International Journal of Innovative
Computing, Information and Control
Volume **13**, Number **6**, December **2017**

ICIC International ©2017 ISSN 1349-4198
pp. **1993–2008**

LEVERAGING ACPI _PSS DATA TO ESTIMATE ENERGY OF PROCESSOR CORE BETWEEN ACPI-CPUFREQ AND INTEL P-STATE DRIVER ON LOW-END INTEL PENTIUM CELERON N2830

Adnan[1], Andani Achmad[1] and Lompo Ramos Emakarim[2]

[1]Department of Electrical Engineering
University of Hasanuddin
Makassar 90245, Sulawesi Selatan, Indonesia
{ adnan; andani }@unhas.ac.id

[2]Global Support Delivery
Hewlett Packard Enterprise
Casablanca Raya Kav. 88, Jakarta 12870, Indonesia
lompo.ramos@hpe.com

Received October 2016; revised April 2017

Abstract. *Measuring energy of processor core which is consumed by a program is a prerequisite when we want to optimize source code of a program in order to gain efficiency of energy. However, it is difficult to find tools to measure energy of processor core especially on low-end processor. In this paper we offer a simple energy estimation as an alternative tool by leveraging an object in ACPI (Advanced Configuration and Power Interface) specification called _PSS (Performance Supported States) which contains information about power of each processor core in each state of its performance states (P-States). To estimate the energy of processor core, we integrate power of each processor core during run time of a program. A program was run repetitively in different conditions on minicomputer Intel DN2820FYK which consists of low-end processor Intel Pentium Celeron N2830. We collected frequency of cores while running the program. We then mapped the frequency into power based on _PSS information to construct graph of power vs. time as a base to estimate the energy by using trapezoid method of integration. Since the processor behavior is also determined by a driver, we compared the effect of drivers between driver ACPI-cpufreq and driver Intel P-State toward energy consumed. We also made some experiments by implementing combinations of policy, frequency, feature of boosts and load. In this paper, we find out that by leveraging _PSS data, we can easily estimate energy of processor core. We also find out that by using driver Intel P-State, it will be better and easier to estimate energy rather than using ACPI-cpufreq.*
Keywords: Energy, Simple, Core, _PSS, Low-end N2830

1. **Introduction.** Efficiency of energy consumed in a computer system can be gained by optimizing source code of a program running in the computer. In many cases, processor core consumes most of a computer's energy. However, measuring how much energy consumed by a processor core in order to compare the energy before and after source code optimization is still difficult especially on low-end processor. Here, we offer a simple method of estimating energy of processor core by leveraging an object of ACPI (Advanced Configuration and Power Interface), called _PSS (Processor Performance States), which can be found on low-end processor. ACPI is an interface developed by many computer manufacturers which enable system configuration and power management directed from Operating System. One of its object is _PSS which contains information of frequency and power of a processor core in each state within processor performance states (P-State). _PSS object had been included in ACPI specification Revision 2 [1]. Hence, any low-end

processor which is compatible with ACPI specification Revision 2 can utilize the _PSS object. The information given by the _PSS is provided by a computer manufacturer such Intel through its firmware. By obtaining this information on a P-State supported device then it is possible to construct a graph of power vs. time as a base for estimating energy of processor core. This method is an alternative to method which is costly and complex that uses additional external power meter [2,3] or software which is only compatible to high-end processor like software called Intel Power Gadget [4] and Intel PCM [5]. The behavior of processor's P-State is controlled by a driver. A well-known driver is ACPI-cpufreq. Intel also releases a new driver called Intel P-State driver as a successor to ACPI-cpufreq. Here, we also compare the differences between the ACPI-cpufreq and Intel P-State when estimating energy of a processor core.

In this paper we start the sections by discussing the motivations and goals in Section 1 and then compare any previous works related to the method of measurement of energy of computer and our contributions in Section 2. We then explain our simple method, i.e., how the _PSS data can be leveraged to estimate energy in Section 3 and show and discuss the results with many combinations in Section 4. We then make our conclusions in Section 5.

2. **Related Works and Contributions.**

2.1. **Hardware power meter and power of entire system.** Some works had been shown to estimate energy by using additional hardware tools. In our opinion, the tools might be too complex and expensive for a software developer when he wanted to know how much energy was consumed by a program. Some works also estimated energy which was consumed by the entire system. In many cases knowing energy consumed by a specific component inside a computer, i.e., processor, is more valuable rather than energy of entire system. In [18], energy was predicted based on system call of a program. The relation between energy and system call was obtained by training the energy and system call which used additional tool called Green-Miner. They claimed that additional hardware was not used. However, the Green-Miner itself required additional hardware, i.e., power supply, Arduino and Ada-fruit. They also estimated energy not on specific component but energy of entire system. They also needed to train the data which add the complexity of the method. In [27], Green Miner was also used to learn the effect of system call data and CPU utilizations towards power consumed by many android applications by comparing 4 machine learning methods. In our opinion, Green Miner itself is complex then the machine learning will add the complexity when implemented in different computer or system because we need to retrain huge android applications in the beginning. In [19], energy of a program was also measured. However, they also used additional expensive tool, i.e., power meter. They also measured entire energy of system not on specific component. In [20], energy of a program was profiled using Power Scope. This tool also used additional hardware, i.e., digital multi-meter to measure DC current. They also did a modification to the kernel which adds the complexity. This tool also measured the entire power which was monitored from the system power supply. In [21], P-State of a processor was also observed to determine what to do in P-State in order to obtain efficiency by proposing some algorithms. However, they also used additional hardware, i.e., ISO-TECH 3005 power meter which was used to measure the entire energy. In [23], a hard work was shown to measure energy of each component of computer, i.e., CPU, disk, and NIC which was consumed by a program. However, to follow their method we need to disassemble a computer system and use additional measurement tool, i.e., National Instrument Data-Acquisition Card. Another complexity when additional hardware measurement tool was

used is synchronizing between power data and runtime of a program. In [24], a comprehensive work had been started to evaluate power on small computer. They directly measured each component in Itsy Pocket Computer. In our opinion, they still needed a very good instrument to measure which equals high cost. In [3,10], AC power and a combined DC power of CPU package and DRAM were measured. In this work external power meter was still used. Also in [13], an external hardware power meter was used to evaluate the power on high-end processors. In [26], energy of a software was estimated more granular by defining point of phases within the software by utilizing state diagram of UML. A better graphical power usage showing phases of a software can be shown. However, Agilent Ammeter and Hewlett Packard power supply as additional hardware were still needed. All those works mentioned in [3,10,18-24,26,27] might be suitable for specific groups who can provide additional hardware tools but might not be for common software developers who want to know how much processor energy is consumed by their programs by a simple method. Furthermore, additional hardware power meter in some works was used to measure entire energy of system but not able to measure energy of specific component, i.e., processor core only. Therefore, a simple method, without additional hardware power meter, is needed to estimate energy of processor core only.

2.2. **Energy based on architecture.** Another work showed a different approach. In [22], energy was estimated by calculating number of capacitances inside a computer and simulated by combining the capacitances calculation and the binary source of the program to obtain energy consumed. In our opinion, this concept was not easy because whenever the architecture of the computer was changed then they must re-calculate the capacitances. This concept may apply for computer manufacturer but not for a common programmer.

2.3. **Intel software and embedded feature in high-end processor.** Some works also showed non-additional hardware approach which was implemented by utilizing either embedded features or software even though additional power meter still used. Those features and software worked only on high-end processor system but could not work on low-end processor. An effort was shown in [11] where a new metric was proposed to measure efficiency of energy consumed. The energy consumed was measured by using a software called Intel Power Governor which measured power of CPU and DRAM without additional hardware. However, Intel Power Governor could only be used on high-end processor, i.e., Intel Xeon E5 series, not on low-end processor, which was located on their Marcher System. In [10], a concept called eDVFS was proposed which P-State was monitored and then optimized to gain efficiency in software. Here, power meter of Yokogawa WT-210 was used for measuring power of entire system while utilizing software called Intel Performance Counter Monitor (PCM) for measuring power on each component. However, PCM itself could only be used on specific processor. For example, when we tested it on Intel NUC DN2820FYKH, it could not be used. In [3,7], power was measured by using Intel's feature called RAPL (Running Average Power Limit) [6]. However, this feature was only found on specific processor which was categorized as high-end. In [8], power was measured by using framework called PAPI (Performance Application Programming Interface). However, this PAPI was also based on RAPL which cannot be found on low-end processor. On ACPI specification version 4, there is an object called _PMM (Power Meter Measurement), which a feature used to measure power [9]. However, this object cannot be found on low-end processor. Besides that, this feature is not used to measure DC power but AC power. All those works in [3,6-11] utilized non-hardware tools, i.e., Intel based Software, RAPL or ACPI's _PMM to measure power and energy found only on high-end processors and not on low-end processors. On the other hand, P-State information which

could be found on both high-end and low-end processor was not yet purposely used to estimate power and energy in those works. Therefore, it is an opportunity to leverage the P-State information to estimate energy of processor core categorized as low-end.

2.4. **P-state driver on Linux.** A tool for measuring power is important in green computing especially on Linux system as explained in [12]. On the other hand, P-State behavior is controlled by driver. Different drivers will show different P-state behaviors. On Linux, such Ubuntu, ACPI-cpufreq driver is used to control the P-State behavior of processor core. Some works like in [7,10,13] discussed about P-State driver. However, they all only discussed about ACPI-cpufreq driver and compared it with their other solutions. In [10], it proposed and claimed eDVFS as a better alternative to policy of on-demand based on driver of ACPI-cpufreq. In [7], it was preferred to read CPU cycles rather than frequency based on driver of ACPI-cpufreq to examine the P-State latency because ACPI-cpufreq was not a good indicator for an actual frequency. In [13], it was compared between ACPI-cpufreq driver with policy of on-demand and their algorithm, i.e., both fuzzy logic and immune inspired algorithm. They concluded that ACPI-cpufreq with policy of on-demand was the worst CPU energy management algorithm while the immune inspired algorithm was the optimal one. Overall, in [7,10,13], they compare ACPI-cpufreq driver with their offered solutions, i.e., eDVFS, CPU cycles based, and immune inspired algorithm. They did not compare with Intel P-State Driver yet. In [28], higher energy consumed was explained as effect of power state of each component hardware. Energy used was compared on different operating systems, i.e., Windows and Linux. However, they did not discuss what driver was used to control the power state especially for processor.

We noticed a website comparing between ACPIcpufreq driver and Intel P-State driver in [14] by using Phoronix benchmark software with kernel Linux 3.15. However, they evaluated it on high-end processor Intel Core i7 4960X Ivy Bridge Extreme Edition and not on low-end processor. Also additional power meter of WATTSUP was used to measure the AC power consumption and performance per watt. Also in [15], both driver ACPI-cpufreq and Intel P-State were compared with Linux kernel 4.7 on high-end processor Intel Xeon E5-2687W v3 Haswell system. However, they did not measure and compare the power. They also did not compare energy consumed between driver ACPI-cpufreq and driver Intel P-State. As far as our efforts, we found no yet works which compared energy of processor core consumed between Intel P-State driver and ACPI-cpufreq on low-end processor without using additional hardware power meter.

2.5. **Contributions.** We emphasize our contributions in this paper, i.e.,

- We estimate energy of core processor, not entire system, by simple method, i.e., leveraging ACPI _PSS data without additional power meter which is costly and complex;
- We estimate energy of core on low-end processor which other tools and other software cannot work on this low-end processor;
- We compare the different behavior and power consumed between driver ACPI-cpufreq and driver Intel P-State on low-end processor.

3. Energy Estimation of Software Programs.

3.1. **Low-end processor supporting P-States.** _PSS namespace, which describes P-States, is found on ACPI specification version 2. Hence, processor that is used as device under test must be compatible with ACPI specification version 2 and above. We used Intel DN2820FYK with Pentium Celeron N2830 Processor as device under test because it is compatible with ACPI version 3 and P-States. N2830 Processor is embedded on board and

it has 2 cores. This processor is also categorized as low-end processor. This low-end N2830 Processor is not compatible with any software and feature for power measurement which is found on high-end processor, i.e., Intel PCM, Intel Power Gadget, RAPL (Running Average Power Limit), and ACPI _PMM. Thus, we cannot use those features in this low-end N2830 processor to measure power and energy.

Frequency TDP (Thermal Design Power) of the processor is 2.159 GHz, and by enabling the boost feature, the frequency of the processor can reach 2.4 GHz. To make sure all of this processor feature was usable, we updated Intel DN2820FYK to its latest firmware. The system is run on Operating System Linux Ubuntu 15.04 with kernel 3.19.0-42-generic. 'Cpupower' command was used to show information about range of frequency of processor, power management policy, and which driver was used.

On ACPI Specification, _PSS contains number of possible states in P-States which provides information about frequency of core, power of core, transition latency, and value of register PERF_CTRL and value of register PERF_STATE for each state. We obtained these value by using tools provided by ACPI on its website [16], i.e., ACPI-dump, ACPI-xtract and ACPI-exec. ACPI-dump was used to gather APCI raw data provided by computer manufacturer. ACPI-xtract was used to convert the raw data into ACPI tables. ACPI-exec was used to look into the _PSS, i.e., P-States information on each core.

3.2. **Collecting and monitor frequency of processor core.** We did collect and monitor frequency of processor core while running a program. The program was strassen.gcc.omp-task, a matrix $N \times N$ multiplication program, which belongs to Barcelona Open MP Tasks Suite [17], a benchmark of task parallelism. We used PERF to collect event data, i.e., frequency and time during execution of the program. This PERF was also run in the Intel DN2820FYKH. We did control and monitor the collection of the data on HP Elitebook 840 connected via Wi-Fi using HP voice tab 7 like on Figure 1.

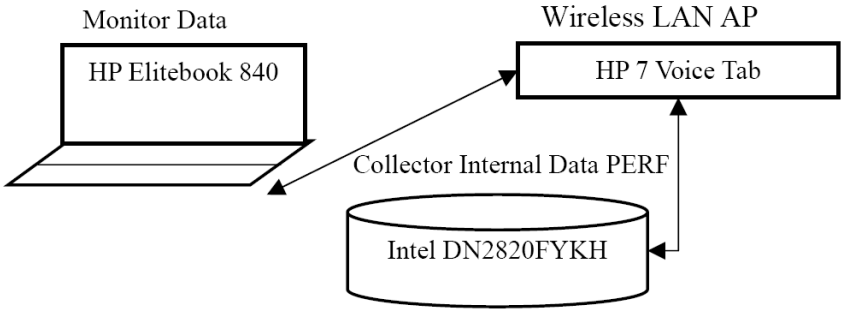

FIGURE 1. Collecting and monitoring data

The same program, i.e., strassen.gcc.omp-task was run respectively on different settings and drivers to see its effect towards power consumed. By default the system was run on Intel P-State driver. We switched the driver into ACPI-cpufreq, by disabling the kernel Intel P-State driver in the grub file. When collecting event data by PERF, we used event of 'power:cpu_frequency' for driver ACPI-cpufreq and event of 'power:pstate_sample' for driver Intel P-State. Before the events were captured, policy on both drivers was set. First, the boost feature was disabled on both drivers. Hence, the frequency of processor would not reach its maximum value, i.e., 2.4 GHz. Second, power management was set to 'powersave' on Intel P-State, while on ACPI-cpufreq it was set to 'ondemand'. It was set because both drivers had different number of power management. However, 'powersave' on Intel P-State and 'ondemand' on ACPI-cpufreq had a similar concept of power management. Third, the minimum frequency and the maximum frequency on both

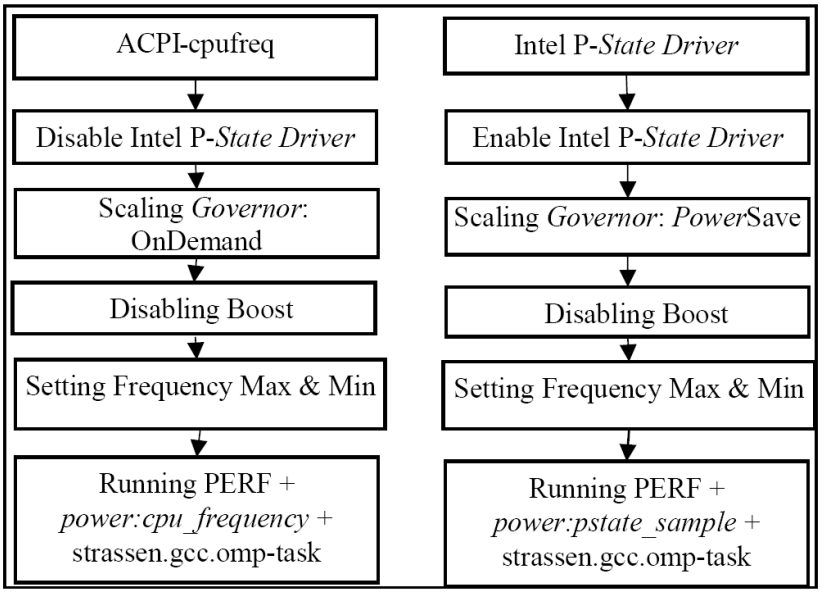

FIGURE 2. Comparison of collecting data between Intel P-State and ACPI-cpufreq

drivers were set to 498 MHz and 2159 MHz respectively. The process of this collecting data was illustrated on Figure 2.

3.3. **Mapping frequency information into power.** Having got the frequency of cores when running a program then we transformed the frequency into power. The relation between Power and Frequency was already explained such in [22], i.e., $P = CV^2af$, where P is power, C is capacitance, V is voltage, and f is clock frequency. In our methodology, the power information of cores on ACPI-cpureq driver was obtained easily by mapping the frequency value of cores with power value based on _PSS information provided by the computer manufacturer. On the other hand, power information of cores (Pt) on Intel P-State was obtained by transforming the frequency value of cores (Ft) using comparison formula below.

$$Pt = \frac{(Ft - F\min)}{(F\max - F\min)} * (P\max - P\min) + P\min \tag{1}$$

where $F\min$, $F\max$, $P\min$, and $P\max$ are respectively minimum frequency, maximum frequency, minimum power and maximum power of P-States listed on _PSS data.

3.4. **Integrating power into energy using trapezoid method.** By using data of power which was obtained on previous subsection, we proceeded energy calculation. We estimated the energy by using integral of power during execution time. The integral was calculated by summing trapeziums formed between two power samples on each core which was also called as trapezoid method of integration [25] which is shown on the following formula. For the sake of simplicity we ignored P-State latencies

$$\begin{aligned} W \approx {} & \{(P_1 + P_2) * (t_2 - t_1)/2\} + \{(P_2 + P_3) * (t_3 - t_2)/2\} \\ & + \cdots + \{(P_n + P_{n+1}) * (t_{n+1} - t_n)/2\} \end{aligned} \tag{2}$$

$$W \approx \sum_{0}^{n} (P_n + P_{n+1}) * (t_{n+1} - t_n)/2 \tag{3}$$

3.5. **Estimating energy on several conditions.** We also estimated energy of the same software in several conditions. First, we estimated the energy in different power management policy. On ACPI-cpufreq there are five power management policies, i.e., ondemand, performance, user space, conservative and power-save. On the other hand, Intel P-State has two power management policies, i.e., power-save and performance. Second, we estimated the energy by enabling feature of boost and by setting different minimum-maximum frequency. We also estimated the energy when the load of the matrix program (size of matrix $N \times N$) was increased from $N = 1024$, 2048, 4096, and 8192.

4. Results.

4.1. **Power and frequency information.** By running the ACPI tools, i.e., acpidump, acpixtract, and acpiexec, we obtained the mapping between the frequency and power as listed on Table 1. The P-State sequence was determined by choosing P0 as the state with the highest frequency. From the table, frequency range between states is 166 MHz and power range between states is 140mW. However, between P0 and P1, frequency range is only 1MHz while power of P0 and P1 is the same.

TABLE 1. Power and frequency information of P-States

Freq (MHz)	Power (mW)	P-State
2159	2000	P0
2158	2000	P1
1992	1860	P2
1826	1720	P3
1660	1580	P4
1494	1440	P5
1328	1300	P6
1162	1160	P7
996	1020	P8
830	880	P9
664	740	P10
498	600	P11

4.2. **Power profile before executing strassen.gcc.omp-task.** We built power profile before a program under test was run. We collected the processor's core frequency during a range of time by running PERF only without running the program under test, i.e., strassen.gcc.omp-task. We called this as base frequency. Then, this base frequency was mapped into power based on Table 1. The results are shown in Figure 3 and Figure 4. Figure 3 shows graph of power of processor core on ACPI-cpufreq during 100.236 Seconds while Figure 4 shows graph of power of processor core on Intel P-State during 79.576703 S. Figure 3 is constructed from 132 samples while Figure 4 is from 581 samples. In Figure 3, Core 0 is stable at 600 mW as the minimum power while Core 1 is fluctuated from 600 mW to 2000 mW. In Figure 4, both cores tend to be stable at around 601.4683133 mW which is mapped from frequency 499.741 MHz. There is a spike at very beginning which indicates that there is a process running at the first time.

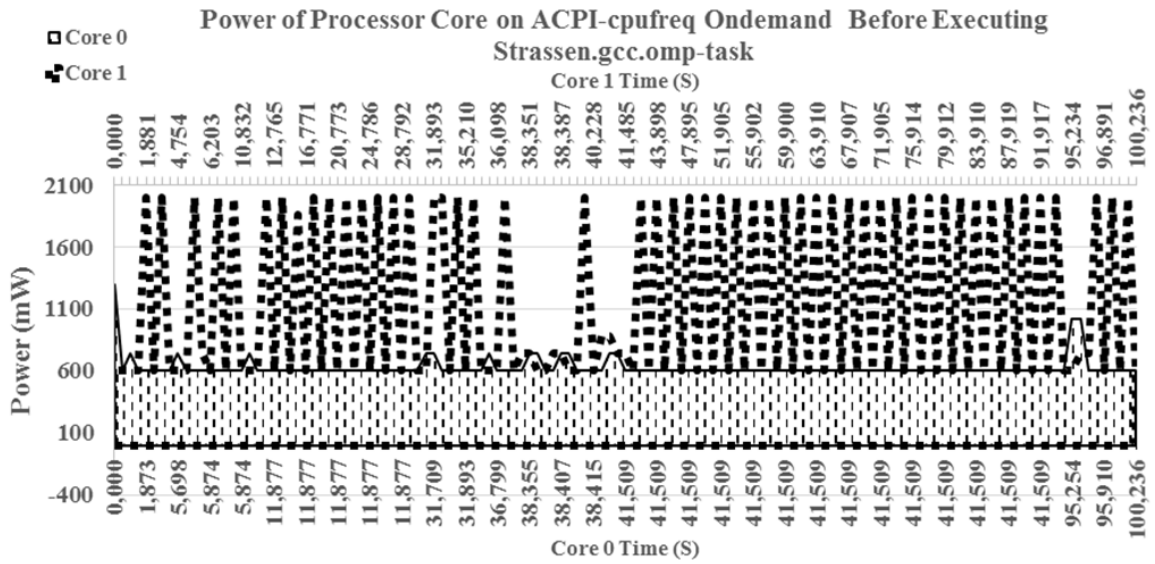

FIGURE 3. Graph of power before executing strassen.gcc.omp-task on ACPI-cpufreq

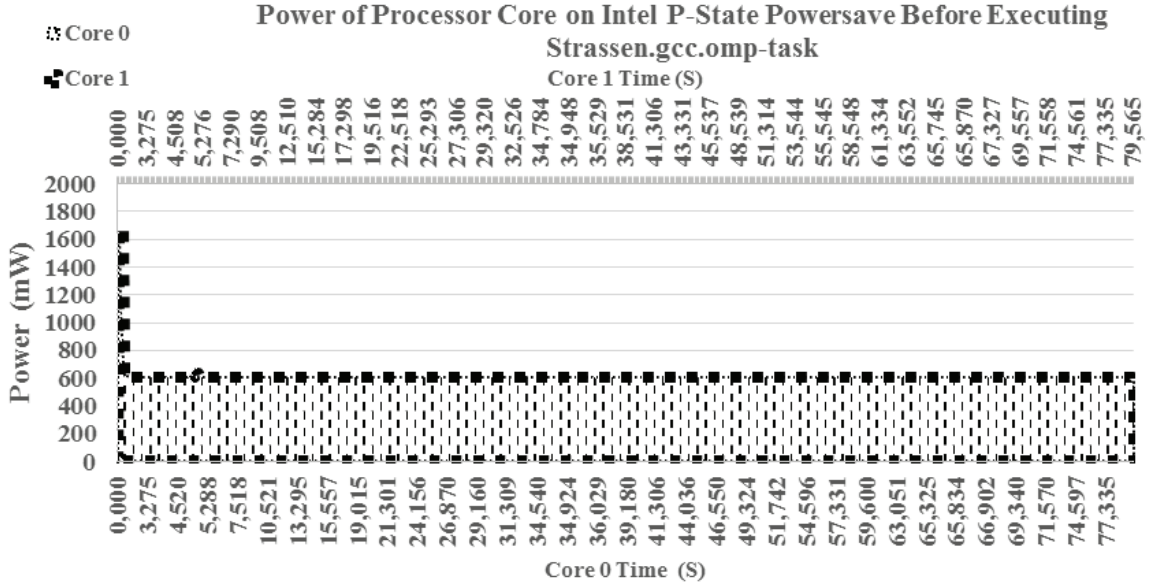

FIGURE 4. Graph of power before executing strassen.gcc.omp-task on Intel P-State

4.3. **Power profile while executing strassen.gcc.omp-task.** We then built power profile while executing program under test. We collected the core's frequency by using PERF while executing program strassen.gcc.omp-task with load $N = 4096$. By mapping the frequency into power, we then built power profile for ACPI-cpufreq in Figure 5 and power profile for Intel P-State in Figure 6. The program strassen.gcc.omp-task solved the matrix problem within 30.264416 Seconds when using ACPI-cpufreq driver. Whereas the matrix problem was solved within 35.26126 Seconds when using Intel P-State driver. In Figure 5, power profile is constructed from 23 samples while in Figure 6 power profile is constructed from much more samples, i.e., 5306 samples. Both in Figure 5 and Figure 6, there is axis for each core. In Figure 5, core 1 is stable at its maximum power 2000 mW except a few milliseconds at the beginning and about 2 seconds at the end which reaches

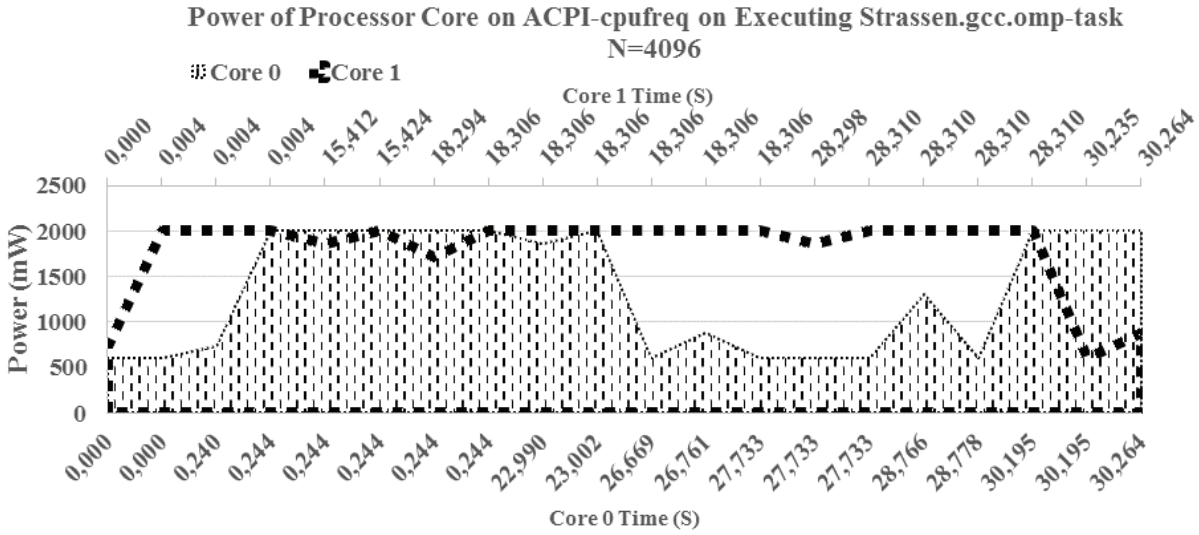

FIGURE 5. Graph of Power while executing strassen.gcc.omp-task on ACPI-cpufreq

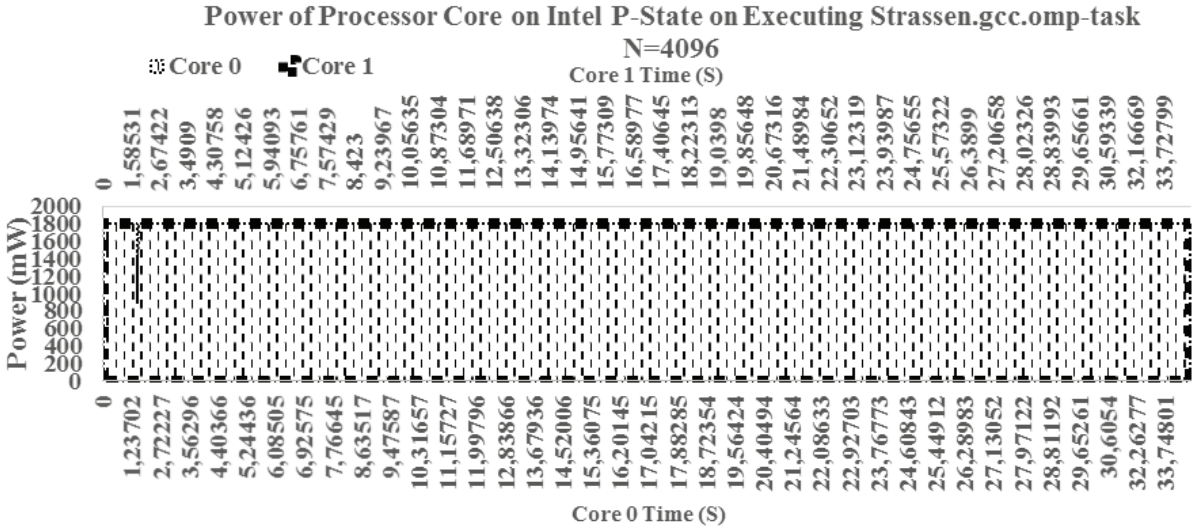

FIGURE 6. Graph of power while executing strassen.gcc.omp-task on Intel P-State

minimum 600 mW. Core 0 is also stable at its maximum except about 2 seconds between on 26.669 S and on 28.778 S which touches its minimum 600 mW. In Figure 6 both cores are stable at about 1795.80747 mW mapped from 1.915886 GHz from the beginning until the end of execution time.

By comparing between Figure 3 and Figure 5 also between Figure 4 and Figure 6, our results show that there is a change on the power profile between before the strassen.gcc.omp-task was executed and while executing strassen.gcc.omp-task. Also, there is change of the cores from its minimum state to its maximum state on ACPI-cpufreq and from 601.4683133 mW to 1795.80747 mW on Intel P-State.

Sample of data which were captured showed that on ACPI-cpufreq, the event 'power:cpu _frequency' would provide data only if there was a change on the core frequency, while on Intel P-State, the event 'pstate:power_sample' provided data on every single interval time even though the core frequency was not changed. Besides that, event 'pstate:power_sample' provided sample of data much more than event 'power:cpu_frequency'. Hence, we can say

that event 'pstate:power_sample' which is event of Intel P-State driver gives better data than event 'power:cpu_frequency' which belongs to ACPI-cpufreq.

4.4. **Energy estimation.** By using trapezoid method of integration, we estimated energy based on power profile in Figure 5 and Figure 6. On ACPI-cpufreq, energy was 52.616333 J on core 0; 56.958585 J on core 1, and 109.574918 J on both cores. Whereas, on Intel P-State, energy was 63.29287J on core 0, 63.32235244 J on core 1, and 126.6152 J on both cores. However, this energy was energy consumed by operating system including all program run in the computer and not the program strassen.gcc.omp-task itself. Hence, we need to subtract it with energy before the strassen.gcc.omp-task was executed by calculating average power in Figure 3 and Figure 4, then multiplying the average power with execution time of strassen.gcc.omp-task. We obtained average power on ACPI-cpufreq, i.e., 0.744907 J/s on core 0; 1.291010 J/s on core 1; and 2.035917 J/s on both cores. We also obtained average power on Intel P-State, i.e., 0.608217769 J/s on core 0; 0.608467064 J/S on core 1; and 1.216684833 J/S on both cores. After multiplying the average power with the execution time of strassen.gcc.omp-task, we obtained energy before the program was run, i.e., 61.61584 J on both cores for ACPI-cpufreq and 42.90184 J on both cores for Intel P-State. Finally by subtracting the energy when the program executed and energy before the program executed, then we estimated energy of processor core consumed by strassen.gcc.omp-task itself, i.e., 47.95907897 J for ACPI-cpufreq and 83.71335977 J for Intel P-State. We can see that the program strassen.gcc.omp-task itself consumed more energy on Intel P-State than on ACPI-cpufreq.

4.5. **Policy, boost, and frequency combinations.** In this subsection we implemented some combinations by changing some items, i.e., policy of power government on both drivers, the boost feature, and minimum-maximum frequencies. We obtained execution time on Table 2 for ACPI-cpufreq. In Table 2 it is shown the effect toward the execution time when ACPI-cpufreq driver was used. When the boost feature was enabled and frequency was set to 2159 MHz for both minimum and maximum, the execution time of the program tended to be the same even though the policy of power government was either ondemand, conservative, userspace, powersave, or performance. When the minimum frequency was changed to 498 MHz, the powersave policy showed the longest execution time while others still had similar execution time. When boost feature was disabled, the longest execution time was achieved when the policy was set to userspace; Powersave was the second longest while performance, ondemand, and conservative had similar execution time. We can see that the boost feature had significant effect toward the execution time.

TABLE 2. Combinations of policy, frequency, and boost feature on ACPI-cpufreq

No	Boost	Freq. Min (MHz)	Freq. Max (MHz)	Execution Time (S)				
				Ondemand	Conservative	Userspace	Powersave	Performance
1	yes	2159	2159	27.915281	27.686746	27.74516	27.676823	27.691734
2	yes	498	2159	27.831756	28.102168	27.80213	114.46815	27.722005
3	no	498	2159	30.188098	30.14639	114.7714	106.87286	30.159552

Table 3 contains combination of policy, boost, and frequency on Intel P-State driver. We also estimated total core's energy. We found that the only thing that affected energy on performance policy was the boost feature. On performance policy, when the boost feature was enabled, the frequency reached its maximum at 2415628 Hz. However, when the boost feature was disabled, the frequency reached its maximum only at 2165800 Hz even though the minimum was set to either 498 MHz, 2159 MHz, or 2.42 GHz. Different thing was shown when the power policy was set to powersave policy. The frequency on

powersave policy was changed if either the maximum frequency was set to change or the boost feature was changed. On powersave policy, the least energy happened when boost was disabled and maximum frequency was set to 2159 MHz. On performance policy, the least consumed energy was shown only when the boost feature was disabled. From Table 2 and Table 3, we can see that the boost feature, maximum frequency, and policy of power management on both drivers affected the execution time and the energy consumed.

TABLE 3. Combinations of policy, frequency, and boost feature on Intel P-State

No	Boost	Set Freq. min (GHz)	Set Freq. max (GHz)	Powersave			Performance		
				Time (S)	Output Freq. max (Hz)	Energy (J)	Time (S)	Output Freq. min (Hz)	Energy (J)
1	yes	2.159	2.159	31.0558	2082467	127.2467	27.7365	2415628	129.9644
2	yes	0.498	2.159	31.0519	2082467	127.2251	27.7347	2415628	129.9472
3	yes	2.42	2.42	27.7295	2415628	129.9249	27.7793	2415628	130.2035
4	yes	0.498	2.42	27.7516	2415628	130.0370	27.7392	2415628	130.0112
5	No	2.159	2.159	33.3234	1915886	126.6492	30.1574	2165800	127.9626
6	No	0.498	2.159	33.2758	1915886	126.3624	30.1573	2165800	127.9268
7	No	2.42	2.42	30.1591	2165800	128.0456	30.1654	2165800	127.7584
8	No	0.498	2.42	30.0694	2165800	127.6710	30.2087	2165800	128.1714

4.6. **Workload change of $N \times N$ matrix.** We changed the load of the matrix $N \times N$ by increasing the number N from 1024, 2048, until 8192 while other settings were the same as Subsection 4.3. We obtained Figure 7 until Figure 9 for ACPI-cpufreq and Figure 10 until Figure 12 for Intel P-State. Load $N = 4096$ is already shown in Figure 5 for ACPI-cpufreq and Figure 6 for Intel P-State. The difference is, in Figure 5 and Figure 6, the scale of the axis time was set irregularly while in Figure 7 until Figure 12, the scaling of the axis time was set regularly. Based on Figure 7 until Figure 9, it was shown that by increasing the load, the execution time becomes longer while power consumed by cores stayed on the maximum P-State. However, there was an exception, i.e., few seconds in the beginning and in the end of execution time where the power went down to a lower P-State which was shown on core 1. Based on Figure 10 until Figure 12, it was shown that the execution time was also increased by increasing the load on which all of the time the power was likely to be stayed on one P-State with fewer declination. We summarized the effect of the load in Figure 13 for ACPI-cpufreq and Figure 14 for Intel P-State. More load caused more execution time and more energy consumed. Based on Figure 13, the

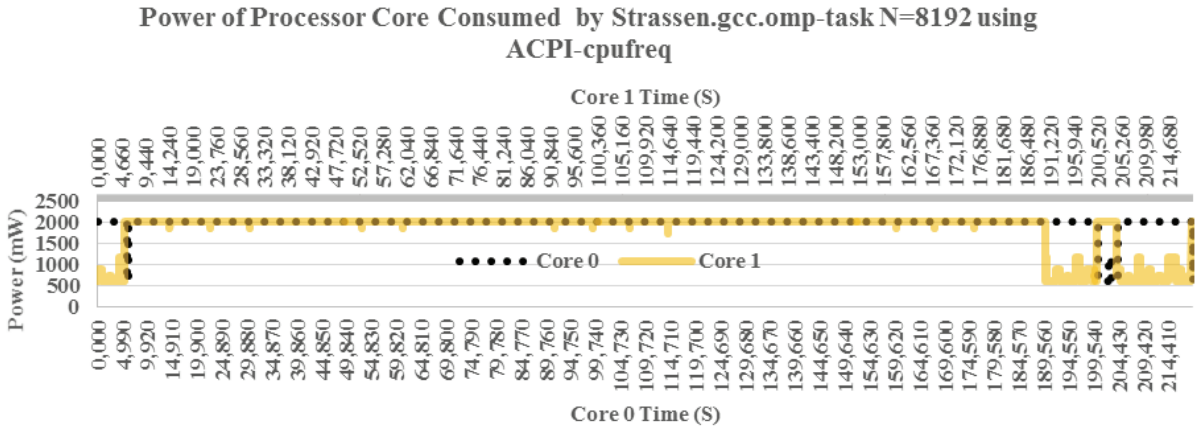

FIGURE 7. ACPI-cpufreq $N = 8192$

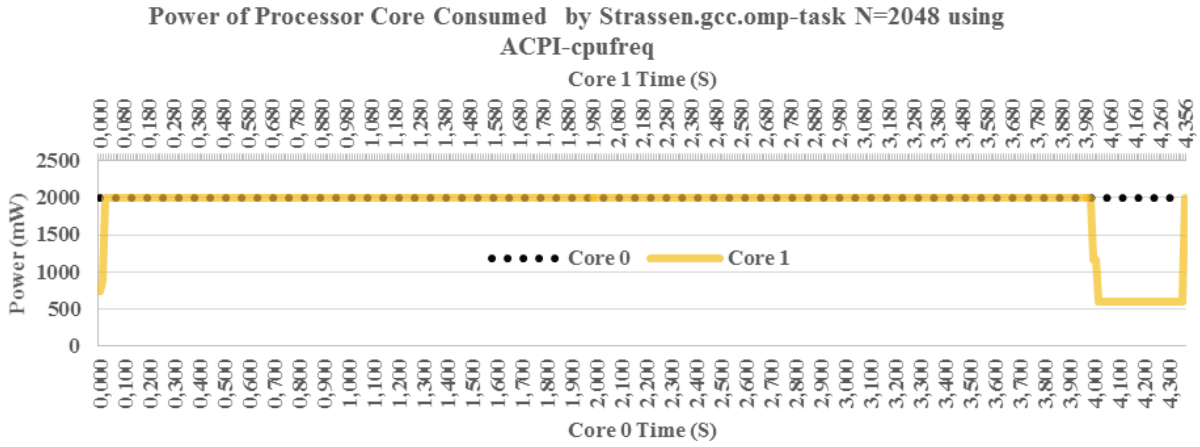

FIGURE 8. ACPI-cpufreq $N = 2048$

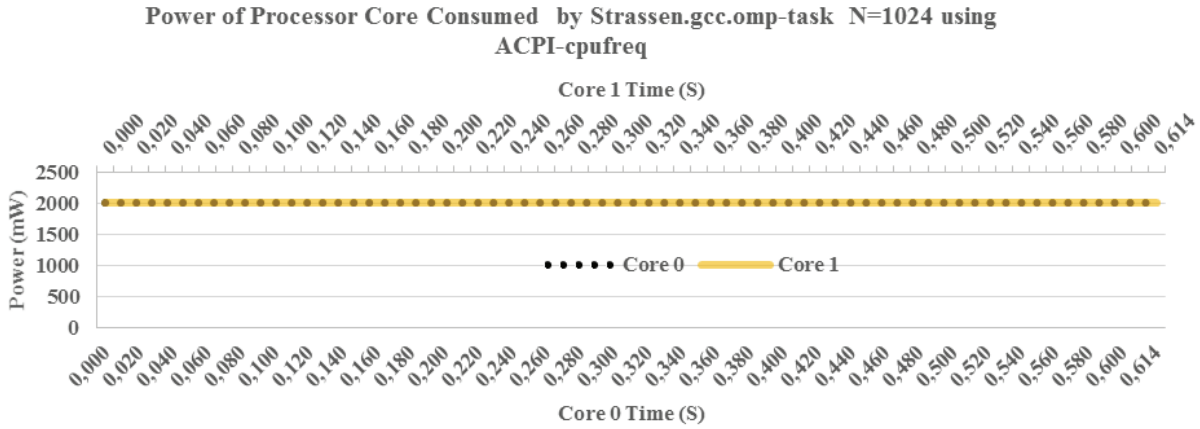

FIGURE 9. ACPI-cpufreq $N = 1024$

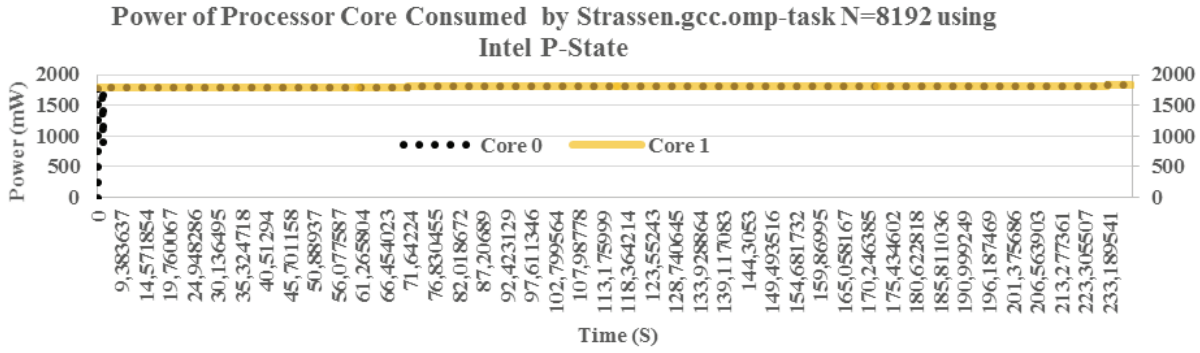

FIGURE 10. Intel P-State $N = 8192$

average power on ACPI-cpufreq showed a declination when the load was increased while based on Figure 14, on Intel P-State the average power remained constant.

4.7. **Discussion.** In this paper, we show that energy of cores on a low-end processor can be easily estimated. The estimation depends on _PSS information which is provided by a computer manufacturer. Collecting frequency of core, mapping the frequency into power based on _PSS, and integrating the power are the important steps. Energy estimation itself is influenced by environment, i.e., the driver chosen, power management policy, the boost feature, and setting of minimum-maximum frequency. Hence, we cannot depend

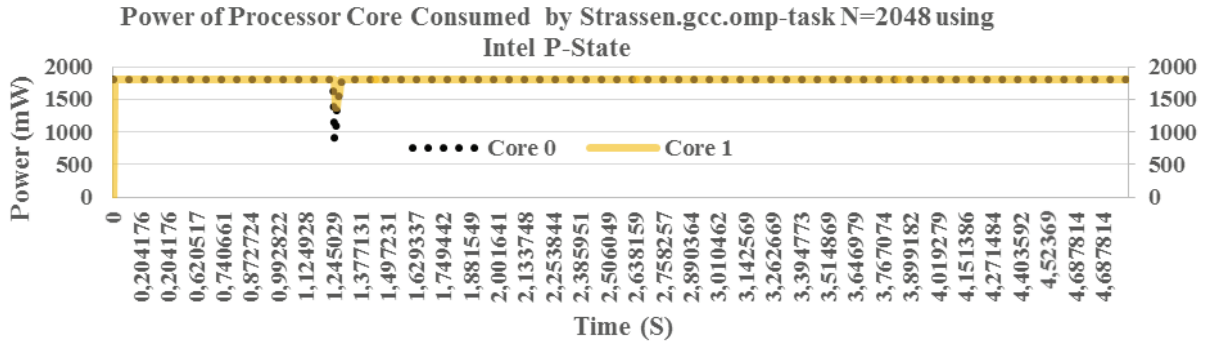

FIGURE 11. Intel P-State $N = 2048$

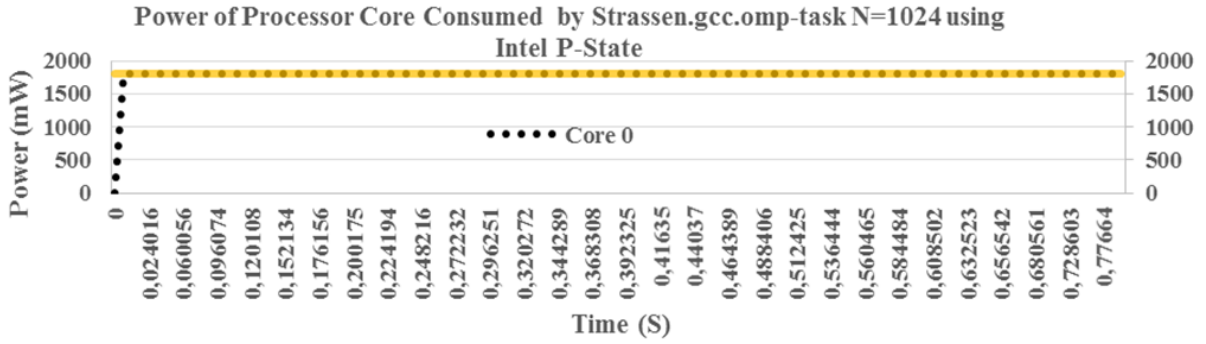

FIGURE 12. Intel P-State $N = 1024$

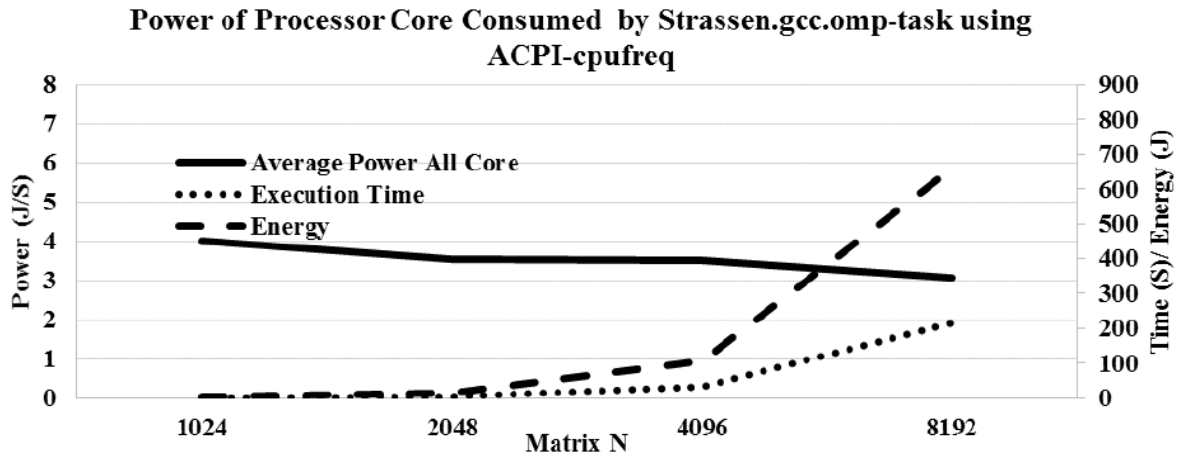

FIGURE 13. Workload effect on ACPI-cpufreq

only on how we modified a source code of a program to achieve an efficient program but we need also be aware of how the environment is set. We show that the same program with different environment settings can yield different energy results.

Intel P-State, with its event 'power:pstate_sample', gives more sampled data than ACPI-cpufreq, with its event 'power:cpu_frequency'. Hence, Intel P-State gives better resolution power graph than ACPI-cpufreq. The difference is that Intel P-State always shows output of event, even though the state is not changed. On the other hand, ACPI-cpufreq shows output of event only when there is a processor core movement from one state into another state. However, it does not mean to be a more exact estimation unless we can prove that there are states which are not sampled when using ACPI-cpufreq with event 'power:cpu_frequency'. Another thing when using ACPI-cpufreq is this driver may

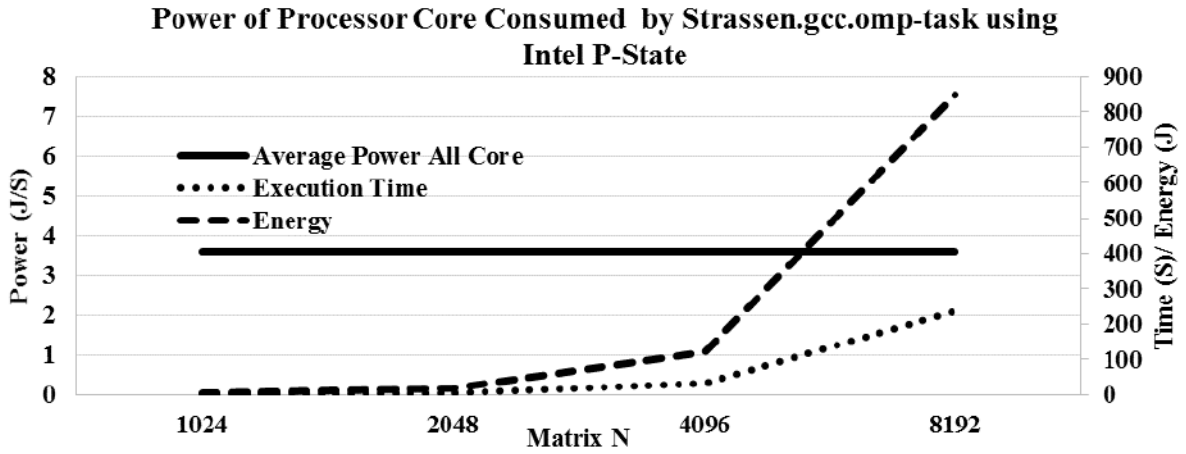

FIGURE 14. Workload effect on Intel P-State

not always give sampled data. In our experiment, many times we captured event data of ACPU-cpufreq but yielded no output especially when the execution time of a program was just few seconds. This might be caused by a reason that there was no P-state movement when a program ran causing no event ACPI-cpurfeq data captured.

When comparing policy of power management, ACPI-cpufreq offers more options than Intel P-State. ACPI-cpufreq provides 5 policies where some policies can show very different execution time. On the other hand, Intel P-State with its 2 policies does not show a significant different execution time and energy consumed between those 2 policies. Most of the time, our results show that the more execution time needed, then the less energy consumed.

Intel P-State and ACPI-cpufreq give similar frequency for all cores in most of the time. Therefore, the energy will depend only on its maximum frequency and execution time. However, ACPI-cpufreq has more frequent movement into any lower states rather than Intel P-State. When boost feature is disabled, frequency of processor core will reach up to only 89.9% of the maximum frequency which will extend execution time of a program and reduce energy consumed.

At least we have 2 threats to validity. First, our methodology depends only on _PSS. Based on ACPI specification, _PSS provides information about power on each state with its maximum value. However, the actual power may be different than its maximum value. So when a core enters into a state, the core may not reach its maximum power in that state. Second, we also ignore the latencies when a core moves from one state into another state. The more frequent of a core moves from one state into another state, the more frequent latency will be shown. Even though, based on the power graphs, movement of the cores is rare compared to the entire execution time. Cores also tend to be stable into one state.

5. **Conclusions.** We have shown that _PSS information can be leveraged to estimate energy of processor's cores easily. This method has 3 important steps, i.e., collecting frequency, mapping frequency into power, integrating the power to calculate the energy. This method is suited used especially on low-end processor where features such as RPAL and _PMM are not available. It is also suited on a condition where measurement tool such as Intel Power Gadget or Intel PCM cannot run on it. By using this method, a programmer who develops a program may estimate easily how much energy consumed by his program. Especially when he runs the program on a low-end processor and considers a green computing. This method also requires no additional hardware measurement tool.

When estimating the energy, things to be aware are the environment setting of the processor, i.e., boost feature, minimum-maximum frequency, power management policy, and driver chosen. Related to the drivers, Intel P-State can provide better information when used to estimate energy rather than ACPI-cpufreq. Intel P-State provides data with more samples, more stable on a certain state and more consistent by always providing data with non-zero output.

A future work needs to be tested on a processor which supports P-State and RAPL. Our estimation methodology based on _PSS approach needs to be compared with other tools like Intel PCM or Intel Power Gadget. Based on our experiences, the comparison should be tested on a high-end processor which supports RAPL.

REFERENCES

[1] Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation, *Advanced Configuration and Power Interface Specification Revision 2.0*, 2000.

[2] C. Xian, L. Cai and Y. Lu, Power measurement of software programs on computers with multiple I/O components, *IEEE Trans. Instrumentation and Measurement*, vol.56, no.5, pp.2079-2086, 2007.

[3] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt and W. E. Nagel, Power measurement techniques on standard compute nodes: A quantitative comparison, *IEEE International Symposium on Performance Analysis of Systems and Software*, pp.194-204, 2013.

[4] *https://software.intel.com/en-us/articles/intel-power-gadget-20.*

[5] *https://software.intel.com/en-us/articles/intel-performance-counter-monitor.*

[6] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3 (3A, 3B & 3C): System Programming Guide*, 2015.

[7] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart and R. Geyer, An energy efficiency feature survey of the Intel Haswell processor, *IEEE International Parallel and Distributed Processing Symposium Workshop*, pp.896-904, 2015.

[8] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra and S. Moore, Measuring energy and power with PAPI, *The 41st International Conference Parallel Processing Workshops (ICPPW)*, pp.262-268, 2012.

[9] Hewlett Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation, *Advanced Configuration and Power Interface Specification Revision 4.0*, 2009.

[10] S. Kim, C. Choi, H. Eom, H. Y. Yeom and H. Byun, Energy-centric DVFS controlling method for multi-core platforms, *SC Companion: High Performance Computing, Networking Storage and Analysis*, pp.685-690, 2012.

[11] S. Abdulsalam, Z. Zong, Q. Gu and M. Qiu, Using the greenup, powerup, and speedup metrics to evaluate software energy efficiency, *The 6th International Green Computing Conference and Sustainable Computing Conference*, pp.1-8, 2015.

[12] G. Pinto, F. Castor and Y. D. Liu, Mining questions about software energy consumption, *Proc. of the 11th Working Conference on Mining Software Repositories*, pp.22-31, 2014.

[13] I. Anghel, T. Cioara, I. Salomie, G. Copil, D. Moldovan and C. Pop, Dynamic frequency scaling algorithms for improving the CPU's energy efficiency, *IEEE International Conference on Intelligent Computer Communication and Processing*, pp.485-491, 2011.

[14] *http://www.phoronix.com/scan.php?page=article&item=intel_pstate_linux315&num=1.*

[15] *http://www.phoronix.com/scan.php?page=article&item=linux-47-schedutil&num=1.*

[16] *https://acpica.org/downloads.*

[17] A. Duran, X. Teruel, R. Ferrer, X. Martorell and E. Ayguad'e, Barcelona OpenMP task suite: A set of benchmarks targeting the exploitation of tied parallelism in OpenMP, *International Conference on Parallel Processing*, pp.124-131, 2009.

[18] S. A. Chowdhury, L. N. Kumar, M. T. I. M. S. M. Jabbar, V. Sapra, K. Aggarwal, A. Hindle and R. Greiner, A system-call based model of software energy consumption without hardware instrumentation, *The 6th International Green Computing Conference and Sustainable Computing Conference*, pp.1-6, 2015.

[19] T. Johann, M. Dick, S. Naumann and E. Kern, How to measure energy-efficiency of software: Metrics and measurement results, *Proc. of the 1st International Workshop on Green and Sustainable Software*, pp.51-54, 2012.

[20] J. Flinn and M. Satyanarayanan, PowerScope: A tool for profiling the energy usage of mobile applications, *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp.2-10, 1999.

[21] I. Anghel, T. Cioara, I. Salomie, G. Copil, D. Moldovan and C. Pop, Dynamic frequency scaling algorithms for improving the CPU's energy efficiency, *IEEE International Conference on Intelligent Computer Communication and Processing*, pp.485-491, 2011.

[22] D. Brooks, V. Tiwari and M. Martonosi, Wattch: A framework for architectural-level power analysis and optimizations, *The 27th International Symposium on Computer Architecture*, pp.83-94, 2000.

[23] C. Xian, L. Cai and Y.-H. Lu, Power measurement of software programs on computers with multiple I/O components, *IEEE Trans. Instrumentation and Measurement*, vol.56, no.5, 2007.

[24] M. A. Viredaz and D. A. Wallach, Power evaluation of a handheld computer, *IEEE Micro*, pp.66-74, 2003.

[25] S. Rosłoniec, *Fundamental Numerical Methods for Electrical Engineering*, Springer-Verlag Berlin Heidelberg, 2008.

[26] R. Yoshimoto, T. Kadono, K. Hisazumi and A. Fukuda, A software energy analysis method using executableUML, *IEEE Region 10 Conference (TENCON)*, 2016.

[27] S. A. Chowdhury and A. Hindle, GreenOracle: Estimating software energy consumption with energy measurement corpora, *IEEE/ACM the 13th Working Conference on Mining Software Repositories (MSR)*, 2016.

[28] A. Hankel, E. Hoekstra, R. van den Hoed and R. van Rijswijk, Measuring software energy efficiency presenting a methodology and case study on DNS resolvers, *The 18th Mediterranean Electrotechnical Conference (MELECON)*, 2016.