Noname manuscript No.
(will be inserted by the editor)

G. Agosta G. Palermo C. Silvano

Efficient Architecture/Compiler Co-Exploration Using Analytical Models

the date of receipt and acceptance should be inserted later

Abstract The hardware/software co-exploration is a critical phase for a broad range of embedded platforms based on the System-On-Chip approach. Traditionally, the compilation and the architectural design sub-spaces have been explored independently. Only recently, some approaches have analyzed the problem of the concurrent exploration of the compilation/architecture sub-spaces. This paper proposes a framework to support the co-exploration phase of the design space composed of architectural parameters and source program transformations. The objective space is multi-dimensional, including conflicting objectives such as energy and delay. In the proposed framework, heuristic co-exploration techniques based on Pareto Simulated Annealing (PSA) have been used to efficiently explore the architecture/compiler co-design space. This space is composed of the product of the parameters related to the selected source program transformations and to the configurable architectures. A first result of this paper consists of showing how the architecture/compiler co-exploration can be more effective than a traditional two-phase exploration. Since the co-exploration space is quite large, to speed up the co-exploration phase by several orders of magnitude over simulation-based approaches, a methodology based on analytical models has been introduced in the co-exploration framework. The goal of analytical models is to quickly evaluate energy/delay metrics at the systems level, while maintaining accuracy with respect to simulation-based co-exploration. The proposed co-exploration framework has been applied to a parameterized SoC superscalar architecture during the execution of a selected set of multimedia applications.

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

Politecnico di Milano, Milano, Italy
E-mail: {agosta,gpalermo,silvano}@elet.polimi.it

1 Introduction

Evaluation of energy-delay metrics at the system-level is of main 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 a *multi-objective DSE* phase is to optimally configure the parameterized *SOC* platform in terms of both energy and performance requirements depending on the given application.

Due to the complexity of recent HW/SW embedded systems, there is a critical need to address architectural optimizations *concurrently* with the tuning of the target application. The HW/SW co-exploration task consists of dynamically profiling the target application compiled using different transformations and executed on different system configurations obtained by varying architectural parameters. Through compiler/architecture co-exploration, the synergy of HW and SW parts of an embedded system can be exploited and power/performance trade-offs can be evaluated.

Only recently, some approaches [1–4] have analyzed the co-exploration of the compilation space and the architectural design space from the energy/delay combined perspective. Although the works in [1,2] investigate the influence of compiler optimization on the architectural exploration, the source code is first optimized *independent* of the target architecture, then the architecture is explored. The Buildabong framework proposed in [3] can be considered as one of the first effort to support compiler/architectural co-exploration to design ASIPs. In this work, an algorithm for pruning the search space to find Pareto-optimal solutions has been proposed. The multi-objective space considered in [3] is 3-dimensional: hardware cost, code size and execution time. The main limitation of this approach is that the energy cost has not yet been considered. However, energy represents one of the main constraints in embedded system design. In the present paper, we addressed the multi-objective co-exploration taking into considerations both energy and delay.

Our work is complementary to the most recent studies in architecture/compiler co-exploration. In our approach, the main goal of the exploration phase is to span *concurrently* the transformation and the architectural spaces, since we want to exploit the intrinsic correlation between architectural and compiler parameters. In fact, in the proposed design flow, the results of the co-exploration in terms of evaluation metrics provide a feedback for the *Architectural Space Exploration (ASE)* module as well as for the *Transformation Space Exploration (TSE)* module.

In particular, our framework has been proposed in [4] to jointly explore the set of all *HW/SW* configurations of the *SOC* platform defined in the combined *System-Level Design Space*: $S_{SD} = S_T \times S_A$, where S_T is the space of program transformations and S_A is the space of architectural parameters. From the *hardware* side, we explore the architecture design space (S_A) 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 (S_T) for embedded applications. The objective of this phase consists of finding 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.

Since the *Compiler/Architecture Design Space* is usually very large, a multi-objective (power/performance) co-exploration approach based on the full search is computationally infeasible, due to the long time required to simulate the wide space of design parameters. In a multi-objective scenario, such as the one we are considering, the goal of the system-level exploration is to derive approximated energy/delay Pareto curves representing the best trade-off between the interesting design requirements.

In the *Compiler/Architecture Design Space*, the time spent in the exploration phase t_{DSE} can be computed as $|S_T \times S_A| \times t_{sim}$ where t_{sim} is the average execution time of a single simulation run for a given architectural platform and target application. As a motivating example, let us consider a simplified scenario, where a 6-dimensional S_A includes data and instruction cache parameters (where each parameter can assume 4 values) and a 2-dimensional S_T includes parameters related to loop tiling and loop unrolling (where again each parameter can assume 4 values). Thus the cardinality of the combined design space is $|S_T \times S_A| = 4096 \times 16 = 65536$. In the case of a target application requiring a simulation time of 1 min, the time needed to exhaustively explore the combined design space would have been approximately 45 days. Therefore, to reduce the exploration time, one of two factors (or both) must be considered: either the number of explored design space points must be reduced, or an estimation technique faster than simulation must be employed.

The main contribution of this paper is twofold: in the first part of the paper, heuristic optimization techniques (such as Pareto Simulated Annealing) have been used to efficiently co-explore the design space; in the second part of the paper, analytical models have been introduced to describe the energy–delay system behavior as a function of the design space parameters, thus speeding up the co-exploration phase, while maintaining sufficient accuracy with respect to simulation–based co-exploration.

The main goal of the second part of the paper is to propose a methodology, based on the introduction of analytical models, to reduce the cost of the co-exploration phase, rather than a complete analytical model of the system behavior. In particular, the analytical models proposed in this paper focus on the energy-delay behavior of the memory hierarchy. This fact does not represent a limitation, since the proposed methodology can be applied to build extended analytical models representing other parts of the system.

The definition of the equation coefficients in the analytical models (namely, *model characterization* phase) requires the simulation of the target application in a set of design space points to derive the equation coefficients that characterize the analytical models. Once the model coefficients have been characterized, analytical models are used to speed up the co-exploration

phase to rapidly predict the energy/delay for each point of the large co-design space, thereby avoiding the simulation for each co-design point.

In this paper, the proposed co-exploration methodology has been applied to a parameterized superscalar *SoC* architecture during the execution of a set of benchmarks to find the best trade-off between the interesting design objectives, mainly energy and delay. A first set of experiments shows that the proposed co-exploration approach is more effective to derive approximate energy/delay Pareto curves than a traditional independent exploration of the transformation and architectural sub-spaces. A second set of experimental results shows how the proposed analytical approach can be effective for the reduction of the exploration time, while maintaining sufficient accuracy with respect to simulation-based co-exploration. In particular, experimental results show that, for the selected set of benchmark, the co-exploration speed-up reached by the proposed analytical-based approach is more than three orders of magnitude with respect to traditional simulation-based co-exploration. To evaluate the accuracy of the analytical-based with respect to simulation-based approaches, the analytical equations derived in the model characterization phase have been applied to estimate energy/delay metrics for a set of validation points. The accuracy is 6.7% for the energy and 13% for the delay.

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 design space co-exploration framework is presented in Section 4. To speed up the co-exploration phase, analytical models are introduced in Section 5, while Section 6 discusses the experimental results carried out to evaluate the effectiveness of the proposed framework for a superscalar *SoC* architecture. Finally, some concluding remarks and possible evolutions of this work have been outlined in Section 7.

2 Background

The overall energy of embedded *SOC* platforms depends on both software and hardware sides. In this section, we survey the most relevant works in the fields of architectural exploration, source code transformations for energy optimization, analytical models for energy and delay estimation and compiler/architecture co-exploration.

2.1 Architectural Exploration

Several system-level estimation and exploration methods have been proposed in literature targeting power-performance tradeoffs from the architectural standpoint [5–8].

The *SimpleScalar* toolset [6] 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* [8] 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 [7] 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 work in [5] proposes a system-level technique to find low-power high performance superscalar processors tailored to specific user applications.

Recently some approaches have been introduced to approximate Pareto-curves for computer architecture design [9], [10], [11]. *Platune* [9] 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. [10] extended *Platune* by applying genetic algorithms to optimize dependent parameters, resorting to the default *Platune* policy when independent parameters are specified by the user.

2.2 Source Code Transformations

According to [12], the most important optimization passes 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. For the purpose of co-exploration, the most relevant techniques are those operating on loops, such as loop unrolling, loop tiling and loop fusion.

Several low-power software design techniques have been proposed in literature [13], [14]. Instruction-level optimization techniques [15] can be automated by applying them in the back-end of the compiler, however their impact on energy is quite limited and they are strongly related to the target architecture.

Other promising techniques are based on source code transformations [16–18]. Panda *et al.* [16] present a survey of the state-of-the-art techniques for the optimization of memory accesses, including software transformations. In [17], code specialization is used for energy reduction, and exploration of the transformation design space is performed. The work proposed in [18] explores the program transformation design space to speed up the execution of DSP applications.

With respect to these previous works, our approach bridges compiler and architecture exploration, showing that exploring the software and hardware parameters can lead to better solutions.

2.3 Analytical Energy/Delay Estimation

Analytical techniques have been proposed in [19] for speeding up the cache/bus power and performance evaluation with respect to simulation, while maintaining sufficient accuracy with respect to simulation-based approaches. In [19], analytical models are limited to represent cache and bus behavior. In our approach, the proposed analytical models cover the global energy/delay system behavior in terms of compiler/architecture parameters such as tile size and cache size.

A software cost model has been proposed in [20] to evaluate the impact of transformations such as loop unrolling and loop tiling on system energy and delay. Analytical and statistical models are proposed in [21] to estimate the energy reduction obtained through loop unrolling and loop fusion.

In our approach, analytical models of the impact of program transformations and architecture parameters on energy and performances are applied within the proposed co-exploration framework, to speed up the co-exploration phase with respect to simulation-based approaches. Our work represents a first attempt to combine both compiler and architecture parameters in a analytical models.

2.4 Compiler/Architecture Co-Exploration

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 ([22,23,1–3]) have analyzed the *concurrent* exploration of the compilation space and the architectural design space from the energy point of view.

The *Avalanche* framework [22] represents one of the first attempts in literature to simultaneously evaluate the energy-performance tradeoffs for software, memory and hardware for embedded systems.

Low-power design optimization techniques for high performance processors have been investigated in [23] from the architectural and compiler standpoints.

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. Moreover, this work first analyzes the source code transformations, independently of the architecture, then the architecture parameters are explored.

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. A technique to prune

the large search space to reduce the exploration phase has been proposed. The objective space of this work is multi-dimensional including hardware cost, execution time and code size. The main limitation of this approach is that the energy cost has not been considered.

Other approaches are based on the introduction of Architecture Description Languages (ADLs) to support the co-exploration phase. The co-exploration approach proposed in [24] is based on the LISA ADL to optimize the processor architecture together with the on-chip communication. In [25], the EXPRESSION language has been proposed to support architectural design space exploration for embedded systems-on-chip and automatic generation of a retargetable compiler/simulator toolkit. Based on the EXPRESSION ADL, the PBExplore approach described in [26] shows an application of the co-exploration methodology to the problem of the optimization of register bypassing.

2.5 Motivations of our Work

The approach presented in this paper is complementary to the recent studies focusing on compiler/architectural energy optimization and estimation. Starting from our preliminary work in [4], the compiler/architecture design spaces are explored concurrently and energy/delay trade-offs have been analyzed. The main contributions of our work can be summarized as:

- The assessment of the usefulness of hardware/software co-exploration with respect to traditional approaches where the hardware and software design spaces are spanned separately;
- The integration of analytical models of energy and delay at system level into the co-exploration framework to speed up the co-exploration phase.

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 S_T and Architectural Space S_A:

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

The main contribution of our approach is that the S_T and S_A spaces are not spanned independently, but rather jointly. In this way, the cardinality of the explored design space is given by $|S_T \times S_A|$.

The *Transformation Space* S_T models the effects of a set of program transformations on the source code, such as function inlining, loop unrolling, loop tiling and loop fusion. The points of S_T correspond to different applications of the optimization passes to the same source program.

For the purpose of co-exploration, we focused on a pair of source code transformations impacting on architectural parameters: loop unrolling and

loop tiling. These two transformations are parameterized in terms of unroll factor and tile size. The sets of possible values for these parameters can be quite large, making these two transformations suitable targets for co-exploration. On the contrary, source code transformations such as function inlining or loop fusion generate a small Transformation Space, since their parameters are binary values representing whether the transformation is applied or not.

In this paper, let us indicate the generic point in the transformation space as the vector $t \in S_T$ where S_T is the transformation space defined as:

$$S_T = S_{p_1} \times \ldots S_{p_l} \ldots \times S_{p_n}$$

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

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: $i(t) =< t, a_1 \ldots a_k >$. For example, if t is Loop Unrolling, an instance of t is $< t, 2 >$, which means Loop Unrolling with unroll factor equal to 2.

Let P be the source program, composed of n transformation points $p_1 \ldots p_n$: $P = \{p_1 \ldots p_n\}$. Let T be a set of m instances of transformations $i_1..i_m$. 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 $< t_i, p_j >$ 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 architectural 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;

In the experimental evaluation reported in Section 6.1, to evaluate the effectiveness of the co-exploration approach, a wide subset of S_A has been used including cache-related parameters as well as parameters related to the processor microarchitecture. On the other hand, in Section 6.2, to validate the analytical models describing the energy and delay behaviors of the caches, the architecture space S_A has been reduced taking into account only the L1 cache related parameters.

In this paper, we indicate the generic point in the architectural design space as the vector $a \in S_A$ where S_A is the architectural space defined as:

$$S_A = S_{p_1} \times \ldots S_{p_l} \ldots \times S_{p_n}$$

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

In the experimental evaluation discussed in the paper, we co-explored a 15–dimensional subset of the full System Design Space ($S_{ES} \subset S_{SD}$) in Section 6.1, while we considered an 8–dimensional subset to reduce the computational cost of PSA-based co-exploration reported in Section 6.2.

4 Design Space Co-Exploration Framework

The design space co-exploration framework described in this section has been previously proposed in [4]. The structure of the framework, shown in Figure 1, is based on a modular implementation including the following *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.*

The 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.

The proposed framework enables the concurrent exploration 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 feedback for both ASE module and TSE modules.

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

The *SUIF Optimizing Compiler* [27] has been chosen due to its modularity: the *SUIF* Intermediate Representation provides an easy-to-use programming interface enabling the construction and integration of optimization passes. The module called *Dependence Analyzer and IR Transformer* receives as input the *SUIF* Intermediate Representation, applies the source code transformations and then generates the optimized intermediate representation for the SUIF back end. The *Source Code Optimizer* in the *Transformation Space Exploration (TSE) Module* selects the source-level transformation passes to be applied.

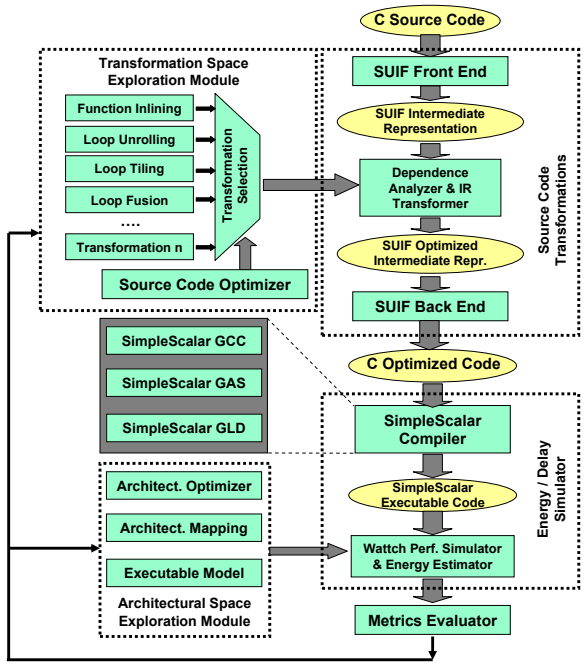

Fig. 1 Basic design space co-exploration framework.

The *SimpleScalar Compiler* integrates the flexible and portable *SimpleScalar* toolset [6], 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* module, 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 [11].

In the *TSE* and *ASE* modules, we implemented a heuristic technique (namely, *Pareto Simulated Annealing* [28]) that is an evolution of the simu-

lated annealing approach to derive the Pareto curves for the multi-objective exploration. The Pareto Simulated Annealing has been chosen since it has been demonstrated in [29] that it is suitable for multi-objective DSE. 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 acceptance rate decreases 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.

The *Performance Simulator and Energy Estimator* module of our framework integrates the *Simple Scalar* architectural simulator and the *Wattch* tool for architectural level power analysis [7]. Then, the *System-Level Metrics Evaluator* is in charge of comparing the different system configurations in terms of system-level metrics. *Energy* and *delay* have been chosen, since they are the most widely used metrics, but the methodology is general and could be easily extended to cover other system-level metrics such as throughput. The motivation of the proposed methodology is to reduce the simulation overhead of the co-exploration phase, so it is not useful to consider static metrics, such as area, to evaluate the effectiveness of our approach.

In our framework, while the delay evaluation is embedded in the cycle-accurate SimpleScalar 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 Simulator (ISS). For multi-objective exploration, the *System-Level Metrics Evaluator* module derives the energy-delay Pareto curve for the set of explored points in the system-level design space. These metrics provide the required feedback for the *TSE* and *ASE* modules to derive the next points to evaluate.

5 Proposed Analytical Approach

The main contribution of this paper consists of introducing a methodology based on analytical models to replace the simulation in the estimation of energy/delay metrics at system-level, to further reduce the time spent during the co-exploration phase.

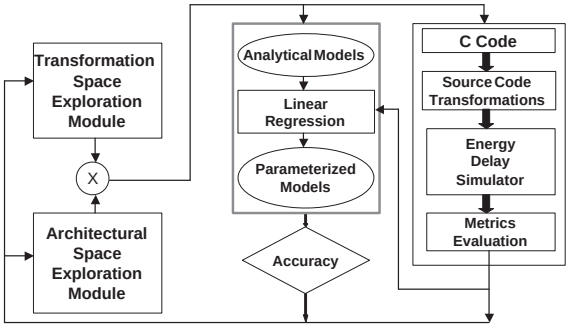

Fig. 2 Proposed design space co-exploration framework using analytical models: Models characterization phase.

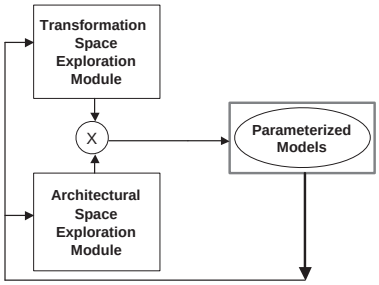

Fig. 3 Proposed design space co-exploration framework using analytical models: Application of parameterized models to speed up the co-exploration.

We propose to empirically derive the analytical equations describing the energy–delay behavior of the modules of the system as a function of the design space parameters. In a first phase, called *model characterization*, we simulate the target application in a set of design space points to derive the equation coefficients that characterize the analytical models. Once the equation coefficients in the models have been characterized, the second phase consists of the application of parameterized models to speed up the co-exploration phase to traverse the system–level design space.

To this end, we modified the basic co-exploration framework proposed in Figure 1 as shown in Figure 2 and 3. In Figure 2, the co-exploration framework has been extended with a linear regression module that analyzes the metric values and tunes a set of coefficients in the analytical models. The resulting parameterized models are verified for accuracy against the metric values obtained by simulation. This first modified framework is used until the accuracy of the parameterized models becomes steady. Then, the simplified framework described in Figure 3 replaces it. In this simplified framework, the simulation and compilation phases of the original framework are entirely replaced by the parameterized models.

The proposed analytical models focus on the energy-delay behavior of the memory hierarchy. This fact does not represent a limitation of the approach, since the proposed approach based on the introduction of analytical mod-

els can be easily extended to other classes of the system-level parameters. However, the effect of the other architectural parameters (such as number of functional units) that have been fixed, is taken into account in the coefficients of the analytical models, dynamically characterized.

In order to simplify the model complexity, the *System-Level Design Space* parameters have been divided in two clusters, assuming that parameters from different clusters are independent to each other, as proved in [30]. The first set, called *Data Side Parameters Cluster*, includes the Tile Size, Data Cache Size, Data Cache Block Size and Data Cache Associativity. The second set, called *Instruction Side Parameters Cluster*, includes the Unroll Factor, Instruction Cache Size, Instruction Cache Block Size and Instruction Cache Associativity.

Each cluster defines a new subset of the full *System-Level Design Space*. For each subset, we first define an analytical model that allows us to estimate the energy and delay metrics from the values of the parameters in the corresponding cluster. In the *model characterization* phase, we simulate a set of randomly chosen points, and we apply a linear regression to obtain values for the coefficients of the equations. Then, we combine the two sets of models (data side and instruction side) into a unified model to represent the system behavior in the full *System-Level Design Space*.

5.1 Analytical Models depending on Data Side Parameters Cluster

Within the subspace defined by the *Data Side Parameters Cluster*, we first estimate the data cache miss rate (MR^D). We assume that the MR^D depends on the inverse of the data cache size (S^D), as larger caches can hold more blocks. The MR^D is affected by the data cache block size (B^D) with a dependence that is experimentally proved to be both linear and quadratic. This is due to the fact that larger block sizes reduce misses, taking advantage of spatial locality, but the miss rate actually grows if the block size is too large with respect to the cache size [31]. The miss rate also has a linear dependence on the data cache associativity (W^D), as shown in [19]. There is a quadratic dependence on the tile size (T), since larger tiles increase the number of misses, and each tile is a matrix of size ($T \times T$). Therefore, the analytical model of data cache miss rate MR^D is:

$$\begin{aligned} MR^D = \mu_0^D + \mu_1^D \tfrac{1}{S^D} + \mu_2^D B^D + \\ + \mu_3^D (B^D)^2 + \mu_4^D W^D + \mu_5^D T^2 \end{aligned} \tag{1}$$

where the μ_0^D to μ_5^D coefficients must be estimated in the model characterization phase.

The next step is to estimate the number of instructions per branch (IPB^D) by considering its inverse dependence on T, since the number of branch instructions to perform increases as the tile size decreases:

$$IPB^D = \beta_0^D + \beta_1^D \tfrac{1}{T} \tag{2}$$

After the estimation of MR^D and IPB^D metrics, we can now provide an estimate of the delay and energy associated with data side parameters:

$$Delay^D = \delta_0^D + \delta_1^D MR^D + \delta_2^D IPB^D \tag{3}$$

$$\begin{aligned} Energy^D = \varepsilon_0^D + \varepsilon_1^D MR^D + \varepsilon_2^D IPB^D + \\ + \varepsilon_3^D S^D + \varepsilon_4^D W^D \end{aligned} \tag{4}$$

The delay is estimated as a linear combination of MR^D and IPB^D, as additional misses and branches increase the execution time of the program proportionally to their respective costs. MR^D and IPB^D also contribute to the energy consumption, as they cause an increase in the activity of the corresponding hardware modules. However, energy is also directly dependent on the size of the data cache, since a larger cache consumes more energy, and on the data cache associativity, due to the increased complexity of the control hardware.

5.2 Analytical Models depending on Instruction Side Parameters Cluster

As before, the coefficients of the models depending on the *Instruction Side Parameters Cluster* have been empirically derived and models coefficients have been estimated by the model characterization phase. Within the subspace defined by the *Instruction Side Parameters Cluster*, we first estimate the instruction cache miss rate (MR^I). The MR^I depends on the inverse of the instruction cache size (S^I), as larger caches can hold more blocks. The MR^I is affected by the instruction cache block size (B^I), but due to the more sequential access patterns (with respect to the data cache), this dependence is on the inverse of the B^I, and no degradation is experienced on larger caches. The miss rate has a linear dependence on the instruction cache associativity (W^I). Since the size of the code linearly depends on the unroll factor U_f, this parameter also affects the miss rate. The overall MR^I model is shown in Equation 5:

$$MR^I = \mu_0^I + \mu_1^I \frac{1}{S^I} + \mu_2^I \frac{1}{B^I} + \mu_3^I W^I + \mu_4^I U_f \tag{5}$$

The next step is to estimate the number of instructions per branch (IPB^I) due to the parameters in the *Instruction Side Parameters Cluster*. Since the distance between two branches increases with growing values of the U_f, there is a direct dependency:

$$IPB^I = \beta_0^I + \beta_1^I U_f \tag{6}$$

We also estimate the variation in the number of instruction per cycle (IPC) as a linear combination of the MR^I and the IPB^I, as shown in Equation 7:

$$IPC^I = \eta_0^I + \eta_1^I MR^I + \eta_2^I IPB^I \tag{7}$$

Having estimated the MR^I, IPB^I and IPC^I, we can now provide an estimate of delay and energy associated with the instruction cache parameters, shown in Equations 8 and 9:

$$Delay^I = \delta_0^I + \delta_1^I MR^I + \delta_2^I IPB^I + \delta_3^I IPC^I \tag{8}$$

$$\begin{aligned} Energy^I = \varepsilon_0^I + \varepsilon_1^I MR^I + \varepsilon_2^I IPB^I + \\ + \varepsilon_3^I IPC^I + \varepsilon_4^I S^I + \varepsilon_5^I W^I \end{aligned} \tag{9}$$

The delay is estimated as a linear combination of the three metrics, since additional misses and branches contribute to the execution time of the program proportionally to the respective costs, and the IPC^I also affects the total execution time. Although the IPC^I term is a linear combination of MR^I and IPB^I, we keep it explicitly in Equations 8 and 9, since it is more accurate to first evaluate the η coefficients and then use the estimated IPC^I term instead of re-evaluating new coefficients for the indirect contributions of MR^I and IPB^I to IPC^I.

The MR^I, IPB^I and IPC^I contribute also to the energy consumption, as they cause an increase in the activity of the respective hardware. However, energy is also directly dependent on the size of the instruction cache, since a larger cache consumes more energy, and on the instruction cache associativity, due to the increased complexity of the control hardware.

5.3 System-Level Analytical Models

Finally, we combine the models associated with the two (instruction and data) clusters to provide estimates of both the instruction side and data side (and related source code transformations) contributions to the overall system energy and delay.

To do so, we must purge the constant components from the instruction side and data side metrics, keeping only the variable components that actually represent the useful information, i.e. the dependence relation between the metric and the design space parameters, and new constant terms δ_0^{Tot} and ε_0^{Tot} are introduced to represent the contributes to the overall system energy and delay that are independent from the design space parameters.

Moreover, we need to combine IPB^I and IPB^D into a single metric, as in Equation 10:

$$IPB^{Tot} = \beta_0^{Tot} + \beta_1^I U_f + \beta_1^D \frac{1}{T} \tag{10}$$

Then, we can estimate the overall delay and energy associated with the system:

$$\begin{aligned} Delay^{Tot} = \delta_0^{Tot} + \delta_1^{Tot} IPB^{Tot} + \delta_1^D MR^D + \\ + \delta_1^I MR^I + \delta_3^I IPC^I \end{aligned} \tag{11}$$

$$\begin{aligned} Energy^{Tot} = \varepsilon_0^{Tot} + \varepsilon_1^{Tot} IPB^{Tot} + \varepsilon_1^I MR^I + \\ + \varepsilon_3^I IPC^I + \varepsilon_4^I S^I + \varepsilon_5^I W^I + \\ + \varepsilon_1^D MR^D + \varepsilon_3^D S^D + \varepsilon_4^D W^D \end{aligned} \tag{12}$$

6 Experimental Results

In this section, two sets of experiments are shown:

- The first set to compare the architecture/compiler co-exploration approach with a traditional independent exploration of the architecture and compiler subspaces;
- The second set to prove the effectiveness of our analytical approach for the reduction of the co-exploration time.

Experimental results are obtained by applying the proposed co-exploration framework to optimize the PISA superscalar architecture during the execution of a set of selected kernels from multimedia industrial applications supplied by STMicroelectronics. The set of target benchmarks are summarized in Table 6. We used application kernels because the loop transformations must be optimally tuned for each loop. In a large application, only the most frequently executed parts of the code (kernels) will be fully co-optimized with the architecture. The choice to use application kernels is also supported by the fact that, for the class of applications represented by the most popular suites of standard benchmarks (such as Mediabench and Powerstone), the percentage of execution time spent in the five most frequently used loops is, on average, over 82% as reported in [32].

Table 1 Description of the target benchmark kernels

Benchmark	**Description**
AES	Symmetric Block Cypher
DCT_IDCT	Un-optimized Discrete Cosine Transform and Inverse Discrete Cosine Transform
FDCT	Fast Discrete Cosine Transform
FIDCT	Fast Inverse Discrete Cosine Transform
FIR 1, FIR 2	Linear-phase FIR digital filter (1D, 2D)
GAMMA	Computes the gamma coefficients
LUEQ	Lower/upper triangular matrix factorization

6.1 Co-Exploration vs. Two-Phase Exploration

In our set of experiments, we co-explored a 15–dimensional subset of the full System Design Space ($S_{ES} \subset S_{SD}$), where the Architectural Space is given by: $S_A = S_{s_i} \times S_{s_d} \times S_{s_{u2}} \times S_{b_i} \times S_{b_d} \times S_{b_{u2}} \times S_{a_i} \times S_{a_d} \times S_{a_{u2}} \times S_{ia} \times S_{im} \times S_{fpa} \times S_{fpm} \times S_{iw}$ and the Transformation Space is simply given by $S_T = S_u$. More in detail, we considered:

- $\mathbf{S_{s_i}} = \{2KB, 4KB, 8KB, 16KB\}$ sizes of the L1 instruction cache;
- $\mathbf{S_{s_d}} = \{2KB, 4KB, 8KB, 16KB\}$ sizes of the L1 data cache;
- $\mathbf{S_{s_{u2}}} = \{16KB, 32KB, 64KB, 128KB\}$ sizes of the unified L2 cache;
- $\mathbf{S_{b_i}} = \{16B, 32B\}$ block sizes of the L1 instruction cache;
- $\mathbf{S_{b_d}} = \{16B, 32B\}$ block sizes of the L1 data cache;

- $\mathbf{S_{b_{u2}}} = \{32B, 64B\}$ block sizes of the unified L2 cache;
- $\mathbf{S_{a_i}} = \{1-way, 2-way\}$ associativity values of the L1 instruction cache;
- $\mathbf{S_{a_d}} = \{2-way, 4-way\}$ associativity values of the L1 data cache;
- $\mathbf{S_{a_{u2}}} = \{4-way, 8-way\}$ associativity values of the unified L2 cache;
- $\mathbf{S_{ia}} = \{1, 2\}$ number of integer ALUs;
- $\mathbf{S_{im}} = \{1, 2\}$ number of integer multipliers;
- $\mathbf{S_{fpa}} = \{1, 2\}$ number of floating point ALUs;
- $\mathbf{S_{fpm}} = \{1, 2\}$ number of floating point multipliers;
- $\mathbf{S_{iw}} = \{2, 4, 8\}$ processor parallel issues;
- $\mathbf{S_u} = \{< t_u, k > | k \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}\}$ where k is the unroll factor;

The co-exploration based on full search would have required the simulation of the System-Level Design Space $S_{SD} = S_A \times S_T$, whose cardinality is given by: $|S_{SD}| = |S_A \times S_T| = 196608 \times 10$. To analyze the effectiveness of the proposed approach, we compared three different exploration strategies (two traditional strategies where the two spaces are explored separately, and our co-exploration strategy) by evaluating the same number of simulation points (1000). The 1000 simulation points are chosen by means of a heuristic optimization algorithm. Due to the large difference in size between the architectural space (196608 points) and the program transformation space (10 points), in the two-phase exploration strategies we always exhaustively searched the Transformation Space.

- *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 4 shows three approximate Pareto curves in the energy-delay space for the LU decomposition benchmark, that has been selected among the benchmarks to represent the energy-delay behavior. 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 clearly show a dominance of the results of the co-exploration with respect to $(S_T + S_A)$ and $(S_A + S_T)$ explorations in finding Pareto points.

Let us now consider the point in Figure 4 corresponding to the minimum delay imposed by $(S_T + S_A)$ exploration. The co-exploration provides a dominating point preserving the delay constraints, while reducing energy by 15% approximately.

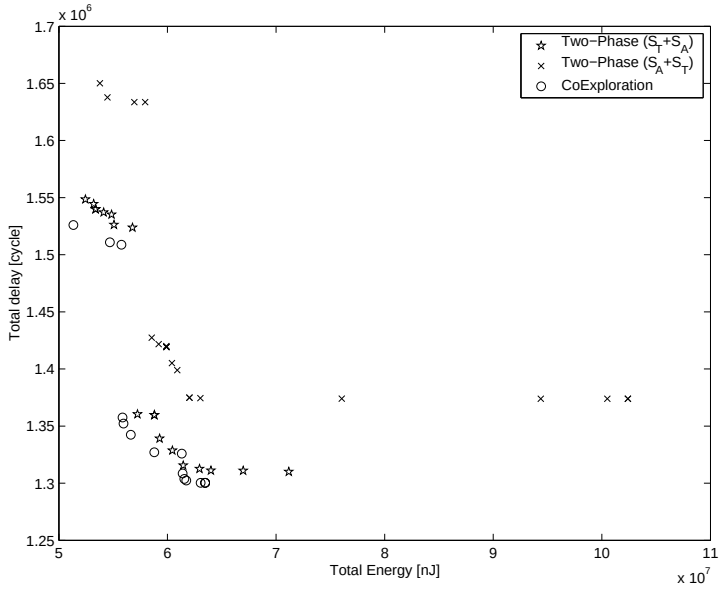

Fig. 4 Comparison between two–phase and co–exploration methodologies in terms of energy-delay Pareto points for the LU decomposition (`LUEQ`) benchmark.

To compare more precisely the approximate Pareto curves derived from different exploration techniques, we applied the *Two–set coverage* metric defined in [33] to the results obtained for all the benchmark suite. 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.

$$C(X', X'') := \frac{|\{\alpha'' \in X''; \exists \alpha' \in X' : \alpha' \succeq \alpha''\}|}{|X''|} \tag{13}$$

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

Table 2 reports the two–set coverage metric (given by Equation 13) computed for all the selected benchmarks. Since the metric is asymmetric, it has been computed for co-exploration with respect to two–phase $(S_A + S_T)$ and vice versa (see second and third columns of Table 2 respectively), and for co-exploration with respect to two–phase $(S_T + S_A)$ and vice versa (see fourth and fifth columns respectively). For 4 over 8 benchmarks, the co-exploration outperforms both $(S_A + S_T)$ and $(S_T + S_A)$ explorations. For these 4 benchmarks, both two-phase explorations do not add any approximated Pareto points with respect to the approximated Pareto curve obtained by the co-exploration. Although for the other 4 benchmarks the co-exploration has a lower coverage, it adds some useful points to the approximate Pareto curve.

Table 2 Comparison of Pareto curves: Two–set coverage of the Co–Exploration (Co) with respect to Two-Phase Explorations (TP) for different benchmarks.

	$S_A + S_T$		$S_T + S_A$	
Benchmark	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
FIDCT	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 3 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
FIDCT	0	0
FIR1	10	5.90
FIR2	0	0
GAMMA	4.40	3.29
LUEQ	4.93	1.03
Average	4.34	1.93

To outline the effects of the *Loop Unrolling* transformation on each benchmark in Table 2, the average unroll factors corresponding to the Pareto points for the different benchmarks and their standard deviations are shown in Table 3. 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, $FIDCT$ and $FIR2$), the co-exploration is outperformed, since the two-phase explorations analyze a reduced design space with respect to the co-exploration.

6.2 Characterization and Validation of Analytical Models

In this section, we provide results on the characterization and validation phases of the proposed energy/delay analytical models by applying our co-exploration framework to the selected set of benchmarks.

In these experiments, we limited the exploration to a 8–dimensional subset of the full System Level Design Space, ($S_{ES} \subset S_{SD}$). *Loop Unrolling* and *Loop Tiling* have been chosen among the possible optimization passes to reduce the number of processor cycles and energy, while among the possible hardware parameters, we have selected the parameters related to L1

caches to exploit data and instruction locality. Globally, we considered $S_A = S_{s_i} \times S_{s_d} \times S_{b_i} \times S_{b_d} \times S_{a_i} \times S_{a_d}$ and $S_T = S_u \times S_t$ where:

- $\mathbf{S_{s_i}} = \{2KB, 4KB, 8KB, 16KB\}$ sizes of the L1 instruction cache;
- $\mathbf{S_{s_d}} = \{2KB, 4KB, 8KB, 16KB\}$ sizes of the L1 data cache;
- $\mathbf{S_{b_i}} = \{16B, 32B\}$ block sizes of the L1 instruction cache;
- $\mathbf{S_{b_d}} = \{16B, 32B\}$ block sizes of the L1 data cache;
- $\mathbf{S_{a_i}} = \{1-way, 2-way\}$ associativity values of the L1 instruction cache;
- $\mathbf{S_{a_d}} = \{2-way, 4-way\}$ associativity values of the L1 data cache;
- $\mathbf{S_u} = \{< t_u, k > | k \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}\}$ where k is the unroll factor;
- $\mathbf{S_t} = \{< t_t, t > | t \in [10, 180]\}$ where t is the tile size varying by increment of 5.

Considering the variation of the above parameters, the System-Level Design Space has a cardinality $|S_{SD}| = |S_A \times S_T| = 3648 \times 340 = 1240320$.

First, the co-exploration framework has been applied to the selected set of benchmarks to derive the coefficients of the analytical equations by simulating a small number of points in the co-design space. To set up the characterization phase, the number of points to be simulated has been determined by trading off the accuracy. For all the benchmarks, Figure 5 shows the variation of the energy and delay average error of estimated with respect to simulated system-level metrics as the number of simulated points used in the model characterization phase increases up to 280. The energy and delay average errors become steady by considering on average 60 simulation points over a design space composed of 1240320 points.

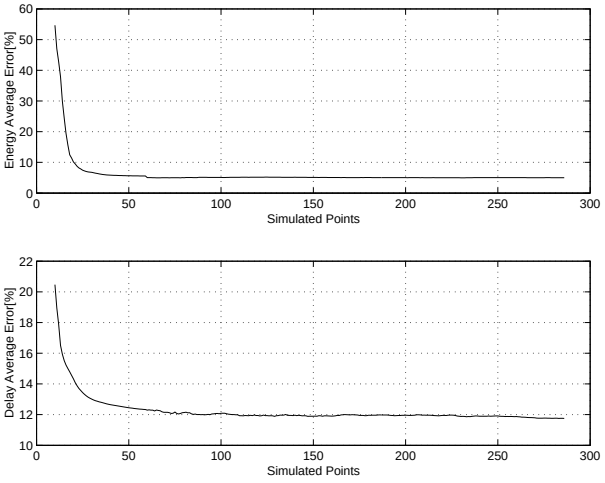

Fig. 5 Energy and delay average error of estimated vs. simulated metrics by varying the number of simulated points used in the *Model Characterization* phase.

Therefore, in the following characterization experiments, we considered 60 randomly chosen simulation points to evaluate the model coefficients for

the set of selected benchmarks. Accuracy data have been collected for all the selected benchmarks, reporting an average error of the proposed system-level analytical models with respect to simulation-based approach of 5% and 12.3% in terms of energy and delay respectively. The accuracy values depend on the accuracy of the underlying SimpleScalar/Wattch simulators [6, 7]. However, as representative examples of the accuracy reached during the characterization phase, the scatter plots in Figures 6, 7, and 8 show the agreement between measured and predicted energy and delay values for the characterization process of the system-level analytical models just related to Gamma, FDCT, and DCT_IDCT benchmarks respectively. In particular, for Gamma results in Figure 6, the average (maximum) error of the proposed analytical models with respect to the simulated values is within 1.11% (2.78%) in terms of energy, while for the estimated delay, the average (maximum) error is within 0.54% (1.29%). For the FDCT benchmark in Figure 7, the average (maximum) errors are respectively within 2.21% (6.17%) for the energy, and 1.86% (4.91%) for the delay. For the DCT_IDCT results in results in Figure 8, the average (maximum) errors are respectively within 3.76% (10.84%) for the energy, and 3.34% (8.27%) for the delay.

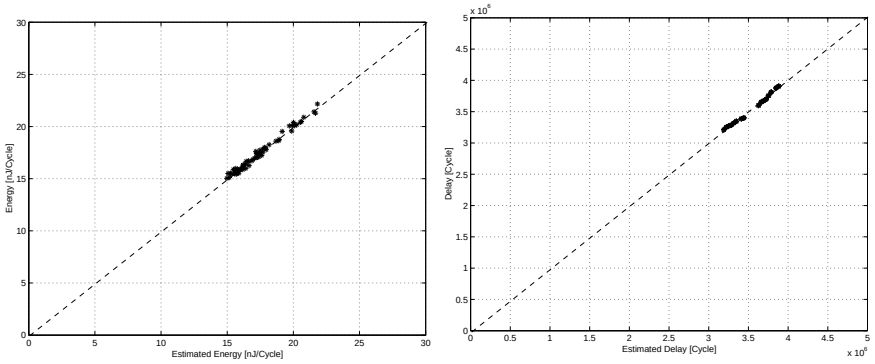

Fig. 6 Model characterization phase: Agreement between simulated energy and delay values with respect to estimated values by analytical models for GAMMA benchmark.

To provide evidence on the speedup of the co-exploration phase obtained by the introduction of analytical models, Table 4 compares the co-exploration times required by three different approaches:

- Exhaustive co-exploration through simulation of each point of the compiler/architecture design space;
- PSA-based co-exploration through simulation of each point of the Partial Pareto Set;
- Co-exploration through analytical models.

The co-exploration time (last row of Table 4) has been approximated as the number of simulated points times the average simulation time for each point. The time required to simulated a single point of the design space (1

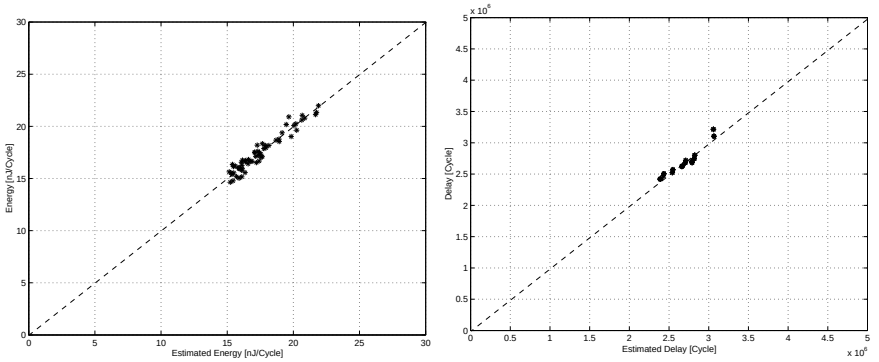

Fig. 7 Model characterization phase: Agreement between simulated energy and delay values with respect to estimated values by analytical models for FDCT benchmark.

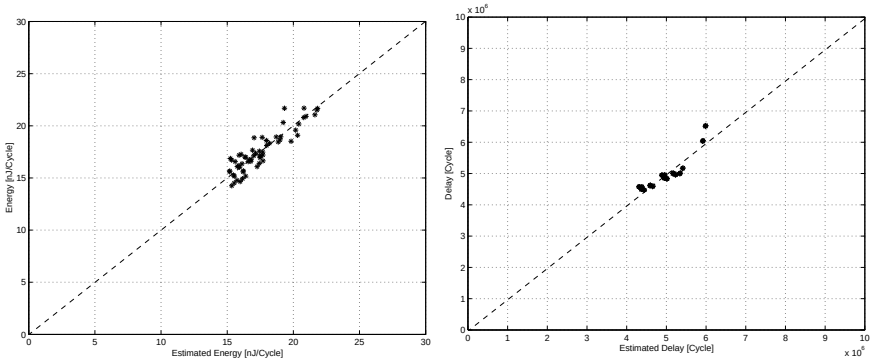

Fig. 8 Model characterization phase: Agreement between simulated energy and delay values with respect to estimated values by analytical models for DCT_IDCT benchmark.

minute) has been averaged over the selected benchmarks running on a Intel P4 1.7 GHz PC with 1GB RAM (but this value can be considered as purely indicative). For the exhaustive co-exploration, the number of simulated points is equal to the cardinality of the system-level design space (1240320), while for the PSA-based co-exploration the number of design points to be simulated has been reduced to 18392. For the analytical approach, the number of simulated points (60) has been restricted to the set of points imposed by the model characterization effort. The exhaustive co-exploration represents an unfeasible approach requiring 29 months to simulate hundred of thousand points, but the co-exploration time can be drastically reduced to 1 hour for the analytical approach. The results demonstrate how the introduction of an optimization technique such as PSA-based can reduce the co-exploration of more than one order of magnitude, while the analytical co-exploration can reduce up to more than three orders of magnitude.

To validate the proposed analytical approach, we applied our framework to all benchmarks to compare the results of the simulation of a sub-set of

Table 4 Comparison of Different Co–Exploration Times.

	Co-exploration		
	Exhaustive	PSA-based	Analytical
Number of explored points	1240320	18392	1240320
Number of simulated points	1240320	18392	60
Ave. single simulation time	1 min	1 min	1 min
Co-exploration time	29 months	13 days	1 hour

design space points to the estimates obtained by the analytical models, since it was unfeasible to simulate the whole set of 1240320 points. A sub-set composed of 300 points has been randomly selected over the whole set of 1240320 points where the 60 points used during the model characterization phase have been excluded. While accuracy data have been collected for all the selected benchmarks, reporting an average error of 6.7% and 13% in terms of energy and delay respectively, the scatter plots in Figures 9, 10, and 11 show the agreement between measured and estimated energy and delay values for the proposed system-level analytical models related to Gamma, FDCT, and DCT_IDCT benchmarks. In particular, for Gamma results in Figure 9, the average (maximum) error of the proposed analytical models with respect to the simulated values is within 2.14% (8.92%) in terms of energy, while for the estimated delay, the average (maximum) error is within 0.54% (1.60%). For the FDCT benchmark in Figure 10, the average (maximum) errors are respectively within 3.00% (13.12%) for the energy, and 1.86% (5.26%) for the delay. For the DCT_IDCT results in results in Figure 11, the average (maximum) errors are respectively within 4.23% (17.60%) for the energy, and 3.30% (8.54%) for the delay.

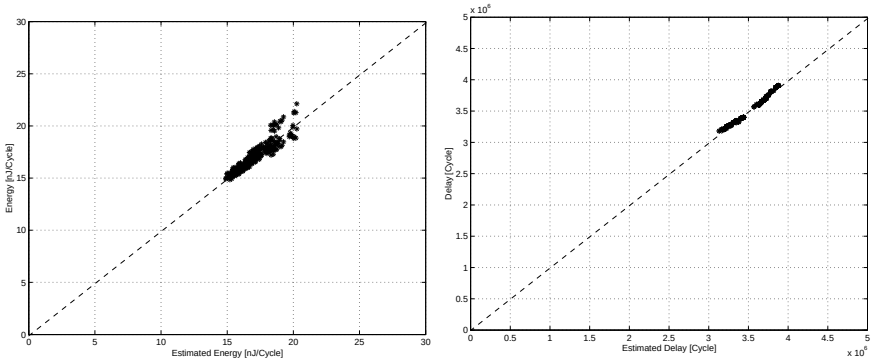

Fig. 9 Model validation phase: Agreement between simulated energy and delay values with respect to estimated values by analytical models for GAMMA benchmark.

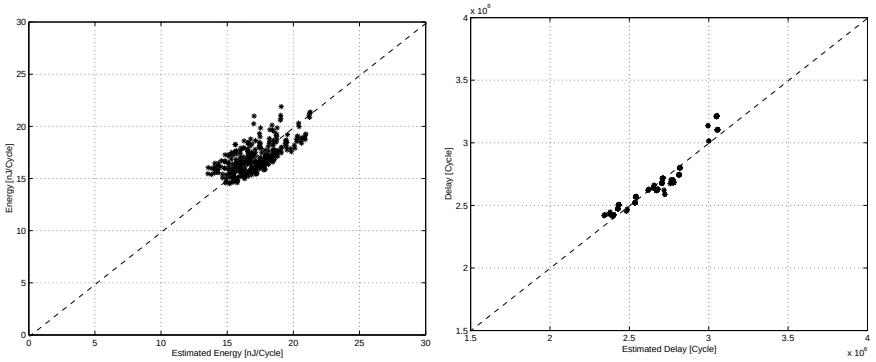

Fig. 10 Model validation phase: Agreement between simulated energy and delay values with respect to estimated values by analytical models for FDCT benchmark.

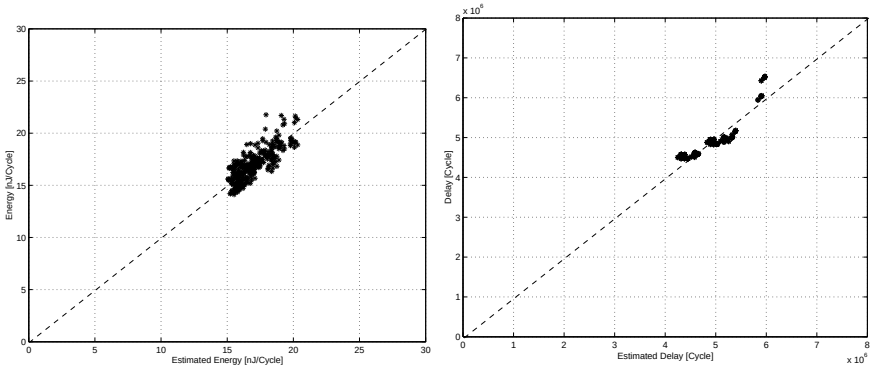

Fig. 11 Model validation phase: Agreement between simulated energy and delay values with respect to estimated values by analytical models for DCT_IDCT benchmark.

7 Conclusions and Future Works

This paper faces the problem of architecture/compiler co-exploration of embedded Systems-on-Chip to evaluate energy/delay trade-offs. The first goal of this paper is to demonstrate how the combined architecture/compiler co-exploration is more effective than a traditional separate exploration of source code transformation and architectural parameters. The second goal of this work is to propose a framework to speed up the co-exploration phase. The methodology is based on an analytical approach in which the energy/delay behavior of the system has been described in terms of system-level parameters (including source code transformation parameters). After the characterization of the coefficients of the analytical models, the parameterized models are then used to rapidly predict the energy/delay system behavior for each possible configuration of the large co-design space. The validation results of the proposed approach show a speedup of more than three orders of magnitude

with respect to traditional simulation-based co-exploration while preserving accuracy within 6.7% for the energy and 13% for the delay.

Further evolutions of the present work are focused on the evaluation of a larger class of source code transformations together with the evaluation of possible correlation between code transformations and architectural parameters. Concerning the definition of the analytical models, we initially focused the architectural space analysis on the parameters related to the I and D caches. In the next future, we plan to extend the analytical models to other classes of the system-level parameters such as the issue rate and number of parallel function units.

References

1. N. Vijaykrishnan et al., "Evaluating Integrated Hardware-Software Optimizations Using a Unified Energy Estimation Framework," *IEEE Trans. on Computers*, vol. 52, no. 1, pp. 59–76, January 2003.
2. M. Kandemir et al., "Influence of Compiler Optimizations on System Power," *IEEE Trans. on VLSI Systems*, vol. 9, no. 6, pp. 801–804, December 2001.
3. Dirk Fischer, Jurgen Teich, Michael Thies, and Ralph Weper, "Efficient architecture/compiler co-exploration for ASIPs," in *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, New York, NY, USA, 2002, pp. 27–34, ACM Press.
4. Giovanni Agosta, Gianluca Palermo, and Cristina Silvano, "Multi-objective co-exploration of source code transformations and design space architectures for low-power embedded systems," in *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, Nicosia, Cyprus, 2004, pp. 891–896, ACM Press, New York, NY, USA.
5. 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 Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 2, pp. 129–137, Apr. 2000.
6. T. M. Austin D. Burger and S. Bennett, "Evaluating Future Microprocessors: The SimpleScalar Tool Set," Tech. Rep. CS-TR-1996-1308, University of Wisconsin, 1996.
7. V. Tiwari D. Brooks and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proceedings ISCA2000*, pp. 83–94, 2000.
8. 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.
9. 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*, vol. 21, no. 11, pp. 1317–1327, November 2002.
10. M. Palesi and T Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign, 2002. CODES 2002*, May 6–8 2002.
11. Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria, "A flexible framework for fast multi-objective design space exploration of embedded systems.," in *PATMOS*, 2003, pp. 249–258.
12. D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler Transformations for High-Performance Computing," *ACM Computing Surveys*, vol. 26, no. 4, pp. 345–420, 1994.
13. L. Benini and G. De Micheli, "System-Level Power Optimization Techniques and Tools," *ACM TODAES*, vol. 5, no. 2, pp. 115–192, 2000.
14. G. De Micheli T. Simunic, L. Benini and M. Hans, "Source-Code Optimization and Profiling of Energy Consumption in Embedded Systems," *Proc. of ISSS00*, pp. 193–198, 2000.

15. V. Tiwari, S. Malik, A. Wolfe, and M. Lee, "Instruction level power analysis and optimization of software," *J. VLSI Signal Processing*, pp. 1–18, 1996.
16. P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg, "Data and memory optimization techniques for embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 6, no. 2, pp. 149–206, 2001.
17. Eui-Young Chung, Luca Benini, and Giovanni De Micheli, "Automatic source code specialization for energy reduction," in *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, New York, NY, USA, 2001, pp. 80–83, ACM Press.
18. Bjorn Franke, Michael O'Boyle, John Thomson, and Grigori Fursin, "Probabilistic source-level optimisation of embedded programs," in *LCTES'05: Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, New York, NY, USA, 2005, pp. 78–86, ACM Press.
19. 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*, vol. 9, no. 4, August 2001.
20. L. Benini E.-Y. Chung and G. De Micheli, "Source Code Transformation based on Software Cost Analysis," *Proc. of ISSS2001*, pp. 153–158, 2001.
21. Carlo Brandolese, William Fornaciari, Fabio Salice, and Donatella Sciuto, "Analysis and modeling of energy reducing source code transformations.," in *DATE*, 2004, pp. 306–311.
22. Y. Li and J. Henkel, "A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems," *DAC-35*, June 1998.
23. 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*, vol. 8, no. 3, June 2000.
24. Andreas Wieferink, Tim Kogel, Rainer Leupers, Gerd Ascheid, Heinrich Meyr, Gunnar Braun, and Achim Nohl, "A system level processor/communication co-exploration methodology for multi-processor system-on-chip platforms," in *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, Washington, DC, USA, 2004, p. 21256, IEEE Computer Society.
25. Ashok Halambi, Peter Grun, Vijay Ganesh, Asheesh Khare, Nikil Dutt, and Alex Nicolau, "Expression: a language for architecture exploration through compiler/simulator retargetability," in *DATE '99: Proceedings of the conference on Design, automation and test in Europe*, New York, NY, USA, 1999, p. 100, ACM Press.
26. Aviral Shrivastava, Nikil Dutt, Alex Nicolau, and Eugene Earlie, "Pbexplore: A framework for compiler-in-the-loop exploration of partial bypassing in embedded processors," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, Washington, DC, USA, 2005, pp. 1264–1269, IEEE Computer Society.
27. R. P. Wilson et al., "SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers," *SIGPLAN Notices*, vol. 29, no. 12, pp. 31–37, 1994.
28. Czyak P. and Jaszkiewicz A., "Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimisation," *Journal of Multi-Criteria Decision Analysis*, , no. 7, pp. 34–47, April 1998.
29. Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria, "Multi-objective design space exploration of embedded systems," *Journal of Embedded Computing*, vol. 1, no. 3, pp. 305–316, 2005.
30. Tony D. Givargis and Frank Vahid, "Parameterized system design," in *Proceedings of the eighth international workshop on Hardware/software codesign.* 2000, pp. 98–102, ACM Press.
31. J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantative Approach, Third Edition*, Morgan Kaufmann Publisher Inc., CA, 2003.
32. Ann Gordon-Ross and Frank Vahid, "Frequent loop detection using efficient nonintrusive on-chip hardware," *IEEE Transactions on Computers*, vol. 54, no. 10, pp. 1203–1215, 2006.

33. E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.