Beyond CPU: Considering Memory Power Consumption of Software

Hayri Acar[1], Gülfem I. Alptekin[2], Jean-Patrick Gelas[3] and Parisa Ghodous[1]
[1]*LIRIS, University of Lyon, Lyon, France*
[2]*Galatasaray University, Istanbul, Turkey*
[3]*ENS Lyon, LIP, UMR 5668, Lyon, France*

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

Abstract: ICTs (Information and Communication Technologies) are responsible around 2% of worldwide greenhouse gas emissions (Gartner, 2007). And according to the Intergovernmental Panel on Climate Change (IPPC) recent reports, CO2 emissions due to ICTs are increasing widely. For this reason, many works tried to propose various tools to estimate the energy consumption due to software in order to reduce carbon footprint. However, these studies, in the majority of cases, takes into account only the CPU and neglects all others components. Whereas, the trend towards high-density packaging and raised memory involve a great increased of power consumption caused by memory and maybe memory can become the largest power consumer in servers. In this paper, we model and then estimate the power consumed by CPU and memory due to the execution of a software. Thus, we perform several experiments in order to observe the behavior of each component.

1 INTRODUCTION

ICTs (Information and Communication Technologies) are responsible around 2% of worldwide greenhouse gas emissions (Gartner, 2007). And according to the Intergovernmental Panel on Climate Change (IPPC) recent reports, CO2 emissions due to ICTs are increasing widely. For this reason, many works tried to propose various tools to estimate the energy consumption due to software in order to reduce carbon footprint.

Since a few years, we have been able to find several research, on the web tools, (Power Supply Calculator, 2014), (eXtreme Power Supply Calculator, 2006), (Computer Power Consumption Calculator) that allow estimating the energy consumed by each component of a computer. Doing so, the user chooses the feature of the component and an estimation is given about related power consumption. However, this approach provides quite vague results so that a developer cannot use them as a guide when developing the software.

That is the reason of the appearance of other measurement means: Measurement of power consumption via hardware devices such as power meter or printed circuits (Kern et al., 2013); (Joseph et al., 2001); (Kamil et al., 2008). Using them, it is more possible to obtain accurate and efficient results for energy consumption. However, using these types of devices is complicated because it is necessary to have these devices and connect them to different components. What is more with this method, it is impossible to measure the energy consumed by virtual machines and applications on process.

In later years, a new methology has appeared which consists of estimating the energy consumed by a software based on mathematical formula established according to the characteristics of each components susceptible to consume power. But, these tools (Kansal et al., 2010); (Wang et al., 2011); (Noureddine et al., 2012), in the majority of cases, takes into account only the CPU and neglects all others components. Moreover, the trend towards high-density packaging and raised memory involve a great increase of power consumption caused by memory and maybe memory can become the largest power consumer in servers (Minas and Ellison, 2012).

In this paper, we will present a methodology to estimate the energy consumed by CPU and memory. Through different experiments we show the performance of the proposed methodology.

Acar, H., Alptekin, G., Gelas, J-P. and Ghodous, P.
Beyond CPU: Considering Memory Power Consumption of Software.
In *Proceedings of the 5th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS 2016)*, pages 417-424
ISBN: 978-989-758-184-7

2 CPU MODELIZATION

For a long time the CPU was considered the largest energy consumer component (Kim et al., 2014) in a computer. That is why, in each research work, the modelization of his structure has been taken into account to estimate the energy consumed by an computer program only.

Several factors contribute to the CPU power consumption and globally it is possible to give the following formula (1) in order to describe the power consumed by the CPU:

$$P_{CPU} = P_{CPU,dynamic} + P_{CPU,sc} + P_{CPU,leak} \quad (1)$$

where $P_{CPU,dynamic}$ represents dynamic power consumption, $P_{CPU,sc}$ corresponds to short-circuit power consumption and $P_{CPU,leak}$, power loss due to transistor leakage currents and varies with the temperature (Zapater et al., 2015). The last two power are due to at the hardware manufacturing. Hence, only the manufacturer can reduce the energy consumption due to hardware. So, it is possible to group this two power in order to obtain a static power on the equation (2):

$$P_{CPU,static} = P_{CPU,sc} + P_{CPU,leak} \quad (2)$$

Thus, it is possible to reformulate the equation (1) as follows (3):

$$P_{CPU} = P_{CPU,dynamic} + P_{CPU,static} \quad (3)$$

In our case, we want to reduce the energy consumed by software. For this, we take account only $P_{CPU,dynamic}$ to have more accurate and efficient results.

The CPU, like many integrated circuit, is a set of switches. So the main power consumption in CPU is due to charge and discharge of capacitors during computations that we can represent with the following figure 1:

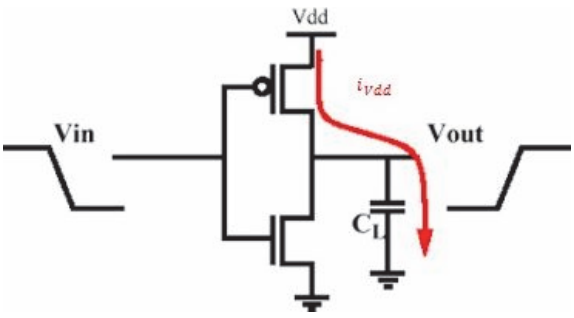

Figure 1: One switch in CPU.

The energy can be expressed (4) as follows:

$$E_{Vdd} = \int_0^{\infty} i_{Vdd}(t) \cdot V_{dd} \cdot dt \quad (4)$$

We also know that the current is given with the following expression (5):

$$i_{Vdd}(t) = C_L \cdot \frac{dv_{out}}{dt} \quad (5)$$

Thus, the expression (4) becomes (6):

$$E_{Vdd} = V_{dd} \cdot C_L \int_0^{\infty} \frac{dv_{out}}{dt} dt$$
$$E_{Vdd} = V_{dd}^2 \cdot C_L \quad (6)$$

We assume that in a switching cycle, there are low-to-high and high-to-low transition. So, we can obtain the power formulate (7) of this gate:

$$P = f \cdot V_{dd}^2 \cdot C_L \quad (7)$$

where f is the frequency.

For N gates, we must multiply the power by N. In a complex circuit the situation is more complicated, as not all the gates commute at the same frequency. Hence, we can define a parameter $\alpha < 1$ as the average fraction of gates that commute at every cycle.

Thus, the next expression of the power (8):

$$P = f \cdot V_{dd}^2 \cdot C_L \cdot N \cdot \alpha \quad (8)$$

By combining the constants as follows (9):

$$\beta = C_L \cdot N \cdot \alpha \quad (9)$$

we obtain (10):

$$P_{CPU,dynamic} = \beta \cdot f \cdot V_{dd}^2 \quad (10)$$

Moreover, we want to obtain the power consumed by the program. Thus, the percentage of the process Id (N_{id}) is multiplied with the previous expression (10) as follows (11):

$$P_{CPU,dynamic,id} = P_{CPU,dynamic} \cdot N_{id} \quad (11)$$

Thanks to these formulas, we can say that there are several ways to reduce the power consumption due to CPU:

Table 1: Possibilities to reduce power consumption of the CPU.

Solutions	Technics
Voltage reduction	Dual voltage CPUs Dynamic voltage scaling Overvolting/Undervolting
Frequency reduction	Underclocking Dynamic frequency scaling
Capacitance reduction	Integrated circuits

Dual voltage CPUs consist of uses a split-rail design to allow lower voltages to be used in the

processor core while the external Input/Output (I/O) voltages remain unchanged.

Dynamic voltage scaling: the voltage used is increased (Overvolting) or decreased (Undervolting) depending upon circumstances.

Underclocking: modify timing settings to run at a lower clock rate than is specified.

Dynamic frequency scaling: the frequency of a microprocessor can be automatically adjusted for saving energy.

Integrated circuits: replace PCB (Printed Circuit Board) traces between two chips.

So, we defined a mathematical formula in order to estimate the power consumed by the CPU. And, we noted the different ways to save energy.

Thus, should be limited to the energy consumption of the CPU or does it take into account other components whose energy consumption could be represent an importance compare to the CPU ? That is why, we will try to model the energy consumption due to Memory.

3 POWER CONSUMPTION OF DRAM

According to (Minas and Ellison, 2012), the power used on servers is increasing and the two largest consumers of power are the processor and the memory. Otherwise, several research works try to optimize systems to reduce DRAM power consumption:

- (Kang et al., 2010);
- (Hur and Lin, 2008);
- (Emma et al., 2008);
- (Zheng et al., 2008);
- (Vogelsang, 2010).

There are also some memory system simulator:

- DRAMSim2 (Rosenfeld et al., 2011);
- Cacti 5.1 (Thoziyoor et al., 2008);
- Micron System Power Calculator (Micron, 2007).

That is why, we choose to study the DRAM in order to model his power consumption. We need to use datasheet values from DRAM manufacturer to establish an expression to estimate the power.

As the CPU, we are interested only by the dynamic power consumed because we can only save energy in this part. Thus, based on (Micron, 2007) we assume that the dynamic power is composed of:

- Activate power;
- Precharge power;
- Read power;
- Write power.

To modelize these powers, we need to understand the functionality of a DDR3 SDRAM. The master operation is controlled by clock enable (CKE) that must be high to allow the DRAM to receive activate, precharge, read, and write commands. And in this situation, commands begin to propagate across the DRAM command decoders, and the activity rises the power consumption.

We regroup all the parameters that we will use to calculate the following powers in the table 2.

3.1 Activate Power

The first command sent to the DRAM, during normal working, is an activate command that chooses a bank and row address in order to allow a DDR3 SDRAM to read or write data. The data, that is stored in the cells of the chosen row, is then transferred from the array into the sense amplifiers. Then, the DRAM past in the active state. The precharge command restores the data from the sense amplifiers into the memory array and resets the bank for the next activate command. This leaves the bank in its precharge condition.

Thus, the following expression (12) can be used to estimate activate power:

$$\begin{aligned} P_{Activate} = {} & Psys(ACT\_PDN) \\ & + Psys(ACT\_STBY) \\ & + Psys(ACT) \end{aligned} \tag{12}$$

where:

$$\begin{aligned} Psys(ACT\_PDN) = {} & IDD3P * Vcc \\ & * BNK\_PRE \\ & * CKE\_LO\_ACT \\ & * (Vdd / Vcc)^2 \\ & * syst\_ck\_freq / 1000 \\ & * Tck\_used \end{aligned} \tag{13}$$

$$\begin{aligned} Psys(ACT\_STBY) = {} & IDD3N * Vcc * (1 \\ & - BNK\_PRE) * (1 \\ & - CKE\_LO\_ACT) \\ & * (Vdd / Vcc)^2 \\ & * syst\_ck\_freq / 1000 \\ & * Tck\_used \end{aligned} \tag{14}$$

$$\begin{aligned} Psys(ACT) = {} & (IDD0 - (IDD3N \\ & * tRAS / tRC + IDD2N \\ & * (tRC - tRAS) / tRC)) \\ & * Vcc * tRC / tRRDsch \\ & * (Vdd / Vcc)^2 \end{aligned} \tag{15}$$

So, activate power depends of many factors. Each term of these equations are summarized on the Table 2.

3.2 Precharge Power

Every activate command, that opens a row, have a precharge command, that closes the row, associated with it.

Precharge power can be formulated with the equation (16):

$$P_{Precharge} = Psys(PRE\_PDN) + Psys(PRE\_STBY) \quad (16)$$

where:

$$Psys(PRE\_PDN) = Idd2P * Vcc * BNK\_PRE * CKE\_LO\_PRE * (Vdd / Vcc)^2 * 1 \quad (17)$$

$$Psys(PRE\_STBY) = IDD2N * Vcc * BNK\_PRE * (1 - CKE\_LO\_PRE\%) * (Vdd / Vcc)^2 * syst\_ck\_freq / 1000 * Tck\_used \quad (18)$$

Precharge power depends also of several factors that are defined on the Table 2.

3.3 Read Power

During active state, data can be read from or written to the DDR3 SDRAM. A read command decodes a specific column address associated with the data that is stored in the sense amplifiers. The data from this column is driven across the I/O, gating to the internal read latch. From there, it is multiplexed onto the output drivers.

Read power can be expressed as follows (19):

$$P_{Read} = (IDD4R - IDD3N) * Vcc * 8 / Blength * RDsch * (Vdd / Vcc)^2 * syst\_ck\_freq / 1000 * Tck\_used \quad (19)$$

Each term of this equation is also described on the Table 2.

3.4 Write Power

The power needed for a write data is similar to the read data except the data propagates in the opposite direction. Data from the DQ pins is latched into the data receivers/registers and is transferred to the internal data drivers that transmit the data to the sense amplifiers across the I/O gating and into the decoded column address location.

Write power is defined with (20):

$$P_{Write} = (IDD4W - IDD3N) * Vcc * 8 / Blength * WRsch * (Vdd / Vcc)^2 * syst\_ck\_freq / 1000 * Tck\_used \quad (20)$$

Each parameter of this formula is also expressed on the Table 2.

3.5 DRAM Total Power

DRAM total power (21) is obtained by summing all the equations (12), (16), (19) and (20) of powers defined in the preceding paragraphs.

$$P_{DRAM} = P_{Activate} + P_{Precharge} + P_{Read} + P_{Write} \quad (21)$$

Moreover, we want to calculate the power consumed by the application. That is why, the usage percent of the process Id (M_{id}) is multiplied with the previous expression (21) as follows (22):

$$P_{DRAM,id} = P_{DRAM} . M_{id} \quad (22)$$

Table 2: Data sheet specifications.

Parameter	Description
Idd2P	Precharge power-down current
Vcc	Voltage
BNK_PRE	The percentage of time that all banks on the DRAM are in a precharged state
CKE_LO_PRE	Percentage of the all bank precharge time for which CKE is held LOW
Vdd	System VDD
IDD2N	Precharge standby current
syst_ck_freq	System CK frequency
Tck_used	Used for current measurements
IDD3P	Active power-down current
CKE_LO_ACT	Percentage of the at least one bank active time for which CKE is held LOW
IDD3N	Active standby current
IDD0	Operating current: One bank active-precharge
tRAS	Used for IDD0 calculation
tRC	Activate-to-activate timing
tRRDsch	The average time between ACT commands to this DRAM
IDD4W	Operating burst write current
Blength	Burst length
WRsch	The percentage of clock cycles which are inputting write data to the DRAM
IDD4R	Operating burst read current
RDsch	The percentage of clock cycles which are outputting read data from the DRAM

Thus, we established also the relation allowing us to estimate the power consumed by DRAM. Hence, we implemented a tool and realize some experiments in order to see the behavior of DRAM compare to CPU.

4 EXPERIMENTS

4.1 Devices Used

We used the laptop ASUS model N751J composed of a CPU Intel Core i7-4710HQ (2.5GHz) and a RAM 16 Go (2 * 8 Go) DDR3 1600 MHz.

To run tests, we developed a tool TEEC (Tool to Estimate Energy Consumption), whose model is shown in Figure 2, in Java programing language because depending on (Noureddine, 2012) Java represent the language with the least power consumption during compilation and execution steps in default parameter settings of the compiler. In this tool TEEC, we use Sigar library (Morgan and MacEachern, 2010) in order to get information about the CPU and the RAM. Moreover, we use also the parameter provides by manufacturers. And, Java Agents allows us to the instrumentation capabilities to an application.

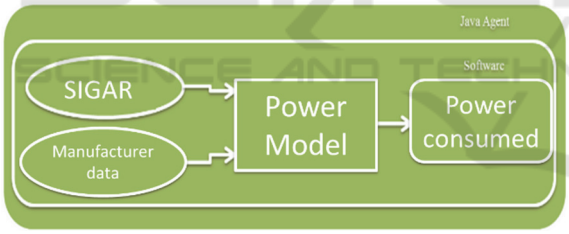

Figure 2: Model of our tool TEEC.

Thus, using TEEC, we realized several different tests in order to observe the variation of the power consumption due to the CPU and the memory and compare them.

4.2 Source Code Adjustment

Based on (Kambadur and Kim, 2014), we realize the following tests in order to see the impacts of source code on CPU and memory power consumption.

4.2.1 Strength Reduction

Strength reduction consists of replacing an operation by a similar operation. The most common example of strength reduction is using the shift operator to multiply and divide. For instance, `a >> 2` can be used in place of `a / 4`, and `a << 1` replaces `a * 2`.

In our case in order to see the behavior of this replacement, we execute the same operation several time (here: 50000 repetitions). So, we can observe the results on the Figures 3a and 3b.

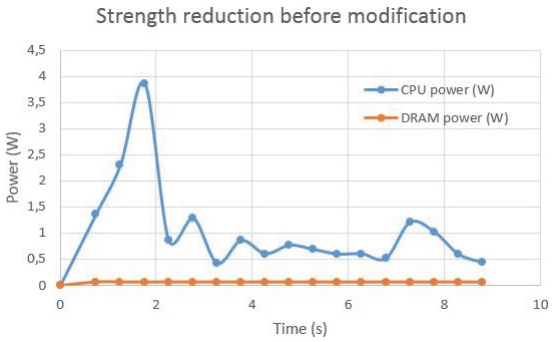

Figure 3a: Strength reduction unoptimized.

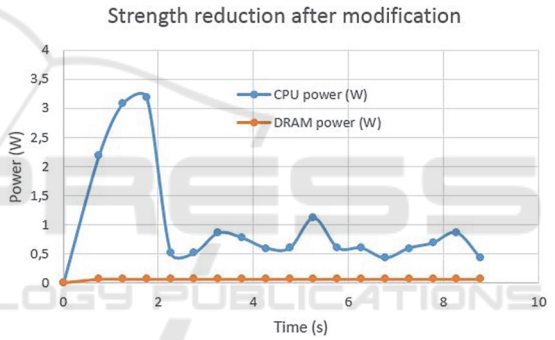

Figure 3b: Strength reduction optimized.

For this test, we observe that the DRAM power consumption remains constant in the two cases and the time elapsed is similar. The CPU power consumption is less important after the strength reduction. This show, the impact of the code source on the CPU power consumption. Moreover, in the two cases, we observe that the CPU power varies and sometimes these values are close to DRAM values. Thus, we can say that the DRAM power consumption is not always neglected in front of CPU power consumption.

4.2.2 Eliminate Common Subexpressions

To remove redundant calculation, we eliminate common subexpressions. This part of code:

```
double a = c * (d / e) * f;
double b = c * (d / e) * g;
```

can be rewritten as:

```
double h = c * (d / e);
double a = h * f;
double b = h * g;
```

We run test in a loop of 50000 repetitions to observe the variation of power. The results are in Figure 4a and 4b.

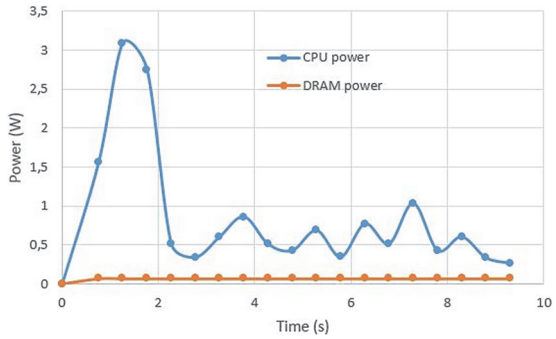

Figure 4a: Subexpression unoptimized.

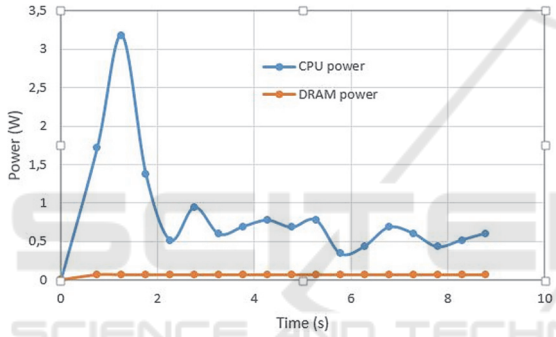

Figure 4b: Subexpression optimized.

In this test, the results show that the CPU and the DRAM power consumption and the elapsed time in the two cases are quite similar. However, we note that the CPU power consumption vary and several times is more close to DRAM power consumption.

4.2.3 Code Motion

Code motion moves code that calculates an expression whose result doesn't change. This is most common with loops, but it can also involve code repeated on each invocation of a method. For example:

```
for (int i = 0; i < a.length; ++i)
 a[i] *= Math.PI * Math.cos(b);
```

becomes:

```
double pico = Math.PI* Math.cos(b);
for (int i = 0; i < a.length; i++)
 a[i] *= pico;
```

The results of this test is represented on the Figures 5a and 5b.

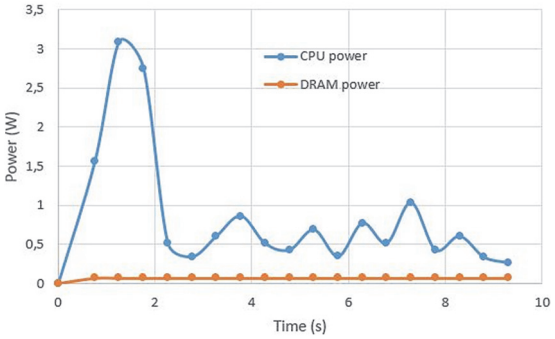

Figure 5a: Code motion unoptimized.

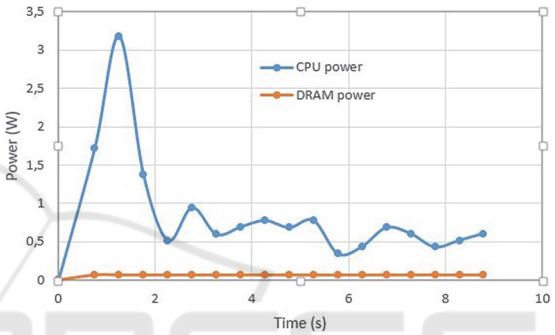

Figure 5b: Code motion optimized.

This test show that in the unoptimized code motion, the time elapsed is slightly greater than optimized code. CPU and DRAM power consumption are quite similar in the two cases. And, sometimes CPU power consumption curve approaches DRAM power consumption curve.

4.2.4 Unrolling Loops

Unrolling loops reduces the number of loop control code by performing more than one operation each time in the loop, and consequently running fewer iterations. With the previous example, if the length of the table a is always a multiple of two, the loop can be rewrite like:

```
double pico = Math.PI* Math.cos(b);
for (int i = 0; i < a.length; i += 2) {
 a[i] *= pico;
 a[i+1] *= pico;
}
```

Figure 6 shows the power consumption of CPU and DRAM depending on the time.

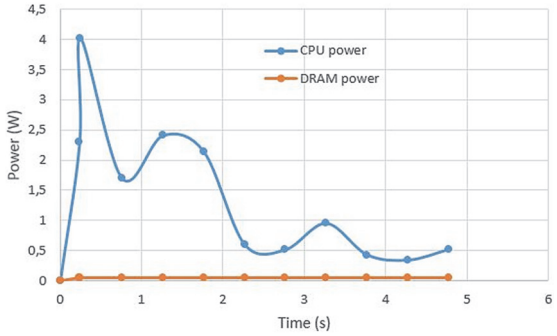

Figure 6: Unrolling loops.

Compare to the Figure 5b, in this case, we observe that at the beginning of the curve, the CPU consumes more power during some time than code motion and then becomes similar. But, in unrolling loops case, the total execution elapsed time is the half of the code motion case. And at the end of the curve in Figure 6, the CPU power is less important than the curve in code motion (Figure 5b). Moreover, in this test, the difference between CPU and DRAM power consumption is less important than code motion case.

Thus, the results reveal that the unrolling loops method is quicker and consumes less CPU power than the code motion method.

5 CONCLUSIONS

A modelization of the CPU and the DRAM has been made in order to understand the behavior and the functionality of each component. Thanks to this model, several mathematical formulas have been established to estimate the power consumption due to each part of each component. Thus, based on this methodology, a tool that allow to measure the power consumed by CPU and DRAM has been implemented and named TEEC (Tool to Estimate Energy Consumption). This tool gives accurate and efficient information about CPU and DRAM power consumption, has been used to perform some experiments. The goal of these tests was to observe the impact of the code source of an application in the power consumption. These experiments have provided several results.

When the code source is optimized, it is possible to reduce the power consumption due to CPU. But, the DRAM power consumption remains quite constant.

Sometimes, it is possible to save energy with an optimization of the code by reducing execution time of an application.

In several cases, after some time of execution, CPU power consumption remains the main energy consumer. However, the DRAM power consumption can't be neglected.

Moreover, some code optimizations don't make any real impact on the CPU and DRAM power consumption.

The contribution to power measurement literature will continue by bringing improvement to the estimation of the consumption of other components; such as, disk and network in order to observe their impact. It will allow us to have a higher accuracy in estimating the energy consumption of a program.

The proposed tool TEEC is expected to be improved, and it is planned to dynamically identifying locations where code consume the largest power. This will allow developers to optimize their own codes to obtain green and sustainable software.

REFERENCES

Gartner, Green IT: The New Industry Shock Wave, Gartner, Presentation at Symposium/ITXPO Conference, 2007.

Power Supply Calculator, February 2014. URL: *http://powersupplycalculator.net/.*

eXtreme Power Supply Calculator, January 2006. URL: *http://outervision.com/power-supply-calculator.*

Computer Power Consumption Calculator. URL: *http://www.matthewb.id.au/power/computer_power_consumption_calculator.html.*

Kern, E., Dick, M., Naumann, S., Guldner, A., Johann, T., 2013. Green software and green software engineering–definitions, measurements, and quality aspects. *ICT4S 2013: Proceedings of the First International Conference on Information and Communication Technologies for Sustainability.*

Joseph, R., Brooks, D., Martonosi, M., 2001. Live, runtime power measurements as a foundation for evaluating power/performance tradeoffs. *Workshop on Complexity Effective Design WCED, held in conjunction with ISCA-28.*

Kamil, S., Shalf, J., Strohmaier, E., 2008. Power efficiency in high performance computing. *IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008.*

Kansal, A., Zhao, F., Liu, J., Kothari, N., Bhattacharya, A., 2010. Virtual Machine Power Metering and Provisioning. *ACM Symposium on Cloud Computing (SOCC).*

Wang, S., Chen, H., Shi, W., 2011. SPAN: A software power analyzer for multicore computer systems. *Sustainable Computing: Informatics and Systems*, Volume 1, Issue 1.

Noureddine, A., Bourdon, A., Rouvoy, R., Seinturier, L., 2012. A Preliminary Study of the Impact of Software Engineering on GreenIT. *First International Workshop*

on Green and Sustainable Software.

Kim, M., Ju, Y., Chae, J., Park, M., 2014. A Simple Model for Estimating Power Consumption of a Multicore Server System. *International Journal of Multimedia and Ubiquitos Engineering.*

Zapater, M. et al., 2015. Leakage-Aware Cooling Management for Improving Server Energy Efficiency. IEEE Trans. Parallel Distrib. Syst. 26(10): 2764-2777.

Minas, L., Ellison, B., 2012. The Problem of Power Consumption in Servers. *Intel Press.*

Kang, U. et al., 2010. 8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology. *Journal of Solid-State Circuits.*

Hur, I. and Lin, C., 2008. A comprehensive approach to DRAM power management. *International Symposium on High Performance Computer Architecture.*

Emma, P., Reohr, W. and Meterelliyoz, M., 2008. Rethinking Refresh: Increasing Availability and Reducing Power in DRAM for Cache Applications. *IEEE Micro.*

Zheng, H. et al., 2008. Mini-Rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency. *Proceedings of Micro.*

Vogelsang, T., 2010. Understanding the energy consumption of dynamic random access memories. *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture.*

Rosenfeld, P., Cooper-Balis, E., Jacob, B., 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *CAL.*

Thoziyoor, S., Muralimanohar, N., Ahn, J., Jouppi, N., 2008. CACTI 5.1. *HP Laboratories Palo Alto.*

Micron Technologies Inc. System Power Calculator. URL: http://www.micron.com/support/power-calc.

Micron, 2007. Calculating Memory System Power for DDR3.

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

Kambadur, M., Kim, M.A., 2014. An experimental survey of energy management across the stack. *ACM International Conference on Object Oriented Programming Systems Languages & Applications.*