Abstract—The fast pace at which technology has been evolving has led to a significant increase of the amount of energy that is consumed by nowadays High Performance Computing systems (HPC). Consequently, it has become highly important to understand how the energy consumption of any given application changes over time, envisaging the possibility to implement real-time power profiling and resources optimization. The work that was developed in the scope of this thesis describes the design and prototyping of an acquisition board (and related software API) composed by several Hall sensors and a microcontroller. Such board is capable of measuring the amount of power demanded by an HPC system, by monitoring the current that passes through the several rails of the main Power Supply Unit (PSU) of a personal computer. For such purpose, a broad set of conditioning modules were studied and implemented, in order to ensure accurate and precise measurements under an ample dynamic range of the measured signals. In particular, an Automatic Gain Controller (AGC) module was implemented in the acquisition board, embracing both the analog and digital domains of the measurement procedure. The results obtained from the experimental evaluation showed that the conceived device is highly suitable for real-time power profiling of HPC systems under complex workloads, by providing fine-grained measures of the power consumption over time, hardly attained by other alternative state-of-the-art devices or systems.

I. INTRODUCTION

The computing science community has attained fantastic results, regarding algorithm’s time performance, resorting to multi-core processors (using CPUs, GPUs and FPGAs). However, this kind of parallelism usually implies higher power demands, due to the minimum level of power required by each active core. In addition, different configurations and systems consume different amounts of power under stress of the same workload. And, even with the same system and configuration, an algorithm has different power demands along time, due to the various tasks running on the processor, which may require distinct resources.

Thus, it would be convenient to somehow understand how and where power is being consumed by a scientific computational application, associated with some configuration (i.e., running with one or more threads in one or more cores). For instance, with that kind of information in hand, the engineer could choose the best combination of the number of nodes and threads for which the algorithm will attain good performances of both time and energy consumption. It could, also, make the decision of exploiting parallelism of specific jobs with GPUs instead of using CPUs or vice-versa, depending on the gain of performance and energy cost. In addition, can combine the real-time power information with power-aware strategies, such as Dynamic Voltage-Frequency Scale (DVFS), decreasing CPU’s power consumption.

This problem can be solved resorting to energy models [1, 2, 3], which rely on specific power indicators (such as CPU utilization) to deliver power consumption information. Other approaches are based on direct power measurements, by the usage of voltimeters and ammeters. In [4, 5], the authors proposed the use of Performance Monitor Counters (PMCs) to find the power of the processor and other devices, such as memory and disk. In [6], Chang et. al all proposed energy models to estimate the components power consumption, alleging that PMCs are not suitable to build fine-grained energy models and that the power profiling thread may interfere with other applications using PMCs.

Other researchers ([7, 8]) defended the use of in-situ measurements, since it provides more accurate measurements and can measure total system’s power and individual devices. Data collection can be done with digital voltmeters and ammeters (Power Scope [3]) and inbuilt sense resistors (WatchDog.com PowerEgg [9], WattsUp? Pro [10]). These approaches have, usually, low time resolutions (2 or 4 Hz), which leads to a major loss of power information, if we bear in mind that a multi-core processor can issue billions instructions per second.

Some of the aforementioned applications have drawbacks, such as not being fast or accurately enough. Thus, failing to fulfill the needs of the computer engineer for a device that can successfully characterize, in real-time, power consumption variations.

Therefore, there is a set of requirements for which a device that seeks to solve this problem must comply to: it must be able to sense all significant rails that supply power to a desktop PC with low power losses; it must provide accurate and precise measurements; it must not introduce a significant time overhead and have a user-friendly interface; it must feature a great time resolution, so fast variations in power demand are distinguishable ($F_s > 1 kHz$); it must exchange data at high rates (> 64 KB/s); it must be powered by the host system and composed by the smallest number of components possible, in order to reduce costs and guarantee a low form factor.

In this paper, an hardware system called Powermeter is presented, which promises to comply with such rigid requirements. The Powermeter is a prototype developed within INESC-ID SiPS group, which is capable of measuring, directly, the 12V, 5V and 3.3V power rails from the motherboard’s power connector and individual components’ power, such as CPU, GPU and HDD. It can also measure the AC power...
requested by the system, using an Automatic Gain Controller (AGC), which dynamically amplifies the input signal, improving Analog-to-Digital Converter (ADC) dynamic range and allowing to distinguish small variations in the system’s power consumption. The design and requirements of the full system, as well as details about the theory and project of the AGC are revisited in later sections of this paper. The device works at $f = 3.3(3)$ kHz, can exchange data at more than 64 KB/s, it is accurate and introduces low time overhead. All these characteristics makes it suitable to characterize with a detail hardly attained by other systems, complex scientific workloads with rapid varying power demands over the time.

This work is organised as follows. Section II describes the Powermeter system hardware architecture, by revealing how can the device attain accurate measurements. In Section III, the software interface is presented, by introducing three major functions, which are used to easily monitor an application. In Section IV, the Powermeter device is evaluated and validated, by comparing its implementations with a system based on internal counters. In addition, the power profiling of two complex scientific applications (NPB EP and BT workloads) is revealed, as well as a characterization of the power consumption of a common desktop computing system. Finally, conclusions are drawn in section V.

II. POWERMETER ARCHITECTURE DESIGN - HARDWARE

The Powermeter sensor devices are inserted directly on the power connectors coming from the PSU, by plugging supply-side power cables into input connectors of the Powermeter and connect the output to wherever the initial connector should plug in (motherboard, CPU, peripherals and others), as the reader may observe in figure 1.

Regarding the sensors, precise low-offset linear Hall-Effect sensors were used, which convert a magnetic field generated by a current to a proportional voltage. The Hall IC has a copper conduction path which has an internal resistance of 1.2 mΩ providing low power losses. The device uses USB protocol to power the device and to suit communications between it and the host, since USB is a fast, versatile and reliable interface. Powermeter uses Microchip PIC18F4550 Microcontroller Unit (MCU), inserted in a common demonstration board (PICDEM FS USB), to establish communication between the host system’s USB and the current sensors on the device. The system communication is interrupt-driven, simplifying the microcontroller code and decreasing the time overhead, since there is no need by the host to poll the device to find if there is data to be collected. The microcontroller and the USB peripheral run both at 48 MHz. This microcontroller also includes timer modules (which were used to generate the necessary time stamps and sampling frequency) and a 10-bit ADC.

A. Signal Acquisition

Figure 2 exposes a general view of the proposed acquisition system. The system properly acquires the output of the DC and AC sensors. The DC acquisition relies on an amplifier stage to increase the range of the low voltage current sensor’s output signal. The AC acquisition introduces a AGC system, which dynamically amplifies the input signal, allowing to distinguish small variations of the signal. The system comprehends an active band-pass filter, which reduces noise and only passes the 50 Hz component of the spectrum, eliminating in the process the offset imposed by the current sensor; and a controller that dictates if the signal gets amplified or attenuated, using a Programmable Gain Amplifier (PGA). Consequently, by the usage of this novel approach the ADC’s dynamic range gets improved. The sensors used to sense AC and DC current of several power rails output a voltage which is proportional to the magnetic field generated by the sensed current. The sensors have different sensitivities, depending on the range of current they are able to sense. For instance, for a 20 A range we have a 100 mV/A sensitivity, whilst for 30 A range, the sensor gives a 66 mV/A output. However, these sensors introduce an offset approximately equal to $V_{cc}/2$. As a result and also because we want a good precision in ADC conversions, the signal coming from the sensor has to be properly conditioned.

B. Automatic Gain Control

This section presents an approach that uses a successive-approximation sampling 10-bit ADC, combined with an ADA
front end, forming a AGC system (figure 3). With a gain that changes automatically based on the analog input value, it uses oversampling at $F_s = 3.3(3) kHz$ to increase the dynamic range of the system to more than 80 dB.

The need to measure signals with a wide dynamic range is quite common in the electronics industry, but current technology often has difficulty meeting actual system requirements. Weigh-scale systems typically use load-cell bridge sensors with maximum full-scale outputs of 1 mV to 2 mV. While the actual sensor data typically takes up only a small portion of the input signal range, the system must often be designed to handle fault conditions. This is exactly the problem of the used current sensors, which outputs a very low-voltage amplitude signal. Thus, a wide dynamic range, high performance with small inputs, and quick response to fast-changing signals, are key requirements. These requirements call for a flexible signal-conditioning block, with low-noise inputs, relatively high gains, and the ability to dynamically change the gain in response to input level changes without affecting performance, while still maintaining a wide dynamic range.

![Fig. 3. AGC Structure](image)

Therefore, an automatic gain controller is proposed. This stage is supposed to, along the time, distinguish small variations in the input signal and amplify them, guaranteeing no loss of the input signal and an improvement of the dynamic range. Figure 3, shows a diagram with the major blocks that constitutes this system. Part of this scheme (the analog part) was done using physical elements like operational amplifiers and digital controlled potentiometers, whilst the digital part was performed by programming the PIC18F4550.

The full system comprehends a subtraction node, which entries are the band-pass and the DAC output signals. The last one is the result of the average of the last 32 samples from the acquired signal. By using this procedure, it is possible to optimally amplify, even more, the small variations in the current signal. The digital value resulted from the average computation is recorded in a variable, so it can be summed to the acquired signal to successfully regenerate the original signal.

The controller system could be accomplished in several ways: one of the alternatives would be to design a Proportional Integral (PI) controller. However, in this particular case, the controller parameters would change in time, since they are dependent on the amplitude of the input signal, which keeps varying in time. This would result in different time responses of the loop. The parameters are also dependent on the sampling frequency, thus for every change in the sampling frequency (during design project), new parameters may have to be computed. Other issue is that the computations needed are not adequate for the PIC18F4550’s 8-bit architecture and would consume much time - a multiplication and division of floats can reach 336 and 2712 clock cycles, respectively, while arithmetic shifts of integers require, approximately, 20 clock cycles.

In order to meet the requirements, another solution was engineered: the Bang-bang controller. The Bang-bang controller (or also denoted as on-off controller), is a simple and effective solution to this problem. In this project, an 'on-off' non-linearity with hysteresis and a dead-zone was implemented (observe figures 4 and 5).

The controller was implemented in the digital domain and changes the gain over the time by controlling a PGA. The controller comprehends three states: one where it increases gain, other where it decreases it and the the dead-zone, where it maintains the current gain unchanged. The controller also includes a maximum search algorithm and a low-pass filter: the search for the maximum is necessary not only to test if the signal amplitude is within the allowable range, but, also, to compute the power of the AC signal, which can be obtained by multiplying it with the 230 Vrms of the AC power; the low-pass is used to avoid false gains changes, due to instantaneous fluctuations in the measured signal. The output of the controller feeds the non-linearity, whose margins were computed taking into account the variations of the current sensor output.

![Fig. 4. On-off Controller](image)

![Fig. 5. Non-Linearity](image)
A. Communication System

The communication between the microcontroller and the host is established through the Universal Serial Bus protocol. The PIC18FX455/X550 device family contains a full-speed and a low-speed compatible USB Serial Interface Engine (SIE) that allows fast communication between any USB host and the PIC microcontroller. It was used the full-speed mode, which grants at the most 12 Mbps.

For this project, the USB transactions are granted using the HID class specification. The communication is interrupt-driven, meaning that the host does not keeps pooling the device waiting for data, which leads to less communication overhead. The host communicates through USB, by using the libusb library API [12], which is an open-source library. The application offers two main interfaces for device I/O: Synchronous and Asynchronous Interfaces.

The Synchronous interface allows the user to perform a USB transfer with a single function call. The main advantage of this model is simplicity: we can do everything with a single function call. However, the application will sleep inside a transfer until the transaction has completed, consequently the entire thread will be useless for that duration.

On the other hand, the asynchronous I/O is a more complex interface, but instead of providing functions that block until the I/O has complete, the interface presents non-blocking functions, which begin a transfer and then return immediately. Due to the demands of this project for developing a device that should not introduce a significant overhead on the normal user application, the used interface for exchanging large amounts of time sensitive data was the Asynchronous one. Nevertheless, for specific cases, such as to initiate synchronization between clocks or to start and stop the sampling process, it was used the Synchronous interface since these kind of requests do not have time restrictions.

In addition, two timer modules embedded in the microcontroller were used to set the sampling frequency of the system and to tag the time at which a sample is acquired: for real-time power profiling it is fundamental to know when a sample was acquired. Therefore, before the sampling process it is necessary to synchronize the microcontroller and host’s clocks. This is achieved by an algorithm that resembles the Network Time Protocol (NTP). The time stamp is synchronized with the 32 LSBs of the host’s clock, providing a resolution in the order of micro-seconds.

There are different kinds of data being transferred between host and device: Asynchronous data and Synchronous data. Hence, it is necessary to distinguish the data sent/received by both systems (the microcontroller and the host). Therefore, the messages are divided into an Header and a Payload. The Header gives the possibility to differentiate the type of data being transferred, while the Payload is the data itself that the device/host wants to transmit. When in asynchronous mode, the buffer size is at the most 256 Bytes, while when in synchronous mode it is just 5 Bytes. Tables I and II shows how data is divided in the synchronous and asynchronous modes.

C. MatLab/Simulink Stability Tests

In order to prove the stability of the proposed system, it was necessary to resort to non-linear feedback analysis tools, since the system relies on a non-linearity. In particular, it was used Popov’s theorems.

In sum, the Popov criterion states that if the non-linearity is delimited by two lines of slope 0 and K, if it belongs to the first and third quadrants and if the open loop transfer function is Hurwitz (i.e., it has all the eigenvalues in the left-half of the complex plan), then the system is globally asymptotically stable iff \( \exists \ q \geq 0 \) such that:

\[
\Re \{(1 + j\omega q)G(j\omega)\} + \frac{1}{K} > 0 \quad \forall \ \omega \in \Re
\]  

(1)

where \(G(j\omega)\) is the open-loop transfer function.

This formulation permits an interesting graphical interpretation: if the Nyquist plot of the system \(\hat{H}(s) = (1 + sq)G(s)\) lies in in the right side of the vertically line that passes through the point \(-\frac{1}{K}, 0\), then the closed loop system is stable [11].

According to figure 5, the non-linearity is in accordance with the theorem and is enclosed by two lines of slope 0 and 1.

Therefore, resorting to MatLab the open-loop transfer function of the system was found to be:

\[
G(s) = \frac{1.286 \times 10^{11}}{(s + 785000)(s + 10)}
\]

(2)

This has lead to the plot of figure 6a, which represents the Nyquist Diagram for equation \(H(s) = (1 + s)G(s)\).

Taking a closer look in the neighbourhood of the vertical line at \( \Re = -1 \) (see figure 6b), it can be observed that the diagram lies at the right of that line, so the Popov criterion states that this system is absolutely stable.

III. POWERMETER ARCHITECTURE DESIGN - FIRMWARE/SOFTWARE

This section seeks to introduce the general procedures of the Powermeter API to successfully acquire data and provide power measurements.
respectively.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Header</td>
<td>Channel 0</td>
<td>Channel 1</td>
<td>Channel 2</td>
<td>Channel 3</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE I**

**SYNCHRONOUS DATA STRUCTURE**

The synchronous structure is just used to start routines, such as clocks synchronization or the sampling process. The byte named **Header** can be **NTP_INIT**, **START** or **STOP** commands (used to start clock synchronization, and start and stop the sampling process, respectively). The payload comprehends the bytes referring to **Channels 0, 1, 2 and 3**. Thus, the user can connect any of the power rails coming from the PSU to be monitored by one of these channels. In addition, it can sample more than one channel at a time without decreasing the sampling frequency of the system.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Header (1Byte)</td>
<td>Payload</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40h</td>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80h</td>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C0h</td>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE II**

**ASYNCHRONOUS DATA STRUCTURE**

The asynchronous data structure is effectively used to exchange large amounts of data. In this structure, the **Header** byte can be {**NTP_HEADER** or **DATA_HEADER**}, referring to the clocks synchronization or the sampling process, respectively. The payload is the data exchanged between both processors, which can be data about clocks synchronization or samples acquired during the sampling process.

**B. Buffering Strategy**

A dual buffer strategy (figure 7) was used to save and send sampled data: every sample is saved in a buffer (**A_buffer**) of size 256 bytes. When this buffer is filled with data, we must send it to the host and still allow the sampling process to continue seamlessly. Consequently, another buffer (**B_buffer**) was necessary to save those samples, whilst **A_buffer** is being utilized only to transfer data. When **B_buffer** is filled, it is used to send data while **A_buffer** will be responsible to store the sampled data. Thus, the buffer to store or to send data alternates between **A_buffer** and **B_buffer** along the time.

**C. Oversampling and Maximum Search Algorithm**

As it was referred before, it was performed oversampling of the acquired samples at f = 3.3(3) kHz, in order to improve the DR. Therefore, it was used a rolling average buffer of 8 samples (figure 8): this buffer has a pointer, which returns to the start of the vector as soon as the buffer is filled with data. While the buffer is not completely filled with samples, the program sends the samples directly without averaging them. This allows a constant sending of useful data even at the start of the program. When a new sample arrives, it is added to the accumulator after the older sample gets subtracted from it. Finally, the output of the accumulation is averaged and the oversample process is over.

In the case of the sensing of the AC sensor, after the oversampling technique, it is also necessary a search for the maximum of the acquired data. This is necessary, since in order to compute the power demanded by the power supply, we have to know the amplitude of the current signal so we can multiply it by the 230 V<sub>RMS</sub> and by the power factor (0.99).

The pseudocode for the **maximum** search is depicted in algorithm 9, which just illustrates the main part of the algorithm. This algorithm returns the peak of the sine wave acquired during AC current sensing. The algorithm is pretty simple, fast and most importantly, effective. It follows the acquired signal and tests if the current sample is higher than the one before and if it is, declares it as temporary maximum. Then, tests if the current sample is lower than the temporary maximum, taking into account the **delta** parameter, which was set to 10 quantization levels, after some experiments. If this comes to be true, then the current sample is an absolute maximum and a search for the minimum value is addressed from this point on. These two routines are inextricably linked and the maximum can be found because the search for the minimum updates the temporary maximum value.

**D. Energy Calculation**

The API returns the energy that was spent between two points of the code of the user application. For this, the user has to call **three** major functions, which use the synchronous structure introduced in section III-A:

- powermeter_api_init(char chanel1, char chanel2, char chanel3, char chanel4);
1: if lookformax then 
2:     if Current Sample < temporary max – delta then 
3:         New absolute maximum found 
4:     lookformax = 0 
5: end if 
6: else 
7:     if Current Sample > temporary max + delta then 
8:         New absolute minimum found 
9:     lookformax = 1 
10: end if 
11: end if

Fig. 9. Maximum Search

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>powermeter_api_init</td>
<td>channels to sample</td>
<td>General Initializations</td>
</tr>
<tr>
<td>powermeter_api_start</td>
<td>Void</td>
<td>Starts Data Sampling</td>
</tr>
<tr>
<td>powermeter_api_stop</td>
<td>Void</td>
<td>Stops Sampling</td>
</tr>
</tbody>
</table>

TABLE III

MAIN API FUNCTIONS

- powermeter_api_start(void);
- powermeter_api_stop(void);

To use the first function, the user must choose which channel(s) to sample (CPU, GPU, HDD or AC, for instance). If there are channels that the user does not want to sample, then it must use NOP as argument. For example, if only CPU channel must be sensed, the call must be: powermeter_api_init(CPU, NOP, NOP, NOP). This function executes every necessary initializations, including libusb library initialization, vectors and files initialization, and synchronization between clocks.

The powermeter_api_start(void) function gives order to start the sampling process, whilst the powermeter_api_stop(void) function, besides stopping the sampling process, also closes opened files, frees allocated memory and calls the routine for the energy computation. Table III summarizes the main functions and their respective features.

The energy is calculated from the power curve by integrating it along time:

$$E_{nergy} = \int_{t_1}^{t_2} P(t) dt$$  \hspace{1cm} (3)

Since the operation occurs in the discrete time domain, the energy spent between start and stop commands, must be computed by using numerical integration methods. There are various methods available to do one-dimensional integration, that are based on interpolation functions: Rectangle Rule - Order 0 Polynomial, Trapezoidal Rule - Order 1 Polynomial and Simpson’s Rule - Order 2 Polynomial.

$$E_{nergy} = \sum_{n=1}^{N} T_{Sampling} \times \frac{P_{wr_{n-1}} + P_{wr_n}}{2}$$  \hspace{1cm} (4)

The energy computation is partially performed in the microcontroller, which results in less time overhead of the software application. With every chunk of data, that it is send to the host, it gets associated an extra slot with the energy of that chunk of samples. Afterwards, the host just has to perform the remaining energy computation (the process stops when the sample corresponding to the time at which the sampling process has ceased is reached).

For this work, it was used the trapezoidal integration rule (see equation 4) since it is a method which results in a small integration error, compared to the rectangle rule and also because it is simple. Simpson’s rule would be a better choice, to reduce the error. However, it would require more sums, multiplications and divisions by numbers which are not a power of 2 and that constitutes a problem for PIC18F4550’s architecture.

IV. EXPERIMENTAL RESULTS

The main goal of this section is to understand how the measuring methods (internal and external measurements) influence algorithm execution (time overhead) and also compare the readings given by the Powermeter and the Running-Average Power Limit (RAPL), which is based on PMCs. The measures obtained with RAPL were performed with a time resolution of 2 ms (maximum update rate [13]).

A. SPLASH-2 Benchamark Tests

SPLASH-2 benchmark suite, included in PARSEC suite [14], was used for this evaluation. This benchmark offers a variety of workloads: for benchmarking purposes, FFT, LU Matrix Decomposition and RADIX sorter algorithm workloads were utilized. Those workloads and their respective Makefile were properly modified to include the calls to Powermeter API or to RAPL and each algorithm was executed in a loop for 1000 times in a row. The results were averaged to get statistical significance. As said before, the main focus of these tests are: evaluate the time execution with/without the power measure systems and evaluate the energy consumption reported by Powermeter and RAPL.

1) FFT: For this workload, the tests were made using a data set of $2^{20}$ complex numbers and the algorithm was distributed in 4 threads (1 by Physical Core) - The total time of execution was 76826729 µs;
2) LU: A 1024x1024 matrix of doubles was used as input and the algorithm was distributed in 4 threads (1 by Physical Core)The total time of execution was 98648873 µs;
3) RADIX: For RADIX workload, a data set of 4292608 32-bit random integers was sorted and the algorithm was distributed in 4 threads (1 by Physical Core)The total time of execution was 96438625 µs.

Observing tables IV and V, it can be concluded that the overhead due to Powermeter or RAPL metering system is not significant, since in the worst case it takes 0.25 % and 1.22 % more time respectively, than the original source code. However, possible due to the higher amount of data exchanged and computed (Powermeter works at f=3.3(3) kHz, whilst RAPL works at f = 500 Hz), Powermeter introduces more time overhead than RAPL. Regarding the energy measurement of
both devices, it is notorious a difference between Powermeter and RAPL, where the former indicates higher energy values than RAPL. This is not strange, since it runs for a longer period of time, has an higher resolution than RAPL and it is not based on PMCs as RAPL, but mainly this is due to the fact that RAPL does not report all the energy consumed by the uncore of the CPU.

B. NAS Parallel Benchmark Tests

Conversely, to evaluate the power profiling and the trade-off between energy and time consumption, the NASA Parallel Benchmarks (NPB’s) with OpenMP [15] were used. This benchmark includes workloads for scientific applications like FT (3-D fast Fourier Transform), BT (Block Tri-diagonal solver of Navier-Stokes equation) and CG (Conjugate Gradient method to compute an approximation to the smallest eigenvalue of a large, sparse, unstructured matrix). Version NPB 3.3.1 was compiled and it offers different benchmark classes: S, W, A, B, C, D, E and F, sorted from the smaller one to the largest problem size. In the conducted tests, only class B was used, which offers a standard problem size and a reasonable high number of iterations (depending on the used workload), thus allowing the profile of an application for a satisfactory period of time.

Figures 10a and 10b represent power profiling graphs of NPB FT benchmark, compiled with -O3 and with CLASS=B and ran with 4 threads, one by processor. In this case, class B translates into a problem size of a 512x256x256 grid and 20 iterations. Both RAPL and Powermeter were used for power profiling this workload. In figure ?? it is shown the different power patterns of CPU, RAM and HDD over time for the EP and BT benchmarks.

In figure 10a, it is clear the fluctuations of the power drained by the CPU over the time. It was found that the number of valleys corresponds exactly to the process iterations executed by the benchmark (N=20). The application initiates with a warm up phase and an initialization phase, followed by N iterations (for CLASS B N = 20). Both RAPL and Powermeter follow the power variations along the time and it is possible to distinguish the computational stage (when power is at its peak) and communication stage (when power comes down). Even so, RAPL counters indicates less power consumption than Powermeter and a huge amount of spikes (some of them reaching values above the maximum power consumption specified by Intel [16]). An in-depth view to one of the process iterations reveals more details, as it can be observed in figure 10b. In this figure, it is clear the difference, in terms of resolution, of RAPL and Powermeter: while the former outputs an average of the power measured, the later indicates the real power at some instant in time and, thereby, much more fluctuations in power behavior are visible; we can, also, differentiate three small valleys in each iteration, which must be related to inner subroutines of the FT benchmark. Thus, while with the Powermeter it is possible to distinguish several different patterns of the power demanded by an application, with RAPL such level of detail is lost, making it inapt for the power characterization in real-time of high-complexity workloads.

Figures 11 and 12 shows the power profiling results of the EP and BT benchmarks, illustrating the power consumption of CPU, HDD and RAM. For each benchmark, the conducted tests results are displayed for 1, 2, 4 and 8 threads, running in different processors. The corresponding source code was compiled with gfortran, -O3 flag and linked to Powermeter API. Each subfigure focuses on the power usage for the first few seconds of the test, in order to clearly show the resulting power behavior patterns.

The embarrassingly parallel benchmark (EP) is essentially computation intensive and communication free. The benchmark consumes a consistent amount of power during its entire execution, since it is perfectly balanced, and each thread executes a CPU-intensive job. Contrastingly, the BT benchmark (represented in figure 12) is more memory-intensive. As a consequence, the power pattern is not consistent over the test run. It was found that the power consumed by the CPU and the memory is interrelated in a way that when memory power goes up, CPU power goes down and vice-versa. In addition,
the machine is at its idle state, compared to the total power consumed by the machine. To understand how significant is that power consumption when active, power is not considered as used for computing. Active power corresponds to the extra power dissipated when the system is no longer in idle mode, but in active mode.

1) CPU: The CPU is powered, directly, by the four 12V cables connecting to EPS12V. This was confirmed by experiments and by the ATX12V power supply design guide. Thus, the sensors are connected directly to this cables. For this test, the LU matrix decomposition provided by SPLASH2 benchmark [14] was used. The test consisted in the decomposition of a 4096x4096 matrix of doubles (128 MB of data) and it was ran in a loop for 25 times, so most of the necessary data will be present in CPU cache, decreasing memory accesses. In addition, it was used P = 1, P = 2, P = 4, P = 6 and P = 8 processors during the test.

2) HDD I/O: The HDD is powered directly by two independent cables (+12V and +5V rails), hence by directly measuring this rails, the disk power consumption can be profiled. In this scenario, a series of tests were performed, consisting in several write operations in the hard disk. Those tests require that the data being written has to surpass largely total RAM size - at least use a dataset two times larger than the total available memory RAM (at Liliana we have approximately 15778 MB).

3) Memory RAM Accesses: To profile power spent in memory accesses STREAM benchmark [17] was used, which measures an effective memory bandwidth on one or more cores. Therefore, a total of three tests were conducted: 1, 2 and 4 threads. With this, it is expected to grasp which power rails are correlated with intensive memory accesses. About the dataset, the benchmark requires that the size of the array to be used, must be four times larger than CPU L3 cache (8 MB).

4) TCP/UDP Data Packets: For this case, the aim is to isolate the power consumed by the Network Interface Chip (NIC), so it was used the iPerf benchmark [18], which can generate TCP and UDP data packets and measure throughput between server and client.

Figure 13 shows the power consumption measured during five different workload tests on the machine. The results are organised in a stacked bar chart, where for each workload (idle, LU 1, LU 2,...) it is presented the power drawn by a component (CPU or HDD) or a rail (12V, 5V and 3.3 V).

By observing figure 13 for the idle bar, the reader can observe how the power is distributed in a desktop system, consuming about 25 W of DC power. Although the CPU only consumes about 8 W, it is still the major portion of power consumption, and together with disk’s power, represent more than 50 % of the system power. The other rails (which are related to RAM, fans, peripherals and others) consumes together the rest of the system’s power (about 12 W).

After a thorough analysis of each bar of figure 13, we can come up with several conclusions. For the case of the workload is running.

### Powermeter was tested in Liliana (machine hosted by SiPS Group at Inesc-ID Lisbon). The machine features the following characteristics:

<table>
<thead>
<tr>
<th>Module</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB</td>
<td>ASUS P8Z77-V LX</td>
</tr>
<tr>
<td>CPU</td>
<td>3.5 GHz Intel i7 3770K (TDP - 77W)</td>
</tr>
<tr>
<td>RAM</td>
<td>G.Skill Sniper DDR3 2x8 GB - 1.866 GHz</td>
</tr>
<tr>
<td>HDD</td>
<td>Seagate Barracuda - 2 TB</td>
</tr>
<tr>
<td>PSU</td>
<td>CORSAIR TX750</td>
</tr>
</tbody>
</table>

**TABLE VI**

**MACHINE’S CHARACTERISTICS**

In this section the goal is to successfully characterize power consumption of the aforementioned machine. A few tests were realized, not only to discover what connectors/rails from the PSU (Power Supply Unit) are directly correlated with the power drawn from the CPU or RAM, for instance, but also to understand how significant is that power consumption when compared to the total power consumed by the machine.

The first test consists, simply, in running the API while the machine is at its idle state (i.e., no intensive workload

![Fig. 11. EP CLASS=B Power Profiling](image)

![Fig. 12. BT CLASS=B Power Profiling](image)
for the LU decomposition of a matrix, it is visible that each component/rail suffers an increase of power consumption. But it is the CPU that presents the higher increase in power usage: while in idle it consumes more or less 8 W, but in stress it achieves, on average, 30 W, 45 W, 61 W, 63 W and 65 W, for $P = 1$, $P = 2$, $P = 4$, $P = 6$ and $P = 8$ processors, respectively. The largest momentaneous power measured was about 68 W. This complies with Intel specification [16], which states a maximum of 77 W of power draw by the CPU package, with no overclocking. Notice that there is not a significant change in power for $P = 4$, $P = 6$ and $P = 8$. This occurs because for $P = 6$ and $P = 8$ processors, Hyperthreading technology emulates at the most eight cores, by having each of the four physical cores running simultaneously two threads.

According to the results, the tendency is that for each doubling in the number of processors used, an extra 15 W of power is required, excluding the cases where the hyper-threading technology is being used. The growth of power in every rail was also measured during this test. Since the EATX12V connector supplies power to the CPU core, the power of the uncore (L3, mem. controller) must come from the motherboard, explaining the growth in power of the 12V rail. Even so, CPU’s fan and system’s fan are also powered by the 12V and 5V. Hence the surge in power on those rails as well.

For the disk intensive benchmark, the reader can observe that there are insignificant changes in all the rails. The only interesting changes, occur for the CPU and, as it should, for the HDD. The increase in CPU power is expected, since the workload runs several writes and rewrites in a loop. During the test, the HDD consumed more power in the writing with putc() command, achieving about 10.5 W of power, whilst when writing in chunks the power was lower. This was expected, since when writing a char at a time, more accesses to the disk are required than when writing a bunch of characters in once. The rewriting consumes more power than when just writing, because it also calls the lseek() system function. Although there was in increase in the power usage, the maximum specified power drained by this component was not achieved (12.75 W): the reasons may be because of the use of a not sufficient large dataset, but most probably is due to the existence of a buffer, which allows the disk to process reads/writes in batches, minimizing disk accesses.

Focusing, now, on the STREAM benchmark, it is visible a general rise of power draw in every rail and component. Nevertheless, if the reader takes a closer look to the 3.3V, 5V and 12V rails, it can perceive that it is only during this workload test that these rails evidence the higher power consumption values. For other tests, the power drained by the 5V rail has changed a little during LU workloads, but the consumption is not very high when compared with the idle power. This is an unambiguous proof that the 5V rail is tightly correlated with the power drained by RAM modules. Despite this benchmark is not CPU intensive (as the reader can observe in CPU bar), the 12V and 3.3V experiences a significant augment in power along this set of tests. Though, since those rails present a very high power consumption during LU workload as well, this is a confirmation that the 12V must power the CPU uncore and fans. Nevertheless, when in intensive work, RAM modules must request extra power from the 12V and 3.3V rails, explaining the peak values observed under the STREAM benchmark test.

Regarding the last workload, intended to stress the NIC, it is clear that there was insignificant or even no difference between the power drained when running the test and the idle state. This happened for all rails and components measured, thus it is a clear proof that the NIC uses all the power it needs whether in active or in idle state.

V. Conclusions

Initially, it was introduced the scope of this work, by revealing what was the problem and the state of the art applications and devices [1, 9, 10, 3] that seeks to solve it. However, it was concluded that those devices are not adequate for the characterization of complex applications, which have an high level of computational requirements, due to their inaccuracy or low time resolution. Therefore, a novel device was introduced, which promises to achieve the aforementioned goal: the Powermeter. The conceived device comprehends several electronic components, including precision Hall-effect current sensors and an AGC structure for handling the AC sensor output signal, by dynamically scaling the signal’s amplitude, improving ADC’s DR. The system samples at a rate of $f = 3.33kHz$, it does not add any significant time overhead and transmits the acquired data at high speed (more than 64 KB/s), while providing accurate and precise power measurements.
To make this possible, it was fundamental to analyse all the initial requirements and project electronic structures that were fitted for the conditioning of the various types of signals coming from an off-the-shelf desktop PSU. Furthermore, an AGC block comprising a band-pass filter and a PGA was proposed. The system was analysed it was evaluated in terms of stability. The results proved that the system is absolutely stable.

Then, the design of the software part of the conceived architecture was presented in section III. In this section, it was revealed the essential procedures for computing the energy with low time overhead, by scheduling part of that computation to the MCU; and it was presented the three main functions of the software API that are used to initiate and stop the power readings: `powermeter_api_init()`, `powermeter_api_start()` and `powermeter_api_stop()`.

The results regarding the ADC and the sensors calibration were presented in section IV. For validation of the system usefulness and reliability, several tests were performed by comparing the readings of computational intensive workloads with both powermeter and internal counters (RAPL). It was also profiled a few parallel benchmarks with different kinds of computational requirements. The results served as a proof of concept of the proposed device and several conclusions were realized: it was demonstrated that RAPL is less detailed and reliable for real time power characterization than Powermeter by not providing enough time resolution and all the uncore power consumption of the CPU; and it was concluded that various applications request distinct resources, evidencing different power patterns in CPU and RAM. Finally, it was identified which rails supply specific components (5V - RAM, 12V, 5V, 3.3 V - fans, NIC and others) and it was realized how the power is distributed in a desktop computing system.

REFERENCES


