Comparative Analyses of Power Consumption in Arithmetic Algorithms Implementation

Alexandre Wagner C. Faria [1]

Leandro P. de Aguiar [1]

Daniel D. Lara [1]

Antonio A. F. Loureiro [1]

Abstract: Historically, energy management in computer science has been predominantly treated as an activity of hardware optimization. A great deal of the effort in this area is concentrated on component activation, deactivation, and resources scheduling in order to provide a reduction of total power consumption. This study focuses on the subject of power consumption from the software developer's point of view, using a reliable power measurement framework to validate the programming literature's premise that programming options – such as multiplication operations - are high consumers of power. Besides some elementary operations and authors' suggestions about alternatives for power consumption reduction on the programming stage, two well-known and widely applied algorithms for large number multiplication were compared: Karatsuba and Toom-Cook. The results lead to conclusions that might help a developer to choose, in some cases, between execution speed or power consumption reduction, or establish a maximum power consumption limit for the software execution.

1 Federal University of Minas Gerais, UFMG.
`{awcfaria,leandrop,daniels,loureiro @dcc.ufmg.br}`

1 Introduction

Energy management can be considered a critical aspect in mobile environments. Historically, most efforts from the research community have focused on developing technologies with more efficient batteries and hardware that consume less energy. However, some recent approaches have also suggested that the application of techniques to design more energy efficient devices should be focused on higher levels of abstraction. The optimization of the energy consumption originated by activities in different software components of a system is an equally (or even more) attractive alternative when the question is the energy management and economy in mobile environments.

Due to the expansion of mobile platforms and the subsequent reduction in price of such technological devices, new and more advanced applications have been used in different sectors of the economy where there is demand and space for mobility. Combined with high speed telecommunication networks, such applications are requiring more and more from the compact hardware in these devices, pushing developers to explore new options and achieve results never before thought possible.

Among most attractive software currently available are the advanced graphical applications such as global positioning systems. These programs contain a wide range of well-known practical applications that, like several others, require manipulation and rendering of "on demand" images procured directly from the Internet. Typically, such applications use high cost algorithms, not only in terms of CPU time, but also in terms of energy consumption, which can represent a critical factor for its viability.

In order to develop more efficient algorithms - in terms of energy consumption - it is necessary to understand, in advance, the possible situations that can arise during the development stage that can cause either a relevant increase or decrease in the energy consumption of the software. However, given the great complexity of variables - physical and logical - that can negatively or positively interfere in consumption, it is necessary to use cautious approaches that allow reliable comparisons and good conclusions during, or even before the development phase, while the developer is defining implementation strategies.

Measuring the energy consumption resulting from several behaviors of an algorithm is not a trivial task. There are many related issues to consider, since the isolation of variables in the experiment, until problems caused by the interference of the measurement process itself. Any of these situations can lead to inconclusive results.

At first sight, one could say that high cost in time algorithms would have a proportionately higher energy consumption cost. However, the literature shows that there are factors that can make a system to react differently in terms of energy consumption. Since such behaviors have not been precisely (and formally) described so that this knowledge can be clearly taught and used by developers, an interesting approach - in terms permitting conscious choices during development analysis and project stages - would be to compare different implementations of a software piece, showing not only the results in terms of memory size and processing time, but also concerning energy consumption. This is the objective of this study.

2 Measurement of Energy Consumption

In order to create a tool that is able to show that some algorithm consumes more or less energy, it is necessary to analyze some of the involved aspects related to the available measurement techniques. There are many problems, such as the interference of other operating system processes and some specificities of the computational platform which can affect the isolation of the variable "energy consumption" introducing noises that make results unreliable or inconclusive. There are different ways to approach this theme from a measurement point of view: from simulation and experimentation to mathematical models based on the variables related to the processing and access to computer resources (memory, I/O, etc.).

In order to find out here an efficient alternative for the stated objectives, it is important to evaluate some of the already created and validated solutions for measurement process in previous work, assessing the performance of each method.

2.1 Measurement Alternatives

Many studies have already been developed in an attempt to create ways of measuring the consumption associated to algorithms. The majority of these attempts, however, create only specific solutions for a very specific demand, which means such measurement tools cannot be used in other different situations. Moreover, because these works use hardware and software tools used in an excessively customized environment, it's very difficult to reproduce the original work, achieving similar results. This can even eliminate the possibility of validations through alternative methods not originally considered by authors.

The majority of used approaches to measure algorithms energy consumption make use of measurements executed directly in the hardware. In [8], Farkas etal. uses the approach of direct measurement using an electrical "shunt resistor" to evaluate the different impacts of coding for mobile applications on pocket PCs. In this technique, the hardware is modified by inserting a low resistance and high precision resistor, which is inserted in series circuit with the main power source circuit. Through a precise voltmeter, the voltage values over the resistor are continuously evaluated, while benchmarks of different Java applications with similar standard behavior are executed. Through Ohm´s Law ($R=V \times I$ and $P(w) = V \times I^2$), values for power consumption are inferred and related to each action executed in the software. Although this measurement system has a maximum error considering the real consumed power of only 0.002 Watts, the reproduction of this environment is totally dependent on the usage of equipments with exactly the same characteristics. In addition, it constitutes as an intrusive and even destructive approach for the hardware (considering that circuits for the majority of modern mobile equipments are constructed using SMDs – superficial mounting devices – electronic components which make it difficult to change the circuit). This type of approach can also be subject to interference from different software components besides the target components involved in the measurement, since there is no way of dissociating the consumption among the components.

In [7], Sinha *et. al.* suggested a methodology for estimating the energy consumption for software which is able to make predictions with a error margin of 3%. Regardless the relative good accuracy, the work has only been developed for processor models StrongARM SA-1 100 and Hitachi SH-4, and considering only some specific individual applications running on a specific operational system. Thus, they have not created any generic models or estimation tools that could be used in other situations.

Other similar approaches for measurement were already developed focusing specifically on consumption reduction for mobile devices. In general, they have in common the restriction of being conceived for specific hardware models, prohibiting the extension for different platforms where some specific hardware functions are not available. These works are typically based on circuit optimization and energy saving policies consisting of enabling and disabling different subsystems considering different demands for processing or using resources. This specific research area has grown faster compared to other areas of study. In this scope, initiatives such as LessWatts.org [6], have created cooperation between operating systems development, and the improvement of hardware functionalities to enable the use of more energy efficient algorithms. *PowerTop* [6] is an example of an optimization tool focused on reducing the consumption for Intel processors running Linux operating systems and has the objective of helping software developers for this platform not only to test, but also to achieve optimal behavior in terms of consumption.

The functioning principle of the *PowerTop* is to maximize energy use, through maintenance of the "low power" state, for the maximum amount of time. This is done in the moments that the availability of all resources, at the same time, is dispensable, and is accomplished through the switching of the so called C-states, which are possible modes for CPU operation during inactivity periods. There are basically five "states" (C0, C1, C2, C3 and C4), and the larger the number, the less energy is consumed by CPU and more time the processor will take to be able to return executing some given instructions. Therefore, the consumption reduction strategy practiced by the *PowerTop* is to keep the processor in states C3 and C4 for as long as possible, trying always to act without harming the system's general performance. This type of energy saving approach, in summary, does not allow us to certainly predict the behavior concerning the power consumption by an algorithm, since the responsibility for high or low consumption is given only by the improvements that this external control is able to practice. Therefore, although the tool is capable of providing a behavior analysis for algorithms, such results are superficial, thus disallowing an accurate determination of the possible power consumption sources.

In the next set of studies, we can find some authors that have used combined approaches – such as, hardware measurement and hardware resource usage – in order to generate power consumption estimates. In [2], Flinn J. et al. developed a tool to determine the consumption profile for applications that combine hardware instrumentation and Kernel measurements at the operating system's level in order to generate usage statistics for system activities. This method uses a digital multimeter connected to a second computer that generates traces and graphs of current consumption, and a modified operating system (BSD with a modified kernel) to offer system functions (system calls) to manipulate memory counters where the activity statistics are registered. At each execution interval of a given benchmark, the values of the Program Counter (PC), Process Identifier (PID), and other additional information - such as the moments in which the processor is manipulating an interruption - are registered simultaneously with the drawing of current values. In the next step, all this data is combined, and the final result uses a symbols table in order to map statistics to specific execution procedures inside the code. By doing so, this approach allows the power cost attribution not only for processes, but also for code regions. In spite of its efficiency (the authors suggest that, with the usage of this tool, it is possible to reduce the energy consumption of a given multimedia application by 46%), the *PowerScope* focuses on the consumption analysis related to all aspects and resources such as videos, disks and networks at the same time (the hardware measurement made with the multimeter is performed on the system´s main power source that provides energy for all the components). Moreover, for considering that the analysis of a single process might omit some critical information about energy consumption, the tool monitors all the processes simultaneously, pondering the consumption of each process throughout the activity that each one has generated. This creates a granularity level that may confuse the detailed analysis of consumption sources inside the algorithms, since other processes which the consumption can be basically summarized as I/O device may disproportionately influence the final results (in this case, the mentioned variables PC, PID, and IRQ are not able to distinguish the exact sources of activity). Additionally, similarly to other already presented methods, setting up the environment for the *PowerScope* is an equally complex and expensive task – since it asks for equipment with requirements that are too onerous, specially considering the practical point of view of software development environments. Other disadvantage it that it demands executing the software to be measured in a specific open source operating system, what not only interferes with the original system, but also make it impossible to use in some situations.

Despite the difficulties mentioned prior, the measurement philosophy used for the *PowerScope*, where hardware instrumentation is combined with resource accounting, is currently the most efficient method available. In [4], Chiyoung Seo ET al. have used the same approach to create a consumption measurement framework for distributed systems developed in Java, promising accuracy with a error margin of 5%. Instead of predicting the resource consumption with the running process like PowerScope does, this framework is based on the idea that consumption can be predicted through a resultant set of constituent software components. Through their research, the authors have determined the energy cost of a software component by calculating the sum of the computational cost – the cost associated with CPU use (memory, operations of IO, etc.) – the communication cost (calculated through the amount of bytes transmitted by the network), and the infrastructure cost (generated by the operating system and JVM overhead). The computational cost is determined for two sources: **Public interfaces**, due to the analyses been oriented to Java applications (the invocation effect of each interface is expressed in terms of different *bytecode*) and **Native methods** *e.g.*

System Calls. Also, considering that for different hardware platforms these costs may vary, the final result depends on successfully obtaining a previous consumption measurement of each one of these elements: consumption of each type of *bytecode*, consumption of each native method to be invoked, and consumption of the monitoring process itself. Although the efficiency of this measurement is adequate, once again, the cost of establishing this environment makes this framework impractical (in this study it was also necessary to use an open code version of Java - the Kaffe - so that the calls to methods could be accounted, which makes the reproduction of this environment even more complex).

2.2 Measurement Technique Based on Counters

Another interesting approach for measurement technique based on both instrumentation and counting is presented by [1], [3] Canturk I. and Martonosi M..This approach consists of using counting techniques based on hardware performance counters - available in the majority of modern processor models - that help with the identification of unitary events (unit-by-unit). According to Canturk, "cicle level" simulation alternatives not only consume a lot of time but also are vulnerable to lapses in accuracy. They are also complicated to use in thermal studies since it can require a long period of use until the processor reaches the correct thermal balance. The basic approach of this work is to use a multimeter for collecting total consumption data, and using estimates based on the counter's readings in order to produce the unit cost analysis. The total consumption is obtained through ampere meter pliers (which eliminate the necessity of an alteration of the power circuit) hardwired to a computer that generates graphs with a historical description of the consumption with timestamps. At the same time, a software developed as a kernel module known as LKM - which eliminates the necessity of a direct alteration in the kernel code of the operational system - is responsible for reading 24 metrics related to processor counters, relative to 22 physical components: bus control, L1 cache, L2 cache, L1 branch prediction unit (BPU), L2 BPU, instruction TLB & fetch, memory to order buffer, memory control, dates TLB, integer execution, floating point execution, integer to register file, floating point to register file, instruction to decoder, traces cache, microcode ROM, allocation, rename, instruction queue1, instruction queue2, schedule and retirement logic. The access rate to components are used as balance factor for distributing total consumption value using a scaling strategy based on micro-architectural and structural properties. Thus, through the accounting of metrics in real time, it is possible for a complex architecture processor (high-end), to measure and estimate the energy consumption with a configurable granularity depending on when and how the system calls of the LKM are made. In conclusion, compared with previous studies, this framework can be considered portable and detailed enough to allow accurate and reliable analyses.

3 Components used in consumption measurements

The purpose of this study does not aim to measure the exactly values of power consumption in Watts or Joules for a given algorithm, but just to conclude whether or not an algorithm's features can lead to more or less power consumption. This would lead to more conscious decisions from software engineers, in terms of energy usage, during the development of mobile applications, for instance. Such objective favors the use of the framework proposed in [1], since the counters based approach uses instrumented measures just for weighting total consumption values. This is only possible because all the resultant equations proposed in this study indicate a proportionality linear relationship, and different values for total consumption do not interfere with the final comparison between the algorithms.

The Canturk´s method is strongly linked to the processor features, which compel the utilization of a identical hardware configuration or at least similar in relation to the counters used in the work. In order to make use of the already built LKM, an Intel Pentium 4 1.4 GHz Willamette processor, the same model used by the author, had to be used. The final architecture consists of the following components:

- Client: with a similar processor used in the reference work, it is responsible for collecting counters values by trigger commands from the system.
- Server: receives collected values from the client and correlate them to time, recording instant values in data files and plotting graphs of power consumption.

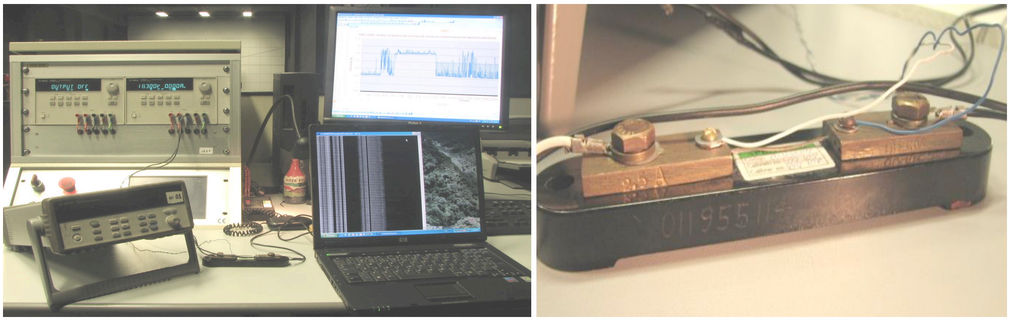
3.1 Changing and validating the model

In the original framework the application had the purpose of comparing consumption measurements, through instruments in real time, with the estimate values from the counters. In order to make possible applying the same framework in this paper, the dependence of the code related to the multimeter communication had to be eliminated. This was made through the identification and extinction of the LKM function, responsible for the data read in the serial port of the client and through the exclusion of all subsequent calls for graph plotting for energy consumption.

The original framework was validated through a series of tests that confirmed it´s accuracy. With the code alteration and the elimination of the dependent hardware, however, a new stage of validation was necessary since the alteration applied could cause distortion in the final behavior. The validation was chosen to use the method proposed in [8], through a precision shunt resistor. The shunt was inserted in series with the power source of a notebook where an algorithm with some simple arithmetic operations were running in loop.

The values of voltage were collected by a data logger model Agilent 34970A with a stabilized power source model Agilent E3632A. The shunt used transformation relation of 60 mV= 2.5A, and the final consumption results were analyzed through the consumption graphs generated by the data logger. Figure 1 shows a photo of the instruments used. The algorithm for validation was executed in two steps: the first step was comprised of only the validation environment; the second step was comprised of only the modified framework. For the validation, the differences in consumption estimations were compared and the results show that, even with the described alterations, the framework remained capable of estimating power consumption with a very small error rate.

Figure 1 –Validation Environment



4 Consumption sources in arithmetic algorithms

Some studies have already been conducted with the intent to identify the possible origins of major energy consumption. A lot of these studies concentrate mainly on hardware construction aspects, providing solid bases for the microprocessor industry to manufacture microprocessors with more economical consumption properties. Few efforts, however, concentrate on the rational use of these processors by creating more efficient programs. The major portion of these studies concentrates on determining the high level causes for power consumption and analyzing the entire process, instead of using different approaches such as: a) analysis of smaller code portions or low level instructions, b) conclusions about rationing of resources during inactivity periods, as show in the proposed work by PowerTop in [6].

The majority of implementation alternatives tried in this study to attempt to find a more economic energy usage are based on the book "Computer Arithmetic – Algorithms and Hardware Designs" [9], by Behrooz Parhami. This decision is basically justified by the explanations given by the author for explaining waste of time and energy. According to Parhami, in his chapter about low consumption arithmetic, there are three basic points that can be considered for consumption reduction: reduction of power waste, where the intention is to reduce the consumption by reducing the arithmetic complexity of the task executions; activity reduction, in which the consequences of an arithmetic operation are studied in order to reduce the final system´s activity; and, finally, the approach of transformations and replacement, in which some aspects of the arithmetic's hardware circuits – such as velocity and simplicity - are sacrificed despite the improvement of energy usage. The study conducted here concentrates on the exploration of the possibilities provided by the two firsts alternatives, once that the transformation and replacement approaches involves hardware modification.

4.1 Reduction of power waste

The most obvious method for reducing power waste is to reduce the number or complexity of arithmetic operations performed. Two multiplications consume more energy than one shift operation followed by one addition [9]. Thus, in the example suggest by Parhami, the expression (***a* x *b + a* x *c***) is worse, in terms of consumption, than computing the expression ***a* x *(b + c).*** In the same way, computing the expression (***16* x *a) - a*** is less wasteful than the expression (**15 *x* a**). Apparently, according to Parhami, although the necessity for optimizing expressions can be understood, even in cases where consumption is not the matter, there are situations in which the reduction of operations involves sacrificing speed. This raises concerns in the cost-benefit analysis, although less obviously, since the recovery of this damage can be done through a momentary acceleration of the clock and/or increasing the processor´s voltage, increasing again the total consumption of the operation.

4.2 Reduction of activity

The reduction of consumption through the reduction of activity is basically looking for actions that are executable using the request of one operation or a group of arithmetic operations. This can be done, for instance, by the reduction of the quantity of binaries changing state from 1 to 0 and 0 to 1. Thus, the signal change ("+" to "-" or "-" to "+") of one variable of the type " *signed* ", tends to consume much less power than changing one value code in a "complement of two", since changing the first one can be done by simply changing the value of one single bit. The following topics introduce some methods that can be utilized to reduce activity.

- Changing the coding information: one standard binary code, for one counter, for instance, involves in average two transitions or inversions, of bit by cycle. An approach that can reduce the consumption is to count according to the Gray code, in the binary representation of the next integer in the counter is different by only one number in relation to the one before.
- Operation reordering: an example of activity reduction by re-ordering can be given by the addition of one list of *n* numbers, in which the group of numbers is divided into two parts consisting of positive and negative values. Each list is added separately and the resulting sums of each group are added, leading to a reduction of activity.
- Pre-computation: in this step, a pre-processing stage is realized in order to provide an activity reduction on the arithmetic's operations involved. A typical example is the decomposition of a complex computation in two or more simple ones. This is done based on the value of one or more input variables.

5 Power consumption reduction by reducing the power waste

A large part of this study was focused on the identification of power reduction using techniques to reduce the power waste. The main goal was to identify possible situations that could be tested to determine what leads to less consumption. Fundamentally, the suggestion proposed consists in changing some fundamental arithmetic operations, by reducing the complexity or altering the number of operations, in order to obtain a better rate of consumption without sacrificing the performance. The following topics show the code alteration, suggested in [9], that can lead to less consumption.

- Multiplication: changing of the operation of multiplication by a group of shift-add leads to a reduction of complexity with a consequent reduction of consumption. This is this way because the multiplication instructions take more time to execute than a shift followed by an addition, even when the multiplication instruction is available in the hardware [9]. Multiplication by power of 2 can be realized using only shift operations and, finally multiplication of constant integers can be optimized using the techniques above for the reduction of activity. The reduction of operators can also be used as an alternative to reduce the power consumption.
- Division: similar to the multiplication, the reduction of operators and the use of shift operations in divisions, by power of 2, can be a valid approach for activity and consumption reduction. According to [9], division is also more expensive than multiplication. One way to replace the operation is the use of successive subtractions and reordering the operations. An example of activity reduction by reordering can be given by adding a list of *n* numbers and dividing the group into two parts consisting of positive and negative values. Then, each list is added separately and the resulting sums are added. This method tends to result in a reduction of activity.
- Subtraction: The subtraction can be done by a sequence of negation of the subtrahend and adding the result to the minuend, which, depending on some features present in the major part of hardwares available, can lead to an activity reduction.

6 Tests and Consumption Comparison

After surveying the possibilities for reducing energy consumption, a procedure to achieve implementations and measurements was developed aiming to identify the algorithms that would possibly generate the most significant results in terms of power efficiency. The following sequence of algorithms were chosen to be implemented and tested:

Implementation of several elementary additions and multiplications, validating the information from [9] in High-end architecture;

Implementation of several operations (Add, Sub, Mul, Div) using literals or variables, validating the information from [9] in a High-End architecture;

Implementation of some multiplications using complex operators and those same operations using simple shifts followed by additions that give the same result in order to validate the information from [9] in a High-End architecture;

Implementation and comparisons of the resultant consumption differences from the usage of shift operation for base 2 numbers, validating the information from [9] in a High-End architecture;

Implementations and comparisons of the resultant differences (long terms differences) implied by execution of more complex algorithms where there is an alternative option. For this step, two multiplication algorithms for large numbers were chosen: Karatsuba and Toom-Cook

6.1 Addition and Multiplication

According to [9], multiplications require more energy than addition operations, independent of hardware implementation. Therefore, the objective of this test was to compare the instantaneous consumption between a loop execution of additions and another of multiplications. Figure 2 shows the graphic generated by the framework where the operations of (a) addition and (b) multiplication were executed in loop. It's easy to see that, in fact, the instantaneous consumption initiated by multiplication is greater.

6.2 Use of Literals or Variables

One of the existing ways to reduce energy consumption by activity reduction, according to [9], is the reduction of the amount of used registers to execute an operation. The example used to illustrate and prove the veracity of this information is the use of a literal instead of a variable to represent some constant during the code writing. The compiled code in the first case will always require at least one less register and, consequently, there will be less activity during the execution.

To show the effects of energy reduction in this case, an algorithm was developed to run in loop with a number of operations using (a) a literal to represent a constant and (b) a variable to represent the constant. Figure 3 shows the comparison results. Although the total consumption using the variable was inferior (since using a literal takes more time to execute), it's possible to observe that the instantaneous consumption of using a literal is almost 20 to 30 percent lower.

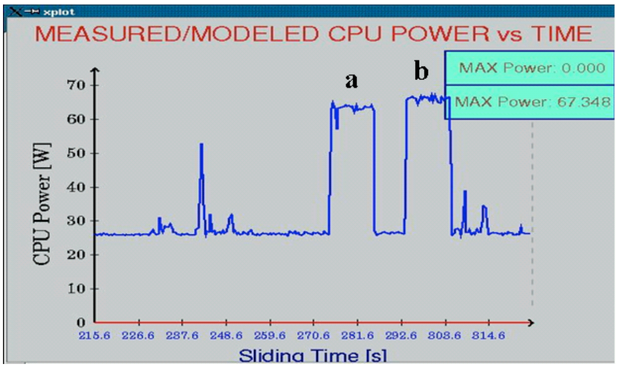


Figure 2 – Comparative of Instantaneous Energy Consumption of (a) Addition and (b) Multiplication Operations

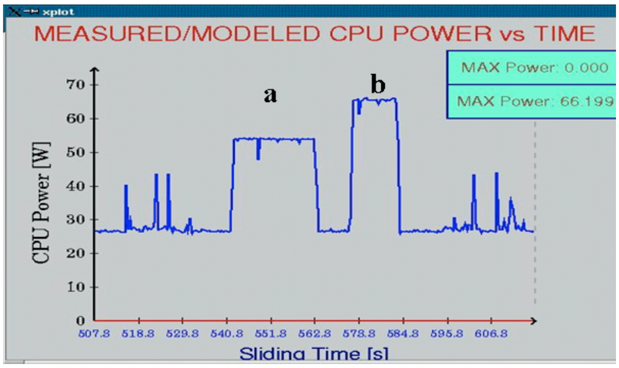


Figure 3 – Comparative of Energy Consumption of a Multiplication algorithm using (a) a literal to represent a constant and (b) a variable to represent the constant number

6.3 Complexity reduction of a multiplication operation.

One of the alternatives for power reduction is to reduce the complexity of the arithmetic operations to be executed. To validate this premise, an algorithm was developed to compare the differences in the consumption of the operations **(a)** a(b+c) and **(b)** (ab) + (bc).

The expected answer for this case is that, in function **(a),** where there is an elimination of an unnecessary multiplication, will require less energy consumption.

This algorithm was implemented and executed for the cases (a) and (b) in a loop to emphasize the consumption effects; the results are presented in Figure (4). Is easy observe that, in this case, the differences between (a) and (b) are not significant (or the differences are not large enough to comparatively measure the error, considering the framework) enough to conclude that (a) is, in fact, more efficient. The justification for this similar behavior can be found by comparing the machine code generated by GCC compiler for both cases.

In summary, there was an optimization by the compiler, such that, the generate code became identical, so there is not significant difference in the rate of consumption.

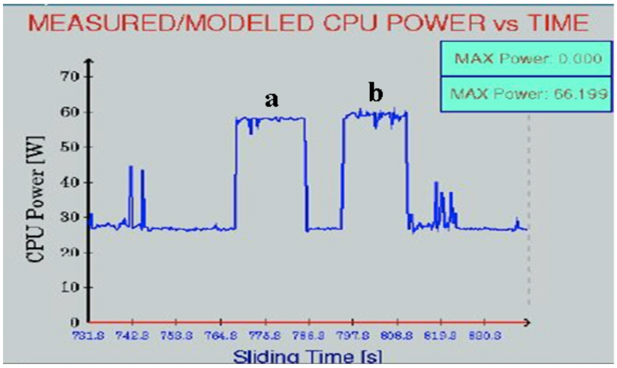


Figure 4. Comparative between algorithms of energy consumption: With optimization (a) and without optimization (b).

6.4 Complexity reduction replacing a multiplication by shift (numbers of base 2).

We already know that, typically, multiplication operations consume more than other basic operations. Another alternative suggested in [9] is to use the operation of shift to multiply numbers on the base 2.

In the implemented example, was used the suggestion of the author of [9]. In the first case (a) was traced the consumption graph of the operation *15a* and the second (b), the equivalent for this operation *(16a)-a*.

The expected effect was the reduction in the consumption of the operation **(a)**. Figure 5 compares the results.

In spite of the long period, the power consumption in operation (a) was lower than (b). However, the operation with shift (b) was executed in superior time (leaving the circuits busy for longer time and thus requiring more consumption). It is easy to observe that the instantaneous consumption of (a), in fact, followed the expected standard with a smaller value.

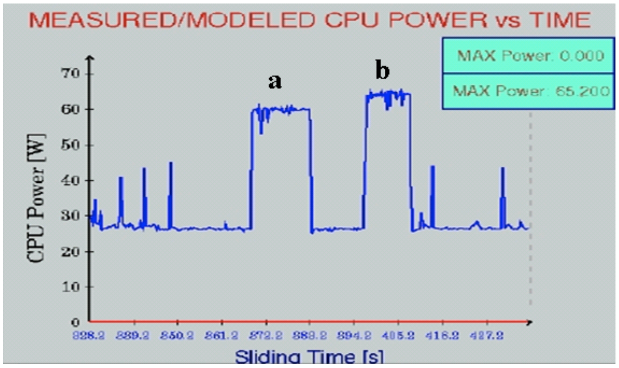


Figure 5. Comparative graphic of energy consumption of a multiplication operation: Optimized with shift (a), and without optimization (b).

6.5 Comparison of the multiplication algorithms: Karatsuba and Toom-Cook

The last chosen approach for the analysis of power consumption was the comparison of two algorithms designed to manage more complex numbers. For this phase, two algorithms for multiplication of large numbers were evaluated. The Karatsuba and the Toom-Cook. The Karatsuba was first implemented by Anatolli Alexevich Karatsuba in 1960 and intends to realize the multiplication of relatively large numbers.

The Toom, also know as Toom3 or Toom-Cook Multiplication, is an algorithm also designed to multiply large numbers with the cost of O (n^1,465). Typically, the Toom operates with larger numbers than the Karatsuba. The cut-off point at which to use one algorithm or the other still needs to be established. This study divided the tests into two phases: In the first phase, small numbers were used in order to test the efficiency of Karatsuba versus Toom-Cook. In the second phase, large numbers were chosen to test the efficiency of Toom-Cook versus Karatsuba.

The results can be observed in Figures 6 and 7. It is possible to see that, independently of the value of the numbers used, in all cases, the instantaneous consumption of the Toom-Cook algorithm was smaller. The differences in time of execution, however, are sufficient enough to create an approximate relation of straight proportionality between the consumption of time and consumption of energy. Although this work doesn't have showed a specific chart of this behavior, it was possibleto conclude that exists a threshold in which even with a superior cost in time, the Toom algorithm consumes less instantaneously.

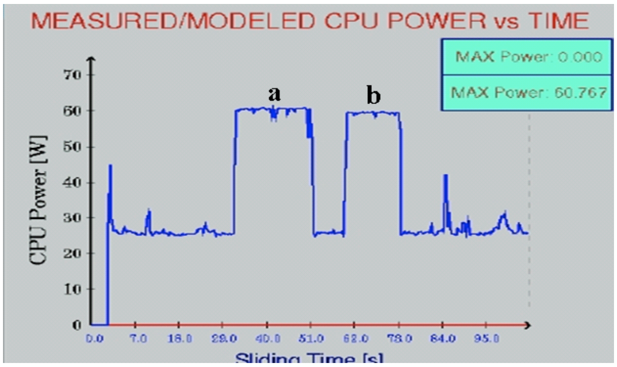


Figure 6. Comparative graph of energy consumption of algorithms: Karatsuba (a) and Toom (b), using large number

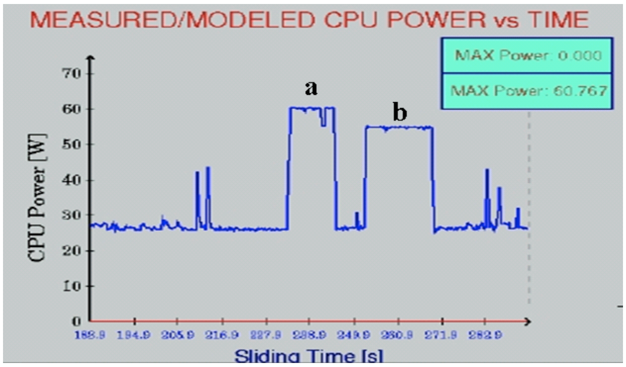


Figure 7. Comparative graph of energy consumption of algorithms: Karatsuba (a) and Toom (b), using relatively large numbers

7 Conclusions

In this paper was performed some arithmetic methods study to reduce power consumption in a high-end computer architecture. In order to evaluate these methods,it was used a framework proposed by Canturk [1]. Laboratory tests were done to validate the framework to make sure it was suitable for this work. The results confirm the potential of changing som arithmetic algorithms, aiming to reduce power consumption. Among the results, the following points can be emphasized:

- The framework, suggested by Canturk, for estimating the energy consumption of algorithms was effective in the measurements, even with the alterations made in this study to eliminate the dependence of some other measuring devices, originally used.

- The instantaneous energy consumption of the algorithm Toom-Cook is always better than the algorithm Karatsuba for any level of number multiplication.
- It is possible to determinate a threshold for which the Toom algorithm consumes less energy, even when considering the superior execution time when compared to Karatsuba.
- In the multiplication realized by *shift-add* in replacement to the normal operation there wasn't reduction in the energy consumption. This happened because, in this case, there was a greater execution time.
- There was a clear reduction in the energy consumption when a constant literal, instead of a constant stored variable, was used in multiplication operations. This information can be used in projects that develop software to limit the maximum power consumption.
- The suggestions of consumption in [9], except the reduction of complexity in the multiplication, were all validated. In the major part of cases, the alternatives for less consumption lead to more execution time. The Results indicate that the differences in the reduction of consumption can be greater than 20%.

These conclusions can be used for the limitation of maximum energy consumption of current built-in system circuits that use software behavior. These conclusions, can also be used to design a system that use batteries with a maximum capacity of peak current,. The reduction of instantaneous consumption, can determine size reduction of devices, once the robustness of the integrated circuits that compose it can be determined by the thickness of the conductors.

8 Future works

Although not all possibilities for power reduction have been approached, this study sufficiently affirmed - with a high level of certainty – that it is possible to implement changes during the development phase of software that can lead to reduced energy consumption by the hardware. In some cases, there can exist a trade-off between time and consumption cost. Thus, we considered important the following extensions for this study:

1. The implementation of others alternatives for the reduction of power consumption that still have not been tested, but suggested by the references.
2. The analyses of the overhead, that exists when there is an eventual choice by software between one or other algorithm, as in the example of Karatsuba and Toom, which is very common in practice.
3. The development of elaborated guidelines for developers that show which decisions of codification can lead to more or less energy consumption.
4. The construction of a tool that uses such recommendations, combined with a group of preferences of the programmer about the choice or aspect more important: consumption

or execution time. Such a tool could be capable of reorganizing the code to prioritize that in agreement with a power consumption knowledge base.

9 References

[1] C. Isci and M. Martonosi “Runtime Power Monitoring in High-End Processors. Methodology and Empirical Data”, International Symposium on Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM

[2] Jason Flinn, M. Satyanarayanan “PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications”, 1999 IEEE Workshop on Mobile Computing Systems and Applications

[3] C. Isci. and M. Martonosi, “Identifying program power phase behavior using power vectors.”, 2003. WWC-6. 2003 IEEE International Workshop on Workload Characterization

[4] Chiyoung Seo, Sam Malek, Nenad Medvidovic, “An energy consumption framework for distributed java-based systems”, 2007, Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering

[5] D. Ramanathan; S.Irani; R.Gupta, “Latency effects of system level power management algorithms Computer Aided Design”, 2000, Proceedings of the IEEE/ACM international conference on Computer-aided design

[6] PowerTop. Available in: http://www.lesswatts.org/projects/powertop/ (Access in 30/04/2008)

[7] A. Sinha, et al. “JouleTrack – A Web Based Tool for Software Energy Profiling”, In Proceedings of Design Automation Conference, 2001. Proceedings DAC.

[8] Keith I. Farkas, Jason Flinn, Godmar Back, Dirk Grunwald, Jennifer M. Anderson, 2000 “Quantifying the Energy Consumption of a Pocket Computer and a Java Virtual Machine”.

[9] Parhami, Behrooz. “Computer Arithmetic - Algorithms and Hardware Designs”, Univ. of California, Santa Barbara. Orford University Press.