Fast Energy Evaluation of Embedded Applications for Many-core Systems

Felipe Rosa[1], Luciano Ost[2], Thiago Raupp[3], Fernando Moraes[3], Ricardo Reis[1],

[1] UFRGS - Instituto de Informática - PGMicro/PPGC
Av. Bento Gonçalves 9500 Porto Alegre, RS - Brazil
{frdarosa,reis}@inf.ufrgs.br

[2] LIRMM – 161 rue Ada, Cedex 05 - 34095 Montpellier, France
ost@lirmm.fr

[3] FACIN-PUCRS - Av. Ipiranga 6681- 90619-900, Porto Alegre, Brazil
{thiago.raupp, fernando.moraes}@pucrs.br

Abstract— **The growing concerns of energy efficiency and performance scalability motivate research in the area of many-core embedded systems. The software development of such systems plays an important role on the system performance, while accounting for a significant part of the total energy consumption. Thus, it becomes imperative to consider the software energy consumption at early stages of the software development. This paper proposes an instruction-driven energy analysis approach that provides an accurate and practical way of evaluating software energy cost at the speed of up to 1.8 MIPS. Results show that the accuracy of our approach varies from 0.06% to 8.05% when compared to a gate-level implementation.**

Keywords **— Instruction-driven energy model, fast and accurate energy evaluation, JIT-based simulation, OVP.**

I. INTRODUCTION

Many-core embedded systems must execute different applications in parallel, requiring high performance allied to low energy consumption, which is more critical than high-speed operation in battery-driven devices [1]. With 200-core chips available in the market [2], energy-efficiency software development becomes a challenge of paramount importance in many-core system design, since it has a significant contribution to the overall system energy budget. To assess the energy impact of complex software stacks (operating system - OS, drivers, etc.), several software and hardware parameters must be tuned and evaluated properly, considering a large design space.

The resulting complexity restricts the use of board-oriented and detailed simulation approaches for software energy evaluation, especially for many-core architectures. While specialized board designs produce accurate results, they require a substantial development effort to setup/port the software stacks. Further, physical boards can be expensive, with limited resources (e.g. number of CPUs, memory), as well as poor debuggability due the lack of internal observability and controllability of its components. Compared with board-oriented approaches, transistor or gate-level simulators are typically slower and the amount of memory required by these approaches is too high. Nevertheless, they facilitate the analysis of various architectural and software options, enabling to identify performance/energy bottlenecks of the target system.

To achieve efficient exploration of many-core systems, the use of *virtual platforms* is a key tool for adequate design-time assessment of the performance-oriented and energy-efficiency strategies. Such frameworks support concomitant hardware and software development, while providing more flexibility and debuggability.

While producing accurate results, quasi-cycle virtual platforms achieve limited simulation speed (around 200 KIPS – kilo instructions per second [3]). In contrast, simulators such as the Open Virtual Platforms (OVP) OVPsim can achieve simulation speeds of up to 100 MIPS, at the cost of limited accuracy. Such simulators target software development and rely on just-in-time (JIT) dynamic binary translation, i.e., dynamic translation and optimization of target machine code to host machine code.

This paper contributes by including an accurate energy model in the OVPSim simulator, making it suitable for fast software energy cost estimation at early design stage. Another contribution is an extensive model evaluation by using several benchmarks, while comparing it to a gate-level implementation.

The rest of this paper is organized as follows. Section 2 discusses related work in instruction-driven energy models. Section 3 provides the fundamental concepts of proposed instruction-driven energy model. Section 4 gives an extensive evaluation of speedup, accuracy and cost of our simulation. Finally, Section 5 points out conclusions and directions for future work.

II. RELATED WORK

To speed up the energy cost evaluation at early stages of the software development, authors are investigating alternatives for fast energy/power estimation using high-level models. Each of these entails different reference models; calibration practices and simulation speed/accuracy tradeoffs, allowing the exploration of different design space aspects. For instance, proposed models are calibrated from prototyping boards or simulation of detailed CPUs descriptions (e.g., gate-level, RTL).

Table I State-of-Art in instruction-driven energy models.

Approach	Reference model	Claimed Accuracy	Benchmark suite	Description
Lee et al. 2001 [6]	ARM7TDMI	Average 2,5% and worst case 6,33%	36 randomly-generated instructions	Model based in a linear regression analysis.
Garcia et al. 2002 [8]	Gate-level estimation using an ARM920	Not available	MPEG-4 video decoder	Inclusion of a power model calibrate in gate-level activity in a System-level Cycle-Accurate simulator
Kalla et al. 2003 [10]	Synthesizable RTL of a SPARC	Energy less than 5% and per-cycle power inside 15% of error	Bubble Sort, Heap Sort, Insertion Sort, Key 3 and 3D image processing	Model based in Active and Stall consumption for each individual module of the architecture, refined with inter-instruction effect. Additionally provides the maximum and the minimum of power.
Nikolaidis et al 2003 [7]	ARM7TDMI	5%	A few instructions	Abstract model for pipeline with static, inter-instruction, and pipeline power.
Konstantakos et al. 2006 [5]	Motorola HC908GP32	Not available	Not available	The instructions are divided in groups by the cycle's length.
D. Lee et al. 2006 [11]	Gate-level estimation for M32R-II and SH3-DSP	Average 3% and worst case 16%	JPEG and MPEG2 encoders, compress, FFT and DCT	Training benchmarks are used in conjunction with a gate level simulator and liner optimization to generate several parameters to describe frames of instructions. Afterward this parameters are utilize together with ISS.
Castillo et al. 2007 [12]	Arm ISS, arm-elf-gdb, for a ARM9TDMI and ARM TRM	Less than 11%	Bubble Sort, FIR, Array, Fibonacci and Quicksort	An online analysis of the source-code without requiring simulation or even compilation. Based in the mean energy per instruction calculate from values provide by ARM Manual. Detailed study about the operators in C e.g. + = >> and their costs in meter of instructions.
Sultan et al. 2009[9]	Synthesizable RTL of a LEON3	Not available	Not available	Propose of an instruction level power model profiling each instruction in different stages of a pipelined processor. The aim is measure the activity generate in the processor and taking in count the capacitance to calculate the power.
Callou et al. 2011 [13]	NXP LPC2106 with an ARM7TDMI-S	7% in average	5 applications	Stochastic approach based on Coloured Petri nets and source code analysis
Bazzaz et al. 2012 [4]	AT91	Less than 6%	8 MiBench benchmark	ISS Model calibrated from real measures. Complete model with static, inter-instruction, and pipeline power.
Proposed approach 2014	Gate-level estimation	Between 0,06% and 8%	19 benchmarks from WCET and in house applications	Instruction-driven model calibrated from the switching activity of the processor internal components. Run-time model developed on the basis of OVP API that monitors the instructions executed by a given CPU.

In the case of prototyping boards, the power information is captured from a precision resistor positioned between the power supply and the power input pin [4]–[7]. The use of physical information can aggregate precision to the high-level models (error varying from 2.5% to 7% as presented in the third column of Table I). However, to measure the power of each instruction, additional and expensive hardware (e.g. high performance oscilloscopes) are required. Another drawback of this approach is the difficulty of accessing/isolating individual modules inside the processor due to internal structure and connections (e.g. Flash, Rom, SPI, AD, and DC).

In simulated-based techniques, the required information is extracted from low-level simulators (e.g. SPICE, gate-level), in which a hardware description is used to execute input benchmark applications and to profile the power of each instruction. For example, in [8] an instruction set simulator (ISS) is enriched with an energy model based on the mean switching activity of the processor, which is modeled by two states, *active* and *NOP*. A similar approach is presented in [9], which considers the average switching activity of an LEON3 processor simulated at RTL level. In this work, the power is computed according to the number of transitions generated in response to a certain instruction that is fetched from BootROM of LEON3.

In [10] is developed a tool called SEA targeting power and energy estimation for an SPARC processor. This work uses a

gate-level simulation to provide energy information to their model. In this approach, the instructions are classified in memory, not-memory and specials, while we adopted seven groups. In turn, the work proposed in [11] combines linear programming, gate-level and ISS simulation to extract energy parameters. The reference energy values are computed according to a fixed number of instructions pre-defined during the energy characterization flow.

Authors in [12] propose obtaining energy values directly from the analysis of the source-code without requiring simulation or even compilation. A further higher-level approach is proposed in [13], in which the source-code is converted in a Colored Petri net model, which is used to estimate the energy cost of a given application.

Reviewed approaches focus on creating instruction-driven models, which compute energy/power values by observing the sequence of executed instructions. One difference between such approaches is the calibration process. For instance, in [4] authors evaluate instructions individually to feed the instruction-driven model, while in [6] fixed length instruction groups are used. Authors argue that different transition scenarios may significantly affect the energy estimation. For that reason, some works such as [4][7][10] also calculate the inter-instruction energy, i.e. the energy required to switch from one to another.

Another distinction lies in the energy evaluation process. For instance, the approach proposed in [12], differs from the other works in the sense that it translates the source code in an intermediary code representation, which is used to estimate the application energy consumption. This approach does not require simulation that may decrease the energy evaluation effort. However, to predict the behavior of loops and branches only by code inspection is not a trivial task that may pose other design/evaluation challenges.

Our contribution distinguishes from previous works by enhancing the OVPSim (JIT-based simulator) with energy evaluation capability allowing faster and accurate exploration of energy-efficiency software development for large-scale architectures. Contrary to the most of reviewed approaches, our approach is ISA/CPU-oriented, and it relies on a run-time approach, thus everything is transparent to the software engineer. Therefore, once calibrated whatever OS/application can be ported, modified and its energy-efficiency can be evaluated without any code modification or re-calibration phase.

III. Instruction-driven Energy Model

This Section presents the instruction-driven energy estimation model that was integrated into the OVPSim simulator. The proposed approach is executed at run-time, i.e. full computation is executed concomitantly with the system simulation, eliminating huge trace files, as well as pre- or post-processing software/application profiling. Manufacturers do not usually provide detailed energy information. In some cases, the average and maximum energy and/or power picks are provided. To overcome this restriction is necessary to acquire information through a characterization phase. Fig. 1 illustrates the proposed evaluation flow that comprises the characterization flow and the simulation phase.

A. Characterization Flow

The first and most important phase is the characterization, which profiles the energy spent by each instruction belonging to the target ISA. The proposed characterization flow is validated taking as reference the Plasma processor [14], a 32-bit RISC processor based on the MIPS architecture with a 3-stage pipeline. The characterization flow is executed once per ISA architecture and it comprises four main steps: (*i*) benchmark development; (*ii*) activity measurement; (*iii*) power acquisition; (*iv*) energy computation. In Fig. 1, the numbers from 1 to 10 are used to describe intermediary files, while the letters from A to D represent the adopted tools.

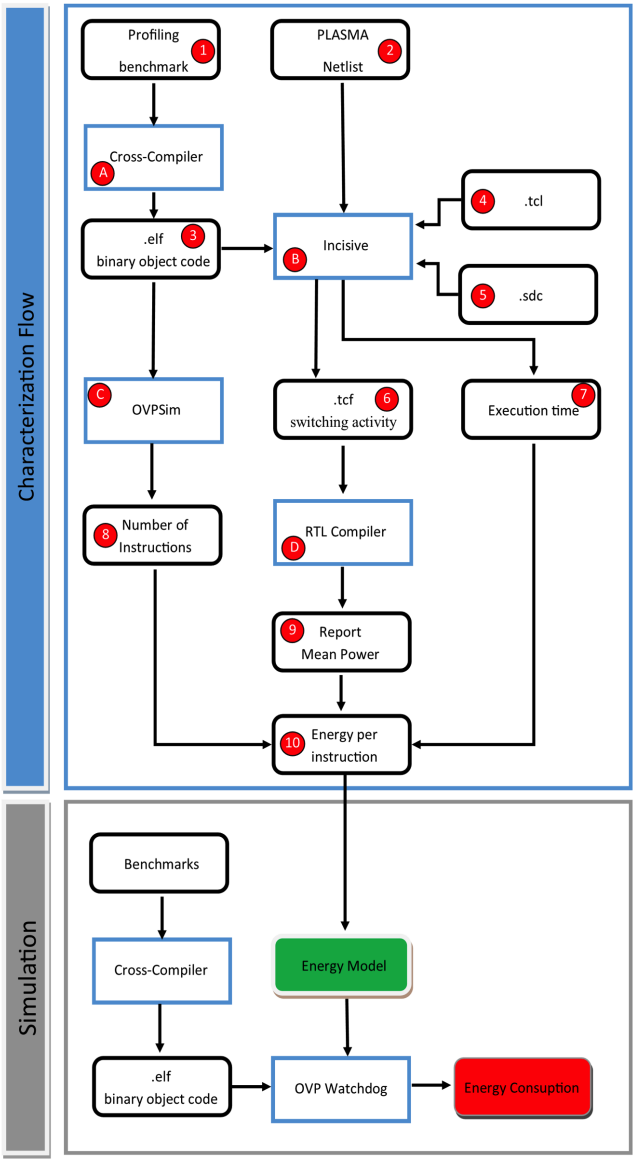

Figure 1 - Proposed instruction-driven evaluation flow.

The first step of the characterization flow encompasses developing the benchmarks that will be used to profile the energy consumption for each instruction (1 in Fig. 1). To reduce the computation cost of our model, instructions are classified in seven groups due their behavior in the processor data-path: (*i*) arithmetic, (*ii*) logical, (*iii*) move, (*iv*) branches, (*v*) load/store, (*vi*) nops, and (*vii*) shifts. One practical example is the close relationship between instructions such as *add* and *addiu* or between *lw* and *sw*. Note that the mnemonic *move* is considered in this work as arithmetic instruction through the use of a pseudo-instruction implementation (performed by an *lui* and *ori*).

To profile the energy consumption of each instruction group (1 in Fig 1), applications were carefully developed in a way that at least 90% of the executed instructions would belong to the target group, including the possible variations of the same instruction (e.g. *add, addi, addiu*, etc.). Previous experiments using higher percentages (e.g. 95%), showed a negligible difference. Note that multiplication and division instructions are modeled as 12 arithmetic instructions each, since our Plasma version takes 12 cycles to execute them. Further, an application benchmark was created to characterize the pipeline stall as a *nop* instruction.

Each application is executed in the OVPSim simulator (C in Fig1) to verify its correctness and to extract the exact number of executed instructions (8 in Fig1). Each application executes, in average, 35 thousands instructions, which requires less than one 1 second of simulation.

The Plasma is synthetized with Cadence RTL Compiler tool (2 in Fig 1) targeting a 65nm low power library from ST Microelectronics. Then, each application is simulated using Cadence Incisive simulation tool (B), taken as inputs: the Plasma netlist (2), the application object code (3), a *tcl* script (4), and the *sdc* file containing the timing constraints (5). The simulating is executed until the end of the application. As a result, a *tcf* file (6) is generated. This file contains statistic information about the switching activity of each cell and wire in the netlist. In addition, the exact execution time of each application is collected (7).

Finally, the power evaluation is executed. Cadence RTL Compiler (D) also performs this task; the tool reads the netlist (2) and computes de average power consumed by each cell based in their switching activity information in the *tcf* (6) file. Subsequently, the tool produces a report containing the average power consumption (9) for the application.

The final step comprises computing the average energy spent by each characterized group (10). Associating the average power (9) and execution time (7) collected in the previous step with the number of instructions (8), the energy consumed per instruction group is obtained using Equation (1). This flow is repeated for each instruction class.

$$Average\ energy = \frac{execution\ time\ (\mu s)\ x\ power\ (mw)}{executed\ instructions}\ (nJ) \quad (1)$$

TABLE II summarizes the energy results for each instruction group. Results in the column "number of instructions" are obtained through the OVPSim simulation. Results in columns "power" and "execution time" are obtained through the gate-level simulation. The total energy consumption ("energy" column) is obtained by multiplying the number of instructions by the total execution time. Then, with the number of executed instructions and the total energy consumed, it is possible to compute the energy consumed by each instruction ("Energy per Inst." column).

TABLE II AVERAGE ENERGY PER INSTRUCTION GROUP (CALIBRATION FREQUENCY 100 MHZ, WHERE 1 US CORRESPONDS 100 CLOCK CYCLES).

Groups	Power (mW)	Execution Time (us)	Energy (nJ)	# of Inst	Energy per Inst (nJ)
Arithmetic	6,456	342,755	2212,826	34764	0,0636528
Jump	6,046	102,600	620,320	10224	0,0606729
Load-Store	4,094	1042,800	4269,223	48561	0,0879146
Logical	4,469	349,735	1562,966	35462	0,0440744
Move	3,129	480,725	1504,189	39363	0,0382133
NOP	2,141	257,155	550,569	26130	0,0210704
Shift	3,824	298,735	1142,363	30362	0,0376247

B. WATCHDOG MODULE

As mentioned before, our energy model relies on monitoring at run-time the instructions executed by a given CPU. The monitoring process was developed based on OVP APIs and integrated into an extension of a Watchdog component proposed in [15]. Fig. 2 shows the three main Watchdog modules: (*i*) disassembler, (*ii*) a hash table with pre-characterized groups of instructions, and (*iii*) internal data structures. Both hash table and energy information are calibrated according to an instruction set architecture (ISA).

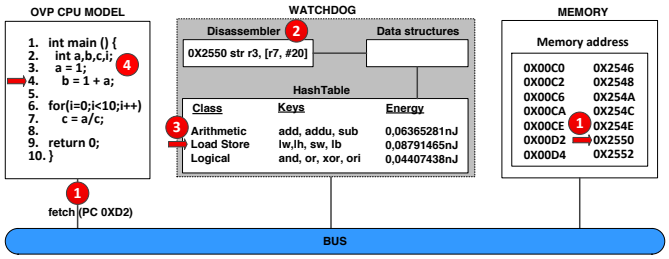

Figure 2 - Block diagram of developed watchdog module.

In the disassembler module, each instruction (i.e. binary code) fetched from the memory unit is converted to a string and afterwards subdivided in other substrings. The purpose here is to isolate the instruction mnemonic from the instruction register arguments in order to feed the hashtable. As a means to disassemble binary code instructions, our implementation employs the *icmDisassemble* function, which is provided by the ICM API. This function call disassembles an arbitrary memory position for each arbitrary CPU instance used in simulation model (for the sake of simplicity only one CPU is used as example in Fig. 2). The underlying function requires arguments such as: (*i*) the CPU model object, (*ii*) the

target memory address, and (*iii*) arguments of the disassembled instruction (e.g. 0X2550 STR r3,r2]).

The second module is a Hashtable, which is used to store the energy information and to speed up the search process during the simulation phase. In turn, the data structure module is responsible to process information provided by previous modules considering the CPU state information.

C. Simulation Phase

In OVPSim all CPUs, busses, and memory models are created at run-time through linked libraries. The same process is applied to our Watchdog module and its internal components. The numbers from 1 to 4 in Fig. 2 are used to describe the Watchdog behavior during the simulation phase.

After the platform simulation begin, whenever an instruction is fetched from the memory (1) a callback is triggered, thus activating the Watchdog. Inside the first module, the binary code of each instruction is acquired using the program counter (PC) register, thus the binary code is disassemble, divided in sub-strings, identifying the instruction that must be executed (2). The identified instruction is employed as a hashtable key to discover which class (e.g. arithmetic, load, logical) such instruction belongs (3). The hashtable implementation was considered to remove the simulation bottleneck inherent to the use of linked lists (model implementation not reported here).

Hence, energy consumption of this instruction is computed, considering the predefined energy information (calibration process). Once, the energy consumption is computed, the instruction is executed in the CPU (4). At the end of the simulation is possible to retrieve energy reports, including individual energy consumed per CPU, number of memory access (read and write operations), among others.

IV. Results

A. Experimental Setup

Application benchmarks that permit exploiting and assessing performance of embedded CPUs were selected from different research domains. For instance, the 11-selected applications of the worst-case execution time [16] (WCET) benchmarks vary in terms of execution time, number of loops, matrixes and array size. Other in-house application benchmarks using well-known algorithms are also employed.

Since OVPSim uses the target CPU's binary code to perform emulation on a host machine, all simulation scenarios were executed multiple times in order to capture meaningful results. Note that we are using the same cross-compile, libraries, and compilations flags in order to create almost identical binaries. Further, we used in the Mentor Graphics Sourcery Tools version 4.8.1 the following flags -mips1 -g -Ttext 00000000.

B. Accuracy of the Model

Fig. 3 compares the energy consumption for each application benchmark, considering results obtained from gate-level simulation (i.e. Cadence Incisive) and the proposed instruction-driven energy model in OVP. Gray bars correspond to the difference between each result, showing the high accuracy achieved with the proposed model (error below 8%).

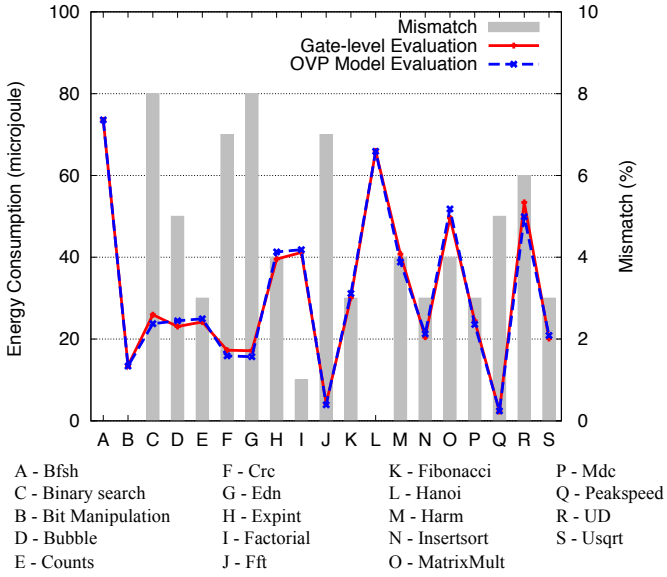

Figure 3 - Application benchmark energy consumption: gate-level simulation vs proposed instruction-driven energy model in OVP.

C. Achieved Speedup

Fig. 4 presents the achieved simulation speeds in MIPS when comparing both the instruction-driven energy model in OVP and the gate-level simulation. Results show the gain in terms of speedup is wide, ranging from 10x to 1500x (gray bars) depending on the application benchmark nature. Note that all analysis using our proposed energy model in OVP required less than a minute of simulation.

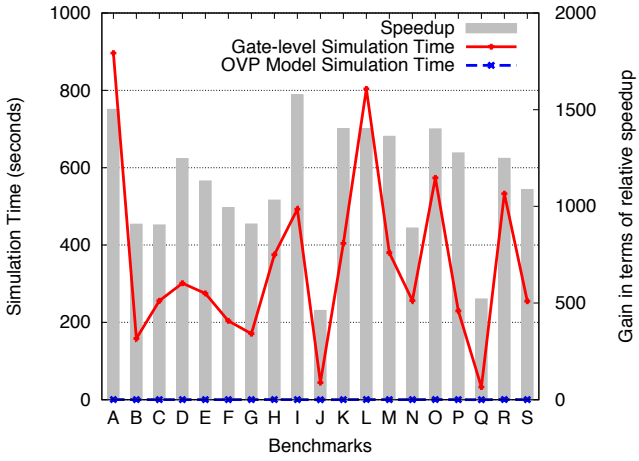

Figure 4 - Gain in terms of speedup: gate-level simulation vs proposed OVP energy model. Letters (A-S) symbolize the application benchmarks described in Fig. 3.

Achieved high accuracy and simulation speedup of the proposed energy model, allied with the design flexibility and debugging features inherent to OVP, introduces an efficient tool capable to assist designers in the software development. In this direction, we claim that software engineers can validate

the functional behavior of the entire software stack (e.g. OS/application) executing it onto a given CPU architecture, using the original OVP. Then, software engineers may use the proposed OVP extension in a still reasonable simulation speed (i.e. 1.8 MIPS) to investigate if target software stack can be executed according to the energy requirements.

D. Model Scalability

Developing high-level models and simulators that scale to hundreds of cores becomes a stringent requirement to investigate many-core systems. This Section investigates the performance scalability of the proposed model when scaling the number of watchdogs and CPU cores.

Fig 5 shows the speedup obtained using the proposed instruction-driven energy model, considering many-core systems varying the number of CPUs from 10 to 1000, each of which executes an instance of FFT. Note that for each CPU there is one Watchdog module instance (as illustrated in Fig 2). Results show the simulation speed remains between 1.6 and 1.8 MIPS. Such results prove that proposed approach is suitable for fast energy exploration of large many-core systems. The gray line represents simulation speed of the proposed energy model, considering the full statistics report generation (e.g. executed instructions, energy per CPU, as illustrated in Fig. 6). In addition to that, the yellow one includes the cost of capturing the number and the type of memory accesses, while black line comprises both.

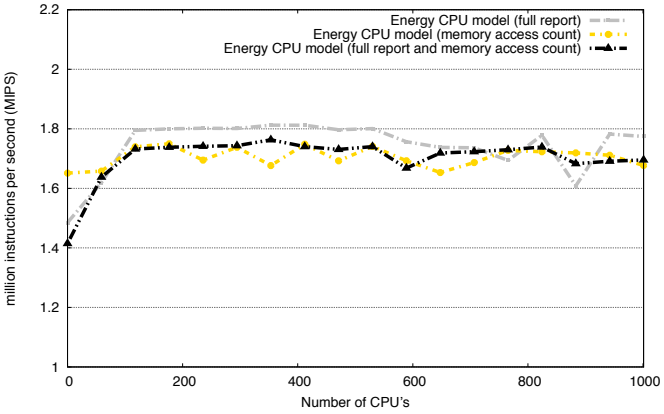

Figure 5 – Simulation speed when scaling the number of watchdogs and CPUs from 10 to 1000.

```
Watchdog ------------------------------------------------
Watchdog CPU 0
Watchdog ------------------------------------------------
Watchdog ------------------------------------------------
Watchdog Total energy of arithmetic instructions     87.869536 nj
Watchdog Total energy of branchs instructions        19.231249 nj
Watchdog Total energy of load instructions         3958.050043 nj
Watchdog Total energy of logical instructions         0.491963 nj
Watchdog Total energy of move instructions            7.976288 nj
Watchdog Total energy of shift instructions          26.868314 nj
Watchdog Total energy of mult instructions           77.531944 nj
Watchdog Total energy of NOP instructions            39.484758 nj
Watchdog Total energy consumption                  4217.504095 nj
Watchdog ------------------------------------------------
Watchdog Number of executed instructions  47835
Watchdog ------------------------------------------------
```

Figure 6 – One CPU report example.

V. Conclusion

This paper addressed the challenge of making JIT-based simulators able to estimate applications' energy consumption. As case study, we presented a fast and accurate instruction-driven energy model, which was integrated into the OVPSim. The proposed model is applicable to different types of CPUs and it relies on run-time basis, eliminating huge trace files, as well as pre- or post-processing software/application profiling. A number of experiments were presented, showing accuracy and an important speedup to obtain energy vales.

References

[1] N. Miura, Y. Koizumi, Y. Take, H. Matsutani, T. Kuroda, H. Amano, R. Sakamoto, M. Namiki, K. Usami, M. Kondo, and H. Nakamura, "A Scalable 3D Heterogeneous Multicore with an Inductive ThruChip Interface," *IEEE Micro*, vol. 33, no. 6, pp. 6–15, Nov. 2013.

[2] "MPPA MANYCORE: a multicore processors family - Many-core processors - KALRAY - Agile Performance." Available at: http://www.kalray.eu/products/mppa-manycore/.

[3] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2013, pp. 475–486.

[4] M. Bazzaz, M. Salehi, and A. Ejlali, "An Accurate Instruction-Level Energy Estimation Model and Tool for Embedded Systems," *IEEE Trans. Instrum. Meas.*, vol. 62, no. 7, pp. 1927–1934, Jul. 2013.

[5] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, and T. Laopoulos, "Energy Consumption Estimation in Embedded Systems," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 4, pp. 797–804, Apr. 2008.

[6] S. Lee, A. Ermedahl, S. L. Min, and N. Chang, "An Accurate Instruction-Level Energy Consumption Model for Embedded RISC Processors" *ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, New York, NY, USA, 2001, pp. 1–10.

[7] S. Nikolaidis, N. Kavvadias, T. Laopoulos, L. Bisdounis, and S. Blionas, "Instruction Level Energy Modeling for Pipelined Processors," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, J. J. Chico and E. Macii, Eds. Springer Berlin Heidelberg, 2003, pp. 279–288.

[8] A. B. Abril Garcia, J. Gobert, T. Dombek, H. Mehrez, and F. Petrot, "Cycle-accurate energy estimation in system level descriptions of embedded systems," in *9th International Conference on Electronics, Circuits and Systems, 2002*, vol. 2, pp. 549–552.

[9] S. Sultan and S. Masud, "Rapid software power estimation of embedded pipelined processor through instruction level power model," in *International Symposium on Performance Evaluation of Computer Telecommunication Systems, 2009. SPECTS 2009*, vol. 41, pp. 27–34.

[10] P. Kalla, J. Henkel, and X. S. Hu, "SEA: fast power estimation for micro-architectures," in *5th International Conference on ASIC, 2003. Proceedings*, 2003, vol. 2, p. 1200 Vol.2–.

[11] D. Lee, T. Ishihara, M. Muroyama, H. Yasuura, and F. Fallah, "An Energy Characterization Framework for Software-Based Embedded Systems," *IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia*, 2006, pp. 59–64.

[12] J. Castillo, H. Posadas, E. Villar, and M. Martínez, "Energy Consumption Estimation Technique in Embedded Processors with Stable Power Consumption based on Source-Code Operator Energy Figures," presented at the XXII Conference on Design of Circuits and Integrated Systems, 2007.

[13] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, and P. Cunha, "Energy Consumption and Execution Time Estimation of Embedded System Applications," *Microprocess Microsyst*, vol. 35, no. 4, pp. 426–440, Jun. 2011.

[14] "Plasma CPU." [Online]. Available: http://plasmacpu.no-ip.org/. [Accessed: 21-Apr-2014].

[15] Rosa, F., Ost, L., Reis, R. and Sassatelli. "Instruction-driven Timing CPU Model for Efficient Embedded Software Development using OVP". In: IEEE ICECS, 2013, pp. 855 – 858.

[16] A. B. Jan Gustafsson, "The Mälardalen WCET Benchmarks: Past, Present And Future.," pp. 136–146, 2010.