Impact of GCC optimization levels in energy consumption during C/C++ program execution

David Branco
Universidade do Minho, Portugal
davidbranco88@gmail.com

Pedro Rangel Henriques
Universidade do Minho, Portugal
pedrorangelhenriques@gmail.com

***Abstract*—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, we are carrying out a master's thesis looking for ways to get this improvement at the code generation phase. First we investigated how the hardware manufacturers have been and are concerned with green issues, aiming to find whether the modern processors are delivered with instruction sets that express the energy required to execute each operation, and whether they include code alternatives cheaper in terms of electric energy. In this paper we intend to describe a second step in this project. We will analyze the influence of GCC optimizations on the energy consumed by a program. We will compare the consumption of a given program compiled without any optimization with other versions of the same program compiled with different levels of optimization. The experiment conducted has shown that actually options -Ofast and -O3 generate a faster code that decreases the energy consumed. We report in the paper the study carried on, presenting the experimental setup and the method followed to measure and compare the resources consumed by a program during execution, and then we discuss the results obtained.**

I. 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 C/C++ 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 optimise 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.

For C/C++ compiler was chosen GCC[2](v.5.1.0) because it is a robust option, well documented, the most widely used and this allows different code optimization levels.

The last fundamental decision is concerned with the target machine, to set up the assembly code that will be generated. Of course, the solution to be developed completely depends on the instruction set of 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 [7][8][9].

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

[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/

II. 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 [10]. 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 [11].

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) [12][13][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].

III. 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.

A. 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: Ubuntu GNOME 14.04.2 LTS 64-bit (Linux Kernel 3.16); Intel® Core i7-4710HQ up to 3.5 GHz, Haswell Family; 8 GB DDR3L 1600MHz; and NVIDIA® GeForce® GT 840M, 2GB DDR3 VRAM.

B. Measurement Software

The energy measurement necessary to make the comparative study was performed using Running Average Power Limit[3](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 [14][15].

Performance Application Programming Interface[4](PAPI) and the Perf[5] 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[6], 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 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.

C. Measured Software

In the experiment described in this paper we have analyzed 6 programs that differ in some aspects such as: 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++;
- 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.

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

Program	PL	IF	OF	CC	ED	ACT (s)	AET (s)
MMC	C	None	None	Low	None	0,077	24,800
Grades	C (Flex and Yacc)	Txt	Html	Low	Few	0,231	32,908
Bzip	C	Wav	Bz2	Medium	None	1,501	23,773
Bzip2	C	Wav	Bz2	Medium	Several	1,787	23,939
Oggenc	C	Wav	Ogg	High	None	3,908	23,338
Pbrt	C++	Pbrt	Exr	High	Lot	41,516	19,265

TABLE I. SOME FEATURES OF THE MEASURED PROGRAMS.

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 I 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.

A brief description of each one of the subject programs follows: MMC the multiplication of matrices with size 1024x1024 using 6 different methods; Grades given the names and grades of students in a given discipline, generates the final evaluation; Bzip and Bzip2 are 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.

IV. 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.

A. 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 [16][17].

- -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 trade-off, 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 levels -Os and -Og for this study. The level -Og perform optimizations which do not contribute to an energy reduction compared to the other 5 discussed. Although the level -Os has some potential for significant improvement in some particular cases, for example where their optimizations are capable to fit the code in the cache, we decided to leave it for future work in order to study them more deeply.

B. 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 5 levels of optimization;
6) Repeat step 1 to 5 for all 6 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[7](for parsing the results of each operation) and GNUPlot[8](to generate plots of each program).

V. Discussion of Results

To illustrate the results obtained, three examples of charts are displayed in Figure 1, Figure 2 and Figure 3. The remaining charts, as well as the HTML pages, can be found in the online repository[9] of this work.

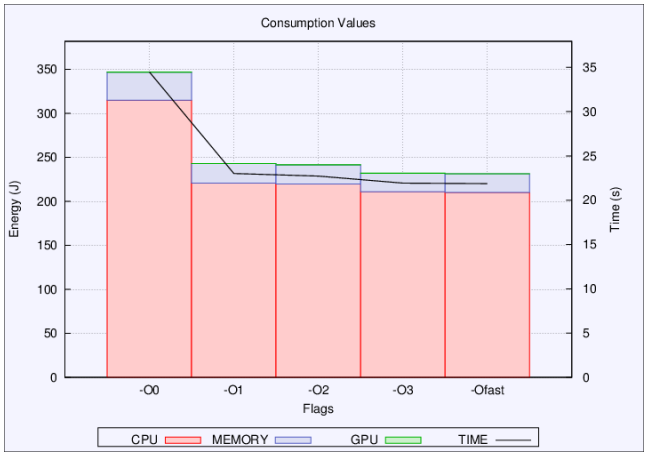

Fig. 1. Plot with the results of MMC measurements.

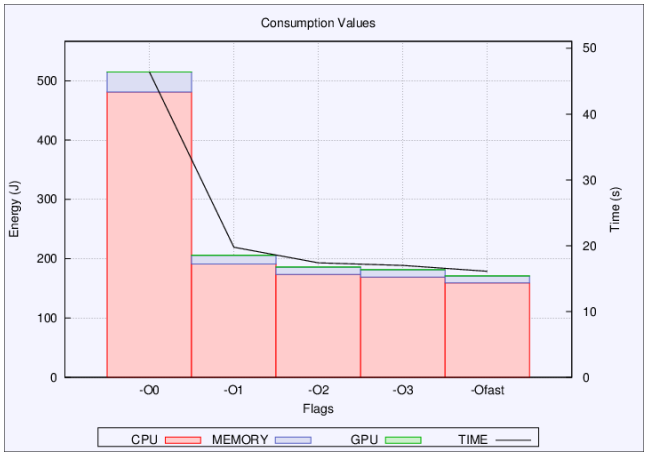

Fig. 2. Plot with the results of Oggenc measurements.

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 75%). For low complexity programs it

[7]https://www.perl.org/

[8]http://www.gnuplot.info/

[9]https://github.com/david-branco/gcc-optimization-energy-article

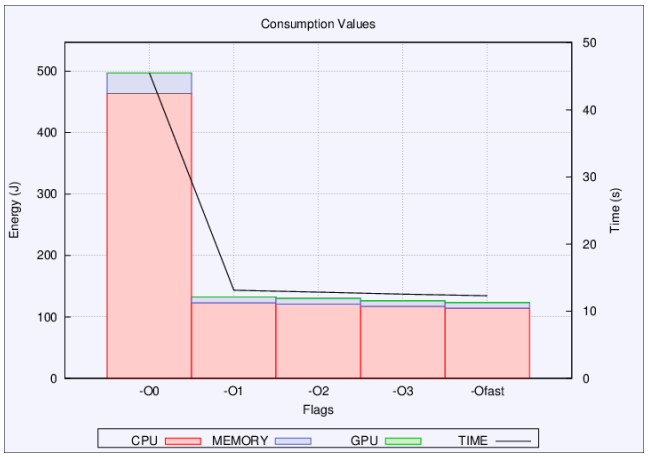

Fig. 3. Plot with the results of Pbrt measurements.

Program \ Level	-O0	-O1	-O2	-O3	-Ofast
MMC	334.478	23.009	22.719	21.930	21.864
Grades	38.141	32.625	31.503	31.170	31.101
Bzip	37.646	20.375	20.312	20.326	20.207
Bzip2	37.641	20.727	20.580	20.254	20.493
Oggenc	46.405	19.773	17.397	17.001	16.118
Pbrt	45.502	13.126	12.851	12.547	12.302

TABLE II. Execution times of programs in seconds by optimization level.

Program \ Level	-O0	-O1	-O2	-O3	-Ofast
MMC	314.970	220.796	219.703	210.974	210.213
Grades	290.560	240.781	227.802	233.465	230.638
Bzip	339.544	174.628	177.310	177.555	176.794
Bzip2	346.031	181.297	183.455	182.846	183.314
Oggenc	480.964	191.291	173.362	169.072	159.202
Pbrt	463.963	122.835	120.753	117.074	114.278

TABLE III. Selected programs and their energy consumption by optimization level.

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.

Considering all optimization levels which are not the default, analyzing the execution time (Table II), memory and CPU (Table III) energy consumption, and ignoring minimal differences of values between levels, it appears that in most cases the -Ofast level is the most efficient unlike -O1 option. 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.

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 collect along the experiment, we can say that the timeline follows the trend of the columns with the consumption of each component by optimization flag.

VI. 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++ 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 [18], 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.

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 II 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..

Acknowledgments

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] T. Guelzim and M. S. Obaidat, *Handbook of Green Information and Communication Systems*. Academic Press, 2013, ch. Chapter 8, pp. 209–227.

[2] R. R. Harmon and N. Auseklis, "Sustainable IT services: Assessing the impact of green computing practices," in *Management of Engineering Technology, 2009. PICMET 2009. Portland International Conference on*, 2009, pp. 1707–1717.

[3] L. Minas and B. Ellison, *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. Intel Press, 2009.

[4] B. Ellison, "The Problem of Power Consumption in Servers," *Energy Efficiency for Information Technology*, pp. 1–17, 2009.

[5] J. Pallister, S. J. Hollis, and J. Bennett, "Identifying compiler options to minimize energy consumption for embedded platforms," *The Computer Journal*, vol. 58, no. 1, pp. 95–109, 2013.

[6] U. Kremer, "Low power/energy compiler optimizations," 2004.

[7] ITCandor, "Intel Leads," 2012. [Online]. Available: http://www.itcandor.com/microprocessor-q312/

[8] ——, "Market Down," 2013. [Online]. Available: http://www.itcandor.com/chip-q213/

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

[10] M. Valluri and L. John, "Is compiling for performance== compiling for power," *Interaction between compilers and computer architectures*, p. 101, 2001. [Online]. Available: http://books.google.com/books?hl=en{&}amp;lr={&}amp;id=pE4guNvx8u4C{&}amp;oi=fnd{&}amp;pg=PA101{&}amp;dq=Is+Compiling+for+Performance+===+Compiling+for+Power?{&}amp;ots=Gf3qnsc59G{&}amp;sig=tW07EpiJySGmojXzJ4HsjotGPWQ

[11] L. N. Chakrapani, P. Korkmaz, V. J. Mooney III, K. V. Palem, K. Puttaswamy, and W. F. Wong, "The Emerging Power Crisis in Embedded Processors: What Can a Poor Compiler Do?" *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 176–180, 2001. [Online]. Available: http://doi.acm.org/10.1145/502217.502246

[12] Z. Pan and R. Eigenmann, "Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning," *International Symposium on Code Generation and Optimization*, no. ii, pp. 319–332, 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1611551

[13] T. Patyk, H. Hannula, P. Kellomaki, and J. Takala, "Energy consumption reduction by automatic selection of compiler options," *2009 International Symposium on Signals, Circuits and Systems*, pp. 1–4, 2009. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5206106{&}contentType=Conference+Publications

[14] *Intel 64 and IA-32 Architectures Optimization Reference Manual*, 325462nd ed.

[15] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using RAPL," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, 2012.

[16] "GNU - GCC," 2014. [Online]. Available: http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[17] "GCC optimization - Gentoo Wiki," 2015. [Online]. Available: https://wiki.gentoo.org/wiki/GCC_optimization

[18] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013, pp. 661–672.