Reducing Complexity of Multi-objective Design Space Exploration in VLIW-based Embedded Systems[1]

VINCENZO CATANIA, MAURIZIO PALESI, DAVIDE PATTI
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania, Italy

Architectures based on Very Long Instruction Word (VLIW) have found fertile ground in multimedia electronic appliances thanks to their ability to exploit high degrees of Instruction Level Parallelism (ILP) with a reasonable trade-off in complexity and silicon cost. Specialization of such architectures involves the configuration of both hardware-related aspects (e.g., register files, functional units, memory sub-system) and software-related issues (e.g., the compilation strategy). The complex interactions between the components of such systems will force a human designer to rely on judgement and experience in designing them, possibly eliminating interesting configurations, and making tuning of the system, for either power, energy or performance, difficult. In this paper we propose tools and methodologies to cope efficiently with this complexity from a multi-objective perspective. We first analyze the impact of ILP oriented code transformations using two alternative compilation profiles to show quantitatively the effect of such transformations on typical design objectives like performance, power dissipation and energy consumption. Next, by means of statistical analysis we collect useful data to predict the effectiveness of a given compilation profiles for a specific application. Information gathered from such analysis can be exploited to drastically reduce the computational effort needed to perform the design space exploration.

Categories and Subject Descriptors: C.3 [**Special-purpose and Application-based Systems**]: Microprocessor/microcomputer applications; G.3 [**Probability and Statistics**]: Statistical computing; I.6.3 [**Simulation and Modeling**]: Applications; J.6 [**Computer-aided Engineering**]: Computer-aided design (CAD)

General Terms: Algorithms, Performance

Additional Key Words and Phrases: VLIW architectures, ILP, Power, Energy, Performances, Design Space Exploration, Hyperblock formation, Multi-Objective optimization, Genetic Algorithms, Statistical analysis

[1]New Paper, Not an Extension of a Conference Paper.

Author's address: V. Catania, University of Catania, Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, V.le Andrea Doria, 6 – 95125 Catania, Italy; M. Palesi, University of Catania, Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, V.le Andrea Doria, 6 – 95125 Catania, Italy; D. Patti, University of Catania, Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, V.le Andrea Doria, 6 – 95125 Catania, Italy.

1. INTRODUCTION

The embedded systems market is without a doubt the largest and most significant application area for microprocessors [WSTS 2006]. There are basically two reasons for this success. The first is the shorter lifecycle for products based on embedded systems, which has led to increased competition between manufacturers. The second is the constant increase in the number, complexity and heterogeneous nature of the functions these products have to offer. The reduction in the time-to-market has also made it infeasible to design a processor from scratch for a specific application. On the other hand, the design of an embedded system is application-specific and so the use of general-purpose microprocessors is often not only inappropriate but also infeasible in terms of performance, cost, power, etc..

Architectures based on *Very Long Instruction Word* (VLIW) processors [Fisher 1983] are emerging in the domain of modern, increasingly complex embedded multimedia applications, given their capacity to exploit high levels of performance while maintaining a reasonable trade-off between hardware complexity, cost and power consumption. A VLIW architecture, like a superscalar architecture, allows several instructions to be issued in a single clock cycle, with the aim of obtaining a good degree of Instruction Level Parallelism (ILP). But the feature which distinguishes the VLIW approach from other multiple issue architectures is that the compiler is exclusively responsible for the correct parallel scheduling of instructions. The hardware, in fact, only carries out a *plan of execution* that is statically established by the compiler according to the hardware resources available.

If we consider the original sphere of application of VLIW systems, that is high performance computing, it is evident that the compilation techniques developed all aim to extract the maximum ILP from the application. Considering an embedded scenario, choosing one compilation policy rather than another may significantly affect other possible objectives, such as dissipated power and/or energy consumption. Energy consumption, for example, could prove to be a decisive factor in battery-powered mobile devices. 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. The rediscovery of VLIW-based systems in embedded environments thus makes it necessary to reconsider ILP-oriented compilation from a perspective other than that of classical performance requirements.

The problem of configuring the parameters of a VLIW-based system thus becomes a *multi-objective optimization* problem, involving interaction between hardware features (e.g., the number and type of functional units, register files) and software aspects (e.g., the compilation strategy). Since the size of the design space increases as the product of the cardinalities of the parameter spaces, exhaustive exploration of the design space is only computationally feasible in systems with a very limited number of parameters. In real cases, the number of parameters and the size of the relative parameter spaces excludes the possibility of exhaustive exploration. In the case study presented in this work, for example, the parameterized VLIW architecture comprises 18 parameters which generate a design space of over 10^9 configurations. Even considering an evaluation time of a few seconds for each configuration, exhaustive exploration would take hundreds of years.

The main focus of this paper is to propose tools and methodologies to cope efficiently with this complexity. In particular, we show how statistical analysis allows to forecast characteristics of the Pareto-optimal set by evaluating few points of the design space. With only a minimal overhead, such preliminary analysis is useful to drastically reduce the compu-

tation effort required by the subsequent design space exploration phase. The contributions this paper makes can be summarized as follows:

—Providing an extensive analysis of the complex interactions between architectural parameters, applications, as well as compiler options on performance, power dissipation and energy consumption of a VLIW based embedded platform.

—Proposing the use of statistical analysis as a viable solution to drastically reduce the size of the design space by cutting off system configurations and compilation profiles which do not add significant and useful information to the tradeoff configurations set.

—Developing an efficient design space exploration approach by coupling statistical analysis techniques with evolutionary computing techniques resulting in a drastic reduction of the exploration time without any significant loss in accuracy.

Methodology assessment is carried out by means of a highly parameterized VLIW based platform whose design space is spanned by processor architecture parameters, memory hierarchy configuration, as well as different compiler optimization strategies.

The rest of the paper is organized as follows. Section 2 summarizes some previous works in the field of system-level exploration of VLIW architectures. Section 3 describes EPIC-Explorer, the simulation and exploration platform used in our experiments, together with the models used to estimate the design objectives and the application set used throughout this work. In Section 4 we analyze quantitatively the possible effects of ILP oriented code transformations on the design objectives. Next, in Section 5, we show how it is possible to use statistical analysis to predict the effect of two alternative compilation profiles. A set of representative Pareto set is discussed in Section 6. Finally, Section 7 summarizes our contribution and outlines some directions for future work.

2. RELATED WORK

There are a number of contributions in the literature regarding system-level exploration of VLIW-based architectures. The main difference between them lies in the objectives to be optimized and the architectural and/or micro-architectural elements the analysis focuses on.

A first area of research regards the design of high performance VLIW application-specific processors. In [Hekstra et al. 1999] Hekstra *et al.* present methods and tools to perform design space exploration concerning the precise functional unit configuration for a 64-bit VLIW core for future TriMedia processors. The exploration aims to identify optimal placement (in terms of area and performance) of the 30 different functional units in the 5 issue slots available. The possibility of introducing application-specific functional units is analyzed in [Middha et al. 2002] by Middha *et al.*. The authors show that the use of an Application Specific Instruction Set Processor (ASIP) in embedded systems allows much more flexible solutions than implementations based on ASIC and much more efficient than implementations based on standard processors in terms of both performance and power consumption. In these works, however, the exploration of the design space is limited to the microprocessor core and do not consider the important effects played by the memory hierarchy.

Another research area focuses on exploration of the design space for VLIW architectures. Lapinskii *et al.* in [Lapinskii et al. 2002] present a kernel-specific and technology-independent methodology for exploration of the design space of clustered VLIW ASIP

data paths. The results obtained on a set of computation-intensive benchmarks showed that the penalties of clustered versus centralized datapaths are often minimal and that clustering indeed unlocks a variety of valuable design tradeoffs. In [Fisher et al. 1996] Fisher *et al.* present a system for the automatic design of VLIW architectures optimized for the execution of a specific application but functionally valid for the execution of other applications. Exploration of the design space is oriented toward joint optimization of the speedup and the hardware costs. Capitanio *et al.* in [Capitanio et al. 1992] describe a fine-grain code partitioning method to limit the number of register file ports in VLIW architectures. The design space obtained with various number of register file ports, the number of partitions and the communication bandwidth between the partitions is then explored, evaluating the performance penalty. An interesting example of multi-objective exploration is PICO-VLIW [Kathail et al. 2002] developed by HP-Labs. It is a tool for synthesis and design space exploration to identify area/performance tradeoffs in a generic VLIW architecture. To allow exploration of the extremely large number of configurations, a hierarchical evaluation approach is adopted, separating exploration of the VLIW core and that of the memory subsystem. In [Fischer et al. 2002] the authors present an exploration algorithm for architecture/compiler co-design of ASIPs in which the design space is spanned by processor architecture parameters as well as different compiler optimization strategies. The design space is explored with the goal to optimize hardware cost, code size and performance. The analysis involves only the microprocessor without considering the memory subsystem which has been proven to be fundamental to meet the overall performance and power design constraints [Hennessy and Patterson 2006; Venkatachalam and Franz 2005; Balasubramonian et al. 2000] especially in a VLIW scenario [Raghavan et al. 2006; Morgan et al. 2005; Abraham and Mahlke 1999]. Although the approach is interesting, it is not general but depends on the system architecture being considered. In addition, like in [Kathail et al. 2002], parameters are explored independently of each other which is a serious drawback in design space exploration as stated in [Givargis et al. 2002; Ascia et al. 2005].

Alongside traditional research aimed at maximizing performance, interest has recently been shown in estimation and architectural exploration from the power and energy perspective. Kim *et al.* in [Kim et al. 2001] present a framework, built on the Trimaran VLIW tool-set [Trimaran], to estimate the amount of energy consumed. The framework is then used to analyze the impact of different architectural choices and compilation optimizations on energy efficiency. An instruction-level power model for VLIW architectures was proposed by Benini *et al.* in [Benini et al. 2002]. The model is based on a hierarchy of dynamic power estimation engines: From the instruction level down to the gate/transistor level. The model is then used to develop a system-level simulation framework for dynamic profiling of power consumption during execution of an application. The model estimates the energy associated with a long instruction as the sum of the energy associated with each single operation of the long instruction and the single pipeline stages. In [Pokam and Bodin 2004] Pokam and Bodin explore the energy-delay tradeoff of ILP enhancing techniques at the compilation level. The methodology used exploits the variations in performance of the program to identify conditions which lead to an increase in energy consumption. They reach the conclusion that there exists a threshold beyond which ILP enhancing optimizations may necessarily turn into diminishing energy reduction returns. The impact on power and performance due to code transformation techniques in VLIW architectures is presented by Masselos *et al.* in [Masselos et al. 1999]. The techniques proposed aim to

optimize power consumption by the memory subsystem. The techniques have a positive impact on performance as well, and were validated on video processing applications on a multimedia VLIW processor. All these approaches, however, are not integrated with a design space exploration methodology and the analysis of the different tradeoffs is carried out manually defining the system configurations to be compared.

In [Srinivasan et al. 2002] Srinivasan *et al.* show that the power/performance analysis cannot be merely based on a CPI-centric view in early-stage definition studies. They present an optimization methodology that starts with an analytical power-performance model to derive optimal pipeline depth for a superscalar processor. Although their work is focused on superscalar architectures, the general results obtained are in line with that we will present in this paper for VLIW architectures.

In this paper we present a system-level analysis of a parameterized VLIW-based platform to evaluate the impact on performance, energy and power from a multi-objective perspective. The huge design space is spanned by both architectural related parameters as well as different compilation profiles. The main contribution this paper intends to make is the use of statistical analysis techniques to capture important properties of the design space which can be exploited to drastically reduce the computation effort required to explore the design space.

3. SIMULATION ENVIRONMENT

To evaluate and compare the performance indexes of different architectures for a specific application, one needs to simulate the architecture running the code of the application. To make architectural exploration possible both the compiler and the simulator have to be retargetable. Trimaran [Trimaran] provides these tools and thus represents the pillar around which we have constructed EPIC-Explorer [Ascia et al. 2003]. EPIC-Explorer is a framework that not only allows us to evaluate any instance of a platform in terms of area, performance and power, but also implements various techniques for exploration of the design space.

In this section, we briefly describe the parameterized VLIW platform used as testbed for the experiments, the general evaluation flow along with the high-level estimation models used to evaluate the performance indexes to be optimized and the set of applications used as benchmarks.

3.1 Reference Architecture

The parameterized system architecture used in this work is based on HPL-PD [Kathail et al. 2000] which is a parametric processor meta-architecture designed for research in instruction-level parallelism of EPIC/VLIW architectures[2]. The HPL-PD opcode repertoire, at its core, is similar to that of a RISC-like load/store architecture, with standard integer, floating point (including fused multiply-add type operations) and memory operations.

Architectural parameters can be classified in three main categories: *register files*, *functional units* and *memory sub-system*. The first two depend on the implementation of the VLIW core and regard the size of the register files, in terms of the number of registers contained in each of them, and the number of functional units for each type of unit supported.

[2]EPIC is an extension of the VLIW approach; where it is not necessary to make a distinction we will only use the term VLIW for the sake of simplicity.

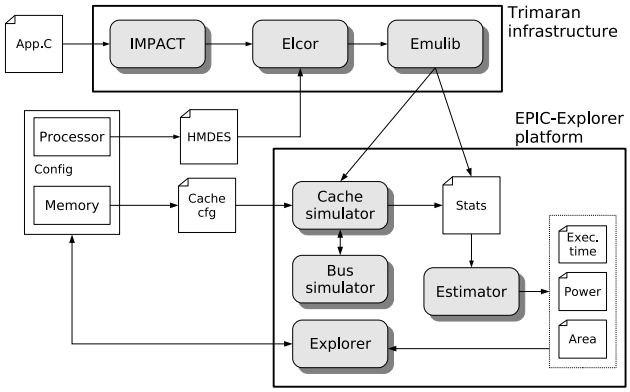


Fig. 1. Block diagram of the framework.

As far as the former are concerned, five different types of register files can be identified: GPR (32-bit registers for integers), FPR (64-bit registers for floating point values) PR (1-bit registers used to store the Boolean values of predicated instructions), BTR (64-bit registers containing information about possible future branches) and CR (32-bit control registers containing information about the internal state of the processor). The functional units involved are: *Integer units*, *floating point units*, *memory units* (associated with load-/store operations) and *branch units* (associated with branch operations). With respect to the memory sub-system, the parameters that can be modified are the *size*, *associativity* and *block size* for each of the three caches: First-level data cache (L1D), first-level instruction cache (L1I) and second-level unified cache (L2U).

3.2 Evaluation Flow

We will now show a functional scheme highlighting the main blocks of the EPIC-Explorer framework and the interface with the Trimaran tools. The input for the whole evaluation flow is the source of the application involved in the optimization and the configuration of the architecture being evaluated. With reference to Figure 1 this input is represented by the blocks *App.c* and *Config*.

The application (App.c) is first compiled by the Trimaran's compiler front-end (IMPACT). This front-end performs ANSI C parsing, code profiling, classical code optimizations and block formation. The intermediate code produced, together with the High Level Machine Description Facility (HMDES) machine specification [Gyllenhaal 1994], represents the Elcor input. The HMDES is the machine description language used in Trimaran, which describes a processor architecture from the compiler's point of view. With reference to the tunable parameters outlined in the previous subsection, the hmdes file specifies the number and type of register files and functional units in the VLIW core. Elcor is Trimaran's back-end VLIW compiler for the HPL-PD architecture and it is parameterized by the hmdes machine description. It performs three tasks: Code selection and scheduling, register allocation, and machine dependent code optimizations. At the end of the compilation flow, Trimaran supplies a simulation library (Emulib) which makes it possible to execute the VLIW code produced by Elcor, generating a file (Stats) containing the execution statistics (e.g., instruction mix, execution cycles, utilization of functional units,

etc.). A cache simulator, along with a bus simulator, is used to gather information about the behavior of the memory hierarchy in terms of miss rate and data/address traffic on the interconnection buses.

Together with the configuration of the system, the statistics produced by simulation contain all the information needed to apply the area, performance and power consumption estimation model implemented in the *Estimator* component of EPIC-Explorer. The results obtained by these models are the input for the *Explorer* component. This component executes an optimization algorithm, the aim of which is to modify the parameters of the configuration so as to minimize the three cost functions (area, execution time and energy/power consumption).

3.3 Estimation Models

Several models have been implemented in the EPIC-Explorer platform in order to estimate the power/energy/performance indexes of each simulated system configuration.

The average power consumed by the processor was estimated using an adaptation of the Cai-Lim model [Cai and Lim 1999] to the VLIW processor. As regards the cache subsystem, a transition-based model was used, according to the equations described in [Kamble and Ghose 1997]. The main memory energy is based on the model in [Shiue and Chakrabarti 1999] and assumes a per main memory access energy of $4.95 \times 10^{-9} J$ based on the data for the Cypress CY7C1326-133 memory chip. The contribution towards power consumption made by the interconnection system was calculated by counting the number of transitions on the bus lines and applying the formula $P_{bus} = 1/2 V_{dd}^2 \alpha f C_l$ where V_{dd} is the supply voltage, α is the switching activity, f is the clock frequency and C_l is the capacity of a bus line. For the on-chip buses we also considered the coupling capacitances between bus lines, using the model in [Henkel and Lekatsas 2001]. For a detailed description of the models used and their adaption to the case of a VLIW based system see [Ascia et al. 2003].

The performance statistics produced by the simulator are expressed in clock cycles. To evaluate the execution time it is sufficient to multiply the number of clock cycles by the clock period. This was set to 200MHz, which is long enough to access cache memory in one single clock cycle.

3.4 Reference Application Set

The class of applications being considered belongs to the MediaBench suite [Lee et al. 1997] and represents quite a broad spectrum of the possibilities of using a VLIW architecture in an embedded multimedia environment. Table I shows the set of applications chosen, with a brief description and the input data set size. All tests and data reported in this work refer to simulations performed on a Pentium III-700MHz with 1.5GB RAM running GNU/Linux.

In the following sections we will show how EPIC-Explorer was used to analyze the influence of a variation in one or more features of the architecture and/or compiler on performance, power dissipation and energy consumption.

4. ILP ORIENTED COMPILATION

The contribution of this section is to analyze the implications of ILP-oriented compilation in a multi-objective design scenario typical of embedded systems.

Table I. Set of Mediabench applications used for the experiments.

Application	Description	Input size (KB)
`adpcm-dec`	Adaptive Differential Pulse Code Modulation speech decoding	1
`adpcm-enc`	Adaptive Differential Pulse Code Modulation speech encoding	295
`fir`	FIR filter	64
`g721-enc`	CCITT G.711, G.721 and G.723 voice compressions	8
`gsm-dec`	European GSM 06.10 full-rate speech transcoding	1
`gsm-enc`	European GSM 06.10 full-rate speech transcoding	8
`ieee810`	IEEE-1180 reference inverse DCT	1
`jpeg`	JPEG image compression and decompression	2
`mpeg2-dec`	MPEG-2 video bitstream decoding	4

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 ILP. 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. However, 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. In fact, conditional branches fragment the code in several *basic blocks*, that is, the maximal straightline code fragments with a single entry/exit point . To achieve high levels of parallelism, the compiler should be allowed to schedule instructions belonging to different basic blocks in parallel. This extremely complex task, that involves moving instructions above branches (speculative execution), makes the presence of several basic blocks an intrinsic limit to the amount of parallelism that can be extracted from the application code. A solution to overcome this limit and achieve an effective ILP-oriented scheduling is to extend the scope of action of the compiler to wider regions of code. A typical example are the *hyperblocks* [Mahlke et al. 1992], formed by combining basic blocks belonging to different execution paths[3].

In order to investigate the effect of such ILP-oriented code transformations we can, for example, consider two different *compilation profiles*, as shown in Table II. It should be pointed out that with the word "profile" we hereby mean a set of values chosen for each of the compiler parameters. This should not be confused with the term profile intended as the result of a code profiling. For space reasons, we do not show parameters that remain unchanged among the profiles (for a complete list of the parameters available see [Schlansker et al. 1996]). In particular, in the following we will refer to two different compilation profiles, namely *N* and *H*: The first is chosen as a default conservative profile, where only

[3]A discussion of the various ILP-oriented code transformation is beyond the scope of this work, see [Schlansker et al. 1996] for a survey.

Table II. The compilation profiles *N* and *H*.

Parameter	Profile	
	N	H
issue width	8	8
branch combining		√
predicated execution		√
classical optimization	√	
modulo scheduling	√	√
min cb weight	-	20
path max op growth	-	2.1
path max dep growth	-	4.25
path min exec ratio	-	0.00075
path min main exec ratio	-	0.05
path min priority ratio	-	0.10
block min weight ratio	-	0.005
block min path ratio	-	0.015
unsafe jsr priority penalty	-	0.005
safe jsr priority penalty	-	0.01
pointer st priority penalty	-	1.0
peel enable	-	√
peel max ops	-	36
peel infinity iter	-	6
peel min overall coverage	-	0.75
peel min peelable coverage	-	0.85
peel inc peelable coverage	-	0.10

Table III. Compilation time for profiles *N* and *H*.

Application	Compilation time (sec)		Slowdown factor
	N	H	
`adpcm-dec`	18	113	6.28
`adpcm-enc`	20	118	5.9
`fir`	21	26	1.24
`g721-enc`	42	203	4.83
`gsm-dec`	660	2285	3.46
`gsm-enc`	813	2946	3.62
`ieee810`	34	118	3.47
`jpeg`	30	123	4.1
`mpeg2-dec`	250	909	3.64

classical C optimization are performed (e.g. loop unrolling, function inlining); on the other side, profile *H* refers to an aggressive ILP oriented compilation, involving some heavy code transformations that the formation of hyperblocks allows to the compiler.

Table III reports, for each application, the compilation times required along with the slowdown factor. Of course, as it could have been expected, choosing profile *H* results in a more complex and time consuming compilation phase. It's intuitive that if compilation time dominates the total time required to evaluate a configuration, then the overhead introduced by profile *H* could severely degrade the time-efficiency of a space exploration algorithm or, seen from an other perspective, could reduce the number of configurations evaluated in a fixed amount of time. A second, but not less relevant issue, is that compiler

Table IV. Results obtained for the prefixed reference configuration and the two compilation profiles *H* and *N*.

Application	Clock Cycles ($\times 10^3$)			Energy (mJ)			Power (W)			IPC
	Total	CPU	Stall	Total	CPU	MEM	Total	CPU	MEM	
`adpcm-dec` (N)	5710	5706	4.1	61.9	55.3	6.5	2.1	1.9	0.2	1.4
`adpcm-dec` (H)	6436	6429	7	85.4	74.0	11.4	2.6	2.3	0.3	1.7
`adpcm-enc` (N)	7430	7426	4.4	76.8	69.2	7.6	2.0	1.8	0.2	1.3
`adpcm-enc` (H)	4689	4683	5.7	75.6	67.3	8.3	3.2	2.8	0.3	2.6
`fir` (N)	238	234	3.5	2.6	2.3	0.2	2.1	1.9	0.2	1.5
`fir` (H)	274	267	7.1	3.2	2.9	0.3	2.3	2.1	0.2	1.7
`g721-enc` (N)	7974	7970	4.2	79.4	72.8	6.5	1.9	1.8	0.1	1.2
`g721-enc` (H)	7484	7475	9.2	94.1	83.8	10.	2.5	2.2	0.2	1.7
`ieee810` (N)	4557	4424	132	71.7	57.8	13.8	3.1	2.5	0.6	2.0
`ieee810` (H)	5807	5797	9.5	106.4	89.5	16.8	3.6	3.0	0.5	2.7
`gsm-dec` (N)	1997	1979	18	19.8	17.9	1.9	1.9	1.7	0.1	1.2
`gsm-dec` (H)	6628	6607	21	78.1	68.4	9.7	2.3	2.0	0.2	1.5
`gsm-enc` (N)	10927	10903	23	112.2	94.8	17.4	2.0	1.7	0.3	1.1
`gsm-enc` (H)	4702	4653	49	59.0	51.1	7.8	2.5	2.1	0.3	1.6
`jpeg` (N)	849	789	59	9.7	8.2	1.4	2.2	1.9	0.3	1.3
`jpeg` (H)	810	749	61	9.6	8.0	1.5	2.3	2	0.3	1.4
`mpeg2-dec` (N)	6033	5837	195	98.6	78.2	20.3	3.2	2.5	0.6	2.1
`mpeg2-dec` (H)	6416	6082	334	90.0	72.9	17.0	2.8	2.2	0.5	1.7

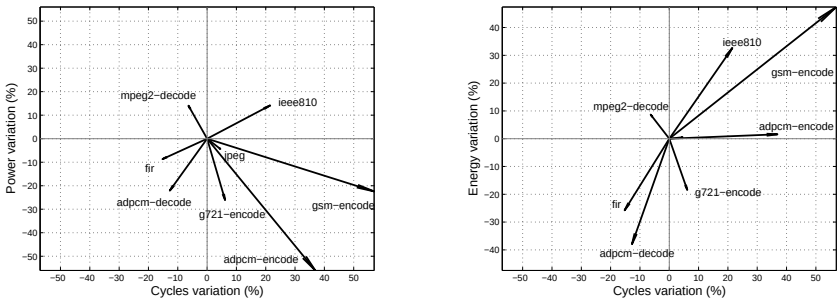
techniques that enhance ILP in object code were pioneered and developed for performance related purposes. Consequently, the rediscovery of VLIW-based architectures in systems where performance is not the only objective makes it necessary to reconsider the effects of ILP oriented compilation from a multi-objective perspective.

A first look of the implications of an ILP oriented compilation in a multi-objective embedded design will be discussed in the following subsections.

4.1 Heterogeneousity of ILP impact

We will start by evaluating the impact of ILP oriented compilation on the objectives (performance, power dissipation, energy consumption) for different applications running on a given fixed architectural configuration. In this first analysis we are not interested in exploration of the architectural parameters: We only want to investigate any "side effects" there may be on the objective magnitudes following an ILP oriented compilation strategy. We therefore set each parameter of the VLIW core and the memory hierarchy and evaluate the applications for each of the two compilation profiles *H* and *N*. In particular, the set of results given below refers to a configuration with a VLIW core comprising 4 integer units, 64 GPR registers, 64 FPR registers and 64 predicate registers. The memory subsystem comprises first-level instruction and data cache, both of 32K bytes, and 128-byte direct mapped blocks. The second-level cache is 256K bytes, with 256-byte blocks and an associativity of 4.

Table IV summarizes the results obtained. From the left, the table gives the values obtained for the three magnitudes: The total number of clock cycles (Clock cycles), the total energy consumption (Energy) and the average power dissipation (Power). Each of the objectives occupies three columns: Alongside the total value, the contributions made by the processor and hierarchy of memories are given separately. The last column on the



(a) (b)

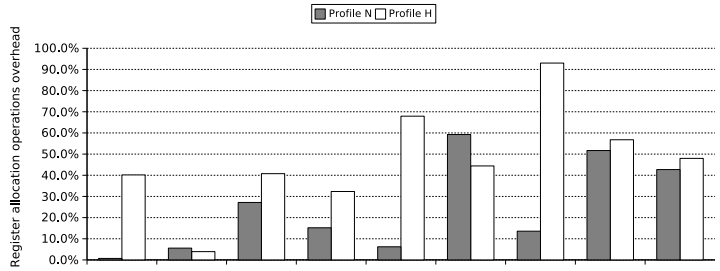
Fig. 2. Multi-Objective Representation Axes graphs for cycles/power (a), and cycles/energy (b).

right gives the average number of instructions executed per cycle (IPC), that represents a real measure of the ILP that the compiler actually achieves. This does not mean, however, that maximization of the ILP is in itself a design objective. For example, referring to the `adpcm-dec`, `fir`, and `gsm-dec` cases in Table IV, an increase in the IPC corresponds to an increase in the total number of execution cycles. The data relating to IPC do not, in fact, provide a reliable validity criterion for any of the performance/power/energy objectives the designer is interested in. We give the IPC values on account of the role that "IPC achieved" historically plays in describing the features of a VLIW architecture.

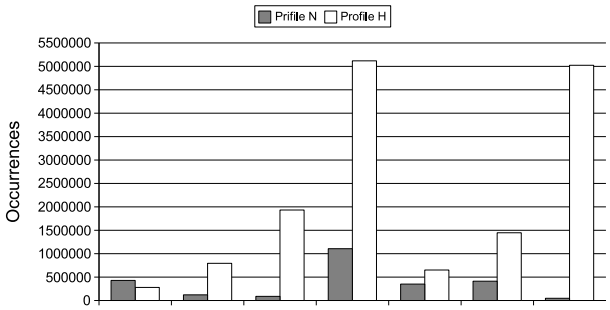
To show more intuitively the heterogeneous nature of the scenarios encountered in Table IV we introduce an alternative representation of the data, consisting of graphs we will refer to as *Multi-Objective Representation Axes* (MRA). Figures 2(a)-(b) show two examples of MRA, to be interpreted as follows. Each MRA refers to a pair of objectives. Each axis of an MRA represents the percent variation of an objective with respect to the value obtained in a previous reference scenario. For a certain application, assuming that switching from profile N to profile H caused an x% and y% variation in the values of the objectives, we plot a *variation vector* that goes from the origin of the axes to the coordinates (x,y). In this way the set of variation vectors makes it possible to represent and compare impact scenarios belonging to different applications in a single graph. The direction of the vectors obtained depend on the "type" of the impact, whereas their magnitude is determined by the % variation for each of the two objectives. For instance, the variation vector of `ieee810` application shown in Figure 2(a) an increase of about 20% in clock cycles and about 15% in power dissipation. In the remaining subsections we investigate each of the design objectives to give an explanation of the variety of impact scenarios encountered.

4.2 Effects on Performance

When considering the performance, that is, the total number of execution cycles, conventional wisdom would suggest that the more ILP is extracted from the application the more performance is gained. In other words, even if we cannot foresee how the chosen compilation profile will impact on other magnitudes such as power and energy, we would expect profile H as being more performance-oriented. However, the presence in both MRA graphs of vectors belonging to quadrants I and IV shows that the use of the ILP-oriented profile H does not necessarily cause an improvement in performance. In connection with this, it is



(a)



(b)

Fig. 3. Registers allocation overhead (a), and distribution of operations for the `gsm-dec` application (b).

useful to consider the total number of clock cycles as being made up of two components:

$$total\ clock\ cycles = CPU\ clock\ cycles + memory\ stall\ clock\ cycles, \quad (1)$$

where the first term is the number of cycles executed by the CPU hypothesizing an ideal memory hierarchy with a 100% hit rate, and the second is the total number of clock cycles due to memory stalls. Using profile H, the VLIW compiler can exploit a set of code transformations (e.g., if-conversion and speculative execution as described in Section 4) which "duplicate" portions of code. This means an increase, at times quite considerable, in the number of instructions to be executed, thus increasing the register pressure. If the processor has insufficient resources, the performance of the code produced will be poor and the first term in Equation (1) will increase considerably. An evident symptom of resource saturation is an increase in the special instructions for *save/restore* operations in registers. If the register allocator encounters a need to free one or more registers these special instructions make it possible to save the status of the register temporarily in the memory and restore it later on. They are, in fact, operations that from the semantic viewpoint that do not perform any "useful work" in the program being executed; instructions that could be avoided if there were an unlimited number of registers.

Figure 3(a) shows the overhead for these operations as compared with the number of useful instructions executed. With reference to `adpcm-dec` and `gsm-dec`, where there is

a considerable degradation in performance, the corresponding increase in the overhead is very high. Taking the `gsm-dec` case as an example, whereas in the first case it is necessary to add 5% more save/restore operations, in the second the amount is as high as 70%. With respect to `gsm-dec`, Figure 3(b) shows how the distribution of operations changes between the two compilation profiles. As can be observed, the most important contribution is due to the increase of integer operations.

Let us now examine the second term in Equation (1). The greater filling of the long instruction slots caused by the increase in ILP leads to an increase in pressure on the caches. Sending fuller instruction bundles means that in the same time interval there will be more instructions to retrieve from the instructions cache and more values to read and/or store in the data cache. Consequently, with the same cache memory, ILP-oriented compilation will lead to an increase in memory stalls. An undersized cache could therefore nullify any benefits of using the compilation profile H for the first term in Equation (1).

4.3 Effects on Power and Energy

Changing to another perspective, in a VLIW architecture the performance considerations made above cannot be used to draw conclusions regarding energy consumption. Irrespective of whether performance improves or deteriorates, the impact on energy consumption can be either positive or negative. Energy consumption is linked to the amount of "work carried out" by the system, but in a multiple issue architecture like VLIW executing fewer cycles does not mean less work[4]. A program could execute almost empty instruction bundles for several cycles, or several instructions in parallel for less time.

On the other side, the magnitude associated with the power objective is given by definition by the following relationship

$$average\ power = \frac{total\ energy}{total\ clock\ cycles \times clock\ period}. \tag{2}$$

From (2) it is clear that, given a certain amount of energy consumption or, alternatively, a certain number of clock cycles executed, no assumption can be made about the resulting average power dissipation. This reflects on the presence of variation vectors in all the quadrants in Figure 2. It should be pointed out, however, that the definition of power excludes the possibility of an application with variation vectors simultaneously in quadrants I(a) and IV(b), or III(a) and II(b).

In short, the results obtained show a variety of scenarios that seems to be tightly application-dependent. In the next section we analyze the effect that the compilation strategy has on the performance/power/energy when the whole design space is being considered.

5. STATISTICAL ANALYSIS

The main goal of this section is to show how a statistical analysis can be used to forecast the general characteristics of the Pareto set obtained by the overall exploration of the design space. A key aspect of this analysis is that it needs to evaluate only a very small set of configurations (in the order of tens), thus resulting in a negligible time overhead for the whole design space exploration phase. After introducing some useful concepts from the Design of Experiments theory [Montgomery 2000], we show how it can be used from both the single and the multi objective viewpoint.

[4]In general the same applies to all architectures that can achieve $IPC > 1$ (e.g., Superscalar).

5.1 Mean Effect Prediction

In Figures 2(a)-(b) we have shown variation vectors for each of the applications considered with a prefixed reference architecture configuration. The next step is to ask whether, given an application and a pair of objectives, there exists a "variation quadrant" that can be associated with it regardless of the architectural configuration being considered. If there were, it would represent something like a property of the application, indicative of the effect of a given compilation profile on each of the objectives.

Let us consider the configuration space shown in Table VII on page 21. An exhaustive evaluation of each compilation profile for each single configuration is obviously an unacceptable option. In this section we propose the use of a statistical approach based on *hypothesis testing* and *confidence interval* procedures. For hypothesis testing we will use an *Unpaired Two-Sample t-test* [Montgomery 2000] to verify *statistical hypotheses* regarding the effect of a given compilation profile on the design objectives. Statistical hypothesis is a conjecture made by the designer on the effects of a parameter of the system being investigated. In the case in point, what we are asking is whether, given a certain application and an objective, choosing a compilation profile has a significant effect on the average value of the objective calculated with respect to the whole configuration space. Referring to the compilation profiles introduced in Section 4, if we use μ_H and μ_N to indicate the *true mean* of the objective, this means verifying whether $|\mu_H - \mu_N| \geq \delta$, where δ represents a *critical difference* we consider to be significant. To know the *true mean* of an objective in the H and N scenarios, it is necessary to know the value taken by the objective in all the possible configurations, which is impossible. The t-test overcomes this difficulty as it is based on limited sampling of the configuration space.

Let us briefly describe the procedure followed for a given application and a certain objective. Given a configuration space $\mathscr{C}$, two randomly chosen subsets $\mathscr{C}_N$ and $\mathscr{C}_H$ both containing a number n of configurations are retrieved. The choice of sample size n is discussed in Subsection 5.2. For each configuration in $\mathscr{C}_N$ the value of the objective is evaluated by compiling using profile N. Likewise, each configuration in $\mathscr{C}_H$ is evaluated using profile H. At the end of this phase we obtain two sets of n elements which we indicate as O_N and O_H, comprising the values taken by the objective in the configurations of the subspaces $\mathscr{C}_N$ and $\mathscr{C}_H$. In our analysis O_N and O_H are thus two samples of the possible values the objective can take. For each of these two samples we can calculate various statistical properties such as mean, variance, etc. Obviously all these magnitudes will depend on the sample used and will only be a more or less reliable estimate of the statistical properties of the objective as compared with the whole of the configuration space.

The t-test procedure allows us to choose the number of configurations n in the subspaces $\mathscr{C}_N$ and $\mathscr{C}_H$ in such a way as to use the statistical properties of the samples O_N and O_H to verify conjectures regarding the true mean μ_H and μ_N of an objective. In particular, it allows us to verify the degree of admissibility of the following two hypotheses:

$$H_0 : \mu_N = \mu_H,$$
$$H_1 : \mu_N \neq \mu_H.$$

Hypotheses H_0 and H_1 are said to be a *null hypothesis* and an *alternative hypothesis*. As the t-test is based on the objective values obtained in two subspaces, $\mathscr{C}_N$ and $\mathscr{C}_H$, which are much smaller than the whole space $\mathscr{C}$, the veracity of the hypotheses will have to be confirmed together with a corresponding level of statistical reliability. This is represented

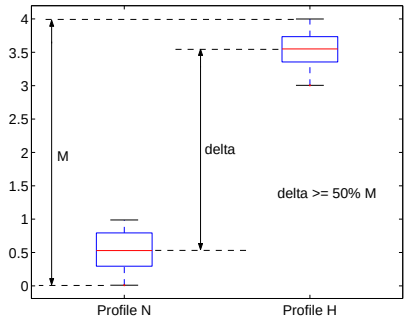


Fig. 4. Critical δ: A visual explanation.

by two error probabilities:

$$\alpha = P(\text{type I error}) = P(\text{reject } H_0 \mid H_0 \text{ is true}),$$
$$\beta = P(\text{type II error}) = P(\text{fail to reject } H_0 \mid H_0 \text{ is false})$$

In other words, α can be defined as the probability of identifying a difference in the true means, when such a difference does not really exists. On the other hand, β is the probability of *not* identifying a difference, when the difference exists. Intuitively, the greater the size of the samples O_N and O_H, the greater the capacity to identify increasingly smaller differences. That is, a specified difference in means is easier to detect for larger sample sizes than for smaller ones. A discussion of the values chosen in this work is addressed in Subsection 5.2. Therefore, having chosen the values of β and n, the final result of the t-test consists of confirming that hypothesis H_0 is accepted or rejected for a certain α value, also called the *significance level* of the test. Intuitively, as increasingly smaller α are used, the result of the test will be more and more prudent in rejecting H_0, that is, stating that the true means are different. A significant role is played by the smallest α value with which H_0 is rejected, called the P-value. A sufficiently small P-value (typically $< 10^{-3}$ [Montgomery 2000]) indicates that hypothesis H_0 is false, that is, a difference between the true means μ_H and μ_N that is comparable to the chosen critical difference δ.

5.2 Choice of the t-test parameters

In the tests performed, we set $\beta = 0.05$, a value commonly accepted in literature as suitably small. As regards the critical difference $\delta = |\mu_N - \mu_H|$, we set a value equal to 50% of M, where M is the maximum range of variation encountered considering the union of samples O_N and O_H. A visual example based on the boxplots is shown in Figure 4. This choice of δ is of course arbitrary, but it intuitively reflects a certain "distance" in the mean distribution of the samples. Preliminary tests performed using different choices for δ did not show a dependence of the results obtained respect to small changes of the δ values used. The value of δ basically "tunes" the sensitivity of the t-test: If the difference between the real means is comparable with the critical value chosen, we want to be sure of intercepting this difference with a high degree of probability. A commonly accepted way to choose the size of n is to use some graphs of β versus sample size n, called *operating characteristic curves* [Ferris et al. 1946]. Using them is possible to relate the desired β to

the minimum sample size needed, given a critical difference δ to be identified. Using the above mentioned operating characteristic curves, we found that a n value of 50 (that is a total 100 configurations for both profiles) was more than sufficient for the β value chosen and the δ values obtained using the criterion outlined above.

The *P-values* given in Table V show that in most cases hypothesis H_0 is rejected. Consequently, in those cases it can be stated with almost certainty that $\mu_N \neq \mu_H$. Only in two applications, `mpeg-dec` and `jpeg`, does there seem to be no average affect of the compilation profile. In these cases, the corresponding P-value is not sufficiently low to confirm the admittedly minimal difference in the confidence interval. For most of the remaining cases, where an effect on the true means has been identified, it is important to understand how significant the estimated $\mu_N - \mu_H$ difference is with respect to the previously defined critical threshold. In the table we give the δ values and the confidence interval estimated for $\mu_N - \mu_H$.

5.3 Mono-objective Analysis

Let us start from a mono-objective perspective. Given an application and an objective, the more the corresponding confidence interval of $|\mu_N - \mu_H|$ is of limited uncertainty and comprises values close to the critical difference δ, the greater the reliability of the hypothesis of a significant impact of the compilation profile on the objective being considered. In these cases – highlighted in bold in the table – we can therefore obtain indications as to whether one compilation profile is better than another for a certain objective. Once again, the type of effect varies considerably from one application to another (e.g., referring to the execution time, profile H seems to be a good choice for `gsm-enc` but not for `gsm-dec`). The remaining cases not highlighted in bold show values of $|\mu_N - \mu_H|$ which are more or less negligible with respect to δ. It should be pointed out again that this does not imply that the choice of a particular compilation profile has a negligible effect on the objectives. The performance/power/energy values obtained for each single architectural configuration can still be strongly influenced by the choice of the compilation profile. What is predicted to be mostly equivalent is the set of values that could be obtained if the whole configuration space was explored. In other words, similar values could be obtained in the two cases, but corresponding to different system configurations.

In short, for a given objective and a given application, if a profile is predicted to lead to better results it should be chosen, otherwise the choice is irrelevant. In both cases, from a mono-objective viewpoint, the analysis conducted allows the exclusion of the compilation profiles from the decisional parameters of the system.

Referring to the compilation/execution time reported in Table I, this may result in a drastic reduction in exploration time. For the sake of example let us consider the performance optimization of `gsm-dec` application and suppose that a mono-objective exploration of the design space requires S system configurations to be visited. The total exploration time is:

$$Exploration\ time = S \times (Evaluation\ time_{profileH} + Evaluation\ time_{profileN}).$$

By exploiting the information obtained from the statistical analysis we realize that it does not need to evaluate visited configurations with profile H since they, on average, perform worse as compared to configurations evaluated with profile N. The new exploration time is:

$$Exploration\ time_{new} = S \times Evaluation\ time_{profileN} + SA\ overhead.$$

Table V. t-test analysis and confidence intervals for the design objectives.

Application	Time (ms)			Power (W)			Energy (mJ)		
	P-value	δ	95% c.i. of $\mu_N - \mu_H$	P-value	δ	95% c.i. of $\mu_N - \mu_H$	P-value	δ	95% c.i. of $\mu_N - \mu_H$
adpcm-dec	**4.975e-10**	**10.47**	**-7.22 ± 3.43**	**4.336e-13**	**0.61**	**-0.5 ± 0.12**	**7.203e-11**	**35.12**	**-27.74 ± 7.3**
adpcm-enc	4.675e-11	15.8	8.17 ± 2.2	**< 2.2e-16**	**0.75**	**-0.89 ± 0.14**	1.520e-05	27.67	-8.56 ± 3.73
fir	**3.165e-10**	**0.41**	**-0.32 ± 0.08**	**1.375e-09**	**0.47**	**-0.30 ± 0.09**	**< 2.2e-16**	**0.84**	**-0.97 ± 0.12**
g721-enc	**4.018e-06**	**11.24**	**-8.41 ± 2.91**	**< 2.2e-16**	**0.46**	**-0.39 ± 0.08**	**< 2.2e-16**	**39.32**	**-32.4 ± 5.9**
gsm-dec	**< 2.2e-16**	**12.93**	**-23.83 ± 2.58**	**1.262e-07**	**0.32**	**-0.24 ± 0.09**	**< 2.2e-16**	35.76	**-56.6 ± 6.43**
gsm-enc	**< 2.2e-16**	**21.97**	**33.25 ± 4.79**	**9.61e-09**	**0.53**	**-0.48 ± 0.14**	**< 2.2e-16**	**47.57**	**55.84 ± 9.82**
ieee810	**1.295e-11**	**9.98**	**7.76 ± 1.84**	6.508e-06	0.98	0.38 ± 0.16	**< 2.2e-16**	**29.41**	**30.82 ± 4.55**
jpeg	0.0002289	2.44	-0.97 ± 0.51	0.1258	0.53	-0.07 ± 0.09	1.276e-05	5.83	-2.31 ± 1.01
mpegdec	0.03342	20.03	-5.28 ± 4.85	0.002332	0.53	0.25 ± 0.16	0.4829	37.5	-3.48 ± 9.88

Where *SA overhead* is the overhead due to the statistical analysis:

$$SA\ overhead = |\mathscr{C}_N| \times Evaluation\ time_{profileN} + |\mathscr{C}_H| \times Evaluation\ time_{profileH}.$$

Since $|\mathscr{C}_N| = |\mathscr{C}_H| \ll S$, the contribution of *SA overhead* is negligible.

We can calculate the overall speedup as the ratio between the old exploration time and the new exploration time:

$$Speedup = 1 + \frac{Evaluation\ time_{profileH}}{Evaluation\ time_{profileN}}. \quad (3)$$

The evaluation time for a single system configuration is the sum of compilation time and simulation time:

$$Evaluation\ time_{profileH} = Compilation\ time_{profileH} + Simulation\ time_{profileH}$$
$$Evaluation\ time_{profileN} = Compilation\ time_{profileN} + Simulation\ time_{profileN}.$$

In first approximation, the simulation time is not affected by the compilation profile being used:

$$T := Simulation\ time_{profileH} \equiv Simulation\ time_{profileN}.$$

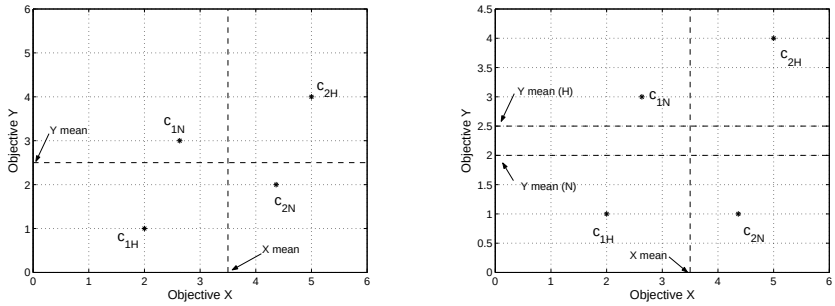
For `gsm-enc`, we have, $T = 14$ sec, $Compilation\ time_{profileN} = 813$ sec, $Compilation\ time_{profileH} = 2946$ sec. Substuiting them in (3) we obtain $Speedup = 4.6$.

5.4 Multi-objective Analysis

Let us now reconsider results of Table V from a multi-objective viewpoint. Given a pair of objectives X and Y (e.g., cycles/energy or cycles/power), we can enumerate four possibilities:

—*Case 1*: A discordant impact on the objectives.
—*Case 2*: No significant impact on the objectives.
—*Case 3*: An impact on only one of the objectives.
—*Case 4*: A concordant impact on both objectives.

Case 1 is a trivial one: Choosing a compilation profile would mean deteriorating the possibility of optimizing one of the two objectives. This clashes with the multi-objective optimization principle and so is to be excluded. In *Case 2* we could make an erroneous



(a) (b)

Fig. 5. Counterexamples: (a) No significant impact on the objectives. (b) An impact on only one of the objectives.

Application	Objectives	
	Ex.time/Power	Ex.time/Energy
adpcm-dec	Case 4	Case 4
adpcm-enc	Case 3	Case 2
fir	Case 4	Case 4
g721-enc	Case 4	Case 4
gsm-dec	Case 4	Case 4
gsm-enc	Case 1	Case 4
ieee810	Case 3	Case 4
jpeg	Case 2	Case 2
mpeg2-dec	Case 2	Case 2

Table VI. Classification of the applications based on the four possible cases for the pairs of objectives execution time/power and execution time/energy.

analogy with single objective analysis and deduce that the choice between profiles N and H is irrelevant. But this would mean excluding a number of potential tradeoffs between the pairs of objectives, as shown in the counterexample of Figure 5(a). Let us imagine that the space $\mathscr{C}$ comprises only two configurations c_1 and c_2, which in the two scenarios H and N (indicated by the subscripts) give the value of the objectives shown. By choosing profile N a priori we would forgo the possibilities offered by configuration c_1 using profile H, which is the best of the results obtainable. Although deliberately simplified, this counterexample shows that it is not possible to simply extend the considerations made regarding the single objectives to the multi-objective case. Likewise, in *Case 3* we might erroneously deduce that if for an objective X the impact is insignificant, while an effect of a certain type is observed for a second objective Y, it is convenient to follow what is suggested by the impact on objective Y. A counterexample of this is shown in Figure 5(b), which is simply a variation of *Case 2*, where the mean for objective Y in scenario N is shifted lower down.

The most interesting scenario is represented by *Case 4*, in which the mean effect is predicted to be significant for both objectives. In this case, as in the single-objective analysis, we can have an indication about the compilation profiles to be chosen. As reported in Table V, most of the predicted effects fall in this case, thus allowing to exploit the statistical analysis in order to reduce the exploration effort in terms of computation time.

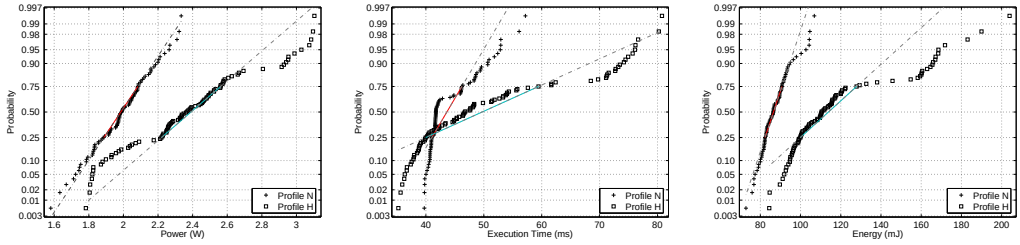


Fig. 6. Normal probability plot of the samples obtained for the `g721-enc` application for both *H* and *N* profiles.

Table VI classifies each application in terms of the case it belongs for the pairs of objectives execution time/power and execution time/energy. We observe that the most useful case (*Case 4*) is also the most frequent (44% for the pair Ex.time/Power, and 67% for the pair Ex.time/Energy).

The results obtained in this preliminary analysis will be the starting point for architectural exploration of the system as a whole, as discussed in Section 6.

5.5 Checking Assumption in the *t*-Test

The hypothesis underlying the comparison of two sample using the *t*-test procedure is that the populations they are taken from are normally distributed. Verification of this assumption by constructing *normal probability plots* shows that the samples approximate this hypothesis quite well and that the approximation does not vary significantly from one application to another. In some cases, the samples relating to energy and execution time exhibit a difference at the outermost points of the reference line (see Figure 6). These cases are, however, acceptable because the central part of the distribution, where almost all the points are concentrated, follows a regular trend. Having verified this assumption, the choice of which variation of the *t*-test to use depends on the way the samples are generated in the two scenarios *H* and *N*. In the tests carried out the comparison is made between two independent sets of configurations, i.e., obtained by repeating the random sampling operation for each of the two groups (*unpaired samples*). The data also showed great lack of homogeneity in the variance values for the samples being compared. The population distribution in scenario *H* always resulted in a slightly higher variance values than scenario *N*. For the reasons outlined above, the *Welch's* version for *t*-test was used in this work, as suggested in [Montgomery 2000]. However, it should be remarked that this is transparent to the *t*-test user, that is, the variant of *t*-test used does not reflect in any change respect to the approach presented in this work.

5.6 Extension to Multiple Compilation Profiles

The use of the statistical analysis introduced in the Subsection 5.1 for the comparison of two different compilation profiles can be easily extended to the case of multiple profiles. Different approaches can be adopted in order to define them, independently from the methodology presented in this work. A first although simplicistic solution could be a blind approach that randomly selects the set of parameters for each profile. One advantage of this solution is that no time overhead is involved in selecting the profiles to be investigated, since all the compilation parameters are considered as being of the same impact on the resulting executable code. On the other hand, using this approach does not allow to focus

```
1  PrifilesSet ProfilesSelection(ProfilesSet P = {p1, p2, ..., pN}) {
2      vector<SystemConfigurationSet> G = RndSampling(N, M);
3      CompileAndSimulate(G, P);
4      ProfilesSet S = {p1};
5      for (i = 2 to N)
6          for (q ∈ S)
7              StatisticalAnalysis(G, pi, q);
8              if (pi ≻ q)
9                  S = (S ⋃ {qi}) \ {q};
10             else if (q ⊁ pi)
11                 S = S ⋃ {q};
12     return S;
13 }
```

Fig. 7. Pseudo-code of the algorithm for selecting the subset of compilation profiles to be explored.

on the most influencing ones, and a great amount of the available exploration time could be wasted in evaluating profiles containing not relevant parameters . To overcome this issue, a sensitivity-based techniques such as [Fornaciari et al. 2002] could be adopted in order to select the most impacting parameters. For a system with n compilation parameters, determination of the degree of sensitivity of each parameter consists of fixing $n-1$ parameters and varying one of them, determining the maximum range of variation of the objective function. One way to fix the parameters is to consider the mean value of their variation set. More sophisticated solutions such as fractional factorial design [Montgomery 2000] would further help in exploiting the available budget of exploration time evaluating only the most promising profiles. Such techniques try to identify the first order interaction between parameters, so that not only the most impacting parameters are included in the profiles, but also the subset of the most representative combinations of them are identified. Finally, as a trivial but not always viable solution, an experienced designer could adopt an heuristic approach, based on his/her knowledge of the system and the meaning of the compilation parameters involved. As stated before, the choice if one these approaches is independent from the methodology presented in this work, therefore we can assume, without loss of generality, that a certain set of compilation profiles has been selected for statistical analysis.

Let $P = \{p_1, p_2, \ldots, p_N\}$ be the set of available compilation profiles. Given two compilation profiles $p, q \in P$ we say that p dominate q (indicated as $p \succ q$) if and only if the statistical analysis operated on p and q classifies the transition from p to q as belonging to the *Case 4*.

The pseudo-code of the algorithm for selecting the subset of compilation profiles to be explored is shown in Figure 7. As input, it requires the set of available compilation profiles. As output, the algorithm returns the set of compilation profiles which should be explored in accordance to the results of the statistical analysis. First, the function *RndSampling* performs a random sampling of the design space selecting N groups (N is the total number of compilation profiles) each containing M system configurations (where M is a user defined parameter). Then, the system configurations of each group are compiled by using the appropriate compilation profile (i.e., system configurations belonging to group number i are compiled by using compilation profile p_i) and simulated (function *CompileAndSimulate*).

Table VII. Configuration space used in the experiments.

Parameter	Parameter space
GPR	16, 32, 48, 64, 128
FPR/PR/CR	64
BTR	8, 12, 16
Integer Units	1, 2, 3, 4, 5, 6
Float Units	1, 2, 3, 4, 5, 6
Memory Units	1, 2, 3, 4
Branch Units	1, 2, 3, 4
L1D/I cache size	128B, 256B, …, 64KB, 128KB
L1D/I cache block size	32B, 64B, 128B
L1D/I cache associativity	1, 2, 4
L2U cache size	128KB, 256KB, 512KB
L2U cache block size	64B, 128B, 256B
L2U cache associativity	2, 4, 8, 16
Space size	2.8×10^9

The two nested loops operate a pair-wise dominance comparison as follows. First a set of profiles S is initialized with p_1. Then, for each compilation profile $p_i \in P$ and for each compilation profile q belonging to S the statistical analysis (as described in Section 5.3) is performed. If compilation profile p_i dominates q, p_i is inserted into S and q is removed. That is, the exploration of the system configurations compiled with q will statistically lead to poor solutions. Otherwise, if p_i does not dominate q as well as q does not dominate p_i, then q is inserted into S. That is, system configurations compiled with q are statistically equivalent to that compiled with p_i. Finally, the set S contains the compilation profiles to be used during the design space exploration.

6. DESIGN SPACE EXPLORATION

The statistical analysis presented in the previous section provides an important criteria to decide, on a case-by-case basis, whether it is possible to choose a compilation profile to be used in the design space exploration phase. The main goal of this section is to show how the results obtained in Section 5 reflect on the Pareto set distribution by the overall exploration of the design space. We will discuss an example for each of the four cases listed in Subsection 5.4. The design space considered is the same used in Section 5 and it is reported in Table VII. It should be pointed out that the *configure-and-execute* design paradigm [Vahid and Givargis 2001] is considered in this paper. In this context, a highly parameterized pre-designed platform is configured according to the application (or set of applications) it will have to execute. For this reason we do not explore the design space generated by the variation of the architecture which is considered being fixed. However, the applicability of the proposed approach is general and does not depend on the characteristics of the design space to be explored. If the designer wants to explore several architectural alternatives, he has to add a new parameter whose variation space is simply the enumeration of all the architectural alternatives. For the sake of example, let us suppose that the designer wants to expand the design space with system configurations in which both the first level cache memory and the second level cache memory can be either unified or separated. In this case, the parameter to be added will encode the following values: first level unified/second level unified, first level separated/second level unified, and first level separated/second level separated.

6.1 Exploration Algorithms

Various approaches have been proposed in the literature for efficient design space exploration in parameterized systems. The aim is to limit the number of configurations to be evaluated to obtain a good approximation of the Pareto optimal front. Among the most interesting we can mention the approach based on parameter dependence proposed by Givargis *et al.* [Givargis et al. 2002], the use of sensitivity analysis proposed by Fornaciari *et al.* [Fornaciari et al. 2002], the heuristic method based on evolutionary computing techniques proposed by Ascia *et al.* [Ascia et al. 2004], and the hybrid approaches proposed by Palesi *et al.* [Palesi and Givargis 2002; Ascia et al. 2002]. Although EPIC-Explorer implements all these exploration algorithms, the experiments were performed using the exploration algorithm, based on genetic algorithms [Ascia et al. 2004] which features a good tradeoff between accuracy and exploration time. All the Pareto sets reported initial population size of 30 individuals, a mutation probability 0.1, crossover probability 0.8, for a total of 50 generations.

6.2 Discussion

Let us start with the first of the four cases listed at the beginning of this section, that is, a discordant impact on design objectives. As an example, Figure 8 shows the Pareto fronts obtained for the `gsm-enc` application when the design objectives considered are power and performance. In this case the positive impact on performance predicted for profile *H* is evident. The improvement in performance is as much as 60% with only a 20% deterioration in dissipated power. It is interesting to note that the best performing configurations (to the right of the dashed line) all use more than 1 integer unit. An increase in the degree of parallelism of the integer units cannot, in fact, be fully exploited by a compilation profile that does not use hyperblock formation. Passing from configuration *A* to configuration *E*, we observe an increase in the number of integer units (up to a maximum of 6) and memory units (up to a maximum of 3). The first set of configurations (*A*), featuring an average consumption ranging between 2.5 W and 2.7 W, use 2 integer units and only 1 memory unit. Increasing the number of integer units to 3 (configurations *B*), there is no appreciable improvement in performance unless the number of memory units is also increased (configurations *C*). Finally, configurations *D* and *E* are equivalent from a performance viewpoint, but *D* is 6% more power-efficient. The decision maker could, however, choose *D* rather than *E* not only on account of the lower power consumption but also because of the lower cost of using only 3 integer units rather than 6.

As reported in Table V some of the results reported fall into *Case 2* (no significant impact on the objectives). Unfortunately, even in this case, statistical analysis is not useful to save exploration time and the design space should be explored including each compilation profile. For example, we show the Pareto set obtained for the `jpeg` application and power/performance objectives.

Figure 9 shows the Pareto fronts obtained for the two profiles *N* and *H*. With heavy power constraints (less than 1.7 W) it is possible to exploit the steepness of the Pareto front, which allows improvements in performance of over 40% without an appreciable deterioration in power consumption. In this case the increase in performance is only due to the increase in the number of GPRs, which goes from 16 (configuration *A*) to 32 (configuration *B*) to 128 (configuration *D*). Configuration *C*, which have 64 GPRs, are dominated by configuration *D*. In configurations *A* and *B* the availability of fewer registers puts more pres-

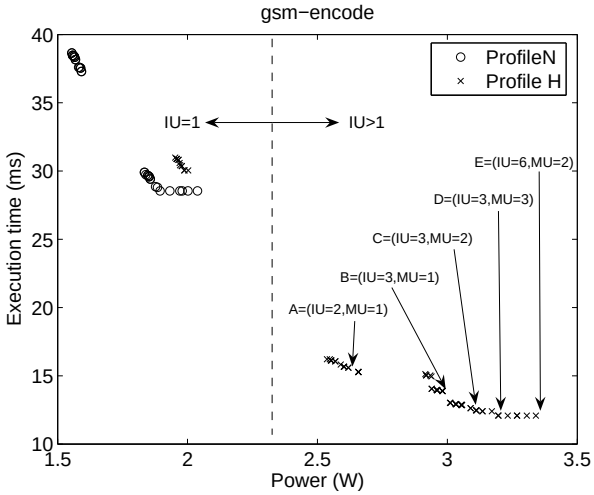


Fig. 8. Example of *Case 1*: Pareto front obtained from the exploration of the design space defined in Table VII for the application `gsm-enc`.

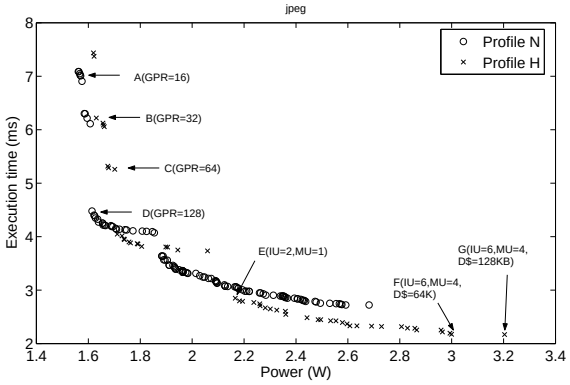


Fig. 9. Example of *Case 2*: Pareto front obtained from the exploration of the design space defined in Table VII for the application `jpeg`.

sure on the memory subsystem. The overall performance is, however, acceptable thanks to the presence of larger second-level caches than those used in configurations *C* (256 KB as compared with 128 KB). Increasing the power constraints beyond the threshold of 2.2 W, the configurations obtained by using profile *H* are dominant. The improvement in performance is, however, limited to 20% and the increase in power consumption is over 45%. In addition, in these configurations the number of integer units increases from 2 to 6, and the number of memory units goes from 1 to 4 as we move along the Pareto front from *E* to *G*. Finally, observing configurations *F* and *G*, which only differ in that *G* uses a 128 KB data cache as compared with the 64 KB used in *F*, no appreciable improvement in performance is observed but there is almost a 7% increase in power consumption.

Changing to *Case 3* (an impact on only one of the objectives) from Table V we found

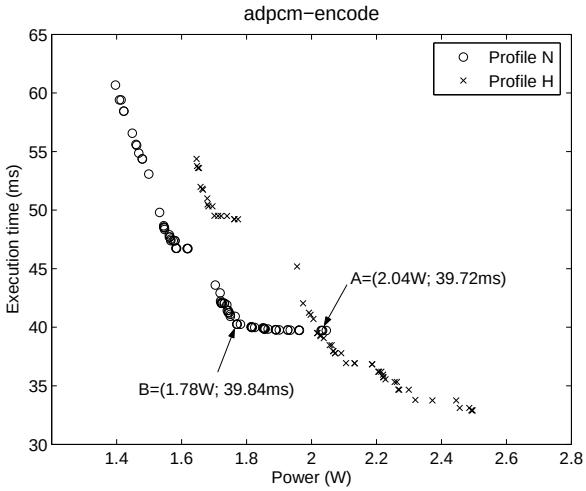
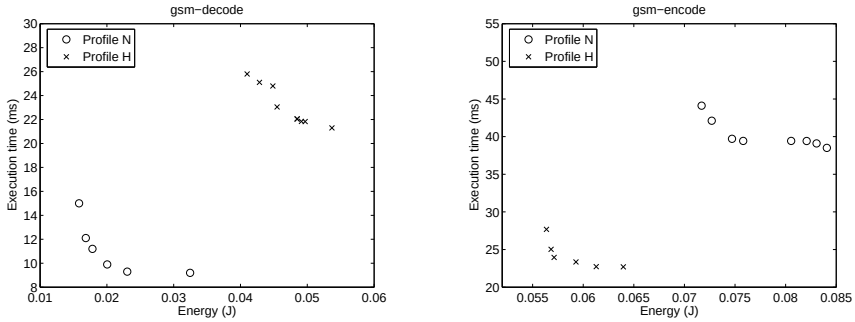


Fig. 10. Example of *Case 3*: Pareto front obtained from the exploration of the design space defined in Table VII for the application `adpcm-enc`.



(a) (b)

Fig. 11. Example of *Case 4*: Pareto front obtained from the exploration of the design space defined in Table VII for the applications `gsm-dec` (a), and `gsm-enc` (b).

that the application `adpcm-enc` falls into this possibility as regards to execution time and power dissipation. Power dissipation is the only objective which was predicted to be negatively impacted by profile *H*, since no mean effect was predicted for the other objective, that is execution time. But, as can be seen in Figure 10, excluding profile *H* would make impossible to reach the Pareto points below the configuration *A*. Once again it should be pointed out that the statistical prediction refers to the distribution of the objectives considering the whole design space. Since we cannot know a priori which points will contribute to the final Pareto set, an indecision about the effect on a time objective does not allow to exclude compilation profile *H*.

Concluding this section, let us now analyse the most common and important of the cases listed at the beginning of the section, that is *Case 4* (a concordant impact on both objectives). For example, from Table V we observe that `gsm-enc` and `gsm-dec` fall into

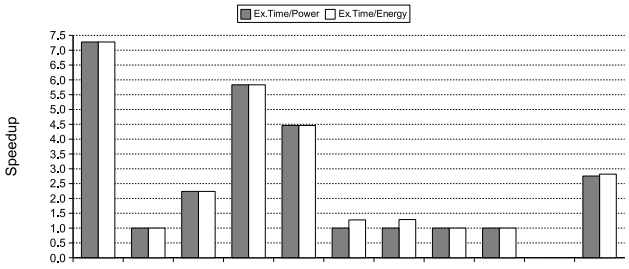


Fig. 12. Speedup in exploration time of the proposed approach as compared to the blind exploration of both the profiles *H* and *N*.

Table VIII. Actual design exploration time (in hours) for a blind approach and for the proposed approach (SA).

Application	Ex.Time/Power		Ex.Time/Energy	
	Blind	SA	Blind	SA
`adpcm-dec`	8	1	6	1
`adpcm-enc`	8	8.2	6	6.2
`fir`	4	2	4	2
`g721-enc`	14	2	11	2
`gsm-dec`	192	43	163	37
`gsm-enc`	204	210	235	184
`ieee810`	14	14.5	13	10
`jpeg`	13	13.4	12	12.4
`mpeg2-dec`	67	69	54	55.7

this possibility when time/energy objectives are considered. For the first one, profile *H* results in an improvement on both objectives. For the second one, the predicted effect is the opposite. This prediction is confirmed by the exploration results reported in Figures 11(a)-(b) which show the Pareto fronts, execution time/energy consumption for both compilation profiles. For `gsm-enc` (`gsm-dec`), the Pareto front obtained using profile *H* dominate (is dominated by) the Pareto front obtained using profile *N*.

Figure 12 summarizes the speedup obtained by performing a statistical analysis phase to predict whether the exploration of a particular scenario will or will not contribute with Pareto optimal solutions. For the pairs of objectives execution time/power and execution time/energy, it shows the speedup in exploration time of the proposed approach as compared to the blind exploration of both the profiles *H* and *N*. To get an idea of the computational effort needed to perform the design space exploration, Table VIII reports the actual design exploration time (in hours) for a blind approach and for the proposed approach (SA). For some applications the use of statistical analysis does not translate in any performance gain, but increases the overall exploration time due to its overhead. However, when the statistical analysis classifies the application as belonging to *Case 4*, the saving in exploration time is quite notable.

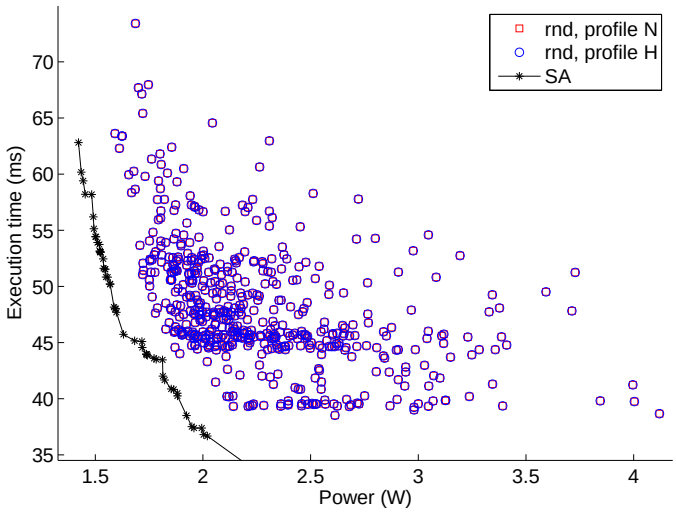


Fig. 13. Pareto-front obtained by the proposed design space exploration technique used in this paper along with 1,054 randomly system configurations for `g721-enc` application.

6.3 Assessment of the Exploration Strategy

To assess the quality of the exploration strategy used in this paper, we compare the results obtained with that obtained if a pseudo-random sampling of the design space is carried out. For the sake of example, let us consider the `g721-enc` application. By using the proposed design space exploration strategy a total of 1,054 system configurations are visited. The resulting Pareto front along with the results of 1,054 randomly generated system configurations are shown in Figure 13. Each randomly generated system configuration is evaluated by using both profile *N* and profile *H*. As can be observed, all the randomly visited system configurations are dominated by the solutions found by the proposed approach.

More in general, let *n* be the number of system configurations visited by using the proposed approach, and let $\mathscr{P}$ be the Pareto set obtained by the proposed approach. Let $\mathscr{P}'$ be the Pareto set extracted from *n* randomly generated system configurations. The Pareto set $\mathscr{P}$ and $\mathscr{P}'$ are compared by means of a set of quality measures as suggested in [Knowles et al. 2006]. The quality measures we considered most suitable for our context are as follows.

(1) *Hypervolume* [While et al. 2006]. It measures the hypervolume of that portion of the objective space that is weakly dominated by the Pareto set to be evaluated. In order to measure this index the objective space must be bounded. If it is not, then a bounding reference point that is (at least weakly) dominated by all points should be used. In this work we define as bounding point the one which has coordinates in the objective space equal to the highest values obtained. Higher quality corresponds to smaller values.

(2) *Pareto Dominance*. Is the percentage of the points of $\mathscr{P}'$ which are dominate by at least on point of $\mathscr{P}'$.

(3) *Distance*. This index summarizes how much two Pareto fronts are close each other.

Table IX. Quality measures for different applications.

Application	Hypervolume		Pareto	Distance	
	SA	RND	Dominance	$dist_{avg}$	$dist_{max}$
adpcm-dec	79	122	100%	0.07	0.19
adpcm-enc	81	139	100%	0.03	0.09
fir	77	162	100%	0.08	0.13
g721-enc	58	87	100%	0.04	0.09
gsm-dec	63	119	100%	0.05	0.08
gsm-enc	77	98	100%	0.03	0.09
ieee810	21	33	100%	0.04	0.09
jpeg	9	16	100%	0.04	0.08
mpeg2-dec	66	83	100%	0.03	0.11
		Average	100%	0.05	0.11

We define the average and maximum distance index as follows:

$$dist_{avg} = \frac{1}{|\mathscr{P}| \times D} \sum_{x \in \mathscr{P}} \min_{y \in \mathscr{P}'} (d(x,y))$$

$$dist_{max} = \frac{1}{D} \max_{x \in \mathscr{P}} (\min_{y \in \mathscr{P}'} (d(x,y)))$$

where $d(\cdot,\cdot)$ is the Euclidean distance and D is a measure of the extension of the Pareto front (i.e., for the two-objective case, it is the diagonal of the smallest area rectangle containing both $\mathscr{P}$ and $\mathscr{P}'$).

For each application, Table IX reports the values of the quality indexes described above. As can be observed, for all the considered applications, the solutions found by the proposed approach dominate the best solutions found by randomly sampling the design space. On average, the percentage distance between the two Pareto front is 5% with a maximum distance of 11%. These results confirm the reliability of the proposed approach.

7. CONCLUSIONS

In this paper we proposed tools and methodologies to cope efficiently the design VLIW-based embedded systems. We presented EPIC-Explorer, an open platform targeted for parameterized *EPIC/VLIW* architectures that provides high level estimation estimation models and several multi-objective design space exploration strategies. We first analyzed the impact on performance, power dissipation, and energy consumption due to the interaction between hardware-related aspects (namely the architectural parameters) and software-related issues (namely the compilation profiles). We used statistical analysis techniques based on Design of Experiments theory, with the attempt to draw some guidelines that help the designer in the process of pruning the design space to be explored. Finally, for a set of representative multimedia and communication applications, we applied a multi-objective design space exploration strategy based on genetic algorithms to study the power/performance and energy/performance tradeoffs.

Future developments will mainly address the analysis of more complex system architectures based on multiprocessors systems-on-chip where the communication between processors is realized through the use of an on-chip network infrastructure (network-on-chip). In this architectural scenario, we would like to investigate on data packetization, network

architectures and routing algorithms. Another important research direction will be related to the problem of selecting the right compilation profiles to be explored. As the selection of the compilation profile is intimately tied to the specific application and target architecture, the way in which different optimizations are configured is vital for design optimization purposes.

REFERENCES

ABRAHAM, S. G. AND MAHLKE, S. A. 1999. Automatic and efficient evaluation of memory hierarchies for embedded systems. In *ACM/IEEE International Symposium on Microarchitecture*. 114–125.

ASCIA, G., CATANIA, V., AND PALESI, M. 2002. Tuning methodologies for parameterized systems design. In *System on Chip for Realtime Systems*, K. A. Publisher, Ed.

ASCIA, G., CATANIA, V., AND PALESI, M. 2004. A GA based design space exploration framework for parameterized system-on-a-chip platforms. *IEEE Transactions on Evolutionary Computation 8,* 4 (Aug.), 329–346.

ASCIA, G., CATANIA, V., AND PALESI, M. 2005. A multi-objective genetic approach for system-level exploration in parameterized systems-on-a-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24,* 4 (Apr.), 635–645.

ASCIA, G., CATANIA, V., PALESI, M., AND PATTI, D. 2003. 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, 65–72.

BALASUBRAMONIAN, R., ALBONESI, D., BUYUKTOSUNOGLU, A., AND DWARKADAS, S. 2000. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *ACM/IEEE International Symposium on Microarchitecture*. 245–257.

BENINI, L., BRUNI, D., CHINOSI, M., SILVANO, C., ZACCARIA, V., AND ZAFALON, R. 2002. A framework for modeling and estimating the energy dissipation of VLIW-based embedded systems. *Design Automation for Embedded Systems 7,* 3 (Oct.), 183–203.

CAI, G. AND LIM, C. H. 1999. Architectural level power/performance optimization and dynamic power estimation. In *Cool Chips Tutorial colocated with MICRO32*. 90–113.

CAPITANIO, A., DUTT, N., AND NICOLAU, A. 1992. Partitioned register files for VLIWs: a preliminary analysis of tradeoffs. *SIGMICRO Newsletter 23,* 1-2, 292–300.

FERRIS, C. L., GRUBBS, F. E., AND WEAVER, C. L. 1946. Operating characteristics for the common statistical test of significance. *Annals of Mathematical Statistics*.

FISCHER, D., TEICH, J., THIES, M., AND WEPER, R. 2002. Efficient architecture/compiler co-exploration for ASIPs. In *Compilers, Architectures, and Synthesis for Embedded Systems*. Grenoble, France, 27–34.

FISHER, J. A. 1983. Very long instruction word architectures and the ELI512. In *Tenth Annual International Symposium on Computer Architecture*. 140–150.

FISHER, J. A., FARABOSCHI, P., AND DESOLI, G. 1996. Custom-fit processors: Letting applications define architectures. Tech. Rep. HPL-96-144, HP Laboratories. Oct.

FORNACIARI, W., SCIUTO, D., SILVANO, C., AND ZACCARIA, V. 2002. A sensitivity-based design space exploration methodology for embedded systems. *Design Automation for Embedded Systems 7*, 7–33.

GIVARGIS, T., VAHID, F., AND HENKEL, J. 2002. System-level exploration for Pareto-optimal configurations in parameterized System-on-a-Chip. *IEEE Transactions on Very Large Scale Integration Systems 10,* 2 (Aug.), 416–422.

GYLLENHAAL, J. 1994. A machine description language for compilation. M.S. thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana IL.

HEKSTRA, G., HEI, D. L., BINGLEY, P., AND SIJSTERMANS, F. 1999. TriMedia CPU64 design space exploration. In *International Conference on Computer Design*. Austin Texas, 599–606.

HENKEL, J. AND LEKATSAS, H. 2001. a^2bc: Adaptive address bus coding for low power deep submicron designs. In *Design Automation Conference*. Las Vegas, Nevada, USA, 744–749.

HENNESSY, J. L. AND PATTERSON, D. A. 2006. *Computer Architecture A Quantitative Approach*, Fourth ed. Morgan Kaufmann.

KAMBLE, M. B. AND GHOSE, K. 1997. Analytical energy dissipation models for low power caches. In *IEEE International Symposium on Low Power Electronics and Design*. 143–148.

KATHAIL, V., ADITYA, S., SCHREIBER, R., RAU, B. R., CRONQUIST, D. C., AND SIVARAMAN, M. 2002. PICO: Automatically designing custom computers. *IEEE Computer 35*, 9 (Sept.), 39–47.

KATHAIL, V., SCHLANSKER, M. S., AND RAU, B. R. 2000. HPL-PD architecture specification: Version 1.0. Tech. rep., Compiler and Architecture Research HP Laboratories Palo Alto HPL-93-80.

KIM, H. S., VIJAYKRISHNAN, N., KANDEMIR, M. T., AND IRWIN, M. J. 2001. A framework for energy estimation of VLIW architecture. In *International Conference on Computer Design*. Austin, TX, USA, 40–45.

KNOWLES, J., THIELE, L., AND ZITZLER, E. 2006. A tutorial on the performance assessment of stochastive multiobjective optimizers. Tech. Rep. 214, Computer Engineering and Networks Laboratory, ETH Zurich. Feb.

LAPINSKII, V., JACOME, M., AND VECIANA, G. D. 2002. Application-specific clustered VLIW datapaths: early exploration on a parameterized design space. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21*, 8 (Aug.), 889–903.

LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*.

MAHLKE, S. A., LIN, D. C., CHEN, W. Y., HANK, R. E., AND BRINGMANN, R. A. 1992. Effective compiler support for predicated execution using the hyperblock. In *International Symposium on Microarchitecture*. 45–54.

MASSELOS, K., CATTHOOR, F., GOUTIS, C. E., AND DEMAN, H. 1999. Low power mapping of video processing applications on VLIW multimedia processors. In *IEEE Alessandro Volta Memorial International Workshop on Low Power Design*. Como, Italy.

MIDDHA, B., GANGWAR, A., KUMAR, A., BALAKRISHNAN, M., AND IENNE, P. 2002. A trimaran based framework for exploring the design space of VLIW ASIPs with coarse grain functional units. In *15th International Symposium on System Synthesis*. Kyoto, Japan, 2–7.

MONTGOMERY, D. C. 2000. *Design and Analysis of Experiments*, 5th ed. Wiley Text Books.

MORGAN, P., TAYLOR, R., HOSSELL, J., BRUCE, G., AND O'ROURKE, B. 2005. Automated data cache placement for embedded VLIW ASIPs. In *IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis*. 39–44.

PALESI, M. AND GIVARGIS, T. 2002. Multi-objective design space exploration using genetic algorithms. In *Tenth International Symposium on Hardware/Software Codesign*. Stanley Hotel, Estes Park, Colorado, USA.

POKAM, G. AND BODIN, F. 2004. Understanding the energy-delay tradeoff of ILP-based compilation techniques on a VLIW architecture. In *11th Workshop on Compilers for Parallel Computers*. Chiemsee, Germany.

RAGHAVAN, P., LAMBRECHTS, A., JAYAPALA, M., CATTHOOR, F., AND VERKEST, D. 2006. Distributed loop controller architecture for multi-threading in uni-threaded VLIW processors. In *Conference on Design, Automation and Test in Europe*. 339–344.

SCHLANSKER, M. S., RAU, B. R., MAHLKE, S., KATHAIL, V., JOHNSON, R., ANIK, S., AND ABRAHAM, S. G. 1996. Achieving high levels of instruction-level parallelism with reduced hardware complexity. Tech. Rep. HPL-96-120, HP Laboratories. Feb. 28.

SHIUE, W.-T. AND CHAKRABARTI, C. 1999. Memory exploration for low power, embedded systems. In *36th ACM/IEEE Conference on Design Automation Conference*. New Orleans, Louisiana, United States, 140–145.

SRINIVASAN, V., BROOKS, D., GSCHWIND, M., BOSE, P., ZYUBAN, V. V., STRENSKI, P. N., AND EMMA, P. G. 2002. Optimizing pipelines for power and performance. In *MICRO*. 333–344.

Trimaran. An infrastructure for research in instruction-level parallelism. `http://www.trimaran.org/`.

VAHID, F. AND GIVARGIS, T. 2001. Platform tuning for embedded systems design. *IEEE Computer 34*, 3 (Mar.), 112–114.

VENKATACHALAM, V. AND FRANZ, M. 2005. Power reduction techniques for microprocessor systems. *ACM Computing Surveys 37*, 3, 195–237.

WHILE, L., HINGSTON, P., BARONE, L., AND HUBAND, S. 2006. A faster algorithm for calculating hypervolume. *IEEE Transactions on Evolutionary Computation 10*, 1 (Feb.), 29–38.

WSTS 2006. World semiconductor trade statistics bluebook. `http://www.wsts.org/`.