POLITECNICO DI MILANO

FACOLTA' DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Specialistica in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

STUDY AND MODELLING OF AN APPROXIMED MEMOIZATION SYSTEM FOR FINANCIAL APPLICATIONS

Relatore: Prof. CHIARA FRANCALANCI

Correlatore: Prof. GIOVANNI AGOSTA

Ing. MARCO BESSI

Tesi di laurea di:

ROBERTO MASSIMINI

Matr. 753025

Anno Accademico 2011 - 2012

A papá che ha sempre creduto in me e che dal Cielo mi protegge.
A mamma, per tutti i sacrifici che ha fatto.

Ringraziamenti

Il primo ringraziamento va a mia mamma, Mariella, che nonostante tutto quello che ci é capitato, non mi ha mai fatto mancare la sua fiducia e il suo supporto. Il secondo ringraziamento va a mio papá Vincenzo, che mi sta guardando dal Cielo e di sicuro sará molto orgoglioso.

Grazie a Cristina per essermi stata vicino e per avermi sostenuto nei momenti in cui il traguardo sembrava irraggiungibile.

Un ringraziamento ad Andrea, Marco, Matteo, Pietro e Roberto, gli amici di una vita, che mi hanno aiutato, supportato (o forse sopportato) in tutto questi anni universitari.

Voglio inoltre ringraziare la mia relatrice, la prof. Chiara Francalanci, per avermi offerto l'opportunitá di realizzare questa tesi.

Un grazie anche ai miei correlatori Giampaolo Agosta e Marco Bessi per avermi seguito in un modo splendido nella realizzazione di questa tesi e per avermi sempre supportato e aiutato nei problemi che di volta in volta si incontravano. Un ringraziamento anche all'ing. Eugenio Capra, che mi ha seguito nella prima parte di tesi per poi affrontare una nuova avventura lavorativa.

Infine un Grazie davvero grande anche a tutti quelli che non sono stati espressamente citati ma che mi hanno accompagnato in questi ultimi anni e che hanno avvicinato un pó questo traguardo.

Grazie.

Roberto.

Abstract

The information comunication and technology sector (ICT) emits each year the same amount of CO_2 as all the word aviation industry. A green ICT can not be obtained only working on the hardware. The software has an important role in its entire life cycle, starting from the use phase until the end of the computer life. The role of the software in the green ICT it is not only on the monitoring and on the optimization of the resources, but it can going over, allowing, for example, to satisfy new requirements with the existings infrastructures. In this thesis is presented an approach based on the memoization in order to improve the energy and the time efficiency of a function whose goal is to compute the financing payments for a big Italian bank. It was analyzed the function, doing a porting from the original coding language (COBOL) to the Java language. It was applied after the memoization approach, in order to obtain an improvement on the energy consumptions. This approach was modified introducing the concept of approximed memoization, in which an output value is taken from a memoized value if the corrisponding input value belongs to the interval memoized value plus or minus a Service Level Agreement (SLA), choosen by the users. The results of the application of this approach are good, finding a timing percentage gain of the 47% in the worst case (high value precision).

Sommario

Il settore informazione, comunicazioni e tecnologia (Ict) emette ogni anno tanta anidride carbonica quanto tutta l'industria mondiale dell'aviazione e dei trasporti aerei. Una ICT davvero green non puo' essere ottenuta intervenendo unicamente sull'hardware. Il software infatti gioca un ruolo molto importante sull'intero ciclo di vita: sia nella fase d'uso sia nella fine vita dei computer. Il ruolo del software nel green ICT non e' solo monitoraggio e ottimizzazione di risorse, ma puo' andare oltre, consentendo ad esempio di soddisfare nuove esigenze con le infrastrutture esistenti. In questa tesi viene presentato un approccio basato sulla memoizzazione per migliorare l'efficienza energetica e temporale di una funzione appartenente ad un grande istituto bancario che effettua il calcolo degli interessi sui mutui. E' stata analizzata la funzione, effettuando un porting dal linguaggio originale (COBOL) al linguaggio Java. Successivamente e' stato applicato l'approccio della memoizzazione in modo da ottenere un miglioramento riguardante l'efficienza energetica. Tale approccio e' stato modificato introducendo il concetto di memoizzazione approssimata, in cui come valore di output viene preso un valore gia' memoizzato se il valore in input appartiene all'intervallo definito da un valore memoizzato piu' o meno uno SLA (Service Level Agreement) definito dall'utente. I risultati relativi all'approccio sono soddisfacenti, trovando un guadagno percentuale del tempo di esecuzione del 47% nel caso peggiore (precisione dei valori elevata).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The emission of CO_2 of the ICT sector each year is equal to the emission caused by all the world aviation industry. The ICT causes the 2% of the world CO_2 emissions and in the 2020 the previsions are that this value will be the double. So, the green ICT can not concentrate only its studies on the hardware part or on the embedded system. It becomes important to study methods that permits to create a green software, in order to optimize and improve the energy efficiency of the total system (hardware and software). Recent studies had highlighted that the CPU is the contributor of the 60 % of the mean consumption of energy in a server system, more than the disks and the memories. Otherwise, while the power absorbed by the microprocessor is proportional to its use, the one absorbed by the memory and the disks is indipendent on the usage, because it depends on the cyclic update of the dynamic RAM and on the rotation of the disks. From this observation the idea that becomes more and more relevant is that it is possible to use the memory instead of the processor to satisfy the functional requirements, gaining a consistent energy gain. It is possible to apply this approach to some computation intensive applications using some "memoization" tecniques. The term memoization was coined by Donald Michie in 1968 and is derived from the Latin word memorandum (to be remembered), and thus carries the meaning of turning [the results of] a function into something to be remembered. While memoization might be confused with memorization (because of the shared cognate), memoization has a specialized meaning in computing. A memoized function "remembers" the results corresponding to some set of specific inputs. Subsequent calls with remembered inputs return the remembered result rather than recalcu-

lating it, thus eliminating the primary cost of a call with given parameters from all but the first call made to the function with those parameters.

It is not possible to apply to all the functions the memoization approach. This method can be applied to a set of functions called "pure", whose main charatteristics are to be deterministic and they can not produce side-effect.

The goal of this thesis is to analyze a function that computes the financing payments for a big Italian bank and then apply the memoization approach on that function. Clearly, the application of the method is possible only if the function is pure.

In the chapter 2 it is presented the Green IT and the Green Software concepts, then it is described the state of the art related to the memoization techniques. After these concepts, is presented the definition of pure function, join to the description of the framework developed by the Politecnico di Milano, that defines the model that is the start point of the thesis works.

In the third chapter it is described the financing payments function with the reasons that have brought to choose it and the parameters that influence the function behaviour. Then there is an illustration on the problems found and the solutions adopted during the porting from the COBOL to a Java pure function. At the end of this chapter it is described the distribution of the input data and also the memory division adopted for the solution.

The chapter 4 presents the solution idea, the approximated memoization. It is described first the quadtree, the structure choosen to store the data in the memory. Then it is described the previsional model, in the modified version, in order to introduce the concept of the approximed memoization. At the end of the chapter there is an accurate analysis on the function approximation, specially on the best curve fitting method.

The fifth chapter presents the results reached with the memoization approach, speaking first on the how the experiments were conducted, with the presentation of the devices use for executing and then measure the energy consumptions. Then it is presented the section in which are illustrated the savings gained with the application of the memoization.

In the last chapter are shown the conclusions on the works done for the thesis and then there is a subsection in which are described future works that could be done for improving the actual status of the project.

Chapter 2

State of the art

This chapter provides an introduction to the basic concepts that will be referred to throughout all the other chapters. In section 2.1, Green IT main concepts will be introduced and described. Section 2.2 provides a detailed overview about the energetic optimization of the software, called Green Software. In particoular, in this section will be described an approach used to improve the performances and to decrease the consumption of a software method. This approach is called *memoization*. To use memoization the function analyzed has to be pure and in section 2.3 will be presented the definition of *pure function* and will be described the overall project on which this thesis work is based.

2.1 Green IT

The information systems, that nowadays are common in our lifes, have an enviromental impact that can't be overlooked. The computer production needs electricity, chemical products and other things, becaming a cause of pollution. Some studies [23] underline the fact that IT infrastructures are responsible for 2% of the CO_2 world emissions, with an impact equal to the aeronautic industry. Each PC,indeed, generates 1 ton of CO_2 per year, and a server needs the energy that causes the emission of the same quantity of CO_2 coming from a SUV running for 25 km [25]. The strong evolution in the recent years have brought to the market smaller and more powerful processors, with the consequent increment of the processing dissipated power. The common Intel® Core processors dissipate 110 W on the average, with a consumption increment of 110% related to the old

486, that had the incredible consumption of 10W on the average. To get a more concrete idea about the energy consumption of the IT systems, we can consider that a modern server blade uses up to 1 kW, the same as a regular refrigerator at home. It is easy to see that a rack of blade servers, with 5 shelves and 8 units each, will need 40 kW, the same as an appartment building. Given this consideration, a medium datacenter will need 250 kW, like a district, while big data centers, like the one that big banks or ISP have and maintain, will need something like 10MW, the same as an entire town.

2.1.1 Enviromental impact of IT and the origin of Green IT

Green IT is an expression that includes many smaller concepts under it, but generally speaking it addresses all problems related to the environmental impact of the energy consumption related to the IT infrastructures.
We can identify three big macro areas behind it:

- IT energy efficiency
- environment friendly management of the IT lifecycle
- IT usage as a way of improving business process efficiency

The first area is related to the IT energy efficiency, which can be improved focusing on the design and management of the infrustructures and data centers and by changing the companies attitude and way of working.

The second area includes all the actions that are related to an environment friendly management of the IT life cycle, from the production to the disposal. The pollution that the IT industry is responsible for is not only caused by the energy consumption, but includes also the toxicants dispersed in the environment. Some studies have discovered that the 70% of the ground pollution caused by lead, cadmium and mercury comes from, directly or indirectly, the IT.
The *"Waste of Electric and Electronic Equipment"* (WEEE), called also *"e-waste"*, are special type of waste that derive from any type of electric or electronic equipment that is disused because is broken or obsolete. That waste is toxic and not biodegradable and it is going to be an higher risk for the enviroment. In order to address the problem the European Union has delivered some guidelines (WEEE 2002/95/CE) which state some rules for the gathering and the reuse of

those IT waste.The studies on Green IT on this field have the goal to decrease the polluting parts during all IT life cycle. It starts from the production process, for instance trying to optimize the packaging process, and it goes till the end of the life-cycle, trying to elaborate way of gathering and reusing old or broken IT components.

The last area focuses the usage of IT as a instrument of governance, measuring and monitoring the *"green"* parameters (like the power consumption, temperature, paper usage,...) of all the business processes, IT or not IT. Recent studies [25] have proved that 86% of ICT departments in Great Britain do not know their CO_2 emissions and 80% of companies do not know the electric bill. A research study from Politecnico di Milano [7] considered 140 Italian small and medium companies and revealed that 89% of IT administrators have no idea of the consumption of their IT infrastructure. As a matter of fact it is hardly impossible to improve what it is not even known; moreover it is quite hard that an IT administrator is interested in investing in lowering its energy consumptions when these consumptions are not even assigned to the IT infrastructures.

As already mentioned, in order to obtain a greener IT, there must be a strategic view of the company and ther must be some cultural and management changes following it. The enviromental impact of a company should be monitored through appropriate "green" KPI (Key Performance Indicator), that can be used by the company decision makers to monitor and improve the energy efficiency of an industrial process. So the IT can be used in two ways:

- To misure the green parameters using sensors or smart networks able to sample and to analyze the data
- To analyze and summarize the data using data mining tools to support the company decisions

2.1.2 Why is Green IT important?

There are at least three good reasons that can explain why the Green IT is important:

- The IT has an important enviromental impact
- The IT energy consumption costs

- The energy need is a limit to the IT scalability

The paragraph before talked about how the IT has an important enviromental impact both for the energy consumption, and so the emission of greenhouse gas, the first cause of global warming, and for the release of polluting substances in the environment. These gas emissions can cause respiratory diseases, acid rains and global climate changes. The only way of stopping these effects is to reduce global emissions and this can be done also by reducing electric power consumption. Therefore talking about Green IT is first of all a social responsability that nowadays can't be omitted.

The CIO and IT responsible interest on Green IT is increasing because the cost of the energy used by the IT systems is becoming a significant part of the *Total Cost of Ownership* (TCO) . The figure 2.1 shows the data related to the

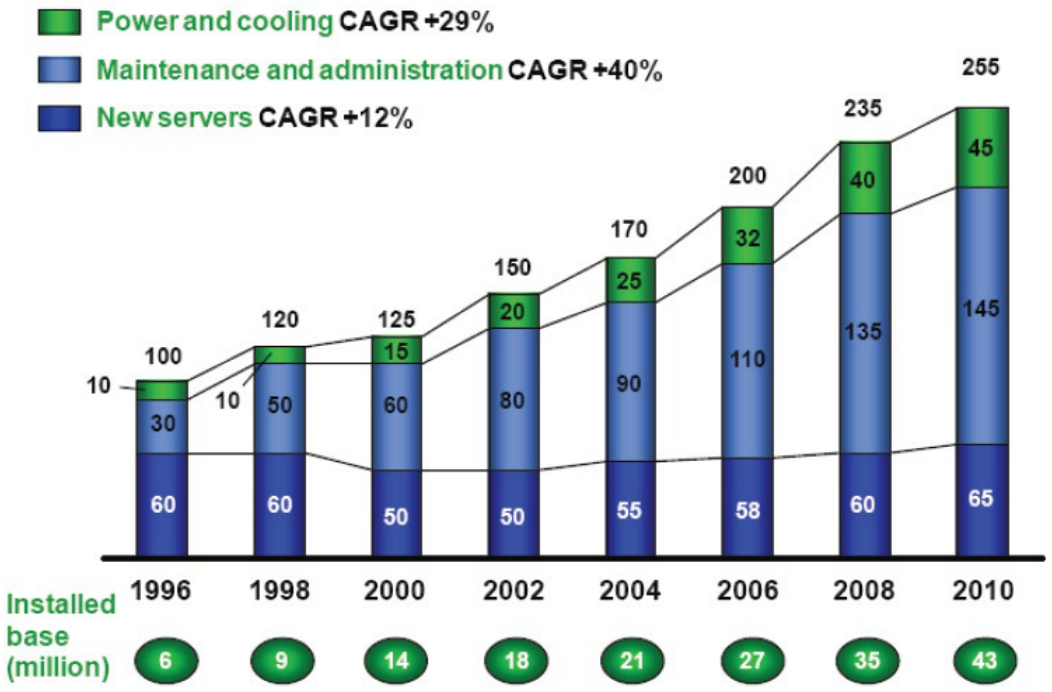

Figure 2.1: Compound annual growth rate of the costs of IT infrastructures. Source:IDC (2006)

servers world cost in the last years. While the hardware cost in the last 12 years is slightly grown, the cost of power and cooling is grown four times. Nowadays, power and cooling cost represent the 60% of the total spending for new IT in-

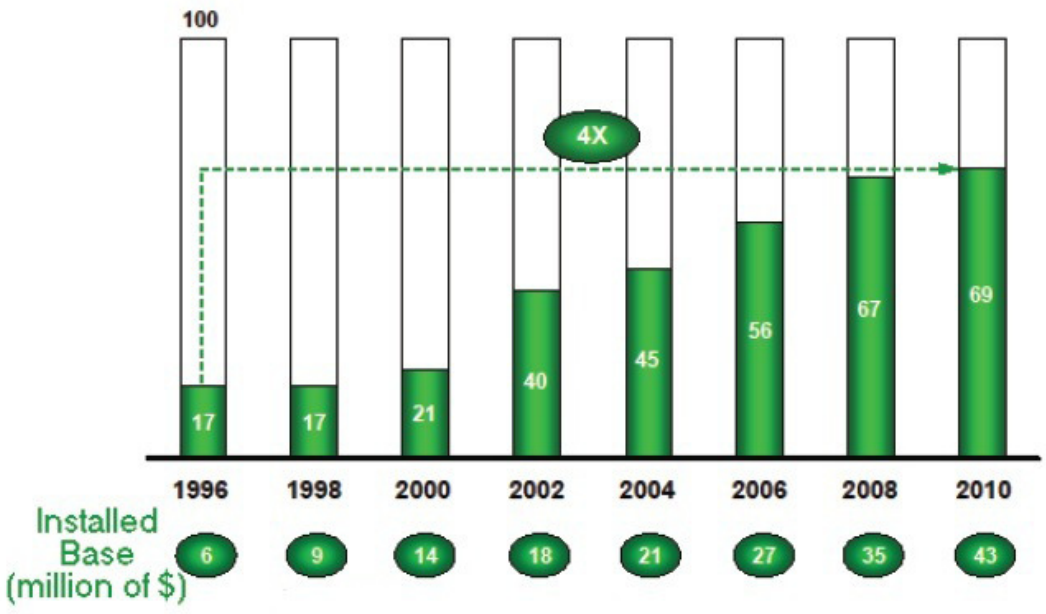

Figure 2.2: Ratio between energy and cooling cost vs new server cost. Source:IDC (2006)

frastructures, with a big impact on the *Total Cost of Ownership* (as is shown in the Figure 2.2). This proportion is expected to rise even more in the next years, also because of the continuous growth of energy unit cost.

Even if the cost of energy is continuously growing and has a significant impact on the energy consumption of a company, in most of the cases energy costs are not charged to the IT budget, which means that the costs of IT power consumption are not yet included in the TCO of IT infrastructures.

In addition, the higher energy consumption of IT equipment is becoming a limit to the data centers scalability of medium and big companies located in higher density living areas. The electric power need grows 8/10% each year and for the electric suppliers grows the risk to be unable to supply that amount of energy in a restricted urban area. To increase the capability of computation in the actual data centers is going to be necessary building new structrures in low-density living areas, with a further enviromental and cost impact.

2.1.3 How to make the Green IT a multi-level approach

Up to now the main focus of this introduction has been on Green IT and on the impact of IT infrastructures on power consumption. Now the focus will be on the reason why IT infrastructures consume so much energy. As a matter of fact it is the transmission of the information itself that consumes energy. One bit, which represent the minimum quantity of information, keeps its information by associating it to the state of a physical system (for instance bits can be kept in a persistent manner on a magnetic storage device such as magnetic tape or disc). In order to make the bit change its state, the state of the physical system associated to must be changed, and this needs energy.

Recent studies conducted at MIT (Massachusetts Institute of Technology) [17] found a lower bound for the power consumption needed to commute a bit from one state to another at a given speed. This bound is due to quantum physic laws and can only be reached when every bit is strictly associated to only one quantum electronic spin. This particular kind of bit commutation can only be reproduced with quantum computers, which are still under study. These computers exploit quantum properties of materials and they require 10^{-25}J in order to commute one bit at 1GHz. Traditional computers only reach 10^{-16}J. Anyway these energy values are still lower then the power consumption discussed before. This difference comes from the fact that a bit commutation is the lowest level of a computer system; over it there are a lot of different infrastructural layers which multiply, also for a factor of 30, the single energy required by one bit commutation.

Figure 2.3 [13] shows the energy consumption distribution for a typical data center (note that the distribution may vary due to different computational load and due to hardware characteristics of the system). From the figure it comes out that Green IT has to be faced from different infrastructural levels of the data center (CPU usage, load distribution on the different servers, cooling ...) since every single components contributes significantly to the overall power consumption. In order to improve energy efficiency of the entire system all the layers must be considered, especially the lower ones, since the power consumption of these layers is amplified by the upper layers. The guideline towards this kind of improvement must be software efficiency, considered in this case as the number of commutation executed by the processor. A non energy efficient software requires a higher number of processor operations, which means more commutation. In

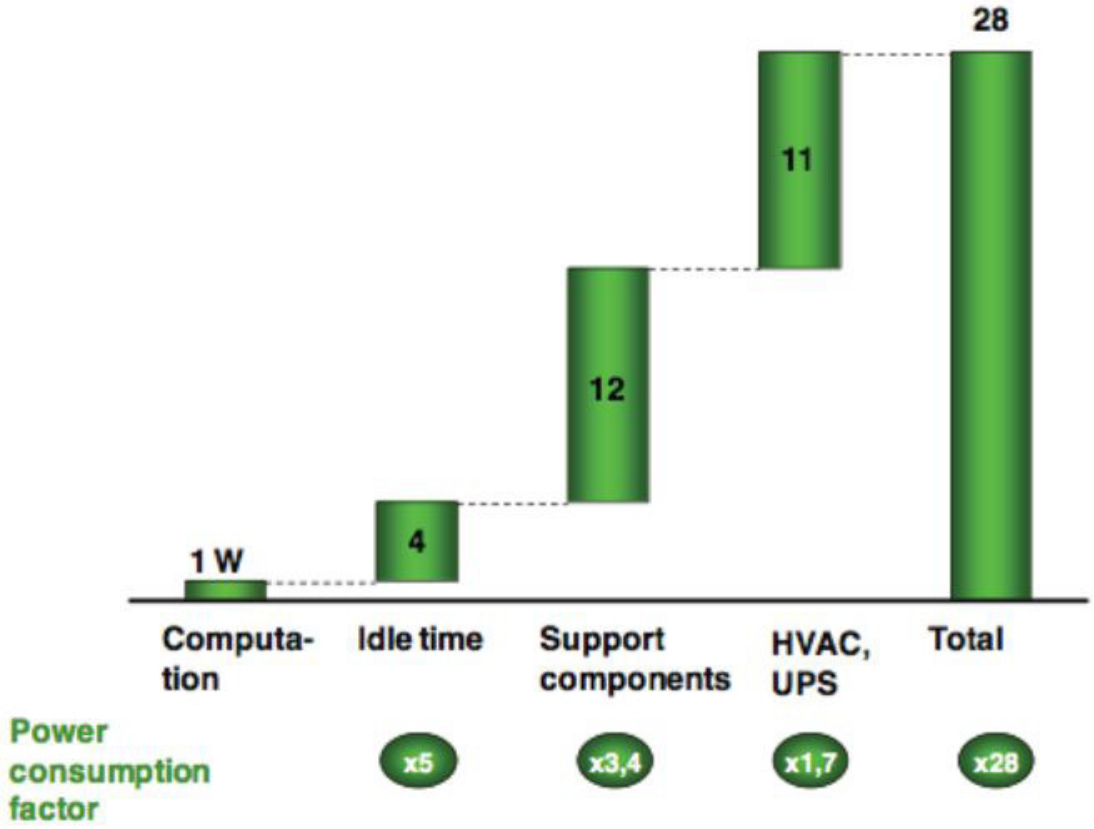

Figure 2.3: Power consumption breakdown in a data center. Source:IBM (2007)

addition it also requires more memory space and it needs to be cooled more.

The energy efficiency of an IT system has been estimated not to be higher the 50% of the theoretic maximal efficiency. As it has been sketched in Table 2.1, different layers in IT systems are characterized by different energy efficiency estimates. Analyzing each layer in detail it comes out the fact that these results are due to many small inefficiencies in widely different areas of an IT system.

At the infrastructural layer there are mainly two inefficiencies: UPSs and cooling systems, which usually cannot be configured and run at the maximum power possible. In order to make them more efficient it should be possible to dynamically adjust them in order to cool down the system only when and where it is needed, with the respect to the real workload of data center.
In addition the placement of shelves and air vents should be driven by thermodynamic studies of the environment; this would lead to gain in efficiency.

At the system layer efficiency improvements can be obtained by using virtu-

Level	Estimated energy efficiency
Infrastructure	50%
System	40%
Server	60%
Processor	0,001%
Software	20%
Network	10%
Database	60%
IT use	30%

Table 2.1: Estimated energy efficiency with respect to the maximal theoretic efficiency of IT systems divided in logical and infrastructural layers.

alization, which allows to balance workload between the different servers.

At the server layer it is possible to obtain high efficiency improvements. As a matter of fact a lot of power is dissipated from the peripheral components; by trying to optimize each of them it is possible to obtain a global gain. For instance, reducing energy consumption by making fans not continuously run but by making them run according to the cooling needs; this approach can make the fans power consumption reduce to 50 %. Other improvements can be obtained by dividing the cache in segments, where each of them must only be powered if it is really used, or by substituting hard disks with solid state memory.

One of the main things responsibile for power dissipation is the conversion from alternate power to continue power. This conversion is typically done in each server, because they have their own power supply. If the conversion is done only by one bigger power supply, energy efficiency would be much higher and the heat dispersion much lower.

Processor is the component for which the efficiency is the smallest one. As mentioned before, today's processor architectures are far from reaching the power consumption lower bound imposed by the quantum physics. Beyond their architectural limits, today's processors do not exploit their potential enough and they are not used in an energy efficient way. It has been proved that lowering the clock frequency and use multi-core processors can reduce power consumption by 50% [23].

The operations executed by the processor are driven by the software, that it implies different layers: from the operating system, through the middleware layer

to the user applications layer. Usually software engineers and programmers do not take into consideration software efficiency enough and sometimes software power consumption is not even considered in the trade-off between costs, quality and performances. Recent studies had taken in consideration how are the internal software parts that have also an impact with its energy efficiency. In the next section will be presented the green software concept.

At network layer, efficiency can be improved thinking about green routing algorithms or optimizing hardware network infrastructures.

Also the database efficiency can have a consistent impact on power efficiency. Data quality is an important aspect since poor quality data can lead to more transactions and more operations, with a consequent power loss. Database data structures, engines and parameters must be tuned in a way to minimize the required operations, therefore to minimize the overall power consumption.

Last but not least there is the way IT is used, which can be the cause of consistent power losses. Recent studies [12] have proved that the correct use of power management technology already available also on personal computers, like screen savers or turn off the system when it is not used, can lower energy consumption by 60%. There are little things that can be done easily but that can save a lot of energy, like allowing the standby on the systems.

There is the need of a more radical change, able to modify deeply companies' cultures and the way of thinking towards IT systems. Both the system administrators and the end users need to change their habits and understand how to behave more "green". Every person involved with the IT system must be correctly instructed and must understand the problems related to energy efficiency. The investment that needs to be done in order to acquire a "green" computer is 15- 20$ more then a regular one, but the power consumption reduction is around 30W given the actual power costs, which makes the investment repaid in less then 2 years, without even considering the reduction of cooling costs [6].

2.2 Green Software

The Green IT is now in the heart of attention both at accademic and at industrial level [3]. Speaking about Green IT, it is possible to point out different fields of actions, like:

- Workspace
- Data Center
- Green Hardware
- Green Software

The section before talked about the first three points. From now on the focus will be on the software, which is the first responsible of the energy consumption caused by the hardware. The software "drives" the hardware and it selects the basic instructions that have to be executed.
The "green software" studies the procedures according to the software has an influence on the energy needs of IT and how to optimize them.

To understand the area of green software can be useful an example with the motor world. If you want to use less fuel to go from Milan to Turin the first thing that you have to do is choosing a car that covers more km with 1 litre of fuel. In the IT world this is the same as using an efficient hardware, that can do a lot of basic operation with 1 Wh of energy. But there are many others solutions that can be used to use less fuel. First of all you can try to travel full load. For example, if you have to do the travel with other 8 people, you can use a minibus instead two cars. This is the equivalent to balance the workloads and using the virtualization, all things that are in the area of green data center. Then you can travel at the speed that minimize the fuel consumption, that usually is different to the maximum speed of the car, otherwise you can choose the best path that minimize the total amount of km. This two latest types of action corrispond in the IT world to the green software. One application can be evaluated in base of its energy efficiency, and not only in base with the other classical parameters, like the response time. In addition, a "well" written application can satisfy the functional requirements with the less number of possible basic operations and, as a consequence, with less energy.

While different studies were done on the hardware, on embedded systems [14] and on data centers, the energy efficiency theme on software is new and unexplored. The software development cycle and the related metholodogies don't care about energy efficiency as a goal. Even the software engineering literature doesn't give the metrics on how to measure the energy efficiency of software.

Most of the IT managers think that the software has a limited impact on the energy consumptions, in particular in the classical transactional systems, like the bank softwares or the Enterprise Resource Planning (ERP) [11]. It is common knowledge that the operating systems have an influence on performances and on energy consumption [28], but not the applications.

The green software possible saving is high. Therefore, in most of the data centers the percentage of incoming energy, used for the computation, is low because of the inefficiencies of the different infrastructure levels: for each Watt of energy used for computation are necessary 5 or more Watts. So the savings at the software level can be amplified by the infrastructural levels above. If are executed less basic operations, less heat is produced, so the energy needed for cooling is lower.

2.2.1 The studies in Green Software

There are three main topics to consider speaking about application optimization.

First of all, it is necessary measuring the software energy efficiency, using appropriate metrics. It is important to underline that the concept of efficiency is different from the concept of consumption, because it is the expression of the consumption of energy needed to execute the work done. A set of metrics for the energy efficiency is useful both for selecting the software that has to be choosen and for the inhouse evaluation of the product and the development team. They can be a good tool for comparing two different system in order to do benchmarking.

If the goal is the energy efficiency improvement, the possible ways that can be chosen can be grouped in two big categories. The first one is the code optimization: a code written better has an effect on the application energy performaces. This tecniques can be traslated in methodologies, guide lines, development tools and also in guide lines for the business levels, to show the developers abilities and their training path. Unfortunately, they are appliable only on inhouse software.

Improving the efficiency optimizing the code is not always feasible or economical. It can be a good idea rewriting and optimizing the code of the routine that are the most executed in the system, but it is not possible thinking to analyze and optimize the entire code of a bank information system or the effort necessary to do this operation can be more expensive than the savings obtained. For this

reason it is suitable developing level system optimization tools, applicable without working at level code. At this level there are a lot of tools and actions that can be developed, like a wise selection of the stack, considering the interaction between the operating system and the application on the total consumption.

2.2.2 The meaning of energy efficiency in software

At different levels can be found some metrics useful for the energy efficiency measure. For example, the Power Usage Effectiveness (PUE) [3] is commonly used to evaluate the efficiency at infrastructural level, including cooling systems and UPS. To evaluate the load balacing and the virtualization level can be used metrics like the mean percentage of the processor usage. In processor and server levels can be used metrics like W/tpm (Watt for transaction per minute), FLOP/Wh (Floating Point Operations for Wh) or MIPS/W (Millions of Instruction Per Second for W) in the mainframes.

However there are not generally and reliable metrics to measure the energy efficiency of the software. The difficulty in defining these metrics is how to measure the "work" done by an application related to one energy unit. In the motor world example, the efficiency of a car can be measured as distance in kilometers with one litre of gas, in the green software area it is easy to know how many gas litres have been consumed (the energy), but it is difficult to measure the distance.

So the problem is defining standard transaction and workloads and then execute them with an application in order to measure the efficiency. Some institutes, like TPC [1], have define standard workloads that can be used, but in most of the cases this benchmarks analyze the entire system, and not the final application. For others applications is easy to define transactions (like the DBMS[2]), but for others there are different types of transactions, and so there are different types of efficiency metrics. The different transactions are related to different modules and code portions of the application, and it is reasonable using different metrics. The goal is to reach a single metric for the entire application, maybe a mean of the different metrics with their use frequency. It is obvious that the definition and the computation of this metrics is more difficult than for the others infrastructural levels.

[1] Transaction Processing Performance Council
[2] DataBase Management Systems

An alternative solution may be identifying others metrics based on the code, in some ways correlated with the energy efficiency, and using like proxys. These metrics could be a convenient solution because they could be easy measurable using appropriate software tools that analyze the code without executing it. However, the studies haven't now reach a good result on this type of metrics.

2.2.3 Writing energy efficient software

Applications that have the same functional requirements, but a different structure, may have different energy consumptions.

Writing a good code has an heavy influence on the efficiency as can be noticed from an analysis of the function XIRR, used by some Italian banks. XIRR computes the investment return rate and it requires the computation of the polynomial zeros. There are different algorithms to do this operation, with a different impact on the energy consumption (Fig. 2.4). Optimizing the implementation

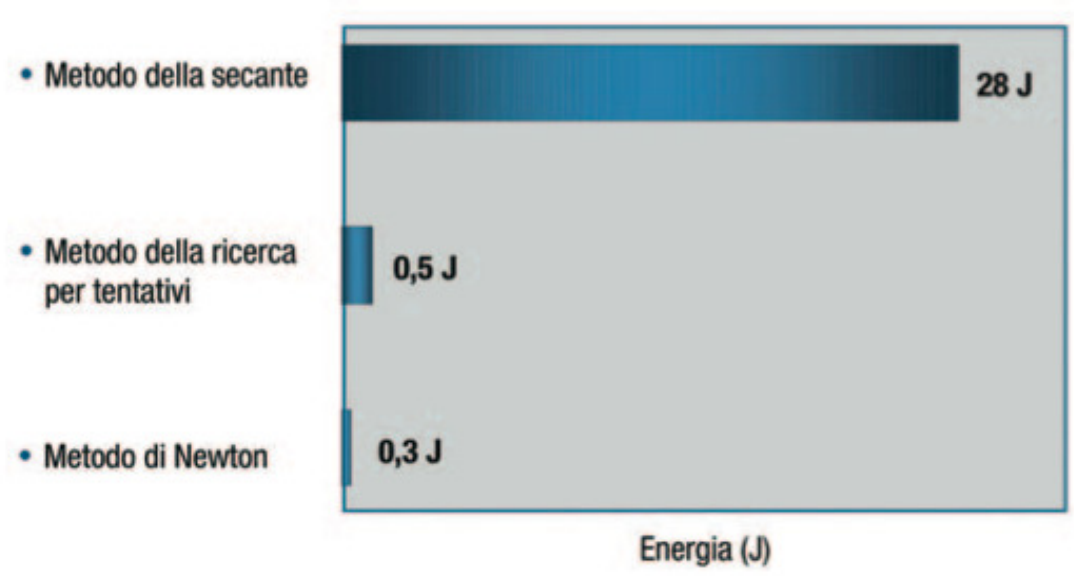

Figure 2.4: XIRR energy consumption

of the function using the most efficient algorithm can reduce three times the energy consumption. This reduction requires developers with deep knowing of the domain and the algorithms. Using this type of developers increases the development costs and can also increase the maintenance costs due to the difficult

interpretation. A good choice is optimize the 20% of the code responsible of the 80% of the consumption using specialized developers.

For big pre-existent code bases, the maintenance cost may be a critical factor. In this case there are different types of optimization methodologies, maybe semi-automatic, appliable at system level. The idea is setting the optimization parameters instead of modifying manually the code.

2.2.4 Consume optimization at system level: Memoization

Analyizing the typical power consumption profiles of the different hardware components of a server (Fig. 2.5), namely the hard disk, the motherboard and the CPU, it is possibile to note that:

- The processor is responsible for the bulk part of power absorbtion (approximately 60% when fully loaded)
- The energy consumption of the processor is a function of the load, while hard disk and memory have an energy consumption that is almost independent of load.

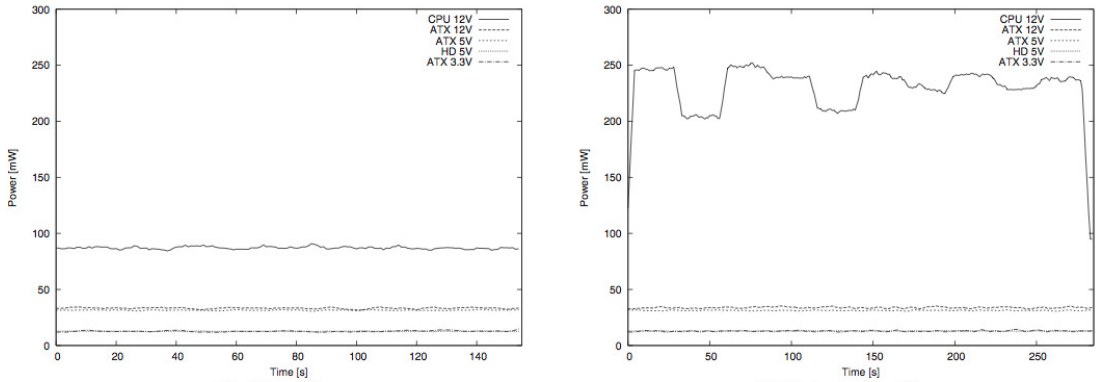

Figure 2.5: Power consumption profiles for server hardware components. HD lines indicate power absorbed by hard disk. CPU the power absorbed by the processor, and ATX power absorbed by motherboard and other components. (a) Idle profile. (b) Under load profile.

In fact, all the other components, except for the CPU, consume almost the same amount of power under load, Fig. 2.5(b), and in idle, Fig. 2.5(a). This is partially

explained by the fact that most of the power of traditional HD drives is used for spinning, and not for reading and writing operations. Similarly, dynamic RAM banks are periodically refreshed irrespective of reading and writing operations.

These observations suggest that computational approaches that make a more intense use of the storage in place of the CPU may have a beneficial impact on energy efficiency. For example, reading a precomputed value inside memory is faster and more energy efficient than computing it. This approach, known as *memoization* [27, 20], has been already applied (mostly in functional languages) to optimize the response time of software applications such as parsing, scientific applications and dynamic programming. Such an approach is usually feasible as the typical parameter range is wide, but finite.

The studies done in this thesis refers to use the memoization for the software energy efficiency, basing the approach on automatic identification of functions that are convinient to tabulate, which are a subset of the pure functions of an application. In fact, memoizing a function leads to energy savings only if the function is invoked several times with the same parameters, so that the results can be read from the memory rather then re-computed. This depends on the scope of the function and on the number of parameters, as the hit-rate probability decreases geometrically with the number of parameters.

Every time that one of these functions is called, a software module called energy manager takes control. This module knows whether the output of the function has been computed before or not. If the result is new, it is stored in the memory. If the required result has already been computed, the module reads the value stored in memory instead of computing it.

As hinted above, memoization techniques involve energy tradeoffs. First, memory is limited, and additional memory banks involve additional fixed power abortion. Accordingly, the total number of values that can be stored is limited. If more than one function needs to be optimized, available memory should be allocated to different functions so that the overall energy saving is maximized. Second, reading instead of computing involves an overhead due to the execution of the energy manager module. These trade-offs are influenced by the range and distribution of the input parameters of the functions to be tabulated. If the values of these parameters have a Gaussian or Pareto distribution, i.e. the functions are often called with the same input parameters, the approach can be effective. On

the other hand, if the range of the input parameters is too wide the approach could be ineffective.

2.3 Pure function memoization for the energy efficiency: GPF Framework

In order to evaluate these trade-offs and to efficiently choose which functions to memoize according to the available resources, a statistical previsional model has been developed [4]. This model is based on the assumption that power consumption is strictly related with execution time.

2.3.1 Proposed approach and software architecture

In this section is presented the proposed approach to exploit memoization of pure functions for power efficiency. Memoization is a programming technique to reuse computed values across a program by storing them in memory. Stored values must be indexed by the function and the input parameters that generate them. The focus is on Java programs. In general, Java is not commonly used for computation intensive applications. However, this is often the case in the financial domain. Moreover, the outcome of interviews to the executives at large European banks is that while currently only 7-8% of the codebase of their instructions is written in Java (the larger part is still in COBOL), Java tends to be used for most of the newly developed applications.

Pure function definition

Pure Functions are those functions (or, equivalently, methods or procedures, depending on the language) that have no side effects (i.e. they have no effect besides producing a return value) and are deterministic (i.e. the return value is determined uniquely by the values of input parameters). This definitions is often referred to as *strong purity* [30].

From the strong purity definition, several weaker definitions can be derived that are useful in applications - a popular one in the context of the Java language is to identify as pure functions those methods that do not modify any pre-exixting object, but may create new objects. For the thesis purposes, however, such a

definition is not appropriate: the newly created object must be different at each invocation, thus preventing a successful memoization.
Definition. A Java method m is a pure function if the following conditions hold:

- Its signature does not include object parameters, except:
  - *this* parameter;
  - array parameters, if the base type is primitive;
- Its return type is a primitive type;
- All methods invoked within m are pure functions;
- For all instructions composing the body of m, the following conditions hold:
  - The instruction does not read or modify static variables;
  - The instruction does not read or modify variables that are not declared within the function;
  - The instruction does not modify array values;

It is worth noting that, contrary to the most common definitions of pure function, this definition does not prevent the generation or catching of exceptions, since these events interrupt the normal flow of execution, but either do not affect the external results or prevent the generation of a return value (thus ensuring that no value is read from the memoization cache if the corresponding input parameter tuple generates an exception). Moreover, it is possible to allocate objects within a pure function, if and only if such objects are not alive at the end of the function.

An example of purity analysis

For instance, consider the source code of the XIRR method reported in the figure 2.6. The XIRR method computes the annualized internal rate of return of a cash flow at arbitrary points in time. The analysis is conducted on the bytecode, which is not shown for the sake of brevity. Initially the analysis assumes the target method to be pure.

The XIRR method takes as input two arrays, respectively of `double` and `int`, representing the cash flow in terms of values and dates, and a guess of the solution used to initialize the algorithm (a `double`). At the bytecode level, the

```
public static double Xirr(double[] value, int
    [] days, double guess){
  int maxConvergenza = 50;
  double adrd = 0.05, xirr = guess, sum;
  char sign = '+';
  for (int i = 1; i <= maxConvergenza; i++) {
    sum = 0;
    for (int j=0; j<value.length; j++) {
      sum += value[j] / Math.pow(1 + xirr,
          days[j] / 365.0);
    }
    adrd = signAdrd();
    xirr = xirr + adrd;
    if(Math.abs(sum) < 0.0000001) { break; }
  }
  return xirr;
}
```

Figure 2.6: XIRR method source code

method signature is `([D[ID)D`, where `[D` represents an array of double. Since the array is composed of elements of primitive type, it does not cause impurity according to the definition of pure function, and the same goes for the other two parameters. Each element of the arrays will be treated as an individual parameter for memoization purposes.

Line 2-4 declare variables of primitive types, and therefore do not change purity. Then, the analysis proceeds on to two `for` loops. It checks the data accesses, both in read and write. The analyzed method only writes to local variables of primitive type, and acccesses both array parameters only for reading. By the definition of pure functions, the `XIRR` method is therefore pure, assuming that `signAdrd()` is a pure function (which is the case).

Software architecture of GPF Framework

The memoization has been exploited in two steps: a compile-time analysis and instrumentation step, and the run-time automatic memoization step [20]. As shown in figure 2.7, in the first step Java bytecode methods are statically analyzed to detect whether they are pure functions or not. Code is analyzed by means of BCEL [8], a Java library for bytecode inspection, in order to identify bytecode instructions classified as impure, i.e. which make the method impure. Impure instructions have been defined according to the criteria proposed above. Methods identified as pure functions are classified according to heuristic criteria to remove small methods for which memoization would not provide a benefit, as the

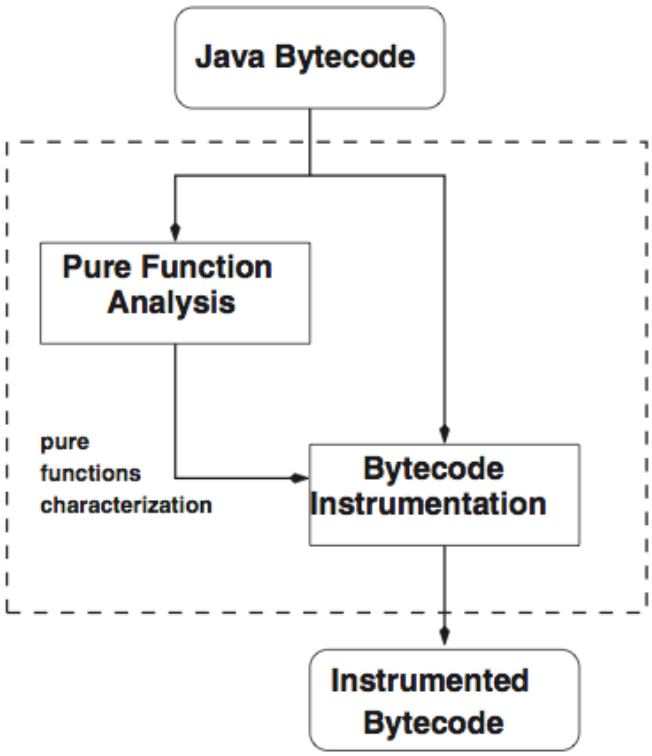

Figure 2.7: Bytecode instrumentation tool architecture: Java Bytecode is first analyzed to collect information about pure functions; candidate pure functions are then instrumented to support memoization

overhead of the energy memory module would be larger than the savings. These criteria are mainly based on a threshold size of the method, which is empirically tuned. The threshold is set based on the number of instructions contained in the function weight by their complexity. In fact, if the execution time of the function is lower than time to access memory, according to the performance model that will be discussed later, that approach will be always inefficient even if hit-rate is 100%. Obviously, this does not take into account the fact that the execution is dynamic and that execution time could be deeply influenced by cycles. However, this threshold permits to preliminary eliminate from the analysis functions that are evidently too simple, such as additions and void functions. The remaining methods are candidates for memoization.

Then, the candidate methods are instrumented, wrapping them in the code

needed to perform the second step.

At runtime, when a candidate method is invoked, the wrapper code is executed first. Each candidate method has an associated data structure (hash table) that contains pairs of parameter tuples and return values. If the current actual parameter tuple produces a match in the hash table, then the corresponding value is returned, skipping the rest of the wrapper code as well as the original method body. Otherwise, the value must be computed using the original method, which is immediately invoked. Before returning the result, however, a trade-off function is invoked to decide whether to cache the computed value or not - this may involve, if there is no available memory, a decision to replace another hash table entry.

Memory is allocated to maximize the overall effectiveness of the approach. The effectiveness of the memoization of a given function depends on the statistic distribution of the input parameters, and on the comparative saving that is obtained when accessing the memory rather than executing the function. To estimate the effectiveness of the memoization, a performance model has been developed.

2.3.2 Previsional Model

The performance model estimates the effectiveness of the approach based on execution time and is used during the function characterization phase.

Performance model

The effectiveness of the approach depends on:

- The energy required by the function for a computation compared with the energy required to read a memorized value.
- The hit rate of the stored values.

The hit rate depends on the variance of the input parameters of the function and on the size of the memory available for the memoization. The developed model allows to estimate the effectiveness of the approach taking the execution time as a proxy of energy consumption. The execution time has been taken into consideration because it is easier to assess the time performance of a function by

means of profiler tools rather than to evaluate its energy consumption, and energy consumption is directly related to execytion times for computation intensive applications. The memoization effectiveness is defined as:

$$\eta = \frac{T}{T_e} \tag{2.1}$$

where T is the average time required to satisfy a function call through the approach and T_e is the time required to execute the function and compute the result. If α is the hit rate then:

$$T = \alpha T_{hit} + (1 - \alpha) T_{miss} \tag{2.2}$$

$$T_{hit} = T_m \tag{2.3}$$

$$T_{miss} = T_m + T_e + \beta T_t \tag{2.4}$$

where T_m is the time to read a memoized value that is stored in the memory, and T_t is the time required to execute the trade-off module that decides wheter or not to store the new value of the function. Initially, the trade-off module will allow to allocate the available memory to the different functions in order to maximize total energy efficiency as estimated by means of the performance model described in this section. After this initial tuning, the trade-off module is invoked with a given frequency (β) to retune the memory allocation. In fact, a continuous execution would deeply affect performances. Eq. 2.2 combined with 2.4 and 2.3 leads to:

$$\eta = \frac{T_m}{T_e} + (1 - \alpha) \left(1 + \beta \frac{T_t}{T_e} \right) \tag{2.5}$$

Eq. 2.5 shows that the maximum effectiveness of the memoization approach, obtainable when the trade-off module is not executed ($\beta = 0$) and when all the input values of the function are stored in memory ($\alpha = 1$), is given by the ratio of the time to access the memory divided by the time to execute the function. As T_m and T_e can be easily measured, this result can be used to identify a priori the set of functions that are worth being memoized.

The hit rate depends on the number of stored values and on the statistical

distribution of the input parameters with which the function is invoked. Supposing that the approach has been applied to the function $y = f(x)$, that is invoked with a Gaussian distribution of the input parameters x, with mean μ and standard deviation σ. If x is a continuous variable, a sampling unit τ will have to be determined. The sampling unit should be decided according to the precision needed by the final users. If S_m is the size of the total available memory and S_d is the size of memory necessary to store a pair (x, y) the number of values that can be memoized is:

$$N_d = \frac{S_m}{S_d} \tag{2.6}$$

The probability that the function f is invoked with a value of the input parameter x that is tabulated as:

$$\alpha = erf\left(\frac{N_d\tau/2}{\sigma\sqrt{2}}\right) \tag{2.7}$$

where erf is the error function defined as:

$$erf = \frac{2}{\pi}\int_0^x e^{-t^2}dt \tag{2.8}$$

Equation 2.5 combined with 2.6 and 2.7 allows to estimate the effectiveness of the approach for a given function with a specific variance of the input parameter σ according to the available memory S_m.

Memory allocation policy

Memory is allocated to memoized methods in order to maximize the overall effectiveness of the approach. This is measured as gain factor obtained for each function ($G_i = 1/\eta_i$) weighted by the frequence of occurrence of that specific function:

$$G_{tot} = \sum_i G_i * f_i \tag{2.9}$$

where f_i is the frequence of occurrence of function i. It has been developed a memory calibration algorithm that defines the portion of memory to be assigned to each method that maximizes G_{tot}. This algorithm favors the methods that have a higher hit rate (α), based on the statistical distribution of their input parameters. It allocates the available memory incrementally to each function by

blocks of $S_{d,i}$ bytes. $S_{d,i}$ values are specific for each function i, according to the size of the input parameters and results to be stored for each function. The algorithm incrementally explores the possible allocation options of a new block of memory and identifies the function that would lead to the highest effectiveness gain if got assigned of that memory block. This is estimated by $G_i / (S_{m,i} + S_{d,i})$, where $S_{m,i}$ is the memory already allocated to function i.

When the memoized application starts, the functions candidate for memoization are known as they had been identified by the pure function analysis module. Their frequences of occurrence, and the mean and standard deviation values of their input parameters are estimated based on previous domain knowledge. A first run of the algorithm defines the initial allocation of the available memory space among the different functions. As long as the application runs, memory is allocated to memoized functions as the invocations happen, until the established portion of memory for each function is consumed. The trade-off module is invoked with a given frequence (β) in order not to create a time and power overhead. When it is invoked, first it runs the memory calibration algorithm with the new values of frequence of occurrence, mean and standard deviation. This leads to a new allocation of the available memory space. Then it eliminates the low-frequency entries in order to re-equilibrate the space according to the new allocation.

Chapter 3

Model and application

This chapter provides a detailed description about the reasons that are the cause of the realization of the thesis project. In the section 3.1 will be described the function chosen for the project, with the problems and the solutions adopted for the trasformation in pure function. It also provides a detailed description of the parameters interested in the computation. In the section 3.2 will be described the data used for the computation of the function.

3.1 Analyzed function

The investigation is in the financial domain because banks are traditionally among the largest IT-consuming organizations. Most financial institutions are worried about IT energy consumption both for cost and scalability reason [16]. In the specific case of financial computation, energy savings have been demostrated in recent works [22].

3.1.1 Why this function

The function chosen for the project belongs to the set of financial functions used by a big Italian bank. From interviews to IT administrators it cames out that the function that did the computation of the financing payments had several problems and so it was a perfect candidate for the project.

The function has three main problems:

- Is MIPS[1] intensive
- Its execution time is too high
- It consume a lot of energy

Analyzing all the three reasons that have brought to select the function, it is easy to say that they are correlated.

As the function is written in COBOL, its execution is done using mainframes[2] and for this type of computers the metric used for measuring the cost of computing is MIPS. Also the contract with the supplier of this technology is based on the MIPS consumed. The supplier sell to organizations the amount of MIPS necessary for the computation.

The function analyzed is MIPS intensive. It requires a big amount of MIPS and this quantity is growing. As a consequence, the cost of the contract has grown too.

The increase of the consumed MIPS generates another increment on the execution time. The function has been developed to work when the bank is closed, and the mainframes are dedicated to compute only this function. The idea was that the computation ended before the opening time. This condition has been expired and now there is the problem that the mainframe works in the opening time too. This is a cause of delays or, in the worst case, stops of the terminals used in the filials. These stops or delays are a critical problem for the bank, because they create also inefficiencies to the clients.

The third problem, related to the first and to the second, is the high energy consumption. This is due to the mainframe computation of the function, that clearly has a consumption higher than in the idle phase of the mainframe. This is not the only cause of the energy consumption. All the other components used with the mainframe, like UPS and coolers, have energy needs and they added to the mainframe consumption. As the mainframe, also the components energy needs change in base of the level of the computation.

In addition, the increment of the execution time causes also an increment of

[1] Million of Instruction Per Second

[2] Mainframe computers are computers used primarily by corporate and governmental organizations for critical applications, bulk data processing, industry and consumer statistics, enterprise resource planning, and transaction processing[9]

the energy consumption because the mainframe and all the components have to work more.

3.1.2 Problems found and solutions used for the COBOL-Java pure functions translation

To analyze the function that computes the financing payments it has been necessary to translate that function from the COBOL language to the Java, because also the framework described in the previous chapter is written in Java.

The translation was done in two phases: the first was the translation from the COBOL to the Java language. The second was re-designing the function in order to have pure functions.

The first translation of the function has some problems due to the differences of the two languages. In COBOL the definition of the variables is totally different from the one used in the modern programming languages: after the type there is the definition of the size of the variable. If the variable is composed of characters this is not an important problem, because there is always the possibility to use the method `sizeof()`, but if the variable is referring to a number, it has to be setted with the correct number of elements (in particular the decimals), otherwise the computation results will be wrong. To accomplish this, the Java class `DecimalFormat` has been used. In particular the code in the listing 3.1 is the one used in the function.

Another stylistic solution adopted was the creation of objects with variables and relative getters/setters. This is useful for the code interpretation and for the data saving, but in the second step of the translation the solution has to be deleted because pure functions use only primitive types.

```
DecimalFormat format1 = new DecimalFormat("#.##");
String numberString = format1.format(number);
number = Double.parseDouble(numberString);
```

Listing 3.1: DecimalFormat code example: transforms the variable number in a double with only two elements in the decimal position

The second step of the translation has product a total re-design of the function. As the definition of pure function says, all methods invoked within the function

candidate to be pure have to be pure. Following this condition, the first step is trying to modify the internal functions.The way to make a function pure is, first of all, having only one value to return and, in particular, it has to be primitive. With this condition all the objects created and mentioned above have to be substituted with a set of variables that represent all the object's fields.

Speaking about the signature of the method, all the data used inside of it have to be in the parameters section of the signature, or they have to be declared inside of the function. There is no possibility to use variables declered outside the function and not in the signature as a parameter, or to use global variables.

Also the parameter types have to be modified to be primitive. The `Calendar` type cannot be used, and for all the dates the solution adopted was using three `int` variables, one for the day of month, one for the month and the last for the year value.

The System class is considered not pure. So all the `System.out.println()` instructions, used for the error warning, have to move in the external methods, in particular in the one that is unpure. The same action has to be taken for the `System.exit` instructions, used for exit from the fuction execution after an error.

Clearly, is impossible to make pure all the functions, because of the presence of the `System` instruction mentioned before. But it is not the only cause of the non purity of the entire function. The class `DecimalFormat` is not pure, and it is not possible deleting these instructions, because are useful to set the decimals in the numbers used for the computation. Where possible this instructions can be moved outside the method, but there are situations that this operation cannot be executed.

Another reason that cause the non-purity of the entire function is that the data are taken from an external CVS file[3]. For the condition "the instruction does not read or modify variables that are not declared within the function" it is not allowed to interact with an external file, even if it contains the data useful for the computation.

3.1.3 Computation interested parameters

The pure functions found with the framework (fig. 3.1) are mainly related to the computation of the financing payment value. There are different functions that

[3]Comma-Separated Values, a file that stores tabular data (numbers and text) in plain-text form.

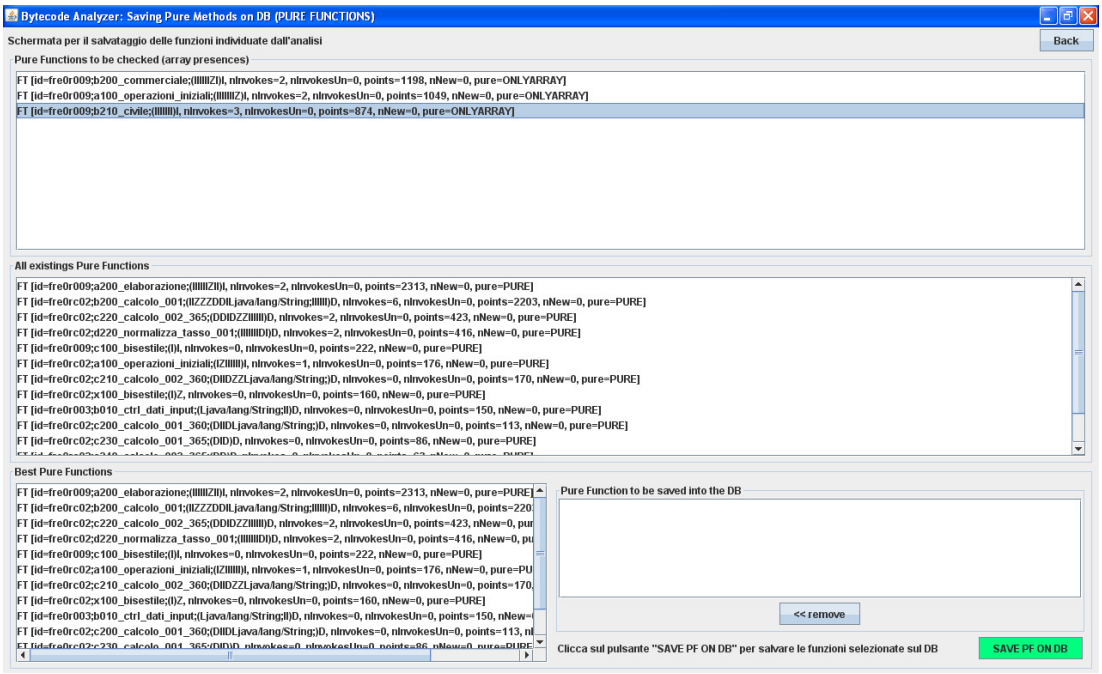

Figure 3.1: Framework analysis of the function

have the duty to compute the value, depending on the data stored in the CSV file. Analyzing these functions it has been noted that they have most of the input parameters in common, and it is possible to highlight them as more important in relation to the others in the data file.

The most used parameters in the function are:

- **Consistenza**: the value of the capital on which is calculated the value of the financing payment.
- **Tasso**: the value of the interest rate used for the computation of the financing payment.
- **Periodicitá**: number of months in the period between the begin and the end date.
- **N giorni**: number of days in the period between the begin and the end date.

This four variable are the most used in the function and they are the only that have an impact on the final value of the financing payment. In each of the

different type of computation of that value, the four parameters written above are presents.

It is important analyze them in such a way to create a new approach useful for the application of the memoization method and for the reduction of the energy consumption.

3.2 Data analysis

In the first section there is an accurate description of the function, but without data this function is useless.

In this section there is an analysis on the real data used by the big Italian bank. This analysis will be useful for the creation of the approach used in the project.

3.2.1 Data distribution

The total number of data is 1.200.000. In the project has been chosen a set of data that has the parameter "componente" equal to the value 3. This choice was done by the Italian bank and it selects the data concerning the post-sale interests. This selection has decreased the number of the data to 408.863.

Doing an analysis on the two parameters Consistenza and Tasso it is possible

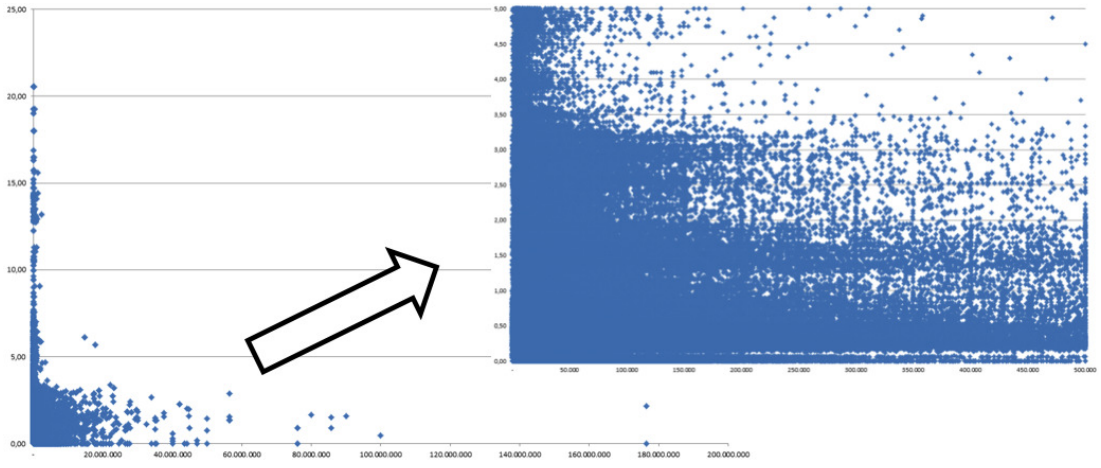

Figure 3.2: Data Distribution using Consistenza on the x-axis and Tasso on the y-axis

to note that there is a concentration of the data (fig. 3.2) in the lower part of

the graph, near the axis.

Looking to the data distribution, it is possible to think on the application of the memoization approach. Most of the data are concentrated in a specific area, and using memoization can be a possible improvement. Instead of computing all the data, taking from the memory the result of a precedent computation can decrease the execution time and so the costs related to it. In fact, reading from the memory is faster than compute the data. Clearly, to take a data from the memory, it has to been computed before.

In case of the data is not in the memory (so, it is never beeen computed), the idea is to compute it and save in the memory, in case of future use.

The idea to use memory instead computing data is good, especially for the savings. The problem is the fixed size of the memory, that cannot permit the total saving of the tuples `<input,output>` .

One possible idea to solve this problem is to memoize the data with an high frequency of occurrences, and then compute the remaining.

To discover what are the data with an high occurrence frequency it is necessary analyze the entire set of inputs. In the table 3.1 is shown the interval of the data taken as input by the function. From these values is possible to create a set that can cover more than an half of the data. This interval is the one shown in the table, in the column memoization data interval.

Data analysis				
	Real data interval		Memoization data interval	
	min	max	min	max
Consistenza	0,26	176510709,30	870,00	50000,00
Tasso	0,00	19,25	0,20	1,00

Table 3.1: Data intervals

Total Values	408863
Values in the interval	220571
Percentage	53,95%

Table 3.2: Interval analysis

The table 3.2 summarize the analysis of the input data. The set chosen can cover the 53,95% of the data. This is useful for the memoization, because this

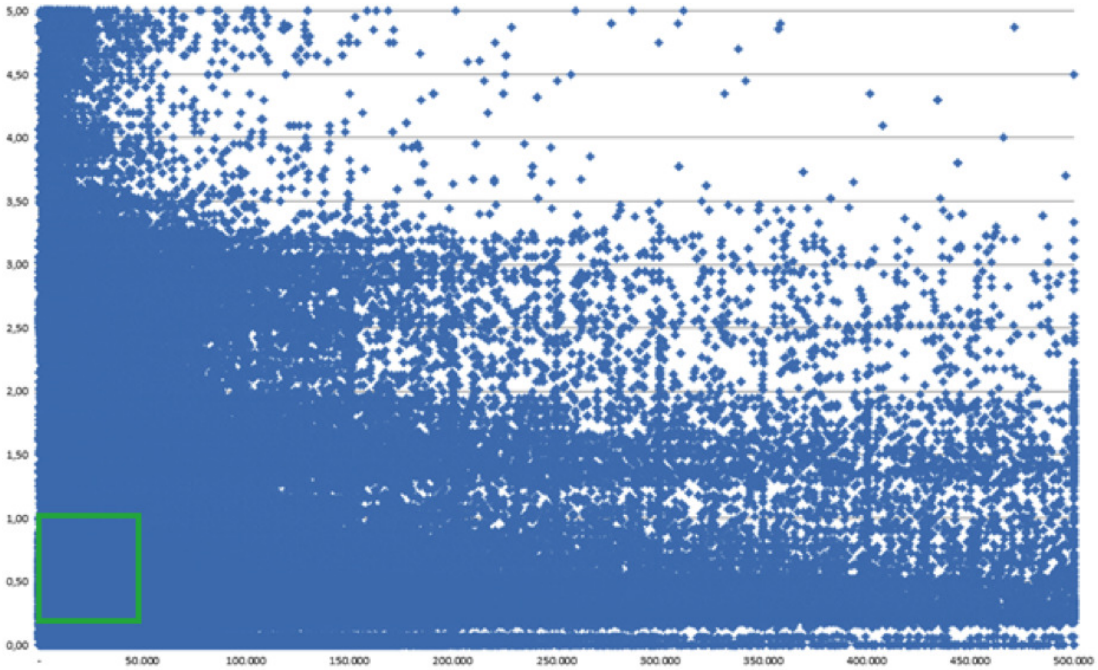

Figure 3.3: Data Distribution with the memoization interval in green

result permits to say that using memoization with this small interval permits to cover more than the half of the data in input. The figure 3.3 shows the distribution of the data and the green rettangle is the interval choose for the memoization.

The final project idea is to memoize the tuples `<input,output>` in the interval, and then to compute the others data input that are not in the memory. If there is some free space in the memory, the value computed can be stored, otherwise the value will be computed another time if it will be necessary.

3.2.2 Memory division

The idea to divide in two parts the solution for the computation has been used to divide also the memory.

The first part of the memory is used for the memoization of the interval. It is the biggest division and it is used for memoize the data that are in the set defined above. The values are memoized in the way to cover completely the interval. If the input data is equal to one saved in the memory, it is not saved, and its result is taken from the tuple memoized.

The second part of the memory is used for the data that are outside the in-

terval. In this case, the value, after the computation, is saved in the memory if there is free-space. Otherwise is deleted.

To improve the memory-usage and to enlarge the memoization interval, it is possible to introduce the concept of SLA (Service Level Agreement).

All the data saved in the memory are precise and they refers to only one tuple `<input,output>`. Instead using this data is possible to use and save an interval, that includes a set of the data. In particular, the idea is to take the output and create an interval. The boundary values are output + SLA and output - SLA. The value of the SLA is defined by the user and it is the highest error that the user wants. Having the output interval, with an inverse operation it is possible to have the input interval, and then save the tuple `<input_interval,output_interval>`.

With this approach, if the input value is in one of the tuple in the memory, the output value saved is used as output value of the computation, even if the input value is not the same data saved in the memory. Adopting this approach can increase the number of possible values that can be saved in the memory and increase also the memoization interval.

3.2.3 Data memory occupation

In the previous sections is done a global analysis of the data used as input in the function.

These data have a defined dimension, based on the variable type declared in the function. In the table 3.3 is shown the memory occupation of the single variables.

Using the structure chosen for the memoization (described in the next chapter), the total amount of memory needed for memoize all the data is about 4 billion of gigabytes. This is a huge quantity of memory, therefore the idea of using an interval of the data mostly used can be ideal because reduces the quantity of memory needed.

Variable	Type	Bit	Variable	Type	Bit
rewc_rc	int	32	no_intero_fn_mese	boolean	1
data_inizio	Calendar	432	si_intero_no_fnmese	boolean	1
data_fine	Calendar	432	cod_divisa	String	48
consistenza	Double	64	tipo_calcolo	int	32
tasso	Double	64	formula	int	32
periodicita	int	32	importo_calc	double	64
intero	boolean	1	tasso_calc	double	64
si_intero	boolean	1	nr_giorni	int	32
no_intero	boolean	1	divisore	int	32
intero	boolean	1	gg_calendario	int	32
Totale bit		1398	**Totale Byte**		175

Table 3.3: Size of variables

Chapter 4

Idea: approximated memoization

In this chapter will be described the model created for the thesis project.

In the first section will be provide a detailed description about the structure used in the memory for the storage of the data.

In the second section will be presented the approximed memoization, with the description of the previsional model created.

In the third section will be analyzed the best method for curve fitting, used for the approximed memoization.

4.1 Data structure: quadtree

As said in the previous chapter, the storage of the data for the memoization approach is done using a particular structure. After some studies, the structure chosen is the *quadtree*.

A quadtree is a tree data structure in which each internal node has exactly four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular, or may have arbitrary shapes. This data structure was named a quadtree by Raphael Finkel and J.L. Bentley in 1974. A similar partitioning is also known as a Q-tree. All forms of Quadtrees share some common features:

- They decompose space into adaptable cells.
- Each cell (or bucket) has a maximum capacity. When maximum capacity

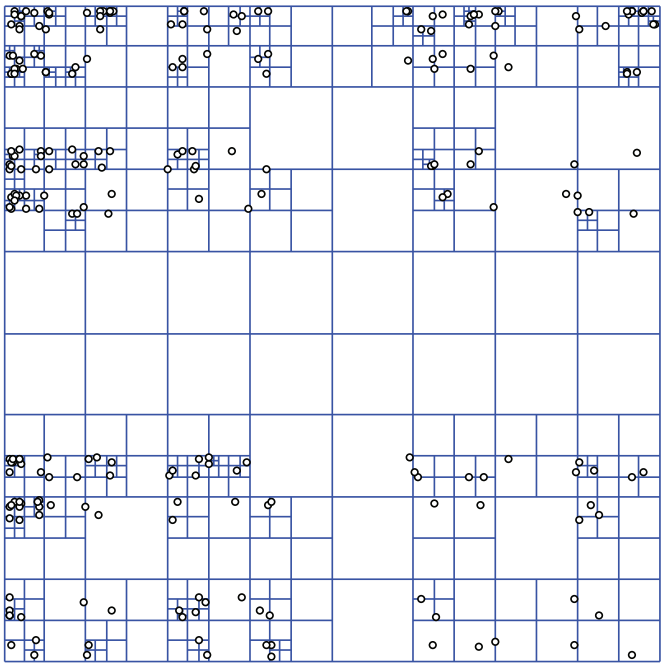

Figure 4.1: A region quadtree with point data

is reached, the bucket splits.

- The tree directory follows the spatial decomposition of the Quadtree.

Quadtrees may be classified according to the type of data they represent, including areas, points, lines and curves. Quadtrees may also be classified by whether the shape of the tree is independent of the order data is processed. Some common types of quadtrees are:

- **The region quadtree**: represents a partition of space in two dimensions by decomposing the region into four equal quadrants, subquadrants, and so on with each leaf node containing data corresponding to a specific subregion (fig. 4.1). Each node in the tree either has exactly four children, or has no children (a leaf node). The region quadtree is a type of trie[1].
  A region quadtree with a depth of n may be used to represent an image consisting of $2^n * 2^n$ pixels, where each pixel value is 0 or 1. The root node represents the entire image region. If the pixels in any region are

[1]trie: ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings

not entirely 0s or 1s, it is subdivided. In this application, each leaf node represents a block of pixels that are all 0s or all 1s.
A region quadtree may also be used as a variable resolution representation of a data field. For example, the temperatures in an area may be stored as a quadtree, with each leaf node storing the average temperature over the subregion it represents.
If a region quadtree is used to represent a set of point data (such as the latitude and longitude of a set of cities), regions are subdivided until each leaf contains at most a single point.

- **Point quadtree**: is an adaptation of a binary tree used to represent two dimensional point data. It shares the features of all quadtrees but is a true tree as the center of a subdivision is always on a point. The tree shape depends on the order data is processed. It is often very efficient in comparing two dimensional ordered data points, usually operating in O(log n) time.

- **Node structure for a point quadtree**: is similar to a node of a binary tree, with the major difference being that it has four pointers (one for each quadrant) instead of two ("left" and "right") as in an ordinary binary tree. Also a key is usually decomposed into two parts, referring to x and y coordinates.

- **Edge quadtree**: are specifically used to store lines rather than points. Curves are approximated by subdividing cells to a very fine resolution. This can result in extremely unbalanced trees which may defeat the purpose of indexing.

The reasons that have brought to choose the quadtree are the easy way to store and to read the values from this structure.

Having quadrants and regions with equal dimensions directs the data to the corrisponding subregion and then in the leaf, where the data will be stored. This way to store the data is useful in case of a not equal distribution of the data. In this case, if a data is the only one in the region, this will become a leaf. On the other hand, if a data belongs to an high concentrated set, the tree structure will be deep.

The second advantage of using quadtrees is the easy way to search data. On each level of the quadtree the search is directed to the subregion in which the data belongs to, with a good decrease of the time to read a value in the memory.

4.2 Modified previsional model

The model estimates the effectiveness of the approach based on execution time and is used during the function characterization phase.

The effectiveness of the approach depends on:

- The energy required by the function for a computation compared with the energy required to read a memorized value
- The hit rate of the stored values

The hit rate depends on the variance of the input parameters of the function and on the size of the memory available for the memoization.

In the chapter 2 the memoization effectivenes (η) was defined as:

$$\eta = \frac{T_m}{T_e} + (1 - \alpha)\left(1 + \beta\frac{T_t}{T_e}\right) \tag{4.1}$$

where T_m is the time to read a memoized value that is stored in the memory, T_e is the time required to execute the function and compute the result, α is the hit rate and β is the trade-off module frequency.

This equation shows that the maximum effectiveness of the memoization approach, obtainable when the trade-off module is not executed ($\beta = 0$) and when all the input values of the function are stored in the memory ($\alpha = 1$) is given by the ratio of the time to access the memory divided by the time to execute the function.

The hit rate is defined as the probability that the function f is invoked with a value of the input parameter x that is tabulated as:

$$\alpha = erf\left(\frac{N_d\tau/2}{\sigma\sqrt{2}}\right) \tag{4.2}$$

where erf is the error function defined as:

$$erf = \frac{2}{\pi}\int_0^x e^{-t^2}dt \tag{4.3}$$

and N_d is the number of values that can be memoized, and is defined as:

$$N_d = \frac{S_m}{S_d} \tag{4.4}$$

Equation 4.1 combined with 4.4 and 4.2 allows to estimate the effectiveness of the approach for a given function with a specific variance of the input parameter σ according to the available memory S_m.

Analyzing the equation 4.2, it is possible to note that the hit rate α is influenced by some parameters:

- N_d: the number of the values that can be memoized
- τ: the sampling unit
- σ: the standard deviation of the Gaussian distribution of the parameter x

The number of values and the standard deviation are fixed values, so the parameter that influences the value of the hit rate is the sampling unit τ.

The sampling unit is related to the input parameter x, if it is a continuous variable. The sampling unit value is decided according to the precision needed by the final users.

With the term precision is defined the measure of how close replicated estimates are from each other. This is the same as asking how much error is there around a mean estimate. In the thesis project, the value of the precision is defined as SLA (Service Level Agreement).

A service-level agreement (SLA) is a part of a service contract where the level of service is formally defined. In the project, the SLA is defined as the precision needed by the final users on the value of the financing payment. For example, if the SLA is setted to the value 0,05 euro, the output value accepted by the user are included in the set with the boundaries equal to: $[realvalue - SLA; realvalue + SLA]$.
The mathematical definition of the SLA is:

$$SLA = f(\tau) \tag{4.5}$$

The service level agreement is a function of the sampling unit. If the sample is large, the output values are less precise. If the sample is small, the output values

have an high precision.

As a consequence, the value of the sampling unit τ can be defined as:

$$\tau = f^{-1}(SLA) \tag{4.6}$$

The problem is now how to define the function $f(x)$ that is unknown.

Of course, given some training data, it is always possible to build a function that fits exactly the data. But in the presence of limited memory usage, it is not possible to use all the given sampling units by considering the repetition of same samples which would lead to a poor performance. The general idea behind the design of a model is thus to look for a fitted function with memoization method. Typically, one would look, in a collection of possible models, for the one which fits well the data. But there is an important question: how many sampling units (τ) have to be used in order to have an accurate function?

The larger the sample size, the more precise is the estimate. Sample size will therefore depend largely on the reliability wanted to place in the estimate. If the request is a very precise estimate, a larger sample of inputs is needed than if a good approximation is wanted. Statistical methods requiring the collection of some pilot data, are available for calculating sample sizes necessary to achieve predetermined levels of precision.

To achieve precision, the idea is saturating memory with input/output data by executing the program during the tuning phase.

It is needed again to control the memory if the input values are same with a new input data. In this way, the output value will be the same. Thus, in order to have good accuracy in the function, the repeated data is not saved in the memory, letting the other sampling units take place in it.

4.3 Model mathematical analysis

There is now the need to determine which sampling units, out of all those available, should be used in the project.

To accomplish this task, the first step is define the function f and then how to obtain this function using some mathematical methods.

4.3.1 The definition of f

The data known are the input values associated with the corrisponding output values and that the function $y = f(x)$ is invoked with a Gaussian distribution of the input parameter x.

To collect the output values and try to approximate the function the first step is setting a tuning phase, where at each execution of $f(x)$ the input and the output values are saved in the memoy. The period of execution of the tuning phase is decided by the system administrator.

After the tuning phase the idea is to approximate the function to a third degree polynomial function, like the following:

$$\tilde{f} = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3 \tag{4.7}$$

To find the 4.7 there are some mathematical methods, called curve fitting, that will be described in the following section, with the selection of the best method that will be used in the thesis project.

4.3.2 Curve Fitting

Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, possibly subject to constraints. Curve fitting can involve either interpolation, where an exact fit to the data is required, or smoothing, in which a "smooth" function is constructed that approximately fits the data.

Fitted curves can be used as an aid for data visualization, to infer values of a function where no data are available, and to summarize the relationships among two or more variables.

Extrapolation refers to the use of a fitted curve beyond the range of the observed data, and is subject to a greater degree of uncertainty since it may reflect the method used to construct the curve as much as it reflects the observed data.

Curve fitting parameters

Before starting the list of the methods used for the curve fitting is important explain the parameters used to say that a fit-statistic is good.

Goodness of Fit-Statistic

To explain how much is good a fit-statistic are used 4 parameters, that are described in the following paragraphs.

Sum of Squares due to Error (SSE) This statistic measures the total deviation of the response values from the fit to the response values. It is also called the summed square of residuals and is usually labelled as SSE.

$$SSE = \sum_{i=0}^{n} w_i * (y_i - f_i)^2 \tag{4.8}$$

where y_i is the observed data value and f_i is the predicted value from the fit. w_i is the weighting applied to each data point, usually $w_i = 1$.

A value closer to 0 indicates that the model has a smaller random error component, and that the fit will be more useful for prediction.

R-Square R^2 This statistic measures how successful the fit is in explaining the variation of the data.

R-square is the square of the correlation between the response values and the predicted response values. It is also called the square of the multiple correlation coefficient and the coefficient of multiple determination. It is defined as:

$$R - square = 1 - \frac{\sum_{i=1}^{n} w_i * (y_i - f_i)^2}{\sum_{i=1}^{n} w_i * (y_i - y_{av})^2} = 1 - \frac{SSE}{SST} \tag{4.9}$$

Here f_i is the predicted value from the fit, y_{av} is the mean of the observed data and y_i is the observed data value. w_i is the weighting applied to each data point, usually $w_i = 1$. SSE is the sum of squares due to error and SST is the total sum of squares.

R-square can take on any value between 0 and 1, with a value closer to 1 indicating that a greater proportion of variance is accounted for by the model. For example, an R-square value of 0.8234 means that the fit explains 82.34% of the total variation in the data about the average.

If you increase the number of fitted coefficients in your model, R-square will increase although the fit may not improve in a practical sense. To avoid this situation,the degrees of freedom adjusted R-square statistic described below has

to be used.

Degrees of Freedom Adjusted R-Square This statistic uses the R-square statistic defined above, and adjusts it based on the residual degrees of freedom. The residual degrees of freedom is defined as the number of response values n minus the number of fitted coefficients m estimated from the response values.

$$v = n - m \tag{4.10}$$

v indicates the number of independent pieces of information involving the n data points that are required to calculate the sum of squares.

The adjusted R-square statistic is generally the best indicator of the fit quality when you compare two models that are nested - that is, a series of models each of which adds additional coefficients to the previous model.

$$adjustedR - square = 1 - \frac{SSE(n-1)}{SST(v)} \tag{4.11}$$

The adjusted R-square statistic can take on any value less than or equal to 1, with a value closer to 1 indicating a better fit. Negative values can occur when the model contains terms that do not help to predict the response.

Root Mean Squared Error (RMSE) This statistic is also known as the fit standard error and the standard error of the regression. It is an estimate of the standard deviation of the random component in the data, and is defined as

$$RMSE = s = \sqrt{MSE} \tag{4.12}$$

where MSE is the mean square error or the residual mean square.

$$MSE = \frac{SSE}{v} \tag{4.13}$$

Like SSE, an MSE value closer to 0 indicates a fit that is more useful for prediction.

4.3.3 Best curve fitting algorithm analysis and selection

In this section will be presented the different tests done to find the best fit-statistic that will be used in the project, describing the methods used to do the tests.

In the following paragraphs are presented three different algorithms in order to do the comparison for finding the best one in fitting data.

Gaussian Non Linear Least Squares The Gaussian model fits peaks, and is given by

$$y = \sum_{i=1}^{n} a_i e^{[-\left(\frac{x_i - b_i}{c_i}\right)^2]} \tag{4.14}$$

where a is the amplitude, b is the centroid (location), c is related to the peak width, n is the number of peaks to fit, and $1 \leq n \leq 8$. In the experiment n = 3.

Polynomial Linear Least Squares Polynomial models for curves are given by

$$y = \sum_{i=1}^{n+1} p_i x^{n+1-i} \tag{4.15}$$

where $n + 1$ is the order of the polynomial, n is the degree of the polynomial and $1 \leq n \leq 9$ (in the experiment $n = 3$). The order gives the number of coefficients to be fit, and the degree gives the highest power of the predictor variable. Polynomials are often used when a simple empirical model is required. The main advantages of polynomial fits include reasonable flexibility for data that is not too complicated, and they are linear, which means the fitting process is simple. The main disadvantage is that high-degree fits can become unstable.

Cubic Spline Cubic spline is a spline constructed of piecewise third-order polynomials which pass through a set of control points. The second derivative of each polynomial is commonly set to zero at the endpoints, since this provides a boundary condition that completes the system of equations. This produces a so-called "natural" cubic spline and leads to a simple tridiagonal system which can be solved easily to give the coefficients of the polynomials. However, this choice is not the only one possible, and other boundary conditions can be used instead.

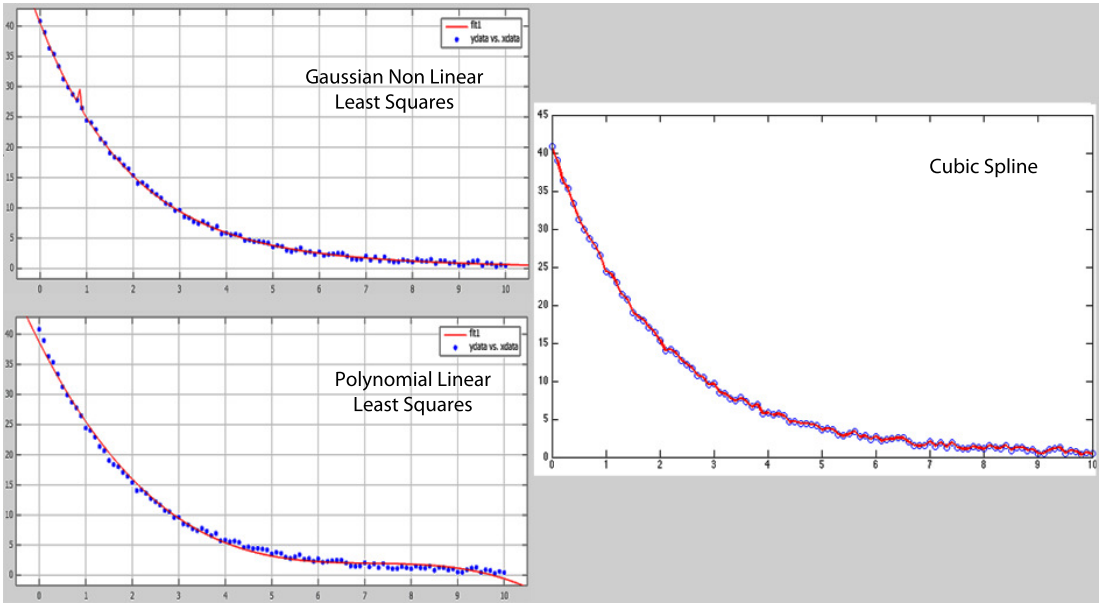

Figure 4.2: Graphs from the test with the function $y = 40 * e^{-0,5*x} + rand(size(x))$

The test is done using the MATLAB Curve Fitting Toolbox and the MATLAB spline tool for the spline cubic interpolation. The functions chosen to do the analysis are the following:

$$y = 40 * e^{-0,5*x} + rand(size(x)) \tag{4.16}$$

$$y = 35 * x^2 + 4 * x - 90 \tag{4.17}$$

$$y = sin(x * rand(size(x))) \tag{4.18}$$

with $x \in [0 : 10]$ with a 0,1 step. While the 4.16 and 4.17 have a path similar to a function, the 4.18 generates a sort of random data.

Results

For each function is done a test with the methods described above, and, for sake of brevity, the results are listed in the table 4.1.

As it can be seen from the result table 4.1, the best fitting method is the cubic spline interpolation. Increasing the randomness of the data, is the only method that can interpolate all the data without losing informations. In the project are used data related to a particular function (not random data), so the

Function $y = 40 * e^{-0,5*x} + rand(size(x))$				
	SSE	**R-Square**	**Adj R-Square**	**RMSE**
Gaussian	8,261	0,9992	0,9991	0,2997
Polynomial	38,09	0,9963	0,9962	0,6266
Spline	0	1	1	0

Function $y = 35 * x^2 + 4 * x - 90$				
	SSE	**R-Square**	**Adj R-Square**	**RMSE**
Gaussian	8,52e+04	0,9993	0,9992	30,43
Polynomial	1,598e-23	1	1	4,059e-13
Spline	0	1	1	0

Function $y = sin(x * rand(size(x)))$				
	SSE	**R-Square**	**Adj R-Square**	**RMSE**
Gaussian	43,92	0,09804	0,01961	0,691
Polynomial	47,95	0,01531	-0,01514	0,7031
Spline	1,2326e-32	1	1	0

Table 4.1: Results of the curve-fitting analysis

spline interpolation is the perfect method to use to fit the data and to find the approximate function.

Now that the algorithm chosen is a spline interpolation, let know something about this type of curve-fitting algorithm.

Spline interpolation is a form of interpolation where the interpolant is a special type of piecewise polynomial called a spline. Spline interpolation is preferred over polynomial interpolation because the interpolation error can be made small even when using low degree polynomials for the spline. Spline interpolation avoids the problem of Runge's phenomenon which occurs when interpolating between equidistant points with high degree polynomials.

A spline is a sufficiently smooth polynomial function that is piecewise-defined, and possesses a high degree of smoothness at the places where the polynomial pieces connect (which are known as knots). The most commonly used splines are cubic spline, i.e., of order 3 and is the order used in the test.

Chapter 5

Results

In this chapter will be presented the results obtained applying the model, presented in the previous chapter, to the function that computes the financing payments.

In the first section will be described all the devices used and the tecniques adopted to execute the function and to collect the results. The final goal is to have results that, after an analysis, can show if there are some reductions on the total consumptions and also on the time execution that actually is the main problem. In this section is presented the environment of the function execution and then it is explained how is measured the energy consumption.

In the second section will be presented the savings obtainable with the application of the approach to the function. There are time savings, related to the execution time of the function, MIPS savings, related to the reduction of the MIPS necessary to the execution and energy savings, related to the power consumption of the mainframes and all the devices connected.

5.1 How the experiments are conducted

As described in the third chapter the function that computes the financing payments is executed using mainframes.

There was not the opportunity to use a mainframe for the project, so the experiments were done using a normal personal computer. It is obvious that there is a big difference from executing the function on a mainframe or on a personal

computer. This difference is accentuated because of the difference between the function original language, COBOL, and the language use for the translation, Java.

In the next section are described the charatteristics of the personal computer used and it is presented also the environment of execution of the function.

5.1.1 Pc technical charatteristics

The original function was created to be executed on mainframes.

Mainframe computers are computers used primarily by corporate and governmental organizations for critical applications, bulk data processing such as census, industry and consumer statistics, enterprise resource planning, and transaction processing. The term originally referred to the large cabinets that housed the central processing unit and main memory of early computers. Later, the term was used to distinguish high-end commercial machines from less powerful units. Most largescale computer system architectures were established in the 1960s, but continue to evolve. Modern mainframes can run multiple different instances of operating systems at the same time. This technique of virtual machines allows applications to run as if they were on physically distinct computers. In this role, a single mainframe can replace higher-functioning hardware services available to conventional servers. While mainframes pioneered this capability, virtualization is now available on most families of computer systems, though not always to the same degree or level of sophistication. Mainframes are designed to handle very high volume input and output (I/O) and emphasize throughput computing. Since the mid-1960s, mainframe designs have included several subsidiary computers (called channels or peripheral processors) which manage the I/O devices, leaving the CPU free to deal only with high-speed memory. It is common in mainframe shops to deal with massive databases and files. Gigabyte to terabyte-size record files are not unusual. [21] Compared to a typical PC, mainframes commonly have hundreds to thousands of times as much data storage online, and can access it much faster.

Mainframe return on investment (ROI), like any other computing platform, is dependent on its ability to scale, support mixed workloads, reduce labor costs, deliver uninterrupted service for critical business applications, and several other risk-adjusted cost factors.

As there was not the opportunity to use a mainframe to analyze the consumptions related to the application of the approach proposed, the computation of the function is done using a personal computer.

A personal computer (PC) is any general-purpose computer whose size, capabilities, and original sales price make it useful for individuals, and which is intended to be operated directly by an end-user with no intervening computer operator. This contrasted with the batch processing or time-sharing models which allowed larger, more expensive minicomputer and mainframe systems to be used by many people, usually at the same time.

The differences of costs and performances between the personal computer and the mainframe is important. It is insane to compare a mainframe to a personal computer, but is possible to analyze the data coming from the pc and then trasform in data related to a mainframe computation.

The technical characteristics of the personal computer used for the computation of the function are presented in the table 5.1.1. The execution of the function

Table 5.1: Technical characteristics of the system used for the testing

Personal computer technical characteristics	
Model	MacBook Pro
Processor Name	Intel Core i7
CPU velocity	2,7 Ghz
CPU number	1
Core number	2
L2 Cache (for each core)	252 KB
L3 Cache	4 MB
RAM	4 GB DDR 3 1333 MHz

was done using the software environment Eclipse.

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system.

Figure 5.1: Energy meter

5.1.2 How to measure the energy consumpion

The increase of the studies on the Green IT makes necessary the execution of accurate measures for monitoring, studying and analyzing the energy consumptions in order to find tecniques and methods for a reduction that could be efficient. An information system is a complex architecture, structured on more than one level. For this reason, the energy consumption are distributed on all the entire system, from the hardware used (CPU, motherboards, hard disks), to the software executed (operating systems, users application). It is important consider all the components and their related energy need.

The only way to correctly analyze the energy consumptions of different software applications and then compare their energy efficiency is to obtain their actual power consumption in an empirical and direct way. There are many different solutions able to perform this task, but all of them require one basic operation: reading the real current absorbed by the computer. Since the voltage is fixed to the nominal fixed voltage (provided by the national energy providers), the power consumption can then be obtained using the formula:

$$Power = I * V * cos\varphi \qquad [W] = [A][V] \tag{5.1}$$

This formula expresses the real power. The term $cos\varphi$ is called *power factor* and varies always between 0 and 1. In the case of processors and computer hardware usually it is a good assumption to assume the $cos\varphi$ to be really close to 1, but this is still an approximation.

One very simple measurement methodology is to use an energy meter, like

the one in figure 5.1. An energy meter is a device that measures the amount of electrical energy supplied to an electrical component.

Another power measurement method is to use a current clamp, like the one

Figure 5.2: Current Clamp

in the figure 5.2, in order to obtain the drained current (and then multiply by the tension to obtain the wattage) or to obtain directly the wattage. A current clamp is an electrical device with two jaws which are possible to open in order to allow clamping around an electrical conductor. This permits to measure the electrical current in the conductor, due to the Hall effect [1], without having to make physical contact with it, or to disconnect it for insertion through the probe. This means that the output value is not biased due to some other voltage drops. Some current clamps have additional functionalities, like support for voltage measure, which allow to measure wattage, or output, which allow to monitor the current value over the time.

The latter meets all the requirements to monitor the overall power consumption of the system if it is used paired with a digital multimeter.

A multimeter or a multitester, also known as a VOM (Volt-Ohm meter), is an

[1]The Hall effect is the production of a voltage difference (the Hall voltage) across an electrical conductor, transverse to an electric current in the conductor and a magnetic field perpendicular to the current. It was discovered by Edwin Hall in 1879. The Hall coefficient is defined as the ratio of the induced electric field to the product of the current density and the applied magnetic field. It is a characteristic of the material from which the conductor is made, since its value depends on the type, number, and properties of the charge carriers that constitute the current.

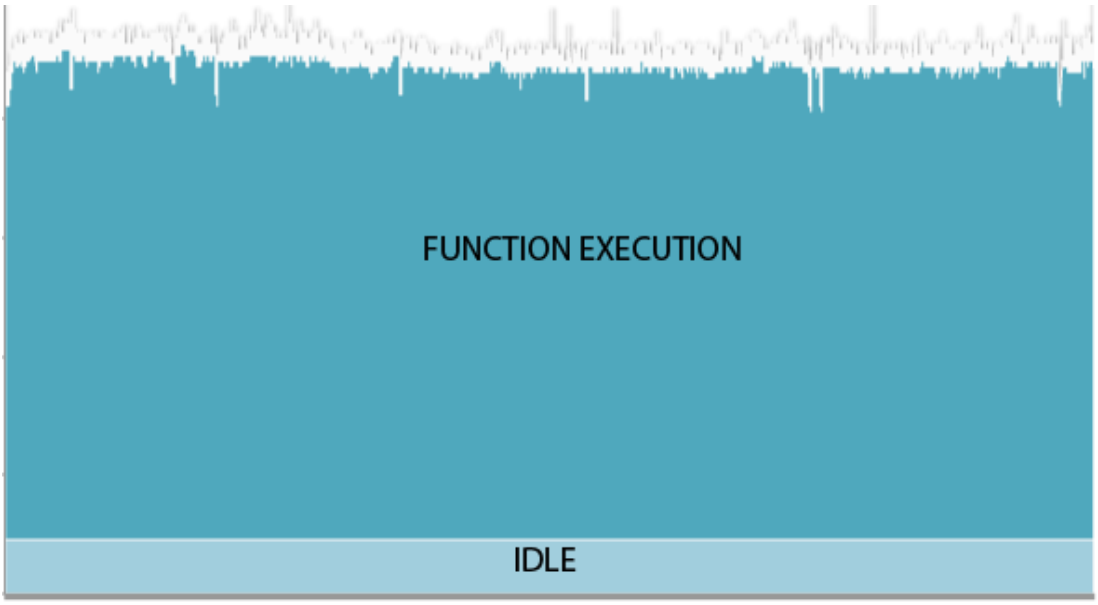

Figure 5.3: Function energy consumption profile vs idle consumption profile

electronic measuring instrument that combines several measurement functions in one unit. A typical multimeter may include features such as the ability to measure voltage, current and resistance.

The multimeter used in the project had a COM port, that made possible the connection with a computer in order to record and acquire the instant values over time. The interface between the computer and the multimeter is a software that adds the functionalities of record and storage the data.

To acquire the data related to the power consumption, the current clamps have to be clamped in the copper wire in which flows the current, and this wire have to be connected to the personal computer used for the computation of the function.

The measure of the current described is influenced by the idle profile of the personal computer. A computer processor is defined as in idle phase when it is not being used by any program. Many programs that use CPU idle time cause the CPU to always be 100% utilized, so that the time spent where the CPU would have been idle is instead spent performing useful computations. This generally causes the CPU to consume more power instead entering in power-safe modes when they are idle.

If the goal is to measure only the consumption caused by the function execu-

tion, it is important to know what is the value of the energy need during the idle phase. Knowing this value, it is possible to subctract it from the consumption measured during the computation. It is possible to see the energy consumption profile of the function in the figure 5.3, where it is shown the difference of the consumptions in the idle phase and during the function execution.

5.2 Model application in the function

The first test done is the application on the function of the modified model described in the fourth chapter. To accomplish this task it was necessary to search and then analyze all the libraries that implement the cubic spline interpolation, finding the one that best fits with the requirements of the model.

The best library that accomplishes the cubic spline interpolation is the one developed by Michael Thomas Flanagan. This class includes all the methods useful for the interpolation and also has the possibility to do the multidimensional spline interpolation. In fact, the first problem of the project was to find an algorithm that permits to have an approximation of a function that is influenced by different indipendent input parameters, not by only a single parameter. The tests done in the beginning were based on two parameters, *consistenza* and *tasso*, as said in the section 3.1.1, then it was introduced a third input value, *n_ giorni*.

The general class in the flanagan library that performs the multi-dimensional spline interpolation is called `PolycubicSpline`. It contains the constructor and the methods for performing a multi-dimensional cubic spline interpolation, i.e. an interpolation within a n-dimensional array of data points, $y = f(x1, x2, x3 \dots xn)$, using a natural cubic splines and where n may take any integer value. The interpolation procedure is recursive.

If the number of dimension, n, of the data array, $y = f(x1, x2, x3 \dots xn)$, is less than five it is suggested to use other classes, like `CubicSpline [n=1]`, `BiCubicSpline [n=2]`, `TriCubicSpline [n=3]` or `QuadriCubicSpline [n=4]` more convenient and slightly more efficient. The test were done using first the `BiCubicSpline`, that contains the constructor and methods for performing an interpolation within a two dimensional array of data points, $y = f(x1, x2)$, using natural bicubic spline. The returned interpolated value is the mean of the values obtained by an interpolation of the originally entered two dimensional array of

tabulated data function values, $y = f(x1, x2)$, and by an interpolation of the transpose of this two dimensional array, $y = f(x2, x1)$. Then, with the addition of the third parameter, the class used was the `TriCubicSpline`, that contains the constructor and methods for performing an interpolation within a three dimensional array of data points, $y = f(x1, x2, x3)$, using a natural tricubic spline. The inner interpolations at the bicubic spline level are obtained as the mean of the values obtained for the two dimensional tabulated values submatrix and for its transpose. At higher levels the only interpolations performed are on the data as entered.

To test the interpolation class it was necessary to create a training set, useful to train the class in such a way to create the function approximation. The training set was created initially selecting the first 50 data coming from the input file, then that number was increased, until it was reached the memory saturation.

Meanwhile it was also created a test set, used for the evaluation of the spline interpolation created with the training set using the flanagan libraries. The creation of the approximed function using the cubic spline interpolation it was done using the constructor shown in the list 5.1, where `x1` and `x2` are the array of the input parameters, and `y` is the matrix of the output values.

```
public BiCubicSpline(double[] x1,double[] x2,double[][] y)
Usage: BiCubicSpline aa = new BiCubicSpline(x1, x2, y);
```

Listing 5.1: Constructor of the interpolation function

The result of this interpolation are unsatisfactory. The approximeted function created with the spline interpolation class produces values not accettable for the project. The problems of this results are maybe attributable to two different causes.

The first reason it is related to the input data. For the creation of the interpolated function, the spline class requires as a parameter three input parameters: two `double` arrays containing the input parameters and a matrix that contains the output values. The two arrays have to be in ascending order, while the matrix is created as the output values coming from the combination of the two arrays, using as indexes the row and the column of the matrix. The construction of the output matrix is shown in the listing 5.2.

In the project, this type of data management creates a *sparse* matrix, a matrix populated primarily with zeros as elements of the table. The presence of

this type of matrix influences the computation of the cubic spline, causing an increment of the difference from the real value and the value obtained with the approximated function given by the spline interpolation.

```
for(int i=0; i<x1.length; i++){
   for(int j=0; j<x2.length; j++)
      yValues[i][j] = 'read statement' or '
                              calculation of f(x1, x2)';
  }
}
```

Listing 5.2: Construction of the matrix y

The second reason of the bad results obtained with the approach is the presence of overfitting.

Overfitting occurs when a statistical model describes random error or noise instead of the underlying relationship. Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model which has been overfit will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data. The possibility of overfitting exists because the criterion used for training the model is not the same as the criterion used to judge the efficacy of a model. In particular, a model is typically trained by maximizing its performance on some set of training data. However, its efficacy is determined not by its performance on the training data but by its ability to perform well on unseen data. Overfitting occurs when a model begins to memorize training data rather than learning to generalize from trend. As an extreme example, if the number of parameters is the same as or greater than the number of observations, a simple model or learning process can perfectly predict the training data simply by memorizing the training data in its entirety, but such a model will typically fail drastically when making predictions about new or unseen data, since the simple model has not learned to generalize at all. In the project, after the creation of the approximed function, the next operation was to verify how good was the interpolation obtained. To accomplish this task it became necessary the adoption the test set, described previously, with the goal to obtain output values near to the real values of the function. The instruction defined in the class to execute the interpolation of new

incoming data is shown in the list 5.3

```
public double interpolate(double xx1, double xx2)
Usage: y1 = aa.interpolate(xx1, xx2);
```

Listing 5.3: Interpolation method

The results obtained from the computation of the interpolation given new data are not good for the goal of the project. When the input data are equal or similar to values used in the training set, the interpolation function gives good output value, with an irrelevant difference from the real value computed. On the other hand the interpolation function, using input data never processed before, processes an awful output value, maybe due to the overfitting present in the input data used for the creation of the function.

With these results it was necessary to reanalyze the entire function using another approach, in order to confirm that the application of the memoization techniques in the project creates an advantage respect to the present situation.

The new analysis was executed in order to have a complete knowledge about

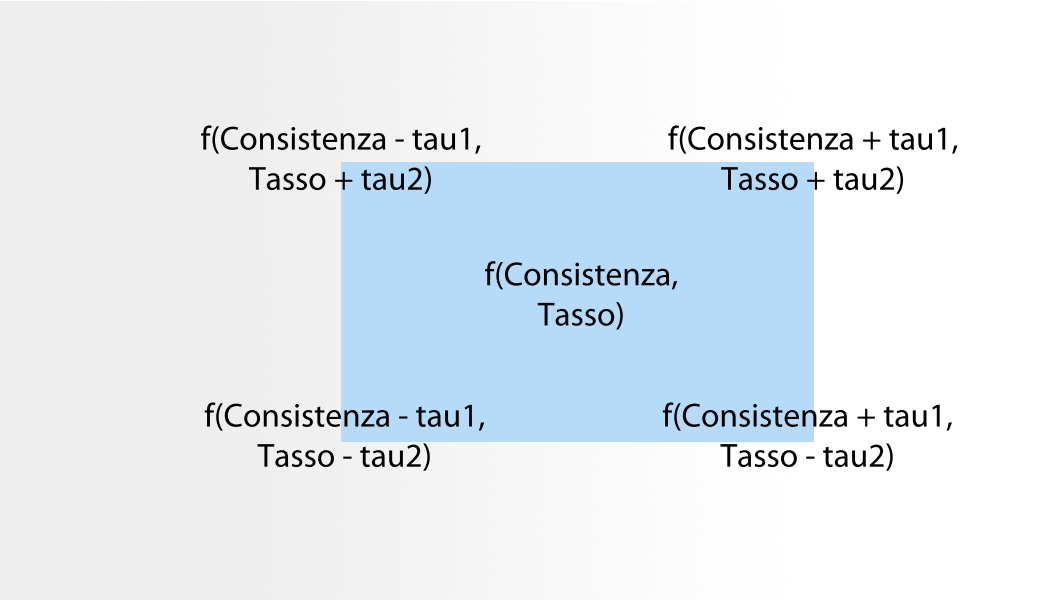

Figure 5.4: Memoization area

the behaviour of the function when there is a little change in the input data. The change in the input data was the presence of τ added or subctructed to each

parameter (different value of τ for different parameters). Then it was calculated the output value executing the function and saving the results. In order to have a deep knowledge on the behaviour of the financing payments function it was necessary to repeat the execution using different values of τ, covering all the user accettable areas.

While the initial idea was adding (or subctracting) a SLA value to the output value and then calculate, with an inverse function, the values of the τ related to every single input parameter, the new idea permits to know the maximum (or minumum) value of the SLA in base of the τ choosen on each single input parameter. In this way, if the SLA found is satisfactory, according to the users requirements, all the input values that are inside of the area, that has as boundaries the combination of the input parameters and τ (shown in the figure 5.4), can be memoized.

5.3 Savings

In this section are described the results of the project, presented as savings obtained with the application of the approximed memoization approach described in this thesis.

The idea is to divide the section in four parts, where in each one division is presented a different type of saving obtained.

The first part contains the description of the time enhancements gained from the procedure optimization, with all the data related to the application of the approximend memoization approach. The time savings are an important goal of the project, because the present situation creates a great inconvenience both for the employees and for the clients. In the second part are described the energy savings, related to the consumption during the function execution of all the devices used in the computation. As third saving gained, there is the reduction of the MIPS used in the computation, with the consequent renegoziation of the MIPS supplier contract. Last but not least, ther is the decription of the cost savings, with a good analysis on the term TCO (Total Cost of Ownership).

5.3.1 Time Savings

In the third chapter of this document it was mentioned that one of the big problems, or maybe the biggest problem, created by the financing payments function computation, was the high execution time. This is due to the increase of the data to compute during the night, when the mainframes are dedicated to execute only this function. Currently the execution doesn't end before the opening of all the filials, so it blocks the terminal used by the employees, with the relative inefficiency for all the clients. Reducing the execution time has also some advantages in terms of the energy consumption and also on the MIPS used for the execution.

The project has the goal to improve the computation in order to decrease the time require to complete the function execution. To accomplish this task, the first solution adopted was the introduction of the memoization approach, i.e. reading a precomputed value inside the memory instead computing it. The studies done in this thesis refers to use the memoization for the software energy efficiency, basing the approach on the automatic identification of functions that are convinient to tabulate. Memoizing a function leads to energy savings only if the function is invoked several times with the same parameters, so that the results can be read from the memory rather then re-computed.

The second improvement was introducing the concept of Service Level Agreement (SLA) into the memoization concept, in order to have less elements in the memory and to reduce the time to search and then read an element from the memory. Basing on the distribution of the input data the memory was divided in two different parts. The first one, that is the bigger, is used for saving the values that are in the memoization interval. In this part all the values are memoized in order to cover completely the memoization interval respecting the output SLA defined.

The second part is used to save the values that are outside the memoization interval, memorizing them precisely.

The last improvement done is the use of the quadtree as data structure in the memory. A quadtree is a tree data structure in which each internal node has exactly four children.

Consumption model

In order to analyze the timing savings that the three actions described above have brought, it was created a consumption model. The application of this model gives the percentage of the execution time reduction.

The goal is to have a profit using the memoization instead of executing the function. It is possible to write this sentence in this way (5.2):

$$T_{ORIG} > T_{MEMO} \tag{5.2}$$

where T_{ORIG} is the execution time of the function, defined in the 5.3, and T_{MEMO} is the memoization time.

$$T_{ORIG} = \sum_{1}^{n} T_e \tag{5.3}$$

where T_e is the execution time of the single input value and n is the number of the input values and is defined as the sum of three values

$$n = m + c + x \tag{5.4}$$

where m is the number of the input data memoized, c is the number of the input value calculated and x is the number of the copies of the input value memoized, so they don't afflict the memory space.

The memoization time is defined as:

$$T_{MEMO} = \sum_{1}^{m+x} T_m + \sum_{1}^{c} (T_m + T_e + \beta * T_t) \tag{5.5}$$

where T_m is the time to read a memoized value that is stored in the memory, β is the frequency value of the invocation of the trade-off module, and T_t is the time required to execute the trade-off module. This module is executed few times a month, so it is possible to set the value of β equal to zero. So, the formula of the memoization time is:

$$T_{MEMO} = \sum_{1}^{m+x} T_m + \sum_{1}^{c} (T_m + T_e) \tag{5.6}$$

Substituting in the 5.2 the definitions of the memoization and the execution time (5.3 and 5.6), the new formula is the one described in the 5.7

$$\sum_{1}^{n} T_e > \sum_{1}^{m+x} T_m + \sum_{1}^{c} (T_m + T_e) \tag{5.7}$$

Doing some calculation on the 5.7 is possible to find a simple formula that can be used for the calculation of the gain obtained with the application of the approach discussed in this thesis.

$$\sum_{1}^{n} T_e > \sum_{1}^{m+x} T_m + \sum_{1}^{c} T_m + \sum_{1}^{c} T_e \tag{5.8}$$

$$\sum_{1}^{m+x} T_e > \sum_{1}^{n} T_m \tag{5.9}$$

$$\sum_{1}^{m+x} T_e - \sum_{1}^{n} T_m > 0 \tag{5.10}$$

From the 5.10 is possible to write the formula that expresses the percentage gain.

$$G_{\%} = \frac{\sum_{1}^{m+x} T_e - \sum_{1}^{n} T_m}{\sum_{1}^{n} T_e} * 100 \tag{5.11}$$

With the 5.11 it is possible to obtain the value of the time percentage gain caused by the application of the memoization approach on the function execution. The interesting terms in the formula are:

- m e x, that are the indexes of the sum. Their value depends on the quantity of memory allocated for the memoization approach.
- T_e, it is the execution time of the function without the application of the memoization approach. It is calculated experimentally using the code in the list 5.4.
- T_m, it is the time to read a memoized value. Its value it was estimated.

The execution time data capture was repeated several times, in order to use for the analysis the mean values. This can assure that the data analyzed are

coming from different execution in the same conditions.

```
inizio = System.nanoTime();
tempo = System.nanoTime()-inizio;
```

Listing 5.4: Code for the evaluation of the execution time. The first instruction is inserted before the reading of the input values used for the function execution, the second is inserted at the end of the execution, before the new reading of the input values.

As done with the energy consumption measures, all the programs that were running in the idle phase were killed, in order to have only the data related to the function and not infected by some other executions.

Results

	Real Interval		Memoized Interval	
Parameters	**Min**	**Max**	**Min**	**Max**
Consistenza	0,26	176.510.709,30	870,00	50.000,00
Tasso	-	19,25	0,20	1,00

Total rows	408.863		**# parameters**	2
Total rows in the interval	202.571		**# cell divisions**	4
% rows in the interval	53,9%		**Mem. access time**	10

	Time [ns]
Max Execution Cost	28.643.217.000
Min Execution Cost	1.000
Mean Execution Cost	31.419.508

Table 5.2: Input data summary table

In the table 5.2 it is possible to see all the informations related to the input data. In particular it is possible to see the statistics of the memoization interval choosen. With that interval, that is little considering the entire interval of the

input data, is possible to cover over the 50% of the entire set of data, making the approach of the memoization a good solution. Futhermore this data distribution allows the application of the approximed memoization approach, in order to save some memory space.

Still in the same table are indicated the values of the execution time measured. In particular it is shown the maximum, the minimum and the mean value coming from the various experiment conducted. In the analysis is used the mean value, because it is possible to see that the distance between the maximum value and the mean value is huge. This means that the maximum value can not rappresent the set of the input data better than the mean value. For the same reason it is not used the minimum value.

The latest informations coming from the table 5.2 are the values of the parameters of the function. In particular, it is shown the number of the input parameters that have an influence on the behaviour of the execution, the number of the division cell of the quadtree (calculated as $2^{\#parameters}$ and the time to access to the memory.

	TAU (EURO)	Quad-Tree depth	Cells	Memory Space (GB)	Tm	Interval Depth	Memoization (GB)
Consistenza Tasso	0,5 0,00002	28 20	24.019.198.012.642.600	4.197.354.853	560	17	1.000,73
Consistenza Tasso	1 0,00005	27 19	6.004.799.503.160.660	1.049.338.713	540	16	250,18
Consistenza Tasso	2 0,0001	26 18	1.501.199.875.790.160	262.334.678	520	15	62,55
Consistenza Tasso	5 0,0005	25 15	375.299.968.947.541	65.583.670	500	13	3,91
Consistenza Tasso	10 0,001	24 14	93.824.992.236.885	16.395.917	480	12	0,98
Consistenza Tasso	20 0,002	23 13	23.456.248.059.221	4.098.979	460	11	0,24

Table 5.3: Memory data table

Using the parameters listed in the table 5.2 is possible to do the analysis,

beginning from the construction of the quadtree structure, used for memoizing the data in the memory. It is possible to see the data related to this analysis in the table 5.3. For each of the two input parameters (tasso and consistenza) it was calculated the depth of the quadtree, finding a different values because of the τ values assigned to the parameters are not the same. The reason of this difference is the domain of the two parameters. *Tasso* belongs to the set of values between 0 and 20, so its τ value has to be a decimal number. On the other hand, *Consistenza* belongs to a set of values bigger than the one described before, so it is a good idea to use an integer τ value.

The formula of the calculation of the depth of the quadtree structure is:

$$depth = log_2(\sqrt{\tau_{input}}) \tag{5.12}$$

The second data in the table is the number of cells that the quadtree has to have if the approach choosen is to memorized all the values in the memory using that structure and the following column converts that number into the memory space required. To accomplish these calculation the formulas adopted are:

$$\#cell = \frac{(\#cell\_divisions^{max(depth)-1} - 1)}{3} + \#cell\_divisions^{max(depth)-1} \tag{5.13}$$

$$MemorySpace = \frac{\#cell * byte\_dimension}{10^9} \tag{5.14}$$

The latter one is the important data that permits to have a comparison between the classical theory of store the data in the memory and the new approach of the memoization. It also explains the need to have a method that improves the actual situation, because, with the highest value of τ, the memory space required is over the 4 billion of GB.

To execute the comparison it is necessary to compute the data coming from the application of the memoization approach. The first one is the time to read a memoized data, that consist on the time to access into the memory and then the time to search and than read the value stored in the quadtree structure. These actions can be written in a formula like the one that follows:

$$MemoizationTime = MemAccessTime * \#Parameters * (QuadtreeDepth) \tag{5.15}$$

where the $QuadtreeDepth$ is the bigger one between the depth values of the two parameters.

The depth of the quadtree created using the approximed memoization approach is calculated with this formula:

$$IntervalDepth = log_2 \left(\frac{Range}{\sqrt{2} * \tau} \right) \tag{5.16}$$

where with $range$ is indicated the range of the interval (table 5.2). The value of the depth with the application of the memoization interval is choosen as the maximum value between the two one generated by the two different parameters. As it is possible to see from the table 5.3, the depth, in the case of the most precise input (where the τ values are the smallest one), is decreased from the value 28 to the value 17. This implies that with the approach of the approximed memoization it is more faster access and read a value in the memory.

The other value that can be useful to compare is the one related to the memory occupation. In the case of the approximed memoization approach that value is calculated with the same one formula written for the normal approach, that it is indicated in the 5.14. Despite of the same formula, the results are totally different. It is possible to see that with the approximed memoization the interval of data is saved in 1 TB (TeraByte). This is an optimum result, because it saves a lot of memory space.

With all of these data, it is possible to compute and then analyze the overall gain that the approximed memoization approach gives to the function in order to reduce the execution time.

The table 5.5 shows the values of the gain obtained with the approach of the memoization method to the function execution. With the label $output\_sla$ it is indicated the mean error committed using the memoization approach instead executing the function. In the worst case the difference from the two values it is 1,60 Euro, that is a good result analyzing the context of the execution. From this table it is possible to see that the memoization approach has pointed out an elevated reduction on the execution time of the function. In particular, there is a reduction of about the 47% using the maximum precision on the output (0,02 Euro).

This reached goal permits to assert that the application of the memoization

			MEMORIA ALLOCATA [GB]													
			0,5		1		2		4		8		16		32	
			Msla	Mmemo	Msla	Mmemo	Msla	Mmemo	Msla	Mmemo	Msla	Mmemo	Msla	Mmemo	Msla	Mmemo
TAU [€]		OUTPUT SLA	0,45	0,05	0,90	0,10	1,80	0,20	3,60	0,40	7,20	0,80	14,40	1,60	28,80	3,20
Consistenza	0,5	0,022	46,08%		46,10%		46,15%		46,24%		46,44%		46,83%		47,60%	
Tasso	0,00002															
Consistenza	1	0,047	46,15%		46,24%		46,44%		46,83%		47,60%		49,16%		52,26%	
Tasso	0,00005															
Consistenza	2	0,089	46,44%		46,83%		47,60%		49,16%		52,26%		58,47%		70,89%	
Tasso	0,00010															
Consistenza	5	0,403	52,26%		58,47%		70,89%		95,73%		99,998%		99,998%		99,998%	
Tasso	0,00050															
Consistenza	10	0,792	70,89%		95,73%		99,998%		99,998%		99,998%		99,998%		99,998%	
Tasso	0,00100															
Consistenza	20	1,600	99,999%		99,999%		99,999%		99,999%		99,999%		99,999%		99,999%	
Tasso	0,00200															

Figure 5.5: Gain obtained with the memoization approach

approach improves the performances and the consumptions comparing to the normal function execution. This is a good start for propose and then apply the memoization approach to other financial applications.

5.3.2 Energy savings

Before starting with the execution of the test, it was done a monitoring of the system, waiting for an equilibrium condition with a costant idle. This approach makes possible the comparison between the instant values of the consumptions during the function execution and the consumptions during the idle phase, even in different sessions, but executed using a similar system. Due to the high variability during the idle phase, the real data on the consumption are not directly comparable outside the same session, even if they are executed on the same system. It is non realistic to mantain a hardware and a software system equivalent during the time. The daily use, without the installation of new software, brings anyway variations. All the values reported in this section are obtained using the mean values of the different tests results. In particular, in order to have the scientific prove of the derived data, the tests are done with the goal to having a confidency below of the 5%.

To give an indication of the scale of the overall consumption of the system used during the test (5.1.1) in the figure 5.6 is shown one of the consumptions trend during the idle phase. The instant mean value of the energy need, during an idle phase (90 s of observation), it is 93,867 W , with an integral consumption of 18374,70 J (about 5,10 Wh).

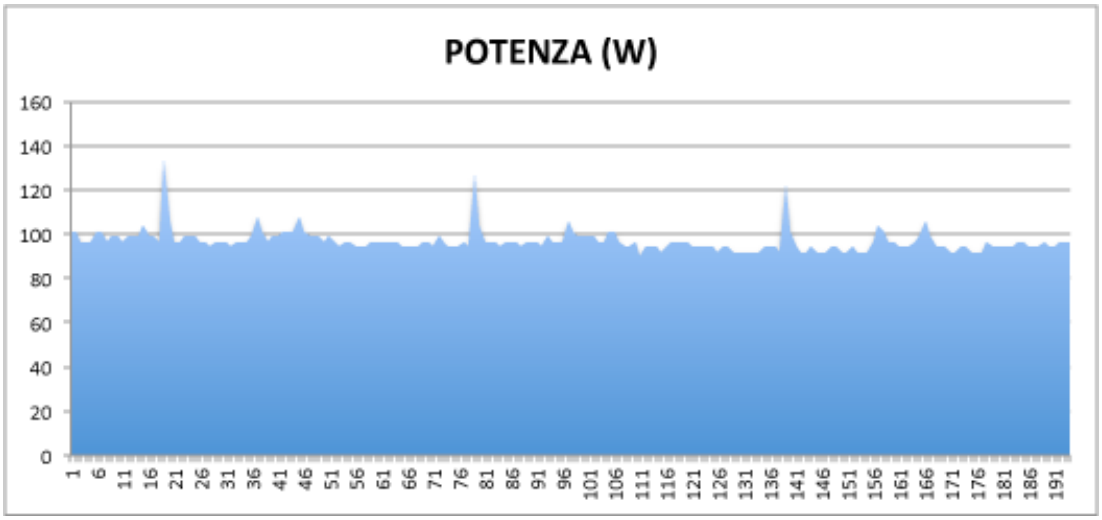

Figure 5.6: Idle consumption of the system.

The analysis of the energy consumption during the function execution are shown in the table 5.4. All the values in that table are computed as a mean value of the different analysis, and the number of the analysis one is associated to a value of confidence under the 5%.

The second column of the table is related to the set of data with the parameter $formula$ equal to 251 or 254, while the third column is the total amount of the data in the input file. The data were measured for the second column and then it was computed the relative field on the third column. The J of the function was trasformed in kWh using this formula:

$$kWhfunction = Jfunction/3600/1000 \tag{5.17}$$

Then, with this data was possible to compute the annual value of the kWh consumed by the function execution and then the total cost of that operation. For the data related to costs was used the ratio euro/kWh setted to 0,189, as a normal contract between the energy supplier and a private citizen.

To have, finally, the data of the global consumption during the function execution, it was used the definition presented in the second chapter (2.3), multiplying the total consumption of the function for the factor 28, that helps to include the energy consumption of all the devices connected or used in order to make the system works.

As the main topic of the thesis is the Green ICT, this number were trasformed in something related to the green area, like the quantity of CO_2 emissions and the number of trees planted in order to label the function as a "Zero Emissions Function". The conversion between the total amount of kWh and the Kg of CO_2 realesed in the air is 0,55 g for Wh. Then, the last conversion, from g to kg convertible by the trees is setted with the default value of 20 kg/year. This is the minimum value of absorption potential that a tree can does in a natural contest with a good growth factor. In this environment the value of the absorption potential ranges from the 20 to the 45 kgCO_2/year, in a time range between the 20 and the 30 years (it is possible to see the CO_2 cycle in the image 5.7).

Energy Consumption		
N	408.863	1.200.000%
Mean function J	2.492.350,63	7.314.970 %
Mean function kWh	0,69	2 %
Annual Function kWh	252,70	742 %
Annual Cost (Euro)	47,76	140 %
Total kWh	7.075,51	20.766,39 %
Total Costs (Euro)	1.337,27	3.924,85%
CO_2 Emission	3.891,53	11.421,51 %
Trees	194,58	571,08 %

Table 5.4: Energy consumption of the normal function execution.

The reduction of the energy need caused by the application of the memoization approach is not a data known, because of the part of the storage on the quadtree structure, at the writing moment, is not done yet. What it is possible to say is that, if we have at minimum a 47% of the timing saving, so the devices are used less than during the normal execution. Maybe the use of this devices has a value decreased more or less about the same percentage of the time saving.

In this scenario the remaining time can be used for the execution of some others operations that maybe in the actual context are not done or they are

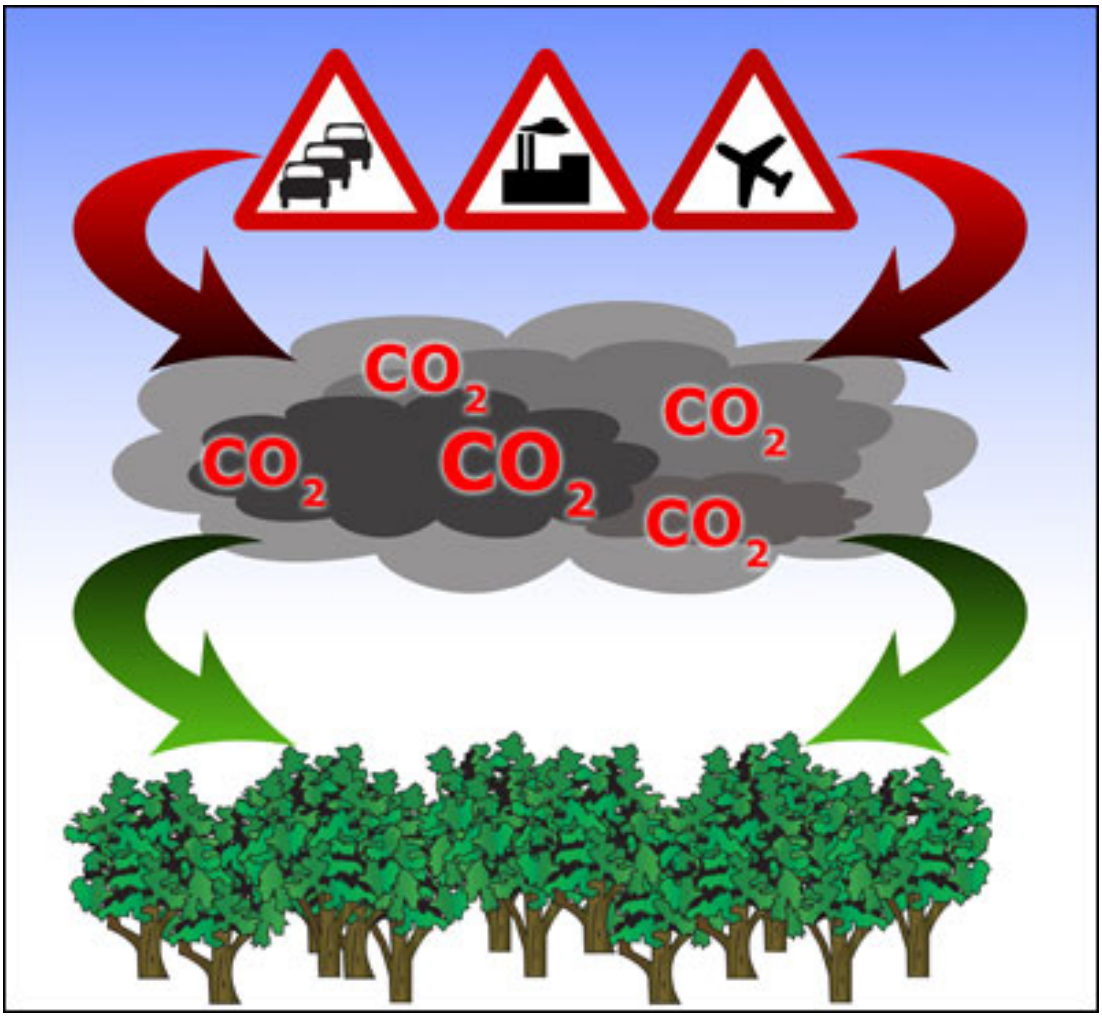

Figure 5.7: CO2 Cycle

postponed to moments in which the load is low.

5.3.3 MIPS savings

The function that computes the financing payments was created in order to be executed on mainframes. Generally speaking, the supplier of the hardware and the software used to run the mainframes stipulates a contract related to the load on the system created by the execution of the function.

It is common to analyze this load and so stipulating the contract between the two parts using the MIPS (Million of Instruction Per Second) as a reference

value.

The MIPS is the unit of measurement of the frequency of instruction executed done by a microprocessor. The instructions are the assembly ones of the processor in exam. These instructions are, in general, simple (like a single sum or a single test to evaluate a condition). A normal computer program is composed by thousands or million of this instructions, automatically created by a compiler. The number of the instruction that a processor can execute in one second, expressed in millions, is its MIPS.

The formula for computing the MIPS value is

$$MIPS = \frac{N_i}{CPU_{Time} * 10^6} \tag{5.18}$$

where N_i is the number of instructions executed by the microprocessor in a time period or on a program, CPU_{time} is the time in milliseconds to execute the N_i instructions, and 10^6 is the ratio between seconds and milliseconds.

Analyzing that there is a big reduction on the execution time, it is possible to assert that also the number of the MIPS is decreased, because the introduction of the memoization approach permits to delete some function execution cycles introducing some reads from memory instructions, that are faster and less MIPS consuming in contrast to the normal function execution.

With this reduction, it is possible to redefine the new contract with the supplier, based on the new quantity of MIPS required for the execution of the function. The new agreement permits to lowering the costs sustained by the client of the system. On the contrary it is possible to maintain the same contract, using the remaining MIPS for the execution of some others functions, reaching anyway a reduction of the amount of money related to all the function executions.

5.3.4 Cost saving

All the results obtained, shown in the subsections before, are now analyzed in order to evaluate their impact on the total cost of ownership (TCO). It will be defined the TCO of the analyzed system, including the evaluation of the hardware components on which it affects directly the memoization approach.

The total cost of ownership (TCO) is a financial estimate whose purpose is to help consumers and enterprise managers determine direct and indirect costs of a

product or system. It is a management accounting concept that can be used in full cost accounting or even ecological economics where it includes social costs. For manufacturing, as TCO is typically compared with doing business overseas, it goes beyond the initial manufacturing cycle time and cost to make parts. TCO includes a variety of cost of doing business items, for example, ship and re-ship, opportunity costs, while it also considers incentives developed for an alternative approach. Incentives and other variables include tax credits, common language, expediated delivery, customer oriented supplier visits.

TCO, when incorporated in any financial benefit analysis, provides a cost basis for determining the total economic value of an investment.

A TCO analysis includes total cost of acquisition and operating costs. A TCO analysis is used to gauge the viability of any capital investment. An enterprise may use it as a product/process comparison tool. It is also used by credit markets and financing agencies. TCO directly relates to an enterprise's asset and/or related systems total costs across all projects and processes, thus giving a picture of the profitability over time.

TCO analysis was popularized by the Gartner Group in 1987 [2].The roots of this concept date at least back to the first quarter of the twentieth century. Many different methodologies and software tools have been developed to analyze TCO. TCO tries to quantify the financial impact of deploying an information technology product over its life cycle. These technologies include software and hardware, and training.

Technology deployment can include the following as part of TCO:

- Computer hardware and programs
  - Network hardware and software
  - Server hardware and software
  - Workstation hardware and software
  - Installation and integration of hardware and software
  - Purchasing research
  - Warranties and licenses
  - License tracking - compliance
  - Migration expenses

  - Risks: susceptibility to vulnerabilities, availability of upgrades, patches and future licensing policies, etc.

- Operation expenses
  - Infrastructure (floor space)
  - Electricity (for related equipment, cooling, backup power)
  - Testing costs
  - Downtime, outage and failure expenses
  - Diminished performance (i.e. users having to wait, diminished money-making ability)
  - Security (including breaches, loss of reputation, recovery and prevention)
  - Backup and recovery process
  - Technology training
  - Audit (internal and external)
  - Insurance
  - Information technology personnel
  - Corporate management time

- Long term expenses
  - Replacement
  - Future upgrade or scalability expenses
  - Decommissioning

Comparing the TCO of existing versus proposed solutions, the consideration should put towards costs required to maintain the existing solution that may not necessarily be required for a proposed solution. Examples include cost of manual processing that are only required to support lack of existing automation, and extended support personnel.

All the voices mentioned above can be grouped into six macro-factors of the TCO cost.

- Hardware purchase costs
- Software developmant and maintenance costs
- Energy and cooling costs
- Hardware maintenance costs
- Server room management costs
- Hardware decommissioning costs

According to some studies of the Politecnico di Milano based on the impact of the energy costs on the total cost of the ICT [5], each euro invested for acquiring new servers needs 0,7 euro for the annual energy cost for supply and cooling all the devices, with a ratio of 0,7:1. In addition the cost of the hardware maintenance is correlated to the acquiring costs, with a ratio of 2:3. Supposing that the software development and maintenance costs are proportional to the hardware ones with a ratio of 3:1, and knowing that the software maintenance requires over that the 50% of the development costs, it is possible to draw a model of the TCO cost, subdivided by the six macro-factors, like the one shown in the figure 5.8.

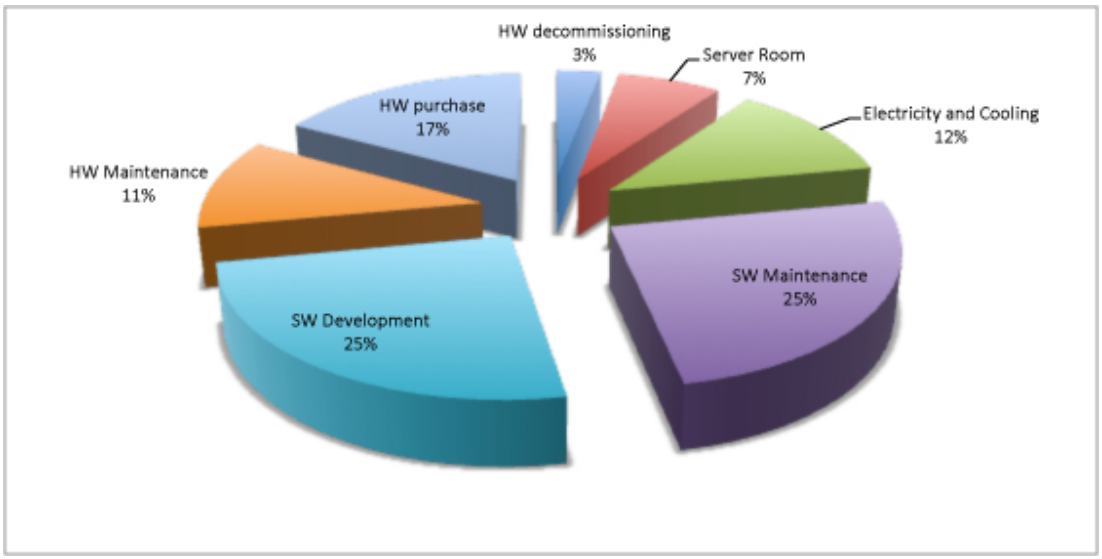

Figure 5.8: TCO Model

Obviously the costs for the hardware purchase and for the software development have to be considered only in the initial phase of the project. On the other

hand, the costs of the hardware decomissioning are present only at the end of the work lifecycle. The cooling and the energy need of the devices costs are present during all the lifecycle of the machines, so the software mainteinance, thinking about not only to corrective or adaptative operations, but also preventive or improvement one.

The use of the memoization approach for the energy efficiency not only has an effect on the cooling or electricty costs, but also on some other voices of the TCO model presented before, with suitable weights. In particular, the savings gained have a direct incidence on the hardware purchase and mainteinance costs, because to satisfy the computational load required the number of the devices used is less than the one used in the actual situation.

For example, what two servers, at the 100% usage, can do, it can be done with only one of this machine, because, in the worst case, the memoization introduces a timing saving of the 47% of the actual execution time.

On the other hand, the energy efficiency is not affected by the software development and mainteinance costs. New implementations or changes don't have big variations on the costs.

Chapter 6

Conclusions

In this chapter are presented the conclusions related to the project done in this thesis. There is an initial part that describes the works done and then the results obtained with the application of the approximed memoization method.

In the second section are then presented some possible future works, that can be done in order to improve the method presented in this thesis.

6.1 Conclusions

In this thesis work it was analyzed the problem of the optimization of the energy consumptions generated by the software, with a particular focus on the improvement on the execution time.

In particular, the studies were done on a particular function, the one that computes the financing payments. This choice was done because a big Italian bank has the problem that this function takes too much time on its execution and so it finish its computation after the fillials opening, causing some problems on the terminals.

Analyzing the context of that function, the solution choosen was to apply the concept of the memoization in order to have an improvement on the consumptions and on the execution time. The memoization is an optimization technique used primarily to speed up computer programs by having function calls avoid repeating the calculation of results for previously processed inputs. A memoized function "remembers" the results corresponding to some set of specific inputs. Subsequent calls with remembered inputs return the remembered result rather

than recalculating it, thus eliminating the primary cost of a call with given parameters from all but the first call made to the function with those parameters.

To "remember" the input parameters and then the output generated using that values, the solution chose was to store the value in the memory, in particular adopting a specific memory structure, called quadtree. A quadtree is a tree data structure in which each internal node has exactly four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions.

The next idea in order to save some memory space and also to reduce more the execution time was to modify the memoization approach. The new proposal was to use an approximation of the function, reducing the set of the input-output values saved in the memory. This operation was executed adding a value, called Service Level Agreement (SLA) to the output values stored.The SLA is a part of a service contract where the level of service is formally defined. In the project, the SLA is defined as the precision needed by the final users on the value of the financing payment. From this value was compute the relative input values and the distance from the real input value and the one compute with the SLA value was called τ.

The idea was to memoized some input values with their τ (clearly also the output value with their SLA) and if the next input value was not inside of the intervals stored in the memory its output value is computed using the approximated function. Otherwise, if it is inside of one of the input intervals memoized, its output value is read from the memory, avoiding all the computing operations.

The approximation method chosen was the spline interpolation and in particular, it was found a Java library that implements this type of interpolation using, as input, more than one parameter. The task of this library was to create an approximated function that, after a training session, it creates the approximated function, that it will be used instead computing the original one.

This goal was not reached, because the data used as input values creates a matrix too sparse and then it is impossible for the library to create a good interpolated function. With this problem, the output created by the Java library was a function afflicted by an overfitting problem.

In order to validate the approach proposed, the idea changed and, instead processing the input given an ouput with the SLA, it was inverted the flow of

computation. It was added to the input parameters a τ value and then it was computed the output with the relative SLA. This operation was useful to understand how the approach can cover the input set and how much savings generates.

The results of the application of the last method descripted were good. It was discovered that the memoization approach on the case study of the financing payments function generates an higher gain on the execution times. The reduction, using the maximum precision on the outputs (0,02 euro), it is about of the 47%.

Related to this reduction, there is an improvement on the energy consumption of the system and so an economical save for the bank.

It is possible to say that, the generality of the approach and its automation permit to apply this method on other financing applications, and this is one of the part described in the next section.

6.2 Future works

The results reached applying the memoization on the financing payments function permits to consider some future works in order to improve the method and maybe extend it to some other applications.

The first future work that have to be done is to apply the memoized function to all the set of the input values. In the thesis, the set was reducted to the parameters that have in the field `formula` the values `251` and `254`. Then it is important to analyze and then introduce an approximation method that can substitute the original function and it can also generates excellent results.

All the results in the thesis were theorical because the part of the storage in the quadtree was not realized yet. So the next step, to analyze how really the memoization approach can reduce the execution time of the function, it is to implement and then test the quadtree structure and all the methods use to save and read in the memory. After this operations a first program pilot can be realized.

If the results coming from the realization of the pilot are good enough it is possible to think that the memoization approach can be extended to some others applications that belong to the economy world, reducing drastically their total energy consumptions.

Appendix A

Java-Cobol

In this chapter it will be presented a kind of manual for the porting from the COBOL language to the Java.

In the firsts sections are presented the two languages, with their own characteristics. Then there will be a detailed section on the conversion from the COBOL to the Java. This section is subdivided into three categories: structures, control instructions and statements.

A.1 COBOL

COBOL is one of the oldest programming languages, primarily designed by Grace Hopper. Its name is an acronym for COmmon Business-Oriented Language, defining its primary domain in business, finance, and administrative systems for companies and governments.[18]

A.1.1 History and specification

The COBOL specification was created by a committee of researchers from private industry, universities, and government during the second half of 1959. The specifications were to a great extent inspired by the FLOW-MATIC language invented by Grace Hopper, commonly referred to as "the mother of the COBOL language." The IBM COMTRAN language invented by Bob Bemer was also drawn upon, but the FACT language specification from Honeywell was not distributed to committee members until late in the process and had relatively little impact.

FLOW-MATIC's status as the only language of the bunch to have actually been implemented made it particularly attractive to the committee.[24]

The scene was set on April 8, 1959 at a Conference on Data Systems Languages (CODASYL) for computer manufacturers, users, and university people, at the University of Pennsylvania Computing Center. The United States Department of Defense subsequently agreed to sponsor and oversee the next activities. A meeting chaired by Charles A. Phillips was held at the Pentagon on May 28 and 29 of 1959 (exactly one year after the Zurich ALGOL 58 meeting); there it was decided to set up three committees: short, intermediate and long range (the last one was never actually formed). It was the Short Range Committee, chaired by Joseph Wegstein of the US National Bureau of Standards, that during the following months created a description of the first version of COBOL. The committee was formed to recommend a short range approach to a common business language. The committee was made up of members representing six computer manufacturers and three government agencies. The six computer manufacturers were Burroughs Corporation, IBM, Minneapolis-Honeywell (Honeywell Labs), RCA, Sperry Rand, and Sylvania Electric Products. The three government agencies were the US Air Force, the Navy's David Taylor Model Basin, and the National Bureau of Standards (now National Institute of Standards and Technology). The intermediate-range committee was formed but never became operational. In the end a sub-committee of the Short Range Committee developed the specifications of the COBOL language. This sub-committee was made up of six individuals:

- William Selden and Gertrude Tierney of IBM
- Howard Bromberg and Howard Discount of RCA
- Vernon Reeves and Jean E. Sammet of Sylvania Electric Products [29]

The decision to use the name "COBOL" was made at a meeting of the committee held on 18 September 1959. The subcommittee completed the specifications for COBOL in December 1959. The first compilers for COBOL were subsequently implemented in 1960, and on December 6 and 7, essentially the same COBOL program ran on two different computer makes, an RCA computer and a Remington-Rand Univac computer, demonstrating that compatibility could be achieved.

A.1.2 Criticism and defense

Lack of structurability

In his letter to an editor in 1975 titled "How do we tell truths that might hurt?" which was critical of several programming languages contemporaneous with COBOL, computer scientist and Turing Award recipient Edsger Dijkstra remarked that "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense."[10]

In his dissenting response to Dijkstra's article and the above "offensive statement," computer scientist Howard E. Tompkins defended structured COBOL: "COBOL programs with convoluted control flow indeed tend to 'cripple the mind'," but this was because "There are too many such business application programs written by programmers that have never had the benefit of structured COBOL taught well..."[26]

Additionally, the introduction of OO-COBOL has added support for object-oriented code as well as user-defined functions and user-defined data types to COBOL's repertoire.

Compatibility issues after standardization

COBOL 85 was not fully compatible with earlier versions, resulting in the "cesarean birth" of COBOL 85. Joseph T. Brophy, CIO, Travelers Insurance, spearheaded an effort to inform users of COBOL of the heavy reprogramming costs of implementing the new standard. As a result the ANSI COBOL Committee received more than 3,200 letters from the public, mostly negative, requiring the committee to make changes. On the other hand, conversion to COBOL 85 was thought to increase productivity in future years, thus justifying the conversion costs.[15]

Verbose syntax

COBOL syntax has often been criticized for its verbosity. However, proponents note that this was intentional in the language design, and many consider it one of COBOL's strengths. One of the design goals of COBOL was that non-programmers-managers, supervisors, and users-could read and understand the code. This is why COBOL has an English-like syntax and structural elements-

including: nouns, verbs, clauses, sentences, sections, and divisions. Consequently, COBOL is considered by at least one source to be "The most readable, understandable and self-documenting programming language in use today. [...] Not only does this readability generally assist the maintenance process but the older a program gets the more valuable this readability becomes."[1] On the other hand, the mere ability to read and understand a few lines of COBOL code does not grant to an executive or end user the experience and knowledge needed to design, build, and maintain large software systems.

Other defenses

Additionally, traditional COBOL is a simple language with a limited scope of function (with no pointers, no user-defined types, and no user-defined functions), encouraging a straightforward coding style. This has made it well-suited to its primary domain of business computing-where the program complexity lies in the business rules that need to be encoded rather than sophisticated algorithms or data structures. And because the standard does not belong to any particular vendor, programs written in COBOL are highly portable. The language can be used on a wide variety of hardware platforms and operating systems. And the rigid hierarchical structure restricts the definition of external references to the Environment Division, which simplifies platform changes in particular.[1]

A.1.3 Data Types

Standard COBOL provides the data types described in the table A.1.

A.2 Java

Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popu-

Data type	Sample declaration	Notes
Character	`PIC X(20)` `PIC A(4)9(5)X(7)`	Alphanumeric and alphabetic-only Single-byte character set (SBCS)
Edited character	`PIC X99BAXX`	Formatted and inserted characters

Numeric fixed-point binary	`PIC S999V99`
	`[USAGE]`
	`COMPUTATIONAL`
	or
	`BINARY`
		Binary 16, 32, or 64 bits (2, 4, or 8 bytes). Signed or unsigned. Conforming compilers limit the maximum value of variables based on the picture clause and not the number of bits reserved for storage.

Numeric fixed-point packed decimal	`PIC S999V99` `PACKED-DECIMAL`	1 to 18 decimal digits (1 to 10 bytes). Signed or unsigned
Numeric fixed-point zoned decimal	`PIC S999V99` `[USAGE DISPLAY]`	1 to 18 decimal digits (1 to 18 bytes) Signed or unsigned.Leading or trailing sign, overpunch or separate
Numeric floating-point	`PIC S9V999ES99`	Binary floating-point
Edited numeric	`PIC +Z,ZZ9.99` `PIC \$***,**9.99CR`	Formatted characters and digits

Group (record)	`01 CUST-NAME.`
	`  05 CUST-LAST PIC X(20).`
	`  05 CUST-FIRST PIC X(20).`
		Aggregated elements.

Table (array)	`OCCURS 12 TIMES`	Fixed-size array

Variable-length table)	`OCCURS 0 to 12 TIMES`
	`DEPENDING ON`
	`CUST-COUNT`
		Variable-sized array

Table A.1: COBOL Data Types

lar programming languages in use, particularly for client-server web applications, with a reported 10 million users. Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1991 and first released in 1995. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General

Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java and GNU Classpath.

A.2.1 Principles

There were five primary goals in the creation of the Java language:[19]

- It should be "simple, object-oriented and familiar"
- It should be "robust and secure"
- It should be "architecture-neutral and portable"
- It should execute with "high performance"
- It should be "interpreted, threaded, and dynamic"

A.3 Cobol to Java

In this section are presented the major traductions from the COBOL to Java adopted in the project, subdividing into three different parts. The first one is the one related to the structures. In the second are described the control instructions, like the conditional and the loop ones. In the last part are described some others instructions normally used when programming.

A.3.1 Structures

In this section are presented the conversions of the structures and the data types.

Date Cobol code:

```
01 dt-inizio
  05 FILLER REDEFINES dt-inizio.
    10 dt-ini-ssaa PIC 9(04)
    10 FILLER REDEFINES dt-ini-ssaa.
      15 dt-ini-ss PIC 9(02).
      15 dt-ini-aa PIC 9(02).
    10 dt-ini-mm PIC 9(02)
    10 dt-ini-gg PIC 9(02)
```

Java Code:

```
Calendar dt_inizio;
```

Numbers - Double Cobol code:

```
05 consistenza PIC S9(15)V9(3).
```

Java Code:

```
double consistenza;
```

In the program, in order to have the same number of decimals, there were used two statements. The first defines the format of the number (with the number of decimals wanted) and the second assigns the new format to the interested variable.

```
DecimalFormat formato = new DecimalFormat("#.#######");
importo_calc = Double.parseDouble(formato.format(importo_calc));
```

Numbers - Integer Cobol code:

```
05 tipo-calcolo PIC 9(03).
```

Java Code:

```
int tipo_calcolo;
```

Boolean Cobol code:

```
01 intero PIC X(01).
  05 si-intero VALUE 'S'.
  05 no-intero VALUE 'N'.
```

Java Code:

```
boolean intero;
```

String Cobol code:

```
01 divisa PIC X(03).
```

Java Code:

```
String divisa;
```

A.3.2 Control instructions

Conditional statements Cobol code:

```
IF <condition>[AND][OR][<condition>]
        statements
[ELSE]
[END-IF]
```

Java Code:

```
if (<condition>[& || <condition>]){
        statements
}[else][elseif <condition>]
```

Case Cobol code:

```
EVALUATE TRUE
    WHEN <condition>[OR][AND][<condition>]
   [WHEN OTHER]
END-EVALUATE
```

Java Code:

```
 switch (variable) {
     case <value>: statement  [{statements}];break;
                 .....
     default:  statement  [{statements}];break;
}
```

A.3.3 Statements

Function call Cobol code:

```
PERFORM function_name.
CALL function_name USING variable
```

The first is a normal function call, while the second is used importing a variable.
Java Code:

```
function_name();
function_name(variable);
```

Assignment Cobol code:

```
MOVE variable1 TO variable2.
```

Java Code:

```
variable2 = variable1;
```

Operation computation Cobol code:

```
COMPUTE variable3 [ROUNDED]= variable1 + [- * /] variable2... .
```

ROUNDED is used for replacing the real value by another one that is approximately equal but has a shorter, simpler, or more explicit representation. Java Code:

```
 variable3 = variable1 + [- * /] variable2..;
```

Cobol code:

```
ADD variable1 TO variable2.
```

Java Code:

```
 variable 2 = variable2 + variable1;
```

Cobol code:

```
DIVIDE variable1
    BY value
    GIVING  quotient
    REMAINDER remainder
END-DIVIDE.
```

Java Code:

```
 quotient = variable1 / value.
 remainder = variable1 % value.
```

Bibliography

[1] Cobol tutorial - introduction to cobol.

[2] Tco definition.

[3] Capra E. Agosta G. Green software. anche le applicazioni consumano energia. *Mondo Digitale*, (1), Marzo 2011.

[4] Agosta G. Bessi M. Capra E. Francalanci C. Automatic memoization for energy efficiency in financial application. *Sustainable Computing: Informatics and Systems*, 2012.

[5] F. Merlo A. Poli C. Francalanci, P. Giacomazzi. Green ict: tecniche di riduzione di costo e problemi aperti. *Politecnico di Milano*, Maggio 2009.

[6] E. Capra. Green ict: scenario tecnico e socio-economico. Technical report, -, 2008.

[7] Francalanci C. Capra E. Green it. sfide e opportunita. *Mondo Digitale*, 2008.

[8] Haase E. Dahm M., van Zyl J. *The bytecode engineering library (BCEL)*.

[9] Oxford English Dictionary. mainframe, n. on -line edition,.

[10] Dijkstra. *E.W. Dijkstra Archive: How do we tell truths that might hurt?* University of Texas at Austin, 2006.

[11] Katz D.M. Cios called clueless about extra costs. *CFO*, 2010.

[12] US Environmental Protection Agency (EPA). Energy conservation: Past & present projects: Green computing guide. Technical report, University of Colorado at Boulder, USA, 2005.

[13] Renzi F. Scenari evolutivi nei sistemi e nella tecnologia e loro impatti sui ced e sui loro consumi energetici. Technical report, IBM, 2007.

[14] Sciuto D. e Silvano C. Fornaciari W., Gubian P. Power estimation of embedded systems: A hardware/software codesign approach. *IEEE Trans. On VL-SI Systems*, 6(2):266–275, 1998.

[15] J. Garfunkel. *The COBOL 85 example book*. New York: Wiley, 1987.

[16] Brown E.G. Lee C. *Topic overview: Green it*. Forrester Reasearch, 2007.

[17] Levitin L.B. Margolus N. The maximum speed of dynamical evolution. *Phisica D120*, pages 188–195, 1998.

[18] Rui Oliveira. *The power of Cobol*. BookSurge Publishing, 2006.

[19] Oracle. Design goals of the java programming language, 1999.

[20] Norvig P. Techniques for automatic memoization with applications to context-free parsing. *Computer Linguistic*, 1991.

[21] Retrieved. Largest commercial database in winter corp. topten survey tops one hundred terabytes. Press release., 2008.

[22] Capra E. Formenti G. Francalanci C. Gallazzi S. The impact of mis software on it energy consumption. Technical report, European Conference of Information Systems, 2010.

[23] Murugesan S. Harnessing green it: Principles and practices. *IT professional*, 10(1):24–33, 2008.

[24] Jean Sammet. The early history of cobol. *ACM SIGPLAN Notices*, pages 121–161, January 2010.

[25] Restorik T. An inefficient truth. Technical report, Global Action Plan Report, www.globalactionplan.org.uk/research.aspx, 2007.

[26] Howard E. Tompkins. *In defense of teaching structured COBOL as Computer Science*. ACM SIGPLAN Notices, 1983.

[27] Robert Harper Umut A. Acar, Guy E. Blelloch. Selective memoization. Technical report, Proceeding of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New York, NY, USA, 2003.

[28] Ellis C.S. Vahdat A., Lebeck A. Every joule is precius: the case for revisiting operating system design for energy efficiency. *ACM SIGOPS European Workshop*, pages 31–36, 2000.

[29] Richard Wexelblat. *History of Programming Languages*. Boston: Academic Press, 1981.

[30] Verbrugge C. Xu H., Pickett C.J.F. Dynamic purity analysis for java programs. Technical report, Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, New York, NY, USA, 2007.