JKAU: Comp. IT. Sci., **Vol**. **8** No. 1, pp: 11 – 18 (1440 A.H. / 2019 A.D.)
Doi: 10.4197/Comp. 8-1.2

Build Power Profiling Tool for Modern CPUs

Naif Aljabri and **Osama Abulnaja**

Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

naljabry0001@stu.kau.edu.sa

Abstract. Reduce the application power consumption is one of the main challenges for the HPC community. Code power profilers are very important for researchers to identify the performance bottlenecks and power consumption for their code. Most of the modern CPUs are equipped with a built-in sensor to allow researchers and HPC engineers to estimate the power consumption of the running applications. To estimate the power consumption for any piece of code running on CPU, you need to eliminate the confounding factors as possible and run the code many times until the average converge. The reason for that is the environment, which has the OS and other processes and services running at the same time with your code and may report incorrect power readings. In this paper, we build a power profiler tool, which saves the researcher time by running and profiling different pieces of code with different types of workloads, and keeps running until the average converge. Furthermore, we identify and eliminate the environment confounding factor which saves the researcher time and gives a realistic result for power consumption experiments.

Keywords: Power, Profiler, CPU, Average converge, Confounding factors.

1. Introduction

Power consumption is the main obstacle to scale the current HPC petascale computing power to reach the next exascale generation. The most powerful supercomputer in the world consumes about 15 megawatts of power [1]. To reach the next generation of exascale computing power in reasonable power budget, we need a lot of effort from architecture designers and software programmers. Most of the modern CPUs are equipped with power sensors to allow the programmers to estimate their code power consumption.

With code profilers, we can identify the power efficient code building blocks and algorithms and use it to build a power efficient HPC applications. CPU environment is very complicated for power estimation experiments. The OS and other processes are running side-by-side with your code which may affect the accuracy of the results. One of the best practices in power estimations experiments to avoid the environment noise is to run your code many times until the average converge. It's very difficult to estimate how many times you need to run your code to reach the average converge. We need to run the code hundreds of times and calculate the accumulated average for each run. Furthermore we need to eliminate as possible the environment confounding factors. Without eliminating these factors, the experiment process will take a lot of time from the researcher to reach the estimated average.

In this paper, we propose a power profiler tool which automatically runs many pieces of code until the average converge. The proposed profiler can deal with any number of code blocks or algorithms automatically. It is select the code block or algorithm from the queue, then start many sessions and read the power sensors and calculate the accumulated average for each session. The profiler capable to run any number of sessions automatically until the average converges for each code block or algorithm. All the data saved in memory during the run and after the session terminated all the data saved in the database for reference. The profiler will stop the experiments automatically when reach the average converge and starts the next code block or algorithm in the queue. Furthermore, we identify and propose several environment confounding factors and how to eliminate these factors to avoid nose in the experiment results.

This paper is organized as follows. Section II briefly lists the related work. In Section III, the proposed profiler is given, followed by the environment confounding factors in Section IV. The results and discussion are given in Section V. Finally, the conclusion and futures works are given in Section VI.

2. Related Work

There are many profilers used by researchers to estimate the power consumption in the CPU environment. Some of these profiles are system-wide profilers (not for a specific piece of code). Also, some of them can be used to profile a piece of code but it doesn't support the automated sessions, running until the average converge or deal with many pieces of code. Here we list some of these profiles.

Intel VTune Amplifier [2], is a commercial performance profiling tool. It's developed by Intel and supports most of the modern Intel processors. It supports the analysis of function, instruction or source code. It supports also the energy profiling for system-wide.

Oprofile [3], it's a kernel based profiler that profiles Linux applications. The samples are taken at fixed interval time. Oprofile is defaulted to configure because it requires a copy of the source code for the exact Linux kernel.

Qprof [4], it's Linux based profiler that collects the samples in real time for any exe file. The user must configure a number of environment variables for sampling. It can support function or instruction level profiling.

Perfsuite [5], is a Linux based profiler used to profile Linux applications using the psrun command. Its use specific API to perform sampling for the application. After the end of executions all the profile data saved in XML file.

Gprof [6], is a cross-platform profiler for function level sampling. It can collect a wide range of functions based metrics. Its use the augmenting compiler component to insert the monitoring functions into the targeted application.

3. Proposed Profiler

We developed our own profiler to perform power consumption experiments for any code block or algorithm running on modern multicore CPUs. The main features of our profiler are:

- Deal with any a number of exe files, just compile your code blocks or algorithms, then pass all the exe file names and location to the profiler.
- Select the number of experiments you want to perform or each exe file.
- For each experiment, the profiler will run each exe file many times and collect the power sensors readings periodically and

calculate the accumulative average until average converge for two decimal places, then write the results to the database.

- Each round of run and each experiment are identified by a unique number, to be easy for the researcher to filter and classify the results to perform statistical calculations on the data.

- In case of the CPU temperature increased due to heavy utilization of CPU, the profiler will hold the next run and wait for the CPU temperature to return to the normal state then resume the next run. This very important feature because if the CPU temperature increased, the power consumption readings will increase and give unrealistic results. The researcher can define the normal temperature degree in the settings of the profiler before starts the experiments. The normal temperature is different from CPU to other based on the ambient temperature or the CPU manufactural specifications.

- Shows live results for each run and the progress of the experiment during the profiling process.

- Shows live sensors reading for CPU type and model, CPU temperature, voltage, power, load and clock cycles. Also, its calculate the execution time to be used to calculate the energy consumption for any exe file. All these data also saved in the database. The researcher may need this information when running the experiments on more than CPU type to know what is the data belongs to each CPU type.

We developed our profiler using Visual Studio C# 2010 under Windows 7 64-bit environment. To run each exe file session we use the *BackgroundWorker* tool. The *backgroundWorker1_DoWork()* function used to run the exe file. In the same time, we run the *Timer()* function to read the CPU power sensor periodically during the run. Once the run finished, the *BackgroundWorker* tool invokes the *backgroundWorker1_RunWorkerCompleted()* function and stop the sensors reading and write all the readings to the database. The average power consumption for the exe file saved in the averages array. The profiler keeps running the same exe file many times until the power average converges in the averages array. Finally, the profiler writes all the averages array to the database. Also, we give the user the ability to give the experiment name and the experiment number in the profile settings before the start the experiments. All this information saved in the database when the profiler writes the samples and the power averages in the database. We don't need to create a new database for each experiment. The user can differentiate between the experiments names, numbers and CPU type from the database.

We use the Open Hardware Monitor library to read the CPU sensors periodically. Open Hardware Monitor library is an open source library that supports most hardware monitoring chips found on today's mainboards [7]. Figures 1&2 show how the proposed profiler works.

4. Environment Confounding Factors

Profiling experiments in modern multicore CPUs are very complicated. There are many processes and services running at the same time with your code during the profiling process and affect your result by incorrect values from the sensors. These confounding factors must be eliminated as possible to give realistic sensors values. In this section, we will list the confounding factors in the modern multicore CPUs and how to eliminate them as possible. Eliminating these factors will give more realistic and save days of the running of your code to reach the average converge. At the beginning of our experiments we spent

weeks to reach the average converge for the average power consumption, and then after months of research about the CPU environment we successfully eliminated most of these factors and save weeks of time and most of our experiments done in the same day.

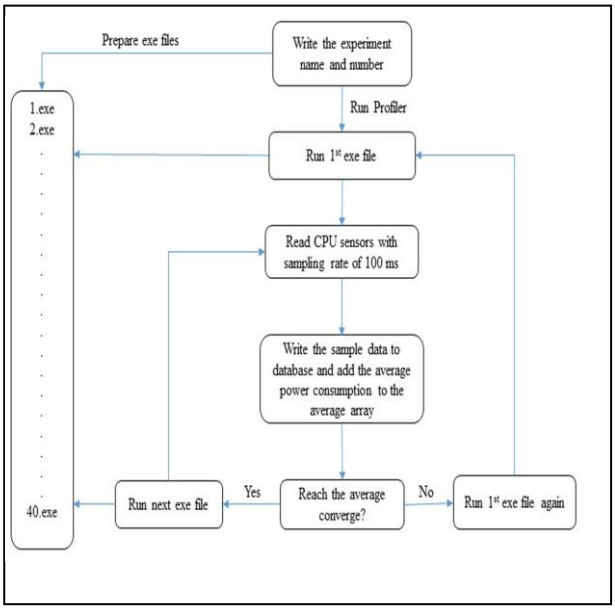

Fig. 1. The flowchart of the proposed pofiler.

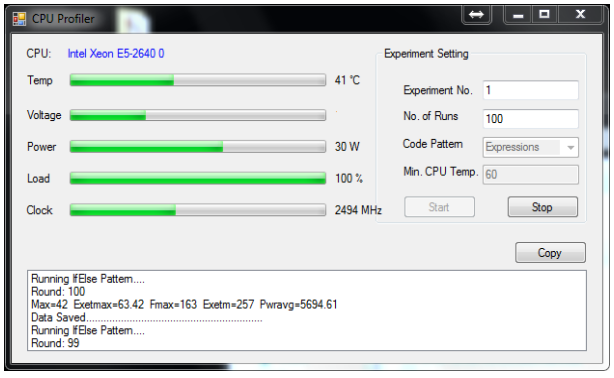

Fig. 2. The proposed profiler picture.

A. CPU Power Management

This is one of the new features in the modern multicore CPU. Every CPU equipped with a voltage regulator and if CPU utilization increased power management tool will decrease the CPU clock cycles and reduce the power through the voltage regulator. In this case, you will read incorrect power values for your code. We need to disable this feature from the BIOS before you run your code to avoid incorrect power values [8].

B. Hyper-Threading

The feature of hyperthreading allows each physical core to work as two virtual core to the OS. There are duplicated components with each physical core to store the program stat and switch to another program for a slice of time. During the run of your program – if the hyperthreading feature is enabled – the CPU sensor will read the power values of your program and any other program running on the two virtual cores which may add noise to your results. You need to disable this feature to deal with one physical core and read the correct power values for your program [9].

C. Context Switching

The context switching is not a new feature in the modern CPUs. It's also used in the old CPUs. The idea of this feature is to allow to programs to share the time with any physical core. For example, if your program is running on core 1 after some time it may switch to another physical core and complete the running. We can eliminate the context switching by using the Set Affinity command. This command allows you to choose one physical core to run your program and prevent your program from switching to any other physical core. Also with this command, we can set the affinity of any program or process running on the CPU to work on a set of cores for example core 3-8 and keep core 1 for your program without any interrupt from any other program or process to use your program core. This will give your results a perfect accuracy and reduce the number of runs to reach the average converge of the power averages [9].

D. Neighbor Core Effect

The effect of the neighbor physical core is validated by many researchers [10]. The neighbor core temperature may affect the core

temperature and increase the temperature of the core and increase the time of the experiment. Our profiler is designed to repeat the run in normal core temperature if the core temperature exceeded the normal temperature, the profiler will hold the next run and resume after the core gets cold and reach the normal temperature. Our experiments if we use core 1 for our program we keep core 2 always idle by using the Set Affinity command. For example, if we have CPU with 8 physical cores, we set the affinity of our program to core 1 and set the affinity for other programs and processes to core 3-8 and keep core 2 idle.

E. OS Processes and Services

In the CPU environment, there are hundreds of processes and services running to operate the OS functions. All these processes and services use all the CPU cores and may affect the reading of power values from the sensors. We can set the affinity to all these services and processes to use the other cores that affect our program running. You can't set the affinity for all the OS processes and services because some of these processes and services are protected by the OS and need some privilege to set their affinity. But at least you can eliminate most of them and move them to other cores.

F. Program Threads

In some cases, your program may contain many threads. Even if you set the affinity of your program to run in a specific core, your program threads may move to other cores during the run. You need to use the command Set Thread affinity before your run your exe file to avoid incorrect readings for your program even if you set the affinity of your exe file to run on a specific core.

G. Data Load Effect

If your program work with a large data set size the best practice for the programmers is to separate them in another file to keep your exe file small and easy to run. After your exe file loaded in the memory, it will read the data from the data file and load it to the memory. This operation consumes a lot of power from the CPU. Data movement from Disk to memory based on many studies is the most power consumption operation in the modern CPU environment [11]. In this case, if your data are separate, when your exe file starts running by the profiler it will read large power values and add some noise to your results. The best way to avoid this large power values during your experiments is to embed your data inside your exe file. Your exe file becomes larger but the OS will take the function of loading the data to the memory before the profiler starts reading the power values from the sensors. Your exe file will not be ready for running before the OS load all the data to the memory.

H. Compiler Optimization Effect

To make your program more efficient when running on some CPU types, the compiler may translate some of your code blocks to different statements. For example, if you want to measure the power consumption for *For Loop Statement*. The profiler, in some cases, changes your For Loop statement to sequential code. This operation called loop unrolling. In this case, during the run of your exe file, the profiler will read the power values of sequential statements instead of your original For Loop statement and give untrusted results. You need to disable the compiler optimization option before you compile your exe file to avoid this issue.

I. Ambient Temperature

Ambient temperature is the air temperature inside the lab used to perform the power consumption experiments. We need to maintain the appropriate ambient temperature for the lab during the run of the power profiling experiments. Keep the lab cold as much as

possible to keep the CPU in normal temperature when performing the workload. If the air temperature is not cold enough to keep the CPU and other components at normal temperature, the experiments will take a very long time to finish [12].

All the above confounding factors will give a very ideal environment for the power profiling experiments, give more realistic results and reduce the experiment time. We use the proposed profiler and these environment setting to obtain the results for some algorithms power consumption estimation for two published papers [13],[14].

5. Results and Discussion

We use the proposed profiler and the environment setting to measure the average power consumption for the generic mergesort algorithm and the optimized 3-way partitioning quicksort algorithms. The dataset size for the two algorithms is one million integer numbers. The dataset numbers were generated randomly using the *rand() % RAND_MAX;* function. The same dataset was used for both algorithms and embedded inside the exe file. We use the HPC machine FUJITSU CELSIUS M720 with Sandy Bridge architecture Intel Xeon CPU E5-2640 Chip (2.50 GHz, 6 Physical Cores). The OS is Microsoft Windows 7 Professional x64-based and the Physical Memory 8.00 GB. We use the algorithms code in [15]. The code was written using C++ language and compiled using GCC 64-bit compiler. We disable the compiler optimization options and we set the affinity for the program and thread to core number 0 and set the affinity of the other OS processes and services to core 2-5 and we keep the neighbor core 1 idle to avoid the temperature effect to core 0. The ambient temperature in average was 20 °C. We pass the two exe files to the proposed profiler and obtained the power consumption results for each algorithm. The average converge was detected in less than 300 runs. Table 1 shows the obtained average power consumption or each algorithm.

Table 1. Algorithms average power consumption.

Dataset Size	Average Power (Watt)		Average Time (ms)	
	Mergesort	***Quicksort***	***Mergesort***	***Quicksort***
1 M	11.92	13.26	12.41	10.93

We notice from the table that mergesort consumes less power than the optimized 3-way partitioning quicksort algorithm. In terms of speed, quicksort was faster than mergesort. During the history of algorithms, most of the algorithms researchers and developers optimize the algorithms for speed. This optimization adds more complexity to the algorithms, which makes the algorithm finish faster but consume more power. One of the main obstacles facing the HPC community these days is power consumption. The experiments of power consumption for algorithms and code blocks are very important to identify the efficient algorithms and code blocks for HPC in terms of power consumption.

6. Conclusions and Future Work

In this paper, we proposed a power profiler for power consumptions experiments in modern multicore CPU environment. The profiler has many features that make the perpetration and collecting of results easier for the researches. We also identify the power consumption experiments confounding factors and how to eliminate these factors to give accurate and realistic power values from the CPU sensors. The optimization of algorithms and code blocks makes them faster and complex. This complexity will affect the algorithms and code blocks power consumption in many cases.

The main objective of this study is to build a CPU power profiler tool which maintains the ideal environment to improve the

accuracy of algorithms and code block power consumption estimation.

We improve the environment by eliminating most of the confounding factors in power estimation experiments to give more accurate and realistic power values from the CPU sensor.

In the future, we will continue improve the profiler tool and identifying more confound factors in other environments like Linux. Also, we will use the new generation of power efficient Intel CPUs architectures like Haswell, Broadwell and Skylake Chips or customizable chips like FPGA (Field Programmable Gate Array), to evaluate many algorithms and code blocks power consumption.

Acknowledgment

This work has been supported by the Deanship of Scientific Research (DSR), King Abdulaziz University (KAU) under grant No. 1-611-1433/HiCi.

References

[1] **Top500.org.** (2018). *November 2017 list of TOP500* Supercomputer Sites. [online] Available at: https://www.top500.org/lists/2017/11/ [Accessed 22 Apr. 2018].

[2] **Software.intel.com**. (2018). *Documentation for Intel® VTune™ Amplifier XE | Intel® Software*. [online] Available at: https://software.intel.com/en-us/intel-vtune-amplifier-xe-support/documentation [Accessed 22 Apr. 2018].

[3] **Oprofile.sourceforge.net.** (2018). *OProfile manual*. [online] Available at: http://oprofile.sourceforge.net/doc/index.html [Accessed 22 Apr. 2018].

[4] *SourceForge.* (2018). QProf. [online] Available at: https://sourceforge.net/projects/qprof/ [Accessed 22 Apr. 2018].

[5] **Perfsuite.ncsa.illinois.edu.** (2018). *PerfSuite*. [online] Available at: http://perfsuite.ncsa.illinois.edu/ [Accessed 22 Apr. 2018].

[6] **Sourceware.org.** (2018). *GNU gprof: Top.* [online] Available at: http://sourceware.org/binutils/docs/gprof/index.html [Accessed 22 Apr. 2018].

[7] **Openhardwaremonitor.org.** (2018). *Open Hardware Monitor.* [online] Available at: http://openhardwaremonitor.org [Accessed 22 Apr. 2018].

[8] **David, H., Gorbatov, E., Hanebutte, U., Khanna, R.** and **Le, C.,** "RAPL: Memory power estimation and capping." In: *ACM/IEEE International Symposium on Low-Power Electronics and Design*, pp:. 189{194, 2010.

[9] **Zecena, I.,** "*Energy consumption analysis of parallel algorithms running on multicore systems and GPUs*", *MSc*., Texas State University-San Marcos, p1, August 2013.

[10] **Liu, G., Fan, M.** and **Quan, G.,** "Neighbor-aware dynamic thermal management for multi-core platform," In: *Proceedings of the Conference on Design, Automation, and Test in Europe*, pp. 187–192, 2012.

[11] **David, H. Gorbatov, E., Hanebutte, U., Khanna, R.** and **Le, C.** "RAPL: Memory power estimation and capping." In: *ACM/IEEE International Symposium on Low-Power Electronics and Design*, pp: 189-194, 2010.

[12] **Aliaga, J. I., Barreda, M., Dolz, M. F., Martín, A. F., Mayo, R.** and **Quintana-Ortí, E. S.,** "Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems," *Clust. Comput.*, **17**(4) : 1335–1348, 2014.

[13] **Al-Hashimi, M., Saleh, M., Abulnaja, O.** and **Aljabri, N.,** "Evaluation of control loop statements power efficiency: An experimental study", *Informatics and Systems (INFOS), 2014 9th International Conference on, Cairo*, pp: 45-48 2014.

[14] **Al-Hashimi, M., Saleh, M., Abulnaja, O.** and **Aljabri, N.,** "On the power characteristics of mergesort: An empirical study," In: *Advanced Control Circuits Systems (ACCS) Systems & 2017 Intl Conf on New Paradigms in Electronics & Information Technology (PEIT), 2017 Intl Conf on. IEEE*, pp: 172–178, 2017.

[15] **Sedgewick, R.** and **Wayne, K.,** *Algorithms*, 4th ed. Addison-Wesley Professional, 2011.

بناء أداة قياس استهلاك الطاقة لوحدات المعالجة المركزية الحديثة

نايف الجابري و **أسامة ابو النجا**

كلية الحاسبات وتقنية المعلومات، جامعة الملك عبدالعزيز، جدة، المملكة العربية السعودية

arushdi@kau.edu.sa

المستخلص. يعتبر تقليل استهلاك الطاقة في التطبيقات أحد أهم التحديات الرئيسة لمجتمع HPC. قياس استهلاك الطاقة للشيفرات البرمجية يعتبر مهم للغاية للباحثين لتحديد اختناقات الأداء واستهلاك الطاقة للشيفرة البرمجية. حديثا، تم تجهيز معظم وحدات المعالجة المركزية بجهاز استشعار مدمج للسماح للباحثين ومهندسي HPC بتقدير استهلاك الطاقة للتطبيقات قيد التشغيل. لتقدير استهلاك الطاقة لأي جزء من الشفرة البرمجية يعمل على وحدة المعالجة المركزية، تحتاج إلى التخلص من عوامل التشويش قدر الإمكان وتشغيل الشفرة عدة مرات حتى يستقر متوسط القراءات. والسبب في ذلك هو البيئة التي بها نظام التشغيل والعمليات والخدمات الأخرى التي تعمل في نفس الوقت مع الشيفرة البرمجية الخاصة بك، وقد تسبب قراءات طاقة غير صحيحة. في هذه الورقة، نقوم ببناء أداة قياس استهلاك الطاقة التي توفر وقت الباحث من خلال تشغيل وقياس استهلاك الطاقة لشيفرات برمجية على أنواع مختلفة من حجم العمل إلى أن يستقر متوسط القراءة. علاوة على ذلك، نقوم بتحديد وإزالة عوامل التشويش على التجربة، مما يوفر وقت الباحث، ويعطي نتيجة واقعية لتجارب استهلاك الطاقة.

الكلمات المفتاحية: الطاقة، قياس الاستهلاك، وحدة المعالجة المركزية، متوسط التقارب، عوامل التشويش.