Hyperblock Formation: A Power/Energy Perspective for High Performance VLIW Architectures

Giuseppe Ascia, Vincenzo Catania, Maurizio Palesi and Davide Patti
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania, Italy
{gascia,vcatania,mpalesi,dpatti}@diit.unict.it

***Abstract*— Architectures based on Very Long Instruction Word (VLIW) processors are an optimal choice in the attempt to obtain high performance levels in mobile devices. The effectiveness of these processors depends on the ability of compilers to provide sufficient instruction-level parallelism (ILP) in program codes. The main factor limiting the possibility of obtaining high ILP levels is the presence of conditional branches, which prevent a VLIW compiler from scheduling instructions belonging to different paths in parallel. Hyperblock formation is the main compiling technique to solve this limit affecting ILP, transforming the code in such a way as to eliminate conditional branches. The paper presents an analysis of the effect of this technique, not only from the well-known perspective of performance gain but from that of power dissipation and energy consumption. The effect of hyperblock formation on these magnitudes is presented for a set of typical embedded multimedia applications, introducing the non-trivial problems this aggressive ILP technique causes in the increasingly widespread scenario of multi-objective performance, energy and power optimization.**

I. Introduction

The spread of systems based on Very Long Instruction Word (VLIW) processors greatly depends on the flexibility these architectures exhibit in applications affected by constraints on performance, energy and power consumption. On the one hand, like superscalar architectures, they meet performance requirements by offering the possibility of executing several instructions per cycle in parallel. On the other, unlike superscalar architectures, the hardware is considerably simplified as it does not have to deal with the scheduling of parallel instructions but confines itself to a *plan of execution* which is statically pre-established in the compilation phase. Simplified hardware means shorter critical paths and greater clock frequencies but also, and above all, less energy consumption and power dissipation.

This shift of responsibility for scheduling from the control hardware to the compiler requires greater awareness in using the compiler than is required in classical architectures. A wrong decision in the compiling stage may be equivalent to choosing an inappropriate hardware configuration for execution of a given application.

If we consider the original objective of a VLIW architecture, i.e. maximization of performance, it is clear that it is closely connected with the capacity of the compiler to schedule in parallel as many instructions as possible, that is, to obtain a high degree of *instruction level parallelism* (ILP). Given the principle on which an architecture based on a VLIW processor works, the number and type of functional units affects the way in which the compiler can schedule the operations in each long instruction. The presence of several instances of a certain functional unit, for example, makes it possible to schedule several operations using that type of unit in the same clock cycle. Tests performed show, however, that even when the number of functional units is increased there is an inherent limit to the degree of ILP that can be obtained, depending on the application involved. The presence of conditional branches is one of the factors that most affects the compiler's capacity to achieve high ILP levels, mainly because it limits the possibility of code shifts. Shifting a portion of code belonging to a particular branch above that branch (speculative execution) is, in fact, extremely complex when there are several branches. No less important is the latency affecting branch instructions and the consequent large bubbles introduced in the pipeline.

An essential technique to overcome this limit is the formation of *hyperblocks* [1]. A hyperblock is formed by combining basic blocks belonging to different execution paths. Conditional branches are removed to form hyperblocks by transforming the original code, an operation called *if-conversion*. If-conversion replaces conditional branches of the code with comparison instructions that set the Boolean content of one of the predicate registers. All the instructions in a certain branch are thus converted to "predicate instructions", the execution of which depends on the value of a source operand contained in a previously set predicate register. In this way, control dependencies are transformed into dependencies between data.

Having been developed as an advanced technique to enhance ILP, the effects on performance are well-documented. Less well-known is the fact that with the spread of VLIW processors in systems in which performance is not the only requirement the increase in performance achieved by hyperblock formation clashes with the need to control power and energy consumption. As we will see, the *maximize ILP* paradigm should therefore be modified in the light of what could be defined as the "collateral effects" that code transformations have on energy consumption and the average amount of power dissipated.

In this paper we present a first attempt to analyse these problems from a quantitative point of view. The analysis is carried out on a set of typical embedded multimedia applications rather than the ad-hoc kernel loops often used to demonstrate the effectiveness of hyperblock formation.

The rest of the paper is organized as follows: the following section deals with work on compiling techniques analysed from the low power/energy point of view; in Section III we describe the experimental framework used, and then in Section IV the experiments performed. Finally, we provide concluding remarks and future directions in Section V.

II. Related Work

The focus of compiler optimizations has traditionally been on improving performance (see for example [2] and the references therein). Compiler optimizations have been widely used to achieve speedups on array-based codes. Such optimizations are becoming increasingly important in embedded multimedia systems. However, some other performance indexes, like energy consumption and power dissipation, play a significant role in this scenario. Unfortunately, there has been little effort to analyze the effect of state-of-the-art compilation techniques both on energy consumption and power dissipation. In [3] Kandemir *et al.* present an experimental evaluation of several high-level compiler optimizations (linear loop transformations, tiling,

0-7803-8834-8/05/$20.00 ©2005 IEEE.

unrolling, fusion, and scalar expansion) on energy consumption for an architectural template based on SimpleScalar. In [4] Kremer discuss general issues and challenges related to compilers for power and energy management and address the question whether power, energy, and performance should be considered separate compiler optimization goals or not. Moreover, through simple examples he shows that optimizing for minimal power dissipation or minimal energy usage may have different metrics, and therefore result in different optimization strategies. Madhavi and Lizy in [5] present a quantitative study wherein they examine the effect of the standard optimization levels of DEC Alpha's *cc* compiler on power and energy of the processor, finding that optimizations that improve performance by reducing the number of instructions are optimized for energy.

In our work we concentrate on the influence of performance-oriented compiler optimizations (specifically hyperblock formations) on power dissipation and energy consumption.

III. Experimental Framework

To carry out the tests presented here we used EPIC-Explorer [6] a parameterized platform for Design Space Exploration of VLIW architectures built on the Trimaran integrated compilation and performance monitoring infrastructure [7]. EPIC-Explorer is a framework that allows us to evaluate any instance of a platform in terms of area, performance and power, exploiting the state of the art in estimation approaches at a high level of abstraction.

A configuration of the system generates an instance that is simulated and evaluated for a specific application. The application written in C is first compiled. Trimaran uses the IMPACT compiler system as its front-end. The code produced, together with the High Level Machine Description Facility (HMDES) [8] machine specification, represents the Elcor input. This language describes a processor architecture from the compiler's point of view. Elcor is Trimaran's back-end for the architecture and is to a large extent parameterized by the machine description facility. The Trimaran framework also consists of a simulator which is used to generate various statistics such as compute cycles, total number of operations, etc..

Together with the configuration of the system, the statistics produced by simulation contain all the information needed to apply performance and power consumption estimation models. The estimation models used are based on activity and inactivity values for each of the functional blocks in the architecture. Periods of activity/inactivity are obtained from the execution statistics, thus providing an estimate of the average power dissipation [9]. The results presented in the following sections use activity/inactivity power values referring to a 0.1μ and $1.3V$ technology. We also assume a clock frequency of 200MHz. For further details about the platform, the reader is referred to [6].

IV. Experimental Results

In the following subsections we will analyze the impact of hyperblock formation on performance, power dissipation and energy consumption. The class of applications being considered belongs to the MediaBench suite [10] and represents quite a broad spectrum of the possibilities of using a VLIW architecture in an embedded multimedia environment. All tests were carried out on a Pentium III-700MHz with 1.5GB RAM running GNU/Linux 2.4.9.

To have an idea of the dimensions of the benchmarks, input data and compiling times, we can refer to Table I. As can be seen, with hyperblock formation the compilation times are much longer (ranging from 350% to 620%). A great impact is observed on the size of the generated assembly code which increase by 250% on average.

Table II gives the results obtained with various numbers of functional units for integer operations, with and without hyperblocks. As we are not interested in variations in the parameters of the memory hierarchy, we assume a 100% cache hit rate. The number of functional units of other kinds is considered to be fixed, as is the number of registers in each of the register files and the cache configuration. The values are given in Table III. This choice is not only necessary for greater legibility but is also the most natural one as the integer units are the most used and therefore the units that best represent an increase in the resources that can be exploited by hyperblock formation to enhance ILP.

TABLE III
Reference architecture.

Functional Units	
Integer Units	1, 2, 3, 4
Floating Point Units	1
Memory Units	1
Branch Units	1
Register Files Size	
General Purpose	64
Floating Point	64
Predicate	64
Control	64
Branch Target	16
Cache Configuration	
L1 Instruction Size/Block size/Associativity	64KB/64B/1
L1 Data Size/Block size/Associativity	64KB/64B/1
L2 Unified Size/Block size/Associativity	256KB/128B/4

A. Performance Perspective

Before analyzing the results from the power/energy point of view, it is worthwhile making some observations about the traditional objective of hyperblock formation, i.e. an increase in performance. This is of fundamental importance because some of the phenomena we observe when analyzing performance have repercussions on the power/energy characteristics.

If, considering a certain application, we run down Table II it is evident that an increase in the number of functional units causes a decrease in the number of execution cycles. Obviously, as already mentioned, the presence of several functional units provides the compiler with more resources for the parallel scheduling of instructions. This is evident in the column referring to the IPC achieved, defined as:

$$IPC = \frac{instruction\ count}{clockcyles} \tag{1}$$

The increase in the IPC with varying numbers of functional units is even more marked if we observe the results obtained by enabling hyperblock formation. As was to be expected, in fact, as soon as the limits imposed by the conditional branches are removed, code moving operations are facilitated. Hyperblock formation, therefore, cannot but increase the IPC. But from a performance point of view, is the ultimate aim that of achieving the highest possible IPC values? The answer is no, as can be observed by comparing `gsm-decode` and `adpcm-decode` with and without hyperblock formation. Analytically this can be accounted for by obtaining the number of cycles from Equation (1):

$$clock\ cycles = \frac{instruction\ count}{IPC}$$

TABLE I

SET OF APPLICATION USED FOR THE EXPERIMENTS.

Application	Compilation time (sec)		Input	code size ratio
	without hyperblocks	with hyperblocks	size (KB)	(with/without hyperblocks)
`g721-encode`	42	203	8	3.2
`gsm-decode`	660	2285	1	2.2
`mpeg2-decode`	250	909	4	3.3
`mpeg2-encode`	390	1492	6	1.8
`adpcm-decode`	18	113	1	2.4

TABLE II

EFFECT OF HYPERBLOCK FORMATION ON POWER, ENERGY, AND PERFORMANCE.

		Without hyperblock formation				With hyperblock formation			
Application	IU	IPC	Cycles	Energy (mJ)	Power (W)	IPC	Cycles	Energy (mJ) (mJ)	Power (W) (W)
`g721-encode`	1	1.1	9345140	72.99	1.56	1.19	10324000	86.52	1.68
	2	1.33	8047890	70.30	1.75	1.74	7772630	79.91	2.06
	3	1.37	8001040	71.57	1.79	1.97	7347280	80.86	2.20
	4	1.39	7970320	72.85	1.83	2.00	7370330	82.86	2.25
`gsm-decode`	1	1.05	2407080	18.34	1.52	1.18	4849870	41.83	1.72
	2	1.2	1981100	17.16	1.73	1.38	4164510	44.59	2.14
	3	1.23	1980030	17.50	1.77	1.40	4479710	48.95	2.19
	4	1.24	1979760	17.85	1.80	1.41	4835360	53.19	2.20
`mpeg2-decode`	1	1.14	8289240	78.03	1.88	1.25	7263270	70.25	1.93
	2	1.28	6027300	76.04	2.27	1.46	5896990	67.02	2.52
	3	1.3	5919980	76.71	2.34	1.49	5836660	68.43	2.59
	4	1.31	5837790	77.42	2.39	1.50	5823250	69.63	2.65
`mpeg2-encode`	1	1.19	46084700	408.09	1.77	1.31	38432201	391.44	1.89
	2	1.4	35802400	395.74	2.21	1.57	31656405	378.12	2.44
	3	1.47	34896900	398.52	2.28	1.61	30943440	382.56	2.56
	4	1.48	34511700	403.10	2.34	1.65	30743420	390.87	2.61
`adpcm-decode`	1	1.13	7455300	58.61	1.57	1.37	8323380	76.99	1.85
	2	1.48	5706770	53.33	1.87	1.74	6577180	71.87	2.19
	3	1.48	5706480	54.33	1.90	1.81	6331410	72.06	2.28
	4	1.48	5706330	55.34	1.94	1.79	6380680	73.38	2.30

A higher IPC is therefore only synonymous with better performance if the increase in the number of instructions executed due to hyperblock formation does not exceed the increase in the IPC. This, however, does not happen, as can be seen, for example, in the instruction mix shown in Figure 1 which refers to the `adpcm-decode` application for 4 integer units.

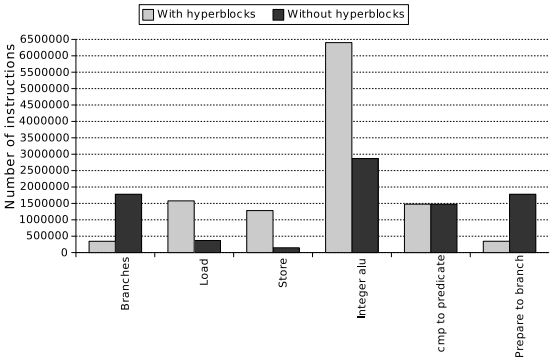

Fig. 1. Instruction mix with and without hyperblock formation for the `adpcm-decode` application for 4 integer units.

So although the code transformations and duplications performed during hyperblock formation certainly increase the IPC, they may cause an increase in the number of instructions that does not compensate for the advantages in terms of performance. The consequences of this emerge when we address the problem from an energy/power perspective.

B. Energy/Power Perspective

Although they are at times erroneously used as synonyms, it should be stressed that in our perspective power and energy objectives are to be considered as two independent quantities. Energy consumption, for example, may be of decisive importance in designing mobile devices running on batteries. Power dissipation, on the other hand, which is linked to the amount of heat the system is subjected to, is a fundamental element for aspects such as packaging, which directly affect the final cost of implementing the system. As we will see, the nature of the architecture considered, which allows several instructions to be scheduled in parallel, as well as the use of a code transformation technique like hyperblock formation, shows up this conceptual difference even more clearly.

If we run down the columns referring to energy consumption with increasing numbers of functional units in Table II, we observe that a decrease in the number of cycles does not correspond to a decrease in energy consumption. The values are relatively constant, following a parabolic trend with a slight initial drop of 4-5% and then a rise. In a traditional architecture we would expect executing fewer cycles to correspond to "less work" and thus less energy consumption. But in architectures with multiple issue like a VLIW architecture,

the reduction in the execution cycles derives from the fact that the instructions are executed in parallel rather than sequentially. So if an instruction of a certain kind, associated with the use of certain functional units, consumes a certain quantity of Joules, moving it back or forward will by no means annul this consumption. The overall energy consumption will therefore depend on the number of instructions executed rather than the number of cycles.

How does hyperblock formation affect this? On the basis of the previous remarks, and reconsidering Equation (1) from another perspective, we get:

$$Energy \propto clockcycles \times IPC$$

So, with the same number of functional units, with the introduction of hyperblock formation we will have:

$$\frac{Energy_{new}}{Energy_{old}} \propto \frac{clock\ cycles_{new} \times IPC_{new}}{clock\ cycles_{old} \times IPC_{old}}$$

As the introduction of hyperblock formation cannot but increase the IPC, in a project with constraints on energy consumption it is necessary to assess whether the corresponding variation in the number of execution cycles balances the variation in the IPC.

While it is clear that following an increase in the number of execution cycles the introduction of hyperblock formation gives no advantages as far as energy consumption is concerned, the opposite does not hold. That is, a decrease in the number of cycles due to hyperblock formation does not mean an advantage in terms of energy. This happens, for example, in `g721encode`, where we have a 3 to 8% increase in performance but the IPC increase due to the introduction of hyperblock formation is such as to increase energy consumption by almost 14%. For the sake of clarity the percent variations are given separately in Table IV.

TABLE IV
EFFECT OF HYPERBLOCK FORMATION ON PERFORMANCE, POWER, AND ENERGY.

Application	IU	IPC	Cycles	Energy	Power
`g721-encode`	1	8.6%	10.5%	18.5%	7.3%
	2	30.8%	-3.4%	13.7%	17.7%
	3	44.3%	-8.2%	13.0%	23.0%
	4	44.2%	-7.5%	13.7%	23.0%
`gsm-decode`	1	12.7%	101.5%	128.1%	13.2%
	2	14.7%	110.2%	159.9%	23.7%
	3	14.0%	126.2%	179.7%	23.6%
	4	13.0%	144.2%	198.0%	22.0%
`mpeg2-decode`	1	10.2%	-12.4%	-10.0%	2.7%
	2	14.0%	-2.2%	-11.9%	11.0%
	3	14.5%	-1.4%	-10.8%	10.5%
	4	14.3%	-0.2%	-10.1%	10.9%
`mpeg2-encode`	1	10.3%	-16.6%	-4.1%	6.7%
	2	12.3%	-11.6%	-4.5%	10.4%
	3	9.6%	-11.3%	-4.0%	12.1%
	4	11.4%	-10.9%	-3.0%	11.7%
`adpcm-decode`	1	21.2%	11.6%	31.4%	17.7%
	2	17.6%	15.3%	34.8%	16.9%
	3	22.3%	11.0%	32.6%	19.5%
	4	20.9%	11.8%	32.6%	18.6%

As can be observed in Table II the power dissipation values follow a more homogeneous trend than the energy consumption values. The dissipated power grows, in fact, both when the number of functional units increases and when hyperblock formation is enabled. Whereas variations in the number of functional units led to a limited oscillation in energy consumption, the same does not apply to average power dissipation, which follows an increasing trend. It is clear that as power is a measure of the speed at which energy is consumed at a given instant, it cannot but increase as the average number of operations executed per cycle, the IPC, grows. Even in the hypothesis that the larger number of functional units are not exploited for parallel scheduling, and thus with a view to increasing the IPC, the power would still be greater because the additional units, although inactive, would still cause static power consumption. By the logic behind hyperblock formation, the transformations it produces cannot but reinforce this greater instantaneous consumption effect and thus lead to an increase in average consumption. So although from the energy point of view a gain is possible in some cases, power dissipation grows quite considerably. In Table IV we can observe values ranging from 17% to 50%.

V. CONCLUSIONS AND FUTURE WORK

In this paper, a first study of hyperblock formation from the power and energy perpective has been carried out. The results showed that the way this aggressive code optimization influences each of these two magnitudes is different, both considering a quantitative and behavioural point of view. We found that average power, assuming a fixed hardware configuration (e.g. same functional units and register files), can increase up to 40-50% due the IPC improvement. On the other hand, effects on energy consumption showed to be less predictable, not being directly related to the performance gain that hyperblock formation attemps to achieve. For our future work we would like to investigate the performance/power/energy interaction between the hyperblock formation compiler optimization technique and the architectural parameters of a VLIW-based system with the aim to define new multiobjective design space exploration strategies.

REFERENCES

[1] S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann, "Effective compiler support for predicated execution using the hyperblock," in *International Symposium on Microarchitecture*, Dec. 1992, pp. 45–54.

[2] M. Wolfe, *High Performance Compilers for Parallel Computing*. Pearson Education POD, 1995.

[3] M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye, "Influence of compiler optimizations on system power," in *Design Automation Conference*, Los Angeles, California, USA, 2000, pp. 304–307.

[4] U. Kremer, *Low Power/Energy Compiler Optimizations*. CRC Press, 2004, ch. 35.

[5] M. G. Valluri and L. John, "Is compiling for performance == compiling for power?" in *Annual Workshop on Interaction between Compilers and Computer Architectures*, Monterrey, Mexico, Jan. 2001.

[6] G. Ascia, V. Catania, M. Palesi, and D. Patti, "EPIC-Explorer: A parameterized VLIW-based platform framework for design space exploration," in *First Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, Newport Beach, California, USA, Oct. 3–4 2003.

[7] "An infrastructure for research in instruction-level parallelism," http://www.trimaran.org/.

[8] J. Gyllenhaal, "A machine description language for compilation," Master's thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana IL, Sept. 1994.

[9] G. Cai and C. H. Lim, "Architectural level power/performance optimization and dynamic power estimation," in *Cool Chips Tutorial colocated with MICRO32*, Nov. 1999, pp. 90–113.

[10] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," in *International Symposium on Microarchitecture*, Dec. 1997.