Using Hardware Performance Counters to Measure Application Energy Consumption

Priyal Suneja
University of Washington
Github: https://github.com/priyalsuneja/single-server-etrace

1 Introduction

Data centers today are responsible for about 1-2% of worldwide electricity consumption [6, 8]. However, there is no established way to measure the energy consumption of our applications, especially in settings like data centers where multiple applications share the same physical hardware. While there are ways to measure power consumption at the hardware level using power monitors, they do not provide the granularity we require to be able to track energy consumption at an application level.

Being able to have fine grained energy measurement will provide developers and operators with the tools to understand and reduce the energy consumption of software. In this project, I use Hardware Performance Counters to estimate the Energy Consumption of Applications. When run on an in-memory graph processing benchmark, this method yields results with a maximum 30% error.

2 Background

2.1 Previous Work

This work is heavily inspired by Contreras et. al's [4] work from the mid 2000s. They used Hardware Performance Counters such as TLB Misses, Data dependencies, Cache misses etc. and processor voltage information to create a linear model to predict power consumption of applications running on Intel XScale Processors.

Their work serves as a proof of concept for this idea, however, it has very different challenges given how much processors have changed over the past 15 years. Nevertheless, it provides unique insight into how this question can be approached, and I relied on their methods at several points.

2.2 RAPL

RAPL, or Running Average Power Limit [5] is an Intel feature introduced in 2011 that enables users to monitor the power consumption of a processor, on-chip GPU and attached DRAM using hardware counters. RAPL splits the chip and DRAM into domains, and reports the energy consumption on a per-domain basis. For example, an entire socket is considered "Power Plane 0", the GPU is considered "Power Plane 1" and so on.

RAPL currently supports the following domains:

- Package: Measures energy consumption of the entire socket, including the cores, integrated graphics and uncore components such as last level caches.
- Power Plane 0: Measures energy consumption of **all** the cores on a socket
- Power Plane 1: Measures energy consumption of integrated processor graphics.
- DRAM: Measures energy consumption of the DRAM attached to the socket's memory controller
- PSys: Measures the thermal and power specifications of a single socket SoC.

Figure 1 (cited from [7]) shows a visual representation of these domains.

While RAPL provides a starting point to measuring energy consumption, it suffers from several limitations:

- Since an entire socket is a single domain, there is no way to measure power on a per-core basis using RAPL.
- It has a maximum sampling rate of 1000 updates per second, which is too slow for modern-day applications, and does not fit our vision of being able to measure energy consumption on a per-function basis.

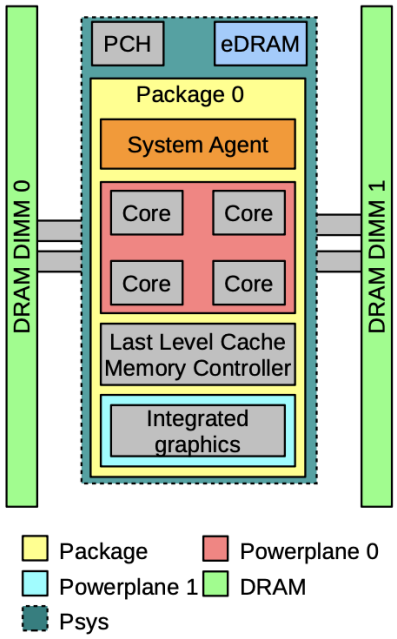

Figure 1: RAPL domains

- Its DRAM values were only introduced in the Haswell architecture in 2013 and are considered unreliable for the older architectures [7].

- Since RAPL is a new feature, it is not present on all the hardware that is currently deployed in data centers, and also has varying accuracy of its results depending on which architecture it is being used on.

2.3 Linux perf

perf [1] is a performance analyzing tool in Linux, which is available to users as a command line utility. perf can profile the entire system, including kernel code, and supports hardware performance counters, software performance counters, and tracepoints.

In this project, I mainly used perf to get access to hardware performance counters.

3 Goal

As a part of my overall project, I planned to build a tool to infer the energy consumption of an application at the software level, providing application, thread and function level granularity. Since RAPL does not provide high granularity in its energy measurements, I used hardware performance counters such as number of instructions, number of L1 and L2 cache misses and number of TLB misses for this purpose. The goal was to be able to predict the power consumption of an application when given the values of a subset of its performance counters.

4 Methodology

My project had two main aspects to it.

The first was to implement a suite of tests that I could run along with perf and RAPL in order to measure the energy spent per unit hardware event.

The second one was to use the energy numbers calculated in part one to estimate how much energy an application with a given hardware counter trace would be spending.

Event	Energy (in 10^{-7}joules)
L1 cache miss	2.096
L2 cache miss	3.556
TLB miss	7.659
Instruction	0.216

Table 1: Energy consumed by individual hardware events

4.1 Calculating energy spent per hardware event

I decided to measure the energy costs of an L1 cache miss, an L2 cache miss, a TLB miss, and an average instruction.

For this, I designed four main tests, each stressing the part of the CPU pipeline that I wished to measure.

The reason for choosing the mentioned counters was their high performance costs and the fact that they are very likely to occur in large numbers during program execution. In addition to these metrics, I also wanted to look at the number L3 cache misses, branch mispredictions and floating point instructions in the application code, but was unable to do so because of the following reasons:

- L3 cache misses: My benchmark that was supposed to trigger L3 cache misses was not causing the correct amount of L3 cache misses, and I was unable to figure out why in time for the deadline.

- Branch mispredictions: It is surprisingly hard to cause branch mispredictions! Out of a 1000 loops, I was only able to generate an average of 25 mispredictions, and hence I was unable to isolate the stats enough to calculate the energy expense of a misprediction.

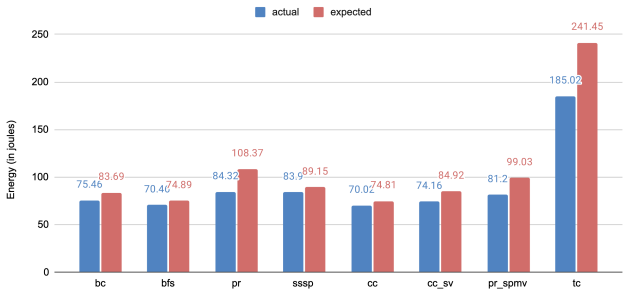

Figure 2: Measured v/s Expected energy values for gapbs

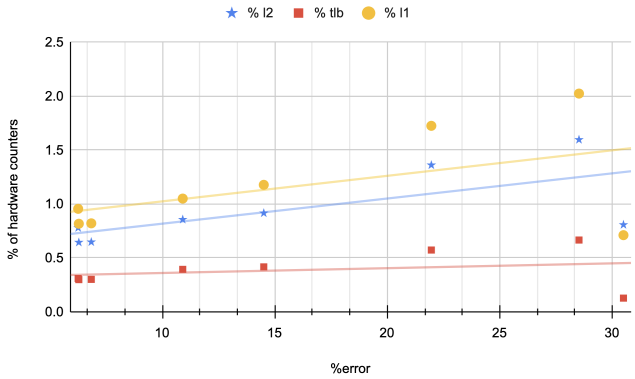

Figure 3: Type of instruction v/s error rate

- Floating Point Instructions: The machine I was using did not support the hardware counters for FP instructions.

My tests that isolate the specified counters are available in my code for the project, under the directory single-server-etrace/tests/msr.

In addition to writing this code, I also had to determine the perf events to use in order to calculate the desired events. I used the following perf events* for the specific event:

- L1 cache misses: L1-dcache-load-misses
- L2 cache misses: l2_rqsts.miss
- TLB misses: dTLB-load-misses
- Total instructions: instructions

* Note that these are case sensitive.

In order to measure the RAPL values for these benchmarks, I directly accessed the model specific registers for my processor. The code for this is also available in the files msr.h and msr.c in the directory single-server-etrace/tests/msr.

4.2 Using energy numbers to predict energy consumption

Once I had the per event energy numbers for the above mentioned events, the next step was to predict the energy consumption for a set of applications. I chose the GAP Benchmark Suite [3] to predict the energy numbers for.

In order to do so, I ran four out of the eight of the tests in the benchmark to get the amount of energy that was being consumed by the tests, and then use those numbers to assign weights to the hardware counters we mentioned above. This would lead to a linear model.

However, doing so resulted in *negative weights* for some parameters, which is incorrect since none of these metrics can result in a reduction in energy consumed by the program. Therefore, instead of going for an exact fit, I looked for weights that would provide a close estimation of the overall energy consumption. The simplest model provided the same weight for every metric, and that is the one I used for my evaluations.

5 Evaluation

For my evaluation, I used the following model to predict the energy consumption of the processes:

$$E = 0.1 * ((l1 * l1_0) + (l2 * l2_0) + (t * t_0) + (i * i_0))$$

Where $l1$ is the number of $l1$ misses the program had, and $l1_0$ is the energy (in joules) spent by one $l1$ cache miss as calculated by my work in Section 4.1. The same pattern has been followed for all other events, where $l2$ refers to number of $l2$ cache misses, t refers to the number of TLB misses, and i refers to the number of instructions. In this model, each event has a weight of 0.1.

Table 1 shows the energy consumed by each hardware event.

Figure 2 shows the measure v/s expected energy consumption values for the GAP BS benchmark. The highest error rate is observed for the ”tc” benchmark, with a value of 30.45%. The lowest error rate is for ”cc”, and has a value of 4.79% .

Figure 3 shows the percentage error on the x axis, and the percentage of the L1, L2 and TLB misses when compared to total events measured on the y axis. This graph shows two significant findings. First, that the L1, L2 and TLB misses combined make a very small percentage of the total events measured (about 3.5% maximum). This is probably because measuring the total number of instructions inflates the total events being measured, and might lead to some insights being lost due to it making everything else seem very insignificant. Second, this shows that the error rate increases with an increase in the

relative amount of L1, L2 and TLB misses when compared to total instructions. This can have many causes, but I suspect it is because L1, L2 and TLB misses impact each other, while I have been measuring them as independent events.

6 Future Work

There are multiple ways to take this work ahead. This also involves things I wasn't able to do because of time restrictions.

- Try out different models to predict energy consumption. This includes seeking out different weight values for each of the events, as well as considering non-linear models.
- Adding more events to the model. Currently, I only look at 4 events. An idea would be to consider more events, and then compare which ones are more important and provide a more accurate prediction for energy consumption.
- Trying this on a different processor. Just like Intel, AMD also supports RAPL[2]. I hope to be able to create a similar model for AMD processors as a part of future work.
- Analyzing the errors that these models have on architectures that do not support RAPL.
- Use this fine grained energy analysis in more complex systems (such as networked or distributed systems).

7 Reflections

I learned a lot of things from this project.

First off, it was quite challenging to do my first project in grad school all alone. When I came in, I wasn't very familiar with the worlds of energy and power measurement. While I am still no expert, I think I am a bit more comfortable in talking about things. It was frustrating at times to have nobody to ask questions too since I work alone, however, the moments when I uncovered things was very worth it. So, this project definitely helped me build my confidence (atleast a little bit), and also helped me practice how to figure things out on my own.

Second, it pushed me to learn more about measurement studies. I have always been terrified of measurement work because it is a perspective towards systems I wasn't taught about. I used to be scared to analyze why benchmarks were behaving the way that they were, especially since I have mostly worked with systems I had no knowledge of. However, this project showed me the power that measurement holds. It can help us detect that something is wrong, or that our understanding of a concept is either incorrect or blurred. It's true that numbers don't lie, and there's no better way than a measurement study to truly accept that.

Lastly, I learned a lot about hardware. While I have always been intrigued by hardware, I don't have a lot of experience working with it. As someone who took her architecture course during the pandemic, I think its safe to say I was starting to forget concept as time went by. However, this project really forced me to brush up on those topics, which enabled me to look at this project with a keener eye.

References

[1] Perf (linux), Jul 2021.

[2] AMD. Processor programming reference (ppr) for amd family 19h model 51h, revision a1 processors (pub).

[3] BEAMER, S., ASANOVIĆ, K., AND PATTERSON, D. The gap benchmark suite, 2015.

[4] CONTRERAS, G., AND MARTONOSI, M. Power prediction for intel xscale/spl reg/ processors using performance monitoring unit events. In *ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.* (2005), pp. 221–226.

[5] INTEL. Intel 64 and ia-32 architectures software developer's manual combined volumes 3a, 3b, 3c, and 3d: System programming guide.

[6] JONES, N. How to stop data centres from gobbling up the world's electricity. *Nature 561* (2018), 163–166.

[7] KHAN, K. N., HIRKI, M., NIEMI, T., NURMINEN, J. K., AND OU, Z. Rapl in action: Experiences in using rapl for power measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst. 3*, 2 (Mar. 2018).

[8] PEARCE, F. Energy hogs: Can world's huge data centers be made more efficient? *Yale Environment 360* (2018).