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

David Branco Pedro Rangel Henriques

Universidade do Minho - Portugal

Informatics'2015
November - Poprad

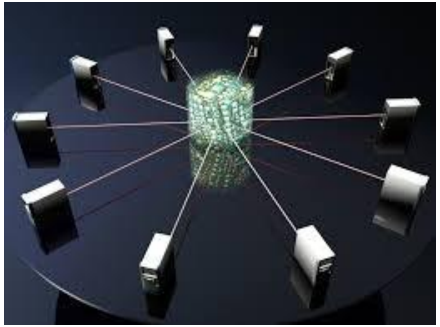
Problem

The rapid development & high demands in computer technology, the fast growth of the Internet & Massification widespread of advanced wireless and mobile devices,

produce:

- Increasing energy costs;
- Increasing cooling requirements;
- Increasing equipment power density;
- Restrictions on energy supply and access;
- Growing awareness of IT impact on the environment.

Measures and Solutions



Context

GreenSSCM project

(Green Software for Space Control Missions)

Main Research Directions :

- Measuring and visualizing energy consumption within software code;
- Detecting anomalous energy consumption in android applications;
- Defining energy consumption plans for data querying processes.

Our Objective

Main stream:

❑ Study how the compile process can influence and minimize the energy consumption (towards a *green compiler*).

1.st step:

❑ Analyse the influence of GCC optimizations on the energy consumed by a program.

❑ Compare the consumption of programs compiled by GCC using different levels of optimization.

Outline

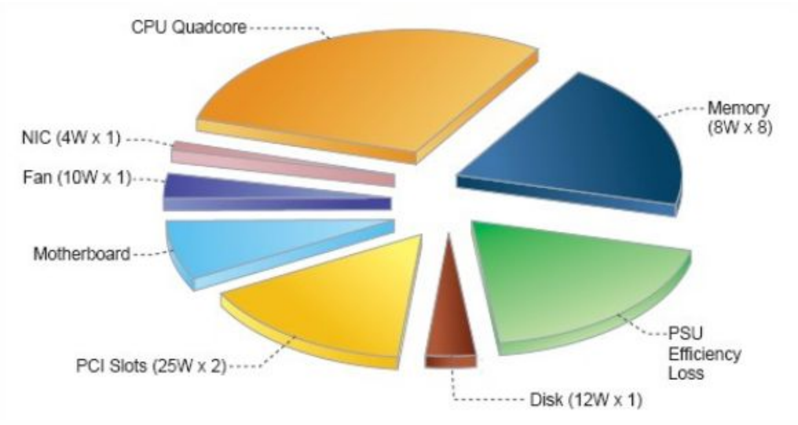
1. *Introduction*
   1. Why Microprocessors ?
   2. *Which Microprocessor Manufacturer ?*
   3. *Why Compilers ?*
   4. *Which Compiler ?*
2. Experimental Setup
3. Methodology
4. Discussion of Results
5. Conclusion
6. Future Work

1. Introduction

I. Why Microprocessors ?

- They are everywhere !!

- Largest consumer of energy.



1. Introduction

II. Which Microprocessor Manufacturer ?

“Intel’s processors are in more than 80 percent of the world’s PCs sold every year and it has 99 percent of the market for servers built on PC chips ….”

Bloomberg Business, 2015

1. Introduction

III. Why Compilers ?

- Great way to get more performance without modifying any hardware or software component.
- With just a few adjustments in the compilation arguments, and these improvements are obtained immediately.

IV. Which Compiler ?

- C/C++ compiler;
- Robust and well documented;
- Widely used.

(V. 5.1.0)

Outline

1. Introduction
2. *<u>Experimental Setup</u>*
   1. *Testing Platform*
   2. *Measurement Software*
   3. *Measured Software*
3. Methodology
4. Discussion of Results
5. Conclusion
6. Future Work

2. Experimental Setup

I. Testing Platform

- Laptop Asus N56JN-DM127H:
  - 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;
  - NVIDIA® GeForce® GT 840M, 2GB DDR3 VRAM.

2. Experimental Setup

II. Measurement Software

Running Average Power Limit (RAPL) by Intel®

Our framework allows to:

- Measure time and energy consumption of a certain operation;
- Performs readings of CPU, RAM and GPU;
- Receive as argument the path for program makefile/executable;
- Managed through a mechanism of flags;
- Set the number of cores, read through perf, etc.;

2. Experimental Setup

III. Measured Software

Programs	Program. Language	Input File	Output File	Code Complexity	External Depend.	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	9.908	23.338
Pbrt	C++	pbrt	exr	High	Lot	41.516	19.265

Outline

1. Introduction
2. Experimental Setup
3. *<u>Methodology</u>*
    I. *Optimizations Flags*
    II. *Measurement Process*
4. Discussion of Results
5. Conclusion
6. Future Work

3. Methodology

I. Optimizations Flags

- ✔ O0: Default Level (disables all optimization flags).
- ✔ O1: Basic Level (up to 39 flags).
- ✔ O2: Recommended Level (up to 73 flags).
- ✘ Os: Reduced Code Size (up to 65 flags).
- ✔ O3: Highest Level (up to 82 flags).
- ✔ Ofast: Disregard strict standards compliance (up to 85 flags).
- ✘ Og: Optimize debugging experience (up to 74 flags).

3. Methodology

II. Measurement Process

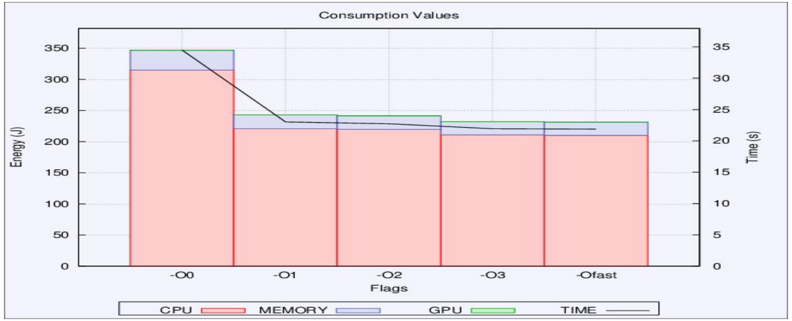
1) Choose a program and its Makefile;
2) Choose the desired optimization level;
3) Execute 100 times the measuring tool for the program and the chosen optimization level;
4) Process the output generated by each invocation of the measuring tool:
    a) Get the energy consumption and time values;
    b) 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.

Outline

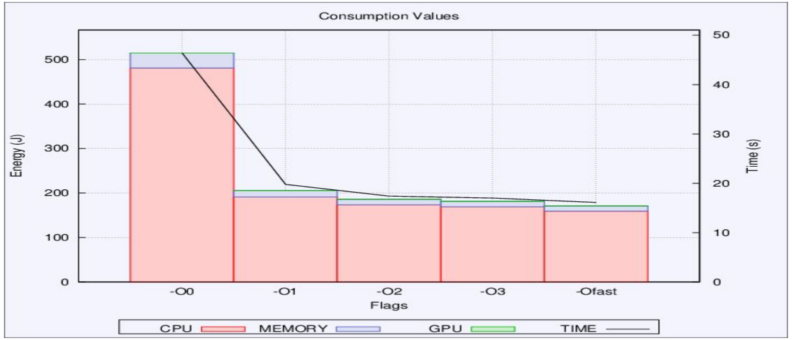
1. Introduction
2. Experimental Setup
3. Methodology
4. *Discussion of Results*
5. Conclusion
6. Future Work

4. Discussion of Results

mmc

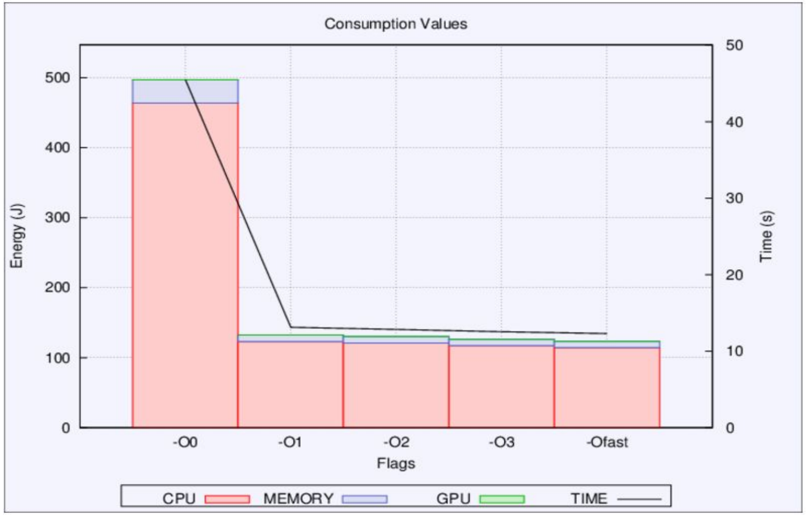


oggenc



pbrt

Flags	-O0	-O1	-O2	-O3	-Ofast
Time (s)	45.502	13.126	12.851	12.547	12.302
GPU (J)	0.228	0.172	0.176	0.174	0.168
Memory (J)	32.979	9.558	9.359	9.139	8.962
CPU (J)	463.963	122.835	120.753	117.074	114.278



Outline

1. Introduction
2. Experimental Setup
3. Methodology
4. Discussion of Results
5. *Conclusion*
6. Future Work

5. Conclusion

⇒ Energy decreases as faster is the code.

⇒ Energy/time consumptions are undoubtedly lower when selected an optimization level different from the default.

⇒ Optimization is greater for more complex source codes.

⇒ In most cases the -Ofast level is the most efficient and -O1 level is the less efficient.

⇒ It is not possible to conclude with certainty GCC's strategies on the matter.

6. Future Work

- Study the application of the algorithms referred in related work in order to obtain specific sets of flags that can further reduce power consumption.

- Look up for the cheapest machine instructions, during the compiler’s code generation/optimization phase, regarding the energy consumption (producing the most economic code).

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

Obrigado !