 1998.

[4] Brunet E., Denis A., Namyst R. and Trahay F., "A multithreaded communication engine for multicore architectures," In proc. of IEEE International Symposium on Parallel and Distributed Processing (IPDPS), Miami, USA, pp. 1-7, 2008.

[5] Broquedis F., Clet-Ortega J., Moreaud S., Furmento N., Goglin B., Mercier G., Thibault S. and Namyst R., “Hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications,” In proc. of the 18th Euromicro International Conference on Parallel, Distributed and Network Based Computing (PDP), Pisa, Italia, pp. 180-186, 2010.

[6] Broquedis F., Furmeto N., Goglin B., Namyst R., and Wacrenier P. “Dynamic Task and Data Placement over NUMA Architectures: An OpenMP Runtime Perspective,” In proc. of 5th International Workshop on OpenMP (IWOMP), Dresden, Germany, pp. 79-92, 2009.

[7] Buyya R. “High Performance Cluster Computing: Architectures and Systems,” Prentice Hall PTR Upper Saddle River, New Jersey, USA, 1999.

[8] Cierniak M., Javeed M. and Li W., “Compile-time Scheduling Algorithms for Heterogeneous Network of Workstations,” The Computer Journal, Vol. 40, Issue 2, pp. 40-46, 1997.

[9] Cermele M., Colajanni M., and Tucci S. “Adaptative Load Balancing of Distributed SPMD computations: A Transparent Approach”, Future Generation Computer Systems 16, Tech. Report, Dipartimento di Informatica, Sistemi e Produzione, Universita’ di Roma, Italia, pp. 2-9, 2000.

[10] Cruz C. “Estrategias Coordinadas Paralelas Basadas en Soft Computing para la solución de Problemas de Optimización,” Tesis doctoral en informática, Departamento de ciencias de la computación e inteligencia artificial, Universidad de Granada, España, 2005.

[11] Chai L, “High Performance and scalable MPU intra-node communications middleware for multicore clusters.” Doctoral Thesis, The Ohio State University, USA, 2009.

[12] Chai L, Gao Q., and Panda D., “Understanding the Impact of Multi-Core Architecture in Cluster Computing: A case study with Intel Dual Core system,” In proc. of IEEE International Symposium on Cluster Computing and the Grid (CCGRID), Rio de Janeiro, Brasil, pp. 471-478, 2007.

[13] Chatterjee N., Pugsley B., Spjut J. and Balasubramonian R., “Optimizing a Multi-Core Processor for Message-Passing Workloads,” In proc. of International Symposium on Performance Analysis of Systems and Software (ISPASS), Boston, USA, pp. 1-8, 2009.

[14] Chen X., Zhonghai L., Jantsh A. and Chen S., “Speedup Analysis of Data-parallel Applications on Multi-core NoCs,” In proc. of 8th International Conference (ASIC), Changsha, China, pp. 105-108, 2009.

[15] Culler D. and Pal J., “Parallel Computer Architecture a harware and software approach,”editorial Meghan Keeffe, San Francisco, USA, 1999.

[16] Darema F., George D., Norton V. and Pfister G., “A single-program-multiple-data computational model for EPEX/FORTRAN,” Journal of parallel computing, Vol 7, Issue 1, pp. 11-24, 1988.

[17] Darema F. “The SPMD Model: Past, Present and Future,” In proc. of the 8th Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (European PVM/MPI), London, United UK, pp. 1, 2001.

[18] Domeika M. “Software development for embedded Multi-core systems,” Elsevier, USA, 2008.

[19] Dummler J., Rauber T. and Runger G., “Scalable computing with parallel tasks,” In proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers (MATGS), Portland, USA, pp. 1-9, 2008

[20] El-Mahdy A. and El-Shishiny H., "An efficient load-balancing algorithm for image processing applications on multicore processors," In proc. of the 1st International Forum on Next-generation Multicore/Manycore Technologies (IFMT), New York, USA, pp. 1-5, 2008.

[21] Fraser D., Kowalik M. and Gepner P., "Performance evolution and power benefits of cluster system utilizing quad-core and dual-core Intel Xeon processors," In proc. of the 7th International Conference on Parallel Processing and Applied Mathematics (PPAM), Gdansk, Poland, pp. 20-28, 2008.

[22] Foster I., “Designing and Building Parallel Programs”, Addison Wesley Publishing Company, 1994.

[23] Fox G, Johnson M, Lyzanga G. Otto S, Salmon S. and Walker D. “Solving problems on concurrent processors,” Pretince Hall, Vol 1, 1988.

[24] Guo J., Bikshandi G., Basilio B., Fraguela, Garzan M and Padu D, “Programming with tiles,” In proc. of the 13th ACM Symposium on Principles and Practice of Parallel Programming (PPoPP), New York, USA, pp. 111-122, 2008.

[25] Hennessy J, Patterson D., and Goldberg D., “Computer Architecture: a quantitative approach, ” Editorial Elsevier Science, Third edition, 2003.

[26] Hoisie A., Lubeck O., and Wasserman H., “Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications,” Journal of High Performance Computing Applications, Vol. 14, Issue 4, pp. 330–346, 2000.

[27] Jeannot E., and Mercire G., “Near-Optimal Placement of MPI Processes on Hierarchical NUMA Architectures,” In proc. of the 16th International Euro-Par Conference on Parallel Processing (Europar), Ischia, Italy, pp. 199-210, 2010.

[28] Lee C., Wang Y., and Yang T., “Global optimization for mapping parallel image processing tasks on distributed memory machines,” University of California at Santa Barbara, USA, Technical Report, 1997.

[29] Liebrock L. and Goudy S., “Methodology for modeling spmd hybrid parallel computation,” Journal Concurrency and Computation: Practice and Experience, Vol. 20, Issue 8, pp. 903–940, 2008.

[30] Mattson T., “Scientific Computation”, in Parallel and Distributed Computing Handbook, McGraw-Hill, pp. 981–1002 , 1996.

[31] Mccool M. D, "Scalable programming models for massively multicore processors," Proceedings of the IEEE , Vol. 96, Issue 1, pp. 816-831, 2008.

[32] Mercier G. and Clet-Ortega, “Towards an Efficient Process Placement Policy for MPI Applications in multicore Environments,” In proc. of the 16th Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (European PVM/MPI), Espoo, Finland, pp. 104-115, 2009.

[33] Moreno L., “Computación Paralela y Entornos Heterogéneos”, *Tesis Doctoral*, Universidad de la Laguna, 2005.

[34] Muresano R, Rexachs D. y Luque E., “Administración Eficiente de Recursos de Cómputo Paralelo en un Entorno Heterogéneo,” En proc. de XX Jornadas de Paralelismo, A coruña, España, pp.427-432, 2009.

[35] Muresano R., Rexachs D., and Luque E. , “How SPMD applications could be efficiently executed on multicore environments?," In proc. of International Conference on Cluster Computing (CLUSTER), New Orleans, USA, pp. 1-4, 2009.

[36] Muresano R., Rexachs D., and Luque E., “Could be improved the efficiency of SPMD applications on heterogeneous environments?.,” In proc. of the XV Congreso Argentino de Ciencias de la Computación (CACIC), Jujuy , Argentina, pp. 34-43, 2009.

[37] Muresano R., Rexachs D., and Luque E., “A tool for efficient Execution of SPMD applications on multicore Clusters,” ?.,” In proc. of 10 International Conference in Computational Sciences (ICCS), Amsterdam Netherland, pp 2599-2608, 2010.

[38] Muresano R., Rexachs D., and Luque E., “Methodology for efficient execution of SPMD applications on multicore Clusters”, In proc. of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID), Melbourne, Australia, pp. 185-195, 2010.

[39] Muresano R., Rexachs D., and Luque E., “Scalability and Efficiency for SPMD Applications on multicore,” presented on State of the Art in Scientific and Parallel Computing Conference, Iceland, Reykjavik, pp 1-4, 2010.

[40] Muresano R., Rexachs D., and Luque E., “Combining scalability and Efficiency for SPMD Applications on multicore,” accepted to be published and presented on the International conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, USA 2011, To appear.

[41] Muresano R., Rexachs D., and Luque E., “An Approach for Efficient Execution of SPMD Applications on Multi-core Clusters,” In S. Pllana, F. Xhafa (Eds) Programming Multi-core and Many-core Computing Systems Ed. Wiley Series on Parallel and Distributed Computing, 6079, 2011. (pp. to appear).

[42] Nielsen I., y Curtis L. Janssen, "Multicore challenges and benefits for high performance scientific computing," Journal of Scientific Programming Complexity in Scalable Computing, Vol. 16, Issue 4, pp. 277-285, 2008.

[43] Oliveira A, Argolo G., Iglesias P., Martins S. and Plastino A. “Evaluating a scientific SPMD application on a computational grid with different load balancing techniques,” In proc. of 5th International School and Symposium Advanced Distributed Systems (ISSADS), Guadalajara, Mexico, pp. 301-311, 2005.

[44] Oliver A. and Robert Y., “Data allocation strategies for dense linear algebra on two-dimensional grids with heterogeneous communication links,” Institut National de Recherche en Informatique (INRIA), Technical Report, 2001.

[45] Olukotun K., Hammond L., and Laudon J., “Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency,” Morgan & Claypool, 2007.

[46] Panshenskov M, and Vakhitov A., “Adaptive scheduling of parallel computations for SPMD tasks,” In proc. of the International Conference on Computational Science and Its Applications (ICCSA), San Franciso, USA, pp. 38–50, 2007.

[47] Pastor L. and Bosque L., “An efficiency and scalability model for heterogeneous clusters,” In proc. of the IEEE International Conference on Cluster Computing (CLUSTER), Washington, USA, pp. 427-434, 2001.

[48] Peng L., Kunaseth M., Dursun H., Nomura K., Wang W., Kalia R., Nakano A., and Vashisht P., “A scalable hierarchical parallelization framework for molecular dynamics simulation on multicore,” In proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, USA, pp.97-103, 2009.

[49] Pinto L., Tomazella L y Dantas M. “An Experimental Study on How to Build Efficient Multi-core Clusters for High Performance Computing,” In

proc. of International Conference on Computational Science and Engineering (CSE), Washington, USA, pp. 33-40, 2008.

[50] Ramanathan R., "Intel Multi-Core Processors: Leading the Next Digital Revolution," Intel Magazine, Vol. 1 , Issue 1, pp. 1-8, 2005.

[51] Shameem A. and Jason R. “Multicore Programming,” Intel Press, USA, 2006.

[52] Sanyal S. and Das S., “Match: *mapping* data-parallel tasks on a heterogeneous computing platform using the cross-entropy heuristic,” In proc. of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Denver, USA, pp. 64b–64b, 2005.

[53] Sorensen J. and Baden S., “Hiding Communication Latency with non-SPMD, Graph-Based Execution,” In proc. of the 9th International Conference on Computational Science (ICCS), Lousiana, USA, pp 1-8, 2009.

[54] T. Lee T. and Lin C., “A stable discretization of the lattice Boltzmann equation for simulation of incompressible two-phase flows at high density ratio”, Journal of Computing physics, Vol. 206 Issue 1, pp. 16-47, 2005.

[55] Quinn M., “Parallel Computing: Theory and Practice,” McGraw-Hill, 1994.

[56] Valiant G, “A bridging model for parallel computation,” Magazine communication of the ACM, Vol. 33, Issue 8, pp.103-111, 1990.

[57] Valiant G, “A bridging model for multi-core computing,” In proc. of the 16th annual European symposium on Algorithms (ESA), Karlsruhe, Germany, pp 154-166, 2008.

[58] Van der Wijngaart R. and Jin H., “NAS parallel benchmark, multi-zone versions,” NASA Advanced Supercomputing (NAS) Division NASA Ames Research Center, Moffett Field, CA, USA Tech. Report, 2003.

[59] Von Neumann J., “The Principles of Large-Scale Computing Machines,” Journal of Annals of the History of Computing, Vol. 10, Issue 4, pp 243-254, 1988.

[60] Vikram K. and Vasudevan V., “Mapping data-parallel tasks onto partially reconfigurable hybrid processor architectures,” IEEE Transactions on

Very Large Scale Integration Systems, Vol. 14, Issue 9, pp. 1010-1023, 2006.

[61] Weissman J, “Prophet: automated scheduling of spmd programs in workstation networks,” Journal Concurrency and Computation: Practice and Experience, Vol. 11, Issue 6, pp. 301–321, 1999.

[62] Weissman J, and Zhao X., “Scheduling parallel applications in distributed networks,” Journal Cluster Computing, Vol. 1, Issue 1, pp. 109-118, 1998.

[63] Wilkinson B. y Allien M. “Parallel Programing”, Pretince Hall, 1999

[64] Winkowycz. M, “Numerical Heat transfer, Part A applications”, Journal of Computation and Methodology, Vol. 60, Issue 4, pp 74-83, 2009.

[65] Wong A., Rexachs D., and Luque E., “Extraction of Parallel Application Signatures for Performance Prediction,” In proc. of IEEE International Conference of High Performance Computing and Communications (HPCC), Melbourne, Australia, pp. 223-230, 2010.

[66] Zheng, H Shu C y Chew T., “A lattice Boltzmann model for multiphase flows with large density ratio”, Journal of Computing physics, Vol. 218 Issue 1, pp. 353-371, 2006.

[67] Zaki O, Lusk E, Gropp W. Swider D. “Toward Scalable Performance Visualization with Jumpshot,” Journal in High-Performance Computing Applications, Vol. 13, Issue 2, pp. 277-288, 1999.

Referencias electrónicas

[L1] AMD, “AMD Dual-Core Processors: Twice the Processing Power of Single-Core Chip” disponible en: http://www.amd.com/us/products/ Technologies/multi-core-processing/Pages/dual-core-processing.aspx, revisado 20 Marzo del 2011.

[L2] AMD, “AMD Quad-Core Processors”, disponible en: http://www.amd.com/us/products/technologies/multi-core-processing /Pages/quad-core-advantages.aspx, revisado 20 Marzo del 2011.

[L3] IBM, “The Cell Chip”, disponible en http://www.research.ibm.com/cell /cell_chip.html, revisado 10 Marzo del 2011.

[L4] Intel Reseller Center, “Intel Dual-core Technology”, disponible en http://www.intel.com/cd/channel/reseller/asmo-na/eng/products/server/ platform/5000/learn/index.htm, revisado 15 Enero del 2011.

[L5] Intel Reseller Center, “Intel® Core™2 Quad Processors”, disponible en http://www.intel.com/products/processor/core2quad/?wapkw=%28 Quad+Core%29, revisado 15 Enero 2011.
AMD, “AMD Dual-Core Processors: Twice the Processing Power of Single-Core Chip.

[L6] Juropa “JUROPA / HPC-FF System Configuration” disponible en http://www2.fz-juelich.de/jsc/juropa/configuration, revisado 23 Febrero 2011.

[L7] MP-Laps V 1.1 “Multiphase Lattice Boltzmann Suite” disponible en http://code.google.com/p/mplabs/ , revisado 26 de abril de 2011

[L8] McCalpin, “STREAM: Sustainable Memory Bandwidth in High Performance Computers” disponible en http://www.cs.virginia.edu/stre am/ revisado en 10 Abril 2009.

[L9] MPI, “The Message Passing Interface (MPI) standard, ” disponible en http://www.mcs.anl.gov/research/projects/mpi/ revisado 03 Octubre 2010.

[L10] Netpipe, “A Network Protocol Independent Performance Evaluator” disponible en: http://www.scl.ameslab.gov/netpipe/. revisado en 10 Abril 2009

[L11] PVM, “Parallel Virtual Machine.” Disponible en http://www.csm.ornl.gov/pvm/ revisado 03 Octubre 2010.

[L12] Top500, “Top 500 lists”, disponible en http://top500.org/lists/2010/11 revisado 03 de Marzo 2011

[L13] Wave Equation, “Solving the 2D Wave Equation” disponible en https://ccrma.stanford.edu/~jos/pasp/Solving_2D_Wave_Equation.html, revisado 08 de Noviembre 2010