Measuring Energy Consumption for Short Code Paths Using RAPL

Marcus Hähnel, Björn Döbel, Marcus Völp, Hermann Härtig
Technische Universität Dresden
Dresden, Germany
{mhaehnel,doebel,voelp,haertig}@tudos.org

ABSTRACT

Measuring the energy consumption of software components is a major building block for generating models that allow for energy-aware scheduling, accounting and budgeting. Current measurement techniques focus on coarse-grained measurements of application or system events. However, fine grain adjustments in particular in the operating-system kernel and in application-level servers require power profiles at the level of a single software function. Until recently, this appeared to be impossible due to the lacking fine grain resolution and high costs of measurement equipment.

In this paper we report on our experience in using the Running Average Power Limit (RAPL) energy sensors available in recent Intel CPUs for measuring energy consumption of short code paths. We investigate the granularity at which RAPL measurements can be performed and discuss practical obstacles that occur when performing these measurements on complex modern CPUs. Furthermore, we demonstrate how to use the RAPL infrastructure to characterize the energy costs for decoding video slices.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Measurement

Keywords

Power Consumption, Operating Systems, RAPL

1. INTRODUCTION

The increasing popularity of mobile devices and the environmental concerns related to green and sustainable IT have qualified energy consumption as a primary concern in the design of electronic systems and of computer systems in particular. At the heart of many approaches towards energy efficient and energy proportional computing lie energy models that allow extrapolating the future energy consumption of the system. Such models are used to schedule applications and resources [11], to adapt application behavior to externally specified energy constraints [4], and to account energy usage to the respective software components [12].

To obtain an energy model, the energy consumption of one or more applications is measured by manually instrumenting the hardware in a lab and concurrently measuring hardware-level performance counters. The energy model then usually constitutes a linear regression model, incorporating the hardware performance counters that best correlate with the manual energy measurement. This approach has drawbacks: manual instrumentation is often done at the level of the computer's power supply [15], resulting in models that are too coarse-grained or inaccurate to model detailed application behavior. This can be leveraged by more fine-grained instrumentation of single processing units [2].

In addition to that, a study by McCullough and colleagues pointed out that linear regression models are inaccurate when it comes to modeling more complex computer systems and workloads [10]. In their paper, they argue that the only way to properly evaluate the energy characteristics of such components would be measurement features built into the platform that can be used by applications dynamically.

Our focus lies on determining the power consumption of short-running code paths, such as the energy cost of a single function within an application or of a system call of the underlying operating system. We believe this information will be helpful for a variety of tasks. As an example, imagine the following three scenarios:

1. *Video encoders* today optimize the encoded frames so that the decoded pictures come at a high ratio of quality per encoded byte. If we were able to measure the energy required to decode a single video frame or even its sub-slices, encoders could optimize for quality per watt or deliver a version of the video that contains a high-quality and a second one that can be decoded at a low-energy (e.g., on a smartphone). The runtime decoder might then decide which frame to use based on the currently available energy budget.
2. *Database systems* usually combine a set of available operators so that they calculate the result for a given query. There are many of such combinations and knowing the energy cost of every single operator may allow the database system to not only select the operator combination that computes a result as fast as possible, but also the one that consumes the least energy.
3. *Hardware devices*, such as hard disks or network interfaces, exist in different variants and can be run in different power modes [5]. However, when it comes to evaluating the energy required to execute a specific

GREENMETRICS '12 London, UK

workload on this device, measuring the device's energy consumption is not sufficient. Instead, we must also measure the energy that the driver of this device consumes in terms of CPU power. It may therefore well be that operating a low-power device might actually consume more energy than unoptimized devices if they cause the CPU and the driver to service more interrupts. A thorough evaluation of such scenarios however is only possible if energy can be measured as fine granular as a single interrupt handler.

Recently, the vendors of desktop and server class systems have equipped their systems with a new tool for obtaining fine-grain energy models: on board energy sensors for measuring the energy consumption of on-core hardware components and of the code that runs on these components. Intel introduced these sensors — called "Running Average Power Limit" (RAPL) — with their Sandy Bridge microarchitecture [6], AMD starting with their Bulldozer [1] CPUs[1].

In this paper we evaluate whether and how RAPL sensors can be used for measuring the energy consumption of short code paths. In Section 2 we describe HAECER, a measurement framework that we built into our operating system to allow application-specific energy measurements. In Section 3, we then go on describing the obstacles we had to overcome to obtain results for small code paths. Section 4 presents first results about slice-accurate energy models for video decoding that we obtained with our framework.

2. RAPL MEASUREMENTS

To obtain exact energy measurements at the granularity of short source code blocks, we ideally would like to get an exact reading of the system's consumed energy at the start and the end of a measurement time frame. In this section, we take a closer look at how Intel's RAPL sensors work (Sec. 2.1), whether they are accurate enough to perform short-time measurements (Sec. 2.2), and how we leverage them in the HAECER measurement framework (Sec. 2.3).

2.1 How RAPL Works

Intel introduced the *Running Average Power Limit* (RAPL) feature with the Sandy Bridge microarchitecture. RAPL is available in newer versions of the Xeon server-level CPUs and provides sensors that allow measuring the power consumption of the CPU-level components listed in Table 1. These available counters limit measurements to CPU and memory controller power consumption. It is impossible to measure energy consumption of I/O devices, but we hope that device vendors will follow Intel's lead and provide energy-related information through their device-specific interfaces.

`RAPL_PKG`	Whole CPU package
`RAPL_PP0`	Processor cores only
`RAPL_PP1`	"A specific device in the uncore"
`RAPL_DRAM`	Memory controller

Table 1: List of available RAPL sensors

RAPL sensors can be configured and examined by reading Machine-Specific Registers (MSRs). On the Intel architecture this is only possible in privileged kernel mode. Hence, we require kernel-level support for energy readings.

[1]At the time of writing this paper, no Bulldozer system was available for us. We will therefore limit ourselves to the Intel RAPL sensors. Extending our approach to the AMD Bulldozer sensors should be straight forward.

2.2 How accurate are RAPL counters?

Intel reports an update rate of the RAPL MSRs once every 1 ms [7, Ch.14]. To perform exact energy measurements at the beginning of short code paths, we therefore need to align the start of this code path to an update point. Unfortunately, updates of RAPL counters do not occur accurately every millisecond, which would have allowed us to delay the beginning of the short code path to the next multiple of 1 ms.

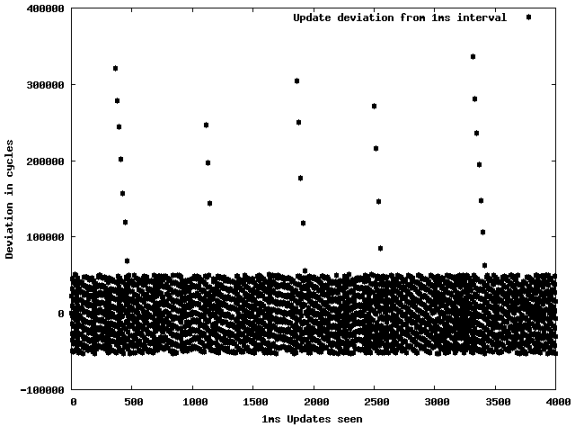

Figure 1: Distribution of updates relative to the specified 1 ms interval

Figure 1 shows the deviation of RAPL counter updates from once every millisecond. We evaluated this update accuracy on an Intel Core i5-2400S CPU running a tight loop at 2.5 GHz in which we continually read the time stamp counter and the RAPL MSR. Updates are expected every 2,500,000 cycles. A deviation of 0 means that an update was seen exactly at a multiple of 1 ms after the start of the benchmark. For values larger than 0 the update occurred after the 1 ms instant, for negative values the update was seen before. We can see that updates are not accurately timed, but jitter in the range of +/- 50,000 cycles. This deviation is high enough to make short-term energy measurements significantly inaccurate. In addition to the mentioned MSR update inaccuracy, we see a small amount of measurements that deviate by more than 100,000 cycles. We explain where these come from and how we handle update inaccuracies in Section 3.

To gain confidence in the RAPL hardware, we also compared the energy consumption reported through RAPL measurements with a measurement we performed on a manually instrumented board kindly provided by TU Dresden's HPC Centre (ZIH). In Figure 2, we see the consumption reported by RAPL compared with the external measurement for a synthetic benchmark that switches between highly utilizing the CPU and sleeping every second. Although there is an offset of nearly 4 W between the curves we see that the curves' characteristics are identical. We attribute this offset to the fact that the external measurement includes addi-

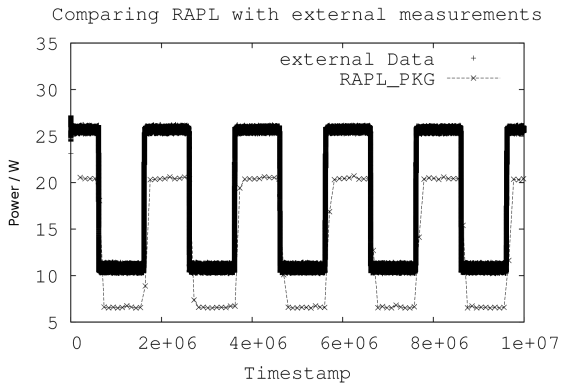

Figure 2: Comparison of RAPL and external measurement

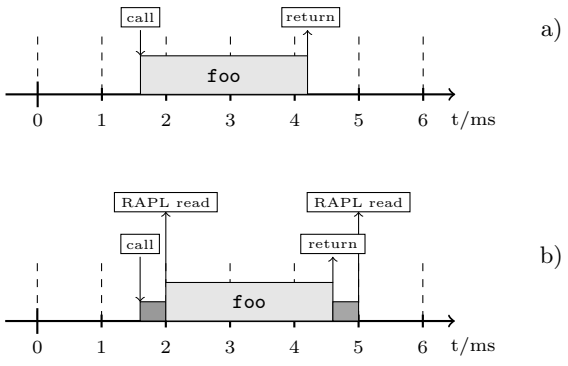

Figure 3: Adjusting measurements to accurately measure short-term energy consumption

tional energy consumers such as for instance the DRAM, which the RAPL hardware do not cover[2].

2.3 HAECER - A framework for short-term energy measurements

Our group develops the L4/Fiasco.OC microkernel [16]. We performed the measurements documented in this paper on top of Fiasco.OC. However, we are convinced that our methodology can equally be applied to other operating systems, such as Linux. As RAPL MSRs can only be read in kernel mode, we added a `read_rapl` system call to the kernel and extended the application-level event monitoring framework Ferret [13] to support also low-latency short-term energy measurements. The resulting framework we call the HAEC Energy Reader — HAECER.

HAECER provides convenience functions for the inherently CPU version dependent setup and initialization of performance counters. Furthermore, HAECER handles obstacles that stem from timing and measurement inaccuracy.

Instead of trying to perform fine-grained measurements using RAPL, we could also implement synthetic microbenchmarks that execute the code we are interested in for thousands of times and then compute power consumption using a coarse-grained measurement. While this works if we are only interested in the cost of executing instructions, it becomes infeasible for the scenarios we described in Section 1: These pieces of software do depend on the data they are currently handling. We will see in Section 4 that the same function for decoding a slice of a video frame consumes different amounts of energy depending on which video data is being processed. To catch this behavior in a microbenchmark, we would have to set up a proper environment for every such benchmark. Our framework allows instrumenting arbitrary applications regardless of such characteristics.

3. MEASURING SHORT-TERM POWER CONSUMPTION

Suppose we want to measure the energy consumption of a function `foo` by manually instrumenting its start and end. Figure 3 a) illustrates the problems we are now facing. The call to `foo` may happen at any instant within a RAPL update interval. (Dashed lines in Figure 3 represent those updates.) If we simply read the RAPL MSR at this point, we read the value from the last update, which means we may read a value that may be up to one update interval too old. Accordingly, if we instrument the end of this function by instrumenting `foo`'s return with a read of the RAPL MSR, we may again read a value that is up to one update interval too old. This may lead to worst-case scenarios:

1. If the call to and return from `foo` fall into a single update interval, the obtained difference in energy consumption will be zero, because we will read the same (inaccurate) MSR value twice.
2. If the call to `foo` happens shortly before an update, we will read an old value and over-estimate foo's energy consumption.
3. If `foo` returns short before the next RAPL update, we will read a too old value and underestimate `foo`'s energy consumption.

We solve these issues in HAECER by aligning instrumentation points with RAPL updates as illustrated in Figure 3 b). The function for measuring the start of a code path begins with a loop that reads the RAPL MSR until it sees a change in the obtained value. Then it goes on to execute the function `foo`. Thereby we make sure that we start measuring the energy consumption at the beginning of an update interval.

We instrument the end of the code path in a similar way by inserting a delay loop that reads the RAPL MSR until it sees a modification. Thereby we measure the energy consumption until the next RAPL update. If we can control the energy spent between the actual return from `foo` and the next update, we can simply subtract this energy from the measured value and thereby get the exact value we would like to measure. To synchronize for the next RAPL update we have to loop reading the RAPL MSR. We evaluated the energy cost of this loop using a microbenchmark and determined that the energy cost per clock cycle remains constant across several executions. Therefore, we can simply count the number of `rdmsr` operations until the next RAPL update and compute from these the energy consumed in the delay loop.

The exact value to subtract for the delay loop of course varies depending on the actual hardware the measurement is performed on and on environmental situations such as tem-

[2]The DRAM sensor documented by Intel is only available in newer versions of the Xeon server-level CPUs.

perature. To remain independent from the test machine, the HAECER framework comes with an initialization function that calibrates the average energy spent for instructions in the RAPL synchronization loop.

In Section 2.2, we observed a small amount of RAPL updates that occurred significantly later than the average updates. We further investigated this issue and repeated the microbenchmark in user and kernel mode, as well as with interrupts disabled to rule out OS-level effects that might step in – the observation remained the same. Therefore, the only viable explanation for us is that this delay is introduced by the CPU switching into System Management Mode [7, Ch.3]. This mode is used internally by the CPU to perform maintenance tasks, such as thermal management and certain legacy device emulations. SMM completely defies itself from control by the operating system[3]. Our experiments show that SMM is entered periodically about every 16 milliseconds on our test machine. To rule out SMM side effects on our short-term measurements, we extended the delay loop at the start of our measurements to wait for the entry into SMM. This left us 16ms to perform our measurements.

Together, the mechanisms explained above allow us to accurately obtain RAPL measurements for the start and end of a short code path. In the next Section we will validate this claim by inspecting the energy consumption of slice-level video decoding. We are aware of two limitations, though: First, the delay loops of course add execution overhead. Therefore, our measurements cannot be left always-on and should not be combined with time measurements. Second, our measurements are accurate only as long as we run a single instrumented application. Enhancing our approach to account for multiple concurrent applications would require modifications to the scheduler and potentially expensive multiplexing of the RAPL sensors. While we expect this to be feasible, we leave it for future work.

4. CASE STUDY: SLICE DECODING ENERGY CONSUMPTION

With the possibility of fine-grained energy measurements in place, we now report on our experiment for evaluating our framework with one of the scenarios discussed in Section 1: slice-granular decoding energy.

Decoding high-resolution videos may require a substantial amount of CPU power even in today's systems. Moreover, video decoding is a common use case for embedded systems, such as mobile phones, where both, the CPU and the available battery power, are a limited resource. Modern video decoding formats such as H.264 [8] are prepared for parallel decoding while maintaining the high compression rates from exploiting the interdependence of subsequent images (called frames). To parallelize frame decoding Wiegand et al. [18] suggests splitting video frames into so called *slices* and decoding slices on the available CPUs. Roitzsch [14] improves on Wiegand's equal split by predicting slice decoding times to better balance the load to the different CPUs or a multicore system.

We believe that larger energy savings become possible when the majority of the slice decoding cores can be kept at low frequencies and hence at low supply voltages [17]. However, this requires accurate information about video decoding energy at the granularity of the individual slices. For a 24 frames-per-second video (i.e., one frame every 40ms), the slice decoding time with 10 slices per frame is already in the order of 4ms. This simple calculation already qualifies slice decoding as a short code path. In reality, slice decoding may be well below one millisecond, which, as we have seen, is below the granularity of RAPL energy sensors.

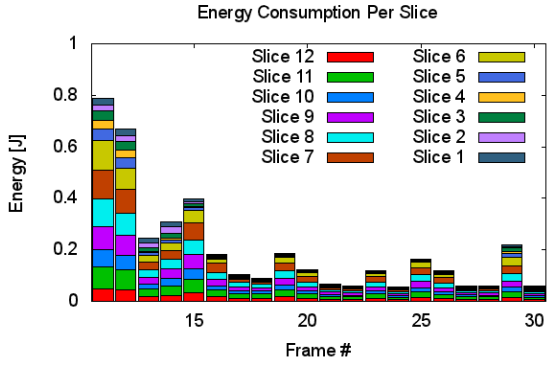

Figure 4: Per-slice energy consumption for H.264 video frames

For our video experiments we ported the instrumentation done by Roitzsch in [14] to a recent version of the `ffmpeg` decoder [3]. We performed our measurements on the trailer for "The Avengers" [9]. The video is 31 seconds long and has been encoded with 12 equally-sized slices per frame. We decoded the video in a single-threaded version of the decoder on an Intel Core-i5-2400S CPU.

Figure 4 shows the per-slice energy consumption measured with the help of our framework[4]. The graphs illustrate that: 1. The energy used for decoding a frame (the sum of 12 slices) varies between 0.1 W and 0.8 W, and 2. The energy used for decoding a single slice (parts of the bars) varies as well. From this we conclude that our assumption that slice decoding energy varies is correct and that research on energy-balanced slice-based video decoding appears to be promising.

To gain trust in the slice-level values presented in Figure 4, we need to validate that these values match the higher-level energy consumption and are not disturbed by energy consumed by our instrumentation. We verified this by instrumenting the same video at the frame level, leaving out all slice-level instrumentation code. Figure 5 shows the frame-granular energy consumption in relation to the sum of slice-level values obtained in the previous experiment. As the difference is not perceptible from this Figure, we additionally plot the relative error between the two measurements (Figure 6). The average relative error per frame is 1.13%, the maximum 3.6% and the minimum 0.5%.

5. CONCLUSIONS

In this paper we evaluated the suitability of Intel's RAPL energy performance counters for performing fine-grained energy measurements on short code paths. We explained the obstacles we encountered when trying to evaluate the energy consumption of code blocks that execute faster than the average RAPL update frequency. We implemented HAECER,

[3]Plus, disabling SMM may break the CPU.

[4]We only show a few frames here for visibility reasons.

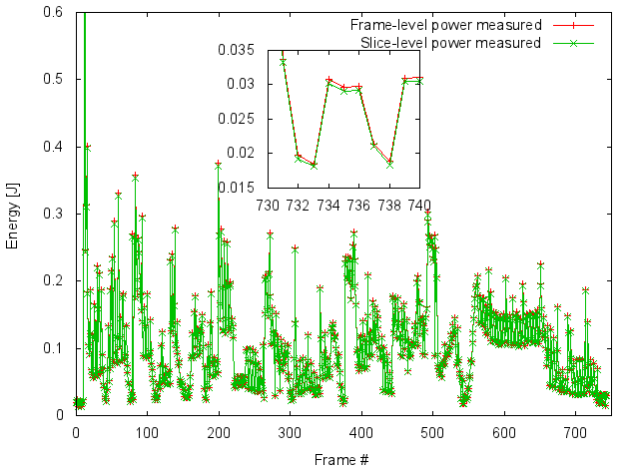

Figure 5: Energy consumption measured at the frame level compared to slice-level measurements

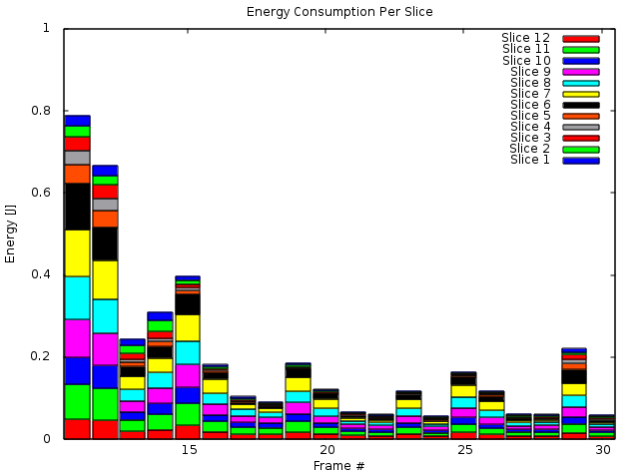

Figure 6: Relative error between slice-level and frame-level energy measurements

a framework for instrumenting applications and performing fine-grained energy measurements at sub-update frequencies and demonstrated the applicability of HAECER for an accurate evaluation of video slice decoding energy.

Acknowledgments

This work was partially funded by the German Research Council (DFG) through the Collaborative Research Center CRC 912 "Highly-Adaptive Energy-Efficient Systems" (HAEC) and the Special Purpose Program "Dependable Embedded Systems" (SPP 1500).

We would like to thank our colleague Michael Roitzsch for his advice on video decoding and on evaluating slice-based energy-consumption.

6. REFERENCES

[1] Advanced Micro Devices. *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors*. 2012.

[2] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Annual Technical Conference*, pages 1–14, Boston, MA, USA, Jun 2010.

[3] FFmpeg project. `http://www.ffmpeg.org`.

[4] J. Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Trans. Comput. Syst.*, 22(2):137–179, May 2004.

[5] J. Haartsen, M. Naghshineh, J. Inouye, O. J. Joeressen, and W. Allen. Bluetooth: vision, goals, and architecture. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(4):38–45, Oct. 1998.

[6] Intel Corp. Intel Xeon processor. `http://www.intel.com/xeon`, 2012.

[7] Intel Corp. *Intel® 64 and IA-32 Architectures Software Developer Manual*. 2012.

[8] ISO/IEC 14496-10. Coding of audio-visual objects, part 10: Advanced video coding.

[9] Marvel Studios. The Avengers – Big Game (31s). `http://trailers.apple.com/trailers/marvel/avengers/`.

[10] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta. Evaluating the effectiveness of model-based power characterization. In *Proceedings of the 2011 USENIX Annual Technical Conference*, USENIX ATC'11, Berkeley, CA, USA, 2011. USENIX Association.

[11] A. Merkel and F. Bellosa. Balancing power consumption in multiprocessor systems. In *Proceedings of the 1st ACM SIGOPS European Conference on Computer Systems 2006*, pages 403–414, New York, NY, USA, 2006. ACM.

[12] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM Europ. Conference on Computer Systems*, pages 29–42, New York, NY, USA, 2012. ACM.

[13] M. Pohlack, B. Döbel, and A. Lackorzynski. Towards runtime monitoring in real-time systems. In *In Proceedings of the Eighth Real-Time Linux Workshop, Lanzhou, P.R. China*, 2006.

[14] M. Roitzsch. Slice-balancing H.264 video encoding for improved scalability of multicore decoding. In *Proceedings of the 7th International Conference on Embedded Sofware, Salzburg, Austria*, EMSOFT'07. ACM, 2007.

[15] D. C. Snowdon, S. M. Petters, and G. Heiser. Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *Proceedings of the 7th International Conference on Embedded Software*, pages 84–93, Salzburg, Austria, Oct 2007.

[16] TU Dresden OS Group. L4/Fiasco.OC microkernel. `http://www.tudos.org/fiasco`, 2012.

[17] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[18] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits Syst. Video Techn.*, 13(7):560–576, 2003.