Processor Energy and Temperature in Computer Architecture Courses: a hands-on approach

Sergio Gutiérrez-Verde Octavio Benedí-Sánchez
Darío Suárez-Gracia Jose María Marín-Herrero† Víctor Viñals-Yúfera
gaZ. Dpto. de Informática e Ingeniería de Sistemas
† Gitse. Dpto. de Ingeniería Mecánica
I3A–Universidad de Zaragoza
C\ María de Luna 1. E-50018 Zaragoza, Spain
http://webdiis.unizar.es/gaz/

Abstract

Performance has driven the microprocessor industry for more than thirty years. Its effort has enabled to multiply by several orders of magnitude the computational power; e.g., the Intel 8080 was able to execute 0.64 MIPS and the newest Core i7 can execute 6400 MIPS. The cost of this fabulous improvement has been a large rise in energy consumption. Nowadays, we have reached a point where one of the most limiting factor for improving performance is energy dissipation.

In order to keep the performance improvement during the next years, it is necessary to study energy and temperature in deep. Nevertheless, most current computer architecture curricula include neither energy nor temperature.

The lack of adequate experimental platforms contributes to the difficulty in teaching these topics. In this paper we propose a possible solution: to instrument a commodity PC for measuring the processor power and temperature during the execution of real programs. The platform is devised for teaching, but it can be used to support research experiments as well. For example, we describe an interesting undergraduate laboratory that analyzes the interaction between compiler optimizations and energy. With this laboratory, students can learn that performance optimizations usually reduce energy but may increase power.

1 Introduction

Recently, designing energy-efficient computers or reducing energy consumption is going beyond marketing strategies or personal experiences to turn into a collective goal for governments, societies, or companies. For instance, Green Computing advocates for an environmentally sustainable computing and communication, with minimal or no impact on the environment. Together with the concepts of total cost of ownership, including the cost of disposal and recycling, the economics of energy efficiency is a key point of Green Computing. So we think that computer engineers should be aware of these issues.

Energy-efficient computers are not only important from a Green Computing perspective, but also from a pure performance point of view. On one hand, in the embedded domain, lowering the energy consumed by the processor increments the device uptime. On the other hand, in the commodity segment, the cooling system affects performance when it is not able to dissipate all the generated heat and forces a processor frequency/voltage reduction.

While the study of many design constrains, such as performance or programmability, may be done by means of white boards or simulators, evaluation of energy and temperature appeals for hands-on laboratories—where students deal with real—for many reasons such as: 1) This approach reinforces their physics background and establish a clear connection between computer architecture and its implementation; 2) They will quickly learn the importance of energy dissipation and temperature by watching for example how fast a processor shutdowns when its fan stops; 3) Energy and temperature simulations require sophisticated environments for being accurate, and since energy depends on both the instructions and their data, the simulation time can be very high and unfordable in two/three hour lab sessions.

The main barrier this hands-on approach faces is the lack of well established platforms for carry on the measurements. Many authors have performed processor power measurements either research oriented such Isci *et al.* or academic oriented like Asín *et al.* [3]. Others such us Mesa-Martínez *et al.* have measured temperature in commodity PCs [17]. But up to our knowledge there is not an ade-

quate platform able to simultaneously measure both magnitudes. The present work extends the Asín *et al.* platform adding temperature monitoring support and automatic synchronization of the sampling process. The resulting platform improves measurement accuracy and data logging capabilities, and at the same time its academic capabilities such ease of use or cost are reinforced.

Platform features are presented by means of a laboratory intended use case for last year undergraduate or master courses. Our final goal is to use this platform with students from both Computer Engineering (Computer Architecture courses) and Mechanical Engineering (Heat Transfer courses) degrees in our institution to make them working together in a common problem. As a session suitable for both kind of students we present a lab dealing with the interaction between compiler optimizations and energy.

Summarizing, the contributions of this work are the following: we improve an existing platform for measuring energy and temperature in commercial processors extending its logging capabilities and improving the sampling accuracy. We present the potential of the platform with a interesting laboratory in which the relation of power and temperature and the impact of compiler optimization in energy and power are analyzed.

This paper is organized as follows. Section 2 comments on the related work. Section 3 describes the measurement platform in detail. Section 4 explains some test for validating the platform. Section 5 describes the example laboratory. Section 6 concludes and present some possible future work lines.

2 Related Work

Energy and temperature have aroused the interest in both industry and academia. In the industrial side, SPEC has introduced SPECpower_ssj2008 focusing on server computer consumption [6], and EEMBC has defined EnergyBench establishing a framework for adding energy to the metrics of the EEMBC's performance benchmarks [5].

Many studies have been conducted in the academic side. Regarding energy, Isci and Martonosi describes a methodology for obtaining per-unit power estimations combining real power measurements with performance counters [14]. Other authors have proposed infrastructures based on an Intel Pentium 4 for characterizing program phases, evaluating compiler optimizations, or studying energy [9, 21, 3].

Temperature measurements have been performed with more sophisticated setups; e.g., Mesa-Martínez *et al.* have presented some power and temperature estimations using an expensive IR thermal imaging equipment [16].

While most previous work focuses on energy and temperature from a research perspective, our work also takes into consideration academia requirements such as simplicity or affordable cost.

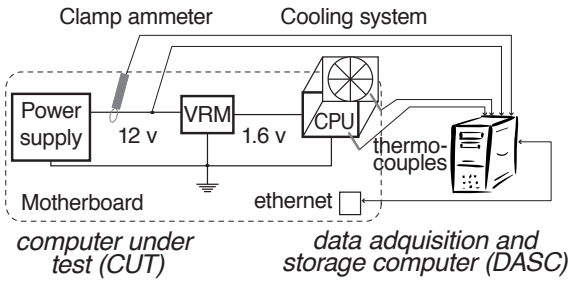
3 Platform description

The measurement platform is based in our previous work and consists of two commodity PCs [3]. One, named computer under test (CUT), is monitored, and another, named data acquisition and storage computer (DASC), acquires and saves all the power and temperature samples gathered from the CUT. Both computers are shown in Figure 1a, the CUT in the left and the DASC in the right.

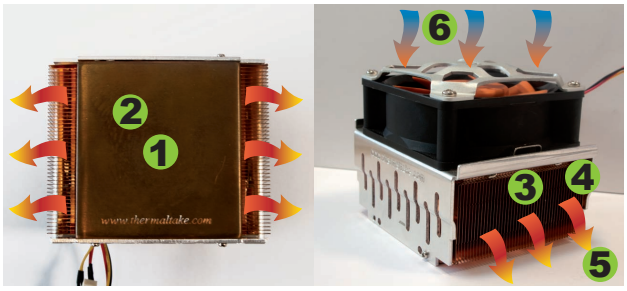
The CUT runs a GNU/Linux system with a 2.6.25 kernel in which all non-required modules and services (X-Windows, printing, USB, ...) have been removed to minimize the energy consumed by the operating system tasks. The processor and the motherboard are a 2.8 Ghz Intel Pentium 4 *Northwood* and an ASUS P4 P8000, respectively. This motherboard employs a dedicated power line between the power supply and the processor voltage regulator manager; thus, it removes the need of hacking the motherboard and simplifies the monitoring of the processor consumption because the product of the voltage of the VRM power line times its current is the power drawn by the processor—assuming negligible the VRM consumption [2]. The above described power line is present on most current PCs, so this technique can be used with other hardware configurations.

The current is measured with a Tektronix TPC-312 clamp ammeter [23]. The output of the clamp ammeter along with the voltage are logged with an Adlink PCI-9112 [10] data acquisition card sampling at 2 Kilosamples/second per channel, 1000× more than the previous version of the platform. At this sampling rate, we are able to observe the main program execution phases, and power traces remain in reasonable sizes, lower than 1 GiByte. All samples are stored in the DASC in order to allow off-line analysis . The DASC system also runs GNU/Linux and the previous LabView software has been replaced by C based code and some perl scripts because they allowed much higher sampling rates and we observed that the real-time visualization of LabView was seldom used. In fact, real-time visualization is useful for debugging the platform, but for that purpose an oscilloscope is preferable. The use of the new programs is straightforward with a small learning time as it was with the LabView based software.

Current processors require large heat sinks with powerful fans for cooling. Cold air flows towards the processor pushed by the fan and gets warmer. The hot air is expelled through the sides of the head sink as shown in Figure 1b. Since the air (a fluid) flows through a solid (each of the narrow channels in between the parallel fins), the whole processor cooling package could be modeled according to a forced-convection thermal model. If some conditions are



(a) Component diagram.



(b) Thermocouple localization in the processor–cooling package. . The cold air, pushed by the fan, flows through the narrow channels in between the fins : it enters top-down and exits horizontally, either by the left and the right sides

Figure 1: Overview of the platform with its main components

meet, and forced convection holds, heat transfer, q, becomes proportional to dissipation area, A, and gradient temperature, ΔT:

$$q = h \times A \times \Delta T$$

Being the constant h an (experimental) number depending mainly on thermal conductivity, speed of the flow, and channel geometry [11]. Acquiring temperature at multiple points will help us to determine the model goodness. Measurements are carried with K-type thermocouples optimized for the temperature range of 0-100 °C that are located at 6 positions: 1) drilled in middle of the heat sink contacting with the processor, 2) drilled in the border of the heat sink—Intel provides some guidelines for the placement at these locations [13], 3) in the lateral edge of a fink placed in the middle of the heat sink, 4) in the lateral edge of a fink placed in a corner of the heat sink, 5) in the free path of the output hot air flow without touching the heat sink, and 6) in the free path of the input cold air flow.

The six measurement points ease the verification of the forced convection model because from this model we know that the temperature of the hot air flow should be much bigger than that of the cold air flow. Also, the temperature should rise as we approach close to the processor; therefore, in the real measures we have to observe that $Temp(5) >> Temp(6)$ and $Temp(T1) > Temp(T2)$.

The acquisition of temperature samples is done with a Picotech TC-08 converter that is connected to a USB port of the DASC [22]. The conversion frequency depends on the number of attached thermocouples. In our case, 6 thermocouples, the data acquisition rate is 0.73 samples/second, so that any individual thermocouple gets sampled every 4,4 s. This rate is much smaller than that of power, but it is enough because the change rate of temperature is much lower than that of power as we will see in Section 4.

Since the platform uses two computers, it is required to synchronize the beginning and the end of the sampling process. The synchronization is acomplished by sending two low-latency Ethernet packets, one just at the beginning of the execution of the program under test and the other just after its end. This synchronization schema is done by a wrapper on the executables that avoids any complexity to the students, even for those without a good shell knowledge. The platform is able to monitor any program independently of its execution time as long as the hard disk drive has space left.

Summarizing, the platform is able to measure the temperature and the energy drawn by the execution of any program in an Intel Pentium 4 processor with high precision and without interfering the computer under test. All the platform software is freely available upon request.

4. Platform Validation

Most changes in the hardware of the platform with regards to the previous version were motivated for increasing the sampling accuracy and for logging power and temperature simultaneously. The objective was to detect power phases during program execution, and to see how changes in energy consumption affected temperature.

As a prove of the accuracy of the platform, Figure 2 shows the temporal evolution of power and temperature for the complete run of `473.astar` (SPEC CFP2006) compiled at the maximum level of optimizations with Intel C compiler[1].

The left Figure, 2a, shows the instant power and temperature at the center of the heat sink (thermocouple 1 in Figure 1b). Note that with this easy experiment students can see how changes in the phases of programs also affects to its energy consumption, and how temperature reacts slowly to the changes in power—justifying the choice of a much lower sample rate for temperature than for power. Besides,

[1]For more methodology details please read Section 5.

this plot also shows how the processor–heatsink–fan system tends towards their thermodynamic equilibrium when power is almost constant after roughly 130 s (this can be noticed in both the 150-300 and 600-800 time windows). Our software package includes a PID controller able to stabilize the processor consumption, or alternatively the temperature, at a given value for performing these kind of experiments in a controlled way.

In order to employ the forced convection model of the processor—cooling package we have to take several steps. The first one is to verify relations among the measured temperatures. As shown in the right Figure, 2b, the output air temperature (T5) is warmer that the input one (T6), and the difference in temperature increases as the processor activity rises. Once the initiation phase is completed, the temperature difference between the processor– headsink package and the input air (T2 - T6) is large (a maximum of almost 30 °C) while the difference between the processor-headsink package and the output air (T2 - T5) is small (less than 5 °C). These differences between both values indicate that the air is absorbing heat from the headsink and spreads it out of the processor–headsink package. Also the temperature in the middle of the heat sink (T1) is bigger than that of the border of the heat sink (T2). All these relations match with the model expectations.

The second step involves considering also the fin temperatures (T3 and T4), determine which gradient temperature has to be computed (ΔT), and tune the experimental constant h. We have some preliminary numbers, allowing us to approximate the package temperature from the power drawn by the processor, but we do not show the numbers because the model is not accurate enought; the h constant does not completely match with the handbook data normally used in thermal engineering.

5 Example Laboratory

This section describes a laboratory to get some insights between compiler optimizations and energy/power and then comments some other challenging experiments using the thermal measurement abilities of the platform.

5.1 Interaction between Compiler Optimization and Energy/Power

One possible application of the platform in academia is its use in computer architecture laboratories. For example, it easily allows to study the interaction between compiler optimizations and energy/power.

The lab would be introduced by explaining the basic relationships among time, energy, and power paying attention to what changes should be expected when the optimization level rises. An outline of such introduction follows.

In a processor without Dynamic Voltage Frequency Scaling (DVFS), the execution time T_{ex} of a program can be expressed as

$$T_{ex} = N_{inst} \times CPI \times T_{cycle} \quad (1)$$

where N_{inst}, CPI, and T_{cycle} represents the total number of instructions, the average number of cycles per instruction, and the cycle time, respectively. For minimizing T_{ex}, compilers focus on reducing the total number of cycles, $N_{inst} \times CPI$. But which are the effects of this reduction on power and energy?

Assuming the simplifying assumption that static bias current does not flow in a microprocessor [20], its total power consumption is given by

$$P_{tot} = P_{dyn} + P_{sta} = C_L V_{dd}^2 f + V_{dd} I_{leak} \quad (2)$$

where P_{tot} is the total sum of the dynamic and static power. The dynamic power, P_{dyn}, is the product of the average capacitance switched per cycle (processor activity), C_L, times the square of the supply voltage, V_{dd}, times the frequency, f. The static power is the product of the supply voltage times leakage current, I_{leak} [19].

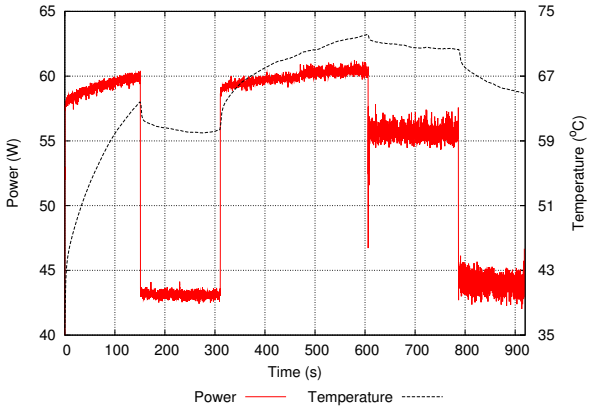
From equations (1) and (2) we observe that compiler optimizations only affect power indirectly. Regarding dynamic power, P_{dyn}, on one hand, it is difficult to establish a relationship between N_{inst} and C_L because executing more, less, or different instructions may or may not change the performed activity per cycle. On the other hand, CPI seems to impact more the dynamic power (C_L) because optimizations that rise/reduce Instruction Level Parallelism (ILP), such as instruction scheduling or dead-code elimination, can increase/decrease activity per cycle, C_L.

Static power is less affected by compiler optimizations since it depends mostly on technological parameters; however, they can affect static power when the optimizations increase/decrease the processor activity and this results in a variation of processor temperature because leakage current depends on temperature [4]. The most straightforward path for reducing static power from compilation is to add special instructions in the code for switching off processor parts as suggested by Zhang *et al.* [26]. These proposals will become more and more important in the future because as technology scales, the percentage of static power is rising [15].

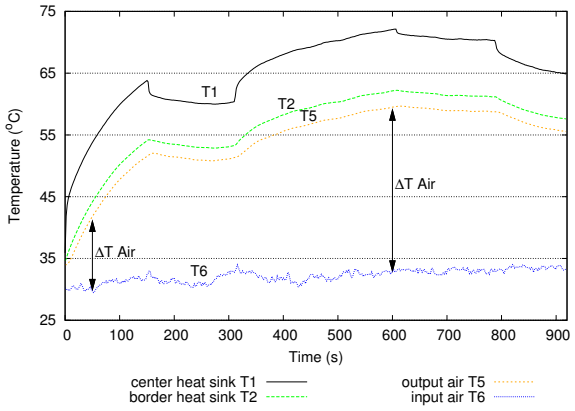
The product of P_{tot} times T_{ex} is the energy consumed by a program

$$\begin{aligned} E_{tot} &= P_{tot} \times T_{ex} = E_{dyn} + E_{sta} \\ &= C_{tot} V_{dd}^2 + V_{dd} I_{leak} \times T_{ex} \end{aligned} \quad (3)$$

where C_{tot} is the total capacitance that has been switched across all execution cycles.



(a) Temperature in thermocouple 1 and Power



(b) Temperature at thermocouples

Figure 2: Temperature and Power temporal evolution during the full execution of 473.astar compiled with `iO3prf` options.

Recalling equations (1) and (3), E_{dyn} is independent of the frequency and

$$C_{tot} = C_L \times N_{inst} \times CPI \quad (4)$$

Thus, execution-time optimization saves energy when they reduce the total number of cycles, $N_{inst} \times CPI$, because we do not expect that compiler optimizations increase significantly C_L. In deep-pipelined processors with complex decoding such as the Intel Pentium 4, this is specially true because the energy consumed in the execution stage is smaller that the energy consumed in the rest.

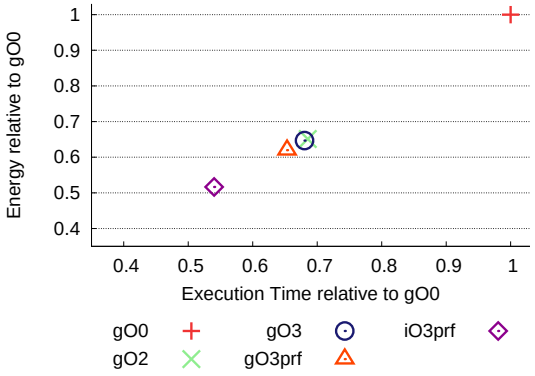
Table 1: Compiler optimization impact summary. ↓, ?, and ↑ means decrement, undetermined, and increment, respectively.

Power	N_{inst} ↓	CPI ↓
dynamic (P_{dyn})	?	↑
static (P_{sta})	?	?

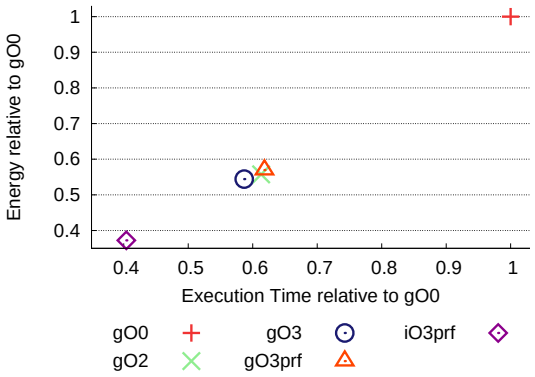
Energy	N_{inst} ↓	CPI ↓
dynamic (E_{dyn})	↓	↓
static (E_{sta})	↓	?

Table 1 summarizes all previous relations and derives the effect of decreasing either N_{inst} or CPI, assuming constant the other factor. As it can be seen, performance-oriented compiler optimizations (focused on reducing $N_{inst} \times CPI$) are beneficial for energy, and may not be power-efficient when their target is to reduce only the CPI because dynamic power can increase. Asking the students to complete this table before the laboratory session is a good assignment for ensuring that students understand the underneath theory.

5.1.1 Experimental Results



(a) Integer



(b) Floating Point

Figure 3: Average Energy and Execution time relative to `gO0`.

Table 2: Tested SPEC CPU2006 benchmarks.

Integer	Input
400.perlbench	-I./lib checkspam.pl 2500 5 25 11 150 1 1 1 1
462.libquantum	1397 8
473.astar	rivers.cfg
483.xalancbmk	-v t5.xml xalanc.xsl

Floating Point	Input
436.cactusADM	benchADM.par
437.leslie3d	-i leslie3d.in
447.dealII	23
453.povray	SPEC-benchmark-ref.ini
454.calculix	-i hyperviscoplastic
470.lbm	3000 reference.dat 0 0 100_100_130_ldc.of

Table 3: Compiler configurations with their respective optimization flags

	Compiler	Flags
gO0	gcc	-O0
gO2	gcc	-O2 -mtune=pentium4 -march=pentium4
gO3	gcc	-O3 -mtune=pentium4 -march=pentium4 -mfpmath=sse,387 -msse2
gO3prf	gcc	-O3 -mtune=pentium4 -march=pentium4 -mfpmath=sse,387 -msse2 -fprofile-generate/use
iO3prf	icc	-O3 -xN -ipo -no-prec-div -prof-gen/use

The previous relations can be verified with the proposed platform by executing multiple programs with different compiler optimizations and acquiring the energy and power measurements. For the sake of brevity, we only show the results for the relation between energy and execution time.

As a benchmark we can choose any program not spending most of the time in I/O to ensure that the impact of compiler optimization is significant in energy and power. Due to its widespread use in industry and academia SPEC CPU2006 has been our choice [8]. In order to reduce the measurement time we select the representative subset proposed by Phansalkar *et al.* [18]. The input sets for each program used in this paper are shown in Table 2. Other events of interest such as fetch stalls or instruction count can be measured with Intel Performance Tuning Utility (PTU); e.g., to compute the energy per instruction value [1].

To check the impact of compiler optimizations in energy and power we suggest to test multiple configurations of the GNU C compiler 4.1.2 (`gcc`) [7] and one configuration of the Intel C compiler 10.1 (`icc`) [12], all listed in Table 3. As a baseline, we use a configuration without optimizations, `gO0`. We also checked a production-level configuration tuned for our processor, `gO2`. Finally, we encourage using more aggressive `gcc` configurations: `-O3` without and with profiling, and `icc` at its maximum level of optimizations with profiling (`iO3prf`).

In integer, the more optimizations are applied, the better the results are. The best `gcc` configuration, `gO3prf` saves 34.7% of execution time and 38% of energy. `IO3prf` increases the gains saving 46% and 48.4% of execution time and energy, respectively. In floating point, optimizations are more effective; i.e., `gO2` (the best `gcc` configuration) saves 41.3% and 45.6% of execution time and energy, respectively. Again, `iO3prf` performs better with 59.6% and 62.8% reductions in execution time and energy.

Gains in execution time and energy are very close suggesting a strong correlation. To support this claim, Figure 4 plots execution time and energy for each benchmark. As can be seen the correlation is strong, which is in line with previous work [24, 21]. We believe that the correlation is due to the fact that the clock net, static consumption, and fetch, decoding, and control parts of the processor consume more than functional units [25]; hence, it seems than reducing the number of executed instructions is more important than its kind for improving energy consumption.

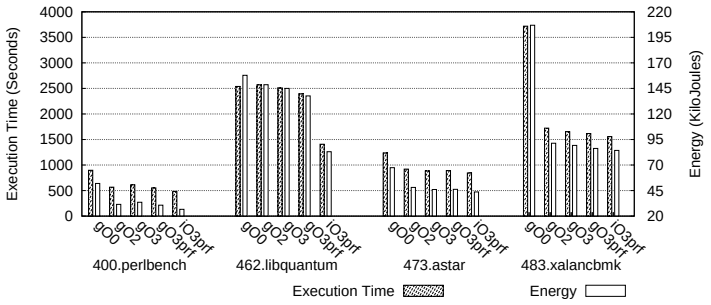
Regarding execution time, `icc` beats `gcc` in all but one benchmark, `447.dealII`. Besides, `icc` consumes less energy in all programs but `470.lbm`. To conclude, both `gcc` and `icc` reduce notably the number of executed instructions (50% and 75% on average for integer and floating point, respectively) and increase the CPI (rising also the Energy per instruction) but `icc` does it in a lower quantity.

Summarizing, the main assignments for this lab can be: to perform the measurements for the program, to verify that the table they have completed before the lab is correct, and to finish extracting the conclusions of the previous paragraphs.

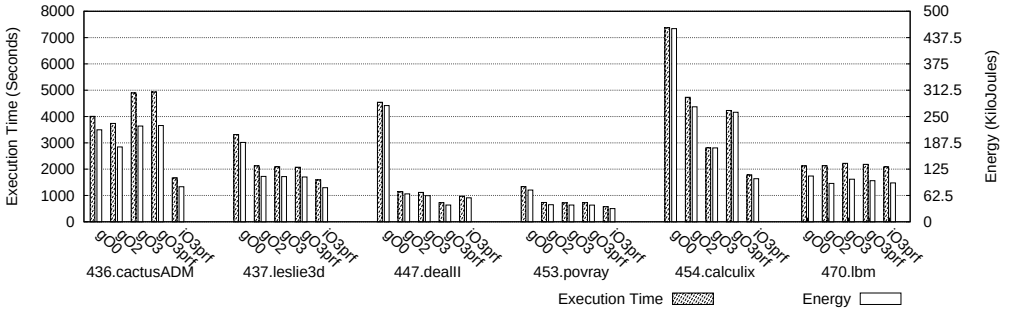
5.2 Other Experiments

The platform can be used with a more research-oriented focus such as master dissertations. For example, an outgoing work in our lab wants to obtain a power/temperature profile of individual instructions.

Since the processors' manual does not document the consumption of the instructions, we can get an estimation with the platform . For example, we have observed that stack operations rise power consumption and heat more the processor, which makes sense because stack instructions require a read/write in the cache and one increment/decrement in the stack pointer register in the same cycle.



(a) Integer



(b) Floating Point

Figure 4: Execution time and Energy per benchmark.

6. Conclusions and Future Work

This paper presents a platform for measuring energy/power and temperature in commodity PCs with an academic focus. In this work, measures are carried out in an Intel Pentium 4, but the platform can be easily ported to any commodity PCs. The acquired data can be stored to perform off-line analysis, and its accuracy enables to detect power and temperature phases.

With the platform students can, for example, study the interaction between compiler optimizations and energy/power. This laboratory enables the student to learn that on average performance optimizations are energy-efficient.

Nowadays,the platform is used and extended by a small group of students. Our next main step is to set up a whole laboratory for using it as a regular laboratory session in our Computer Architecture and Heat Transfer courses. Our ongoing work is to obtain a simple linear equation relating measured power, fan speed, and dissipating surface to compute output air temperature for using it during the introduction of the laboratories.

Our future work will try to extend the platform reducing the granularity of the sampling process. Now, the platform does not know at which code fragment or function each sample belongs. We believe that this ability will help us finding the most heat-producing instruction sequences to continue our studies on per instruction energy estimations.

Acknoledgements

The authors would like to thank the anonymous reviewers for their suggestions on this paper. Darío Suárez Gracia and Víctor Viñals Yúfera were supported in part by the Gobierno de Aragon grant gaZ: Grupo Consolidado de Investigacion, the Spanish Ministry of Education and Science under contracts TIN2007- 66423, TIN2007-68023-C02-01, and Consolider CSD2007- 00050, and the European Union Network of Excellence HiPEAC-2 (FP7/ICT 217068).

References

[1] *Intel Performance Tuning Utility 3.1 Update 3.* `http://software.intel.com/en-us/articles/intel-performance-tuning-utility-31-update-3`, 2007 edition.

[2] Analog Devices. *ADP3180, 6-Bit Programmable 2-, 3-, 4-Phase Synchronous Buck Controller*. Analog Devices, 2003.

[3] A. Asín Pérez, D. Suárez Gracia, and V. Viñals Yúfera. A proposal to introduce power and energy notions in computer architecture laboratories. In *WCAE '07: Proceedings of the 2007 workshop on Computer architecture education*, pages 52–57, New York, NY, USA, 2007. ACM.

[4] D. Brooks, R. P. Dick, R. Josepth, and L. Shang. Power, thermal, and reliability modeling in nanometer-scale microprocessors. *IEEE Micro*, 27(3):49–62, May-June 2007.

[5] E. T. E. M. B. Consortium. EnergyBench ™ version 1.0 power/energy benchmarks. http://www.eembc.org/benchmark/power_sl.php, 2008.

[6] S. P. E. Corporation. SPECpower_ssj2008 benchmark suite. http://www.spec.org/power_ssj2008/, 2008.

[7] Gcc team. *GCC 4.1.2 Manual*. http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/. Free Software Foundation, February 2008.

[8] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, 2006.

[9] C. Hu, J. McCabe, D. A. Jiménez, and U. Kremer. Infrequent basic block-based program phase classification and power behavior characterization. In *Proceedings of The 10^{th} IEEE Annual Workshop on Interaction between Compilers and Computer Architectures*. ACM Press, 2006.

[10] A. T. Inc. Adlink pci-9112 data acquisition card. http://www.adlinktech.com/PD/web/PD_detail.php?cKind=&pid=29&seq=&id=&sid=, 2008.

[11] F. P. Incropera, D. P. DeWitt, T. L. Bergman, and A. S. Lavine. *Fundamentals of Heat and Mass Transfer*. Wiley, 6^{th} edition, 2007.

[12] Intel. *Intel C++ Compiler 10.1 Profesional edition*. http://www.intel.com/cd/software/products/asmo-na/eng/277618.htm, 2007 edition.

[13] Intel. *Intel® Pentium® 4 Processor in the 478-Pin Package Thermal Design Guidelines*. Intel Corporation, 1^{st} edition, May 2002.

[14] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93, Washington, DC, USA, 2003. IEEE Computer Society.

[15] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Number 4 in Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2008.

[16] F. J. Mesa-Martinez, M. Brown, J. Nayfach-Battilana, and J. Renau. Measuring performance, power, and temperature from real processors. In *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*, page 16, New York, NY, USA, 2007. ACM.

[17] F. J. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau. Power model validation through thermal measurements. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 302–311, New York, NY, USA, 2007. ACM.

[18] A. Phansalkar, A. Joshi, and L. K. John. Analysis of redundancy and application balance in the spec cpu2006 benchmark suite. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 412–423, New York, NY, USA, 2007. ACM.

[19] J. Rabaey. *Low Power Design Essentials*. Springer, 2009.

[20] J. M. Rabaey, A. Chandrakasan, and B. Nikolić. *Digital Integrated Circuits. A design perspective*. Prentice Hall Electronics and VLSI series. Prentice Hall, second edition, 2003.

[21] J. S. Seng and D. M. Tullsen. The effect of compiler optimizations on pentium 4 power consumption. In *Seventh Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT'03*, page 51, 2003.

[22] P. Technologies. *USB TC-08 Temperature Logger User's Guide*. Pico Technologies Limited, 2007.

[23] Tektronix. Tektronix tpc-312 current probe. http://www2.tek.com/cmswpt/psdetails.lotr?ct=PS&ci=13540&cs=psu&lc=EN, 2008.

[24] M. Valluri and L. John. Is compiling for performance == compiling for power? In *Fifth Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT'00)*, page 51, 2001.

[25] W. Wu, L. Jin, J. Yang, P. Liu, and S. X.-D. Tan. A systematic method for functional unit power estimation in microprocessors. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 554–557, New York, NY, USA, 2006. ACM.

[26] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Compiler-directed instruction cache leakage optimization. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, page 208. IEEE Computer Society, 2002.