The Impact of Source Code in Software on Power Consumption

Hayri Acar, Gülfem Alptekin, Jean-Patrick Gelas, Parisa Ghodous

THE IMPACT OF SOURCE CODE IN SOFTWARE ON POWER CONSUMPTION

Hayri Acar, Gülfem I. Alptekin, Jean-Patrick Gelas and Parisa Ghodous
[1]*LIRIS, University of Lyon, Lyon, France*
[2]*Galatasaray University, Istanbul, Turkey*
[3]*ENS Lyon, LIP, UMR 5668, Lyon, France*
{hayri.acar, jean-patrick.gelas, parisa.ghodous}@univ-lyon1.fr, gisiklar@gsu.edu.tr

ABSTRACT

Writing sustainable, power efficient and green software necessitates understanding the power consumption behavior of a computer program. One of the benefits is the fact that developers, by improving their source code implementations, can optimize power consumption of a software. Existing power consumption models need to be improved by taking into account more components susceptible to consume energy during runtime of an application. In this paper, we first present a detailed classification of previous works on power consumption modelization. Then, we introduce TEEC (Tool to Estimate Energy Consumption) model in order to estimate the power consumed by CPU, memory and disk due to the execution of an application at runtime. The main goal is to guide developers to improve their source code for optimizing energy consumption. TEEC enables determining the part of the code consuming the highest power. This will help to obtain a less energy consuming software with the same functionalities.

Keywords: Sustainable Software, Green Software, Power Consumption, Energy Efficiency, Green IT.

1. INTRODUCTION

The 2015 Paris Climate Conference, COP 21 (Conference of the Parties), the conference have reaffirmed the objective of keeping the rise in temperature below 2°C before the end of the century, by controlling the global greenhouse gas emissions [1]. Information and Communication Technologies (ICT) represents around 2% of worldwide greenhouse gas emissions (GGE) [2]. Moreover, the number of mobile users is increasing due to new technologies, such as mobile Internet, cloud computing, Internet of things, etc. Thus, it is predicted, if nothing is done, that ICT global GGE will be 4% by 2020 [3].

Writing sustainable, power efficient and green software necessitates understanding the power consumption behavior of a computer program. One of the benefits is the fact that developers, by improving their source code implementations, can optimize power consumption of a software. Existing power consumption models need to be improved by taking into account more components susceptible to consume energy during runtime of an application.

In this paper, we first present a detailed classification of previous works on power consumption modelization. Then, we introduce TEEC model (Tool to Estimate Energy Consumption) in order to estimate the power consumed by CPU, memory and disk due to the execution of an application at runtime. The main goal is to guide developers to improve their source code for optimizing energy consumption. TEEC enables determining the part of the code consuming the highest power. This will help to obtain a less energy consuming software with the same functionalities.

This paper is organized as follows. In Section 2, we present a detailed survey of the related works on power modeling and measurement. Then, we describe the modelization of different components in terms of power consumption in Section 3. In Section 4, we represent our proposed model TEEC, followed by experiments in Section 5. We validate the accuracy of TEEC in Section 6. Finally, Section 7 concludes the work.

2. RELATED WORKS

In related literature, it is possible to find several online tools [4, 5], which aim to estimate the power consumption arising from different components like CPU, memory, disk, network card, etc. However, these power calculators are not accurate enough and

give a global estimation on consumed energy. We believe that there is a need to have a tool, which can accurately estimate the power consumption of an application. For this purpose, researchers have used different methodologies that we can classify into three main categories: hardware methodologies, software methodologies and hybrid methodologies.

2.1 Software Methodology

This type of methodologies estimate the consumed power based on mathematical formula, which is established according to the characteristics of each component susceptible to consume power. We followed systematic review methodology [6] to analyze previous works in literature. We respect the systematic mapping process [7].

2.1.1 Research Type Facet

We summarize research approaches respecting research type facet in Table 1.

Name	Description
Validation Research	Investigated techniques are novel and have not yet been implemented in practice.
Evaluation Research	Techniques are implemented in practice and an evaluation of the technique is conducted.
Solution Proposal	A solution for a problem is proposed, the solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution is shown by a small example or a good line of argumentation.

Table 1: Research Type Facet

2.1.2 Research Nature Facet

In Table 2, related works on component-based power estimation models are summarized.

Name	Description
CPU	It consists of the studies, where the CPU is taken into account in order to establish a power estimation model of software.
Memory	It consists of the studies, where the memory is taken into account in order to establish a power estimation model of software.
Disk	It consists of the studies, where the disk is taken into account in order to establish a power estimation model of software.

Table 2: Research Nature Facet

Therefore, we will use these two criteria (research work and research type facet) when classifying the works. The final classification is represented in Table 3, where we give also related mathematical equations, together with the brief description of each study.

Using the information in Table 3, we establish a diagram in two bubble plots that is represented in Figure 1.

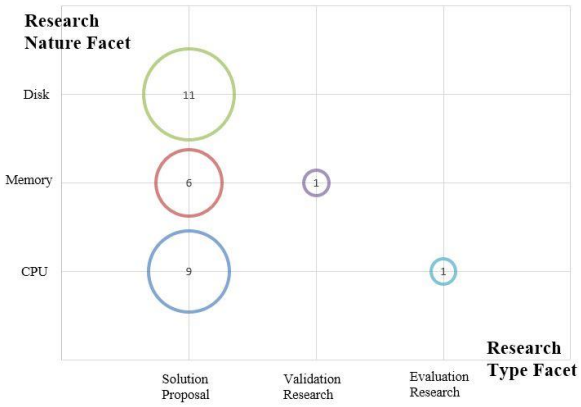

Figure 1: Systematic map in a bubble plot of research type and nature facets

So, we observe that the majority of studies for calculating power consumption of software takes into account only one component and neglects others. Moreover, the most remarkable research type facet is solution proposal.

Study	Research Type Facet	Research Nature Facet	Formula	Description
Wattch [8]	Solution Proposal	CPU	$P_d = C.V_{dd}^2.a.f$ Pd: dynamic power consumption, C: load capacitance, Vdd: supply voltage, f: clock frequency and a: fraction between 0 and 1.	A framework for estimating CPU power consumption at the architectural level.
Framework proposed by Gupta and Singh [9]	Solution Proposal	CPU	$RawPower = \frac{UserTime + KernelTime}{10^9}.CpuUsage$ RawPower: power consumed by each process, UserTime: execution time spent in user mode, KernelTime: execution time spent in kernel mode and CpuUsage: CPU usage of each process in the process list.	A framework of CPU power modeling in order to minimize power consumption.
PowerAPI [10]	Solution Proposal	CPU	$P_{soft} = P_{comp} + P_{com}$ Pcomp: CPU power consumed and Pcom: network card power consumption.	Tool that estimates the CPU energy consumption of running processes.
Power model by Bertran [11]	Solution Proposal	CPU	$P_{total} = \sum_{i=1}^{numcomp} A_{ri}.P_i + P_{static}$ Pi: weight of component i, Ari: activity ratio and Pstatic: static power consumption of all components.	Power model that estimates the power consumption due to CPU component.
Span [12]	Solution Proposal	CPU	$P(ai,fi)_{prettotal} = \sum_{k=1}^{cores} P(ai,fi,k)_{pret} + P(fi)$ aj: target benchmark, P(aj,fi,k)prêt: generated at per core level and P(fi): power pilot for frequency fi	Manually, specific code can be added in order to locate parts of source code power consumer.
Simwattch [13]	Solution Proposal	CPU	$P_d = C.V_{dd}^2.\alpha.F$ Pd: dynamic power consumption, C: load capacitance, Vdd: supply voltage, F: clock frequency and α: fraction between 0 and 1.	Power simulator that estimates CPU power consumption.
CAMP [14]	Solution Proposal	CPU	$P_d = A.C_{effective}.V^2.f$ A: the fraction of cycles a specific event occurs, Ceffective: effective capacitance, V: voltage and f: clock frequency.	Estimates the power consumption due to CPU in runtime.
Joulemeter [15]	Solution Proposal	CPU	$E_{sys} = E_{CPU} + E_{Mon} + E_{Disk} + E_{static}$ CPU, monitor, disk and static energy.	For a given process, estimates only CPU power consumption.
SoftWatt [16]	Solution Proposal	CPU, Memory, Disk	$P = P_{CPU} + P_{memory} + P_{disk}$ CPU, memory and disk power	Just estimates power consumption without give information about source code.
vEC [17]	Solution Proposal	Memory	$E = E_{bus} + E_{cell} + E_{pad} + E_{main}$ Ebus: data and address bus energy, Ecell: cache energy, Epad: data and	Virtual Energy Counters, to estimate the energy consumption of

			address pad energy, Emain: main memory energy.	software.
CACTI-D [18]	Solution Proposal	Memory		A comprehensive memory modeling tool.
Proposed by Vogelsang [19]	Solution Proposal	Memory	$P = \sum_i 0.5.C_i.V_i^2.f_i$ C: capacitance, V: voltage, f: frequence and i: all charging and discharging events.	Power model based on DRAM architecture in order to operate power usage.
DRAMsim [20]	Solution Proposal	Memory		Memory system simulator.
Memory System Power [21]	Solution Proposal	Memory	$P(TOT) = P(PRE_{PDN}) + P(PRE_{STBY}) + P(ACT_{PDN}) + P(ACT_{STBY}) + P(WR) + P(RD) + P(REF) + P(termW) + P(termRoth) + P(termWoth)$	A detailed study on DDR3 SDRAM power consumption.
SimplePower [22]	Solution Proposal	Memory		Framework to evaluate the memory system.
Proposed by Bisson [23]	Solution Proposal	Disk		Reduce hybrid disks power consumption.
Proposed by Benjamin [24]	Solution Proposal	Disk	$FirstLBACycl = N_{heads}.\sum_i L_i$ FirstLBACyl: function for the first LBA on that cylinder and Nheads: constant number of heads in the drive.	Simulator based on mathematical hard disk timing model.
Hylick tool [25]	Solution Proposal	Disk	$Energy \approx LBN^3$	Estimates power consumption due to hard drives to find runtime power profile.
Engel Proposition [26]	Solution Proposal	Disk	$P = \frac{2\sigma}{m.V_x}.P = \beta.P.(2\sigma)^2$ β: inverse temperature, m: mass, Vx: velocity along one axis, P: pressure and σ: radius.	Examines different phases of hard disks.
MIND [27]	Solution Proposal	Disk	$E_{Raid} = \sum P_{Raid,i}.tin_i + \sum E_{action,j}.counts_j$	Measure power consumption of different phases.
Dempsey [28]	Solution Proposal	Disk	$P_{Drive} = V_{Drive}.I_{Drive} = V_{Drive}.\frac{V_{Resistor}}{R_{Resistor}}$ Vdrive: voltage of the power supply to the disk drive and Idrive: current.	Disk power consumption simulator.
Vesper [29]	Solution Proposal	Disk	$T_{total} = T_{seek} + T_{rotation} + T_{transfer}$ Ttotal: total time of a disk operation, Tseek: seek time, Trotation: rotation delay and Ttransfer: transfer time	Disk power simulator based on the different time passed on each stage.
Proposed by Zhu [30]	Solution Proposal	Disk	$E_i(t) = P_i(t - Ti) + Ci$ Pi: power dissipation in mode I, Ti	Disk power management to save energy.

			and Ci: time and energy required to spin-down and spin-up from power mode i to 0.	
Lewis Proposition [31]	Solution Proposal	Disk	$E_{hdd} = P_{spinup}.t_{su} + P_{read}.\sum Nr.t_r + P_{write}.\sum Nw.t_w + \sum Pidle.t_{idle}$	Real-time energy estimation model that gives server energy consumption.
SODA [32]	Solution Proposal	Disk	$P_{spm} = n.b.wspm^{2.8}$ n: number of platters and b: viscous friction coefficient.	Sensitivity based optimization of disk architecture.
Tempo [33]	Solution Proposal	Disk		Measure Power consumption of the disk during data transfers and disk head seeks

Table 3: Research works classification

2.2 Hardware Methodology

Research works, using hardware methodologies in order to measure the power consumed by components, can be grouped in two categories. First [34, 35], power meters are used to measure directly the voltages and currents in devices to obtain the power. Second way [36] consists to connect power sensors directly into the component that we want to measure the power consumption. This approach is particularly used by high performance servers.

Hardware methodologies are more accurate than software methodologies. However, it is impossible to measure the power consumed by programs on process and virtual machines. Moreover, this method is expansive and circuits consume also power.

2.3 Hybrid Methodology

Hybrid methodology [37, 38] is also a research area, since it enables taking the accuracy of hardware methodologies and the simplicity of software methodologies. However, this way of measurement methodology is more difficult to establish, in practice.

3. POWER MODEL

The power consumption of the software is composed of two parts: static and dynamic. Static power consumption is due to the manufacturer component's features. Therefore, we cannot modify this part. Hence, we are interested only in dynamic power consumption, which depends on source code of software. In order to model the power consumption of different components, we take into account only dynamic part of power consumption.

3.1 Power Model of CPU

As shown in Table 3, the power consumption equation of CPU is, in the majority of cases, the multiplication of frequency, square of voltage and a constant. So, we propose our formula (1) that is distinguished from others concerning the constant part:

$$P_{CPU} = \beta \, . \, f \, . V_{dd}^2 \qquad (1)$$

where $\beta = C_L \, . N \, . \alpha$, the constant, C_L is the capacitance, N represents the number of gates and $\alpha < 1$ as the average fraction of gates that commute at each cycle, f is the frequency and V_{dd} corresponds to voltage.

The difference of the proposed equation is in the constant part. In order to obtain the power consumed by a specific process, we multiply (1) by the percentage of the process id N_{id} (2):

$$P_{CPU,\ id} = P_{CPU} \, . N_{id} \qquad (2)$$

3.2 Power Model of Memory

Dynamic DRAM power is composed of four states: activate, precharge, read and write. So, power consumption can be expressed as (3):

$$P_{DRAM} = P_{Activate} + P_{Precharge} + P_{Read} + P_{Write} \qquad (3)$$

We multiply previous equation (3) by the usage percent M_{id} of the process id to obtain Eq. (4):

$$P_{DRAM,\ id} = P_{DRAM} \, . M_{id} \qquad (4)$$

3.3 Power Model of Disk

A disk executing a sequence of requests is composed of four mode: active, idle, standby and sleep.

The dynamic disk power consumption is obtained when the disk is in active mode. Thus, we can deduce the following equation (5):

$$P_{Disk} = P_{Active} = P_{Read} + P_{Write} \qquad (5)$$

where P_{Read} is the read power and P_{Write} is the write power.

3.4 Total Power Consumption

Based on previous equation, it is possible to define the global power consumption due to software by adding Eq. (2), (4), and (5) in order to obtain the following expression (6):

$$P_{Soft} = P_{CPU,id} + P_{DRAM,id} + P_{Disk} \quad (16)$$

4. TEEC (TOOL TO ESTIMATE ENERGY CONSUMPTION)

4.1 Green Process

All development processes of a computer program requires following a specific sequence in order to complete the project. In addition, after each phase, a green analysis step can be involved in order to check if the considered step has respected all criteria that allow reducing energy consumption. If the criteria of a phase are not validated by the green analysis, depending uncommitted specifications, a return to the previous step or even return until the requirement analysis step can be performed.

The process described in [10] presents a comprehensive progress of a development project. Thus, we offer our descriptive diagram in Figure 2.

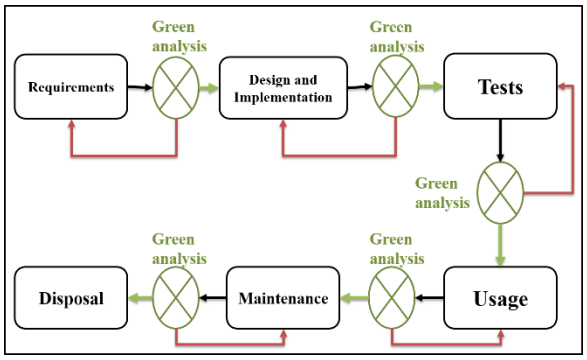

Figure 2: Green Software Engineering Process

Requirements: It is the first step in order to build a software product. This stage corresponds to the descriptions of the tasks that will be performed by the product. The aim is to meet customer demands.

Design: The defined requirements are considered in order to create system architecture. The classes and the relationships among them are defined at this stage.

Implementation: In this step, the program is implemented in respect to its design. Developers should choose the most appropriate programming language.

Tests: This step allows checking if the software meets its requirements, to discover faults or defects. The tests will be defined at the end of requirements phase (QCHP) before design and implementation step, to show that the specifications have been understood. Use of different tests will allow developers to see if the requirements are correct and consistent.

The proposed energy consumption measurement tool (TEEC) will be used in order to know whether the program can be improved.

Usage: This step defines how the software product can be used by the user in a green manner. The responsibility belongs to the user, but also to the engineers themselves. The user should be trained to use the software, because the fact that improper handling can cause errors in the program.

Maintenance: Newer versions or enhancements usually involve changes. The developers need to handle them. Furthermore, developers need to know the cost is proportional to the energy waste. Several types of errors in the program can cause the return to the implementation phase, but sometimes even more complicated errors can cause the developer to return to the first step of requirement analysis. The maintenance process must be carried out in the most energy efficient manner.

Disposal: Software and hardware must be replaced when it is not profitable to up to date them, or when it is no longer used, or when it has become obsolete. This step considers both the software and the hardware running the code. Disposal of old hardware also causes energy consumption.

Green analysis: This step can be added at the end of each one in order to improve energy efficiency. This stage will evaluate the greenness of the software.

4.2 Design and Implementation

According to [10], Java programming language is stated as the language with the least energy consumption during compilation and execution stages. Thus, Java is chosen as the development language.

Sigar library [40] allows getting information about the CPU usage, including the percentage of usage of each process and the number of cores used. Thus, the id of the ongoing process can be identified and retrieved. Moreover, the form of global variable data providers is formed that allows estimating the energy and assigning a corresponding value.

Java agents are utilized, which are software components that provide with the instrumentation capabilities to an application, such as re-defining the content of class that is loaded at run-time.

Our proposed model TEEC, with whom we can provide an estimation of power consumption of each component is illustrated in Figure 3.

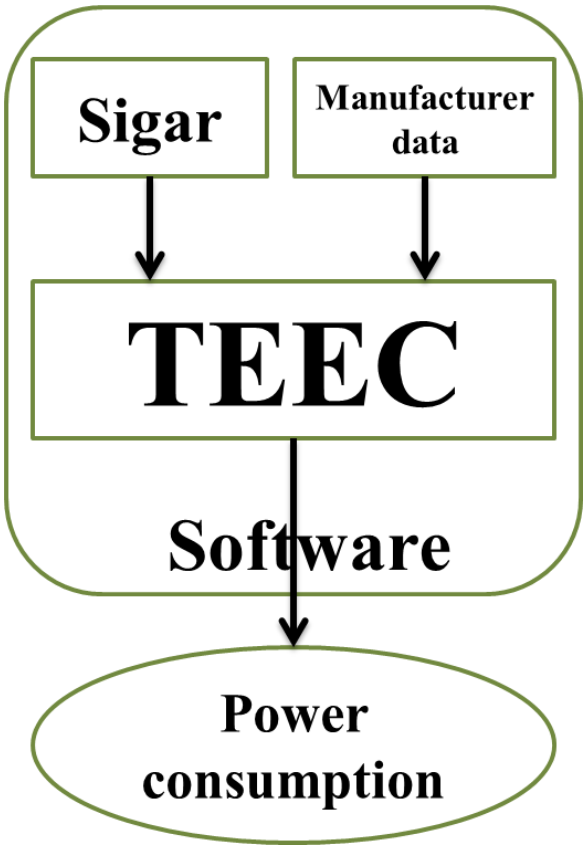

Figure 3: TEEC

So, using this model, an estimation of power consumption due to each component during runtime of software can be provided.

5. EXPERIMENTS

We carry out our tests on a notebook ASUS N751JK-T7238H, running Windows 8.

Thus, using TEEC, different tests have been executed with unoptimized and optimized methods in order to observe the variation of the power consumption due to the CPU, the memory and the disk and compare them

5.1 Tests Description

Loops have an important effect on the performance of a program and provide efficient way for repeating a piece of code as many times as required. Java has three types of loop control structures which are: *while, do-while and for*. If we do not know the number of required iterations, then *while* loop can be used. The *do-while* loop is always executed at least once and then the condition is checked at the end of the loop. If we know how many iterations are required, then we *for* loop

Therefore, it is interesting to study some methods that are used during a development of a program in order to examine possible improvement.

5.1.1 Array copy

It is better to use an *int* data type than *byte* or *short* data types for a loop index variable, because of its efficiency. The fact to use *byte* or *short* data type as the loop index variable involves implicit type cast to *int* data type.

It is always efficient to copy arrays using System.arraycopy() than using a loop. Table 4 shows the difference between optimized and unoptimized source code.

Unoptimized	Optimized
`for (int j = 0; j < a.length; j++) b[j] = a[j];`	`System.arraycopy(a, 0, b, 0, b.length);`

Table 4: Array copy

5.1.2 Locality of Reference

Elements close to each other in memory are faster to access. We can observe this principle with the programs described in Table 5. Locality of reference in an array is used.

In the unoptimized version, the loop reads the values of 100 elements in an array. In the optimized version, the loop loads 100 elements, but they are spaced 100 elements apart from each other.

Unoptimized	Optimized

`for (int i = 0; i < 1000000; i++) {`
`  int sum = 0;`
`  for (int x = 0; x < 50000; x += 100) {`
`  sum += values[x];`
`  }`
`}`
	`for (int i = 0; i < 1000000; i++) {`
	`  int sum = 0;`
	`  for (int x = 0; x < 500; x++) {`
	`  sum += values[x];`
	`  }`
	`}`

Table 5: Locality of Reference

5.1.3 Array and array list

Arrays are harder to use than ArrayLists, but they have a speed advantage, even on simple element accesses. In Table 6, we represent a sum of two 100-element collections: an array and an ArrayList.

Unoptimized	Optimized

`for (int i=0; i < 1000000; i++)`
`{`
`  int sum = 0;`
`  for (int v = 0; v < list.size(); v++)`
`  sum += list.get(v);`
`}`
	`for (int i = 0; i< 1000000; i++)`
	`{`
	`  int sum = 0;`
	`  for (int v = 0; v < array.length; v++)`
	`  sum += array[v];`
	`  }`

Table 6: Array and array list

5.1.4 Integer list loop

There are several ways to iterate elements of an integer list. In Table 7, we compare two different ways.

Unoptimized	Optimized

for (Integer i : list)
count++;
	int size = list.size();
	for (int i = 0; i < size; i++)
	count++;

Table 7: Integer list loop

5.1.5 Char array and StringBuilder

We can replace a StringBuilder with a *char* array in some programs as in Table 8.

Unoptimized	Optimized

for (int i = 0; i < 1000000; i++) {
StringBuilder builder = new StringBuilder();
for (int v = 0; v < 1000; v++)
builder.append('?');
String result = builder.toString();
}
	for (int i = 0; i < 1000000; i++)
	{
	char[] array = new char[1000];
	for (int v = 0; v < 1000; v++)
	array[v] = '?';
	String result = new String(array);
	}

Table 8: Char array and StringBuilder

5.1.6 Binary search

As showed in Table 9, the BinarySearch method searches an integer in a sorted array of integers. This is more practical to use compare to a *for* loop.

Unoptimized	Optimized

for (int i = 0; i < 10000000; i++) {
int index = -1;
for (int j = 0; j < values.length; j++)
{
if (values[j] == 80)
{
index = j;
break;
}
}
}
	for (int i = 0; i < 10000000; i++)
	{
	int index = Arrays.*binarySearch*(values, 80);
	}

Table 9: Binary search

5.2 Results

We develop two JAVA projects in order to regroup all the optimized and unoptimized methods previously defined. We obtain the following power and energy related relationships (Figure 4, 5, 6 and 7).

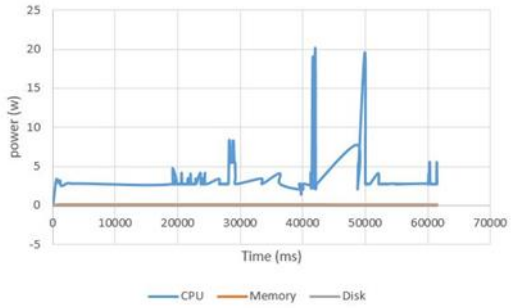

Figure 4: Power consumption of an unoptimized code

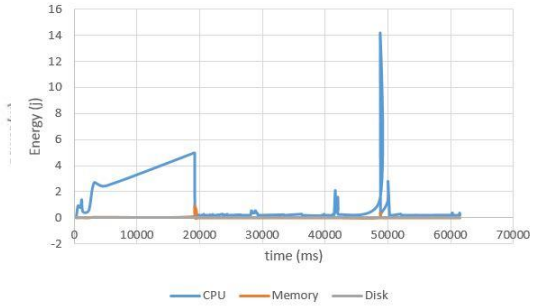

Figure 5: Energy consumption of an unoptimized code

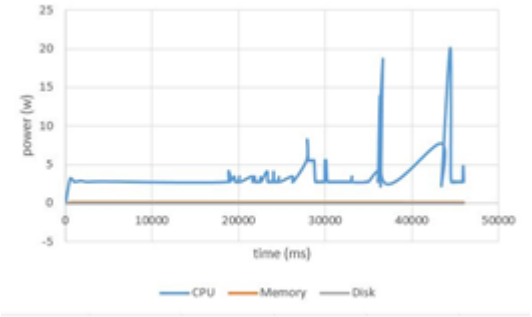

Figure 6: Power consumption of an optimized code

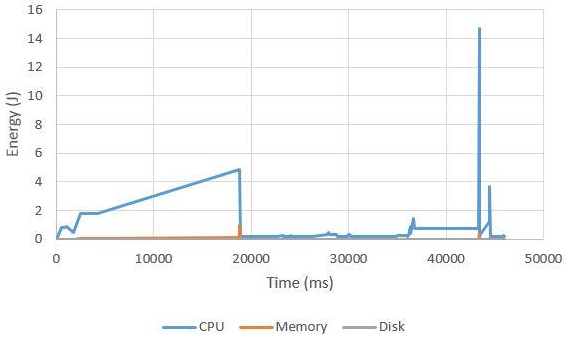

Figure 7: Energy consumption of an optimized code

Therefore, we observe that globally the power consumption of CPU dominates memory or disk consumption. If we examine the results obtained each 50 ms, we can note that the power consumption of

disk can be neglected for these cases, but in some cases power consumption of memory must be taken into account. In addition, we can note that the power consumption of the unoptimized code is higher than the one of the optimized code and the total execution time of optimized code is less than the one of the unoptimized code. Consequently, it is a great interest to develop optimized parts of code in order to obtain green, sustainable and efficient software.

So, going more in details, for each method code, we measure the time elapsed during the execution of the tests and results are represented in Table 10.

Functions	Unoptimized	Optimized
	Time (ms)	
Array copy	359	312
Locality of reference	18140	17219
Compare array to array list	22047	17297
Compare integer list loop	7734	7391
Char array StringBuilder	11235	2421
Binary search	2250	438

Table 10: Functions time execution

Hence, optimized codes are found faster than unoptimized codes. Particularly, we can remark a faster execution of the following optimized methods: “Locality of reference”, “Compare array to array list”, “Char array StringBuilder” and “Binary search”.

6. VALIDATION

To validate our experiments, we use a powermeter ‘wattsup ?PRO’ as shown in Figure 8. We connect this powermeter to the notebook via USB port. This device saves in his memory the power consumed by all process in runtime. So, we connect WattsUp to the notebook and then we wait until the power reach a stationary state. Then, we execute the unoptimized code, followed by the optimized code. We then transfer the results using the application WattsUpUSB and the results are depicted in Figure 9.

Figure 8: wattsup?PRO

Comparing to the results obtain with TEEC, even if we make a measurement in each second, we can say that in all of the case, optimized code test is faster and reveals less power than unoptimized code test. Each optimized and unoptimized curves present some increase of power as we observed with TEEC.

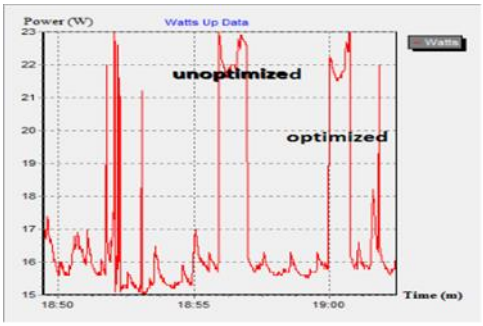

Figure 9: unoptimized and optimized functions power consumption

7. CONCLUSION

In addition to the CPU, a modelization of memory and hard disk have been made to describe the consumption behavior of each component. The proposed tool, named TEEC, takes into account all these three components. Mathematical expressions have been established in order to calculate the power consumption of each component.

The accuracy of TEEC has been tested over several optimized and unoptimized functions and validated against a real powermeter.

The results revealed that the power consumption of memory should not always be neglected when compared to the CPU power consumption, whereas power consumption of hard disk can be neglected. We observed that the optimization of source code is required in order to contribute to the reduction of the greenhouse gas emissions.

Going further, we will extend the capability of TEEC by integrating other components power consumption (such as network interface cards, etc.). Then, we will use the output of TEEC to guide developers in order to build greener software in real time and analyze the results.

REFERENCES

1. COP 21, 2015, "*Adoption of the Paris Agreement,"* Paris. URL: https://unfccc.int/resource/docs/2015/cop21/eng/l09.pdf
2. Gartner, 2007, "*Green IT: The New Industry Shock Wave*, Gartner," *Presentation at Symposium/ITXPO Conference*.
3. Green Touch, "*ICT Industry Combats Climate Change"*. URL: http://www.greentouch.org/?page=how-the-ict-industries-can-help-the-world-combat-climate-change.
4. Power Supply Calculator, 2015, URL: http://powersupplycalculator.net/
5. eXtreme Power Supply Calculator, 2016, URL: http://outervision.com/power-supply-calculator
6. Kitchenham, B., 2013, "A systematic review of systematic review process research in software engineering," *Information and Software Technology*, Vol. 55, pp. 2049-2075.
7. Petersen, K., Feldt, R., Mujtaba, S. and Mattsson, M., 2008, "Systematic Mapping Studies in Software Engineering," *the 12th international conference on Evaluation and Assessment in Software Engineering*, pp. 68-77.
8. Brooks, D., Tiwari, V. and Martonosi, M., 2000, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proceedings of the 27th International Symposium on Computer Architecture*, pp. 83-94.
9. Gupta, P.K. and Singh, G., 2011, "A Framework of Creating Intelligent Power Profiles in Operating Systems to Minimize Power Consumption and Greenhouse Effect Caused by Computer Systems," *Journal of Green Engineering*, Vol. 01, No. 2, pp. 145-163.
10. Noureddine, A., Bourdon, A., Rouvoy, R and Seinturier, L., 2012, "A Preliminary Study of the Impact of Software Engineering on GreenIT", *First International Workshop on Green and Sustainable Software*, pp. 21-27.
11. Bertran, R, Gonzales, M., Martorell, X., Navarro, N. and Ayguadé, E., 2010, "Decomposable and Responsive Power Models for Multicore Processors using Performance Counters," *Proceedings of the 24th ACM International Conference on Supercomputing*, pp. 147-158.
12. Wang, S., Chen, H. and Shi, W., 2011, "SPAN: A software power analyzer for multicore computer systems," *Journal of Sustainable Computing: Informatics and Systems*, Vol. 01, No. 1, pp. 23-34.
13. Chen, J., Dubois, M. and Stenstrom, P., 2003, "Integrating complete-system and user-level performance and power simulators," *IEEE International Symposium on Performance Analysis and Software*, pp. 1-10.
14. Powell, .D., Biswas, A., Emer, J.S. and Mukherjee, S.S., 2009, "CAMP: atechnique to estimate per-structure power at run-time using a few simpleparameters," *IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 289-300.
15. Kansal, A., Zhao, F., Liu, J, Kothari, N. and Bhattacharya, A.A., 2010, "Virtual Machine Power Metering and Provisioning," *Proceedings of the 1st ACM symposium on Cloud Computing*, pp. 39-50.
16. Gurumurthi, S. et al., 2002, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, pp. 141.
17. Kadayif, I. et al., 2001, "vEC: Virtual Energy Counters," *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pp. 28-31.
18. Thoziyoor, S. et al., 2008, "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies*," 35th International Symposium on Computer Architecture*, pp. 51-62.
19. Vogelsang, T., 2010, "Understanding the Energy Consumption of Dynamic Random Access Memories," *43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 363-374.
20. Wang, D. et al., 2005, "DRAMsim: A Memory System Simulator," *ACM SIGARCH Computer Architecture News*, Vol. 33, No. 4, pp. 100-107.
21. Janzen, J., 2008, "Calculating Memory System Power for DDR3", *designline, vol. 10, No. 2.*
22. Ye, W. et al., 2000, "The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool," *Design Automation Conference*, pp. 340-345.
23. Bisson, T. et al., 2007, "A Hybrid Disk-Aware Spin-Down Algorithm with I/O Subsystem Support," *IEEE International Performance, Computing and Commutating Conference*, pp. 236-245.
24. Parsons, B.S. and Pai, V.S., 2013, "A mathematical hard disk timing model for full system simulation," *IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 143-153.
25. Hylick, A. et al., 2008, "An Analysis of Hard Drive Energy Consumption," *IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, pp. 1-10.
26. Engel, M. et al., 2013, "Hard-disk equation of state: First-order liquid-hexatic transition in two

dimensions with three simulation methods," *Physical Review E 87,* Vol. 87, No. 4.

27. Liu, Z. et al., 2011, "MIND: A Black-Box Energy Consumption Model for Disk Arrays," *International Green Computing Conference and Workshops*, pp. 1-6.

28. Zedlewski, J. et al., 2003, "Modeling Hard-Disk Power Consumption," *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pp. 217-230.

29. DeRosa, P. et al., 2006, "Realism and simplicity: disk simulation for instructional OS performance evaluation," *Proceedings of the 37th SIGCSE technical symposium on Computer Science Education*, pp. 308-312.

30. Zhu, Q. et al. 2004, "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management," *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, pp. 118.

31. Lewis, A., Ghosh, S. and Tzeng, N.F., 2008, "Run-time Energy Consumption Estimation Based on Workload in Server Systems," *Proceedings of the 2008 conference on Power aware computing and systems*, pp. 4.

32. Zhang, Y, Gurumurthi, S. and Stan, R.M., 2007, "SODA: Sensitivity Based Optimization of Disk Architecture," *Proceedings of the 44th annual Design Automation Conference*, pp. 865-870.

33. Molaro, D., Payer, H. and Le Moal, D., 2009, "Tempo: Disk Drive Power Consumption Characterization and Modeling," *IEEE 13th International Symposium on Consumer Electronics*, pp. 246-250.

34. Hu, C., Jimenez, D.A. and Kremer, U., 2005, "Toward an evaluation infrastructure for power and energy optimizations," *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium*, pp. 230.

35. Kamil, S., Shalf, J. and Strohmaier, E., 2008, "Power efficiency in high performance computing," *IEEE International Symposium on Parallel and Distributed Processing*, pp. 1-8.

36. Giri, R.A. and Vanchi, A., 2010, "Increasing data center efficiency with server power measurements," *Intel Information Technology*.

37. Ge, R. et al., 2009, "Powerpack energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 21, No. 5, pp. 658-671.

38. Isci, C. and Martonosi, M., 2006, "Phase characterization for power: evaluating controlflow-based and event-counter-based techniques," *The Twelfth International Symposium on High-Performance Computer Architecture*, pp. 121-132.

39. Sara S., M. and Imtiaz, A., 2013, "A Green Model for Sustainable Software Engineering," *International Journal of Software Engineering and Its Applications*, Vol. 7, No. 4, pp. 55-74.

40. Morgan, R. and MacEachern, D., 2010, URL: https://support.hyperic.com/display/SIGAR/Home.