

# Multi-Objective Co-Exploration of Source Code Transformations and Design Space Architectures for Low-Power Embedded Systems

Giovanni Agosta   Gianluca Palermo   Cristina Silvano

Politecnico di Milano, Milano, Italy  
 {agosta,gpalermo,silvano}@elet.polimi.it

## ABSTRACT

The exploration of the architectural design space in terms of energy and performance is of mainly importance for a broad range of embedded platforms based on the System-On-Chip approach. This paper proposes a methodology for the co-exploration of the design space composed of architectural parameters and source program transformations. A heuristic technique based on Pareto Simulated Annealing (PSA) has been used to efficiently span the multi-objective co-design space composed of the product of the parameters related to the selected program transformations and the configurable architecture. The analysis of the proposed framework has been carried out for a parameterized superscalar architecture executing a selected set of benchmarks. The reported results show the effectiveness of the proposed co-exploration with respect to the independent exploration of the transformation and architectural spaces to efficiently derive approximate Pareto curves.

## Keywords

Hardware/Software Co-Exploration, Source Code Transformations, Low-Power Design, Embedded Systems

## 1. INTRODUCTION

Evaluation of energy-delay metrics at the system-level is of mainly importance for embedded applications characterized by low-power and high-performance requirements. The growing spread of *System-On-Chip (SOC)* embedded applications based on the platform-based design approach requires a flexible tuning framework to assist the phase of *Design Space Exploration (DSE)*. The overall goal of the *DSE* phase is to optimally configure the parameterized *SOC* platform in terms of both energy and performance requirements depending on the given application.

Only recently, some approaches ([1], [2], [3]) have analyzed the *concurrent* exploration of the compilation space

and the architectural design space from the energy/delay combined perspective. Our work is complementary to these recent studies, since we are focusing on the problem of the *Hardware-Software Co-exploration* from the power and performance concurrent perspective.

From the *hardware* side, we explore the architecture design space for parameterized microprocessor-based systems. The goal of this phase is to find the best architecture mainly in terms of the parameterized core processor, degree of *Instruction Level Parallelism (ILP)*, number of levels in the memory hierarchy, cache-related parameters, system-level bus topology, width of address and data buses, etc. From the *software* side, we explore the space of source program transformations for embedded applications. The goal of this phase is to find the best set of ordered program optimization passes (such as function inlining, loop unrolling, loop blocking, etc.) by estimating their effects on performance and energy.

This paper proposes a modular and flexible framework to jointly explore the set of all *HW/SW* configurations of the *SOC* platform defined in the combined *System-Level Design Space*, defined as:  $S_{SD} = S_T \times S_A$  where  $S_T$  is the space of program transformations and  $S_A$  is the space of architectural parameters. In other approaches previously published in literature ([1], [2]), the source code is first optimized independently of the target architecture, then the architecture is explored. In our approach, the exploration phase spans concurrently the transformation and the architectural spaces. In the proposed design flow, the results of the co-exploration in terms of evaluation metrics represent a feedback for the *Architectural Space Exploration (ASE)* module as well as for the *Transformation Space Exploration (TSE)* module.

Since the *System-Level Design Space* is usually very large, a multi-objective exploration approach based on the full search exploration is computationally infeasible, due to the long simulation time required to explore the wide space of parameters. The goal is to efficiently explore the multi-objective design space to find a good *approximation* of Pareto curves, representing the best compromise between the interesting design objectives (in our case, energy and delay). In this paper, to derive the Pareto curves for the multi-objective exploration, a heuristic technique (namely, *Pareto Simulated Annealing* [4]) has been used, that is an evolution of the well-known Simulated Annealing approach.

The proposed co-exploration methodology has been applied to a parameterized superscalar *SOC* architecture dur-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'04, March 14-17, 2004, Nicosia, Cyprus  
 Copyright 2004 ACM 1-58113-812-1/03/04 ...\$5.00.

ing the execution of a set of industrial benchmarks to find the best trade-off between the interesting design objectives, mainly energy and delay. In our experiments, *Loop Unrolling* and *Function Inlining* have been chosen among the possible source-level optimization passes. The set of tunable hardware parameters is mainly related to the memory hierarchy (in terms of cache size, block size, and associativity), number of processor functional units, and processor parallel issues. Each component of the target architecture is provided with the corresponding system-level executable model to dynamically profile the given application to derive the information related to power and performance metrics. From the experimental results, we verified that the co-exploration represents a more powerful approach with respect to the independent exploration to find energy/delay Pareto-curves. Furthermore, our analysis demonstrates that the choice of the optimal set of source-level transformations strongly depends on the architectural characteristics of target processor and memory subsystem.

The paper is organized as follows. A review of the most significant works appeared in literature concerning the design space exploration problem is reported in Section 2. Section 3 describes the *System-Level Design Space*, while the proposed co-exploration framework is described in Section 4. Section 5 discusses the experimental results carried out to evaluate the effectiveness of the proposed framework for a superscalar target architecture. Finally, some concluding remarks and possible evolutions of this work have been outlined in Section 6.

## 2. BACKGROUND

The overall energy of embedded *SOC* platforms depends on both software and hardware sides. From the software side, several low-power software design techniques have been recently proposed in literature [5], [6]. Instruction-level optimization techniques [7] can be automated by applying them in the back-end of the compiler, however they impact on energy is quite limited and they are strongly related to the target architecture. Other promising techniques are based on source code transformations [8], [9], [10]. One of the most critical issues afforded in these works is the estimation of the energy saved for each transformation at the highest level of abstraction. A coarse-grained static high-level code metrics is code compactness, while more accurate metrics are based on code profiling. Other system-level estimation and exploration methods have been recently proposed in literature targeting power-performance tradeoffs from the system-level standpoint [11], [12], [13], [14], [15], [16].

The *SimpleScalar* toolset [12] is based on a set of *MIPS*-based architectural simulators focusing on different abstraction levels to evaluate the effects of some high-level algorithmic, architectural and compilation trade-offs. The *SimpleScalar* framework provides the basic simulation-based infrastructure to explore both processor architectures and memory subsystems. However, *SimpleScalar* does not support power analysis. Based on the *SimpleScalar* simulators, *SimplePower* [15] can be considered one of the first efforts to evaluate the different contributions to the energy budget at the system-level. More recently, the *Wattch* architectural-level framework has been proposed in [13] to analyze power with respect to performance tradeoffs with a good level of accuracy with respect to lower-level estimation approaches. *Wattch* provides a framework to explore different system

configurations and optimization strategies to save power, in particular focusing on processor and memory subsystems.

The *Avalanche* framework [8] evaluates simultaneously the energy-performance tradeoffs for software, memory and hardware for embedded systems. The work in [11] proposes a system-level technique to find low-power high performance superscalar processors tailored to specific user applications. Low-power design optimization techniques for high performance processors have been investigated in [14] from the architectural and compiler standpoints. A trade-off analysis of power performance effects of *SOC* architectures has been presented in [16], where the authors propose a simulation-based approach to configure the parameters related to the caches and the buses.

Recently some approaches have been introduced to approximate Pareto-curves for computer architecture design [17], [18], [19]. *Platune* [17] is an optimization framework that exploits the concept of parameter independence to individuate approximate Pareto curves without performing the exhaustive search over the whole design space. More recently, Palesi et al. [18] extended *Platune* by applying genetic algorithms to optimize dependent parameters, resorting to the default *Platune* policy when independent parameters are specified by the user.

Most of the research works in energy optimization and/or estimation have focused on the hardware modules composing the system and have not yet analyzed the interaction between the hardware and software sides of the system. Only recently, some approaches ([1], [2], [3]) have analyzed the *concurrent* exploration of the compilation space and the architectural design space from the energy point of view.

The energy estimation framework in [1], [2] supports both hardware and software optimizations. The framework is based on *SimpleScalar* and *SimplePower* toolsets, where a high-level optimization module has been added to implement source-code optimizations. The main goal of this work is to estimate the effects of performance-oriented compiler transformations on the overall system (processor core plus memory subsystem) and in particular on each module of the system. The focus on the work described in [1], [2] is on energy estimation and optimization rather than the investigation of energy-delay trade-offs.

A co-exploration approach for *ASIPs* has been recently proposed in [3] where the authors span the space of processor architecture parameters as well as of different compiler optimization strategies. The objective space of this work is multi-dimensional including hardware cost, execution time and code size, however the energy cost has not been considered. A technique to prune the large search space to reduce the exploration phase has also been proposed in [3].

The present approach is complementary to these recent studies focusing on compiler/architectural energy optimization and estimation.

## 3. SYSTEM-LEVEL DESIGN SPACE

When considering a customizable *SOC* platform, the System Level Design Space is composed of the set of all the feasible software/hardware configurations of the platform defined as the product of the Transformation Space and Architectural Space:

$$S_{SD} = S_T \times S_A$$

In our approach, the  $S_T$  and  $S_A$  spaces are not spanned

independently, but rather jointly.

The *Transformation Space*  $S_T$  models the effects of a set of program transformations on the source code. The points of  $S_T$  correspond to different applications of the optimization passes to the same source program. The most important optimization passes [20] can be classified in three categories: dataflow passes, passes that simplify the control and enlarge the code, and passes that modify the access pattern to data.

Dataflow passes, such as *CSE*, *DCE*, *Constant Propagation*, *Constant Folding* and *Copy Propagation* are usually beneficial and independent of the architecture. In our framework, we always add a standard dataflow pass (as performed by `gcc`) at the end of all transformation processes.

*Function Inlining*, *Loop Unrolling*, *Code Specialization* are code-size affecting passes, providing an explicit trade-off between performances and code growth, directly controlled by the transformation depth, a factor that depends on the specific transformation. This parameter is the unroll factor for the *Loop Unrolling*, the number of inline points for *Function Inlining*, and a combination of the number of specialization points and the number of different specialized versions of the same code for *Code Specialization*. These transformations have been included in our architectural-compilation exploration, since their effects impact the energy and they strongly depend on the architectural parameters.

In particular, *Function Inlining* provides an immediate benefit when the overhead is comparable to the execution time. However, this beneficial effect is usually limited, unless the call point is located within a frequently executed loop body. As in other code-growing transformations, the main limitation is the need to replicate code sections. With respect to data and code locality, this transformation can modify the size of basic blocks, affecting the layout of code in the instruction cache. There is no direct effect on data locality, but additional transformations, like dataflow optimizations, may take advantage of new opportunities arising from the merger of basic blocks.

*Loop Unrolling* operates by replicating the loop body  $k$  times, where  $k$  is known as the *unroll factor*, and adjusting the loop index update operations so that the number of iterations decreases by a  $k$  factor. Additional recovery code is inserted to guarantee that loop runs that involve a number of iterations  $i$  that is not a multiple of  $k$  are completed by executing first the unrolled body  $i/k$  times, and then the original body  $i \bmod k$  times. The most interesting property of *Loop Unrolling* is its beneficial effect on data cache locality, however *Loop Unrolling* causes a considerable increase in code size, roughly equal to  $k$  times the size of the original loop body.

Access patterns-affecting passes (such as *Loop Tiling*, *Loop Skewing*, and *Modulo Scheduling*) fully depend of the target machine parameters, so their effects must be modelled within the optimization framework. Parameters for these passes are the access patterns, such as the size and shape of the tiles in the *Loop Tiling* transformation.

A transformation point is any point in a program where a transformation can be applied. An instance of a transformation  $t$ , parameterized by a set of  $k$  parameters, is composed of  $t$  and a value for each of its  $k$  parameters:  $i(t) = \langle t, a_1 \dots a_k \rangle$ . For example, if  $t$  is *Loop Unrolling*, an instance of  $t$  is  $\langle t, 2 \rangle$ , which means *Loop Unrolling* with unroll factor equal to 2.

Let  $P$  be the source program, composed of  $n$  transforma-

tion points  $p_1 \dots p_n$ :  $P = \{p_1 \dots p_n\}$ . Let  $T$  be a set of  $m$  instances of transformations  $i_1 \dots i_m$ . For the sake of simplicity, we assume that all transformation instances  $i_j$  can be applied to all  $p_i$ . Where this is not possible,  $i_j(p_i) = \perp$ , we redefine the transformation instance as the identity transformation,  $i_j(p_i) = p_i$ . We define the Transformation Space  $S_T$  as the power set of  $T \times P$ .

We can easily extend this definition to the case of limited application points for each transformation, by removing all sets that include an illegal  $\langle t_i, p_j \rangle$  pair. Obviously,  $|S_T|$  is  $O(2^{|T| \cdot |P|})$ , where  $|T|$  is in turn a function of the number and range of the parameters required for each transformation.

The *Architectural Space*  $S_A$  considers all possible combinations of the configurable parameters, such as:

- Number of levels in the memory hierarchy and positioning (on-chip, off-chip);
- Unified vs split data/instruction cache, at any hierarchy level;
- Cache configuration parameters (cache size, block size, associativity etc.);
- Micro-architectural-level parameters, such as number and type of processor functional units;
- Issue width sizes for ILP processors;
- Width and topology of address and data buses;
- Encoding techniques for data and address buses;

In the rest of the paper we will indicate the generic point in the architectural design space as the vector  $a \in \mathcal{A}$  where  $\mathcal{A}$  is the architectural space defined as:

$$\mathcal{A} = S_{p_1} \times \dots \times S_{p_1} \dots \times S_{p_n}$$

where  $S_{p_i}$  is the ordered set of possible configurations for parameter  $p_i$  and “ $\times$ ” is the cartesian product.

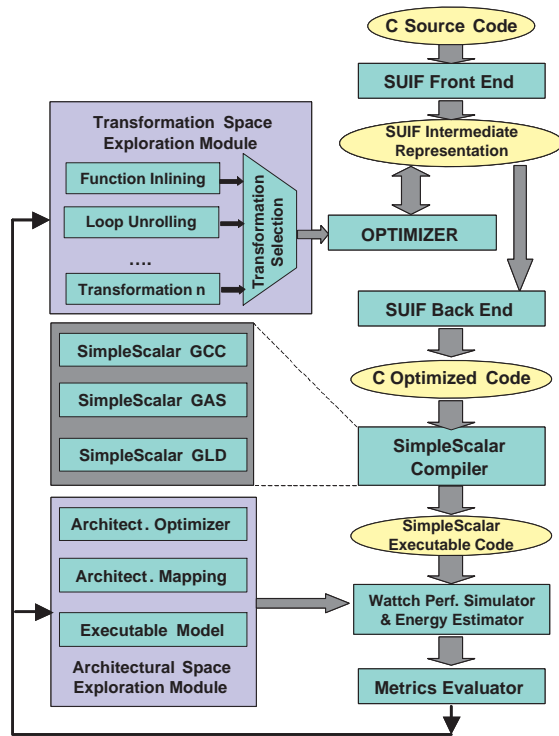
## 4. DESIGN SPACE CO-EXPLORATION FRAMEWORK

The proposed co-exploration framework is shown in Figure 1 and it is based on a modular implementation including *open-source* and *in-house* modules:

- *SUIF Optimizing Compiler*
- *Transformation Space Exploration (TSE) Module*
- *SimpleScalar Compiler*
- *Architectural Space Exploration (ASE) Module*
- *Wattch Simulator and Energy Estimator*
- *System-Level Metrics Evaluator*

Our high-level co-exploration framework receives as input the given application (i.e. a source code written in C) and explores the *System-Level Design Space* to find the optimal system configurations in terms of source-code transformations and architectural parameters.

In our approach, we concurrently explore the space of source-level transformations and architectural parameters by spanning the whole *System-Level Design Space* ( $S_T \times S_A$ ). In our design flow, the results of the co-exploration in terms of system-level evaluation metrics represent a feed-back for the *ASE* module as well as for the *TSE* module.



**Figure 1: Proposed Design Space Co-Exploration Framework**

Let us analyze in more detail each module of the proposed framework described in Figure 1.

The *SUIF Optimizing Compiler* [21] has been chosen due to its modular characteristics: the *SUIF Intermediate Representation* provides an easy-to-use programming interface enabling the construction and integration of optimization passes.

The *Transformation Space Exploration (TSE) Module* receives as input the *SUIF Intermediate Representation* and selects the source-level transformation passes to be implemented. To explore the Transformation Space, we need to reduce it to tractable size. Traditional approaches are based on a combination of heuristics and user directions techniques. Where traditional heuristics try to model the architectural features of the target machine and their impact on the transformations, our heuristics should address the interaction between architecture and transformation at a higher level.

The *SimpleScalar Compiler* integrates the flexible and portable *SimpleScalar* toolset [12], a collection of publicly-available simulation tools generating the executable code targeted toward the *SimpleScalar* architecture, a derivation of the superscalar *MIPS-IV* instruction set architecture.

The *Architectural Space Exploration (ASE) Module* receives as input the description of the possible design configurations (i.e. the target *architectural design space*) and generates the executable model of the system to be simulated by *Wattch*. The *Architecture Optimizer* module is responsible for choosing, from the design space, a set of candidate optimal points to be evaluated. Once selected a point in the design space, it is mapped into a specific instance of the target architecture by the *Architecture Mapping* mod-

ule, providing the evaluation of each point by means of an executable model.

Since in our case the *System-Level Design Space* is very large, a multi-objective exploration approach based on the full search exploration is computationally infeasible, due to the long simulation time required to explore the wide space of parameters.

The goal is to efficiently explore the multi-objective design space in order to find a good approximation of Pareto curves representing the best compromise between the interesting design objectives, mainly energy and delay. The problem of the efficient construction of Pareto curves has been already addressed in the past [19]. In the present framework, we implemented a heuristic technique (namely, *Pareto Simulated Annealing* [4]) that is an evolution of the simulated annealing approach to derive the Pareto curves for the multi-objective exploration.

Simulated annealing is a Monte Carlo approach for minimizing multivariate functions. The term “Simulated Annealing” derives from the analogy with the physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure. In the Simulated Annealing algorithm a new configuration is constructed by imposing a random displacement. If the cost function of this new state is lower than that of the previous one, the change is accepted unconditionally and the system is updated. If the cost function is greater, the new configuration is accepted probabilistically; the possibility decrease with the temperature. This procedure allows the system to move consistently towards lower cost function states, yet still ‘jump’ out of local minima due to the probabilistic acceptance of some upward moves. The PSA is an evolution of SA for multi-objective optimization. At each step of PSA, the starting point is not a single configuration but a set of points called *Partial Pareto Set*, PPS. At each step the PPS is updated with all the non-dominated new solutions. All dominated solutions in the PPS are removed, while the new dominated solutions are accepted with a probability depending on the distance from the partial Pareto front. PSA is intrinsically parallel, because the generation of each new solution from the PPS can be executed by different processors. In our case, this characteristic has been used to reduce the final simulation time.

The *Wattch Simulator and Energy Estimator Module* proposed in [13] integrates the *Wattch* framework for architectural-level power analysis. The foundations of the *Wattch* power modelling infrastructure are parameterized power models that are integrated into the Simple Scalar architectural simulators. The *Wattch* framework has been proved as a fast high-level tool, while maintaining a good level of accuracy with respect to lower-level tools [13].

The *System-Level Metrics Evaluator* is in charge of comparing the different system configurations in terms of system-level metrics. In particular, we focus on the energy and the delay associated with the processor core, the system-level buses and each level of the memory hierarchy. While the delay evaluation is embedded in the cycle-accurate simulation environment, the use of each system resource must be extracted and imported into the *Wattch* energy models defined to evaluate the overall energy metric. All these statistics are gathered by profiling the functional and memory behavior of the processor during the simulation of the embedded application by means of a cycle-accurate Instruction-Set Sim-

ulator (*ISS*).

Different metrics for the combined evaluation of the system characteristics have been implemented in the *System-Level Metrics Evaluator* module. In the case of multi-objective exploration, the module derives the energy-delay Pareto curve for the set of explored points in the system-level design space.

To compare Pareto curves derived from different exploration techniques, we used the *Two-set coverage* [22] metric. Let  $X'$ ,  $X'' \in X$  be two sets of decision vectors. The function  $C$  maps the ordered pair  $(X', X'')$  to the  $[0,1]$  interval.

$$C(X', X'') := \frac{|\{\alpha'' \in X''; \exists \alpha' \in X' : \alpha' \succeq \alpha''\}|}{|X''|} \quad (1)$$

Note that the value  $C(X', X'') = 1$  means that all points in  $X''$  are dominated by or equal to the points in  $X'$ . As opposite,  $C(X', X'') = 0$  represents the situation when none of the points in  $X''$  are covered by the set  $X'$ . Given the nature of the Pareto curve,  $C(X', X'')$  is not necessarily equal to  $C(X'', X')$ .

## 5. EXPERIMENTAL RESULTS

In this section, we analyze the experimental results obtained by applying the proposed co-exploration framework to optimize a superscalar microprocessor-based system during the execution of a set of selected industrial benchmarks written in C language (see Table 2).

In our set of experiments, we explored a 14-dimensional subset of the full System Design Space,  $S_{ES} \subset S_{SD}$ . *Loop Unrolling* and *Function Inlining* have been chosen among the possible optimization passes to reduce the number of processor cycles and energy, while among the hardware parameters we analyzed the parameters related to the memory hierarchy (to exploit data and address locality), the number of processor functional units and the processor parallel issues. Each instance of the virtual architecture has been described in terms of the following parameters:

- $S_{s_1}, S_{s_d}, S_{s_{u2}}$  are the ordered sets of the possible sizes of the I/D L1 caches (from 2 KByte to 16 KByte) and unified L2 cache (from 16 KByte to 128 KByte).
- $S_{b_1}, S_{b_d}, S_{b_{u2}}$  are the ordered sets of the possible block sizes of the I/D L1 caches (from 16 Byte to 32 Byte) and unified L2 cache (from 32 Byte to 64 Byte).
- $S_{a_1}, S_{a_d}, S_{a_{u2}}$  are the ordered sets of the possible associativity values of the I/D L1 caches (from 1 way to 2 ways for the I-cache and from 2 ways to 4 ways for the D-cache) and unified L2 cache (from 4 ways to 8 ways).
- $S_{ia}, S_{im}$  are the ordered sets of the possible number of integer ALUs and multipliers (from 1 to 2).
- $S_{fpa}, S_{fpm}$  are the ordered sets of the possible number of floating point ALUs and multipliers (from 1 to 2).
- $S_{iw}$  is the ordered set of the possible processor parallel issues (from 2 to 8).
- $S_u = \{ \langle t_u, k \rangle \mid k \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\} \}$  where  $k$  is the unroll factor.

To analyze the effectiveness of the proposed approach, we compared three different exploration strategies by evaluating the same number of simulation points (1000):

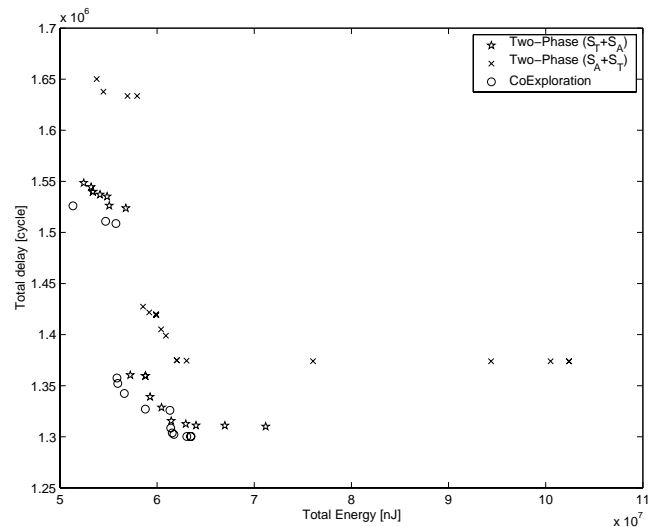


Figure 2: Comparison between Two-Phase and Co-Exploration methodologies in terms of energy-delay Pareto points for the LU decomposition (Lueq) benchmark.

- *Two-Phase  $S_A + S_T$  Exploration*, where we performed *separately* the heuristic search of the Architectural Space followed by the exhaustive search of the Transformation Space. The cardinality of the design space is:  $|S_{SD}| = |S_A| + |S_T|$
- *Two-Phase  $S_T + S_A$  Exploration*, where we performed *separately* the exhaustive search of the Transformation Space (given the target architecture) followed by the heuristic search of the Architectural Space. The cardinality of the design space is:  $|S_{SD}| = |S_T| + |S_A|$
- *Co-Exploration*, where we performed *jointly* the heuristic search on the System-Level Design Space, that is the product of the Transformation Space and the Architectural Space. The cardinality of the design space is:  $|S_{SD}| = |S_T \times S_A|$

Figure 2 shows three approximate Pareto curves in the energy-delay space for the LU decomposition benchmark, that has been selected because it clearly represents the energy-delay behavior common to all the benchmarks. The first approximate Pareto curve (empty dots) is related to our co-exploration methodology applied to the LU decomposition benchmark, while the other two approximate Pareto curves (crosses and stars) represent the results of the two-phase explorations. The curves show a dominance of the results of the co-exploration in finding Pareto points.

To compare the Pareto curves derived for the different benchmarks, Table 1 reports the two-set coverage metric (given by Equation 1) computed for all the selected benchmarks. Since the metric is asymmetric, it has been computed for co-exploration with respect to two-phase  $S_T + S_A$  and vice versa (see second and third column of Table 1 respectively), and for co-exploration with respect to two-phase  $S_A + S_T$  and vice versa (see fourth and fifth column Table 1 respectively). The average unroll factors corresponding to the Pareto points for the different benchmarks and their standard deviations are shown in Table 2, to outline the effects of the *Loop Unrolling* transformation on each benchmark.

**Table 1: Comparison of Pareto curves: Two-set coverage of the Co-Exploration (Co) with respect to Two-Phase Explorations (TP) for different benchmarks.**

Benchmark	$S_A + S_T$		$S_T + S_A$	
	C(Co,TP)	C(TP,Co)	C(Co,TP)	C(TP,Co)
AES	0.9	0	1	0
DCT_IDCT	0.07	0.7	0.09	0.8
FDCT	0.14	0.64	0.12	1
IDCT	0.56	0.8	0	0.9
FIR1	0.96	0	0.82	0
FIR2	0.43	0.5	0.7	0.5
GAMMA	0.83	0	1	0
LUEQ	1	0	1	0
Average	0.61	0.33	0.59	0.4

**Table 2: Average unroll factor and related standard deviation for Co-Exploration Pareto points for different benchmarks.**

Benchmark	Unroll Avg.	Unroll St.Dev.
AES	11	1.07
DCT_IDCT	0	0
FDCT	4.36	4.18
IDCT	0	0
FIR1	10	5.90
FIR2	0	0
GAMMA	4.40	3.29
LUEQ	4.93	1.03
Average	4.34	1.93

We note that, wherever the *Loop Unrolling* transformation provides some benefits in terms of delay and energy, the co-exploration dominates the two-phase exploration techniques, except for the case of the FDCT benchmark, where the high standard deviation is due to the fact that for the large part of the co-exploration space the optimal unroll factor is equal to zero. In this case, and in general when the optimal unroll factor is equal to zero (i.e. when the Loop Unrolling is not useful at all such as in *DCT\_IDCT*, *IDCT* and *FIR2*), the co-exploration is outperformed, since the two-phase explorations analyze a reduced design space with respect to the co-exploration.

## 6. CONCLUSIONS AND FUTURE WORKS

Our work presents a co-exploration of the architectural and transformation spaces supporting the design of mixed hardware and software systems for embedded low-power applications. The results of the exploration would have been biased towards the co-exploration, in case the full search of the design space had been possible. Being the full search computationally infeasible due to the large number of points in the system-level design space, the heuristic exploration approach represents the critical factor to obtain an effective co-exploration method in terms of accuracy and efficiency. Experimental results (obtained by a heuristic technique based on Pareto Simulated Annealing) have shown that better energy-delay Pareto points can be reached by the co-exploration of the architectural and transformation spaces with respect to the two-phase separate explorations for the benchmarks for which the selected transformations have been beneficial.

Our on-going work is directed towards the co-exploration of architectures based on *Very Long Instruction Word* pro-

cessors and the implementation and analysis of other source code transformations, such as *Unimodular* and *Affine* transformations.

## 7. REFERENCES

- [1] N. Vijaykrishnan et al. Evaluating Integrated Hardware-Software Optimizations Using a Unified Energy Estimation Framework. *IEEE Trans. on Computers*, 52(1):59–76, January 2003.
- [2] M. Kandemir et al. Influence of Compiler Optimizations on System Power. *IEEE Trans. on VLSI Systems*, 9(6):801–804, December 2001.
- [3] D. Fischer, J. Teich, M. Thies, and R. Weper. Efficient architecture compiler co-exploration for asips. *ACM CASES2002*, 2002.
- [4] Czyzak P. and Jaszekiewicz A. Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimisation. *Journal of Multi-Criteria Decision Analysis*, (7):34–47, April 1998.
- [5] L. Benini and G. De Micheli. System-Level Power Optimization Techniques and Tools. *ACM TODAES*, 5(2):115–192, 2000.
- [6] G. De Micheli T. Simunic, L. Benini and M. Hans. Source-Code Optimization and Profiling of Energy Consumption in Embedded Systems. *Proc. of ISSS00*, pages 193–198, 2000.
- [7] V. Tiwari, S. Malik, A. Wolfe, and M. Lee. Instruction level power analysis and optimization of software. *J. VLSI Signal Processing*, pages 1–18, 1996.
- [8] Y. Li and J. Henkel. A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems. *DAC-35*, June 1998.
- [9] E. De Greef F. Balasa L. Nachtergaele F. Cathoor, S. Wuytack and A. Vandecappelle. Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design. *Kluwer*, 1998.
- [10] L. Benini E.-Y. Chung and G. De Micheli. Source Code Transformation based on Software Cost Analysis. *Proc. of ISSS2001*, pages 153–158, 2001.
- [11] T. M. Conte, K. N. Menezes, S. W. Sathaye, and M. C. Toburen. System-level power consumption modeling and tradeoff analysis techniques for superscalar processor design. *IEEE Trans. on VLSI Systems*, 8(2):129–137, Apr. 2000.
- [12] T. M. Austin D. Burger and S. Bennett. Evaluating Future Microprocessors: The SimpleScalar Tool Set. Technical Report CS-TR-1996-1308, University of Wisconsin, 1996.
- [13] V. Tiwari D. Brooks and M. Martonosi. Watch: A Framework for Architectural-Level Power Analysis and Optimizations. *Proceedings ISCA2000*, pages 83–94, 2000.
- [14] N. Bellas, I. N. Hajj, D. Polychronopoulos, and G. Stamoulis. Architectural and Compiler Techniques for Energy Reduction in High-Performance Microprocessors. *IEEE Transactions on VLSI Systems*, 8(3), June 2000.
- [15] N. Vijaykrishnan, M. Kandemir, M.J. Irwin, H.S. Kim, and W. Ye. Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower. *ISCA 2000*, June 2000.
- [16] T. D. Givargis, F. Vahid, and J. Henkel. Evaluating Power Consumption of Parameterized Cache and Bus Architectures in System-on-a-Chip Designs. *IEEE Transactions on VLSI Systems*, 9(4), August 2001.
- [17] T. D. Givargis and F. Vahid. Platune: a tuning framework for system-on-a-chip platforms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(11):1317–1327, November 2002.
- [18] M. Palesi and T Givargis. Multi-objective design space exploration using genetic algorithms. In *CODES 2002*, May 6–8 2002.
- [19] G. Palermo, C. Silvano, and V. Zaccaria. A flexible framework for fast multi-objective design space exploration of embedded systems. *PATMOS03*, Sept. 2003.
- [20] D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler Transformations for High-Performance Computing. *ACM Computing Surveys*, 26(4):345–420, 1994.
- [21] R. P. Wilson et al. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers. *SIGPLAN Notices*, 29(12):31–37, 1994.
- [22] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.