UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

JUAN SEBASTIAN PIEDRAHITA GIRALDO

Adaptable VLIW Microprocessor for Energy Efficiency

Thesis presented in partial fulfillment of the requirements for the degree of Master in Microeletronics

Advisor: Prof. Dr. Antonio Carlos Schneider Beck

Porto Alegre
2016

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Piedrahita Giraldo, Juan Sebastian

Adaptable VLIW Microprocessor for Energy Efficiency / Juan Sebastian Piedrahita Giraldo – 2016.

90 f.:il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2014. Advisor: Antonio Carlos Schneider Beck.

1. Introduction. 2. Adaptable Computer Architectures for Energy Efficiency. 3. VLIW Design. 4. Evaluation of Energy Savings on a VLIW Processor through Dynamic Issue-width Adaptation. 5. Time-based Power Gating for VLIW Processors. 6. Leveraging Compiler Support on VLIW Processors for Efficient Power Gating. 7. Hardware-Software Power Gating Comparison. 8. Conclusions.

“The real danger is not that computers will begin to think like men,
but that men will begin to think like computers.”
– Sydney Harris

ACKNOWLEDGMENTS

This work would not be possible without the help of many people that provide me with advices, guidance, ideas, love and many reasons to wake up each morning. I am very grateful to my advisor Prof. Antonio Carlos Beck for always helping me with his invaluable experience and for teaching me all the required skills to become a better researcher. Likewise, thank you to all the people from the Embedded Systems Lab, which gave me feedback, recommendations and excellent ideas.

To my family, which is always in my heart and motivates every single step I take.

To Thati, for being a source of endless love and unconditional support. For filling me with countless smiles.

To my friends in Porto Alegre for the unforgettable moments and long laughs. Mauricio, Nathalia, David, Jose, Luis, Sammy, Franky.

And to Brazil, for being the country that made me happier.

Microprocessador VLIW Adaptável para Eficiência Energética

RESUMO

O consumo de energia tem sido uma variável cada vez mais importante nos projetos de implementação de microprocessadores nas últimas décadas. A arquitetura VLIW é um exemplo representativo desta tendência, devido ao seu design simples e desempenho competitivo, resultado da exploração do paralelismo entre instruções (ILP) em tempo de compilação. Neste trabalho, é realizada uma análise da economia de energia obtida através da adaptação da microarquitetura dos processadores VLIW de acordo com as diferentes fases dos programas executados. Primeiramente, o potencial de otimização é abordado, através da execução de um grupo de benchmarks no processador configurável ρ-vex, e estudando o impacto da largura do processador (i.e.: número de *issues*) na performance, consumo de energia, e área. A partir desta informação, um experimento levando em conta o caso ótimo (usando um oráculo) foi realizado com o objetivo de variar dinamicamente a largura do processador de acordo com a fase do programa, considerando duas granularidades diferentes. A economia de energia usando este tipo de adaptação pode ser de até 81,5% comparado com uma versão estática do mesmo processador executando o grupo de benchmarks MiBench. Com base nestes resultados, duas técnicas de *power gating* nas unidades funcionais são propostas. A primeira é baseada em lógica adicional, inserida no processador, para controlar os circuitos de *power gating* associados com cada unidade funcional. Mostra-se que estas unidades podem ser desabilitadas em até 63% do tempo de execução para os multiplicadores e 30% para as ALUs, com um custo em performance de 13%, em média. A segunda técnica proposta propõe uma técnica para ser usada em conjunto com o compilador para aplicar *power gating* nas unidades funcionais, assim como nos blocos do banco de registradores. Esta operação é realizada inserindo instruções específicas em tempo de compilação, tendo em conta a análise das probabilidades de instruções de saltos e informação dos blocos básicos, obtidos através de instrumentação de código. Utilizando este tipo de estratégia, é possível economizar até 20% em energia com perda marginal de desempenho.

Palavras chaves: VLIW. Processador Adaptativo. Consumo de energía.

Adaptable VLIW Microprocessor for energy efficiency

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, due to their static ILP exploitation. In this work, we study the energy savings that could be obtained by adapting such microarchitecture according to the current program phase. First we analyze the potential of optimization, by executing a set of benchmarks on the ρ-vex configurable softcore VLIW processor, and by modifying the number of issues. With this data in hand, we develop an oracle experiment to dynamically vary the issue width of the processor according to the phase behavior, considering two different phase granularities. The potential energy savings using this policy could be as high as 81.5% when compared with the static version, executing the MiBench set. Taking into account this information, two techniques for power gating the functional units are proposed. The first approach is based on additional hardware logic to control the power gating circuitry of each Functional Unit. Our results show that these units can be put to sleep on average 63% of the execution cycles for the multipliers and 30% for the ALUs, at a performance loss of 13%. The second approach handles intelligent use of the compiler for power gating the Functional Units as well as blocks of the Register File. We do so by inserting customized instructions at compile time, based on the analysis that involves probabilities of conditional branches and basic block information obtained via dynamic profiling. By using this technique, it is possible to save up of 20% in the total energy consumption with marginal losses in performance.

Keywords: VLIW. Adaptive processor. Energy consumption

LIST OF FIGURES

LIST OF ACRONYMS

CMOS	Complementary Metal Oxide Semiconductor
VLIW	Very Long Instruction Word
VEX	VLIW Example
VHDL	VHSIC Hardware Description Language
IPC	Instructions Per Cycle
ALU	Arithmetic Logic Unit
CFG	Control Flow Graph
ILP	Instruction Level Parallelism
FU	Functional Unit
DVFS	Dynamic Voltage and Frequency Scaling
WCET	Worst Case Execution Time
EDP	Energy-Delay Product

INDEX

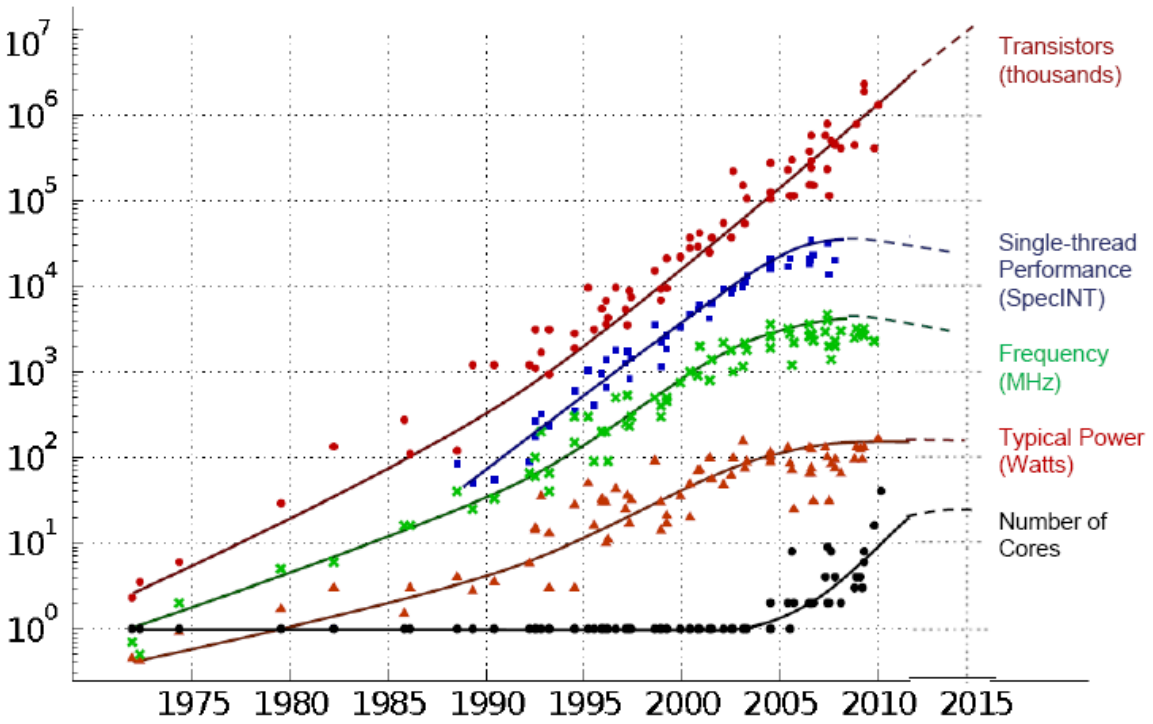
1 INTRODUCTION

1.1 Motivation

The microprocessor industry witnessed the birth of a new design paradigm in the last decades, moving away from a design approach focused on performance towards one focused on energy efficiency. This trend can be attributed to two major facts. The first one is the power wall that the industry faced when the raising frequency used in the new high-end processor designs produced higher values of power consumption (Figure 1.1), which could not be managed with the current packaging technology COLWELL (2013). The deep-pipelined and aggressive out-or-order designs were discouraged since the success of these strategies is profoundly linked with increasing clock rates. The second fact was the prevalence of embedded systems in the electronics market whose energy consumption is very restricted due to mobile battery constraints RAKHMATOV;VRUDHULA (2003). Thus 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 BECK; LISBÔA; CARRO (2012).

This design pressure for adaptability regarding energy savings might be found in all levels of abstraction within any modern microprocessor. In the architectural level, it can be found heterogeneous systems, which are multicore systems composed of cores with different computer organizations KUMAR et al. (2005). Each one of them presents different values of power consumption and performance, representing distinct points in the design space. In these systems, each thread is allocated for its best possible core depending on either specific performance or energy requirements, making possible improvements in energy efficiency. On the other hand, at the gate and transistor level, technological approaches like DVFS modify frequency and voltage supply depending on the current performance demands or power constraints ISCI; BUYUKTOSUNOGLU; MARTONOSI (2005).

Figura 1.1 - Thirty five years of microprocessor trend data.



Reference: MOORE (2011)

At the architectural level, VLIW design is a microarchitecture solution for the demands imposed by energy efficiency 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. However, the energy resources of a modern VLIW microprocessor are also limited and then its use must be intelligently managed to avoid any kind of energy waste. Therefore, VLIW microprocessors must also be designed to dynamically adjust the availability of resources to the computational demands.

The typical commercial VLIW processors are static and the resources that are available in the CPU are not always in tune with the demands generated by the workload. In this group of static resources, we can find the issue-width and the register file size. The issue-width influences the availability of execution units, which determines the ILP available for the compiler, whereas the register file size determines the number of registers that the compiler is able to manage. By choosing high values for these parameters, performance is likely to increase but it will also increase power consumption as a drawback. Since different

programs exploits the availability of functional units and registers at different levels, the presence of idle units in this kind of resources will most likely be a source of energy waste.

Moreover, not only different applications present varying demands 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 because of being memory bound sections. These intervals with similar behavior are defined, in this work, as phases SHERWOOD et al. (2002). If the resources are set to comprise the phase with the highest demand of hardware resources (in order to achieve the best performance), the execution of a phase with low computing demands will increase the energy consumption of the system since the idle resources will continuously consume energy. On the other hand, if the resources are set to the lowest demanding phase, the performance will be highly affected (and may also negatively influence energy, since it will take longer to finish execution). Therefore, for a given application, the presence of different phase behavior will very likely affect the performance and energy consumption of the system.

Since high performance is needed in many practical problems, the implementation of a great quantity of FUs in a VLIW processor is sometimes obligatory. 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 is found. This situation demands the use of effective techniques for power management of idle FUs, dynamically decreasing the amount of energy consumed by parts of the circuit that are not useful for computing purposes. In this group of techniques, we can find prior circuit-level approaches like clock gating BOLZANI et al. (2009), EMNETT; BIEGEL (2000) and power gating HU et al. (2004). While the first one only attempts to decrease the dynamic power consumption, which is a result of the clock tree switching; the second one also reduces leakage power of the logic by disabling the power lines for specific parts of the circuit. The implementation of power gating is more challenging in terms of design because of the overhead of this technique: since the turn-on and turn-off processes of the power lines of the circuitry spend additional time and energy, this penalty must be taken into account by the power gating policy. Nevertheless, leakage power has been showing an increasing impact on newer technologies CHANDRAKASAN; BRODERSEN (2012). Thus, the use of power gating for specific parts of the die could be mandatory in the next microprocessor designs to avoid excessive power dissipation.

1.2 Objectives

Having in mind the aforementioned scenario, the purpose of this thesis is to address the feasibility of an adaptable VLIW processor and propose design solutions for the implementation of its microarchitecture. We present results and measurements of the potential benefits produced by this scheme through synthesis and simulation tools.

Summarizing, this work has three main purposes:

- Describe quantitatively the impact of architectural design choices like issue-width for energy consumption, performance, and area on a VLIW processor.
- Analyze the potential energy savings that might be obtained by dynamically adapting the VLIW microarchitecture according to the program phase.
- Propose, describe and implement a feasible adaptable VLIW processor for energy efficiency; and compare the advantages and challenges of a purely hardware-based approach versus a compiler-based approach.

1.3 Thesis Organization

The rest of this work is organized as follows. Chapter II describes a summarized bibliographic review over adaptable computer architectures for energy efficiency. Chapter III describes work related to VLIW microprocessors and presents ρ-VEX, a VLIW processor that is used along this work for simulation purposes. Chapter IV shows the potential of optimization by analyzing the impact of design choices on performance and energy consumption as well as the evaluation of an oracle methodology for potential energy savings through issue-width adaptation. Chapter V and Chapter VI compares two main implementations for adaptable functional units through power gating: a hardware approach based on hardware counters (Chapter V) and a software approach based on power gating instructions (Chapter VI). Chapter VII presents a comparison in terms of results between the two previously mentioned strategies. Chapter VIII summarizes our conclusions and points out to future directions of research and development.

2 ADAPTABLE COMPUTER ARCHITECTURES FOR ENERGY EFFICIENCY

In this chapter, it is presented a survey about adaptable computer architectures for energy efficiency and the implementation challenges that are associated with this kind of design. More specifically, how we can dynamically reconfigure the behavior of modern microprocessors depending on the running application in order to minimize energy consumption. We are going to review a set of researches related with this issue and we will show the different practical problems that arise whenever an adaptable processor is implemented. The purpose of this chapter is to show the different methodologies that have been applied for the energy efficiency problem, presenting a wide range of paradigms, from Heterogeneous Systems to Reconfigurable Computing.

The rest of this chapter is organized as follows. In section 2.1 it is presented the problem of power consumption in CMOS technology as well as the technological techniques that have been developed to decrease its impact. In section 2.2 there is an overview about dynamic behavior of workloads studying the variability that applications exhibit in terms of application metrics like IPC, branch miss prediction error rates, etc. In section 2.3 the related work about adaptable architectures for energy efficiency is presented, remarking the differences between each one of the possible approaches. Finally, section 2.4 presents the conclusions that can be extracted from the previous discussion and the perspectives that can be expected from future adaptable computer architectures.

2.1 Managing Power consumption in CMOS technology

The CMOS paradigm has evolved as the primary technology used to design, implement and verify new integrated circuits. The shift from bipolar circuits to CMOS brought new advances in power consumption since the energy this kind of circuits demanded was mainly represented by switching activity whereas static consumption was almost insignificant. Paradoxically, the same reason that gave impulse to CMOS technology has been

the primary challenge in the last years. The increasing power density as well as the increasing importance of leakage power have become rough challenges for hardware engineers. Since the advent of a new technology that replaces CMOS will not be mature within a short term, the efforts have been focused on the implementation of techniques for decreasing power consumption without essentially changing the manufacturing process.

Since the 1990s, power consumption has been a primary requirement at the same level that performance and cost for any digital design. The embedded systems and mobile applications, which are restricted by battery lifetime, are two big forces that have guided the rise of power-aware computing. In desktop machines, the key constraint is thermal, so currently, high performance processors have encounter a “power wall” and their operating frequencies have been severely restricted.

Likewise, power consumption is becoming a main concern for the High Performance Computing (HPC) community, since moving from petascale to exascale demands better energy efficient systems HEMMERT (2010). For instance, the Titan supercomputer at Oak Ridge National Laboratory has 18,688 NVIDIA Tesla K20X GPUs, which demand almost 8.2 MW of power while generating 2.14 GF/W. This energy efficiency is far from the required to enable exascale systems, which would have to be higher than 50GF/W VILLA, et al. (2014). The future process technology improvements would account for about 4.3X of the required energy efficiency whereas an additional 1.9X could be derived from circuit improvements like lower VDD operating voltage. Therefore, about 2.5X of the needed enhancements would have be generated by architectural and system level design decisions which, if successful, would allow efficient scaling-up of node performance VILLA, et al. (2014).

The problem of low power techniques has been historically and mainly addressed by circuit engineers. The apparently historical unique focus on the circuit-level is understandable because of availability of CAD tools for power consumption estimation and a more precise analytical approach. However, in the recent past the importance of architecture decisions has gained interest, so the problem has been also managed at a higher level of abstraction. Often the design decisions on higher levels of abstraction produce a higher impact on system behavior in terms of performance and energy. The latest efforts in this direction have been present at the level of caches, execution units, cores, etc.

In the next subsections the CMOS power consumption sources are explained as well as the specific techniques that have been developed to deal with them.

2.1.1 Sources of Power consumption

Basically we can divide power consumption for CMOS into two big categories: dynamic and leakage power. The first is related with energy spent during switching activity intervals and the second one includes all the power consumption sources when the circuit is in idle state.

2.1.1.1 Dynamic Power

Dynamic power consumption has been the most important part of the power budget and the primary metric that has guided the progresses in power aware computing. It is caused by the switching activity that CMOS gates generate when there are state changes in any of their outputs. In that case, the capacitances that compose the circuit are charged and discharged, which consumes energy in the process.

It is given by the known equation:

$$P = CV^2Af$$

Here, C is the load capacitance, V is the voltage that acts as supply, A is the activity factor and f is the operating frequency. The details of each one of these variables are described below.

Capacitance (C): It is the capacitance that is generated by the intrinsic characteristics of the transistors and the contribution of wires and aggregated chip sub-structures. Since the capacitance due to transistor parasitics are inherent to the technology, the architecture decisions have not a big impact on this one. However, the load capacitance largely depends on the wire lengths of on-chip modules, so the variation of computer organization could increase or decrease the total power consumption. As an example, a processor composed of four simple cores would have shorter wire lengths compared with a big core since the most part of the connections is restricted to smaller regions. The same principle applies to independent banks of cache compared with a large cache since the address and data lines are only spanned in a relatively small region.

Supply Voltage (V): It corresponds to the voltage of supply lines connected to CMOS gates. Since the contribution to P is quadratic, its changes have enormous influence on the total power consumption. Historically, this variable has dropped steadily with each new technology which has made possible higher levels of integration.

Activity factor (A): This variable expresses the fraction of time that the wires are transitioning. A value of 1 would mean that the transistor outputs are switching at the same rate as the clock speed. There are many techniques, like clock gating, that aim to reduce activity factor by disabling unused logic at specific moments.

Frequency (*f*): The clock rate is a variable that is associated with operating frequency in any synchronous circuit. The supply voltage and frequency are not independent parameters, so usually they are correlated. A system that needs higher operating frequency needs higher supply voltage since the charge and discharge of capacitances is faster in that situation. For that reason, the effect of these variables (V^2f) has a cubic impact on power consumption. Techniques like DVFS (Dynamic Voltage and Frequency Scaling) have been developed to reduce this effect by modifying frequency and voltage supply depending on the current performance demands ISCI, et al. (2005).

2.1.1.2 Leakage Power

The power consumption that is produced when the transistors are not changing their current outputs is called leakage power. The amount of its contribution is not predominant when it is compared with dynamic power but the trend is to increase with each new semiconductor fabrication improvement. As the technology scales down below 100nm, the channel length decreases, which increases the amount of leakage power in the total power dissipated KUMARI et al. (2014).

Leakage power is divided into gate leakage and sub-threshold leakage. The first one is related to the amount of current that is tunneled through the transistor gate and its relative importance is increasing with technological shrinking. On the other hand, sub-threshold leakage is the most representative part of leakage power and it is caused by the non-ideality of gate dynamics. The ideal value of static drain current in a CMOS circuit is zero. However, since it behaves exponentially, there is a little contribution of current even when the voltage source is less than threshold voltage. Mathematically it is expressed as:

$$P = V\left(ke^{-qV_{th}/(ak_aT)}\right)$$

Here, V is the supply voltage, V_{th} is the threshold voltage, T is the temperature and a, q and k_a are constants that depend on fabrication processes. It is noted that leakage power is higher if Vth is lower. Since lowering threshold voltage is a necessity to achieve higher operating frequencies, it causes an increasing for power consumption.

2.1.2 Techniques for power reduction

There have been proposed many different techniques to decrease the impact that a variety of sources have over power consumption both as dynamic power and as static power. These methods span from low level techniques, like using high threshold transistors, to high

level approaches, like using multicore processors. In this section, we explore two specific methods that can be applied at the microarchitecture level and, as a result, important for the current research. They are clock gating and power gating.

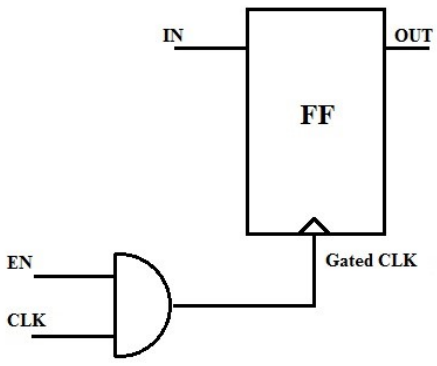
The first one attempts to reduce dynamic power caused by clock tree and the second one is focused on decreasing leakage generated by idle circuits. Each one resolves different problems so they can be applied orthogonally.

2.1.2.1 Clock Gating

One of the main methods to reduce dynamic power consumption is to disable switching activity that does not have a direct influence on the results of a computation. In the case of clock gating, the main purpose is to prune those parts of the clock tree that arrive to flip flops and latches which are not changing their outputs. Therefore, this action reduces dynamic power generated by clock switching activity. It could be applied to simple circuits, modules an even at the core level, depending on the granularity that is being addressed.

At the circuit level, the process of clock gating is depicted in Figura 2.1. When an AND gate is introduced, the flip flop input capacitance is replaced by the AND gate capacitance. Since this value is lower than the former one, the energy that is spent discharging and charging it is much smaller.

Figura 2.1 - Clock gating circuitry for a flip flop. The clock is controlled by the behavior of enable signal.



One specific technique for clock gating is Deterministic Clock Gating, which enables and disables the clock tree for execution units or different stages of a pipeline some amount of cycles in advance LI et al. (2004). So if one specific module is not used during a part of the execution, its clock signals could be pruned. The idleness of a structure must be known some

cycles in advance to reduce problems of performance drop. The enable and disable signals for clock gating are transmitted like a bubble through the pipeline.

Although the idea had been conceived a long time ago, the first real application on a superscalar pipeline was not made until 2003. By applying deterministic clock gating LI et al. (2003) disabled and enabled latches and stages of the pipeline according to the ideas presented here. They reduced power in 21% and 19%, on average, for floating point and integer SPEC2000 benchmarks, respectively.

A number of commercial processors implement some kind of clock gating to take advantage of the reduction in dynamic power without significant losses in performance. In the following paragraphs some of the most remarkable ones are presented.

Intel XScale: It is a low power processor that, besides its extensive DVFS features, implements deterministic clock gating CLARK et al. (2001). The most basic units that are controlled via clock gating are the Local Clock Buffers (LCBs), which generate clock pulses that are fed into pulse-clocked latches. Each LCB has enable signals that stops the production of those signals and consequently disables the clock tree for some parts of the processor. Each LCB must control at least five latches to avoid losses due to power overhead associated with extra-circuitry.

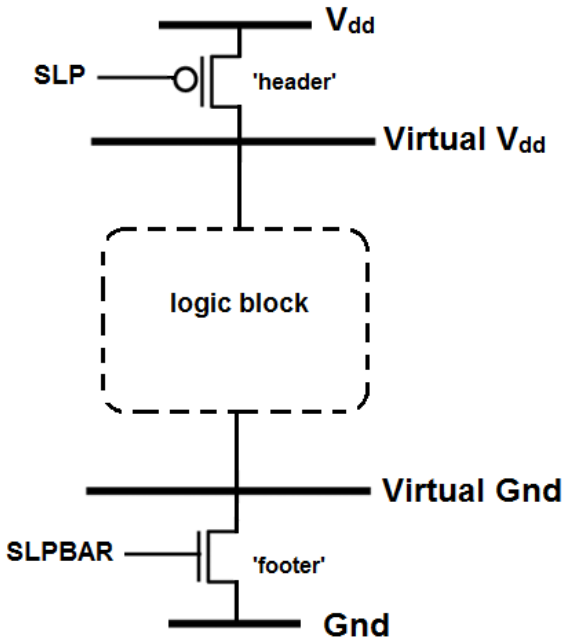
Power5: The power savings obtained via clock gating in this processor are about 25% without losses in performance CLABES et al. (2004). All the clock gating domains are programmable, so there is a big control over the dynamics of the mechanisms.

2.1.2.2 Power Gating

The main idea behind Power gating is to decrease static power by dynamically disabling the power lines of specific parts of the circuit when there is no switching activity. To achieve this objective a global policy controls the dynamics of power gating signals and, likewise, some additional and special circuits carry out the connection and disconnection of power lines.

The circuitry for each domain is composed of a header and/or footer transistor and the corresponding logic (see Figura 2.2). When a sleep signal is asserted, the sleep transistors disconnect Vdd and the virtual Vdd. Likewise, when it is de-asserted the power supply is available again.

Figura 2.2 - Power gating circuitry. The sleep signal controls the availability of power lines for each transistor.

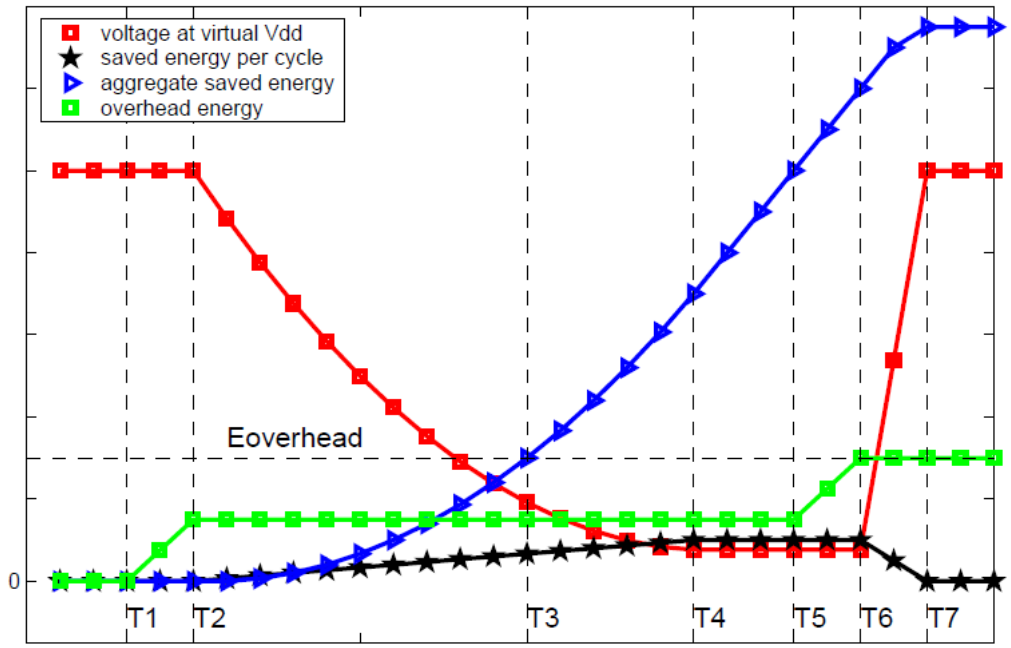


Refernce: SAHA et al. (2013)

Since the charge and discharge of power lines is not currently accomplished within one single clock cycle, there is a time penalty for each one of these processes. This kind of penalty is manifested as a potential performance loss when a specific circuit is needed for computational purposes and its functionality is not available at that moment.

The process of power gating has a set of phases that it is important to mention briefly (See Figura 2.3). The inactivity period begins at T_0, and at T_1. The Power gating control unit takes a decision and produces the signals to disable the domain. From T_1 to T_2, the signal is distributed to the header and it consumes an overhead energy of $E_{overhead1}$. At T_2 the connection between V_{dd} and virtual V_{dd} is asserted and the latest one begins to decrease. The process continues until T_4, when the virtual V_{dd} is completely discharged. Although the reduction in leakage power begins at T_2, it is only zero at T_4. At T_5, it is detected busy activity again and the signals for the header are enabled, which produces an overhead energy cost equals to $E_{overhead2}$. At T_6, the virtual V_{dd} begins its process of recharging until T_7.

Figura 2.3 - Key intervals in the power gating cycle.



Reference: HU et al. (2004)

The break-even point T_3 is defined as the point when the amount of power savings equals the amount of energy overhead that the reconfiguration demands:

$$E_{overhead} = E_{overhead1} + E_{overhead2}$$

Depending on the specific characteristics of the header, the block size, the decoupling capacitances, etc. this value could vary. There have been several researches about this topic. For instance, in DROPSHO et al. (2002) the worst-case leakage behavior relative to the dynamic energy is modeled; whereas in HU et al. (2004) an analytical model is developed for the calculation of break-even point, finding a value of 10 cycles for break-even point using some typical technological parameters.

Due to the overhead of power and performance that power gating produces, it generates more modifications at the architectural level when compared with clock gating. An intelligent policy must be implemented to avoid applying power gating in all situations and restricting its use only for those cases when the potential savings are better than the incurrent cost. Many commercial products for low power markets implement power gating through the intervention of Operating System, like the Intel Atom Processor E5xx. This kind of processors are designed with optimized power utilization, being capable of applying power gating for Video Decode (VDX), Video Encode (VED), Graphics (GFX), and Display (DSP) modules in runtime YEO et. al (2011).

Therefore, power gating has been implemented through two main approaches: by microarchitectural techniques, which use hardware logic to measure, decide and apply power gating at the circuitry level; and by software techniques, which use the knowledge about the behavior of the program to carry out power gating decisions. In the first group we can find researches based on an accurate measurement of idle periods FLAUTNER et al. (2002), KAXIRAS; HU; MARTONOSI (2001). The efforts are focused on memory resources like SRAM hardware, measuring intervals of time when some sub-modules are not currently being used and applying power gating when these periods of time are detected. HU et al. (2004) proposes the use of hardware counters to detect idle periods and apply power gating when these intervals surpass a specific threshold. Two strategies are analyzed: time based power gating and branch prediction power gating. On the other hand, RELE et al. (2002) use compiler technology to detect low ILP segments of the execution to generate power gating directives for rarely used FUs on a superscalar MIPS processor. Park et al. PARK et al. (2010) use profile information of functional units to insert ON/OFF instructions during the code execution of an out-of-order ARM processor. ROY et al. (2006) describe the insertion of sleep instructions at the entry point of specific iterative code segments by capturing the nesting loop properties of the program.

Other researches have addressed the issue in VLIW processors. UCHIDA et al. (2012) proposes a scheduling technique for fine-grained power gate on VLIW processors. LIAO; BASILE; HE (2002) use power gating through Virtual power/ground rails Clamp (VRC) and Multithreshold CMOS (MTCMOS). It is based on a microarchitecture technique that counts the active frequency of cache ways and specific components of the datapath in a given time window. This information is used to determine which idle units could be disabled via power gating. NIEDERMEIER et al. (2010) present an analysis of fine-grain power gating. The processor is partitioned into three power domains and they are controlled through control registers that determines the activation or deactivation of the different resources.

2.2 Dynamic Behavior of Workloads

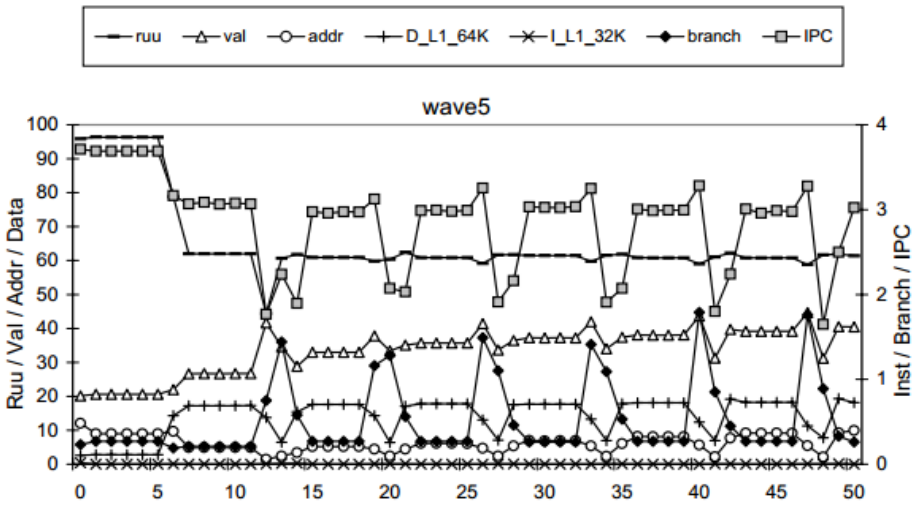
The sections of code processed by a generic microprocessor changes depending on the current state of execution. It means that one specific time segment could be focused on certain parts of the application whereas other could be rarely executed. It makes the CPU exhibits disparate behaviors in terms of performance and use of resources in different moments. This characteristic is not only evident when the applications are compared between each other but even when the analysis is carried out between specific time intervals of the same program.

Most part of the applications that are normally executed in a computing system exhibit a series of phases along time SHERWOOD et al. (2002). Each phase is characterized by a group of variables that are relatively constant for that period of time. For instance, during one part of the execution the program could be showing memory bounded behavior, in another it could be repeatedly stall on branch miss-predictions, and in others the arithmetic operations could be the most part of the processed instructions and the processor performance would be higher. One key observation which must be noted here is that since this behavior is a product of the way the processor executes different segments of the code, large scale phases could be detected just by measuring the ratio in which different parts of the program are being executed DHODAPKAR; SMITH (2003). For example, if a program is composed by two main segments of code A and B, a phase 1 could be characterized by a high proportion of executed code A and a low proportion of B whereas a phase 2 could be primarily composed by executed code B and a low proportion of code A.

Even though the existence of phases is very intuitive and evident (see Figura 2.4), the justification about when a new phase begins, and even its quantitative and qualitative definition, is not trivial. Program phases show fractal-like behavior, so large phases are composed of lower level phases and in the limit each instruction could be classified as a distinctive phase HIND; RAJAN; SWEENEY (2003). It remarks the idea that a program phase is not an absolute concept but it is only a model to classify the behavior of a program at different levels of granularity. Phase detection algorithms are not methods for direct detection of phases but for detecting changes in program behavior that are interpreted as a change in current program phase.

SHERWOOD; SAIR; CALDER (2003b) simulated a set of programs from SPEC95 benchmark and obtained measurements of IPC, branch prediction, address prediction, value prediction, cache performance and reorder buffer occupancy. It was found that programs exhibit phase and cyclic behavior in a large scale and this inherent pattern is repeated along time. The method that was used to discriminate between phases was Basic Block Vectors (BBV), which is based on saving information about each basic block of the program within each time interval. A record of all the basic blocks executed by a specific interval is saved so that a distribution of BB is obtained. If two time intervals belong to the same phase they would have a similar distribution of basic blocks use and conversely, if they are not part of the same phase, the Manhattan Distance between the two BBV would be higher.

Figura 2.4 - Time varying behavior for wave5 program from SPEC95 benchmark. The X axis is in terms of 100 million of executed instructions.



Reference: SHERWOOD et al. (2002)

Since the number of executed basic blocks within a time window of millions of instructions is very large, recording the complete information about the use of all the basic blocks is almost impractical in terms of hardware and software. For this reason it was used compression of the BBVs to reduce their high dimensionality. The results remark the great variability of the program metrics between different phases and the similarity that could be found when the measurements comparisons are done within different parts of the same phase.

In a similar work, DHODAPKAR; SMITH (2002) use the specific set of instructions that are executed within a time window to classify phases. Program changes are detected when the dissimilarity between two consecutive instruction working sets are greater than a preset threshold. Likewise, as the approach that was earlier described by SHERWOOD; SAIR; CALDER (2003b), a compression phase allows to retain only the information necessary for classification and to obtain a digital representation with a minimal number of bits.

Other strategies like HUANG; RENAU; TORRELLAS (2003) use subroutines to identify phase behavior. 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 time that is spent in each sub-routine is greater than a preset value, it is detected as a new phase.

DWARKADAS et al. (2010) implement a system that uses the amount of conditional branches to detect phase changes. One of the arguments that is used by this kind of approach is that the quantity of branches throughout a phase remains relatively constant and a major

change for this value represents a modification of the current phase. For this purpose, it is used a threshold that indicates the maximum value of difference between two adjacent time windows to be classified as belonging to the same phase. The threshold is modified dynamically throughout the execution of the program.

2.3 Adaptable Architectures to minimize Energy consumption

The idea of an adaptable computing system suitable for the specific characteristics of the workload has been developed in several previous researches. Firstly, it is presented a heterogeneous system perspective with two focuses: approaches which optimize power limited by performance constraints and others whose most important metric is performance whereas power is restricted to a budget. Secondly, it is presented the concept of reconfigurable computing and the main papers that have been related with this field. Finally, there is a review about self-adaptable microarchitecture and the challenges that this perspective presents in comparison with other techniques.

2.3.1 Heterogeneous Systems

Heterogeneous System is a design approach proposed to improve the energy efficiency on multicore chips. In this kind of systems, the workload is scheduled among a group of cores with different computer organization complexities. The objective is allocating the threads to the best core in terms of performance or energy efficiency. For instance, if an application has high thread level parallelism, the workload could be divided into a set of simple cores, taking advantage of its inherent parallel algorithm. However, if one part of the workload is strictly sequential, it is not useful using many cores since a single thread could be allocated to a more complex processor (e.g, out-of-order core). Having in mind Amdahl's Law, this decision decreases the impact that a strictly sequential segment of code produces over the total execution time.

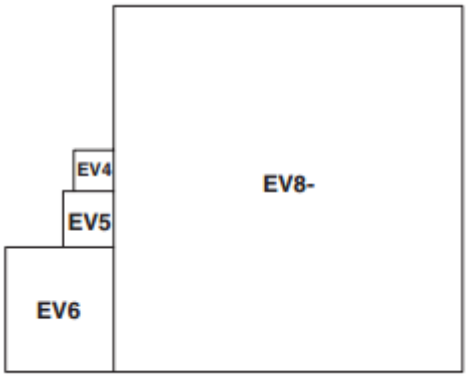
Many strategies have been proposed to adapt thread allocation policy in heterogeneous systems. In the following sub-sections it is presented some of the most relevant researches, as well as the advantages and the challenges for each approach. The related work is divided into two groups: those researches that attempt to obtain power savings according to performance constraints; and others that optimize performance constrained by a constant power budget.

2.3.2.1 Optimizing power with performance constraints

KUMAR et al. (2004) present one of the first researches focused on saving energy via dynamically scheduling in heterogeneous systems. It used a simulation of a multi-core system to evaluate how the optimal core for energy efficiency changes along execution time. It was

used a group of cores with different complexity levels, namely four Alpha cores – EV4 (Alpha 21064), EV5 (Alpha 21164), EV6 (Alpha 21264), and a single threaded version of EV8 (Alpha 21464). The physical organization is depicted in Figura 2.5.

Figura 2.5 - Relative sizes of the cores used for simulation.



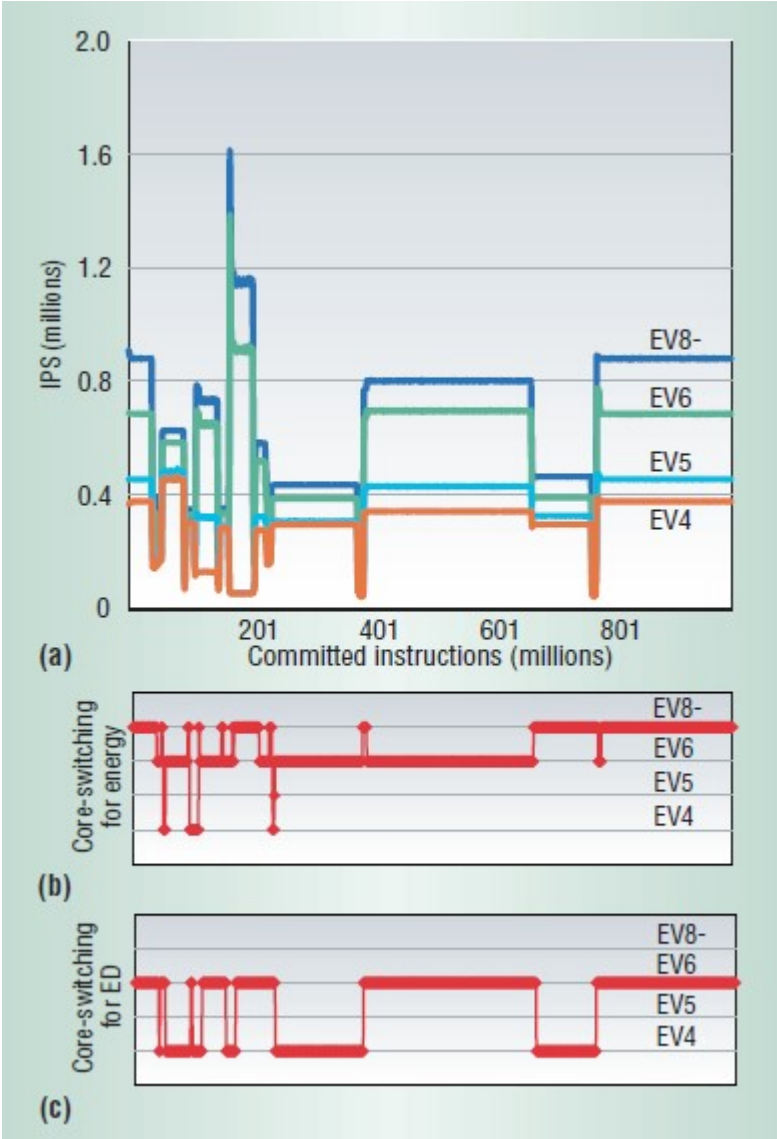
Reference: KUMAR et al. (2004)

As can be seen, the increasing complexity of the core microarchitecture is the reason behind the big difference between the area used by EV4 and EV8. The former approximately has only 10% of the total area allocated for the second one. All the cores share a single ISA which reduces problems due to binary compatibility.

In all simulations it was assumed that a single thread was running in one core at a time. Power and area were calculated from application statistics collected in real applications. By running the same thread for all cores it was found which core was the best option for each phase depending on two objective functions: energy-delay (the product of energy and delay) and energy-delay^2 (the product of energy and the square of delay).

Using the best core for each application phase and following the mentioned methodology, it was obtained a reduction about 63% and 50% for energy-delay and energy-delay^2 respectively. This oracle experiment for a specific application can be observed in Figura 2.6, where it is represented for each phase the best core in terms of energy efficiency.

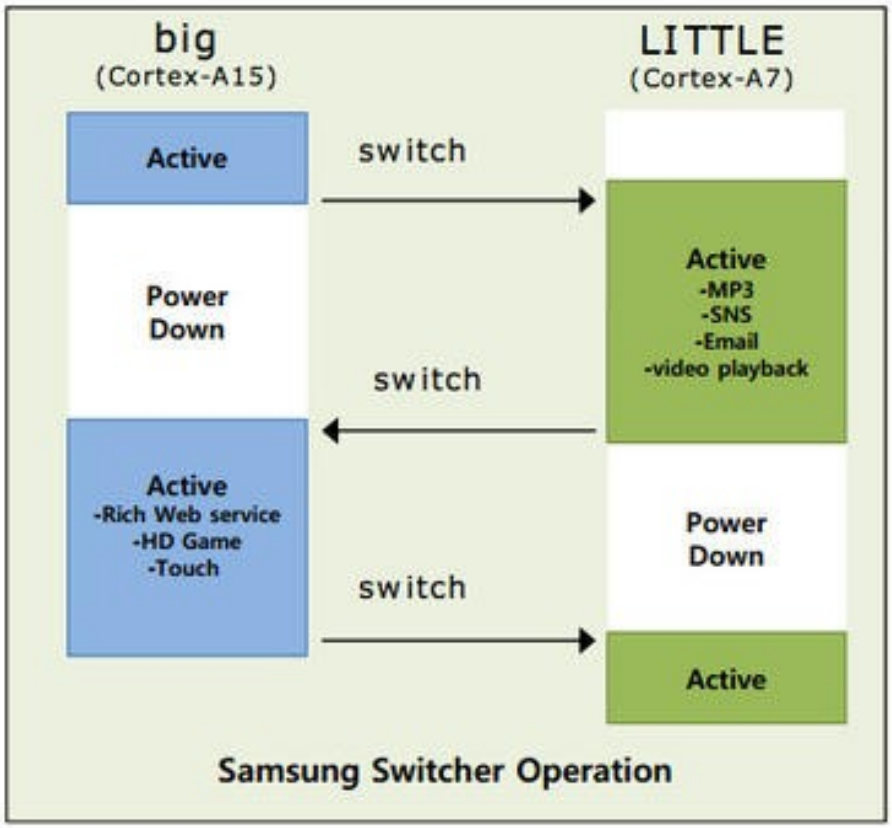
Figura 2.6 - (a) Performance of application applu on the four cores. (b) Oracle using energy metric (c) Oracle using energy-delay product.



Reference: KUMAR et al. (2004)

Recent technological and commercial devices are based on the heterogeneous processing architecture like the Big.LITTLE Technology JEFF et al. (2012). The referred system uses two types of processor: “LITTLE” processors, which are designed for maximum power efficiency; and “big” processors, which are designed to provide maximum performance (Figura 2.7). These two kinds of processors use the same instruction set, which provides coherence for their programmability. Depending on the performance required for the executed threads, each one can be allocated to a big or LITTLE core. Its use has been mainly present in the Mobile market.

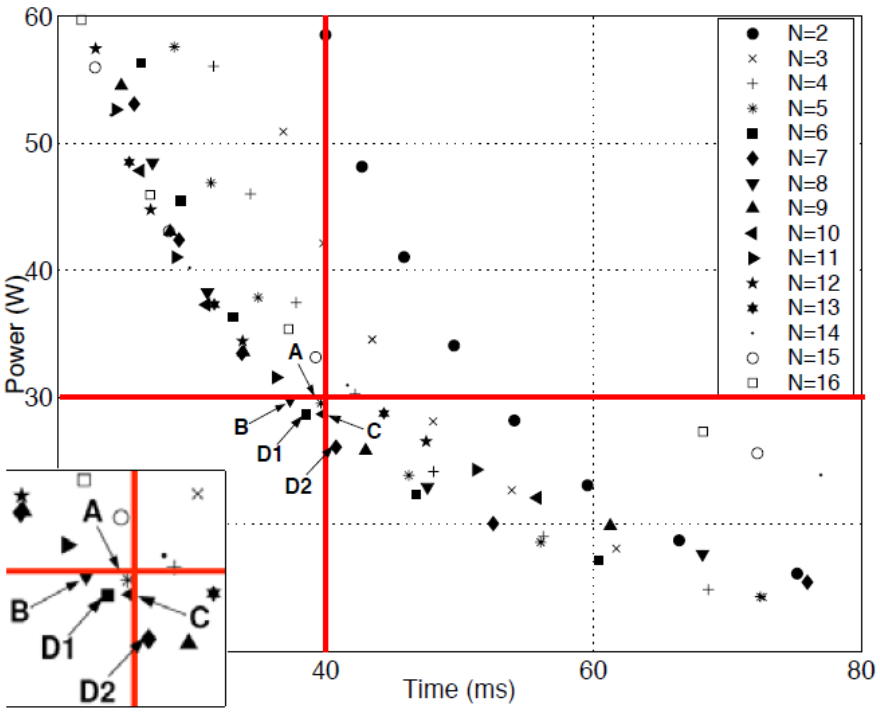
Figura 2.7 - Context Switching for big.LITTLE architecture. The demanding tasks are allocated to the more complex Cortex-A15 while the others are executed by the Cortex-A7.



Reference: JEFF et al. (2012)

LI; MARTINEZ (2006) present, in the context of optimization algorithms for minimizing power consumption in heterogeneous systems, a group of heuristics for dynamic optimization using a design space composed of two main dimensions: the number of active processors and voltage/frequency scaling. A group of benchmarks was simulated with different values for these two variables. The results of power and performance can be observed in Figura 2.8.

Figura 2.8- Power consumption and execution time of CMP configurations with varying number of processors N and voltage/frequency levels for an instance of BSOM's parallel region (a parallel data mining application). Target execution time and power are 40 ms and 30 W respectively.



Reference: LI; MARTINEZ (2006)

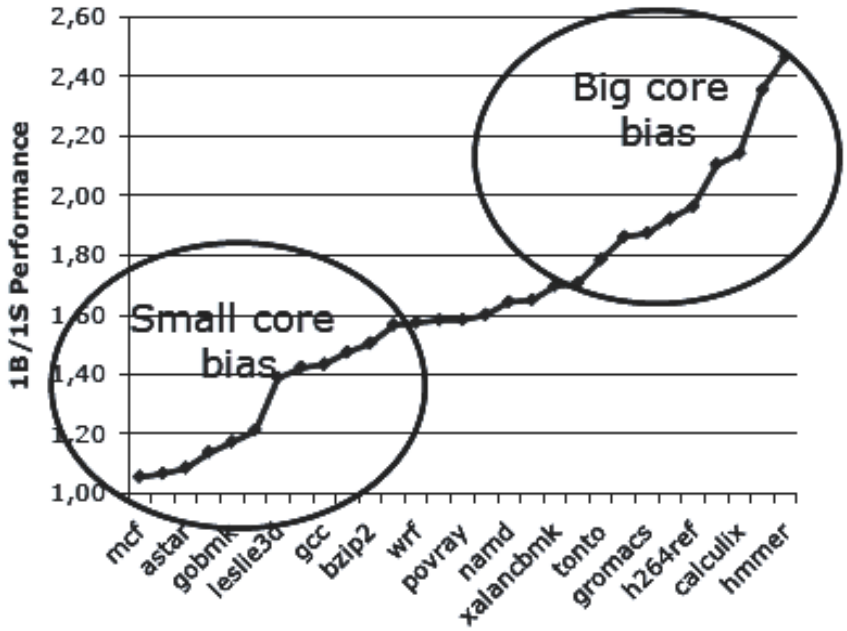
As normally, an exhaustive search is beyond the scope of any design exploration. Therefore, a hill climbing method was used to find the optimal number of processors in terms of power consumption. Taking into account the results of performance and power obtained via simulation, binary search was applied to the number of processors. A variable DVFS was set to accomplish the minimal performance target. Software support is required to schedule the applications to the right number of processors during optimization phase and steady phase.

2.3.2.2 Optimizing performance with a given power budget

ISCI et al. (2006) describe a homogeneous system, which aimed to maximize performance under a power budget by applying per-core DVFS (Dynamic Voltage and Frequency Scaling). Different policies and objective functions were used to evaluate how several heuristics impact on the overall results of power and performance. It was obtained a degradation of 1% of performance compared with an ideal oracle, constrained to a specific power budget. The concept of a global power manager is shown, which uses different frequencies and voltages for each core instead of a generic policy for all the cores. This has an enormous advantage over other approaches because it allows adjusting the power level to the performance requirements of each core, producing a better granularity management.

KOUFATY; REDDY; HAHN (2010) analyze the problem of selecting the core that best suits the resource needs of each thread in a heterogeneous system. They evaluate key metrics to find the best core, allowing the scheduler to be aware of such characteristics for a better performance. Two big contributions can be obtained from this work. First, it is based on online application monitoring without sampling performance metrics on each core or offline profiling. Secondly, the cores, unlike many other researches, exhibited different microarchitectures which allows better understanding of heterogeneous system in the context of multithreading. One of the assumptions that this work used was the correlation between stalls and bias core (Figura 2.9). Bias, in this research, is defined as the affinity of a thread to a core type. A thread has big core bias if its big/small core speedup is large and conversely, a thread has small core bias if its big/small core speedup were small. Taking into account this concept, often the applications that present a high amount of external or internal stalls have smaller performance gains when they are running on a big core, when compared with others which do not have this characteristic. Therefore, a system that measures the number of stalls was simulated along with an algorithm for estimation of bias application depending on the number of stalls and CPI. The results showed the benefits of a policy that takes into account the hardware metrics derived from applications for efficient thread scheduling.

Figura 2.9– Big/Small Core Speedup for different applications. A thread has big core bias if its big/small core speedup is large. Conversely a thread has small core bias if its big/small speedup is small



Reference: KOUFATY; REDDY; HAHN (2010)

GHIASI; GRUNWALD (2003) present the use of an asymmetric multicore approach to deal with thermal emergencies. The management is accomplished through the use of multiple operation frequencies for the different units, so the general policy attempts to optimize performance taking into account the constraints that impose the exceptions generated by abnormal temperature behavior.

ANNAVARAM; GROCHOWSKI; SHEN (2005) implement a real design of an asymmetric multiprocessor with multithreading applications. Both static and dynamic environments are simulated, and it is measured the performance gain under fixed power budgets. This is evaluated on a physical 4-way Xeon SMP Server and using a big group of multi-threaded benchmark programs. It was found 38% wall clock speedup for the AMP (Asymmetric multiprocessor) compared to a standard SMP (Symmetric Multiprocessor) that uses the same power.

Although the benefits that heterogeneity brings to microprocessor design are clear, the implementation of real systems face new challenges in terms of software development which are not present in homogeneous systems.

Usually software engineers develop code with the assumption that all cores have the same performance and capabilities, but this is not true when a heterogeneous multicore system is being used. This implies that if software design is not aware of microarchitecture features, maybe there will be problems for scalability and predictability.

BALAKRISHNAN et al. (2005) show a work about these issues and how they could be managed to take advantage, in terms of performance and energy efficiency, of heterogeneous systems without generating new problems due to asymmetry. It was used a 4-way 2.8 GHz Intel Xeon multiprocessor (Shasta series), hyper threading was disabled in all processors and heterogeneity was achieved via DVFS for each core. Running a set of SPEC benchmarks, the conclusions that were found from this work can be summarized as:

- Asymmetry affects predictability when there are shared memory resources for a specific application.
- With some workloads, the problems that arise with asymmetry could be mitigated with an operating system aware of heterogeneity. In other applications, it is necessary the application itself to be aware of the available microarchitecture features.
- The sequential performance of a heterogeneous system with a fast core is better than a homogeneous one, as can be expected from previous discussions.

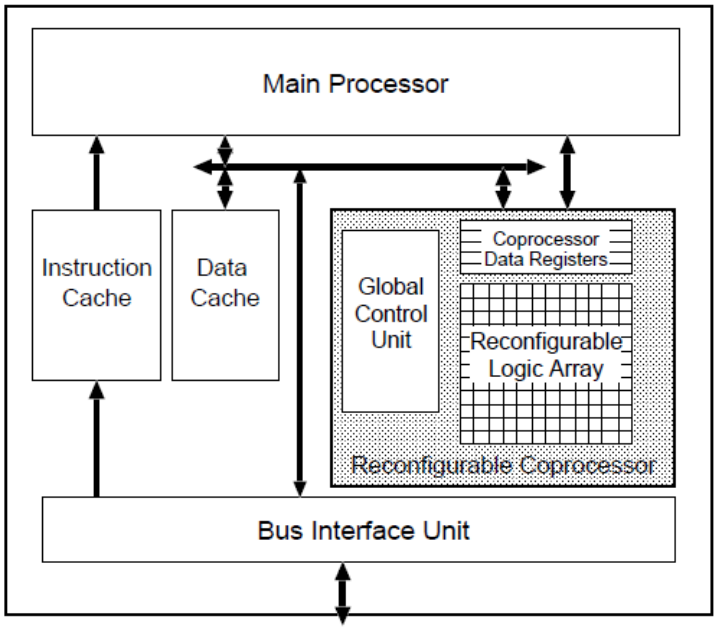
Finally, it is noted the big importance of the development of mechanisms for exchanging information between hardware and software counterparts for good predictability and scalability results. New tools and interfaces need to be developed to expose the operating system and the scheduler to the varying features of the cores that are part of a heterogeneous system.

2.3.2 Reconfigurable Computing

Reconfigurable Computing is an approach aimed to adapt the computer organization of the system depending on the varying behavior of the workload. The computer organization is modified to exploit application particularities, improving the performance of specific code segments or modifying the availability of computing resources for energy efficiency CARDOSO; DINIZ; WEINHARDT (2010). In this kind of solution, reconfigurable hardware can be modified on-the-fly to generate specific hardware structures. For example, some parts of a program could require only 12-bit fixed point precision arithmetic, while others could demand the use of 32-bit precision for parallel computation of Fast Fourier Transform. Since the most part of the computing systems do not have data structures optimized for each one of these tasks, the overall efficiency (in terms of performance and energy) might be boosted if the regular data flow were configured into dedicated hardware structures. Furthermore, the rapid emergence of FPGAs for computing purposes has multiplied the opportunities for the application of reconfigurable computing into many commercial and scientific domains.

Some researchers have been based on the implementation of a main processor that normally executes the instruction set and a coupled coprocessor which is reconfigured for acceleration purposes (Figura 2.10). For example, LYSECKY; STITT; VAHID (2004) present the implementation of a WARP processor that uses a Field Programmable Gate Array (FPGA) to improve the performance of specific parts of the running code. It is completely transparent for the programmer because there is no need for a special compiler for the FPGA unit. While not all the benchmarks could take advantage of this technique, it significantly improves performance for some applications. Many others projects, can be classified as coarse grained architectures – e.g. Montium Tile Processor SMIT et al. (2004)– which are commonly used to improve filter algorithms for communication and multimedia applications with little control flow, and fine-grained approaches – e.g. Chimaera YE et al. (2000) – which are dominated by control statements. FL; CARRO (2010) address a wider discussion of these kinds of architectures and the implemented challenges associated.

Figura 2.10 - Block diagram of a microprocessor with reconfigurable coprocessor.



Reference: MIYAMORI; OLUKOTUN (1998)

2.3.3 Challenges and Other Architectures

Three important aspects must be taken into account when such adaptable microarchitectures are proposed. First, the metric which is subject to optimization through adaptability depends on the specific system. For example, some processing units could be power constrained so an adaptability solution could be focused on this metric to accomplish better results. On the other hand, systems which have higher performance objectives could use microarchitecture adaptability to enhance performance depending on the workload. Researches like KEDZIERSKI et al. (2010) trade-off between power and performance by dynamically partitioning a shared cache among threads based on the phase behavior of the program. It dynamically reverts back to a performance centric cache partitioning scheme if the power savings are not possible. The results show energy savings around 51.5% in a L2 cache, which corresponds to 11.5% of the total energy of the processor.

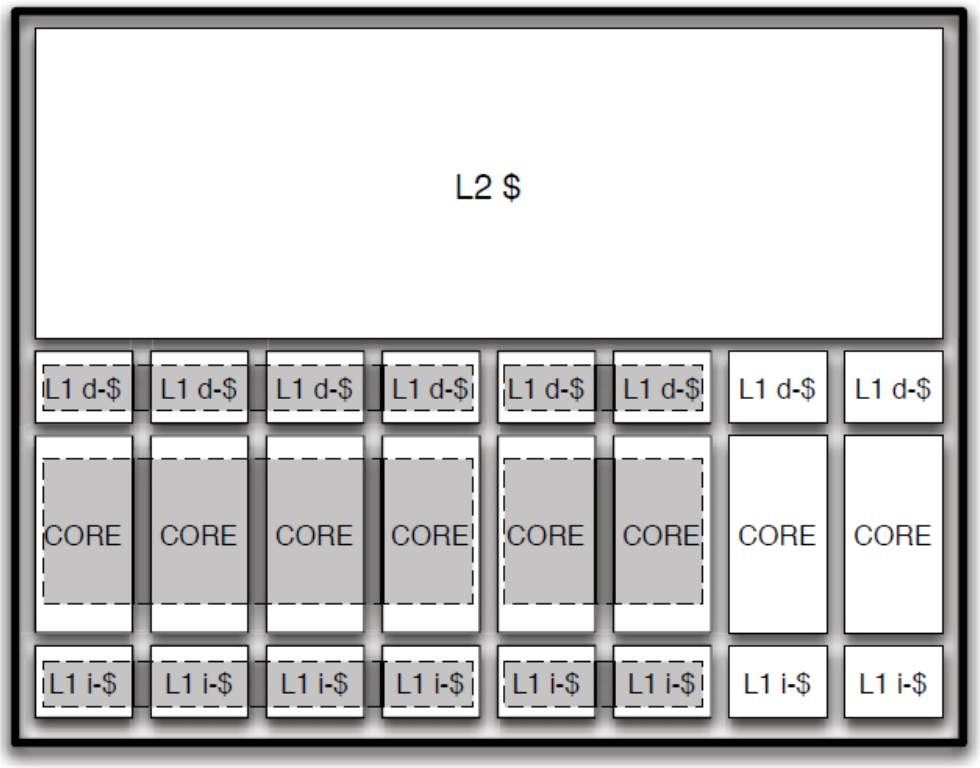
Secondly, the frequency of microarchitecture adaptation is essential for the results and savings obtained. Every reconfiguration has an overhead in terms of performance and power since it involves a physical modification of the system, therefore the reconfiguration rate must be carefully planned to increase benefits and to avoid potential losses. HU et al. (2004) explore power gating to manage the availability of execution units depending on the current use. The cost of performance and power that are associated with this technique are the

variables that limit the reconfiguration rate. Since there is a cost that must be spent to accomplish the disconnection of the power supply lines, there is a minimum time for the availability of execution units. If the reconfiguration does not respect this kind of constraint, the amount of energy used to adapt the system would be higher than the energy savings originated from the use of the technique. Using this approach, and obeying the last restriction, they found that the execution units could be put to sleep for up to 28% of the execution time at a performance loss of only 2%.

Finally, the physical resources that are going to be modified as well as the specific techniques to accomplish this task must be determined. The kind of resources could be the size and associativity from a cache, the availability of execution units, the fusion of cores, etc. ALBONESI (1999) implements cache reconfiguration through modification of the level of associativity to meet the goal of managing energy budget. Some sub-set of ways are disabled depending on the demands to reduce cache switching activity. The policy is applied for the whole execution time of a program, so one cache size is constant for each application. It was obtained a reduction about 2% for performance and savings nearly to 40% in cache power dissipation.

It must be noted that some microarchitecture adaptable approaches attempt to reconfigure the system at higher levels with the objective of making a higher impact for performance and energy efficiency. In this kind of strategy, Reconfigurable Computing is not only restricted to specific datapath modules of a single processor but the entire system is behaviorally and structurally modified. IPEK et al. (2007) evaluate the use of core fusion to better adjust the application demands to the characteristics of the CPU. They use two custom instructions visible for the operating system, FUSE and SPLIT, which combine independent cores or split one large core into simpler ones (Figura 2.11). It accommodates software diversity and incremental parallelization in chip multiprocessors CMPs. The results showed that it provides a single execution model across all configurations, as well as it does not need additional programming effort maintaining ISA compatibility.

Figura 2.11 - Diagram of an eigh-core CMP with core fusion capability. It depicts a configuration example of two independent cores, a two-core fused group, and a four-core fused group.



Reference: IPEK et al. (2007)

2.4 Critical Analysis and Contributions

As was presented along this chapter, the need for more energy efficient computer architectures has established one of the main design requirements for new embedded systems. This design pressure is the reason behind the adoption of dynamically adaptable systems which, besides being optimized for a specific range of points in the design space, shows selective use of the resources depending on the workload. While speed-up was the variable that guided IC industry during the first decades of development, and power saving was the requirement that arose when the industry reached the technological limits, adaptability promises may deliver a mid-term between them. This new design paradigm encourages the IC engineers to provide the best performance for any application but at the same time using the resources efficiently depending on the needs and requirements of the moment.

The technological evolution of CMOS industry has showed an increasing importance for the management and control of power consumption in an integrated circuit. Though dynamic power has been the main contribution for total power consumption, static power has gained more attention because of higher values for sub-threshold leakage and gate leakage

related with newer technologies. A variety of physical techniques has been proposed to address this problem like power gating, clock gating and DVFS.

The related work showed the adoption of two main approaches to accomplish workload adaptation for energy savings: systems which schedule the current running code for the resources available (i.e. heterogeneous systems) and others which adapt the resources available for the current running code (i.e. reconfigurable computing). Each one of them presents different advantages in terms of energy savings and performance depending on the impact that the adaptation policy has over the normal execution of the microprocessor.

In terms of design effort for an adaptable system, the complexity that is demanded from hardware and software is higher compared with a static one. The system ought to be efficient in terms of measuring the state of the execution and implementing the adaptation policy. The former refers to the correct measuring of hardware metrics that characterize each application, as well as each phase when the granularity is finer. The quantity and quality of the variables that are measured determine how much improvements can be derived from an adaptable solution. Secondly, an adaptation policy must include all the algorithms and extra hardware sub-units that are needed to carry out the physical adaptation process. The selection of a hardware or software approach has different impacts and they must be studied carefully during the design phase.

2.4.1 Our approach

Based on all the current research compiled in this survey, we present in the following chapters the design challenges, methodologies and the impact of an adaptable computing solution for a VLIW processor. In chapter IV the influence of issue-width for performance, power and area for a VLIW microprocessor is addressed. According to a methodology inspired by the seminal and aforementioned work KUMAR et al. (2004), we explore the potential optimization that could be achieved by a dynamic issue-width VLIW microprocessor.

Then, two methodologies for microarchitecture adaptation based on power gating are described: a hardware-based and a software-based approach. The former one uses additional logic into the microprocessor with the purpose of applying power gating as was previously presented. The objective of this kind of approach is to implement into hardware all the necessary modules to detect potential idle parts of the integrated circuit based on historical behavior and generating all the suitable signals to control the power gating circuitry. The specific type of heuristics used to classify a physical domain as idle depends on the specific research and different methods have been proposed SHIN et al. (2010). In our case we

implemented Time-based power gating HU et al. (2004). It is aimed to turn off execution units by power gating them after observing a streak of idle cycles. Additional VHDL logic was implemented into our VLIW processor with the objective of implementing the hardware counters responsible for detecting long-idle periods.

As was described in this chapter, implementing a power gating policy in hardware is not trivial due to the associated power and timing overhead. For that reason, we propose the use of software directives for power gating VLIW datapath resources. This strategy has clear advantages and has been used in recent investigations PARK et al. (2010). First, the use of compiler technology allows identifying idle periods in advance, which is a difficult and expensive task when implemented in hardware RELE et al. (2002). Secondly, the additional hardware logic that is needed to apply power gating is reduced, since the complexity of the problem is moved to the compiler.

Finally, we implemented the use of customized power gating instructions in VLIW processors to disable idle FUs and blocks of the RF. This is done by obtaining and analyzing the execution profile of a given application, and then identifying the maximum RF use and the idle periods of the FUs. To achieve this goal, enhancements in the compiler and the microarchitecture of the VLIW processor were done. Our approach presents the following advantages in comparison to previous power gating techniques:

- The employed configurable VLIW architecture helps reducing the extra hardware logic needed for scheduling. A superscalar processor needs a more complex scheduler to manage adaptive FUs, since the available resources are continuously changing and additional hardware logic to implement this adaptability must be added. Taking into account that the scheduler logic of out-of-order processors is complex by nature, this overhead could produce a significant impact on the design. In a VLIW processor, the logic is simplified since all the scheduling decisions are made before execution.
- The VLIW code of an application is normally composed of available slots that are not used for computing purposes, which are usually filled with NOP instructions. These available slots allow the insertion of customized instructions for power gating, modifying the code without increasing its size. In a superscalar processor, the code grows since the power gating instructions must be added to the binary code.
- The proposed use of customized instructions for power gating allows managing the availability of the FUs together with the RF. The combined application of compiler-based power gating to these two resources is a methodology which, to the best of our knowledge, has not yet been addressed.

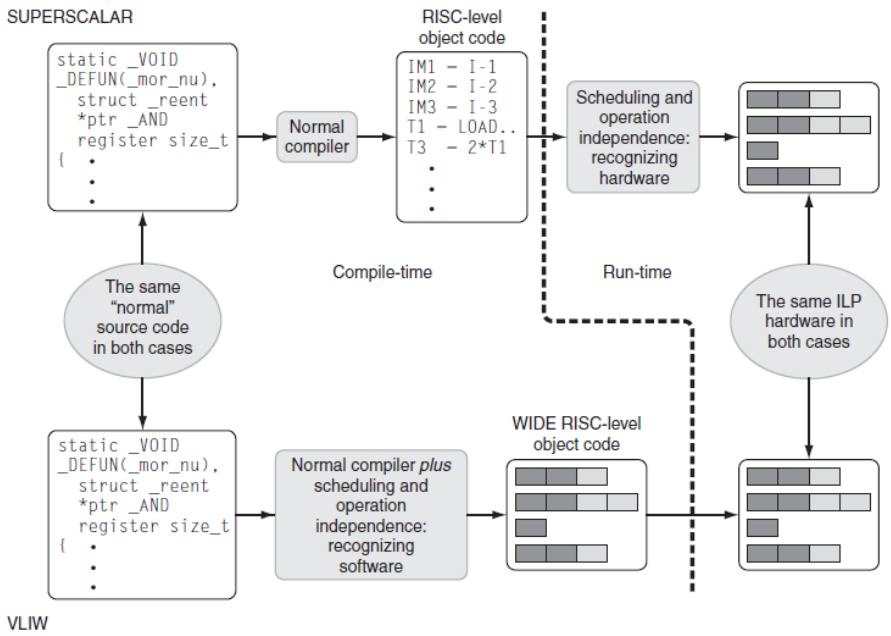
The use of power gating instructions on VLIW processors takes advantage of the customizability of the VEX architecture (which is the ISA used along this work) and demands enhancements on the microarchitecture. Unlike other power gating techniques on VLIW processors based on additional control hardware LIAO; BASILE; HE (2002), our approach is completely based on software directives. This means that area and power overhead are completely minimized and just extra simple logic is required to decode the customized instructions. Other researches, based on re-scheduling VLIW instructions to apply power gating UCHIDA et al. (2012), LIAO; BASILE; HE (2007), are intended to reduce energy with no concerns regarding performance. Our algorithm does not modify the original performance-driven code but takes advantage of the availability of unused slots to insert new instructions. In other researches, which use software directives to apply power gating on VLIW processors NIEDERMEIER et al. (2010), the decisions about the control of the power domains are taken by previous knowledge about the application and its phases. For instance, if the programmer knows that in one specific interval of execution there are no floating point operations, this unit is disabled via power gating. In contrast, our approach proposes a complete fine-grained framework that does not require any a priori knowledge of the programmer about phase behavior. Furthermore, some researches have applied power gating to control different datapath resources FLAUTNER et al. (2002), LIAO; BASILE; HE (2002), but none of them has addressed the use of this kind of technique to manage the availability of the FUs and the RF jointly.

3 VLIW DESIGN

3.1 VLIW basics

VLIW architectures are an alternative to superscalar designs, exploiting ILP through compiler instead of using hardware resources. The compiler is responsible for building long instruction words, which are composed of various independent operations that will be executed at the same time. The main function of VLIW hardware is to split each word and distribute the operations among the functional units (FUs) at run-time. The exploitation of ILP is done through the use of several functional units with a simple control logic, avoiding the expensive dynamic scheduling hardware of contemporary superscalar processors (Figure 3.1). In this way, all the computation efforts for this task are put on the compiler which lightens the load that is normally handled by the hardware. It considerably reduces power consumption and design complexity FISHER; FARABOSCHI; YOUNG (2005).

Figure 3.1 - Execution in a VLIW versus Superscalar.



Reference: FISHER; FARABOSCHI; YOUNG (2005)

The Instruction Set Architecture of a VLIW processor is normally composed of RISC instructions, which must be assembled by the compiler in order to use all the functional units efficiently. It requires the presence of sufficient ILP in the application to keep all the resources relatively busy. Some of the compiler techniques that are used for this purpose are software pipelining, scheduling code along basic blocks speculatively, reducing the number of operations, etc.

As any other computer architecture, VLIW processors have some technological disadvantages that must be taken into account. In this group of features we can find examples such as code size, which significantly increases due to aggressive scheduling policies; higher memory and register file bandwidth because of the use of larger instruction words; no binary compatibility between different VLIW processors with different type and number of functional units, etc.

3.2 Commercial VLIW processors

A great part of the commercially available VLIW processors uses a fixed issue width, such as TMS320C611 from Texas Instruments, S231 from STMicroelectronics or TriMedia series from NXP. Some efforts for reconfigurable VLIW systems can be found in ZHONG; LIEBERMAN; MAHLKE (2007) and for superscalar systems in SANKARALINGAM et al. (2003) and IPEK et al. (2007). 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. By contrast, the current research is mainly focused on analyzing the influence that such adaptive architectures have over energy consumption and other metrics.

3.3 VEX Architecture

The architecture that we use in this work is the VEX instruction set architecture FISHER; FARABOSCHI; YOUNG (2005). It defines a 32-bit clustered VLIW ISA that is scalable and customizable to individual application domains. It offers the use of multi-cluster machines, with each cluster being an independent VEX implementation. VEX does not support floating point operations. The resources that are found by default in a VEX cluster are 4 ALU units, 2 multiplier (MUL) units, 1 branch control (CTRL) unit, 1 memory access

(MEM), 64 32-bit general purpose registers (GR) and 8 1-bit branch registers. Many syllables compose a VEX instruction, depending on the issue-width. Each syllable can be understood as a RISC instruction.

A set of rules are stablished, those that all implementations must obey (such as register connectivity, the base ISA, architecture state, memory coherency) and those that each specific implementation can define (kind and number of functional units, latencies, issue width, number of clusters, and custom instructions). This last feature allows us to enhance the capabilities of the ISA, adding custom instructions, without affecting the compatibility with the VEX architecture. Therefore, the tools that are developed for this kind of ISA (compilers, simulation environment, etc.) can be used to implement new capabilities.

Hewlett-Packard provides a VEX software toolchain, which has a C compiler and a simulator. These tools can be parametrized through the loading of machine models which could be specific for each VLIW implementation. The VEX C compiler was derived from Lx/ST200 C compiler, which was at the same time an enhancement of the Multiflow C compiler. The VEX simulator produces a binary executable through the translation of the target executable binary code.

3.4 ρ -VEX processor

The processor that was used along this work for simulation purposes was the ρ-VEX, which is a configurable processor implemented in VHDL and that implements the aforementioned VEX architecture WONG; VAN AS; BROWN (2008). 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 (Figure 3.2).

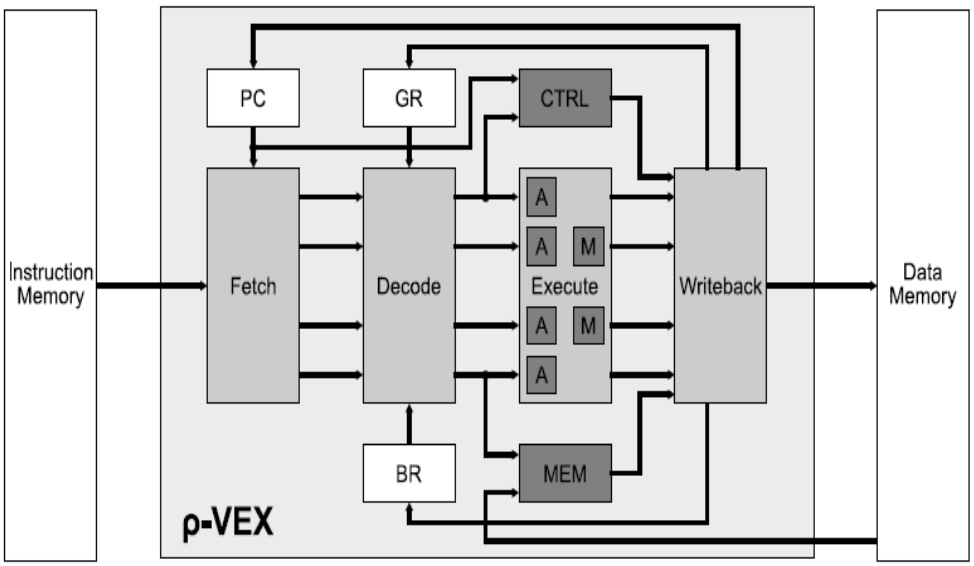
The execution stage is parameterizable, since the number of ALUs and MULs can be changed. The CTRL unit handles all branch and jump operations, whereas all load and store operations are performed by the MEM unit. The register file, branch register, and program

counter are written back at the respective unit, to ensure that all the targets are modified at the same time. The write target of each operation is determined at the decode stage.

The extensibility of ρ-vex processor is implemented through two mechanisms that are provided by VEX architecture. First, the use of custom instructions via pragmas inside the application code allows enhancing the functionalities of the architecture. With only a few added lines of VHDL into the ρ-vex code it is possible to add a custom functionality. Secondly, the VEX machine models allow to define different parameters for the ρ-vex processor. In this group of variables, it is possible to modify the following properties:

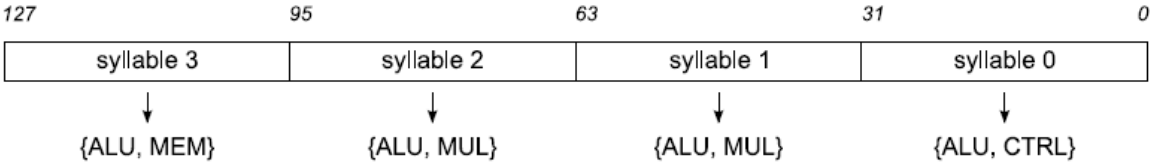
- Syllable Issue-width
- Number of ALU units
- Number of MUL units
- Number of GR registers. (Up to 64)
- Number of BR registers. (Up to 8)
- Width of memory buses
- Types of accessible FUs for each syllable.

Figure 3.2 - ρ-vex organization for 4-issue-width.



Reference: WONG; VAN AS; BROWN (2008).

Figure 3.3 - Instruction Layout for ρ-vex processor

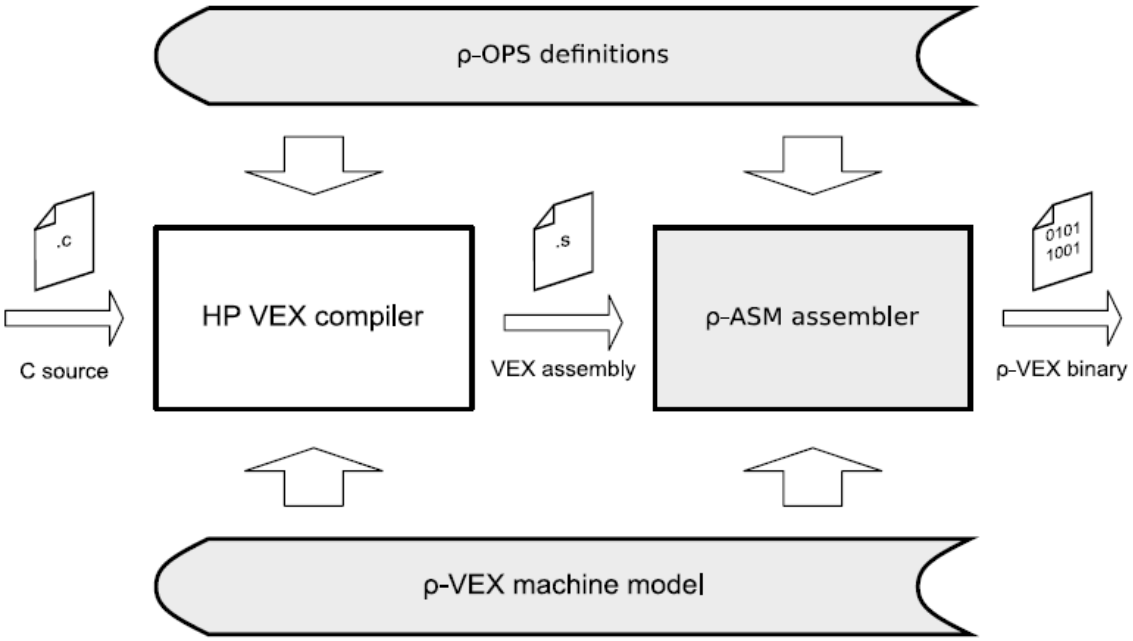


Reference: WONG; VAN AS; BROWN (2008).

3.5 Application development for ρ-vex processor

The development of experiments and programs for the ρ-vex processor can be summarized into two steps. The first one is the compilation of the C code with the VEX compiler. The machine model must be passed as a parameter for the compiler when a custom configuration is used. The second step is the generation of an instruction ROM for ρ-VEX, which is generated through the assembler p-ASM. The machine model definitions must be used in this step too. This whole process is depicted in Figure 3.4.

Figure 3.4 - ρ-VEX application development framework



Reference: WONG; VAN AS; BROWN (2008).

The ρ-VEX design organization used in this work was the following: register file of 64 registers, 8 issue-width (the issue-width was modified in some experiments to measure the potential of optimizations, in chapter 4), ALUs in all issue slots, one memory and one branch unit (due to ρ-VEX's design restrictions), and 4 multipliers. This configuration is similar to

other VLIW processors (e.g., Intel Itanium) WONG; VAN AS; BROWN (2008). The programs used in this work were compiled with the VEX compiler from HP labs using optimization O3 and the specific machine model for each experiment carried out.

The synthesis to obtain the power dissipation and area was carried out using an 180nm library from X-FAB X-FAB (2015) and Encounter RTL Compiler from Cadence Tools CADENCE ENCOUNTER (2015). The module synthesized was the ρ-VEX core, without any peripheral or memory attached. The operation frequency was set to 500 Mhz. The activity factor that was assumed for the calculation of dynamic power consumption was of 30%, which is a value that has been traditionally used for system level analyses of microprocessors GEUSKENS; ROSE (2012). This variable assumes that even when some parts of the circuit are not used along specific parts of the execution (e.g. Functional Units), the total switching activity is averaged to a specific value of 30%.

4 EVALUATION OF ENERGY SAVINGS ON A VLIW PROCESSOR THROUGH DYNAMIC ISSUE-WIDTH ADAPTATION

As already discussed, one of the main issues when it comes to designing a VLIW processor from scratch is about project decisions, such as choosing the right issue-width and the register file size. The issue-width influences the availability level 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 also presents as drawback increasing the area and power dissipation.

In this chapter, we study the potential energy savings that might be obtained by adapting VLIW issue-width according to the current program phase. Based on the measuring of ILP for different phases, we study an optimal scenario where we combine both performance and low energy consumption by adapting all the resources available to the average ILP of each phase.

Therefore, this chapter has two main purposes:

- Describe quantitatively the impact of issue-width for energy consumption, performance, and area on a VLIW microprocessor.
- Analyze the potential energy savings that could be obtained by dynamically adapting the issue width on a VLIW microprocessor 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).

By considering that the VLIW issue-width is dynamically changed along the program execution, the potential energy savings using this policy could be as high as 81.5% when compared with the static version.

The rest of this chapter is organized as follows. Section 4.1 shows the potential of optimization by analyzing the impact of design choices on performance and energy consumption. Section 4.2 discusses two approaches for evaluating the phases of an

application. Section 4.3 describes the oracle experiment performed to evaluate the energy savings potential of choosing the most appropriate issue-width for a given phase of the program. Finally, Section 4.4 summarizes our conclusions.

4.1 Potential of Optimization

In this section, different values for VLIW issue-width are evaluated in order to assess the potential optimization that can be achieved through issue-width adaptation. The impact on area, performance, energy consumption and area is addressed.

Figure 4.1 depicts the area of different issue-widths, varying from 1- to 8-issue (this range was used because of ρ-vex restrictions). 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 Figure 4.2. The 8-issue dissipates 2.1 times more power than the 4-issue and 6.86 times more power than the single-issue.

Figure 4.1- Area comparison between different issue-widths

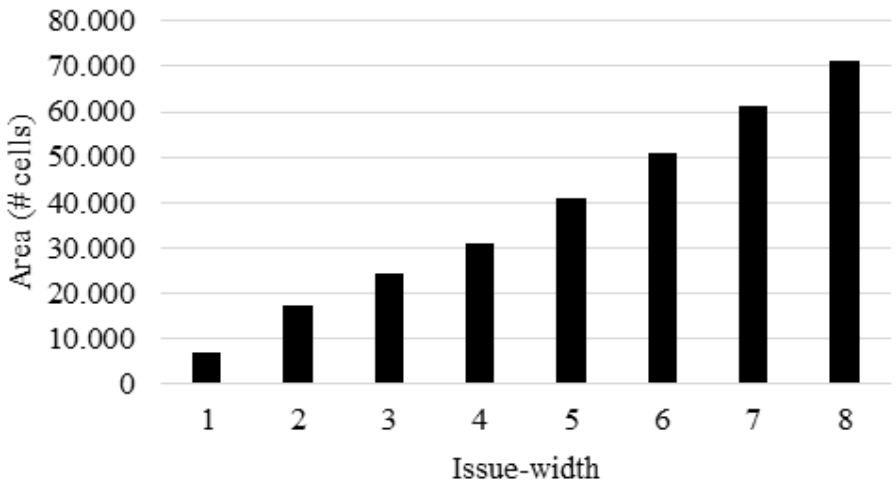
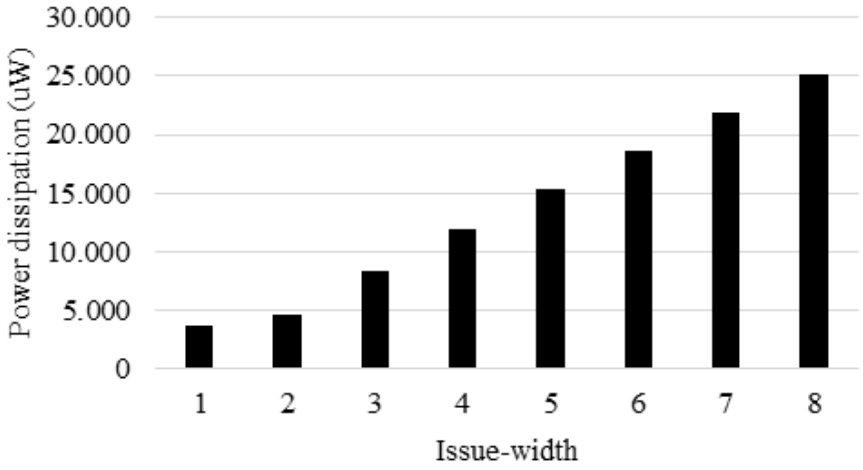


Figure 4.2 - Power comparison between different issue-widths.



In Figure 4.3, 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%.

Figure 4.3 - Speedup compared to the 4-issue VLIW

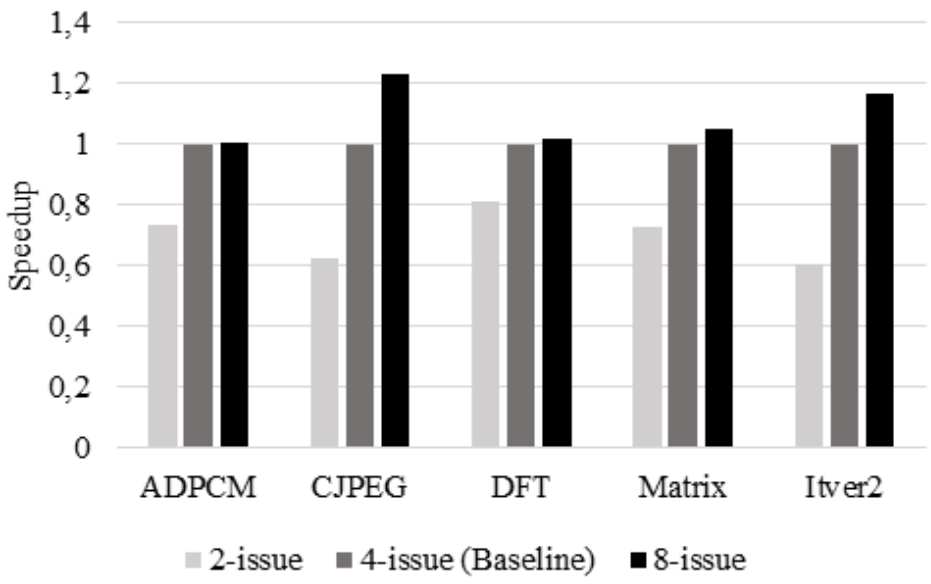
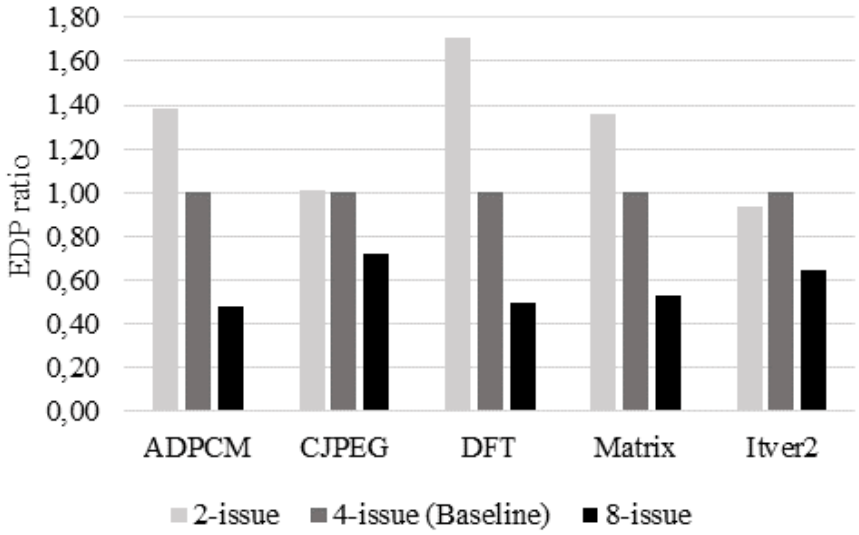


Figure 4.4 - EDP ratio for different applications



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 effectively exploit ILP for wider issues.

Figure 4.4 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.

4.2 Dynamic adaptation

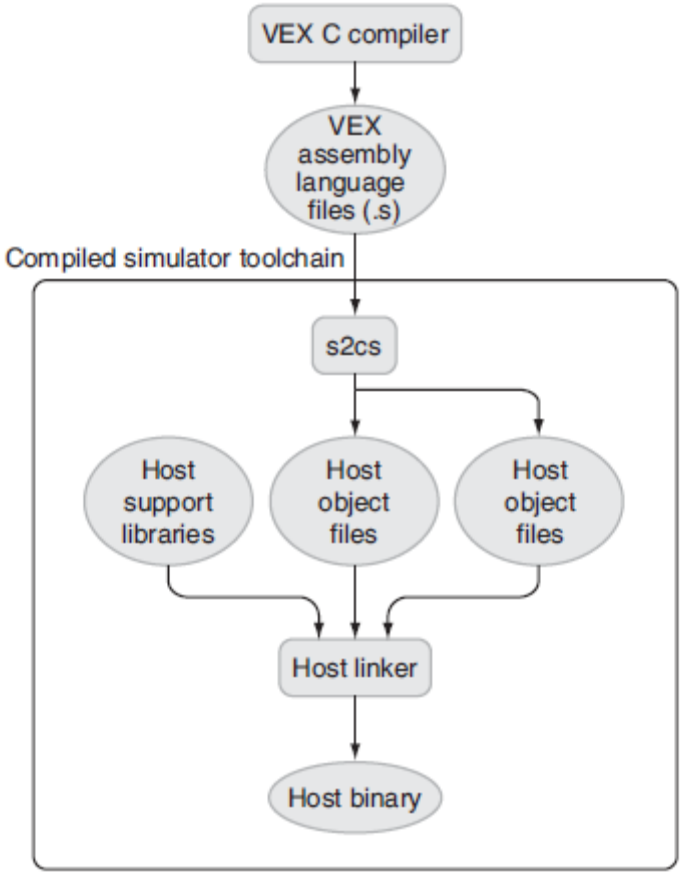
The aforementioned analysis highlights the enormous potential that an exploration of the design space could produce in terms of energy savings if microarchitectural adaptation was available at run-time. For instance, if one part of a program does not use certain issue slots, it is not necessary that 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.

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. In this way this information will be used to determine the issue-width that best matches the IPC of each phase.

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. 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. Therefore, we measured the evolution of IPC throughout time to detect the phase changes and hence the dynamic demand of computing resources.

4.2.1 Methodology

Figure 4.5 - Diagram Flow for the VEX simulator toolchain



Reference: FISHER; FARABOSCHI; YOUNG (2005)

The HP's VEX simulator FISHER; FARABOSCHI; YOUNG (2005) was modified to obtain the IPC at run-time, extracting the number of issues used by each instruction word. The VEX simulator is an architecture-level simulator that uses compiled simulator technology to achieve a speed of many equivalent MIPS. The simulation system also comes with a fairly complete set of POSIX-like libc and libm libraries (based on the GNU newlib libraries), a simple built-in cache simulator (level-1 cache only), and an API that enables other plug-ins used for modeling the memory system. The VEX compiled simulator uses a binary translator to generate an executable binary for the host platform that contains the operations for simulating a program compiler. Finally, the application's execution on the VEX architecture is simulated. To exemplify the action of this toolchain, in Figure 4.6 we can see an example of one original VLIW instruction in assembly, whereas in Figure 4.7 it is presented the compiled-simulated code. Note that each operation in the assembly code has an associated function in

the compiled-simulated code that counts the number of occurrences for that kind of instruction along the execution of the program.

Figure 4.6 - Example of a VLIW instruction in assembly

```
135   .trace 3
136   L0?3:
137       c0    cmplt $b0.0 = $r0.2, $r0.0   ## bblock 1, line 115,  t52(I1),  t40,  0(SI32)
138       c0    mov $r0.57 = (~0x13)   ## [spec] bblock 3, line 0,  t29,  (~0x13)(I32)
139       c0    mov $r0.58 = $r0.3   ## [spec] bblock 3, line 0,  t28,  t39
140       c0    mov $r0.59 = $r0.2   ## t40
141       c0    mov $r0.60 = $r0.3   ## t39
142   ;;                              ## 0
```

Figure 4.7 - Example of the compile-simulated code

```
  sim_icache_fetch(45 + t_thisfile.offset, 5);
{
    _ADD_CYCLES(1);
    _INC_BUNDLE_CNT(5);
    _CMPLT(reg_b0_0, reg_r0_2, 0); /* line 137 */
    _MOV(reg_r0_57, (unsigned int) -20); /* line 138 */
    _MOV(reg_r0_58, reg_r0_3); /* line 139 */
    _MOV(reg_r0_59, reg_r0_2); /* line 140 */
    _MOV(reg_r0_60, reg_r0_3); /* line 141 */
}
```

In general terms, the modification of the VEX simulator for IPC measurements was based on the addition of profiling functions to the host platform compiled-simulated code. The objective of these profiling functions was to count the number of operations for each instruction executed as well as to count the number of total instructions processed in specific intervals of time. An example of this modification is shown in Figure 4.8.

Figure 4.8 - Insertion of profiling functions inserted into compile-simulated code

```
  sim_icache_fetch(45 + t_thisfile.offset, 5);
{
    _ADD_CYCLES(1);
    _INC_BUNDLE_CNT(5);
    _CMPLT(reg_b0_0, reg_r0_2, 0); /* line 137 */
        profiling_instruction();
    _MOV(reg_r0_57, (unsigned int) -20); /* line 138 */
        profiling_instruction();
    _MOV(reg_r0_58, reg_r0_3); /* line 139 */
        profiling_instruction();
    _MOV(reg_r0_59, reg_r0_2); /* line 140 */
        profiling_instruction();
    _MOV(reg_r0_60, reg_r0_3); /* line 141 */
        profiling_instruction();
        count_instructions();
}
```

The function *profiling_instruction()* counts an additional operation processed. In this case, since there are 5 operations into the VLIW instruction *profiling_instruction()* is invoked 5 times. The function *count_instructions()* counts the number of instructions in the current time window, thereby it is only invoked one time in this example. Each time a new instruction is executed, *count_instructions()* increases an internal counter. Depending on the window size, this internal counter is reset when it reaches a predetermined threshold.

The approach for obtaining IPC at run-time was chosen according to two implemented methodologies, which differ in the way they handle the instruction window sizes for phase measurement and therefore the way *count_instructions()* is implemented. They are called coarse-grained and fine-grained approaches, which are explained in the next sub-sections.

The programs used were extracted from Mibench, which is a free, commercially representative embedded benchmark suite GUTHAUS et al. (2001). 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*.

4.2.2 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 Figure 4.9a, Figure 4.9c, and Figure 4.9e (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.

4.2.3 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 Figure 4.9b, Figure 4.9d, and Figure 4.9f 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.

4.2.4 Coarse vs. fine-grained approaches

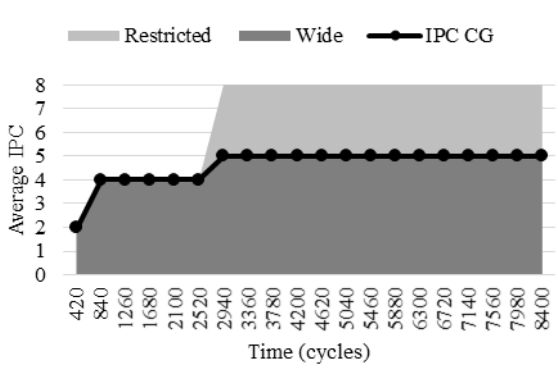
As can be observed from the data obtained, the applications exhibit dynamic behavior that could be successfully exploited via microarchitecture adaptation through coarse or fine-grained approach.

From the results we can observe that each application exhibits completely different dynamics, 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.

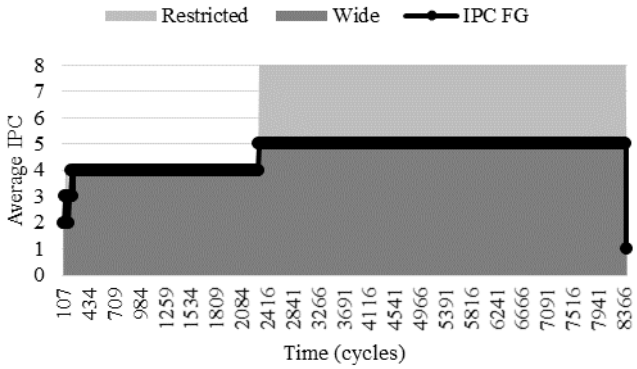
The benefits that coarse and fine-grained approaches for dynamic adaptation are different as well as the implementation challenges associated. The first approach aims to give an outlook of the dynamics of the program by averaging IPC along a big number of instructions, while the second produces higher 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.

The measurement of IPC through the coarse-grained approach would have 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.

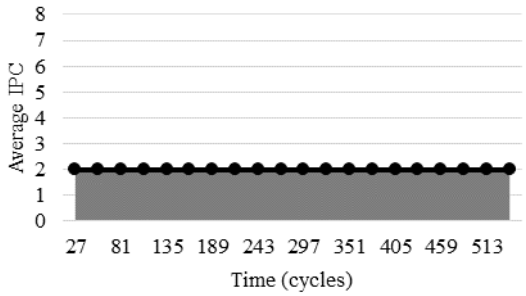
Figure 4.9 - Average IPC during the execution



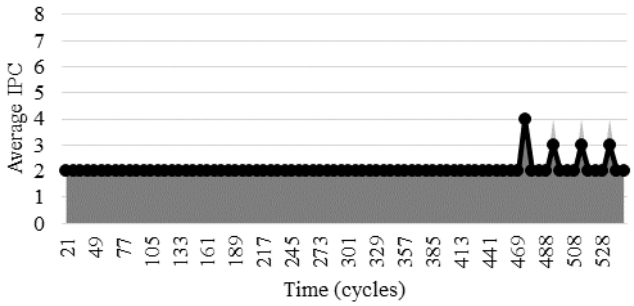
a) Basicmath coarse-grained



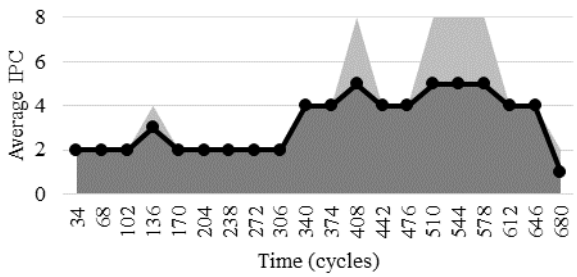
b) Basicmath fine-grained



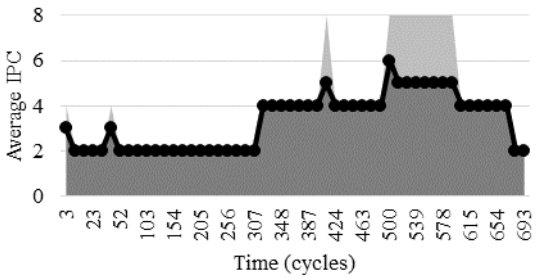
c) StringSearch coarse-grained



d) StringSearch fine-grained



e) sha coarse-grained



f) sha fine-grained

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. The behavior of this memory is similar to a branch prediction table and an equivalent microarchitecture could be used for its implementation.

4.3 Oracle Heuristics for Dynamic Issue-width Selection

Based on the previous experiments, 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. Since we are interested in obtaining the maximum potential energy savings, no technological overhead is taken into account for each reconfiguration process.

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. The oracle experiment choose the best suited issue-width for each interval of execution, knowing in advance the ILP for that period of time. 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 Figure 4.2 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.

Figure 4.10 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 Figure 4.10 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-

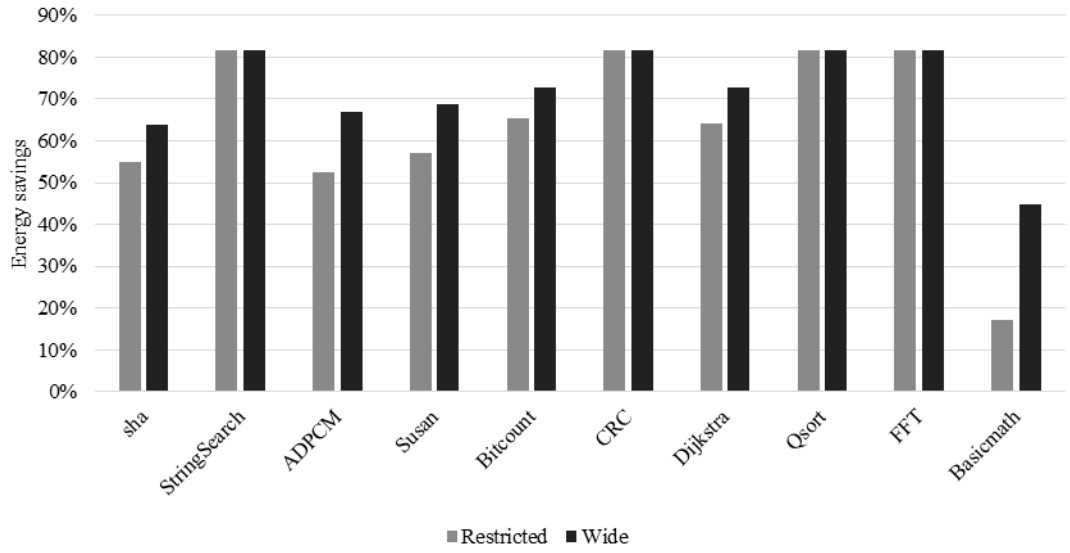
widths, all of which the wide approach is also able to choose. That is, for phases that have an average IPC of 2, 4, or 8 (i.e., the values that the restricted adaptation is able to choose), the wide adaptation (that can choose from 1- to 8-issue) will choose the same issue-width as the restricted, having 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 Figure 4.9a. 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. 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.

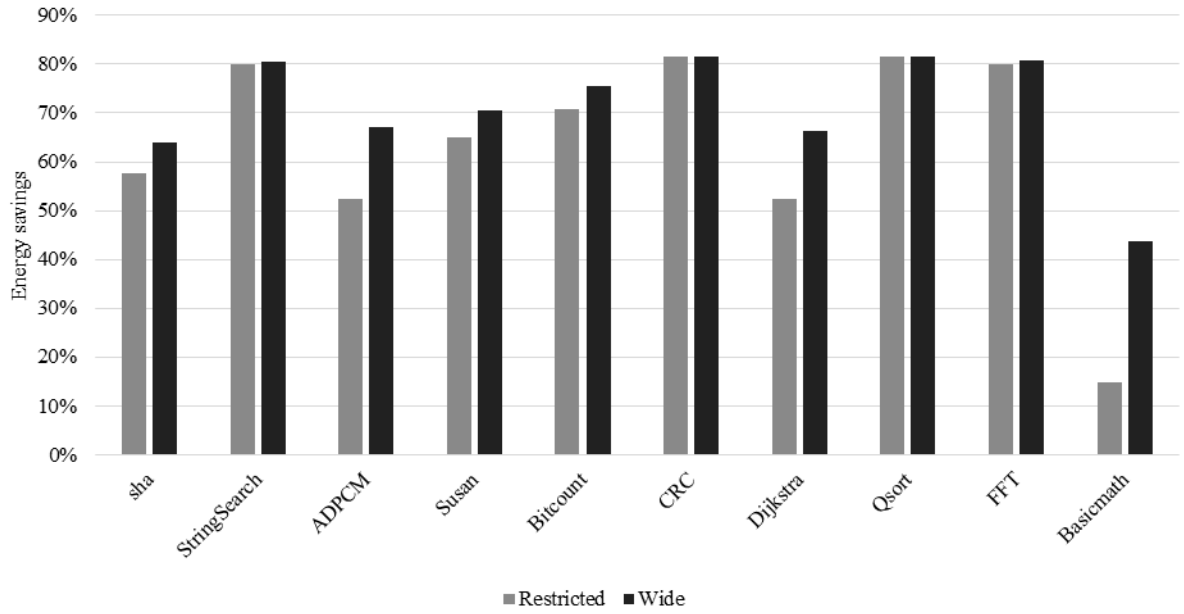
4.5 Critical Analysis

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

Figure 4.10 - Energy Savings



a) Coarse-grained approach



b) Fine-grained approach

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.

It is worth noting that in a real application, the decision about using wide or restricted adaptation is dependent on the available project resources. If a larger group of available issue-width values is handled, the complexity of the hardware would be significantly increased. Specifically, extra logic must be added in order to support a larger group of microarchitectures. This means that the overhead of a wider adaptation, in terms of area, could make unaffordable its implementation if this cost is high.

As was mentioned along this chapter, the ideal solution for an energy efficient VLIW processor would be having as many Functional Units available and Registers as possible when needed, and turning them off if the application does not offer enough ILP. In the rest of this thesis, we propose the use of specific power gating techniques for FUs and RF to exploit the availability of long idle periods to disable these kind of hardware modules. In this way we not only could decrease dynamic power consumption but also static power consumption efficiently.

There will be presented a hardware and a software approach in Chapter V and Chapter VI respectively. The advantages and disadvantages of each one of these methodologies will be discussed in these chapters as well as a final discussion about their comparison will be extended in Chapter VII.

5 TIME BASED POWER GATING FOR VLIW PROCESSORS

The results presented along Chapter IV evidence the potential energy savings that might be obtained through an adaptable VLIW processor. In this chapter, we explore the use of hardware techniques to accomplish such objective. In order to adapt the amount of available resources suited to each interval of execution, the use of additional logic embedded into the processor is proposed to detect idle periods of the FUs. According with this detection, the FUs can be disabled or enabled via power gating depending on the demand and state of the corresponding resource.

With this in mind, the ρ-vex processor was modified by inserting new logic to carry out the detection of idle periods of the FUs, their activation/deactivation and the measurement of the performance losses. Through the use of hardware counters to detect long idle periods, the units are turned off when their inactivity surpasses a user-defined threshold. In this way, the historical behavior of the Functional Units is used to successfully apply power gating without significant time penalties. The turn off and turn on processes are simulated by adding new registers that save the power state of each one of the resources. Furthermore, the impact on the performance is calculated by counting the overhead associated with each wake-up process (i.e the case when a FU is needed but such unit is not available yet).

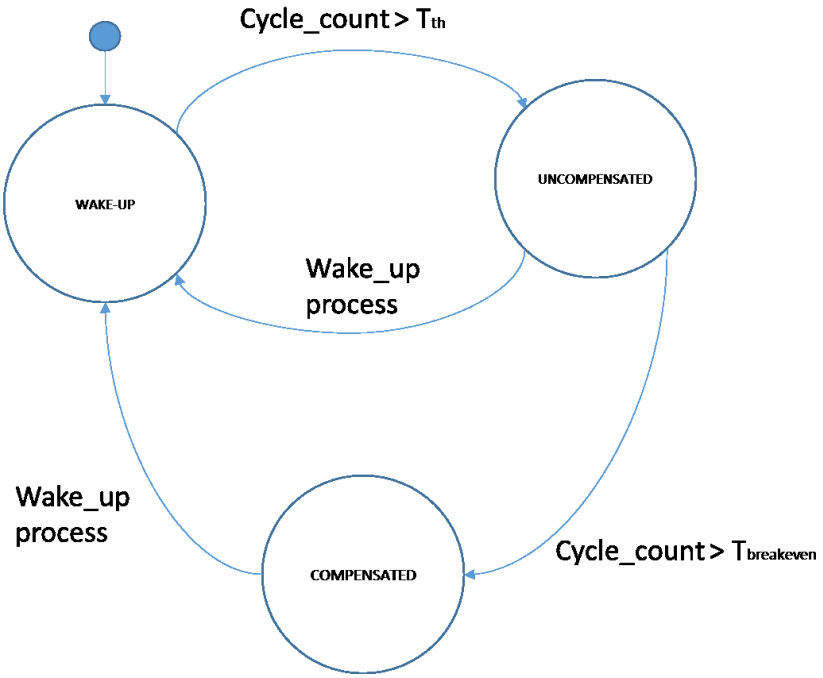
Our results show that the FUs can be put to sleep on average 63% of the execution cycles for the multiplier units and 30% for the ALUs, at a performance loss of 13%. By varying the threshold for detecting idle inactivity, it is possible to observe different impacts on the performance-power tradeoff. Overall, our results prove that hardware techniques can be used effectively to power-gating execution units on a VLIW processor.

The rest of this chapter is organized as follows. Section 5.1 shows the conceptual basis of a time based power gating. Section 5.2 presents the methodology used to implement this kind of hardware approach. Section 5.3 discusses the results obtained through the simulation of a group of benchmarks. Finally, Section 5.4 summarizes our conclusions.

5.1 Time Based Power Gating

Time-based power gating HU et al. (2004) is a power gating technique based on additional logic to detect long-idle periods for the Functional Units and the corresponding turn-on and turn-off decisions. It is aimed to turn off execution units by power gating them after observing a streak of idle cycles. In order to implement this idea, a finite state machine (FSM) is added for each execution unit of the VLIW processor, as depicted in Figure 5.1. The initial state of the FSM is WAKEUP. If the execution unit is idle during a number of cycles that exceeds some threshold T_{th}, power gating can be applied and the state changes to an interim state called UNCOMPENSATED. If the execution unit remains at the same idle state after $T_{breakeven}$ cycles, then it moves to COMPENSATED state. As can be expected, in these two states the execution units are turned off but only at the COMPENSATED state there are positive energy savings. When an execution unit is in COMPENSATED or UNCOMPENSATED state and an instruction needs it, the execution unit must wake up. In that case, it is necessary a number of T_{wakeup} cycles to carry out the wake-up process. If the unit is not completely power-down the time that the execution units takes to wake up will be less than T_{wakeup} because the voltage difference will be smaller between the two states (enabled and disabled). However, we take a conservative approach by assuming wake up process is equal to T_{wakeup} cycles. Therefore, the real performance impact will be much higher since we are assuming the most pessimistic setting.

Figure 5.1 - State machine of an execution unit when power gating is inserted.



The energy savings as well as the performance impact are dependent on the aforementioned parameters: $T_{breakeven}$, T_{wakeup} and $T_{idledetect}$. Whereas the first two variables are constraints defined by circuit design limits, the third one is a design decision which can be adjusted taking into account the trade-off between energy savings and performance losses. If a large $T_{idledetect}$ is used, the performance would not be significantly affected because of less number of wake-up moments but at the same time the contribution of short idle periods to energy savings would be small. Conversely shorter $T_{idledetect}$ allows to exploit these idle intervals but with higher cost for performance.

5.2 Methodology

The 8 issue-width ρ-vex with 4 multipliers was modified to apply time-based power gating through modifications into the VHDL code. For each multiplier and ALU unit a FSM, as described in the last section, was added to carry out the application of time-based power gating. Besides, the microarchitecture was enhanced with a power control register of 12 bits (8 bits for ALU units and 4 for multipliers), which saves the power state of each execution unit (1 for enabled or 0 for disabled).

The additional logic implemented into the VHDL code has five main objectives as was argued in the Section 5.1:

(1) Detect idle periods greater than a preset threshold for each functional unit.

(2) Change the power state when a potential idle period is detected. In other words, modify the corresponding bit in the power control register to ‘0’.

(3) Activate the execution unit in case a new instruction demands the use of this resource. Modify the corresponding bit in the power control register to ‘1’.

(4) Count the effective fraction of execution time that each execution unit is disabled.

We report experimental results based on traces of a set of 8 WCET benchmarks, *adpcm, mm_40, x264, matrix, fir, crc, ndes* and *dft* GUSTAFSSON et al. (2010). This group of benchmarks was selected because of its compatibility with ρ-vex processor. For all the results, in reporting average statistics, we use geometric mean across the corresponding benchmark suite.

5.3 Results

The total idle cycles for each functional unit were calculated via simulation as well as the total execution cycles for each benchmark. For the rest of this chapter, the fraction of

cycles spent in the sleep mode by an execution unit of a given type *P* is determined as follows:

$$\frac{1}{n * total\ execution\ cycles} * \sum_{i=0}^{i=n} (cycles\ in\ sleep\ mode)_i$$
$$where\ n = number\ of\ units$$

In this equation, it is calculated, for each functional unit, the ratio between the time spent in sleep mode and the total execution time. The idle cycles correspondent to all the instances of a specific type of FU are summed and they are weighted by the contribution of each instance. For example, since there are 8 ALUs in our simulations, for each ALU it is obtained the total cycles in sleep mode; they are summed, and the result is divided by the total execution cycles times the number of functional units, in this case 8.

The values of *P* (corresponding to each FU) were calculated for each benchmark and they were averaged along the benchmarks to obtain the metrics hat are shown in the next figures. Figure 5.2 and Figure 5.3 show the power savings for ALU units and Multiplier units. These figures show the impact of the parameters $T_{idledetect}$ and $T_{breakeven}$ on the expected number of disabled cycles. T_{wakeup} is fixed for these charts.

Figure 5.2 - Percent of cycles in sleepmode for ALUs units (y-axis) with different $T_{idledetect}$ (x-axis) and $T_{breakeven}$ = one of 5, 10,15, or 20 cycles. T_{wakeup} is fixed at 3 cycles.

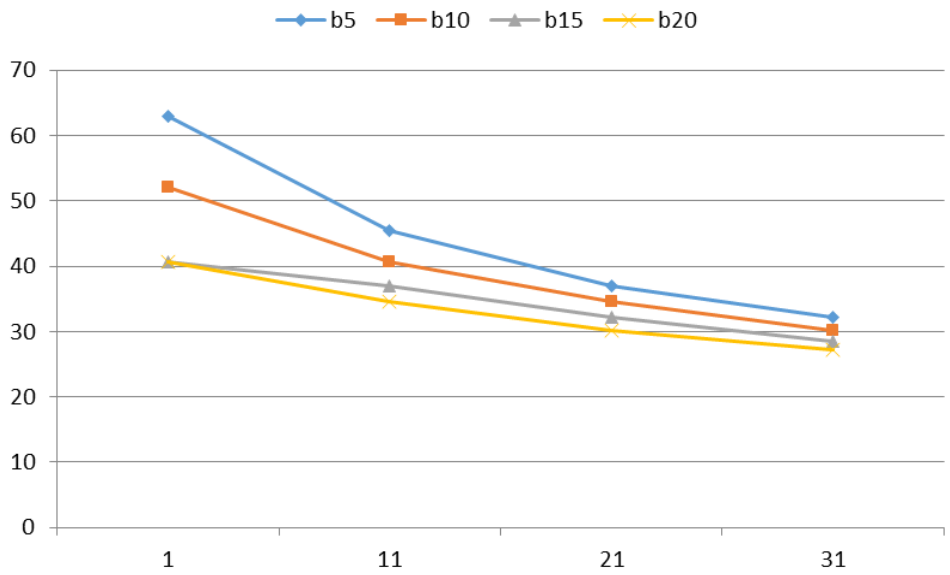


Figure 5.3 - Percent of cycles in sleep mode for Multiplier units (y-axis) with different $T_{idledetect}$ (x-axis) and $T_{breakeven}$ = one of 5, 10,15, or 20 cycles. T_{wakeup} is fixed at 3 cycles.

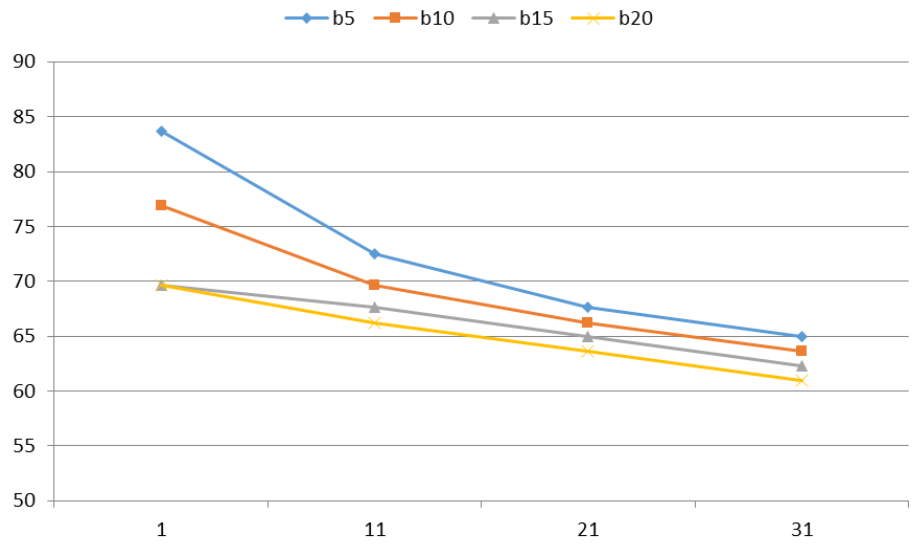


Figure 5.4 shows performance losses when the power gating overhead is taken into account. In this case, each time a wake-up process is carried out, the contribution of this overhead is summed to the total extra-cycles. In the same way that the first metric shown in the last figures, the performance losses were averaged along all the benchmarks simulated. The parameter $T_{breakeven}$ point was fixed to 10 cycles and $T_{idledetect}$ and T_{wakeup} were modified to observe their impact on performance.

Figure 5.4 - Average IPC of WCET benchmarks (y-axis) with different $T_{idledetect}$ (x-axis) and T_{wakeup} values. $T_{breakeven}$ is fixed at 10 cycles. IPC is normalized to the base case where power gating is disabled.

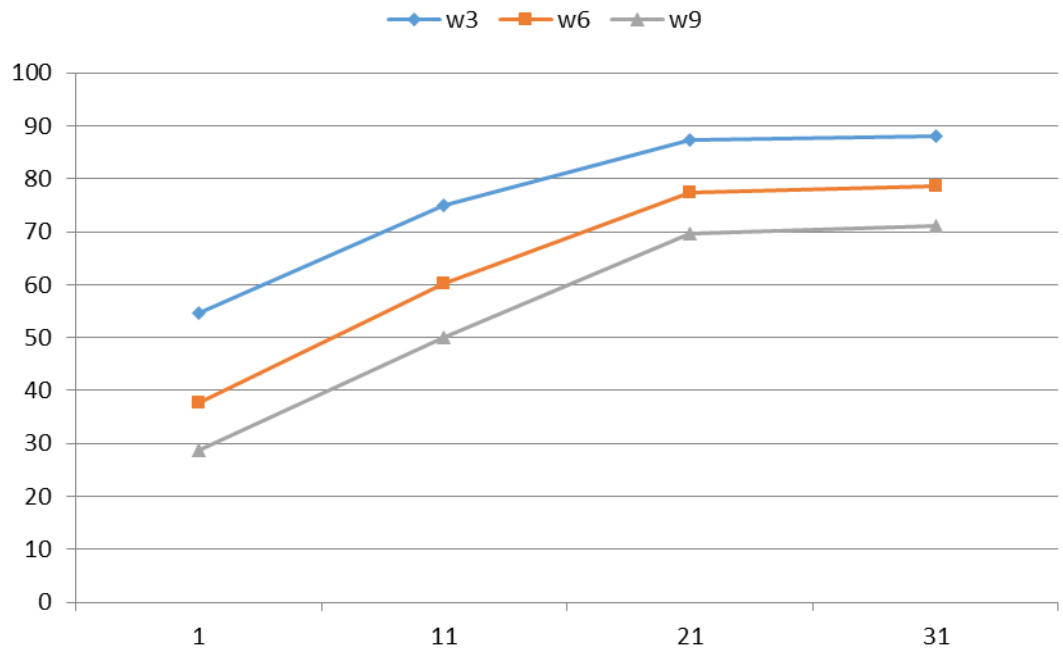


Figure 5.5 - Energy Savings for each application using $T_{idledetect}$ fixed to 21 cycles and $T_{breakevent}$ point to 10 cycles.

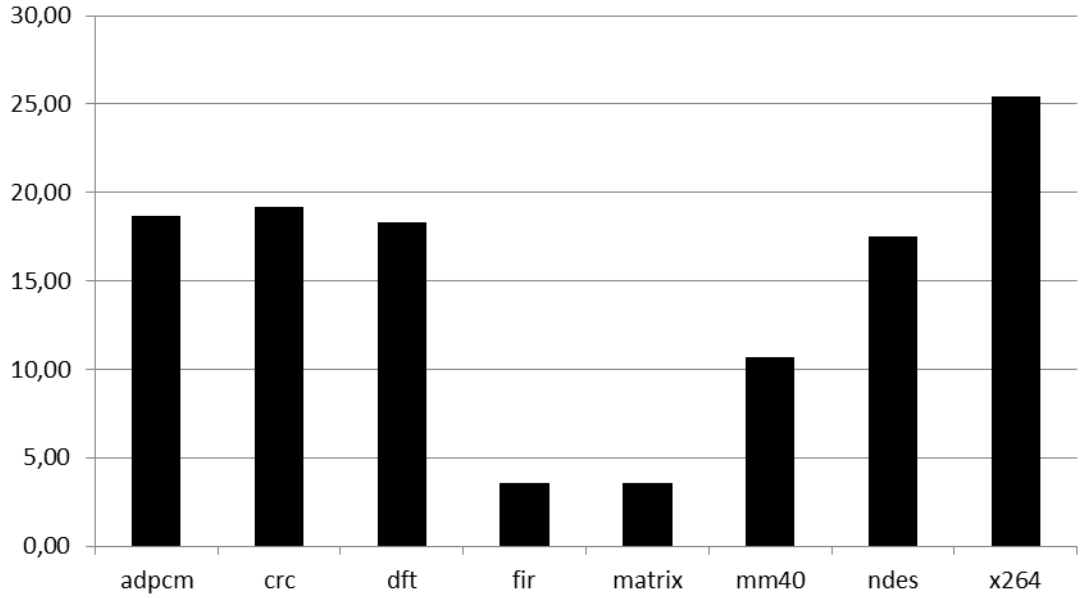


Figure 5.5 depicts the energy savings that are obtained using threshold fixed to 21 cycles and break-even point to 10 cycles, for all the benchmarks. The effective use of the functional units was used to calculate this metric, assuming a constant energy consumption for the Multiplier units as well as for the ALU units. On average the mean power savings are 14,63%, spanning from 3,57% for *matrix* and 25,44% for *x264*. This wide range remarks the difference in terms of benefits that this technique shows depending on the application. There are some programs that have more quantity of long idle periods for the functional units so time-based power gating could be used to take advantage of this behavior. On the other hand, there are applications that have a large quantity of shorter idle periods and the use of large thresholds prevent the use of power gating in those situations. This is the case of *fir* and *matrix* which have many short idle periods that are not detected with the threshold used in this setting.

As we can see from the figures, the percentage of cycles spent in sleep mode for every functional unit decreases almost exponentially with increasing $T_{idledetect}$ values. The performance, on the other hand, improves significantly when $T_{idledetect}$ increases from 1 cycle to 11 cycles, and then gradually reaches the performance of the base case, where power-gating is disabled. The big performance jump from 1 cycle to 11 cycles of $T_{idledetect}$ indicates the presence of short idle periods and these are not amenable to power gating. Though the number of such idle periods are large, power-gating in this case causes a significant

performance loss since each of these periods would incur timing overhead equals to T_{wakeup} cycles. The presence of smaller values for $T_{breakeven}$ and $T_{wakeup,}$ as well as long idle periods help to achieve larger energy savings and decreases the performance impact.

Figure 5.2 and Figure 5.3 show a particular behavior for energy savings around $T_{idledetect}$=11 cycles. Before this turn point, the curve drops very fast, indicating that very short idle periods dominate the total length of idle periods in WCET benchmarks. Since in integer applications branches, loads and stores appear once every 4 or 5 instructions, it correlates with the predominant idle periods.

It must be noted that the results that were achieved with this work are very similar to those found in HU et al. (2004). One of the differences is the value for $T_{idledetect}$ to achieve a specific amount of power savings. For instance, the mentioned research found a reduction of 10% for IPC using $T_{idledetect}$ = 6. To achieve the same IPC reduction we would have to use nearly $T_{idledetect}$=31. This means that it is necessary to neglect more short idle periods in our case to achieve the same energy savings. In exchange, for the same $T_{idledetect}$ we achieve better results in terms of power savings. For example, if we fix $T_{idledetect}$= 6, the aforementioned work found power savings nearly to 10% compared with the baseline case. For the same value of $T_{idledetect}$, we found reduction for power savings nearly to 50% for ALU units.

This behavior could be caused by the difference of processors and compilers used in the two researches. The cited paper used an out-of-order processor, whereas we use an in-order processor, namely a VLIW unit. This means that our system is not capable of processing other instructions until the current instruction is executed. In our case, the turn-on of a functional unit has a big impact in performance since it is necessary to execute all the operations of an instruction to process the next group of operations. Conversely, an out-of-order processor could process more instructions even if an execution unit is not ready yet. Some other functional units could handle operations that are independent while the wake-up process of a FU is completed.

In conclusion, the presented hardware-based approach for power gating Functional Units in a VLIW processor shows positive energy savings, which supports its feasibility and potential benefits. Using a time-based power gating solution, the Multiplier units can be put to sleep about 63% of the total execution time and the ALU units about 30%. It generates power savings about 14,63% with performance losses near to 13% for a typical configuration.

6 LEVERAGING COMPILER SUPPORT ON VLIW PROCESSORS FOR EFFICIENT POWER GATING

In this chapter it is evaluated a compiler-based approach for power gating functional units and register file in a VLIW processor. As it is shown in this chapter, intelligent use of the compiler allows for power gating at a finer grain saving considerable amounts of power. It is done so by inserting customized instructions at compile time, based on the analysis that involves probabilities of conditional branches and basic block information obtained via dynamic profiling. By using the compiler technique, it is possible to save up of 20% in the total energy consumption with marginal losses in performance. Unlike the technique that was described in Chapter V, which relies exclusively on hardware resources, this approach is based on the use of software directives to manage the availability of circuit domains throughout the execution trace.

The rest of this chapter is organized as follows. Section 6.1 shows our compiler power gating approach as well as the implementation challenges faced. Section 6.2 discusses the evaluation methodology and the resources that were used. Section 6.3 describes the results obtained by following this approach. Section 6.4 summarizes the conclusions.

6.1 Compiler-based approach for Power Gating

The methodology proposed is based on 1) obtaining the execution profile of the applications 2) using this information to determine the best locations for power gating instructions and 3) measuring the impact of these customized instructions on energy savings and performance losses.

6.1.1 For the Functional Units

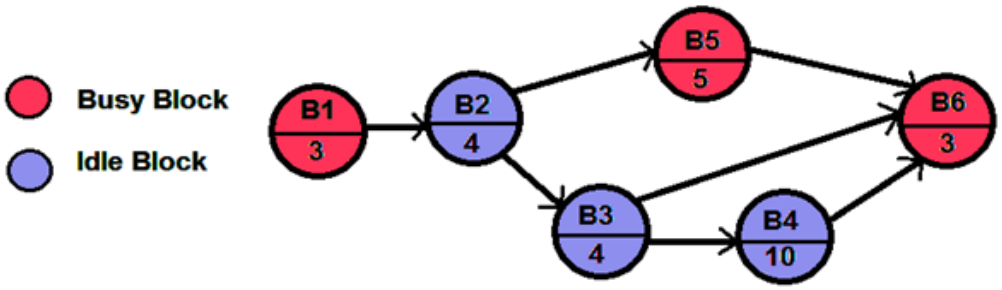
The proposed code based solution for the application of power gating for functional units uses the analyses of the control flow graph (CFG) and inserts power gating instructions according to it. More precisely, based on profiling, we obtain information about the conditional branches and/or loops to evaluate the best locations for power gating instructions,

taking into account the impact on performance and power savings. The problem of inserting power gating instructions can be reduced to finding the optimal locations for OFF instructions (which disables a functional unit) and ON instructions (which wakes up a functional unit) to maximize the energy savings. With this purpose we build the CFG of a program, which is a data structure that comprises the transition probabilities, the percentage of use of each FU, and the amount of cycles; always considering each basic block separately. An example of a CFG obtained via dynamic profiling is depicted in Figure 6.1. There are paths that do not use the current functional unit during a specific time interval, like the one composed of basic blocks B2, B3 and B4. The transition probability from B2 to B3 is 70%, whereas from B3 to B4 is 90%. If we are interested in knowing the expected number of cycles that the FU could be put to sleep if we insert an OFF instruction at the beginning of B2, we must take into account this information. This number will comprise the number of instructions in B2 (since we are assuming that the OFF instruction is processed), plus the expected number of idle cycles due to the transition to B3 (This will be the amount of cycles for B2 weighted by its transition probability of 70%), plus the expected number of idle cycles resultant from the transition from B3 to B4. In this case, the expected number of idle cycles is the number of instructions of B4 weighted by the probability of this specific path B2-B3-B4: the transition probability from B2 to B3, which is 70%, multiplied by the transition probability from B3 to B4, which is 90%.

Therefore, we can obtain the following expected number of cycles for an OFF instruction inserted at the beginning of B2:

$$T = 4+(0.7 *4) + (0.7 * 0.9 *10) = \mathbf{13,1}$$

Figure 6.1 - CFG of a program. Each circle is a Basic Block with an identifier and the number of instructions.



This value must be higher than the technological break-even point to obtain positive power savings, since otherwise the amount of energy used to disable the FU will be greater than the saved energy. Taking into account this example and using a technological break-even point equals to 10 cycles (which is consistent with the technology parameters used as was described in Chapter II) we can see that inserting an OFF instruction at the top of the path and an ON instruction at the final of this one would generate energy savings.

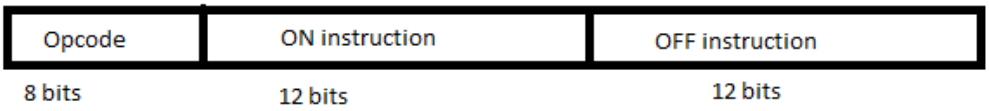
We can generalize this average path calculation to any basic block. The implementation of this function is made recursively, obtaining the average number of idle cycles by weighting the contribution of the average number of idle cycles of each path by its transition probability. We can express the last statement in a recursive mathematical expression:

$$T = P_1 T_1 + P_2 T_2$$

Where P_1 and P_2 are the transition probabilities of the path 1 and path 2 respectively, and T_1 and T_2 are the average number of idle cycles following the path 1 and 2, respectively. If one path uses the FU, T_1 or T_2 will be equal to zero, since there are no idle cycles in that path. In this work, this algorithm is applied for all the basic blocks and for each CFG associated with each FU.

The additional customized instruction for power gating FUs has a standard layout which encodes all the necessary information to disable and enable the required units. 2 bits are necessary to control each FU (3 possibilities: disable, enable, normal operation). Since each VLIW instruction is composed of a set of syllables of 32 bits each, it is possible to encode more power gating directives into a unique word, so different functional units can be disabled and enabled by only processing one power gating instruction. The 8 most significant bits are used as opcode (i.e., indicate that the current instruction is for power gating). The remaining 24 bits were divided into two groups of 12 bits each (Figure 6.2). The Most Significant Bits (MSB) are responsible for the ON operations and the least significant bits (LSB) for the OFF operations. Since there are 12 functional units, each bit of the MSB part and each of the LSB part is associated with a specific functional unit.

Figure 6.2 - Power gating instruction for the FUs.



Although the technique proposed is based on software directives to apply power gating, it is also necessary some hardware support to execute the additional customized instructions. Given that the analyses of the CFG is carried out before execution, the extra-logic is drastically reduced. A status register of 12 bits was added to the ρ-VEX processor to hold the state of each functional unit (ON or OFF), called, in this work, of Power Status Register (PSR). Additional logic was added to modify the PSR according to the power gating instruction processed. Power Gating Circuitry, using the same principle as headers [18], uses the information of the PSR to enable or disable the FUs according to the encoded bits. Verification monitors were implemented to count the number of saved cycles for each execution unit as well as the cycles overhead due to wake up instructions. The cycles overhead counter is increased each time a FU is required for execution but it is disabled at that point. For instance, if a new basic block is processed and one instruction belonging to this one requires the use of a multiplier that is disabled, the counter is augmented with the value of needed wake-up cycles (3 cycles taking into account current technology HU et al. (2004)).

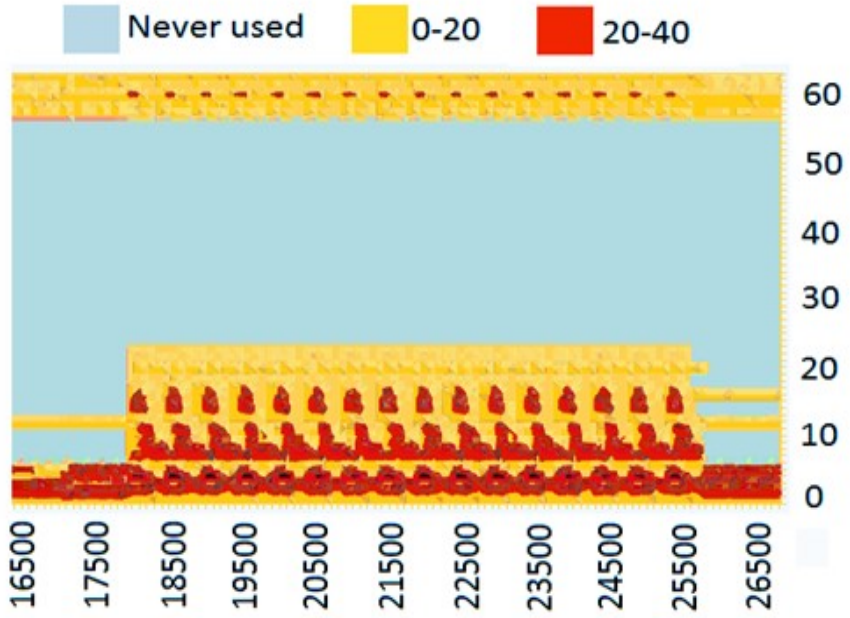
The hardware complexity associated with the decoding process of power gating instructions is marginal with our scheme. The information about which FUs must be enabled or disabled is encoded directly in the instruction with almost zero decoding overhead. However, it is important to note that the insertion of a power gating instruction in a basic block is limited by the availability of issue slots. In the proposed methodology, the power gating instructions are inserted at the first instruction of each basic block, so at least one syllable must be available there (i.e., the word must have at least one NOP operation). For narrower issue-width processors, whose availability of unused slots is reduced, such behavior could be a source of performance losses because an additional VLIW instruction would need to be inserted.

6.1.2 For the Register File

Since all the applications demand different amounts of resources, the overall use of the RF is dependent on the program that is being executed by the processor. For instance, Figure 6.3 depicts the register file use along time for a typical WCET benchmark. The y-axis represents each register and the x-axis represents time in terms of cycles. It was calculated the amount of references for each register in intervals of 100 cycles. If one register is used for an arithmetic/logic operation, branch operation, load/store operation, it is counted like a reference for that register. This information is encoded in the gray scale of each portion of the graph. The most part of the register file use is depicted in flat clear gray, which represents the registers that are never used along the execution of the program. The active registers are

clustered in contiguous areas of the RF. These are allocated in groups that are positioned in the most and least significant indexes of the RF (i.e., close to r0 and r63). As can be observed, the usage rate of each register use can be constant for a long period of time, while the behavior of the register file can be correlated with the phases of the program SHERWOOD et al. (2003a). This means that big portions of the hardware can be disabled in case that some registers are not needed for computational purposes.

Figure 6.3- Number of references for each register using windows of 100 instructions for *ndes*. The y axis represents each register that is part of the RF.



Therefore, we can apply power gating to the registers that are not used along the execution. To implement this functionality in hardware, we divided the register file into 8 groups of 8 registers. The enabling/disabling process of each group is managed statically by power gating instructions that informs the groups of registers that are going to be used for each program. The power gating instructions are inserted in the first instructions of the application taking into account the maximum RF use. The format of the customized instruction can be seen in Figure 6.4. To simplify decoding, 8 bits controls the activation of the RF blocks and other 8 bits the deactivation of them.

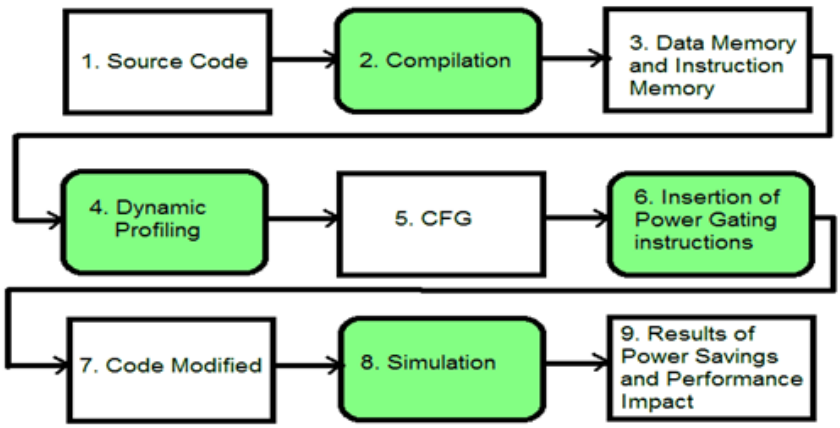
Figure 6.4 - Power gating instruction for the RF.

6.2 Methodology

The following applications were evaluated: *adpcm, crc, dft, fir, matrix, mm40* and *ndes,* which belong to the WCET benchmark set. The methodology flow is depicted in Figure 6.5. Benchmarks are compiled with the VEX compiler from HP labs (Step 1). The generated instruction and data memory files are loaded to the ρ-VEX processor, so it is possible to obtain full traces of the execution, using the Modelsim Software. Dynamic profiling was used to obtain the CFG of each application (Step 4 and 5). Each CFG is represented through a database that contains the necessary information for each basic block, as depicted in Section IV.

Then, the CFG is analyzed to generate all the power gating instructions and to insert them at the best possible locations, as shown in the last section (Step 6). It was implemented using ANSI C. As it is automatically done at the binary level, there is no need for recompilation or any modifications at the source code. Finally, the modified benchmark, with the additional power gating instructions, is ready to execute on the ρ-VEX processor (Step 8) and therefore it is possible to measure power savings and performance overheads (Step 9).

Figure 6.5 - Diagram flow for Profile-Based power gating. Each gray box is a step (process) while the rectangles are data obtained from the last step.

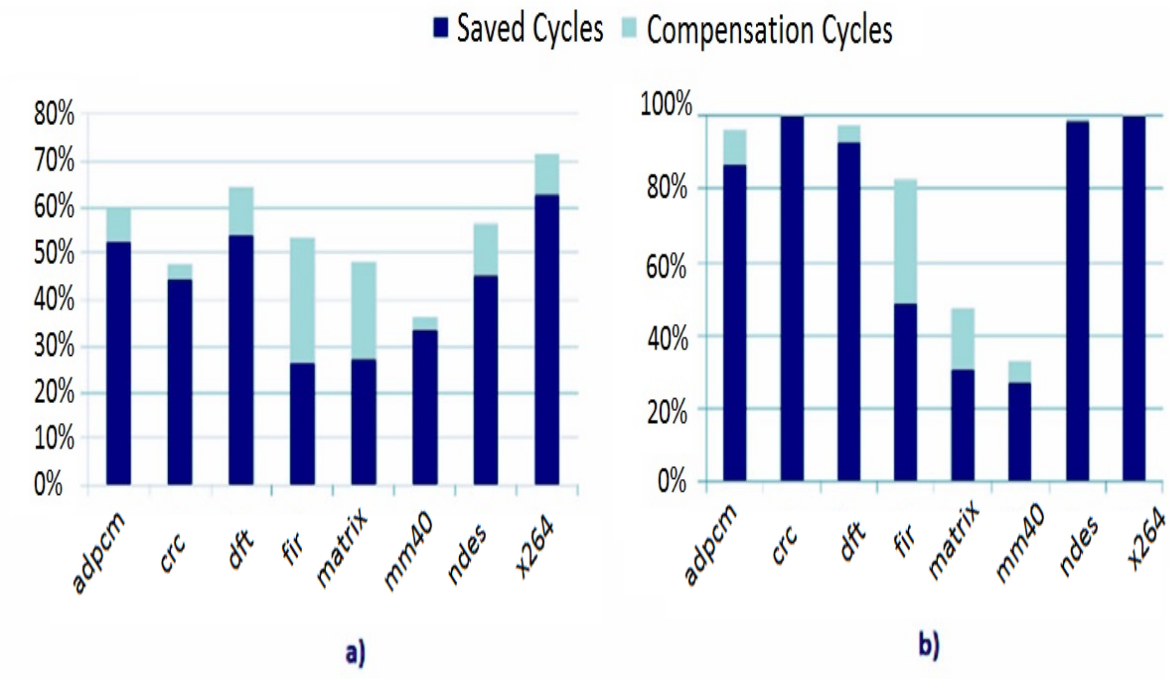


6.3 Experimental Results

The fraction of cycles spent in sleep mode by an execution unit of a given type for each benchmark simulated is depicted in Figure 6.6. It is observed that the total number of cycles that the functional units can be disabled is significant: 54,63% for ALUs and 81,90% for multipliers, on average. It is important to remember that to achieve positive power savings from power gating, there is a period of time needed to compensate the energy overhead, which we call compensation cycles. The subtraction between the total idle time that the FUs remain disabled and the compensation cycles is the effective period of time that the block is saving energy. Taking into account this behavior, the effective time that the FUs are saving energy is 43,11% for ALUs and 72,93% for multipliers.

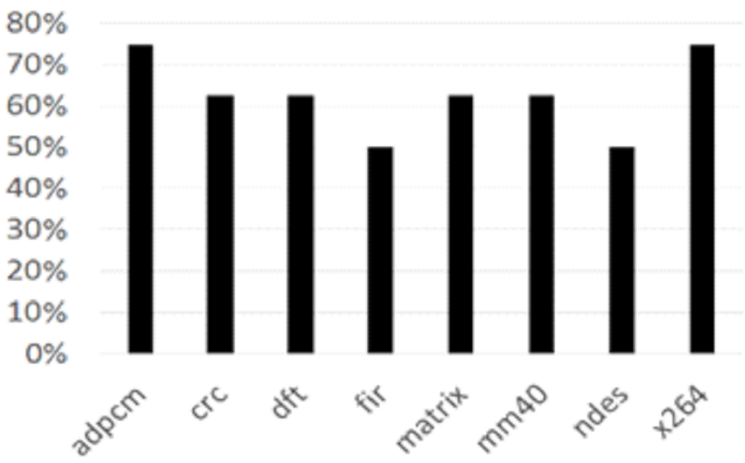
Some applications, like *crc* and *mm40*, show a small proportion of compensation cycles, which means that the idle periods are relatively long so the compensation cycles are not very significant. On the other hand, others, like *fir* and *matrix*, spend more cycles in compensation mode since there are a big quantity of short idle periods (i.e., the OFF/ON process is repeated many times). For some programs, like *ndes* and *x264*, the amount of compensation cycles for the multipliers is near to zero, since they are not used along program execution. Therefore, they can be disabled at the beginning of the execution and so the turn-off process is only carried out one time. In these cases, the effective energy savings for the multiplier units is almost 100%. Moreover, it can be seen that the use of functional units depends heavily on the application: *crc* does not use multipliers extensively; whereas the *mm40* presents the opposite behavior, so idle periods in such cases are not so common. However, in overall, the amount of saved cycles from the multipliers is significantly greater than for the ALUs for all the benchmarks, since they are usually less used (even though their presence is obviously necessary).

Figure 6.6 - Number of cycles that the FUs are disabled through power gating for a) ALUs and b) Multipliers. The dark blue portion represents the amount of cycles that saves energy, whereas the clear bright bluw portion represents the number of cycles that are needed to compensate the energy overhead derived from power gating.



The total cycles in the sleep mode for the blocks in the RF for each benchmark simulated is depicted in Figure 6.7. The amount of number of cycles that the registers can be disabled is 74,38%, on average. As can be observed, the difference in terms of power savings for each benchmark is significantly different, since there are some programs like *fir* that uses more blocks of registers along the execution than others, like *adpcm*.

Figure 6.7 - Number of cycles that the RF is disabled through power gating.



The performance losses were also calculated for each application (Figure 6.8), with an average value of 8.64%. The performance overhead is a consequence of how many times an instruction is processed and whether the needed functional units are ready or not for executing the operations at a given moment. Any time that this situation occurs (the functional unit has not been completely awake to execute an instruction that is ready), the system must stall and wait until the wake-up process is completed. Clearly, the performance impact is not equal for all the benchmarks. Some of them, such as *fir* and *matrix*, are more affected. This situation arises when there are more recurring wake-up processes in the execution of an application, which increase the total execution time. One of the causes of this behavior is in cases there is a big difference in the type or number of resources demanded by basic blocks that are sequentially executed. For example, if one basic block uses 2 ALUs, and another basic block that is most of time executed after this one uses 5 ALUs, the wake-up process of 3 ALUs will be carried out as this path is processed. This will result in performance degradation, even though energy is saved - and our approach focus on the latter. However, if there are practical limits on performance losses for an application, performance constraints can be added to our algorithm.

Figure 6.8 - Performance reduction resulting from the insertion of power gating instructions.

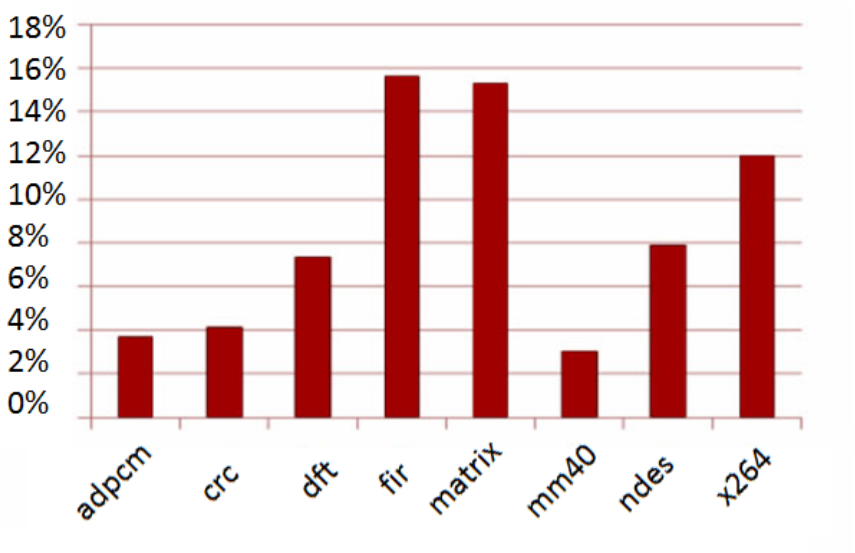
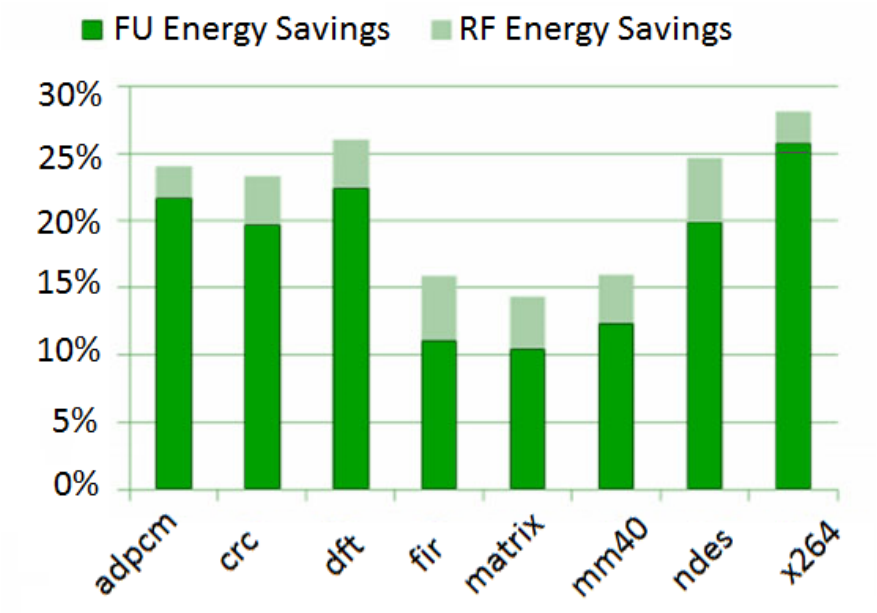


Figure 6.9 - Energy savings obtained through the insertion of power gating instructions. The dark green represents the contribution of the FUs and the bright green portion represents the RF contribution.



The energy savings due to the insertion of power gating instructions, considering the performance penalties, is depicted in Figure 6.9. This quantity of energy savings is considered taking into account the contribution of power gating applied to the RF and to the FUs. It is observed that the average savings in energy are about 20% of the total energy consumed by the original ρ-VEX processor. The range of variation is between 14,35% for *matrix* and 28,12% for *x264*. The same variability observed between benchmarks before can be extended to analyze this metric.

7 HARDWARE-SOFTWARE POWER GATING COMPARISON

The difference of power energy savings results between the simulations focused on the potential of optimization (Chapter IV) and the simulations focused on the application of power gating techniques (Chapter V and VI) are notorious and are caused by the methodologies and objectives of each approach. In chapter IV we found that, on average, the potential energy savings are 70% across the benchmarks simulated, whereas for the application of power gating of the functional units the results are 15% and 20% for a hardware-based and software-based approach, respectively. The approach showed in Chapter IV had as objective describing the energy savings that could be obtained if the complete microarchitecture of the processor was changed from one issue-width to another. It means that all the hardware units are changed including the specific configuration of the functional units, the register file, the load/store unit, etc. which are modules that are tailored for each issue-width. Since hardware reconfiguration always presents timing and power overhead, the adaptation of all the microprocessor would not be feasible in real situations at fine-granularity, so the results of this chapter can be seen as the upper-limit that an adaptable VLIW processor could achieve. Conversely, the presented power gating techniques in Chapter V and VI aimed to show the implementation of a VLIW processor that just apply the adaptation policy for the functional units. The results shown take into account the drawbacks derived from the application of the technique and are intended to present the real challenges that are faced with an adaptable VLIW microprocessor.

There are remarkable differences between the hardware and the software approach that are important to point out. In general, the former offers best results in terms of energy savings as well as less performance losses for almost all the benchmarks that were used along this work.

The compiler-based approach can put to sleep the functional units more times than the time-based approach. In Figure 7.1 and Figure 7.2, it is depicted the comparison for ALU

units and Multiplier units for each one of the benchmarks using the two aforementioned techniques. The threshold, the break-even point and the wakeup cycles of the time-based power gating were set to 21, 10, and 3, respectively, to obtain similar performance losses with the compiler-based power gating. In this way, we can compare the two frameworks by setting the same performance losses margins. It is observed that the average number of cycles that the functional units can be disabled is 54,63% for ALUs and 81,90% for multipliers through compiler-based approach; and 30% for ALUs and 63% for multipliers through time-based power gating. The results showed that with application binary analysis it is possible to detect shorter idle time periods that are overlooked by the time-based power gating. It is remarkable that for some benchmarks, like *fir* and *matrix,* the total sleep time through compiler-based approach is significantly larger. This means that in these benchmarks the long idle periods are not so common, and it is necessary a static or dynamic application analyses to take advantage of the presence of shorter idle periods inherent to the code. Moreover, the time that the multiplier units can be disabled is larger than for ALU units, since these functional units are used with less frequency. For some applications they can be put to sleep throughout all the execution time saving significant amounts of energy.

Figure 7.1 - Comparison of total sleep time for ALU units between Time-based Power Gating and Compiler-based approach.

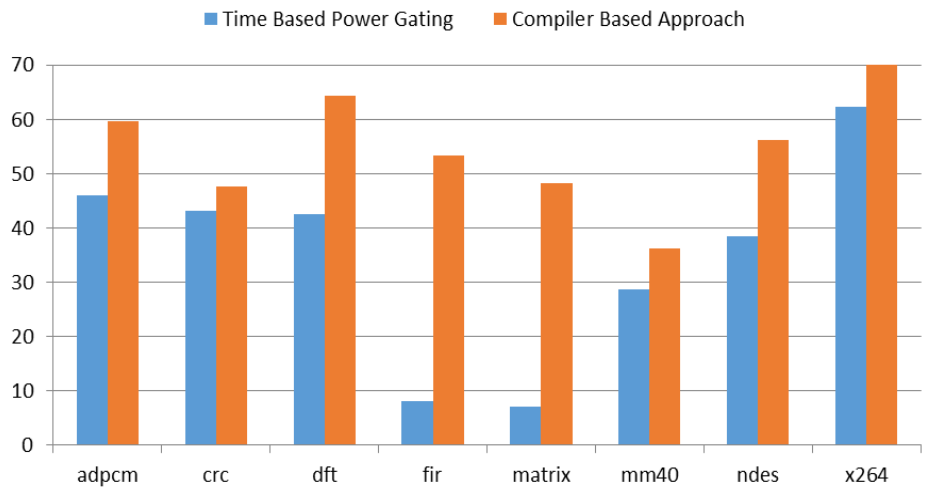
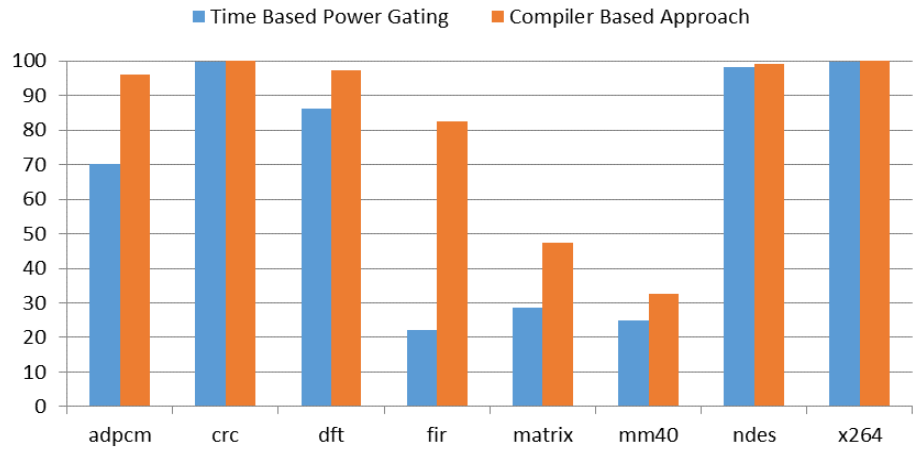
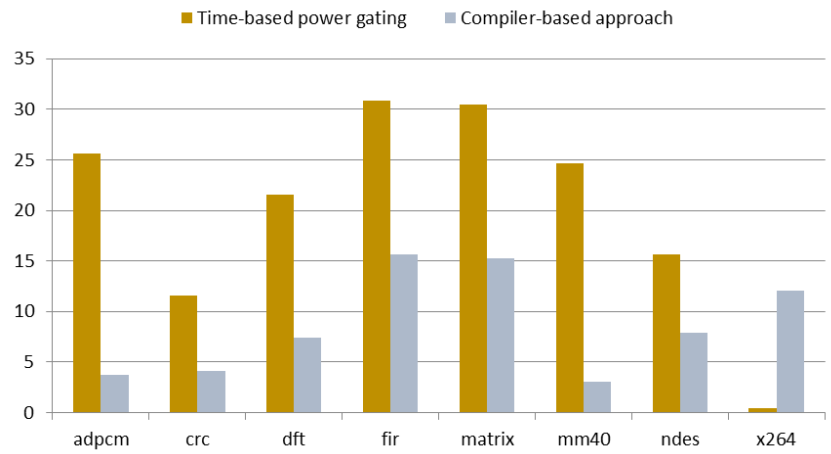


Figure 7.2 - Comparison of total sleep time for Multiplier units between Time-based Power Gating and Compiler-based approach.



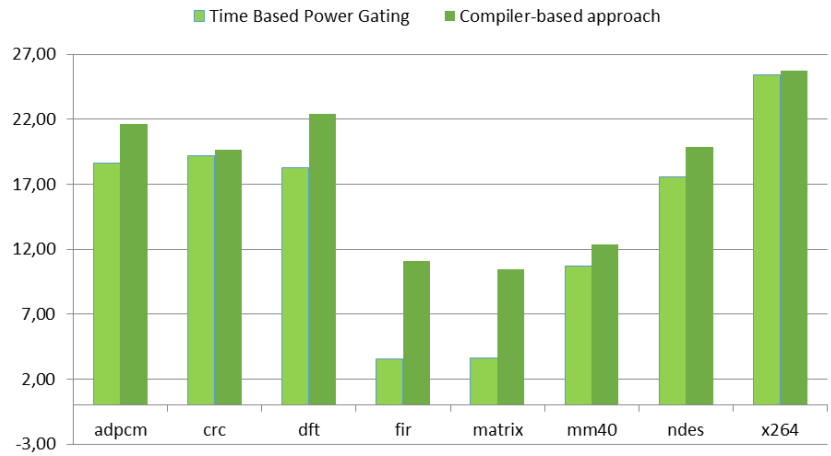
In Figure 7.3 it is shown the comparison of performance losses. This metric represents the reduction in IPC that it is expected regardless the employed technique. In the same way as before, the threshold, the break-even point and the wakeup cycles of the time-based power gating were set to 11, 10, and 3, respectively, to obtain similar power savings with the compiler-based power gating. The performance losses for the compiler-based approach are on average 8.64% and for the time-based approach 20,11%. In general, for almost all the benchmarks the reduction is about two or three times better for the software technique. This situation appears because by using application binary analyses we can set the necessary resources of each basic block before it begins its execution. In this way, often all the functional units that are required by a basic block are enabled when the instructions belonging to that basic block demand a specific resource. Conversely, for the time-based approach the functional units are enabled only when they are required by a new instruction. This lack of prediction is due to the dynamic nature of the hardware approach in contrast with the static compiler-based technique, producing a significant timing overhead. The only exception of this pattern is *x264*, which presents a larger performance overhead for the compiler-based power gating. This situation arises because this application presents many short idle periods which can be exploited for energy savings through the compiler technique. However, they are not long enough to obtain significant improvements in terms of energy consumption. Since the developed power gating instruction algorithm always put one power off instruction if there are potential positive energy savings, the performance could be affected without significant improvements for power consumption, which is what happens to this specific case.

Figure 7.3 - Comparison of performance losses between Time-based Power Gating and Compiler-based approach.



In Figure 7.4 it is depicted the comparison of energy savings using the same methodology as before for the threshold, break-even point and the wakeup cycles, to obtain similar performance losses with the compiler-based power gating as was made for the Figure 7.1 and Figure 7.2. We can see a link between those figures with Figure 7.4 since the calculations for energy savings were made based on the amount of sleep time cycles of the functional units. For all the benchmarks, the energy savings are larger for the compiler-based approach with varying degrees of improvements. Despite the difference in energy savings between the two approaches, this value is not so large (only 5%). The reason behind this result could be the presence of disabled functional units that are inactive independent of the applied methodology. For example, since the multipliers are not extensively used in some benchmarks, both software and hardware approach are able to disable them, so the net energy savings difference between the two methodologies is reduced.

Figure 7.4 - Comparison of energy savings between Time-based Power Gating and Compiler-based approach.



8 CONCLUSIONS

This thesis has presented the motivation, implementation challenges and potential benefits of an adaptable VLIW processor for energy efficiency. The overall results show that this kind of design is feasible and the improvements in relation to previous approaches is significant.

The first part of the work has covered the most recent developments in adaptable computer architecture, specifically those focused on energy efficiency optimization. It was noted that this kind of designs offer a great variety of advantages over their static counterparts since the resources are handled dynamically. On the other hand, often the design complexity is increased and the overhead in terms of energy and performance must be taken into account into the final solution.

We have then detailed the problems that arise with traditional static VLIW microprocessors and explained how these issues could be addressed through the implementation of an adaptable computing system. The impact in area, performance and power of an adaptable issue-width processor was obtained. The potential energy savings for a dynamic issue-width VLIW processor were calculated and two adaptation policies were implemented: coarse granularity and fine granularity. The results showed that the potential energy savings could be as high as 80%, which remarks the beneficial impact that an adaptable behavior could bring to a VLIW processor.

Having in mind this scenario, we proposed two different techniques for the implementation of an adaptable VLIW processor: a hardware-based and a software-based power gating solution. The first one was based on the availability of hardware counters to detect long idle periods for the functional units and the use of power gating circuitry to disable the resources when these inactive traces are detected. The results showed that the functional units can be put to sleep for 63% of the total execution time for the multiplier units and 30% for the ALUs with a penalty of 20% in performance. The second solution proposed the insertion of customized instructions into the VLIW code to handle the turn on and turn off

of the functional units via power gating. Since this approach moves away the computational efforts from the hardware towards the software, the hardware design overhead is decreased and the power consumption of such adaptable processor is lower. The obtained results showed that the ALU units can be disabled 54% of the total execution time and the Multiplier units 81% following this policy. The total energy savings are on average 20% with an impact on performance near to 8%.

8.1 Future Work

The conclusions obtained from this dissertation open a wide range of research topics that could be source of new projects in the future work.

The application of power gating might be suited to other datapath structures in the same way that these techniques were applied to the functional units and the register file. For example, the same as the Multiplier units and the ALU units present long idle periods, modules of a cache memory are used selectively and present idleness depending on the application. It means that, a significant part of the cache is inactive during specific periods of time wasting dynamic and static power consumption. This phenomenon is dynamic throughout the execution time. A compiler-based approach could use static or dynamic profiling to detect the expected cache use of each phase, and hence be able to insert customized instructions to control the availability of idle circuits via power gating.

The greedy nature of the algorithm used for power gating instruction insertion presents some drawbacks that can be handled with improvements for this compiler technique. As was observed, there are some programs like x264 that have some idle paths in the CFG of some functional units, that are long enough to obtain positive energy savings but that are not significantly long to justify the performance losses. For example, using 10 cycles for the break-even point any idle path with just 11 cycles could be exploited to disable the functional unit. Whereas there will be energy savings since it is greater than the minimum cycles required, the performance losses could be higher if the wakeup of the functional unit is continuously required. To manage this situation, the influence of the performance impact could be inserted into the algorithm to avoid disabling the functional unit when these kind of recurring wake-up processes are present.

The application of power gating and clock gating together is an interesting research topic that could be addressed in future work. As was observed along this Master Thesis, these two techniques are orthogonal since the application of one of them does not constraint the

application of the other one. Since clock gating could be applied cycle by cycle, power gating could be used selectively to take advantage of longer idle periods.

REFERENCES

ALBONESI, David H. Selective cache ways: On-demand cache resource allocation. En Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on. IEEE, 1999. p. 248-259.

ANNAVARAM, Murali; GROCHOWSKI, Ed; SHEN, John. Mitigating Amdahl's law through EPI throttling. En Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on. IEEE, 2005. p. 298-309.

BALAKRISHNAN, Saisanthosh, et al. The impact of performance asymmetry in emerging multicore architectures. En ACM SIGARCH Computer Architecture News. IEEE Computer Society, 2005. p. 506-517.

BECK, Antonio Carlos Schneider; LISBÔA, Carlos Arthur Lang; CARRO, Luigi. Adaptable embedded systems. Springer Science & Business Media, 2012.

BOLZANI, Leticia, et al. Enabling concurrent clock and power gating in an industrial design flow. En Proceedings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, 2009. p. 334-339.

BORKAR, Shekhar. Design challenges of technology scaling. Micro, IEEE, 1999, vol. 19, no 4, p. 23-29.

CADENCE ENCOUNTER, R. T. L. Compiler v. 8.10. Available at: www. cadence. com. Accessed November, 2015, vol. 11.

CARDOSO, João MP; DINIZ, Pedro C.; WEINHARDT, Markus. Compiling for reconfigurable computing: A survey. ACM Computing Surveys (CSUR), 2010, vol. 42, no 4, p. 13.

CHANDRAKASAN, Anantha P.; BRODERSEN, Robert W. Low power digital CMOS design. Springer Science & Business Media, 2012.

CHANG, Chin-Hao; LIU, Pangfeng; WU, Jan-Jan. Sampling-based phase classification and prediction for multi-threaded program execution on multi-core architectures. En Parallel Processing (ICPP), 2013 42nd International Conference on. IEEE, 2013. p. 349-358.

CLARK, Lawrence T., et al. Standby power management for a 0.18 μm microprocessor. En Proceedings of the 2002 international symposium on Low power electronics and design. ACM, 2002. p. 7-12.

COLWELL, Robert. The chip design game at the end of Moore's law. En 2013 IEEE Hot Chips 25 Symposium (HCS). IEEE, 2013. p. 1-16.

DHODAPKAR, Ashutosh S.; SMITH, James E. Comparing program phase detection techniques. En Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2003. p. 217.

DHODAPKAR, Ashutosh S.; SMITH, James E. Managing multi-configuration hardware via dynamic working set analysis. En Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on. IEEE, 2002. p. 233-244.

DROPSHO, Steve, et al. Managing static leakage energy in microprocessor functional units. En Microarchitecture, 2002.(MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on. IEEE, 2002. p. 321-332.

DUNN, Darrell. The best and worst cities for data centers. InformationWeek, 2006, vol. 23.

DWARKADAS, Sandhya, et al. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. U.S. Patent No RE41,958, 23 Nov. 2010.

EMNETT, Frank; BIEGEL, Mark. Power reduction through RTL clock gating. SNUG, San Jose, 2000.

FISHER, Joseph A.; FARABOSCHI, Paolo; YOUNG, Cliff. Embedded computing: a VLIW approach to architecture, compilers and tools. Elsevier, 2005.

FL, Antonio Carlos Schneider Beck; CARRO, Luigi. Dynamic Reconfigurable Architectures and Transparent Optimization Techniques: Automatic Acceleration of Software Execution. Springer Science & Business Media, 2010.

FLAUTNER, Krisztián, et al. Drowsy caches: simple techniques for reducing leakage power. En Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on. IEEE, 2002. p. 148-157.

GEUSKENS, Bibiche; ROSE, Kenneth. Modeling microprocessor performance. Springer Science & Business Media, 2012.

GHIASI, Soraya; GRUNWALD, Dirk. Thermal management with asymmetric dual core designs. Dept. Comput. Sci., Univ. Colorado, Boulder, Tech. Rep. CU-CS-965-03, 2003.

GUSTAFSSON, Jan, et al. The Mälardalen WCET benchmarks: Past, present and future. En OASIcs-OpenAccess Series in Informatics. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

GUTHAUS, Matthew R., et al. MiBench: A free, commercially representative embedded benchmark suite. En Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on. IEEE, 2001. p. 3-14.

HEMMERT, Scott. Green hpc: From nice to necessity. Computing in Science & Engineering, 2010, vol. 12, no 6, p. 8-10.

HIND, Michael J.; RAJAN, Vadakkedathu T.; SWEENEY, Peter F. Phase shift detection: A problem classification. Technical report, IBM, 2003.

HU, Zhigang, et al. Microarchitectural techniques for power gating of execution units. En Proceedings of the 2004 international symposium on Low power electronics and design. ACM, 2004. p. 32-37.

HUANG, Michael C.; RENAU, Jose; TORRELLAS, Josep. Positional adaptation of processors: application to energy reduction. En Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on. IEEE, 2003. p. 157-168.

IPEK, Engin, et al. Core fusion: accommodating software diversity in chip multiprocessors. En ACM SIGARCH Computer Architecture News. ACM, 2007. p. 186-197.

ISCI, Canturk, et al. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. En Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture. IEEE Computer Society, 2006. p. 347-358.

ISCI, Canturk; BUYUKTOSUNOGLU, Alper; MARTONOSI, Margaret. Long-term workload phases: Duration predictions and applications to DVFS. Micro, IEEE, 2005, vol. 25, no 5, p. 39-51.

JEFF, Brian. Big. LITTLE system architecture from ARM: saving power through heterogeneous multiprocessing and task context migration. En Proceedings of the 49th Annual Design Automation Conference. ACM, 2012. p. 1143-1146.

JIANG, Hailin; MAREK-SADOWSKA, Malgorzata; NASSIF, Sani R. Benefits and costs of power-gating technique. En Computer Design: VLSI in Computers and Processors, 2005.

ICCD 2005. Proceedings. 2005 IEEE International Conference on. IEEE, 2005. p. 559-566.

KĘDZIERSKI, Kamil, et al. Power and performance aware reconfigurable cache for CMPs. En Proceedings of the Second International Forum on Next-Generation Multicore/Manycore Technologies. ACM, 2010. p. 1.

KOUFATY, David; REDDY, Dheeraj; HAHN, Scott. Bias scheduling in heterogeneous multi-core architectures. En Proceedings of the 5th European conference on Computer systems. ACM, 2010. p. 125-138.

KUMAR, Rakesh, et al. Heterogeneous chip multiprocessors. Computer, 2005, no 11, p. 32-38.

KUMAR, Rakesh, et al. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. ACM SIGARCH Computer Architecture News, 2004, vol. 32, no 2, p. 64.

KUMARI, Khushboo, et al. Review of Leakage Power Reduction in CMOS Circuits. American Journal of Electrical and Electronic Engineering, 2014, vol. 2, no 4, p. 133-136.

LI, Hai, et al. DCG: deterministic clock-gating for low-power microprocessor design. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2004, vol. 12, no 3, p. 245-254.

LI, Hai, et al. Deterministic clock gating for microprocessor power reduction. En High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on. IEEE, 2003. p. 113-122.

LI, Jian; MARTINEZ, Jose F. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. En High-Performance Computer Architecture, 2006. The Twelfth International Symposium on. IEEE, 2006. p. 77-87.

LIAO, Weiping; BASILE, Joseph M.; HE, Lei. Leakage power modeling and reduction with data retention. En Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design. ACM, 2002. p. 7 WANG, Meng, et al. Real-time loop scheduling with leakage energy minimization for embedded VLIW DSP processors. En null. IEEE, 2007. p. 12-19.14-719.

LYSECKY, Roman; STITT, Greg; VAHID, Frank. Warp processors. En ACM Transactions on Design Automation of Electronic Systems (TODAES). ACM, 2004. p. 659-681.

MARKOFF, John; HANSELL, Saul. Hiding in plain sight, Google seeks more power. New York Times, 2006, vol. 14, p. 1-2.

MIYAMORI, Takashi; OLUKOTUN, U. A quantitative analysis of reconfigurable coprocessors for multimedia applications. En FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on. IEEE, 1998. p. 2-11.

MOORE, Chuck. Data processing in exascale-class computer systems. In: The Salishan Conference on High Speed Computing. 2011.

NIEDERMEIER, Anja, et al. The challenges of implementing fine-grained power gating. En Proceedings of the 20th symposium on Great lakes symposium on VLSI. ACM, 2010. p. 361-364.

PARK, Danbee, et al. Optimal algorithm for profile-based power gating: A compiler technique for reducing leakage on execution units in microprocessors. En Proceedings of the International Conference on Computer-Aided Design. IEEE Press, 2010. p. 361-364.

RAKHMATOV, Daler; VRUDHULA, Sarma. Energy management for battery-powered embedded systems. ACM Transactions on Embedded Computing Systems (TECS), 2003, vol. 2, no 3, p. 277-324.

RELE, Siddharth, et al. Optimizing static power dissipation by functional units in superscalar processors. En Compiler Construction. Springer Berlin Heidelberg, 2002. p. 261-275.

ROY, Soumyaroop; RANGANATHAN, Nagarajan; KATKOORI, Srinivas. A framework for power-gating functional units in embedded microprocessors. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2009, vol. 17, no 11, p. 1640-1649.

SAHA, Dipankar, et al. Implementation of the Cluster Based Tunable Sleep Transistor Cell Power Gating Technique for a 4x4 Multiplier Circuit. arXiv preprint arXiv:1310.3203, 2013.

SANKARALINGAM, Karthikeyan, et al. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. En Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on. IEEE, 2003. p. 422-433.

SHERWOOD, Timothy, et al. Automatically characterizing large scale program behavior. ACM SIGOPS Operating Systems Review, 2002, vol. 36, no 5, p. 45-57.

SHERWOOD, Timothy, et al. Discovering and exploiting program phases. Micro, IEEE, 2003, vol. 23, no 6, p. 84-93.

SHERWOOD, Timothy; SAIR, Suleyman; CALDER, Brad. Phase tracking and prediction. En ACM SIGARCH Computer Architecture News. ACM, 2003. p. 336-349.

SHIN, Youngsoo, et al. Power gating: Circuits, design methodologies, and best practice for standard-cell VLSI designs. ACM Transactions on Design Automation of Electronic Systems (TODAES), 2010, vol. 15, no 4, p. 28.

SMIT, Gerard JM, et al. Lessons learned from designing the MONTIUM-a coarse-grained reconfigurable processing tile. 2004.

UCHIDA, Mitsuya, et al. Energy-aware SA-based instruction scheduling for fine-grained power-gated VLIW processors. En SoC Design Conference (ISOCC), 2012 International. IEEE, 2012. p. 139-142.

VILLA, Oreste, et al. Scaling the power wall: a path to exascale. En Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, 2014. p. 830-841.

WONG, Stephan; VAN AS, Thijs; BROWN, Geoffrey. ρ-VEX: A reconfigurable and extensible softcore VLIW processor. En ICECE Technology, 2008. FPT 2008. International Conference on. IEEE, 2008. p. 369-372.

X-FAB, 0.18 micron modular cmos technology. Available at: www.xfab.com/technology/cmos/018-um-xc018/. Accessed November, 2015.

YANG, Se-Hyun, et al. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches. En High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on. IEEE, 2001. p. 147-157.

YE, Zhi Alex, et al. CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit. ACM, 2000.

YEO, L. C., et al. Dynamic power gating implementation on intel embedded media and graphics driver. Intel Corporation, White Paper, 2011, vol. 325293.

YOU, Yi-Ping; LEE, Chingren; LEE, Jenq Kuen. Compilers for leakage power reduction. ACM Transactions on Design Automation of Electronic Systems (TODAES), 2006, vol. 11, no 1, p. 147-164.

ZHONG, Hongtao; LIEBERMAN, Steven A.; MAHLKE, Scott A. Extending multicore architectures to exploit hybrid parallelism in single-thread applications. En High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on. IEEE, 2007. p. 25-36.