Energy Consumption Analysis of Algorithms Implementations

Mohammad Rashid, Luca Ardito, and Marco Torchiano
Politecnico di Torino
Department of Control and Computer Engineering, Turin, Italy
Email: name.surname@polito.it

Abstract—Context: Mobile devices, typically battery driven, require new efforts to improve the energy efficiency of both hardware and software designs. Goal: The goal of this work is to analyze the energy efficiency of different sorting algorithms implementations. Method: We set up an experiment on an ARM based device, measuring the energy consumption of different sorting algorithms implemented in different programming languages. Result: The algorithms and languages exhibit significantly different energy consumption, the ARM assembly language implementation of Counting sort is the greenest solution. Conclusion: We provide the basic information to select algorithms, and we identified the main factors affecting energy consumption.

I. INTRODUCTION

With the increasing popularity of mobile and embedded systems, energy management has become a critical challenge. Mobile devices are typically battery driven, and the importance of energy efficiency has led to rethink hardware and software designs. However, algorithm design is still relatively untouched, as a matter of fact, complexity models do not consider the energy consumed by an algorithm.

Current mobile and embedded devices run programs written with different programming languages. For this reason it is also important to consider how energy efficient is a software written with a particular programming language. Thus it is important to analyze the algorithm energy efficiency (in theory), and its implementation energy efficiency (in practice) because nowadays we only know the impact in terms of memory usage and execution time.

To do so, we will focus on sorting algorithms because they are well known with regard to complexity and implementation.

The remainder of this paper is structured as follows: Section II introduces the related work, Section III describes the context of our work, including instrumentation and research questions, Section IV presents results while Section V discusses them and, finally, Section VI provides conclusions and future works.

II. RELATED WORK

Recent research efforts have been spent on analyzing algorithms energy efficiency with regard to time and space complexity. Energy-aware algorithms have previously been examined by Jain et al. in [1]. In this paper authors defined a theoretical algorithm complexity model for energy. However, they only presented a theoretical explanation by using staple algorithms such as random number generation and sorting to verify the model.

Bunse et al. [2] examined energy efficient sorting algorithms and, based on experimental result, they presented an approach for saving energy by choosing the appropriate sorting algorithm. Based on an abstract memory model in [3] Roy et al., presented a energy model that is a (weighted) sum of the time complexity of the algorithm and the number of "parallel" I/O accesses made by the algorithm. Energy efficient sorting was explored in [4], where authors proposed an external sorting benchmark for evaluating the energy efficiency of a wide range of systems. But this benchmark was more focused on hardware criteria rather than exploring the software domain.

Relatively little work focuses on understanding energy consumption in algorithm implementation. In [5] authors examined the effects of compiler energy optimization with different levels of architectural optimization. But this study only focus on GCC Compiler Optimization of embedded systems.

The contribution of this paper is to analyze algorithm implementation energy efficiency. Therefore in our study we performed experiments aimed at accessing energy impact of algorithm execution using different programming languages (ARM assembly, C, Java).

III. EXPERIMENT DESIGN

The aim of our research is to compare the impact of some sorting algorithms with different computational complexity with regard to energy consumption. For this purpose we used a ARM-based device: a Raspberry Pi. We performed an experiment, which measures the energy consumed by different sorting algorithm over three different languages ARM assembly, C and Java. The choice of different languages allows us to evaluate the trade-offs deriving from increasing coding abstraction levels.

A. Goal Description and Research Questions

We define our goal through the Goal-Question-Metric (GQM) approach [6]. This approach, applied to our experiment, leads to the definition of the model presented in Table I.

B. Variable selection

For this experiment we selected three independent variables, or factors:
RQ3 focuses on the relationship between energy and memory usage. In particular we investigate the caching, branching and instructions per cycle for each task. We formulate the null hypothesis as:

\[ H_{m_0} \] 

there is a null correlation between memory usage (i.e. cache, branch and instructions) and energy consumed to perform the task.

### D. Instrumentation and Experiment Design

The selected usage scenarios have been implemented in ARM, C, and Java language. In order to obtain a statistically relevant data set, we created a task for each data size and language composed of five repetition of the same sorting algorithm, and each task has been repeated 30 times.

1) **Hardware Instrumentation:** The experiment has been performed on a Raspberry Pi running the Raspbian Linux distribution.

The energy consumption data was acquired through a NI USB-6210 DAQ. With this device we get current with a sampling frequency of 1kHz, and it is placed between the power source and the Raspberry Pi. The NI USB-6210 sends the collected data to a laptop.

2) **Software Setup:** We separate one single execution from another with a sleep and a while(1) loop, which last a predefined amount of time. By doing this we created tags in our collected data because the sleep has the lowest energy consumption value, while the while(1) loop has the highest energy consumption value.

The sleep power consumption value is subtracted from the task power to obtain the actual power consumed by the algorithm execution.

We used Perf profiler\(^2\) for collecting memory performance on each task execution.

### E. Analysis Methodology

Considering the first hypothesis \((H_{e_0})\) we check the distribution of energy based on these factors: **Language**, **Algorithm** and **Data Size**. In practice we check the relationship between **Energy vs. time** according to the fundamental relationship: \( E = P \cdot t \).

In addition for \((H_{t_0})\) and \((H_{c_0})\), we investigate the effects of the independent factors on the task average power consumption. For this purpose we fit a linear model based on task per different **Data Size** and **Language** and **Algorithm**. Regression is based on the following linear regression equation:

\[
\text{Power} = c_{\text{time}} \cdot \text{time} + c_N \cdot N_{\text{Bubble}} \cdot i_{\text{Bubble}} + c_{\text{Counting}} \cdot i_{\text{Counting}} + c_{\text{Merge}} \cdot i_{\text{Merge}} + c_{\text{Quick}} \cdot i_{\text{Quick}} + c_C \cdot i_C + c_{\text{Java}} \cdot i_{\text{Java}} \quad (1)
\]

Where \(i_X\) is an indicator variable for a specific level of a factor, e.g. \(i_{\text{Bubble}}\) is an indicator variable for level **Bubble** of factor **Algorithm**. Indicator variables are equal to 1 when the

---

\(^1\)https://perf.wiki.kernel.org/index.php/Main_Page Last Visited 06 June 2015

factor assumes the relative level and 0 otherwise. We added the time factor to verify that time has no effect on power.

We observe that, for the first factor in the equation (i.e. Language) we report the indicator variables for all levels, while for the second factor we report the indicator variable for two out of three levels because the coefficient for the first level is implicit.

Concerning the hypothesis ($H_{m0}$) our work explores the effect of memory management for each task execution. In particular we perform a regression of Power vs. the HW indexes, IPC, PCT_CM, and PCT_BM.

F. Threats to validity

We can identify a few different threats, which could affect the validity of our experimental results.

The first threat is related to the energy consumed by the operating system (I/O, process scheduling, etc.), which may alter the energy measures. We computed the “idle” energy consumption and then we subtracted it to our measurements in order to obtain the energy consumption due to the sole sorting algorithm task. The second threat is related to the management of unnecessary signals as noise. To overcome this problem we use low pass filter with cutoff frequency 40Hz to reduce signal noise. And finally the absolute values of energy and power consumption are specific to a single device, though they are representatives of a very popular category of devices with similar specifications. We expect to find similar trends and ratios in devices, e.g. mobile phones, which use similar ARM-based processor architectures. The statistical analyses we conducted assume that all experiments are independent: in our setup the array is re-initialized before every sort execution, and when distinct process executions are isolated from each other because no permanent storage is used.

IV. RESULTS

The distributions of the average power and total energy per different size, algorithm, and language are shown in Figure 1 by means of boxplots. According to a Kruskal-Wallis test the three factors have a significant effect on Energy ($p<0.001$ in all cases), as we can visually observe in the Figure.

Concerning the relationship between time and energy, first of all we fit a regression equation of Energy vs. time, according to the fundamental relationship:

$$\text{Energy} = \text{Power} \cdot \text{time}$$

The regression estimates the coefficient Power = 0.158, it is significant ($p<0.01$); overall the regression $R^2 = 89.1\%$.

The coefficients and the relative significance levels for the regression of Power vs. time, Algorithm, Language, and N are reported in Table II.

3 To understand the main causes of the effect of the Language and Algorithm factors, we performed a regression of Power vs. the HW indexes. The coefficient and statistical significance are reported in Table III.

---

**Fig. 1. Power and Energy for all configurations**

**TABLE II**

| Time         | Estimate | Std. Error | t value | Pr(>|t|) |
|--------------|----------|------------|---------|----------|
| time         | -3.3e-06 | 1e-05      | -0.31   | 0.75     |
| N            | 5.6e-07  | 2.3e-08    | 24      | <0.001   |
| AlgorithmBubble | 0.137        | 0.0027     | 51      | <0.001   |
| AlgorithmCounting | 0.119       | 0.0023     | 52      | <0.001   |
| AlgorithmMerge  | 0.107       | 0.0023     | 47      | <0.001   |
| AlgorithmQuick | 0.105       | 0.0023     | 46      | <0.001   |
| LanguageC     | -0.033     | 0.002      | -17     | <0.001   |
| LanguageJava | 0.036      | 0.0019     | 19      | <0.001   |
As expected, we obtained different levels of energy consumption for different algorithms and languages. We have a good linear regression of Energy vs. time. The percentage explained by the sole duration of the task is slightly more than 89%. The residuals can be explained in terms of different (average) power consumption by the different algorithms and languages.

By looking at the upper part of Figure 1 we can observe that among algorithms, Counting sort exhibits better performances, followed by Quicksort. We did not run Bubble sort on the largest array because it would be absolutely non comparable.

Concerning the language performance, we observe a consistent ranking with ARM assembly as the least energy hungry language and Java as the most energy hungry.

Looking at the (average) power during task (bottom diagram in Figure 1) we observe a similar order among algorithms as for energy. This visual analysis is confirmed by the coefficients relative to the four algorithms in the regression reported in Table II.

By looking at the coefficients in Table II we observe that the effect of languages on average power is roughly an order of magnitude smaller than the algorithm. In particular language C appears more efficient than ARM which, in turn, is more efficient than Java. This order is different from the one observed for the total energy, where ARM is consistently smaller than C, it often exhibits a higher (average) power consumption pattern.

An overall view of the above results can be obtained by looking at how the median values of the dependent variables (for each combination of factor levels) as shown in Figure 2.

We can observe that while in terms of time ARM is consistently smaller than C, it often exhibits a higher (average) power consumption.

The different power consumption can be explained mainly by means of the the PCT_CM that has the largest coefficient (see Table III). The task executed with the three languages actually exhibit different median values for such indicator: 0.17% for ARM, 0.11% for C and 0.86% for Java. Partially counter-intuitive is the negative sign for the PCT_BM coefficient. We speculate that a larger number of branch misses slows down the processor, therefore lowering the power consumption.

We observe that the both the percentage of cache misses and that of branch misses are statistically significant predictors, while the instructions per cycle is not.

V. DISCUSSION

VI. CONCLUSIONS AND FUTURE WORKS

We analyzed the energy and power consumption of different sorting algorithms and the relative implementation in three distinct languages. Both the algorithm and the language significantly affect the total energy consumption.

A large part of the energy consumed is determined by the time performance, i.e. computational complexity of the solution, though roughly 10% is determined by a specific power consumption pattern.

Our study provides the basic information to chose a specific sorting implementation to minimize energy consumption.

As further work we plan investigating in more details the hardware factors that affect energy consumption and how they are related to specific language idioms.

REFERENCES


