Universitat Autònoma de Barcelona

Departament d Arquitectura d Ordinadors i Sistemes Operatius

Performance model for hybrid MPI+OpenMP Master/Worker applications

Thesis submitted by Abel Castellanos Carrazana for the degree of Philosophy Doctor by the Universitat Autònoma de Barcelona, under the supervision of Dr. Tomàs Margalef and Dr. Andreu Moreno. Barcelona (Spain),

June 26, 2014

Departament d Arquitectura d Ordinadors i Sistemes Operatius

Thesis submitted by Abel Castellanos Carrazana for the degree of Philosophy Doctor by the Universitat Autònoma de Barcelona, under the supervision of Dr. Tomàs Margalef and Dr. Andreu Moreno.

Performance model for hybrid MPI+OpenMP Master/Worker applications

Author
Abel Castellanos Carrazana

Advisors
Tomàs Margalef Burrull, Andreu Moreno Vendrell

Director signature Director signature Author signature

Acknowledgements

First I want to thanks to my parents,
to suffer the pain of my distance for several years.
To Andreu, Tomàs, Joan, Eduardo, Ania
for your guidance, patience and sympathy.
To Carlos,
one of the best persons I meet.
To Hugo, Ronald and Tomàs Artes,
for offering your friendship and support.
To all my friend and patners at CAOS,
for everything I learn from each one of you,
and last and most important,
to my wife and my son,
for suffer by my absence in the last four year but support me all the time,

Thanks you all!

Abstract

In the current environment, various branches of science are in need of auxiliary high-performance computing to obtain relatively short-term results. This is mainly due to the high volume of information that needs to be processed and the computational cost demanded by these calculations. The benefit to performing this processing using distributed and parallel programming mechanisms is that it achieves shorter waiting times in obtaining the results. To support this, there are basically two widespread programming models: the model of message passing based on the standard libraries MPI and the shared memory model with the use of OpenMP. Hybrid applications are those that combine both models in order to take the specific potential of parallelism of each one in each case. Unfortunately, experience has shown that using this combination of models does not necessarily guarantee an improvement in the behavior of applications. There are several parameters that must be considered to determine the configuration of the application that provides the best execution time. The number of process that must be used,the number of threads on each node, the data distribution among processes and threads, and so on, are parameters that seriously affect the performance of the application. On the one hand, the appropriate value of such parameters depends on the architectural features of the system (communication latency, communication bandwidth, cache memory size and architecture, computing capabilities, etc.), and, on the other hand, on the features of the application.

The main contribution of this thesis is a novel technique for predicting the performance and efficiency of parallel hybrid Master/Worker applications. This technique is known as model-based regression trees into the field of machine learning. The experimental results obtained allow us to be optimistic about the use of this algorithm for predicting both metrics and to select the best application execution parameters.

Keywords

MPI, OpenMP, hybrid applications, Master/worker, performance prediction, Model-based regression tree.

Resumen

En el entorno actual, diversas ramas de las ciencias, tienen la necesidad de auxiliarse de la computación de altas prestaciones para la obtención de resultados a relativamente corto plazo. Ello es debido fundamentalmente, al alto volumen de información que necesita ser procesada y también al costo computacional que demandan dichos cálculos. El beneficio al realizar este procesamiento de manera distribuida y paralela, logra acortar de manera notable los tiempos de espera en la obtención de los resultados. Para soportar ello, existen fundamentalmente dos modelos de programación ampliamente extendidos: el modelo de paso de mensajes a través de librerías basadas en el estándar MPI, y el de memoria compartida con la utilización de OpenMP. Las aplicaciones híbridas son aquellas que combinan ambos modelos con el fin de aprovechar en cada caso, las potencialidades específicas del paralelismo en cada uno. Lamentablemente, la práctica ha demostrado que la utilización de esta combinación de modelos, no garantiza necesariamente una mejoría en el comportamiento de las aplicaciones. Existen varios parámetros que deben ser considerados a determinar la configuración de la aplicación que proporciona el mejor tiempo de ejecución. El número de proceso que se debe utilizar, el número de hilos en cada nodo, la distribución de datos entre procesos e hilos, y así sucesivamente, son parámetros que afectan seriamente el rendimiento de la aplicación. El valor apropiado de tales parámetros depende, por una parte, de las características de arquitectura del sistema (latencia de las comunicaciones, el ancho de banda de comunicación, el tamaño y la distribución de los niveles de memoria cache, la capacidad de cómputo, etc.) y, por otro lado, de la características propias del comportamiento de la aplicación.

La contribución fundamental de esta tesis radica en la utilización de una técnica novedosa para la predicción del rendimiento y la eficiencia de aplicaciones híbridas de tipo Master/Worker. En particular, dentro del mundo del aprendizaje automatizado, este método de predicción es conocido como arboles de regresión basados en modelos análiticos. Los resultados experimentales obtenidos permiten ser optimista en cuanto al uso de este algoritmo para la predicción de ambas métricas o para la selección de la mejor configuración de parámetros de ejecución de la aplicación.

Palabras clave

MPI, OpenMP, aplicaciones híbridas, Master/worker, predicción de rendimiento, Arboles de regresión basados en modelos analíticos.

Resum

A l'entorn actual, diverses branques de la ciència tenen la necessitat d'auxiliar-se en la computació d'altes prestacions per obtenir resultats en un termini relativament curt. Això és degut fonamentalment, a l'alt volum d'informació que necessita ser processada i també al cost computacional que requereixen aquests càlculs. El benefici al realitzar aquest processament de forma distribuïda i paral·lela, aconsegueix escurçar notablement els temps d'espera en la obtenció de resultats. Per suportar això, existeixen fonamentalment dos models de programació àmpliament extesos: el model de pas de missatges mitjançant llibreries basades en l'estàndard MPI, i el de memòria compartida amb l'ús d'OpenMP. Les aplicacions híbrides són aquelles que combinen ambdós models per tal d'aprofitar en cada cas, les potencialitats específiques del paral·lelisme en cadascun d'ells. Lamentablement, la pràctica ha demostrat que l'utilització d'aquesta combinació de models, no garantitza necessàriament una millora en el comportament de les aplicacions. Existeixen diversos paràmetres que han de ser considerats per determinar la configuració de l'aplicació que proporciona el millor temps d'execució. El número de procés que s'ha d'utilitzar, la quantitat de fils a cada node, la distribució de dades entre processos i fils, i així successivament, són paràmetres que afecten sèriament al rendiment de l'aplicació. El valor apropiat d'aquests paràmetres depèn, per una banda, de les característiques de l'arquitectura del sistema (latència de les comunicacions, l'ample de banda de comunicació, el tamany i la distribució dels nivells de memòria cache, la capacitat de còmput, etc.) i, per altra banda, de les característiques pròpies del comportament de l'aplicació.

La contribució fonamental d'aquesta tesi rau en l'ús d'una tècnica novedosa per la predicció del rendiment i l'eficiència d'aplicacions híbrides de tipus Master/Worker. En particular, dins del món de l'aprenentatge automatitzat, aquest mètode de predicció es conegut com a arbres de regressió basats en models analítics. Els resultats experimentals obtinguts permeten èsser obtimista en quant a l'us d'aquest algoritme per la predicció d'ambdues mètriques o per la selecció de la millor configuració de paràmetres d'execució de l'aplicació.

Paraules clau

MPI, OpenMP, aplicacions híbrides, Master/worker, predicció de rendiment, Arbres de regressió basats en models analítics.

Contents

1 Introduction **1**

1.1 Hybrid MPI+OpenMP applications clasification 5

1.2 Performance factors 7

1.3 Motivation 10

1.4 Objectives 11

1.5 Thesis outline 12

2 Background **15**

2.1 Performance Analysis and Tuning 15

2.1.1 TAU 16

2.1.2 Scalasca 17

2.1.3 Paraver 17

2.1.4 Active Harmony 17

2.1.5 Periscope 18

2.1.6 HPCToolkit 19

2.1.7 Elastic 20

2.1.8 ATLAS 21

2.2 Behavior prediction proposals 21

2.2.1 LogP models for communication 21

2.2.2 Analytical models for performance 23

2.2.3 Prediction based on simulation 26

2.2.4 Prediction based on heuristics 28

2.2.5 Machine learning methods 28

3 Analytical Model for Master/Worker hybrid applications 31
3.1 Introduction 31
3.2 General performance model 32
3.3 Communication time prediction 35
3.4 OpenMP overhead 38
3.5 Computation time estimation 39
3.5.1 Sampling measurements using PAPI library 41
3.5.2 Setting up the overlaps 42
3.6 Prediction results 43
3.7 Prediction model limitations 49
3.8 Conclusion 51

4 Model-based regression tree 53
4.1 Introduction 53
4.2 Model-based regression tree 56
4.3 Defining the general expression. 60
4.4 Party package in R. 61
4.4.1 Objective functions 63
4.4.2 Adding weight to training data set 65
4.4.3 Setting the minsplits argument 66
4.4.4 M-fluctuation test 67
4.4.5 Searching for the best application parameter 69
4.5 Methodology to generate the prediction tree 70
4.6 Experiments platform design 73
4.7 Conclusion 74

5 Experimental evaluation 77
5.1 Introduction 77
5.2 Applications 78
5.2.1 Kmeans 78
5.2.2 Nbody 78
5.2.3 Heat transfer 79
5.3 Experimental platforms 80

5.4 Prediction on homogeneous cluster 82
5.4.1 Performance prediction 84
5.4.2 Best performance configuration 86
5.4.3 Efficiency index prediction 92
5.4.4 Best efficiency index configuration 96
5.5 Prediction on NUMA architecture 100
5.5.1 Efficiency index prediction 102
5.5.2 Best efficiency index configuration 107
5.6 Discussions 113

6 Conclusions and Open Lines 115
6.1 Conclusions 115
6.2 Open Lines 117
6.3 List of publications 118

Bibliography 121

List of Tables

3.1 Matrix-mult model constant parameters 46
3.2 Matrix-mult (Last cache & Tlb) regression arguments 47
3.3 Jacobi model constant parameters 48
3.4 Jacobi (Last cache & Tlb) regression arguments 48

4.1 Main parameters for mob_control class in R. 63
4.2 Main parameters for the mob function in R. 64
4.3 Objective functions for MBRT. 64

5.1 Kmeans application variables for training and validation data sets 82
5.2 NBody application variables for training and validation data sets 82
5.3 Heat transfer application variables for training and validation data sets . . 83
5.4 Functions and parameters for MBRT (Performance predition). 83
5.5 Real vs Prediction best configuration (Kmean). 89
5.6 Real vs Prediction best configuration (Nbody). 90
5.7 Real vs Prediction best configuration (Heat transfer). 91
5.8 Functions and parameters for MBRT (Efficiency index prediction). 92
5.9 Real vs Prediction best efficiency index configuration (Kmean). 97
5.10 Real vs Prediction best efficiency index configuration (Nbody). 98
5.11 Real vs Prediction best efficiency index configuration (Heat transfer). . . . 99
5.12 Kmeans application variables for training and validation data sets (NUMA). 100
5.13 NBody application variables for training and validation data sets (NUMA). 101
5.14 Heat transfer application variables for training and validation data sets (NUMA). 101
5.15 Functions and parameters for MBRT (Efficiency index predition) on NUMA. 102
5.16 Real vs Prediction best efficiency index configuration (Kmeans) on NUMA. 110
5.17 Real vs Prediction best efficiency index configuration (Nbody) on NUMA. . 111
5.18 Real vs Prediction best efficiency index configuration (Heat transfer) on NUMA. 112

List of Figures

1.1 Architecture distribution in the TOP500 list. 4
1.2 Hybrid MPI+OpenMP classification hierarchy. 6
1.3 Communication patterns examples for SPMD applications. 9
1.4 Imbalance between threads in hybrid applications with multiple overlapping. 9

3.1 Characterization of blocking communications 37
3.2 Overhead time for PARALLEL OMP FOR scheduling varying the number of iterations per thread. 39
3.3 Matrix multiplication prediction. 45
3.4 Matrix-mult prediction error grouped by number of threads. 46
3.5 Jacobi Model prediction vs real time with workload size of 44000. 48
3.6 Jacobi prediction error grouped by number of threads. 49
3.7 $Ovrlp_{cache}$ and $Ovrlp_{tlb}$ fluctuation behavior in Matrix mult benchmark. . . 50
3.8 $Ovrlp_{cache}$ and $Ovrlp_{tlb}$ fluctuation behavior in Jacobi benchmark. 50

4.1 Prediction result for sin(x) function using model-based regression tree. . . . 55
4.2 An example of the model-based regression tree. 59
4.3 M-fluctuation test example. 68
4.4 Methodology to create the Model-based regression tree. 71
4.5 Basic examples of error distribution. 72
4.6 Experiments platform design. 75

5.1 Surface heat state at first iteration and last for heat transfer. 79
5.2 (a) IBM Node architecture. 81
5.3 Catwoman Node architecture (NUMA). 81
5.4 Relative error distribution for validation data set grouped by workload (Kmean). 84
5.5 Relative error distribution for validation data set grouped by workload (Nbody). 84

5.6 Relative error distribution using validation data set grouped by workload (Heat transfers). 85
5.7 Relative error of efficiency index using the validation data set grouped by workload (Kmeans). 93
5.8 Relative error of efficiency index using the validation data set grouped by workload (Nbody). 94
5.9 Relative error of efficiency index using the validation data set grouped by workload (Heat transfer). 94
5.10 Relative error of efficiency index using the validation data set grouped by workload (Kmeans) on NUMA. 103
5.11 Distribution of internal nodes in the MBRT using Kmeans on NUMA. . . . 104
5.12 Relative error of efficiency index using the validation data set grouped by workload (Nbody) on NUMA. 105
5.13 Distribution of internal nodes in the MBRT using Nbody on NUMA. 105
5.14 Relative error of efficiency index using the validation data set grouped by workload (Heat transfer) on NUMA. 106
5.15 Distribution of internal nodes in the MBRT using Heat transfer on NUMA. 107

List of Code

3.1 Output example using EPCC OpenMP Benchmark. 39
3.2 Code example with function calls to the custom library. 41
4.1 R code example for created the model-based regression tree. 62

Chapter 1

Introduction

Nowadays, the interaction between different fields of study has become a fact for scientists in areas such as physics, biology, mechanics and so on. But all of them have also been affected by the emergence of computational science. These research fields are highly affected by the possibilities to solve difficult problems of different nature that came with the introduction of High Performance Computing (HPC) . As the application becomes more complex and performs more sophisticated computations, the increasing demand for computing resources marks the need for the massive parallelization of systems and applications.

These parallel systems are computer environments formed by a set of processing units that work in conjunction to solve a problem. Just now, one way to increase the power of those systems is to have more cores embeded inside a processor. Although such systems have performance limits, they are more powerful than the ordinary desktop PC that we can find in any home. That is the reason why multicore processors are widely used around the world and are integrated in most computing nodes, from personal computers to supercomputer processing nodes. In this context, every computing node in a parallel/distributed system includes several cores that can be exploited to reduce the execution time of parallel applications. One way of exploiting such features is to distribute application processes to many nodes of the system and execute different numbers of threads at the core level in each node. But even though it is the most popular way to increase the computational demands, there are a variety of architectures of parallel systems. These architectures can be clasified based on the Flynn taxonomy [1] but the result do not clearly express the result of the significant differences of each one. The best way to clasified the actual variety of parallel achitectures must be based on the programming paradigms that are most used in these systems.

The **shared memory architecture** (SMA) refers to a multiprocessing design where several processors access the globally shared memory. They can be divided into two main groups: the symmetric multiprocessor (SMP) also known as uniform memory access (UMA) and the non-uniform memory access (NUMA) [2]. The SMP architecture uses a central bus that connects all the processors with the main memory. This design guarantees that the cost of accessing this shared memory will be equal for all the processors, regardless of the memory address accessed. The contention problems on the bus caused by concurrent access to the bus by the processing units will be the main cause for degradation of the applicationsś scalability. On the contrary, the NUMA architectureś memory distribution is not symmetric. The latency of accessing the main memory will depend on the distance between the processor and the memory slot where the data is accessed. In this case, each proccesor has its own part of the global memory. The global memory will be the sum of all these parts. If one processor tries to access a memory adress located close to another proccesor, it has to be done through the interconnection network between processors. The memory accesses generated in one processor to its associated memory will have a lower latency than those accesses to memory associated with other processors.

The **parallel distributed architectures** feature a system composed of several independent nodes connected through an interconnection network. In these systems, each node has its own memory, which is not shared among all of them. If there is an access to data located in the local memory of another node, there should be explicit communication between those two nodes. These systems are highly scalable, and they allow to teh introduction new architectures on each node without too much effort. For these reasons, they have become one on the most popular architectures in HPC. In general, parallel distributed systems can be split into three groups: clusters, constellations and massively parallel processors (MPPs) [3].

A **cluster** can be understood as a parallel system formed by independent nodes connected through an interconnection dedicated network. Dongarre et al. limit the scope of the definition of a cluster to a parallel computer system comprised of an integrated collection of independent nodes , each of which is a system in its own right capable of independent operation and derived from products developed and marketed for other standalone purposes. Moreover, a commodity cluster is a cluster in which the network(s) as well as the compute nodes are commercial products available to the market for procurement and independent application by organizations (either end users or separate vendors) other than the original equipment manufacturer. A special case of a commodity cluster is the Beowulf-class PC clusters that contains mass-market components for both hardware and software to achieve the best performance without any limited dependence on any single vendor. Beowulf-class clusters and clusters of workstations were at one time distinct system types, but, with the blurring or elimination of any meaningful differences between PCs and workstations in capability, the differentiation between the two types of clusters has also largely lost any meaning. Nowadays, it is quite common for these systems to be formed by multiprocessors (SMP or NUMA).

A **constellation** is a cluster of large SMP nodes scaled such that the number of processors per node is greater than the number of nodes. Using this definition, a constellation becomes a cluster when the number of nodes equals or exceeds the number of cores per node. In this case, the dominant parallelism level is located inside the node. Today, the eight core node is standard, which implies that you need eight nodes (or more) to be called a cluster. Therefore, a modern day cluster should have 64 or more cores. There was a time when 64 processors (single core) was considered a large cluster.

Unlike clusters, massively parallel systems (Massively Parallel Processing, MPPs) are arranged with a very high level of coupling systems. In the MPP systems nodes, are independent (that is, they contain their own memory and copy of the operating system) and are connected via a high speed network, but are managed as a single system similar to a multiprocessor. In general, the performance of these systems is better than the clusters, because some components are often designed specifically for these systems (e.g. the interconnection network).

Nowadays, Cluster and MPPs monopolize the list of most parallel architectures according to the TOP500 [4].

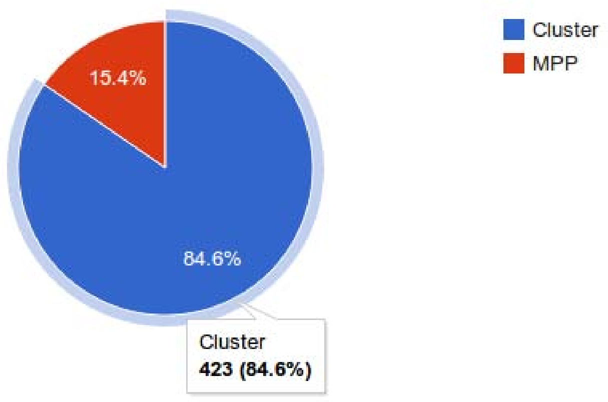

Figure 1.1: Architecture distribution in the TOP500 list.

Figure 1.1 shows a statistic about the most popular HPC architectures used by the most important research center in the world. The percentage values are rounded, which is why there is not another architecture included in the graph. It is clear that there is a high popularity of these two architectures, while the rest tend to disappear.

There are several alternatives of programming models for programming the applications that will be executed in such environments. Each one has advantages and disadvantages in terms of simplicity and ability to leverage the benefits of the underlying hardware and portability. One of the most commonly used programming models is the hybrid approach, with MPI [5] processes communicated processes using message passing and OpenMP threads exploited inside each node [6] and [7]. This combination of parallel programming models is usually called hybrid programming. In the literature, we can find the same term used to reference programming models that combine MPI with other programming model like CUDA [8] for accelerators or StarSs [9] itself, but, so far, the hybrid MPI+OpenMP hybrid combination remains one of the most popular ways to join these two levels of parallelism. The main reason that explains why programmers from different fields of science are still using it now is its inherent simplicity and the rapid growth of the learning curve to assimilate it. A well coding MPI+OpenMP program tries to benefit from the advantages of parallelism at both levels. But, even when we have several applications developed based on these models, there are several differences in the way these two logics combine in the final application. The next sections will cover the different kinds of hybrid MPI+OpenMP applications.

1.1 Hybrid MPI+OpenMP applications clasification

Figure 1.2 shows a general overview for the different kinds of hybrid applications based on the first conceptual proposal by R. Rabenseifner et al. [10]. The conceptual relationship between the different traditional programming models to develop a parallel application can be seen. On the one hand, we have the pure MPI applications, which are those where the implicit parallelism is expressed using a message passing model between all the processes involved in the execution. On the other hand, there are the pure OpenMP applications where the parallelism is implemented inside each computational node. In this case, there is no message passing communication between the threads, because all of them share a common memory space. The responsibility for the creation, destruction and synchronization all these threads is done by the library at the beginning and end of each parallel section that is specified by the programmer.

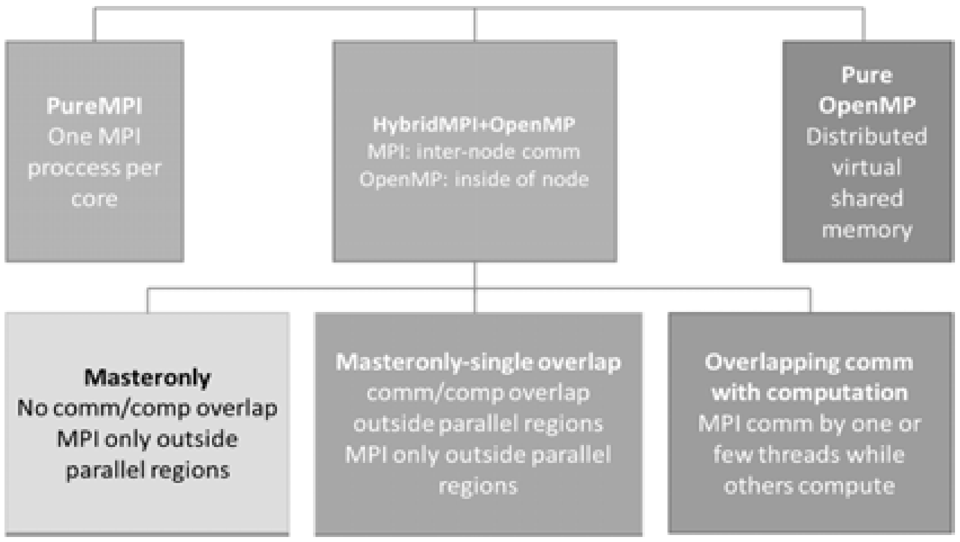
PureMPI
One MPI proccess per core
HybridMPI+OpenMP
MPI: inter-node comm
OpenMP: inside of node
Pure OpenMP
Distributed virtual shared memory
Masteronly
No comm/comp overlap
MPI only outside parallel regions
Masteronly-single overlap
comm/comp overlap outside parallel regions
MPI only outside parallel regions
Overlapping comm with computation
MPI comm by one or few threads while others compute

Figure 1.2: Hybrid MPI+OpenMP classification hierarchy.

An even more detailed description of such applications should consider the location of communications within the logic of our program and whether these communication functions overlap with the computational region of the application. Accordingly, we would have hybrid Masteronly applications, Masteronly-single overlap applications and applications with multiple overlap between computation and communication.

In the first case, the Masteronly applications the communication function calls are made outside the parallel region, so the overlap between regions of computation and communication i only observed between the different processes involved. In contrast, the Masteronly single overlap applications maintain communication function outside the parallel regions, but, in this case, these communication functions are asynchronous, so there will be an overlap between this function and the parallel region of OpenMP code that is between the send call and the wait call. The applications with multiple computation-communication overlap are those where communication function is called inside the OpenMP region. In this case, the overlapcan be seen between different processes and threads.

1.2 Performance factors

Even when a expert programmer in HPC knows the logic of its applications and the details of the architecture where these application will be executed very well, it is very difficult to launch them ensuring that their running times will be close to the minimum or that their efficiency will enrich the maximum value. The applications usually have several parameters that influence their behavior, as well as the fact that the workload of these applications can change over the course of their execution. This problem must necessarily be addressed from an automatic selection of the best parameters of execution. Trying to tackle the challenge consists of being able to develop applications that can enrich a good performance and efficiency and must start to identify the most important factors that influence the behavior of parallel applications. It will allow us to focus all our the effort on improving the application performance by tuning these factors at runtime.

The most important performance factors for parallel hybrid application are the following:

Load balancing: if the workload of each process involved in the execution of the application is quite dissimilar it will cause performance degradation in these applications because processes that finish the jobs early will have to wait for the rest. Thanks to the existence of the two levels of parallelism in the hybrid applications, the imbalance effect will be lower than pure MPI application, but the degradation will be not small enough to ignore the importance of this factor.

Number of MPI processes: Based on the number of MPI processes used in the execution, the general workload will be divided into smaller fragments. In an ideal case, in the same way we increase the number of MPI process, we expect to reduce the processing time on each MPI process.

Number of threads per process MPI: Define the number of threads to be used in the parallel OpenMP region in each MPI process. Similar to the number of processes, an increment in this value will decrease the workload for each thread. If this effect does not occurs, the performance degradation must be explained by the architecture limitation bandwidth produced when several threads are concurrently accessing the main memory. Another cause could be the time penalty generated by data misses on the different levels of cache memory when a thread tries to access this data.

Affinity: Describe the allocation policy for the MPI processes or threads. The MPI processes can be placed on different nodes that do not share a common memory, or they can be placed on the same node. At the same time, the threads can share cache memory level based on a specific allocation or not. Here, we can have a lot of combinations for affinity, but, in the general, the applications can benefit mainly from two types of affinity: the close and far affinity. A close affinity means that the processes or threads are allocated trying to sharethe different levels of cache memory as much as possible, and the far affinity is just the opposite. The applications that have a regular pattern to access the data and frequently reuse this data usually benefit from the close affinity policy.

Communication pattern: In the SPMD applications, we can usually find a regular pattern in the communication between the MPI processes involved in the execution. In figure 1.3, there are a few examples of different patterns in one, two or three dimensions. The number of MPI messages sent and received will increase in proportion to the number of process neighbors for each MPI process. Usually, when the communication pattern becomes more complex, the size of each individual message will decrease. Depending on the characteristics of the application, this increase could benefit or harm the general performance of the whole application. In the Master/Worker and Pipeline parallel applications, it makes no sense to consider this performance factor because the pattern in this case is determined by the number of processes involved.

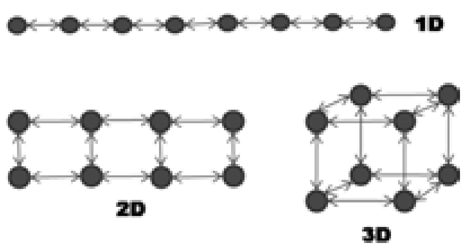

Figure 1.3: Communication patterns examples for SPMD applications.

OpenMP parallelism quality: It is the only factor we identify that cannot be changed at runtime. It describes how much influence there is of the parallelism in these regions on the final performance. There is a chance to improve the source code of this region based on the characteristics of the hardware on which the application is executed. Unfortunately, an improvement in the performance of these parallel regions caused by modifying the code is not a trivial task, as it requires extra effort from the developer to seek the best modification that effectively contributes to this purpose.

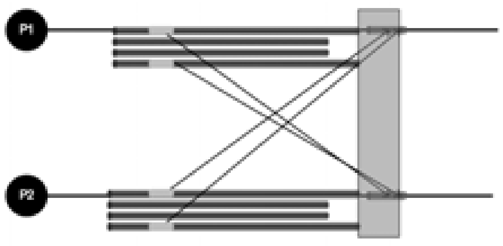

Figure 1.4: Imbalance between threads in hybrid applications with multiple overlapping.

Workload in the communication threads: It is only present in hybrid applications with multiple overlapping factors. For a fine adjustment of this factor, we must decrease the workload of the threads that perform communication and computation relative to those threads that only perform computation. In this manner, we will ensure that all the threads will finish their task at the same time (Fig. 1.4). There is previous research [11] using parallelism implemented with pthreads that shows a dramatic improvement in the performance of parallel applications with the logic adapted to allow for the multiple overlap.

In short, for those hybrid applications on which this thesis is focused (the Master/Worker applications with blocking MPI communications), the performance factors to consider are: number of MPI process, number of threads per process, affinity policy for the threads and processes, application workload and other particular application parameters that will have an impact on the performance.

1.3 Motivation

Nowadays, the steady increase in demand for more computing power in HPC is a reality. This come from the community of scientists that has to deal with complex problems comming from different fields. It is now common to see supercomputers with tens or hundreds of thousands of processors. The parallel applications running on these supercomputers are able to calculate the solution for a problem in a relatively short time but, unfortunately, the performance achieved in several cases is not as good as expected. Even when the execution time of these applications is acceptable, it does not mean that the resources have been used in the right way. Depending on the kind of application, we can achieve a similar performance or efficiency, significantly reducing the number of resources involved in the computation. However, this decision is not fixed on the execution time of those applications. If the workload varies at runtime, the best application parameters have to be dynamically tuned as a consequence of these changes.

That is the reason why, if it is possible to predict the performance and efficiency of applications, we can then automatically search for a best configuration of application parameters that allow us to adjust them at runtime.

1.4 Objectives

The primary goal of this thesis is to ***design and validate a technique for selecting the configuration of application parameters that allow us to reach the best performance or efficiency for a given workload in parallel hybrid MPI+OpenMP Master/Worker applications***. We plan to explain the general methodology that allow us to reach this goal through an incremental improvement process.

In order to accomplish these objectives, there are some intermediate goals that have to be fulfilled in order to guarantee the success of our proposal:

- Define a technique to predict the performance of hybrid applications for any combination of application parameters .
- Evaluate whether global results are precise enough to ensure a proper selection of the configuration parameters of the application.
- Evaluate the techniques for predicting the efficiency index.
- Validate the accuracy of searching for the best parameters of the application that guarantee proximity to the optimal efficiency.

This proposal study is limited to the following conditions:

- The applications selected to validate our work must have a regular computation region that does not depend on the data values.
- The workload of the application should be able to be characterized.
- The application has to be previously well-balanced to guarantee that the processing execution time is similar amoung all workers.
- The communication behavior must be regular. This means that the latency and bandwith of MPI communications should be stable even when the message size or the number of worker changes.

- The validation of our proposal will be bounded to the search space defined by the limits of the training set of observations.

Throughout this thesis, two different approaches are presented. The first attempt to predict the performance is based on defining an analytical model. Thanks to the experience of working on this aproach, we detect the most important limitations of this technique, which led us to change the strategy of prediction to a technique based on regression trees.

1.5 Thesis outline

Chapter 2: Backgrounds. This chapter will cover the most important tools for performance analysis as presented nowadays. Additionally, an exhaustive review of the different techniques for predicting performance will be covered. These techniques are basically joined in three groups: proposals that used simulation, analytical models to predict the performance of both the computation and communication time or even the total execution time.

Chapter 3: Analytic Model for Master/Worker hybrid applications. In this chapter we find a detailed explanation of the performance model we proposed. Tt explains the parts of the model and the methodology for its construction in detail. The chapter also covers the main problem the model has and why it cannot be generalized to predict performance for any workload.

Chapter 4: Model-based regression tree. This chapter will detail the most important contribution of this thesis. Here, the model-based regression tree is introduced as a technique for predicting the performance and efficiency in hybrid application. Details of its parameters and construction will also be cover. In addition, the general methodology is explained to get a better version of the prediction tree based on an iterative and incremental process of analysis of partial results.

Chapter 5: Experimental evaluation. This chapter details the experimental validation of the model-based regression technique, as well as explaining the full description of how the experiments were designed. To validate our proposal, we used three applications in two different architectures.

Chapter 6: Conclusions and Open Lines. Finally, in this chapter, we review the principal conclusion presented in this work and conclude this thesis. This is followed by an outline of open lines and discussion of future works.

Chapter 2

Background

This chapter will cover the most important tools for performance analysis as presented nowadays. Additionally, an exhaustive review of the different techniques for predicting performance or other metrics will be covered. These techniques are basically joined in four groups: proposals that used simulation, analytic models, heuristic and machine learning methods.

2.1 Performance Analysis and Tuning

The develop of efficient parallel hybrid applications is a challenging task that requires a high degree of expertise. Usually, when running parallel application, performance problems appear that limit efficiency, especially as the number of tasks involved increases. To reduce the effect of the performance problems during the execution of the application, it is often necessary for the programmer to carry out a performance improvement process the development of the application. First, a gathering phase is needed to collect the most important behavioral information about the execution. Then, through the analysis of the collected information, performance problems can be detected and possible action to avoid them are determined. Finally, we can change the different application parameters with the goal of resolving the problems and trying to increase performance as much as possible.

In the last decade, different tools and techniques have been proposed to tackle these performance issues and to help developers during the performance improvement phase. The next section will cover some of the most popular tools used by the scientific community.

2.1.1 TAU

The TAU (Tuning and Analysis Utilities) performance system is a integrated toolkit that allows for the instrumentation, offline analysis and visualization of parallel applications [12, 13]. This product can be executed in the majority of the current HPC platforms and can be used to analyze applications written in C, C++, Fortran, Java or Python. It has support to different standard libraries for message passing (e.g MPI or MPICH) and multi-threading (e.g Pthreads or OpenMP). The flexible instrumentation layer provided by TAU allows for the used of different instrumentation techniques, including dynamic instrumentation using the DynInst API [14] or automatically using PDT (Program Database Toolkit) [15]. The instrumentation of the MPI function can be done by using PMPI [16] while the OpenMP directives can be instrumented with Opari [17]. Additionally, this tool allows us to take hardware counters from the processor using different libraries [18].

The parallel profile analysis environment of TAU is composed of sevaral tools. To profile information visualizing, we must use ParaProf. This is a powerful tool that provides several graphics and report styles with the option of filtering part of the information collected. The reports can be generated using the real metrics of using derived metric calculated using this tool. The trace files generated by TAU can be transformed to other formats like SLOG-2 [19], OTF [20] or EPILOG [21]. This is quite helpful because this tool does not force the user to use its own environment; on the contrary, it is posible to see these trace files with an alternative software (e.g JumpShot [22]). Finally, the PerfDMF tool is a laboratory that allow us to manage the results obtained from several experiments. With all this information collected in a database, the user can creat groups by experiment categories or application parameter values and so on. Beyond this advantage, using this tool it is possible to apply correlation studies and clustering techniques over all the data.

2.1.2 Scalasca

Scalasca [23, 24] is a post-mortem performance analysis tool. This program proposed an incremental performance analysis procedure adopting a strategy of successively refined measurement configurations. The more distinctive features that distinguish it from other tools are its ability to detect wait states and the bottleneck in the applications with a very large number of processors and its ability to combine these with summarized local measurements. With this information, it is easier to identify problems, particularly those related to communication and synchronization. The profiling information and the analysis results can be visualized with a tool itegrated into the Scalasca environment. The two tools, Scalasca and TAU have much in common. In fact, it is possible to transform the profiling and trace output files from one to the other quite simply.

2.1.3 Paraver

Paraver is a flexible tool developed in the Barcelona Supercomputing Center [25] for the analysis and visualization of parallel application performance [26]. This tool generates a flexible trace file that allows us to perform different kinds of analysis. This trace can be obtained from the source code of parallel application coded using MPI or OpenMP. In addition, it has an option to show the selected events from the execution in a picture, and it includes a module to make a quantitative analysis of the different metrics or the analysis of several concurrent traces. Additionally, we can get an output text file with detailed information on the application behavior.

2.1.4 Active Harmony

Auto tuning refers to the automated search for values to improve the performance of a target application. In this case, performance is an abstract term used to represent a measurable quantity. A common example of performance for auto tuning is time, where the goal is to minimize execution time. Other possible examples include minimizing power or maximizing floating point operations per second. In general, the Active Harmony framework [27] seeks to minimize performance values and handles maximization via negation.

This project is focused on the dynamic accommodation of the parallel application to the network and resources capacities of the execution environment. This is achieved by automatically testing the different algorithms and tuning actions of application parameters. The architecture they provide is based on a client-server model. Using the collected information from the application previously instrumented, the server carries out the tuning action. The lastest update to this tool is focused on the field of online tuning of automatically generated code [28]. The goal is discover the best configuration of tuning parameters to improve the performance of the application. In an early version of Harmony [29], a greedy algorithm to handle automatic selection of parameters was used. Cristian et al. [30] introduce the use of the Nelder-Mead method for parameter selection that guarantees getting closer to the optimal. This search strategy uses a simplex-based method to estimate the relative slope of a search space without calculating gradients, but, similar to our proposal, this strategy is fed by several execution observations that are collected at runtime.

2.1.5 Periscope

Periscope is a distributed performance analysis tool [31]. under development at Technische Universitaet Muenchen in the ISAR and SILC projects. It consists of a frontend and a hierarchy of communication and analysis agents. Each of the analysis agents, i.e., the nodes of the agent hierarchy, searches autonomously for inefficiencies in a subset of the application processes. The application processes are linked to a monitoring system that provides the Monitoring Request Interface (MRI). The agents attach to the monitor via sockets. The MRI allows the agent to configure the measurements: to start, halt, and resume the execution, and to retrieve the performance data. The monitor currently only supports summary information. The application and the agent network are started through the frontend process. It analyzes the set of processors available, determines the mapping of application and analysis agent processes, and then starts the application and the agent hierarchy. After startup, a command is propagated down to the analysis agents to start the search. The search is performed according to a search strategy selected when the frontend is started. At the end of the local search, the detected performance properties are reported back via the agent hierarchy to the frontend. Periscope starts its analysis from the formal specification of performance properties as C++ classes. The specification determines the condition, the confidence value and the severity of performance properties.

Recently, an extension of Periscope (Periscope Tunning Framework) has been exposed under the AutoTune project [32]. This extension aims to help developers in the process of tuning a parallel application. The framework identifies tuning alternatives based on expert knowledge and evaluates them within the same run of the application. In the end, PTF produces a report on how to improve the code.

2.1.6 HPCToolkit

HPCToolkit [33] is an integrated suite of tools for measurement and analysis of program performance on computers ranging from multicore desktop systems to the nation s largest supercomputers. By using statistical sampling of timers and hardware performance counters, HPCToolkit collects accurate measurements of a program s work, resource consumption, and inefficiency and attributes them to the full calling context in which they occur. HPCToolkit works with multilingual, fully optimized applications that are statically or dynamically linked. Since HPCToolkit uses sampling, measurement has low overhead (1-5%) and scales to large parallel systems. HPCToolkitś presentation tools enable rapid analysis of a program s execution costs, inefficiency, and scaling characteristics, both within and across the nodes of a parallel system. HPCToolkit supports the measurement and analysis of serial codes, threaded codes (e.g. pthreads, OpenMP), MPI, and hybrid (MP-

Ithreads) parallel codes. The analysis performed by HPCToolkit is based on trying to correlate different performance metrics with the logic structure of the application. In addition, it allows for the creation of a description of the application that is independent from the hardware where it was executed. With this information, HPCToolkit can predict the behavior of applications in other architectures [34].

2.1.7 Elastic

Elastic (Large Scale Dynamic Tuning Environment) is a tool that performs dynamic and automatic tuning of pure MPI applications [35]. Its objective is to improve the performance of parallel applications at runtime by adapting them to the variable conditions of the system. Through different stages (instrumentation, information gathering, analysis and tuning), this tool dynamically modified the application by applying given solutions. By using dynamic libraries for instrumentation, the actions of recompiling or restarting the application are not needed. ELASTIC operates following the closed tuning loop of automatic and continuous monitoring and the analysis and tuning of a parallel application without stopping, recompiling or rerunning it. In the monitoring phase, ELASTIC uses event tracing to collect information about the application at the task level. This information is sent to the nodes of the tuning network where automatic performance analysis is conducted. After detecting a performance problem, tuning orders are inserted into the application tasks at runtime with the aim of improving its performance.

The lastest contribution to this product is the viability of using ELASTIC for dynamic tuning in the large-scale computing area. The scalability of this product arises from its architecture, structured as a hierarchical tree of nodes (the tuning network), whose topology can be adapted to accommodate the size of the parallel application.

2.1.8 ATLAS

The ATLAS [36] libraries provide a different approach. In this case, they develop a methodology for the automatic generation of efficient linear algebra routine based on the architecture that will be used. Basically, this methodology is based on generating optimized code to determine the correct blocking reduce the penalty for data cache misses and loop unrolling factors in order to perform an optimized on-chip operation. This is an automatic tuning strategy for shared-memory parallelism, but it is focused on a specific library and cannot be generalized to other applications or libraries.

2.2 Behavior prediction proposals

There are several approaches to tackling the goal of predicting performance, efficiency or other measurements that express the degradation of a parallel behaviour. In general, all the proposals can be grouped into four main groups: the analytical models, prediction support by simulation, the use of a heuristic algoritm and the prediction based on machine learning techniques. In addition, there are some solutions that are formed by combining of some of these techniques [37, 38] and can be used intelligently, considering the constraints of each proposal and the application environment in which they can be applied. From now on, the most important ideas provided in the recent years will be described.

2.2.1 LogP models for communication

The Hockney model [39] is a simple model that assumes that the time to send a message between two nodes with size m has a linear behaviour. It says that:

$$T = a + bm \tag{2.1}$$

where a is commonly known as latency. This is the interval between the start of the communication and when the receiver process catches the first byte of the message. The b argument means the ratio of time per byte sent and which is equivalent to the reciprocal bandwidth of the communication channel. This model does not consider some aspects from the real applications that will cause inaccurate predictions like the congestion of network interconnection or the overlap between different messages sent from many processes.

The LogP model [40] is a trade-off proposal created to be a realistic abstraction of the real behaviour of communication in homogeneous environments. At the same time, it is a simple model, not too sophisticated, that does not asume an infinite bandwith in the interconnection network or a zero time delay. Basically, it is formed by four parameters:

Communication latency (L): Upper limit of the communication delay to send a wordsized message between two nodes assuming that there will be no conflicts on the network.

Overhead (o): It is the time spent by one process doing a communication task (sending or receiving a message) During this time, the process cannot do any computation tasks.

Gap (g): The minimum time interval between the transmission and the reception of two consecutive messages sent by the same process. The inverse of this value determines the effective bandwith.

Number of process (P): Number of processes involved in the execution of the application.

The first three parameters will be measured in time units. They will characterize the performance of point-to-point communications on the parallel system. One important assumption is that the eviroment is considered as a asynchronous machine where there is no synchronization throughout all the processes, so that every task performed by each proccess is independent from the others. In addition, the interconnection network must have a finite bandwith defined by the maximum number of mesages per time unit. This model is only defined in those cases where the size of the messages sent is remarkably small.

The fact that LogP model parameters are independent of the system favors the development and the analysis of portable algorithms. At the same time, it allows us to avoid the need for specific details of the different machines that will be used. The reduced number of parameters presented in this family of models has been crucial its becoming one of the most popular for modeling different situations. Based on its simplicity and accuracy, there are some derived models [41] that cover the identified shortcomings in the original LogP model. One of the most simple updates to this model is the substitution of the network parameters (L,o y g) for functions that depend on the size of the message. The main disadvantage of this proposal is that it increases the complexity of the result model. That is why other extensions like P-logP [42] or LogGP [43] are more popular. The LoPC [44] and LogGPC [45] models are extensions of LogP and LogGP respectively adding the parameter C to characterize the congestion of the interconnection network. The LogGPS [46] model considers the cost of introducing the synchronization libraries of high-level communication to send large messages.

2.2.2 Analytical models for performance

If we look for those proposal that try to model the time consumed for one entire iteration of the application, the analytic model for Master/Worker applications developed by Cesar et al. [47] is a good starting point. This proposal divides the analytic model into two different formulas. There is one approach for those applications with blocking communication and another for those that use non-blocking communication. Additionally, this model considers that the processing time of each worker is proportional to the difference in computation volume of each worker when the number of workers is incremented or decreased. This is a strong restriction that cannot be assumed in the current context. By the time this model was proposed, the multicore architecture did not have the popularity that it now possess. That is why these models do not consider the problem of performance prediction using multicore nodes. Applying this model for predicting performance on multicore environments will significantly increase the prediction error because the model does not take into account the typical problems of memory contention effect and data cache sharing. But, even when this contribution is not applicable in our context without having been adapted, there is a performance index proposal to evaluate the trade-off between efficiency and performance that can be applied with some small adjustments. Next, in section 4.4.5, we will explain each term of the Master/Worker performance index expression in greater depth.

Andreu et al. [48] present an adaptation of a factoring load balancing algorithm of parallel-loops to Master/Worker applications. They developed a completely new set of expressions using the task processing rate as a random variable instead of the task processing time used for parallel-loops. In order to test many different cases, the assessment of the new algorithm has been done through simulations but was also validated through real execution of synthetic programs. The new algorithm has proven to lead to significantly better results than other policies, such as Fixed Size Chunking, Static Scheduling and Factoring with factor two (the most used version of factoring), but the expression proposed is developed for MPI pure applications and does not consider the existence of parallel regions within the MPI process. This proposal can be used to guarantee a dynamic load balancing behavior in hybrid applications, but has to first be adapted to this new reality.

Guevara et al. [49] take advantage of the knowledge of the Master/Worker and Pipeline models and combine them with a resource management policy for obtaining a global model. This contribution can be applied to those complex applications that combine the different patterns mentioned above, but, still similar to the previous models, it was developed to be applied to single processor architecture. So it cannot be applied to hybrid applications without being modified. In general, these three approaches have the same problem in being adopted into the hybrid application environment, but the variability inherent to multicore processing time is not covered in an analytic model, and it is very difficult for a single mathematical expression to be capable of modeling this such complex behavior. We face this reality when we test the first approach presented in this thesis. The results will be covered in future chapters.

Cesar Allande et al. [50] proposed a characterization of the performance of parallel regions to estimate cache misses and execution time. The model exposed is used to select the number of threads and the affinity distribution for each parallel region in OpenMP pure application. The model is applied for SP and MG benchmarks from the NAS Parallel Benchmark Suite using different workloads on two different multicore, multisocket systems. It is evaluated using runtime measurements on a partial execution of the application in order to extract the application characteristics. The authors define the affinity as a vector of the number of threads per socket. A similar idea will be used in our final proposal (Chapter 4) of this thesis in order to define the training information of our prediction technique. The main disadvantage of this idea is that for each execution of the application, some partial iteration need to be sacrificed in order to estimate the best number of threads and affinity. On the other hand, this technique is only accurate for those memory bound applications and is not generalizable to other cases. Our proposal tries to face the problem in more generally by using the concept of affinity as one of the variables for the training information and several historical execution observations as the database to model the general behavior.

Diego Rodríguez propose a refreshing approach in [51]. The main objective of this study is the development of a performance analysis environment that allows us to obtain, highly accurate analytical models of application parallel systems. In particular, the modeling procedure was developed by model selection techniques based on the Akaike information criterion [52], wich is an objective tool to quantify the suitability of a particular model regarding a finite set of models. In order to implement these techniques, there are some libraries that facilitate the generation of analytic model variations from the linear relationship between its components (usually known as multi linear regression models) [53].

The methodology proposed by Diego provides an automatic mechanism generation for analytical models constructed from the performance information that can be obtained after the execution of applications on a particular system. Basically, the generated models are validated to fit the performance of the Whetstone serial benchmark, the collective communication behavior in the MPI library, the performance of NPB-NAS and HLP parallel benchmarks and the result of a task scheduling algorithm in a cluster. The main objection that could be made to using this method is that the results do not show how accurate it is for prediction using configuration not included in the training example. That is why the predictive ability of this proposal is not validated. Based on our experience, even when we have a linear model that fits very well with the behaviour of some outcome variables from parallel applications, it is quite difficult for this model to be generalizable for all prediction spaces. Usually, the arguments of this model will suffer a high variation according to the combination of parameters of the application we use.

2.2.3 Prediction based on simulation

A simulator is a complete environment that emulates the behavior of the different elements that characterize a real system. Therefore, simulation techniques provide a controlled environment of the system for experiments without disturbing the actual system. This technique is particularly suitable for developing new architectures because it allow us to estimate the behavior of applications systems which have not been physically implemented. On the one hand , it is necessary for the system to be sufficiently faithful to the original system for the results to be realistic. However, the simulation environment must be simple enough to avoid a large resource consumption and high cost of development, as well as obtaining relatively small simulation times. In fact, one of the main disadvantages of this approach lies in the high cost of evaluation, which proibits its use in certain situations.

[54] [55] and [56] are contributions that try to predict the effect of cache on the performance of parallel applications. Basically, all of them need a simulation tool of the progression of the cache state over time. The simulator input is the stack distance profile recorded for each thread. With the results provided by the simulator, two techniques for predicting cache misses are proposed. Both are based on a deep correlation study to identify application parameters with higher impacts on performance. The first one uses a piece-wise polynomial regression, and the second uses a neural network. Neither proposal covers the problem of predicting when the profile does not represent all the execution for a particular workload. The third proposal introduces the use of the Markov model to generate an arbitrary length memory access trace with given workload characteristics. This model is generated using the real memory access trace file to configure all the probability distribution functions for each transition on the model. Simulation of a variety of application traces from the SPEC2000 benchmark has been used to demostrate that the synthetic memory reference stream is generally similar to its original form. In all these cases, the research focuses on the prediction behavior for a memory access pattern. Even when these results are accurated, this approach is restricted to predicting the memory missed ratio so it can be extended in order to predict performance or efficiency.

Dimemas [57] is a performance analysis tool for message-passing programs. It enables the user to develop and tune parallel applications on a workstation, while providing an accurate prediction of their performance on the parallel target machine. The Dimemas simulator reconstructs the time behavior of a parallel application on a machine modeled by a set of performance parameters. Thus, performance experiments can be done easily. The supported target architecture classes include networks of workstations, single and clustered SMPs, distributed memory parallel computers, and even heterogeneous systems. The analysis module performs critical path analysis reporting the total CPU usage of different code blocks, as well as their importance to the program execution time. Based on a statistical evaluation of synthetically perturbed traces and architectural parameters, the importance of different performance parameters and the benefits of particular code optimization can be analyzed.

2.2.4 Prediction based on heuristics

In recent studies, the last-level cache miss rate was used as a heuristic to predict whether threads or processes sharing a multicore CPU are suffering performance degradation [58, 59, 60, 61]. In those works, the last level cache (LLC) miss rate was used to decide when the threads should be scheduled on separate chips to avoid cache contention. While suitable for coarse-grained scheduling decisions, the miss rate is not sufficient to estimate performance degradation with greater precision.

Furthermore, relying on a single indicator of performance (the miss-rate) to estimate the effect of sharing multiple resources is a fragile strategy. It may work as long as memory controllers and prefetch bandwidth are key contended resources on multicore systems, but if the hardware bottlenecks change, the heuristic will stop working. Furthermore, this method does not easily allow the integration of other shared resources into the model. Machine learning can adjust to changes in hardware and can be extended to model any new resources that emerge as important for contention.

2.2.5 Machine learning methods

Lee et al. [62] introduced methods of inference and learning for performance modeling of parallel applications. They applied statistical techniques such as clustering, association, and correlation analysis to understand the application parameter space. After that, two techniques (piece-wise polynomial regression and artificial neural networks) were developed for predicting performance. The most important disadvantage of both proposals is the huge amount of work and extensive knowledge required for selecting the subset of application parameters necessary for training the neural network or for generating the piece-wise polynomial regression. Dealing with large data sets of training observation is the major disadvantage to using machine learning techniques. In our proposal, we have also had to accept it. The prediction made in this proposal is limited to the performance prediction with a fixed workload, but varies the application parameter configurations. In adidition, the results presented are limited to performance prediction accuracy. They do not cover other important aspects like efficiency.

A practical method for estimating performance degradation on multicore processors [63] was presented by Tyler et al. In this case they used an early version of the regression tree (REPTree) included in the Weka package [64]. By testing several attribute selection algorithms included in this product, the CfsSubset it is finally selected as the one that achieved the lowest error. To reduce the error rate and to avoid over-fitting, an evaluation of different accuracy-improving techniques is also included. The results are divided into two kind of prediction, the performance fidelity and power efficiency. The results pruve that machine learning can indeed be used to build reasonably accurate models which estimate degradation within 16% of the true value on average; however, inacurate estimetes can occur if the test is very different from the applications in the training set.

Our final approach will be based on the same machine learning technique but using a different regression tree algorithm. Unlike a previous case in which we used this as our method for prediction using any value of workload accordingly, our data set of sampling measurements from short execution of the application is ten times bigger than the data set used in this proposal. Using a version of the model-based regression tree that included an instability test as the primary partition criterion, we reduce the prediction error by using a less sophisticated general regression function that can be used for Master/Worker applications. This general function does not suffer from any major changes when applied to the prediction for the real applications included in this thesis. On the contrary, it allows for the inclusion of candidate variables for splitting, even when they are not included in the general regression model. This is an important improvement to the classic method, because it allows us to used a more general regression function without sicrifiying the prediction accuracy. handIn addition, we extend the study to search for the best parameter configuration to maximize efficiency with the best performance limited to this restriction.

Chapter 3

Analytical Model for Master/Worker hybrid applications

In this chapter, an analytical model approach is proposed in order to predict the performance of Master/Worker hybrid applications. The model is formed of two main parts. The first one predicts the behavior of performance when there is no time penalty caused by access to the main memory. The second part use a regression function to estimate this penalty. The most important disadvantages and problems in prediction will also be covered.

3.1 Introduction

The Master/Worker paradigm is a well-known parallel programming structure because it enables us to express, in a natural way, the complexity inherent to the behavioral characteristics of a wide range of high-level parallel application patterns. In this kind of parallel applications, the Master process is responsible for distributing the data to a set of Worker processes, then each worker makes some kind of computation with the received data and sends the results back to the Master. This logic is repeated for each iteration of the application. Depending on the nature of the problem, the Master process might have to wait for the results from all of the Workers before sending them new data. This behavior forces the complete application execution to be organized in iterations. That is the reason why trying to predict the execution time of one iteration will guide us to predicting the total execution time. Moreover, if the prediction of one iteration is accurate enough, it allows us to search for the best configuration of parameters in order to obtain the best possible performance. In consequence, the total execution time for the application will be reduced in proportion to how reduced it is for one iteration.

3.2 General performance model

Any model designed to predict the performance of hybrid Master/Worker applications has to deal with two kinds of predictions corresponding to the two levels of parallelism involved. The first is the Master/Worker message passing paradigm, which must consider the communication from the master to the workers, the worker processing time and the communication from the last worker to the master. The second one is related to the OpenMP region according to the number of threads used.

If it is assumed that the load is balanced among the Worker processes, the performance of the application mainly depends on two factors: the number of Workers in the system and the number of cores dedicated to each Worker process. Taking into account that the workload might change over time, the model can be applied during the execution of a Master/Worker application to determine dynamically the adequate configuration of the system and/or the application to obtain the best possible performance.

So, the first goal is to develop a performance model that determines the execution time of one iteration of the Master/Worker application. The model can be used to tune these two parameters to improve the performance and the efficiency of the applications at runtime.

One iteration of the Master/Worker applications involves the following steps:

- The Master process makes some processing before distributing the data to the Workers.
- The Master process distributes the data to the Workers.
- All the Workers receive the corresponding data.
- The threads of each Worker compute the result data.
- The Workers send the results back to the Master process.

The new performance model we present here takes all of these issues into account and is based on a previously proposed methodology for developing performance models for hybrid applications [65]. Our proposal extends a previous model [47] considering the complexity of multicore architectures. A general expression to estimate the execution time of one iteration of a Master/Worker hybrid application can be derived as follows:

$$\begin{aligned} T_{iter} &= \mu_m(W) + \lambda_{m-w}(W, Workers) + \\ &\mu_w(W, Workers, Thr) + \lambda_{w-m}(W, Workers) \end{aligned} \tag{3.1}$$

where W is the workload. If the workload is too complex to be characterized with a single value, W must be defined as a vector $W = (w_1, w_2, ..., w_n)$. The $Workers$ parameter is the number of MPI processes acting as workers. Depending on how the applications split the workload among all the workers, it must be expressed as a vector of number of the workers by dimension $Workers = (wk_1, wk_2, ..., wk_n)$. Finally, Thr is the number of threads per worker.

This equation includes the terms representing the steps mentioned above:

- μ_m is defined as the processing time spent by the master on the preparation of a new set of tasks.
- μ_w is the processing time spent by the last worker to finish its task.

- λ_{m-w} is the sum of all communication time from master to every worker.
- λ_{w-m} is the time spent by the last worker to send back the result data to the master.

Additionally, processing time spent by the last worker is expressed in the following equation:

$$\mu_w = \frac{CPI_{ideal} * Inst_{dat} * Dat_{out}}{Frec_{cpu}(Thr) * Workers * Thr} + \tag{3.2}$$

$$\begin{aligned} & Miss_{LL}(W, Workers, Thr) * Lat_{mem} * Ovrlp_{cache} + \\ & Tlb(W, Workers, Thr) * Lat_{tlb} * Ovrlp_{tlb} + \\ & \Theta(W, Workers, Thr) \end{aligned} \tag{3.3}$$

Where

- CPI_{ideal} is the ideal average cycle per instruction of the parallel OpenMP code assuming that the program does not access the main memory.
- $Inst_{dat}$ is the amount of instructions needed to process a single result data.
- Dat_{out} is the volume of global result data as results from the execution of single iteration of all the workers.
- $Frec_{cpu}(Thr)$ is a function that returns cpu frequency in Hertz based on the number of threads that have been used.
- $Miss_{LL}$ is a function that returns the number of last level data cache misses.
- Lat_{mem} Latency of reading a data from main memory.

- $Ovrlp_{cache}$ is the percentage of overlap between instruction spent time and data cache miss time penalty.
- Tlb is a function that returns the number of data translation lookaside buffer misses.
- Lat_{tlb} is the main memory latency when the lookaside buffer generates page fails.
- $Ovrlp_{tlb}$ Overlap percentage between instruction spent time and data translation lookaside buffer misses time penalty.
- Θ is a function that estimates the overhead caused by the OpenMP region in the creation, synchronization and destruction of all the threads.

The ideal CPI_{ideal} is a constant value that is usually hard to calculate. CPU architecture and the kind of instructions used in the parallel region are factors that affect the accuracy of this value. Besides, this value is affected by data cache misses, data dependency, etc. For complex applications, this value can be estimated by taking measurements of cycles and instructions at runtime for small workloads that fit on cache.

There are some constant parameters of the model that need to be set based on the information obtained from a previous characterization of the system. Basically, the constants that are related to the architecture of node will be estimated using different benchmarks. For those terms concerning data cache misses penalty and data translation lookaside buffer misses, a different strategy has been applied. In this case, runtime measurements (hardware counters and execution time) will be taken from some previous iteration of the application as input for applying a regression technique. A thorough description of the measurement phase is presented in the following sections.

3.3 Communication time prediction

In this proposal, we estimate communication time in Master/Worker applications by sacrificing the simplicity of the previous model [47] in order to represent more complex relations between message size and the total communication time. However, this new estimation does not consider the cost of communication per byte as a constant. In fact, there are studies in the literature like [66] that justify the use of benchmarks [67] to reach a more accurate characterization of MPI communication. So, for the evaluation of the communication time, MPIBench [68] has been used.

This benchmark allows us to measure the latency average for each kind of communication defined in the MPI library based on a specific number of measurements for each communication. The main parameters for a valid execution of this program are summarized as:

1. **min_size:** is the start message size for each single communication.
2. **max_size:** is the end message size for each single communication.
3. **iteration:** number of repetitive communications made to calculate the average.
4. **step:** defines the value for increasing the message size between iteration of the benchmark.

The results obtained are the basic information used to evaluate the functions $\lambda_{m-w}(W, Workers)$ and $\lambda_{w-m}(W, Workers)$. For the first expression, the result is the addition of all individual communications from the master to a particular worker λ^{i}_{m-w}. Stated formally, the communication time spent by the Master process is modeled as follows:

$$\lambda_{m-w}(W, Workers) = \sum_{i=0}^{Workers} \lambda^{i}_{m-w}(W, Workers) \tag{3.4}$$

Figure 3.1 shows the latency achieved from point-to-point blocking communication between two process, with varying message sizes. Depending on the architecture used to execute this benchmark, the results will show more clearly that there is not a linear relationship between the latency and the size of the message. In order to achieve enough information to construct these

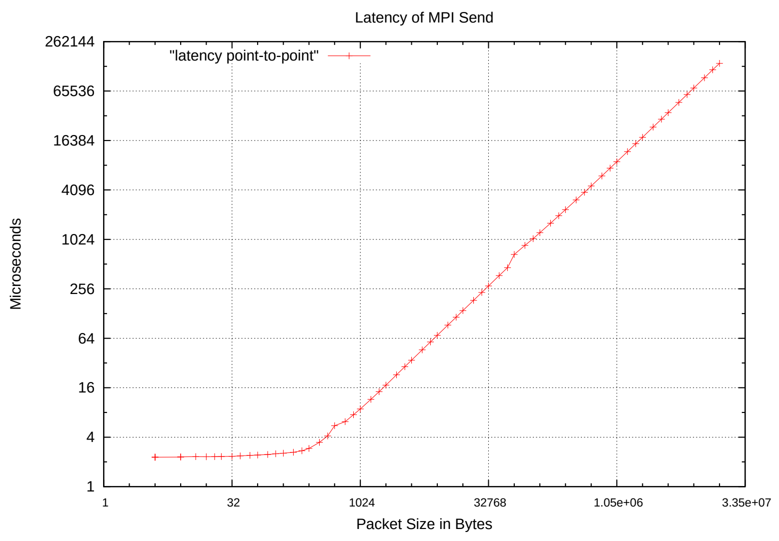

Figure 3.1: Characterization of blocking communications

two functions, the MPIBench needs to be executed for all the types of communication functions provided in the MPI library. This leads us to evaluate the communication behavior for blocking, non-blocking communication and single communication with package message.

This behavior might be affected by the number of communications that are involved at the same time in the execution of the application. Bearing in mind that if the application has to be well-balanced previously, each communication will be carried out in turns because the worker processes must finish their task in the same order the master sent it to each one of them. That is the reason why this approach must give back good prediction results if the variance of the communications is not high.

If applications are planned to be executed on different network architectures or using a different message passing library such as MPICH [69] or even if the libraries are compiled with a different tool, a new characterization has to be made. It will guarantee a specific characterization based on the combination previously mentioned.

3.4 OpenMP overhead

The parallel region implemented using OpenMP pragmas might be negligible for some architectures, but this penalty does not have to be the same for others. Indeed, multicore architectures with a considerable number of cores show significant overhead in the parallel region depending on how complex the logic of the parallel region is. So, a function $\Theta(W, Thr)$ to calculate this time is needed in order to take this overhead into account.

The main causes for OpenMP overhead are described as follows:

- OpenMP Barriers
- Critical sections
- Reduction functions (+,-,*,Max,Min,etc)
- Creation, synchronization, scheduling and destroying threads

The time spent on the creation and destruction of threads depends on the amount of threads. However, the cost of the scheduling, synchronization and the rest of OpenMP pragmas also depends on the number of iterations executed by each thread on the outermost parallel for clause. In order to obtain all these overheads, we use the EPCC OpenMP Microbenchmark [70] to evaluate the overhead for all these pragmas. The benchmark has to be evaluated by varying the number of iterations and threads to get fragmented linear function for each case.

Finally, function Θ internally calculates the number of iterations for a specific number of threads and uses an interpolation to estimate the overhead for every OpenMP clause. The result is the addition of all of these overheads. The following image shows a typical output from this benchmark.

```
Running OpenMP benchmarks on 1 thread(s)
Computing PARALLEL FOR time

Sample_size     Average   Min      Max       S.D.        Outliers
 20             0.9755   0.8500   1.2780    0.196065    0

PARALLEL FOR time =            0.975500 microseconds +/- 0.384287
PARALLEL FOR overhead =        0.206895 microseconds +/- 0.396885
```

Listing 3.1: Output example using EPCC OpenMP Benchmark.

If we use the benchmark to evaluate the overhead for each of the previous pragmas, it is easy to get a prediction for any value of iterations of the parallel region. Figure 3.2 shows the overhead prediction results for different numbers of threads and iterations of the parallel region. As expected, the overhead penalty becomes greater as soon as the number of threads increases.

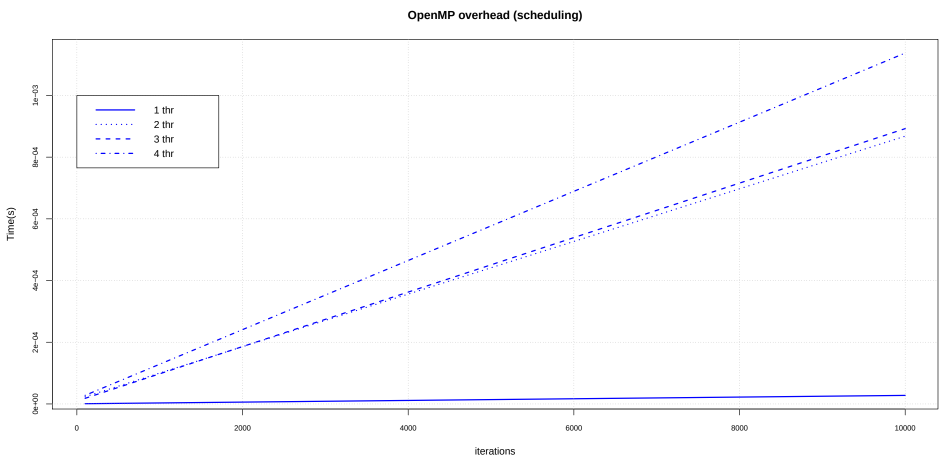

Figure 3.2: Overhead time for PARALLEL OMP FOR scheduling varying the number of iterations per thread.

3.5 Computation time estimation

The computation time of each worker is part of the performance expression where the largest prediction errors of the proposed model are concentrated.

The complexity inherent to current processor architectures makes the prediction harder. Cache size, number of cache lines, data dependency on the code that will be executed, replacement latency and so on, are some examples of the features that have large impact on the model accuracy.

For this reason, the use of some measurements in the initial iterations of the applications is necessary. The number of iterations for taking sampling measurements is not decided arbitrarily, but rather depends mainly on the computational complexity of the parallel OpenMP region. In the case of the microbenchmark of matrix multiplication, the complexity of the parallel region is $O(N^3)$. We propose using at least the first four iterations as a start-up database for applying regression techniques because this is the minimum number allowed to calculated the arguments of the regression function.

This allows us to predict the behavior of data cache misses and data translation lookaside buffer misses varying the number of workers, threads per worker and workload. On each of these four iterations, a high workload imbalance is generated among all workers. Additionally, different number of workers and threads are used on each parallel region for each worker. To generate all these variations in the application configuration, dynamic instrumentation libraries [14] can be used in order to change the behavior of the application. Once all these measurements have been taken, it is not necessary to keep changing the number of workers, threads or imbalance. If the number of iterations for taking measurements is increased, the performance model prediction will be more accurate.

At this point, measurements of hardware counters can be maintained since they do not generate too much overhead (about 1% of the execution time). These measurements generate additional information that would be useful in the prediction process at runtime.

To set up the constant Lat_{mem} and Lat_{tlb} included on the model, lmbench [71] is used. This tool is a microbenchmark that allows us to estimate the different access latency to each memory level. As our model does not consider the effect of data cache misses on the first levels of cache memory hierarchy, only main memory and disk average latency values have been included in the model.

3.5.1 Sampling measurements using PAPI library

As part of the model, information relatives to hardware counter measurements of the parallel region execution is needed. We explained the different tools developed by the scientific community to allow this possibility in the previous chapter. All these tools allow the use of different measurement libraries, but not all of them are supported on all the architectures. One of the most popular libraries with support for several architectures is PAPI [72]. That is the reason why we included it as the central part of our measuring library. In this case, we need to develop a custom library for instrumenting the application code because the PAPI library restricts the use of multiple hardware counters. Furthermore, there are some counters that are mutually exclusive depending on the limitation of the architecture in taking hardware samples.

```
Sample_Init();
#pragma omp parallel num_threads (threads) shared(matrixA,
    matrixB)
{
  unsigned long tID;

  tID=omp_get_thread_num();
  Sample_SetCPU(tID);
  Sample_ON(type_of_meassure,tID);

  #pragma omp for private(i, j, k, result)
  for(i = 0;i < rows_a;i++)
  {
      ...
  }
  Sample_OFF(tID);
}
Sample_End(hw_counters);

Sample_Close();
```

Listing 3.2: Code example with function calls to the custom library.

This problem is mainly caused by hardware limitation that does not allow samples from specific couples of counter to be taken together. For example, on several AMD architectures and/or operating systems, it is not possible to measure last level cache misses and Tlb at the same time. The following code 3.2 shows an example of a code instrumented with our library.

This custom library allows several counters to be measured by sampling them at random in the different iteration of each thread. The functions that start with **Sample_** prefix are basically the code inserted to allow the measuring of the hardware counters. Once all iterations of the threads are finished, the **Sample_End** function calculates each hardware counter value in proportion to the number of OpenMP For-clause iterations in each thread. The final result is returned in the **hw_counters** structure. The number of counters that will be measured simultaneously and the number of them that must be measured alternatively must be defined in the configuration file. If the number of measures is reduced, this sampling technique will be more accurate, but, in the same way, we can not take samples of counters that are mutually exclusive in the same experiment.

3.5.2 Setting up the overlaps

As expressed in the model 3.3, $Miss_{LL}$ and Tlb are two functions for predicting latest level data cache misses and main memory data access lookaside buffer misses. Both functions are built using the same technique. A function for choosing between different kinds of regression is applied using a sample data set as input data. In this function, the outcome variable is the time penalty, and the independent variables are the workload, the number of workers and the number of threads. At this point, the nonlinear regression with the smallest sum of square error is selected as the best candidate for prediction. For example, in the case of a matrix multiplication application, the best regression function selected by its smallest error is given by the expression:

$$\frac{\beta_3 W^3 + \beta_2 W^2 + \beta_1 W + \beta_0}{Workers * Thr} \tag{3.5}$$

In this function, each term of the polynomial on the fraction s numerator has its corresponding coefficient. If the workload does not change drastically, the $Ovrlp_{cache}$ and $Ovrlp_{tlb}$ values must be fixed during the prediction phase. To calculate these constants, real measurements of last level data cache misses(LL), main memory data access lookaside buffer misses(TLB) and execution time for the OpenMP region will be used. Subtracting the predicted time spent only for the instructions that have been executed (3.2) to the time spent in the parallel region, it will give back an estimated time penalty for data access misses (3.6). Using this time as an output variable and real LL and TLB measurements as independent variables, we apply a linear regression technique in order to estimate both parameters (formula 3.7).

$$T_{penalty} = \mu_{realw}(W, Workers, Thr) - \frac{CPI_{ideal} * Inst_{dat} * Dat_{out}}{Frec_{cpu}(Thr) * Workers * Thr} \tag{3.6}$$

The regression expression to estimate the LL and TLB overlap factor must be:

$$T_{penalty} \sim \beta_2 * LL_{real} * Lat_{mem} + \beta_1 * TLB_{real} * Lat_{tlb} + \beta_0 \tag{3.7}$$

Where LL_{real} and TLB_{real} are the real measurements taken in the sampling phase and $\mu_{realw}(W, Workers, Thr)$ in the measure execution time for the parallel region on the worker. The two values of latency are taken from the result of the memory benchmark. At this point, the most important disadvantage of this analytical model appears. The calculated values of overlap do not remain fixed for all the possible combinations of parameters and workload.

3.6 Prediction results

To validate the performance model developed in the previous section, an experimental study has been carried out. Basically, two applications have been used in order to test the model. The architecture where both applications were executed is the cluster IBM. The details of this architecture are covered in depth in chapter 5.

The first application is a matrix block multiplication. It calculates the result of an expression expressed in postfix notation. For example, $A(5,5)B(5,5)*$ represents a multiplication operation between matrices A and B where both have 5 rows per 5 columns. The master process is responsible for creating all matrices with random values to be multiplied, transposing the second and sending the appropriate blocks from the first and the second matrix to each worker. In turn, each worker calculates its matrix block and then sends the resulting matrix block back to the master which updates the global result matrix with all the data received. The master process uses a single core for the transpose operation. The Workers use 4 cores for the matrix multiplication operation.

The second application is a Master/Worker version of the Jacobi method from numerical linear algebra. The Jacobi method is an algorithm for determining the solutions of a system of linear equations. The matrix involved in the parallel calculation is split by the Master process into similar blocks and, the Master then sends each block to each worker consequently. Once each worker finishes its task, it sends back the error that has been computed and the first and last rows of its block. The master process sums up all the errors into the global error and once again sends each worker the first and last row from its neighboring workers.

Figure 3.3 shows the real execution time of the matrix multiplication application and the execution time predicted by the performance model for the case of multiplying matrices of 3500 x 3500 using different numbers of workers (from 2 to 30). For this experiment, in order to have enough sampling information for the regression techniques that must be applied, four previous executions have been used in the sampling phase. These samples have been taken using 2, 8, 20 and 28 workers varying for all of them, the number of threads and the workload from 2250 to 3250. In Figure 3.3, the solid line marked with triangles represents the prediction returned by the performance model varying the number of workers. The dotted line with the plus symbols is the real execution time. As can be seen, the prediction curve does not exactly fit the real one. The errors generated by the regression function for cache misses and tlb misses are the fundamental part of this behavior. Moreover, the most significant point to be considered is that the real and predicted execution times exhibit the same tendency.

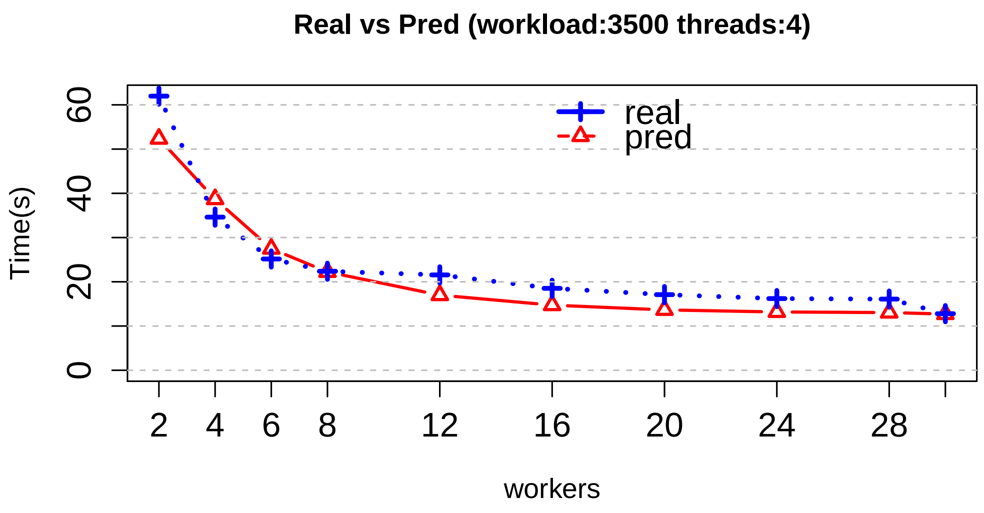

Figure 3.3: Matrix multiplication prediction.

Figure 3.4 shows the error obtained when the performance model is evaluated varying the number of threads and the number of workers. The error is calculated based on the difference between real execution time and predicted execution divided by the first one. The prediction errors are grouped by the number of threads, and the different bar textures represent the result varying the number of workers. The error has been normalized in the range from -1 to 1, where positive values mean an overestimated prediction, and the negative values are just the opposite. For all of the cases, the errors are less than 20%, except in the case with 30 workers with a single thread. Taking into account that the previous workload and the real one have a 16% of variation leads us to conclude that using the prediction value to determine the adequate number of Workers is quite a successful approach.

The performance model is evaluated using the values shown in table 3.1.

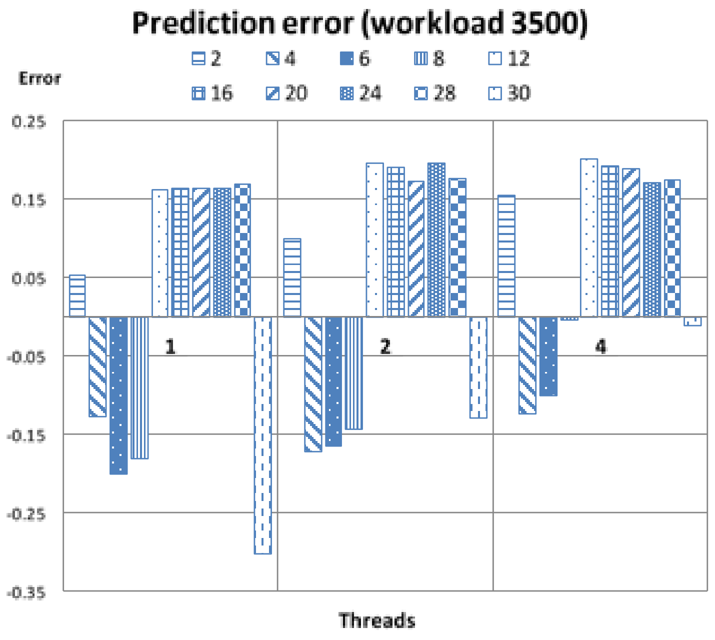

Figure 3.4: Matrix-mult prediction error grouped by number of threads.

These are the main constants used in the model. As previously explained, Lat_{mem} and Lat_{tlb} are obtained by applying a memory access benchmark. $Ovrlp_{cache}$ and $Ovrlp_{tlb}$ are calculated based on measurements results from the previous execution with a workload of 3250. The negative overlap values shown in the table can be understood as the result of the compensation effect produced by the regression technique. CPI_{ideal} is calculated based on previous executions, but, if it is not accurate enough, it will cause negative values in the overlap constants. Increasing the number of previous execution measurements is one strategy to avoid this situation.

Table 3.2 shows the arguments of the best regression function for the pre-

Table 3.1: Matrix-mult model constant parameters

$Ovrlp_{cache}$	$Ovrlp_{tlb}$	Lat_{mem}	Lat_{tlb}	CPI_{ideal}
0.17597	-0.44831	7.5e-08	1.2e-07	0.93

Table 3.2: Matrix-mult (Last cache & Tlb) regression arguments

	Function	β_3	β_2	β_1
OpenMP region 1	$Miss_{LL}$	9.9927e-01	1.7959e+01	-2.9014e+04
	Tlb	1.0596e-01	1.0523e+02	-1.5111e+05

diction of data cache misses and tlb misses. Unlike the previous application, Jacobi has two OpenMP parallel regions. Consequently, the performance model for this application is extended to have two expressions to predict the behavior of each parallel region. The first region includes the main part of the Jacobi method while the second region is just a logic to make a copy of the result matrix used by the worker.

The results obtained from the evaluation of the performance model using the Jacobi Master/Worker application are shown in Figure 3.5. Once again, the dotted line represents the real execution and the solid one is the prediction of the model. This experiment was executed to predict the behavior using a workload of 44000 using two threads and varying the number of workers from 4 to 30. The measurement phase used information relative to iteration with 2, 4, 8, 16 and 24 workers using in each case 32000, 34000, 36000, 40000 and 42000 values for the workload.

In this case, we achieved errors in the results smaller than in the previous Matrix-mult application. The prediction curve fits the real one very well, and the error has also decreased. Figure 3.6 shows the prediction error grouped by number of threads. For 1 thread and 2 threads the errors are even lower than 10%. It is only the prediction for 4 threads that reaches around 17% error. The values of $Ovrlp_{cache}$ and $Ovrlp_{tlb}$ were estimated in the previous iteration, where the workload was 42000. So, the prediction was made using information about execution with a 9% lower workload. In order to predict data cache misses and tlb misses, the function that searches for the best regression function returned only one expression for each case.

The expression $\frac{\beta_2 W^2 + \beta_1 W}{Workers * Thr}$, is the best regression selected for its smallest error for both parallel regions of this application. Obviously, the parameters

Table 3.3: Jacobi model constant parameters

	$Ovrlp_{cache}$	$Ovrlp_{tlb}$	CPI_{ideal}
OpenMP region 1	0.30934	-13.94077	1.13237
OpenMP region 2	-1.117e-07	3.982e+01	0.79694

Table 3.4: Jacobi (Last cache & Tlb) regression arguments

	Function	β_2	β_1
OpenMP region 1	$Miss_{LL}$	179.36e-03	-609.1023
	Tlb	46.03e-04	-68.4915
OpenMP region 2	$Miss_{LL}$	100.21e-03	-0.24
	Tlb	12.245e-04	766.17e-04

of each term of the polynomial are different for each case.

The constant parameters generated by the model for each parallel region as well as the arguments of the regression function for predicting $Miss_{LL}$ and Tlb are summarized in the table 3.3 and 3.4. As the application has two very different parallel regions, it is advisable to calculate each one separately, because the values obtained in each one might be significantly different.

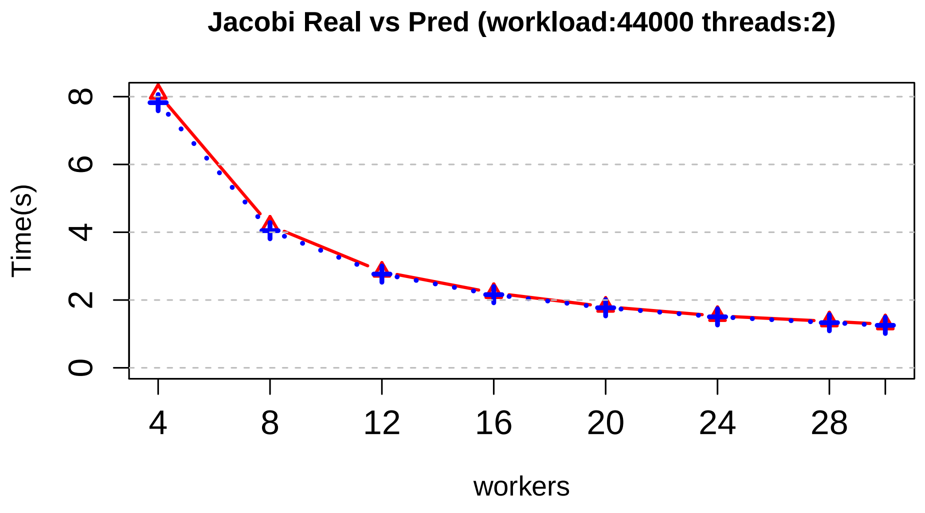

Figure 3.5: Jacobi Model prediction vs real time with workload size of 44000.

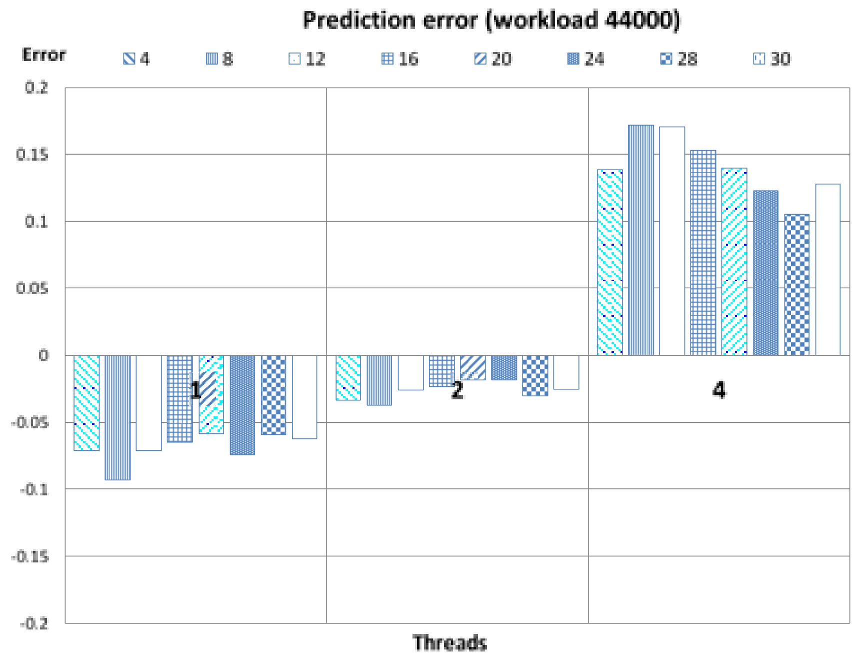

Figure 3.6: Jacobi prediction error grouped by number of threads.

3.7 Prediction model limitations

There is an error involved in the prediction when we used both values of overlap to predict the performance for a new workload. This error will be small if the difference between the previous workload, when overlap values were calculated, and the new one is under 10%. The main cause that explains this behavior is that the values of $Ovrlp_{cache}$ and $Ovrlp_{tlb}$ are suffering from a very high level of variation when the workload changes drastically from one iteration to the next. Even if the $Miss_{LL}$ and TLB functions achieved a good accuracy in the prediction, the influence of the overlap variables in the final result is significant. Figures 3.7 and 3.8 show the values of these two variables for the Matrix multiplication and NBody applications. These values were calculated using the real measurements of execution varying the number of workers, threads and workload. Both graphs clearly shows that there is no regularity in the distribution, but the huge range of variance of both values is even more important. In the Matrix multiplication benchmark, the range

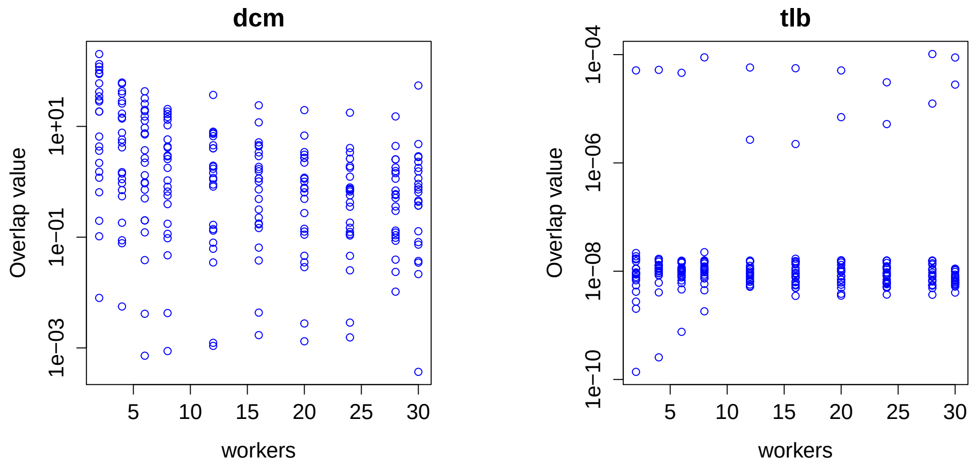

Figure 3.7: $Ovrlp_{cache}$ and $Ovrlp_{tlb}$ fluctuation behavior in Matrix mult benchmark.

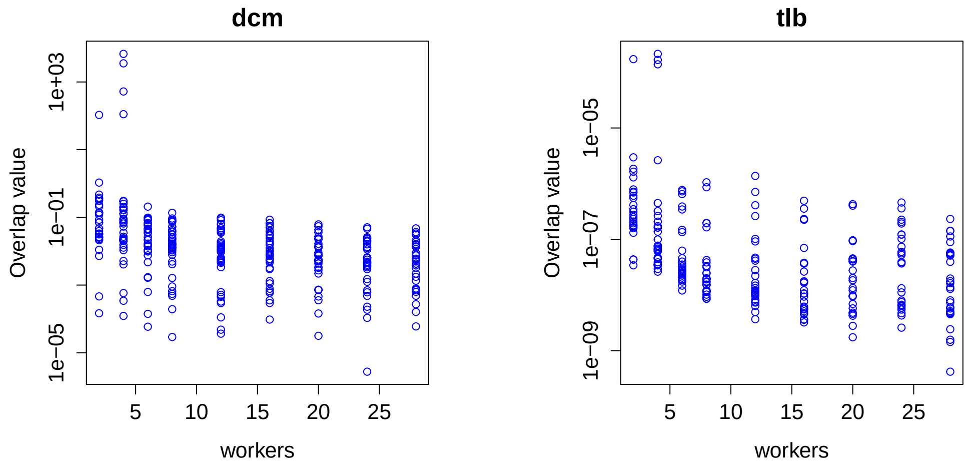

Figure 3.8: $Ovrlp_{cache}$ and $Ovrlp_{tlb}$ fluctuation behavior in Jacobi benchmark.

of $Ovrlp_{cache}$ goes from $1e-10^3$ to more than $1e-10$. In the case of Jacobi, the range is from $1e-10^5$ to more than $1e+10^3$. A similar behavior can be seen for $Ovrlp_{tlb}$ variable distribution in each application. This problem is clearly the most important weakness of the performance model presented in this chapter, because trying to model this behavior becomes an intractable problem even though both values are calculated repeatedly at each iteration in order to reduce error.

3.8 Conclusion

In this chapter, a performance model has been presented. The model is composed of an analytical part and two regression functions for predicting the latest level cache misses and TLB misses. To validate the prediction results, two applications have been used. The expression can be used to predict the performance for small changes in the workload between iterations, but it is not generalizable to any prediction basically because the prediction regression functions to estimated last level cache misses and tlb are calculated with a small sample of observations.

However, the prediction curve for any workload in both applications is remarkably similar to the result that can be fitted with general polynomial with equal grade to the complexity of the application. Even when we reduce the prediction error for the last level data cache misses and TLB misses using a multi regression model, the main focus of prediction error is located in the prediction of the overlap variables. In general, this behavior cannot be modeled with a single regression function because the arguments of this function do not remain fixed if we try to achieve a good accuracy. This led us to propose a different approach, where splitting the prediction into different regions and applying regression to each part is the central part of a new algorithm.

Chapter 4

Model-based regression tree

This chapter will cover a semi-analytic technique for application performance prediction. It is known as the model-based regression tree. Basically, this algorithm is a supervised learning method based on recursively partitioning the training space of observation and generating a binary tree with different regression model on the leaf. A particular version of this algorithm is also covered as well as an explanation of the methodology to create a well-accurate prediction tree. Finally, there is a review of how the experiments platform is designed.

4.1 Introduction

Regression models predict the value of a dependent numeric variable from the values of independent variables, also referred to as predictors (in statistics, predictors are also referred to as regressors). The regression task is the problem of inducing or learning a regression model from a table of measured values of the dependent and independent variables. The simplest approach to the regression task is linear regression, where the dependent variable is modeled as a linear combination of the predictors. But, using regression to model the behavior of performance of parallel application guided us to the the problem of variability in the arguments when a single prediction function were used. The performance or affinity behavior can be understood as couple of functions with several application parameters that will not have, necessarily, the same trend for different application variables (parameters). An interesting strategy could start by using several regression functions to model different regions of the behavior. This will guide us to good results if the performance of efficiency can be modeled based on the same function prototype with different parameters for each prediction region. At the same time, this approach introduces two additional problems: how many regression functions we need to fit the whole prediction space and where are the boundaries of those functions for predicting their respective region. If it is possible to use automatic method that split the prediction space based on some criteria and then calculates for each resulting regions the best regression function we have the chance to accomplish a good challenge. These regression function will have different arguments for each variable involved, even when they match an unique prototype of regression function. On the other hand, the resulting set of regression functions must guarantee that the prediction error will be accurate enough to be be use, later, for prediction using new values of application parameters.

In the last decade, the incorporation of (simple) parametric models into trees has been receiving increased interest. Local regression trees are based on the assumption that using smoother models in the tree leaves can bring about gains in accuracy. Research in this direction was mainly motivated by the fact that constant fits in each node tend to produce large and thus hard to interpret trees. Several algorithms have been suggested both in the statistical and machine learning communities that attach parametric models to terminal nodes or employ linear combinations to obtain splits in inner nodes. These techniques, usually called tree-based regression models or model-based regression trees (MBRT), are known for their simplicity and efficiency in dealing with domains with large numbers of variables and cases. The regression models obtained with this methodology can be represented in the form of a tree. This tree consists of a hierarchy of nodes, starting with a top node known as the root node. Each node of the tree contains logical tests on the variables, with the exception of the bottom nodes of the hierarchy. These latter are usually known as the leaves of the tree. The leaves contain the predictions of the tree-based model. Each path from the root node to a leaf can be seen as a conjunction of logical tests on the variables. These conjunctions are logical representations of subareas of the overall regression surface being approximated.

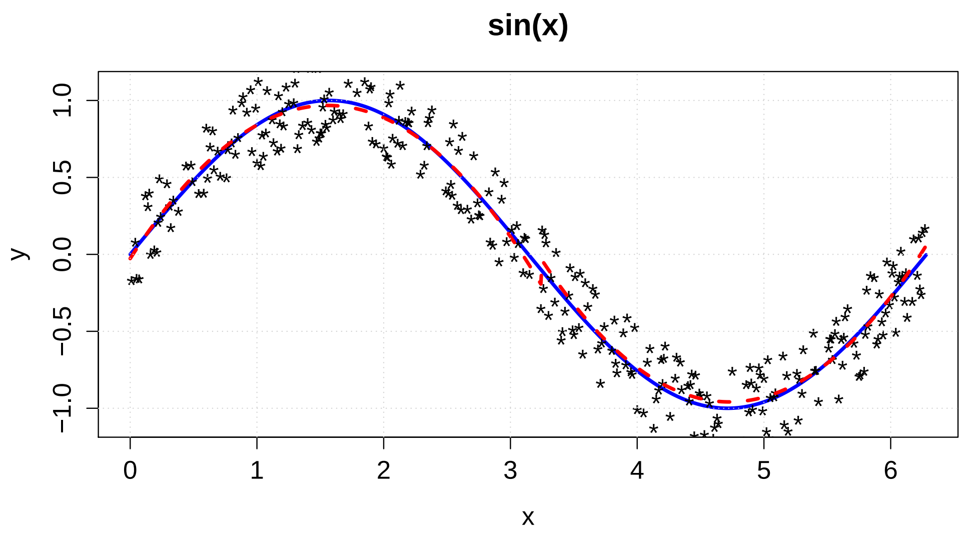

Figure 4.1: Prediction result for sin(x) function using model-based regression tree.

Figure 4.1 shows a simple example where this technique has been used. In this case, the MBRT fit the curve of the sin function, assuming that this behavior is unknown at the starting point. The sin function is plotted in blue and the fitted dotted curve in red is the result from applying the MBRT using the samples as input into the model. The points under and over the curve are the samples generated with a random error. The algorithm divides the observations into two parts trying to minimize the sum of least square error of both regions. For each part, a two-grade polynomial regression is applied. The results also show very well fitted functions with a split point between both predictions at approximately 3.2. Both regression functions have different arguments that was calculated based on the information in each partition. In real problems, the algorithm has to deal with more predictor variables like number of workers, number of threads per worker, process affinity and so on.

4.2 Model-based regression tree

The model-based regression tree (MBRT) approach is a particular variation of regression trees [73]. The main difference from the classical regression trees is basically that the outcome is a tree where each node is associated with a fitted parametric model instead of a tree with fixed real value on the leaf. The goal of applying this technique is to predict the iteration computation time of a Worker for a given workload and a number of workers and threads. The implementation used was developed by Zeileis at al. [74] in a package of partition methods developed for R [75].

The algorithm is fed with a data set of several observations of the application iterations. This data set is formed by measurements of iteration execution time for different values for the number of workers, the number of threads in each worker, workload and other application features that influence the performance. It is advisable to add secondary variables calculated from the primary variables that express the interaction effect between them on the final prediction. This data set is commonly known as a training data set. To validate the accuracy of prediction, a smaller data set is used. This validation data set is composed of observations using workloads that are not included in the training set. Once we have identified the most relevant parameters, the algorithm works based on the following steps:

- Feed a regression model with a data set of training observations.
- Test for instability over a set of candidate splitting variables.
- If there is some overall instability, split the model with respect to the variable associated with the highest instability selecting the value that minimizes the global prediction error.

- Repeat the procedure in each of the child nodes until some stopping conditions are met.

The basic idea of this technique is to recursively split the application variable space into two parts by selecting a specific value of one of the variables that minimizes the sum of the square error. The algorithm calculates the instability p-value for each application variable. The M-fluctuation tests applied were described by Zeileis et al. [76]. This will give back the variable with the highest instability. This value is considered statistically significant if the result of this test returns a p-value lower than 0.05 by default. For example, we have different combinations of the number of workers and the number of threads. First, the algorithm selects the number of threads since, this is the variable that generates the most instability in the performance of the application. Once it is identified, the algorithm divides the training set into two parts (e.g. the first part is lower than or equal to two threads, and the second is the rest) and calculates the regression function that best fits each part. With these two functions already generated, the sum of the square error for each prediction is calculated and added into the final error for this partition. The lowest result will indicate which partition value is the best at minimizing the overall error for this node of the tree.

Before continuing with the explanations of the implementation of this method, some overall concepts have to be defined:

1. **internal node:** will be those nodes where the split action of the training data set of observations occurs.

2. **leaf node:** nodes in the final level of the tree where the regression functions for prediction were calculated and stored for future predictions.

3. **candidate splitting variable:** Are the previously identified variables of the application that might have an impact on the performance of the application. These variables can be application parameters or derived variables from a formula that involves the basic application parameters.

4. **splitting variable:** are those candidate splitting variables that, once the tree is built, are included as split action on the internal nodes.

5. **training data set:** is the tabular information with observations from the executions of the application used to construct the MBRT. Each observation included communication time, computation time and waiting time for each process, as well as all the values of the candidate splitting variables in each case.

6. **validation data set:** similar to the training data set but only used to validate the accuracy of the MBRT, using observations that were not included in the construction of the tree.

7. **general expression:** Is a general prototype function composed of linear combinations of candidate splitting variables. The arguments for this function will be different in each leaf node of the tree.

At this point, one important restriction for using this method is that the parametric model must be expressed as a linear combination of variables. The algorithm stops the generating child nodes on the tree when one of the following two conditions are not met. The first one is related to the size of the subset from the primary data set. We can define the minimum size for the data set on the leaf of the tree. This restriction is useful because this size has to be large enough for the regression can be applied. Additionally, it is advisable not to use a fairly small value in order to avoid over-fitting. This behavior can be observed when a statistical model describes random error or noise instead of the underlying relationship. The second condition refers to how significant the split action is. When M-fluctuation tests do not return a high significance for any of the variables, the algorithm stops at this node of the tree.

Figure 4.2 shows a simple example of the tree generated using this algorithm. The tree is composed of 11 internal nodes, 12 terminal nodes, three parameters for the regression functions and five candidate splitting variables were used. The plots in the terminal nodes give partial scatter plots for each

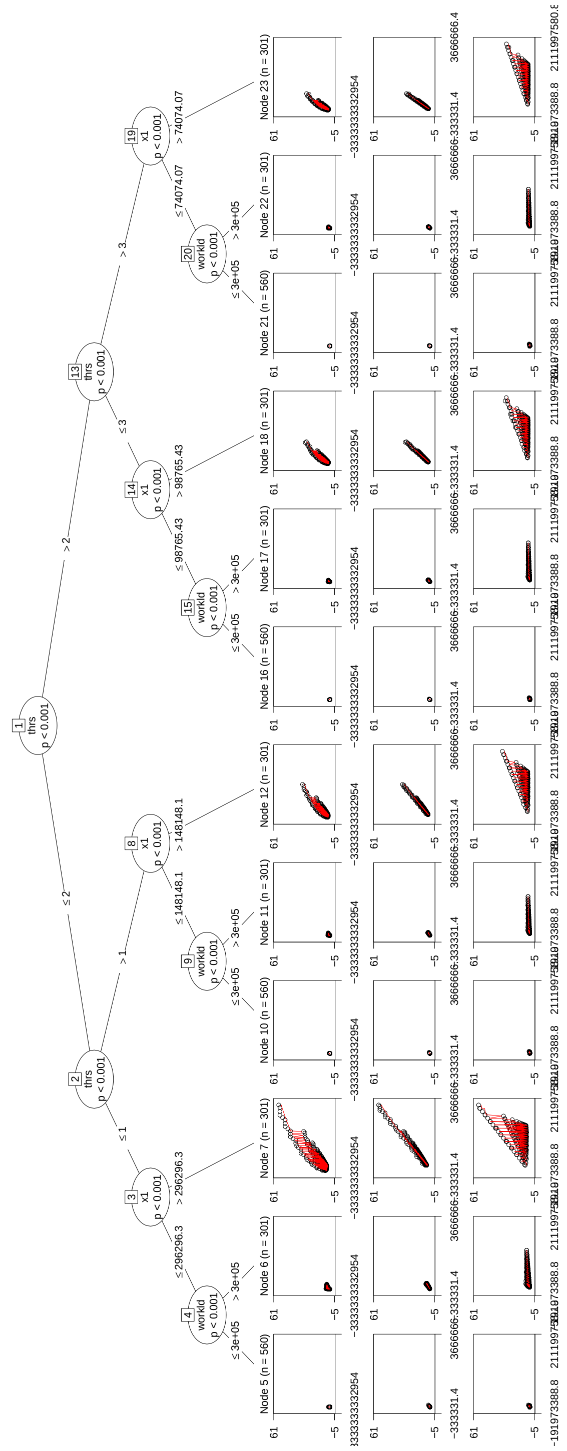

Figure 4.2: An example of the model-based regression tree.

argument in the regression expression. In the first and second level of the tree, the split action is done by the number of threads. This means that, at these levels, the greater instability in regression prediction is generated by the variation of the number of threads. This result suggests that application performance significantly changes if the number of threads increases or decreases.

4.3 Defining the general expression.

There are two important inputs for generating the recursive partitioning tree: the general regression model and the candidate splitting variables. In a first approach, the behavior could be modeled using a general regression function for μ_w in the following way [77] (where β_2, β_1 and β_0 are now from the parameters, and W, $Workers$ and Thr are the variables):

$$\mu_w = \frac{\beta_2 * W^2 + \beta_1 * W + \beta_0}{Workers * Thr} \tag{4.1}$$

This equation can be expressed as:

$$\mu_w = \beta_2 * x_2 + \beta_1 * x_1 + \beta_0 \tag{4.2}$$

Where $x_2 = \frac{W^2}{Workers*Thr}$ and $x_1 = \frac{W}{Workers*Thr}$. In this way, the expression is a linear expression that could be approximated by regression.

In this way, the execution time of one iteration of a Master/Worker hybrid application is estimated as the addition of several terms. Each term involves an error that is propagated to the iteration execution time resulting in a poor iteration time estimation.

To overcome this difficulty, a new approach, is proposed. In this new approach the iteration execution time (T_{iter}) is modeled as a single function that includes all the features of the application and the architecture. This function could be expressed as:

$$T_{iter} = \beta_3 * X_2 + \beta_2 * X_1 + \beta_1 * Workers + \beta_0 \tag{4.3}$$

The communication time spent will depend on the number of processes involved in the execution of the application, but it also depends on the workload. The master process sends $process - 1$ messages to all the worker processes and the workers then respond to the master, but only the last worker is considered because the rest of this communication overlaps. That is the reason why the iteration will again depend on the number of process $Workers$.

However, the values of β_i parameters obtained from regression methods do not represent the complete W, $Workers$ and Thr variable space and the prediction error achieved was considerably high if a single function with fixed β_i parameters was used for the whole variable space. For example, increasing the number of threads in a Worker process implies a cache sharing among these threads, and the number of cache misses can then increase, which can degrade the application performance. So, developing a unique model that takes into account all the platform and applications features is not feasible, and some heuristic must be introduced to create a model that reproduces the observed behavior of the system instead of trying to estimate the effect of each particular feature globally.

For those regular applications where the size of the MPI messages is not constant, the message size is usually proportional to the workload of the application. As the MBRT essentially divides the tree using this variable, in each terminal node, the β_0 parameter will vary according to the size of the message.

4.4 Party package in R.

R is a functional language for statistical computation and graphics [75]. It can be seen as a dialect of the S language (developed at AT&T), for which John Chambers was awarded the 1998 Association for Computing Machinery (ACM) Software award, which mentioned that this language forever altered how people analyze, visualize and manipulate data . R can be quite useful just by using it interactively. Still more advanced uses of the system will lead the user to developing his own functions to systematize repetitive tasks, or even to adding or changing some functionalities of the existing add-on packages, taking advantage of their being open source.

There are several examples of these packages for recursion partition and regression. In particular, The community developers in R provide three of them: the rpart [78], partykit [79] and party package [80]. We used the last one basically because it allows us to build a tree where the outcome is a continuous variable, but, furthermore, it allows us to use different group of variables for regression and for creating a partition in each internal node of the tree. Additionally, this library assembled various high- and low-level tools for building tree-based regression and classification models. It includes conditional inference trees (ctree), conditional inference forests (cforest) and parametric model trees (mob). At the core of the package is ctree, an implementation of conditional inference trees which embeds tree-structured regression models into a well-defined theory of conditional inference procedures. This non-parametric class of regression trees is applicable to all kinds of regression problems, including nominal, ordinal, numeric and censored, as well as multivariate response variables and arbitrary measurement scales of the covariates.

Taking a look at how MBRT methods must be used in the R library the two lines of code can be seen. The first line is only for creating the object with all the information needed to execute the MBRT algorithm.

```r
data$wght <- sapply(data$t, function(x) get_weight(x,2))

ctrl <- mob_control(alpha=0.05,bonferroni=TRUE, minsplit=25,
    objfun=deviance, verbose = FALSE)

model <- mob(t~w2+w1+workers | w2+w1+workers+workld+local_
    workld+thrs,data=data,control=ctrl,model=linearModel,
    weights=data$wght)
```

Listing 4.1: R code example for created the model-based regression tree.

The alpha parameter is related to the significance p-value that must be

Table 4.1: Main parameters for mob_control class in R.

mob_control function	
Parameter	Description
bonferroni	logical. Should p values be Bonferroni corrected?
minsplits	integer. The minimum number of observations (sum of the weights) in a node.
objfun	function. A function for extracting the minimized value of the objective function from a fitted model in a node.
verbose	logical. Should information about the fitting process of mob be printed to the screen?

used to consider a significant instability on each candidate splitting variable. The rest of the arguments for both functions are summarized in Table 4.1.

The second function (mob) builds the MRBT based on the information in the training data. In this case, the data is a data frame that contains all the variables involved in the process. The weight parameter must be a vector of integer values to define the importance of one sample with respect to the others Table 4.2). It is important to highlight that the variables involved in the regression function do not have to be the same as the candidate splitting variables. This gives us the option to split the tree using some variables even if we do not have sufficient information to include them in the regression model. Adding more variables as candidates for splitting the tree does not necessarily produce a more complex tree, but it will increase the computational time of this method to create the final tree. Because our analysis is not performed at runtime of the application, the generation time of the tree can be considered negligible.

4.4.1 Objective functions

As explained in the previous section, minimizing the sum of result values from the objective function is the basic goal of this technique. That is why the objective must be a function that preserves the addition operation ($f(x+y) = f(x)+f(y)$). By default, the mob library in R provides two different objective

Table 4.2: Main parameters for the mob function in R.

mob function	
Parameter	Description
$t \sim x2 + x1 + workers$	General linear regression expression.
$w2 + w1 + workers + workld +$ $local\_workld + thrs$	Candidates splitting variables
data	Tranning data set of observations.
model	LinearModel or gLinearModel.
weight	Is a vector of weight for each sample on the training data set.

Table 4.3: Objective functions for MBRT.

Function	Name
$\sum_{i=1}^{n}(y_i - y_i) * w_i$	residuals
$\sum_{i=1}^{n}(y_i - y_i)^2 * w_i$	deviance
$\sum_{i=1}^{n} \frac{\lvert y_i - \hat{y_i} \rvert}{y_i} * w_i$	relative error

function for lineal regression (deviance and residual). We added a new one (relative error) in order to evaluate the benefits of each one in the final prediction. In the table, y_i is the real execution time, and y_i is the prediction time of the regression model. The w_i is the weight assigned to each training sample. We have discarded, in principle, the residual objective function as a good candidate for constructing the tree. The main reason for doing so is related to the compensation effect produced when negative and positive values are all added together. This will hide the real error in the prediction generated in the branches of the tree.

By default, the regression arguments in each node of the tree are generated to minimize the weighted sum of the square error objective function ($\sum\limits_{i=1}^{n}(y_i - y_i)^2 * w_i$). This is basically the way the multiple linear regression is applied. Unfortunately, the library does not allow for changes to this logic. It may be interesting to evaluate the results of prediction using the relative error as the minimizing function to the regression algorithm. Consequently, if relative error is selected as the objective function, it will be used in the splitting action decision, but not in the regression function internally.

4.4.2 Adding weight to training data set

Optimizing the weighted fitting criterion to find the parameter estimates allows the weights to determine the contribution of each observation to the final parameter estimates in the regression. It is important to note that the weight for each observation is given relative to the weights of the other observations so that different sets of absolute weights can have identical effects. This method also shares the ability to provide different types of easily interpretable statistical intervals for estimation, prediction, calibration and optimization.

The biggest disadvantage of weighted least squares, which many people are not aware of, is the fact that the theory behind this method is based on the assumption that the weights are known exactly. This is almost never the case in real applications, of course, so estimated weights must be used instead. The effect of using estimated weights is difficult to assess, but experience indicates that small variations in the weights due to estimation do not often affect a regression analysis or its interpretation. However, when the weights are estimated from small numbers of observations, the results of an analysis can be very badly and unpredictably affected.

The Optimal results, which minimize the uncertainty in the parameter estimators, are obtained when the weights, w_i, used to estimate the values of the unknown parameters are inversely proportional to the variances at each combination of predictor variable values:

$$w_i \sim \frac{1}{\sigma_i^2} \tag{4.4}$$

Unfortunately, however, these optimal weights, which are based on the true variances of each data point, are never known. Estimated weights have to be used instead. When estimated weights are used, the optimality properties associated with known weights no longer strictly apply. However, if the weights can be estimated with high enough precision, their use can significantly improve the parameter estimates compared to the results that would be obtained if all of the data points were equally weighted. In the literature, there are different algorithms to calculate the weights in linear least square regression [81], but all of them return real values of weight. As the party library does not allow the definition of integer values of weights, we can-not use these algorithms in our work. This is the reason why we whre forced to define a new weight function.

First, we need to define a function to calculate the top weight value for all the observations:

$$Max_{weight} = \max_{1}^{i \leq Obs}(\log_2(y_i * 10^4)) \tag{4.5}$$

and then, the function to assign the weight to each observation must be:

$$w_i = Max_{weight} - \log_2(y_i * 10^4) + 1 \tag{4.6}$$

The main idea of this expression is to try to penalize large values of performance time. Taking into account that experiments with high execution time are a minority within the training set, the differences in prediction of these cases will be larger in magnitude even when they are not in relative error. That is why the weight function is designed to assign a greater weight to the cases with small execution times. First of all, the function normalized all the execution time to start at one. This is the reason why all values are multiplied by 10^4. As the second transformation, the weight is assigned using the logarithmic function with base two. The second expression used the top weight of all the observations to re-assign the weights in reverse order.

4.4.3 Setting the minsplits argument

To avoid over-fitting, this implementation does not have the possibility to apply a final step of pruning the tree. Recursive partition trees usually allow for this possibility. Therefore, the parameter **minsplits** can be used to restrict the number of nodes in the tree to a specific range. A value of 25 means that the smaller size of the training partition in any leaf node has to be equal to or greater than this value. So, previously, the parent internal node of this leaf did not take this restriction into account when dividing the data set.This is clearly a method to control the size and depth of the tree generated and, consequently, to control the over-fitting. But selecting the best value is not trivial, because,as the **minsplits** argument increases, the prediction accuracy will also increase. This is a typical example of a trade-off between good accuracy with generalization in prediction vs. over-fitting. The best way to deal with this kind of phenomenon is to do an iterative study of prediction error results varying the value **minsplits** argument.

4.4.4 M-fluctuation test

Structural change is of central interest in many fields of research and data analysis: to learn if, when and how the structure of the mechanism underlying a set of observations changes. In parametric models, structural change is typically described by parameter instability. If this instability is ignored, parameter estimates are generally not meaningful, inference is severely biased and predictions lose accuracy. Therefore, a lot of literature on tests for structural change or parameter instability emerged, in particular in the econometrics community, using the tests as tools for diagnostic checking against misspecification [82]. But, more generally, such tests can also be used as explorative tools that can help to understand the structure in the data. The generalized fluctuation tests fit a parametric model to the data via ordinary least squares (OLS) or equivalently via maximum likelihood (ML), using a normal approximation and deriving a process which captures the fluctuation of the recursive or OLS residuals [83] or the recursive or rolling-moving estimates and rejects them if this fluctuation is improbably large.

The resulting class of tests for parameter instability, which are based on M-estimation scores, contains many of the tests mentioned above as special cases and unifies the approaches with the construction of test statistics. If we assume n possible vector independent observations

$$y_i \sim F(\theta_i) \qquad (i = 1, ..., n) \tag{4.7}$$

distributed according to some distribution F with k-dimensional parameters θ_i. We also assume that the index $i = 1, ..., n$ stems from the ordering with respect to some external variable. The method is interested in testing the null hypothesis

$$H_0 : \theta_i = \theta_0 \qquad (i = 1, ..., n) \tag{4.8}$$

against the alternative that at least one component of θ_i varies over some variable. It is based on functional central limit theorems for these fluctuation processes; both under the hypothesis and local alternatives it is shown how structural changes in parametric models, with a special emphasis on regression models, can be discovered by test statistics that capture the fluctuation in the M-score processes. The statistical p-value will describe how unstable the response variable is with respect to the predictor variable in each case. The lower p-value indicates what the predictor variable associated with the highest parameter instability is.

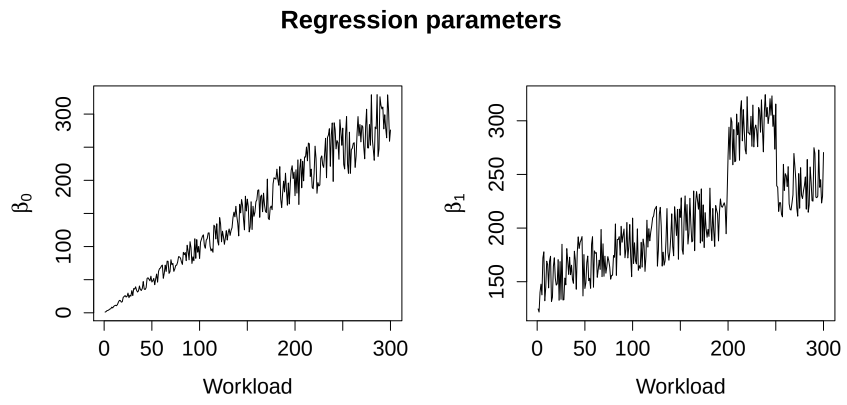

Figure 4.3: M-fluctuation test example.

Figure 4.3 shows a hypothetical example of the power of this test. Let us say we have a lineal regression with several variables, and the values of the arguments for two of them are plotted. The M-fluctuation test has the ability to detect the variable predictor β_1 as the one with more statistical significance for generating instability. The variable β_0 has a similar variability, but, in this case, there is more stability. The variable β_1 suffers a remarkable variation in its tendency in the region between 200 and 250. This example shows that applying a split action in the tree using β_1 will generate three lineal regressions that must fit the real behavior better.

4.4.5 Searching for the best application parameter

Let say we obtain a very accurate prediction results. The main idea to search for the best execution parameters with a given workload goes to evaluating all the leaf node of the tree that match with the workload. Once we have this subset of predictions, the next step is to find The minimum output value from all the regression functions evaluated using the given workload. The regression selected will give us the final parameters with which the application should be executed. This algorithm can be used for both, the selection of the best performance settings, and the selection for the best parameters configuration of efficiency. This thesis validates both proposals.

In order to predict the efficiency for a particular combination of parameters, the first step is defining the expression for evaluated the efficiency. Our approach is based on an extension of the efficiency expression for Master/-Workers application defined by Cesar [47]. More formally, the efficiency index $E(W, Workers, Thrs)$ is define as $\frac{T_c}{Workers * T_{avail}(W, Workers, Thrs)}$, where T_{avail} is $\sum_{i=0}^{Workers} tavail_i$, and $tavail_i$ is the time worker i has been available for doing useful work, which for those applications we are working with, will be the whole iteration time T_{iter}. Consequently, the efficiency index will be defined as $\frac{T_c}{Workers * T_{iter}(W, Workers, Thrs)}$, and finally, the efficiency index as:

$$E_{index}(W, Workers, Thrs) = \frac{T_{iter}(W, Workers, Thrs)}{E(W, Workers, Thrs)} = \frac{Workers * T_{iter}(W, Workers, Thrs)^2}{T_c} \tag{4.9}$$

where

$$T_c = \sum_{i=0}^{Workers} \mu_w^i(W, Workers, Thrs) \tag{4.10}$$

T_c is the sum of all the computation time spent on each worker, so μ_w^i will be the computation time spent for the worker i. The original definition of efficiency index was performance index but have changed the terminology to avoid being ambiguous. In order to searching for the best efficiency index, this work subjects to applying the first derivate of the expression to find the minimum value that will be the best configuration but in our context, as we have multiple regression functions, the result is not continuous over the parameters space. So, selecting the best configuration must be done using an exhaustive search. The efficiency index is not a classic measure efficiency. It is based on the time relationship between communication and the computation time spent by the parallel application.

4.5 Methodology to generate the prediction tree

Once we present the main concepts that are involved in the algorithm for constructing the MBRT, the following explanation will cover the methodology that allow us to have a well-accurate MBRT. Figure 4.4 shows a diagram with the general overview of the methodology proposed and the process composed of six steps that has to be followed to enrich the goal. In the first step, we have to define the general regression function based on the knowledge previously achieved from the behavior of the application. At this point, the number of parallel regions, the complexity of each one of them and the functions for MPI communication have to be taken into account to define the regression expression.

In the second step, the application parameters that might influence the performance of the application need to be considered as candidate splitting variables. Once these two steps are completed, we create the training and validation data set, varying for each case the workload and the values of application parameters. For all these combinations,the architecture selected for the experimental validation will be executed. At this point, taking measurements of the whole execution is not needed. We only take the first ten iterations, assuming that, if the application is regular, these measurements will be representative of the complete execution.

Once we record all the performance measurements, the next step will be the tree generation using the selected training data set, followed by the step where boxplot for error distribution using training data takes place. At this step, it is important to check for similar error distribution for each splitting variable value.

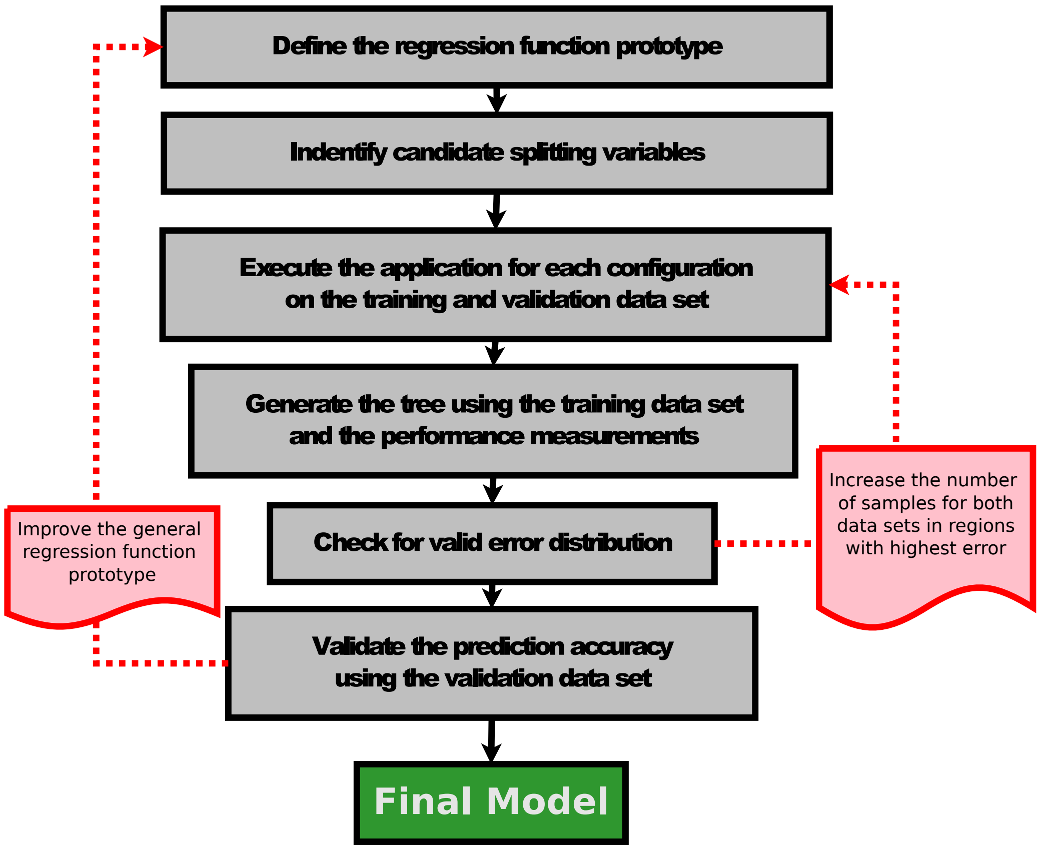
Define the regression function prototype
Indentify candidate splitting variables
Execute the application for each configuration on the training and validation data set
Generate the tree using the training data set and the performance measurements
Check for valid error distribution
Validate the prediction accuracy using the validation data set
Final Model
Improve the general regression function prototype
Increase the number of samples for both data sets in regions with highest error

Figure 4.4: Methodology to create the Model-based regression tree.

Figure 4.5 shows two examples of similar and not similar error distribution for a variable in the range of one to five. These graphs can be understood in the following way. The bottom and top of the box indicate the end of the first and third distribution quartiles respectively, and the line inside the box is always the median value. The lowest datum is still within 1.5 interquartile range (IQR) of the lower quartile, and the highest datum is still within $1.5 * IQR$ of the upper quartile where $IQR = Q3 - Q1$. The small circles represent outlier error predictions, which are those values outside the highest datum. The boxplot with similar results shows slight differences between the error distributions for different values. This is a typical case that we can find in real values. It is very uncommon to achieve exactly the same distribution. As in the graphs, we can consider two distributions to be similar if there is not a big difference in the median and the values of quartiles respectively. In this case, the difference is not bigger than five. In the second case, there are big differences in means and dispersion for the error for each value. This is the kind of results that shows us the probability of making a mistake in the further best application parameters prediction. If some distributions are remarkably higher than others, the probability of not selecting a configuration that returns this error in prediction will also increase.

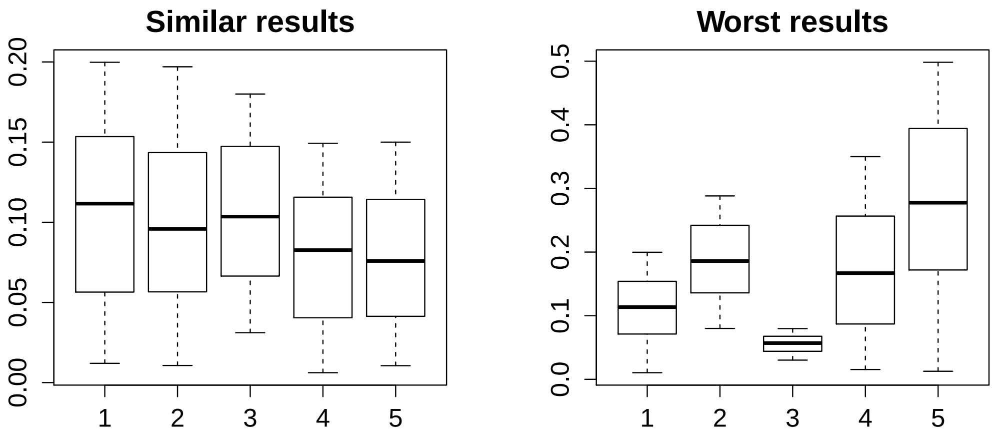

Figure 4.5: Basic examples of error distribution.

In this step of the methodology, we have to check for similar distribution among all the splitting variables. If the results do not guarantee that, we need to go back to the second step in order to increase the training samples by adding more experiments to the region where higher error distribution is identified. This iterative process will prove whether or not the accuracy prediction problem in a region is because the training sample is too small to be representative of all the behavior. In cases of where better results are not achieved when the size of the training data set is increased, we ry to better rethink the general regression function prototype.

The final step is composed of the task of validating the MBRT with the validation data set. Once it is sure that the error distribution between all the splitting variables are similar, the predictive capacity must tested. The most important aspect to keep in mind is that the error for the new values of workload must be, in general, around 5%, and the dispersion must be not higher than 20%. It will be perfect if the outliers prediction is under 5% for all predictions made with the validation sample. If we consider that all these conditions are fulfilled, this is the step to take a look at the prediction table for the best configuration of parameters.

4.6 Experiments platform design

As we explain below, the MBRT technique is fed with several observations of performance behavior, varying in each case the application configuration parameters and the workload. Dealing with this huge amount of information can be tedious and can demand many hours studying the results for the different configurations of the prediction tree. In order to assist in this task being performed more comfortably, we explain in detail how the experiment platform for creating the prediction model is designed.

First, a custom library is created to generate tabular information (.CSV file) related to the measurements that must be taken in each important region on the application. The typical regions that must be instrumented are each OpenMP region in the worker, the MPI send function used in the master and the worker and the Wait and Receive function. Once we have the output from all the executions, a library implemented in R will automatically read all the files in the output folder. The library splits the information into two parts, one for training and the other for validation, and uses the first one to construct the MBRT based on some fixed arguments for this algorithm that are defined in the configuration file. Once the tree is generated, the library saves this predictor tree in a binary file in order to have historical information about the different versions of trees. In a second step, this library generates the bar plot pictures for the error distribution of each splitting variable. To do this task, the library at first checks if each variables was used to split the tree.

At this point, the expert must check if the results are acceptable. If indeed they are, the expert can request the error distribution for the validation data set. At this point, the library reloads the binary serialization of the tree and evaluates the prediction for each case and, finally, generates the boxplot error distribution and the prediction table for the best application configuration parameters. Thanks to the design of this structure for analyzing the data, we save a lot of time in a tedious iterative process. Figure 4.6 shows a general overview of the platform we described.

4.7 Conclusion

The principal objective of this chapter is to present the model-based regression tree techniques as a technique for predicting performance or efficiency in hybrid applications. The MBRT is a machine learning algorithm that allows us to fit a behavior using several regression functions. We used a version of this algorithm developed in R. The main reason for doing this is because this version includes a test for evaluating the variable s instability among all that is included as the candidate splitting variable. In the next chapter we plan to evaluate the MBRT trying for predicting performance and efficiency. If the results are sufficiently accurate, we can additionally use this technique for the selection of the best parameters to run the application in each case.

Additionally, the methodology proposed to generate a good accurate prediction tree is covered, as well as including a full description of the experiment s library design to reduce the time to process all the information that came from the experiment s results.

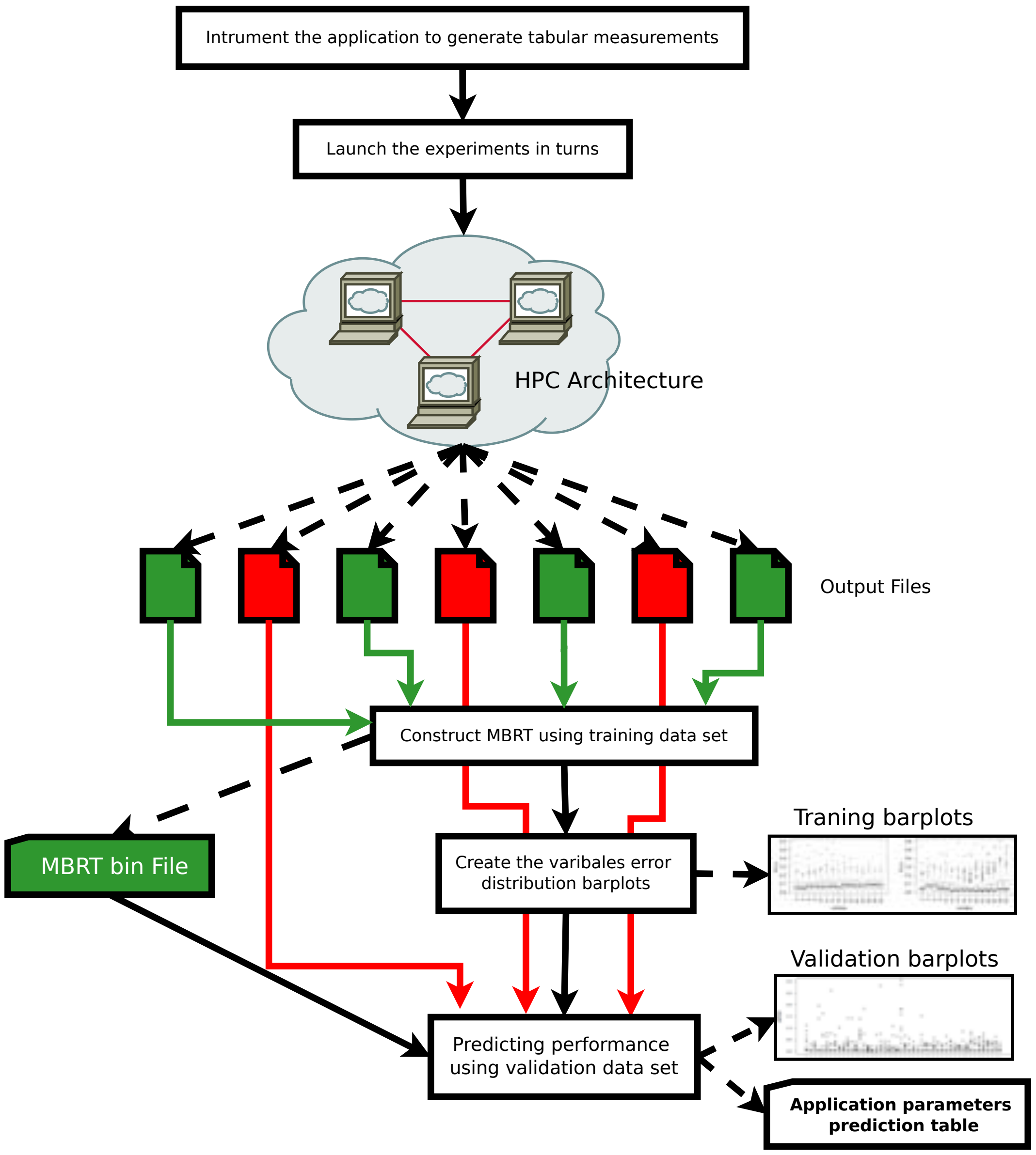
Intrument the application to generate tabular measurements
Launch the experiments in turns
HPC Architecture
Output Files
Construct MBRT using training data set
MBRT bin File
Create the varibales error distribution barplots
Traning barplots
Validation barplots
Predicting performance using validation data set
Application parameters prediction table

Figure 4.6: Experiments platform design.

Chapter 5

Experimental evaluation

5.1 Introduction

To validate the accuracy of our proposal, an experimental study has been carried out. Three applications have been selected in order to test the MBRT technique for predicting performance and efficiency index. Based on the methodology proposed, the results present here are the best ones we were able to achieve by repeatedly refining the tree parameters in each case. The first application is the Kmeans clustering algorithm. The second one is the classical Nbody problem, and the third one is a Heat transfer. All the applications were modified to adjust their logic to the Master/Worker communication pattern. Basically, all collective communications were transformed into a single communication from the Master process to each Worker and the return from each Worker to the Master. Additionally, a logic to process each message from the Workers into the global result was added into the logic of the Master process. In all the cases, the training and validation data sets of observations were collected from several executions, varying the application parameters and the workload. Each run was halted at the tenth iteration in order to reduce the time spent on this phase of the methodology. The three applications involved in this study have parallel regions with quadratic complexity $O(N^2)$. First, the results achieved using the IBM cluster will be presented and, next, the prediction results on the NUMA node. In each case, the performance prediction and efficiency prediction will be predicted using the MBRT technique. The prediction result error is calculated by the following expression:

$$Error = \frac{|pred - real|}{real} \tag{5.1}$$

5.2 Applications

5.2.1 Kmeans

The Kmeans [84] clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. This algorithm aims to partition n observations into k clusters, in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. The results we present here were achieved using 125 different attributes for each data object. The data objects to be clustered are evenly partitioned among all processes while the cluster centers are replicated using the master process as the global result collector. Additionally, the global-sum reduction for all cluster centers is performed at the end of each iteration by the master process in order to generate the new cluster centers. In all the experiments, the algorithm is executed to calculate the value of 10 clusters, varying the number of data object.

5.2.2 Nbody

The classical N-body problem [85] simulates the evolution of a system of N bodies, where the force exerted on each body arises from its interaction with all the other bodies in the system. Nbody algorithms have numerous applications in areas such as astrophysics, molecular dynamics and plasma physics. The simulation proceeds over time steps, each time computing the net force on every body and thereby updating its position and other attributes. If all pairwise forces are computed directly, this requires operations at each time step. In this version of the algorithm, the master process first sends the initial state of all the bodies to all the workers and the range of bodies that each worker has to process. Once this initialization phase is concluded, the workers compute the resulting force for its range of bodies and sends the final state of its bodies back to the master. The master process collects all the information and sends the updated state of each body to all the workers processes.

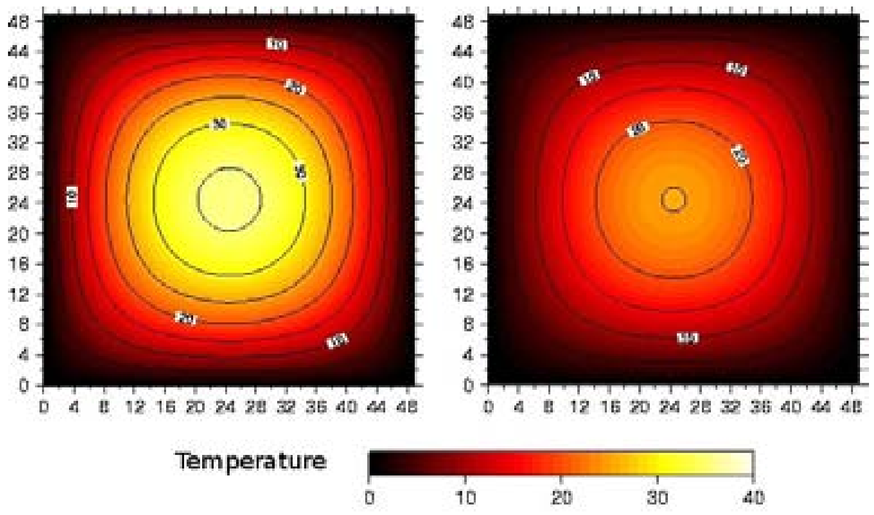

Figure 5.1: Surface heat state at first iteration and last for heat transfer.

5.2.3 Heat transfer

Many computational problems in geosciences involve finite difference approximations of partial differential equations [86]. Here, the goal is to learn to write a straight-forward finite difference model in parallel. Time-transient heat conduction in 2D is used as an example. Finite difference approximations excel as solutions to heat transfer through bodies of reasonably complex geometry and variable bulk thermal conductivity. The solution given here is simplified. The computational grid is rectangular and grid points are evenly spaced. Thermal diffusivity is held constant in the x and y directions over the entire grid. The initial temperature is high in the middle of the domain and zero at the boundaries. The boundaries are held at zero throughout the simulation.

The basic features of this finite difference model are illustrated by Figure 5.1, were we can see the evolution of the surface temperature at the beginning of the simulation and at the end. In this application, the master process partitions the finite difference grid among the nodes by rows. Each worker process solves a fraction of the grid and send to the Master the row on the upper and lower border in a single MPI message. The master then joins all the rows received and sends the respective borders back to each worker. This is a summary of an iteration of the application.

5.3 Experimental platforms

The experiments have been carried out on two different platforms. The first one is an IBM homogeneous cluster with 32 Nodes with 2 x Dual-Core Intel(R) Xeon(R) CPU 5160 at 3.00GHz with 4MB L2 cache (2x2) and 12 GB Fully Buffered DIMM 667 MHz of memory per node. The second computing platform used to test the proposed method is a NUMA architecture that consists of one PowerEdge C6145 node. This node has 4 AMD OpteronTM6376 of 16 cores with 128GB of DDR3 1600 MHz. Figures 5.2 and 5.3 show a general overview of the memory hierarchy in each case.

Taking a look at the IBM architecture we can see that each node is composed of four cores without hyper-therading sharing the level two of cache memory in groups of two cores. Level one of cache is split into two parts, one for the data cache and the other for instruction cache and it is associated with only one core respectively. In all the experiments executed on this architecture, there are two additional split variables as input to the MBRT algorithm l2_0 and l2_1. Each one will count the number of threads that share level two of cache memory. If the application used three threads, l2_0 must be equal to two and l2_1 must be one. The NUMA structure architecture is more complex. It is built of 64 cores without hyper-threading organized in groups of 16 cores in each socket. Each couple of cores shares the second level of cache memory. The third cache memory level in this case is shared by 16 cores in the same NUMA domain, while each socket consists

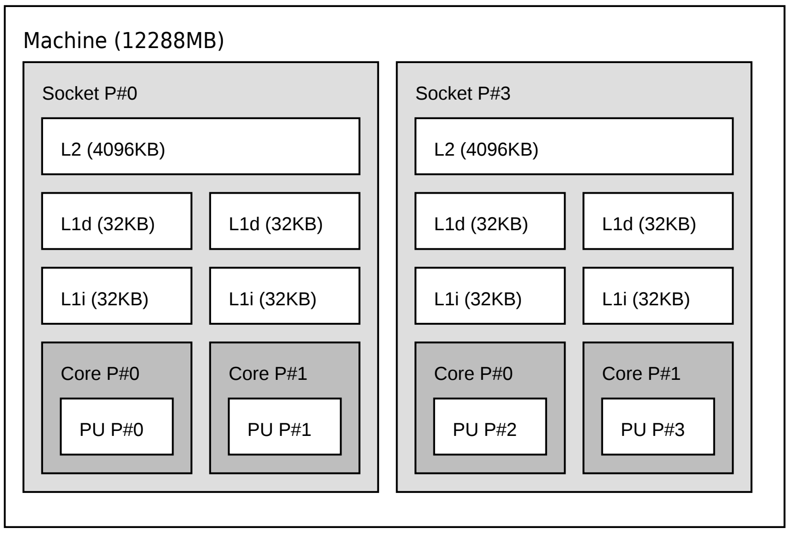

Figure 5.2: (a) IBM Node architecture.

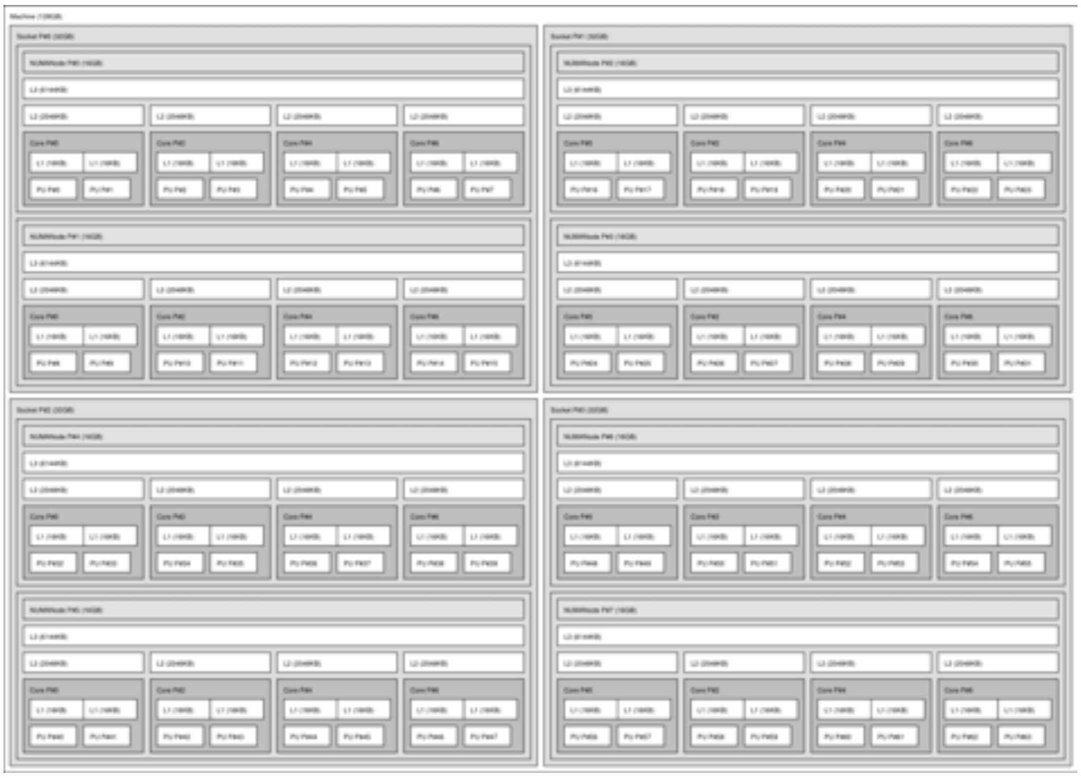

Figure 5.3: Catwoman Node architecture (NUMA).

of two NUMA domains.

Table 5.1: Kmeans application variables for training and validation data sets

Parameter	**Training**	**Validating**
Threads	1,2,3,4	1,2,3,4
Workers	2-28 in steps of 2	2-28 in steps of 2
Workload	200,400,600,800,1000,1200,1600	300,500,700,900,1100,1500,1800
	2000,2200,2800,3100,3500,4000	2100,2500,3000,3200,3800,4100
	4200,4600,4900,6000,10000,14000	4500,4800,5000,8000,12000,15000
	16000,20000,24000,26000,30000	18000,22000,25000,28000,50000
	70000,1e+05,3e+05,5e+05,7e+05	90000,2e+05,4e+05,6e+05,8e+05
	9e+05,1.2e+06,1.8e+06,2.2e+06	1e+06,1.5e+06,2e+06,2.5e+06
	2.8e+06,3.2e+06,3.8e+06,4.2e+06	3e+06,3.5e+06,4e+06,4.5e+06
	4.8e+06,5.2e+06,5.8e+06,6.2e+06	5e+06,5.5e+06,6e+06,6.5e+06
	6.8e+06,7.2e+06,7.8e+06,8.2e+06	7e+06,7.5e+06,8e+06,8.5e+06
	8.8e+06,9.2e+06,9.8e+06	9e+06,9.5e+06,1e+07

Table 5.2: NBody application variables for training and validation data sets

Parameter	**Training**	**Validating**
Threads	1,2,3,4	1,2,3,4
Workers	2-28 in steps of 2	2-28 in steps of 2
Workload	25,70,130,200,300,400,550,650,750	50,100,150,250,350,500,600,700,800
	850,950,1100,1300,1500,1750,1900	900,1000,1250,1400,1600,1800,1950
	2000,2200,2400,2600,2800,3000	2100, 2300, 2500, 2700, 2900, 3100
ccion	3200,3400,3600,3800,4000,4200	3300, 3500, 3750, 3900, 4100, 4300
	4400,4600,4750,4900,5100,5300	4500, 4700, 4800, 5000,5250, 5400
	5500,5780,5900,6300,6800,7300	5600, 5800, 6000, 6500, 7000, 7500
	7800,8300,8800,9200,9750,12000	8000, 8500, 9000, 9500,10000,14000
	15000,19000,22000,25000,29000	17000,20000,24000,27000, 30000
	32000,35000,39000,42000,45000	34000,37000,40000,43500,50000
	55000,65000,75000,85000	60000,70000,80000,90000

5.4 Prediction on homogeneous cluster

The experiments using this architecture were designed to study the prediction accuracy for performance and efficiency index restricting the use of only one MPI process per node of the cluster. The training set is the input for constructing the model-based regression tree. The validating set was done using different values for the workload that were not included in the training

Table 5.3: Heat transfer application variables for training and validation data sets

Parameter	**Training**	**Validating**
Threads	1,2,3,4	1,2,3,4
Workers	2-28 in steps of 2	2-28 in steps of 2
Workload	50,100,180,300,500,900,1250,1800, 2100,2300,2700,2900,3100,3300,3700,3900, 4100,4300,4700,4900,5100,5300,5700,5900, 6100,6300,6700,6900,7100,7300,7700,7900, 8100,8300,8700,8900,9100,9300,9700,9900, 10500,28000,11500,12500,13500,15000, 17000,20000,23000,33000,44000,37000, 39000,42000,50000,60000,70000,80000, 90000	80,150,250,400,750,1000,1500, 2000,2200,2500,2800,3000,3200,3500, 3800,40004200,4500,4800,5000,5200, 5500,5800,6000,6200,6500,6800,7000, 7200,7500,7800,80008200,8500,8800, 9000,9200,9500,9800,1000011000, 12000,13000,14000,16000,18000, 21000,25000,30000,35000,38000, 40000,43500,45000,55000,65000, 75000,85000

set. Both sets are similar in dimension. A complete description of both sets is specified in Tables 5.1, 5.2 and 5.3.

Based on the specific characteristics of each application, the general regression function prototype suffered some variations for each case (See Table 5.4).

Table 5.4: Functions and parameters for MBRT (Performance predition).

Application	**Regression function prototype**	**Candidate splitting variables**
Kmeans	$T_{iter} \sim \beta_3 w2 + \beta_2 w1 + \beta_1 workers + \beta_0$	w2,w1,workers,workload, local_workld,threads,l2_0,l2_1
Nbody	$T_{iter} \sim \beta_4 w2 + \beta_3 w1 + \beta_2 workers + \beta_1 local\_workld + \beta_0$	w2,w1,workers,workload, local_workld,threads,l2_0,l2_1
Heat transfer	$T_{iter} \sim \beta_4 w2 + \beta_3 w1 + \beta_2 workers + \beta_1 workload + \beta_0$	w2,w1,workers,workload, local_workld,threads,l2_0,l2_1

The communication time for all applications is proportional to the number of workers where the application is executed. That is the reason why the *workers* variable is included in all the functions. As the application Kmeans has a constant message size based on partial information of the cluster provided by each worker, the communication spent time in this case only depends on the number of workers. In the Nbody, the communication time depends on the local workload. The message is formed of the state information about all the particles processed by the worker. Heat transfer is an application where the message sent is the top and bottom boundaries for the 2D surface calculated in each worker. In this case, the size of each individual message will be proportional to the workload.

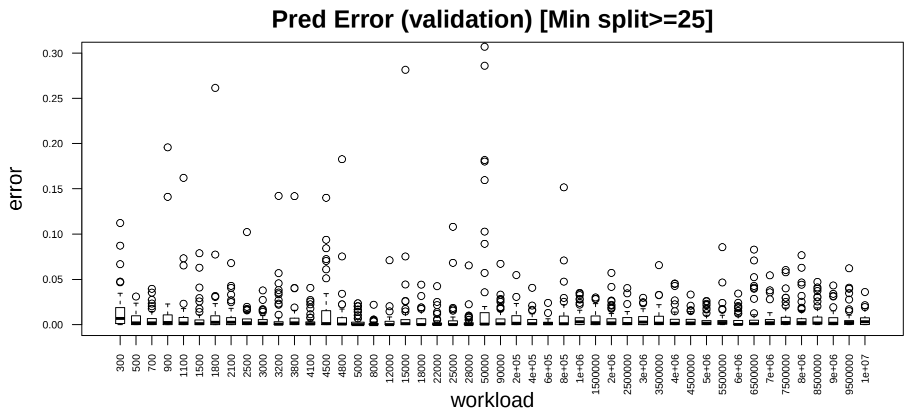

Figure 5.4: Relative error distribution for validation data set grouped by workload (Kmean).

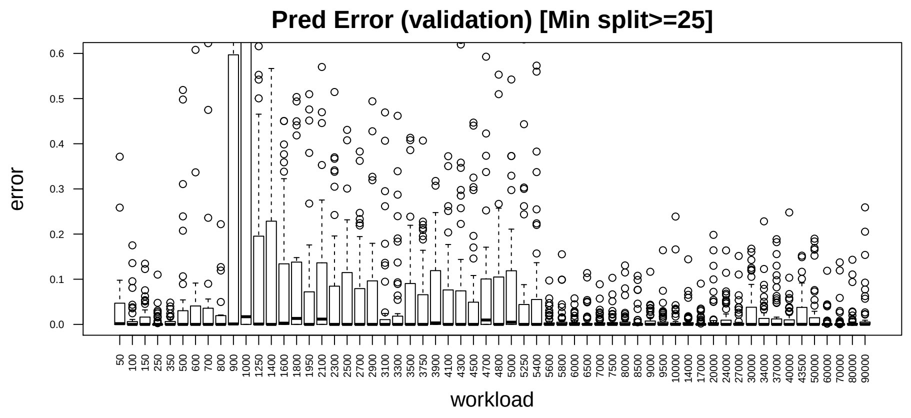

Figure 5.5: Relative error distribution for validation data set grouped by workload (Nbody).

5.4.1 Performance prediction

Taking a look at the performance execution time prediction results for the validation data set presented in the Figure 5.4 gives us an idea of the accuracy of these techniques for predicting performance using new values of workloads.

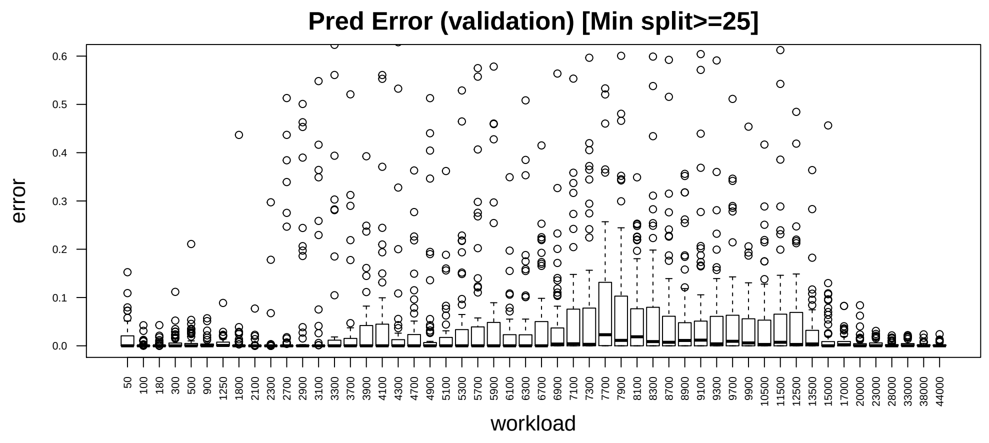

Figure 5.6: Relative error distribution using validation data set grouped by workload (Heat transfers).

The graph shows the error distribution grouped by different workloads for Kmeans application using the min split partition value of 25. The median error in the prediction is under 10% for all the different workloads, but additionally, the error dispersion for most validation samples is less than 5%. The outlier prediction errors still represent only 3 % of the validating set. These results can be considered as a typical example of well accurate performance prediction results because all the error distribution attributes are bound in an acceptable range.

The prediction results achieved with the NBody applications are shown in Figure 5.5. The model-based regression tree was generated with the default p-value and the same minimum split size for the leaf, but the results are slightly different compared with the previous application. The median error in prediction is under the 5% for all the cases, but the error distribution is not as accurate as Kmeans. The error distribution is clearly not homogeneously distributed, and there is a region where the error dispersion is significantly high. On this region, the upper part of the error distribution is usually bigger than the lower part. For those predictions with workloads from 900 to 1400, the error dispersion is greater than 60%. In the case of prediction from 1600 to 5000, the error dispersion can reach up to 30%. These two regions are clearly the main instability prediction sources where performance prediction errors and the best configuration of application parameter prediction error were mostly located. In the next section, the effect of those regions on the selection of the best application parameter configuration will be explained in depth. The outlier predictions represent 9%.

Looking into the prediction error results for the Heat transfer application (Fig. 5.6), we can see a general small error for all the workloads. In the range between 3900 and 17000, the biggest errors are found, but the median error is under the 5% for all the cases. The top 75% values of all of the distribution for any workload are under 15% error in prediction. There are only four results with the top over 20% in the range of 7700 to 8700. The outlier predictions are 7% of all the results for the validation data set, but many of them reach 60% error. On the one hand, the existence of several outlier predictions with a big error directly affects the ability to select the most appropriate parameters for the application, but, on the other hand, at least the prediction error is relatively low.

One reason that explains the appearance of outlier predictions is that we are testing the MBRT technique using experiments with really small iteration times. Given this as a fact, the noise introduced by the operating system becomes significant for the final accuracy.

5.4.2 Best performance configuration

Table 5.5 shows the differences between the configuration parameters for the best execution recorded in the validation data set and the best configuration predicted by the MBRT. The first four columns are the real values of Workers, Threads and best execution times for a particular workload. The last three columns are the parameters predicted and the real execution time using this configuration. The worst prediction selections of application parameters are highlighted in red, and the rows in yellow are those that do not match the configuration prediction where the iteration time mismatch is close to the real one. It can be observed that, in most cases, the predicted best configuration is the configuration that provides the best execution time, and, in some cases, there are other configurations that achieve better execution times, but the execution time provided by the proposed configuration is very close. There are only a few cases where the execution time is significantly different from the one reached by the proposed configuration.

The configuration prediction for workload 300 differs from the best in one thread (10 workers with 4 threads vs. 10 workers with 3 threads). It means that the difference in the number of computational resources is 11 cores. Usually, a small error in the number of threads using this architecture will generate a higher error in the number of computational resources used. Even when it is a bad prediction, the difference between the best time and the predicted time is small, so the prediction is almost guaranteed to be closer to the global minimum performance. Similar results can be seen in the prediction for workloads with values of 1800, 2100, 3000, 3200 , 4500 and 1800 where the difference between prediction and the best configuration is found in the number of workers. For all of these cases, the difference is 2 or 3 workers, but the best performance times compared with the performance for all the configuration predicted are quite similar. In these results, only one configuration prediction is very far from the best one. This happens for a workload of 900 using 18 workers with 2 threads. The prediction suggests using 10 workers with 4 threads. This large error can be explained because the prediction result matches one of the outlier predictions that can be seen in Picture 5.4.

Table 5.6 summarizes the best configuration prediction results for the Nbody application. The results are not as successful as the Kmeans application. In this case, the predictions for workloads 900, 1250, 1600, 1950, 2100, 2300 and 2500 are distinctly different with respect to the best execution for each one. The causes behind the appearance of these large errors can be seen in the prediction error distribution for the training data set 5.5. As the performance prediction error in the range of 900 to 5400 is higher than the rest, the probability of predicting an improper configuration of application parameters using this workloads will also increase. This problem can be avoided by increasing the training observations in this region. In fact, the results presented have been obtained using this methodology. The predictions using workloads 1000 and 1950 do not match exactly with the best configuration of parameters, but, in this case, the real performance time for the predictions is close to the real best one. For the rest of the workloads, the parameter selection is highly accurate.

The best configuration for performance predictions using the Heat transfer application are shown in Table 5.7. In summary, 13% of all the parameters suggested by the prediction using MBRT are wrong choices. There is only 2% of the predicted configurations that are not similar to the best one, but, in these cases, the difference in execution time is quite similar. The rest of the parameter predictions match the best one in each case exactly. The wrong prediction for workloads 2700, 2900, 3100 and 3700 are caused by the outlier performance predictions recorded in the previous barplot. There are several outliers with over 50% of error, which is one of the causes that can lead to problems in the prediction. The outlier appearance will be not a problem if all of them are under 20% error or they are very few compared with the size of the validation sample. The predictions for 9700, 10500 and 12500 suffer from the same problems with the outliers.

Table 5.5: Real vs Prediction best configuration (Kmean).

Real				Prediction		
Workload	**Workers**	**Threads**	**Best time**	**Workers**	**Threads**	**Time**
300	10	4	0.0006168	10	3	0.0006429
500	12	4	0.0007251	12	4	0.0007251
700	12	4	0.0008287	12	4	0.0008287
900	18	2	0.0008989	10	4	0.0009318
1100	16	2	0.0009360	16	2	0.0009360
1500	20	2	0.0010431	20	2	0.0010431
1800	18	4	0.0011265	20	4	0.0011617
2100	20	2	0.0011657	18	2	0.0012151
2500	20	2	0.0012628	20	2	0.0012628
3000	24	2	0.0013854	26	2	0.0014197
3200	20	2	0.0013988	24	2	0.0014831
3800	26	2	0.0015206	26	2	0.0015206
4100	26	2	0.0015549	26	2	0.0015549
4500	24	2	0.0016370	22	2	0.0016543
4800	24	2	0.0017144	24	2	0.0017164
5000	24	2	0.0017389	24	2	0.0018181
8000	24	2	0.0022768	24	2	0.0022831
12000	24	4	0.0029913	24	4	0.0029913
15000	28	4	0.003457	24	4	0.0046466
18000	24	4	0.0041204	28	4	0.0042806
22000	28	4	0.0045482	28	4	0.0045482
25000	28	4	0.0050322	28	4	0.0050322
28000	28	4	0.0055912	28	4	0.0060161
50000	28	3	0.011702	28	3	0.011702
90000	26	4	0.0188799	24	4	0.020413
2e+05	28	4	0.038121	28	4	0.038121
4e+05	28	4	0.0779791	28	4	0.0779791
6e+05	28	4	0.1188076	28	4	0.1188076
8e+05	28	4	0.1558503	28	4	0.1558503
1e+06	28	4	0.1881112	28	4	0.1881112
1.5e+06	28	4	0.2807814	28	4	0.2807814
2e+06	28	4	0.3738786	28	4	0.3738786
2.5e+06	28	4	0.4632191	28	4	0.4632191
3e+06	28	4	0.5542924	28	4	0.5542924
3.5e+06	28	4	0.6501051	28	4	0.6501051
4e+06	28	4	0.738608	28	4	0.738608
4.5e+06	28	4	0.830799	28	4	0.830799
5e+06	28	4	0.919348	28	4	0.919348
5.5e+06	28	4	1.0110899	28	4	1.0110899
6e+06	28	4	1.1046614	28	4	1.1046614
6.5e+06	28	4	1.1890828	28	4	1.1890828
7e+06	28	4	1.2842709	28	4	1.2842709
7.5e+06	28	4	1.3798037	28	4	1.3798037
8e+06	28	4	1.4589468	28	4	1.4589468
8.5e+06	28	4	1.557233	28	4	1.557233
9e+06	28	4	1.648762	28	4	1.648762
9.5e+06	28	4	1.7448824	28	4	1.7448824
1e+07	28	4	1.8314853	28	4	1.8314853

Table 5.6: Real vs Prediction best configuration (Nbody).

Real				Prediction		
Workload	**Workers**	**Threads**	**Best time**	**Workers**	**Threads**	**Time**
50	2	3	0.0002731	2	3	0.0002731
100	2	4	0.0003131	2	4	0.0003131
150	4	3	0.000467	4	3	0.000467
250	2	4	0.0006306	2	4	0.0006306
350	6	4	0.0008699	6	4	0.0008699
500	6	4	0.0012411	6	4	0.0012411
600	6	4	0.0014872	6	4	0.0014872
700	8	4	0.0018249	8	4	0.0018249
800	10	4	0.0020739	10	4	0.0020739
900	8	4	0.002439	20	2	0.0030912
1000	10	4	0.002757	8	4	0.0029204
1250	6	4	0.0043443	4	4	0.0056199
1400	4	4	0.0061357	4	4	0.0061357
1600	6	4	0.0067109	4	3	0.0104806
1800	4	4	0.0091444	4	4	0.0091444
1950	8	3	0.0097078	4	4	0.0105328
2100	6	4	0.0108928	4	3	0.015389
2300	6	4	0.0115614	4	3	0.0191191
2500	8	4	0.0122368	4	4	0.0163447
2700	6	4	0.0168237	6	3	0.0203117
2900	6	4	0.016433	6	4	0.016433
3100	6	3	0.0258621	6	3	0.0294567
3300	6	4	0.0235467	6	4	0.0235467
3500	8	3	0.0261408	8	3	0.0261408
3750	4	3	0.0546644	4	3	0.0558362
3900	8	3	0.0304743	8	3	0.0304743
4100	6	4	0.0305986	6	3	0.03637
4300	10	3	0.030979	10	3	0.03097901
4500	8	3	0.0366514	8	3	0.0366514
4700	12	4	0.0283032	12	4	0.0283032
4800	10	4	0.0308126	10	4	0.0308126
5000	8	4	0.0433158	8	4	0.0433158
5250	6	4	0.1053822	6	3	0.1060726
5400	6	4	0.0537166	4	4	0.0638894
5600	24	4	0.0294887	24	4	0.0294887
5800	24	4	0.0308569	24	4	0.0308569
6000	24	4	0.0310106	24	4	0.0310106
6500	24	4	0.0342993	24	4	0.0342993
7000	20	4	0.0370362	20	4	0.0370362
7500	28	4	0.0392284	28	4	0.0392284
8000	26	4	0.0425798	26	4	0.0425798
8500	24	4	0.0452163	24	4	0.0452163
9000	28	4	0.0495081	28	4	0.0495081
9500	26	4	0.0506968	26	4	0.0506968
10000	28	4	0.0562127	26	4	0.0564312
14000	28	4	0.0909772	28	4	0.0909772
17000	28	4	0.1232944	28	4	0.1232944
20000	28	4	0.1554981	28	4	0.1554981
24000	28	4	0.220228	24	4	0.2349681
27000	24	4	0.2923574	28	4	0.3003843
30000	26	4	0.3338369	26	4	0.3338369
34000	26	4	0.4097942	26	4	0.4097942
37000	28	4	0.4596396	26	4	0.4909842
40000	28	4	0.5342135	28	4	0.5342135
43500	28	4	0.6194289	28	4	0.6194289
50000	28	4	0.7997748	26	4	0.8424526
60000	28	4	1.1100477	28	4	1.1100477
70000	28	4	1.5817435	26	4	1.5817435
80000	28	4	1.9203103	28	4	1.9203103
90000	26	4	2.5653735	28	4	2.3234218

Table 5.7: Real vs Prediction best configuration (Heat transfer).

Real				Prediction		
Workload	**Workers**	**Threads**	**Best time**	**Workers**	**Threads**	**Time**
50	3	2	0.00013	3	2	0.00013
100	3	4	0.00016	3	4	0.00016
180	3	3	0.00020	3	3	0.00020
300	3	4	0.00028	3	4	0.00028
500	5	4	0.00041	5	4	0.00041
900	8	4	0.00067	8	4	0.00067
1250	11	4	0.00098	11	4	0.00098
1800	11	4	0.00146	11	4	0.00147
2100	8	4	0.00179	8	4	0.00171
2300	8	4	0.00222	8	4	0.00249
2700	13	4	0.00252	11	1	0.00491
2900	9	4	0.00286	21	2	0.00339
3100	11	4	0.00305	9	2	0.00445
3300	11	4	0.00347	11	4	0.00346
3700	13	4	0.00396	13	2	0.00498
3900	9	4	0.00457	9	4	0.00459
4100	13	4	0.00497	13	4	0.00497
4300	19	4	0.00515	19	4	0.00515
4700	9	4	0.00676	9	4	0.00678
4900	9	4	0.00695	9	4	0.00695
5100	9	4	0.00743	9	4	0.00741
5300	9	4	0.00811	9	4	0.00811
5700	11	3	0.00963	8	4	0.01072
5900	8	4	0.01148	8	4	0.01148
6100	11	4	0.01001	11	4	0.01002
6300	8	4	0.01324	8	4	0.01324
6700	8	4	0.01452	8	4	0.01456
6900	9	4	0.01280	8	2	0.06214
7100	11	4	0.01256	11	4	0.01260
7300	9	2	0.02212	5	4	0.02404
7700	7	3	0.02327	7	3	0.02327
7900	8	4	0.01901	8	4	0.01901
8100	8	4	0.01983	8	4	0.01983
8300	9	3	0.02418	9	3	0.02418
8700	9	3	0.02605	9	3	0.02605
8900	11	4	0.02424	11	4	0.02421
9100	8	3	0.02843	8	3	0.02843
9300	8	2	0.03922	8	2	0.03922
9700	11	4	0.022576	5	4	0.04111
9900	7	4	0.03027	7	4	0.03027
10500	11	2	0.03811	8	2	0.05003
11500	5	4	0.05715	5	4	0.05715
12500	11	2	0.05105	5	4	0.07002
13500	8	2	0.08249	8	2	0.082492
15000	11	2	0.07244	11	2	0.07244
17000	21	4	0.04980	21	4	0.04980
20000	23	4	0.06074	23	4	0.06074
23000	23	4	0.07027	23	4	0.07027
28000	29	4	0.09244	29	4	0.09256
33000	31	4	0.11583	31	4	0.11607
38000	31	4	0.14249	31	4	0.14249
44000	31	4	0.17777	31	4	0.17780

5.4.3 Efficiency index prediction

Once the MBRT is validated to predict the performance of parallel applications, one important subject is the evaluation of this technique to predict the behavior of the efficiency. In order to accomplish this task, we first take the efficiency index expression explained previously in Chapter 4. The first step, is to evaluate this response variable for all the observations from the training data set in order to construct the final prediction tree. Once this is done, the E_{index} can be included as the output prediction for the MBRT. The regression expression prototype for each application will be the same as that used for the performance prediction because the behavior follows a similar trend. Thus, the summary of the arguments for applying the technique is presented in the following table:

Table 5.8: Functions and parameters for MBRT (Efficiency index prediction).

Application	**Regression function prototype**	**Candidate splitting variables**
Kmeans	$E_{index} \sim \beta_3 w2 + \beta_2 w1 + \beta_1 workers + \beta_0$	w2,w1,workers,workload, local_workld,threads,l2_0,l2_1
Nbody	$E_{index} \sim \beta_4 w2 + \beta_3 w1 + \beta_2 workers + \beta_1 local\_workld + \beta_0$	w2,w1,workers,workload, local_workld,threads,l2_0,l2_1
Heat transfer	$E_{index} \sim \beta_4 w2 + \beta_3 w1 + \beta_2 workers + \beta_1 workload + \beta_0$	w2,w1,workers,workload, local_workld,threads,l2_0,l2_1

In the future, if we want to search for the best application configuration with highest level of efficiency, the prediction with the smallest value indicates the best selection.

Picture 5.7 shows the prediction results for the efficiency index using the same training set to construct the tree and validation set for generating this bar plot. The final tree is formed of several internal nodes in the following proportion: 8.4% of all the internal nodes use *local_workld* for the splitting action, around 4.2% of these nodes split the tree by the number of threads, 16.6% of the internal nodes use the number of workers, 17.1% of the split actions were done by w_1 and 41% by the the derived variable w_2. Finally, 12% of all the splits were done using the application workload. Like the trees obtained for predicting the performance, the affinity variables were not taken into account in the partitioning. This is a clear sign that the information provided by adding these variables is not significant. The applications involved in this study do not suffer much variation in their behavior with respect to a change in their values. In the barplot, the number of outliers is around only 3% of all the predictions. They are concentrated in the range between 300 and 200000. The rest of the outliers are under 20% prediction error. The tops of all the distributions are under 20% error. Based on this result, we expect a high accuracy when we search for the best configuration for this application. It is quite interesting that the importance of the number of workers is similar to the importance of the relationship $\frac{Workload}{Workers*Thrs}$ defined in w_1.

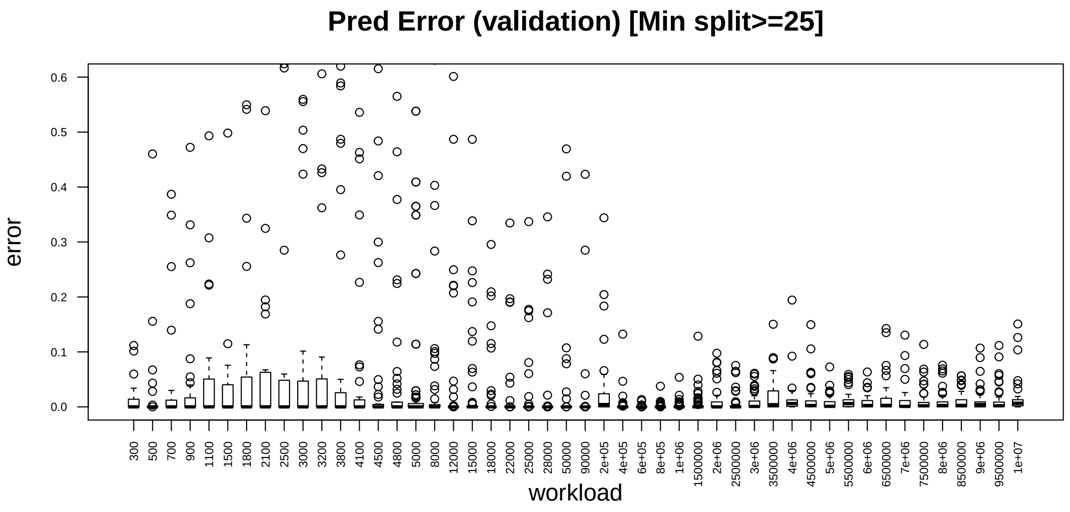

Figure 5.7: Relative error of efficiency index using the validation data set grouped by workload (Kmeans).

The prediction boxplot results using the Nbody are presented in Picture 5.8. In this case, the results are even more accurate than the those we have seen with Kmean. The tops of the boxplot distribution of error for most of the cases are close to 2%, except for workloads 1400 and 1600. In this case, the outliers predictions reached 5% of all the predictions, but many of them are under 20% error. To reach these results, the final tree is composed of

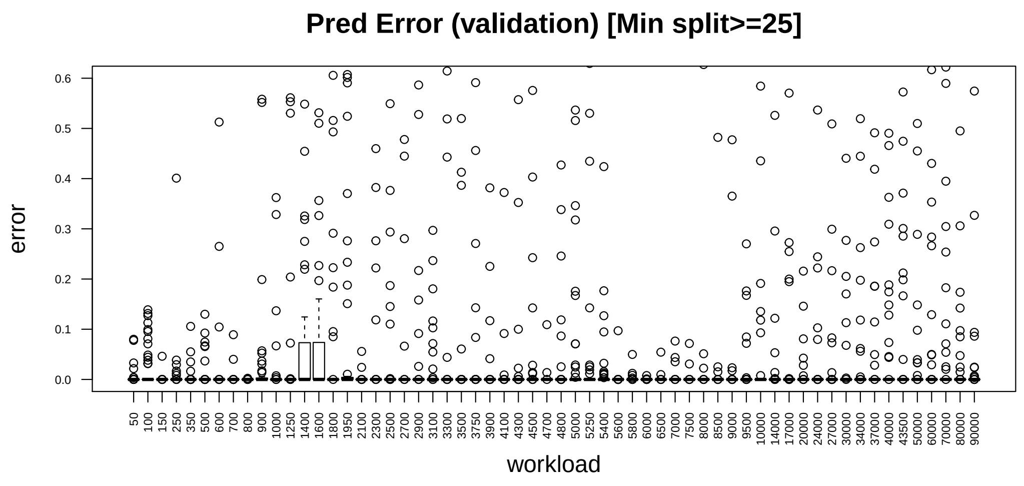

Figure 5.8: Relative error of efficiency index using the validation data set grouped by workload (Nbody).

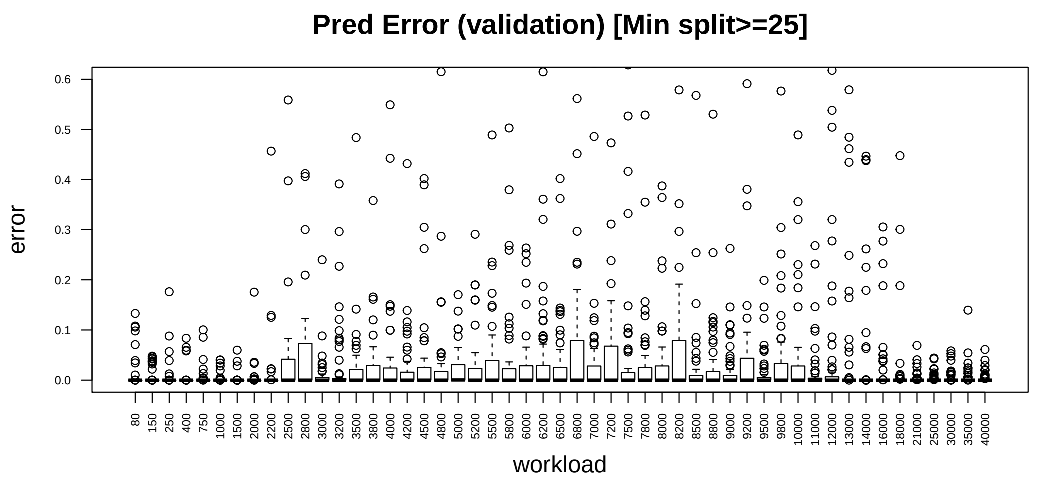

Figure 5.9: Relative error of efficiency index using the validation data set grouped by workload (Heat transfer).

10% of internal nodes that use *local_workld* as the split variable, 2.8% of all the splits were done by using the number of threads per MPI process, 16% of the internal nodes use w_0, 18.1% of split actions used w_1 and 44.7% used the number of workers. The split actions using the global workload of the application represent 7.2% of the whole. The local workload of each worker and the affinity variables were not considered for splitting the tree. Looking at these numbers, the most important variable that generates instability in the behavior is the number of workers. This conclusion is not surprising; we must expect it, considering that this is the application parameter with the largest range of possible values that divides the workload in each term of w_2 and w_1. In this application, the importance of using the number of threads is even lower than in Kmeans. This means that the application does not suffer too much degradation in its behavior from the contribution of the variable number of thread.

The errors achieved for predicting the efficiency index using the Heat transfer application are summarized in Picture 5.9. The resulting tree is formed by 13.5% of internal nodes that use the local workload for splitting, 4.1% divide the tree by the number of threads, 32.5% of the internal nodes performed a partitioning action using w_2, 18% for w_1 and 19% of these nodes divide the tree by the number of workers. The global workload has been used in 12.21% of all the nodes and only 0.02% of the splitting actions were done by the affinity variable $l2_0$. The outlier prediction in this case reaches up to 8% of all the predictions. The higher outliers are concentrated in the range between 2500 and 18000 of the workloads, but the rest are under 15% error. In the error distribution, The upper quartile is under the 20% error for all the cases. In the case of these results, the most important cause that could provoke errors in the selection of the parameters of the application is the relatively high value achieved in the number of outliers.

To summarize, the results presented here are better when compared with those obtained for the prediction performance. The behavior of the efficiency index suffers less instability in predicting their behavior. In the next section, we take a look at the results achieved when we try to select the best application configuration.

5.4.4 Best efficiency index configuration

The application parameter prediction using the Kmeans application (see Table 5.9) shows highly accurate results. Similar to previous results, the yellow rows represent the predictions that do not match with the best one but where the value of the efficiency index is close to it. The red rows are the worst predictions. In this case, only 2% of all the predictions are consider far from the best one. 12% are close to the best one, even when they are predictions where the best configuration and the prediction do not match exactly. The rest of the predictions match exactly. The errors are found in the range of workloads between 300 and 90000. If we look at the bar plot of the error distribution, we can see that this is the region where the highest values of the distribution of outliers are found. This correlation is evidence of the strong relationship that exists between these two results.

The results achieved using NBody are shown in Table 5.10. The predictions were the parameters do not match even when the efficiency index is close to the best one are the 15%. They are found in using the workloads 250, 350, 5250, 1400, 3700, 40000 and so on. The worst predictions are 5% in this case. For this results, the source of error are the several outliers presented in the range between 1250 to 5400 and between 9500 to 90000. Thanks to outliers does not exceed 5 %, the effect caused by them does not generate more errors in this table. The worst predictions are located using 17000, 20000 and 43500 for the value of the global workload of the application.

Table 5.11 summarizes the results of the comparison of the best application configuration parameters with the predicted configuration for reaching the best efficiency index using the Heat transfer. In summary, the results achieved for Kmeans show that 11.7% of all predictions are close to the best efficiency index, but the application parameters are different from the best one. The 9% of the predictions are a combination of application parameters where the error achieved in the efficiency index is above 40%. The rest of the predictions match the best configuration of parameters exactly.

Table 5.9: Real vs Prediction best efficiency index configuration (Kmean).

Real				Prediction		
Workload	**Workers**	**Threads**	**Best pindex**	**Workers**	**Threads**	**Pred pindex**
300	3	2	0.00013	5	4	0.00014
500	5	4	0.00016	5	4	0.00016
700	7	4	0.00016	7	4	0.00016
900	7	4	0.00018	7	4	0.00018
1100	11	4	0.00018	11	4	0.00018
1500	11	4	0.00019	13	4	0.0002
1800	13	3	0.00021	11	4	0.00023
2100	11	4	0.00021	11	4	0.00021
2500	15	2	0.00023	15	3	0.00023
3000	15	3	0.00024	15	3	0.00024
3200	15	2	0.00024	15	2	0.00024
3800	17	2	0.00026	17	2	0.00026
4100	15	2	0.00026	15	2	0.00026
4500	15	2	0.00028	15	2	0.00028
4800	19	2	0.00028	19	2	0.00028
5000	15	2	0.0003	21	4	0.0003
8000	25	2	0.00034	25	1	0.00046
12000	25	4	0.00041	25	4	0.00041
15000	25	2	0.00048	25	2	0.00048
18000	25	4	0.00053	25	4	0.00053
22000	27	4	0.0006	29	4	0.0006
25000	29	4	0.00063	29	4	0.00063
28000	29	4	0.00069	29	4	0.00069
50000	29	3	0.00141	29	3	0.00141
90000	27	4	0.00207	27	4	0.00207
2e+05	29	4	0.00402	29	4	0.00402
4e+05	29	4	0.00836	29	4	0.00836
6e+05	29	4	0.01295	29	4	0.01295
8e+05	29	4	0.01684	29	4	0.01684
1e+06	29	4	0.01981	29	4	0.01981
1500000	29	4	0.0295	29	4	0.0295
2e+06	29	4	0.03888	29	4	0.03888
2500000	29	4	0.04837	29	4	0.04837
3e+06	29	4	0.05691	29	4	0.05691
3500000	29	4	0.06747	29	4	0.06747
4e+06	29	4	0.07653	29	4	0.07653
4500000	29	4	0.0851	29	4	0.0851
5e+06	29	4	0.09589	29	4	0.09589
5500000	29	4	0.10427	29	4	0.10427
6e+06	29	4	0.11295	29	4	0.11295
6500000	29	4	0.12367	29	4	0.12367
7e+06	29	4	0.13083	29	4	0.13083
7500000	29	4	0.14128	29	4	0.14128
8e+06	29	4	0.15226	29	4	0.15226
8500000	29	4	0.15867	29	4	0.15867
9e+06	29	4	0.16987	29	4	0.16987
9500000	29	4	0.18015	29	4	0.18015
1e+07	29	4	0.18697	29	4	0.18697

Table 5.10: Real vs Prediction best efficiency index configuration (Nbody).

Real				Prediction		
Workload	**Workers**	**Threads**	**Best pindex**	**Workers**	**Threads**	**Pred pindex**
50	3	1	0.00041	3	1	0.00041
100	3	3	0.00013	3	3	0.00013
150	3	2	0.0003	3	2	0.0003
250	3	3	0.00015	3	4	0.00015
350	3	4	0.00045	3	3	0.00049
500	3	4	0.00028	3	4	0.00028
600	5	3	0.00079	5	3	0.00079
700	5	4	0.00085	5	4	0.00085
800	7	4	0.001	7	4	0.001
900	7	4	0.00115	7	4	0.00115
1000	3	4	0.00065	3	4	0.00065
1250	7	4	0.00183	7	4	0.00183
1400	5	4	0.00215	5	4	0.00215
1600	7	4	0.00254	7	4	0.00254
1800	5	4	0.00314	5	4	0.00314
1950	5	4	0.00333	5	4	0.00333
2100	7	4	0.00373	7	4	0.00373
2300	7	4	0.00402	7	4	0.00402
2500	3	4	0.00297	3	4	0.00297
2700	7	4	0.00595	7	4	0.00595
2900	7	4	0.00547	7	4	0.00547
3100	7	3	0.00841	7	3	0.00841
3300	7	4	0.0077	7	4	0.0077
3500	3	4	0.0056	3	4	0.0056
3750	3	4	0.01453	3	4	0.01453
3900	9	3	0.00991	9	3	0.00991
4100	7	4	0.00965	7	4	0.00965
4300	11	3	0.01056	11	3	0.01056
4500	9	3	0.01133	9	3	0.01133
4700	13	4	0.01099	13	4	0.01099
4800	11	4	0.01081	11	4	0.01081
5000	9	4	0.01596	9	4	0.01596
5250	5	2	0.02834	3	4	0.0315
5400	5	4	0.01731	5	4	0.01731
5600	11	4	0.01105	11	4	0.01105
5800	17	4	0.01176	17	4	0.01176
6000	13	4	0.01149	13	4	0.01149
6500	15	4	0.01285	15	4	0.01285
7000	17	4	0.0137	17	4	0.0137
7500	15	4	0.01481	15	4	0.01481
8000	21	4	0.01688	21	4	0.01688
8500	21	4	0.01767	21	4	0.01767
9000	17	4	0.01893	17	4	0.01893
9500	17	4	0.01955	17	4	0.01955
10000	17	4	0.02137	17	4	0.02137
14000	23	4	0.03371	29	4	0.03562
iter 17000	25	4	0.04309	3	4	0.27957
20000	29	4	0.05369	3	4	0.17037
24000	27	4	0.07004	27	4	0.07004
27000	25	4	0.09105	25	4	0.09105
30000	27	4	0.10418	27	4	0.10418
34000	27	4	0.12277	27	4	0.12277
37000	29	4	0.14029	27	4	0.14911
40000	29	4	0.16243	27	4	0.16547
43500	29	4	0.18494	19	4	0.23923
50000	29	4	0.23319	27	4	0.24054
60000	29	4	0.3132	25	4	0.34728
70000	27	4	0.4339	27	4	0.4339
80000	29	4	0.52667	27	4	0.55206
90000	27	4	0.68957	27	4	0.68957

Table 5.11: Real vs Prediction best efficiency index configuration (Heat transfer).

Real				Prediction		
Workload	**Workers**	**Threads**	**Best pindex**	**Workers**	**Threads**	**Pred pindex**
80	3	1	0.00016	3	1	0.00016
150	3	1	0.00011	3	1	0.00011
250	3	1	0.00012	3	1	0.00012
400	3	3	0.00014	3	3	0.00014
750	3	3	0.00024	3	3	0.00024
1000	5	4	0.00027	5	4	0.00027
1500	7	4	0.00038	7	4	0.00038
2000	8	4	0.00054	8	4	0.00054
2200	7	4	0.0006	7	4	0.0006
2500	7	4	0.00069	7	4	0.00069
2800	8	4	0.00077	8	4	0.00077
3000	7	4	0.00083	7	4	0.00083
3200	7	4	0.00089	7	4	0.00089
3500	8	4	0.00109	8	4	0.00109
3800	7	4	0.00115	7	4	0.00115
4000	7	4	0.00129	7	4	0.00129
4200	7	4	0.0014	7	4	0.0014
4500	7	4	0.00145	7	4	0.00145
4800	7	4	0.00165	8	1	0.00356
5000	7	4	0.00179	7	4	0.00179
5200	7	4	0.0019	7	4	0.0019
5500	9	4	0.00197	7	4	0.00201
5800	9	4	0.00215	8	4	0.00249
6000	9	4	0.00222	5	3	0.00357
6200	11	4	0.00242	11	4	0.00242
6500	7	4	0.00268	5	4	0.00351
6800	9	4	0.00275	8	4	0.00331
7000	9	4	0.0029	9	4	0.0029
7200	8	3	0.00365	8	3	0.00365
7500	9	4	0.00322	8	2	0.00476
7800	8	4	0.00418	8	4	0.00418
8000	9	2	0.0053	9	2	0.0053
8200	7	4	0.00418	7	4	0.00418
8500	9	2	0.00576	9	2	0.00576
8800	8	2	0.00663	8	2	0.00663
9000	11	2	0.00553	5	4	0.00644
9200	9	2	0.00677	8	2	0.00737
9500	7	3	0.00646	7	3	0.00646
9800	7	3	0.00677	7	3	0.00677
10000	7	3	0.00758	7	4	0.00781
11000	9	2	0.00949	8	2	0.0105
12000	9	2	0.01056	5	4	0.01096
13000	7	3	0.01192	7	3	0.01192
14000	8	2	0.02976	8	2	0.02971
16000	11	3	0.03307	11	3	0.03311
18000	11	4	0.01413	11	4	0.01413
21000	11	4	0.01713	11	4	0.01713
25000	21	4	0.0211	21	4	0.0211
30000	21	4	0.0265	21	4	0.0265
35000	23	4	0.0322	23	4	0.0322
40000	23	4	0.03937	23	4	0.03937

5.5 Prediction on NUMA architecture

The experiments using this architecture are designed to study the prediction accuracy for the efficiency index metric. The training and validation data sets are similar in dimension, and they are composed of several combinations of processes, threads and affinity settings for different workloads. A complete description of both sets is specified in Tables 5.12, 5.13 and 5.14. In these tables, each row of possible values of processes is related to a similar row for the number of threads, so we can only have one thread per MPI process if we have 63 workers. We have not included the study of performance prediction even when the results were really good. The main reason is because that the best parameter settings usually match the largest possible amount of computing resources. On the other hand, the differences in execution time using the same number of resources for any configuration is under 10%.

Table 5.12: Kmeans application variables for training and validation data sets (NUMA).

Parameter	**Training**	**Validating**
Threads	1-16	1-16
	1-8	1-8
	1-4	1-4
	1-2	1-2
	1	1
Workers	2 and 3	2 and 3
	2,3-7 in steps of 2	2,3-7 in steps of 2
	2,3-15 in steps of 2	2,3-15 in steps of 2
	2,3-31 in steps of 2	2,3-31 in steps of 2
	2,3-63 in steps of 2	2,3-63 in steps of 2
Workload	200,400,600,800,1000,1200,1600	300,500,700,900,1100,1500,1800
	2000,2200,2800,3100,3500,4000	2100,2500,3000,3200,3800,4100
	4200,4600,4900,6000,10000,14000	4500,4800,5000,8000,12000,15000
	16000,20000,24000,26000,30000	18000,22000,25000,28000,50000
	70000,1e+05,3e+05,5e+05,7e+05	90000,2e+05,4e+05,6e+05,8e+05
	9e+05,1.2e+06,1.8e+06,2.2e+06	1e+06,1.5e+06,2e+06,2.5e+06
	2.8e+06,3.2e+06,3.8e+06,4.2e+06	3e+06,3.5e+06,4e+06,4.5e+06
	4.8e+06,5.2e+06,5.8e+06,6.2e+06	5e+06,5.5e+06,6e+06,6.5e+06
	6.8e+06,7.2e+06,7.8e+06,8.2e+06	7e+06,7.5e+06,8e+06,8.5e+06
	8.8e+06,9.2e+06,9.8e+06	9e+06,9.5e+06,1e+07

The experiment group that can use a number of workers from 2 to 63 can use only one thread per MPI process. Those experiment groups in the class of 2 to 31 workers can only have one or two threads per MPI process. In this case, the couple of threads per process always share all the different cache memory levels. In the class of experiments where the number of processes is

Table 5.13: NBody application variables for training and validation data sets (NUMA).

Parameter	Training	Validating
Threads	1-16	1-16
	1-8	1-8
	1-4	1-4
	1-2	1-2
	1	1
Workers	2 and 3	2 and 3
	2,3-7 in steps of 2	2,3-7 in steps of 2
	2,3-15 in steps of 2	2,3-15 in steps of 2
	2,3-31 in steps of 2	2,3-31 in steps of 2
	2,3-63 in steps of 2	2,3-63 in steps of 2
Workload	25,70,130,200,300,400,550,650,750	50,100,150,250,350,500,600,700,800
	850,950,1100,1300,1500,1750,1900	900,1000,1250,1400,1600,1800,1950
	2000,2200,2400,2600,2800,3000	2100, 2300, 2500, 2700, 2900, 3100
	3200,3400,3600,3800,4000,4200	3300, 3500, 3750, 3900, 4100, 4300
	4400,4600,4750,4900,5100,5300	4500, 4700, 4800, 5000,5250, 5400
	5500,5780,5900,6300,6800,7300	5600, 5800, 6000, 6500, 7000, 7500
	7800,8300,8800,9200,9750,12000	8000, 8500, 9000, 9500,10000,14000
	15000,19000,22000,25000,29000	17000,20000,24000,27000, 30000
	32000,35000,39000,42000,45000	34000,37000,40000,43500,50000
	55000,65000,75000,85000	60000,70000,80000,90000

Table 5.14: Heat transfer application variables for training and validation data sets (NUMA).

Parameter	Training	Validating
Threads	1-16	1-16
	1-8	1-8
	1-4	1-4
	1-2	1-2
	1	1
Workers	2 and 3	2 and 3
	2,3-7 in steps of 2	2,3-7 in steps of 2
	2,3-15 in steps of 2	2,3-15 in steps of 2
	2,3-31 in steps of 2	2,3-31 in steps of 2
	2,3-63 in steps of 2	2,3-63 in steps of 2
Workload	50,100,180,300,500,900,1250,1800,	80,150,250,400,750,1000,1500,
	2100,2300,2700,2900,3100,3300,3700,3900,	2000,2200,2500,2800,3000,3200,3500,
	4100,4300,4700,4900,5100,5300,5700,5900,	3800,40004200,4500,4800,5000,5200,
	6100,6300,6700,6900,7100,7300,7700,7900,	5500,5800,6000,6200,6500,6800,7000,
	8100,8300,8700,8900,9100,9300,9700,9900,	7200,7500,7800,80008200,8500,8800,
	10500,28000,11500,12500,13500,15000,	9000,9200,9500,9800,1000011000,
	17000,20000,23000,33000,44000,37000,	12000,13000,14000,16000,18000,
	39000,42000,50000,60000,70000,80000,	21000,25000,30000,35000,38000,
	90000	40000,43500,45000,55000,65000,
		75000,85000

limited to the range of 2 to 15, the range of threads each process can have in its parallel OpenMP region is between the range of 1 and 4 threads. Depending on the number of threads used, they have to share the level L2 cache or not with the others, but they always share the L3 cache memory. The experiments that are grouped in the category of 2 to 7 workers have the use of threads limited to the range of 1 to 8. In this case, each process reserves all the cores that share the L3 cache memory and can use them depending on the amount of threads defined for each experiment. Finally, the experiment group that uses 2 or 3 workers can between 1 and 16 threads. All these threads will be located on the same socket and will share L3 cache memory. There are other possible configurations of affinity that can be covered, but, in these experiments, they are restricted to those that use the close affinity between threads.

The regression function prototype defined for each MBRT and the candidate splitting parameters are summarized in Table 5.15. The regression functions are the same as those used bellow, but there are three groups of parameters for defining the affinity. In this architecture, the variables added as splitting candidates for MBRT are related to the number of MPI processes on the same socket (from s0 to s3), the number of MPI processes that share each L3 cache (from pl3_0 to pl3_7) and the number of threads that shares the same L3 cache (from tl3_0 to tl3_7).

Table 5.15: Functions and parameters for MBRT (Efficiency index predition) on NUMA.

Application	**Regression function prototype**	**Candidate splitting variables**
Kmeans	$E_{index} \sim \beta_3 w2 + \beta_2 w1 + \beta_1 workers + \beta_0$	w2,w1,workers,workload, local_workld,threads,s0-s3, pl3_0-pl3_7, tl3_0-tl3_7
Nbody	$E_{index} \sim \beta_4 w2 + \beta_3 w1 + \beta_2 workers + \beta_1 local\_workld + \beta_0$	w2,w1,workers,workload, local_workld,threads,s0-s3, pl3_0-pl3_7, tl3_0-tl3_7
Heat transfer	$E_{index} \sim \beta_4 w2 + \beta_3 w1 + \beta_2 workers + \beta_1 workload + \beta_0$	w2,w1,workers,workload, local_workld,threads,s0-s3, pl3_0-pl3_7, tl3_0-tl3_7

5.5.1 Efficiency index prediction

The results presented in Table 5.10 show the accuracy achieved using the MBRT to predict the efficiency index in Kmeans application. 75% of the quartiles for all the errors distribution are below 10% error for all the workloads except in the first three cases. The upper quartile is below 20% in prediction error. There are only upper quartiles above 30% for the workloads 100, 200, 300 and 500. The number of outlier prediction is around 3% of all done for the validation data set. In this case, the results were achieved using a min split value of 120. Even with the high parameters in this case, the prediction results become very accurate. We recommend using the largest possible value for the mini split because this action decreases the possibility of outlier occurrence and also allows us to avoid over-fitting.

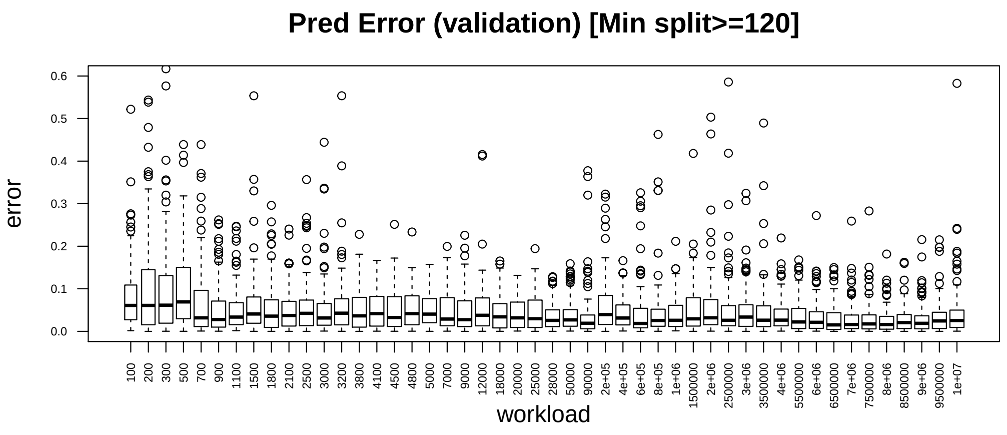

Figure 5.10: Relative error of efficiency index using the validation data set grouped by workload (Kmeans) on NUMA.

The tree that generates these results is structured by 13.6% of all splitting nodes that use w_2 for applying this action. 9% of the splitting actions were done by w_1 and 20% by the number of workers. The Kmeans application is also sensitive to the local workload of each worker. There are 15.9% of all the internal nodes that use this variable. The affinity configuration for the MPI process becomes relevant in this architecture. This is why we can see that 17% of all the splitting was done using one of the s variables, and 5.45% of all the actions are based on any of the $pl3$ process affinity variables. Even when this is a small percentage, the split actions that involve process affinity variables are located at the top of the MBRT because the most important variation in efficiency index using these variables, is detected globally and not in particular cases.

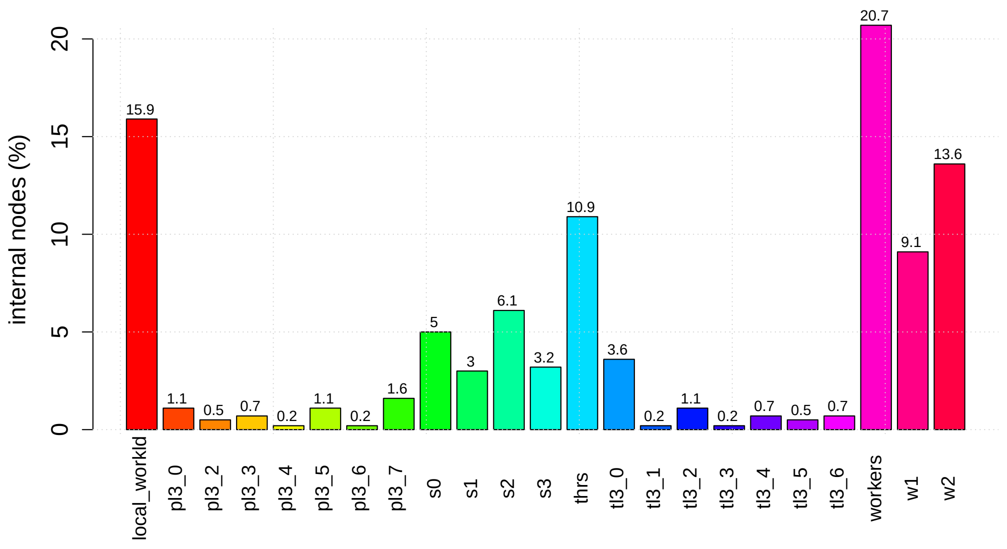

Figure 5.11: Distribution of internal nodes in the MBRT using Kmeans on NUMA.

A similar situation occurs with the number of threads and their affinity. The number of threads is important in the efficiency index behavior, with 10.9% of internal nodes that split the tree using this variable. The percentage of internal nodes that use any thread affinity variables is over 7%, and most are located at the top of the tree.

The distribution of error for predicting the efficiency index using Nbody is shown in Picture 5.12. In this case, the result was achieved using a minimum split value of 20. The number of outlier prediction is around 10%, but, fortunately, these values do not exceed 20% error. The 75 percent quartile is around 10% of prediction error. Taking a look at how the final MBRT was constructed 5.13, we can see similar behavior when compared with the Kmeans application. For Nbody application, 35.9% of all split actions were done using the derived variable w_2. 18.1% used w_1. The percent of internal nodes using the number of workers as the partition variable is 13.9%. The specific workload of all the workers was used in 9.7% of all the cases.

Additionally, the affinity associated with the MPI processes is relevant information that is also included as partitioning criteria. This information was used in 3.49% of the internal nodes, mostly on the top level of the final tree.

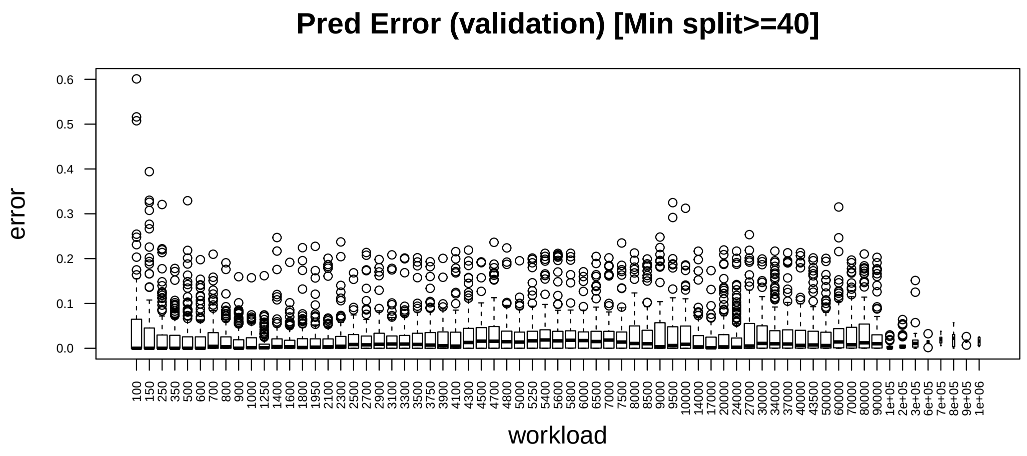

Figure 5.12: Relative error of efficiency index using the validation data set grouped by workload (Nbody) on NUMA.

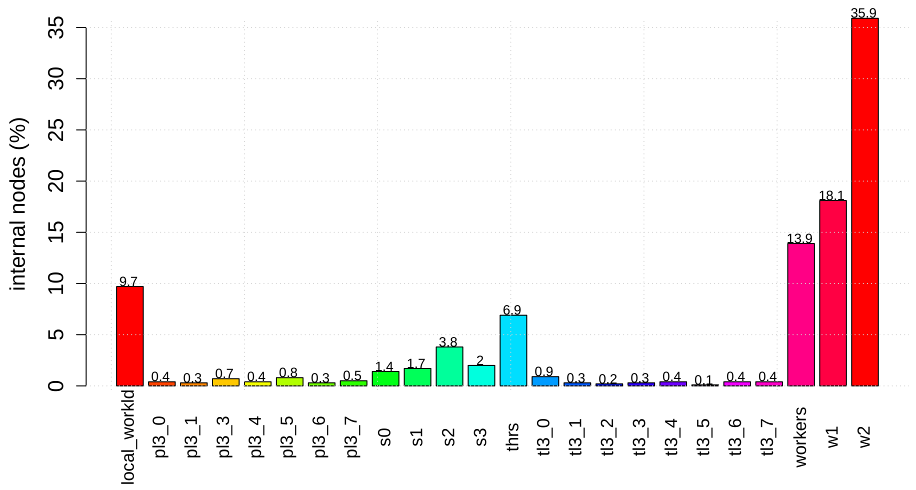

Figure 5.13: Distribution of internal nodes in the MBRT using Nbody on NUMA.

The number of threads was used in 6.9% of all the split actions and the thread affinity variables in 3% of all the internal nodes. In summary, in this application, the affinity variables have less importance in the final structure of the tree. Taking into account that, for the same number of computational resources, the difference in performance time between two configurations is no more than 20% of time, the low importance of affinity variables should be an expected result.

The bar plot for Heat transfer application shows a very regular behavior. The top quartiles of all the distributions is around 20% and the 75% of all predictions error is under 10%. The number of outliers is limited to 3%. Even when these distributions are slightly higher than previous applications, the median and upper quartiles are quite similar for all the workloads.

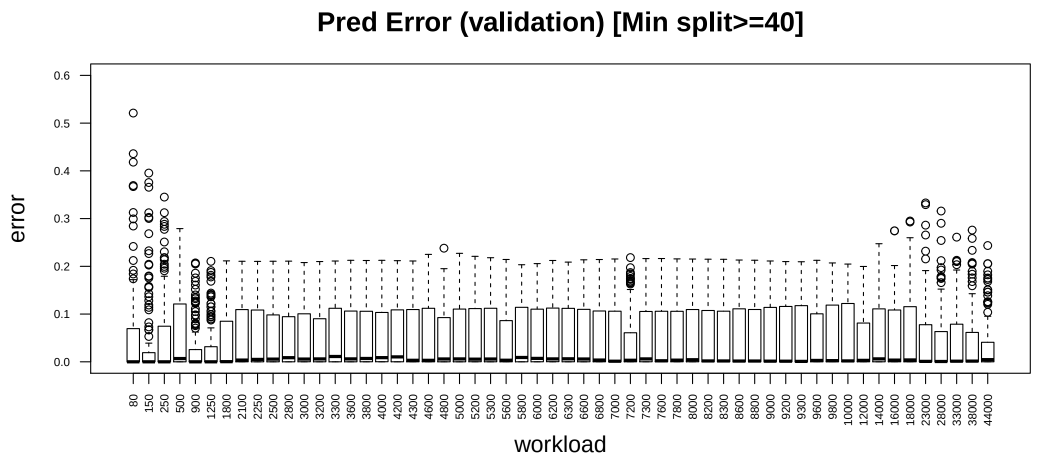

Figure 5.14: Relative error of efficiency index using the validation data set grouped by workload (Heat transfer) on NUMA.

The internal node structure of the tree that generates these predictive results is shown in Picture 5.15. There are 9.6% of the internal nodes using the local workload as the splitting variable. The derived variables w_2 and w_1 were used in 30.8% and 21% of all the cases, respectively. The number of workers used was 16.6%. The split actions using the process affinity variables are 10.4% of the internal nodes, while 3% are based on the number of processes sharing L3 of cache memory. Finally, the thread affinity defined by the $tl3$ variables were used in the 3.2% of the cases. Both groups of variables are located in the top region of the tree. Keeping all these results in mind, the next section will cover the results of these experiments when the best and predicted application configurations for efficiency index are compared.

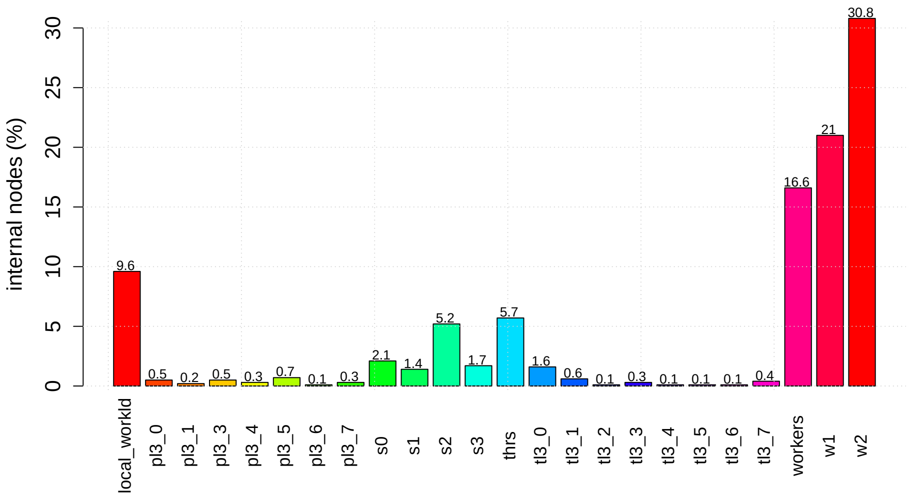

Figure 5.15: Distribution of internal nodes in the MBRT using Heat transfer on NUMA.

5.5.2 Best efficiency index configuration

Table 5.16 summarizes the results obtained by comparing the value of the minimum efficiency index for a given workload with the value predicted by the MBRT. The results for Kmeans applications show 6% of all the efficiency index predictions with a really high gap between them. 25% of all minimum efficiency index predictions are close to the real one, but suggest a different configuration of workers, threads and affinity. The worst predictions are caused by the outliers present in the error graph distribution. In these cases, the affinity suggested by the MBRT does not match the real best one. For workloads 700 and 900, the best configuration and the predicted one are the same: four processes per socket except in the last socket and the third level of cache memory shared by only two processes. The results of affinity in thread subjects the use of two threads per socket to sharing the cache memory L2. For predictions in the range of 1500 to 2100, the predictions subject four processes to use per NUMA socket with two threads per MPI process. The affinity selected in these cases subjects to use two processes sharing L3 but not L2 and two threads per process that do not share the cache at level two. For the range of 4100 to 4800, the affinity suggested is quite peculiar. The best highest level of efficiency is achieved using 26 processes with two threads per process. The MPI processes must have eight processes per socket, except the last socket that only has two, sharing the L3 in this case. Accordingly, the two threads per process have to share cache L2. In the other cases, the more efficient configuration parameters are mostly four processes sharing L3 with two threads sharing L2 or eight processes sharing L3, with only one thread per process. In summary, for this application using the NUMA architecture as described, there is a general tendency to achieve the highest efficiency using more MPI processes than threads. This is evidence that the communication penalty is less than the overhead of the parallel OpenMP regions. If we consider that, in this application, the size of MPI messages sent is very small, this behavior should not be surprising.

The prediction results using Nbody application are summarized in Table 5.17. The results are more accurate in this case. There are 10.4% predictions where the best application configuration do not match with the best configuration predicted. Even with these cases, the efficiency predicted value and the real one are very similar. There is only one prediction for the workload of 80000 that is very different from the best one. In the workload range from 250 to 1400, the real and the predicted best configuration are the same. The application parameters in this range have to be four workers per socket with 16 threads associated with each one. The MPI processes do not share L3 cache memory level, but the 8 thread will. The L2 cache is shared with a pair of threads. In the range of the workload between 1600 and 4700, the most common affinity is suggested in the one that used eight processes with eight threads per process. Each process has a complete L3 cache for its operation and does not share it with the others. The threads are mapped to each core in the range of eight assigned to each process. In this case, they share L2 in pairs and L3 cache is shared by all eight threads. There are only two cases in this range of workloads where the application parameters are different. For 2300, the best configuration is four workers with 16 threads, and, for a workload of 4100, the best configuration is 16 workers with four threads. In the first case, the workers do not share L3, and, the second case, the processes share L3 in groups of four processes. A similar situation can be detected in the range of workloads between 4800 and 7000. The most common configuration selected by the MBRT using a workload range between 7500 and 90000 is 32 workers with two threads. Only the L2 level of cache memory is not shared by the other processes for this configuration. In this application, we find a tendency to use more processes than threads in some cases, but, in other cases, it is just the contrary. The communication impact is greater in this application since messages are proportional to the workload. This is the reason why there are several best configuration selections with higher numbers of threads than workers.

The results achieved using the Heat transfer application are really accurate 5.18. There is only one suggested configuration that does not match the best efficiency exactly. This is the case of using a value of workload of 150. There are essentially four different configurations of affinity suggested in these results. In the range from 2500 to 9300, the output of the MBRT selects eight processes with eight threads per process as the best efficiency configuration. In this configuration, there are two process per socket and 16 threads per socket. The processes do not share the L3 cache, but the eight threads associated with each process share this memory level. Using workloads from 9600 to 23000, the best configuration is the one that uses 16 processes with four threads. The processes share L3 with only one other process, and the threads share L2 in twos. Using 32 workers with two threads for workloads for those workloads between 2800 to 44000 implies the use of eight processes per socket and two threads sharing the L2 cache memory. Discarding this prediction, the rest of the suggested configurations in the validation data set match the best efficiency configuration exactly.

Table 5.16: Real vs Prediction best efficiency index configuration (Kmeans) on NUMA.

Real				Prediction		
Workload	**Workers**	**Threads**	**Best pindex**	**Workers**	**Threads**	**Pred pindex**
100	4	4	0.00002	8	1	0.00003
200	6	4	0.00003	8	2	0.00003
300	6	4	0.00004	8	2	0.00004
500	10	2	0.00005	6	4	0.00006
700	12	2	0.00006	12	2	0.00006
900	12	2	0.00006	12	2	0.00006
1100	16	4	0.00007	12	2	0.00007
1500	16	2	0.00008	16	2	0.00008
1800	16	2	0.00009	16	2	0.00009
2100	16	2	0.0001	16	2	0.0001
2500	18	2	0.00011	16	2	0.00012
3000	20	2	0.00012	20	2	0.00012
3200	26	2	0.00012	14	4	0.00015
3800	32	2	0.00015	26	2	0.00016
4100	26	2	0.00014	26	2	0.00014
4500	26	2	0.00014	26	2	0.00014
4800	26	2	0.00015	26	2	0.00015
5000	24	2	0.00017	24	2	0.00017
7000	28	2	0.00019	28	2	0.00019
9000	30	2	0.00022	30	2	0.00022
12000	32	2	0.00028	32	2	0.00028
18000	32	2	0.00035	32	2	0.00035
20000	32	2	0.00042	32	2	0.00042
25000	32	2	0.00046	32	2	0.00046
28000	32	2	0.00051	32	2	0.00051
50000	32	2	0.00087	32	2	0.00087
90000	32	2	0.00154	32	2	0.00154
2e+05	64	1	0.00336	32	2	0.00339
4e+05	32	2	0.00666	64	1	0.00713
6e+05	64	1	0.00972	64	1	0.00972
8e+05	64	1	0.01325	64	1	0.01325
1e+06	64	1	0.01663	64	1	0.01663
1500000	64	1	0.02463	64	1	0.02463
2e+06	64	1	0.03269	64	1	0.03269
2500000	32	2	0.04095	32	2	0.04095
3e+06	64	1	0.04866	32	2	0.05438
3500000	32	2	0.05758	64	1	0.05968
4e+06	64	1	0.06616	32	2	0.06661
5500000	64	1	0.09064	64	1	0.09064
6e+06	64	1	0.10038	64	1	0.10038
6500000	32	2	0.10811	64	1	0.10854
7e+06	62	1	0.11942	64	1	0.12014
7500000	32	2	0.12393	64	1	0.12583
8e+06	64	1	0.1323	64	1	0.1323
8500000	62	1	0.1415	64	1	0.14263
9e+06	64	1	0.14771	64	1	0.14771
9500000	64	1	0.15504	64	1	0.15504
1e+07	64	1	0.16384	32	2	0.16564

Table 5.17: Real vs Prediction best efficiency index configuration (Nbody) on NUMA.

Real				Prediction		
Workload	**Workers**	**Threads**	**Best pindex**	**Workers**	**Threads**	**Pred pindex**
100	3	6	0.00003	3	10	0.00003
150	3	14	0.00003	3	14	0.00003
250	4	14	0.00005	4	14	0.00005
350	4	16	0.00008	4	16	0.00008
500	4	14	0.00013	4	14	0.00013
600	4	16	0.00013	4	16	0.00013
700	4	16	0.00016	4	16	0.00016
800	4	16	0.00022	4	16	0.00022
900	4	16	0.00027	4	16	0.00027
1000	4	16	0.0003	4	16	0.0003
1250	4	16	0.00043	4	16	0.00043
1400	4	16	0.00057	4	16	0.00057
1600	8	8	0.00067	8	8	0.00067
1800	8	8	0.00083	8	8	0.00083
1950	8	8	0.00097	8	8	0.00097
2100	8	8	0.0011	8	8	0.0011
2300	4	16	0.00133	4	16	0.00133
2500	8	8	0.00148	4	16	0.00164
2700	8	8	0.0017	8	8	0.0017
2900	8	8	0.00188	8	8	0.00188
3100	8	8	0.00217	8	8	0.00217
3300	8	8	0.0024	8	8	0.0024
3500	8	8	0.0027	8	8	0.0027
3750	8	8	0.00303	8	8	0.00303
3900	8	8	0.00328	8	8	0.00328
4100	16	4	0.00363	16	4	0.00363
4300	8	8	0.00398	8	8	0.00398
4500	8	8	0.00429	8	8	0.00429
4700	8	8	0.00463	8	8	0.00463
4800	16	4	0.00477	16	4	0.00477
5000	16	4	0.00518	8	8	0.00522
5250	16	4	0.00567	16	4	0.00567
5400	16	4	0.00596	16	4	0.00596
5600	16	4	0.00641	16	4	0.00641
5800	16	4	0.00686	32	2	0.00717
6000	16	4	0.00729	16	4	0.00729
6500	16	4	0.00843	16	4	0.00843
7000	16	4	0.00967	16	4	0.00967
7500	32	2	0.01123	16	4	0.01203
8000	32	2	0.01267	32	2	0.01267
8500	32	2	0.0142	32	2	0.0142
9000	16	4	0.01555	16	4	0.01555
9500	32	2	0.0175	8	8	0.01791
10000	16	4	0.01911	16	4	0.01911
14000	32	2	0.03641	32	2	0.03641
17000	32	2	0.05297	32	2	0.05297
20000	32	2	0.07277	32	2	0.07277
24000	32	2	0.10377	32	2	0.10377
27000	32	2	0.13052	32	2	0.13052
30000	32	2	0.16063	32	2	0.16063
34000	32	2	0.20678	32	2	0.20678
37000	32	2	0.24436	32	2	0.24436
40000	16	4	0.293	16	4	0.293
43500	32	2	0.3365	16	4	0.3583
50000	32	2	0.44602	32	2	0.44602
60000	32	2	0.64369	30	2	0.68689
70000	32	2	0.89041	32	2	0.89041
80000	16	4	1.17575	14	4	1.36039
90000	32	2	1.46538	16	4	1.49009
1e+05	8	8	1.93545	8	8	1.93545
2e+05	8	8	2.72845	8	8	2.72845
3e+05	8	8	3.47865	8	8	3.47865
6e+05	4	16	4.04516	4	16	4.04516
7e+05	4	16	5.15	4	16	5.15
8e+05	4	16	6.4225	4	16	6.4225
9e+05	4	16	7.9944	4	16	7.9944
1e+06	4	16	9.5179	4	16	9.5179

Table 5.18: Real vs Prediction best efficiency index configuration (Heat transfer) on NUMA.

	Real			**Prediction**		
Workload	**Workers**	**Threads**	**Best pindex**	**Workers**	**Threads**	**Pred pindex**
80	3	4	0.00001	3	4	0.00001
150	3	14	0.00001	3	8	0.00001
250	3	16	0.00002	3	16	0.00002
500	4	16	0.00003	4	16	0.00003
900	4	16	0.00008	4	16	0.00008
1250	4	16	0.00015	4	16	0.00015
1800	4	16	0.00029	4	16	0.00029
2100	4	16	0.00038	4	16	0.00038
2250	4	16	0.00044	4	16	0.00044
2500	8	8	0.00053	8	8	0.00053
2800	8	8	0.00064	8	8	0.00064
3000	8	8	0.00073	8	8	0.00073
3200	8	8	0.00082	8	8	0.00082
3300	8	8	0.00088	8	8	0.00088
3600	8	8	0.00104	8	8	0.00104
3800	8	8	0.00113	8	8	0.00113
4000	8	8	0.00126	8	8	0.00126
4200	8	8	0.00135	8	8	0.00135
4300	8	8	0.00142	8	8	0.00142
4600	8	8	0.00161	8	8	0.00161
4800	8	8	0.00175	8	8	0.00175
5000	8	8	0.00189	8	8	0.00189
5200	8	8	0.00203	8	8	0.00203
5300	8	8	0.00211	8	8	0.00211
5600	8	8	0.0023	8	8	0.0023
5800	8	8	0.00248	8	8	0.00248
6000	8	8	0.00265	8	8	0.00265
6200	8	8	0.00281	8	8	0.00281
6300	8	8	0.0029	8	8	0.0029
6600	8	8	0.00318	8	8	0.00318
6800	8	8	0.00336	8	8	0.00336
7000	8	8	0.00353	8	8	0.00353
7200	8	8	0.00375	8	8	0.00375
7300	8	8	0.00384	8	8	0.00384
7600	8	8	0.00414	8	8	0.00414
7800	8	8	0.00437	8	8	0.00437
8000	8	8	0.00461	8	8	0.00461
8200	8	8	0.00484	8	8	0.00484
8300	8	8	0.00493	8	8	0.00493
8600	8	8	0.00527	8	8	0.00527
8800	8	8	0.00551	8	8	0.00551
9000	8	8	0.00576	8	8	0.00576
9200	8	8	0.00601	8	8	0.00601
9300	8	8	0.00611	8	8	0.00611
9600	16	4	0.0065	16	4	0.0065
9800	16	4	0.00676	16	4	0.00676
10000	8	8	0.00706	8	8	0.00706
12000	16	4	0.00978	16	4	0.00978
14000	16	4	0.01307	16	4	0.01307
16000	16	4	0.01703	16	4	0.01703
18000	16	4	0.02123	16	4	0.02123
23000	16	4	0.03408	16	4	0.03408
28000	32	2	0.05002	32	2	0.05002
33000	32	2	0.06886	32	2	0.06886
38000	32	2	0.09046	32	2	0.09046
44000	32	2	0.12059	32	2	0.12059

5.6 Discussions

In this section, we present the experimental results achieved for three Master/Worker applications using two different architectures. First, the prediction for performance and the best configuration for this goal were presented using a homogeneous cluster. The second group of results shows the prediction efficiency accuracy and the results achieved, trying to suggest the best application parameters to reach the optimal value of efficiency index. In summary, there is a strong correlation between the error distribution and the selection of the best configuration that suggests that, if it is possible to reduce the error performance and the amount of outlier predictions by increasing the training data set with more samples on the critical region, the application parameters prediction will not be far from the best configurations.

Additionally, the results show that even when the predictions for performance are well accurate, the results achieved for predicting efficiency index are even better. This is an encouraging result because we could achieve maximum efficiency without sacrificing too much performance using the configuration predicted by the MBRT.

Chapter 6

Conclusions and Open Lines

6.1 Conclusions

We began our work by studying the different proposals for performance predictions on hybrid MPI/OpenMP applications. An important aspect of this study was to identify the main advantages and drawbacks of each method and the particular scope that each has. In general, the main disadvantage that we detected is that they are not generalized to prediction for any workload (in some cases) and do not go further in trying to evaluate each method to suggest a suitable parameter configuration of the application. There are some proposals where the prediction is based on simulation using the memory access pattern trace file, but, in this case, it is more appropriate to be used for predicting performance on architecture that we do not have. The cost of this simulation is really high, which is why it is not applicable for application tuning at runtime.

As a first approach, we develop a performance model using an analytic expression. The model tries to model the communication time and the overhead of OpenMP regions using the information provided by two benchmarks. To predict the data cache miss penalty, time on the last level and the tlb miss time, the model applies two regression functions using performance behavior observations. This model can be accurate for performance prediction where the workload is around 15% different from the previous one, but, if the gap is higher, the prediction error will not be small enough to be used in the search for the best configuration parameters of the application. The main weakness of this model is the assumption that there are unique values for each parameter of the two regression functions. The actual behavior suggests that these parameters change according to the parameters the application executed.

The most important approach we present is the use of the model-based regression tree to predict the execution time and the efficiency for an iteration of Master/Worker applications. The study is also extended to predicting the best configuration parameters of the application. The model is tested using three applications on two different architectures, obtaining for all the cases a globally mean error lower than five percent. As we achieve a small error in the performance prediction, we can determine dynamically the adequate number of Workers and threads in order to reach the best possible performance. There are a few restrictions prior to applying this method: applications used must have a regular computation region that does not depend on the data values. As this proposal tries to model the iteration time in fragments for the different performance behavior zones of the application parameter space, it can be applied to those applications without too much variability in the behavior of communications. In other words, the dispersion of the communication time has to be not too far from the mean in order to achieve good prediction accuracy. Additionally, there is another strong restriction: the application has to be well-balanced previously to guarantee that the processing execution time is similar among all the workers. This is an important topic because, if it is not guaranteed, the worker does not finish its task in the same order as they received it. As the workload is being used on each regression function as an input variable, this method can only be applied for those applications where the workload can be characterized.

The method returns good global results without investing too much effort in identifying the general regression function and splitting parameters, but it does not guarantee always the best prediction. There are two main causes that explain that result: first, the errors are located mainly in regions where the prediction errors are higher using the training set, and, second, the final prediction was an outlier from the MBRT. This suggests that, if it is possible to reduce the performance error and the amount of outlier predictions by increasing the training data set with more samples on the critical region, the application parameter prediction will not be far from the best configuration.

6.2 Open Lines

The work presented in this thesis will allow for further investigation in several directions. First, it would be nice to validate the technique in a bigger homogeneous cluster with more cores embedded inside each node. This will give an idea of how powerful this technique is for prediction in large-scale systems. In addition, it will allow us to evaluate the prediction accuracy using more computational resources. In this architectures, the communication time will penalize the performance of the application even more and behavior will allow us to increase the study to higher values of the application workload. Further studies about the influence in the final prediction of the min split arguments will benefit the final results. In this case,

This research can be extended to the prediction of SPMD applications. In these cases, the general regression prototype has to be modified in order to model the collective communications that are typically used by this kind of applications. There is a lot of bibliography covering how to model collective communication, so adjusting this technique to this case will not require not much effort.

The library selected for applying the MBRT has some important disadvantages in its implementations. First, it does not allow the inclusion of other kinds of objective functions like relative error as the objective function within regression algorithm. A second problem is related to information that is lost when the tree is constructed. In a few terminal nodes, there are some arguments β_i for the regression function with undefined values. Any prediction that used this nodes will generate also an undefined output. The algorithm discards these results because they do not offer any valuable information. A great future work will be to recode this module of the library to fix these two bugs. During our work, we stay in touch with the authors to suggest all of these potential improvements to them.

At the end, it will be important to include a pruning logic for the result tree. Even when we can control the over-fitting problem by using an appropriate value of minsplit argument, it is important to cut some terminal nodes based on the idea that the effect of this operation does not increase the general error distribution.

6.3 List of publications

The work and results for this thesis have been published or have been under revision in the following papers:

1. **A. Castellanos, A. Moreno, J Sorribes, T. Margalef. Factores de rendimiento en aplicaciones híbridas (MPI+OpenMP) in XXIII Jornadas de Paralelismo SARTECO 2012.** [87]
   This paper was the starting point for this research. First, we study the impact of all possible performance factors on hybrid applications and try to find those which are candidates to be included in performance model or other modelation techniques. Basically, this study was a revision of all the publications about this topic with experimentals results that validate each performance factor proposal.

2. **A. Castellanos, A. Moreno, J Sorribes, T. Margalef. Performance model for Master/Worker hybrid applications in Proceedings The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2013). pp. 210-217.** [88]
   In this paper, we present a naive performance model for hybrid Master/Worker application. The model defines the computation part of the analytic expression as a fraction of the serial execution time divided by

the number of computational resources. This was the main weakness of this proposal. It does not include

3. **A. Castellanos, A. Moreno, J Sorribes, T. Margalef. Performance model for Master/Worker hybrid applications on multicore clusters in Proceedings IEEE 15th International Conference on High Performance Computing and Communication (HPCC 2013). 2013.** [77]
This proposal was an extension of the previous performance model. In this version, we modify the expression to add the time penalty of data cache misses and TLB misses produced by recurrent access to the main memory in parallel OpenMP regions. In order to calculate the arguments of each regression function, we propose using some iteration of the application to take measurements of hardware counters. The model can predict the performance very accurately if the workload that we are predicting is close to the workloads that have been used to calculate the arguments in the regression functions and the overlap variables. Unfortunately, this approach can-not be generalized to predictions that use workloads significantly different from the previous one, because we detect high variation in the parameters of the model when the workload varies drastically.

4. **A. Castellanos, A. Moreno, J Sorribes, T. Margalef. Predicting performance of hybrid Master/Worker applications using model-based regression trees in Proceedings IEEE 16th International Conference on High Performance Computing and Communication (HPCC 2014). 2014. (to be published)** [89]
This work presents the results we achieved using the model-based regression techniques on homogeneous cluster. We test the proposal using two hybrid applications. As an extension to our work, we evaluated the precision of this technique to searching for the best performance parameter configuration.

5. **A. Castellanos, A. Moreno, J Sorribes, T. Margalef. Efficient execution of hybrid Master/Workers applications using model-based regression trees in Proceedings 26th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2014). 2014. (Submitted)** [90]
   This work presents the result we achieve using the Model-based regression techniques for predicting efficiency on parallel Master/Worker applications. In addition, the results for searching the best application parameters were presented. The proposal is validated using three applications.

Bibliography

[1] Michael Flynn. Some computer organizations and their effectiveness . In: *Computers, IEEE Transactions on* 100.9 (1972), pp. 948 960.

[2] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2012.

[3] J. Dongarra, T. Sterling, H. Simon, and E. Strohmaier. High-performance computing: clusters, constellations, MPPs, and future directions . In: *Computing in Science Engineering* 7.2 (2005), pp. 51 59. ISSN: 1521-9615. DOI: `10.1109/MCSE.2005.34`.

[4] *TOP500 List of best supercomputers*. Accessed: 2014-04-27. 2014. URL: `www.top500.org`.

[5] Rolf Hempel. The MPI standard for message passing . In: *High-Performance Computing and Networking*. Ed. by Wolfgang Gentzsch and Uwe Harms. Vol. 797. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1994, pp. 247 252. ISBN: 978-3-540-57981-6.

[6] M. Sato. OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors . In: *System Synthesis, 2002. 15th International Symposium on*. 2002, pp. 109 111.

[7] L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming . In: *Computational Science Engineering, IEEE* 5.1 (1998), pp. 46 55.

[8] CUDA Nvidia. *Programming guide*. 2008.

[9] Jesus Labarta. StarSS: A programming model for the multicore era . In: *PRACE Workshop'New Languages & Future Technology Prototypes' at the Leibniz Supercomputing Centre in Garching (Germany)*. 2010.

[10] R. Rabenseifner, G. Hager, and G. Jost. Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes . In: *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*. 2009, pp. 427 436. DOI: `10.1109/PDP.2009.43`.

[11] Ronal Muresano, Dolores Rexachs, and Emilio Luque. Methodology for Efficient Execution of SPMD Applications on Multicore Environments . In: *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. 2010, pp. 185 195. DOI: `10.1109/CCGRID.2010.67`.

[12] Sameer S. Shende and Allen D. Malony. The Tau Parallel Performance System . In: *Int. J. High Perform. Comput. Appl.* 20.2 (2006), pp. 287 311. ISSN: 1094-3420. URL: `http://dx.doi.org/10.1177/1094342006064482`.

[13] *TAU - Tuning and Analysis Utilities*. Accessed: 2014-04-27. URL: `http://www.cs.uoregon.edu/research/tau`.

[14] Bryan Buck and Jeffrey K. Hollingsworth. An API for Runtime Code Patching . In: *Int. J. High Perform. Comput. Appl.* 14.4 (2000), pp. 317 329. ISSN: 1094-3420. URL: `http://dx.doi.org/10.1177/109434200001400404`.

[15] Kathleen A. Lindlan, Janice Cuny, Allen D. Malony, Sameer Shende, Forschungszentrum Juelich, Reid Rivenburgh, Craig Rasmussen, and Bernd Mohr. A Tool Framework for Static and Dynamic Analysis of Object-oriented Software with Templates . In: *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*. SC 00. Dallas, Texas, USA: IEEE Computer Society, 2000. ISBN: 0-7803-9802-5. URL: `http://dl.acm.org/citation.cfm?id=370049.370456`.

[16] Ewing Lusk, S Huss, B Saphir, and M Snir. *MPI: A message-passing interface standard*. 2009.

[17] Bernd Mohr, Allen D. Malony, Sameer Shende, and Felix Wolf. Towards a Performance Tool Interface for OpenMP: An Approach Based on Directive Rewriting . In: *In Proceedings of the Third Workshop on OpenMP (EWOMP'01.* 2001.

[18] Shirley Browne, Jack Dongarra, Nathan Garner, George Ho, and Philip Mucci. A portable programming interface for performance evaluation on modern processors . In: *International Journal of High Performance Computing Applications* 14.3 (2000), pp. 189 204.

[19] Anthony Chan, William Gropp, and Ewing Lusk. An efficient format for nearly constant-time access to arbitrary time intervals in large trace files . In: *Scientific Programming* 16.2 (2008), pp. 155 165.

[20] Andreas Knupfer, Ronny Brendel, Holger Brunst, Hartmut Mix, and Wolfgang E Nagel. Introducing the open trace format (OTF) . In: *Computational Science–ICCS 2006.* Springer, 2006, pp. 526 533.

[21] Bernd Mohr and Felix Wolf. KOJAK A tool set for automatic performance analysis of parallel programs . In: *Euro-Par 2003 Parallel Processing.* Springer, 2003, pp. 1301 1304.

[22] Omer Zaki, Ewing Lusk, William Gropp, and Deborah Swider. Toward scalable performance visualization with Jumpshot . In: *International Journal of High Performance Computing Applications* 13.3 (1999), pp. 277 288.

[23] Markus Geimer, Felix Wolf, Brian JN Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The Scalasca performance toolset architecture . In: *Concurrency and Computation: Practice and Experience* 22.6 (2010), pp. 702 719.

[24] *Scalasca.* Accessed: 2014-04-27. URL: `http://www.scalasca.org/`.

[25] *Barcelona Supercomputing Center.* Accessed: 2014-04-27. URL: `http://www.bsc.es`.

[26] Jesús Labarta, Sergi Girona, Vincent Pillet, Toni Cortes, and Luis Gregoris. DiP: A parallel program development environment . In: *Euro-Par'96 Parallel Processing.* Springer. 1996, pp. 665 674.

[27] *Active Harmony.* Accessed: 2014-04-27. URL: `http://www.dyninst.org/harmony`.

[28] Ananta Tiwari and Jeffrey K Hollingsworth. Online adaptive code generation and tuning . In: *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International.* IEEE. 2011, pp. 879 892.

[29] Jeffrey K Hollingsworth and Peter J Keleher. Prediction and adaptation in active harmony . In: *Cluster Computing* 2.3 (1999), pp. 195 205.

[30] Cristian Ţăpuş, I-Hsin Chung, Jeffrey K Hollingsworth, et al. Active harmony: Towards automated performance tuning . In: *Proceedings of the 2002 ACM/IEEE conference on Supercomputing.* IEEE Computer Society Press. 2002, pp. 1 11.

[31] *Periscope Performance Measurement Toolkit.* Accessed: 2014-04-27. URL: `http://www.lrr.in.tum.de/~periscop/`.

[32] Renato Miceli, Gilles Civario, Anna Sikora, Eduardo César, Michael Gerndt, Houssam Haitof, Carmen Navarrete, Siegfried Benkner, Martin Sandrieser, Laurent Morin, et al. AutoTune: a plugin-driven approach to the automatic tuning of parallel applications . In: *Applied Parallel and Scientific Computing.* Springer, 2013, pp. 328 342.

[33] *HPCToolkit.* Accessed: 2014-04-27. URL: `http://hpctoolkit.org/`.

[34] Gabriel Marin and John Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models . In: *ACM SIGMETRICS Performance Evaluation Review* 32.1 (2004), pp. 2 13.

[35] Eduardo César Andrea Martínez Anna Sikora and Joan Sorribes. ELASTIC: A Large Scale Dynamic Tuning Environment . In: *The special issue on Automatic Performance Tuning for HPC Architectures* (). Accepted.

[36] R. Clint Whaley and Jack J. Dongarra. Automatically Tuned Linear Algebra Software . In: *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing.* SC 98. San Jose, CA: IEEE Computer Society, 1998, pp. 1 27. ISBN: 0-89791-984-X. URL: `http://dl.acm.org/citation.cfm?id=509058.509096`.

[37] Sadaf R Alam and Jeffrey S Vetter. A framework to develop symbolic performance models of parallel applications . In: *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International.* IEEE. 2006, 8 pp.

[38] Eric Grobelny, David Bueno, Ian Troxel, Alan D George, and Jeffrey S Vetter. FASE: A framework for scalable performance prediction of HPC systems and applications . In: *Simulation* 83.10 (2007), pp. 721 745.

[39] Roger W Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2 . In: *Parallel computing* 20.3 (1994), pp. 389 398.

[40] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. *LogP: Towards a realistic model of parallel computation.* Vol. 28. 7. ACM, 1993.

[41] Kimberley Keeton, Thomas Anderson, and David Patterson. LogP quantified: The case for low-overhead local area networks . In: *Proc. 1995 Hot Interconnects.* 1995.

[42] Thilo Kielmann, Henri E Bal, and Sergei Gorlatch. Bandwidth-efficient collective communication for clustered wide area systems . In: *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International.* IEEE. 2000, pp. 492 499.

[43] Albert Alexandrov, Mihai F Ionescu, Klaus E Schauser, and Chris Scheiman. LogGP: incorporating long messages into the LogP model one step closer towards a realistic model for parallel computation . In: *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures.* ACM. 1995, pp. 95 105.

[44] Matthew I Frank, Anant Agarwal, and Mary K Vernon. *LoPC: modeling contention in parallel algorithms.* Vol. 32. 7. ACM, 1997.

[45] Csaba Andras Moritz and Matthew I Frank. LoGPC: Modeling network contention in message-passing programs . In: *ACM SIGMETRICS Performance Evaluation Review.* Vol. 26. 1. ACM. 1998, pp. 254 263.

[46] Fumihiko Ino, Noriyuki Fujimoto, and Kenichi Hagihara. LogGPS: a parallel computational model for synchronization analysis . In: *ACM SIGPLAN Notices.* Vol. 36. 7. ACM. 2001, pp. 133 142.

[47] E. Cesar, A. Moreno, J. Sorribes, and E. Luque. Modeling Master/-Worker applications for automatic performance tuning . In: *Parallel Computing* 32.7-8 (2006), pp. 568 589. ISSN: 0167-8191. DOI: `10.1016/j.parco.2006.06.005`. URL: `http://www.sciencedirect.com/science/article/pii/S0167819106000263`.

[48] A. Moreno, E. Cesar, J. Sorribes, T. Margalef, and E. Luque. Task distribution using factoring load balancing in Master Worker applications . In: *Information Processing Letters* 109.16 (2009), pp. 902 906. ISSN: 0020-0190. DOI: `http://dx.doi.org/10.1016/j.ipl.2009.04.014`. URL: `http://www.sciencedirect.com/science/article/pii/S0020019009001434`.

[49] A. Guevara, E. Cesar, J. Sorribes, T. Margalef, E. Luque, and A. Moreno. A Performance Tuning Strategy for Complex Parallel Application . In: *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on.* 2010, pp. 103 110.

[50] Anna Sikora Cesar Allande Josep Jorba and Eduardo Cesar. A Performance Model for OpenMP Memory Bound Applications in Multisocket System . In: *The International Conference on Computational Science (ICCS 2014).* 2014, to be published.

[51] Diego Rodríguez Martínez. Modelado Analítico del rendimiento de Aplicaciones en sistemas paralelos . PhD thesis. Universidad de San-

tiago de Compostela, 2011. URL: `www.ac.usc.es/system/files/DiegoRM_tesis_17x24.pdf`.

[52] Yosiyuki Sakamoto, Makio Ishiguro, and Genshiro Kitagawa. Akaike information criterion statistics . In: *Dordrecht, The Netherlands: D. Reidel* (1986).

[53] Vincent Calcagno and Claire de Mazancourt. glmulti: an R package for easy automated model selection with (generalized) linear models . In: *Journal of Statistical Software* 34.12 (2010), pp. 1 29.

[54] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture . In: *Proceedings of the 11th International Symposium on High-Performance Computer Architecture.* HPCA 05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 340 351. ISBN: 0-7695-2275-0. DOI: `10.1109/HPCA.2005.27`. URL: `http://dx.doi.org/10.1109/HPCA.2005.27`.

[55] Chi Xu, Xi Chen, R.P. Dick, and Z.M. Mao. Cache contention and application performance prediction for multi-core systems . In: *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on.* 2010, pp. 76 86.

[56] Rahman Hassan, Antony Harris, Nigel Topham, and Aris Efthymiou. A Hybrid Markov Model for Accurate Memory Reference Generation. In: *IMECS.* 2007, pp. 550 555.

[57] Rosa M Badia, Jess Labarta, Judit Gimenez, and Francesc Escale. DIMEMAS: Predicting MPI applications behavior in Grid environments . In: *Workshop on Grid Applications and Programming Tools (GGF8).* Vol. 86. 2003, pp. 52 62.

[58] Rob Knauerhase, Paul Brett, Barbara Hohlt, Tong Li, and Scott Hahn. Using OS observations to improve performance in multicore systems . In: *IEEE micro* 28.3 (2008), pp. 54 66.

[59] Andreas Merkel, Jan Stoess, and Frank Bellosa. Resource-conscious scheduling for energy efficiency on multicore processors . In: *Proceedings of the 5th European conference on Computer systems.* ACM. 2010, pp. 153 166.

[60] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing shared resource contention in multicore processors via scheduling . In: *ACM SIGARCH Computer Architecture News.* Vol. 38. 1. ACM. 2010, pp. 129 142.

[61] Sergey Blagodurov, Sergey Zhuravlev, Alexandra Fedorova, and Ali Kamali. A case for NUMA-aware contention management on multicore systems . In: *Proceedings of the 19th international conference on Parallel architectures and compilation techniques.* ACM. 2010, pp. 557 558.

[62] Benjamin C. Lee, David M. Brooks, Bronis R. de Supinski, Martin Schulz, Karan Singh, and Sally A. McKee. Methods of inference and learning for performance modeling of parallel applications . In: *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming.* PPoPP 07. San Jose, California, USA: ACM, 2007, pp. 249 258. ISBN: 978-1-59593-602-8. DOI: 10.1145/1229428.1229479. URL: http://doi.acm.org/10.1145/1229428.1229479.

[63] Tyler Dwyer, Alexandra Fedorova, Sergey Blagodurov, Mark Roth, Fabien Gaud, and Jian Pei. A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads . In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.* IEEE Computer Society Press. 2012, p. 83.

[64] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update . In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10 18.

[65] L. M. Liebrock and S. P. Goudy. Methodology for modelling SPMD hybrid parallel computation . In: *Concurrency and Computation: Practice and Experience* 20.8 (2008), pp. 903 940. ISSN: 1532-0634. DOI: 10.1002/cpe.1214. URL: http://dx.doi.org/10.1002/cpe.1214.

[66] William Gropp and Ewing Lusk. Reproducible Measurements of MPI Performance Characteristics . In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Ed. by Jack Dongarra, Emilio Luque, and Tomàs Margalef. Vol. 1697. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1999, pp. 681 681.

[67] Nor Asilah Wati Abdul Hamid and Paul Coddington. Comparison of MPI Benchmark Programs on Shared Memory and Distributed Memory Machines (Point-to-Point Communication) . In: *Int. J. High Perform. Comput. Appl.* 24.4 (2010), pp. 469 483. ISSN: 1094-3420. DOI: 10.1177/1094342010371106. URL: http://dx.doi.org/10.1177/1094342010371106.

[68] Duncan Grove and Paul Coddington. Precise MPI Performance Measurement Using MPIBench . In: *In Proceedings of HPC Asia*. 2001.

[69] Patrick Bridges, Nathan Doss, William Gropp, Edward Karrels, Ewing Lusk, and Anthony Skjellum. User s Guide to MPICH, a Portable Implementation of MPI . In: *Argonne National Laboratory* 9700 (1995), pp. 60439 4801.

[70] J. Mark Bull and Darragh O Neill. A Microbenchmark Suite for OpenMP 2.0 . In: *In Proceedings of the Third Workshop on OpenMP (EWOMP01)*. 2001, pp. 41 48.

[71] Carl Staelin. lmbench: an extensible micro-benchmark suite . In: *Software: Practice and Experience* 35.11 (2005), pp. 1079 1105.

[72] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters . In: *Supercomputing, ACM/IEEE 2000 Conference*. 2000, pp. 42 42. DOI: 10.1109/SC.2000.10029.

[73] Luís Fernando Raínho Alves Torgo. Inductive Learning of Tree-Based Regression models . PhD thesis. Universidade do Porto, 1999. URL: `http://www.dcc.fc.up.pt/~ltorgo/PhD/thesis.ps.gz`.

[74] Achim Zeileis, Torsten Hothorn, and Kurt Hornik. party with the mob Model-Based Recursive Partitioning in R . In: *Institute for Statistics and Mathematics WU Wirtschaftsuniversitat Wien, nd Web* 20 (2012).

[75] Michael J Crawley. *The R book*. John Wiley and Sons, 2012.

[76] Achim Zeileis and Kurt Hornik. Generalized M-fluctuation tests for parameter instability . In: *Statistica Neerlandica* 61.4 (2007), pp. 488 508.

[77] Abel Castellanos, Andreu Moreno, Joan Sorribes, and Tomàs Margalef. Performance model for Master/Worker hybrid applications on multi-core clusters . In: *High Performance Computing and Communication (HPCC 2013), IEEE 15th International Conference on*. IEEE. 2013, pp. 210 217. ISBN: 978-0-7695-5088-6/13. DOI: `10.1109/HPCC.and.EUC.2013.39`.

[78] Terry Therneau, Beth Atkinson, and Brian Ripley. *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-8. 2014. URL: `http://CRAN.R-project.org/package=rpart`.

[79] Achim Zeileis, Torsten Hothorn, and Kurt Hornik. Model-Based Recursive Partitioning . In: *Journal of Computational and Graphical Statistics* 17.2 (2008), pp. 492 514.

[80] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased Recursive Partitioning: A Conditional Inference Framework . In: *Journal of Computational and Graphical Statistics* 15.3 (2006), pp. 651 674. DOI: `10.1198/106186006X133933`. eprint: `http://www.tandfonline.com/doi/pdf/10.1198/106186006X133933`. URL: `http://www.tandfonline.com/doi/abs/10.1198/106186006X133933`.

[81] *NIST/SEMATECH e-Handbook of Statistical Methods*. Accessed: 2014-04-27. 2014. URL: `http://www.itl.nist.gov/div898/handbook/pmd/section4/pmd452.htm`.

[82] Lars Peter Hansen. Large sample properties of generalized method of moments estimators . In: *Econometrica Journal of the Econometric Society* (1982), pp. 1029 1054.

[83] Chia-Shang James Chu, Kurt Hornik, and Chung-Ming Kuan. The moving-estimates test for parameter stability . In: *Econometric Theory* 11.04 (1995), pp. 699 720.

[84] *NU-MineBench data mining benchmark suite.* Accessed: 2014-04-27. URL: `http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html`.

[85] *Astrophysical N-body Simulation Tools.* Accessed: 2014-04-27. URL: `http://sourceforge.net/projects/nbody/`.

[86] *Modeling Heat Transfer in Parallel.* Accessed: 2014-04-27. URL: `http://www.cas.usf.edu/~cconnor/parallel/2dheat/2dheat.html`.

[87] Andreu Moreno Abel Castellanos and Tomàs Margalef. Factores de rendimiento en aplicaciones híbridas (MPI+OpenMP) . In: *in XXIII Jornadas de Paralelismo SARTECO 2012.* 2012, pp. 210 217. URL: `www.jornadassarteco.org/programa-jp2012/js2012/papers/paper_117.pdf`.

[88] Abel Castellanos, Andreu Moreno, Joan Sorribes, and Tomàs Margalef. Performance model for Master/Worker hybrid applications . In: *Proceedings of The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2013).* CSREA Press, 2013, pp. 365 371. ISBN: 1-60132-256-9.

[89] Abel Castellanos, Andreu Moreno, Joan Sorribes, and Tomàs Margalef. Predicting performance of hybrid Master/Worker applications using model-based regression trees . In: *High Performance Computing and Communication (HPCC 2014), IEEE 16th International Conference on.* (Accepted). 2014.

[90] Abel Castellanos, Andreu Moreno, Joan Sorribes, and Tomàs Margalef. Efficient execution of hybrid Master/Workers applications using model-based regression trees . In: *26th International Symposium on Computer Architecture and High Performance Computing*. (Submitted). 2014.