Evaluation of Energy Savings on a VLIW Processor through Dynamic Issue-width Adaptation

Juan S. P. Giraldo[1], Anderson L. Sartor[1], Luigi Carro[1], Stephan Wong[2] and Antonio C. S. Beck[1]
[1] Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Brazil
juan.giraldo@ufrgs.br, {alsartor, carro, caco}@inf.ufrgs.br
[2] Computer Engineering Laboratory, Faculty of EEMCS, Delft University of Technology, The Netherlands
j.s.s.m.wong@tudelft.nl

***Abstract*—The development of energy efficient hardware has been a trend in microprocessor design for the last two decades. VLIW processors are a representative example, since they have a simpler design and competitive performance, because their ILP exploitation is done statically by the compiler. In this paper, we study the energy savings that could be obtained by adapting such microarchitecture according to the current program phase. Our contribution is twofold. First, by executing a set of benchmarks on the ρ-vex configurable softcore VLIW processor, and by modifying the number of issues, we show the potentials of energy reduction. Then, with this information in hand, we developed an oracle experiment to dynamically vary the issue width of the processor according to the phase behavior, considering two different phase granularites. The potential energy savings using this policy could be as high as 81.5% when compared with the static version, executing the MiBench set.**

Keywords—VLIW; adaptive processor; energy consumption

I. INTRODUCTION

The microprocessor industry witnessed the birth of a new design paradigm in the last decades. The prevalence of mobile devices and the increasing implementation problems that arise with higher operating frequencies and aggressive out-of-order processors put energy consumption in evidence. The growing performance that electronics market demands from new computer systems generates an important trade-off: the consumer needs the highest performance with the least possible energy consumption [1].

VLIW design is a microarchitecture solution for such requirements, since complexity is moved from hardware to software. A superscalar processor exploits ILP (Instruction Level Parallelism) through expensive dynamic scheduling hardware, whereas in VLIW processors a compiler does most of this work, statically. It results in a simpler hardware design that still takes advantage of multiple execution units without incurring high resources overhead.

On the other hand, one of the main issues when it comes to designing a VLIW processor from scratch is about project decisions, such as choosing the right number of issues and the register file size. The number of issues influences the level of availability of execution units, which determines the ILP available for the compiler, and the register file size determines the number of registers that the compiler will be able to manage. By choosing high values for these parameters, performance will likely be increased. However, it will also have the drawback of increasing the area and power dissipation.

Moreover, the resource demands vary according to the application and workload. However, not only different applications will present distinct needs for resources: even the needs of a single application may vary throughout time. For instance, some parts of the program may exploit more ILP by computing several arithmetic operations, while others may present less ILP for being memory bound. These intervals with similar behavior are defined as phases [2]. Therefore, for a given application, several phases, with different hardware needs, may be present, which will impact the performance and energy consumption of the system.

Let us consider a scenario where the resources are set to comprise the phase with the highest ILP in the application, in order to achieve the best performance. In this case, when processor executes a phase with low ILP, the idle resources will continue to dissipate power, increasing the energy consumption of the system. On the other hand, if the resources are set to the lowest possible ILP, the performance will be highly affected. Therefore, the optimal scenario is to combine both performance and low energy consumption by having all the resources available for high ILP phases and turn off the idle hardware when a phase with low ILP starts executing.

Therefore, this work has two main purposes:

- Describe quantitatively the impact of the aforementioned architectural design choices for energy consumption, performance, and area.
- Analyze the potential energy savings that could be obtained by dynamically adapting the VLIW microarchitecture according to the program phase, using two different granularities: coarse (granularity of 5% of the total number of executed instructions) and fine (granularity of basic blocks).

The rest of this paper is organized as follows. Section II describes work related to VLIW microprocessors and adaptable systems. Section III shows the potential of optimization by analyzing the impact of design choices on performance and energy consumption. Section IV discusses two approaches for evaluating the phases of an application. Section V describes the oracle experiment performed to evaluate the energy savings

978-1-4673-8276-2/15/$31.00

potential of choosing the most appropriate issue-width for a given phase of the program. Finally, Section VI summarizes our conclusions.

II. Related Work

A. VLIW Processors

As already discussed, VLIW architectures are an alternative to superscalar designs, which exploits ILP through compiler technology instead of using hardware resources. The compiler is responsible for building long instruction words, which are composed of various independent operations that will execute at the same time. The main function of a VLIW hardware is to split each word and distribute the operations among the functional units (FUs) at run-time.

A great part of the commercially available VLIW processors utilizes a fixed issue width, such as TMS320C611 from Texas Instruments, S231 from STMicroelectronics or TriMedia series from NXP. Some efforts for reconfiguring VLIW systems can be found in [3], and for superscalar systems in [4] and [5]. Their focus is on performance improvements for multicore systems through core fusion and selective use of the processors involved. This means that the adaptability of the processor is carried out by merging simpler cores into a more complex one and by disabling the processing units that are not necessary, depending on the application at hand. In these papers, only performance is considered. By contrast, the current research is mainly focused on analyzing the influence that such adaptive architectures have over energy consumption and other metrics.

B. Phase behavior of programs

One of the main ideas that this work leverages is the concept of dynamic behavior of programs. A great part of the applications have different behavior on even the largest of scales. For example, along the execution time, the program could be completely memory bound; it can repeatedly stall on branch mispredicts; or it could mostly be executing arithmetic instructions. A phase can be defined as a set of intervals within a program's execution that have similar behavior, regardless of temporal adjacency [2]. In this way, a phase can reoccur multiple times through the program's execution.

In [6], it was developed an analysis of all SPEC 95 programs to evaluate the dynamic behavior of a number of variables, such as branch prediction, instructions per cycle (IPC), RUU (Register Update Unit) occupancy, cache behavior etc. It used basic blocks as a basic unit for further measurement analyzes. The results showed that programs exhibit phases, which means that the variables mentioned above are stable within specific time intervals due to the cyclic behavior of the running application. Other approaches, like [7], use subroutines to classify the phase behaviors. It uses a hardware call stack to measure the time that each part of the code is using the CPU, taking into account nesting. If the time that is spent in one sub-routine is greater than a preset value, it is counted as a new phase and the associated information is saved in memory.

Several works have already explored coarser grains for phase behavior analysis [8] [9], which means that each phase is composed of millions of instructions. The metrics that are extracted from each one of the phases, such as IPC or branch miss-prediction, are averaged along a big quantity of cycles. These efforts have been motivated by finding ways to optimize the global behavior of programs via software. However, a system based on such a coarse-grained phase behavior could be losing important information about the particularities of the code that only would arise when a finer granularity is used. The drawback of fine-grained approaches is that the overhead must be effectively managed to avoid performance drops or increment of the energy consumption.

C. Clock and power gating

Clock and power gating are techniques to reduce power dissipation of the system, and, consequently, energy consumption. The former is applied by disabling the clock system of specific components of the system, therefore, saving dynamic power; the latter, turns the component completely off, saving both static and dynamic power. Due to its simplicity, clock gating may be applied cycle-by-cycle, but it may create an overhead on the critical path of the system depending on the design [10] [11]. It is usually applied by CAD tools, but the gains of applying it to larger modules are much higher [12], as presented on [13] [14].

On the other hand, power gating has timing and energy costs for turning the module on or off. The cost for reaching the break-even point (i.e., the point in which the energy spent to turn the module on is compensated by the energy savings of using that technique) is of 10 cycles, for typical technology parameters [15]. In addition, there are techniques that exploit both clock and power gating, applying each of these techniques when they are more suitable [10].

D. Proposed approach

This work will evaluate the impact of both fine and coarse-grained approaches on performance and energy consumption of VLIW processors, by considering that certain parts of the hardware are turned off through clock/power gating. We modify different design variables of the VLIW processor and evaluate their impact on system metrics like IPC, energy and area.

III. Potential of Optimization

In this section, different VLIW configurations will be considered in order to assess the potential optimization that can be achieved.

A. Methodology

The processor that was selected for this analysis was the ρ-VEX, which is a configurable processor implemented in VHDL [16]. The processor architecture is based on the VEX instruction set architecture.

The ρ-VEX core has a five-stage pipeline, and it can be configured at design time to have different number of issue slots (e.g., 2, 4, or 8). Each operation is encoded as a syllable and the number of syllables per instruction word is defined by the number of issue slots. The pipeline's fetch stage is responsible for retrieving the instruction word from memory and distributing one syllable for each issue slot. The other pipeline stages are not shared by the issue slots, which are: decode, execution 0, execution 1, and write-back. The execution 1 stage performs access to the data memory or executes instructions that need more than one cycle to be computed (e.g., multiply instruction).

Each issue slot may contain different functional units from the following set: Arithmetic Logic Unit (ALU) (always present), multiplier, memory, and branch units. In addition, other parameters can be changed such as the register file and memory size.

The ρ-VEX design organization used in this work was the following: issue-width from one to eight, register file of 64 registers, ALUs in all issue slots, one memory and one branch unit (due to ρ-VEX's design restrictions), and a number of multipliers that vary from one to four according to the issue-width. This configuration is similar to other VLIW processors (e.g., Intel Itanium [17]).

The synthesis to obtain the power dissipation and area was carried out using an 180nm library from X-FAB [18] and Encounter RTL Compiler from Cadence Tools [19]. The module synthesized was the ρ-VEX core, without any peripheral or memory attached. The programs used to evaluate performance were compiled with the VEX compiler from HP labs, and the total number of execution cycles was measured via hardware counters.

B. Results

Fig. 1 depicts the area between different issue-widths, varying from 1- to 8-issue. The 8-issue has 10.5 times more area than the simplest configuration (1-issue), and 2.3 times more than the 4-issue, due to the instantiation of more functional units and more read/write ports in the register file. This increase in area also leads to an increase in the core's power dissipation, which is presented in Fig. 2. The 8-issue dissipates 2.1 times more power than the 4-issue and 6.86 times more power than the single-issue.

In Fig. 3, the performance for five applications is compared as we change the issue-width of the processor, and the speedup is calculated taking the 4-issue configuration as the baseline. The following applications were considered: *ADPCM*, *CJPEG*, *DFT*, *Matrix multiplication* and *Itver2*. The 8-issue is always faster than the 4-issue for these benchmarks, varying from 0.5% (*ADPCM*) to 23% (*CJPEG*), with an average speedup of 10%. On the other hand, the 2-issue is always slower (values below one), ranging from 22% (*DFT*) to 65% (*Itver2*) of slowdown, with an average slowdown of 44%.

The difference in performance between the 4-issue and 2-issue processors is more remarkable than between the 4- and 8-issue versions, because of the limited parallelism that the compiler can exploit from the source code. Since the requirement for parallelizing a set of operations is that all operations must be executed simultaneously without any data dependencies between them, increasing the issue-width requires a larger group of independent operations. For instance, a 2-issue processor only needs to find 1 relationship in which the data from the two instructions (2-issue) are not dependent from each other, while a 4-issue processor needs to find 6 independent relationships (instruction 1 must be independent from 2, instruction 1 from 3, instruction 1 from 4, instruction 2 from 3, instruction 2 from 4, and instruction 3 from 4). Using the same reasoning, an 8-issue processor needs to find 28 independent relationships to use all the available slots. As can be seen this increase is not linear in relation with the issue-width and therefore it is more difficult to exploit ILP effectively for larger values.

Fig. 4 presents the Energy-Delay Product (EDP) ratio, having the 4-issue as the baseline, for the same set of applications. With the EDP is possible to evaluate the trade-off between energy consumption and performance. The best EDP is obtained when executing the application on the 2-issue in almost all benchmarks (up to 71% lower), with the exception of the Itver2 application, in which the 4-issue presents better EDP. The 8-issue has higher EDP (ratio below one) on all applications when compared to the other configurations. Therefore, the goal is to have the performance of the 8-issue with the energy consumption of a simpler design, e.g., 2- or 4-issue. This can be achieved by disabling parts of the hardware that are idle in a given moment, consequently, reducing the energy consumption and not affecting the performance.

IV. Dynamic adaptation

The aforementioned results highlight the enormous potential that an exploration of the design space could produce in terms of energy savings if microarchitectural adaptation was available at run-time. For instance, if one part of a program does not use certain issue slots, it is not necessary that they remain active during this portion of time. Instead, they could be disabled through a variety of techniques (clock gating, power gating, etc.) to avoid unnecessary energy consumption. In order to evaluate the potential gains from using these techniques, we will consider that switching for enabling or disabling the hardware is done with zero delay.

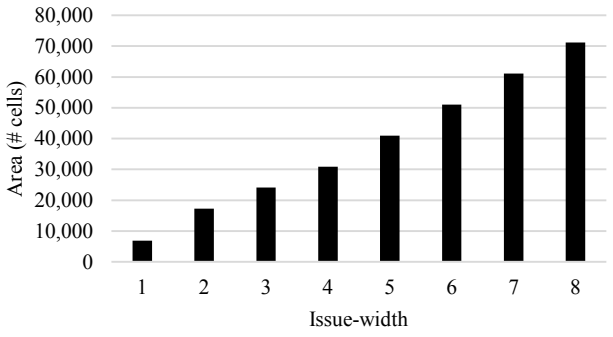


Fig. 1. Area comparison between different issue-widths

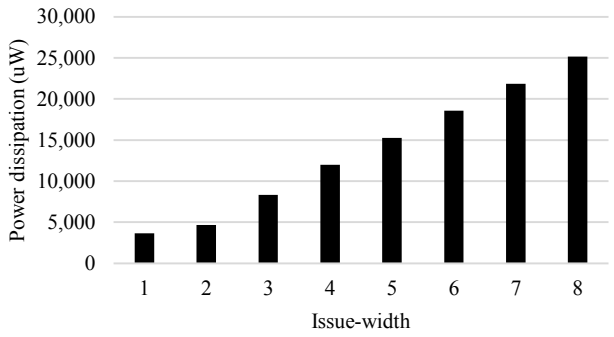


Fig. 2. Power dissipation between different issue-widths

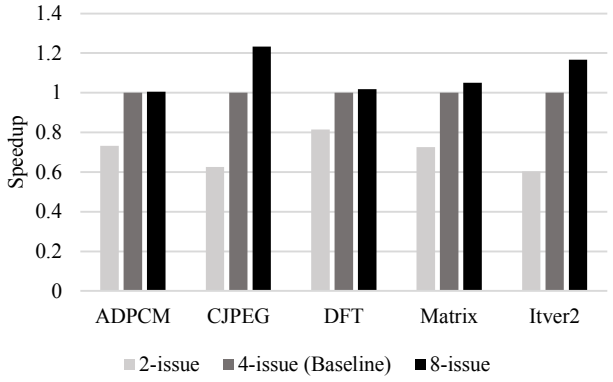


Fig. 3. Speedup compared to the 4-issue VLIW

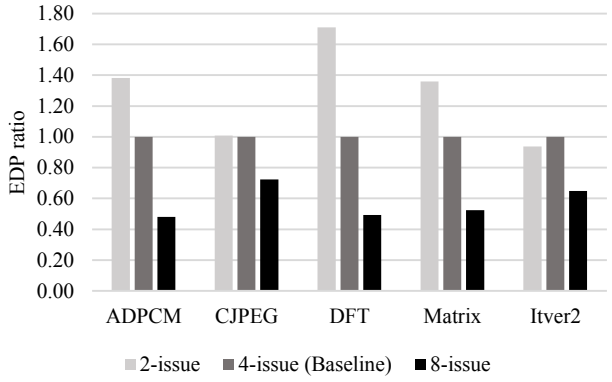


Fig. 4. EDP ratio for different applications

Taking this as our guideline, we use architectural simulation to dynamically evaluate the IPC, which reflects the utilization of the functional units along the execution time. If the processor is using a high number of FUs at one specific moment, it will result in high IPC values, as more instruction parallelism could be explored. On the other hand, when it comes to a specific phase of a program, where the IPC is significantly below the number of available functional units, valuable energy in resources that are not actually being used would be wasted. For example, if a program is running on an 8-issue width processor and the IPC for a phase is 1.5, it means that most of the FUs are idle during great part of the execution.

A. Methodology

The HP's VEX simulator [20] was modified to obtain the IPC at run-time, extracting the number of issues used by each instruction word. The VEX simulator works by translating an already compiled target binary (in our case a binary generated by the VEX compiler) to a C program. Using this generated C program, the host's C compiler is used to create an executable compatible with the host instruction set architecture. Finally, the application's execution on the VEX architecture is simulated. In addition, the simulator produces instrumentation code that is used to count the execution cycles and other statistics about the application. More details about the VEX simulator are available in [21].

Moreover, the simulator was modified to calculate the average IPC for specific intervals according to two implemented methodologies, which differ in the way they handle the instruction window sizes for phase measurement. They are called coarse-grained and fine-grained approaches, and will be discussed in the next sub-sections.

The programs used were extracted from Mibench, which is a free, commercially representative embedded benchmark suite [22]. They were compiled using VEX compiler for the 8-issue configuration. It was selected a number of 10 applications, due to the restriction on the availability of libraries from VEX compiler. The selected programs were *Basicmath*, *Bitcount*, *Qsort*, *Djikstra*, *Sha*, *CRC*, *StringSearch*, *ADPCM*, *Susan*, and *FFT*.

B. Coarse-grained approach

This method aims to visualize the big picture of IPC dynamics for program behavior. For that, the total execution time of each application was divided into intervals with the same number of cycles; and the average IPC value for each one of these intervals was calculated. Since some applications are larger than others, the same length of time interval for all benchmarks would not reflect their particularities. Therefore, it was established a granularity of 5% of total execution time for each benchmark (e.g. if one program is composed of 1000 instruction words, the length of each time interval would be of 50 instructions).

The dotted line in Fig. 5a, Fig. 5c, and Fig. 5e (the gray background will be explained in the next section) shows the results obtained with this methodology. Three different benchmarks are shown: *Basicmath*, *StringSearch,* and *sha*, which illustrate different and representative behaviors. *Basicmath* shows an evident phase behavior, being primarily composed of two stable phases. *StringSearch* is stable and does not present changes on the IPC that suggests any transition phase. Finally, *sha* has an IPC that changes drastically between intervals.

C. Fine-grained approach

This approach uses the basic block as the basic grain unit, so the IPC measurement is applied for each one of them. The Fig. 5b, Fig. 5d, and Fig. 5f show the results using this granularity. The three benchmarks shown (*Basicmath*, *StringSearch,* and *sha*) demonstrate three different behaviors: presence of phases, stable behavior, and erratic behavior. However, the fine-grained approach highlights the differences of IPC between adjacent basic blocks which allows us to observe IPC changes with a higher level of detail than the coarse-grained approach.

D. Coarse vs. fine-grained approaches

As can be observed from the data obtained with coarse- and fine-grained approaches, each application exhibits completely different behaviors, in terms of average IPC, number of phases and even the presence or absence of them. For example, a program like *sha* shows a wide range of variation between values whereas *StringSearch* presents a stable behavior that is not affected by time on a large scale.

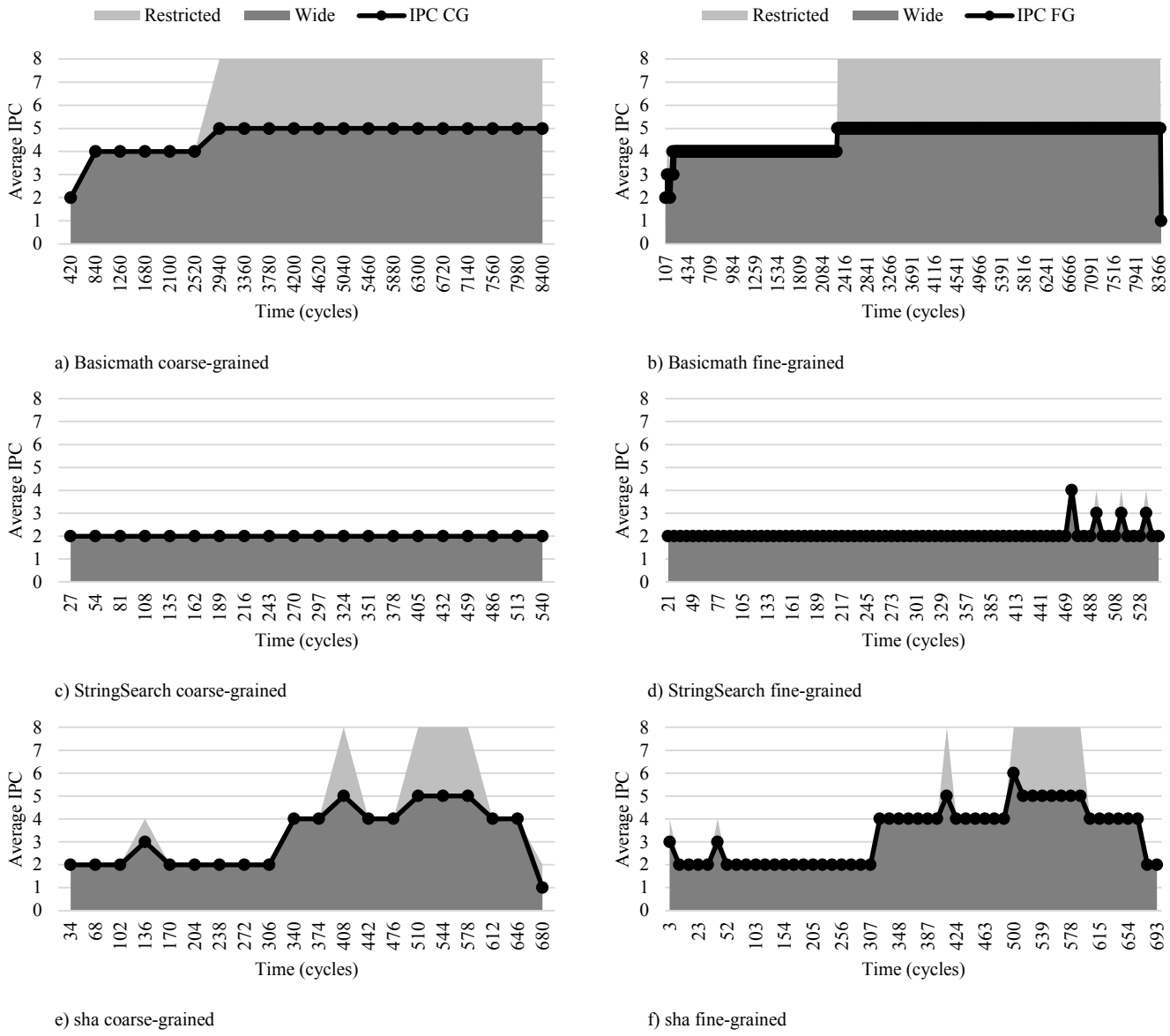


Fig. 5. Average IPC during the execution

The first approach aims to give an outlook of the dynamics of the program by averaging IPC along a big number of instructions, while the second produces higher precision in terms of IPC since the window sizes are smaller. So, for example, *StringSearch* presents different behavior comparing both techniques. Using the fine-grained approach, in the last part of the execution time, the number of execution units would be adjusted to 3- or 4-issue, whereas with coarse-grained approach, this variation of IPC measurement would not be detected and only be set to an averaged value.

In an adaptive processor, the measurement of IPC through the coarse-grained approach has the advantage of requiring a simpler implementation. The system could measure IPC only in some intervals through sampling of the execution time. The hardware structures that are needed for this task are simple hardware counters and storage to save the last IPC measurements.

On the other hand, the fine-grained approach demands more resources but it could allow better granularity optimization. In a hardware implementation, it is necessary a memory structure to save the last basic blocks visited. This means that after each new basic block is processed, its IPC must be saved. The most important advantage of this approach is that when the processor is fetching an already processed basic block, its IPC will be known in advance. This kind of information is imperative if we want to allocate the right quantity of hardware resources for a given part of the code.

V. ORACLE HEURISTICS FOR DYNAMIC ISSUE-WIDTH SELECTION

It was developed an oracle experiment for choosing the best issue-width in a given moment of the application's execution, considering performance and energy consumption. It is based on the assumption that at any time the processor could change the computer organization from one specific issue-width to another to accomplish a global optimization policy.

Hence, the purpose of this framework is to measure the energy savings when the microarchitecture of the system is modified at run-time from one configuration to another. We used the data of IPC measurements that were obtained with both coarse and fine-grained approaches. For each interval, the oracle chooses the issue-width that minimizes the energy consumption without incurring big performance losses. For that, it is selected the nearest integer to the current IPC. For instance, if the IPC for an interval is 2.7, it is chosen a 3-issue width processor for this interval.

The data on power dissipation for each issue configuration was presented in Fig. 2 and for each granularity (fine and coarse), two scenarios are considered as follows. The first is called restricted adaptation, in which the number of issue slots can be modified between 2, 4, and 8. The second, called wide adaptation, is able to adapt the issue width from 1 to 8 (1, 2, 3,…8). For example, if the IPC is calculated to be 5.4, the first approach will choose an 8-issue processor whereas the second one will use a 6-issue processor.

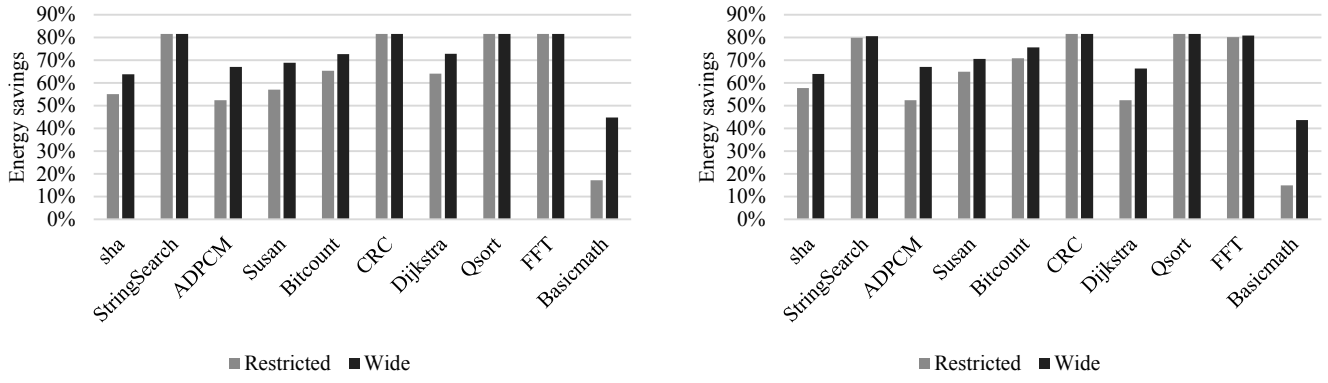
Fig. 6 depicts the energy savings that can be obtained by applying the restricted and wide adaptations on both fine and coarse-grained approaches when compared to the static 8-issue processor. The energy consumption was estimated based on the power dissipation of each core configuration and the time that each of these configurations was active. The results derived from this procedure show that the energy savings that could be obtained via an adaptation of issues could be as high as 81.5%. This means that one processor that could dynamically enable and disable its available execution units would consume only a fifth part of the total energy consumption of an 8-issue processor.

Let us assess Fig. 5 again, now focusing on the gray background: light gray is for when the restricted approach is used, while dark gray is for when the wide on is employed. Note that the restricted will always choose an issue-width equal or larger than the wide adaptation for a given phase, because the former can only choose between three distinct issue-widths, all of which the wide approach is also able to choose. That is, for phases that have an average IPC of 2, 4, or 8 (i.e., the values that the restricted adaptation is able to choose), the wide adaptation (that can choose from 1- to 8-issue) will choose the same issue-width as the restricted, having the same energy savings for that given phase. On the other hand, applications such as *Basicmath* present up to 28.8% of difference between the wide and restricted adaptations, because there is a large part of the application in which the average IPC of the phase is 5. Therefore, the wide adaptation would choose six issue slots, while the restricted would choose eight issue slots, as depicted in Fig. 5a. The reduction obtained with the wide-adaptation is higher because the processor can better adapt to the behavior of the application. On average, the wide adaptation is able to save 71% of energy and the restricted 63%.

By using a finer grain, the processor adapts itself faster to changes in the application's behavior. On the one hand, this may decrease the energy consumption as the issue-width will be changed faster when the application reaches a phase with low ILP. On the other hand, it also may choose a higher issue-width that would not be detected on the coarse granularity, resulting in more energy consumption. Therefore, on average, both fine and coarse-grained approaches achieve similar energy savings because each granularity can consume less or more energy than the other in specific moments of the application's execution.

VI. CONCLUSIONS AND FUTURE WORK

We first focused on evaluating the consequences of architectural decisions over metrics like area, energy, and performance, showing the big impact that these choices could produce into the design. The complexity of a processor, in terms of number of available functional units, improve the measured performance at the expense of increasing the demanded resources and, consequently, increasing the power dissipation and energy consumption. The performance comparison between applications demonstrates that each program has different



a) Coarse-grained approach

b) Fine-grained approach

Fig. 6. Energy savings

implicit ILP, meaning that some programs could benefit more from a VLIW processor with a higher number of execution units.

Then, we investigated the effects of issue-width adaptation during run-time on performance and energy. It was noted that there are remarkable variations of ILP throughout time, which evidences the presence of phases due to the cyclic behavior of the code. The implemented oracle experiment showed that the potential energy consumption reduction between a system with adaptive issue-width and one with eight issue slots could be as high as 81.5%. The results evidence the great benefits in terms of energy savings that an adaptive architecture brings to a VLIW design.

As future work, we will develop a dynamic phase detection in order to allow the issue-width adaptation to be applied to any application without previous knowledge of its behavior. In addition, we will implement a mechanism to perform the trade-off between performance and energy consumption in a given program phase, according to the system and user needs.

REFERENCES

[1] A. C. S. Beck, C. A. L. Lisbôa, and L. Carro, *Adaptable embedded systems*. Springer Science & Business Media, 2012.

[2] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," *Micro, IEEE*, vol. 23, no. 6, pp. 84–93, 2003.

[3] H. Zhong, S. A. Lieberman, and S. A. Mahlke, "Extending multicore architectures to exploit hybrid parallelism in single-thread applications," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*. IEEE, 2007, pp. 25–36.

[4] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ilp, tlp, and dlp with the polymorphous trips architecture," in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*. IEEE, 2003, pp. 422–433.

[5] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez, "Core fusion: accommodating software diversity in chip multiprocessors," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 186–197.

[6] T. Sherwood and B. Calder, "Time varying behavior of programs," UC San Diego, Tech. Rep., 1999.

[7] M. C. Huang, J. Renau, and J. Torrellas, "Positional adaptation of processors: application to energy reduction," in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*. IEEE, 2003, pp. 157–168.

[8] B. Calder, T. Sherwood, E. Perelman, and G. Hamerly, "Method and apparatus for identifying similar regions of a program's execution," Sep. 21 2010, uS Patent 7,802,236.

[9] C.-H. Chang, P. Liu, and J.-J. Wu, "Sampling-based phase classification and prediction for multi-threaded program execution on multi-core architectures," in *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013, pp. 349–358.

[10] L. Bolzani, A. Calimera, A. Macii, E. Macii, and M. Poncino, "Enabling concurrent clock and power gating in an industrial design flow," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 334–339.

[11] F. Emnett and M. Biegel, "Power reduction through rtl clock gating," *SNUG, San Jose*, 2000.

[12] S. Kaxiras and M. Martonosi, "Computer architecture techniques for power-efficiency," *Synthesis Lectures on Computer Architecture*, vol. 3, no. 1, pp. 1–207, 2008.

[13] H. Li, S. Bhunia, Y. Chen, K. Roy, and T. Vijaykumar, "Dcg: deterministic clock-gating for low-power microprocessor design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 3, pp. 245–254, 2004.

[14] N. Mohyuddin, K. Patel, and M. Pedram, "Deterministic clock gating to eliminate wasteful activity due to wrong-path instructions in out-of-order superscalar processors," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*. IEEE, 2009, pp. 166–172.

[15] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proceedings of the 2004 international symposium on Low power electronics and design*. ACM, 2004, pp. 32–37.

[16] S. Wong, T. Van As, and G. Brown, "ρ-vex: A reconfigurable and extensible softcore vliw processor," in *ICECE Technology, 2008. FPT 2008. International Conference on*. IEEE, 2008, pp. 369–372.

[17] H. Sharangpani and K. Arora, "Itanium processor microarchitecture," *Micro, IEEE*, vol. 20, no. 5, pp. 24–43, 2000.

[18] 0.18 micron modular cmos technology. X-FAB. [Online]. Available: http://www.xfab.com/technology/cmos/018-um-xc018/

[19] Encounter rtl compiler. Cadence. [Online]. Available: http://-www.cadence.com/products/ld/rtl_compiler

[20] Vex toolchain. Hewlett-Packard Laboratories. [Online]. Available: http://www.hpl.hp.com/downloads/vex/

[21] J. A. Fisher, P. Faraboschi, and C. Young, *Embedded computing: a VLIW approach to architecture, compilers and tools*. Elsevier, 2005.

[22] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.