UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Detecting Thread Clusters in High-Performance Computing Applications

by

Isaac Boixaderas Coderch

Director: Petar Radojković (Barcelona Supercomputing Center (BSC))
Ponent: Xavier Martorell Bofill (Arquitectura de Computadors, UPC)

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
Degree in Computer Science
Specialization in Computing

25 May 2015

Abstract

This project proposes a way of detecting whether significant differences among the threads involved in an execution of a high-performance computing (HPC) application exist, as well as an efficient algorithm for clustering the threads based on such differences.

Each thread has numerous measurements corresponding to different hardware counter events such as number of instructions, clock cycles or L1 cache misses.

Methods were proposed that calculate the differences among the threads using the Kolmogorov-Smirnov test statistic as a value of similarity in order to establish a degree of divergence between every pair of threads.

After establishing a threshold that indicates whether threads belong or not to the same cluster, the grouping algorithms were applied.

Three clustering algorithms (leader algorithm, K-medoids and CLARANS) were compared in order to evaluate which one better fulfilled the project objectives, being thus a fast and robust algorithm.

Tests showed that the algorithm with the best performance regarding clustering quality is CLARANS, although it requires some more computational time than the leader algorithm, but less than K-medoids.

Resum

Aquest projecte proposa una manera de detectar si existeixen diferències significatives entre els threads involucrats en l'execució d'una aplicació de "hihg-performance computing (HPC)", així com també un algorisme eficient per agrupar els threads en funció de les seves diferències.
Es van realitzar vàries mesures per a cada thread, corresponents a diferents "hardware counter events" com per exemple: nombre d'instruccions, cicles de rellotge o fallades de cache a L1.
Es proposen mètodes que calculen les diferències entre els threads utilitzant l'estadístic del test de Kolmogorov-Smirnov com a valor de similitud per tal d'establir cert grau de divergència entre cada parell de threads.
Després d'establir un llindar que indica quan els threads corresponen al mateix grup, els algorismes d'agrupament es van aplicar.
Es van comparar tres algorismes de "clustering" (l'algorisme del líder, K-medoids i CLARANS) per tal d'avaluar quin és el que millor compleix els objectius del projecte, és a dir, que sigui un algorisme ràpid i robust.
Els tests van mostrar que el que ofereix un millor rendiment en quan a qualitat dels grups és el CLARANS, tot i que requereix més temps de computació que l'algorisme del líder, però menys que el K-medoids.

Resumen

Este proyecto propone una manera de detectar si existen diferencias significativas entre los threads involucrados en la ejecución de una aplicación de "high-performance computing (HPC)", así como un algoritmo eficiente para agrupar los threads en función de sus diferencias.
Se realizaron varias mesuras para cada thread, correspondientes a diferentes "hardware counter events" como por ejemplo: numero de instrucciones, ciclos de reloj o fallos de cache a L1.
Se proponen métodos que calculan las diferencias entre los threads utilizando el estadístico de Kolmogorov-Smirnov como valor de similitud para establecer cierto grado de divergencia entre cada par de threads.
Después de establecer el umbral que indica cuando los threads corresponen al mismo grupo, el algoritmo de agrupamiento se puede aplicar.
Se compararon tres algoritmos de "clustering" (el algoritmo del líder, K-medoids y CLARANS) para evaluar cual es el que mejor cumple los objetivos del proyecto, es decir, que sea un algoritmo rápido y robusto.
Los tests mostraron que el que mejor rendimiento ofrece en calidad de grupos es el CLARANS, aunque requiere de un mayor tiempo de computación que el algoritmo del líder, pero menos que el K-medoids.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

High-performance computing (HPC) is recognized as a driving force for innovation in science and technology. HPC clusters are indispensable tools for research, development, and data analysis in many fields of science and engineering. It is part of a global market funded by all kind of companies, financial institutions, governments, military defense, academia, etc. It has high revenues that are growing day by day and it is expected that they will keep increasing in the following years.

HPC applications can be executed with numerous cores, i.e. there can be a large amount of threads (the smallest unit of processing that can be performed in an operating system) within the same process. The analysis of large-scale HPC applications that comprise thousands of threads is one of the main challenges in HPC. The analysis of all of them, one by one, can be very expensive with respect to time and resources.

Existing tools for HPC applications analysis, such as Paraver [1] and Scalasca [2], focus on visualization of the application's behaviour, while most of the analysis is not automatic, it is performed by the user. This may become a problem when the number of threads is large, resulting in tedious visualization analysis. Existing HPC analytic tools could significantly benefit from systematic approaches that could reduce the amount of data that is analysed by the user.

This project is capable of grouping application threads into different clusters with similar behaviour. Once they have been clustered, instead of analysing thousands of threads, a user could focus its effort on a handful of threads from different clusters, without being necessary to analyse them all but only one or some from each cluster.

Each application thread is described with a number of statistics of interest: number of instructions executed, branch instructions, float operations, clock cycles and L1, L2 and L3 cache misses. For each thread, numerous samples of the mentioned hardware counter events are collected. The first step of the project is to analyse the data in order to find shortcuts that may help the clustering algorithms to perform in a faster and more precise manner.

The use of the adequate algorithms is crucial for the performance with which the program has to identify the clusters, as well as manipulating the data as fast as possible. Experiments with different options for each algorithm are contrasted in order to identify the one that best fits with the project requirements and, finally, the algorithms that showed a better performance are compared in order to offer a general vision of their strengths and weaknesses.

Chapter 2

State of the art

2.1 Clustering overview

Cluster is the unsupervised classification of a set of patterns into clusters (groups) based on their similarity. In easy terms, after the classification each cluster consists of objects that are similar to each other, while dissimilar objects belong to other clusters. There are degrees of similarity between groups; one cluster can be more related to one than to other clusters.

There are many techniques for representing and measuring the similarity between data elements and clustering them. The best choices depend on the kind of data, the types of clusters we prefer depending on the goal of the analysis, the definition of similarity, etc.

The problem of data clustering has been studied and is commonly applied in a wide variety of fields such as marketing [3], medicine (leukaemia) [4] or biotechnology (genetis) [5], among others. It has become of great importance due to the fast evolution of computers and its power for classifying and understanding data with the appropriate algorithms.

There are numerous types of clusters, but the most common ones are partitional or hierarchical clusters, with hard or fuzzy clustering and approached with an incremental or non-incremental technique:

Hierarchical clustering: Consists in building a hierarchy with the objects being nested in different inner-clusters. Closer objects in the hierarchy are more similar than more distant objects. Data is usually represented using dendrograms. See figure 2.1a for a dendrogram representation.

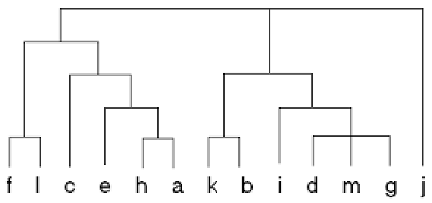
Partitional clustering: Consists in decomposing the data directly into a set of one-level (unnested) disjoint clusters. See figure 2.1b as an example.

Hard clustering: When each object belongs to a single cluster.

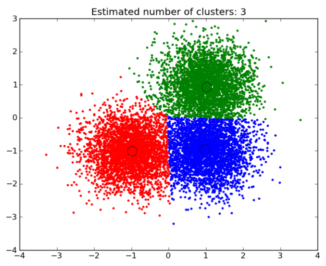
Fuzzy clustering: When each object can belong to more than one cluster. Each object has a certain degree of membership in each cluster.

Incremental clustering: When the algorithm works with an item at a time and decide how to cluster it given the current set of objects that have already been processed.

Non-incremental clustering: When the algorithm uses the information all the objects at once.



(A) Dendrogram



(B) Hard partitional clusters

FIGURE 2.1: Hierarchical and partitional clustering examples

The problem of partitional clustering has been approached from diverse fields of knowledge like graph theory [6], artificial neural networks [7], evolutionary computing [8] or swarm intelligence [9]. However, most algorithms assume that the number of clusters is previously known (or easily approachable), greatly facilitating the clustering task as it constitutes a valuable criterion for guiding the search. In our case, this information is not provided, thus, as finding the exact number of clusters is a difficult work, our approaches require some extra computational time in order to guarantee a robust partitioning.

In this project, a hard partitional clustering approach is used for all clustering methods. This is because we are mostly interested in finding significant differences among threads belonging to execution of a program. Although it would be possible to compute a hierarchical solution, small differences between threads are not of our concern and it is preferred to find main cluster partitions that are clearly defined in some of the data.

Let $P = \{P_1, P_2, ..., P_n\}$ be a set of n patterns (objects) to be clustered. Formally, a hard paritional clustering tries to find a partition $C = \{C_1, C_2, ..., C_k\}$ of k clusters that have to fulfill the following properties:

1. $C_i \neq \emptyset, \forall i \in \{1, 2, ..., k\}$
2. $C_i \cap C_j, \forall i \neq j \wedge i, j \in \{1, 2, ..., k\}$
3. $\bigcup_{i=1}^{k} C_i = P$

Typically, pattern clustering involves the following stages [10]:

1. Pattern representation (optionally: feature selection/extraction)
2. Definition of a pattern proximity measure appropriate to the data domain
3. Clustering
4. Data abstraction
5. Assessment of the output

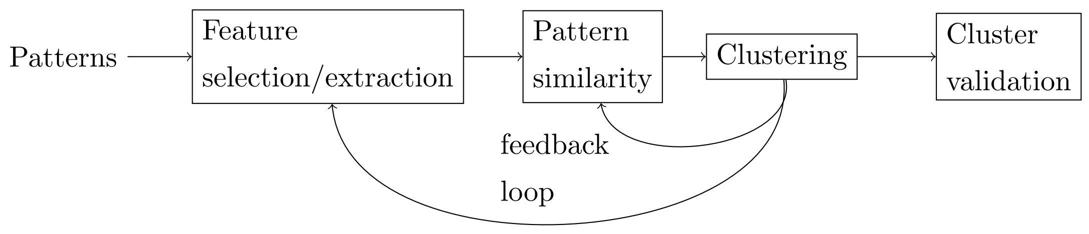


Figure 2.2: Clustering stages

Each of the previous stages is described as follows:

1. Pattern representation refers to the information available to the algorithm, for example: number of classes, number of patterns, etc.
   Feature selection is the process of identifying the most effective subset of the original features to use in clustering.
   Feature extraction is the use of one or more transformations of the input features to produce new (better) features.
2. Pattern similarity is usually measured by a distance function defined on pairs of objects. Euclidean distance between data points is widely used.
3. Clustering is the application of an algorithm for grouping the data based on the similarities/dissimilarities between objects.

4. Data abstraction is the process of extracting a simple and compact representation of the data set. In general, it provides a compact description of a cluster. There are some considerations that can help in the process: how to distinguish a good solution (clustered data) from a bad one? Are these real clusters?

5. Cluster validity is the assessment of a solution (algorithm's output). Often, the analysis uses a specific criterion of optimality, but it is usually subjective.

In this project, these stages are properly applied in order to guarantee a good clustering accuracy.

2.2 State of the art in clustering algorithms

K-means [11] is one of the most used algorithms [12]. It was first published in 1955 and, the fact that it is still widely used, indicates the difficulty of developing a general purpose and fast clustering algorithm.
When the centroids cannot be calculated due to the properties of the data, K-means cannot be used. Then, there are alternatives based on K-means idea such as K-medoids [13], including numerous variations, being PAM (Partitioning Around Medoids) [14], CLARA (Clustering LARge Applications) [14] and CLARANS [15] the most well known. Although there are other alternatives to K-medoids like DBSCAN [16].
There is a good review of clustering algorithms and their main properties in [17].

2.3 Actual solutions to the problem

There are different software tools that can be used for distinguishing similar and dissimilar threads from high-performance applications. The ones presented in the following subsections are good examples of these performance optimization tools. They offer more options and functions for performing a good and accurate analysis of the processes than this project. However, they consist, basically, in visualization tools, needing the user to concentrate in the interaction with the application in order to analyse the data.

Paraver

Paraver [1] is used in order to obtain a global perception of the behavior of the application by visual inspection.

Developing parallel applications that use the resources correctly is not an easy task. Performance analysis with Paraver support developers on the evaluation, adjustment and optimization of their codes. Its performance analyzer uses traces in order to explore the collected data with a reliable flexibility.

All performance tools developed at Barcelona Supercomputing Center [18] are freely available as open-source projects pretending to help the developer to detect performance problems and understand the applications' behavior.

The following picture shows the threads behavior of a high-performance application. The traces are processed and then displayed with Paraver, providing an image like the following:

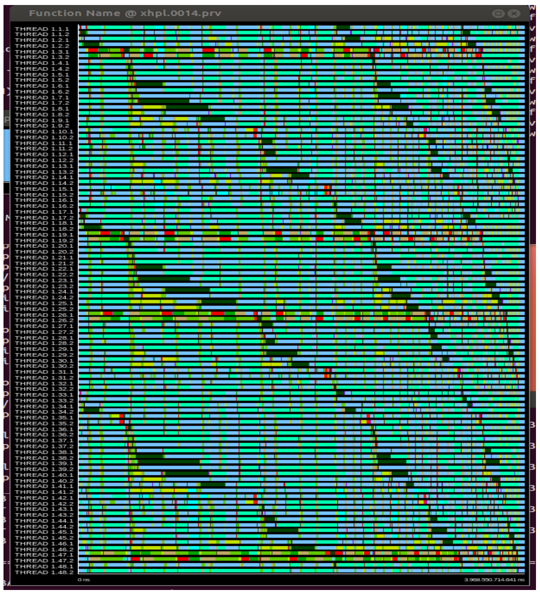


FIGURE 2.3: Paraver threads visualization

Each line corresponds to a thread. The thread behaviour in the execution can be followed by observing the different colors of its line from left to right. In the figure above, the colors represent the frequencies in which it is executed in the CPU. Based on these colors, a user can decide whether two threads behave similarly or not.

Scalasca

Scalasca [2] is a software toolset that helps on the performance optimization of parallel programs executed on large-scale systems with numerous cores. It achieves this goal by measuring and analysing the applications at runtime based on the concurrent behavior via event tracing. The analysis identifies possible performance bottlenecks and offers a way for exploring their reasons.

Specially, Scalasca works well for identifying wait states in applications with a bast amount of threads and to summarize these measurements. The scalability of Scalasca and its effectiveness have been proven by analysing real-world applications in a variety of computer systems.

Chapter 3

Project Planning

The project started on the 1st of September 2014 and it was finished on the 15th of May 2015.
There were some changes with respect to the initial planning. First of all, the analysis took more time due to some unexpected results that produced some changes to the methodology, causing a delay of several weeks. After this, the posterior tasks were also shifted in time. As the initial planning stated that the project would be defended on April and now it is going to be defended at the end of May, this delay did not represent a big problem. If it had not been possible to change the date, there would be some parts of the project that would have been sacrificed in order to submit it on time.

3.1 Task description

First contact

Before starting the project, there were necessary tasks to be done in order to obtain a better understanding of the basis. It started on the 1st of September and ended on the 19th of the same month.

Meeting with the supervisors: This was the first step, which was necessary for discussing the work that needed to be done, the objectives and the methodology.

Reading about project's background: This task consisted in reading and learning more about statistical testing (hypothesis, assumptions...), clustering (types of clusters, state of the art in algorithms...) and HPC applications (how they are parallelized, MPI, OpenMP...).

Installing software: It was required to install the R environment [19] in a Linux Mint distribution on a personal laptop as well as external R packages that were needed for certain statistical tools.

Learning R: After the installation of the software, it was needed to learn basics of R programming language before starting with the development.

Project planning

This part corresponded to the "Gestió de Projectes" (GEP) course. It started on the 15th of September and ended on the 12th of October with the GEP's final presentation. The tasks consisted in six deliverables for planning the whole project, providing a general overview and estimation in the following aspects:

- Project's introduction and scope
- Definition of the tasks and project planning
- Project's economic management and sustainability
- Contextualization (state of the art) and bibliography
- GEP's final presentation

Project development

This is the main phase, the one in which all the data is analysed and the project is implemented. It lasted from the 22th of Semptember to the 9th of April. It consists in various approaches of the artificial intelligence, where each one is built on the top of its predecessor, improving it by applying more accurate or more efficient techniques. It consisted in the four following tasks:

Analysis: It consisted in analysing the data sets in order to find valuable information for guiding the clustering algorithms. It lasted more than expected and it embraced all the time planned for the project development.

First approach: It consisted in developing a naive clustering approach based on the results obtained by the analysis until that point.

Intermediate approach: It consisted in programming a better clustering algorithm (firsts implementations of leader algorithm and K-medoids 9).

Final implementation: It consisted in developing the final implementations of the clustering algorithms based on all the analysis.

For the tasks related to implementing the different approaches (first, intermediate and final implementation), there were the following steps involved:

Reading material in order to have the basic background before starting the implementation.

Implementing the necessary code in order to obtain clustering solutions that can guide the posterior analysis and implementations.

Testing the implementation in order to prevent software development errors and adjusting the possible failures.

Writing a draft of the last three steps in order to have a continuous information compilation about the project's development.

Experiments and validation

After the final implementation, it was needed to validate the different clustering techniques in order to compare the quality of the clusters and the time needed for each algorithm to compute them.
The task consisted in trying different options for each clustering technique and **evaluate their strengths and weaknesses** in different situations.

Project writing

The task consisted in **writing the final thesis** to be submitted. The parts from the GEP course had to be rewritten in order to adapt them to the last state of the project. The drafts of project development and the experiments had to be written and following a logical flow.
It started at the 24th of April and ended at the 15th of May, 2 days before the submission.

Final stage

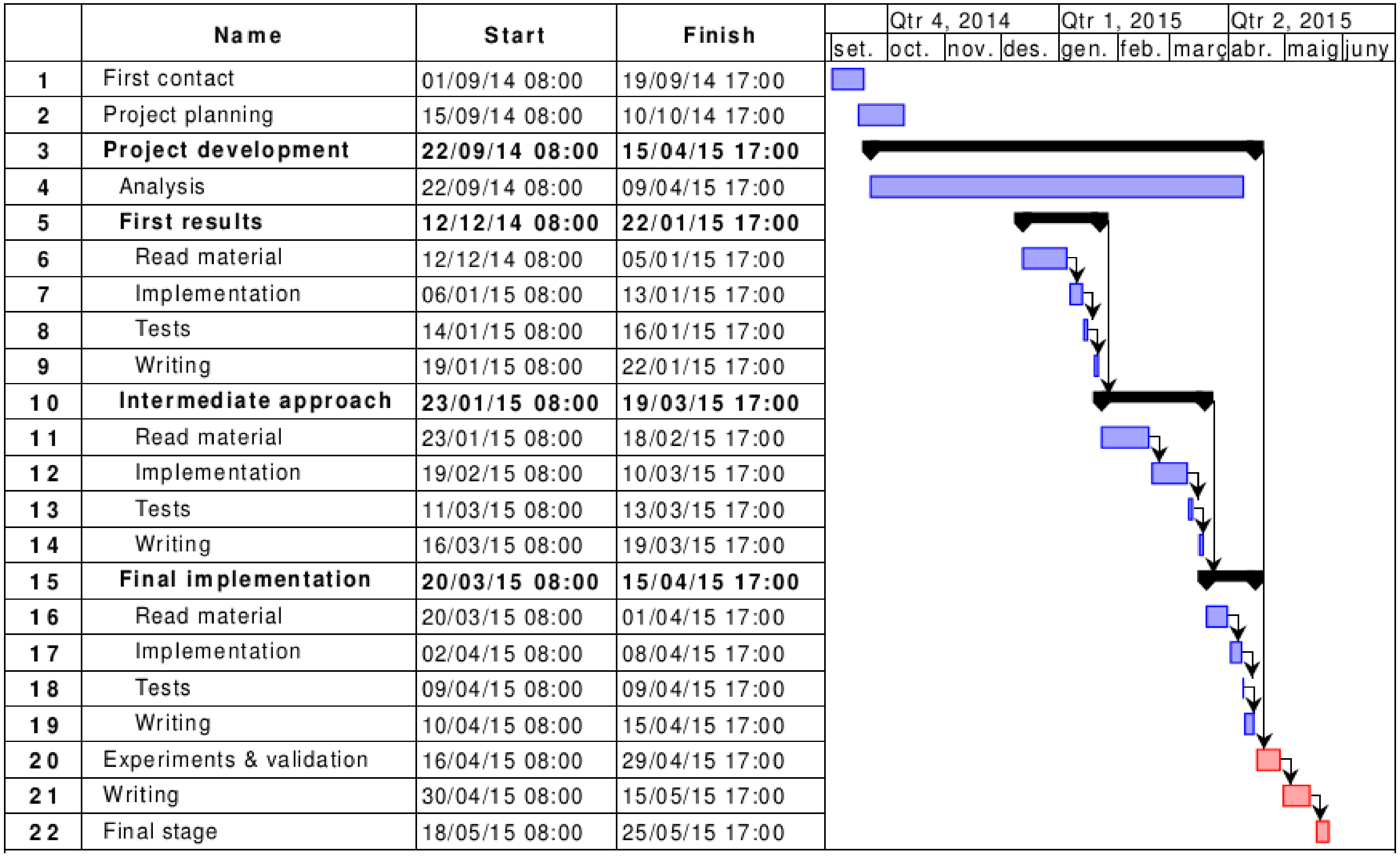
This is the last part, consisting in corrections and improvements before the submission and the preparation of the defense. It takes from the 16th of May to the 25th of the same month, when the defense of the project is scheduled. The tasks involved are:

Revision of the whole document before the submission.

Slides preparation for the defense.

Defense

3.2 Gantt chart



	Name	Start	Finish
1	First contact	01/09/14 08:00	19/09/14 17:00
2	Project planning	15/09/14 08:00	10/10/14 17:00
3	**Project development**	**22/09/14 08:00**	**15/04/15 17:00**
4	Analysis	22/09/14 08:00	09/04/15 17:00
5	**First results**	**12/12/14 08:00**	**22/01/15 17:00**
6	Read material	12/12/14 08:00	05/01/15 17:00
7	Implementation	06/01/15 08:00	13/01/15 17:00
8	Tests	14/01/15 08:00	16/01/15 17:00
9	Writing	19/01/15 08:00	22/01/15 17:00
10	**Intermediate approach**	**23/01/15 08:00**	**19/03/15 17:00**
11	Read material	23/01/15 08:00	18/02/15 17:00
12	Implementation	19/02/15 08:00	10/03/15 17:00
13	Tests	11/03/15 08:00	13/03/15 17:00
14	Writing	16/03/15 08:00	19/03/15 17:00
15	**Final implementation**	**20/03/15 08:00**	**15/04/15 17:00**
16	Read material	20/03/15 08:00	01/04/15 17:00
17	Implementation	02/04/15 08:00	08/04/15 17:00
18	Tests	09/04/15 08:00	09/04/15 17:00
19	Writing	10/04/15 08:00	15/04/15 17:00
20	Experiments & validation	16/04/15 08:00	29/04/15 17:00
21	Writing	30/04/15 08:00	15/05/15 17:00
22	Final stage	18/05/15 08:00	25/05/15 17:00

FIGURE 3.1: Planning Gantt chart

Chapter 4

Economical management and budget

4.1 Costs identification

The costs derived from in this project are the following: human resources, direct costs (hardware and software resources) and indirect costs. The part consisting in writing the final thesis is not taken into account in the calculations below.

4.2 Human Resources

In order to guarantee a proper development of this project, if it was developed in a real company, the necessary staff would be:

Project manager: Responsible of the planning, execution and closing of the project.

Data analyst: Responsible of inspecting, cleaning, transforming, and modeling the data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making.

Software developer: Responsible of researching, designing and implementing the programs.

Software tester: Responsible of executing the programs with the intent of guarantee a correct performance of the software.

Each of these roles has an associated remuneration. Doubles pays are not considered. In Spain, the real cost of a worker, from the point of view of the company, includes the gross salary plus the social contributions, which are the 29.7% of the gross salary. They include a 23.6% for common contributions, 5.5% for unemployement and 0.6% for job training.

Role	**Gross salary (€/month)**	**Social contr. (€)**	**Total month (€/month)**	**Total hour (€/hour)**
Project manager	6,000	1,782	7,782.0	48.64
Data analyst	3,200	950.4	4,150.4	25.94
Software developer	2,600	772.2	3,372.2	21.08
Software tester	2,500	742.5	3,242.5	20.27

TABLE 4.1: Cost per worker (month)

Taking into account the estimated hours for the project, each worker has a different number of hours assigned. The table below shows the total cost of human resources for the project, using the calculations of salary per hour shown in the table above.

Role	**Dedication (hours)**	**Price (€/hour)**	**Project cost (€)**
Project manager	100	48.64	4,864
Data analyst	200	25.94	5,188
Software developer	300	21.08	6,324
Software tester	60	20.27	1216.2
TOTAL			**17,592.2**

TABLE 4.2: Cost per worker for the project

4.3 Direct costs

Hardware Resources

The project was mainly developed in a normal laptop, except for the computations that required a high amount of resources, which were executed on MareNostrum, although any supercomputer with more than 1,024 cores could have been used for this purpose. For this budget, it is considered the price of renting 1,024 at Sabalcore [20], a company dedicated to offer HPC services in the cloud, for 3 days.

Product	**Price (€)**	**Units**	**Useful life (years)**	**Amortization (€)**
ASUS X550LDV	723	1	4	40.854
TOTAL				**40.854**

Table 4.3: Cost of hardware resources

Product	**Price (€/hour)**	**Units**	**Time (days)**	**Total cost (€)**
Core	0.128	1,024	3	9,437.184
TOTAL				**9,437.184**

Table 4.4: Cost of renting cores at Sabalcore

Software Resources

For the development and the analysis, the last version of RStudio (ersion 0.98) [] was used in a Linux Mint 17 operating system. The writing part was done in LaTeXusing TexStudio. As all these resources are freely available online, therefore there are no costs associated to software resources.

Product	**Price (€)**
RStudio Desktop ver.0.98.1103	0.00
TexStudio ver. 2.9.4	0.00
Linux Mint 17	0.00
TOTAL	**0.00**

Table 4.5: Cost of software resources

4.4 Indirect costs

The indirect costs comprise, the electricity consumption of the laptop and of the Internet. On average, a normal computer consumes 60W/hour.

Indirect cost	**Price**	**Amortization (€)**
Laptop electricity	23.7 €/year	5.357
Internet	45 €/month	122.054
TOTAL		**127.411**

Table 4.6: Indirect costs

4.5 Total budget

Grouping all the costs together, the total budget is of 27,197.611 €.

Resource	**Cost (€)**
Human Resources	17,592.2
Direct costs	9,478
Indirect costs	127.411
TOTAL	**27,197.611**

Table 4.7: Total cost of the project

Chapter 5

Sustainability and social engagement

5.1 Economical aspects

As stated in 4.5, the estimated cost for the whole project is of 27,197.611€. It is not strictly necessary to keep the project updated once it is finished, although some changes and improvements (12) would be possible and desirable.

This project cannot be launched in a competitive market as there are other freely available software tools that can be used to achieve the same goals, even if this is not with the same level of comfort. See the section 2.3 for more information about these tools.

The hardware requirements for the project vary depending on the part. In order to produce the data needed for the analysis, the execution of high performance applications like the ones used in this project is needed (see chapter 6). Therefore, a computer capable of using numerous cores in parallel is necessary. For realistic thread measurements, it would be important to execute each application with at least 128 or 256 cores and to assume the main memory demand.
For the preprocessing of the data and the execution of some of the clustering algorithms (see chapter 9), if the application to be preprocessed or clustered is executed with more than 512 cores, it is important to use a computer with sufficient main memory. For example, for preprocessing a single trace file from an application executed with 1,024 cores, 10 GB of RAM are needed.
For some of the analysis, the programming and the clustering part, a normal computer can be used.

The software requirements for this project are not high. All algorithms are programmed in R language [21], well known as an open source language with its main IDE, RStudio, freely available online in [19]. The programming part of the project was developed under Linux Mint as operating system [22], a widely used Linux distribution for desktop environments.

From the point of view of human resources, one worker is enough for analysing the data, developing the clustering algorithm and testing.

Unquestionably, with more resources, more programmers or analysts, the project could be developed in less time. However, it is difficult to achieve a lower cost in the hardware part because the project is developed for HPC applications. Therefore, a computer capable of executing such applications is needed in order to perform an efficient and accurate analysis of the data.

5.2 Social aspects

This project is directly applicable in HPC, which is used by a considerable number of companies and institutions (for research, data analysis...), and this number is still increasing.
However, the project is designed for the analysis of threads in HPC applications, thus it cannot be part of everyday life because of the small and the low power devices that are commonly used. Neither it offers a way of speeding up or improving the performance of any application or computation in research. Therefore, it is not expected to have a direct impact on the society in a direct way.

As shown in chapter 2, this is not the sole project that can be used for differentiate and cluster threads, but most of them need the user to interact directly in order to obtain the results. With this approach the user does not have to interact with the application, saving time to him/her. Therefore, the project is not directly necessary, but it is going to be helpful because of the time the user can dedicate to a non-automatizable tasks instead of interacting with the application.
This project can be used instead of other programs in order to save some time and to group the threads more accurately, but the other applications are more versatile and offer visual support. Therefore, it is not going to substitute them.

5.3 Environmental aspects

MareNostrum [23] and other servers used in this project were only used in very punctual moments, being negligible in this context. The project was mainly developed in an ordinary laptop. Since there is no need of a special computer, an old one can be used, avoiding in this way to contribute to the negative impact regarding electronic waste materials.

The use of a computer is necessary in practically all 630 hours assigned to the development of the project. A laptop consumes in average 60W/hour when it is turned on and processing. Therefore, the total energy estimated for the development of the project is 60*630 = 37,8 kW. This could be reduced by using a low power laptop, but it is necessary to consider that, for testing the application, the project uses big files (between 300MB and 6,000MB, depending on the application). Therefore, the total number of hours dedicated for testing would increase because the performance of these kind of computers is lower than the normal ones. Probably, the best option regarding energy efficiency, would be to use a low power laptop for development and a normal one for testing.

Once the project is finished, the laptop is going to be used in other projects and for daily usage. After 3 or 4 years, it can be given to a non-profit organization that can reuse it, for example, the ONG from UPC, "Tecnologia per a Tothom" (TXT) [24], which repairs and gives old computers to people with less resources from numerous countries from all over the world.

Since this project uses electrical products for its development, it pollutes the environment to a certain extent. In one hand, the energy used comes from power plants, where big amounts of CO_2 and other greenhouse effect-contributing gases are produced every year to produce energy. In the other hand, the process from which computers are manufactured is not free of pollution: hundreds of kilograms of fossil fuel and about 1,500 litres of water are used for the production of one computer, 83% of the energy used for one computer is generated in the manufacturing process (the other 17% is for the daily use) [25].
Asus, the manufacturer of the laptop used for the project, is known to be one of the more committed brands to sustainability from its internal practices to its production processes, to its green products and its environmentally-aware employees [26].

The written part of the project is going to be submitted via Internet in PDF format. Therefore, it is not planned to print it unless one or more professors from the defense ask for it. In that case, it is going to be printed using recycled paper, which is helpful for the environment and it can be easily recycled.

Although the generated pollution for this project is practically negligible, it does not help the environment in any case. However, it was tried to minimize the total amount of generated pollution whenever it was possible.

Chapter 6

Experimental environment

In this chapter there are described the hardware platform and the applications used for the analysis, experimetns and tests of the project.

6.1 Applications

Production HPC applications

We analyse applications that are part of the Unified European Application Benchmark Suite (UEABS) [27]. UEABS is comprised of 12 scientific HPC applications that represent a good coverage of the production HPC applications running on large-scale Tier-0 and Tier-1 HPC systems in Europe. All UEABS applications are parallelized using Message Passing Interface (MPI). Each UEABS application has either one or two input datasets. When there are two datasets, Test Case A (small dataset) is designed to run on Tier-1 sized systems, up to approximately 1,000 x86 cores, while Test Case B (large dataset) is designed to run on Tier-0 sized systems, up to approximately 10,000 x86 cores. When there is only one dataset , both system sizes are suitable.

Table 6.1 summarizes the applications used from UEBABS in the study. The first two columns of the table list application names and the area of science that each application targets. The third shows the number of cores used in the experiments. 10 out of 12 UEABS applications were executed, Code_Saturne and GPAW being discarded due to installation and set up issues. Initially, each installed application was were executed on 256 cores when performing the first analysis on the data but, for the final results shown in this thesis, almost all applications have been executed on 1,024 cores. QuantumEspresso could only be executed with 256 cores and SPECFEM3D is designed to be executed only on specific numbers of cores, 864 in Test Case A and 11,616 in Test Case B.

Application	**Science area**	**Number of cores**
ALYA	Computational mechanics	1,024
BQCD	Particle physics	1,024
CP2K	Computational chemistry	1,024
GADGET	Astronomy and cosmology	1,024
GENE	Plasma physics	1,024
GROMACS	Computational chemistry	1,024
NAMD	Computational chemistry	1,024
NEMO	Ocean modeling	1,024
QuantumEspresso	Computational chemistry	256
SPECFEM3D	Computational geophysics	864

TABLE 6.1: UEABS applications used in the study

Linpack

High-performance Linpack Benchmark [28] solves a dense system of linear equations. The Linpack benchmark is not distributed with a given input dataset, but allows the user to scale the size of the problem in order to achieve the best performance on a given system. In each Linpack run, the user specifies the number of Linpack processes and sets the problem size according to the amount of available physical memory. Linpack documentation recommends to set problem size to approximately 80% of the available memory [2]; the remaining memory is required for the operating system and monitoring software. High-performance Linpack was executed on 1,024 cores.

6.2 Hardware platform

The experiments presented in the paper were executed on the MareNostrum supercomputer [23], one of six Tier-0 HPC systems in the Partnership for Advanced Computing in Europe (PRACE) [29]. MareNostrum contains 3,056 compute nodes connected with the Infiniband network. Each node contains two Intel Sandy Bridge-EP E5-2670 sockets that comprise eight cores operating at 2.6 GHz. Although Sandy Bridge processors support hyper-threading at core level, this feature is disabled, as in most of the HPC systems. Therefore, at the operating system level, MareNostrum compute node is seen as 16 virtual CPUs (2 sockets x 8 cores). Sandy Bridge processors are connected to main memory through four channels, and each channel is connected to a single 4 GB DDR3-1600 DIMM, which makes 32 GB of main memory per node. In all the experiments, one application process per core was executed, i.e. 16 processes per compute node and with hyperthreading turned off.

MareNostrum compute nodes comprise 32 GB of DRAM memory per node, that is 2 GB per core.

Chapter 7

Methodology

7.1 Tools

The hardware and software tools used for the development of this project are the following:

Hardware:

Asus X550L for programming, writing and testing the implementations.

MareNosrum for generating the trace files. See chapter 6 for more information.

Software:

Linux Mint 17 (64 bits) as main operating system.

R environment version 3.0.1 for executing and testing the programs.

Sublime Text 3 as main R programming editor.

Sharelatex as a LaTeXeditor for the thesis.

7.2 Evaluation

The project has been structured following the clustering stages defined by A.K. Jain and R.C. Dubes [10] and reviewed in [17], consisting on the following stages: pattern representation (feature selection/extraction), pattern proximity, clustering and cluster validation.

Pattern representation

As explained in 6, 11 HPC applications (10 from UEABS [27] and 1 from Linpack benchmark [28]) were executed on MareNostrum with 256 to 1,024 cores, depending on the application. Paraver [1] was used for measuring and storing the values of different hardware counter events. The instruction(s) that gives the order for capturing these values are directly placed in the code of each application. Every time a counter is read, it is reset to zero (for each thread, every read contains a value which is counted since the last reading). The hardware counter events analysed in this project are the following: number of instructions completed, clock cycles, L1 cache misses, L2 cache misses, L3 cache misses, performed floating point operations and number of branch instructions. Therefore, for each thread and hardware counter there are various measurements. The number of measurements varies depending on the thread, application, execution, etc. but based on the tested programs, this usually varies between 2,500 and 4,000 for an application executed with 1,024 cores.

Once an application has completed its execution, all the measurements remain stored in a file. Afterwards, the file is preprocessed and saved in csv file format. After the preprocessing, a file has the following format:

Thread number	**Inst.**	**Clock cycles**	**L1 misses**	**L2 misses**	**L3 misses**	**Float inst.**	**Branch inst.**
1	1436	8534	184	45	33	4	298
1	289	1428	57	28	2	0	85
1	280	679	36	2	0	0	83
1	9763	84283	1386	755	479	63	1959
...							
2	1436	3132	196	29	13	3	301
2	270	1290	70	13	2	0	130
...							
1,024	359	854	42	20	6	1	146

Table 7.1: Preprocessed data format. Hard counter event values from the GENE application

Pattern proximity

The statistic from the Kolmogorov-Smirnov test (K-S test) [30] explained in appendix A has been used in this project in order to be able to establish a value of similarity between a pair of threads (patterns).

It was considered that the threads from different executions of the same application (executed in the same machine and with the same number of cores) can be considered similar. For example, if an application is executed two times with N cores and each thread is compared with the same number from both executions (i.e. N comparisons), the values obtained with the statistic of the K-S test would indicate that the threads are similar. This is because, although there would be differences in the measurements due to different environment states in the moment of the execution (like the decisions of the operating system based on the processes being executed on the machine), the workload of each thread should be equitably divided in each execution, as the tasks to be run are the same in all executions of the same application.

Clustering

Three clustering algorithms were selected, implemented and adapted based on the results obtained in the analysis in order to achieve a better performance and a faster execution. They are the leader algorithm, an incremental clustering algorithm proposed by Hartigan in 1975 [31], an improved K-medoids algorithm for large data sets proposed by H.-S. Park and C.-H. Jun in 2009 [13] based on local searching and CLARANS, an algorithm proposed by T.Ng. Raymond and H. Jiawei in 2002 [15] based on global searching.

Cluster validity analysis

The silhouette metric [32] for clustering validation has been used in order to rank the quality of the clusterings obtained with the previous algorithms.
Tests using different functions of the above mentioned algorithms were performed in order to use the ones that achieved a better relation between quality and time.

Chapter 8

Pattern similarity

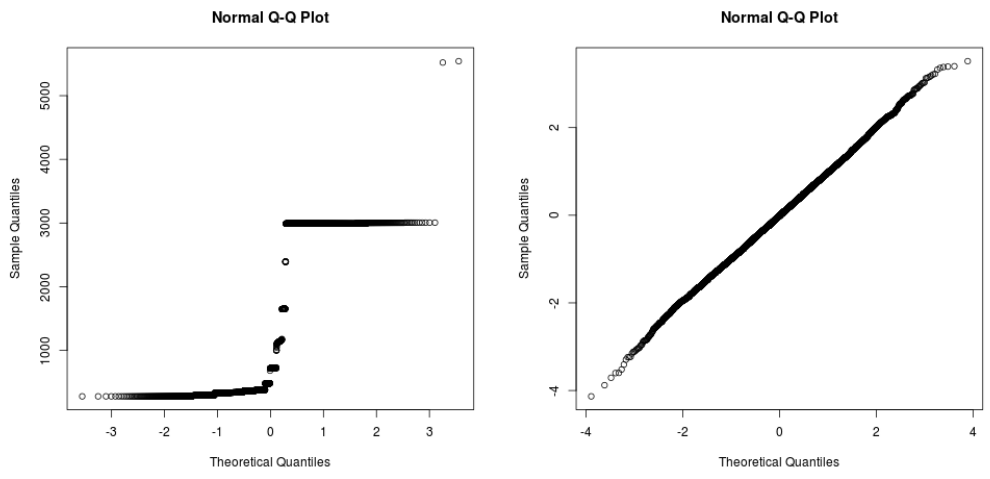
As explained in chapter 2, a relation of similarity or dissimilarity has to be specified in order to compare two objects (threads in our case) and determine their grade of similitude.
The same population can be compared in a lot of different valid ways. The comparison of two objects may vary depending on the motivation of the clustering. For example, two people can be compared based on the colour of their eyes, their gender, their age, etc.
The whole clustering depends on similarity values, as the algorithms are going to include (or not) two threads in the same cluster based on their likeliness. Therefore, it is important to choose its calculation carefully, being it as robust as possible and, in our case, as there is a lot of data, to be processed as fast as possible.

8.1 Data distribution

As explained in 7, hardware counter events are measured for each thread during an execution of an HPC application. The counters are: number of instructions completed, clock cycles, L1 cache misses, L2 cache misses, L3 cache misses, performed float point operations and branch instructions.
The number of measurements per thread vary in each program execution. This depends, basically, on the application, the number of cores and the environment of the process, like the decisions taken by the operating system related to process management. In general, based on the thread traces of the eleven tested applications, with 1,024 threads it is normal to have between 2,000 and 4,000 measurements per thread counter.

Before applying any test for comparing the values of different threads, it is useful to know what kind of data has to be processed. If it follows a known distribution, there are a

variety of specific tests that make inferences about the parameters of the distribution [33], producing more accurate results. One of the assumptions of the majority of parametric tests is that the population follows a normal (or almost normal) distribution.
Various quantile-quantile (Q-Q) plots of the traces of several threads from different applications and with different hardware counter events, showed that our data do not follow a normal distribution, even more, it is far from it. Below there is a Q-Q plot comparing the samples from LINPACK application corresponding to the clock cycles of thread number 2 with a normal distribution.



(A) LINPACK thread 2 clock cycles (B) Normal distribution

FIGURE 8.1: QQ plots with normal distributions

As seen in figure 8.1a, clock cycles of thread number two are quite far from following a normal distribution. If that was the case, it may look like 8.1b. As with in 8.1a, the rest of the threads and hardware counters from all other applications behave similarly.

8.2 The Kolmogorov-Smirnov test

As it is clear that the data do not follow a normal distribution, a nonparametric (distribution-free) test has to be used for the experiments. These are based on fewer assumptions about the distribution of the data. The cost of fewer assumptions is the loss of power compared to parametric tests, that is, when the alternative is true, they may be less likely to reject the null hypothesis.

The two-sample K-S test [30] (explained in appendix A) is one of the most useful and general nonparametric methods for comparing two samples, as it is sensitive to differences in both location and shape of the empirical cumulative distribution functions of the two samples. Furthermore, it performs well when there is a large number of elements, when the sample sizes have different lengths and it is fast as it only needs to traverse the samples once. These properties are not common in all nonparametric tests and are valuable in this project as they perfectly fulfil the requirements of our data.

As in this project there are seven types of measurements for each thread (instructions, clock cycles, etc.), it would be desirable to apply a nonparametric multivariate test that is capable of deciding whether two samples belong to the same distribution or how close the two samples are based on all seven hardware counters, that are in seven dimensions. These kind of tests (e.g. MANOVA) are usually used in normal distributions. Although there are nonparametric alternatives as the multivariate Cramver-von Mises criterion [34] or a proposed multivariate K-S test for goodness of fit (one-sample) performing well in two dimensions [35].

The Cramer-von Mises criterion (which is used to judge the goodness of fit of a cumulative distribution function compared to a given empirical distribution function) was also assessed with an R package with a multivariate version of the test [36]. However, when it was tried to execute it with all samples, the main memory requirements were highly superior than those available. After trying random sampling in order to reduce the size of the samples, the test executed well but the time needed for its calculations was high (more than a minute), a lot more than the univariate K-S test that gives a result with less than a second. This delay is crucial for the fast execution of the clustering algorithms 9. Therefore, it was decided not to use it and to continue with the K-S test.

First results with the K-S test

It was expected to not obtain statistical evidence against the null hypothesis when two threads are similar. For the tests, it was chosen a significance level of $\alpha = 0.05$ (5%) as a matter of good scientific practice [37].

When comparing pairwise all the threads of an application executed with N threads (i.e. $\frac{N(N-1)}{2}$ comparisons for each one of the seven ..parameters..), the p-values were almost zero in all cases and for all applications, that means the null hypothesis is rejected with high confidence and we can state that there is strong statistical evidence against the null hypothesis, that is, there is evidence that the samples are different.

In order to contrast the last results, two executions of each application were run with the same number of threads as their first execution.
These extra-runs were used for comparing the threads between different executions of the same application, instead of the threads involved in only one execution.

It is expected that if the same application is run with the same number of threads two or more times, the behaviour of the threads corresponding to the same number are going to behave in a highly similar way.
However, small p-values were obtained again when comparing the threads of the three different executions of each program.

Based on these results, it was concluded that the hardware counter events from the traces of different threads in the same execution or between different executions of a program should not be considered as equal or similar when performing accurate analysis, as we have shown that, with a very high probability, there are significant differences between the groups of values.
This significant differences are due to the changes in the environments of the applications where they are run. For example, although they are executed on the same machine, its operating system may take different decisions on the different processes based on its state in the moment of the execution.

8.3 Similarity values with the K-S test

As it was not possible to observe if two threads are similar or not (if those similarities exist) with hypothesis testing, it was needed a more flexible value of similarity (divergence/distance) for comparing the distributions.
There are various nonparametric tests for this purpose as Wilcoxon signed-rank test [38], Hellinger distance [39], K-S test statistic [30], Kullback-Leibler divergence [40], etc.

With the K-S test, instead of using the p-value for rejecting or not the null hypothesis, the statistic D can be used for establishing a degree of divergence between the empirical distributions. The D statistic gives values between 0 and 1 when comparing two samples, being 0 the value that indicates that both samples come from the same distribution (in this extreme case the p-value would be 1) and 1 that the samples are completely different ($p - value = 0$).

The K-S test statistic as a value of similarity was chosen for the reasons explained in the second section of this chapter, that is, the samples can have different lengths, it is robust when the samples are large and it is fast.

In order to contrast the results, the same tests were performed with the Hellinger distance, as this test also fits our data requirements. When compared with the values obtained with the K-S statistic, there were high values of correlation (between 0.8 and 0.99), showing that both tests have similar criterion, although the range of values diverges due to the mathematical properties of each test. However, the implemented version for R of the Hellinger distance requires around two times more computational time than K-S statistic. As the execution time is important for this project and the capability of calculating divergences is similar, it is concluded that in this case it is better to use the K-S statistic.

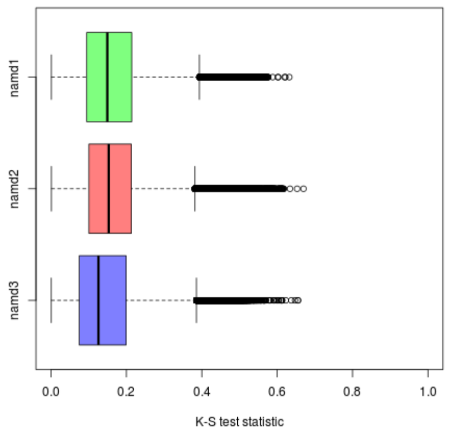
8.4 Applications' threads similarity

Most clustering algorithms do not need more than a function for determining the grade of similarity between the objects to be grouped. However, it may be helpful for the algorithms to have a threshold indicating when it is sure that two threads do not belong to the same cluster. This threshold is a requirement for some incremental algorithms for large data sets (as in our case). For example, the leader algorithm [31] explained in chapter 9.

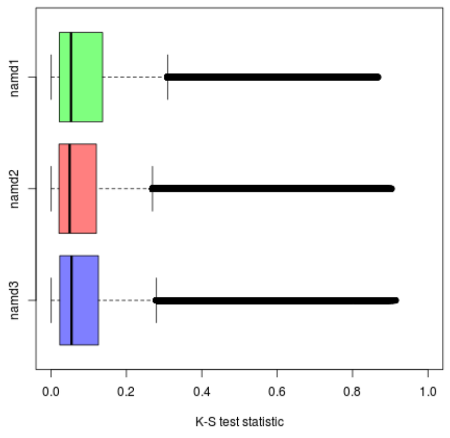
The following analysis show that these thresholds may exist in some applications.

8.4.1 Within application similarity

Given the hardware counter events measurements from an execution of an application, the K-S test did not show any evidence of similarity. However, when the values of the comparisons are measured with the K-S test statistic, it can be observed that different executions of the same application seem to behave similarly.
The figures below show the boxplots of different hardware counter events and applications in order to appreciate the different ranges obtained with the statistic. For each application and hardware counter, the values of the comparisons are obtained by comparing the threads pairwise, i.e. thread number 1 with thread number 2, thread number 1 with number 3... until the comparison between thread 1,023 and 1,024 (in this section, the measurements belong to the executions with 1,024 cores, as explained in chapter 7).
Therefore, each boxplot contains $\frac{1,204 \cdot (1,024 - 1)}{2} = 523776$ values.



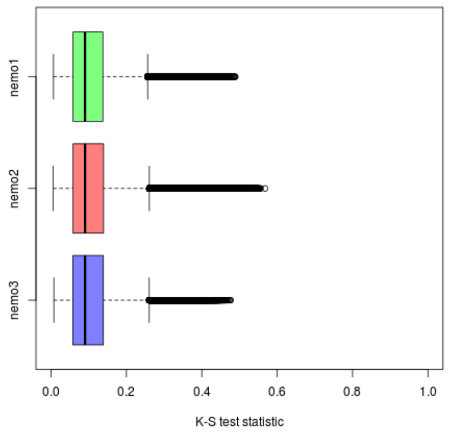
(A) Instructions



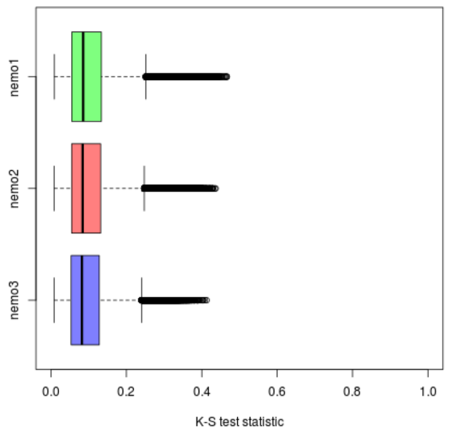
(B) L2 cache misses

FIGURE 8.2: Hardware counter events comparison in single NAMD executions

Figure 8.2 shows the value distributions of the number of the comparison of the number of instructions and L2 cache misses between threads in the application NAMD. All three executions (the fist one in green, the second in red and the third in blue) present similar value distributions, although they have different ranges depending on the hardware counter that is analysed.



(A) Clock cycles



(B) L1 cache misses

FIGURE 8.3: Hardware counter events comparison in single NEMO executions

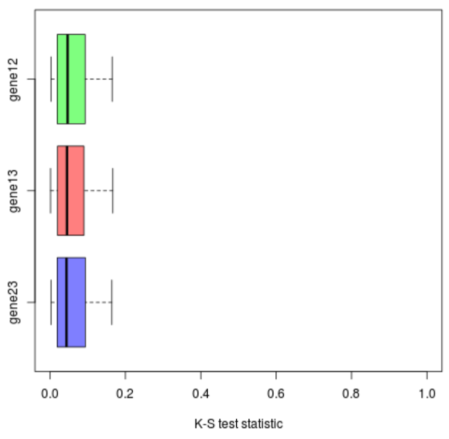
Similarly as before, figure 8.3 shows that the ranges of NEMO vary between hardware counters but the distributions remain similar between executions of the same application.

This pattern is present in practically all the applications and hardware counters, except for execution number of BQCD that shows a different distribution of the number of instructions with respect to the other two executions, and execution number one of GROMACS that presents significant differences regarding the number of instructions and clock cycles.

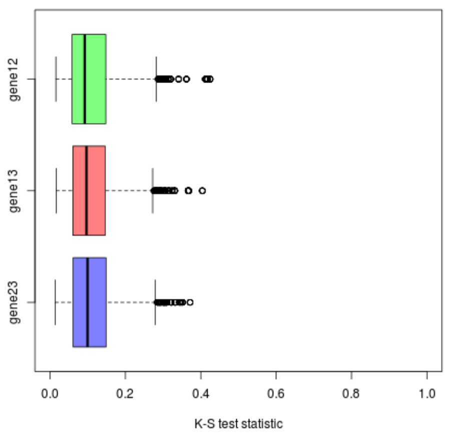
Based on all boxplots, it is clear that the threads from same applications and hardware counter event behave similarly, regardless of the execution.

8.4.2 Similarity between same application executions

As stated before, the same thread from two different executions of the same application (run with the same number of cores) can be considered similar. As explained in chapter 7, there are three executions per application. For each application, the 1,024 threads were compared between all three executions, i.e. 1,024 comparisons (for each hardware counter event) for run 1 with run 2, run 1 with run 3 and run 2 with run 3. The following boxplots show some examples of how the values from the comparisons of different hardware counter events of the same applications are distributed and the ranges they embrace.



(A) Instructions

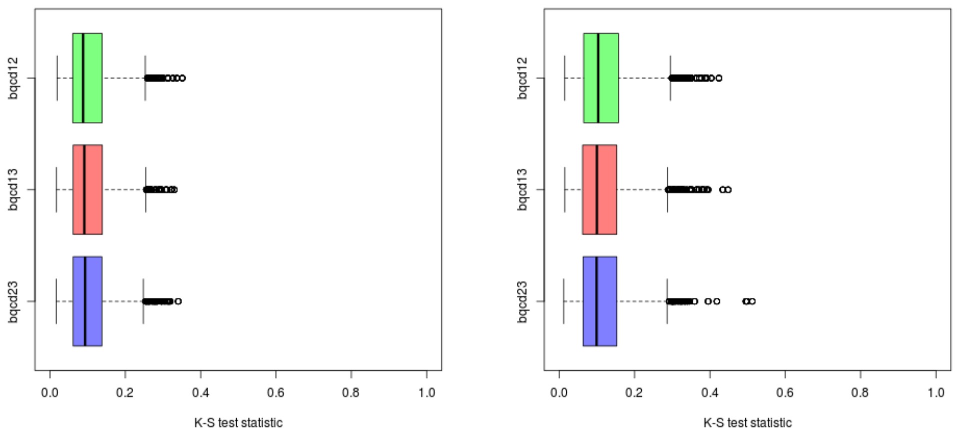


(B) Clock cycles

FIGURE 8.4: Hardware counter events comparison in different GENE application executions

Each boxplot above contains the 1,024 values from the comparisons with the K-S test between the threads of GENE application runs. In figure A there are the comparisons of the number of instructions executed, GENE execution 1 with execution 2 (green), execution 1 with execution 3 (red) and execution 2 with execution 3 (blue). The same in shown in figure B but with clock cycles instead of instructions.
It is clear that the range of values of a hard counter event is similar, independently of

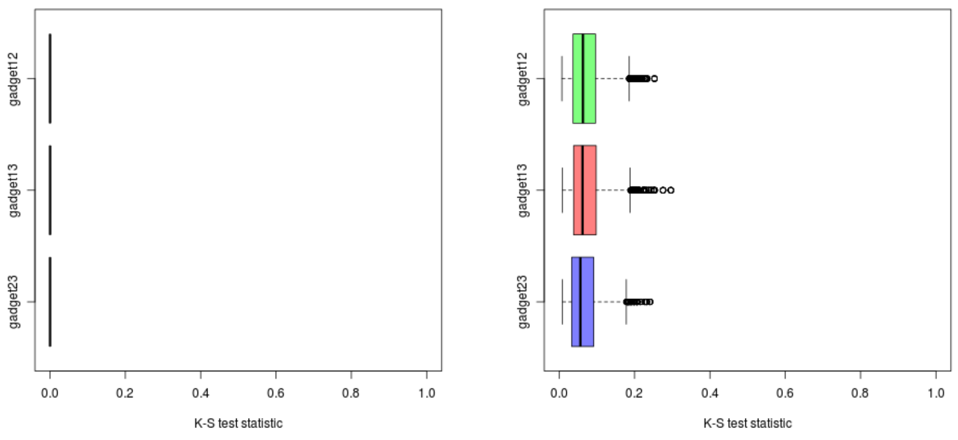
the executions compared. With the instructions, for example, it goes from 0 to 0.16 and with the clock cycles between 0 and 0.4, approximately.



(A) Clock cycles (B) L1 cache misses

Figure 8.5: Hardware counter events comparison in different BQCD application executions

Figure A shows the comparisons among the number of clock cycles executed in the BQCD application. The same is shown in figure B but with L1 cache misses.
Again, the range of values of a hard counter event is similar, independently of the number of executions compared. In this case, it goes from 0 to 0.38 with the clock cycles and between 0 and 0.5 with the L1 cache misses, approximately.



(A) Instructions (B) L2 cache misses

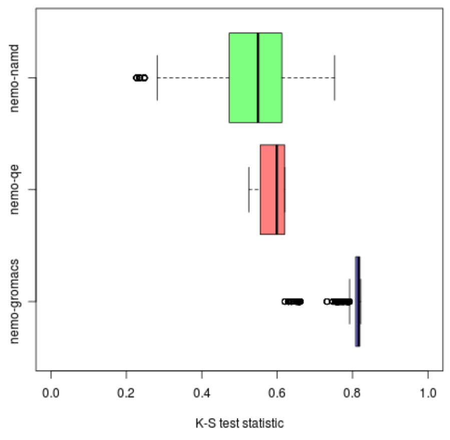
Figure 8.6: Hardware counter events comparison in different GADGET application executions

For GADGET application, the K-S statistic values of the instructions between executions show that these are highly similar, with values between 0 and 0.0004. In this case, the corresponding p-values of all the comparisons are equal to 1, indicating that the null hypothesis can be accepted with high confidence, i.e. they are drawn from the same distribution. Therefore, in this case it can be assumed that there are no significant differences.
For L2 cache misses the values are higher than the instruction, going from 0 to 0.3 (again with p-values equal to zero).

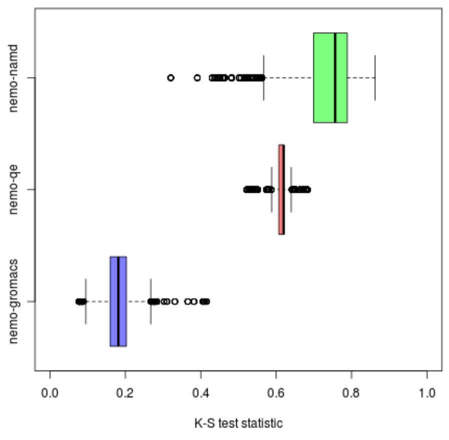
Based on the results of all hardware counters and all applications, it can be seen that a general threshold (similarity value) for determining when two threads belong to the same cluster or not cannot be established.
Indeed, applications such as ALYA are inherently more volatile (with values between 0 and 0.8 in some cases) than applications like GADGET or CP2K, which present lower values than all other applications.

Similarity between different application executions

In order to see if the previous observations have a significant meaning, random threads from different applications were compared for detecting if real differences between the comparisons exist.



(A) L1 cache misses

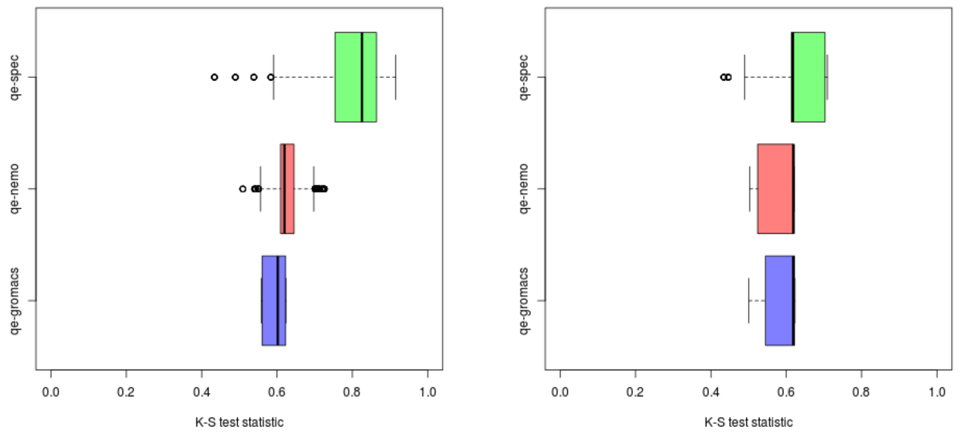


(B) L2 cache misses

FIGURE 8.7: Hardware counter events comparison between NEMO and other applications

Figure 8.7 shows the comparison between NEMO and NAMD (green), NEMO and QuantumEspresso (red) and NEMO and GROMACS (blue), regarding number of instructions

(A) and L2 cache misses (B). As before, the ranges of the values differ depending on the hardware counter events and the applications compared. L2 cache misses of NEMO and NAMD are quite close compared with the other values. However, in general they tend to be further than 0.4 in all hardware counters and applications.



(A) Clock cycles (B) Branch instructions

FIGURE 8.8: Hardware counter events comparison between QuantumEspresso and other applications

Figure 8.8, shows the comparisons between QuantumEspresso and GROMACS (green), QuantumEspresso and NEMO (red) and QuantumEspresso and SPECFEM3D (blue), regarding clock cycles (A) and number of branch instructions (B). As stated before, it seems clear that the values tend to be higher than the ones from the comparison within an execution of an application and that different executions of the same application.

8.4.3 All together

When the previous observations are put together, there were observed three different patterns among the comparisons of the applications.

The first one, fulfilled by most applications, including some hardware counter events of BQCD, GADGET, CP2K, etc. shows no difference between the threads within the same application execution and between executions, although differences with other applications are clear. In these cases, it is stated that there do not exist significant differences among the threads. See the figure below:

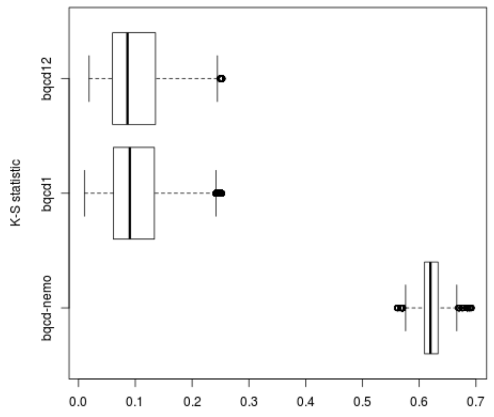


FIGURE 8.9: BQCD comparison between executions 1 and 2, within execution 1 and with NEMO

It can be seen that the range of values for the comparison of threads within the first execution of BQCD is very similar to the range of values from the comparisons of two BQCD executions.

The second pattern, fulfilled by GENE and QuantumEspresso, shows differences between the threads within the same application execution and between executions, being the second ones more similar than the first ones. For example:

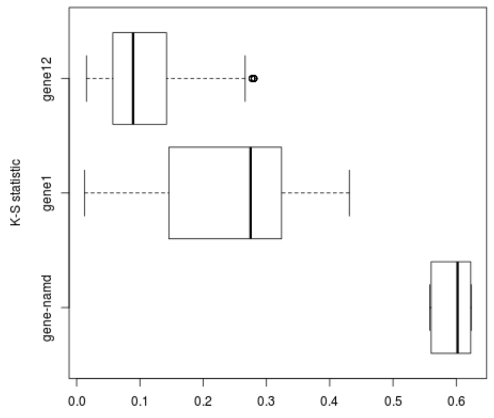


FIGURE 8.10: GENE comparison between executions 1 and 2, within execution 1 and with NAMD

Approximately the 50% of the comparisons from execution 1 of GENE exceed the maximum value of the comparison between two executions.

The third pattern, fulfilled by ALYA, LINPACK, etc. shows that the main range of values is similar as the first type of pattern. However, there are long tails of outliers within their executions.

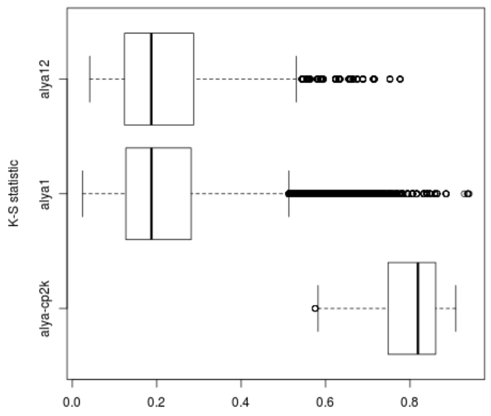


FIGURE 8.11: ALYA comparison between executions 1 and 2, within execution 1 and with CP2K

In the last two cases, real significant differences between the threads exist, as well as it may be possible to detect thread clusters.

8.4.4 Other observations

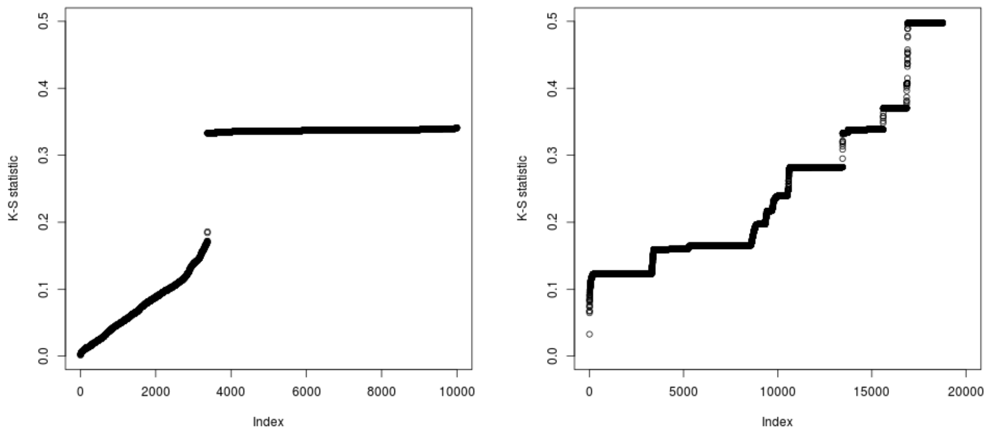
Architectural and application hardware counters

The preliminary results from the executions of some applications with 32 to 256 cores seemed to show a difference between the comparison of application and architectural hardware counter events. The values tend to be larger and more disperse for the architectural ones. However, the final analysis with all the applications executed with 1,024 cores did not show these differences that clear.

Application events are those that do not depend on the environment, i.e. they are inherently connected with the application. For example, the number of instructions and branch instructions executed or floating-point operations performed in an application are always the same, it does not matter where or when the application is executed, they have to be executed a determined number of times and this number must be constant. This is not what is observed with architectural events such as clock cycles or L1, L2 or L3 cache misses. Their behaviour depend on where they are executed and the state of the machine in a specific moment, as they are dependant on the environment. For example, the operating system may decide to keep a process on the CPU more or less time depending on external factors. This may lead to more volatile values, as they change in every execution.

Gaps within application comparisons

It was observed that in the applications that showed signs of possible clusters, GENE and QuantumEspresso, when performing a pairwise comparison between all its inter-application threads, depending on the hardware counter analysed there can be observed clear gaps among the comparison values. This strongly indicates the probable presence of a pattern. For example, see the shape of the sorted values from the comparison of the number of instructions in both applications:



(A) GENE (B) QuantumEspresso

FIGURE 8.12: Gaps on the comparisons of instructions in GENE and QuantumEspresso applications

In figure 8.12a the gap is obvious. In each plot, there are $\frac{1,024 * 1,023}{2} = 523776$ points in total but, just around the values 0.18 and 0.33 there is no single point. A similar thing happens with QuantumEspresso, although it is not that clear, there can be appreciated zones with few values compared to the other high dense parts of the graphic. Based on these gaps, it is easier to identify the threshold which indicates whether two threads belong to the same cluster or not. It is going to be a reliable threshold if its value is greater than the values from the comparison between another execution of the same application. In this case, it would indicate that there are what can be considered, in this case, significance differences among the threads of an application.

Instructions and Branch instructions

It was found that the behaviour between number of instructions and branch instructions is highly similar. They have very strong correlation, as well as slow values of the K-S statistic when both similarity samples are compared.

Application	**Correlation**
ALYA	0.99
QuantumEspresso	0.97
GROMACS	0.91
CP2K	0.98

FIGURE 8.13: Correlation values between instructions and branch instructions with ALYA, QuantumEspresso, GROMACS and CP2K

This high correlation values between both hard counter events is fulfilled by all applications.

8.4.5 Pattern similarity conclusions

Based on all the results, every time that a clustering has to be performed, two Paraver trace files corresponding to two executions of the application are needed. This is for detecting when the comparison values can be considered as similar or not.
When the comparisons between different executions and the same executions have similar values, it is concluded that there are no significant clusters.
If there are differences, the program tries to find the gaps among the values in order to establish that value as threshold.
If no gaps are detected within the comparisons, the maximum value from the comparison between executions is considered as the threshold.
For applications that present weird behaviors, such as SPECFEM3D, for which the comparisons between executions ranges from 0 to 0.8 (completely different), the visual support of Paraver is required, as there is hard to identify reliable clusters in such a volatile application.

Chapter 9

Clustering

There were a lot of considered possibilities as clustering approaches, there is a good review of state of the art in the clustering algorithms at [17], although other options such as DBSCAN [16] were also considered. Besides the ones explained below, prototypes of evolutionary algorithms such as [41], as well as search based techniques like [42] were also implemented. However, after the first tests, it was clear that the time needed for its completion was highly superior than the expected one, being hard to cluster applications executed with merely 128 cores.

The chosen algorithms were selected for performing well with large data sets, as well as reporting solid solutions in the firsts tests. They were modified in order to adapt them to the specific requirements of the desired clustering 2. Afterwards, they were tested with the final application executions, see chapter 10 for the results.

9.1 Leader

Problem definition

The leader algorithm was proposed by Hartigan in [31]. The idea is to provide a fast way for constructing a partition of the given objects, i.e. a hard partitional clustering. It can be considered a greedy algorithm that, in general, do not reach the optimal solution, but it is due its greediness that it is really fast, being a suitable option for calculating a first initialization for more complex algorithms or as a unique option in large data sets.

Given n objects from a data set and a threshold that indicates whether two objects belong to the same cluster or not, it assigns one leader for each cluster and constructs the clusters based on the distance of other objects to the leaders.

It is an incremental algorithm, i.e. it works with an item at a time and decide how to cluster it given the current set of objects that have already been processed. This can lead to different clusterings depending on the order in which the data is processed.

Algorithm

Input parameters:

data set of n objects

threshold: number that indicates when two ojects do not belong to the same cluster (the distance between the threads is greater than the threshold).

```
leaders[1] = object 1
foreach object in data set:
    foreach leader in leaders:
        if distance between object and leader < threshold:
            assign the object to the cluster of the leader
            break this bucle
        endif
    endfor

    if object was not assigned to a cluster:
        assign the object to a new cluster
        assign the object as the leader of the new cluster
    endif
    endfor
endfor
```

Improvements

1. Instead of selecting the first cluster for which the distance between the objects is less than the threshold, traverse all leaders and chose the cluster with the minimum distance.

2. Instead of selecting one fixed leader for each cluster, select 3 random leaders and compare them to the object to be grouped. If 2 or more values are less than the threshold, accept it to the cluster. This improvement can also be applied with the previous one.

Complexity

With the original algorithm, in the worst case the cost in time is $O(n^2)$. This is because the worst scenario is when each cluster is formed for only one object. In this case, each

time an object has to be assigned to a cluster, it checks the distances to the previous objects before creating its own cluster, this results in $\frac{n(n-1)}{2}$ comparisons. However, this is a really specific case and, based on the results of the analysed data, the number of clusters for each application is always much smaller than n.
The cost in space is always $O(n)$, as it is just needed to store the n elements and the leaders.

The costs remain the same with the improvements, as the

9.2 Fast algorithm for K-medoids clustering

Problem definition

It is an algorithm for K-medoids clustering, running in a similar way as K-means [11]. It was proposed by H. Park et al. [13]. It is an improvement for large data sets over typical K-medoids algorithms such as PAM [14] or CLARA [14].

Given n objects from a data set and an integer $k \leq n$. It consists in a local heuristic, i.e. it moves from solution to solution in the space of candidate solutions by applying local changes, until a (local) optimal solution is found. It starts with an initial set of medoids and adjusts the remaining objects in order to maximize the cost function.

Algorithm

Input parameters:

data set of n objects

k: number of clusters. It is initialized with the leader algorithm.

```
mincost = calculate the cost of the solution
while (1):
    newmedoids = for each cluster find the object that minimizes the total
                 distances to the other objects of the cluster
    newcost = calculate the cost of newmedoids
    if newcost == mincost:
        return newmedoids
    else:
        mincost = newcost
    endif
endwhile
```

The function that calculates the cost of a given solution is defined as the sum of the distances from all objects to their closest medoid. This function has to be minimized in order to find the optimum.

The main part is to select k elements from the data set as the initial medoids. The performance of the algorithm may vary according to the method for selecting these elements. The ones tried in this project are the following (see the results of the performance and execution time of each one in chapter 10):

1. Leader initialization gives a good starting point, a reliable solution from which this K-medoids algorithm can locally search for better ones.
2. Random selection of the medoids among all the possible objects.

Complexity

The performed operations for each iteration are:

Initialization of medoids takes constant time, $O(n)$, with either the leader initialization or the random selection of medoids, as they only check each object one time.

Minimization of distances takes $O(n^2)$, as it needs to compare pairwaise the elements of each cluster.

Calculation of the cost of a solution takes $O(nk)$, as it calculates the distance of each point to all medoids and chooses the closest one.

In each step of the algorithm, the space between medoids and the remaining objects decreases. As there is a finite number of possible assignments, it has to converge at some point.

Assuming that the algorithm converges after I iterations, the total cost is: $O(n) + O(In^2) + O(Ink) = O(Ikn^2)$.

With the implemented solution, the cost in space is $O(n^2)$, because each time a distance is calculated, i.e. the K-S test is applied to a sample, its value is stored. As this iterative algorithm calculates distances to medoids and to the other objects of their cluster, it would be expected that the same distances are used numerous times. By storing them, when they have to be used again, the access to the memory position is immediate.

9.3 CLARANS

Problem definition

The CLARANS global searching algorithm was proposed by T. Raymond and H. Jiawei [15]. It accomplishes to cluster data more time efficiently than its predecessors PAM and CLARA with the same performance.

Graph abstraction of the problem:
Given n objects from a data set and an integer $k \leq n$, the process of finding k medoids can be viewed as a searching through a graph. The graph is denoted by $G_{n,k}$ and a node represents a set of k objects (the selected medoids) $\{O_1, .., O_k\}$. The set of nodes in the graph is the set $\{\{O_1, ..., O_k\} | O_i \in \text{data set}, 1 \leq i \leq k\}$.

Two nodes are connected by an arc (i.e. they are neighbors) if their sets differ by one object. Formally, the node $S1 = \{O_11, ..., O_1k\}$ and the node $S2 = \{O_21, ..., O_2k\}$ are connected if and only if $|S1 \cap S2| = k - 1$. Therefore, each node has $k(n-k)$ neighbors. Note that, as each node represents a collection of k medoids, then each node corresponds to a clustering of the data. That is because each object that is not part of the set of medoids can be assigned to the closest medoid based on the similarity function between objects (explained in 8), giving the full clustering.

CLARANS is considered a randomized algorithm. Approaches like PAM [14] lead to high costs for large values of n (large data sets) and k, because all $k(n-k)$ neighbors of a node have to be analyzed, and this is time consuming. CLARANS does not check every neighbor of a node, but it draws a sample of neighbors in each step of a search.

Algorithm

Input parameters:

data set of n objects

k: number of clusters It is initialized with the leader algorithm.

numlocal: number of local minima obtained

maxneighbor: maximum number of checked neighbors with worse cost than the current solution

```
current = select a random node
for i = 1 to numlocal:
    for j = 1 to maxneighbor:
        S = select a random neighbor of the current node
        D = differential cost between current and S
        if S is a better option:
            current = S
            j = 1
        endif
    endfor

    if cost of the current node < mincost:
        mincost = cost current node
        bestnode = current node
    endif
endfor

return bestnode
```

Different options regarding the initialization of a node were tested: the evaluation of solutions using alternative clustering indexes (cost) different than the ones proposed in the original paper and the values of numlocal and maxneighbor.
The proposed options are the following (the information about their performance can be found in chapter 10):

Dunn index is well-known clustering validation index.

Silhouette is a robust clustering validation metric. It is also used for the cluster validation of this project.

Sum of medoids distance is the sum of the distance from each object to its closest medoid.

Complexity

Each node has $k(n-k)$ neighbors that can be examined (as they are picked randomly, not locally). The selection of random nodes has cost $O(k)$ and the cost functions $O(n^2)$, as they have to compare the objects from clusters pairwaise. Therefore, in the worst case, in each iteration the algorithm checks practically all the neighbors before finding a better option.
The total cost is: $O(numlocal * maxneighbor * n^2)$.

As in the previous solution, the cost in space is $O(n^2)$ because each time a distance is calculated, it is stored to memory.

Chapter 10

Cluster validation

After the clusters are created, it is recommended to evaluate the quality of the clustering. This can be achieved by using a metric for evaluating clustering algorithms. Eréndira Rendón et.al. wrote good review on this topic [43]. Two well known indexes that can be applied to the kind of clustering of this project are Dunn index [44] and silhouette [32]. These were both tested with different real solutions from the clustered applications, but it was concluded that the Dunn index is too sensitive to miss-placed objects, giving very low values for clusterings that, in this case, could be considered reliable. Silhouette is more robust in this aspect, as it considers all the differences between all the threads in different clusters and within the same cluster.
A better explanation of silhouette can be found in appendix B.

The following sections show the different quality clusterings and execution times for its computation for the algorithms (and the possible improvements over them) discussed in chapter 9. The tests were performed considering the executions of the applications with 64, 128, 256, 512 and 1,024 cores.

As explained in chapter 8, the applications that showed clear and higher values between them and different executions of the same application were used for the tests.

10.1 Results

10.1.1 Leader algorithm

As explained in chapter 9, the leader algorithm pretends to perform fastly with large data sets but, in exchange, its greediness can lead to worse solutions.

For this algorithm, a version that assigns a thread to the same cluster as the first leader for which the distance is lower than the threshold, named greedy leader, and a version that, for each thread, traverses all the leaders and chooses the closest one, named best leader, were tested. It was also tested an improvement over the best leader, named random leaders, that instead of selecting one fixed leader per cluster, it chooses three leaders randomly from a cluster and accepts a new object to that group if at least two comparisons are below the threshold. The obtained results are following:

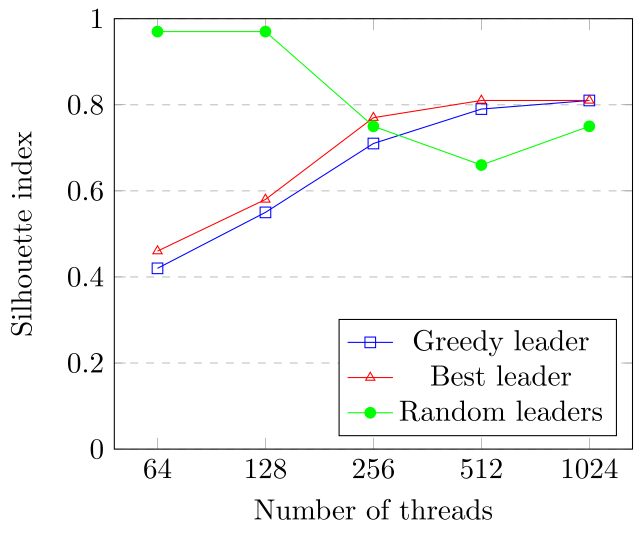


FIGURE 10.1: Leader algorithm execution quality clustering with GENE instructions

The greedy leader and the bets leader strategies perform similarly but, as it was expected, the version that incrementally checks the distance between all the leaders and chooses the closest one has higher values than the version that just assigns a thread to the first leader for which the distance is under the threshold.

The random leaders performs worse than expected, being not better than the other two options when there is a larger number of threads.

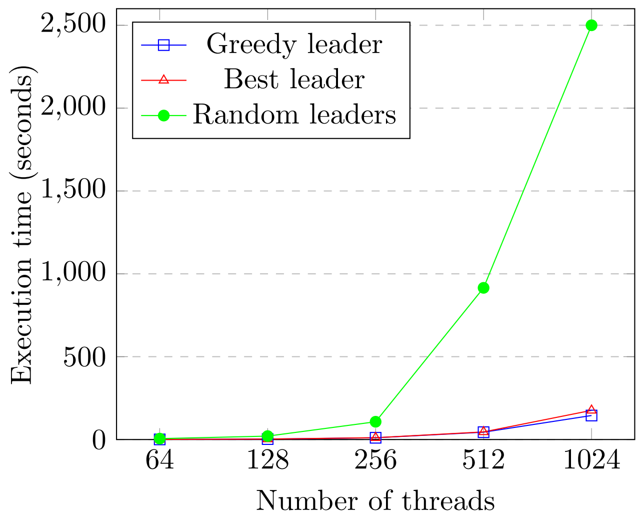


FIGURE 10.2: Leader algorithm execution time comparison with GENE instructions

Figure 10.2 shows that the greedy leader executes with less time than the others. However, the differences between the greedy and the best leader are not remarkable, being positive to sacrifice a small amount of time for the leader version that chooses the best leader in order to achieve a better clustering quality.
With not many threads, the random leaders is better than the other ones, although it always needs a lot more computational time as the number of threads increase in order to calculate the results.

For the next algorithms, the best leader strategy is used as initialization of the medoids because it is fast and it gives relatively reliable results.

The results from QuantumEspresso showed the same behavior with the greedy leader and best leader approaches and, in general, it gave bad results for the random leaders.

10.1.2 Fast K-medoids

For this version of K-medoids, as it is based on local searching, the initialization of the medoids play a key role for converging to a good optimum value with as less time as possible.

The chosen initialization functions are: one picks one element from each cluster of the solution given by the leader algorithm (leader initialization) as medoids and another one that picks k medoids randomly.

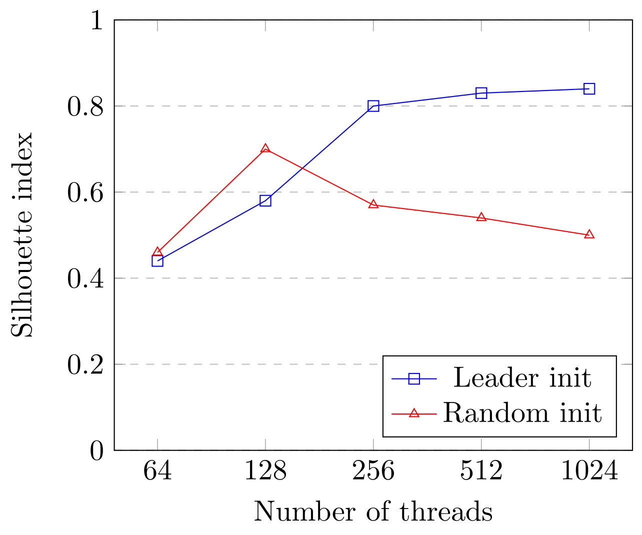


FIGURE 10.3: K-medoids improved algorithm quality clustering comparison with GENE instructions

Based on figure 10.3, it seems that the initialization of the medoids based on the execution of the leader algorithm tends to give better results than the random initialization. In QuantumEspresso, all the silhouette values for the leader initialization are higher.

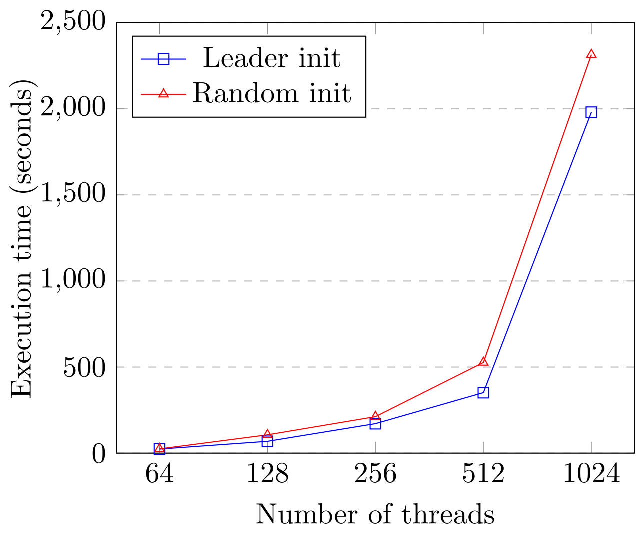


FIGURE 10.4: K-medoids improved algorithm execution time comparison with GENE instructions

It is clear that the random medoids initialization lasts some more time than the one based on the leader, being 7 minutes the difference in execution time with 1,024 cores and, as stated before, with worse solutions.

These results are due to that, as this version of K-medoids consists in local searching, the clustering performed by the leader algorithm is generally closer to a better local optimum than the randomly selected medoids. These results were expected, as our clustering is based on a threshold of similarity, it is normal that the leader algorithms is close to a good optimum. Therefore, the elected version for the fast K-medoids algorithm is the initialization with the leader algorithm.

10.1.3 CLARANS

CLARANS, as the algorithm of the previous section, is an algorithm based on K-medoids. While the previous one performs a local search, CLARANS searches in a wider space of solutions, as it picks them randomly from a given solution and checks whether its random neighbors have a better cost. With this strategy, it is easier to reach a global optimum.

There are various parameters that can be tested in order to obtain a better performance related to time and quality of the clustering. The first one is the number of traversed local minimums and the maximum number of neighbours to be checked for each local minimum. The second test pretends to find out if there is a real difference between initializing the medoids randomly or with the leader algorithm. The third one checks different metrics for clustering in order to observe their performance.

Local minimums and maximum neighbors

Different values of these two input parameters are tested. The results reported in the CLARANS first article, show that low values of local minimums (between 2 and 5) and higher values of maximum neighbors (between 250 and 1,000) lead to reliable results. However, as in this case it is hard to compare two objects due to the big number of samples, low values of maximum neighbors were also used. The results are the following:

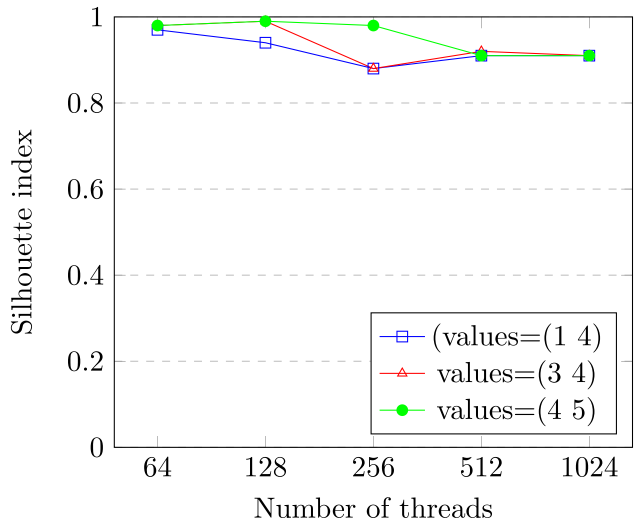


FIGURE 10.5: CLARANS quality clustering comparison between different numlocal and maxneighbor values with GENE instructions

As it was expected, higher values of both parameters lead to better solutions. Based on the blue line, it seems that CLARANS performs better when looking for local optimums, as the number of local minimums is equal to one, thus it performs a local search. In this case, it obtains better results than the previous algorithm, fast K-medoids, that also uses local searching.

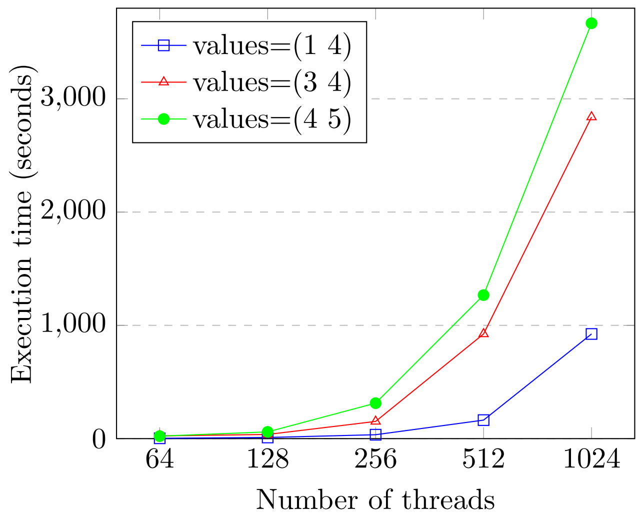


FIGURE 10.6: CLARANS execution time comparison between different numlocal and maxneighbor values with GENE instructions

It is clear that the greater the values, the greater the execution time. However, the time spent increases faster than expected. With values of local minimum of 3 and 4 and maximum neighbors 4 and 5, the time needed for the clustering exceeds 2,800 seconds (45 minutes), being of near 950 seconds (16 minutes) the execution time for values 1 and 4. Based on these observations, it is better to use the lowest value for the local minimum, as the algorithm seems to perform well for finding local optimums (almost as good as the other tested values) and it lasts much less.

Medoids initialization

For initializing the medoids, as done before with the fast K-medoids algorithm, a version with a leader initialization and another one with a random initialization were assessed. Both use localmin=1 and maxneighbor=4.

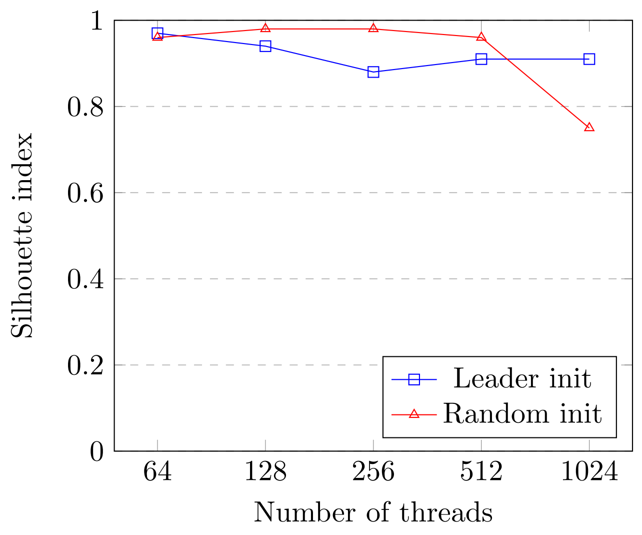


FIGURE 10.7: CLARANS quality clustering comparison between different medoid initializations with GENE instructions

Both initializations give acceptable silhouette values, but the random version behaves better then the leader one with 128, 256 and 512 cores. However, with 1,024 it performs quite low compared with the leader. Although starting from a random solution may lead to a suitable optimum, this is not assured, and sometimes it may give bad solutions due to its initial random position, as the algorithm only checks for one local minimum.

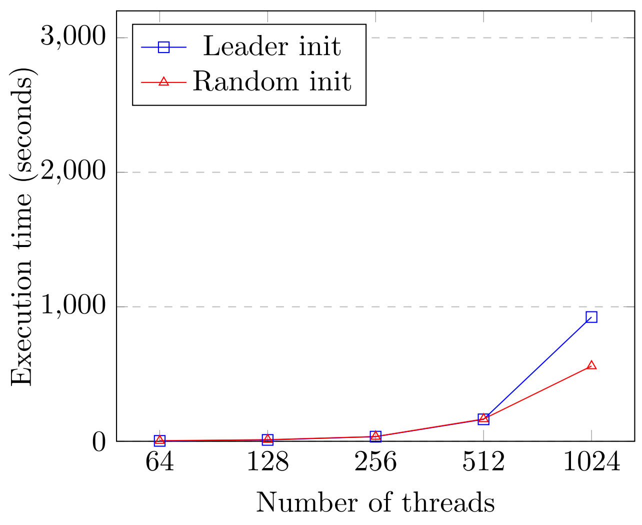


FIGURE 10.8: CLARANS execution time comparison between different medoid initializations with GENE instructions

With higher values of threads, it seems that the leader initialization tends to spend some more time, but it also depends on the random values chosen for the random initialization. Based on these results, a good option would be to use the leader initialization, as it gives a good position for reaching a good local optimum, although it might not the best one; while with random values it is not known whether it is going to chose a good position or not. Another solution would be to use a localmin=2 in order to be able to explore another solution.

Quality functions

Another important part of the CLARANS algorithm is to be able to identify a reliable solution precisely. The cost function used for the previous experiments correspond to the sum of distances between every thread and its closest medoid. With the following tests, this function is compared to the Dunn index and the silhouette metric, both explained at the beginning of this chapter.

The test was stopped after evaluating the execution with 128 cores because the computations lasted too much (15 minutes), an unacceptable time for clustering just 128 threads. The values obtained were the following:

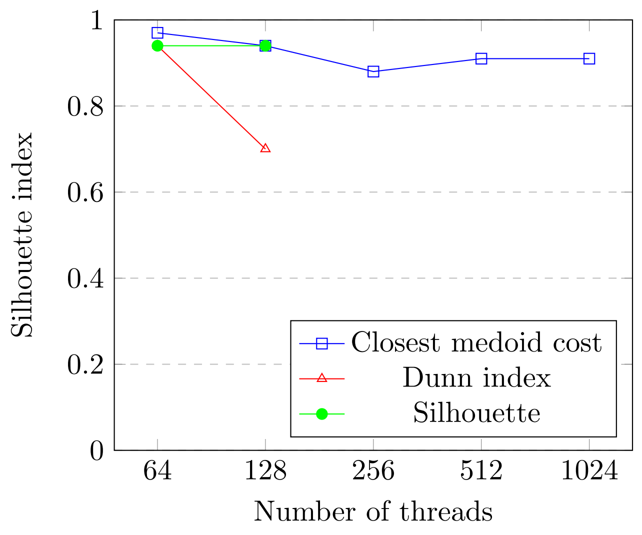


FIGURE 10.9: CLARANS quality clustering comparison between different cost functions with GENE instructions

It is important to note that the clusterings obtained with QuantumEspresso match perfectly with all the previous observations.

Based on the results from the tests, it seems a reliable option to use the CLARANS algorithm with the best leader initialization, localmin=1, maxneighbor=4 and with the closest medoid distance cost.

10.2 Final results

Putting all together, the best options for each of the three algorithms are compared regarding their silhouette value and the execution time:

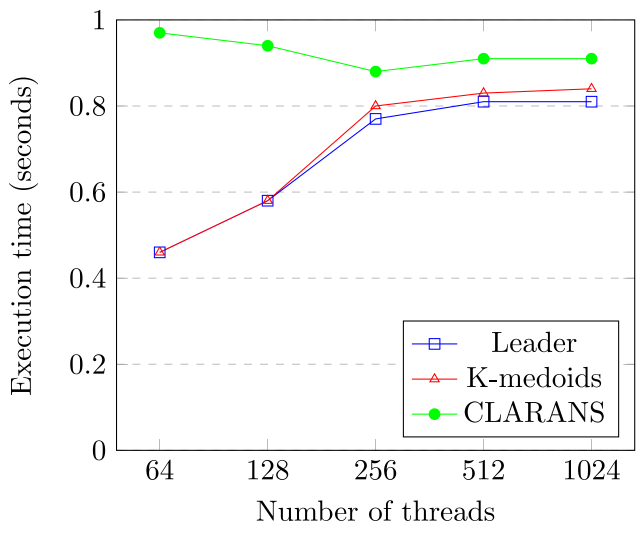


FIGURE 10.10: Silhouette values for the leader algorithm, K-medoids and CLARANS with GENE instructions

Figure 10.10 shows that the performances of the best options for the leader algorithm and the fast K-medoids are highly similar, meaning that the leader algorithm is near an optimum solution. CLARANS achieves clearly a greater clustering quality than the other two options.

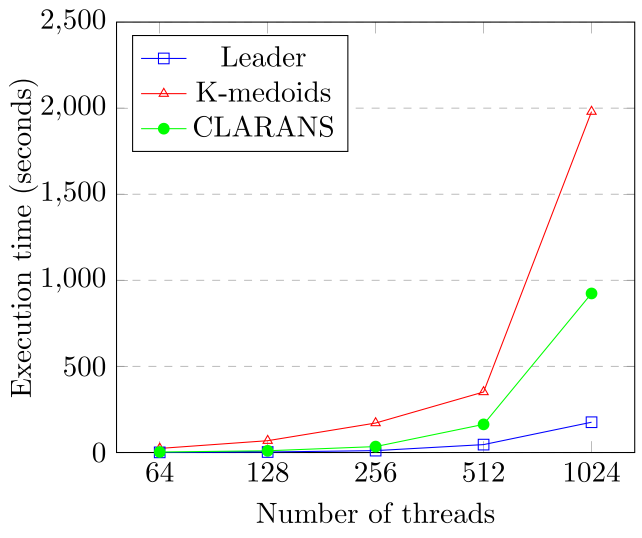


FIGURE 10.11: Time execution for the leader algorithm, K-medoids and CLARANS with GENE instructions

Figure 10.11 shows that the K-medoids is the algorithm that needs more computational time and, based on the previous figure, its performance is not far from the leader algorithm, that needs much less time.

CLARANS' execution time is in between the other two algorithms, while the performance is highly superior. With 1,024 threads it can find a solution with approximately 15 minutes.

Based on these results, it seems a good option to choose CLARANS as the better clustering algorithm, although in some circumstances where it is preferable a fast execution than a good performance, the leader algorithm is a reliable option.

Chapter 11

Conclusions

It has been shown that significant differences between the threads involved in an execution of a hard-performance computing application may exist, although such differences are not always perfectly clear and a profound analysis is required in order to detect them. This process requires two trace files obtained with Paraver [1], from the application to be clustered. From these files, different hardware counter events have been analysed. These are: number of instructions, clock cycles, L1, L2 and L3 cache misses, number of floating-point operations and branch instructions.

In order to compare the samples of two threads, the Kolmogorov-Smirnov test statistic [30] has been used for establishing a value of similarity between them. It has been shown that the applications for which such clusters exist seem to show certain gaps when they are all compared pairwise. These gaps can be used for establishing a threshold of similarity between pairs of threads, which indicates whether two threads belong to the same cluster (less distance than the threshold) or not (greater distance). If no gaps are detected with the comparisons within the application, the threads from both executions have to be compared in order to establish the threshold, i.e. the maximum distance value from the two-execution comparison.

An artificial intelligence has been developed in order to give a clustering of the threads when clear groups are recognized within an application. As there is a lot of data to be analysed, it is important to use techniques that compare as less threads as possible. Three clustering algorithms have been carefully chosen and adapted to the project's requirements in order to obtain more robust clusters in less time. These are: the leader algorithm [31], an two K-medoids like algorithms for large data sets, a simple and fast algorithm [13] based on local searching and another named CLARANS [15] based on global searching. In order to state the strengths and weaknesses of each algorithm, different tests have been developed for such purpose.

The tests show that the algorithm with the best performance regarding clustering quality is CLARANS, although it requires some more computational time than the leader algorithm. The fast version of K-medoids showed no big differences in quality compared to the leader algorithm and it is the option that needs more time for finding the solutions, being it the worst of the options.

Chapter 12

Future work

12.1 Multivariate test for non-parametric data

The Kolmogorov-Smirnov test that was used to compare samples from two threads can only be used in one dimension. Therefore, it is just able to compare the threads based on one of the seven hardware counter events measured in the applications (number of instructions, clock cycles, L1, L2 or L3 cache misses, number of floating-point operations or branch instructions). Although at R package with the Cramer-von Mises criterion [34], i.e. a distribution-free test that accepts multiple dimensions, the time required for each comparison was too high.

It still remains to find a test or divergence method capable of performing well with non-parametric data and in a short time with multiple dimensions.

12.2 Architectural vs application hardware counters

The preliminary results obtained with the execution of some of the applications with 32 to 256 cores seemed to indicate a difference between the comparison of application and architectural hardware counter events, but the final analysis with all the applications executed with 1,024 cores did not show these differences that clearly. See section 8.4.4 for more information about architectural and application counters.

However, as stated before, these differences were not clear with the last analysis. It remains pending to perform a deeper analysis in order to find out whether they exist or not.

12.3 Deeper analysis of the applications

There could be other ways of detecting clear significant clusters within an application. Performing more types of analysis and tests would be a good option of finding them in order to use this knowledge for incorporating it to the actual clustering algorithms.

12.4 Test more applications

In order to find new clues about other kinds of patterns followed by the threads from an application execution, it would be preferable to execute more applications in order to perform similar and new kinds of analyses to them.

Appendix A

Kolmogorov-Smirnov test

In this project, the Kolmogorov-Smirnov test (K-S test) [30] has been used for comparing the samples of the thread traces. It is a nonparametric test of the equality of one-dimensional probability distributions. It can be used to compare an empirical sample with a reference continuous probability distribution (one-sample K-S test, using it as a goodness of fit test) or to compare two empirical samples (two-sample K-S test).
In the two samples case, the null distribution of the K-S statistic is calculated under the null hypothesis (H0) that the samples are drawn from the same distribution.

The two-sample K-S statistic determines its value based on the maximum distance between the empirical distribution functions (ECDFs) of the respective samples. An empirical distribution function is the cumulative distribution function associated with the empirical measure of the sample. It is usually denoted by $\hat{F}_n(x)$ or $\hat{P}_n(X \leq x)$ and it is defined as

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} I(X_i \leq x)$$

where $X_1, .., X_n$ is an independent and identically distributed (iid) sample from an unknown distribution and $I()$ is the indicator function, defined as

$$I(X_i \leq x) = \begin{cases} 1 & X_i \leq x \\ 0 & otherwise \end{cases}$$

In words, the ECDF of a random variable at x gives the probability that the random variable X_i is less than or equal to that number x. See figure...

The two-sample K-S test hypothesis are:

H_0: Samples drawn from the same distribution
H_1: Samples drawn from different distributions

The test statistic is defined as

$$D = sup|\hat{F}_1(x) - \hat{F}_2(x)|$$

where $F1$ and $F2$ are the ECDFs and *sup* is the supremum, i.e. the greatest element of a set.
The null hypothesis is rejected at level α if

$$D > c(\alpha)\sqrt{\frac{n + n'}{n \cdot n'}}$$

where n and n' are the lengths of the samples and the value $c(\alpha)$ varies depending on the chosen significant level. For $\alpha = 0.05$, $c(\alpha) = 1.36$.

Appendix B

Silhouette

Silhouette is a method is a method for clusters validation. It was first described by Peter J. Rousseeuw in 1986 [32].

For each object i, $a(i)$ is the average dissimilarity between i and all other objects that are in the same cluster. Let $b(i)$ be the lowest average dissimilarity of i to one of the other clusters, not including i's cluster. Silhouette is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Based on the silhouette definition, it can take values between $-1 \leq s(i) \leq 1$. When $s(i)$ is closer to 1, it is required that $a(i) \ll b(i)$. If $s(i)$ is close to one it means that the object i well clustered, this is because as $a(i)$ indicates the dissimilarity of i to its own cluster, a small value means that it is well matched. Furthermore, if $b(i)$ is a large value, it means that the distance from i to its closest neighbor cluster is big.
Alternatively, when $s(i)$ is close to minus one, then by the same logic it is not well clustered. If $s(i)$ is close to zero indicates that the object is on the border of two clusters.

Bibliography

[1] Barcelona Supercomputing Center. Paraver: a flexible performance analysis tool., . URL http://www.bsc.es/computer-sciences/performance-tools/paraver/general-overview.

[2] M. Geimer, F. Wolf, B.J.N. Wylie, E. Ábrahám, D. Becker, and B. Mohr. The scalasca performance toolset architecture. *Concurrency and computation: Practice and experience*, (22):702–719, 2010.

[3] M.N. Tuma, S. Scholz, and R. Decker. The application of cluster analysis in marketing research: A literature analysis. *Business Quest Journal*, (14), 2009.

[4] NIH Bethesda Maryland Biometry Branch, National Cancer Institute. The detection of disease clustering and a generalized regression approach. *Cancer Research*, (27), 1967.

[5] France Laboratoire de Génétique Cellulaire, INRA Toulouse. Multiple sequence alignment with hierarchical clustering. *Nucleic Acids Research*, (16):10881–10890, 1998.

[6] C.T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, C-20(96):68–86, 1971.

[7] N.R. Pal, J.C. Bezdek, and E.K. Tsao. Generalized clustering networks and kohonen's self-organizing scheme. *IEEE Transactions on Neural Networks*, 4:549–557, 1993.

[8] A. Abraham, S. Das, and A. Konar. *Metaheuristic Clustering*. Springer, 2009.

[9] A. Abraham, S. Das, and S. Roy. *Soft Computing for Knowledge Discovery and Data Mining*. Springer, 2008.

[10] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall College Div, March 1988.

[11] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability 1*, (22):281–297, 1967.

[12] A.K. Jain. Data clustering: 50 years beyond k-means. *19th International Conference in Pattern Recognition*, (31):651–666, 2010.

[13] H.-S.. Park and C.-H. Jun. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, (36):3336–3341, 2009.

[14] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: an Introcution to Cluster Analysis*. John Wiley $ Sons, 1990.

[15] T.Ng Raymond and H. Jiawei. Clarans: a method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, (14):1003–1016, 2002.

[16] E. Martin, H.-P. Kriegel, J. Sander, and X. Xu. *A density-based algorithm for discovering clusters in large spatial databases with noise*. Number 14. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996.

[17] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[18] Barcelona Supercomputing Center. Barcelona supercomputing center., . URL http://www.bsc.es.

[19] RStudio. Rstudio. URL http://http://www.rstudio.com.

[20] Sabalcore. Sabalcore. URL http://www.sabalcore.com.

[21] R Project for Statistical Computing. R. URL http://www.r-project.org.

[22] Linux Mint. Linux mint. URL http://www.linuxmint.com.

[23] Barcelona Supercomputing Center. Marenostrum iii system architecture, 2013. URL http://www.bsc.es/marenostrum-support-services/mn3.

[24] UPC. Tecnologia per a tothom. URL http://txt.upc.edu/index.php.

[25] S. Morgan. *Waste, Recycling and Reuse*. Evans Brothers, December 2006.

[26] ASUS. Asus recycling. URL http://recycling.asus.com/ASUSRecycle/PageContentShow.aspx?page=AboutSite.

[27] Partnership for Advanced Computing in Europe (PRACE). Unified european applications benchmark suite, July 2013. URL http://www.prace-ri.eu/ueabs.

[28] LINPACK HPC Benchmark. Linpack hpc benchmark. URL http://www.netlib.org/linpack.

[29] Prace research infrastructure. Partnership for advanced computing in europe (prace). URL http://www.prace-ri.eu.

[30] H. Cramer. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, pages 68–78, 1951.

[31] J.A. Hartigan. *Clustering Algorithms*. Johm Wiley & Sons, 1975.

[32] P.J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Computational and Applied Mathematics*, (20):53–65, 1987.

[33] S. Geisser and W.O. Johnson. *Modes of Parametric Statistical Inference*. John Wiley & Sons, February 2006.

[34] H. Cramér. On the composition of elementary errors. *Scandinavian Actuarial Journal*, pages 141–180, 1928.

[35] A. Justel, D. Peña, and R. Zamar. A multivariate kolmogorov-smirnov test of goodness of fit. *Statistics & Probability Letters*, pages 251–259, 1997.

[36] C. Franz. *Multivariate nonparametric CramerTest for the two-sample-problem*. R Foundation for Statistical Computing, 2014. URL http://cran.r-project.org/web/packages/cramer/index.html.

[37] R.M. Craparo. *Significance level*. Sage publications, ...

[38] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 6: 80–83, 1945.

[39] E. Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, pages 210–271, 1909.

[40] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 1(22):79–86, 1951.

[41] S. Das, A. Chowdhury, and A. Abraham. A bacterial evolutionary algorithm for automatic data clustering. *Evolutionary Computation*, pages 2403–2410, 2009.

[42] T. Hofmann and J. Buhman. Pairwaise data clustering by deterministic annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (19):1–14, 1997.

[43] E. Rendón, I. Abundez, A. Arizmendi, and E.M. Quiroz. Internal versus external cluster validation indexes. *International Journal of Computers and Communications*, (5):27–34, 2011.

[44] J.C.Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics 3*, (3):32–57, 1973.