Coordinated Scheduling and Dynamic Performance Analysis in Multiprocessor Systems

Julita Corbalán González

Departament d'Arquitectura de Computadors

Universitat Politècnica de Catalunya (UPC)

Barcelona (SPAIN), June 2002

A THESIS SUBMITTED IN FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor per la Universitat Politècnica de Catalunya

Coordinated Scheduling and Dynamic Performance Analysis in Multiprocessor Systems

Author: Julita Corbalán González

Advisor: Jesús José Labarta Mancho

Co-Advisor: Xavier Martorell Bofill

Approved, Thesis Committee

....

....

....

....

....

Its time...

to design and build computing systems capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads we put upon them. These autonomic systems must anticipate needs and allow users to concentrate on what they want to accomplish rather than figuring how to rig the computing systems to get them there.

AUTONOMIC COMPUTING
IBM’s Perspective on the State of Information Technology
IBM 2001

Abtract

The performance of current shared-memory multiprocessor systems depends on both the efficient utilization of all the architectural elements in the system (processors, memory, etc), and the workload characteristics. This Thesis has the main goal of improving the execution of workloads of parallel applications in shared-memory multiprocessor systems by using real performance information in the processor scheduling.

In multiprocessor systems, users request for resources (processors) to execute their parallel applications. The Operating System is responsible to distribute the available physical resources among parallel applications in the more convenient way for both the system and the application performance.

It is a typical practice of users in multiprocessor systems to request for a high number of processors assuming that *the higher the processor request, the higher the number of processors allocated, and the higher the speedup achieved by their applications*. However, this is not true. Parallel applications have different characteristics with respect to their scalability. Their speedup also depends on run-time parameters such as the influence of the rest of running applications.

This Thesis proposes that the system should not base its decisions on the users requests only, but the system must decide, or adjust, its decisions based on real performance information calculated at run-time. The performance of parallel applications is an information that the system can dynamically measure without introducing a significant penalty in the application execution time. Using this information, the processor allocation can be decided, or modified, being robust to incorrect processor requests given by users. We also propose that the system use a target efficiency to ensure the efficient use of processors. This target efficiency is a system parameter and can be dynamically decided as a function of the characteristics of running applications or the number of queued applications.

We also propose to coordinate the different scheduling levels that operate in the processor scheduling: the run-time scheduler, the processor scheduler, and the queueing system. We propose to establish an interface between levels to send and receive information, and to take scheduling decisions considering the information provided by the rest of levels. In particular, we propose that the processor scheduler decides when a new application can be started and let the decision about which application to start to the queueing system.

The evaluation of this Thesis has been done using a practical approach. We have designed and implemented a complete execution environment to execute OpenMP parallel applications. We have introduced our proposals, modifying the three scheduling levels (run-time library, processor scheduler, and queueing system): At the run-time level we have implemented some techniques to improve the run-time behavior in a multiprogrammed multiprocessor system, including the coordination with the O.S. scheduler. We have also proposed a mechanism to dynamically measure the performance of parallel applications. At the processor scheduling level, we have mainly proposed several scheduling policies to include the performance information in the processor allocation policy. We have also specified the mechanism to coordinate the processor scheduler with the queueing system. At this level we have also done proposals to work both in space-sharing and in gang scheduling policies. At the queueing system we have mainly incorporated the coordination with the processor scheduling level.

Results show that the ideas proposed in this Thesis of (1) measuring the applications performance at run-time to decide and / or adjust the processor allocation, (2) imposing a target efficiency to ensure the efficient use of resources, and (3) coordinating the different scheduling levels, significantly improve the system performance. If the evaluated workload has been previously tuned, in the worst case, we have introduced an slowdown around 5% in the workload execution time compared with the best execution time achieved. However, in some extreme cases, with a workload and a system configuration not previously tuned, we have improved the system performance in a 400%, also compared with the next best time.

The main results achieved in this Thesis can be summarized as follows:

- The performance of parallel applications can be measured at run-time. The requirements to apply the mechanism proposed in this Thesis is to have malleable applications and shared-memory multiprocessor architectures.
- The performance of parallel applications must be considered to decide the processor allocation. The system must use this information to self-adjust its decisions based on the achieved performance. Moreover, the system must impose a target efficiency to ensure the efficient use of processors.
- The different scheduling levels must be coordinated to avoid interferences between levels.
- Malleability is a desired application characteristic that benefits both the application and the system. The application because applications do not have to wait for a certain amount of resources to become available, and to the system because it can better distribute the available resources among applications.

Dedicatoria

A mis padres y a Jose

Agradecimientos

Es viernes, son las siete, y llevo desde las tres mirando ficheros en C. Creo que lo mejor es que lo deje y dedique la poca capacidad que me queda a escribir algo que sea fácil, como por ejemplo agradecerle a todas las personas que han hecho que esta tesis tenga sentido.

Podría empezar por mis directores de tesis como la mayoría de la gente pero creo que el primer lugar se lo merecen mis padres y mi marido. Ellos son mi vida y por lo tanto esta tesis es tan suya como mía. A ellos no se lo agradezco, se la dedico. A mi familia en general, a mis hermanos, a mis suegros, a mis cuñados, en fin, a todos ellos gracias por hacer que todo esto tuviera sentido y apoyarme siempre, espero que sientan que en parte esta tesis es suya también. A algunos amigos muy especiales, a Julio, Mari, y Rubén (su peque), ellos también forman parte de mi familia.

Ahora si que podemos pasar al ámbito más "profesional". Por supuesto se lo quiero agradecer a mis dos directores de tesis, a Jesús y a Xavi. A Jesús más que agradecerle que confiara en mi, que sería lo típico que podría decir, le agradezco que no me haya hecho perder el tiempo, y el poder trabajar con él, ha sido muy constructivo. A Xavi tendría que dedicarle un capítulo especial en mis agardecimientos, porque cuando llegué a las mil horas que me había dedicado dejé de contar.

También, aunque no sea mi jefe, me gustaría agradecerle a Nacho que confiara en mi, ya que es posible que sea gracias a él que yo haya acabado siendo "de sistemas". Me gustaría agradecerle también a Mateo que me echara una mano en algunos momentos, y también algunas palmaditas en la espalda que a veces vienen bien.

Y si repaso por orden cronológico podría nombrar a mis compañeros del proyecto NANOS, a todos en general, Marc, Toni, Eduard, Nacho, Jesus, Xavi, empiezo a repetirme pero es que a algunos tengo mucho que agradecerles. A todos ellos tengo que agradecerles que siempre han estado dispuestos a escuchar mis rollos, a veces incluso voluntariamente :-). A algunos creo que incluso les debo parte del nombre de mi tesis. Si no recuerdo mal Toni me inspiró el nombre de mi primer artículo publicado.

A partir del momento en que me decidí a hacer la tesis también empecé a tratar con otros compañeros que al final se han convertido en una constante diaria: Ernest, Xavi, Jesus (Corbal), Daniel, Ernest, Fernando, Ayose, Oliver, Fran, Xavi (Verdu). Algunos hace poco tiempo que los conozco pero otros han dejado ya su huella. Por ejemplo Jesus, gracias a él se que existe la palabra *hilarante* (y que sus gustos por el cine no coinciden con

los mios). A Daniel le quiero agradecer que haya aumentado mi paciencia hasta niveles que rozan el pasotismo, ahora la vida es más fácil. Ernest, otro "de sistemas", un apoyo a la hora de comer para que "los otros" no nos coman. Xavi es el mismo Xavi de siempre es que no se como me aguanta hasta para comer, creo que queda claro que le debo mucho :-). Todos son buenos amigos.

Algunos compañeros también han contribuido con su granito de arena, Por ejemplo Álex, que me dejó una herramienta de lo mas útil para hacer gráficas, ay, si la hubiera conocido antes. Felix, que también ha puesto su granito de arena con otra útil librería. Otros que ya no están, no es que les haya pasado nada es que se han ido a la privada,como Albert, a él le debo un trocito de mi tesis, sus *dituls* son muy útiles, si algún dia lee esto que lo sepa. También a otros que han desarrollado algunas herramientas que sin ellas hubiera sido una tarea mucho más complicada evaluar la tesis, como a Nacho, gracias a su scpus he podido sacar unas trazas muy útiles. Y como no a Jordi Caubet, que ha tenido una paciencia infinita conmigo.

A mi amigo Pepe, que cuando estuvo aquí haciendo tesis fue el mejor amigo, cuando se fue a Murcia fue el mejor amigo y cuando ha vuelto sigue siendo el mejor amigo,pero ahora tiene más secretos porque es un chico Intel :-). En fin, gracias Pepe.

También por supuesto a gente de esa que está en la sombra, que parece que no tienen nada que ver, pero que si no fuera por ellos sería más díficil. En general a todos los de sistemas, en especial a Victor, que son mucho años :-), y a los del CEPBA, que han tenido mails mios a diario durante unos cuantos años.

This work has been supported by the Spanish Ministry of Education under grant CYCIT TIC98-0511, TIC2001-0995, the ESPRIT Project NANOS (21907) and the Direcció General de Recerca of the Generalitat de Catalunya under grant 1999FI 00554 UPC APTIND. The research described in this work has been developed using the resources of the European Center for Parallelism of Barcelona (CEPBA).

Index

CHAPTER 1

Introduction

Abstract

This Thesis focuses on the efficient execution of workloads of parallel applications in multiprogrammed multiprocessor environments. In particular, we will defend two main ideas: the first one is that all the scheduling levels must be coordinated to achieve a good system performance. The second one is that the processor scheduling must consider real performance information to decide the processor allocation, and to impose a target efficiency to running applications to ensure the efficient use of resources.

In this Chapter, we introduce the main subjects of this Thesis, its motivation, and our contributions.

1.1 Introduction

Multiprocessor architectures appeared as the natural extension to uniprocessor systems. The common characteristics among all the multiprocessor systems is that they have multiple processors that may be used to execute multiple applications at the same time, one application in multiple processors (parallel application), or combinations of the two cases (multiprogrammed systems that execute parallel applications).

The first approach to schedule concurrent applications on these systems was to directly apply uniprocessor policies extended to the case of several processors. However, users, system administrators, and researchers, quickly observed that uniprocessor policies do not exploit the potential of these systems.

The problem was to consider that a multiprocessor system had the same characteristics and problems that a uniprocessor system. Multiprocessor systems have their own goals, applications, and architectural characteristics, and they must be taken into account to schedule, not only processors, but any physical resource. *Goals*, because multiprocessor systems are oriented to increase the throughput of the system and the speedup of individual applications. *Applications*, because parallel applications have frequent synchronizations. These synchronizations imply that the delay of some of the processes can result in a delay of the complete application. *Architectural characteristics,* because multiprocessors have two elements that in most of the cases determine the application and the system performance: the memory system and the interconnection network. In multiprocessor systems, the different memory and network organizations have a direct effect in how applications must be scheduled.

For these reasons, since multiprocessor systems appeared, the scheduling of applications has been an important research subject in this kind of systems, from the point of view of job scheduling, to the point of view of run-time libraries that schedule parallel loops.

As we have commented, all the components of the system (processors, memory, network, and I/O) influence in the performance of a multiprocessor system. In this Thesis, we will focus in the problem of how to schedule workloads of parallel applications in shared-memory multiprocessor systems, taking into account all the elements of the system, but focusing in the processor scheduling.

1.2 The scheduling problem

The scheduling problem consists of how to assign physical processors to application threads. The scheduling problem can be divided in three levels: job scheduling, processor scheduling, and loop scheduling.

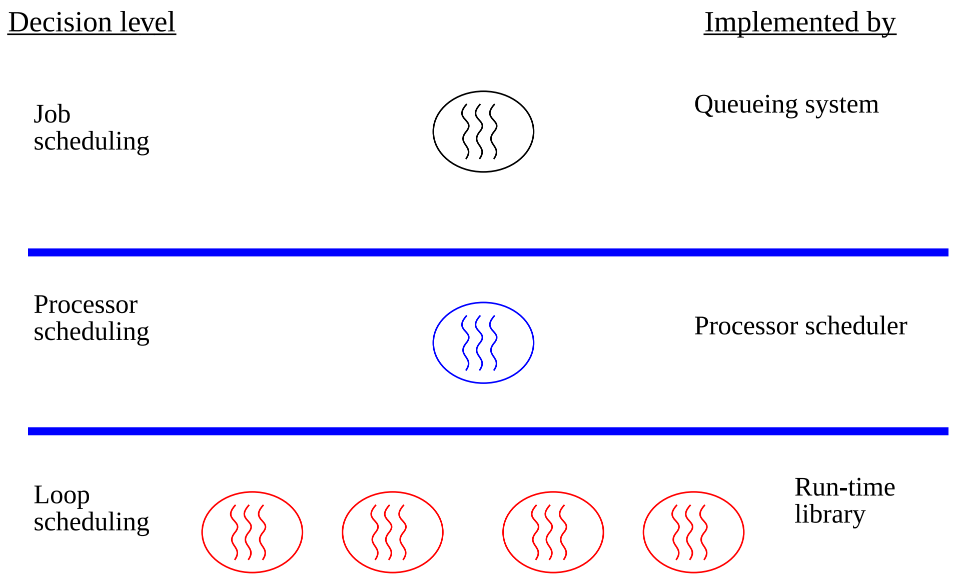

Figure 1.1: Scheduling levels

Figure 1.1 shows the three scheduling levels. The first one, the most external, is the job scheduling. At this level the problem consists of deciding which job should be executed. This level of decision is implemented by the queueing system. These decisions are taken at a low frequency compared with the other levels. The second decision level is the processor scheduling. It decides how many processors to allocate at any moment to each running application. This level of decision is implemented by the processor scheduler. The last one is the loop scheduling problem, and it decides how to distribute the computation among the processors allocated to the application. It is implemented by the run-time library that controls the application parallelism. This very short-term scheduler is normally not considered by the O.S, but it is a responsibility of the application itself.

In commercial systems, these three levels are loosely coordinated. Decisions taken by each level are taken without cooperation with the others levels. This behavior generates situations such as applications running with more kernel threads than available processors, or that there are free processors and queued applications at the same time.

In some previous research works, it was proposed an interface between the processor scheduling level and the loop scheduling level to adjust the number of running threads to the number of physical processors. In some of these proposals, the run-time level informs the processor scheduler about the number of processors requested, and the processor scheduler informs the run-time about the number of processors allocated to the application. With this first level of interaction, the overall performance was significantly improved. These proposals are described in Chapter 2.

1.3 Our Thesis

In this Thesis, we propose to extend other proposals with three main points:

- To achieve the best overall system performance, and the best individual application performance, **all the scheduling decisions must be coordinated**. That means, to provide an interface between the queueing system and the processor scheduler, and an interface between the processor scheduler and the run-time library allowing such coordination.
- **It is necessary to include real performance information** in the processor scheduling decisions. We consider real performance information those values measured at run-time.
- **It is necessary to impose a target efficiency to running applications** to ensure the efficient use of processors.

The first point is the coordination between levels. Coordination means that each scheduling level will provide information about its internal status to levels that communicate with it, and that it will receive information from them. Coordination also means that scheduling decisions taken at each level will consider all this information. Coordination will allow the system to avoid incorrect situations that degrade the system performance such as the ones commented previously, where there can be free processors and queued applications at the same time. Or the inverse situation, where the queueing system can start a new application when the system is heavily loaded.

The second point, the use of real performance information in addition to other user provided information, will allow the processor scheduler to improve its scheduling decisions. Our third point is not only to consider the application performance, but also to impose a target efficiency to be achieved by running applications. The goal of this target efficiency is to ensure the efficient use of resources. Not to consider the application performance could result in an inefficient processor allocation such as to allocate a small number of processors to a parallel application that scales very well and a lot of processors to a parallel application that does not scale at all. This last case even can result in an increment in the execution time of the application.

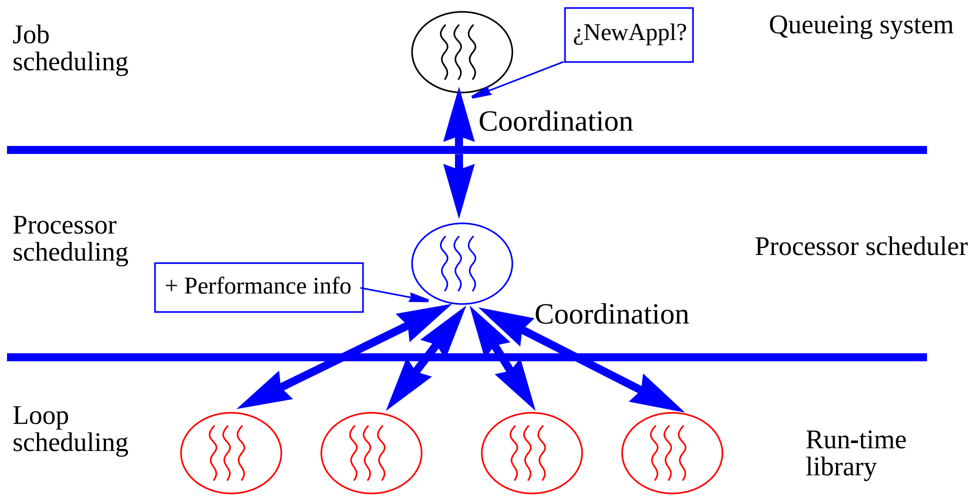

Figure 1.2: Coordinated scheduling

Figure 1.2 shows the main proposals of this Thesis: coordinate the three scheduling levels to improve the system performance, and include performance related information to decide the processor scheduling.

Some works have previously proposed to allocate processors as a function of the application performance. They always assume that application performance is known before the application execution, *a priori*. Details about these proposals are presented in Chapter 5. However, we believe that the system can not rely on users be neither experts nor honest. There are also some other related problems to the use of *a priori* information:

- Sometimes, it is not possible to evaluate parallel applications because their performance depends on input data and the number of combinations makes it impossible to calculate all the possibilities.
- Sometimes, the optimal number of processors for a particular application may not be optimal for the overall system performance, for instance if the load is very high.
- The performance of a high number of applications depends on run-time parameters such as the memory mapping, the number of process migrations, or the influence of the rest of running applications (concurrently executed).

To demonstrate our Thesis, we propose an execution environment with the following characteristics: The performance of parallel applications will be measured at run-time, the processor scheduler will impose a target efficiency to running applications to receive processors, and finally, parallel applications will be malleable to be able to react to the processor scheduling decisions.

1.4 Contributions of this Thesis

To demonstrate our ideas, we present a practical approach based on implementing mechanisms and policies in a real system. The particular contributions that demonstrate the main points of this Thesis are divided into three parts.

In the first part of this Thesis, we propose a complete execution environment that includes a run-time library that performs a dynamic performance analysis, and a new scheduling policy that incorporates the concepts of: use of real performance information, impose a target efficiency, and coordination with the queueing system.

The dynamic performance analysis is implemented by the **SelfAnalyzer**. SelfAnalyzer is a run-time library that measures the speedup of parallel applications at run-time, and also estimates the execution time of parallel applications. SelfAnalyzer exploits the iterative behavior that have a lot of parallel applications, which have a predictable behavior since they repeat the same code several times. The SelfAnalyzer measures the execution time of several iterations with different number of processors and calculates the speedup as the ratio between two of these measurements.

The new coordinated scheduling policy is called **Performance-Driven Processor Allocation Policy (PDPA)**. PDPA takes two decisions: the processor allocation and the multiprogramming level. Regarding the processor allocation, PDPA is a dynamic space-sharing policy that decides a processor allocation based on the performance of running applications, and imposes a target efficiency. With respect to the multiprogramming level, PDPA decides to increment the multiprogramming level when there are free processors and all the running applications have an stable allocation. PDPA will show us the potential and the benefits of a policy that considers application performance and ensures a target efficiency in parallel applications in front of policies that do not consider this point.

In the second part of this Thesis, we present a new methodology to incorporate these three points to any previously proposed processor scheduling policy. The goal of this methodology is to incorporate the concepts of (use real performance information / ensure target efficiency / coordinated scheduler) to other criteria exploited by other policies. With this aim, we present **Performance-Driven Multiprogramming Level** (PDML). We have applied PDML to two space-sharing policies: Equipartition and Equal_efficiency. We have named the resulting policies Equip++ and Equal_eff++. Results will show that after applying PDML, the resulting policies detect and correct situations where applications with bad performance were using a high number of processors due to inefficient allocations decided by the original policies.

In the last part of this Thesis, we incorporate the concepts of (use of real performance information / ensure target efficiency / coordinated scheduler) to a different set of policies, gang scheduling policies. Gang scheduling policies perform time-sharing among applications. Applications are grouped into slots, and at each quantum expiration the scheduler selects a new slot to execute in a round-robin way. Traditionally, these policies apply a simple dispatch, that means that applications receive as many processors as they request. We will show that the ideas proposed in this Thesis are also valid in gang scheduling policies.

We propose two contributions to this kind of policies. In the first one, we propose to apply the PDML methodology to a traditional gang scheduling policy. We call the resulting policy **Performance-Driven Gang Scheduling** (PDGS). PDGS evaluates the performance of active applications every a certain quantum, and adjust their allocation if they do not reach the target efficiency.

The second proposal to improve gang scheduling is a new re-packing algorithm. Repacking algorithms decide how applications are grouped in the different slots. We propose the **Compress&Join** algorithm. The goal of Compress&Join is to reduce the number of slots, which is one of the main sources of overhead of gang scheduling policies. Compress&Join reduces the processor allocation of applications in a proportional way to the application performance. With this reduction in the processor allocation, it is possible to fit the same number of applications in a small number of slots. In both cases, PDGS and Compress&Join, we coordinate the processor scheduler with the queueing system to decide when a new application can be started.

1.5 Overview of the Thesis environment

Since we will demonstrate our ideas using a practical approach, we believe that it is important to briefly describe the characteristics of the Thesis execution environment. Figure 1.3 shows its main elements, and it is fully described in Chapter 3.

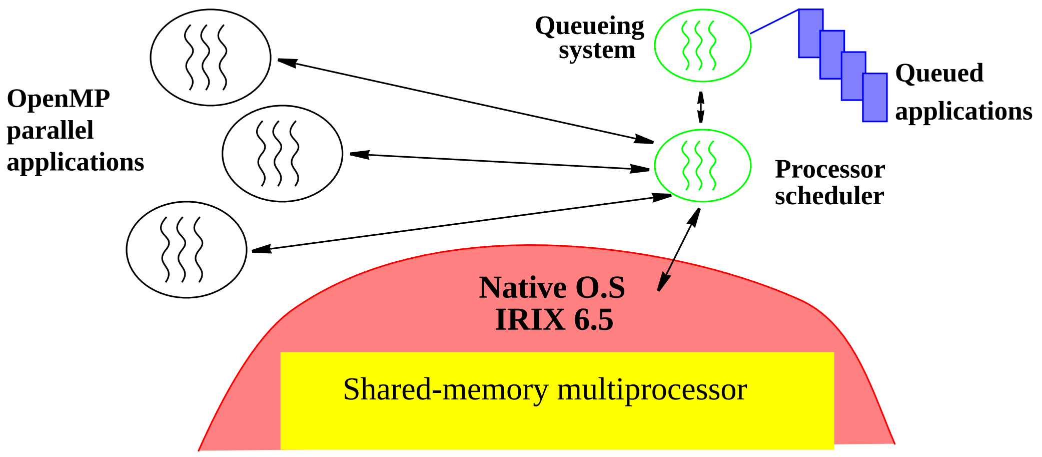

Figure 1.3: Thesis execution environment

This Thesis has been developed in an shared-memory multiprocessor environment. In particular, in a CC-NUMA machine with 64 processors. We have selected this architecture because of its availability, and because it is representative of systems with a medium-high number of processors used in commercial environments and supercomputing centers.

One of the characteristics of our proposed environment is that applications must be malleable to adapt their parallelism to the number of processors available. In this Thesis, we have used the OpenMP programming model because in OpenMP the parallelization does not only depend on the number of physical processors, but also it depends on the algorithm and the loop scheduling policy applied. With the OpenMP model, and the support of the run-time library, applications can be malleable. In this Thesis, we have used the NthLib as run-time library. The NthLib uses the processor scheduler interface to request for processors and to check the number of processors allocated to the application. The NthLib is able to react to changes in the number of processors allocated to the application.

The last elements in the execution environment are the processor scheduler and the queueing system. These two elements have been implemented at user level to implement and evaluate all the proposals presented in this Thesis. The processor scheduler implements the processor scheduling policy and enforces its decisions by means of the native operating system. As we have commented, it provides an interface used by the run-time library. In this Thesis, the processor scheduler also provides another interface used by the queueing system. The queueing system implements the job scheduling policy, that decides when to start the execution of jobs submitted to the system. In our proposed execution environment, the processor scheduler will inform the queueing system about the convenience of starting a new application, and the job scheduling policy (implemented by the queuing system) will decide which particular application to start. In this Thesis, the job scheduling policy implemented is a FIFO, and the job selected is always the first queued job.

1.6 Organization of the Thesis document

This Thesis is organized as follows: Chapter 2 describes the main elements of a multiprocessor system: multiprocessor architectures, scheduling policies, and programming models. We focus on elements that are more related to this Thesis such as CC-NUMA architectures or space-sharing policies.

Chapter 3 presents the particular characteristics of our execution environment. We describe the queueing system, ***Launcher***, the processor scheduler, ***CPUManager***, and the improvements introduced in this Thesis in the run-time library, ***NthLib***.

Chapter 4 presents ***SelfAnalyzer***, a run-time library that dynamically calculates the performance of parallel applications.

Chapter 5 presents ***Performance-Driven Processor Allocation (PDPA)***, a coordinated scheduling policy that decides both the processor allocation and the multiprogramming level.

Chapter 6 describes ***Performance-Driven Multiprogramming Level (PDML)***, a new methodology that transforms previously proposed processor allocation policies to include job performance analysis and coordination with the queueing system.

Chapter 7 presents two new techniques based on the use of job performance analysis and job malleability to improve gang scheduling policies: ***Performance-Driven Gang Scheduling (PDGS)*** and ***Compress&Join*** algorithm.

Finally, Chapter 8 presents the conclusions and the future work of this Thesis.