IMPACT OF GCC OPTIMIZATION LEVELS IN ENERGY CONSUMPTION DURING PROGRAM EXECUTION

David BRANCO, Pedro Rangel HENRIQUES
*Algoritmi Research Center - Department of Informatics, University of Minho - Gualtar - 4710-057, Braga, Portugal,
tel. (+351) 253 604 470, E-mails: {davidbranco88,pedrorangelhenriques}@gmail.com

ABSTRACT

In this paper we report an experiment conducted to measure and compare the resources consumed by a program during execution, and then we discuss the results obtained when compiled without any optimization and compiled with different levels of optimization, in order to understand the relationship between a program's energy consumption and its optimization level. Namely, we will analyze the optimizations performed by the GNU Compiler Collection (GCC) on C, C++, Objective-C and GO programs. Java was also considered, but thereafter abandoned. In the paper we describe the experimental setup and the method followed in the study carried on to get a deeper knowledge about the factors that actually impact on the energy consumption of a given program. With the study described, the main lesson extracted is that the optimizations that generate a faster code are the more convenient in terms of green computing because they also decreases the energy consumed. The work report was developed in the context of a research project aimed at studying patterns for energy consumption at runtime in order to transform the original software into executable programs that consume less energy; in this context we are specially interested to look for ways to get this improvement at the code generation phase.

Keywords: *Energy Consumption, Optimization, GCC, Test Cases, Imperative Languages, Compile Generator*

1. INTRODUCTION

Currently we live in a period in which technology evolves very quickly and the number of those who use it, causing the associated energy consumption, reaches very high values in financial and environmental terms. So with a strong concern and an increasing need to reduce energy consumption in all the information and communication infrastructures (ICTs), emerged Green Computing which aims precisely at computers and related resources more efficiently while maintaining or increasing overall performance [1] [2].

Being microprocessors - or commonly, the CPUs or just processors - the heart and brain of any computer (and of a huge number of devices used on a daily basis), naturally they are one of the components with great impact on energy consumption. According to the information from Intel Labs in 2008 the processors are even the largest consumers of energy inside servers with values between 45W to 200W per multi-core CPU (depending on the type of server and workload) [3] [4]. They had a very important role in world's development and will continue to affect the life at several levels of their population. For all those reasons, the optimization of aspects related with microprocessors is a crucial way to combat excessive energetic consumption of IT industry.

The code optimizations performed by a compiler are a great way to get more performance without modifying any hardware or software component. With a just a few adjustments in the compilation arguments and these improvements are obtained immediately. While this approach has mainly the aim of reducing the execution time or required space by a program, in this article we will also see the potential that may have in reducing energy consumption [5] [6].

In this paper, we address the energy impact that running optimized compiled code has in the energy consumption of the CPU. In particular, the study of programs compiled by GCC with optimization flags for a target machine based on an Intel CPU. Although the CPU is the most important computational resource of this study it was also examined the impact in memory and GPU in order to perform a more complete analysis.

The study here described is a part of a master's thesis undergoing in the context of the project GreenSSCM[1] which intends to optimize energy consumption via software, transforming high level software code into more suitable software code. These performs the same tasks but consumes less energy.

The chosen programming languages are C and C++ according to the general decision taken in the context of GreenSSCM. In addition, they are also nowadays two of the programming languages more used [7].

For C/C++ compiler was chosen GCC[2](v.5.3.0) because it is a robust option, well documented, the most widely used and this allows different code optimization levels.

Taking into consideration that GCC is an integrated distribution of compilers for several major programming languages [8], two more programming languages have been added to this study. In this way we could anayze in more detail the behavior of GCC and its optimization procedures to get a deeper knowledge about the impact of optimizations on energy consumption for a larger and more diverse set of programming environments. After a preliminary analysis, we decided to consider Objective-C and Go because they are also widely used today [7] and still subject of improvements in the most recent GCC versions (unlike Java, for example, for which there is not any relevant update since version 4.5 of April 2010 [9] [10]). Although Apple has

[1]Green Software for Space Control Mission, a research project involving the company VisionSpace Technologies and Universidade do Minho running under Compete and supported by QREN.
website: http://visionspace.dnsdynamic.com/GreenSSCM

[2]https://gcc.gnu.org/

decided to completely replace the use of GCC by LLVM[3] and Clang[4] in their systems to compile Objective-C programs, actually Objective-C was not "forgotten" by GCC and, albeit some stagnation in support and improvement, it makes sense to study the impact of GCC compilation on Objective-C runtime performance [11].

After choosing the source languages to compile with GCC, the last fundamental decision to complete the experiment setup is concerned with the choice of the target machine. The assembly code that will be generated, and the optimization effects/impacts that can be reached, depend, of course, on the machine selected. Accounting for that Intel is the manufacturer with the largest market share in recent years (57% in 2012, 65% in 2013), and is currently present in over 80 percent of the computers sold worldwide, it is imperative to use an Intel microprocessor for more relevant results [12] [13] [14].

This paper is organized as follows. In section 2 it is referred a similar study in our context for embedded systems. In section 3 the main elements are described to carry out this experimental study. In section 4, the GCC optimizations are specified and explained the measurement process. In section 5 are shown and analyzed the results obtained. Lastly, in section 6 a Conclusion is devised focusing the main considerations of the present study.

2. RELATED WORK

Despite the fact that energy consumption subject is only a research topic in recent years (largely due to the massive widespread of quite advanced wireless and mobile devices), already at the beginning of this century appeared some projects considering compilers precisely as a way to combat it. In 2001 the standard optimization levels -O1 to -O4 were evaluated to understand the effect of a few individual optimization of DEC Alpha's cc compiler on power and energy consumption of the processor. The authors concluded that when the optimizations decrement the number of instructions to be executed, also the energy consumption is reduced [15]. In the same year, another study was conducted to explore the effect of the compilers for existing processor architectures addressing the same problem. They concluded that the compiler optimizations has enough potential to achieve some reduction in energy consumption, but it would be necessary to expose more innovating micro-architectural features to the compilers, in order to obtain substantial gains in energy saving [16].

There are also some studies in which algorithms are designed to select combinations of compiler optimization flags that, for a given input program, generate a machine code with a better performance at runtime (without taking into account the energy issues or flags that have emerged in more recent versions of GCC) [17] [18] [5]. If there is indeed a relationship between the optimizations applied by the GCC and the energy consumption of programs, such algorithms (or variants thereof) may be very useful. This investigation will be left for future work.

Considerations for energy efficiency are especially relevant at the level of embedded platforms. In [5] they use GCC, 10 benchmarks and 5 different embedded platforms to analyze the energy consumption of a large number of compile options. Through hardware power measurements and some case studies explore various hypotheses and conclude, among other many things, the execution time and energy consumption are correlated in most general cases.

Until this day there is very little work that explores widely the impact that the various compilation options provided by compilers (including GCC) has on the energy consumption of the software that use them [5].

3. EXPERIMENTAL SETUP

The three main elements to carry out this experimental study are: a platform for taking measurements (a laptop); the software that makes measurements; and also the software packages that will be measured.

3.1. Testing Platform

The study was accomplished on a laptop Asus N56JN-DM127H, running under Linux. The hardware/software resources most relevant characteristics for the required analysis are: Arch Linux 64-bit (Linux Kernel 4.4.5-1); Intel® Core i7-4710HQ up to 3.5 GHz, Haswell Family; 8 GB DDR3L 1600MHz; and NVIDIA® GeForce® GT 840M, 2GB DDR3 VRAM.

3.2. Measurement Software

The energy measurement necessary to make the comparative study was performed using Running Average Power Limit[5](RAPL) interface. RAPL allows, among other features to read the Machine-Specific Registers containing information about the energy consumed by the CPU, RAM and GPU during a given period of time [19] [20].

Performance Application Programming Interface[6](PAPI) and the Perf[7] were also considered as alternatives to RAPL to take the desired measures. However after analysing pros and cons of each one, these options were discarded. On one hand PAPI is an event-driven tool what makes its use inadequate in our context. On the other hand, Perf only allows to obtain information about the CPU and we were interested in investigating the other possible sources of consumption, the memory and the GPU.

RAPL was used through an extension developed by the team of this study to an existing tool[8], that simply is reading the referred register. This extension keeps all other features previously present, such as setting the number of the CPU cores that will be measured, and adds the ability to measure the consumption of a certain operation and also

[3]http://llvm.org/
[4]http://clang.llvm.org
[5]https://01.org/blogs/tlcounts/2014/running-average-power-limit--rapl
[6]http://icl.cs.utk.edu/papi/
[7]https://perf.wiki.kernel.org/index.php/Main_Page
[8]https://github.com/deater/uarch-configure/tree/master/rapl-read

the time spent for its completion. For this study the operation above referred is the compilation and/or the execution of a program, being only necessary to inform the path to the respective makefile or the executable. This extension can also be used in other areas of study, because it allows to measure the compilation of a program in any language that contains their dependencies expressed in a makefile or even any other executable. The distinct features that this tool provides are managed through a mechanism of flags that are passed as arguments when the tool is invoked.

3.3. Measured Software

In the experiment described in this paper we have analyzed 12 programs that differ in some aspects such as: programming language, main objective, code complexity, external dependencies and also compile and execution time. Despite these differences, were all chosen according to the following criteria:

- Open source code, allowing to analyze in more detail the complexity and how the objectives are achieved;
- Running under Linux environment;
- Coded in one of the programming languages chosen in the context of GreenSSCM project, C or C++, Objective-C or Go;
- Not interactive, this is independent of user interaction during execution, thus avoiding waiting for input that would have interference in the measured values and also allows automate this part of the process;
- No graphical interface;
- Total execution time less than 60s, to prevent possible overflows.

Although the many particularities of the chosen programs, there are some characteristics common to all of them that can be used in order to be possible to perform a comparative study, without knowing in detail the code of each studied programs.

In Table 1 we characterize the selected programs regarding some of their features. The parameters considered are: (PL)programming language; the (IF)input and (OF)output files; (CC)code complexity (low, medium or high); and the (ED)amount of external dependencies (none, few, several or lot). Average compile and execution times(ACT, AET) are also included just to figure out a quantitative description of their size/complexity.

Program	PL	IF	OF	CC	ED	ACT (s)	AET (s)
MMC	C	None	None	Low	None	0.17	26.16
Grades	C (Flex and Yacc)	Txt	Html	Low	Few	0.23	32.84
Bzip	C	Wav	Bz2	Medium	None	1.47	23.41
Bzip2	C	Wav	Bz2	Medium	Several	1.97	23.38
Oggenc	C	Wav	Ogg	High	None	3.83	22.91
Pbrt	C++	Pbrt	Exr	High	Lot	43.08	19.42
Matmul	Go	None	None	Low	None	0.13	15.05
PGo	Go	None	None	Low	None	0.15	13.06
Sudoku	Go	Txt	None	Medium	None	0.24	24.43
Matmulobjc	Obj-C	None	None	Low	None	0.19	3.88
Miscellany	Obj-C	None	None	Medium	None	0.40	9.47
Sorting	Obj-C	None	None	Medium	Few	0.26	35.62

Table 1 Some features of the measured programs.

A brief description of each one of the subject programs follows:

MMC: Multiplication of matrices with size 1024x1024 using 6 different methods.

Grades: Generates the final grades of the students in a course from their marks.

Bzip and Bzip2: File compression tools (replacing the input file).

Oggenc: Perform file format conversion (creating a new file).

Pbrt: Executes the rendering of images using ray tracing.

Matmul: Multiplication of two matrices with size 2000x2000.

PGo: Modular random level generator (roguelike type).

Sudoku: Solves 20 extremely hard Sudokus repeated 1024 times;

Matmulobjc: Multiplication of matrices with size 800x800 using 6 different methods.

Miscellany: Collection (more than 1000 lines) of practical exercises.

Sorting: Applies 6 of the best known sorting algorithm to an array of 8000 positions.

4. METHODOLOGY

After defining the software and hardware environment for the study, and the set of programs to test, in this section we will describe the main decisions taken in what concerns the parameterization of GCC in order to configure it for fair comparisons. We also define the strategies followed to get the measures and to obtain significant statistical results.

4.1. Optimizations Flags

The GCC compiler has hundreds of flags related to the optimization of the generated code specific to a given machine. Due to the specificity of each of the flags and the fact that sometimes they are mutually exclusive, this study will only focus on the several optimization levels specified by the compiler switch -O. There are 7 different levels of optimization and each contains a number of individual flags that are enabled or disabled (depending of the level). Below are described the referred levels with some more detail [21] [22].

-O0: Default level (disables all optimization flags). Reduce compilation time and make debugging produce the expected results.

-O1: Basic optimization level (enables up to 39 flags). Reduce code size and execution time without taking much compilation time.

-O2: Recommended optimization level (enables up to 73 flags). Enables all existing in -O1 and the remaining that do not include a space-speed tradeoff, increasing both compilation time and the performance of the generated code.

-Os: Reduced code size (enables up to 65 flags). Enables all -O2 options that do not increase the size of the generated code. Useful for target machines with limited disk storage space and/or CPUs with small cache sizes (with dramatic improvement of performance).

-O3: Highest level of optimization (enables up to 82 flags). Enables all existing in -O2 and further some very heavy in both time compile and memory usage terms. It is not guaranteed that the code generated is better in terms of performance than the previous set.

-Ofast: Disregard strict standards compliance optimization (enables up to 85 flags). Enables all -O3 options and more 3 that are not valid for all standard-compliant programs.

-Og: Optimize debugging experience (enables up to 74 flags). Offer a good debugging experience enabling all optimizations that do not interfere with debugging and reasonable level of optimization while maintaining fast compilation.

After a preliminary test phase, we decided not taking into consideration the level -Og for this study because this level performs optimizations which do not contribute to an energy reduction compared to the other 6 discussed.

4.2. Measurement Process

After having selected the programs to be studied and defined the set of optimizations to be compared, it was mandatory to setup the measurement process to apply to all programs in order to cover all the desired cases. This process can be described by the following steps:

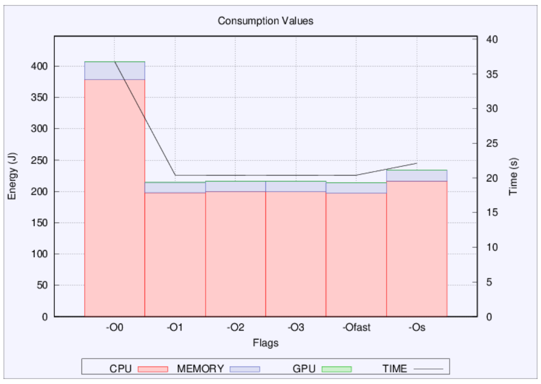
1. Choose a program and its Makefile;
2. Choose the desired optimization level;
3. Execute 100 times the measuring tool for the program and the chosen optimization level;
4. Process the output generated by each invocation of the measuring tool:
   (a) Get the energy consumption and time values;
   (b) Ignore the 10 highest and lowest values;
   (c) Compute the average of the remaining 80 values;
   (d) Generate a table and plot with the results in an HTML page.
5. Repeat step 2, 3 and 4 for all 6 levels of optimization;
6. Repeat step 1 to 5 for all 12 programs.

All measurements relating to a program were performed uninterruptedly while avoiding fluctuations related to the testing platform (e.g. pre-loading of data, memory heating, etc.). During the measurement process all executions of the intended programs were forced to run on just one core of the CPU, through the tool flag -n, to ignore efficiency issues related to the parallelization that some programs may allow. The processing of the output (referred in item 4 above) was done using essentially PERL[9](for parsing the results of each operation) and GNUPlot[10](to generate plots of each program).

5. DISCUSSION OF RESULTS

To illustrate the results obtained, five examples of charts are displayed: three running C/C++ (Figure 1, Figure 2 and Figure 3), one running Go (Figure 4) and one more example running Objective-C (Figure 5).

The remaining charts, as well as the HTML pages, can be found in the online repository[11] of this work.



[9]https://www.perl.org/
[10]http://www.gnuplot.info/
[11]https://github.com/david-branco/gcc-optimization-energy-article-extended

Fig. 1 Results of Bzip measurements (C program).

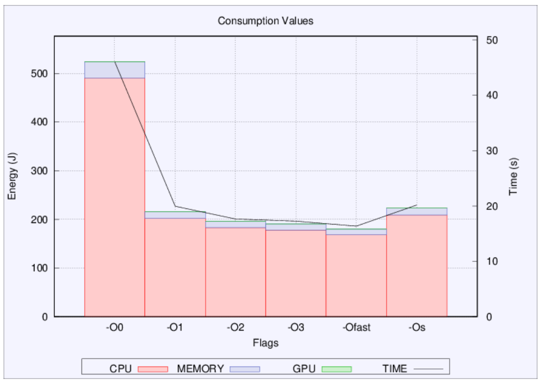


Fig. 2 Results of Oggenc measurements (C program).

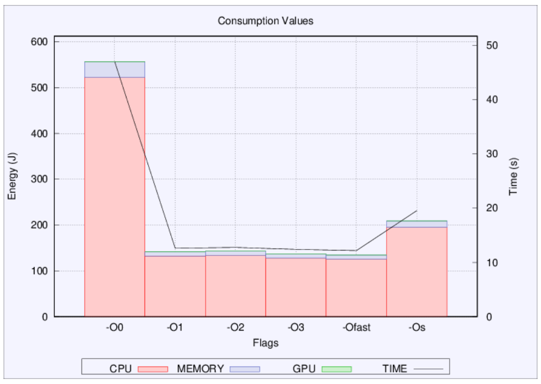


Fig. 3 Results of Pbrt measurements (C++ program).

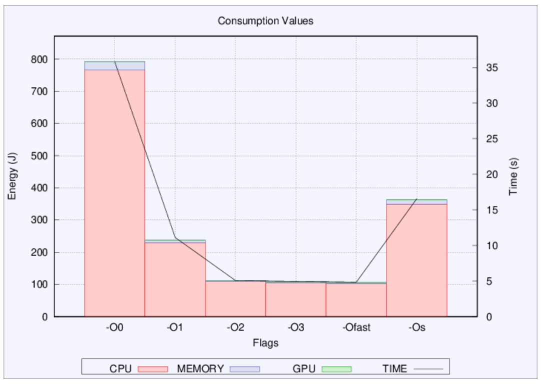


Fig. 4 Results of PGo measurements (Go program).

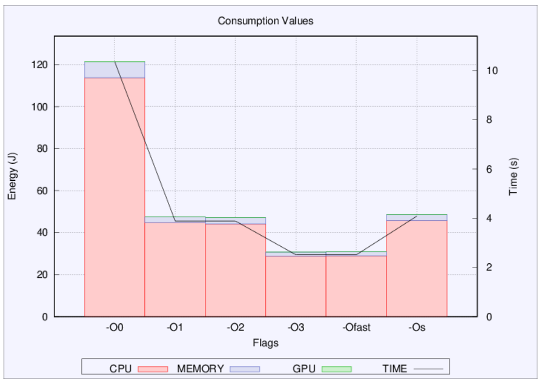


Fig. 5 Results of Matmulobjc measurements (Objective-C program).

In all analyzed charts it was found that, due to the classes of the programs selected, the energy consumed by the GPU is reduced. It is clear that energy and time consumptions are undoubtedly lower when selected an optimization level different from the default, and that the optimization is greater for more complex source codes (in the case of Pbrt there was a reduction of almost 76%). For low complexity programs it was confirmed that none of these levels is much higher or lower than the remaining, with only a minimal difference (in the extreme case may not even exist) among some of them because of individual optimizations that each enables/disables.

Level / Program	-O0	-O1	-O2	-O3	-Ofast	-Os
MMC	34.361	24.552	24.402	23.649	24.265	25.740
Grades	40.038	32.378	31.239	31.226	31.060	31.091
Bzip	36.755	20.408	20.396	20.361	20.399	22.115
Bzip2	36.755	20.493	20.366	20.477	20.427	21.773
Oggenc	46.068	19.958	17.648	17.256	16.359	20.177
Pbrt	47.008	12.614	12.761	12.413	12.175	19.547

Table 2 Execution times of C/C++ programs in seconds by optimization level.

Level / Program	-O0	-O1	-O2	-O3	-Ofast	-Os
MMC	354.934	237.568	239.599	240.192	238.445	227.809
Grades	352.228	257.912	246.786	248.958	245.701	249.664
Bzip	406.914	214.438	216.539	215.882	213.496	234.162
Bzip2	402.516	212.324	212.327	215.639	213.947	229.156
Oggenc	523.992	216.264	195.844	190.472	180.038	213.51
Pbrt	556.222	141.461	142.815	136.84	134.378	209.164

Table 3 Selected C/C++ programs and their energy consumption (CPU and memory) in Joules by optimization level.

Considering all selected programs and optimization levels which are not the default, analyzing the execution time (Tables 2, 4 and 6), memory and CPU energy consumption (Tables 3, 5 and 7), and ignoring minimal differences of values between levels, it appears that in most cases the -Ofast level is the most efficient unlike -O1 and -Os options. It is also perceptible that, although there is no much difference between the -O2 and -O3 levels, -O3 is slightly more efficient especially when the program complexity increases. Although regarding the results presented, -Os was not one of the best levels in the analyzed parameters, it is know that -Os has potential for significant improvement in some particular cases, for example where their optimizations are capable to fit the code in the cache.

Level / Program	-O0	-O1	-O2	-O3	-Ofast	-Os
Matmul	22.247	21.396	8.420	8.429	8.434	21.347
PGo	35.803	11.117	5.066	4.961	4.853	16.567
Sudoku	36.586	24.557	14.951	13.156	13.162	26.169

Table 4 Execution times of Go programs in seconds by optimization level.

Level / Program	-O0	-O1	-O2	-O3	-Ofast	-Os
Matmul	290.775	246.092	105.301	105.514	105.541	231.753
PGo	791.741	237.263	112.529	108.662	106.343	362.635
Sudoku	414.359	265.79	164.581	145.177	146.065	286.753

Table 5 Selected Go programs and their energy consumption (CPU and memory) in Joules by optimization level.

Level / Program	-O0	-O1	-O2	-O3	-Ofast	-Os
Matmulobjc	10.354	3.885	3.882	2.511	2.513	4.085
Miscellany	22.442	13.681	9.038	10.447	10.448	13.665
Sorting	37.623	35.692	35.663	35.787	35.748	36.031

Table 6 Execution times of Objective-C programs in seconds by optimization level.

Level / Program	-O0	-O1	-O2	-O3	-Ofast	-Os
Matmulobjc	121.262	47.532	47.053	30.714	30.867	48.664
Miscellany	248.844	129.711	95.549	107.859	107.713	129.914
Sorting	456.098	435.695	435.908	439.173	433.084	438.992

Table 7 Selected Objective-C programs and their energy consumption (CPU and memory) in Joules by optimization level.

One of the main objectives of this work was to determine if programs optimized at compilation time also have an optimized energy consumption during execution. Although the data obtained clearly demonstrate a great optimization of energy consumption, when selected optimization levels which are not the default, however it is also noticeable that the execution time of programs previously optimized also decreases dramatically (generally for shorter times was obtained lower consumption).

Thus, it is not possible to conclude with certainty GCC's strategies on the matter. We saw that in all cases energy consumption is directly related to execution time. In fact analyzing all graphs, their depict the data collected along the experiment, we can say that the timeline follows the trend of the columns with the consumption of each component by optimization flag.

6. CONCLUSION

In this paper we defined the objectives of a master's thesis in the context of GreenSSCM research, and described an experimental test aimed at studying the impact of GCC optimization on the energy consumed by the compiled C, C++, Go and Objective-C programs at runtime. We concluded that energy decreases as faster is the code and so we can affirm that GCC optimizations techniques have a positive impact in favor of green computing concerns. To software developers, this conclusion means that they do not need an extra effort when/if they decide to have their code more efficient concerning both execution time and energy consumption.

The framework developed to perform the necessary measurements is also an important result of this work. It was developed in a rather generic and comprehensive manner, allowing the analysis of several operations and programming languages, in order to be possible its usage in other GreenSSCM projects. Also, the overall output produced by the measurements performed in this study are important because they can be used as a good workbench for other green oriented research.

The obtained results are in line with what was the initial intuition of GreenSSCM team members and also the conclusions already reported in [5] for embedded systems. After finish this experimental work, we were aware that the time-energy relationship was already described and explained in the Technical Report [23], corroborating our experimental findings.

However a lot of work remains to be done to understand the strategies followed in those optimization algorithms in order to understand if there is still room for more reduction.

Study the impact of optimization the parallel of a program, analyze more programming languages that GCC can handle, or analyze programs that run on GPU are some of the possible research directions aiming at directly complement the work here reported.

Concluded the study and taking into account the results obtained, namely the fact that there are some improvements in energy consumption produced by compiler optimizations (albeit indirectly), some other working hypotheses rise up to proceed within this scope. A first one is to study the application of the algorithms referred in Section 2 in order to obtain specific sets of flags that can further reduce power consumption. Another one is to look for special types of machine instructions that can be chosen by the compiler during the code generation/optimization phase regarding the energy consumption. This is, we intend to analyze the information provided by the current machines' Instruction Sets to verify if the energy consumption cost is provided (explicitly available) to be considered by the compiler's optimization algorithms.

ACKNOWLEDGEMENT

This work is co-funded by the North Portugal Regional Operational Programme, under the National Strategic Reference Framework (NSFR), through the European Regional Development Fund (ERDF), within project GreenSSCM - NORTE-07-02-FEDER-038973.

We would like to thank all previous reviewers of this paper for their advices and guidelines for future work.

REFERENCES

[1] GUELZIM, TARIK AND OBAIDAT, MOHAMMAD S.: *Handbook of Green Information and Communication Systems, Chapter 8, pp. 209–227. Academic Press, 2013. ISBN 978-0-12-415844-3*

[2] HARMON, R. R. AND AUSEKLIS, N.: Sustainable IT services: Assessing the impact of green computing practices, pp. 1707–1717. Management of Engineering Technology, PICMET 2009, 2009.

[3] MINAS, LAURI AND ELLISON, BRAD: *Energy efficiency for information technology: How to reduce power consumption in servers and data centers. Intel Press, 2009. ISBN -1-934053-20-1*

[4] ELLISON, BRAD: The Problem of Power Consumption in Servers, pp. 1–17. Energy Efficiency for Information Technology, 2009.

[5] PALLISTER, J. AND HOLLIS, S. J. AND BENNETT, J.: *Identifying compiler options to minimize energy consumption for embedded platforms*, The Computer Journal **58**, No. 1 (2013) 95–109

[6] KREMER, ULRICH: *Low power/energy compiler optimizations. Low-Power Electronics Design, CRC Press, 2005.*

[7] Tiobe - the software quality company. http://www.tiobe.com/tiobe_index

[8] Using the GNU Compiler Collection (GCC). https://gcc.gnu.org/onlinedocs/gcc/G_002b_002b-and-GCC.html

[9] The GNU Compiler for the Java™Programming Language. https://gcc.gnu.org/java/

[10] GCC 4.5 Release Series. https://gcc.gnu.org/gcc-4.5/

[11] AppleInsider - Apple's top secret Swift language grew from work to sustain Objective C, which it now aims to replace. http://appleinsider.com/articles/14/06/04/apples-top-secret-swift-language-grew-from-work-to-sustain-objective-c-which-it-now-aims-to-replace

[12] ITCandor - Microprocessors Intel Leads A $94 Billion Market. http://www.itcandor.com/microprocessor-q312/

[13] ITCandor - Microprocessor Market Down 5% In Q2 2013. http://www.itcandor.com/chip-q213/

[14] Intel Forecast Shows Rising Server Demand, PC Share Gains. http://www.bloomberg.com/news/articles/2015-07-15/intel-forecast-shows-server-demands-makes-up-for-pc-market-woes

[15] VALLURI, MADHAVI AND JOHN, LIZY: Is compiling for performance== compiling for power. Interaction between compilers and computer architecture, 2001.

[16] CHAKRAPANI, L. N. AND KORKMAZ, P. AND MOONEY III, V. J. AND PALEM, K. V. AND PUTTASWAMY, K. AND WONG, W.: *The Emerging Power Crisis in Embedded Processors: What Can a Poor Compiler Do?, pp. 176–180. Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2001. ISBN 1-58113-399-5*

[17] PAN, ZHELONG AND EIGENMANN, RUDOLF: Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning, ii pp. 319–332. International Symposium on Code Generation and Optimization, 2006. ISBN 0-7695-2499-0

[18] PATYK, TOMASZ AND HANNULA, HARRI AND KELLOMAKI, PERTTI AND TAKALA, JARMO: *Energy consumption reduction by automatic selection of compiler options, pp. 1–4. 2009 International Symposium on Signals, Circuits and Systems, 2009. ISBN 978-1-4244-3785-6*

[19] Intel 64 and IA-32 Architectures Optimization Reference Manual, 325462-053. http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html

[20] HÄHNEL, MARCUS AND DÖBEL, BJÖRN AND VÖLP: Measuring energy consumption for short code paths using RAPL, 40. ACM SIGMETRICS Performance Evaluation Review, 2012.

[21] Options That Control Optimization. http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[22] GCC optimization - Gentoo Wiki. https://wiki.gentoo.org/wiki/GCC_optimization

[23] JEE WHAN CHOI AND BEDARD, D. AND FOWLER, R. AND VUDUC, R.: *A Roofline Model of Energy, pp. 661-672. IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS), 2013. ISSN 1530-2075*

BIOGRAPHIES

David Branco was born on 01. 07. 1988. In 2014 he graduated in Computer Science at University of Minho and, at present, he is a MSc student in Software Engineering at the same University. Since 2015 he is working on his master thesis in the area of Green Computing and Code Generation, namely studying ways to improve the optimization phase of compilers in order to choose the best machine instructions concerning the reduction of energy consumption. He is a co-author of some papers in this area.

Pedro Rangel Henriques got a degree in ”Electronics Engineering”, at FEUP (Porto University), and finished a Ph.D. thesis in ”Formal Languages and Attribute Grammars” at Minho University. In 1981 he joined the Computer Science Department of University of Minho, where he is a teacher/researcher. Since 1995 he is the coordinator of the ”Language Processing group” at ”Algoritmi Research Center”. He teaches different courses in the broader area of programming: Programming Languages and Paradigms; Compilers; Language Engineering; etc.