Compiler-Assisted Energy Optimization for Clustered VLIW Processors

A Thesis

Submitted for the Degree of

Doctor of Philosophy

in the Faculty of Engineering

by

Rahul Nagpal

Department of Computer Science and Automation

Indian Institute of Science

Bangalore – 560 012, India

March 2008

सर्व मंगल मांगल्ये शिवे सर्वार्थ साधिके ।
शरण्ये त्र्यम्बके गौरि नारायणि नमोऽस्तु ते ॥

Abstract

Clustered architecture processors are preferred for embedded systems because centralized register file architectures scale poorly in terms of clock rate, chip area, and power consumption. Although clustering helps by improving clock speed, reducing energy consumption of the logic, and making the design simpler, it introduces extra overheads by way of inter-cluster communication. This communication happens over long wires having high load capacitance which leads to delay in execution and significantly high energy consumption. Inter-cluster communication also introduces many short idle cycles, thereby significantly increasing the overall leakage energy consumption in the functional units. The trend towards miniaturization of devices (and associated reduction in threshold voltage) makes energy consumption in interconnects and functional units even worse and limits the usability of clustered architectures in smaller technologies.

In the past, study of leakage energy management at the architectural level has mostly focused on storage structures such as cache. Relatively, little work has been done on architecture level leakage energy management in functional units in the context of superscalar processors and energy efficient scheduling in the context of VLIW architectures. In the absence of any high level model for interconnect energy estimation, the primary focus of research in the context of interconnects has been to reduce the latency of communication and evaluation of various inter-cluster communication models. To the best of our knowledge, there has been no such work in the past from the point of view of energy efficiency targeting clustered VLIW architectures specifically focusing on smaller technologies.

Technological advancements now permit design of interconnects and functional units with varying performance and power modes. In this thesis, we propose scheduling algorithms that aggregate the scheduling slack of instructions and communication slack of data values to exploit the low power modes of interconnects and functional units. We also propose a high level model for estimation of interconnect delay and energy (in contrast to low-level circuit level model proposed earlier) that makes it possible to carry out architectural and compiler optimizations specifically targeting the interconnect. Finally, we present synergistic combination of these algorithms that simultaneously saves energy in functional units and interconnects to improve the usability of clustered architectures by achieving better overall energy-performance trade-offs.

Our compiler assisted leakage energy management scheme for functional units reduces the energy consumption of functional units approximately by 15% and 17% in the context of a 2-clustered and a 4-clustered VLIW architecture respectively with negligible performance degradation over and above that offered by a hardware-only scheme. The interconnect energy optimization scheme improves the energy consumption of interconnects on an average by 41% and 46% for a 2-clustered and a 4-clustered machine respectively with 2% and 1.5% performance degradation. The combined scheme obtains slightly better energy benefit in functional units and 37% and 43% energy benefit in interconnect with slightly higher performance degradation. Even with the conservative estimates of contribution of functional unit and interconnect to overall processor energy consumption, the proposed combined scheme obtains on an average 8% and 10% improvement in overall energy-delay product with 3.5% and 2% performance degradation for a 2-clustered and a 4-clustered machine respectively. We present a detailed experimental evaluation of the proposed schemes using the Trimaran compiler infrastructure.

Acknowledgments

I would like to express my deep gratitude to my research supervisor, Prof. Y. N. Srikant, for providing me an opportunity to work under his supervision. Prof. Y. N. Srikant is an excellent adviser who is always willing to listen, encourage, and give insightful comments and valuable criticism. This dissertation would have been a dream for me without his support, inspiration, and constant encouragement.

I would like to thank Prof. Amrutur Bhardwaj for many important technical discussions especially towards end of my thesis. I am also thankful to the faculty of the Department of Computer Science and Automation, Prof. Priti Shankar, Prof. Narahari, Prof. Narasimha Murthy, and Prof. Mathew Jacob in particular, for clarifying doubts, refining ideas, and for the constant encouragement.

Particular thanks go to the Philips Research for funding the part of this research work. Thanks also to Microsoft Research Labs, Redmond, USA for providing me an exciting period of 3 moths of internship which helped me to understand the industry style of research.

My heartful thanks to my dear lab mates at compiler lab who have provided wonderful company and stimulating environment; without them this process would have been much difficult. In particular, I would like to thank Arun and Anand Vardhan. Many thanks to the lab staff of the Compiler Laboratory, Pushpraj in particular, for their support and assistance. I would also like to thank my other friends in IISc for a constant source of mild distraction that prevented me from burning out during the course of my research.

I can not thank my parents enough for their countless sacrifices. They have constantly talked me out of worries and it is due to their support and encouragement that I could face most difficult times and still could come this far.

Finally, Thanks to all who supported me directly or indirectly in this period of my life.

Publications Based on the Thesis

Conferences

1. Rahul Nagpal and Y. N. Srikant. Compiler-assisted Leakage Energy Optimization for Clustered VLIW Architectures, Proceedings of the International Conference on Embedded Software (EMSOFT'06), Seoul, Korea, October 2006.

2. Rahul Nagpal and Y. N. Srikant. Exploring Energy-Performance Trade-offs for Heterogeneous Interconnect Clustered VLIW Processors, Proceedings of the International Conference on High Performance Computing (HiPC'06), Bangalore, India, December 2006.

3. Rahul Nagpal, Arvind Madan, Amrutur Bharadwaj, and Y. N. Srikant. INTACTE: An Interconnect Area, Delay, and Energy Estimation Tool for Micro-architectural Explorations, Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'07), Salzburg, Austria, October 2007.

4. Rahul Nagpal and Y. N. Srikant. Register File Energy Optimization for Clustered VLIW Architectures, Proceedings of the International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07), Gramado, RS, Brazil October 2007.

5. Rahul Nagpal and Y. N. Srikant. Compiler-assisted Instruction Decoder Energy Optimization for Clustered VLIW Architectures, Proceedings of the International Conference on High Performance Computing (HiPC'07), Goa, India, December 2007.

Poster Presentations

1. Rahul Nagpal. Energy Efficient Cross-path Scheduling for Clustered VLIW Processors, Poster Presentation at International Conference on Language, Compilers, and Tool Support for Embedded Systems (LCTES'06), Ottawa, Canada, June 2006.

2. Rahul Nagpal and Y. N. Srikant. Energy Optimization for Clustered VLIW Architectures, Poster Presentation at Third International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES'07), L'Aquila, Italy, July 2007.

Journals

1. Rahul Nagpal and Y. N. Srikant. Compiler-Assisted Energy Optimization for Clustered VLIW Architectures (under Review)

2. Rahul Nagpal and Y. N. Srikant. Compiler-Assisted Power Optimization for Hot Spots in Clustered VLIW Architectures (under Review)

3. Rahul Nagpal, Arvind Madan, Amrutur Bharadwaj, and Y. N. Srikant. An Interconnect Area, Delay, and Energy Estimation Methodology and its application for Micro-architectural Explorations (under Preparation)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Embedded systems have proliferated in our daily lives in the form of wearable computers, telecommunication appliances, consumer electronic devices, system controllers, and entertainment objects among many others. An embedded system can be loosely defined as a *"System employing a collection of hardware and software to perform a closely knitted set of functions and often working in a reactive and a time-constrained environment as a part of a larger system"*. These systems are often employed to run multiple sophisticated algorithms demanding widely varying operation rates in hard or soft real-time mode. For example in a cellular phone, the speed of decoding must match that of normal conversation to ascertain a proper communication. Likewise, the controller of an atomic reactor should monitor several fluctuating parameters and exercise various controlling actions in a spur of the moment to avoid disastrous situations. If deadlines are violated, the quality of service may degrade to an unacceptable level or may even lead to catastrophic results in the case of a hard real-time system. Small size and less power consumption are critical factors in the design of hand-held battery-operated embedded systems. In general, less power consumption is desirable in all kinds of embedded systems because it reduces the operating cost of a device. Small form-factor necessary for many handheld embedded devices coupled with the requirement for powerful computing capabilities makes the temperature and associated reliability problems (specially for safety critical applications) another important consideration in the design of embedded systems. Thus embedded systems can be characterized by very high performance demand (to operate in real-time), low power consumption, low temperature, low cost, small form-factor, and high reliability.

The increasing ubiquity of embedded systems has opened up many new research issues as the design challenges posed by these systems are ostensibly different from those offered by general purpose systems due to their specific and conflicting requirements. New architectures leveraging the improvement in semiconductor technology have been developed. These exploit the abundant fine-grained instruction-level-parallelism (ILP)[1] available in embedded applications to satisfy their high performance requirements. Superscalar architectures[2] and very long instruction word (VLIW)[3] architectures are two traditional ILP design philosophies[1]. Both superscalar and VLIW processors have multiple pipelined functional units, which are connected to a single unified register file in parallel to attain better performance. A superscalar processor[2] uses dedicated hardware for scheduling instructions at run time. However, it suffers from the problems such as complicated design, large chip-area, and high power consumption attributed to the complicated runtime scheduling logic[4].

A VLIW architecture[3] gets rid of scheduling hardware and associated problems by exposing instruction latencies and delegating the task of scheduling to a compiler. High operation rates as required by emerging real time embedded applications can be attained by increasing the number of parallel functional units in a VLIW architecture. However, as the number of arithmetic units in a processor increases to higher levels, register storage and communication between arithmetic units become critical factors dominating the area, cycle time, and power dissipation of the processor. The cycle time is determined by the sum of wire delay and gate delay along the critical execution paths of a processor pipeline. Wire delays have become significant for the 0.25 micron CMOS process generation and centralized monolithic architectures (both superscalar and VLIW) which use long wires for connecting spatially separated resources may not benefit from the advancements in semiconductor technology[5][6]. The ever increasing trend towards miniaturization of devices makes utilizing huge transistor budget in a manner that enables high clock speed, low design complexity, and less energy consumption even more challenging[5]. However, resolving this challenge can enable the deployment of embedded systems for many performance-demanding never-before embedded applications at a lower cost. Another challenge posed by this technological advancement is the rising level of the leakage energy consumption in the logic. The increase in the transistor density requires reducing the supply voltage in order to operate the circuit reliably. The reduction in supply voltage also requires reduction in the threshold voltage in order to maintain the speedup and this leads to an exponential rise in the leakage component of the energy consumption[7]. With the 65nm and smaller technologies currently in fabrication, the leakage energy is on par with the dynamic energy consumption. In future technologies the leakage energy will further dominate the overall energy consumption[8].

Clustered VLIW architectures(CVA)[9][10][11] have been proposed to overcome the difficulties with centralized architectures and to make them suitable for use in embedded systems. A clustered VLIW architecture[11] has more than one register file and connects only a subset of functional units to a register file. Groups of small computation clusters can be interconnected using some interconnection topology and communication can be enabled using any of the various inter-cluster communication models[12]. Clustering avoids area and power consumption problems of centralized register file architectures while retaining high clock speed which can be leveraged to get better performance. Texas Instrument's VelociTI[13], HP/ST's Lx[14], Analog's TigerSHARC[15], and BOPS' ManArray[16] are examples of the recent commercial micro-architectures developed based on clustered ILP philosophy. IBM's eLite[17] is a research proposal for a novel clustered architecture. Apart from clustered VLIW architectures, many other architectural philosophies[18][19][20][21][22] have also used distribution in its varied form to tackle the scalability problem in the past. This trend is expected to continue in the future also with ever growing number of transistors on the chip

Though clustering helps to combat the scalability problem by making components simpler and thereby increasing clock rate and reducing dynamic energy consumption of functional components, an interconnection network is required for the communication among different components. This communication happens over long wires having high load capacitance which in effect takes more time and incurs more energy consumption[5][23]. This problem is becoming severe with each upcoming process technology. As a result, clustered architectures are becoming more communication bound in terms of the performance and energy consumption. Apart from the interconnects, functional units are another major source of energy consumption in clustered architectures. The frequent accesses to functional units raises the temperature level and increases the leakage energy consumption which is specifically a concern in smaller technologies. Moreover, the contention for limited number of slow interconnects leads to many short idle cycles and that further increases the leakage energy consumption in functional units.

Clustered VLIW architectures rely on compile-time scheduling. The static scheduling simplifies the issue logic by alleviating the need for a dedicated hardware for scheduling. Thus, a significant fraction of the total energy consumption in clustered VLIW architectures is attributed to interconnects and functional units. Though, the exact percentage depends upon the architecture and circuit details, earlier studies report that a very high percentage (20% to 30%) of total processor energy consumption is attributed to interconnects[24]. Similarly, a large fraction (30% to 35%) of static energy conumption in a VLIW architecture is attributed to functional units[25]. An architecture level model developed in [26] also confirms that the leakage energy consumption in functional units constitutes a noticeable fraction of the overall processor leakage energy consumption despite having a smaller transistor count compared to the caches. Thus, optimizing energy in interconnects and functional units in clustered architectures is becoming more and more important from one process generation to another.

However, the functional units and interconnects are often underutilized in clustered VLIW architectures. Apart from other usual causes such as data dependencies, the under-utilization of functional units is also due to the contention for limited number of slow interconnect channels that introduces many short idle cycles for functional units. At the same time since the functional units are distributed among clusters, there is also more contention for functional resources which leads to the underutilization of interconnects. Finally, the contention for functional and interconnect resources in clustered VLIW architecture combine in a synergistic fashion and lead to greater available slack in clustered architectures as compared to VLIW architectures.

The advancements in VLSI technology now enable designing interconnects and functional units with different power and performance modes. For example [27][28] show that using 45nm technology, it is possible to design wires consuming 1/5 the energy but having twice the delay[27]. [29] proposes to use interconnect composed of wires with different characteristics to improve the ED^2 of the superscalar processor. Similarly, the capabilities of dual-threshold domino logic[30] with sleep mode (that can transition between active mode and sleep mode and vice versa without any performance penalty but with moderate energy penalty) can be utilized to do leakage energy management for short idle cycles in functional units. In this thesis, we propose a compiler-assisted approach that exploits the greater available slack and leverages on these advancements in VLSI technology to improve the usability of clustered VLIW architecture in smaller technologies, targeting the two major source of energy consumption namely interconnects and functional units.

1.1 The State-of-art

In the past, study of leakage energy management at the architectural level has mostly focused on storage structure such as cache[31][32][33]. Relatively, a little work has been done on architecture level leakage energy management in the context of functional units[34][35] of superscalar processors. Albonesi et al.,[34] propose a purely hardware based scheme in the context of a superscalar architecture for aggressive leakage energy management which is specifically useful for smaller technologies. However, being a purely hardware based scheme, it suffers from the problem of limited program view and thus the benefits are severely (on average, by 30%) affected by frequent transitions from active mode to sleep mode and vice-versa because of many short idle periods. Our compiler-directed approach for leakage energy management in functional units in the context of clustered VLIW processors leverage on global view of the schedule and greater available slack in the context of clustered VLIW architectures to directly improves over [34]. Though there has been some work in the direction of energy efficient scheduling in the context of VLIW architectures[36][25][37], to the best of our knowledge, there has been no such work for aggressive leakage energy management in the context of clustered VLIW architectures specifically targeting smaller technologies.

As compared to reducing energy consumption in function blocks, study of energy efficiency in interconnects is still in its infancy. Before proposing any architectural or compiler optimization for saving interconnect energy, one needs a model for high level estimation of interconnect delay and energy. However, prior research in interconnect modeling and analysis has mostly dealt with specific circuit level issues [28][38][39] and is not directly usable to make high level micro-architectural trade-offs. For example, an architect would be interested in knowing what are the available trade-offs in terms of pipeline latency and power, for a given bandwidth and interconnect distance. This information could be used at a higher level of design to obtain the overall optimum for the system. Hence, there is a need for a model or a tool for the interconnect, which can give reasonably accurate design points and their associated area and power costs for various architecture level constraints like bandwidth, latency etc. Similar models are available for caches[40], register files[41], and functional components[34]. Availability of an interconnect model will be very helpful for architects to involve interconnect in early design trade-offs. We fill this gap by proposing a high level interconnect energy and delay estimation model to get a fast but reasonably accurate estimates of interconnect delay, area, and power for a given technology, wire length, bit-width, clock frequency and latency. In contrast to earlier circuit level models, the proposed model is directly useful for the architects and compiler designer to perform energy optimization in interconnects.

In the absence of any high level model for interconnect energy estimation, primary focus of research in past had been to reduce the latency of communication[42] and evaluation of various inter-cluster communication models[12]. We introduce the idea of heterogeneity in interconnects of clustered VLIW architecture. We also propose a notion of communication slack of data values and a scheduling scheme that exploits the communication slack to save significant energy in heterogeneous interconnect in the context of clustered VLIW architectures. Again to the best of our knowledge, we are not aware of any work targeting reduction of energy consumption in interconnects in clustered VLIW architectures.

The novelty of our approach also lies in an integrated scheduling algorithm that simultaneously reduces the energy consumption in functional units as well as interconnects. The contention for a limited number of functional and communication resources in a clustered VLIW architecture leads to increased cycles of execution on a clustered machine as compared to an equivalent VLIW machine. Our combined scheme aggregates the scheduling slack of instructions and communication slack of data values in a synergistic fashion to convert the inherent idleness of functional and communication resources in clustered architectures to energy gains.

1.2 Our Approach

VLIW and clustered VLIW architectures are optimized for peak performance in order to meet real-time performance requirements of embedded applications. However, functional units are underutilized due to the inherent variations in the ILP of programs. The idleness is even more pronounced for clustered VLIW architectures because of the contention for a limited number of slow interconnects which manifests itself in the form of many short idle cycles. We observe that functional units are idle for more than 70% cycles on an average in our collection of media benchmarks for a 2-clustered machine with moderate number of of functional units. This idleness is not only because of data and resource dependencies but also because not all scheduling regions of the programs have the same amount of ILP. We also observe that roughly 50% of total 70% idle cycles have durations less than or equal to 10 cycles.

One way to reduce the number of idle cycles (attributed to contentions for cross-paths) and thereby improve performance is to use a high-speed high-bandwidth cross-path for communication of data values among clusters. Previous studies have reported that performance degrades by 12% when the latency of communication is doubled for a four clustered architecture, and that increasing the interconnection bandwidth from one to two improves the performance by as much as 10%[43]. We also observe a similar benefit in performance by using two bidirectional single-cycle cross-paths between clusters as compared to the configuration which uses only one cross-path. However, having both the cross-paths optimized for low latency results in high energy consumption in interconnects. This is because improving the latency of a communication channel requires closely spaced repeaters which increase the area and energy overheads of repeaters[27]. A high speed path for communication of data values among clusters indeed enables better performance, but we argue that not all data values are critical enough to be communicated on a high speed path, that many communications are non-critical and can still happen on a slow path without affecting performance. We introduce the notion of communication slack in the context of data values that captures the criticality of communication and we observe that on an average, 60% of communications can sustain a latency of three cycles or higher.

Thus a more suitable option to reduce the idleness in functional units without incurring a high energy overhead is to use heterogeneous interconnects between clusters with some paths optimized for latency and others for energy. We propose a scheduling mechanism that exploits the communication slack to steer the non-critical communication over the slower but energy-efficient wires while assigning critical communication over the fast but more energy-consuming wires. Such a configuration which uses one bidirectional single-cycle cross-paths and one bi-direction three-cycle cross-path between clusters reduces the number of idle cycles by 13% on an average as compared to the configuration which uses only one cross-path.

Though a high bandwidth cross-path mitigates the contentions for cross-path and improves performance to some extent, the variation in ILP of programs coupled with cross-path contention still manifest itself in the form of many short idle periods. Short idle periods render traditional leakage energy management schemes unusable. A hardware based scheme proposed by Albonesi et al.[34] utilizes the dual-threshold logic (and its capability of fast transition to and from low-leakage mode at moderate energy penalty) to perform leakage energy management for short idle cycles. However, effective energy savings of this scheme is low because of the high energy cost and frequent transitions. We propose a scheme that exploits instruction slack to aggregate idleness in functional units and improves effective leakage energy savings by reducing frequent transitions. We also evaluate the energy benefit of our compiler-directed technique in the context of instruction decoder.

Finally, we propose a combined scheme that exploits the communication slack of data values as well as instruction slack to reduce the energy consumption in functional units and interconnects. The proposed scheme tries to keep idle functional unit idle while maximizing the utilization of active functional units. Similarly, the proposed scheme also exploits the communication slack of data values to utilize the low-power cross-path as much as possible. The major contributions of this thesis can be stated as follows:

1. We propose a compiler-directed leakage energy optimization technique in the context of VLIW/clustered VLIW architectures, targeting the underutilized components such as functional units and instruction decoder and evaluate its benefits. Significant benefits are obtained over and above that offered by a hardware based technique (which already obtains good leakage energy benefits) with negligible performance loss. Apart from obtaining energy benefits without performance loss, the proposed technique also reduces peak power and step power consumption that impact the reliability of the chip[44][45]. The proposed technique is particularly useful for smaller technologies demanding aggressive leakage energy management.

2. We also propose a high level model for interconnect energy and delay estimation that can be used by architects, micro-architects, and compiler designers to explore the interconnect design space and evaluate interconnect energy optimization. The proposed model fills the gap between architect's need and circuit level models. The model takes architectural parameters such as length, bit-width, latency, and target technology and provides a set of interconnect options with varying degree of area, pipelining, and power budget using pre-characterized estimates of circuit parameters for different interconnect components[46].

3. We introduce the notion of heterogeneity in interconnects and propose a compiler-directed technique for energy optimization in interconnects of clustered VLIW architectures. We evaluate the energy benefits of the proposed techniques using the proposed interconnect

energy model[47][46].

4. We propose integrated schemes for energy optimization in functional components as well as interconnects simultaneously in a novel fashion and evaluate the energy benefits of the combined schemes[48].

5. We also present a detailed performance evaluation of all the schemes by adapting Trimaran compiler infrastructure for clustered architectures.

1.3 Outline of the Thesis

Chapter 2 provides a brief survey of earlier work in the context of energy modeling, energy efficient architectures, energy optimization in hardware, software (both at OS level and at compiler level) as well as HW/SW co-optimization techniques. Chapter 3 describes the clustered VLIW architectures and our experimental framework. Chapter 4 describes a compiler-directed scheme for leakage energy optimization in functional units and instruction decoder of clustered VLIW architectures. We also provide an example that shows the energy benefits of the proposed scheme and a detailed performance evaluation for VLIW and clustered VLIW architectures showing the energy benefits of the proposed scheme. Chapter 5 presents a model that we have developed for interconnect delay and energy estimation. We also present results that show various interconnect design trends in different technologies and a detailed validation of the model based on circuit-level spice estimations. Chapter 6 describes compiler-directed interconnect energy optimization techniques in the context of clustered VLIW architectures. An example is presented to show the functioning of the scheme. A detailed performance evaluation is presented to show the benefit of the proposed scheme. Chapter 7 describes integrated techniques for energy optimization in both functional units and interconnects. Experimental results show the energy benefits of proposed scheme. Chapter 8 presents conclusions and future directions of this work.

Chapter 2

Related Work

This chapter presents a brief survey on earlier work in energy optimization. Section 1 provides basic power equations for CMOS circuits and demystifies various power-performance metrics in widespread use. Section 2 describes earlier work on circuit/logic level techniques for power optimization. Section 3 describes energy-efficient architectural design techniques. Section 4 deals with energy-efficient optimization at the software level i.e., at the operating system level, compiler level as well as application level. Section 5 describes some hardware/software co-optimization techniques for energy optimization. Section 6 briefly discusses some recent developments and advanced techniques for energy optimization found in commercial systems. Section 7 presents different approaches to energy modeling. Section 8 concludes this chapter with a mention of the limitations of the existing work.

2.1 Power Basics and Metrics

The power consumption in a CMOS circuit can be reasonably approximated by the three component Equations 2.1, 2.2, 2.3[7]. The parameters used in these equations are described in Table 2.1. The first equation represents dynamic or switching power. Switching power is the power consumed due to some work being performed i.e., activity happening in the circuit. In other words, it is the power consumed when the outputs of the gate switch, which electrically means charging and discharging of the output capacitance of the gate.

Activity factor represents the fraction of the gates that undergo switching because not all gates switch in every cycle. Dynamic power is the dominant component of power up to 90 nm technology with the leakage being roughly 10% of the total power.

$$P = ACV^2f + VI_{leak} + \tau AVI_{short}f \tag{2.1}$$

$$f_{max} \propto (V - V_{threshold})^2/V \tag{2.2}$$

$$I_{leak} \propto exp(-qV_{threshold}/(kT)) \tag{2.3}$$

However, with 65 nm and smaller technologies, leakage power represented by the second term of the equation becomes more dominant. Leakage power is consumed irrespective of whether activity happens in the circuit or not, i.e., it is present irrespective of any work being performed or otherwise. The major reason why leakage dominates in smaller technologies is as follows. The increase of transistor density from one technology to another requires reducing operating voltage in order to maintain the reliability of circuit operation. Though reducing voltage helps to reduce dynamic power quadratically, it also impacts the maximum operating frequency which, as Equation 2.2 shows has roughly linear dependence on operating voltage. Thus, in order to maintain speedup in the event of reducing voltage, threshold voltage should also be reduced. However, reduction in threshold voltage leads to exponential rise in leakage current as given by Equation 2.3. In 65 nm technology, leakage power is on par with dynamic power and it further increases in smaller technologies and this presents a significant power challenge.

The third component of power is attributed to short-circuit current that flows momentarily between ground and power supply during switching of the gate. Unlike leakage current, this component of power neither dominates nor exponentially increases.

Apart from these three power components, two other factors related to power have significant impact on design and reliability of circuits. Peak power is the maximum limit on power above which a circuit undergoes catastrophic damage and thus, the circuit should be operated below peak power all the time. Step power is the sharp change in power requirement of a circuit from one time step to another. This is also called ground bounce or di/dt noise which has an effect on voltage levels and can lead to erroneous behavior. Thus, it is required to avoid sharp changes in the power requirements of circuit for reliable circuit operation[7].

Whereas power is the correct measure for wall powered devices such as usual desktop PC and servers, energy is the correct measure for devices powered by batteries (such as cell phone, laptop etc.) which store a fixed amount of charge. This is because the amount of energy consumed

Table 2.1: Parameters in Power Equations 2.1, 2.2, 2.3

A	Activity factor
C	Total capacitive load at the output of the gate
V	Operating voltage
$V_{threshold}$	Threshold voltage
f	Operating frequency
I_{short}	Short circuit current
f_{max}	Maximum operating frequency dictated by V and $V_{threshold}$
T	Operating temperature
k	Boltzmann's constant
q	Electronic charge
τ	Duration of flow of the short circuit current

for performing a task decides the lifetime of the battery in these devices. The computational efficiency of two devices is often compared using the metric, MIPS/watts. The impact of energy optimization is compared using product of energy and execution time popularly known as energy-delay product. The energy-delay product gives a fair measure of comparison and is immune to tricks. Techniques for dynamic voltage and frequency scaling are often compared using $energy - delay^2$ that gives more weightage to performance loss than to energy savings as compared to just energy-delay product. $Energy - delay^2$ is more useful for applications where high performance degradation is not desirable for energy savings.

2.2 Circuit and Logic Level Techniques

Various circuit and logic level techniques are available to design circuits with reduced effective capacitance, thereby reducing dynamic power consumption. Circuit level techniques have also been developed to reduce leakage power as well as power of interconnects. This section presents available techniques for reducing power consumption at the circuit design level.

2.2.1 Dynamic Energy Optimization in Logic

Reducing transistor width reduces power. However, it introduces more delay and thus, the implementation of this technique is done effectively by changing the width of a transistor based on its distance from the critical path[49]. Switching of a transistor leads to the switching of adjacent transistors as well and increases the overall power consumption. Thus, transistors are often grouped together such that those that switch often are placed near the output of the circuit. This restructuring is often applied at multiple levels. For example, gates are often restructured in a chain topology and this reduces switching as compared to tree topology[50][51]. Half frequency clock synchronizes at rising as well as falling edge. In general, automation tools known as EDA tools are used to come up with the best implementation of a circuit by exploring many possible implementations to meet the desired delay, area, and power goals[52]. Half frequency clocks enable significant power savings by cutting down the operating frequency to half. Similar benefits can also be achieved by using half swing clocks that swings only to half the voltage[53]. However, implementation of half frequency and half swing clocks poses significant design problems that also exacerbates the already troublesome clock skew problem[53]. Apart from logic gates, the flip-flop which is the basic storage element, is also a major source of power consumption, especially because of spurious activity. Self-gating flip-flops and conditional capture flip-flop check this spurious activity and hence reduces power consumption due to spurious input changes[54][55].

2.2.2 Leakage Energy Optimization in Logic

As described earlier, leakage energy is expected to rise exponentially with technology scaling. Various circuit level techniques which aim to reduce the leakage energy have been proposed. The straightforward way of reducing leakage by increasing threshold voltage impacts speedup[7]. One technique exploits the stacking effect that says that two or more transistor stacked on each other consume less overall leakage because stacking induces a slight reverse bias between the gate and source of the bottom transistor. Thus, stacking increases the effective threshold voltage of the bottom transistor and hence reduces the leakage current. Another technique proposes to use a high threshold sleep transistor to isolate the leaky circuit element from supply voltage and ground, and thereby reduces leakage drastically when the circuit is inactive. However, it involves careful sizing of the sleep transistor, as otherwise, the overhead of activating it will be high[56]. A Multiple threshold circuit uses high threshold transistors on the critical path and low threshold transistors on non-critical paths but this technique calls upon a careful and difficult design[57]. Adaptive body biasing technique dynamically adjusts the threshed voltage of a circuit by applying a voltage to the body of a transistor depending on whether the circuit is active or not[58]. Leakage equation 2.3 shows the exponential dependence of leakage on temperature which is specifically a concern for smaller technologies. Higher leakage leads to higher total energy consumption energy which in turn leads to more heat dissipation and hence high temperature. High temperature in turn increases leakage energy which increases temperature again. This vicious cycle can lead to a thermal runway which may eventually damage the chip. Thus, circuit designers are looking at new ways of cooling the chip to check the temperature and its impact on leakage, such as refrigerating the chip and circulating cryogenic fluids such as liquid nitrogen[59]. Apart from reducing the leakage, reducing temperature also improves the performance and reliability of the chip.

2.2.3 Interconnect Energy Optimization

An interconnect is a place of high electrical activity and hence causes heavy energy consumption. Many proposals target to reduce the energy consumption in interconnects by reducing spurious transitions or by using only a part of the bus. Bus inversion transfers the actual values or inverted values depending on which causes less number of transitions. Apart from transitions, cross-talk among nearby wires also adds significantly to interconnect energy consumption. Shielding wires are introduced to reduce coupling among the wires. Self- shielding codes encode the value in such a way as to reduce cross-talk among neighbor wires[60]. Low-swing buses transfer a value with lower voltage and amplify it back to reduce energy consumption. However, this comes at the cost of extra hardware[61]. Bus segmentation divides a bus into segments and instead of keeping the whole bus active keeps only the desired segment active[62]. A network-on-chip extends the segmented bus idea to the whole chip by composing an interconnection network by rows and columns, which in turn are divided into segments and demarcated by tri-state buffers. Different segments are selectively updated as data passes through them[63].

2.3 Architectural Techniques for Power Optimization

Earlier work on architectural techniques for power optimization as described below has mostly focused on energy consumption of caches and issue queues that contribute significantly to overall processor energy consumption. Techniques proposed earlier either reduce the energy consumption of an access or reduce the total number of accesses. Splitting memory into smaller sub-banks and splitting banks into sub-banks enables energy savings by keeping only parts of the memory active[64]. Further a compiler can help in clustering data into banks, thereby providing more energy savings[65]. Filter caches provide significant energy savings by introducing another level of cache between processor and L1 cache to filter many accesses[66]. Another idea is buffering a block from the cache and if access is from that block this saves energy. Scratch pad memory is software controlled memory (filled ahead of time) that resides on the chip and can be accessed in a single cycle[67]. Low power trace cache designs try to reduce the number of instruction cache accesses. Instead of accessing both instruction cache and trace cache, branch confidence estimation can be used to access either from the trace cache or from the instruction cache[68].

An adaptive Cache changes the state of a cache at the granularity of a line, block, or set[69]. A decay based approach simply puts the cache lines that are not in use after some time interval into drowsy or sleep mode[70][32]. Another approach determines hot spots by deciding the branch outcome (sufficiently taken branch target is a hot spot) and activates the cache lines accommodating the hot spot[71]. Deadline elimination based schemes power down cache lines containing basic blocks that have reached their final use[72]. The adaptive instruction queue technique divides the instruction issue queue into several partitions and activates only the desired partition[73]. Certain schemes use program IPC and compare it with IPC in an earlier interval or threshold to adjust the issue queue size[74]. Yet another approach deactivates the youngest part of instruction queues based on its contribution to overall IPC[75].

2.3.1 Dynamic Voltage Scaling

DVFS changes the processor voltage and frequency with changing workload of a processor to gain energy benefits without loosing performance. However, this deceptively simple idea involves various complications. Predicting workload is complicated because of arbitrary preemption and presence of modern features like pipelining, out-of-order execution, caches etc. Earlier work has focused on worst case execution time determination based on static analysis and measurement based prediction as well as combination of both. Moreover, the pipeline model and the cache model have been integrated into the compiler/simulator for better estimation. Nondeterminism and anomalies in real-time systems add to the complication of effective dynamic voltage and frequency scaling. Finally, DVFS mostly targets dynamic energy and often impacts performance. The rising level of leakage energy in smaller technologies can overwhelm the benefits of aggressive DVFS. The I/O which is powered at high voltage can also nullify the benefits of even a tremendous saving in energy in the microprocessor part because of longer execution[76].

These are the different approaches to DVS. An interval based approach determines the usage of CPU in previous interval(s) to determine the processor speed for the next interval. They differ in determination of future processor utilization[77]. These are good for regular intervals but not for irregular intervals. Inter-task approaches monitor the hardware using hardware performance counters while the process executes, and determines the different frequencies for different processes based on these to save energy[78]. Inter-task DVS however suffers from two drawbacks.

- The approach is unaware of program structure and don't utilize it to determine processor speed. Flaunter et al.,[33] try to address this problem.

- Task workloads are usually unknown and it works based on the assumption that perfect knowledge of workload is available. Some also use history based workload prediction.

These limitations cause problems for irregular workloads. In contrast, Intra-task approaches adjust the processor speed and voltage within a task. One approach divides each task into time-slots and sets the processor speed to the slowest needed to finish the execution up to this time slot based on earlier worst-case execution time of the whole task[79][80]. Compiler level intra-task algorithm due to Shin and Kim,[81] determines the execution time of different program paths by profiling and inserts instructions to adjust frequency at the start of a different path, based on how far away is a path from the critical path. Hsu and Kremer's[82], approach is to generate a table of execution time and energy consumption for different regions and for different frequency combination and choose the best frequency combination for each region.

MCD architectures stand in between the fully synchronous and fully asynchronous architectures and are often called as globally asynchronous locally synchronous architectures[83]. An MCD architecture divides a processor into multiple domains for which frequency and voltage can be individually chosen. Absence of global clock makes MCD architectures, specifically useful to deal with the problem of clock skew which is increasingly troublesome for smaller technologies. DVS can provide more benefit for MCD architectures. However, communication among different domains in MCD architectures make the DVS problem even more complicated in the context of MCD architectures. Runtime DVS algorithms for MCD use issue queue usage for each domain to get a load estimate and accordingly determine the frequency for each domain. Whereas, earlier algorithms make their decision after a fixed interval[84], later improvisation adaptively decides when to apply DVS[85]. A profile based approach[86] executes the program to obtain a runtime dependence graph of the program. A Shaker algorithm[86] then works on the graph from the root to the leaves and from the leaves to the root, to determine the operations with excessive energy dissipation and stretch their execution to get energy benefits until no operation has slack or give energy benefits. The feedback from this algorithm is used to select clock frequencies for different regions[86].

2.3.2 Leakage Energy Management of Other Devices

As mentioned earlier, leakage energy is dominating the overall energy consumption in smaller technologies. Resource hibernation or putting the idle resources into low leakage mode is a popular technique to reduce the leakage energy consumption. This technique has been applied on various components and at various levels.

Displays are the largest consumer of power. The traditional way is to dim the display after a threshold amount of no-response time. Face-off technique photographs the display perception and powers down the display if the face is not recognized. The associated cost of photography is being addressed by less energy consuming heat sensors to control the photography[87]. Zoned back-lighting technique is based on the assumption that the user is interested in only a part of the display. This scheme keeps only the focused window bright and rest of windows are dimmed[88].

Disk drives are another large consumers of power and most of this is attributed to rotating platters. One technique puts a platter into hibernation after a threshold amount of time[89]. Low threshold means more savings but also the risk of more energy and performance loss because of bumps. High thresholds may miss potential savings. An adaptive scheme increases and decreases the threshold dynamically to adapt to the usage and provides more effective energy savings[89]. Another scheme that is more useful in servers (where idleness is not very frequent) is to charge the RPM based on input queue size and system response time[90]. Operating systems can further help by clustering disk usage requests[91].

2.4 Software Level Power management

2.4.1 Compiler-Directed Power Management

Traditionally, performance oriented compiler optimization are used to provide implicit energy benefits. [92] demonstrated that loop transformations provide energy benefits in memory but increase the CPU energy dissipation. [93] found that optimizations that reduce workload to improve performance such as common subexpression elimination, copy propagation and loop invariant code motion also reduce the CPU energy dissipation. However, correlation between performance optimization and optimization for low power is not true in general. Moreover, not all performance oriented optimizations improve peak power dissipation. For example, software pipelining and loop unrolling actually increase peak power dissipation because more ILP is exploited. Thus, many compiler techniques have been developed to reduce power and energy explicitly in different components. Compiler optimizations can reduce the number of memory accesses by keeping the data closer to the processor and reduce access energy consumption by judicious assignment of data to memory locations. [94] proposes various compiler techniques using state-preserving and state-destroying low-leakage cache modes that determine last usage of instruction and place the corresponding instruction cache lines into the appropriate state. [95] exploits data locality and proposes compiler techniques to keep only a part of the data cache (needed by current computation) active in order to save energy in data caches.

As compared to storage structures, compiler level techniques for reducing energy consumption in functional components and interconnects are rather scarce and are mostly proposed in the context of superscalar and flat VLIW architectures. Zhang et al., have proposed a rescheduling scheme to reduce dynamic and leakage energy in the functional units of a VLIW processor by exploiting the remnant slack of a performance-oriented schedule[61]. Kim et al.,[25] have proposed a leakage energy management scheme for VLIW processors that approximates the ILP available in the program using heuristics. The calculation is at the coarser loop level granularity assuming that there is little variation in the ILP within the loop. Their scheme keeps only canonical subset of functional units that is sufficient to exploit this approximated ILP in the active state. Gupta et al.,[35] propose a novel data structure called power-aware flow graph. Their leakage energy management scheme in the context of superscalar processors works over this graph to determine larger program regions called power blocks which offer opportunities to save leakage energy. ISA and architectural support is needed to switch the functional unit on and off at the boundaries of power blocks and nullify spurious switching. Kim et al.,[37] have proposed a modulo scheduling algorithm that produces a more balanced schedule for software pipelined loops with an objective to reduce the peak power and step power. Lee et al.,[96] propose an optimal horizontal rescheduling and a heuristic vertical rescheduling algorithm that exploits the slacks in already scheduled code to minimize the transition activity on instruction buses (used to transfer instruction from instruction memory to decoder), thereby reducing the power consumption due to transition activity on the instruction bus. We compare and contrast our proposal with these earlier approaches in the area of compiler-directed energy management in the respective chapters.

Remote compilation and remote execution are other challenging areas for energy optimization. It has been observed that it is energy efficient to compile remotely if compilation time exceeds the execution time[97]. However, the best strategy would be to execute, even the code remotely and just transfer the result just in time[98].[98] also investigates various static and dynamic methods of partitioning the code for remote compilation and execution. While the static method decides at compile time which method would be executed remotely, dynamic methods are superior in the sense that they dynamically make these decisions based on each call-site[98].

2.4.2 Application and OS Level Power Management

Application transformation and adaptation techniques abstract an application as a software architecture graph that consists of processes, event handlers, and inter-process communication mechanisms (IPCM). It considers the base energy consumption of applications and then repetitively applies transformations to applications such as margin processes and replaces costly IPCM with cheaper IPCM to attain an energy optimized version of the application[99]. Other popular techniques are: trading the accuracy of computation with energy consumption[100] and reducing energy consumption by trading off the fidelity[88]. Application usage is monitored and signal is given when the usage falls below the requested level. Application lowers the quality of service until resources are plentiful. Echos system abstracts the resources as monetary objects. The applications are distributed currency based on currently desired battery rate and the currency is consumed to use resources. The application is interrupted when the currency is depleted[101].

Application hints can be very useful in improving the energy efficiency of systems. Some software architectures consider systems as composed of two APIs. An API that allows application to communicate with the OS and another API that allows the OS to communicate with the hardware. An application can specify its power performance trade-off by giving the usage, deadline etc., for resources and these trade-offs are used by the OS for managing resources[102]. For example, an application can check if the disk drive is in sleep mode and whether it is costly to access it, Then the data can be accessed through the network. Thus, an application has abstract control over devices which is mapped by the OS to exact control. Similarly, a compiler can cluster disk accesses and network accesses by applying transformations and then add hints for the OS for clustered accesses to the disk. The OS can use this hint to process a batch of disk accesses and put the disk into sleep mode after processing[103].

2.5 Hardware/Software Co-Optimization

A cross-layer scheme applies power management at different levels i.e., at hardware, operating system, compiler, and application levels. The four major advances in this area are as follows :

Forge system[104] is an integrated power management framework for networked multimedia systems (specifically targeting video frame requests) that uses DVS, leakage control mode, and other architectural adaptation such as cache ways and register file size at architecture level. It has control knobs at OS and compiler levels to control these adaptations and a distributed middle-ware (at mobile device and proxy server) that takes feedback from the mobile device about energy statistics and decides the network traffic regulation and quality of service. Different heuristics are used at different levels but the whole system works in coordination with feedback, for integrated energy management[104].

Grace[105] is a cross-layer adaptation framework targeting real-time multimedia workload and integrating dynamic voltage scaling, power-aware task scheduling, and QoS settings. Grace works with two layers: a global layer that makes decisions less frequently because of high cost and a local layer which is more active in decision making. The local layer controls CPU frequency, task scheduling and adapts QoS parameters within the task. The information about a task is also broadcast to other tasks. Major variations such as low- battery level and large workload variations trigger the global adapter[105].

Another four layer system proposed by Fei[106] has two system layers of Hardware and OS and two user level layers of middle-ware and application. The application declares its energy requirement for a range of QoS levels before getting admitted into the system. This information coupled with the information about battery levels (determined by OS) is used by the middle-ware to determine the systems settings. These setting are communicated to the OS before admitting the application process in to the system[106].

Another work[107] proposes a mixed Compiler-OS approach targeting real-time systems with fixed deadlines and worst case execution time requirements where the compiler emits code to put remaining worst-case execution time values into register and OS reads these value to set system settings accordingly[107].

2.6 Power Management in Commercial Processor

The Pentium 4 processor though designed from the performance perspective also features a thermal detection and response mechanism that inhibits the processor's clock from reaching most parts of the processor if the temperature enters the danger zone. It also features a register using which the operating voltage and frequency can be set to either high mode (wall-powered/high performance) or low mode (battery powered/energy efficient)[108]. Pentium M processor is specifically designed for the handheld domain. Thus, the architecture has several features that balance performance and energy consumption. The major features targeting energy efficiency include hardware for predicting idle units and inhibiting the clock signal, activating only a part of the bus needed for requisite communication, and execution stacking that clusters together units that perform similar functions. It also supports six different voltage-frequency settings for DVFS and low leakage transistors in the cache for reducing leakage energy[109]. Intel PXA27x processors use a profiler to determine the memory bounded-ness of programs using hardware performance counters and use this to decide the power modes of the processor.

Transmeta Crusoe Processor uses a code morphing engine to translate x86 instructions into the native VLIW packets with on-the fly optimization thereby providing energy efficiency by migrating various hardware functionalities into software and reducing processor real-estate and hence energy[110]. Later generations are proposed to have features to reduce the leakage energy consumption as well. IBM has developed an extensive OS power management module that abstract away the specification of policies using operating point (frequency and voltage settings) and operating state (active, sleep, idle) which is mapped automatically to the architecture thereby providing the independence of policy from architecture[111]. ARM's intelligent energy management (IEM) software consists of three layers that coordinate together to decide optimal processor frequency. At the bottom is the baseline algorithm that uses previous workload to predict future workload and the required frequency settings. At the top is the application dependent decision making algorithm based on clock frequency. In between is the interface using which an application can communicate its performance requirement to the baseline algorithm. The performance prediction at the three levels are combined to obtain a confidence rating which is used to decide the final processor frequency. The powerwise adaptive voltage scaling is combined with IEM where voltage for the chosen frequency is decided and modulated dynamically based on continuous monitoring of temperature and other environmental factors. This provides 45% more energy savings compared to a scheme which computes a table of frequency and corresponding voltages offline[112].

Apart from reducing power consumption, recently, there has been an increase in interest in building alternative power sources. Research in this direction is fueled by limited capacity of batteries and the advancements in battery technology are far slower compared to the demand[113]. Two alternative technologies getting specific attention are fuel cells and MEMS. Fuel cells consist of anode, cathode and a separating membrane. Consumption of fuel as it moves between electrodes produces electrons that generates electricity. The major advantages of fuel cells over batteries are the ease of refueling and significantly high energy densities. The associated disadvantages are the higher amount of heat, high cost due to usage of metallic and mechanical components, and the need for high safety measures before these can be allowed in airplanes[114]. MEMS systems are miniature versions of large scale gigantic gas turbine engines which work on the same principle to convert mechanical energy into electrical energy and offer high energy density. The major obstacle in their adoption is the high heat density that could raise chip temperatures to dangerous levels, which is attributed to hot exhaust gases produced by jet engine, inflammability, and power dissipation of turbines[115].

2.7 Energy Estimation and Energy Modeling

Software energy estimation is important to evaluate the benefit of any energy optimization as well as system design. However, it is inherently difficult because exact estimation is often dependent on low-level circuit details which are not readily available. Moreover, estimation may itself involve an energy cost and may suffer from estimation errors depending on the granularity of the estimation. Energy estimation approaches calculate the total energy of software execution as the aggregate of the energy consumption of individual activities in the system. The major differences among different approaches are the definition of what constitutes an activity, how the activities are counted, and how the energy cost for each activity is determined.

The instruction level energy model proposed by Tiwari et al., considers instruction as an activity[116]. The prerequisite to energy estimation in this method is the characterization of the target architecture which involves executing each instruction over and over again in a loop and physical measurement of current to determine the associated energy consumption. The earlier model proposed by Tiwari et al., ignored the inter-instruction effect. The model proposed later takes into account the inter-instruction effect by determining the energy consumption for each pair of instructions and taking into account the pipeline stalls and cache misses[117]. The major advantage of this methodology is that it can be used for off-the-shelf processors for which the circuit level details are not readily available. The disadvantage with this method are the lengthy characterization phase and measurement errors. Later improvisation on this model grouped similar instructions together to reduce the characterization errors[118][119]. Other extensions to this work looked at modeling other system activities to improve the accuracy of model such as buses[120].

Another approach motivated by cycle accurate simulation of program execution is the activity based energy modeling. This approach determines activity in different components of the architecture as the program executes and based on pre-estimated per-activity cost of each component, the total energy of execution is determined. A pioneering work in this direction is the Wattch energy simulator that is based on the simplescalar execution simulator[121]. Other simulators proposed differ in the way they model the activities in different micro-architectural components and how the energy consumption for each activity is determined[122][123]. The advantage with this approach is that energy estimation is systematic and is based on all runtime events. A disadvantage is the need to determine energy for different micro-architectural events that require detailed implementation-level knowledge of the target architecture.

The sampling based approach to energy modeling interrupts the software execution periodically to collect instantaneous power values. The values are later used to determine the energy consumption by different processes[124]. The disadvantage with this method is the performance and energy overhead of constant sampling as well as wrong correlation of energy to a different process. Hardware performance counters have also been used to obtain sampling intervals of finer granularity based on critical events in the execution of the program[78].

2.8 Limitations of the Earlier Work

As described in the earlier sections, earlier work for energy optimization has mostly concentrated at circuit level or architectural level. The work that has happened at software level has mostly focused on reducing dynamic energy consumption using techniques such as DVFS. Some of the work at the compiler level also targets to organize data in memory structures such as caches or to reduce the cache accesses to improve the energy consumption. Though the role of software level energy optimization techniques in general and compiler optimization techniques for energy savings in particular have been recognized for quite long, following are the limitations of earlier works in the direction of software level energy optimization.

1. There is limited work on compiler level leakage energy optimization techniques (specifically), though leakage is the dominant component of power in current (65 nm) and future technologies. The role of a compiler is rather limited in the context of superscalar architectures but in the context of VLIW and clustered VLIW architectures, the compiler has a major role to play, not only for performance optimization but also for energy optimization. Though there have been some proposals in the context of VLIW architectures, they mostly address leakage energy management at the coarser granularity whereas the increasing significance of leakage in future technologies demands aggressive leakage energy management. In the context of clustered VLIW architectures, where dynamic energy is already checked because of simplification of components, leakage is even more significant. There is hardly any proposal that targets leakage energy savings in the context of clustered VLIW architectures. We propose and evaluate our compiler-directed instruction scheduling algorithm that assists a hardware based scheme in achieving better energy savings in

the context of VLIW and clustered VLIW architectures.

2. Energy modeling work in the past has mostly focused on modeling energy in logic and storage components such as caches. Interconnect energy modeling has started getting attention only very recently. Recent work on interconnect energy modeling is only useful for low level modeling of interconnects and is not very useful for architects or compiler designers to evaluate energy optimizations for interconnects. With the trend towards more and more decentralized/clustered architectures, the interconnect becomes the major performance and power bottleneck. Thus, there is a need for high level interconnect energy models that can be used by architects and compiler designers to evaluate the energy optimization. We propose one such model.

3. In the absence of any high level model for interconnect energy estimation, the primary focus of research in the past had been to reduce the latency of communication[42] and evaluation of various inter-cluster communication models[12]. However, studies clearly report that the trend towards decentralized architectures make the interconnect not only a major performance bottleneck but also a power bottleneck. We propose a compiler/architecture co-optimization technique for interconnects in the context of clustered VLIW processor.

4. Most of the prior research focuses on optimizing energy in storage structures such as caches and scheduling hardware etc. In the context of embedded clustered VLIW processors, the absence of scheduling hardware and small caches require more aggressive energy management techniques for other dominant energy consumers such as functional units and interconnects. Our work is particularly pertinent to embedded VLIW/clustered VLIW processors specifically targeting components such as functional units and interconnects.

Chapter 3

Clustered VLIW Architectures and Experimental Setup

This chapter gives an overview of clustered VLIW architectures, describes our cluster scheduling approach, and the experimental framework. Section 1 introduces the clustered architectures. Section 2 discusses some popular clustered architectures specifically Texas Instruments' VelociTI and HP/ST's Lx. Section 3 describes various steps of our cluster scheduling approach. Section 4 gives a brief overview of the trimaran compiler infrastructure. We also describe the modification made to the trimaran compiler infrastructure to adapt it for clustered architectures. Section 5 presents a general description of our experimental setup.

3.1 Clustered Architectures

Processors based on new architectures that support high levels of parallelism needed to meet the demand of high performance in embedded applications have been developed. These have been made possible due to advances in semiconductor technology. These architectures exploit the fine-grained instruction-level parallelism (ILP)[1] available in abundance in embedded applications to attain high performance. ILP gains in performance by parallel execution of the lowest level computer operations in contrast to coarse grained parallel processing where performance benefits come from executing large sections of code in parallel on independent processors[1]. Two major ILP design philosophies are superscalar architectures[2] and VLIW architectures[3]. Though both these support multiple pipelined functional units connected to a single unified register file in parallel to achieve better performance, the major difference between the two philosophies lies in the way parallelism is detected and parallel instructions are executed.

Superscalar processors[2] have dedicated hardware responsible for scheduling instructions at run time. A branch predictor is employed to avoid pipeline stalls that occur in the event of control transfer. However, the complicated hardware needed to carry out dynamic scheduling leads to complicated design and high power consumption. Presence of a large number of functional units leads to commensurate increase in the number of read and write ports which in turn increase the chip area, cycle time, and power consumption. Furthermore, execution time becomes unpredictable because parallel instructions are created based on run-time conditions which change during different executions of the same program. Thus, the dynamic features of a superscalar processor make it difficult to assure a performance guarantee which is of utmost importance for real-time applications. These problems of superscalar processors make them less suitable for embedded systems[125].

Another design philosophy is VLIW architecture[3]. Here, the major point of deviation from superscalar architectures is that instruction latencies are exposed to the compiler and the compiler is responsible for scheduling. This eliminates the need for scheduling hardware. However, embedded applications operating in real time have performance requirements as high as 10^{10} to 10^{11} operations per second and even higher rates are expected in the future with the growing sophistication of applications[126]. To meet the high performance requirements of these applications more number of functional units are required to operate in parallel. As the number of arithmetic units in a processor increases, register storage and communication between arithmetic units become critical factors dominating the area, cycle time, and power dissipation of the processors. The single central register file that has traditionally been used to interconnect ALUs and provide short term storage does not scale well with large numbers of arithmetic units. For N arithmetic units, area of register files grows as N^3, delay as $N^{3/2}$, and power dissipation as N^3. The area of register files dominates the area of the ALUs for more than 7 ALUs, the delay of the register files dominates the latency of a floating point multiplier for more than 58 ALUs, and the power dissipation of the register file dominates the power dissipation of a floating point multiplier for more than 8 ALUs[126].

Performance measured as the execution time of a program is a product of three components namely, cycle time, cycles per instruction, and instruction count. Performance can be improved by reducing any of these components without having a compensating effect on the other components. Cycle time is determined by the sum of wire delay and gate delay along the critical execution paths of a processor pipeline. Wire delays have become significant for the 0.25 micron CMOS process generation and centralized monolithic architectures (both superscalar and VLIW) which use long wires for connecting spatially separated resources may not benefit from the advancements in semiconductor technology[5][6].

Traditional ILP architectures allow any ALU to read from and write to any storage location and thus provide both storage and communication among arithmetic units in a very general manner. Partitioning and clustering techniques can be effectively applied to monolithic ILP architectures to overcome the problems mentioned above to achieve better performance and scalability[9][10][11]. Complex centralized structures and resources can be partitioned to make them simple and fast. However, in order to achieve the same functionality as that of the original hardware without compromising on the performance, these partitioned resources must be carefully grouped together into small clusters that interact with each other. It has been shown that a large number of register instances are used only a few number of times and that they are used up soon after they are created[127]. This temporal locality of register reference can be exploited by partitioning the register file. The area, delay, and power consumption of a register file can be significantly reduced by restricting the communication between ALUs and registers, so that each arithmetic unit can read and write a limited subset of registers. A partitioned register file has an additional advantage that it can have a smaller number of registers per local register file than any other scheme for a given number of architected registers, and hence can potentially provide faster register access time. Clustered VLIW architecture (CVA) based on this idea has been proposed to overcome the difficulties with centralized VLIW architecture and to make them suitable for use in embedded systems[9][10][11].

Figure 3.1 and 3.2 depict the general clustered architecture and an individual cluster respectively. A clustered VLIW architecture[11] has more than one register file and connects only a subset of functional units to a register file. Groups of small computation clusters can be fully or partially connected using some interconnection topology. Connectivity can also be distinguished in terms of whether they are connected using a *point-to-point* network or a *bus-based* network. Inter-cluster communication can be provided in various ways as in the send-receive model, the

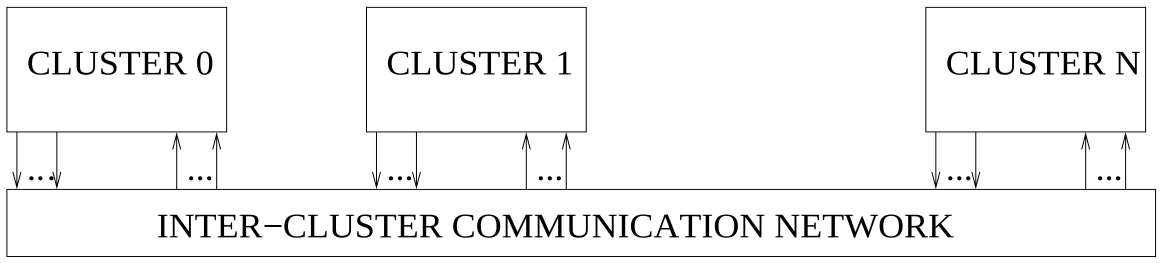


Figure 3.1: A General Clustered VLIW Architecture

extended operand model, the extended result model, and the broadcast model[12].

- A send-receive model enables ICC by explicit copy operations in regular VLIW issue slots.

- Extended operand model extends some of the operands with the cluster id field and thus allows an instruction to read some of the operands from some of the other clusters *without any extra delay* and without storing them in the local register file.

- Extended result model has an encoding of the cluster with the destination operand and it provides a sort of inter-cluster move (contrast to inter-cluster copy in other models) by storing the result of an operation in the register file of the other cluster. A genuine extension of this model is the multi-cast model where the result of an operation can be stored in more than one cluster.

- Broadcast model provides ICC by a set of shared registers that can be read and written in all the clusters.

Functional units can be grouped in either heterogeneous or homogeneous way. In heterogeneous clustering, each cluster has a different combination of functional units. Homogeneous clusters are generally preferred to heterogeneous clusters because they help in reducing the number of distinct components on a chip, and hence the complexity of VLSI design and the validation process[12]. Clustering avoids area and power consumption problems of centralized register file architectures while retaining high clock speed which can be leveraged to get better performance. Several commercial microprocessors such as Texas instrument's VelociTI[13], HP/ST's Lx[14], Analog's TigerSHARC[15], and BOPS' ManArray[16], have adopted clustered ILP architecture

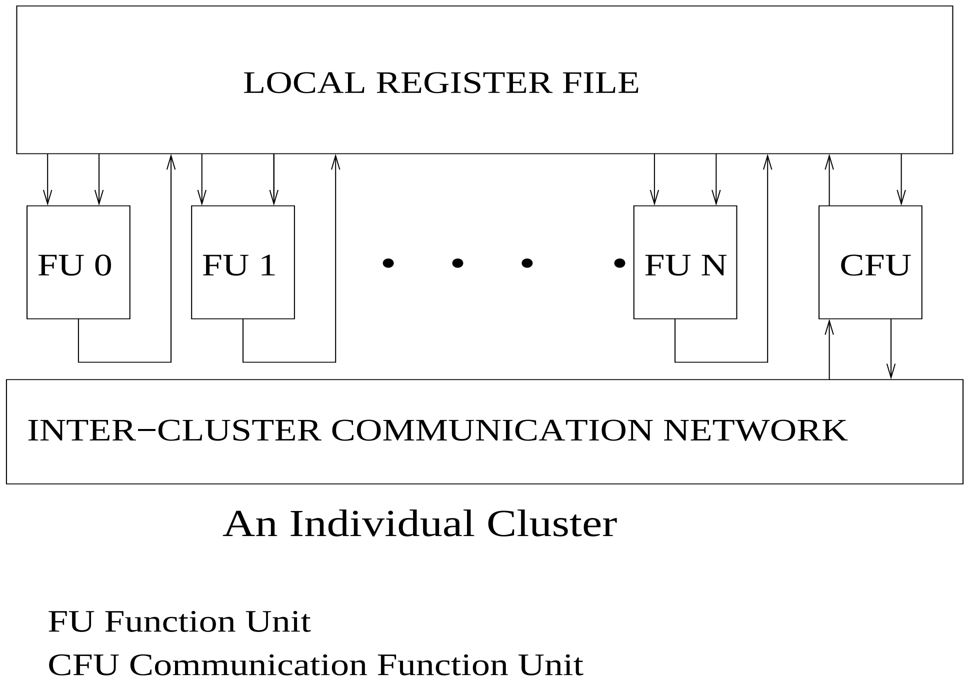


Figure 3.2: A Cluster

philosophy to meet the high performance requirements of embedded applications. IBM has proposed the eLite architecture[17] with novel features such as vector processing, facility to switch functional units on and off and also change clock speed and supply voltage.

3.1.1 Clustering Overheads

Though clustering enables higher clock speed, it also incurs overheads in terms of execution cycles and code size. Clustering overheads are due to various interrelated factors. Inter-cluster communication delays are governed by spatial distance between communicating clusters as well as the latency and bandwidth of the interconnection mechanism. Thus, communication among clusters costs extra cycles due to latency and limited bandwidth of the communication channel. Furthermore, encoding of the inter-cluster communication in the instruction set architecture (ISA) may also constrain operation scheduling because some of the issue slots and resources may be in use for inter-cluster communication and this may lead to delay in scheduling regular operations. Replication of values among clusters leads to increased register pressure. Thus the code size overhead can lead to extra cycles due to instruction cache stalls. Moreover, register requirements of instructions may cause spilling and hence extra cycles due to data cache stalls. Thus, due to interrelated nature of the different factors, their contributions to the clustering overhead are not always independent of each other[12].

As in terms of performance, clustering also provides mixed benefit in terms of energy. Though, clustering simplifies the components which gives benefit in dynamic energy consumption, the extra cycles of execution (because of reasons mentioned above) increase idleness of components which in turn increase leakage energy in components. As mentioned earlier, significance of leakage is particularly high in the current and future technologies. Another impact of clustering an architecture is the need for an interconnect for inter-cluster communication. The inter-cluster communication network is composed of long wires having high load capacitance. Thus, communication on inter-cluster communication network also contributes to the additional energy consumption in clustered architectures.

3.2 Some Clustered Architectures

3.2.1 VelociTI Architecture

VelociTI is a load-store RISC-style VLIW architecture[128]. Figure 3.3 depicts the block diagram of the architecture. Figure 3.4 shows the CPU data paths of TMS320C64X processor, one member of the VelociTI architecture family. The architecture follows a homogeneous clustering philosophy and has two clusters (named A and B). Each cluster has a 32x32 register file and 4 functional units (named L, S, M, and D). Having a VLIW architecture, it allows parallel fetch, decode and execution of multiple instructions that compose a VLIW instruction word. During execution, each instruction is performed on a single functional unit. Eight 32 bit instructions supply control for eight independent functional units. The instruction set consists of simple, atomic, and completely independent instructions. Memory operations have been decoupled from arithmetic operations. This feature lowers the number of data fetches for a particular algorithm and thus CPU power consumption as well. The most frequent instructions can be executed on the largest number of functional units. Every instruction can execute on at least two functional units. Six registers (A0,A1,A2,B0,B1,B2) can be used for conditional execution of instructions and all instructions can be executed conditionally. However, some instructions can be conditionally executed only by specific units. Most of the instructions are of unit latency.

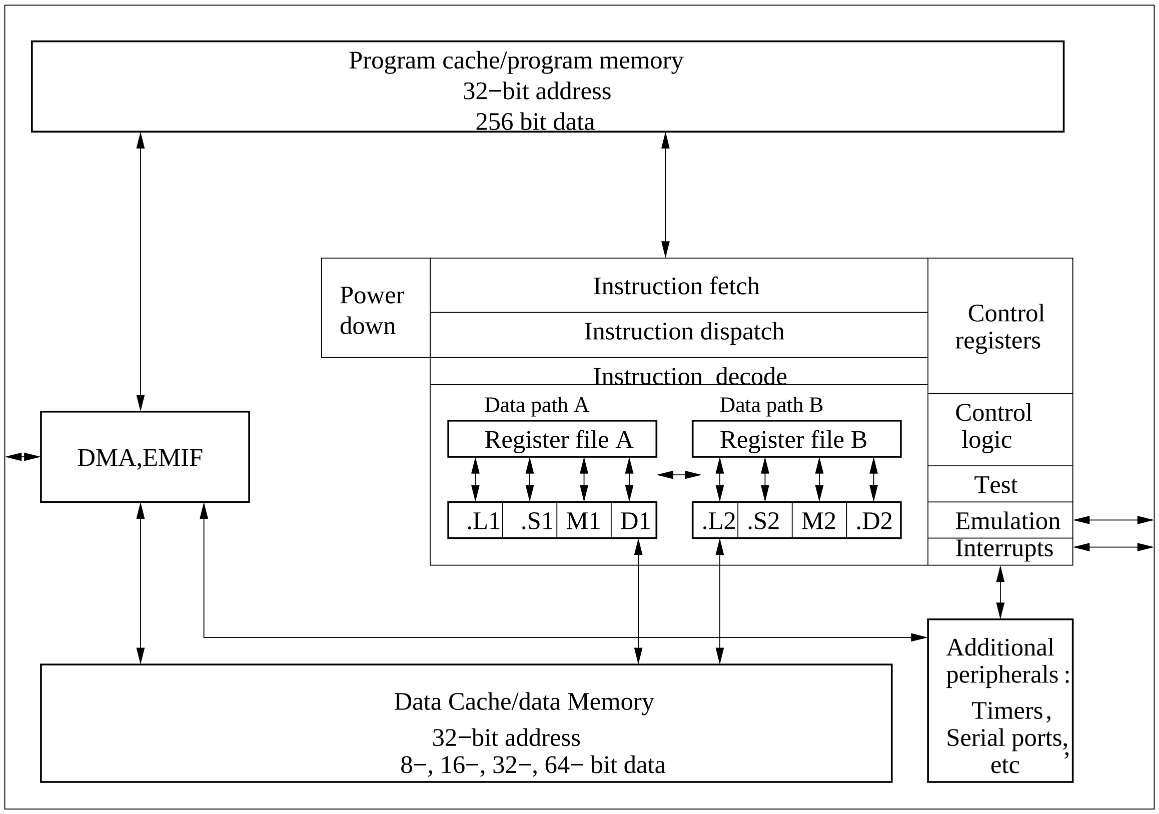


Figure 3.3: VelociTI Architecture Block Diagram

16x16 bit multiply has a 2 cycles latency while load has a latency of 5 cycles and branch instruction has a latency of 6 cycles.

Like the instruction set, the register file is also highly orthogonal. Any register can be an operand of any instruction or any type of functional unit. Each functional unit reads directly from and writes directly to the register file within its own data path. That is L1, S1, D1, and M1 units write to register file A and L2, S2, D2, and M2 units write to register file B. Each functional unit has two 32-bit read ports for the source operands src1 and src2 (one each) and a 32-bit write port into a general-purpose register file. The register files are connected to the functional units of the opposite-side register file via 1X and 2X cross-paths. These cross-paths allow functional units from one data path to access a 32-bit operand from the opposite side register file. The 1X cross-path allows the functional units of data-path A to read their source from register file B and the 2X cross-path allows the functional units of data path B to read their source from register file A. All eight functional units have access to the register file on the

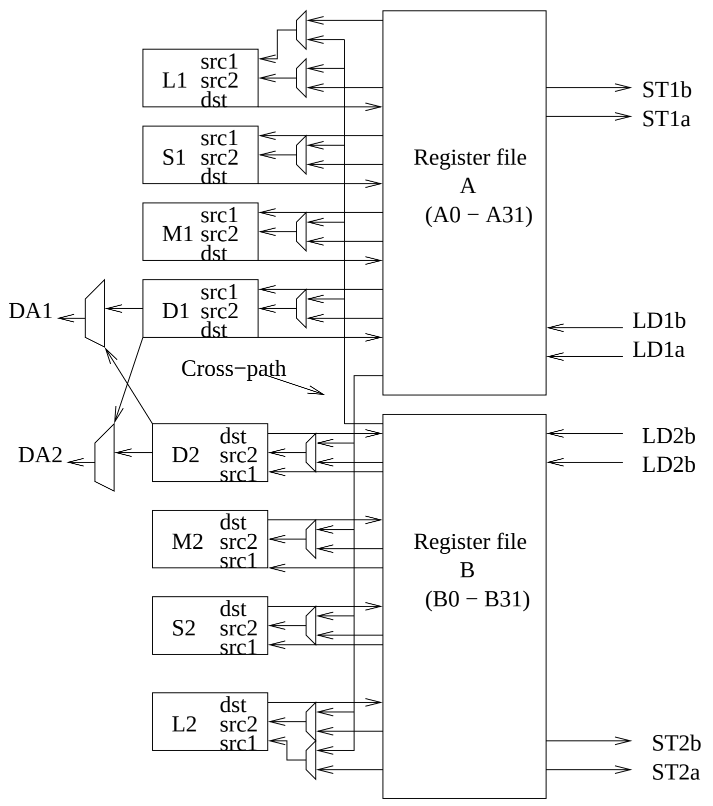


Figure 3.4: TMS320C64X CPU Data Paths

opposite side, via a cross-path. The src2 input of M1, M2, S1, S2, D1, D2 units are selectable between the cross-path and the register file on same side. In the case of L1 and L2, both src1 and src2 inputs are selectable between the cross-path and the register file on the same side . Only two cross-paths, 1X and 2X, exist in the C6X architecture. Thus, only two cross-path source reads per cycle are possible. There are four 32-bit paths (LD1a and LD1b for side A and LD2a and LD2b for side B) for loading data from memory to register files. Similarly there are four 32-bit paths (ST1a and ST1b for side A and ST2a and ST2b for side B) for storing register values to memory. The data address path DA1 and DA2 are each connected to the D unit in both the data paths. This allows data addresses generated by one path to access/store data to/from any register.

The pipeline is divided into three phases. The fetch phase covers four pipeline stages, the decode phase covers two pipeline stages and the execute phase covers 5 pipeline stages. Load and store follow the same pipeline flow and thus the result stored in one execute packet can be read by the next execute packet and this avoids read-after-write conflicts typically found in DSP processors. The VelociTI architecture lets the simplest of CPU instructions determine the cycle time of the processor. The critical path corresponds to the time for register-to-register ALU operation such as an ADD instruction. More complex instructions, such as multiply, are implemented with a one-cycle latency. To access high performance, synchronous on-chip memories, instruction fetch and data access are performed in multiple pipeline stages. The VelociTI pipeline is unprotected and fully exposed to the compiler. Run-time interdependencies such as pipeline interlocks between phases used in other DSPs are difficult to predict at compile time. The VelociTI CPU model at compiler time fully reflects the execution and completion order of instructions at run time[13].

3.2.2 Lx Architecture

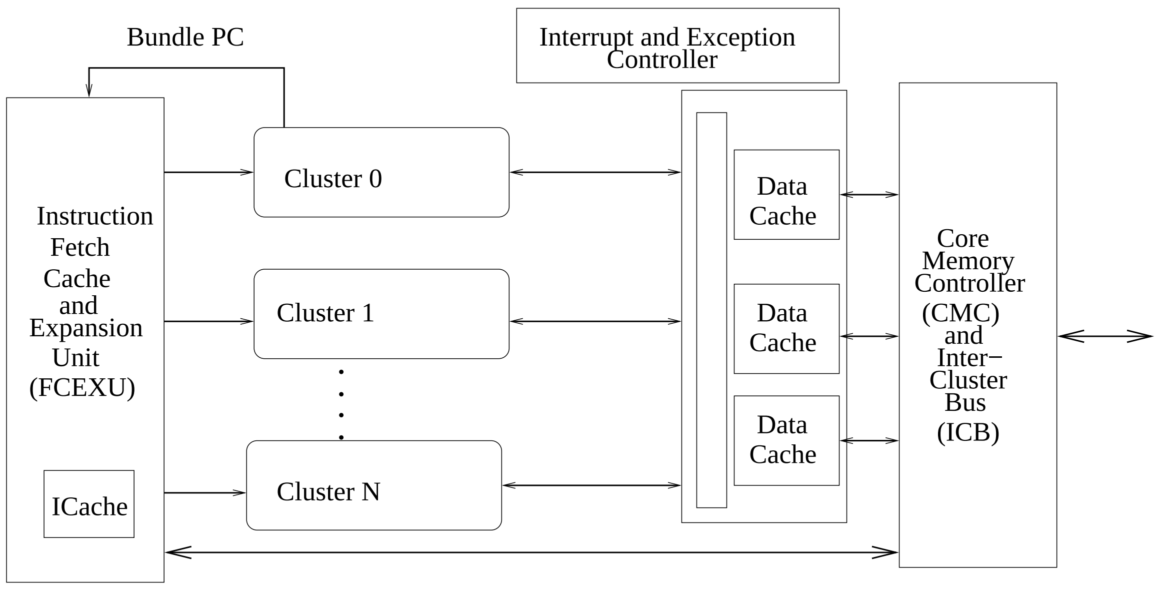

Bundle PC
Interrupt and Exception Controller
Instruction Fetch Cache and Expansion Unit (FCEXU)
ICache
Cluster 0
Cluster 1
Cluster N
Data Cache
Data Cache
Data Cache
Core Memory Controller (CMC) and Inter−Cluster Bus (ICB)

Figure 3.5: Lx Architecture Block Diagram

Lx is a statically scheduled VLIW architecture form HP Labs targeted to embedded applications[14]. Figure 3.5 depicts the block diagram of Lx architecture. Lx is scalable (with number of clusters varying between 1 to 4) and customizable to specific applications or application areas through the addition of application specific operations. Clusters are composed of a mix of register banks, constant generators, and functional units and can be heterogeneous having different unit/register mixes. However, all the clusters are controlled by a single program counter, driven by the same execution pipeline and fed synchronously from the same logical instruction cache so that they run in lockstep. Inter-cluster communication, is achieved by explicit register-to-register move. Lx defines a scalable and flexible communication mechanism based on a simple pair of send-receive instruction primitives that move values among registers.

An Lx cluster is a 4-issue VLIW core composed of four 32 bit integer ALUs, two 16x32 multipliers, one Load/Store unit, and one branch unit. The Cluster also includes 64 32-bit general-purpose registers and 8 1-bit branch registers (used to store branch conditions, predicates, and carries). The ISA consists of a simple RISC instruction set with minimal predication support through select instructions. The memory repertoire includes base+offset addressing, allows speculative execution and software pre-fetching. The control unit supports a two-step branch architecture, where compare and branch operations are decoupled and the compare-branch latency is exposed to the compiler. The ISA includes a complete set of compare and logic operations. There are no architecturally visible delay slots after the taken branch. It has a simple six-stage in-order pipeline: F D R E1 E2 W[14].

3.2.3 Other Clustered Architectures

Analog's TigerSHARC[15] is a register based load-store architecture. It has a mix of features from both VLIW and superscalar architectures and thus is also known as a static superscalar architecture. The TigerSHARC architecture implements a model where the program specifies instruction-level parallelism only, and the hardware dynamically resolves instruction timing. ILP is determined prior to run-time by a compiler or programmer as in a VLIW architecture. However in contrast to a VLIW architecture, it has features such as fully interlocked register files and all the computation and memory access instructions come with a regular two-cycle delay pipeline.

IBM's eLite[17] is a statically scheduled RISC-style clustered LIW DSP architecture designed for very low power implementations targeting algorithms found in communication standards, audio and video encoding/decoding, and voice-over-IP. It supports 3-wide ILP by allowing each LIW instruction packet to have one, two, or three variable size instructions whose individual size can be 16, 20, 24, 30, or 60 bits.

The RAW microprocessor[129] consists of identical tiles interconnected through a pipelined two dimensional mesh. Each tile has a RISC processor with a register file and a cache. The highly distributed RAW architecture is fully exposed to the RAW compiler for scheduling, communication, and synchronization.

3.3 Cluster Scheduling

Instruction scheduling involves reordering and grouping instructions in order to achieve performance benefits in terms of execution time and power consumption among others. It is performed on the low level intermediate code (after all machine independent optimization). The problem of scheduling becomes harder in the context of clustered architectures because a scheduler has to decide not only when to schedule (temporal concern) but also where to schedule (spatial concern). In a monolithic micro-architecture, the earliest time a data-ready operation can be scheduled for execution depends only on the availabilities of the functional unit and the resource to store the result. In contrast, the earliest time a data-ready operation can be scheduled on the clustered processor depends not only on the resource availability as in a traditional architecture but also on the delay in accessing the source operands. In other words, an operation can become data-ready at different times on different clusters depending on the proximity to the cluster(s) in which source operands are produced. In order to carry out effective cluster scheduling, a cluster scheduler is required to accurately gather and effectively utilize information regarding availability of operands in a cycle, availability of resources and cross-path in current and earlier cycles, as well as resource and cross-path requirements that may arise in the future. Essentially, effective scheduling of a clustered VLIW architecture demands resolving the conflicting goals of exploiting hardware parallelism as well as minimizing communication among clusters.

Earlier proposals for scheduling on clustered VLIW architectures can be broadly classified into two main categories. The phase-decoupled approach to scheduling[130][131][132] works on a data flow graph (DFG) and partitions instructions into clusters to reduce inter-cluster communication while approximately balancing the load among clusters. The annotated DFG is then scheduled using a traditional list scheduler while adhering to earlier spatial decisions. However, the phase-decoupled approach to scheduling suffers from the well known phase-ordering problem. A spatial scheduler has only an approximate knowledge (if any) of the usage of cross paths, functional units, and load on clusters. This inexact knowledge often leads to spatial decisions which may unnecessarily constrain a temporal scheduler and may lead to a suboptimal schedule. We follow an integrated approach to scheduling[133][43][134][135] that tries to combat the phase-ordering problem by combining spatial and temporal scheduling decisions into a single phase. A description of the earlier approaches for cluster scheduling is available in chapter 4. In what follows, we give a general high level description of the various steps involved in our scheduling approach.

3.3.1 Preprocessing

The preprocessing phase builds scheduling regions of the code. Different region formation techniques such as loop unrolling, superblock formation, and hyperblock formation aim to enhance the available ILP in the program in order to achieve better schedule. After region formation, the next step is to construct a dependence graph of the region with nodes as instructions and edges between nodes for dependences between the instructions. The dependence graph is preprocessed to determine various topological directed acyclic graph (dag) properties such as height of a node, depth of a node, the number of successors of a node, and the static slack associated with a node. These properties are stored in each node and referred during scheduling. In some cases, properties such as available slack in scheduling an instruction and the number of scheduled successors of an instruction are dynamically determined. The root node of the dependence graph is used for initializing the ready-list which contains instructions ready to be scheduled in a cycle.

3.3.2 Instruction Selection

Each scheduling step involves instruction selection from the ready-list for scheduling based on a priority function. Selection of instructions from the ready list can be based on any priority order derived from different DAG properties[136] such as :

- *Earliest start time (EST)* of an instruction which is defined as the earliest time step at which an instruction can be scheduled without violating any dependency in the DFG.
- *Latest start time (LST)* of an instruction which is defined as the latest scheduling step of the instruction such that a valid minimum length schedule can still be achieved on an infinite resource machine.

- *Slack or mobility* of an instruction which is defined as the difference between EST and LST.
- Maximum distance of an instruction from the sink node of DFG.
- Number of children of an instruction.

We select instructions from the ready-list based on the scheduling slack of an instructions and the number of successors of an instruction. After one scheduling step is over, the ready-list is updated to include new instructions that are ready to be scheduled. The dynamic properties of instructions such as dynamic scheduling slack and the number of unscheduled successors are determined when an instruction is inserted to the ready-list.

3.3.3 Cluster Assignment

Instruction selection is followed by selection of the cluster to which an instruction can be assigned. The scheduler determines all the alternatives for scheduling an instruction on different clusters taking into account the availability of functional units and cross-paths. The selection of a cluster for binding an instruction from a set of feasible clusters is done based on parameters such as the earliest time at which an instruction can be scheduled on a cluster and/or the communication cost of scheduling an instruction on a particular cluster, depending on the objective of the scheduler. The communication cost metric takes into account the communications that happen in the current cycle as well as in future. Figure 3.6 presents an example to illustrate the current and future communications that happen because of a binding. Let us assume that V1 is mapped to cluster 1 (shown with a light circle) and V2 and V3 are mapped to cluster 2 (shown with dark circles). If a scheduling decision is taken to bind V4 on cluster 1, these are the communications that will be required as a side effect of this binding:

1. Communicate V2 from cluster 2 to cluster 1 using inter-cluster move instruction in one of the earlier cycles.
2. As a side effect of binding V4 on cluster 1, V5 will incur a communication in future no matter whether it is bound to cluster 1 or cluster 2. This is because V3, the other parent of V5 is already bound to cluster 2. This is the required future communication as a side effect of the binding as shown with a darker arrow from V3 to V5.

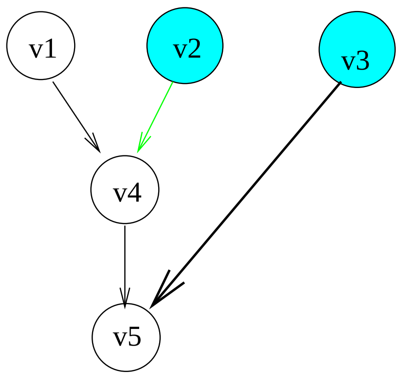


Figure 3.6: Communications Required as a Result of a Cluster Assignment

A more detailed description on calculation of communication cost and the cluster selection criteria is available in later chapters.

3.3.4 Functional Unit Assignment

Assignment of functional units is not completely independent of cluster assignment. Many times a cluster which has an active functional unit to schedule an instruction is preferred over a cluster which requires explicitly activation of a functional unit. In other situations, the selection among available functional units within a cluster is done based on how long a functional unit has been in sleep mode. Depending on the objective of the scheduler, the cluster assignment decision is made based on the availability of active functional units and/or communication cost of scheduling an instruction in a cluster.

3.3.5 Cross-path Assignment

A cross-path assignment scheme decides the cross-path to be used for transferring the data needed by an instruction from other clusters to the target cluster (where the instruction would execute). The scheme in general tries to minimize the cost of transfer by choosing the minimum energy cross-path that can transfer the data values without explicitly stretching the schedule. It is important to note that cross-path scheduling is only relevant for heterogeneous interconnect clustered VLIW architectures which offers different cross-paths varying in terms of latency and energy cost. A detailed description of heterogeneous interconnect clustered VLIW processors and cross-path assignment schemes is available in later chapters.

3.4 Trimaran Framework

We have used the Trimaran suite[137] for our experimentation. Trimaran was developed to conduct state-of-the-art research in compilation techniques for ILP architectures with a specific focus on VLIW class of architectures. Figure 3.7 depict our research framework built on trimaran infrastructure. Trimaran infrastructure is basically composed of three major components namely IMPACT front-end, ELCOR backend and simulator. IMPACT performs typical front end related functionalities such as parsing, type checking, and a large suite of high-level (i.e. machine independent) classical optimization. Apart from classical optimizations, IMPACT also performs lot of instruction-level parallelism enhancing optimizations followed by forming scheduling regions. The transformed and optimized C program is then passed to back-end called ELCOR in the form of a textual intermediate representation called Rebel. ELCOR internally represent the program using a dependence graph based intermediate representation which is suitable for performing various machine dependent optimization. ELCOR is fully parameterized using an extensible machine description called MDES. The machine description MDES provides facility to specify the target machine attributes such as instructions, their latencies and resource requirements and number and type of registers. The machine description is queried to perform machine dependent optimization such as instruction scheduling, and register allocation using the dependence graph representation of the program. The final component is a cycle-level simulator which is configurable by a machine description and provides run-time information on execution time, branch frequencies, and resource utilization.

We have modified the Trimaran suite to generate and simulate code for a variety of clustered VLIW configurations. The machine description module has been upgraded to describe various clustering related parameters such as the number of clusters, number and types of functional units in each cluster, interconnection network parameters such as number and types of cross-paths between different clusters, and their latency parameters. These parameters are fed to the parameterized machine-dependent optimization modules in the back-end. Major modifications have been performed in the Trimaran scheduler and register allocator module (which was originally written for a class of flat VLIW architectures) to faithfully account for the conflicts due

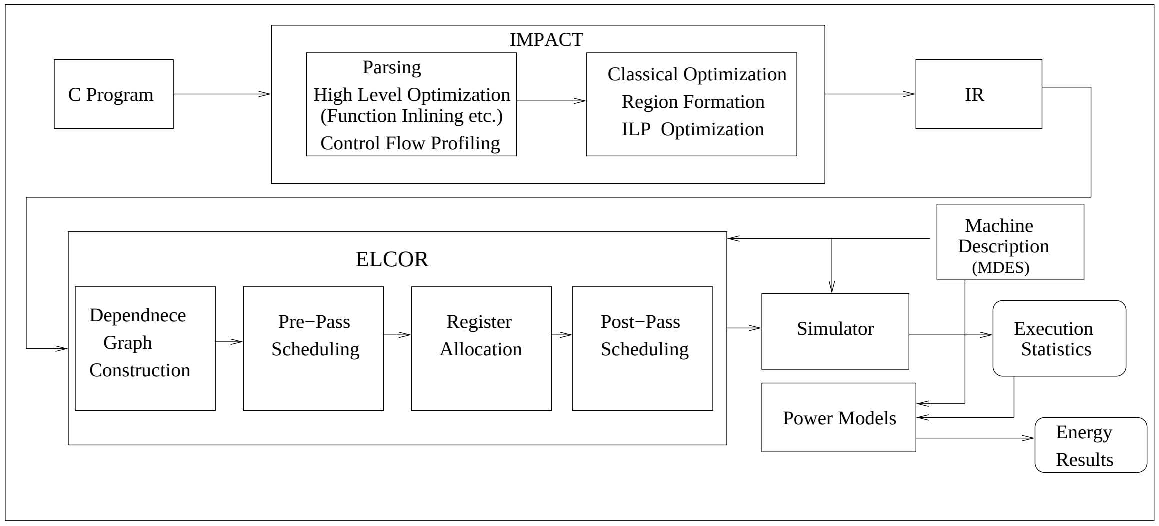


Figure 3.7: Implementation Framework on Trimaran Infrastructure

to limitations on the number of available functional units and registers in a cluster as well as the limitations on the number of available cross-paths between clusters. The scheduler has been modified to implement the scheduling algorithms described in later chapters. The execution statistics obtained from trimaran simulator and parameters of the target machine as obtained from machine description are used in power models to obtain the energy results.

3.5 Experimental Setup

This section describes our general experimental setup. The specific details of experiments (if any) can be found in individual chapters along with performance results. We have used twelve benchmarks out of which nine are from mediabench[138][139] *(viz. cjpeg, djpeg, rawcaudio, rawdaudio, g721encode, g721decode, md5, des, and idea)*, two from netbench[140][141] *(viz. crc, and dh)*, and one *(susan)* is from MiBench[142][143]. Table 3.1 gives the description of different benchmark programs. We have tried other benchmarks from these suits as well but these are the only ones which compiled successfully and executed correctly in the Trimaran framework.

The results are presented for an unclustered flat VLIW, a two-cluster VLIW, and a four-cluster VLIW machine. Table 3.2 gives latency of different operations. Table 3.3 gives the configuration of different clustered VLIW architectures. The unclustered VLIW configuration

Table 3.1: Description of Benchmark Programs

Benchmark Suit	Program	Description
Mediabench	cjpeg	Image Compression
	djpeg	Image Decompression
	rawcaudio	Adpcm Encoding
	rawdaudio	Adpcm Decoding
	g721encode	Speech Encoding
	g721decode	Speech Decoding
	md5	Message Digest
	des	Data Encryption Standard
	idea	Block Cypher
Netbench	crc	Checksum calculation
	dh	Diffie-Hellman Encryption Decryption
MiBench	susan	Image Recognition

Table 3.2: Latencies of Operations

Integer	1
Integer Multiply	3
Integer divide	8
FP	3
FP Multiply	3
FP divide	8
Load	3
Store	1
Branch	3
Inter-Cluster Move	1/3

Table 3.3: Clustered VLIW Configurations

Base VLIW	4 ALUS, 2 Load-store units, 1 Branch unit, 64 Registers
2-Clustered VLIW	2 ALUS, 1 Load-store units, 1 Branch unit, 32 Registers
4-Clustered VLIW	1 ALUS, 1 Load-store units, 1 Branch unit, 16 Registers

has 4 ALUs, 2 load-store units, 1 branch unit, and 64 registers. The 2-clustered configuration has 2 ALUs, 1-load store units, 1 branch unit, and 32 registers in each cluster, whereas the 4-clustered configuration has 1 ALU, 1-load store unit, 1 branch unit and 16 registers in each. The inter-cluster cross-paths allow communication of two data values between clusters in each cycle. The latency of inter-cluster move operation can be 1 cycle or 3 cycle depending on the cross-path used. This is further explained in detail in chapter 5 and chapter 6.

Chapter 4

Leakage Energy Management for Functional Units

4.1 Introduction

The ongoing improvements in the semiconductor technology bring along various challenges[144]. One such challenge is the rising level of the leakage energy consumption in the logic. The transistor density doubles every eighteen months by packing more logic into the same area. However, this increase in the transistor density requires reducing the supply voltage in order to operate the circuit reliably. The reduction in supply voltage also requires reduction in the threshold voltage in order to maintain the speedup and this leads to an exponential rise in the leakage component of the energy consumption[7]. With the 65nm and smaller technologies currently in fabrication, the leakage energy is on par with the dynamic energy consumption. In future technologies, the leakage energy will further dominate the overall energy consumption[8].

VLIW and clustered VLIW architectures rely on compile-time scheduling. This simplifies the issue logic by alleviating the need for a dedicated hardware for scheduling. Thus, a significant fraction of the total leakage energy consumption in VLIW architectures is attributed to components such as functional units and instruction decoder. The frequent access of these components raises the temperature level and makes the leakage energy consumption even worse. Though, the exact percentage depends upon the architecture and circuit details, earlier studies report that 30% to 35% of the static energy conumption in a VLIW architecture is attributed to functional units[25]. An architecture level model developed in [26] also confirms that the leakage energy consumption in functional units constitutes a noticeable fraction of the overall processor leakage energy consumption despite having a smaller transistor count compared to the caches. Thus, optimizing leakage energy in functional resources is becoming more important by each process generation.

However, these architectures are often designed targeting embedded domains where the real-time performance is of utmost importance. Thus, the design is often optimized for the peak performance and as a result, the functional units are underutilized due to the inherent variations in the ILP of the programs. Clustered VLIW architectures improve over the VLIW architectures by solving the scalability problem (in order to obtain a better clock rate) by distributing functional units among different clusters[11]. However, contentions for the limited number of slow inter-cluster communication channels introduce many short idle cycles and makes the utilization of functional units even worse. This further exacerbates the leakage energy consumption problem in clustered VLIW architectures specially in smaller technologies.

The underutilization of functional resources can be exploited to reduce leakage energy consumption. Some earlier work in this area reports leakage energy management at a coarser granularity of loop level[25] or block level[35]. However, the rising level of leakage energy in current and future process technologies requires aggressive leakage energy management even for short idle periods. One such purely hardware based scheme in the context of a superscalar architecture is due to Albonesi et al.,[34]. Their scheme utilizes the unique characteristics of dual-threshold domino logic with sleep mode that can transition between active mode and sleep mode without any performance penalty[30]. However, such a fast transition incurs moderate amount of energy penalty. Their scheme puts any integer ALU into low leakage mode after one cycle of idleness. Their results confirm the benefits of such an aggressive scheme. However, being a purely hardware based scheme, the benefits are severely (on average, by 30%) affected by frequent transitions from active mode to sleep mode and vice-versa because of many short idle periods.

In this chapter, we propose and evaluate our compiler-directed instruction scheduling algorithm that assists such a hardware based scheme in achieving better energy savings in the context of VLIW and clustered VLIW architectures. Though our primary focus is clustered VLIW architectures, we also present results of applying the proposed scheduling technique in the context of flat VLIW architectures for the sake of completeness and comparison. Whereas the hardware scheme suffers from a limited program view, a compiler can analyze whole program regions and is capable of orchestrating the mapping between operations and functional units in the context of VLIW and clustered VLIW architectures. The proposed scheme exploits the scheduling slacks of the instructions to maximize the simultaneous idle time and usage of functional units, thereby reducing the number of transitions drastically. This reduction in the number of transitions leads to significant improvements in the energy savings over those obtained by a purely hardware based scheme. Moreover, since the proposed scheme keeps a limited number of functional units active and use them as much as possible, it generates a more balanced schedule which helps to reduce the peak power and the step power[37]. The proposed scheme is particularly targeted toward embedded clustered VLIW processors where the absence of scheduling hardware and the small caches require more aggressive energy management techniques for other dominant sources of energy consumption such as functional units and instruction decoders. The simplification of the components in clustered VLIW architectures makes dynamic energy consumption a less of a concern whereas increasing significance of leakage in smaller technologies demands aggressive leakage energy management in functional units.

The rest of the chapter is organized as follows. Section 2 provides a motivation for this work along with some quantitative results. Section 3 describes our new instruction scheduling algorithm and presents an example to show the benefits of the proposed scheme. Section 4 provides detailed experimental results and analysis. Section 5 presents the application of the proposed technique and experimental results in the context of instruction decoder. Section 6 describes the earlier work in the area of instruction scheduling for clustered VLIW architectures, architectural approaches for energy management, and energy-aware scheduling for VLIW architectures. Section 7 concludes this chapter with a summary.

4.2 Motivation

The VLIW and clustered VLIW class of architectures are in widespread use in the embedded domain. The primary reason for their success in this domain is high level of ILP in the embedded workload and the suitability of a compiler scheduling algorithm to map this explicit ILP to available hardware. In order to satisfy the demand for high performance in embedded

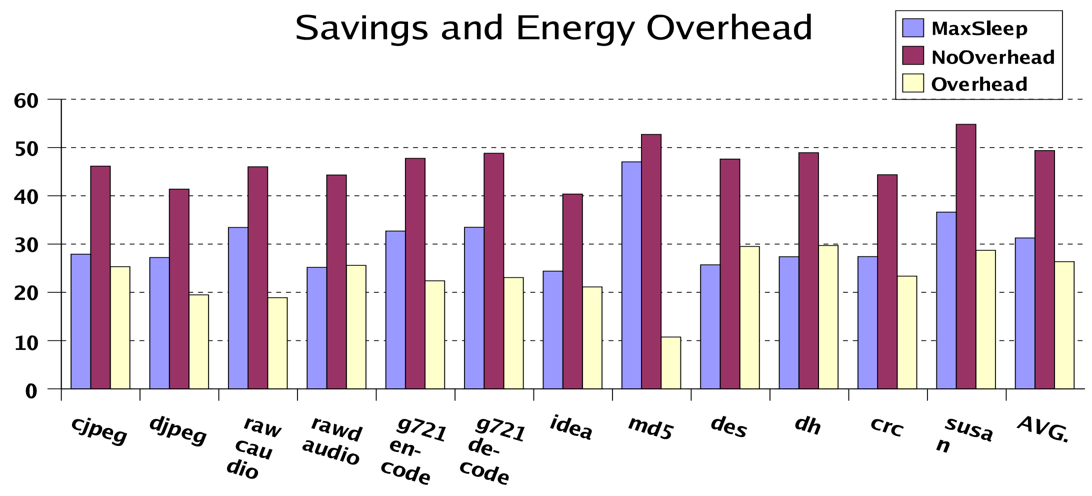


Figure 4.1: % Savings for 'MaxSleep' and 'NoOverhead' Policies

applications (most of which are real-time applications), these architectures use more and more number of functional units. However, the inherent variations in the ILP of the programs lead to underutilization of functional units. We observe that integer ALUs are idle for 60% of the time on an average for a collection of media benchmarks. This idleness figure is for a VLIW configuration having only a moderate number of ALUs so as to achieve 95% of the peak performance (details of our experimental setup and energy model appear in a later section). The idleness is even more pronounced for a clustered VLIW configuration because of the contention for a limited number of slow interconnects which manifests itself in the form of many short idle cycles.

A hardware based scheme such as 'MaxSleep' proposed in [34] puts a functional unit into low leakage mode after an idleness of one cycle and thus saves leakage energy (refer the related work section for a description on 'MaxSleep'). However, if there are many short idle cycles then there are many transitions and the transition overheads adversely affect the benefit gained by such a scheme. Figure 4.1 presents the energy savings obtained by a 'MaxSleep', energy savings obtained by a 'NoOverhead' scheme which is a hypothetical scheme (same as 'MaxSleep') but does not incur any transition energy overheads and % energy overhead of 'MaxSleep' due to transitions as compared to that of 'NoOverhead' scheme for integer ALUs in a 2-cluster configuration. These results clearly indicate that the 'NoOverhead' scheme is able to achieve an average savings of 50% in total energy, where as the average savings for 'MaxSleep' is only

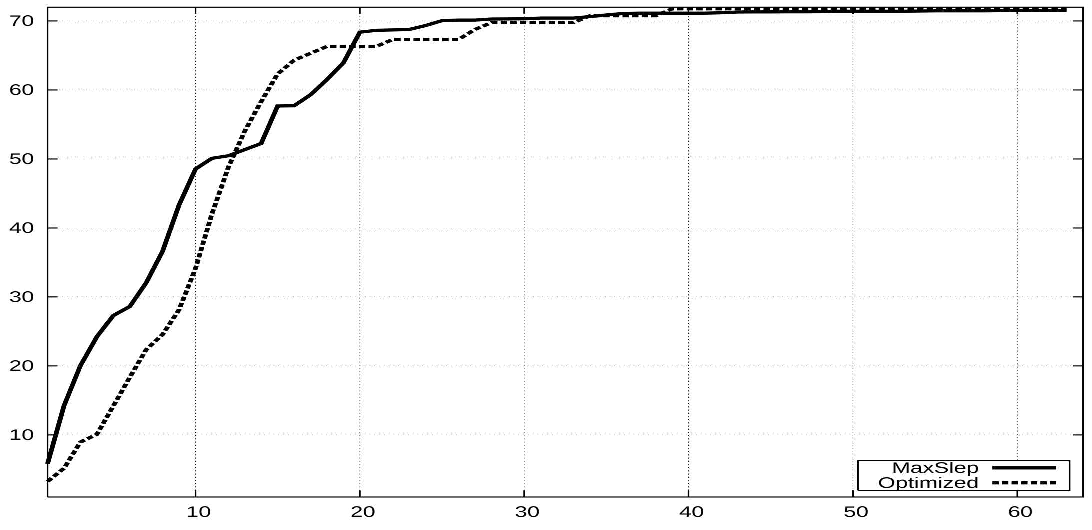


Figure 4.2: % Cumulative Distribution of Idle Cycles

31%. 'MaxSleep' has an average energy overhead of 26% (due to transitions) as compared to the 'NoOverhead' scheme. These results are also in agreement with the results presented in [34]. Thus, reducing the number of transitions will increase the idleness duration for functional units and improves the energy benefits of a hardware based scheme.

Motivated by this, we have developed a scheduling algorithm in the context of VLIW and clustered VLIW architectures that leverage the available slack in scheduling instructions in order to keep the idle functional units idle for a longer duration while maximizing the utilization of active functional units. Figure 4.2 shows the average cumulative distribution of idle cycles in integer ALUs for a 2-clustered machine on our collection of benchmarks. The graph for 'MaxSleep' clearly shows many small idle cycles constitute a large percentage of overall cycles. 50% of total 71% idle cycles have a duration less than or equal to 10 cycles. The graph after applying our scheduling scheme is shown with title 'Optimized'. This shows that the many small idle cycles have been converted to large idle cycles by reducing transitions and only 34% of overall idle cycles are now less than 10 cycle. Idle cycle of length between 10 to 20 cycles constitute 32% of total idleness for 'Optimized' scheme while for the 'MaxSleep' scheme this is only 18%. This clearly shows that our scheme is able to exploit the slack to reduce the number of transitions thereby increasing the duration of idle periods.

Algorithm 1 Energy Efficient Scheduling for Functional Units

```
1: if (Scheduling for a clustered configuration) then
2:    $ClusterScheduling \leftarrow 1$
3: end if
4: Initialize ReadyList with root operations of the dependence graph of the
        region to be scheduled
5: $CurrentCycle \leftarrow 0$
6: while (ReadyList is not empty) do
7:    Initialize EarlyCycle with CurrentCycle, and LateCycle with SchedulingCycle de-
      termined using performance oriented scheduling
8:    $slack = LateCycle - EarlyCycle$
9:    while (Not all operations in ReadyList have been tried once) do
10:      $(CurrentOperations \leftarrow UnSchedList.pop())$
11:      $AlternativeList \leftarrow DetermineSchedulingAlternatives($
                                   $CurrrentOperation, ClusterScheduling)$
12:      if $(IsEmpty(AlternativeList))$ then
13:         CONTINUE
14:      end if
15:      $TargetCluster \leftarrow 0$
16:      $SUCCESS \leftarrow FALSE$
17:      if (ClusterScheduling) then
18:         $TargetCluster \leftarrow DetermineBestCluster(CurrentOperation)$
19:         $AlternativeList \leftarrow DetermineSchedulingAlternatives($
                                   $CurrentOperation, TragetCluster)$
20:      end if
21:      while $(CurrentAlternative = AlternativeList.pop())$ do
22:         if (FU in CurrentAlternative are active) then
23:            Schedule CurrentOperation using CurrentAlternative in CurrentCycle on
               TargetCluster.Cluster using TragetCluster.CommOption
24:            $SUCCESSS \leftarrow TRUE$
25:         end if
26:      end while
27:      if $(!SUCCESS\ and\ Slack \leq SLACK\_THRESHOLD)$ then
28:         $FallBackAlternative \leftarrow DetermineBestAlternative($
                                   $AlternativeList, TargetCluster)$
29:         Wakeup FU in FallBackAlternative
30:         Schedule CurrentOperation using FallBackAlternative in CurrentCycle on
            TragetCluster.Cluster using TragetCluster.CommOption
31:      else
32:         ReadyList.add(CurrentOperation)
33:      end if
34:   end while
35:   $CurrentCycle \leftarrow CurrentCycle + 1$
36:   $ReadyList.update()$
37:   $updateFUStatus()$
38: end while
```

Procedure 2 DetermineBestCluster

```
1: $FirstTarget.Cluster \leftarrow -1$
2: $FirstTarget.CommCost \leftarrow 1000000$
3: $FirstTarget.CommOption \leftarrow NULL$
4: $SecondTarget.Cluster \leftarrow -1$
5: $SecondTarget.CommCost \leftarrow 1000000$
6: $SecondTarget.CommOption \leftarrow NULL$
7: for (CurrentCluster ranging from FirstCluster through LastCluster) do
8:   Compute the Cross-path Requirements in CurrentCommOption
9:   Compute the Communication Cost in CurrentCommCost
10:  if (FU and Cross-paths required by CurrentOperation are available in CurrentCycle
     for CurrentCluster) then
11:    if (FU under consideration is in active mode and FirstTarget.cost > Current-
       CommCost) then
12:      $FirstTarget.CommCost \leftarrow CurrentCommCost$
13:      $FirstTarget.CommOption \leftarrow CurrentCommOption$
14:      $FirstTarget.Cluster \leftarrow CurrentCluster$
15:    else
16:      if ($SecondTarget.cost > currentCommCost$) then
17:        $SecondTarget.CommCost \leftarrow CurrentCommCost$
18:        $SecondTarget.CommOption \leftarrow CurrentCommOption$
19:        $SecondTarget.Cluster \leftarrow CurrentCluster$
20:      end if
21:    end if
22:  end if
23: end for
24: if ($FirstTarget.cluster != -1$) then
25:   RETURN $FirstTarget$
26: else
27:   RETURN $SecondTarget$
28: end if
```

4.3 The Scheduling Algorithm

The Elcor backend of the Trimaran infrastructure has a cycle scheduling algorithm designed and implemented for flat VLIW architectures[137][145]. We have modified this algorithm to perform leakage energy optimization for VLIW as well as clustered VLIW architectures. Another loop has been added inside the main scheduling loop of the cycle scheduler to perform cluster scheduling in an integrated fashion. The integrated approach[43][134][135] to cluster scheduling makes the cluster assignment decision during temporal scheduling. This is in contrast to phase-decoupled approaches[130][131][132] which perform cluster assignment prior to temporal scheduling. Essentially, our integrated scheduling algorithm for leakage energy optimization consists of the following main steps.

1. Prioritizing the ready instructions

2. Assignment of a cluster to the selected instruction

3. Assignment of functional unit to selected instruction in target cluster

In what follows, we describe how each of these step is performed in our algorithm. An outline is shown in Algorithm 1.

4.3.1 Prioritizing the Ready Instructions

Instructions in the ReadyList are prioritized using a priority function that uses instruction slack and number of consumers of the instruction. Instructions with less slack should be scheduled early and are given higher priority over instruction with more slack to avoid unnecessary stretching of the schedule. Instructions with the same slack values are further ordered in the decreasing order of the number of consumers. An instruction with a large number of successors is more constrained in the sense that its spatial and temporal placement affects scheduling of more number of instructions and hence should be given higher priority. Giving preference to an instruction with many dependent instructions also enables better future scheduling decisions by uncovering a larger portion of the graph.

Scheduling slack of an instruction is defined as the difference between the earliest start time and the latest finish time of the instruction. Traditionally, slack is determined statically during dependence graph analysis before the scheduling begins, assuming a machine with infinite resources of each type. This calculation is inherently pessimistic as any real machine will have contentions for resources which prolongs the execution time. Since our algorithm exploits slack of instructions to delay their execution in order to save energy without affecting performance, a better quantification of available slack is of utmost importance. We quantify the slack of instructions while scheduling a region for the specific target machine by taking resource constraints into account. We first schedule the instruction for a base VLIW configuration using a simple cycle-by-cycle scheduler. The schedule time of the instructions is stored during this phase. In the second phase, this schedule time (Late cycle) is used to determine the slack of the instruction. In our implementation, slack is dynamically updated for all the operations in the ready list after every cycle. The earliest schedule time of an instruction is set to the current cycle, before scheduling for the current cycle begins (Early cycle). The slack is then determined as a difference of the Early cycle and the Late cycle. The dynamic update of slack after each cycle ensures that any consumed slack is taken into account while scheduling instructions in the future cycles.

4.3.2 Cluster Assignment

Once an instruction has been selected for scheduling, we make a cluster assignment decision. The primary constraints are:

- The chosen cluster should have at least one free resource of the type needed to perform this operation.
- Given the bandwidth of the channels among clusters and their usage, it should be possible to satisfy the communication needs of the operands of this instruction on the cluster by scheduling these communications in the earlier cycles (so that operands are available at the right time).

Note that if we are scheduling for a plain VLIW architecture with no clustering, we assume that there is only one cluster (numbered 0) that is holding all the resources and the same algorithm is used [1]. Selection of a cluster from the set of the feasible clusters is done as follows. A cluster with an active functional unit of the type needed to schedule the operation is given preference.

[1]Procedure *DetermineSchedulingAlternatives* returns all possible alternatives for scheduling operation under consideration.

If no such cluster is available or more than one such cluster is available, the one which reduces the communication cost gets preference (as explained in pseudo code of Procedure 2).

The communication cost is computed by determining the number and type of communications needed by a binding in the earlier cycles as well as the communication that will happen in the future. Future communications are determined by considering the successors of this instruction which have one of their parents bound on a cluster different from the cluster under consideration. This is due to the fact that if the instruction is bound to the cluster under consideration, it will surely lead to communication(s) in the future while scheduling the successors of the instructions. Although, we have experimented with many other heuristics for cluster assignment, the above mentioned heuristic seems to generate the best schedule in almost all cases[134].

4.3.3 Functional Unit Binding

A functional unit binding scheme decides the binding of a chosen instruction to a functional unit. The algorithm maintains a FU map that explicitly keeps track of the status of each functional unit. A functional unit is marked to be in sleep mode after one cycle of idleness and activated on next use.

If the functional unit required for the instruction under consideration is active in the target cluster, it is bound as usual. otherwise, the available slack of the instruction is considered. If the slack is below a threshold (we use the threshold value of 0 in our experiment) the functional unit required by the instruction is woken up. In case there is more than one alternative available (for activating), the functional unit which is in sleep mode for a longer duration (as returned by Procedure *DetermineBestAlternative*) is woken up in order to amortize the cost of waking up. In case the instruction possesses enough slack, its scheduling is deferred to a future cycle and it is put back in the ReadyList. Note that the next time this instruction is picked up for scheduling, its earliest scheduling time and hence the slack get updated. This guarantees that the slack of an instruction reduces monotonically and eventually comes below the threshold ensuring that it is scheduled. Hence the algorithm is guaranteed to terminate.

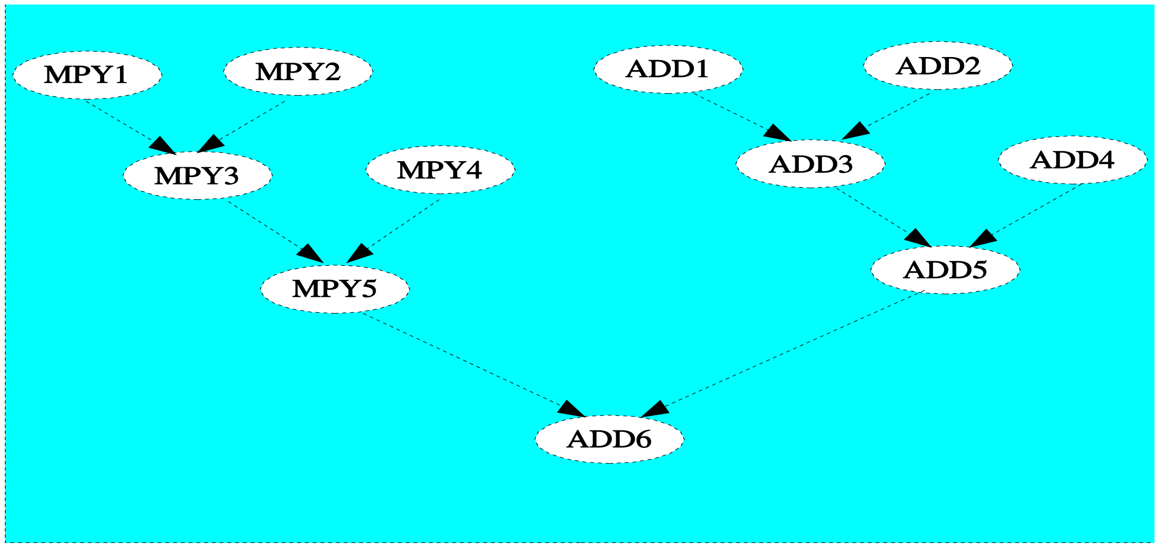
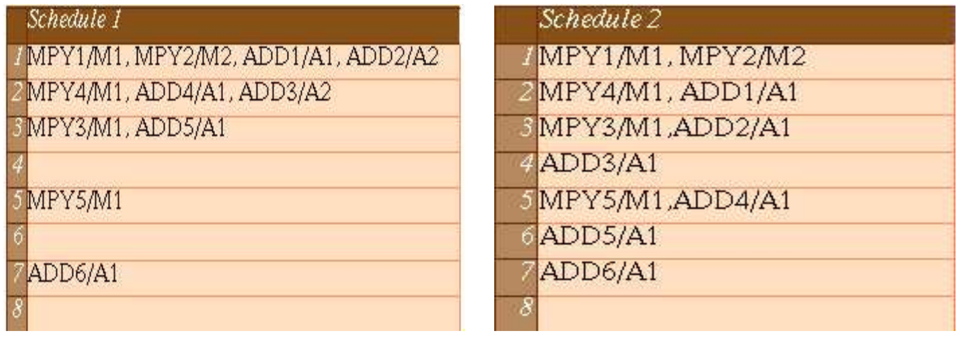


Figure 4.3: An Example Data Dependency Graph

4.3.4 An Example

In this subsection, we present an example to illustrate how the available slack of instructions is exploited by the proposed scheduling algorithm to get energy benefits without hurting performance. Figure 4.3 shows an example data dependency graph and Figure 4.4 and Figure 4.5 shows some schedules. Let us first discuss schedule 1 and schedule 2 for a plain VLIW architecture having two adders (namely A1 and A2) and two multipliers (namely M1 and M2) both of which are pipelined. We assume that the latency of an add operation is one cycle and the latency of a multiply operation is two cycles. Schedule 1 is generated by a traditional performance-oriented scheduler which schedules the instructions as early as possible and uses the slack value of instructions to break any contentions for resources and the total schedule length is 8 cycles.

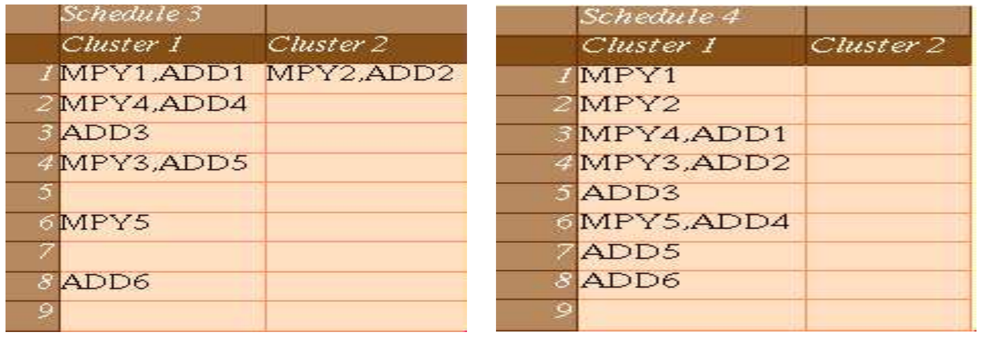
Our energy efficient scheduler realizes the criticality of MPY operations and available slack for ADD operations and schedules the same data dependence graph as shown in schedule 2. Since deferring the execution of any MPY operation leads to stretching of schedules, they are scheduled in the same way as in the performance-oriented schedule 1. However, the scheduling of ADD operations is delayed as well as serialized, capitalizing on available slack of add operations. Notably, the scheduler determines the slack value available in scheduling an operation by first doing a performance-oriented scheduling pass on data-dependence graph and uses the estimate of schedule length from this pass to calculate the exact slack value available in scheduling an



	Schedule 1
1	MPY1/M1, MPY2/M2, ADD1/A1, ADD2/A2
2	MPY4/M1, ADD4/A1, ADD3/A2
3	MPY3/M1, ADD5/A1
4	
5	MPY5/M1
6	
7	ADD6/A1
8	

	Schedule 2
1	MPY1/M1, MPY2/M2
2	MPY4/M1, ADD1/A1
3	MPY3/M1,ADD2/A1
4	ADD3/A1
5	MPY5/M1,ADD4/A1
6	ADD5/A1
7	ADD6/A1
8	

Figure 4.4: Schedules for VLIW Architecture (a) Schedule 1 (b) Schedule 2



	Schedule 3	
	Cluster 1	Cluster 2
1	MPY1,ADD1	MPY2,ADD2
2	MPY4,ADD4	
3	ADD3	
4	MPY3,ADD5	
5		
6	MPY5	
7		
8	ADD6	
9		

	Schedule 4	
	Cluster 1	Cluster 2
1	MPY1	
2	MPY2	
3	MPY4,ADD1	
4	MPY3,ADD2	
5	ADD3	
6	MPY5,ADD4	
7	ADD5	
8	ADD6	
9		

Figure 4.5: Schedules for Clustered VLIW Architecture (a) Schedule 3 (b) Schedule 4

instruction which is used to generate the schedule for energy efficiency. Schedule 2 takes same execution cycles as schedule 1 but is better than schedule 1 in many ways. First of all, schedule 2 make use of only one adder compared to schedule 1 which clearly reduces the leakage energy consumption because the second adder is always in low-leakage mode. Secondly, there are fewer transitions from active to low leakage mode and vice versa in schedule 2 as compared to schedule 1. Assuming availability of hardware mechanisms that put a functional unit into low leakage mode after 1 cycle of idleness, the number of transitions from active mode to low leakage mode and vice-versa for M1, M2, A1, and A2 are 2, 2, 4, and 2 for schedule 1 and 2,2,2, and 0 for schedule 2. Schedule 2 is also more balanced as compare to schedule 1 in terms of resource usage. The resource usage vector of the first schedule is (4,3,2,0,1,0,1,0) and that of second is (2,2,2,1,2,1,1,0). Thus, cycle to cycle variation in resource usage is clearly reduced in schedule 2 as compared to schedule 1. Which in turn helps in reducing step power and peak power dissipation [37]. Thus, it is clear that the proposed scheme is capable of reducing leakage energy consumption, transition energy overheads, as well as peak power and step power dissipation without affecting the performance.

Consider schedules 3 and 4 generated for a 2-clustered VLIW architecture (equivalent to above mentioned VLIW architecture) having 1 adder and 1 multiplier in each cluster and a bidirectional cross-path between the two clusters with 1 cycle transfer latency. Schedule 3 is generated by a performance-oriented scheduler. The extra delay of inter-cluster communication stretches the schedule from 8 cycles to 9 cycles as compared to the corresponding VLIW schedule 1. ADD3 (MPY3 resp.) is scheduled in cycle 3 (cycle 4 resp.) because it takes a cycle to transfer the result of ADD2 (MPY2 resp.) from cluster 2. The Total schedule length is 9 cycles.

Scheduling the same set of operations using our energy-efficient scheduler generates schedule 4. The major point to note is that the scheduler leverages the available slack due to inter-cluster communication to map all the operation to just cluster 1, keeping cluster 2 completely idle, thereby saving even more leakage energy. The number of transitions from active mode to low leakage mode and vice-versa for M1, M2, A1, and A2 are 2, 2, 4, 2 for schedule 1 and 2, 0, 2, 0 for schedule 2 respectively. Finally schedule 2 is much more balanced : The resource usage vector of first schedule is (4,2,1,2,0,1,0,1,0) and that of the second is (1,1,2,2,1,2,1,1,0).

4.4 Experimental Evaluation

We present results for an unclustered, a two-cluster machine and a four-cluster VLIW machine. The unclustered VLIW configuration has 4 ALUs, 2 load-store units, 1 branch unit, and 64 registers. The 2-clustered configuration has 2 ALUs, 1-load store units, 1 branch unit and 32 registers in each cluster, whereas the 4-clustered configuration has 1 ALU, 1-load store unit, 1 branch unit and 16 registers in each cluster. The number of functional units selected for the VLIW configurations are such that the performance achieved using this configuration is within 95% of the peak performance achieved by using many more functional units. This moderate number of functional resources guarantees that the benefits reported have not been obtained by trivially putting the numerous idle functional units into the low leakage mode. Also, we report results only for Integer ALUs which are heavily used and pose a challenge for any leakage energy management scheme. Thus, the benefits reported here have not been magnified by the leakage energy benefits of the load-store, branch, and FP units which are mostly idle.

4.4.1 Energy Model

We have used the same analytical energy model as in [34] to directly compare the energy benefits of the proposed scheme over the pure hardware based scheme proposed in[34]. According to this model the the total energy in a functional unit composed of dual threshold domino logic is determined as given in Equation 4.1 and 4.2. Different parameters used in these equation are given in Table 4.1

$$\begin{aligned} E^{'}_{total} =& DynamicEnergy + LeakageEnergy + \\ & TransitionEnergy + SleepModeEnergy \end{aligned} \tag{4.1}$$

$$\begin{aligned} E^{'}_{total} =& n_A(\alpha E_A + (1-D)E_{S_1}) + (n_A D + n_{UI})(\alpha E_{s_0} + (1-\alpha)E_{s_1}) + \\ & M_z((1-\alpha)E_A + E_{Sleep}) + n_Z E_{s0} \end{aligned} \tag{4.2}$$

The first component of the Equation 4.2 accounts for dynamic energy which is determined

Table 4.1: Parameters of Energy Model in Equations 4.2

$\mathbf{n_A}$	Number of active cycles
$\mathbf{n_{UI}}$	Number of uncontrolled idle cycles
$\mathbf{M_Z}$	Number of transitions
$\mathbf{n_Z}$	Number of sleep cycles
$\mathbf{E_A}$	Active mode energy per cycle
$\mathbf{E_{S_1}}$	High leakage mode energy per cycle
$\mathbf{E_{S_0}}$	Low leakage mode energy per cycle
$\mathbf{E_{sleep}}$	Energy to activate low leakage (sleep) mode
$\mathbf{D}$	Duty Cycle
α	Activity Factor
$\mathbf{p}$	Ratio of the high leakage energy to the active mode energy
$\mathbf{s}$	Ratio of high leakage energy to the low leakage energy

as number of active cycles, $\mathbf{n_A}$, times the maximum dynamic energy per cycle($\mathbf{E_A}$) and the high leakage energy, $\mathbf{E_{S_1}}$, during the precharge portion, (1-D), of the clock cycle. The second component of the equation takes into account the leakage energy expended in uncontrolled idle cycles, $\mathbf{n_{UI}}$, as well as for the duration in active cycles when the clock is high ($\mathbf{n_A} * \mathbf{D}$). There are two components of the leakage energy expended per cycle, viz., fraction of the number of internal nodes, α, that are places into low leakage state, $\mathbf{E_{S_0}}$ and the remaining nodes (1-α) that are not discharged and have general high leakage energy, $\mathbf{E_{S_1}}$. The third component takes into account the transition energy. This is determined as number of transitions, $\mathbf{M_Z}$, times the additional energy cost of precharging the internal dynamic nodes that would have not been discharged otherwise i.e. (1-α)$\mathbf{E_A}$ and energy cost of activating and distributing the sleep signal across FU i.e $\mathbf{E_{sleep}}$. The final component is the sleep mode low leakage energy when all the internal dynamic nodes have been discharged and dissipates energy $\mathbf{E_{S_0}}$ for $\mathbf{n_Z}$ cycles.

$$E_{S_0} = s * E_{S_1}, 0.0001 \leq s \leq 0.01 \; and \; E_{S_1} = p * E_A, 0 \leq p \tag{4.3}$$

After simplifying and normalizing the equations with respect to active energy using equation 4.3, the following model for total energy consumption is obtained :

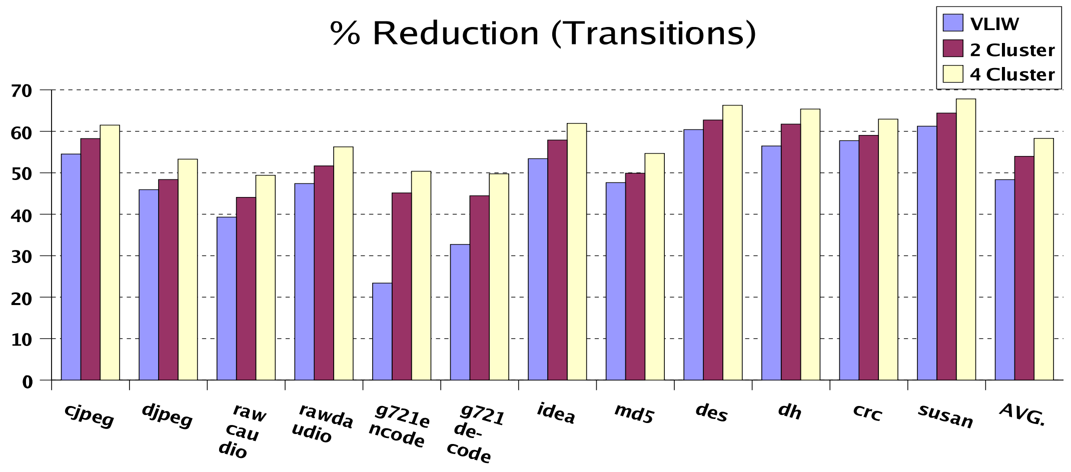


Figure 4.6: % Reduction in Transitions with Scheduling w.r.t. Hardware Only Scheme

$$\begin{aligned} E_{total} =& n_A(\alpha + (1 - D)p) + (n_A D + n_{UI})(\alpha sp + (1 - \alpha)p) + \\ & M_z((1 - \alpha) + E_{Sleep}/E_A) + n_Z sp \end{aligned} \tag{4.4}$$

The technology parameters that we have used (s=0.01 and $E_{Sleep}/E_A = 0.01$) are also the same as in [34] in order to compare the benefits of our scheduling algorithm to the hardware-only scheme ('MaxSleep') proposed in [34]. Considering the current 65nm fabrication technology where leakage energy is on par with dynamic energy, we set p to 0.5 .α is activity factor and **D** is the duty cycle of the clock. We use a typical value of 0.5 for both of these parameters in our simulation as in [34].

4.4.2 Results

We have performed a detailed experimental evaluation of the proposed scheme in terms of the reduction in the number of transitions and the associated energy savings. We present results for the hardware-only scheme from [34] called 'MaxSleep' as well as for our scheduling scheme that assists the hardware based scheme. We call this scheme 'Optimized'. The results are presented in comparison with a hypothetical scheme called 'NoOverhead' that is the same as 'MaxSleep' but does not incur any of the energy overheads of transitions. This scheme represents a theoretical ideal against which a leakage energy management scheme can be compared for its effectiveness.

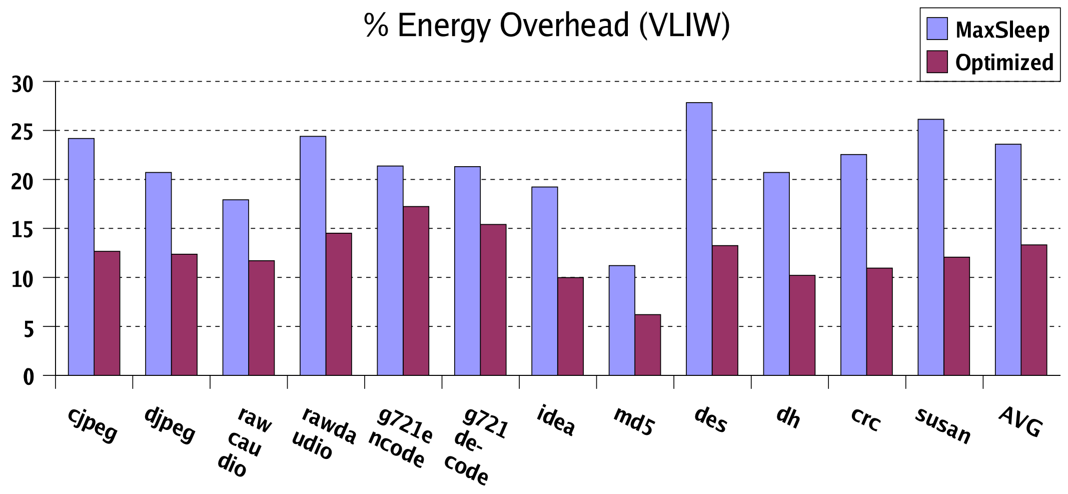


Figure 4.7: % Increase in Functionl Unit Energy w.r.t Hypothetical No-overhead Scheme (VLIW)

Figure 4.6 shows the percentage reduction in the number of transitions due to our algorithm as compared to the hardware-only scheme. We observe that the number of transitions reduce by 48.34%, 53.97%, and 58.29% for VLIW, 2-Clustered VLIW, and 4-Clustered VLIW respectively. The reduction in the number of transitions depends on the total available slack in scheduling instructions as well as the distribution of the idle cycles in the benchmark. Benchmarks like des, dh, crc, and susan have many short idle cycles and our algorithm is able to exploit the available slack in these applications to avoid many transitions. In the case of g721encode and g721decode, the available slack is relatively less and consequently the reduction is also less.

Figure 4.7 shows the energy overhead of 'MaxSleep' and 'Optimized' schemes as compared to the 'NoOverhead' scheme. 'MaxSleep' and 'Optimized' schemes show average energy overheads of 23.59% and 13.32% respectively as compared to the 'NoOverhead' scheme. The proposed 'Optimized' scheme reduces the total energy overhead by 11.85% over the 'MaxSleep' scheme which is significant taking into account that it is a purely software based scheme and does not incur any hardware overhead. These results are in agreement with the results presented in [34], where the author mention that the 'MaxSleep' scheme incurs 30% more energy overheads than the 'NoOverhead' scheme (some difference compared to our evaluation is due to change in workload). In [34] the evaluation is based on the spec benchmark in the context of superscalar architecture, whereas our evaluation uses the media benchmark in the context of VLIW and

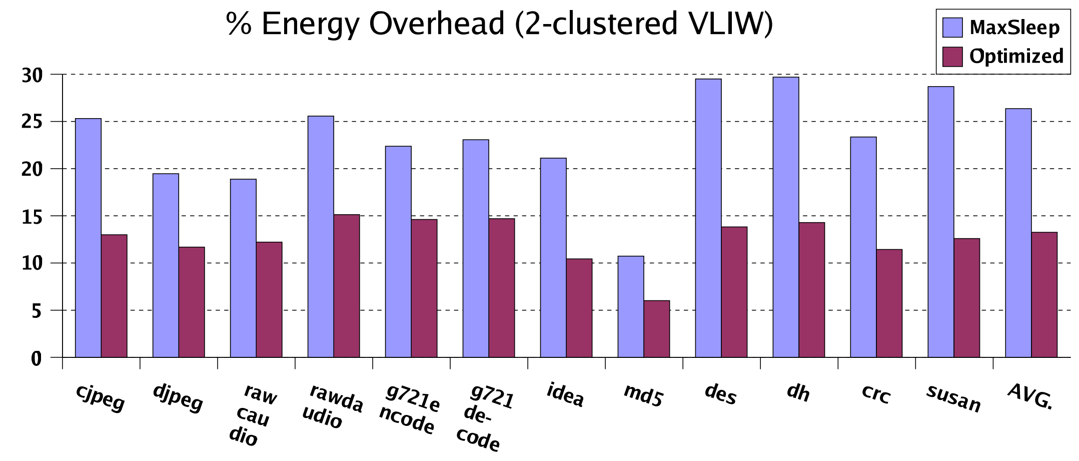


Figure 4.8: % Increase in Functionl Unit Energy w.r.t Hypothetical No-overhead Scheme (2 Cluster)

clustered VLIW architectures.

The benefit of our scheme is even more pronounced in the context of clustered architectures. In the context of 2-clustered architecture 'MaxSleep' and 'Optimized' have average energy overheads of 26.36% and 13.26% respectively as compared to the 'NoOverhead' scheme (Refer Figure 4.8). The energy benefits of 'Optimized' over Maxsleep is 15.11% in the context of 2 clustered architecture. For a 4-clustered configuration, 'MaxSleep' and 'Optimized' incur 27.02% and 12.15% overhead as compared to 'NoOverhead' scheme (Refer Figure 4.9). The 'Optimized' scheme improves over the 'MaxSleep' scheme on the average by 16.92% in the context of 4-clustered architectures. The reasons for more savings in the context of clustered architectures are as follows. Clustering brings along extra contentions for a limited number of slow cross-paths (for inter-cluster communication). This leads to many short idle cycle during which functional units are waiting for operands to arrive from other clusters. A purely hardware based scheme with traditional scheduling algorithm undergoes transitions for such many short idle cycles and suffers the associated energy penalty. In contrast to the performance-oriented scheduling algorithm which is designed for utilizing the resources spread over different clusters to achieve a better performance, our energy-aware scheduling algorithm sometime limits the spreading of operations, if it can fetch some energy benefits without hurting performance. Thus, some of the extra slack which is available while scheduling for clustered architectures due to contention for

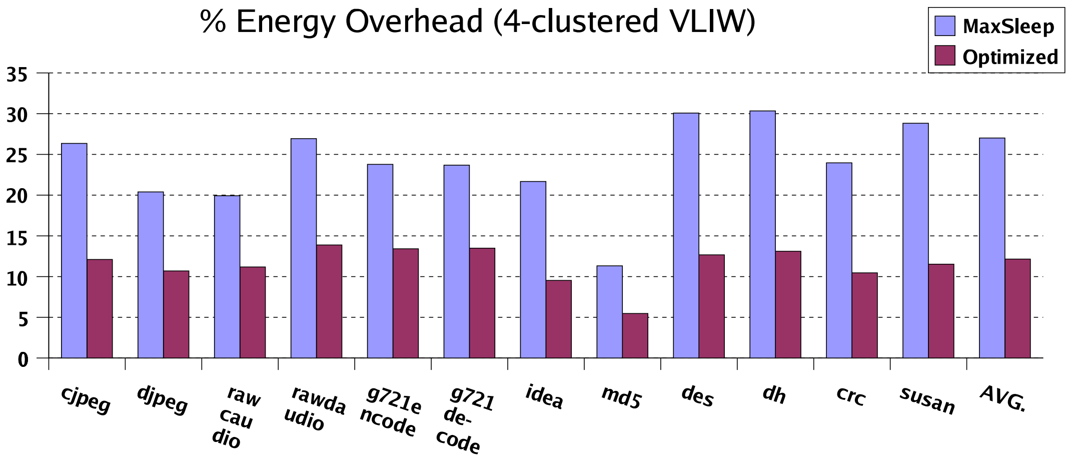


Figure 4.9: % Increase in Functionl Unit Energy w.r.t Hypothetical No-overhead Scheme (4 Cluster)

inter-cluster communication is utilized to gain energy benefits in our algorithms.

Finally, our energy aware algorithm suffers a very marginal performance loss of 0.2% in the context of VLIW architceture as compared to the performance oriented scheduler. The performance loss in the context of 2-clustered and 4-clustered architecture is 0.3% and 0.5% respectively. The reason for this performance loss is inherent inaccuracies in determining the available slack. In certain cases this leads to a few cycles of performance penalty due to serialization of operations. However, the results clearly show that it is rare and its overall effect on performance is negligible. This is because our algorithm is conservative in exploiting slack to save energy.

4.4.3 Sensitivity Analysis

Results presented in the last subsection are for the 65nm technology currently in fabrication. In this subsection we present the benefit of over 'Optimized' scheme over 'MaxSleep' scheme for other technology nodes such as 90nm (where leakage is 20% to 30% of overall energy) and 45nm (where leakage is 60% to 70%) of overall energy. Assuming leakage fraction of 25%, 50% and 65% for 90nm, 65nm and 45nm, Figure 4.10 presents the benefit of 'Optimized' scheme over 'MaxSleep' scheme. It is clear that the benefit of our scheme are higher for even larger

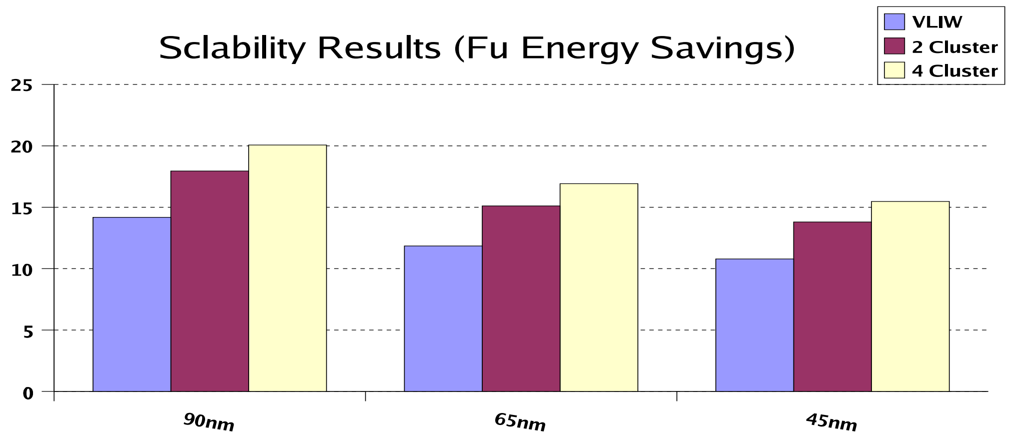


Figure 4.10: Scalability Results for Functional Unit Energy Savings

technologies such as 90nm where the leakage fraction is relatively lesser. This is because our scheme is geared towards reducing the transitions and associated energy overheads of transitions which are more in bigger technologies. Thus, our scheme save significant leakage energy over hardware scheme even in smaller technologies by reducing the transition energy overheads. Consequently, the benefit in even smaller technologies such as 45nm are slightly less but still significant. Because when the overall contribution of leakage is more and corresponding dynamic energy contribution is less thereby the extra transitions and the impact of savings by reducing these transitions is also slightly less. On average savings of our compiler directed scheme over hardware only scheme are 14.18%, 17.95%, and 20.07% in the context of VLIW, 2-clustered and 4-clustered architectures respectively in 90 nm technology. Whereas the corresponding benefits are 10.79%, 13.8% and 15.47% in 45 nm technology.

4.5 Application in the Context of Instruction Decoder

The proposed technique is also applicable in the context of instruction decoder which is another combinational circuit in VLIW processor. The application in the context of instruction decoder is also motivated by the fact that instruction decoder in the context of VLIW and clustered VLIW

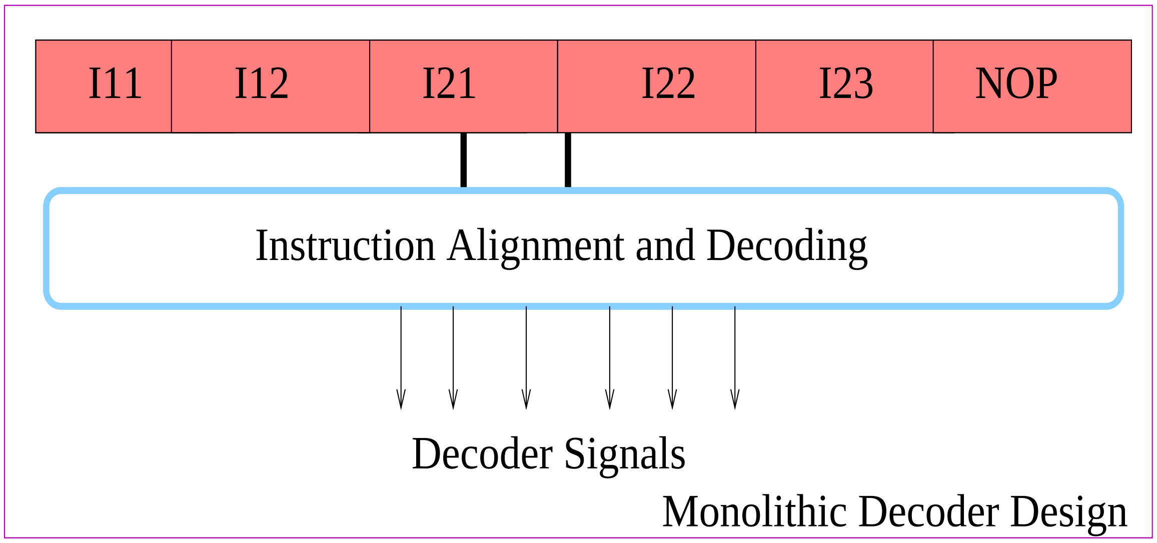


Figure 4.11: Traditional Monolithic Decoder Design

architectures is required to decode up to 8 (or more depending on issue width) instructions in parallel. However, the full issue width is rarely utilized which leads to unnecessary leakage energy consumption in the decoder circuitry. Moreover, frequent access to the instruction decoder raises the temperature level and make not only the leakage energy consumption even worse in the instruction decoder but also makes decoding circuitry a hot-spot in the chip[146][147]. However, unlike functional units, the traditional monolithic design of instruction decoder inhibit the leakage energy management in instruction decoder. We consider a split instruction decoder as described in next subsection that enable the leakage energy optimization. We also evaluate the benefit of the proposed compiler scheduling algorithm in the context of instruction decoder.

4.5.1 Split Decoder Design

Decoding activity involves dividing a fetch packet into execute packets and then decoding individual micro-instructions in each execute-packet to issue signals. A parallel-bit is dedicated in a VLIW micro-instruction that specifies whether the next micro-instruction is in the same execute-packet (i.e., executes in the same cycle) or starts a new execute-packet. A traditional monolithic design of instruction decoder as shown in Figure 4.11 inhibits any fine grained control for hibernating parts of the decoder circuit that are idle. A decoder circuit can be easily pipelined and split as shown in Figure 4.12. This provides the benefits of ease of design and verification of circuit and performance benefits of pipelining[148] and also enables leakage energy

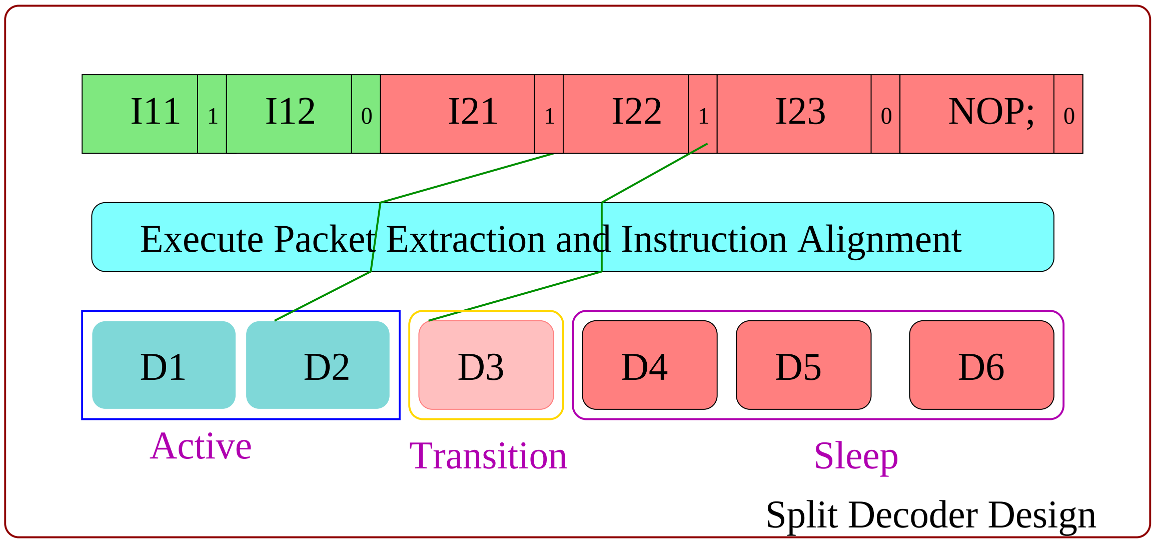


Figure 4.12: Split Decoder Design

savings at the granularity of individual decoders. The performance benefits of pipelining the decode stage have already been identified and such a design is in use in many high performance commercial DSPs including the Texas Instruments' VelociTI[13]. However, we capitalize on the energy management capability of such a design as follows.

Due to variations in the ILP of the programs, the full issue width of the processor is rarely utilized continuously and hence several decoder will be idle most of the time. The split decoder design can leverage the capabilities of dual-threshold domino logic for fast transition from active mode to sleep mode and vice versa in less than a cycle (as used in [34] for functional units) to save tremendous amount of leakage energy in mostly idle decoder circuit. However, in order to avoid the explicit penalty of activating a sleeping decoder, it is required to issue the activating signal one cycle in advance. Fortunately, the parallel-bit that specifies the parallel instructions in the current execute-packet can be used to drive the activation signal. To avoid introducing any new hardware, in our machine model, we always keep first decoder active, and use the parallel-bits in execute packet to drive the active signal for the required number of decoders. It is important to note that these signals are activated during the first stage of decoding when the execute packet is being extracted and aligned from the fetch packet. Thus, by the time the micro-instructions reach stage two for actual decoding, the required number of decoders are in active state to perform the decoding.

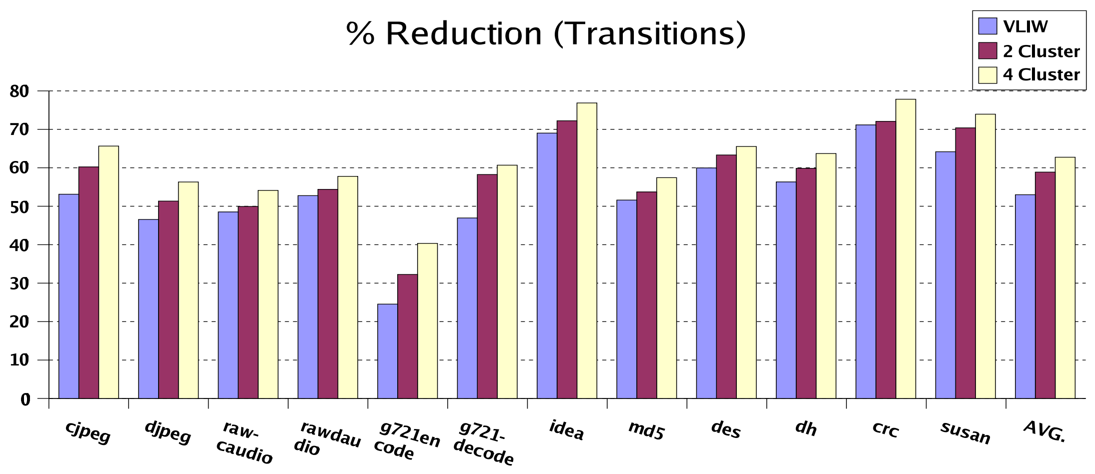


Figure 4.13: % Reduction in Transitions with Scheduling w.r.t. Hardware only Scheme for ID

4.5.2 Experimental Evaluation

The result of applying the proposed compiler-assisted scheme in the context of split instruction decoder (ID) design are depicted in Figure 4.13 and Figure 4.14.

Figure 4.13 shows the percentage reduction in the number of transitions due to our algorithm as compared to the hardware-only scheme. We observe that the number of transitions reduce by 53%, 58.88%, and 62.74% for VLIW, 2-Clustered VLIW, and 4-Clustered VLIW respectively.

Figure 4.14 shows the energy overhead of 'MaxSleep' and 'Optimized' schemes as compared to the 'NoOverhead' scheme. 'MaxSleep' and 'Optimized' schemes show average energy overheads of 27.29% and 14.99% respectively as compared to the 'NoOverhead' scheme. The proposed 'Optimized' scheme reduces the total energy overhead by 14.46% over the 'MaxSleep' scheme which is significant taking into account that it is a purely software based scheme and does not incur any hardware overhead.

The benefits of our scheme are more pronounced in the context of clustered architectures. In the context of 2-clustered architecture 'MaxSleep' and 'Optimized' have average energy overheads of 29.37% and 14.6% respectively as compared to the 'NoOverhead' scheme (Refer Figure 4.15). The energy benefits of 'Optimized' over Maxsleep is 17.3% in the context of 2-clustered architecture. For a 4-clustered configuration, 'MaxSleep' and 'Optimized' incur 29.88%, and 13.7% overhead as compared to 'NoOverhead' scheme (Refer Figure 4.16). The 'Optimized'

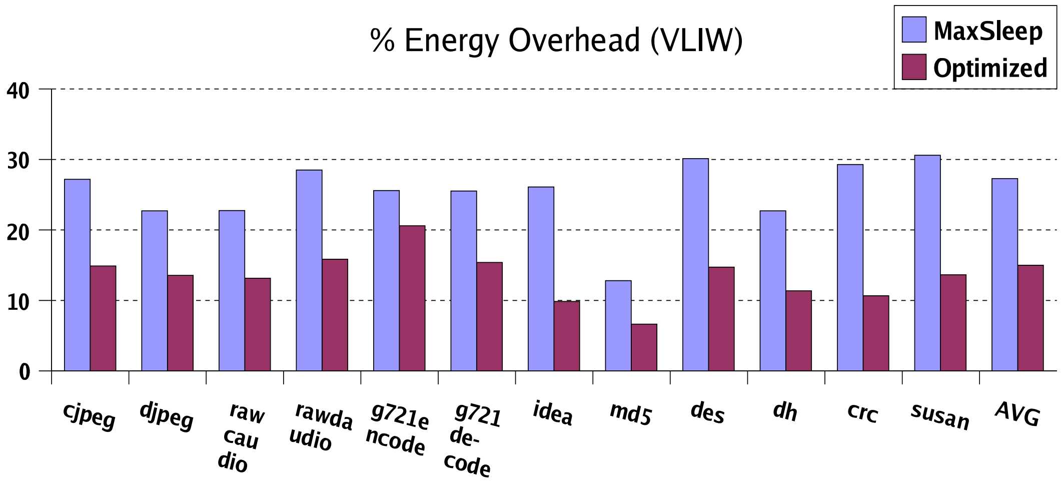


Figure 4.14: % Increase in Energy w.r.t Hypothetical No-overhead Scheme for ID (VLIW)

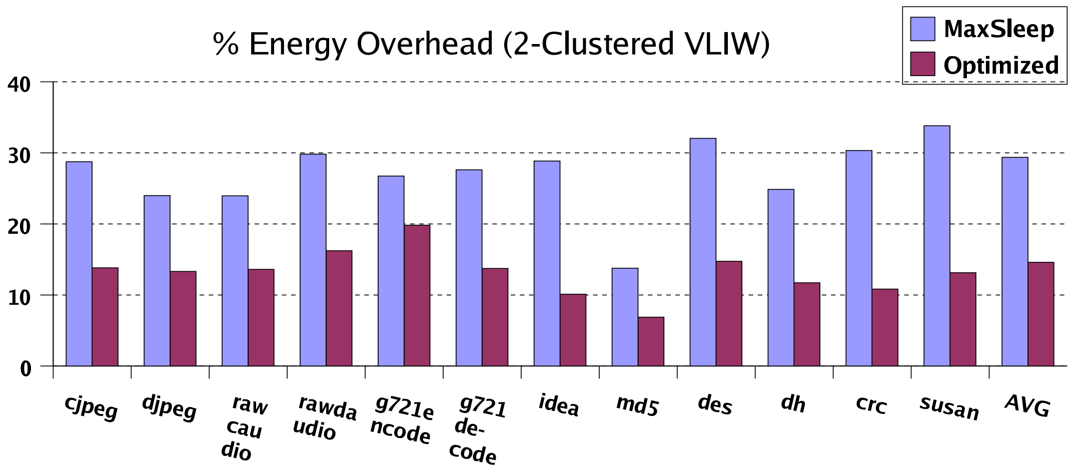


Figure 4.15: % Increase in Energy w.r.t No-overhead Scheme for ID (2 Cluster)

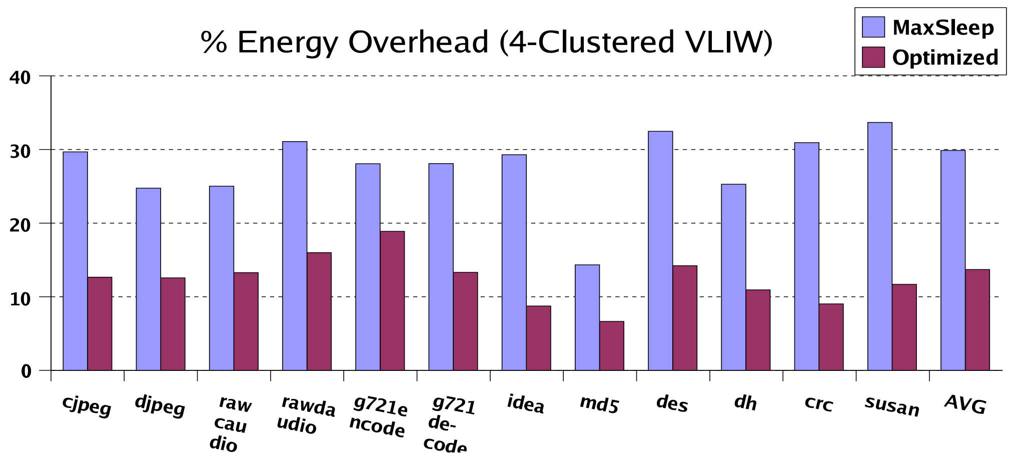


Figure 4.16: % Increase in Energy w.r.t No-overhead Scheme for ID (4 Cluster)

scheme improves over the 'MaxSleep' scheme on the average by 18.74% in the context of 4-clustered architectures. The reasons for more savings in the context of clustered architectures is same as explained above i.e. some of the extra slack which is available while scheduling for clustered architectures due to contention for inter-cluster communication is utilized to gain energy benefits in our algorithms.

4.6 Related Work

In this section, we compare and contrast our work with some of the earlier related work in the area of instruction scheduling for clustered architectures, architectural approaches for leakage energy management, and energy aware scheduling for VLIW architectures. Reader is referred to chapter 2 for a general treatment on earlier work in the area of circuit, architectural, and software techniques for energy optimization.

4.6.1 Scheduling for Clustered Architectures

Earlier proposals for scheduling on clustered VLIW architectures can be broadly classified into two main categories. The phase-decoupled approach to scheduling[130][131][132][149] works on a Dataflow graph (DFG) and partitions instructions into clusters to reduce inter-cluster communication while approximately balancing the load among clusters. The annotated DFG is then scheduled using a traditional list scheduler while adhering to earlier spatial decisions. However, the phase-decoupled approach to scheduling suffers from the well known phase-ordering problem. A spatial scheduler has only an approximate knowledge (if any) of the usage of cross paths, functional units, and load on clusters. This inexact knowledge often leads to spatial decisions which may unnecessarily constrain a temporal scheduler and may lead to a suboptimal schedule. We follow an integrated approach to scheduling[133][150][43] that tries to combat the phase-ordering problem by combining spatial and temporal scheduling decisions into a single phase. In what follows, each of these proposals is briefly described.

Faraboschi et al.,[131] have proposed a two phase algorithm called partial component clustering (PCC). In the first phase the DAG to be scheduled is partitioned into several components using a depth-first search algorithm and a maximum component size threshold. These components are then clustered while striving for minimum ICC and balancing load among clusters. The second phase is characterized by improving the initial binding of instructions using an iterative algorithm to reduce the schedule length or ICC. The major objective of this algorithm is to minimize the communication among clusters. Gonzalez et al.,[130] have proposed a graph partitioning based approach that is used as a pre-partitioning phase to modulo scheduling. The first phase is graph coarsening in which a maximum edge weight matching algorithm is used to find a graph matching and collapsing the match nodes into one. This is repeated until the number of instructions are same as the number of clusters. This is followed by the partition refining phase that considers possible movement of instructions among clusters with a view to get better execution time. Lapinskii et al.,[132] have proposed an effective binding algorithm for clustered VLIW processors. Their algorithm performs spatial scheduling of instructions among clusters and relies on a list scheduling algorithm to carry out temporal scheduling. They compute the cost of allocating an instruction to a cluster using a cost function that considers the load on the resources and buses to assign the nodes to clusters. Mahlke et al., have proposed a graph-partitioning based approach similar to [130] which they call as region-based hierarchical operation partitioning (RHOP)[149]. RHOP assigns a weight to each node and edge of DFG to be partitioned. Nodes are assigned weights so as to represent the load on resources while edges are assigned weights to represent the cost of introducing a inter-cluster move operation between node connected through the edge. Edge weights are used to coarsen the graph from finer nodes to coarser nodes until the number of partition reaches a threshold. This is followed by an uncoarsening phase that improves the initial partition by considering the movement of coarsened nodes available at this point to another clusters.

Ozer et al.,[133] have proposed an algorithm called unified-assign-and-schedule (UAS). UAS extends the list scheduling algorithm with a cluster assignment decision while scheduling. After picking the highest priority node, it considers clusters in some priority order and checks if the operation can be scheduled in a cluster along with any communication needed due to this binding. They have proposed various ways of ordering clusters for consideration such as, no ordering, random ordering, magnitude weighted predecessors (MWP) and completion weighted predecessors (CWP). Leupers[150] proposed an algorithm for integrated scheduling of clustered VLIW processors. They use simulated annealing, a heuristic search algorithm for cluster binding followed by a list scheduling algorithm. Kailas et al.,[43] have proposed an algorithm for combined binding, scheduling and register allocation for clustered VLIW processors. Their greedy algorithm binds a node to the cluster in which it can be executed at the earliest. This may lead to a high inter-cluster communication in the future (while scheduling successor of this node) due to the unavailability of a communication slot to schedule the required MV operation and this may stretch the schedule as well. They also propose an on-the-fly register allocation scheme. Insertion of spill code and scheduling of spill code is also integrated in the main algorithm.

4.6.2 Architectural Approaches for Leakage Energy Management

Study of leakage energy management at the architectural level has mostly focused on storage structure such as cache. Yang et al., propose power supply gating of L1 cache cells[31]. Kaxiras et al., dynamically adjust the interval after which a cache line is put into low leakage mode[32]. Flaunter et al., propose a state-preserving drowsy cache design and a simple control scheme which is able to deliver most of the leakage energy benefits[33]. A detailed description of leakage energy management techniques for storage structure is available in chapter two.

In contrast to storage structures, little work has been done on architecture level leakage energy management in the context of functional units. Our work directly improves over the work due to Albonesi et al.,[34]. This work proposes and evaluates an architectural policy for aggressively controlling leakage energy in integer ALUs. The 'MaxSleep' policy puts a functional unit into low leakage mode after one cycle of idleness. This scheme depends on dual threshold domino logic circuit with sleep mode proposed in [30] which has no delay penalty of transition between active mode and sleep mode. However, their performance evaluation using the same analytical energy models (as described in section) in the context of spec benchmarks for superscalar architectures shows that for technology such as 65nm, the leakage energy benefit gained by such an aggressive scheme is significant (i.e., on an average 30% energy overhead when compared to a hypothetical 'NoOverhead' scheme that is same as 'MaxSleep' scheme but does not incur any transition energy).

Our experimental evaluation using the same analytical energy model in the context of VLIW/Clustered architecture demonstrates the effectiveness of our compiler-directed scheme in reducing the spurious transitions thereby assisting the purely hardware based scheme ('MaxSleep') in achieving better leakage energy savings.

Decoder Energy Optimization

To the best of our knowledge, only work for energy optimization in the context of instruction decoder is due to Kuo et al.,[148]. Kuo et al.,[148] consider instruction decoding as in superscalar architectures and propose to split (horizontally partition) instruction decoder circuitry into two or more sub-decoders based on execution frequencies of different instructions. They also propose to do pipelining (vertical partitioning) of the instruction decoder to achieve energy and area benefits. The experimental results of Kuo et al., based on physical synthesis clearly demonstrates that the horizontal and vertical partitioning of the instruction decoder is in general useful in reducing the design complexity, power consumption, area overhead, and delay because of simplification of circuitry. In contrast to the work of Kuo et al.,[148], partitioning of instruction decoder in our work is geared more toward VLIW and clustered VLIW architectures that demands decoding of large number of instructions in parallel. Thus, compared to functionally asymmetric partitioning of Kuo et al, we consider partitioning of instruction decoder circuitry into functionally identical individual sub-decoders each of which can be controlled independently. The pipelining of decoder as considered by us is more natural in VLIW context where a fetch packet needs to be broken into execute packets and the current execute packet needs to be aligned before actual decoding can begin. Apart from general benefits of a partitioned design as demonstrated by Kuo et al., partitioned decoder design in our proposal also provides an opportunity for fine grained leakage energy management in the instruction decoder.

4.6.3 Energy-Efficient Scheduling

Earlier compiler level techniques for reducing energy consumption in functional components are mostly proposed in the context of superscalar and flat VLIW architectures. In what follows, we compare our work in the context of energy efficient scheduling for clustered VLIW architectures with these earlier proposals.

Zhang et al.,[36] have proposed a rescheduling scheme to reduce dynamic and leakage energy in the functional units of a flat VLIW processor by exploiting the remnant slack of a performance-oriented schedule. In contrast, our approach works on raw unscheduled code with all the available slack for scheduling and complements a hardware based mechanism for leakage energy management. Kim et al.,[25] have proposed a leakage energy management scheme for flat VLIW processors that approximates the ILP available in the program using heuristics (as the exact estimation problem is itself NP complete). The calculation is done at the loop level granularity assuming that there is little variation in the ILP within the loop. Their scheme keeps only a canonical subset of functional units that is sufficient to exploit this approximated ILP active. In contrast, our approach in the context of clustered VLIW architectures adaptively applies leakage energy management at a finer granularity based on available ILP thereby subsumes the benefit of their approach and delivers better results even in the presence of a variation in the ILP within a loop. Gupta et al.,[35] propose a novel data structure called power-aware flow graph. Their leakage energy management scheme in the context of superscalar processors works over this graph to determine larger program regions called power blocks which offer opportunities to save leakage energy. ISA and architectural support is needed to switch on and off the functional unit at the boundaries of power blocks and nullify spurious on-off. Our approach in the context of clustered VLIW architectures is to nullify spurious on-off without any special hardware support. Kim et al.,[37] have proposed a modulo scheduling algorithm that produces a more balanced schedule for software pipelined loops with an objective to reduce the peak power and step power dissipation. Though our algorithm is not directly designed towards improving the peak power and step power dissipation, it generates a more balanced schedule. As we demonstrated using an example, our scheme tries to keep the minimum number of functional unit active and tries to use the active functional units as much as possible while keeping the idle functional units in idle state for longer durations.

4.7 Conclusions

In this chapter, we have proposed a new energy-aware instruction scheduling algorithm for VLIW and clustered VLIW architectures that is capable of reducing the number of transitions in functional units and instruction decoder by exploiting the scheduling slack of instructions. The experimental evaluation reveals that the proposed scheme is able to reduce the number of transitions in functional units by approximately 48%, 54%, and 58% for VLIW, 2-Clustered VLIW, and 4-Clustered VLIW respectively. This results in 11.85% energy savings in functional units in the context of VLIW architecture while 15.11% and 16.92% energy savings in functional units in the context of 2-clustered and 4-clustered VLIW architecture respectively, as compared to a purely hardware based scheme. The corresponding energy benefits in instruction decoder are 14.5%, 17.3%, and 18.7% for a VLIW, 2-clustered, and a 4-clustered VLIW architecture respectively. The reasons for more savings in the context of clustered VLIW architecture is that our scheme aggregate the extra idleness in functional components in clustered architectures (attributed to contention for limited number of slow cross-paths for inter-cluster communication) to gain the energy benefits. Our sensitivity analysis shows that the compiler scheduling scheme gives significant benefits even in technologies such as 90nm and 45nm over a purely hardware based scheme. In addition, the proposed scheme is able to generate a more balanced schedule that helps in reducing the peak power and step power dissipation of the processor.

Chapter 5

INTACTE: An Interconnect Area, Delay, and Energy Estimation Model

5.1 Introduction

On-chip interconnect for communication among spatially separate resources introduces major performance, area, and energy bottlenecks for decentralized architectures. In Embedded domain, trend towards using fine grained distribution to achieve scalability has been visible for quite some time[11][151][152]. Multicore architectures take the idea of *scalability by distribution* even furthers[153][154][155].

Interconnects are known to consume significantly high energy and area and are known to be a major source of performance bottlenecks[156] for single-core decentralized architectures (such as clustered superscalar and clustered VLIW). In context of multi-core architectures, it has been observed that interconnects can easily consume power equivalent to one core, area equivalent to three cores, and delay that account for over half the L2 access latency[157]. [157] clearly demonstrates that design trade-offs made considering the interconnect as an independent entity can often be quite opposite to the design trade-offs that are optimal from power and performance point of view. Co-designing interconnects early along with other components when high level architectural design trade-offs are being made is highly desirable for high level synthesis of embedded SoCs.

In order to quantitatively evaluate different interconnect design trade-offs for decentralized architecture, one needs a reasonably accurate and fast model for the area, delay, and power for these choices. Prior research in interconnect modeling and analysis has mostly dealt with specific circuit level issues [28][38][39] and is not directly usable to make high level micro-architectural trade-offs. For example, an architect would be interested in knowing what are the available trade-offs in terms of pipeline latency and power, for a given bandwidth and interconnect distance. This information could be used at a higher level of design to obtain the overall optimum for the system. Similarly, it will be very useful to know the power and performance of the interconnect at different operating voltages and frequencies, in order to evaluate dynamic voltage and frequency scaling schemes. Hence, there is a need for a tool for the interconnect, which can give reasonably accurate design points and their associated area and power costs for various architecture level constraints such as bandwidth, latency etc. Similar models are available for caches[40], register files[41] and functional components[34]. Availability of an interconnect model will be very helpful for architects to involve interconnect in early design trade-offs.

This chapter proposes an interconnect modeling tool to get fast but reasonably accurate estimates of interconnect delay, area, and power for a given technology, wire length, bit-width, clock frequency, and latency. The tool solves an optimization problem of minimizing power by finding the appropriate wire size, repeater size, and repeater spacing for varying degrees of pipelining and area. We are currently limiting our work to cover point-to-point interconnects only, as most of the high performance long distance interconnects will be of this form[158]. The tool outputs a set of interconnect designs for a cross section of area and degrees of pipelining, all of which meet the frequency and latency constraints. In addition, for each design a set of power and performance numbers are also given across a range of supply voltages. These choices enable the user to explore the micro-architecture design space for the system which includes this interconnect.

The area, delay, and power estimation for the interconnect is built upon the corresponding values for the low level component such as wires, repeaters, flops and buffers, which are in turn obtained via accurate HSPICE[159] characterization. However, this one time characterization is done in advance, and hence the tool itself is fast enough to explore many interconnect choices rapidly. Furthermore, the power model is parameterized with respect to the activity factor (probability of switching of any bit) and the coupling factor (probability of relative switching between adjacent bits). An architect can profile the target workload to get these quantities in order to further improve the accuracy with respect to the target workload. We have also validated the estimates of delay and power obtained by the tool with HSPICE simulation and we found that the error is less than 15% in the worst and below 12% on average. Our tool based approach to architectural modeling of interconnect parameters is analogous to that of CACTI[40].

The proposed tool can be used by architects/compiler writers in many different ways. Since different on-chip interconnects have different performance requirements, the tool can be used to customize the interconnect design to meet these goals at minimum power. The impact of different interconnect choices with latency and power trade-offs can be evaluated at the architectural level in concert with compiler optimizations. Thus, the tool enables the co-design of interconnects along with the other components early in the design phase and its impact on the overall system power and performance can be evaluated upfront. As mentioned earlier this has become very important in new process generations[157]. The major contributions of this work are:

1. A tool which provides estimates of area and power for a power efficient interconnect to meet target bandwidth and latency requirements for a range of technologies. The tool optimizes for power by finding the optimal values of the wire widths, repeater sizes and spacings, which can meet the target bandwidth and latency.

2. For each input requirement, the tool provides a range of design choices with respect to area and degrees of pipelining which can be used by the user to explore micro-architectural trade-offs at the system level. Furthermore, the tool provides estimates of power, bandwidth and latency for a range of voltages, lower than the nominal. This allows characterization of the design for dynamic voltage and frequency scaling.

3. A detailed HSPICE validation of the tool varying different parameters such as length, pitch, and technology that confirms that tool has known degree of unidirectional error (15% in worse and below 12% on average).

We have specifically used the tool to evaluate the energy benefits of heterogeneous interconnects in the context of clustered architectures using the proposed interconnect model. The next chapter describes our scheduler for heterogeneous interconnect clustered VLIW architectures and the application of the proposed tool in evaluating interconnect energy benefits for the proposed scheduling algorithm. As another illustrative example, we have used the tool to find the optimum degree of pipelining of the wire which minimizes the overall power (Section 4).

The rest of the chapter is organized as follows. We describe the tool and its implementation in Section 2 and the associated delay and power models in Section 3. Section 4 gives the experimental and validation results for the tool. Section 5 puts our work in the context of earlier work in the area of interconnect modeling. Section 6 concludes this chapter.

5.2 INTACTE Tool Description

The core motivation behind the tool is to fill the gap between an architect's requirements of the interconnect and what the circuit level interconnect models provide. Figure 5.1 depicts the tool, its inputs, and its outputs. The user provides the target technology, wire length, number of bits, frequency, and latency (in number of cycles). There are a number of other parameters which have the default values and can be overridden by the user. The supply and threshold voltages are automatically derived from the specified technology based on the predictive technology models (PTM)[160]. Activity factor is the probability of switching of a bit and the coupling factor is probability of relative switching of two adjacent bits. Both can be obtained by profiling the workload to override the default value of 0.5. For long interconnect running at a high frequency, pipelining the interconnect becomes mandatory. Wire length estimates can be obtained from a prior design or with some initial rough floor planning and models such as Rents rule[39].

The design variables that the tool considers for the interconnect optimization are as follows:

1. The wire width (w) and wire spacing (s). Increasing wire width reduces resistance and increasing wire spacing reduces coupling capacitance. Both of these reduce number of repeaters, up to a certain point. Too large a wire width or too small a wire spacing leads to large wire capacitance which is counter productive. Wire width and spacing decides the overall area taken by the interconnect which can be given as an optional constraint by the user or tool work in a loop back manner for set of nominal area values.

2. Repeater Size (S) and Spacing (l_r). Long wires have to be broken up with periodic repeaters to reduce the impact of the wire resistance. There is an optimal repeater size and spacing for minimum delay. But lower sizes and increased spacing can be used to reduce power while meeting target delay[28].

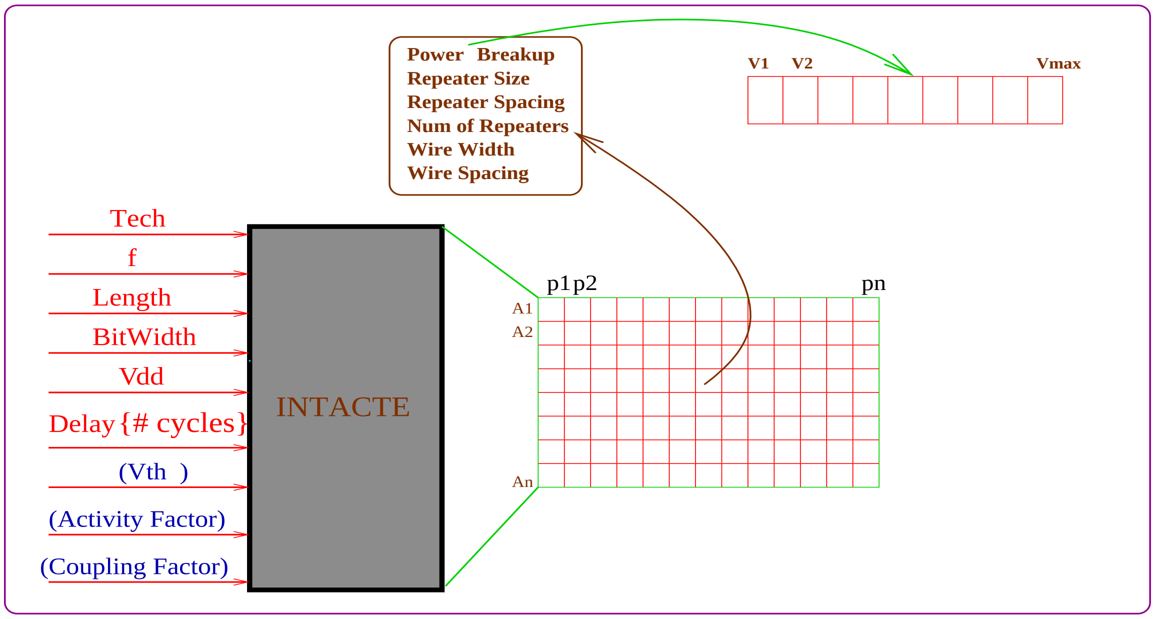


Figure 5.1: Overview of the INTACTE

3. Degree of Pipelining (p). Long interconnects will need intermediate flop stages in order to meet the frequency target.

4. Supply (Vdd) and Threshold Voltage (Vth). These circuit level parameters can be used to trade off dynamic power and leakage power of the interconnect.

Ideally the tool should find the optimal values for the above variables which will lead to a design with minimum power, while meeting the target performance. Unfortunately, the optimization is very complex to solve as it is a mixed integer nonlinear programming problem. Besides, the analytical formulas relating power and delay to all the design variables are also quite complex. Hence we take a pragmatic approach of a mixed analytical and search technique for finding the optimum values.

The tool explores a limited range of areas and pipeline depths. For any given area, the wire pitch is obtained as the length and number of bits are known. For any pipeline depth, the wire length in any pipeline is obtained by assuming equal pipeline segments. These two calculations result in a smaller optimization problem of finding the optimal wire width, repeater sizes, and spacing for a given wire pitch and unpipelined segment, which meets the target cycle time. This problem is solved by using the well known delay and power models for the repeaters and the wires [28]. We use a built-in optimization function in MATLAB[161] to solve this problem. In addition to considering the activity and coupling factors for dynamic power, we have also considered the leakage power which has been ignored in some of the previous work[39]. We have taken care to include the flop overheads as well as the pre-drivers after the flop into the timing and power calculations for the unpipelined segments. The delay and power models for the repeaters, buffers, flops and the wires have been calibrated with HSPICE simulations of these components over four different technology nodes using the PTM SPICE models[160]. Once the problem for an unpipelined segment is solved, the total power for the overall interconnect is easily obtained by scaling it by the number of pipe segments. At this level, the flop and clock power are also included. Thus, a design which minimizes the power for a given area, length, pipeline depth and target frequency is obtained and this is repeated for a set of areas and pipeline depths. Of course, it is also possible to override this iterative behavior to output the results for a specific area and pipeline depth too. Additional information like the breakup of power between different components is also provided which is of interest to a micro-architect. With emerging interest in dynamic voltage and frequency scaling[162][163], it is of interest to see the performance-power trade-offs possible in the interconnect. Hence the tool additionally estimates the power and performance (delay of each segment) for a range of supply voltages lower than the nominal value. The other design parameters like width, sizes, and spacings are kept the same as that obtained for the nominal value. So, in this respect, the power, performance numbers are suboptimal when compared to re-optimizing the design again for specific supply voltages. Nevertheless, these values will be of interest to the architect to evaluate the feasibility and opportunities of dynamic voltage and frequency scaling[163]. One can still obtain optimal design values for any other voltage, by explicitly specifying that voltage, which will then override the default internal voltage value. Thus, the tool allows the architect to choose the best interconnect options that suit their requirements.

The model retains its accuracy because determination of the delay, area, and power are carried out using low level circuit estimation of resistance and capacitance of interconnect components such as wires, repeaters, buffers, and flops using HSPICE[159]. However, these and other technology and voltage dependent parameters are precomputed for different technology nodes and voltage steps. Thus, the estimation is still fast enough (of the order of seconds) compared to a full blown HSPICE[159] estimation (of the order of hours) attribute to manual work and expertise involved in determining low level interconnect parameters. Moreover, pre-estimation of these values for different technology nodes also makes the model capable of providing reasonably accurate estimates for delay, area and power across technologies. We will next briefly go over the detailed models for delay and power used within the tool.

5.3 Modeling The Interconnect

We consider an interconnect as a set of lines where each line consists of number of pipelined segments. The length of interconnect and the number of lines are given as input by the architect. The architect can also give degree of pipelining as an input or the optimization is performed in an iterative manner for a set of feasible degree of pipelining. The length of a pipelined segment is determined by the length of interconnect and degree of pipelining. Each pipeline segment is made of set of wire segments demarcated by repeaters, a flop, and a set of buffers to drive the first repeater of the pipeline segment. The optimization is essentially performed for a single pipelined segment. The four optimization variables are repeater size, repeater spacing, wire width, and wire spacing. These are varied to obtain a delay that satisfies the latency specified by the architect while minimizing the power. Algorithm 3 gives an outline of the optimization process.

In what follows, we describe how the delay and power of interconnect is characterized in terms of delay and power of a pipeline segment which in turn is determined by delay and power of individual components such as wires, repeaters, flops, and buffers. Fig. 5.3 shows the schematic of a set of parallel wire segments driven by repeaters at the end. A repeater is an inverter with equivalent capacitance (C_{gate}) at the input, and a series combination of an equivalent resistance (R_t) and equivalent capacitance at the output(C_p). A wire is modeled as a R-C π section (refer Figure 5.2). To calculate the power and delay of a wire segment and associated repeaters, all the parasitics such as r_t, c_p, c_{gate}, r_w, and c_w are characterized for 4 different technology nodes (90, 65, 45, 32) and 32 different voltage steps differing by 15 mV as described in Table 5.1. The power and delay of flops and buffers are calculated by characterizing these values using HSPICE[159] (Refer Table 5.1 for details).

Algorithm 3 Outline of Optimization Problem

MINIMIZE:

$$P_{total} = BitWidth * p * P^{seg}_{total}$$

WHERE:

$$P^{seg}_{total} = P^{seg}_{wire} + P^{seg}_{rep} + P^{seg}_{buff} + P^{seg}_{flop}$$

$$P^{seg}_{wire} = P^{seg}_{wire\_dyn}$$

$$P^{seg}_{rep} = P^{seg}_{rep\_dyn} + P^{seg}_{rep\_leak} + P^{seg}_{rep\_short}$$

$$P^{seg}_{buffer} = P^{seg}_{buffer\_dyn} + P^{seg}_{buffer\_leak} + P^{seg}_{buffer\_short}$$

$$P^{seg}_{flop} = P^{seg}_{flop\_dyn} + P^{seg}_{flop\_leak} + P^{seg}_{flop\_short}$$

SUBJECT TO:

$$(D_{total} = p * D^{seg}) \leq Delay$$

$$BitWidth * (w + s) <= W, \quad w >= 4 * \lambda, \quad s >= 4 * \lambda$$

WHERE:

$$D^{seg} = D^{seg}_{wire} + D^{seg}_{rep} + D^{seg}_{buff} + D^{seg}_{flop}$$

VARY:

$$rep\_size(S), rep\_space(l_r), wire\_width(w), wire\_space(s)$$

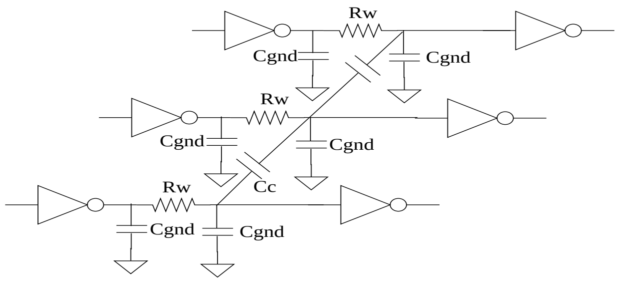


Figure 5.2: A Section of an On-chip Bus

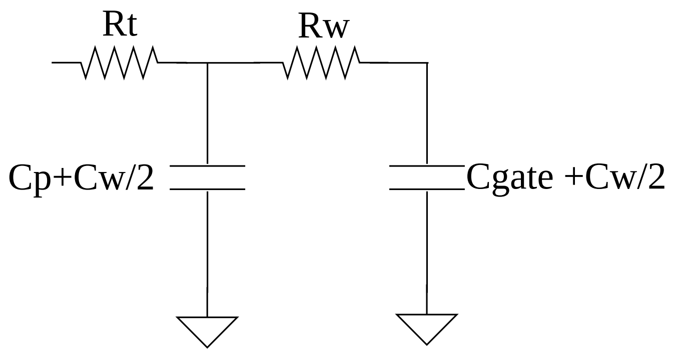


Figure 5.3: π Model of the Interconnect

5.3.1 Delay Characterization

Delay of a pipeline segment is calculated as sum of the delay of wire segments, repeaters, flops and buffers. A minimum sized flop may not have enough drive strength to drive a repeater at a very high speed. Therefore a series of buffers are introduced such that each stage (including the flop) drives a load of not more than 4 times its size. Thus the number of buffers (N_b) is given by $\lceil (log(S_r)/log4) \rceil$ where S_r is the ratio of the repeater size to the minimum possible size (i.e. $4*\lambda^1$) where size of i^{th} buffer ($Size^i_{buff}$) is $4^{i-1} * 4 * \lambda$. The delay equation for an interconnect having p pipelined segment each of length L^{seg} and n_r repeaters (per segment) is determined as follows :

$$D_{total} = p * D^{seg} \tag{5.1}$$

Equation 5.2 calculates the delay of a pipelined segment which has four components namely delay of wire, delay of all repeaters in segment, delay of flop at the beginning of pipe segment, and sum of delay of all buffers required to drive the first repeater respectively (refer Table 5.1. for definitions of symbols) .

$$D^{seg} = (R_t * ((C_p + C_{gate}) * n_r + C_w) + R_w * (C_{gate} * n_r + C_w/2)) + D_{flop} + \sum_{i \in (1..Nb)} D^i_{buff} \tag{5.2}$$

where

$^1\lambda$ is defined as half the feature size.

Table 5.1: Symbols for Various Interconnect Components

r_t	Output resistance of 1 μm repeater size[1]
c_p	Output capacitance of 1 μm repeater size[1]
c_{gate}	Input capacitance of 1 μm repeater size[1]
r_w	Resistance of 1 μm wire length[2].
c_w	Capacitance of 1 μm wire length[2].
c_{gnd}, c_c, c_f	Ground, coupling, and fringing capacitance components of c_w. [2]
D_{flop}	Delay of min sized (4*λ NMOS) flop[1]
P_{flop_dy}	Dynamic power/GHz of min sized flop[1]
P_{flop_leak}	Leakage power of min sized flop[1]
D_{buff}	FO4 delay of min sized inverter[1]
P_{buff_dy}	Dynamic power/GHz of min sized inverter[1]
P_{buff_leak}	Leakage power of min sized inverter[1]

$$R_t = r_t/S,\ C_p = c_p * S,\ C_{gate} = c_{gate} * S,\ C_w = c_w * L_{seg}\ and\ R_w = r_w * L_{seg}$$

5.3.2 Power Characterization

The total power is determined by multiplying power of a pipeline segment (P_{total}^{seg}) with the total number of pipelined segments (p) and total number of wires (BitWidth). Thus the total Interconnect power is given by Equation 5.3:

$$P_{total} = BitWidth * p * P_{total}^{seg} \tag{5.3}$$

Whereas the calculation of power for each pipeline segment for a given repeater size, spacing, wire width and wire spacing is done by calculating the dynamic, leakage, and short circuit power for each of the component as follows:

$$P_{total}^{seg} = P_{dy}^{seg} + P_{sc}^{seg} + P_{leak}^{seg} \tag{5.4}$$

[1]Spice characterized
[2]Calculated using PTM[160] models and ITRS parameters[164]

Dynamic Power

Dynamic or switching power due to switching of repeaters, pipeline registers and its corresponding buffers, and the wires is given by Equation 5.5 where, f is the frequency of operation, AF is activity factor and CF is coupling factor. The activity factor is determined by averaging the transitions on each line for a execution trace of a benchmark. Similarly, the coupling factor is determined by averaging the coupling between adjacent lines (depending on the direction of switching) for a execution trace of a benchmark.

$$\begin{aligned} P_{dy}^{seg} =& (AF * (C_{gate} + C_p) * n_r + C_{wp}) * f * V_{dd}^2 + P_{flop\_dy} * f + \\ & \sum_{i \in (1..Nb)} (Size_{buff}^i * P_{buff\_dy}^1 * f) \end{aligned} \tag{5.5}$$

$$C_{wp} = ((c_{gnd} + c_f) * AF + c_c * CF) * L^{seg} \tag{5.6}$$

Static Power

Static power is consumed when the transistors are idle. This is due to the finite OFF state current flowing in transistors in sub-threshold region and is given by Equation 5.7 where leakage current of a 1 μm repeater (I_{leak}) is determined using equations in [165].

$$P_{leak}^{seg} = (I_{leak} * S * n_r + P_{flop\_leak} + \sum_{i \in (1..Nb)} (Size_{buff}^i * P_{buff\_leak}^1)) \tag{5.7}$$

Short-circuit Power

The short circuit power of repeaters (the finite duration (t_r) in which both PMOS and NMOS are on) is calculated by equation 5.8 where I_{sc} is the short circuit current of 1 μm repeater. The short circuit power of flops and buffers are included in the dynamic power while characterizing these elements.

$$P_{sc}^{seg} = I_{sc} * S * V_{dd} * t_r * f \tag{5.8}$$

5.4 Experimental Results

In this section, we present a small subset of results that we obtained using the tool. These results exhibit various trends in the interconnect energy and serve to demonstrate how accurately our tool models the interconnects. The results are presented for interconnects of different lengths modeled at different technology nodes, with varying degree of pipelining, pitch values, and operating at different frequencies. We also present and describe validation results for different interconnect configurations obtained using HSPICE[159].

Figure 5.4 shows the change in the power as degree of pipelining is increased for two different technology nodes (90 nm and 65 nm) and for two different frequency values (2 Ghz and 1 Ghz). Increasing the pipeline stages for a particular frequency and technology first reduced the power and then there is an increase in the power. In the left part of the graph, power reduction due to decrease in repeater size and number (as a result of increase in degree of pipelining) overwhelms the power overheads due to flops and buffers. However, the situation is opposite for higher degree of pipelining (as shown in right part of the graph) where the power overheads due to flops and buffers exceed the benefits because of already small repeaters. Thus, the inflexion point corresponding to the optimal degree of pipelining shifts to the right for higher frequencies and for lower technology nodes. This reinforces the need for higher degree of pipelining in interconnects running at high frequencies and/or smaller technologies. The reduction in the power of interconnect for smaller technologies is attributed to reduction of transistor capacitance that leads to lower dynamic and short circuit power of repeaters and flops. Figure 5.5 brings out this fact more clearly by showing that the dynamic power of repeater reduces significantly whereas leakage power of repeater increases for smaller technology nodes. However, the leakage power is a small fraction of overall power of repeater in Figure 5.5 or interconnect in Figure 5.4 because we consider a workload with high activity factor in these configurations. The component wise power breakup and leakage trend in interconnect for different activity factors are presented in Figure 5.6 and Figure 5.7 respectively which are discussed later.

Figure 5.4 also shows the HSPICE simulated power estimation for the 90 nm (2 GHz) interconnect to validate our tool. We observe that the error in estimating power using our model is 10.3% at the worst and 7.8% on an average for this configuration. The error estimates for a 1 μm repeater running at 1 GHz is shown separately in Figure 5.5 which shows that the error in estimation of repeater power is at most 14.6% across technologies. The important point

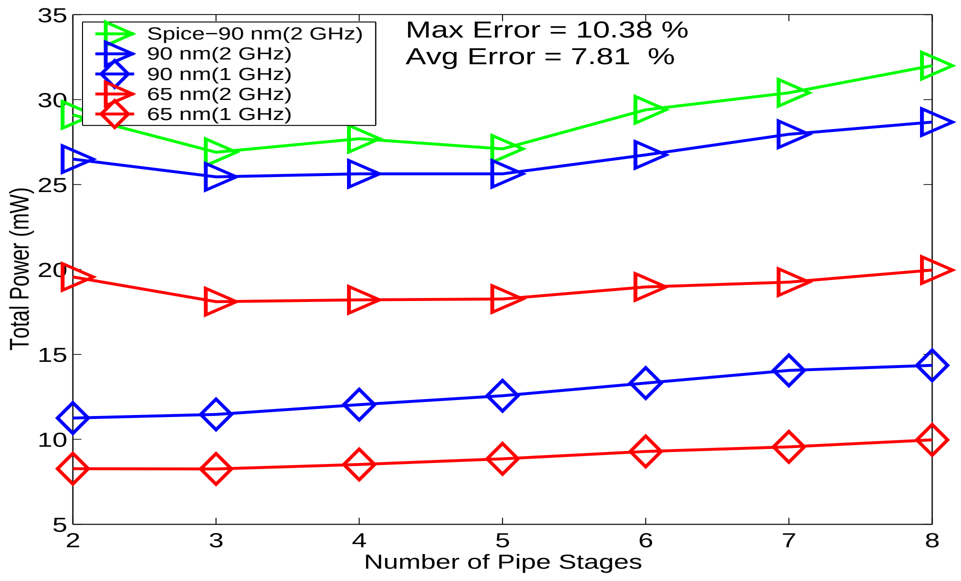


Figure 5.4: Degree of Pipelining vs Power for 5 mm Interconnect with $12 * \lambda$ Pitch

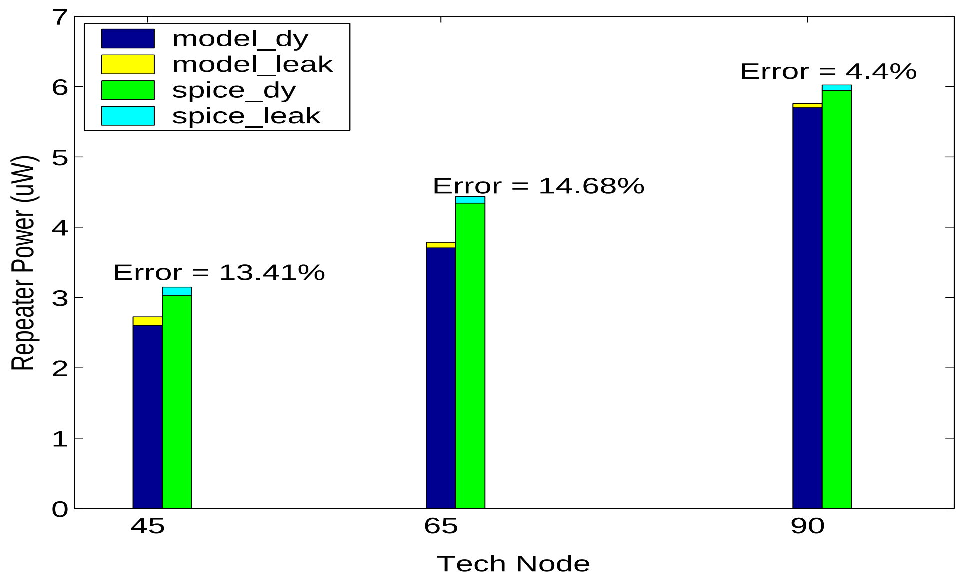


Figure 5.5: Validation of Dynamic and Leakage power for 1 μm Repeater Operated at 1GHz for Different Technology Nodes

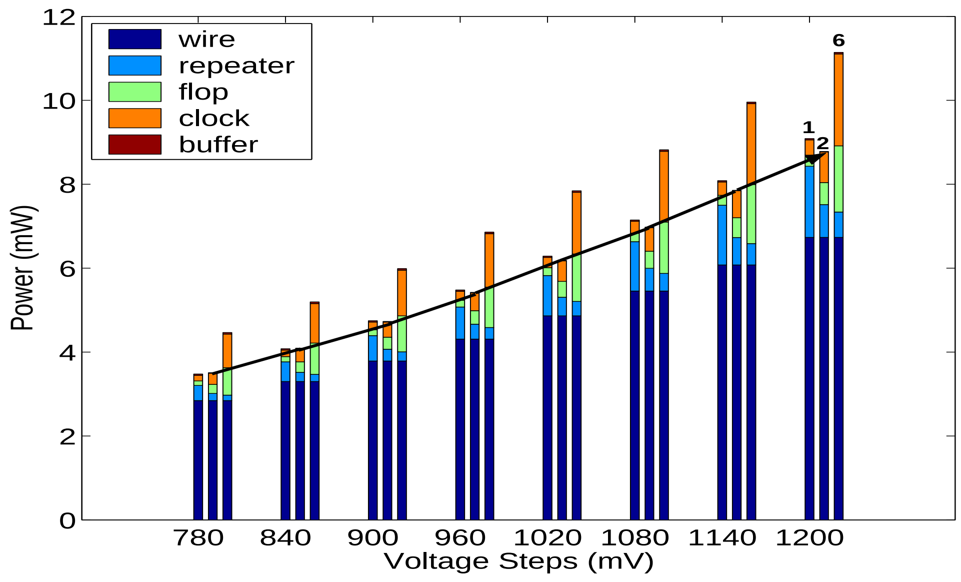


Figure 5.6: Component Wise Power Breakup for a 5 mm Interconnect with $16 * \lambda$ Pitch in 90 nm Tech Node Running at 1 GHz.

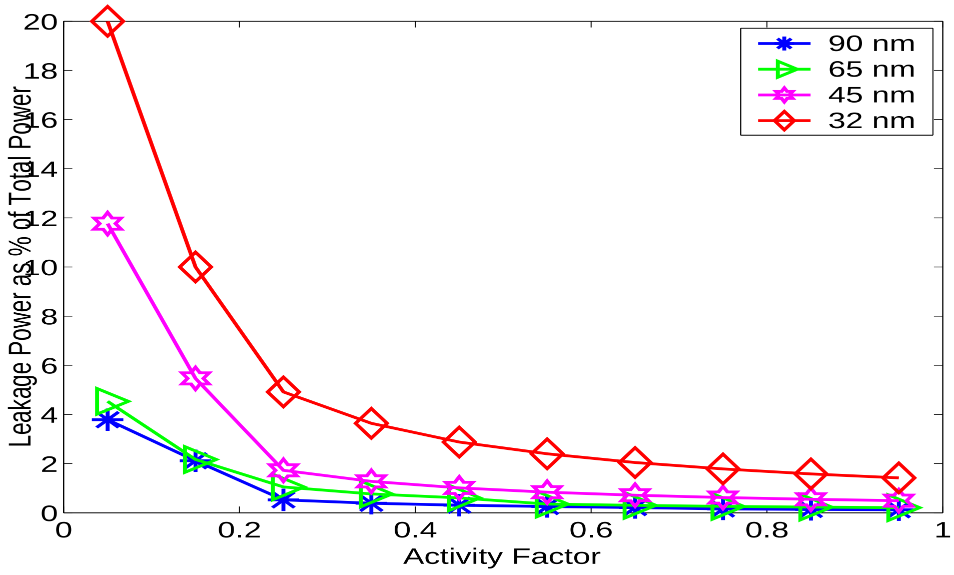


Figure 5.7: Leakage as % of Total Power for Different Activity Factors for Optimally Pipelined 5 mm Interconnect with 12*λ Pitch Running at 1 GHz

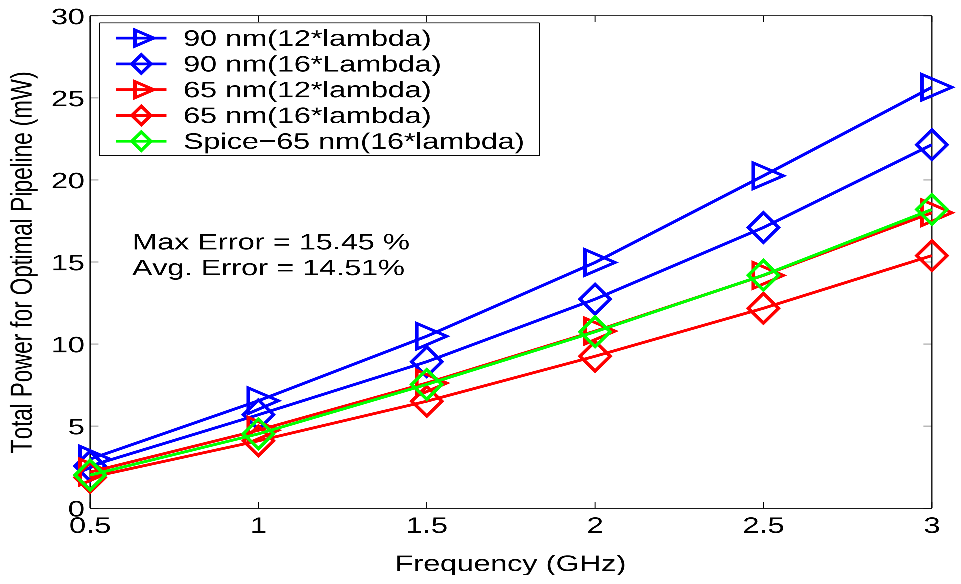


Figure 5.8: Frequency vs. Power for Optimal Degree of Pipelining for 4 mm Interconnect

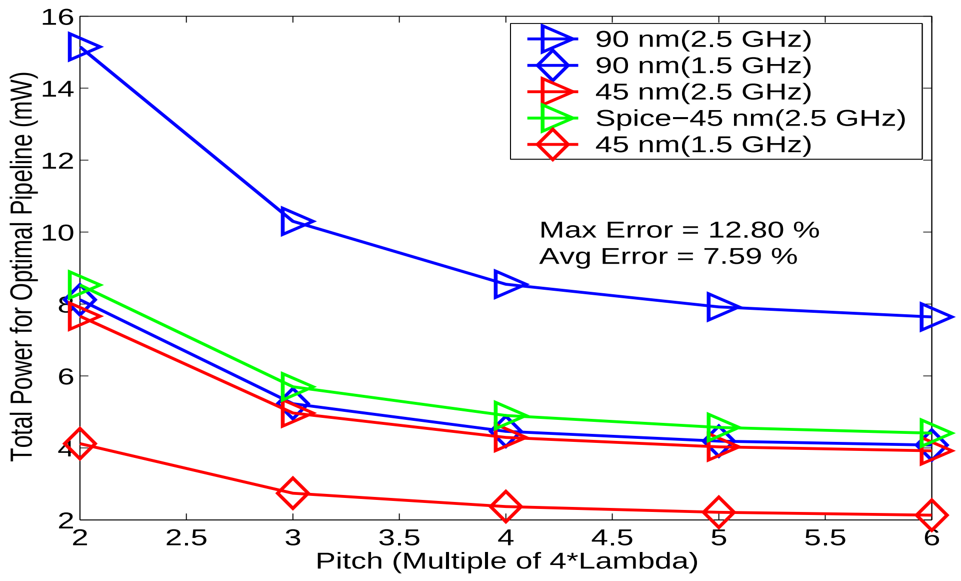


Figure 5.9: Pitch Vs Power for Optimal Degree of Pipelining for 2 mm Interconnect

to note is that our model has a discreet unidirectional error.

Figure 5.6 depicts component wise power breakup for 8 different voltage steps decreasing by 60 mV from operating voltage (1.2 V) for three different degrees of pipelining (2 being the optimal degree of pipelining in this configuration). It is clear from the graph that the wire power is the major component in the overall power of interconnect and clock power is the next top contributor. Figure 5.6 also shows the recurring trend that increasing the degree of pipelining first reduces power till optimal degree of pipelining (middle bar in this case) and than there is an increase because of reason explained earlier. The another trend depicted is the reduction in overall power w.r.t reduction in voltage which is quadratic in nature as shown by plot connecting the high points of the middle bar for different voltage steps.

Figure 5.7 depicts the leakage power percentage of total power for a 5 mm interconnect which is optimally pipelined and running at 1 GHz for a range of activity factors. The leakage power is high (20%) for smaller technologies such as (32 nm and 45 nm) and for low activity factor as expected. Though the fraction is not as high as in combinational circuits because as Figure 5.6 depicts that wire (which doesn't have a leakage component) makes a major fraction of interconnect power.

Figure 5.8 shows the change in power w.r.t frequency for optimal degree of pipelining for two different technology nodes (90nm and 65nm) and for two different wire pitch values ($12 * \lambda$ and $16 * \lambda$). The graph clearly shows the linear change in power w.r.t to the frequency for both the technology nodes. Increasing wire pitch within a technology decreases coupling capacitance which in turn reduces repeater size and number that leads to reduction in power. Again the power reduces in smaller technologies because of the reason explained above. The HSPICE validated graph for 65 nm and $16^*\lambda$ shows that the maximum error is 15.45% whereas the average error is 14.51% for varying frequency.

Figure 5.9 clearly brings out the trend of reduction in power for optimal degree of pipelining with increasing wire pitch for two different frequencies (2.5 GHz and 1.5 GHz) in two different technology nodes (90 nm and 45 nm). The reduction in power is proportional to inverse of the pitch. As mentioned above, increasing pitch actually reduces coupling capacitance which in turn decreases the load on repeaters and makes it possible to reduce repeater size and number of repeaters. Increasing wire pitch also reduces the optimal degree of pipelining as the signal can travel more distance for the same time period. Reduction in the degree of optimal pipeline reduces required number of flops which further reduces power. The trend towards linear reduction in power with reduced frequency is also visible in this graph so as the trend towards reduction in power in smaller technologies for the same length of interconnect. The HSPICE validation for 45 nm at 2.5 GHz shows that the maximum error is 12.8% whereas the average error is 7.59%.

5.5 Related Work

This section discusses some of the earlier work in the context of interconnect modeling and a comparison of our work with the earlier work. Banerjee et al., analyze the effect of changing repeater size and spacing on the power and delay of interconnects[27]. They observe that the delay variation is very shallow near the minimum delay point, which can be utilized to minimize power consumption. However, the wire width and spacing is fixed and its impact on power is not considered in this work. [28] considers the effects of wire dimension on bandwidth (irrespective of power) by considering two cases of same wire width and spacing and minimal spacing. In contrast, we propose a complete tool for modeling different interconnects across technologies. INTACTE optimizes the power by varying all the four parameters (i.e. repeater size, repeater spacing, wire width and wire spacing) in order to obtain minimal power for the desired interconnect. As our results show, wire width and spacing has significant impact on power and minimal spacing leads to comparatively higher power consumption.

[157] presents strong evidences of interconnects being one of major performance and power bottleneck in multi-core systems and a methodology of co-designing interconnect with other processor components. The study is based on earlier circuit level estimates of interconnect parameters[166][167][23]. [156] observes that different interconnects in processor have different bandwidth and latency requirements and interconnects composed of wires with different characteristics meet the power-performance goals of a system in a much better way. The evaluation has been performed based on guesstimates on circuit level study performed in[28].

Gupta et al.,[39] propose a methodology for first level power estimation of interconnect. They take into account activity factor and coupling factor in a similar fashion. They also propose a wire length estimation model which is complementary to our work. The most important limitation of their method is non-consideration of pipelining in interconnect and its overheads in terms of power and delay which is indispensable for global and semi-global interconnect they target. Many important components of power (such as leakage in repeaters and clock power) are not modeled in their work. It is also not clear how easy it is to obtain power estimates of desired interconnect across technologies by using their model.

[38] considers the impact of coupling between adjacent wires on power using a sophisticated method. The proposed method takes into account the time difference between transitions on adjacent wires using a timescale parameter called charge time which essentially represents the correlation time length between two events. The proposed method relies on layout information to be able to calculate coupling in a better fashion. Since we propose a high level methodology for interconnect energy modeling, in absence of detailed layout information, a simple calculation of coupling as done by profiling workload suffices to give reasonable accuracy in our model.

Orion is a simulator proposed for delay and power modeling specifically targeting off-chip interconnects[168]. The approach used is event driven that takes into account the events during execution to determine the power consumption in various logical interconnect components such as FIFO, arbiter, and crossbar. They lack a link model and rely on standard published data for accounting the power of links. However, link is an essential part of the communication and they also recognize the need for a parameterizable model for link power to be able to perform architectural design trade-offs[168]. Our work complements their work by providing a thoroughly validated model for optimizing the power of link used to connect the logic modules.

5.6 Conclusions

In this chapter, we presented a tool that fills the gap between architect's need and circuit level models for design of interconnects. The tool takes architectural parameters such as length, bit-width, latency and target technology and provides a set of interconnect options with varying degree of area, pipelining, and power budget using pre-characterized estimates of circuit parameters for different interconnect components. The major motivation behind development of this tool has been co-designing interconnect with other architectural components that is highly desirable for high level synthesis and design of embedded SoCs. The proposed tool is not only useful to make micro-architectural and architectural trade-offs but also to evaluate various architectural and compiler optimizations. Though, the scope of the tool is much general, we have specifically used the tool to evaluate the benefits of heterogeneous interconnects in the context of clustered VLIW architectures as described in the next chapter.

Chapter 6

Energy Optimization for Interconnects

6.1 Introduction

Clustered VLIW architectures helps to combat the scalability problem by making components simpler and thereby improving performance and energy consumption. However, an interconnection network is required for the communication of data among different clusters. This communication happens over long wires having high load capacitance which in effect takes more time and consumes more energy consumption[5][23]. Earlier studies report that a very high percentage (20% to 30%) of total processor energy consumption is attributed to interconnects[169][24]. Thus, interconnect is becoming major performance and power bottleneck in distributed architectures. Clearly, clustered architectures are attractive only if their benefits outweighs the performance and energy penalties due to interconnection network. Thus, efficient means of using interconnects are important for clustered VLIW architectures. The primary goal so far has been reduction in the latency of communication to minimize communication delays[12][42].

Wire delay is a function of its RC time constant where R and C are the resistance and the load capacitance of the wire respectively. As described in the last chapter, R and C are both linear functions of wire length and thus the wire delay has quadratic dependency on the wire length. It is made linear by dividing the wire into segments and by using repeaters[23]. Closely spaced repeaters can help to improve latency but have more area and energy overheads. By tuning the repeater size and spacing between successive repeaters, different latency and power profiles can be obtained for wires. It has been shown that using 45nm technology, it is possible to design repeaters consuming 1/5 the energy but having twice the delay[27][28]. Though VLSI technology enables design of interconnects with wires having different energy characteristics, to the best of our knowledge, there have been no efforts in terms of using energy efficient interconnects for clustered VLIW architectures.

In this chapter, we propose and evaluate a new energy-aware instruction scheduling algorithm which exploits interconnects of different characteristics in the context of clustered VLIW architectures. The proposed algorithm takes into consideration the interconnect characteristics, and communication slacks of data values together with the scheduling slacks of instructions while steering the communication to an appropriate interconnect, thereby reducing energy consumption without much performance degradation. We consider different flavors of interconnects such as latency-optimized and energy-optimized together with the variation in degree of clustering to perform a detailed performance evaluation. This helps in understanding the energy-performance trade-off in using different varieties of clustered architecture and in making design decisions for clustered architectures targeting embedded domains.

The rest of the chapter is organized as follows. In section 2, we present the motivation for this work with some quantitative results. Section 3 gives a detailed description of our energy-aware instruction scheduling algorithm with a brief mention of implementation details. We also give an example in section 3 to illustrate the notion of communication slack and how it is exploited by our algorithm to yield energy benefits without performance degradation. Section 4 presents a detailed performance evaluation of the proposed algorithm. Section 5 presents related work and we conclude in section 6.

6.2 Motivation

Previous studies have reported that performance degrades by 12% when the latency of communication is doubled for a four clustered architecture, and that increasing the interconnection bandwidth from one to two improves the performance by as much as 10%[43]. We also observe in our experiments that increasing the inter-cluster communication bandwidth indeed gives similar performance benefits. However, we argue that not all data values need to be communicated on a

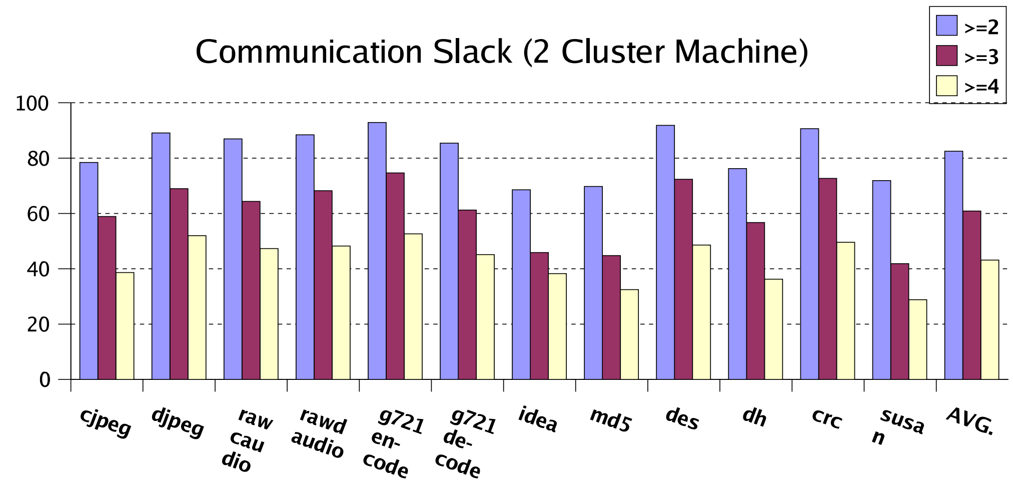


Figure 6.1: Communication Slack for Two-Cluster Machine Model

high speed path. Though some communications are critical and delaying them can have severe impact on performance, we observe that many communications are non-critical and can still happen on a slow path without affecting performance. *We define the communication slack of a data value on clustered architectures as the number of cycles between the time when the data value to be communicated becomes available (due to completion of execution of the producing instruction) and when the instruction that requires the data value is actually scheduled.* Various causes that can affect the available communication slack of a data value on clustered architecture are as follows :

1. Data dependency among instructions adds to the available communication slack of data values because different parents of an instruction may produce results at different points in time.

2. Limitation on the available number of functional units makes an instruction requiring communication getting scheduled many cycles after it actually becomes ready to be scheduled.

3. Limitations on the number of available cross-paths, their bandwidth, and latency of cross-path communication are another factors that add to the communication slack of data values.

4. Finally, the peculiarities of cluster scheduling algorithms also add to the communication slack of data values.

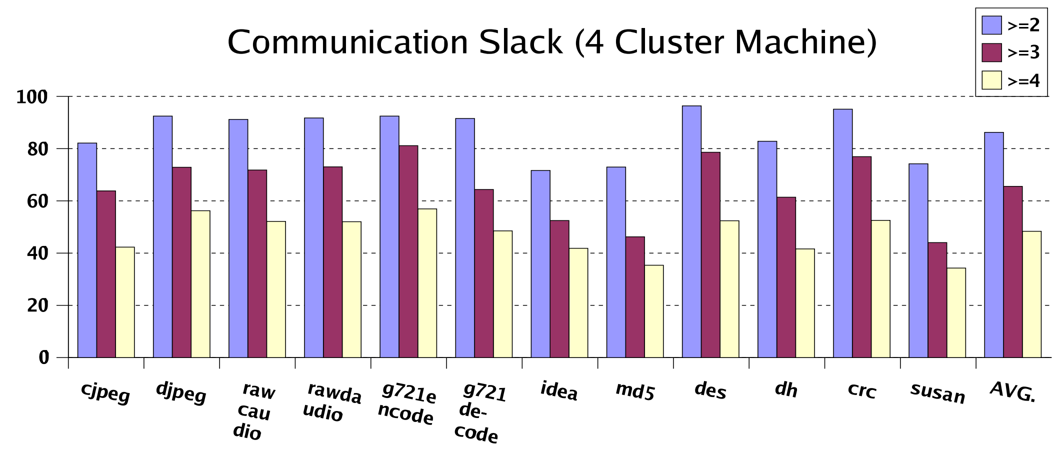


Figure 6.2: Communication Slack for Four-Cluster Machine Model

Figure 6.1 presents quantitative results to substantiate our arguments. This figure presents the percentage of required communication that has a slack of three cycles (two cycles and four cycles) or more. These results are for a two-cluster machine which has two high speed bidirectional cross-paths between clusters. We observe that all the benchmarks have many communications with high slack values. In particular *djpeg, g721encode, des, and crc* have 70% to 75% of communications with slack value of three cycles or higher. On an average, we observe that 60.88% (82.51% and 43.16%) of communications can sustain a latency of three cycles (two cycles and four cycles respectively) or higher. The corresponding results for a four-cluster machine are shown in Figure 6.2. It is clear that that the available communication slack is even higher for a four-clustered machine. Thus, even though having a cross-path with inter-cluster communication bandwidth of two is desirable from a performance point of view, having both the wires optimized for low latency is an over kill. This is because improving the latency of communication channel requires closely spaced repeaters which increase the area and energy overheads of repeaters[27]. A more suitable option is to design interconnect with some paths optimized for latency and others for energy[28]. Thus critical communication can take place over the fast but more energy-consuming wires, and the other not-so-critical communication can happen on the slower but energy-efficient wires. Such a design is going to be beneficial only if the target workload has a sufficient number of communications that are non-critical or as we call it have enough communication slack. Further mechanisms (software or hardware) that can steer the communications to the appropriate cross-path depending upon the communication slack of the data value should be available. Our instruction scheduler is one such mechanism.

6.3 The Scheduling Algorithm

The Algorithm described in chapter 4 only exploits the instruction slack to optimize the leakage energy in functional components. In this section, we present our instruction scheduling algorithm that specifically exploits the communication slack of data values as well as scheduling slack of instructions to steers non-critical communication to slow-interconnects thereby saving energy in interconnects. The proposed algorithm is implemented by modifying the list scheduling algorithm (designed and implemented for flat VLIW architecture) in the ELCOR backend of Trimaran infrastructure[137][145]. As mentioned earlier, we take an integrated approach[43][134][135] to cluster scheduling that makes the cluster assignment decision during temporal scheduling as compared to the phase-decoupled approaches[130][131][132] which perform cluster assignment prior to temporal scheduling. Essentially, our integrated clustered scheduling algorithm consists of the following main steps.

1. Prioritizing the ready instructions
2. Assignment of a cluster to the selected instruction
3. Assignment of cross-paths for transferring data values (from other clusters) to the target cluster.

In what follows, we describe how each of these step is performed in our algorithm. An outline of our algorithm is shown in Algorithm 4. A pseudo code of complete scheduling framework is available in chapter 7.

6.3.1 Prioritizing the Ready Instructions

Instructions in the ReadyList are prioritized using a priority function that uses instruction slack and number of consumers of the instruction respectively. The definition of instruction slack and motivation behind using the above mentioned criteria to prioritize instructions is same as explained in chapter 4.

Algorithm 4 Energy Efficient Scheduling for Interconnects

```
 1: Initialize ReadyList with root operations of the dependence graph of the
    region to be scheduled
 2: $CurrentCycle \leftarrow 0$
 3: while (ReadyList is not empty) do
 4:    Initialize EarlyCycle with CurrentCycle, and LateCycle with SchedulingCycle
       determined using performance oriented scheduling
 5:    $slack = LateCycle - EarlyCycle$
 6:    while (Not all operations in ReadyList have been tried once) do
 7:       $(CurrentOperations \leftarrow UnSchedList.pop())$
 8:       Initialize MinCommCost, MinCommOption and MinCommEnergy
 9:       for (CurrentCluster ranging from FirstCluster through LastCluster) do
10:          Compute the Cross-path Requirements in CurrentCommOption
11:          Compute the Communication Cost in CurrentCommCost and Communication
             Energy Cost in CurrentCommEnergyCost
12:          if (FU and Cross-paths required by CurrentOperation are not available in
                  CurrentCycle for CurrentCluster) then
13:             CONTINUE
14:          end if
15:          if $((CurrentCommCost <= MinCommCost))$ then
16:             MinCommCost=CurrentCommCost
17:             MinCommOption=CurrentCommOption
18:             MinCommEnergy=CurrentCommEnergyCost
19:             TargetCluster=CurrentCluster
20:          end if
21:       end for
22:       if $(CurrentCommEnergyCost < COMM\_THRESHOLD$ or $Slack <$
             $SLACK\_THRESHOLD)$ then
23:          while (CurrentComm=CurrentCommOption.pop()) do
24:             Determine the EarlyCommCycle, LateCommCycle, and the CommSlack
                for CurrentComm
25:             Schedule the CurrentComm using minimum energy consuming cross-path
                between EarlyCommCycle and LateCommCycle
26:          end while
27:          Schedule CurrentOperation in CurrentCycle on TargetCluster
28:       else
29:          ReadyList.add(CurrentOperation)
30:       end if
31:    end while
32:    $CurrentCycle \leftarrow CurrentCycle + 1$
33:    $ReadyList.update()$
34:    $updateFUStatus()$
35: end while
```

6.3.2 Cluster Assignment

Once an instruction has been selected for scheduling, we make a cluster assignment decision. The constraints for a cluster to be considered for binding are as follows and are same as mentioned earlier in chapter 4:

- The chosen cluster should have at least one free resource of the type needed to perform this operation
- Given the bandwidth of the channels among clusters and their usage, it should be possible to satisfy the communication needs of the operands of this instruction on the cluster by scheduling these communications in the earlier cycles.

Selection of a cluster from the set of the feasible clusters is done based on the following criteria. We determine the earliest time when we can schedule the operation under consideration on each of the clusters in the feasible cluster list while adhering to all the resource and communication constraints. The operation is primarily assigned to that cluster where it can be scheduled at the earliest after accommodating all the communications. In case of a tie in this metric, the operation is assigned to the cluster that minimizes communication requirements. The communication cost is computed by determining the number and type of communications needed by a binding in the earlier cycles as well as the communication that will happen in the future. Future communications are determined by considering the successors of this instruction which have one of their parents bound on a cluster different from the cluster under consideration. This is because if the instruction is bound to the cluster under consideration, it will surely lead to communication(s) in the future while scheduling the successor of the instructions in the future (refer chapter 3 for an example). Although, we have experimented with many other heuristics for cluster assignment, the above mentioned heuristic seems to generate the best schedule in almost all cases.

6.3.3 Cross-path Binding

The cross-path assignment scheme is designed to minimize the energy consumption due to inter-cluster communication without affecting runtime performance. In order to meet this objective, the low power cross-paths are used in preference to the high power cross-paths wherever possible. More precisely, the assignment of cross-paths to communications is done as follow. To schedule a communication, its earliest scheduling cycle, latest scheduling cycle, and slack values are determined first. The earliest scheduling cycle for a communication is the cycle in which the data value to be communicated is produced in the source cluster, plus one. The latest scheduling time for communication is the scheduling cycle of first consuming instruction, minus one. The difference between the earliest scheduling cycle and the latest scheduling cycle is the communication slack. In order to avoid delaying the consuming instruction and the consequent possible stretch of the schedule, a communication is assigned to a least energy consuming cross-path that can transfer the data value within the available slack for communication. Thus the cross-path assignment scheme maximizes the usage of low power cross-paths subject to the availability of slack in the communication, and thus, as far as possible, performance degradation is minimized and energy saving is maximized.

The algorithm exploits the instruction slack also and converts it to communication slack to aggressively save the communication energy. Thus, the algorithm defer an instruction that requires communications with total energy cost above communication threshold (COMM_THRESHOLD) if it possess a slack that is above a slack threshold (SLACK_THRESHOLD). The average energy cost of communications associated with a binding is determined according to following cost metric.

$$\begin{aligned} CommEnergyCost =& (A * \sum FastComm + B * \sum SlowComm + \\ & C * \sum FutureComm)/TotalComm \end{aligned} \tag{6.1}$$

Where FastComm represents number of transfer that can happen on fast cross path. SlowComm represents number of communication that can happen over slow cross path and FutureComm represents future communication which will happen on fast or slow cross-path depending on availability. TotalComm is the total number of communication that happen as a side effect of this binding. The selection of A, B, and C is architecture specific and depends on available communication options in a clustered architecture and their relative cost. A=1.0, B=0.33 and C=.67 work well in practice for the kind of heterogeneous interconnect, we consider in our experiments. Of course, the weight can be chosen to reflect the heterogeneous interconnect under consideration. We have found that SLACK_THRESHOLD = 1 and COMM_THRESHOLD=.67 is a suitable choice of scheduler parameters for the heterogeneous interconnect under consideration. The above heuristic leads to scheduling the instruction under consideration at a later cycle preferably on a slow but more energy efficient cross-path but with minimum possibility

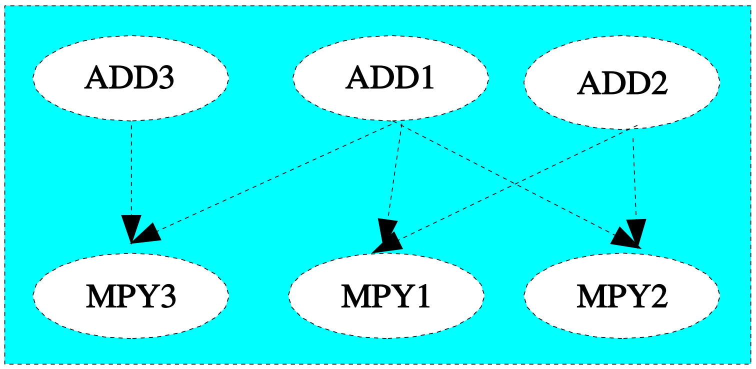
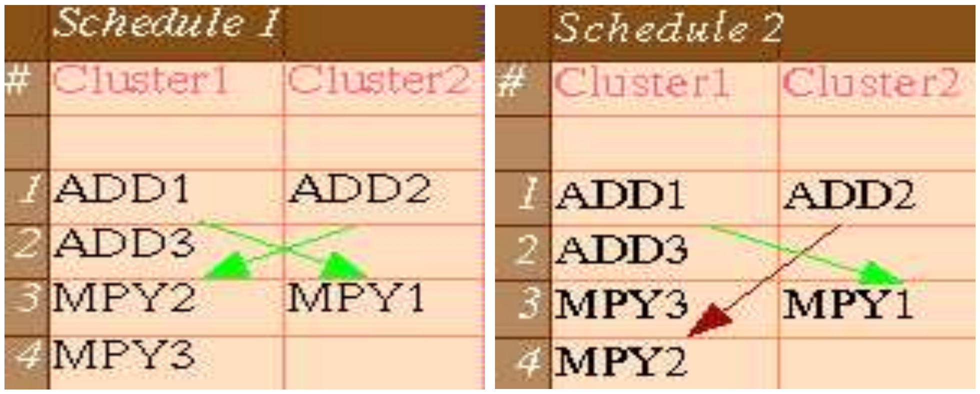


Figure 6.3: An Example Data Dependency Graph



Schedule 1

#	Cluster1	Cluster2
1	ADD1	ADD2
2	ADD3	
3	MPY2	MPY1
4	MPY3	

Schedule 2

#	Cluster1	Cluster2
1	ADD1	ADD2
2	ADD3	
3	MPY3	MPY1
4	MPY2	

Figure 6.4: Possible Schedules For Clustered VLIW Architecture (a) Schedule 1 (b) Schedule 2

of stretching the schedule. Again it is important to note that the next time this instruction is picked up for scheduling, its earliest scheduling time and hence the slack get updated. This guarantees that the slack of an instruction reduces monotonically and eventually comes below the threshold ensuring that it is scheduled. Hence the algorithm is guaranteed to terminate.

6.3.4 An Example

In this subsection, we present an example to illustrate the notion of communication slack and how it is exploited by the proposed scheduling algorithm to get energy benefits without hurting performance. Figure 6.3 shows a portion of a data dependency graph and Figure 6.4 shows two possible schedules for this dependency graph. We Assume a two-clustered machine with each cluster having an adder, a multiplier and a fast cross-path. Schedule 1 has ADD1 and ADD2 scheduled on adders of cluster 1 and cluster 2 respectively in cycle 1. To perform multiplication, the results of these operations are transferred to the other cluster in cycle 2. The remaining addition operation ADD3 is also initiated in cycle 2 on cluster 1. The results of ADD1 and ADD2 can be used in cycle 3 on cluster 1 and cluster 2 respectively to perform MPY2 and MPY1 multipliers. Though MPY3 does not require any inter-cluster communication, it is still executed in cluster 1 at cycle 4 because of non-availability of a multiplier in cycle 3. The scheduler decides to schedule MPY2 ahead of MPY3 in schedule 1 assuming that MPY2 is on the critical path. However, MPY3 gets preference if it is on the critical path as shown in schedule 2. Note that in this case, MPY2 needs to be scheduled in cycle 4 on cluster 1 again because cluster 1 has only one multiplier. *The important point to note here is that the scheduler when scheduling MPY2 in cycle 4 in cluster 2 has the knowledge that it can take two cycles to transfer the result of ADD2 over the communication channel without stretching the schedule.* In such a situation if a slow but more energy-efficient cross-path is available, our schedulers decide to steer communication to such a cross-path (as shown with darker arrow in schedule 2). Notably, even though three additions are ready to be scheduled in the first cycle only two of them can be scheduled (only two adders are available in this case). Similarly though the addition operations finish in opposite clusters in cycle one the results can not be utilized for multiplications in cycle 2 because it takes at least one cycle to transfer the results to the other clusters. This shows how contention among computation and communication resources in clustered architectures manifests itself in the form of greater computation and communication slack. Notably, the contention for resources is more in clustered architectures as compared to flat architectures because of distribution of resources. Our scheduling algorithm leverages this increased slack and takes into consideration the criticality of an instruction and the available cycles to communicate requisite data values while scheduling an instruction in a given cycle. Accordingly, communication is assigned to the most energy-efficient cross-path that can transfer the value in the available communication cycles.

6.4 Experimental Evaluation

We have performed a detailed experimental evaluation of the proposed algorithm in terms of execution time and interconnect energy benefits for a 2-clustered and a 4-clustered machine model. The cluster configurations are same as described earlier in Chapter 3. The inter-cluster cross-paths allow transfer of two data values between clusters per cycle. The interconnect energy benefits of using LP configuration (having 1 fast and 1 slow cross-path) with the proposed scheduling algorithm are presented with respect to the LL configuration (having 2 fast cross-path) as described in the following subsection.

6.4.1 Energy Model

For determining the benefits in interconnects, we have used the INTACTE energy model described in last chapter[46]. INTACTE is a CACTI like tool which takes as input high level architectural parameters for interconnects such as delay, degree of pipelining, technology, voltage, length, and pitch etc and provide the energy estimates of the interconnect that meets the specified delay constraint in target technology at specified voltage. For our experiments, we have determined the energy cost of 1 mm interconnect at three different technologies i.e. 90 nm, 65 nm, and 45 nm and for 1.5 GHz clock. The delay for fast interconnect (called L) is set to 1 clock cycle and delay for slow interconnect (called P) is set to 3 clock cycles. The dynamic and leakage energy estimates given by INTACTE are used along with the usage statistics determined by trimaran simulation to calculate the interconnect energy estimates. We refer the reader to the last chapter for details on INTACTE energy estimation methodology.

6.4.2 Results

We have performed a detailed experimental evaluation of the proposed algorithms in terms of execution time, energy benefits, as well as impact of technology scaling on the energy benefits.

Figure 6.5 presents the percentage increase in execution time of using LP configuration (having 1 fast and 1 slow cross-path) with the proposed scheduling algorithm over LL configuration (having 2 fast cross-path). The average percentage increase in execution time is 1.8% and 1.5% for 2-clustered and 4-clustered machine respectively. Figure 6.6 depicts that the percentage energy benefit of using LP configuration (as compared to LL configuration) scheduled using Algorithm 4 is 41.5% and 46.8% respectively for 2-clustered and 4-clustered machine. These results are determined using the INTACTE[46] interconnect energy model. Programs having more communications with high slack values *viz. djpeg, g721encode, des, and crc* suffer only a marginal performance degradation and gives significant energy benefit because many of the communication in these programs are scheduled on slow cross-path by the proposed scheduling

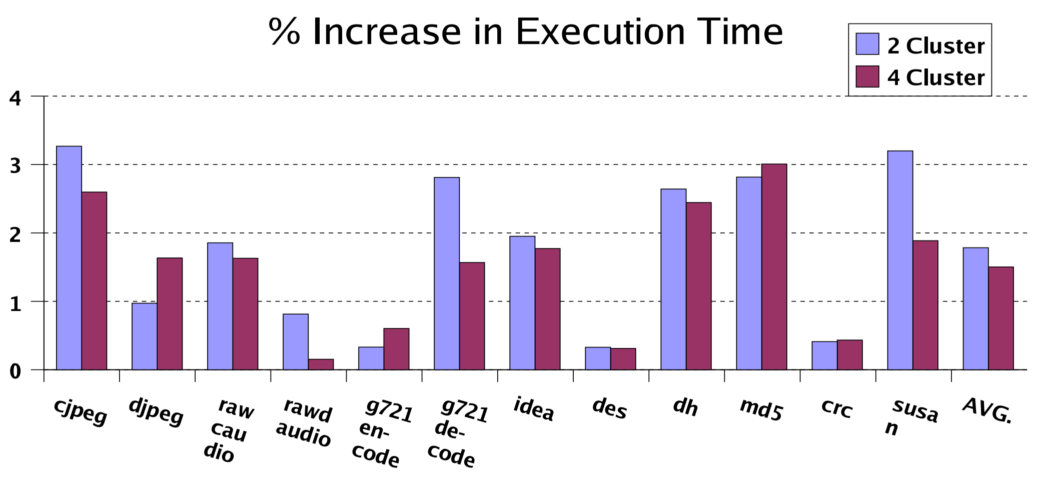


Figure 6.5: % Increase in Execution Time of LP Conf. w.r.t. LL Conf.

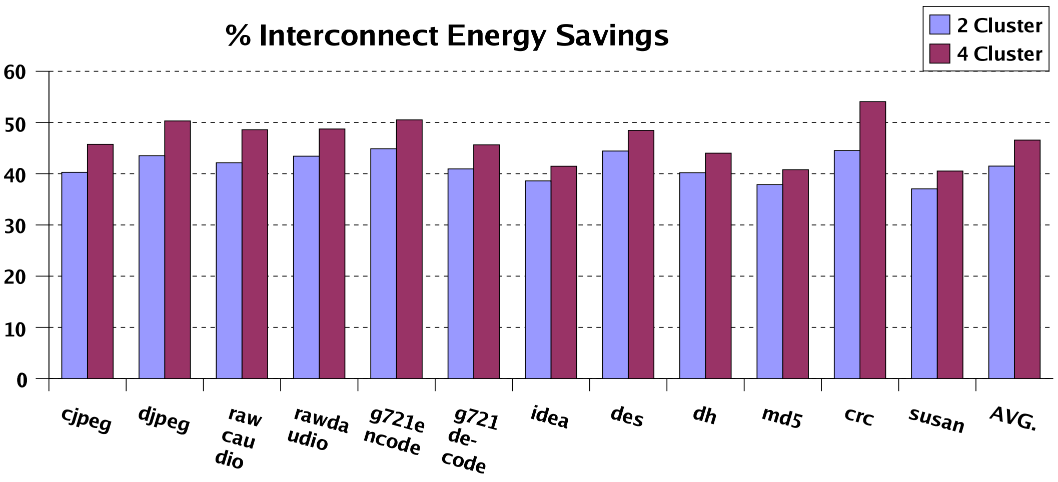


Figure 6.6: % Energy Benefit of LP Conf. w.r.t. LL Conf.

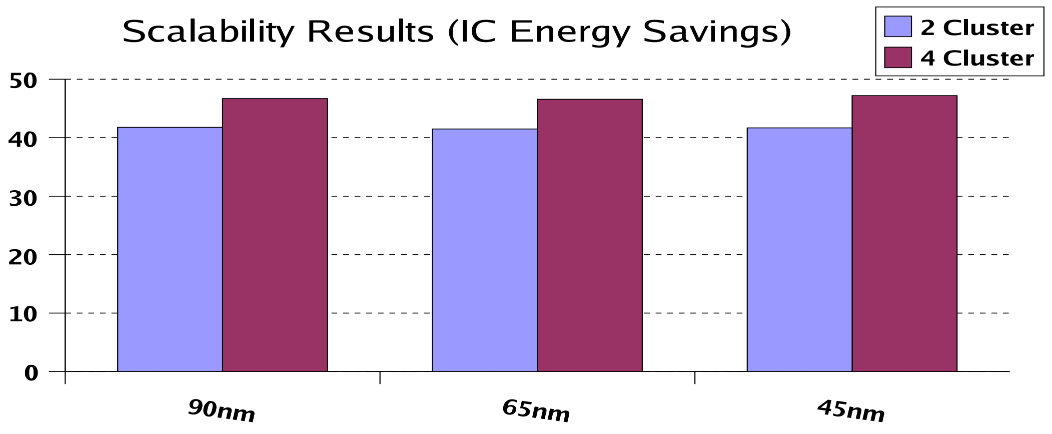


Figure 6.7: Scalability Results for Interconnect Energy Savings

algorithm 4. In Contrast, programs with fewer communications with high slack values *viz. idea, md5, and susan* suffer a moderate performance degradation with the LP configuration and give relatively less energy benefits. However, a very small overall performance degradation occurs with the LP configuration whereas a significant amount of communication energy savings are obtained. This shows the effectiveness of our communication scheduling mechanism that selectively maps communications with high latency tolerance onto a high latency cross-path and communication with low latency tolerance to low latency cross-path.

6.4.3 Sensitivity Analysis

Figure 6.7 depicts the impact of technology scaling on the benefit of our compiler scheduling scheme for heterogeneous interconnect for three different technology nodes namely 90nm 65nm and 45nm. The parameters for different technology nodes for scalability analysis were obtained from INTACTE. From Figure 6.7, it is clear that our scheme gives roughly the same benefits across different technology nodes with slight variations. This is because the smaller technologies though leads to increase in leakage part in some interconnect components such as repeaters, buffers, and flops (note that wire does not have the leakage component), there is also reduction in dynamic component of the voltage in all the components including wires. This leads to roughly the same benefits of using the slower interconnect over faster interconnect across different technology nodes. Thus the scheme gives appreciable benefit across technologies and is particularly useful for more decentralized architectures where interconnects increasingly make major proportion of the total energy consumption of processor.

6.5 Related Work

In this section we compare and contrast our work with some of the earlier work in the context of efficient interconnect design. Reader is referred to chapter 2 for a general description on circuit level techniques for interconnect energy savings and chapter 5 for a description on interconnect modeling related work.

As compared to reducing energy consumption in function blocks, study of energy efficiency in interconnects is still in its infancy. Previous work has concentrated on improving latency for interconnects in the context of decentralized architectures. Gonzalez et al.,[42] have evaluated different kinds of interconnects with different topologies and concluded that a point-to-point interconnect with an effective steering scheme is more efficient than a bus-based interconnect. Their experimental results also demonstrate that an asynchronous interconnect offers a performance comparable to an idealized interconnect at a low hardware implementation cost. Terechko et al.,[12] has proposed various inter-cluster communication models for clustered architecture and perform a quantitative analysis to compare their benefits.

Balasubramonian et al.,[29]. have used the interconnect energy estimates proposed in [28] to evaluate techniques such as cache pipelining, exploiting narrow bit-width operands, and interconnect load balancing in the context of superscalar architectures with heterogeneous interconnect. In contrast, our work is more focused on how communication slack in the context of clustered VLIW architecture can be exploited to gain the energy benefits. Our results demonstrate that compile-time instruction scheduling utilizing a larger view of program can combine the instruction scheduling and communication scheduling in a profitable manner. On the other hand, a architecture with dynamic scheduling suffers from the problem of limited program view and incurs overheads and complexities of extra hardware for exploiting heterogeneous interconnects at run-time. Thus, the choice of a heterogeneous interconnect is more suitable and beneficial for statically scheduled VLIW architectures as compared to dynamically scheduled architectures.

6.6 Conclusions

In this chapter, we proposed a new energy-aware instruction scheduling algorithm for clustered VLIW architectures that is capable of exploiting interconnect characteristics to get energy benefits without showing high performance degradation. The major conclusion that we draw form this work is that clustered architecture with heterogeneous interconnect offers better energy-performance trade-offs when used with an effective scheduling algorithm as compared to a cluster VLIW architecture with homogeneous interconnect optimized for latency. Experimental results using INTACTE model for interconnect energy estimation demonstrate that our instruction scheduling algorithm achieves 41.5% and 46.8% reduction in communication energy respectively for a 2-cluster and a 4-cluster machines with a marginal 1.8% and 1.5% degradation in performance. Scalability Analysis clearly shows that our techniques gives similar benefits across technologies. Thus, we believe that our technique will become more important with the continuing trends towards decentralized architectures with interconnects consuming more and more chip area.

Chapter 7

Integrated Energy Optimization for Functional Units and Interconnects

7.1 Introduction

In this chapter, we present an integrated scheduling algorithm that combines the schemes proposed earlier for energy savings in functional units and interconnects to simultaneously reduce the energy consumption in functional units as well as interconnects. The contention for limited number of functional and communication resources in clustered VLIW architecture leads to increased cycles of execution on a clustered machine as compared to an equivalent VLIW machine. Our combined scheme aggregates the scheduling slack of instructions and communication slack of data values to convert the inherent idleness of functional and communication resources in clustered architecture to energy gains. The rest of the chapter is structured as follows. Section 2 describes our combined scheme to optimize energy for functional units as well as interconnects. Section 3 describes our scheduling framework using pseudo code for major routines. Section 4 presents experimental results and a detailed analysis of results. We conclude in section 5.

7.2 The Scheduling Algorithm

Algorithm 5 is a combined algorithm that performs both leakage energy management in functional units as well as energy optimization in interconnect. However, It is important to note that the combined Algorithm 5 gives preference to leakage energy management in functional units when exploiting slack and uses any left over slack for energy optimization in interconnects. This avoids excessive performance degradation as well as extra transitions in the functional units and associated energy overheads. Another reason for giving preference to functional units when exploiting slack is that its contribution to overall processor energy consumption is more and it is one of the hot-spot and hence demands more aggressive energy saving.

We have implemented the combined algorithm in the ELCOR backend of the trimaran compiler infrastructure by modifying the core scheduling algorithm. This algorithm as well as the scheduling algorithms proposed in earlier chapters use the common scheduling routines as explained with pseudo code in section 3. All the algorithms essentially have the same steps (as described below) but they differ with respect to the criteria and preference for cluster selection, functional unit binding, and cross-path assignment policies. It is also important to note that the decisions with respect to selecting cluster, functional units, and cross-paths are interleaved and the order given below is used just for the ease of explaining the algorithm.

1. Prioritizing the ready instructions

2. Assignment of a cluster to the selected instruction

3. Assignment of functional unit to selected instruction in target cluster

4. Assignment of cross-paths for communicating the data values to target cluster

In what follows, we describe how each of these step is performed and how the combined scheduling Algorithm 5 proposed in this chapter differs in its functioning from algorithms proposed in the earlier chapters.

7.2.1 Prioritizing the Ready Instructions

Instructions in the ReadyList are prioritized using a priority function that uses instruction slack and number of consumers of the instruction. The procedure to calculate and update the dynamic slack of instructions while scheduling and the rationale behind this criterion for prioritizing of instructions is already explained in chapter 4 and chapter 6.

7.2.2 Cluster Assignment

The feasible set of clusters where an instruction under consideration can be scheduled is determined based on compute and communication resource availability as explained earlier. As described earlier, selection of a cluster from the set of the feasible clusters is done based on two criteria. Criterion one is to give preference to cluster that have an active functional unit to schedule the operation under consideration. Criterion two is to give preference to cluster that reduces the overall cost of communication. Algorithm 5 uses criterion one as a primary criterion for cluster selection and criterion two is used as secondary criterion. This is because Algorithm 5 gives preference to saving energy in functional units and also avoids performance loss.

The communication cost is computed by determining the number and type of communications needed by a binding in the earlier cycles as well as the communication that will happen in the future as explained in the last chapter.

7.2.3 Functional Unit Binding

The functional unit binding criterion are the same as those used by the earlier algorithm explained in chapter 4. We briefly explain the criterion here for clarity and completeness. The algorithm maintains an FU map that explicitly keeps track of the status of each functional unit. A functional unit is marked to be in sleep mode after one cycle of idleness and activated on next use.

If the functional unit required for the instruction under consideration is active in the target cluster, it is bound as usual. otherwise, the available slack of the instruction is considered. If the slack is below a threshold (we use the threshold value of 0 in our experiment) the functional unit required by the instruction is woken up. In case there is more than one alternative available (for activating), the functional unit which is in sleep mode for a longer duration is woken up in order to amortize the cost of waking up. In case the instruction possesses enough slack, its scheduling is deferred to a future cycle and it is put back in the ReadyList. Note that the next time this instruction is picked up for scheduling, its earliest scheduling time and hence the slack gets updated. This guarantees that the slack of an instruction reduces monotonically and eventually comes below the threshold ensuring that it is scheduled. Hence the algorithm is guaranteed to terminate.

7.2.4 Cross-path Binding

The cross-path assignment scheme is designed to minimize the energy consumption due to inter-cluster communication without affecting runtime performance. As described in the last chapter, the cross-path assignment scheme maximizes the usage of low power cross-paths subject to the availability of slack in the communication, and thus, as far as possible, performance degradation is minimized and energy saving is maximized. The calculation of communication slack is also done in the same fashion as described in the last chapter.

Combined Algorithm 5 differs from Algorithm 4 explained in chapter 6 in an important way. The Algorithm 4 (explained in chapter 6) uses a more aggressive scheme that exploits the instruction slack also and converts it to communication slack in order to defer instructions and save energy in interconnects. In contrast, the combined Algorithm 5 exploits instruction slack only for leakage energy management in functional units (as Algorithm 1) and uses the communication slack only for interconnect energy optimization. This limits the excessive performance degradation and improves the overall energy-delay product in a much appreciable way.

7.3 Scheduler Implementation

This section explains the implementation of scheduler in detail with the pseudo code for various procedures (refer Algorithm 5 and Procedure 6 to Procedure 11) mimicking the code-base of our scheduler infrastructures. The cycle scheduler as implemented in Trimaran[137] targets a flat VLIW class of architectures[145]. It maintains a *ReadyList* of operations whose predecessors have already been scheduled. In each iteration of the main scheduling loop, the highest priority operation is selected from the *ReadyList* and scheduled in the current cycle if it satisfies all the resource constraints. In our implementation, slack and hence priority is recalculated at the beginning of each cycle for all the operations in the *ReadyList* using the schedule time determined in an earlier performance oriented scheduling pass. After taking the highest priority instruction, *DetermineBestAlternative* determines all scheduling alternatives of *CurrentOperation* in all clusters. *DetermineBestAlternative* scans each cluster one by one and checks if the instruction under consideration can be scheduled in the current cycle on the cluster under consideration using some functional unit (though it can be in sleep mode) and available cross-path for requisite communication (for this scheduling). It prioritizes clusters which match this feasibility criterion into two categories. *FirstTarget* stores the cluster that can accommodate the instruction under consideration in the current cycle with minimum communication cost and also has an active functional unit. *SecondTarget* stores the cluster that can accommodate the instruction under consideration in the current cycle with minimum communication cost but does not have an active functional unit.

Algorithm 5 Energy Efficient Scheduling for Fus and ICs (priority to Fus)

```
1: Initialize ReadyList with root operations of the dependence graph of the region to be
   scheduled
2: $CurrentCycle \leftarrow 1$
3: while (ReadyList is not empty) do
4:    Initialize EarlyCycle with CurrentCycle, and LateCycle with SchedulingCycle de-
      termined using performance driven schduling
5:    $slack = LateCycle - EarlyCycle$
6:    while (Not all operations in ReadyList have been tried once) do
7:       $CurrentOperations \leftarrow UnSchedList.pop()$
8:       $ClusterPriority \leftarrow 1$ # Scheduling for clustered configuration
9:       $Target \leftarrow DetermineBestAlternative($
                   $CurrentOperation, CurrentCycle, ClusterPriority)$
10:      if $((TargetCluster == -1)$ or $(Slack \geq SLACK\_THRESHOLD$
              $and\ (Target.Wakeup == 1)))$ then
11:         ReadyList.add(CurrentOperation)
12:         CONTINUE
13:      end if
14:      $ScheduleComm(Target.CommOption)$
15:      $ScheduleFu(CurrentOp, Target.fu, Target.Cluster, CurrentCycle)$
16:   end while
17:   $CurrentCycle \leftarrow CurrentCycle + 1$
18:   $ReadyList.update()$
19: end while
```

To make these decisions, it uses procedures *DetermineBestFuAlternative* and *DetermineBest-CommAlternative* to return the best functional unit for scheduling the instruction under consideration and the best option for communicating the requisite data values for the instruction under consideration and its associated cost. The best functional unit for performing the operation is one which is available and active. If no functional units is active, this procedure returns the functional unit which is in sleep mode for the longest time as FallBackAlternative. For determining the best alternative for communication, procedure *DetermineBestCommAlternative* considers each required communication one by one, and for each such communication it chooses

Procedure 6 DetermineBestAlternative

```
1: $INPUT$: ThisOp, ThisCycle, ClusterPriority
2: $OUTPUT$: Determines best scheduling alternative for this binding
3: $FirstTarget.Cluster \leftarrow -1$;
4: $FirstTarget.CommCost \leftarrow$ 1000000;
5: $FirstTarget.CommEnergyCost \leftarrow$ 1000000;
6: $SecondTarget.Cluster \leftarrow -1$
7: $SecondTarget.CommCost \leftarrow$ 1000000;
8: $SecondTarget.CommEnergyCost \leftarrow$ 1000000;
9: for (CurrentCluster ranging from FirstCluster through LastCluster) do
10:   if (FU required by ThisOp is available in ThisCycle for CurrentCluster) then
11:     if (Cross-paths required by ThisOp are available in ThisCycle for CurrentCluster)
        then
12:       $CommAlternative \leftarrow DetermineBestCommAlternatives($
                  $ThisOperation, CurrentCluster, ThisCycle)$
13:       $FuAlternative \leftarrow DetermineBestFuAlternatives($
                  $ThisOperation, CurrentCluster, ThisCycle)$
14:       if ((FU under consideration is in active mode) and (FirstTar-
          get.CommEnergycost $\geq$ CommAlternative.CommEnergyCost)) then
15:         $FirstTarget.CommCost \leftarrow CommAlternative.CommCost$
16:         $FirstTarget.CommEnergyCost \leftarrow CommAlternative.CommEnergyCost$
17:         $FirstTarget.CommOption \leftarrow CommAlternative.CommOption$
18:         $FirstTarget.Cluster \leftarrow CurrentCluster$
19:         $FirstTarget.Fu \leftarrow FuAlternative.Fu$
20:       else
21:         if (SecondTarget.CommEnergycost $\geq$ CommAlternative.CommEnergyCost)
            then
22:           $SecondTarget.CommCost \leftarrow CommAlternative.CommCost$
23:           $SecondTarget.CommEnergyCost \leftarrow CommAlternative.CommEnergyCost$
24:           $SecondTarget.CommOption \leftarrow CommAlternative.CommOption$
25:           $SecondTarget.Cluster \leftarrow CurrentCluster$
26:           $SecondTarget.Fu \leftarrow FuAlternative.FallBackFu$
27:         end if
28:       end if
29:     end if
30:   end if
31: end for
32: if (($FirstTarget.cluster != -1$) and ($ClusterPriority == 0$ or
          $FirstTarget.CommCost \leq SecondTarget.CommCost$)) then
33:   $FinalTarget \leftarrow FirstTarget$
34:   $FinalTarget.Wakeup \leftarrow 0$
35: else
36:   $FinalTarget \leftarrow SecondTarget$
37:   $FinalTarget.Wakeup \leftarrow 1$
38: end if
39: RETURN $FinalTarget$
```

Procedure 7 DetermineBestFuAlternative

```
1: $INPUT$: ThisOp, ThisCluster, ThisCycle
2: $OUTPUT$: Best functional unit and fallback functional unit for this Binding
3: for (CurrentFu ranging from FirstFu through LastFu in ThisCluster) do
4:    if (CurrentFu is available in ThisCycle and Current Fu can execute ThisOp) then
5:       if (CurrentFu is active) then
6:          $Target.Fu \leftarrow CurrentFu$
7:          $Target.FallBackFu \leftarrow -1$
8:          RETURN Target
9:       else
10:         if ($CurrentFu.SleepPeriod > Target.SleepPeriod$) then
11:            $Target.FallBackFu \leftarrow CurrentFu$
12:            $Target.SleepPeriod \leftarrow CurrentFu.SleepPeriod$ # Length of sleep mode
13:         end if
14:      end if
15:   end if
16: end for
17: RETURN $Target$
```

Procedure 8 DetermineBestCommAlternative

```
1: $INPUT$: ThisOp, ThisCluster, ThisCycle
2: $OUTPUT$: Best communication alternatives and cost for this binding
3: for (All Communiction required for scheduling ThisOp on ThisCluster in ThisCycle)
   do
4:    $CurrentCommOption \leftarrow DetermineBestCommOption(CurrentComm)$
5:    Target.CommOption.add(CurrentCommOption)
6:    Target.CommCost+=DetermineCommunicationCost(CurrentCommOption)
7:    Target.CommEnergyCost+=DetermineCommEnergyCost(CurrentCommOption)
8: end for
9: RETURN $Target$
```

Procedure 9 DetermineBestCommOption

```
1: $INPUT$ : ThisComm
2: $OUTPUT$: A tuple CommOption (Comm, Cross) that associates best cross-path with
   ThisComm
3: CommOption.Comm $\leftarrow$ ThisComm
4: Determine the EarlyCommCycle, LateCommCycle and the CommSlack
   for ThisComm
5: Determine the free minimum energy consuming cross-path, target_cross that can tran-
   fer CurrentComm between EarlyCommCycle and LateCommCycle
6: CommOption.Cross $\leftarrow$ target_cross
7: RETURN $CommOption$
```

Procedure 10 ScheduleComm

```
1: INPUT: ThisCommOption
2: while (CurrentComm=ThisCommOption.pop()) do
3:     Schedule CurrentComm.Comm on CurrentComm.CrossPath
       on CurrentComm.ScheduleCycle
4:     Mark CurrentComm.CrossPath busy from CurrentComm.ScheduleCycle for
       CurrentComm.CrossPath.Latency Cycles
5: end while
```

Procedure 11 ScheduleFu

```
1: INPUT: ThisOp, ThisFu, ThisCluster, ThisCycle
2: Schedule ThisOp on ThisFu on ThisCluster on ThisCycle
3: Mark ThisFu in ThisCluster Busy from ThisCycle for ThisFu.Latency Cycles
```

the best communication option using procedure *DetermineBestCommOption*. Procedure *DetermineBestCommOption* returns the minimum energy consuming cross-path that is available and can schedule the communication under consideration within the available communication slack associated with the communication under consideration. The procedure *DetermineBestCommOption* finally aggregates the cost of all communication so determined to calculate the overall cost of the binding. It also computes the average energy cost of the binding by averaging the energy cost of all requisite communication for this binding using the cost function described in earlier chapter 6. Finally, the procedure *DetermineBestAlternative* decides between the Target clusters determined using two criteria mentioned above. Algorithm 4 described in chapter 6 uses *FirstTarget* or *SecondTarget* as *FinalTarget* depending on which one has the minimum communication cost. Algorithm 1 described in chapter 4 and Algorithm 5 of this chapter use *FirstTarget* if it is not empty otherwise they use *SecondTarget*.

The top level scheduling algorithm proceeds after getting information about *FinalTarget*. Algorithm 1 puts back the instruction into the *ReadyList* if it requires waking up a resource in *TargetCluster* (i.e., there is no cluster in current cycle with an active functional unit to schedule the instruction) and instruction slack is above a threshold. Thus, this scheme exploits scheduling slack of instructions to save energy in function resources by keeping them idle for a longer period of time while reducing transitions. Algorithm 4 puts the instruction back in the *ReadyList* if the *FinalTarget*.CommCost is above a threshold and instruction also has enough slack. By doing this, it converts the instruction slack into communication slack in order to save energy in interconnects. However, if this is not the case the instruction is scheduled in Target.Cluster on *Target.Fu* using *Target.CommOption*. Procedures *ScheduleComm* and *ScheduleFu* associate the communication and computation resources for this binding and mark them busy. Finally, Algorithm 5 does cluster selection based on communication cost subject to the availability of active resources and delays scheduling an instruction in the current cycle if it possesses enough slack and requires waking up a resource. Of course, the cross-path assignment scheme still assigns the communication required by this instruction to slower cross-paths subject to availability and saves energy in interconnects. However, an important difference between 5 and Algorithm 4 is that unlike Algorithm 4, instruction slack is not exploited to delay instructions in Algorithm 5 to save communication energy. As described earlier, the reason for giving preference to functional units when exploiting slack is because its contribution to overall processor energy consumption is more and it is one of the hot-spot and hence demands more aggressive energy saving. At the same time, this also avoids excessive performance degradation as well as extra transitions (and associated energy overheads) in functional units.

7.4 Experimental Evaluation

We have performed a detailed experimental evaluation of the proposed combined algorithm in terms of execution time, energy benefits, and overall energy-delay product for a 2-clustered and a 4-clustered machine model. The cluster configurations are same as described earlier in chapter 3. The functional unit energy benefits are presented with respect to the hardware only scheme ('MaxSleep') described in chapter 4. The interconnect energy benefits of using LP configuration (having 1 fast and 1 slow cross-path) with the proposed scheduling algorithm are presented with respect to the LL configuration (having 2 fast cross-paths). The usage statistics for functional units and interconnects (from trimaran simulation) is plugged into energy models to obtain the energy benefits for respective components. The energy model for determining functional unit energy benefit is same as the one described in chapter 4. The interconnect energy benefit is obtained by using INTACTE energy model in the same way as described in chapter 7. We present the energy results for the combined scheme for the current 65nm fabrication technology. Refer chapter 4, chapter 5, and chapter 6 for a detailed sensitivity analysis for functional unit energy savings and interconnect energy savings across technologies.

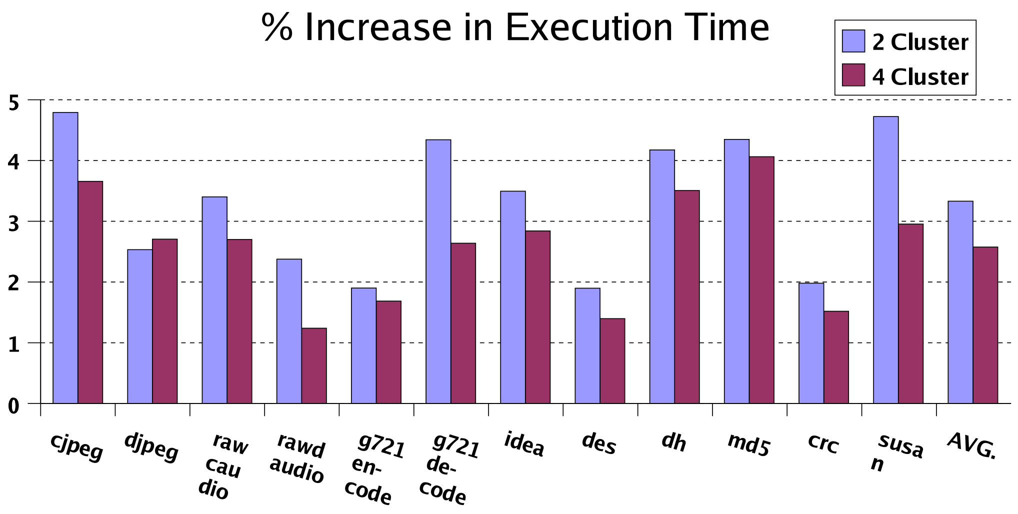


Figure 7.1: % Increase in Execution Time of LP conf. w.r.t. LL conf. for Algorithm 5

7.4.1 Results

Figure 7.1 presents the percentage increase in execution time by applying Algorithm 5 that exploit the instruction slack to save leakage energy in functional units by reducing the transition and also migrates the communication with high slack value to slow cross-path. The percentage increase is presented with respect to clustered architecture with LL configuration scheduled by performance oriented scheduler and have a hardware based scheme to optimize leakage energy in functional units[34]. The average increase in execution time is 3.3% and 2.5% for 2-clustered and 4-clustered architectures which is higher than the average increase in execution time for Algorithm 4 that only optimizes the interconnect energy. This is because the combined scheme uses instruction slack for doing the leakage energy management in functional units and this leads to more cases where two communication simultaneously need the fast cross-path. In other words some of the instruction slack that was used up implicitly in Algorithm 4 is no longer available (because it is already used up in Algorithm 5 the for better leakage energy management in functional units) and this shows up in the form of extra execution cycles in the combined scheme.

Figure 7.2 shows the percentage saving in communication energy of scheduling Algorithm 5 on LP configuration as compared to LL configuration. Algorithm 5 reduces the average communication energy by 37.1% and 43.1% which is slightly lesser than Algorithm 4 because the

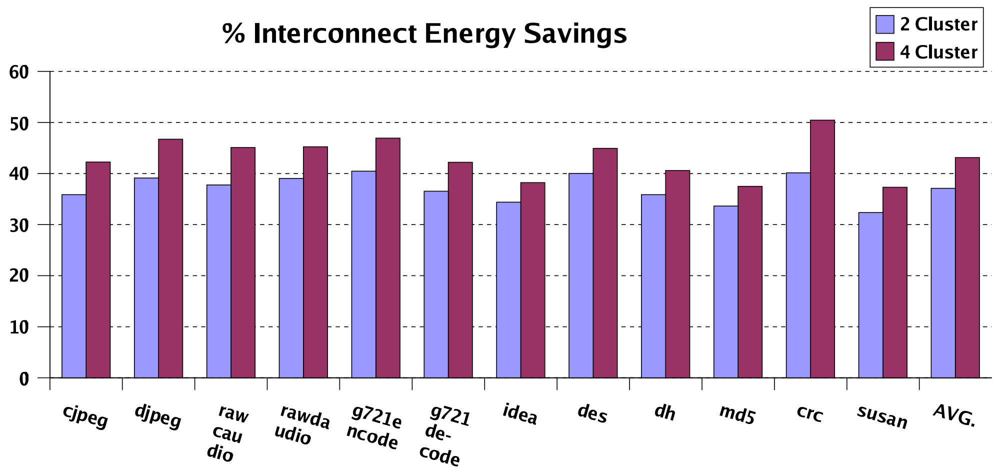


Figure 7.2: % Energy Benefit of LP conf. w.r.t. LL conf. for Algorithm 5

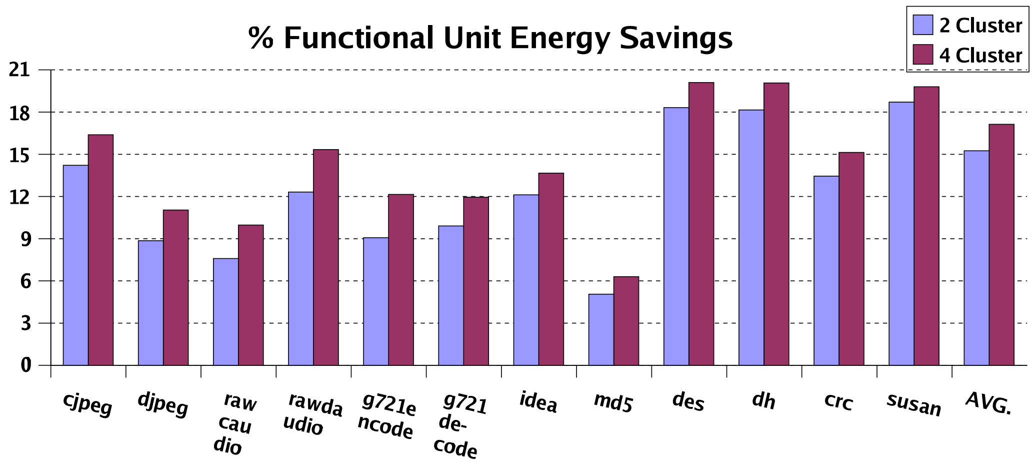


Figure 7.3: % Functional Unit Energy Benefits of Scheduling over H/W only scheme for Algorithm 5

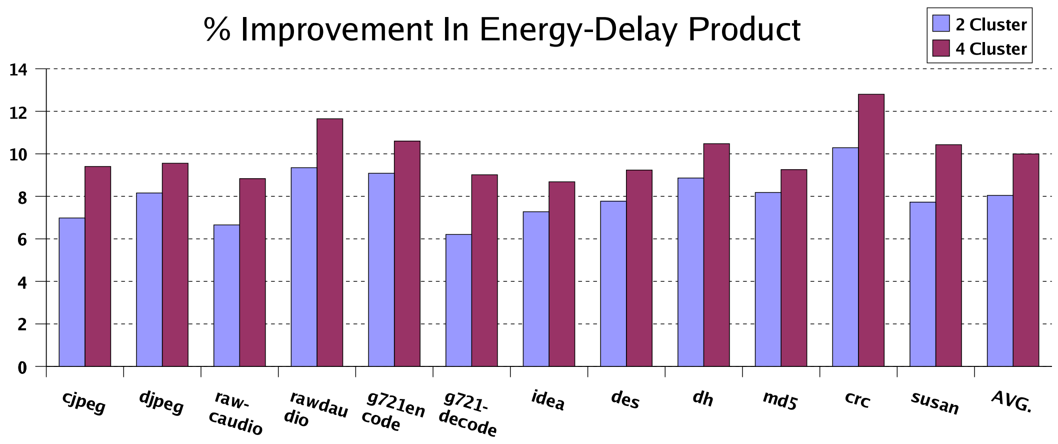


Figure 7.4: % Overall Benefit in EDP of Scheduling Algorithm 5 on LP conf. as compared to LL conf. and H/W only scheme for Fu Transitions

Algorithm 5 does not exploit instruction slack for saving communication energy explicitly and it gives priority to saving energy in functional units. However, there is still significant communication energy savings achieved by the combined Algorithm attributed to available communication slack. Figure 7.3 presents the percentage savings in functional unit energy of the combined Algorithm as compared to the hardware only scheme. The average saving in functional unit energy is 15.3% and 17.2% for 2-clustered and 4-clustered machine respectively. As expected the percentage savings are roughly the same as Algorithm 1 because the combined algorithm gives first priority to functional unit energy savings. The slight increase in functional unit energy for the combined scheduling algorithm is attributed to the increased execution time (due to usage of slow cross-path) that increases the idleness duration in the functional units.

Figure 7.4 gives the percentage savings in energy-delay product of processor by using the combined scheme conservatively assuming that functional units constitute 30% of processor energy and interconnect constitute 20% of processor energy (though the actual figure can vary from system to system and has strong dependence and circuit, design style and technology parameters). We observe that even with these conservative assumption, the overall energy-delay product of the processor is improved on an average by 8.1% and 10% for 2-clustered and 4-clustered architectures respectively which is a significant saving.

7.5 Conclusions

In this chapter, we have presented a combined energy-aware instruction scheduling algorithm that exploits instruction slack and communication slack together to simultaneously save energy in functional units and interconnect. A detailed experimental evaluation using trimaran framework confirms that proposed schemes are capable of providing significant energy savings thereby improving the usability of clustered architectures specifically in smaller technologies. The combined scheme obtains slightly better energy benefit in functional units and 37% and 43% energy benefit in interconnect with slightly higher performance degradation. Even with the conservative estimates of contribution of functional unit and interconnect to overall processor energy consumption, the proposed combined scheme obtains on an average 8% and 10% improvement in overall energy-delay product with 3.5% and 2% performance degradation for a 2-clustered and a 4-clustered machine respectively.

Chapter 8

Conclusions and Future Directions

8.1 Conclusions

The increasing ubiquity of embedded systems has opened up many new research issues. The design challenges posed by these systems are ostensibly different from those offered by general purpose systems due to their specific and conflicting requirements. Embedded systems can be characterized by factors such as, very high performance demand (to operate in real-time), low power consumption, low cost, less chip area, low temperature, and small time to market. The ever increasing trend towards miniaturization of devices makes utilizing huge transistor budget in a manner that enables high clock speed, low design complexity, and less energy consumption even more challenging[5]. However, resolving this challenge can enable the deployment of embedded systems for many performance-demanding never-before embedded applications at a lower cost.

Traditional ILP architectures (such as superscalar and VLIW) connect multiple pipelined functional units to a single unified register file in parallel to attain better performance. However, as the number of arithmetic units in a processor increases, register storage and communication between arithmetic units become critical factors dominating the area, cycle time, and power dissipation of the processor. Thus, the centralized monolithic architectures (both superscalar and VLIW) which use long wires for connecting spatially separated resources may not benefit from the advancements in the semiconductor technology specifically because of the increasing significance of wire delays in smaller technologies[5][6]. Another challenge posed by the technological advancement is the rising level of the leakage energy consumption in the logic[7]. With the 65nm and smaller technologies currently in fabrication, the leakage energy is on par with the dynamic energy consumption. In future technologies the leakage energy will further dominate the overall energy consumption[8].

A clustered VLIW architecture[11] solves the scalability problem associated with centralized VLIW architectures by having more than one register file and connecting only a subset of the functional units to a register file. Groups of small computation clusters can be interconnected using some interconnection topology and communication can be enabled using any of the various inter-cluster communication models[12]. Though clustering helps to combat the scalability problem by making components simpler and thereby increasing clock rate and reducing dynamic energy consumption of functional components, an interconnection network is required for the communication of data values among different components. This communication happens over long wires having high load capacitance which in effect takes more time and incurs more energy consumption[5][23]. Apart from the interconnects, functional units are another major source of energy consumption in clustered architectures. The contention for limited number of slow interconnects leads to many short idle cycles and that further increases the leakage energy consumption in functional units. The frequent accesses to functional units raises the temperature level and makes the leakage energy consumption which is specifically a concern in smaller technologies even worse. Clustered VLIW architectures rely on compile-time scheduling. The static scheduling simplifies the issue logic by alleviating the need for a dedicated hardware for scheduling. Thus, a significant fraction of the total energy consumption in clustered VLIW architectures is attributed to interconnects and functional units[24][25][26]. Thus, optimizing energy in interconnects and functional units in clustered architectures is becoming more and more important from one process generation to another.

In the past, study of leakage energy management at the architectural level has mostly focused on storage structure such as cache[31][32][33]. Relatively, little work has been done on architecture level leakage energy management in functional units[34][35] in the context of superscalar processors and energy efficient scheduling in the context of VLIW architectures[36][25][37]. To the best of our knowledge, there has been no such work in the past targeting clustered VLIW architectures specifically focusing on smaller technologies. In the absence of any high level model for interconnect energy estimation, the primary focus of research in the context of interconnects had been to reduce the latency of communication[42] and evaluation of various inter-cluster communication models[12]. Again to the best of our knowledge, we are not aware of any work that aims to reduce energy consumption in interconnects in clustered VLIW architectures.

However, functional units and interconnects are often underutilized in clustered VLIW architectures. Apart from other usual causes such as data dependencies, the under-utilization of functional units is also due to the contention for limited number of slow interconnect channels that introduce many short idle cycles in the operation of functional units. At the same time, since the functional units are distributed among clusters, there is also more contention for functional resources which leads to underutilization of interconnects. Finally, the contention for functional and interconnect resources in clustered VLIW architectures combine in a synergistic fashion and leads to greater available slack in clustered architectures as compared to VLIW architectures. The advancements in VLSI technology now also enable designing interconnects and functional units with different power and performance modes. It is possible to design wires consuming 1/5 the energy but having twice the delay[27]. Similarly the capabilities of dual-threshold domino logic with sleep mode can be utilized to do leakage energy management for short idle cycles in functional units[30][34]. In this thesis, we proposed a compiler-directed approach that leverages on these advancements in the VLSI technology to improve the usability of clustered VLIW architecture in smaller technologies, targeting the two major source of energy consumption, namely interconnects and functional units.

More specifically, we proposed a compiler directed leakage energy optimization technique in the context of VLIW/clustered VLIW architectures targeting the underutilized components such as functional units that reduce the average energy consumption of functional units by 15.1% and 16.9% in the context of a 2-clustered and a 4-clustered VLIW architectures respectively, with negligible performance degradation, on the top of a hardware-only scheme (already obtaining significant leakage energy benefits). Apart from obtaining benefit without performance loss the proposed technique also serves to reduce peak power and step power consumption that impacts the reliability of the chip[44]. We also evaluate the benefit of the proposed compiler scheduling algorithm in the context of instruction decoder, considering a split instruction decoder that enables the leakage energy optimization in instruction decoder. The average energy benefits in instruction decoder are 17.3% and 18.7% for a 2-clustered, and a 4-clustered VLIW architecture respectively.

In the context of interconnects, we propose a high level model of estimation of interconnect delay and energy (in contrast to low level circuit level models proposed earlier) that make it possible to do architectural and compiler optimization specifically targeting the interconnect. The proposed model fills the gap between the architect's need and circuit level models. The model takes architectural parameters such as length, bit-width, latency and target technology and provides a set of interconnect options with varying degree of area, pipelining, and power budget using pre-characterized estimates of circuit parameters for different interconnect components. We introduce the notion of heterogeneity in interconnects and propose a compiler directed technique for energy optimization in interconnect of clustered VLIW architectures. The interconnect energy optimization scheme (evaluated using the proposed model[46]) reduces the energy consumption of interconnects on an average by 41.5% and 46.8% for a 2-clustered and a 4-clustered machine respectively with 2% and 1.5% performance degradation[47]. We also propose an integrated scheme for simultaneous energy optimization in functional components as well as interconnects in a novel fashion and evaluate the energy benefits of the combined schemes[48]. The combined scheme obtains slightly better energy benefits in functional units and 37.1% and 43.1% energy benefit in interconnects with slightly higher performance degradation. Even with the conservative estimates of contribution of functional unit and interconnect to overall processor energy consumption, the proposed combined scheme obtains on an average 8% and 10% improvement in overall energy-delay product with 3.5% and 2% performance degradation for a 2-clustered and a 4-clustered machine respectively.

8.2 Future Directions

Our work is the first effort in the direction of improving the energy efficient of clustered VLIW architectures thereby improving their usability in future technologies. These are some of the ways in which this work can be extended :

1. We have focused on leakage energy savings in the context of clustered VLIW architectures. However, temperature is another important concern for handheld devices because of their small form factor. The leakage energy has a symbiotic and exponential dependence on temperature. Reduction in leakage energy reduces temperature which in turn reduces the leakage energy and vice versa. The proposed technique can also be evaluated to measure its temperature benefits. This will also give insights to extend the present technique and

develop new techniques that try to improve leakage as well as keep the temperature below the threshold (as dictated by the packaging of the chip).

2. The focus of our research has been to optimize energy in functional units and interconnects which represent two major sources of energy consumption in the absence of complicated scheduling hardware (as in superscalar processors). However, the other components of the clustered VLIW processors such as register file, fetch units etc can also be looked at from the point of view of energy savings. Specifically register file is one of the hot-spots along with the functional units. Techniques that target to keep the temperature of register file below the packaging-directed threshold will be specifically useful. We refer the reader to [170] for some of our preliminary results in this direction.

3. The interconnect energy model we have proposed is limited to design of the point to point interconnects which represent the most important and a major fraction of all on-chip interconnects. The proposed model can be extended to other kinds of interconnects such as bus based interconnects that may be of use in multi-core architectures.

Bibliography

[1] B. Ramakrishna Rau and Joseph A. Fisher. Instruction-level Parallel Processing: History, Overview, and Perspective. *Journal of Supercomputing*, pages 9–50, July 1993.

[2] William Johnson. *Superscalar Microprocessor Design.* Prentice-Hall, 1991.

[3] Joseph A. Fisher. Very Long Instruction Word Architectures and the ELI-512. In *25 years of the International Symposium on Computer Architecture (selected papers)*, pages 263–273. ACM Press, 1998.

[4] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. In *Proceedings of 24th Annual International Symposium on Computer architecture*, pages 206–218. ACM Press, 1997.

[5] D. Matzke. Will Physical Scalability Sabotage Performance Gains. *IEEE Computer*, September 1997.

[6] R. Ho, K Mai, and M. Horowitz. The Future of Wires. *Proceeding of IEEE*, April 2001.

[7] Trevor N. Mudge. Power: A First Class Design Constraint for Future Architecture and Automation. In *Proceedings of the 7th International Conference on High Performance Computing*, pages 215–224, London, UK, 2000. Springer-Verlag.

[8] Dennis Sylvester and Himanshu Kaul. Power-Driven Challenges in Nanometer Design. *IEEE Design and Test of Computers*, 18(6):12–22, 2001.

[9] Andrea Capitanio, Nikil Dutt, and Alexandru Nicolau. Partitioned Register Files for VLIWs: A Preliminary Analysis of Tradeoffs. In *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pages 292–300. IEEE Computer Society Press, 1992.

[10] Keith I. Farkas, Paul Chow, Norman P. Jouppi, and Zvonko Vranesic. The Multicluster Architecture: Reducing Cycle Time Through Partitioning. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 149–159. IEEE Computer Society, 1997.

[11] Paolo Faraboschi, Geoffrey Brown, Joseph A. Fisher, and Giuseppe Desoli. Clustered Instruction-level Parallel Processors. Technical report, Hewlett-Packard, 1998.

[12] Andrei Terechko, Erwan Le Thenaff, Manish Garg, Jos Van Eijndhoven, and Henk Corporaal. Inter-Cluster Communication Models for Clustered VLIW Processors. In *Proceedings of International Symposium on High-Performance Computer Architecture*, page 354, 2003.

[13] Texas Instruments Inc. TMS320C6000 CPU and Instruction Set reference Guide. `http://www.ti.com/sc/docs/products/dsp/c6000/index.htm`, 1998.

[14] Paolo Faraboschi, Geoffrey Brown, Joseph A. Fisher, Giuseppe Desoli, and Fred Homewood. Lx: A Technology Platform for Customizable VLIW Embedded Processing. In *Proceedings of 27th Annual International Symposium on Computer architecture*, pages 203–213, 2000.

[15] J. Fridman and Zvi Greefield. The TigerSHARC DSP Architecture. *IEEE Micro*, pages 66–76, 2000.

[16] G. G. Pechanek and S. Vassiliadis. The ManArray Embedded Processor Architecture. In *Proceedings of Euromicro Conference*, pages 348–355, 2000.

[17] J. Derby and J. Moreno. A High-performance Embedded DSP Core with Novel SIMD Features. In *Proceedings of 2003 International Conference on Acoustics, Speech, and Signal Processing*, 2003.

[18] J.M. Parcerisa R. Canal and A. Gonzalez. Dynamic Cluster Assignment Mechanisms. In *Proceedings of Sixth IEEE International Symposium on High Performance Computer Architecture*, 2000.

[19] Gurindar S. Sohi, Scott E. Breach, and T. N. Vijaykumar. Multiscalar Processors. In *25 Years ISCA: Retrospectives and Reprints*, pages 521–532, 1998.

[20] Ujval Kapasi, William J. Dally, Scott Rixner, John D. Owens, and Brucek Khailany. The Imagine Stream Processor. In *Proceedings of 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02)*, page 282, Washington, DC, USA, 2002. IEEE Computer Society.

[21] Pedro Marcuello and Antonio Gonzalez. Clustered Speculative Multithreaded Processors. In *ICS '99: Proceedings of 13th International Conference on Supercomputing*, pages 365–372, New York, NY, USA, 1999. ACM Press.

[22] James E. Smith. Instruction-Level Distributed Processing. *Computer*, 34(4):59–65, 2001.

[23] R. Ho, K Mai, and M. Horowitz. The Future of Wires. *Proceedings of IEEE*, 89(4):490–504, 2001.

[24] Hangsheng Wang, Li-Shiuan Peh, and Sharad Malik. Power-driven Design of Router Microarchitectures in On-chip Networks. In *Proceedings of Symposium on Microarchitecture*, page 105, 2003.

[25] H. S. Kim, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Adapting Instruction Level Parallelism for Optimizing Leakage in VLIW Architectures. In *Proceedings of Conference on Language, Compiler, and Tool for Embedded Systems*, pages 275–283, 2003.

[26] J. Adam Butts and Gurindar S. Sohi. A Static Power Model for Architects. In *MICRO 33: Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pages 191–201, New York, NY, USA, 2000. ACM Press.

[27] Kaustav Banerjee and Amit Mehrotra. A Power-Optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs. In *Proceedings of IEEE Transactions on Electron Devices*, pages 2001–2007, November 2002.

[28] Man Lung Mui, Kaustav Banerjee, and Amit Mehrotra. A Global Interconnect Optimization Scheme for Nanometer Scale VLSI with Implications for Latency, Bandwidth and Power Dissipation. In *IEEE Transactions on Electron Devices*, pages 195–203, 2004.

[29] Rajeev Balasubramonian, Naveen Muralimanohar, Karthik Ramani, and Venkatanand Venkatachalapathy. Microarchitectural Wire Management for Performance and Power

in Partitioned Architectures. In *Proceedings of International Symposium on High-Performance Computer Architecture*, pages 28–39, 2005.

[30] Volkan Kursun and Eby G. Friedman. Low swing Dual Threshold Voltage Domino Logic. In *GLSVLSI '02: Proceedings of the 12th ACM Great Lakes Symposium on VLSI*, pages 47–52, New York, NY, USA, 2002. ACM Press.

[31] Se-Hyun Yang, Babak Falsafi, Michael D. Powell, Kaushik Roy, and T. N. Vijaykumar. An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, page 147, Washington, DC, USA, 2001. IEEE Computer Society.

[32] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Cache decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proceedings of the 28th Annual International Symposium on Computer architecture*, pages 240–251, New York, NY, USA, 2001. ACM Press.

[33] Krisztin Flautner, Steve Reinhardt, and Trevor Mudge. Automatic Performance Setting for Dynamic Voltage Scaling. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 260–271, New York, NY, USA, 2001. ACM Press.

[34] Steven Dropsho, Volkan Kursun, David H. Albonesi, Sandhya Dwarkadas, and Eby G. Friedman. Managing Static Leakage Energy in Microprocessor Functional Units. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 321–332, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[35] Siddharth Rele, Santosh Pande, Soner Onder, and Rajiv Gupta. Optimizing Static Power Dissipation by Functional Units in Superscalar Processors. In *Proceedings of 11th International Conference on Compiler Construction*, pages 261–275, 2002.

[36] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, D. Duarte, and Y-F. Tsai. Exploiting VLIW Schedule Slacks for Dynamic and Leakage Energy Reduction. In *Proceedings of International Symposium on Microarchitecture*, pages 102–113, 2001.

[37] H.S Yun and J. Kim. Power-aware Modulo Scheduling for High-Performance VLIW Processors. In *Proceedings of 2001 International Symposium on Low Power Electronics and Design*, pages 40–45. ACM Press, 2001.

[38] Taku Uchino and Jason Cong. An Interconnect Energy Model Considering Coupling Effects. In *Proceedings of the Conference on Design automation*, pages 555–558, 2001.

[39] Pallav Gupta, Lin Zhong, and Niraj K. Jha. A High-level Interconnect Power Model for Design Space Exploration. In *Proceedings of the International Conference on Computer-aided design*, page 551, 2003.

[40] S. Wilton and N. Jouppi. CACTI: An Enhanced Cache Access and Cycle Time Model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.

[41] J. H. Tseng and K. Asanovic. Energy-Efficient Register Access. In *Proceedings of the Symposium on Integrated circuits and systems design*, page 377, 2000.

[42] Antonio Gonzalez Joan-Manuel Parcerisa, Julio Sahuquillo and Jos Duato. Efficient Interconnects for Clustered Microarchitectures. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, pages 291–300, 2002.

[43] Krishnan Kailas, Ashok Agrawala, and Kemal Ebcioglu. CARS: A New Code Generation Framework for Clustered ILP Processors. In *Proceedings of 7th International Symposium on High-Performance Computer Architecture*, page 133, 2001.

[44] Rahul Nagpal and Y. N. Srikant. Compiler-assisted Leakage Energy Optimization for Clustered VLIW Architectures. In *Proceedings of the International Conference on Embedded Software*, pages 233–241, 2006.

[45] Rahul Nagpal and Y. N. Srikant. Compiler-Assisted Instruction Decoder Energy Optimization for Clustered VLIW Architectures. In *Proceedings of the International Conference on High Performance Computing*, pages 405–417, 2007.

[46] Rahul Nagpal, Arvind Madan, Amrutur Bhardwaj, and Y. N. Srikant. INTACTE: An Interconnect Area, Delay, and Energy Estimation Tool for Microarchitectural Explorations. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 238–247, 2007.

[47] Rahul Nagpal and Y. N. Srikant. Exploring Energy-Performance Trade-Offs for Heterogeneous Interconnect Clustered VLIW Processors. In *Proceedings of the International Conference on High Performance Computing*, pages 497–508, 2006.

[48] Rahul Nagpal and Y. N. Srikant. Compiler-Assisted Energy Optimization for Clustered VLIW Processors. Technical Report, Dept. of CSA, Indian Institute of Science (http://www.archive.csa.iisc.ernet.in/TR), 2008.

[49] J. Ebergen, J. Gainsley, , and P. Cunningham. Transistor Sizing: How to Control the Speed and Energy Consumption of a Circuit. *Proceedings of 10th International Symposium on Asynchronous Circuits and Systems*, pages 51–61, 2004.

[50] E. Kursun, S. Ghiasi, and M. Sarrafzadeh. Transistor Level Budgeting for Power Optimization. In *Proceedings of the 5th International Symposium on Quality Electronic Design*, pages 116–121, Washington, DC, USA, 2004. IEEE Computer Society.

[51] Anup Kumar Sultania, Dennis Sylvester, and Sachin S. Sapatnekar. Transistor and Pin Reordering for Gate Oxide Leakage Reduction in Dual Tox Circuits. In *Proceedings of the IEEE International Conference on Computer Design*, pages 228–233, Washington, DC, USA, 2004. IEEE Computer Society.

[52] Rob A. Rutenbar, L. Richard Carley, Roberto Zafalon, and Nicola Dragone. Low-power Technology Mapping for Mixed-Swing Logic. In *ISLPED '01: Proceedings of the 2001 International Symposium on Low power electronics and design*, pages 291–294, New York, NY, USA, 2001. ACM.

[53] J. Madsen, Mahadevan, S., and K Virk. *Interconnect Centric Design for Advanced SoC and NoC.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004.

[54] A. G. M. Strollo, E. Napoli, and D. De Caro. New Clock-Gating Techniques for Low-Power Flip-Flops. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pages 114–119, New York, NY, USA, 2000. ACM Press.

[55] S. Kong B., Kim and Y. Jun. Conditional-Capture Flip-Flop for Statistical Power Reduction. *IEEE Journal on Solid State Circuits*, 36(8):1263–1271, 2001.

[56] Benton H. Calhoun, Frank A. Honore, and Anantha Chandrakasan. Design Methodology for Fine-Grained Leakage Control in MTCMOS. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pages 104–109, New York, NY, USA, 2003. ACM Press.

[57] Hyo-Sig Won, Kyo-Sun Kim, Kwang-Ok Jeong, Ki-Tae Park, Kyu-Myung Choi, and Jeong-Taek Kong. An MTCMOS Design Methodology and its Application to Mobile Computing. In *ISLPED '03: Proceedings of the 2003 International Symposium on Low power electronics and design*, pages 110–115, New York, NY, USA, 2003. ACM.

[58] Michael Liu, Wei-Shen Wang, and Michael Orshansky. Leakage power Reduction by Dual-vth Designs Under Probabilistic Analysis of Vth Variation. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pages 2–7, New York, NY, USA, 2004. ACM Press.

[59] R. Krane, J. Parsons, and A. Bar-Cohen. Design of a Candidate Thermal Control System for a Cryogenically Cooled Computer. *IEEE Transaction on Components, Packaging, and Manufacturing Technology*, 11(4):545–556, 1998.

[60] Clark N. Taylor, Sujit Dey, and Yi Zhao. Modeling and Minimization of Interconnect Energy Dissipation in Nanometer Technologies. In *Proceedings of the 38th Conference on Design automation*, pages 754–757, New York, NY, USA, 2001. ACM Press.

[61] Hui Zhang and Jan Rabaey. Low-swing interconnect interface circuits. In *Proceedings of the 1998 International Symposium on Low power electronics and design*, pages 161–166, New York, NY, USA, 1998. ACM Press.

[62] W.-B. Jone, J. S. Wang, Hsueh-I Lu, I. P. Hsu, and J.-Y. Chen. Design Theory and Implementation for Low-Power Segmented Bus Systems. *ACM Transaction on Design and Automation of Electronic Systems*, 8(1):38–54, 2003.

[63] William J. Dally and Brian Towles. Route Packets, Not Wires: On-Chip Inteconnection Networks. In *Proceedings of the 38th Conference on Design automation*, pages 684–689, New York, NY, USA, 2001. ACM Press.

[64] Kanad Ghose and Milind B. Kamble. Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation. In *Proceedings of*

the 1999 International Symposium on Low power electronics and design, pages 70–75, New York, NY, USA, 1999. ACM Press.

[65] V. De La Luz, M. Kandemir, and I. Kolcu. Automatic Data Migration for Reducing Energy Consumption in Multi-Bank Memory Systems. In *Proceedings of the 39th Conference on Design automation*, pages 213–218, New York, NY, USA, 2002. ACM Press.

[66] Johnson Kin, Manish Gupta, and William H. Mangione-Smith. The Filter Cache: An Energy Efficient Memory Structure. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 184–193, Washington, DC, USA, 1997. IEEE Computer Society.

[67] Preeti Ranjan Panda, Nikil D. Dutt, and Alexandru Nicolau. On-chip vs. Off-Chip Memory: The Data Partitioning Problem in Embedded Processor-Based Systems. *ACM Transaction on Design and Automation of Electronic Systems*, 5(3):682–704, 2000.

[68] J. S. Hu, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir. Using Dynamic Branch Behavior for Power-Efficient Instruction Fetch. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03)*, page 127, Washington, DC, USA, 2003. IEEE Computer Society.

[69] Michael Powell, Se-Hyun Yang, Babak Falsafi, Kaushik Roy, and T. N. Vijaykumar. Reducing Leakage in a High-Performance Deep-Submicron Instruction Cache. *ACM Transaction on Design and Automation of Electronic Systems*, 9(1):77–90, 2001.

[70] Nam Sung Kim, Krisztin Flautner, David Blaauw, and Trevor Mudge. Drowsy Instruction Caches: Leakage Power Reduction Using Dynamic Voltage Scaling and Cache Sub-Bank Prediction. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 219–230, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[71] J. S. Hu, A. Nadgir, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir. Exploiting Program Hotspots and Code Sequentiality for Instruction Cache Leakage Management. In *Proceedings of the 2003 International Symposium on Low power electronics and design*, pages 402–407, New York, NY, USA, 2003. ACM Press.

[72] Mohan G. Kabadi, Natarajan Kannan, Palanidaran Chidambaram, Suriya Narayanan, M. Subramanian, and Ranjani Parthasarathi. Dead-Block Elimination in Cache: A Mechanism to Reduce I-cache Power Consumption in High Performance Microprocessors. In *Proceedings of the 9th International Conference on High Performance Computing*, pages 79–88, London, UK, 2002. Springer-Verlag.

[73] Alper Buyuktosunoglu, Stanley Schuster, David Brooks, Pradip Bose, Peter W. Cook, and David H. Albonesi. An Adaptive Issue Queue for Reduced Power at High Performance. In *Proceedings of the First International Workshop on Power-Aware Computer Systems-Revised Papers*, pages 25–39, London, UK, 2001. Springer-Verlag.

[74] R. Iris Bahar and Srilatha Manne. Power and Energy Reduction via Pipeline Balancing. In *Proceedings of the 28th Annual International Symposium on Computer architecture*, pages 218–229, New York, NY, USA, 2001. ACM Press.

[75] Daniele Folegnani and Antonio Gonzalez. Energy-Effective Issue Logic. In *Proceedings of the 28th Annual International Symposium on Computer architecture*, pages 230–239, New York, NY, USA, 2001. ACM Press.

[76] Thomas L. Martin and Daniel P. Siewiorek. Nonideal Battery and Main Memory Effects on CPU Speed-Setting for Low Power. *IEEE Transaction Very Large Scale Integr. Syst.*, 9(1):29–34, 2001.

[77] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *1st USENIX Symposium on Operating Systems Design and Implementation*, pages 13–23, New York, NY, USA, 1994. ACM Press.

[78] Frank Bellosa. The Benefits of Event Driven Energy Accounting in Power-Sensitive Systems. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 37–42, New York, NY, USA, 2000. ACM Press.

[79] Wanghong Yuan and Klara Nahrstedt. Energy-efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of the nineteenth ACM Symposium on Operating systems principles*, pages 149–163, New York, NY, USA, 2003. ACM Press.

[80] Jacob R. Lorch and Alan Jay Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *Proceedings of the 2001 ACM SIGMETRICS International Conference*

on Measurement and Modeling of Computer Systems, pages 50–61, New York, NY, USA, 2001. ACM Press.

[81] Dongkun Shin, Jihong Kim, and Seongsoo Lee. Low-energy Intra-Task Voltage Scheduling Using Static Timing Analysis. In *Proceedings of the 38th Conference on Design automation*, pages 438–443, New York, NY, USA, 2001. ACM Press.

[82] Chung-Hsing Hsu and Ulrich Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming language design and implementation*, pages 38–48, New York, NY, USA, 2003. ACM Press.

[83] Greg Semeraro, Grigorios Magkils, Rajeev Balasubramonian, David H Albonesi, Sandhya Dwarakadas and Michael L Scott. Energy-Efficient Processor Design using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *International Symposium on High Performance Computer Architecture*, 2002.

[84] Greg Semeraro, David H. Albonesi, Steven G. Dropsho, Grigorios Magklis, Sandhya Dwarkadas, and Michael L. Scott. Dynamic Frequency and Voltage Control for a Multiple Clock Domain Microarchitecture. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 356–367, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[85] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W. Clark. Voltage and Frequency Control With Adaptive Reaction Time in Multiple-Clock-Domain Processors. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 178–189, Washington, DC, USA, 2005. IEEE Computer Society.

[86] Grigorios Magklis, Michael L. Scott, Greg Semeraro, David H. Albonesi, and Steven Dropsho. Profile-based Dynamic Voltage and Frequency Scaling for a Multiple Clock Domain Microprocessor. In *Proceedings of the 30th Annual International Symposium on Computer architecture*, pages 14–27, New York, NY, USA, 2003. ACM Press.

[87] A. B. Dalton and C. S. Ellis. Sensing User Intention and Context for Energy Management. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems*, pages 151–156, 2003.

[88] Jason Flinn and M. Satyanarayanan. Energy-Aware Adaptation for Mobile Applications. In *Proceedings of the seventeenth ACM Symposium on Operating systems principles*, pages 48–63, New York, NY, USA, 1999. ACM Press.

[89] Xiaodong Li, Zhenmin Li, Francis David, Pin Zhou, Yuanyuan Zhou, Sarita Adve, and Sanjeev Kumar. Performance Directed Energy Management for Main Memory and Disks. In *Proceedings of the 11th International Conference on Architectural support for programming languages and operating systems*, pages 271–283, New York, NY, USA, 2004. ACM Press.

[90] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proceedings of the 30th Annual International Symposium on Computer architecture*, pages 169–181, New York, NY, USA, 2003. ACM Press.

[91] A. E. Papathanasiou and M. L. Scott. Increasing Disk Burstiness for Energy Efficiency. Technical report, University of Rochester, Rochester, NY, USA, 2002.

[92] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. Influence of Compiler Optimizations on System Power. In *DAC '00: Proceedings of the 37th Conference on Design automation*, pages 304–307, New York, NY, USA, 2000. ACM Press.

[93] M. Valluri and L. John. Is Compiling for Performance == Compiling for Power. In *The 5th Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT-5)*, January 2001.

[94] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Compiler-Directed Instruction Cache Leakage Optimization. In *MICRO 35: Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 208–218, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[95] W. Zhang, M. Karakoy, M. Kandemir, and G. Chen. A compiler Approach for Reducing Data Cache Energy. In *ICS '03: Proceedings of the 17th Annual International Conference on Supercomputing*, pages 76–85, New York, NY, USA, 2003. ACM Press.

[96] Chingren Lee, Jenq Kuen Lee, Tingting Hwang, and Shi-Chun Tsai. Compiler optimization on VLIW instruction scheduling for low power. *ACM Transaction on Design and Automation of Electronic Systems*, 8(2):252–268, 2003.

[97] Jeffrey Palm, Han Lee, Amer Diwan, and J. Eliot B. Moss. When to Use a Compilation Service? In *LCTES/SCOPES '02: Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems*, pages 194–203, New York, NY, USA, 2002. ACM Press.

[98] G. Chen, B. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli. Energy-Aware Compilation and Execution in Java-Enabled Mobile Devices. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 34.1, Washington, DC, USA, 2003. IEEE Computer Society.

[99] T. K. Tan, A. Raghunathan, and N. K. Jha. Software Architectural Transformations: A New Approach to Low Energy Embedded Software. In *DATE '03: Proceedings of the Conference on Design, Automation and Test in Europe*, page 11046, Washington, DC, USA, 2003. IEEE Computer Society.

[100] Adve S. Sachs D. and D. Jones. Cross-Layer Adaptive Video Coding to Reduce Energy on General Purpose Processors. In *Proceedings of the International Conference on Image Processing*, pages 109–112, 2003.

[101] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource. In *ASPLOS-X: Proceedings of the 10th International Conference on Architectural support for programming languages and operating systems*, pages 123–132, New York, NY, USA, 2002. ACM Press.

[102] Gupta R. Pereira C. and Srivastava M. PASA: A Software Architecture for Building Power Aware Embedded Systems. In *IEEE CAS Workshop on Wireless Communication and Networking*. IEEE, 2002.

[103] Heath T., Pinheiro E., Hom J., Kremer U., and Bianchini R. Code Transformations for Energy-Efficient Device Management. *IEEE Transaction on Computers*, 53(8), 2004.

[104] Shivajit Mohapatra, Radu Cornea, Nikil Dutt, Alex Nicolau, and Nalini Venkatasubramanian. Integrated Power Management for Video Streaming to Mobile Handheld Devices.

In *MULTIMEDIA '03: Proceedings of the Eleventh ACM International Conference on Multimedia*, pages 582–591, New York, NY, USA, 2003. ACM Press.

[105] Wanghong Yuan and Klara Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 149–163, New York, NY, USA, 2003. ACM Press.

[106] Yunsi Fei, Lin Zhong, and Niraj K. Jha. An Energy-Aware Framework for Coordinated Dynamic Software Management in Mobile Computers. In *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, pages 306–317, Washington, DC, USA, 2004. IEEE Computer Society.

[107] Nevine AbouGhazaleh, Bruce Childers, Daniel Mosse, Rami Melhem, and Matthew Craven. Energy Management for Real-Time Embedded Applications with Compiler Support. In *Proceedings of the 2003 ACM SIGPLAN Conference on Language, compiler, and tool for embedded systems*, pages 284–293, New York, NY, USA, 2003. ACM Press.

[108] Carmean D. Gunther S., Binns F. and Hall J. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technical Journal*, 2001.

[109] Gochman S., Ronen R., Anati I., Berkovits A., Kurts T., Naveh A., Saeed A., Sperber Z., and Valentine R. The Intel Pentium M processor: Microarchitecture and performance. *Intel Technical Journal*, 7(2):21–59, 2003.

[110] Crusoe Processor Product Brief: Model TM5800, 2003.

[111] B. Brock and K. Rajamani. Dynamic Power Management for Embedded Systems. In *In Proceedings of the IEEE International SOC Conference*, pages 416–419. IEEE, 2003.

[112] Sandeep Dhar, Dragan Maksimovic, and Bruno Kranzen. Closed-Loop Adaptive Voltage Scaling Controller for Standard-cell ASICs. In *ISLPED '02: Proceedings of the 2002 International Symposium on Low power electronics and design*, pages 103–107, New York, NY, USA, 2002. ACM Press.

[113] Kanishka Lahiri, Sujit Dey, Debashis Panigrahi, and Anand Raghunathan. Battery-Driven

System Design: A New Frontier in Low Power Design. In *Proceedings of the 2002 Conference on Asia South Pacific design automation/VLSI Design*, page 261, Washington, DC, USA, 2002. IEEE Computer Society.

[114] C. Dyer. Fuel Cells and Portable Electronics. In *Symposium On VLSI Circuits Digest of Technical Papers*, pages 124–127, 2004.

[115] Epstein A. Millimeter-scale, Micro-Electromechanical Systems Gas Turbine Engines. *J. Eng. Gas Turb. Power*, 126:205–226, 2003.

[116] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. In *ICCAD '94: Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided design*, pages 384–390, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[117] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction Level Power Analysis and Optimization of Software. In *VLSID '96: Proceedings of the 9th International Conference on VLSI Design: VLSI in Mobile Communication*, page 326, Washington, DC, USA, 1996. IEEE Computer Society.

[118] Mike Tien-Chien Lee, Masahiro Fujita, Vivek Tiwari, and Sharad Malik. Power Analysis and Minimization Techniques for Embedded DSP Software. *IEEE Transaction on Very Large Scale Integration Systems*, 5(1):123–135, 1997.

[119] Amit Sinha and Anantha P. Chandrakasan. JouleTrack: A Web Based Tool for Software Energy Profiling. In *DAC '01: Proceedings of the 38th Conference on Design automation*, pages 220–225, New York, NY, USA, 2001. ACM Press.

[120] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel. An Accurate and Fine Grain Instruction-Level Energy Model supporting Software Optimizations. In *In International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS)*, 2001.

[121] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *ISCA '00: Proceedings of the 27th Annual International Symposium on Computer architecture*, pages 83–94, New York, NY, USA, 2000. ACM Press.

[122] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The Design and use of Simplepower: A Cycle-Accurate Energy Estimation Tool. In *DAC '00: Proceedings of the 37th Conference on Design automation*, pages 340–345, New York, NY, USA, 2000. ACM Press.

[123] George Z. N. Cai and Chee How Lim. Architectural Level Power/Performance Optimization and Dynamic Power Estimation. In *Cool Chips Tutorial in Conjunction with the 32nd International Symposium on Microarchitecture*, 1999.

[124] Jason Flinn and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[125] J. Eyre and J. Bier. The Evolution of DSP Processors. Technical Report, Berkeley Design Technology,, 2000.

[126] Scott Rixner, William J. Dally, Brucek Khailany, Peter Mattson, Ujval J. Kapasi, and John D. Owens. Register Organization for Media Processing. *Proceedings of International Symposium on High Performance Computer Architecture*, pages 375–386, 2000.

[127] Manoj Franklin and Gurindar S. Sohi. Register Traffic Analysis for Streamlining Inter-Operation Communication in Fine-Grain Parallel Processors. *SIGMICRO Newsl.*, 23(1-2):236–245, 1992.

[128] N. Seshan. High VelociTI Processing. *IEEE Signal Processing Magazine*, March 1998.

[129] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal. Baring it all to software: Raw machines. *Computer*, 30(9):86–93, 1997.

[130] Alex Aleta, Josep M. Codina, Jesus Sanchez, and Antonio Gonzalez. Graph-partitioning based Instruction Scheduling for Clustered Processors. In *Proceedings of International Symposium on Microarchitecture*, pages 150–159, 2001.

[131] Giuseppe Desoli. Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach. Technical Report, Hewlett-Packard, 1998.

[132] Viktor S. Lapinskii, Margarida F. Jacome, and Gustavo A. De Veciana. Cluster Assignment for High-performance Embedded VLIW processors. *ACM Transaction on Design and Automation of Electronic Systems*, 7(3):430–454, 2002.

[133] Emre Ozer, Sanjeev Banerjia, and Thomas M. Conte. Unified Assign and Schedule: A New Approach to Scheduling for Clustered Register File Microarchitectures. In *Proceedings of International Symposium on Microarchitecture*, pages 308–315, 1998.

[134] Rahul Nagpal and Y. N. Srikant. Integrated Temporal and Spatial Scheduling for Extended Operand Clustered VLIW Processors. In *Proceedings of International Conference on Computing Frontiers*, pages 457–470, 2004.

[135] Rahul Nagpal and Y. N. Srikant. Pragmatic Integrated Scheduling for Clustered VLIW Processors. *Software Practice and Experience*, 38(3):227–257, 2008.

[136] Smotherman M., Krishnamurthy S., Aravid P., and Hunnicutt D. Efficient DAG Construction and Heuristic Calculation for Instruction Scheduling. In *Proceedings of the 24th Annual International Symposium on Microarchitecture*, pages 93–102. ACM Press, 1991.

[137] Trimaran System. `http://www.trimaran.org/`.

[138] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. *International Symposium on Microarchitecture*, 1997.

[139] MediaBench . `http://cares.icsl.ucla.edu/MediaBench/`.

[140] B. Mangione-Smith Gokhan Memic and W. Hu. NetBench: A Benchmarking Suit for Network Processor. *CARES Technical Report*, 2002.

[141] NetBench. `http://cares.icsl.ucla.edu/NetBench/`.

[142] Jeffrey Ringenberg Matthew Guthaus and Dan Ernst. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. *IEEE 4th Annual Workshop on Workload Characterization*, 2001.

[143] MiBench. http://www.eecs.umich.edu/mibench/.

[144] Shekhar Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, 19(4):23–29, 1999.

[145] Santosh G. Abraham, Waleed M. Meleis, and Ivan D. Baev. Efficient Backtracking Instruction Schedulers. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, pages 301–308, 2000.

[146] Srilatha Manne, Artur Klauser, and Dirk Grunwald. Pipeline Gating: Speculation Control for Energy Reduction. In *Proceedings of the International Symposium on Computer architecture*, pages 132–141, Washington, DC, USA, 1998. IEEE Computer Society.

[147] Keith D. Cooper and Todd Waterman. Understanding Energy Consumption on the C62x. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, 2002.

[148] Wu-An Kuo, Tingting Hwang, and Allen C.-H. Wu. Decomposition of Instruction Decoders for Low-power Designs. *ACM Transaction on Design and Automation of Electronic Systems*, 11(4), 2006.

[149] Michael Chu, Kevin Fan, and Scott Mahlke. Region-based Hierarchical Operation Partitioning for Multicluster Processors. *SIGPLAN Notices*, pages 300–311, 2003.

[150] Rainer Leupers. Instruction scheduling for clustered VLIW DSPs. In *PACT '00: Proceedings of 2000 International Conference on Parallel Architectures and Compilation Techniques*, page 291, Washington, DC, USA, 2000. IEEE Computer Society.

[151] J. M. Parcerisa R. Canal and A. Gonzalez. Dynamic Cluster Assignment Mechanisms. In *Proceedings of Sixth IEEE International Symposium on High Performance Computer Architecture*, 2000.

[152] Jung Ho Ahn, William J. Dally, Brucek Khailany, Ujval J. Kapasi, and Abhishek Das. Evaluating the Imagine Stream Architecture. In *Proceedings of International Symposium on Computer architecture*, page 14, 2004.

[153] ARM MPCore. http://www.arm.com.

[154] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. POWER5 System Microarchitecture. *IBM Journal of Research and Development*, 49(4/5):505–521, 2005.

[155] Christopher G.and Matthew Eastwood, Jie Wu, and Addison Snell. Intel Brings Dual-Core Capabilities to Itanium 2 with Montecito Processor. *Manufacturing Insights*, July 2006.

[156] Rajeev Balasubramonian, Naveen Muralimanohar, Karthik Ramani, and Venkatanand Venkatachalapathy. Microarchitectural Wire Management for Performance and Power in Partitioned Architectures. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, pages 28–39, 2005.

[157] Rakesh Kumar, Victor Zyuban, and Dean M. Tullsen. Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads and Scaling. In *Proceedings of the International Symposium on Computer Architecture*, pages 408–419, 2005.

[158] William J. Dally and Brian Towles. Route Packets, Not Wires: On-Chip Inteconnection Networks. In *Proceedings of the Conference on Design automation*, pages 684–689, 2001.

[159] HSPICE. `http://www.synopsys.com/products/hspice.html`.

[160] Predictive Technology Model. `http://www.eas.asu.edu/~ptm/`.

[161] MATLAB. `http://www.mathworks.com/products/matlab/`.

[162] Chung Hsing Hsu and Ulrich Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *Proceedings of Conference on Programming language design and implementation*, pages 38–48, 2003.

[163] Alex Aleta, Josep M. Codina, Antonio Gonzalez, and David Kaeli. Heterogeneous Clustered VLIW Microarchitectures. In *Proceedings of International Symposium on Code Generation and Optimization*, March 2007.

[164] International Technology Roadmap for Semiconductors. `http://www.itrs.net/`.

[165] BSIM4.6.0. `http://www-device.eecs.berkeley.edu/~bsim3/bsim4.html`.

[166] T. N. Theis. The Future of Interconnection Technology. *IBM Journal of Research and Development*, 44(3), 2000.

[167] J. D. Warnock, J. M. Keaty, J. G. Clabes J. Petrovick, C. J. Kircher, B. L. Krauter, P. J. Restle, B. A. Zoric, and C. J. Anderson. The Circuit and Physical Design of the POWER4 Microprocessor. *IBM Journal of Research and Development*, 46(1), 2002.

[168] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and Sharad Malik. Orion: A Power-Performance Simulator for Interconnection Networks. In *Proceedings of the International Symposium on Microarchitecture*, pages 294–305, 2002.

[169] Nir Magen, Avinoam Kolodny, Uri Weiser, and Nachum Shamir. Interconnect-power Dissipation in a Microprocessor. In *Proceedings of International workshop on System Level Interconnect Prediction*, pages 7–13, 2004.

[170] Rahul Nagpal and Y. N. Srikant. Register File Energy Optimization for Snooping Based Clustered VLIW Architectures. In *Proceedings of the International Symposium on Computer Architecture and High Performance Computing*, pages 161–168, 2007.