Experimental Energy Profiling of Energy-Critical Embedded Applications

Kameswar Rao Vaddina*, Florian Brandner*, Gerard Memmi*, Pierre Jouvelot†
*LTCI - TÉLÉCOM ParisTech - University of Paris-Saclay, Paris, France
†MINES ParisTech, PSL Research University, Paris, France
Email: {kameswar.vaddina, florian.brandner, gerard.memmi}@telecom-paristech.fr,
pierre.jouvelot@mines-paristech.fr

***Abstract*—Despite recent advances that have greatly improved the performance of embedded systems, we still face many challenges with regard to energy consumption in energy-constrained embedded and communication platforms. Optimizing applications for energy consumption remains a challenge and thus is a compelling research direction, both on the practical and theoretical sides. This paper presents a new experimental bench for energy profiling of non-performance-critical embedded and mobile applications and reports preliminary results obtained on two embedded boards. The experiments are driven by an online energy monitoring mechanism using National Instruments' cDAQ and LabVIEW running on a host machine. The host monitors a target device, which runs a set of benchmarks. We describe the experience gained from using and modding two different target boards, namely an Nvidia Jetson TX1 and a TI AM572x evaluation module. In particular, we confirm, and thus further validate, the existence of the Energy/Frequency Convexity Rule for CPU-bound benchmarks. This rule states that there exists an optimal clock frequency that minimizes the CPU energy consumption for non-performance-critical applications. We also show that the gain of frequency scaling is highly dependent on workload characteristics. Any future energy-management approach should take these behavioral traits into consideration.**

I. Introduction

Continuous CMOS technology scaling (Moore's law) increases the on-chip power density due to the higher transistor integration. As power density increases, many factors like power dissipation, leakage, data activity, and electro-migration contribute to higher on-chip temperatures. The increase in temperature leads to an increase in leakage power, thereby increasing the total energy dissipation and thus forming a part of a vicious circle significantly limitting system performance.

The bulk of today's computing does not happen on desktops, laptops, servers, or data centers, but rather on embedded media devices like mobile phones [1]. The embedded computing applications running on those devices demand better energy efficiency and flexibility in operation, while delivering better performance per Watt. At the same time, they cannot compete with application-specific integrated circuits (ASIC) in terms of energy efficiency. Indeed, a well-designed ASIC can achieve an efficiency of 5 pJ/op in a 90-nm CMOS process, whereas a very efficient embedded processor would require about 250 pJ/op. That means the embedded processor may consume about 50 times more energy than a custom designed ASIC [1].

Today's system-on-a-chip (SoC) platforms have a lot of software acting in unison, trying to deliver a seamless user experience. The firmware, operating system, device drivers, system-level software, and applications coordinate with each other and with the hardware. All these pieces of software have some impact on the system power. It is thus imperative to observe and measure the impact of software on power requirements, so that developers can benefit from this information and optimize their code for energy-critical systems.

The system-wide analysis of energy issues in embedded systems calls for suitable experimental and theoretical tools. Our paper makes the following contributions in that regard.

- We describe a new experimental platform for a highly accurate on-line measurement mechanism that allows us to perform energy monitoring and profiling of embedded and non-performance-critical applications.
- We show how profiling can be done by physically measuring the power on the power rails, i.e., the output of the power management unit of the target board, with the help of precise sensors and data acquisition devices.
- We detail how our measurement setup is different from several other setups found in the literature (see for instance [2], [3], and [4]) in that they have direct access to the CPU power values. Other setups often only provide access to platform-level energy consumption, which includes the energy consumed by the peripheral devices, power management unit, GPU, memory, et cetera. A more complete state of the art can be found in [6].
- We detail the practical experience we gained from trying to adapt our two embedded target boards in order to interface with our probe and data acquisition infrastructure.
- Our tests confirm the existence of the Energy/Frequency Convexity Rule discovered in our previous work [2]. This rule states that the curve relating energy consumption to frequency exhibits a convex behavior, with an optimal frequency that minimizes energy, for compute-bound applications. We also show that the energy consumption is highly dependent on the workload characteristics.

We revalidate the Energy/Frequency Convexity Rule using much more precise methodology and tools than in our previous work [2], [6], which is of interest on its own. The level of accuracy achieved in this work will allow us in the future to precisely correlate the segments of a software program to its energy trace. This will help us to understand the program

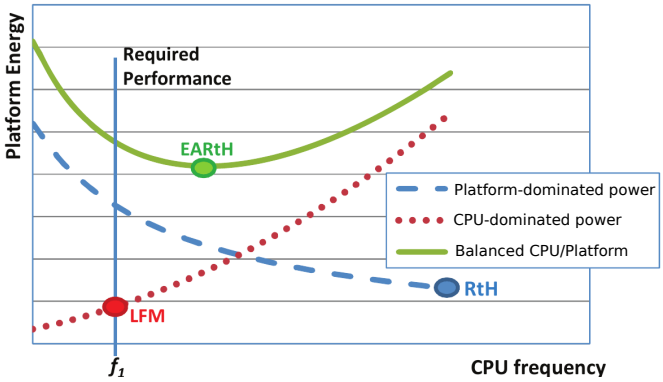

Fig. 1. Conceptual view of platform energy consumption by Rotem et al. [5].

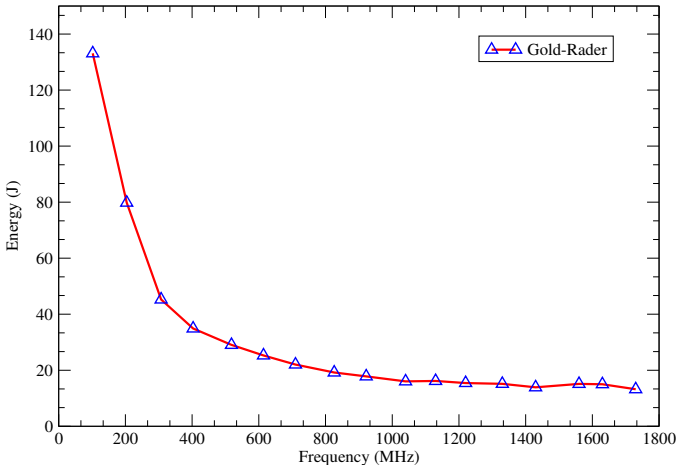

Fig. 2. Energy vs. Frequency: energy consumption for the Gold-Rader bit-reverse algorithm with varying clock frequency on an Nvidia Jetson TX1.

behavior better, allowing us to make fine-grained energy optimizations. We also believe that our paper will be of use to researchers interested in delving into the realm of practical, energy-related testing for embedded systems.

The remainder of this paper is structured as follows. In Section II, we describe the conceptual view of total platform energy consumption, i.e., the consumption of the CPU and the rest of the platform. We then describe (Section III) the design constraints of power management units and the challenges faced when handling the voltage signal. Section IV covers the experimental setup for the TI AM572x board. The results obtained by running various benchmarks are discussed in Section V. Finally, we conclude in Section VI and provide remarks on future work.

II. Platform Energy Consumption

Rotem et al. present a conceptual view of the energy consumption of a whole platform [5], which includes the energy consumed by the CPU and the rest of the board (platform). The model results in a curve, relating CPU/platform energy consumption to clock frequency, as illustrated by Figure 1. When the CPU power dominates the total power requirements, then the energy follows the red dotted curve and the minimum energy is achieved when the CPU operates in the Lowest Frequency Mode (LFM). In their model, they show that CPU power increases steadily, but we show that, in fact, the CPU power already exhibits a certain degree of convexity (see below). But, when the rest of the platform dominates the total power, then the energy follows the blue dashed line. In this case, the most energy-efficient strategy would be Race-to-Halt (RtH) [7], i.e., the processor operates at maximum frequency in order to complete the computation as fast as possible. However, when the power is balanced between the CPU and the platform, then the energy curve follows the green solid line. In this case, the energy consumption is minimized when the processor operates at an optimal frequency somewhere between the LFM and the RtH strategies. Rotem et al. [5] propose a technique to identify this minimum CPU/platform energy point at run time and call their approach Energy-Aware Race to Halt (EARtH).

In our pursuit to confirm and further validate the Energy/Frequency Convexity Rule, we started our experiments on the Nvidia Jetson TX1 platform, which consists of a carrier board and a processing module with the actual TX1 SoC. Only the schematics of the carrier board are public, while the module's schematics are not released. Even worse, there is no physical access to the CPU's power management unit on the module, making it impossible to place our measurement probes without permanently damaging the TX1 module. So, we measured the total energy consumption of the entire carrier board (CPU and platform). As can be seen in Figure 2, the platform energy dominates the CPU energy consumption. Similar to Rotem et al. (Figure 1), no convexity can be seen. This is not surprising, since the Energy/Frequency Convexity Rule has only been validated for CPU-bound computations.

III. Measuring PMU Output

A Power Management Unit (PMU) is a discrete integrated circuit (or a system block of a SoC) that is used to manage the power delivery requirements of a computer system. A typical PMU block diagram is shown in Figure 3. The PMU usually regulates the supply voltage, selects the sequencing of power sources, and even handles battery management (state of health and charging, voltage, current, and temperature). They generally offer two power-saving techniques, namely Dynamic Voltage Scaling (DVS) and Dynamic Frequency Scaling (DFS). The combination of both techniques is known as Dynamic Voltage and Frequency Scaling (DVFS).

The voltage regulator is an integral and very important part of the PMU. It operates in a feed-forward loop and allows to either increase or decrease the voltage at its output ports according to sensor feedback. The sensing is done with the help of two internal and external sense pins, which allow the regulator to monitor the output load and then act in order to make sure that the voltage stays at the proper level.

Fig. 3. A typical power management unit.

A. Physical Access

We first looked at the PandaBoard single-board development platform in order to study how to tap into the power rails going to a CPU. There are two PMUs on PandaBoard. But, while trying to attach the measurement probes by de-soldering an inductor at the output of the PMU, the PMU got dislocated and the subsequent soldering of the PMU onto the pads could not make the PandaBoard boot (a lesson in the need to practice on cheap components first). Following this practice run, we acquired the very recent TI AM572x board, which has been used for the rest of the experiments described in this paper.

The interplay between the PMU and the overall system design can be confusing at times and makes accurate energy measurements a difficult endeavor. Manufacturers are not required to publish board schematics. This is problematic in academic research, even more in our case, since we need direct access to the power rails for our measurements. In addition, we need to be able to control the PMU's configuration, in order to obtain meaningful results. These schematics are held private by the board manufacturers in most cases. Access is typically subject to negotiations and non-disclosure agreements.

Downstream of the PMU (e.g., CPU, GPU, DDR), the design constraints on the board tend to be very tight. PMUs impose strict constraints on the reactance as well as capacitance of power rails. This in turn imposes strict design constraints that need to be respected by the designers. Most of them react to this by burying the power rails in the board's mid layers in order to have strong control over the rail capacitance, thereby denying developers access to the power rails. Even if one did have access to the board schematics, located the power rails, and somehow managed to physically access them, the system would be unlikely to still meet the PMU's design constraints afterwards. Ultimately, after all these efforts, one would risk merely to end up with a broken board.

B. Wiring Inductance

Adding any wiring by means of shunt resistance and a voltmeter across it leads to additional inductance, which induces noise into the voltage signal as the load changes. Such additional inductance of the wiring and the voltmeter means that the voltage regulator can no longer properly sense the load. This often causes the embedded system to crash.

C. Signal Conditioning

The problem of attaching the monitoring probes to the PMU circuit can be seen as a signal-conditioning problem. Consider, for instance, the ARM Cortex-A9 processor of the PandaBoard. Depending on load, this processor operates in a range from 500 mV to 1.5 V. For instance, in order to operate correctly, the power regulator needs to supply at least 500 mV when the processor is idle. If we add any load (e.g., measuring instruments), the resulting voltage drop impacts the input voltage of the ARM processor. This will shutdown the processor/SoC, as the input to the processor cores will not be the expected 500 mV.[1] After careful analysis of the datasheets of the SoC, NI modules (input impedance 12 mOhms), and regulator, we concluded that it is safe to use NI modules at the regulator outputs (input to the ARM) as the voltage drop across the modules is negligible.

There are alternative ways to assess the amount of power required by the CPU. One such method involves measuring current at the input of the regulator and calculating the input-side power. According to the conservation of energy principle, the input power should be equal to output power. That means that the input power added to the power of the regulator should be equal to the output power. The datasheet provides the efficiency of the regulator for different output voltages (the efficiency of the regulator decreases at lower output voltages). Thereby, we could get the power required by the processor by simply multiplying the input power by the efficiency of the regulator. This assumes that the CPU is the only subsystem module being supplied by the PMU. If there are other attached subsystems (like memory, image and video processing units, DSPs), the amount of energy consumed by them needs to be accounted for in order to get the CPU power trace.

D. Temperature Issues

The shunt resistor is sensitive to temperature variations. If the temperature increases due to the passage of high currents, a phenomenon known as Joule self-heating occurs. This self-heating of the shunt resistance increases its resistance, which in turn leads to an increased voltage drop across it. This might eventually cause the SoC to shut down and may also impact the measurements. The chemistry of different shunt resistors also act differently with the increase in temperature.

Moreover, with an increase in temperature, the leakage power of the SoC increases exponentially, thereby contributing to the total power requirements. This does not affect the measurements per se, but helps to understand the power numbers obtained from profiling.

IV. Experimental Setup

Our main experimental setup consists of an AM572x EVM development board from Texas Instruments (TI) that is equipped with a high-performance Sitara™ SoC. The SoC is implemented using a 28-nm process and is comprised of several subsystems. In this work, we are interested in the microprocessor unit (MPU) subsystem, which, in turn, consists of two ARM Cortex-A15 cores. The AM572x EVM board offers current sense resistors for all its submodules, including the MPU. These resistors provide access to the power supply rail and allow continuous power monitoring in real time during software

[1] Assuming a drop of 120 mV due to the measurement probe, the supply would only be 380 mV, depending on the load.

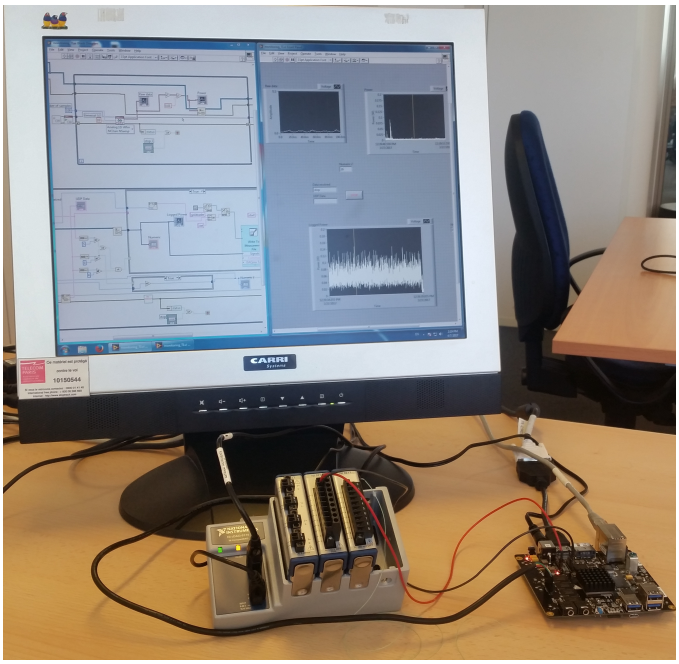

Fig. 4. The host machine running LabVIEW, the NI cDAQ along with the NI 9215 voltage input module, and the TI AM572x EVM baseboard.

execution. We modified the board by soldering male headers across the current sense resistors to easily connect probes. The resistor's value is 0.01 Ohm and has been chosen to provide the best possible dynamic range during data acquisition.

We use a compact data acquisition device from National Instruments (NI cDAQ-9174) for all our data acquisition needs. NI CompactDAQ is a portable platform that integrates connectivity and signal conditioning into I/O modules that can directly interface with many different sensors. We use an NI 9215 voltage input module to measure the voltage drop across the sense resistor. We then calculate the current using Ohm's law and multiply it with common-mode voltage to get power values. Figure 4 shows our complete experimental setup, which includes the Windows host machine running LabVIEW software, the NI cDAQ with the NI 9215 voltage input module plugged in, and the TI AM572x development board. The NI cDAQ shown in Figure 4 also includes an NI 9227 current input module and an NI 9211 temperature module, which have not been used in our experiments yet.

A. Host-to-Board Synchronization

Our monitoring platform (see Figure 5) relies on the LabVIEW software, which is able to monitor the target board in real time, while logging all measurements (along with additional meta-information). This allows us to obtain a complete trace of the energy consumption during a benchmark run. However, the host and target board need to be synchronized in order to reliably determine the start and end of a benchmark run and consequently start/stop data logging.

Usually, the processes running on the Windows host and the target embedded system could affect the precise timing

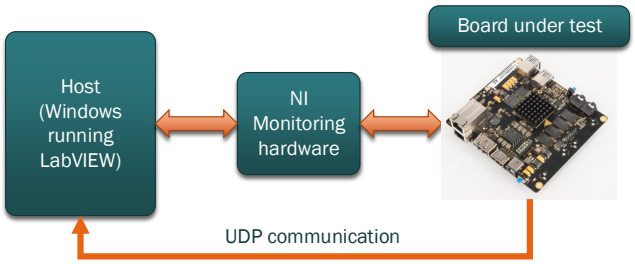

Fig. 5. Block diagram of our energy monitoring platform.

and reliability of the measurements. If the time characteristics of the measurement and automation systems are critical for the safety of the application, then one needs to use a real-time operating system (RTOS). But, in our case, there are no other tasks running on the host system that could impact the timing and reliability of the measurements. A standard Windows host thus suffices for our needs. In addition, we implemented a User-Datagram-Protocol-based (UDP) interface, which allows us to remotly start/stop the data logging process on the host.

The Windows host runs LabVIEW, which acquires and analyzes the measurement data coming from the cDAQ modules. Native to LabVIEW is a visual programming language (G) that uses a dataflow model instead of more traditional sequential lines of code. We used a producer-consumer model to code the host-side data acquisition logic. That means we continuously acquire data from the NI cDAQ, but only start logging when we receive a *start* message from the embedded system via UDP. We stop logging when we receive a *stop* message.

B. Benchmarking and Data Acquisition

For all our measurements, unrelated software and services (WiFi, window manager, services, ...) have been disabled in order to minimize interference with data acquisition. The benchmark runs are controlled by a bash command-line script running on the target board. The script first changes the scaling governor for a single CPU core from *ondemand* to *userspace*. This allows us to manually control the CPU's clock frequency and supply voltage settings. The TI AM572x EVM board, by default, only supports three frequency/voltage settings: OPP_NOM, OPP_OD, and OPP_HIGH. Each of these settings corresponds to a different supply voltage (1.06 V, 1.16 V, and 1.21 V, respectively). The frequency at each of these voltage levels corresponds to the maximum possible value (1000 MHz, 1176 MHz, and 1500 MHz). However, additional frequency settings can be enabled by adding corresponding entries to the Device Tree Source (DTS) file.

A device tree is a tree data structure of *nodes* and their *properties*. Properties are usually key-value pairs, and nodes may contain both properties and child nodes. The device tree describes the physical devices in the system that cannot be dynamically detected (I2C, SPI, flash, bus). The tree is constructed from a binary blob (Device Tree Blob, or DTB) that is parsed by the kernel at boot time. The blob in turn is derived from board/SoC-specific DTS files.

We modified the DTS file by adding corresponding entries (properties) to the *operating point table* child node, the properties of which include *operating point frequency* and *operating point voltage*. This file is then cross-compiled into a DTB file and then flashed onto the boot disk, allowing us to perform measurements while varying the clock frequency in steps of 100 MHz between 100 MHz and 1500 MHz.

The bash script starts the data logging process on the host machine by sending a *start* message before setting the desired scaling frequency. The sampling rate used is 1600 samples/sec. With the help of the *taskset* command, the script controls CPU affinity and forces the benchmark to run on a specific core. The script finally sends a *stop* message to the host machine after the benchmark completes. This terminates the logging process and prepares the LabVIEW software for another benchmark run. Once all benchmark runs for all frequency settings have been completed, the frequency scaling governor is set back to *ondemand*.

For our experiments, we use two cryptographic benchmarks, namely SHA and Blowfish, from the BEEBS suite [8] and the *bit-reverse* algorithm. The bit-reverse algorithm is the part of the ubiquitous Fast Fourier Transform (FFT) algorithm that deterministically rearranges elements in an array. We used the Gold-Rader implementation [9], which is often considered the reference algorithm for FFT applications. The benchmarks are run 3 times and it has been found that there are no significant variations in terms of energy consumption.

V. Discussion

We primarily investigated different platforms and workloads in order to evaluate some of the commonly used energy-management heuristics. For this, we measured various characteristics (execution time, average power, ...) while varying the clock frequency in steps of 100 MHz between 100 MHz and 1500 MHz. Here, we present our main findings.

Figure 6 shows the performance of the benchmarks, i.e., execution time, depending on the CPU's clock frequency on the TI AM572x platform. The execution time decreases in a clear non-linear fashion for our CPU-bound applications as the frequency increases. This is hardly surprising and underlines the importance of frequency scaling, including DFS, for performance. While the energy consumption of the SoC, when running a benchmark, also depends on the benchmark's execution time (recall that leakage power and other factors that are independent of CPU load may play a role here), the clock frequency may have a more significant impact.

Figure 7 shows that the average power per millisecond increases steadily with the frequency for all three benchmarks (SHA, Blowfish and Gold-Rader) in an almost-linear fashion. This accounts for the power of the entire MPU subsystem, i.e., both ARM cores, where one core is running the benchmark and the other is idle – but not powered off. Note that the subsystem also includes the core's private L1 cache, a shared L2 cache, and the necessary interconnect. These components cannot be controlled separately in software, but may transition themselves into low-power modes independently from the

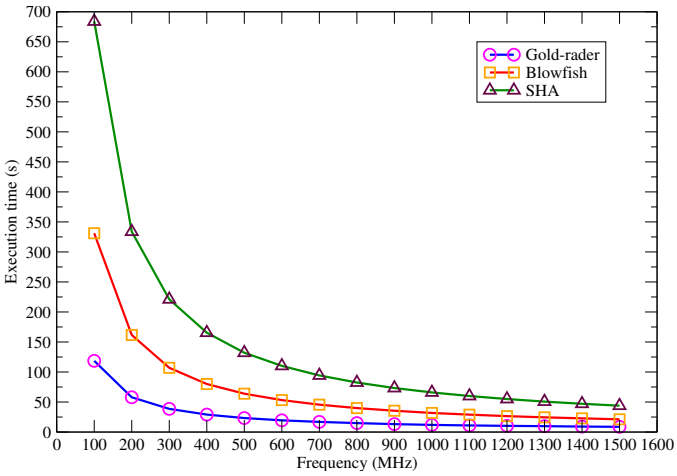

Fig. 6. Execution time vs. Frequency: performance of the benchmarks with varying clock frequency on the TI AM572x platform.

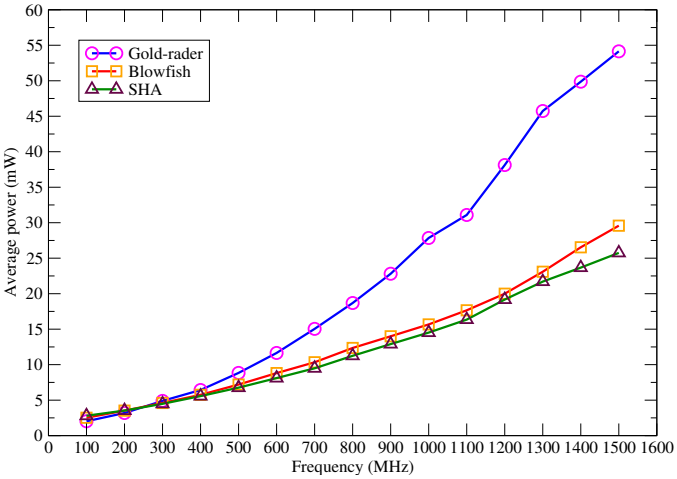

Fig. 7. Average power vs. Frequency: Power consumption of the benchmarks with varying clock frequency on the TI AM572x platform.

CPUs. Reducing the clock frequency minimizes the average power, but this may not be the best strategy, as the non-linear increase in execution time may offset potential gains.

Energy-management approaches often neglect that frequency scaling is highly dependent on workload characteristics. Figure 8 compares the accumulated total CPU energy consumption for each of our three benchmarks with varying frequency on the TI AM572x platform (accounting for the same subsystems as for Figure 7). It is obvious that the energy consumption curves are convex, each having an optimal frequency point (f_{opt}) where the energy consumption is minimized. This is consistent with previous experiments revealing the existence of the Energy/Frequency Convexity Rule for compute-intensive applications running on an Exynos-based platform in a Samsung Galaxy S2 phone [2], [6].

The time it takes to execute a workload increases more than linearly when decreasing the CPU's frequency, while leakage power continues to be drawn during the time the CPU is active. This leakage contributes to the convexity on the lower end of

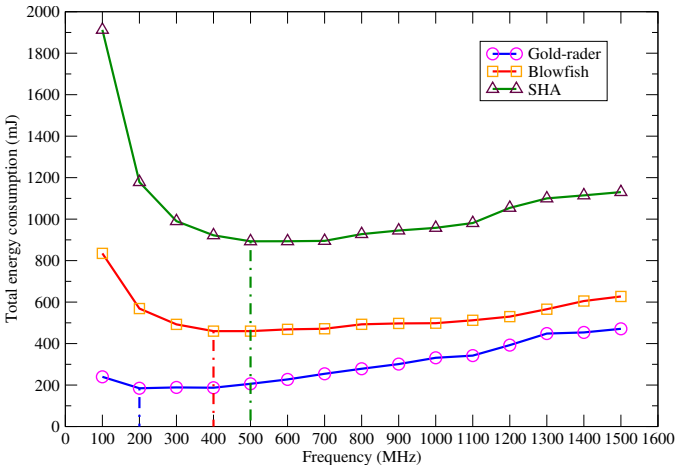

Fig. 8. Total energy vs. Frequency: energy consumption of our three benchmarks with varying clock frequency on the TI AM572x platform. The dotted lines represent the optimum frequency for each benchmark.

the frequency range, i.e., when the operating frequency f is lower than the optimal frequency f_{opt}. For the region where the operating frequency is greater than the optimal frequency ($f > f_{opt}$), the energy consumption is attributed to the increase in frequency, which contributes to dynamic power. In our experiments, we set the CPU target voltage to 1.21 V for the entire frequency range to observe the effect of frequency scaling in isolation. Although the target voltage is fixed, the actual voltage can be anywhere between 0.95 V and 1.25 V. The actual value is determined dynamically by the PMU and may depend on many factors, including production variations, silicon defects, temperature, and the actual CPU load.

These energy profiles allow us to make well informed decisions. For instance, it can be seen from Figure 6 and Figure 8 that the curves for the total energy consumption are relatively flat around the optimal operating points (f_{opt}). This would allow us to trade energy efficiency against performance. For instance, the performance of the Gold-rader benchmark can roughly be doubled by increasing the clock frequency by about 200 MHz from the optimal operating point. The energy consumption would stay almost unchanged and only increase by about 2 mJ. We can similarly improve the performance of the Blowfish and SHA benchmarks by 50% and 40% respectively by increasing the clock frequency by about 200 MHz from the optimal operating point, while incurring only a marginal increase in energy consumption.

Figure 8 shows that the optimal frequency points (f_{opt}) are different for all three benchmarks. So, for a new workload that enters into the system, having an energy-management strategy that tries to minimize the energy consumption by chasing the f_{opt} point depending on its current operation may not work. One possible approach would be to try to classify applications according to energy usage. The operating system would then use the profile of every newly incoming application and apply the best energy management strategy for that particular profile. For example, we could make offline profiling of applications and then cluster them into groups using a *k-means* clustering mechanism. When a new application is introduced, a run-time classification into one of the clustering groups is performed based on the slope of the energy-frequency curve. Of course, such an adaptive scheme works best with applications exhibiting predictable and/or slow-changing power requirements.

VI. Conclusion and Future Work

Optimizing embedded and communicating systems that host energy-critical applications remains a challenge. In this paper, we have presented a new experimental bench for energy profiling of non-performance-critical applications. Our setup provides direct access to the CPU's power rail, enabling precise energy readings. We detailed our experience of acquiring fine-grained energy measurements using two development platforms and we reconfirmed the Energy/Frequency Convexity Rule for CPU-bound benchmarks. Results also showed that frequency scaling depends on the workload characteristics, resulting in workload-specific optimal clock frequencies.

Our future work will be to mathematically model the Energy/Frequency Convexity Rule in light of the conceptual view of the platform energy consumption described in Section II. We would like to extend it to an algorithmic approach for platform-level energy management. This could be done by first classifying applications according to their energy usage and then building an energy-aware execution model.

Acknowledgements

We thank Texas Instruments for providing us the AM572x EVM board, the main platform for our experiments, and Karim Ben Kalaia for his help and advice.

References

[1] Dally, W. J., Balfour, J., Black-Shaffer, D., Chen, J., Harting, R. C., Parikh, V., Park, J., and Sheffield, D. "Efficient embedded computing". Computer 41, no. 7 (2008).

[2] De Vogeleer, K., Memmi, G., Jouvelot, P., and Coelho, F. "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices." International Conference on Parallel Processing and Applied Mathematics (2013).

[3] Hager, G., Treibig, J., Habich, J., and Wellein, G. "Exploring performance and power properties of modern multicore chips via simple machine models." Concurrency and Computation: Practice and Experience 28, no. 2 (2016).

[4] Le Sueur, E., and Heiser, G. "Dynamic voltage and frequency scaling: The laws of diminishing returns." Workshop on Power-Aware Computing and Systems (2010).

[5] Rotem, E., Ginosar, R., Weiser, C., and Mendelson, A. "Energy-aware race to halt: A down to EARtH approach for platform energy management." IEEE Computer Architecture Letters 13, no. 1 (2014).

[6] De Vogeleer, K., Memmi, G., and Jouvelot, P. "Parameter Sensitivity Analysis of the Energy/Frequency Convexity Rule for Application Processors." Sustainable Computing: Informatics and Systems 15 (2017).

[7] Dhiman, G., Pusukuri, K. K., and Rosing, T. "Analysis of dynamic voltage scaling for system-level energy management." Workshop on Power-Aware Computing and Systems (2008).

[8] Pallister, J., Hollis, S., and Bennett, J. "BEEBS: Open benchmarks for energy measurements on embedded platforms." arXiv preprint arXiv:1308.5174 (2013).

[9] Gold, B., Stockham, T. G., Oppenheim, A. V., and Rader, C. M. "Digital processing of signals." McGraw-Hill (1969).