



A Sensitivity-Based Design Space Exploration Methodology for Embedded Systems

WILLIAM FORNACIARI

Politecnico di Milano, Dip. di Elettronica e Informazione, Milano, Italy 20133

fornacia@elet.polimi.it

DONATELLA SCIUTO

Politecnico di Milano, Dip. di Elettronica e Informazione, Milano, Italy 20133

sciuto@elet.polimi.it

CRISTINA SILVANO

Università degli Studi di Milano, Dip. di Scienze dell'Informazione, Milano, Italy 20135

silvano@dsi.unimi.it

VITTORIO ZACCARIA

Politecnico di Milano, Dip. di Elettronica e Informazione, Milano, Italy 20133

zaccaria@elet.polimi.it

Abstract. In this paper, we propose a system-level design methodology for the efficient exploration of the architectural parameters of the memory sub-systems, from the energy-delay joint perspective. The aim is to find the best configuration of the memory hierarchy without performing the exhaustive analysis of the parameters space. The target system architecture includes the processor, separated instruction and data caches, the main memory, and the system buses. To achieve a fast convergence toward the near-optimal configuration, the proposed methodology adopts an iterative local-search algorithm based on the sensitivity analysis of the cost function with respect to the tuning parameters of the memory sub-system architecture. The exploration strategy is based on the Energy-Delay Product (EDP) metric taking into consideration both performance and energy constraints. The effectiveness of the proposed methodology has been demonstrated through the design space exploration of a real-world case study: the optimization of the memory hierarchy of a MicroSPARC2-based system executing the set of Mediabench benchmarks for multimedia applications. Experimental results have shown an optimization speedup of 2 orders of magnitude with respect to the full search approach, while the near-optimal system-level configuration is characterized by a distance from the optimal full search configuration in the range of 2%.

Keywords: Design space exploration, power and performance optimization.

1. Introduction

Many multimedia applications implemented today on portable embedded systems have stringent real-time constraints together with the necessity of minimizing power consumption (e.g., portable MP3 decoders and digital cameras). The increasing complexity of today's embedded systems is forcing the designers to determine an optimal system structure as early as possible in the design process due to the strictly time-to-market requirements. The selection of the most important modules of the system and their conformation with respect to the application and the possible alternatives must be performed at the highest levels of abstraction. In fact, it is impossible in terms of design

time to synthesize and analyze each part of the design space, even considering the optimization of only two parameters such as power and performance. Furthermore, at the highest abstraction levels, several degrees of freedom can be exploited to achieve a comprehensive design optimization while maintaining a good level of accuracy of the estimated parameters. Accuracy and efficiency must be traded-off to meet the system-level overall energy and delay requirements, avoiding costly re-design phases.

Decreasing energy consumption without significantly impacting performance is a must during the design of a broad range of high-end embedded portable applications. Evaluation of energy-delay metrics at the system-level is of fundamental importance for embedded applications characterized by low-power and high-performance requirements. The capability of collecting a direct feed-back on the impact of different design options at the highest abstraction levels implies a shorter development time, avoids costly design re-cycles, while enabling the possibility to early re-target the architectural design choices.

Aim of this work is to study a new system-level design methodology for the efficient exploration of the architectural parameters for application-specific multimedia systems, characterized by stringent energy and delay constraints. For this class of systems, we are focusing on the design exploration of the architectural parameters for the memory sub-systems, mainly cache size, cache block size, and set associativity. The exploration strategy is based on the Energy-Delay Product (EDP) metric taking into consideration both performance and energy constraints. In particular, we show how the given metric is affected by the variation of cache parameters and we prove the importance of including energy in the given metric, since an increase in the cache size and block size can reduce the number of cycles but it does not necessarily reduce the energy.

The system-level architecture we are focusing on in the paper includes separate instruction and data level-one caches. The target of our work is to find a possibly optimal or near-optimal configuration of the I-cache and D-cache without performing the exhaustive analysis of the space of the chosen parameters (mainly cache size, block size and associativity). The paper proposes a heuristic method to reduce the simulation time, discarding from the analysis the less promising system configurations. The method is based on the sensitivity analysis of the system behavior with respect to the most relevant system-level parameters. In such a way, the resulting design exploration phase cost increases linearly with respect to the design space size.

The cornerstone of our strategy is the dynamic profiling of the memory references obtained by tracing the software execution in terms of transition activity on system-level buses and activation of the system resources. Bus traces, derived from the execution of several application programs by an Instruction Set Simulator and filtered by a behavioral model of the caches, are then analyzed from the energy-delay combined perspective to evaluate the cost associated with different architectural configurations of the system caches.

The effectiveness of the proposed methodology has been validated by applying it to the design optimization of the memory sub-system for a real-world representative case study (a MicroSPARC2-based system) during the execution of the Mediabench suite [21], that can be considered representative of a wide class of embedded multimedia applications. Experimental results show the reduction of the number of alternatives analyzed during

the design exploration phase, while achieving either the same optimal system-level configuration obtained by the exhaustive system-level exploration or a sub-optimal system-level configuration with a maximum error of 1.67%.

The main contributions of this work can be summarized as follows:

- definition a system-level exploration methodology to quickly analyze different configurations of the memory sub-system from an energy /delay combined perspective;
- flexibility in terms of capability to include different levels in the memory hierarchy, where each level can be characterized by different architectures (mainly in terms of cache size, cache line size, and set associativity) and implementation technologies;
- high efficiency in terms of simulation time spent for the exploration phase with respect to the simulation time spent for the exhaustive exploration of the parameters space;
- good level of accuracy in the estimation of the energy-delay figure of merit associated with each point of the design space;

The paper is organized as follows. The next section presents the most relevant approaches for system-level exploration appeared in literature so far, while Section 3 proposes our design space exploration framework. The sensitivity-based optimization methodology is discussed in Section 4. The results derived from the application of the proposed methodology to the selected case study have been reported in Section 5. Finally, Section 6 summarizes the main contributions of this work and outlines the directions for our future research.

2. Background

Several system-level estimation and exploration methods have been recently proposed in literature targeting power-performance tradeoffs from the system-level standpoint. Among these works, the most significant methods related to our approach can be divided into two main categories: (i) system-level power estimation and exploration in general, and (ii) power estimation and exploration focusing on cache memories.

In the first category, the SimplePower approach [31] can be considered one of the first efforts to evaluate the different contributions to the energy budget at the system-level. The SimplePower exploration framework includes a transition-sensitive, cycle-accurate data-path energy model that interfaces with analytical and transition sensitive energy models for the memory and bus sub-systems. The SimplePower energy estimation environment consists of a compilation framework and an energy simulator that captures the cycle-accurated energy consumed by the SimpleScalar architecture, the memory system and the buses. In particular, the energy estimates related to the memory system includes the energy consumed by the I-cache and D-cache, the address and data buses,

the address and data pads and the off-chip main memory. Concerning the processor's energy model, SimplePower considers only the energy consumed by the core's data path, neglecting the control path. The SimpleScalar estimation framework has been designed to evaluate the effects of some high-level algorithmic, architectural and compilation trade-offs on energy. Although the proposed system-level framework is quite general, the exploration methodology reported in [31] is limited to a search over the space of the following parameters: cache size, block buffering, isolated sense amplifiers, pulsed word lines and eight different compilation optimizations (such as loop unrolling).

The *Avalanche* framework presented in [23] evaluates simultaneously the energy-performance tradeoffs for software, memory and hardware for embedded systems. The approach is based on system-level and takes into consideration the interdependencies between the various system modules. Their target system features the CPU, D-cache, I-cache and main memory, while the impact of buses is not accounted for. For this reason, the estimates will be less accurate in advanced submicron technologies, for which routing capacitances will play a significant role in the system's power consumption.

The work in [9] proposes a system-level technique to find low-power high-performance superscalar processors tailored to specific user applications. More recently, the Wattch architectural-level framework has been proposed in [5] to analyze power vs. performance tradeoffs with a good level of accuracy with respect to lower-level estimation approaches. Low-power design optimization techniques for high-performance processors have been investigated in [2] from the architectural and compiler standpoints.

A trade-off analysis of power/performance effects of SOC (System-On-Chip) architectures has been recently presented in [14], where the authors propose a simulation-based approach to configure the parameters related to the caches and the buses. The approach is viable only when we can model analytically or statistically (with a high level of accuracy) the energy and delay behavior of the system's modules. Moreover the reported results are limited to finite ranges of the following parameters: bus width, bus encoding, cache size, cache line size and set associativity. In [15] the same authors created a framework for the design space exploration where the optimization phase is based on parameter dependency models.

In the second category approaches dealing with power estimation and exploration for the memory hierarchy, the authors of [20] propose to sacrifice some performance to save power by filtering memory references through a small cache placed close to the processor (namely *filter cache*). A similar idea has been exploited in [3], where memory locations with the highest access frequencies are mapped onto a small, low-energy, and application-specific memory that is placed close to the core processor.

Power and performance tradeoffs in cache architectures have been also investigated in [1]. A model to evaluate the power/performance tradeoffs in cache design has been proposed in [29], where the authors discuss also the effectiveness of novel cache design techniques targeted for low-power (such as vertical and horizontal cache partitioning). An analytical power model for several cache structures has been proposed in [18]. The model accounts for technological parameters (such as capacitances and power supplies) as well as architectural factors (such as block size, set associativity and capacity). The process models are based on measurements reported in [32] for a 0.8 μm process

technology. The analytical model of energy consumption for the memory hierarchy has been extended in [17] and [28] where the cache energy model is included in a more general approach for the exploration of memory parameters for low-power embedded systems. The authors consider as performance metrics the number of processor cycles and the energy consumption. The paper shows how these metrics can be affected by some cache parameters such as cache size, cache line size, set associativity and tiling size, and off-chip data organizations.

Finally, a number of approaches deal with the code optimization in order to reduce the memory accesses and data placement in the memory hierarchy in such a way to minimize power consumption [4], [7], [6], [10], [25], [30].

The approach proposed in this paper is an extension of our previous work [13], [12] and it is an attempt to afford the problem of the system-level exploration approach from a power-performance comprehensive perspective. Taking into consideration the most relevant modules of the target embedded system architecture and integrating all the modules into an overall system-level simulation environment, our main goal is to explore the energy/performance effects of the variations of several parameters related to the memory subsystem like cache size, associativity and cache line size.

3. Design Space Exploration Framework

Among the factors influencing the success of an electronic product, time-to-market is becoming the most crucial one. To cope with the necessity of enabling a fast design space exploration while ensuring a good level of optimization in the final architecture, the use of customizable platforms is becoming more and more important [19]. However, even considering simple microprocessor-based architectures composed of a CPU and a memory hierarchy, the discovery of the optimal system configuration by trading off power and performance still leads to the analysis of too many alternatives. Our work aims at overcoming such problems by providing a methodology and a design framework to drive the designer towards optimal solutions in a cost-effective manner. To achieve such a goal, we increased the abstraction level of the analysis and we identified a suitable set of metrics to carefully predict both the application requirements and their impact on the architecture characteristics. The framework we developed is composed of several basic interacting elements, as shown in Figure 1.

The methodology that we propose is composed of three steps: definition of the system architecture, system characteristics extraction (or *tuning phase*) and design space exploration to determine the optimal system configuration. The main features of such activities are reported in the next subsections.

3.1. System-Level Description

The first step of the design flow is to capture the relevant characteristics of the embedded application, the customizable implementation platform and the designer's goals.

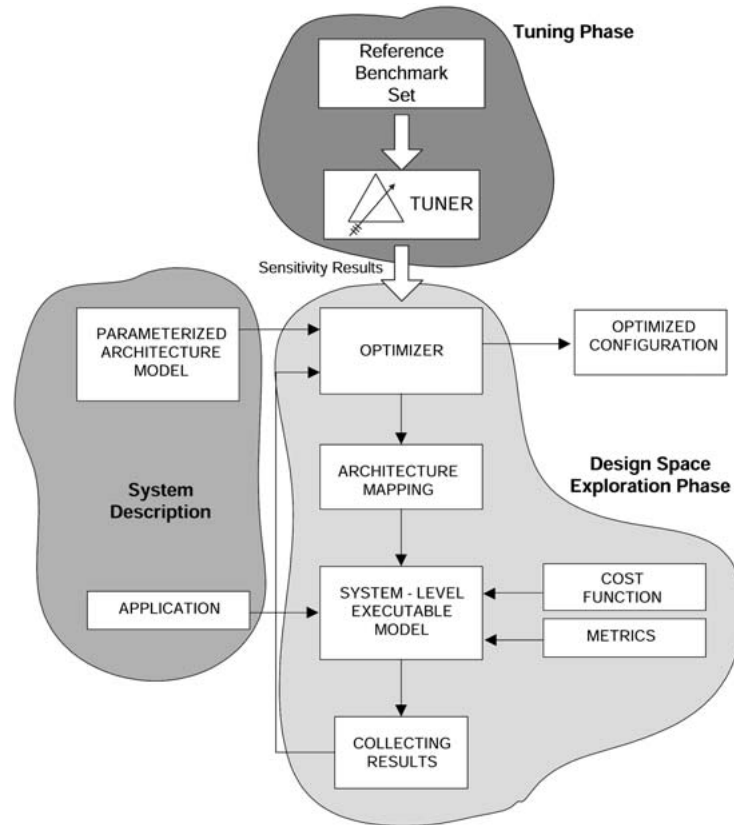


Figure 1. The proposed design space exploration flow.

Concerning the target architecture, the design space is defined in terms of the following parameters:

- Number of levels in the memory hierarchy;
- Cache configuration parameters;
- On-chip vs off-chip positioning of cache (L1 and L2);
- Unified vs separate data/instruction cache, at any hierarchy level;
- Width of address and data buses.
- Encoding strategy for the information flowing on both data and address buses.

In the case study reported in Section 5, we analyze the instruction and data cache parameters of the target architecture.

During this phase, the designer is required to specify the boundaries of the design space for each parameter (for example, the maximum size to be considered for the cache size) and other possible freezing constraints on some parameters.

3.2. *Design Space Exploration*

The tradeoff analysis at the system-level plays a primary role during the design space exploration to early re-target the choice of system parameters and speed up design time. Even if a unique and comprehensive goal function to be optimized is selected, an exhaustive analysis of the typical design space still remains an hard task. Let us consider a simple architecture, to be configured according to three parameters, for each one being possible 8 different values: the resulting number of possible configurations to be compared is over 500 for each of the applications to be considered for the system. In such cases, a *brute force* approach could be impractical if the time required by a single simulation is high. Thus some heuristics to perform a near-optimal search must be applied to find out acceptable solutions while avoiding impractical simulation times.

In this study, we propose a new heuristic algorithm to explore the design space of microprocessor-based platforms with a configurable memory hierarchy. The heuristic algorithm is implemented in the *optimizer* module of the framework (see Figure 1) that is responsible for the generation/evaluation of new alternatives of the system configuration in order to find the near-optimal one. The heuristic algorithm is driven by the characteristics of the system extracted by means of a sensitivity analysis performed during the tuning phase. Hence, the heuristic algorithm is called the *sensitivity-based search* algorithm.

The goal of the heuristic algorithm is to move towards a restricted set of candidate optimal configurations, by avoiding to perform the exhaustive analysis of the design space. This heuristic method must be tuned only once for a specific platform by analyzing its behavior while executing a representative set of benchmarks. The goal is to identify the impact of the different parameters on the cost function to be optimized by performing a comparative analysis of the magnitude of the discrete partial derivatives of the cost function. This information is used to efficiently drive the strategy of the sensitivity-based search engine, that will be used to optimize each new application for that specific platform. More in detail, the optimization strategy begins from a suitable starting point in the design space and analyzes sequentially the parameters in order of decreasing sensitivity.

In practice, instead of considering the simultaneous optimization of n parameters, requiring $O(k^n)$ simulations (where k is the average range of variability of each parameter), the heuristic explores sequentially the region of variation of the most important parameters producing near-optimal solutions (in terms of minimization of the cost function). In fact, the analysis considers one parameter at a time, according to the importance provided by the sensitivity analysis. The number of simulations, in the average case, will be linear with respect to the range of variations of each parameter and,

as we will show in the experimental results (see Section 5), the optimal solution is almost always identified thanks to the accurate sensitivity analysis performed during the tuning phase.

3.3. Computing the Cost of a Configuration

The *cost function* of a system provides a quantitative analysis of the system characteristics and provides a mean to optimize the system by discarding non-optimal system configurations and considering several metrics simultaneously. Here, both performance and power are considered relevant for an embedded system, and should be captured by a general and flexible goal function. These metrics have been extensively discussed since last decade [20], [16] and, based on this literature, we adopted the Energy-Delay Product (*EDP*) metric to compare alternative system designs.

To compute such a general goal function, we focus on the energy and the delay associated with the processor and memory parts of a system, considering the contributions of the processor core, the system-level buses and each level of the memory hierarchy. While the delay calculation is embedded in the cycle-accurate simulation environment (the *System Executable Model*, see Figure 1), the use of each system resource has to be extracted and imported into analytical 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*).

In our study the analytical energy models include the following contributions:

- processor core;
- processor I/O pads;
- processor-to-L1 on-chip buses;
- on-chip L1 I- and D-caches;
- L1-to-L2 off-chip buses;
- L2 unified *SRAM* cache;
- Main memory (*DRAM*).
- L2-to-*DRAM* off-chip buses;

All the energies and delays presented in this work are normalized with respect to the instructions executed. This is done in order to compare the behavior of different

applications on the same system architecture and to compare different architectures on the same application.

For the processor core, a suitable model or tool able to provide the power consumption at the instruction-level is needed [22], [26], [27]. In this version of the design environment, we used a simple instruction-level model that accounts for the average power consumed by the processor in normal operating conditions and we multiplied it by the average execution time of an instruction.

For the system-level buses (such as the processor-to- $L1$ buses), the energy model can be simply expressed as: $E \approx (C_{load} V_{dd}^2 n_{trans})/2$, where C_{load} is the bus load capacitance, V_{dd} is the power supply voltage and n_{trans} is the number of transitions on the bus lines, depending on the hamming distance between subsequent vectors the bus stream.

For on-chip $L1$ caches, we use the analytical energy model developed in [18], that accounts for:

- Technological parameters (such as capacitances and power supplies);
- Architectural parameters (such as block size and cache size);
- Switching activity parameters (such as number of bit line transitions).

This cache energy model has been used in recent works (such as in [20]), where the switching activity parameters have been computed either by using application-dependent statistics or by assuming typical values (such as half of the address lines switching during each memory request). In our framework, we directly import the actual values of hit/miss rates and bit transitions on cache components, that have been derived by the system-level simulation environment to account for the actual profiling information depending on the execution of embedded applications.

Finally the characterization of the main memory has been carried out by directly importing information derived from the data-sheet into our model.

4. Sensitivity-Based Optimization

The optimization methodology we are proposing is based on the *sensitivity* analysis of the system with respect to the set of configuration parameters. Before detailing the sensitivity based optimization, let us introduce the notion of architectural space and the sensitivity associated with the cost function.

4.1. Architectural Space Definition

The architectural optimization algorithm that we are proposing is a local search algorithm whose target is to find a configuration (or *point*) in the *architectural space* whose cost is (locally) minimum.

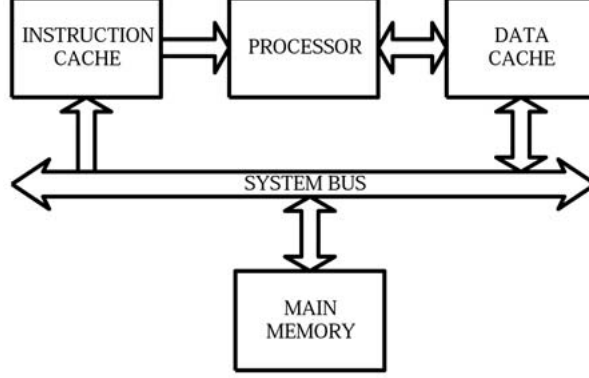


Figure 2. The target system architecture.

Given a set $P = \{p_1, \dots, p_n\}$ of n architectural parameters, the architectural space \mathcal{A} can be defined as:

$$\mathcal{A} = S_{p_1} \times \dots \times S_{p_1} \dots \times S_{p_n} \quad (1)$$

where S_{p_i} is the ordered set of possible configurations for parameter p_i and “ \times ” is the cartesian product. As an example of architectural space, let us consider the configurable system architecture shown in Figure 2 composed of a processor, a separated I- and D-L1 caches, the main memory and the system buses. Let us consider the exploration of the design space of this architecture in terms of six parameters: cache size, block size and associativity of both D- and I-L1 caches. In this case, each instance of the configurable architecture is described as a 6-tuple $[c_i, b_i, v_i, c_d, b_d, v_d] \in \mathcal{A} = S_{c_i} \times S_{b_i} \times S_{v_i} \times S_{c_d} \times S_{b_d} \times S_{v_d}$ where:

- S_{c_i}, S_{c_d} are the ordered sets of the possible sizes of the I- and D-caches (e.g., $\{16\text{KB}, \dots, 256\text{KB}\}$).
- S_{b_i}, S_{b_d} are the ordered sets of the possible block sizes of the I- and D-caches (e.g., $\{16\text{B}, \dots, 64\text{B}\}$).
- S_{v_i}, S_{v_d} are the ordered sets of the possible associativity values of the I- and D-caches (e.g., from direct mapped to 8-way set associative).

Assuming that each set S of possible configurations is discrete and ordered ($S = \{s_1, s_2, \dots, s_o \dots, s_q\}$), we can define a *neighborhood* operator “ \rightarrow ”:

$$s_o \rightarrow n = \begin{cases} s_{o+n} & 1 \leq o+n \leq q \\ \perp & \text{otherwise} \end{cases} \quad (2)$$

where “ \perp ” stands for “indefinite value.” As an example of application of the neighborhood operator, if we consider the following I-Cache size space:

$$S_{c_i} = \{16\text{KB}, 32\text{KB}, 64\text{KB}, 128\text{KB}, 256\text{KB}\} \quad (3)$$

we have that $(64\text{KB} \rightarrow 1) = 128\text{KB}$ while $[16\text{KB} \rightarrow (-1)] = \perp$.

The concept of the neighborhood operator is used to define positive and negative *perturbations* of a configuration. Given an architecture $a = [a_{p_1} \dots a_{p_l} \dots a_{p_n}] \in \mathcal{A}$ and a parameter p , we define a perturbation $\delta_p(a, y)$ as a vector $b = [b_{p_1} \dots b_{p_l} \dots b_{p_n}] \in \mathcal{A}$ where

$$b_{p_l} = \begin{cases} a_{p_l} & p_l \neq p \\ a_{p_l} \rightarrow y & p_l = p. \end{cases} \quad (4)$$

We indicate $\delta_p^+(a)$ for $\delta_p(a, 1)$ and $\delta_p^-(a)$ for $\delta_p(a, -1)$. In other words, a perturbation is an operation that modifies only one element of the configuration at a time (i.e., the element corresponding to p), by applying the neighborhood operator.

In the following section, we introduce the problem of finding the architectural optimal configuration with respect to a defined cost function and its solution through the sensitivity-based optimization algorithm.

4.2. Sensitivity of the Cost Function

The aim of this work is the definition of an efficient algorithm to find a suitable configuration of a given architecture that is optimized for performance and energy consumption. To assess the optimization problem, we need to introduce a cost function associated with the architecture under consideration and to the application to be executed. A cost function is defined as a function $f_M : \mathcal{A} \times \mathcal{K} \rightarrow \mathbb{R}$, where \mathcal{K} is the set of applications that can be executed on the architecture, \mathbb{R} is the set of real numbers representing the cost of the given system configuration and M is the machine model under consideration. The machine model is used to differentiate among the cost functions associated with different families of architectures (e.g., $M = \text{“SunSPARC”}$ or “x86”).

In this paper, we assume that the cost function is expressed as the product of two fundamental parameters, i.e., energy and delay:

$$f_M(a, k) = \text{Energy}_M(a, k) \times \text{Delay}_M(a, k) \quad (5)$$

where $\text{Energy}_M(a, k)$ is the energy dissipated by the system in configuration a during the execution of application k and $\text{Delay}_M(a, k)$ is the number of cycles necessary to the

system in configuration a to execute k . Since both energy and delay associated with the overall system can be hardly expressed by analytical functions, we adopt an executable model (i.e., a simulation-based model) to compute the values of the energy and delay parameters associated with each point of the parameter space.

The basic idea of our work is to exploit the influence of the architectural parameters on the cost function to minimize the simulations to be carried out during the optimization phase. In this paper, we propose the concept of *sensitivity* to express the variability of the cost function in the neighborhood of the minimum for the given set of reference benchmark applications. The reference benchmarks have been chosen in order to stress the functionality of the architecture in terms of memory accesses, use of function units and variability of all the relevant parameters.

The sensitivity of the cost function with respect to a parameter p and a set of reference applications \mathcal{H} is defined as:

$$\sigma(p, f_M) = E_{h \in \mathcal{H}} \left[\frac{\Delta_p(f_M(a_{opt}(h), h))}{f_M(a_{opt}(h), h)} \right] \quad (6)$$

where E is the mean operator over the set of reference applications \mathcal{H} , $a_{opt}(h)$ is the full search optimal configuration found for h , M is the machine model and $\Delta_p(f(a, h))$, is defined as:

$$\Delta_p(f(a, h)) = |f(a, h) - \max(f(\delta_p^+(a), h), f(\delta_p^-(a), h))|; \quad (7)$$

Equation 6 defines the average percentage variation of the cost function with respect to a small perturbation from its minimum. More in detail, the variation $\Delta_p(f(a, h))$ is the maximum absolute variation due to a small positive or negative perturbation on the optimum of the cost function for a given application h .

4.3. Sensitivity-Based Local Search Algorithm

Our main goal is to efficiently identify an optimal (or sub-optimal) configuration that minimizes the cost function. This is achieved by taking into account not only the architectural parameters and their influence on the cost function, but also the target application that will be executed on the embedded system.

Given the architectural space \mathcal{A} , the application k and the cost function f_M , the optimization problem can be stated as

$$\text{find } a_{opt} \text{ such that } f(a_{opt}, k) = \min_{\forall a \in \mathcal{A}} f_M(a, k). \quad (8)$$

To solve the problem expressed by Equation (8), we define an optimization methodology based on two steps. The first step is a tuning phase in which the system under consideration is analyzed in terms of *sensitivity*. The second step consists of the actual local search algorithm, that exploits the sensitivity information to perform the efficient exploration of

```

TuningPhase( $f_M, \mathcal{H}, P$ )
{
  Input:  $f_M$  = cost function derived from machine model
          $\mathcal{H}$  = set of reference applications to be optimized;
          $P$  = set of configuration parameters;
  Output:  $\sigma$  = set of sensitivity values for each parameter;

  foreach  $h \in \mathcal{H}$ 
  {
     $a_{opt}(h)$  = find full search optimum of  $h$ , given  $f_M$ ;
    foreach  $p \in P$ 
    {
       $\sigma(p, f_M) = \sigma(p, f_M) + \left[ \frac{\Delta_p(f_M(a_{opt}(h), h))}{f_M(a_{opt}(h), h)} \right] \times \frac{1}{|H|}$ 
    }
  }
}

```

Figure 3. The algorithm used during the tuning phase.

```

Sensitivity BasedLocalSearch( $f_M, k, \sigma, P, a_0$ )
{
  Input:  $f_M$  = cost function derived from machine model
          $k$  = application to be optimized;
          $\sigma$  = vector of sensitivity values;
          $P$  = set of configurable parameters;
  Output:  $a_{opt}$  = optimal configuration for application  $k$ ;

  foreach  $p \in L$  /*visit the parameters in order of sensitivity*/
  {
    find  $n$  such that  $f_M(\delta_p(a_{opt}, n), k) \leq f_M(\delta_p(a_{opt}, y), k), \forall y$ ;
     $a_{opt} = \delta(a_{opt}, n)$ ;
  }
  } until improvements on  $f_M$  are negligible
}

```

Figure 4. The sensitivity-based local search algorithm.

the architecture. More precisely, the tuning phase is applied only once for a particular set of reference applications \mathcal{H} and a machine model M , while the local-search algorithm can be applied to any new application to be optimized on the system.

The basic steps of the tuning phase are shown in Figure 3. Given a set \mathcal{H} of reference applications, the tuning consists of performing a full architectural space exploration for each $h \in \mathcal{H}$, by computing the finite difference Δ_p and finally the average sensitivity for each parameter p of the architecture. A set of reference applications has to be selected to reflect the typical behavior of the given architecture in terms of accesses to the memory hierarchy and to the function units.

Once the sensitivity $\sigma(p, f_M)$ has been characterized, the sensitivity-based local search algorithm reported in Figure 4 can be applied to any new application for which the optimal architecture must be found. The algorithm receives as inputs:

- the cost function f_M and its sensitivity information σ ;
- the application k to be optimized;
- the starting point for the exploration phase a_0 .

The starting point is computed as the average of the optimal architectures found during the tuning phase:

$$a_0 = E_{h \in \mathcal{H}} [a_{opt}(h)]. \quad (9)$$

The sensitivity-based algorithm selects iteratively one parameter at a time in order to decrease the sensitivity and to optimize the cost function along the dimension corresponding to that parameter (by means of acceptable perturbations). The stopping criterion is defined as the point beyond which further improvements to the cost function result in negligible advantages.

As a clarifying example, let us consider an application k running on a system with only three variable parameters (I-cache size c_i , I-cache block size b_i and I-cache associativity v_i) and let us assume that the sensitivity analysis has shown that the cost function f_M is affected first by c_i , second by v_i and third by b_i . Given a suitable a_0 (see Equation 9), the optimization algorithm executes the following steps:

1. $a_{opt} = a_0$
2. $oldcost = f_M(a_{opt}, n)$;
3. fix the component b_i and v_i of a_{opt} and find the optimal c_i such that $f_M(a_{opt}, k)$ is minimized;
4. fix the component c_i and b_i of a_{opt} and find the optimal v_i such that $f_M(a_{opt}, k)$ is minimized;
5. fix the component c_i and v_i of a_{opt} and find the optimal b_i such that $f_M(a_{opt}, k)$ is minimized; $newcost = f_M(a_{opt}, n)$.
6. If $\left(\frac{newcost - oldcost}{oldcost} > threshold \right)$ go back to step 2.

where *threshold* is a fixed value that defines the stopping criterion for the search of the local optimum.

Therefore, for each new application, the result of the sensitivity analysis of the system will be preserved, so that instead of considering the simultaneous optimization of p parameters only a narrow region of possible configurations producing sub-optimal solutions will be explored.

The sensitivity based optimization requires a number of simulations equal to:

$$g \times \sum_{p_i \in P} |S_{p_i}| \quad (10)$$

where P is the set of configurable parameters, S_{p_i} is the ordered set of configurations associated with p_i and g is the number of iterations over the whole set of parameters of the algorithm.

The speed-up with respect to a full-search algorithm is thus equal to:

$$\text{Speedup} = \frac{|\mathcal{A}|}{g \times \sum_{p_i \in P} |S_{p_i}|} = \frac{\prod_{p_i \in P} |S_{p_i}|}{g \times \sum_{p_i \in P} |S_{p_i}|}. \quad (11)$$

The obtained speed-up represents a relevant improvement in terms of simulation time if the number g of iterations remains low. As a matter of fact, by choosing the starting point a_0 as in Equation 9, we increase the probability that the number of iterations g remains low, since it is most likely that the optimum is in the neighborhood of a_0 .

5. Case Study

In this section, we introduce the chosen SPARC-based architecture and how it can be efficiently optimized by the application of the sensitivity-based local search algorithm. Then, we discuss the experimental results derived from the application of our methodology by exploring the given system architecture. The target system is composed of the following modules:

- A 100 MHz MicroSPARC2 Processor Core (operating at 3.3 V and without on-chip caches).
- Separated and configurable I- and D-caches with one-cycle hit time. The speed of the processor-to-memory bus is 100 MHz.
- A 32Mbyte DRAM composed of 16×16 -Mbit blocks and characterized by a 7-cycle latency. The power model for this memory has been derived from [24]. The speed of the bus cache-DRAM is 100 MHz.

The cache energy model corresponds to $0.8 \mu\text{m}$ CMOS technology, however the equations in the analytical cache energy model can be easily modified to reflect a more updated process technology by simply changing the values of the capacitance parameters to account for the new technological and layout features.

Each instance of the virtual architecture has been described as a *6-tuple* $[c_i, b_i, v_i, c_d, b_d, v_d] \in \mathcal{A} = S_{c_i} \times S_{b_i} \times S_{v_i} \times S_{c_d} \times S_{b_d} \times S_{v_d}$ where:

- S_{c_i}, S_{c_d} are the ordered sets of the possible sizes of the I- and D-caches (e.g., {2KB, 4KB, 8KB, 16KB, 32KB, 64KB}).
- S_{b_i}, S_{b_d} are the ordered sets of the possible block sizes of the I- and D-caches (e.g., {4B, 8B, 16B, 32B}).
- S_{v_i}, S_{v_d} are the ordered sets of the possible associativity values of the I- and D-caches (e.g., {1, 2, 4, 8}).

The architecture has been explored by using a SPARC executable system module, called **MEX**, that consists of the SPARC V8 Instruction Set Simulator with a configurable memory architecture. **MEX** exploits the Shade [8] library to trace the memory accesses generated by a program executed by the SPARC V8 and simulates the target memory architecture to obtain accurate memory access statistics among which:

- Number of accesses to the memory hierarchy (on both I- and D-buses);
- Average cache miss rate (on both I- and D-buses);
- Address sequentiality (on both I and D address buses) [11];
- Bus transition activities (on both I and D address buses);
- Delay (D) (average clock cycles per instruction, measured in terms of $[CPI]$);
- Energy (E) dissipated by the architecture during program execution measured in terms of *Joule per Instruction* $[JPI]$.

The benchmarks used to validate the model are part of the Mediabench suite [20], a set of multimedia benchmarks that can be considered a *de-facto* standard for multimedia applications. The benchmarks have been executed with the standard options provided in their makefiles and have been executed by **MEX** to provide system statistics.

To provide the energy-delay trend for the target system, Figure 5 reports the scatter plot of the energy and the delay for each point in the architectural space (actually composed of 9126 points) for the `gsmdecoder` benchmark. In this picture, we can observe the system with the absolute minimum delay (point A), with the minimum energy (point B) and with the minimum EDP value (point C). The EDP optimal point is, with respect to the most performant configuration A , 1.5% slower but saves as much as 7.9% in energy consumption while, with respect to the less power consuming configuration B , C is 1.9% faster and 1.3% more power consuming. Note that, going from A to C (B to C), the percentage of energy (performance) savings are always greater than performance (energy) losses.

Figure 6 reports the scatter plot of the Energy and the Delay for the `mesa` benchmark. In this case, there is a substantial tradeoff to be exploited since the EDP optimal point is, with

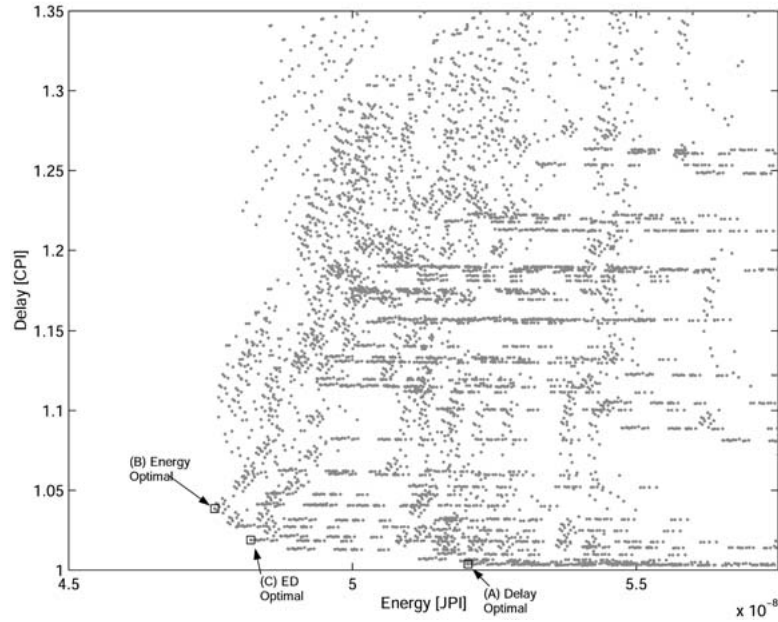


Figure 5. Normalized delay [CPI] vs. energy [JPI] scatter plot for the gsmdecoder benchmark.

respect to the most performant configuration *A*, 4.8% slower but saves 31.8% in energy consumption. Finally, with respect to the less power consuming configuration *B*, the optimal EDP point is 9.4% faster and only 2.6% more power consuming.

Let us analyze more in detail the benchmark mesa in order to better understand the behavior of the system in terms of delay and energy metrics. Figure 7 shows the delay and the energy-delay product in the neighborhood of the optimal configuration of the system for the benchmark mesa ($a_{opt} = [16384, 16, 4, 16384, 4, 4]$) by varying the cache sizes and block sizes.

Note that, by increasing the cache size while keeping a constant block size tends to minimize the delay since more blocks are present in the cache and the miss probability decreases. However, increasing the cache increases also the energy consumption and this implies a trade-off between energy consumption and performance. This is clearly shown in the lower part of Figure 7 where the energy-delay curves are convex.

Figure 8 shows the delay and the energy-delay product in the neighborhood of the optimal configuration of the system for the benchmark mesa by varying the cache sizes and set associativity of both caches. Increasing the cache size with a constant associativity increases the energy consumption. The increased associativity leads to reduced delay without impacting substantially the overall energy-delay product, since 4-way and 8-way set associative caches give an overall better *ED* product with respect to the direct-mapped one.

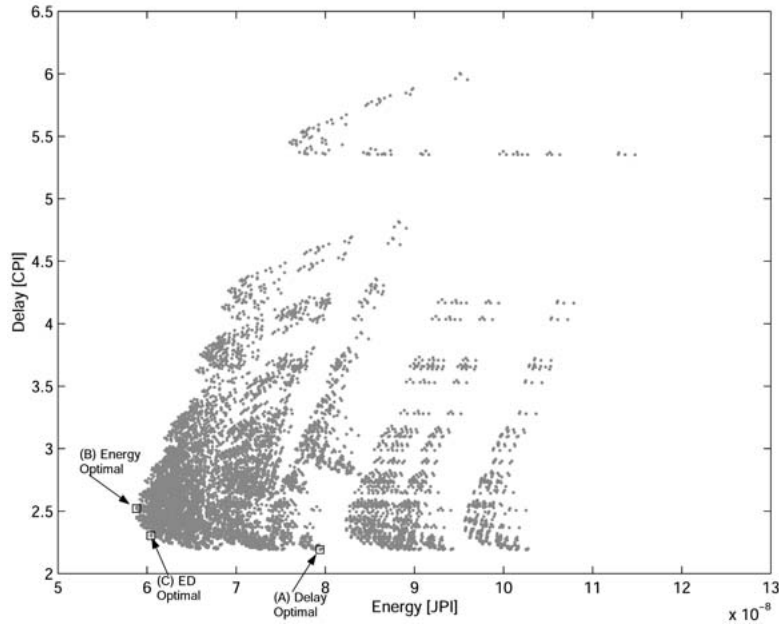


Figure 6. Normalized delay [CPI] vs. energy [JPI] scatter plot for the mesa benchmark.

Figure 9 shows the delay and the energy-delay product in the neighborhood of the optimal configuration of the system for the benchmark mesa by varying the block sizes and associativity of both caches. We can observe how increasing the block size with a constant associativity increases the delay (since data transfer from the main memory becomes longer). However, larger associativity implies less delay, since it reduces cache misses. Furthermore, for the I-cache, increasing the cache block size leads to energy reduction since, with fixed associativity and fixed I-cache size, the number of cache misses is reduced. The power model of the cache is very sensitive to the number of cache blocks, so it is possible to find an optimal block size that minimizes both energy consumption and delay. Concerning the D-cache, the very high rate at which the delay increases with respect to the block size makes negligible the advantages obtained through the reduction of the cache complexity.

5.1. Application of the Tuning Phase

The tuning phase is based on the execution of the MediaBench suite [20], a set of complete multimedia applications mainly written in C. All the applications are public domain and derive from image processing, communication and DSP applications. Among the Mediabench, we selected a sub-set of control-intensive and data-intensive applications to

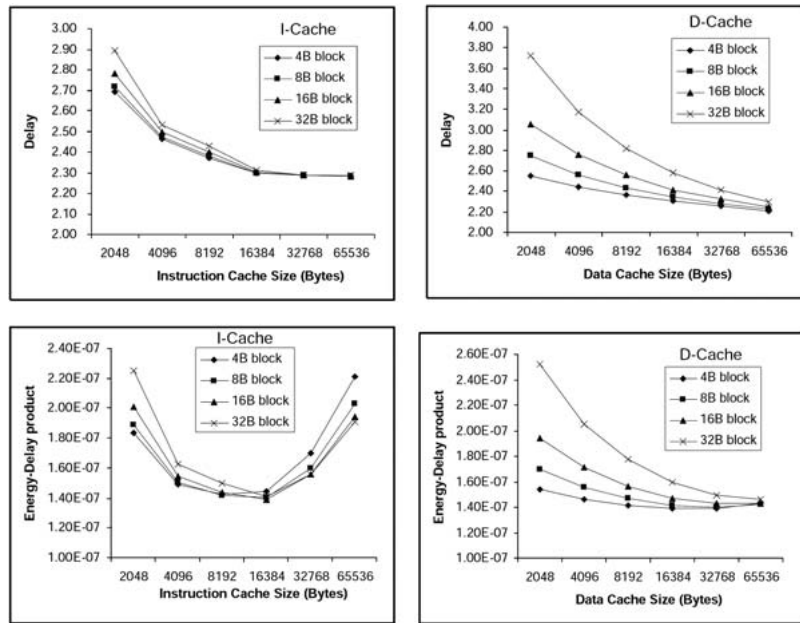


Figure 7. Delay and ED product for mesa benchmark, by varying cache sizes and block sizes.

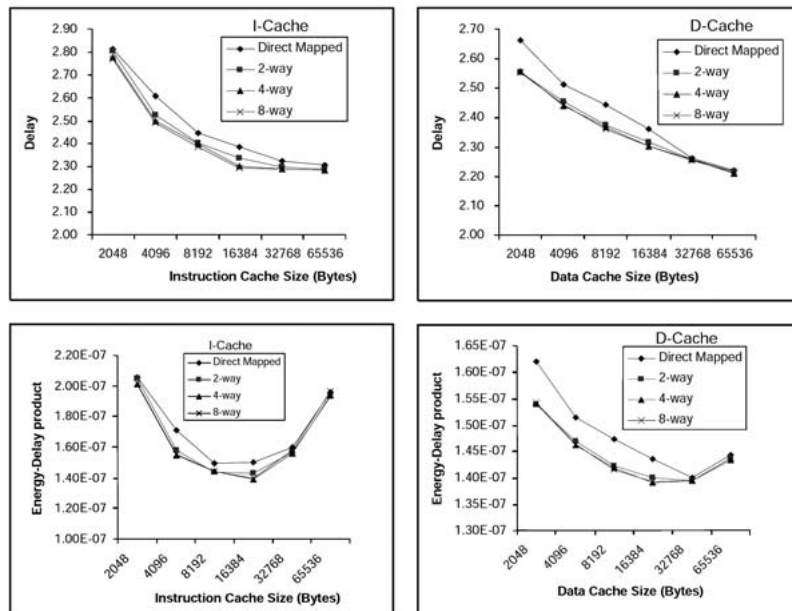


Figure 8. Delay and ED product for mesa benchmark, by varying cache sizes and associativity.

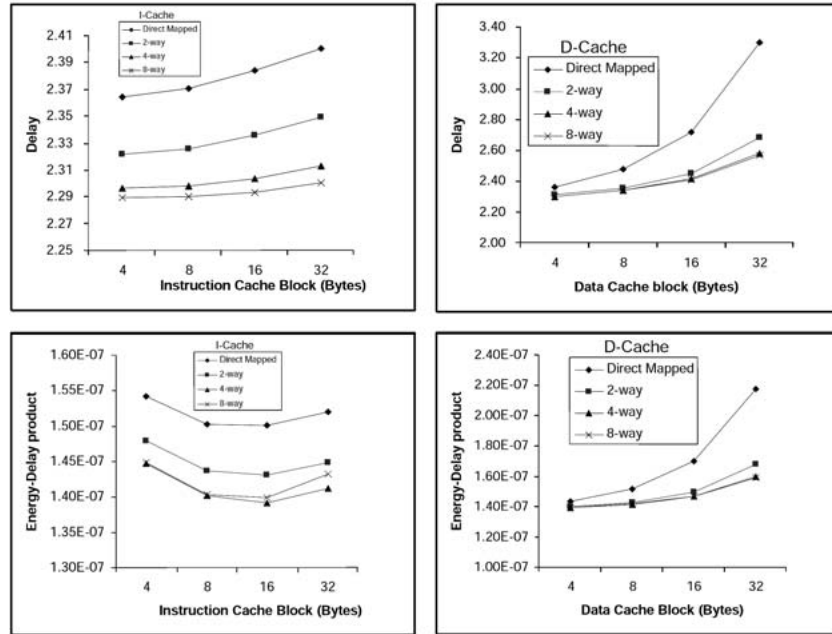


Figure 9. Delay and ED product for mesa benchmark, by varying block sizes and associativity.

be used as reference application set \mathcal{H} to characterize the sensitivity of the system. The reference set \mathcal{H} is composed of the following applications:

G.721 VOICE COMPRESSION ENCODER AND DECODER

These applications consist of the reference implementations of the CCITT (International Telegraph and Telephone Consultative Committee) G.711, G.721, and G.723 voice compressions algorithms.

GHOSTSCRIPT

This application is an interpreter for the PostScript language characterized by a very high I/O activity but no graphical display.

MESA

Mesa is a 3D graphics library very similar to OpenGL. Among the available applications using this library we selected *osdemo*, which executes a standard rendering pipeline.

MPEG DECODER

This is an implementation of an MPEG2 decoder, a standard for high-quality digital video compression, based on the inverse discrete cosine transform.

PGP ENCODER

PGP is digital signature and encryption application based on “message digests,” i.e., 128-bit cryptographically strong one-way hash value of the message. Data encryption is based on the RSA asymmetric encryption scheme.

RASTA

This application performs speech recognition by means of speech analysis techniques such as PLP, RASTA, and Jah-RASTA.

Table 1 shows the optimal parameter values found for the reference benchmarks and their sensitivity indexes. The last row (*MEAN*) indicates the mean sensitivity value associated with the parameter as well as the optimum feasible value of the parameter, i.e., the feasible value of the parameter closest to the mean of the optimal values.

If we consider the mean sensitivity, as Figure 10 shows, the most important parameters are in order, the I-cache size (c_i), the I-cache associativity (v_i), the D-cache size (c_d), the D-cache associativity (v_d) and the I- and D-cache blocks (b_i and b_d). This order will be used by the sensitivity optimizer to efficiently optimize any new application for the given architecture.

5.2. Sensitivity-Based Optimization of the μ SPARC Architecture

To validate the methodology, we applied the sensitivity based optimization to another sub-set of Mediabench. The sub-set is disjoint from the set of reference benchmarks used during the tuning phase. The sensitivity values used for such type of optimization have been presented in the previous section. For validation purposes, the following applications have been used:

ADPCM ENCODER AND DECODER

Adaptive differential pulse code modulation filters for audio coding.

EPIC ENCODER AND DECODER

Image compression/decompression fixed point tools based on a bi-orthogonal wavelet decomposition, combined with a run-length/Huffman entropy coder.

Table 1. The Set of Optimal Parameters and Sensitivity Indexes for the Chosen Set of Reference Applications

Mediabench	c_i	$\sigma(c_i)$	b_i	$\sigma(b_i)$	v_i	$\sigma(v_i)$	c_d	$\sigma(c_d)$	b_d	$\sigma(b_d)$	v_d	$\sigma(v_d)$
g721decode	8192	106.7%	16	0.8%	2	41.3%	8192	1.2%	8	0.3%	2	0.5%
g721encode	8192	118.9%	16	0.8%	2	42.4%	8192	1.2%	8	0.2%	2	0.5%
gsdec	8192	6.2%	8	2.5%	8	0.3%	8192	2.7%	4	1.8%	2	8.8%
mesa	16384	12.1%	16	1.4%	4	2.8%	16384	2.0%	4	1.9%	4	0.6%
mpegdec	8192	8.6%	8	1.8%	4	2.3%	8192	2.4%	4	0.9%	4	0.6%
pgp	8192	45.7%	16	0.6%	2	5.9%	16384	15.4%	4	0.6%	2	1.4%
rasta	32768	20.4%	16	3.1%	8	3.0%	65536	17.7%	16	1.0%	4	4.9%
MEAN	~ 16384	51.1%	~ 16	1.6%	~ 4	15.9%	~ 16384	6.8%	~ 8	0.8%	~ 2	2.8

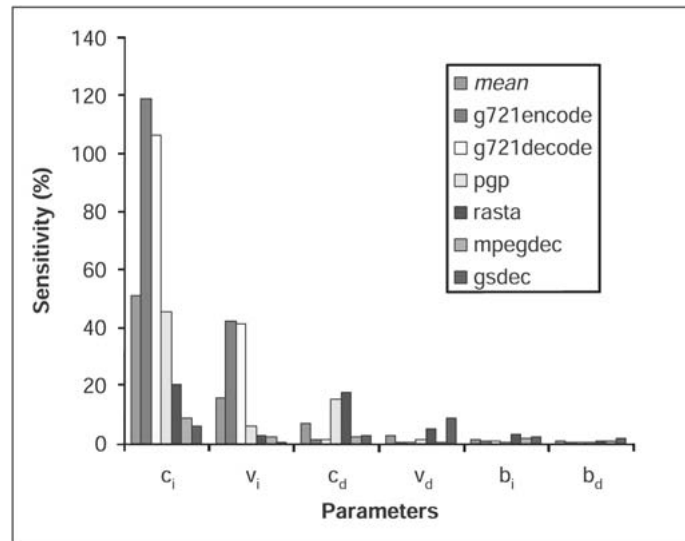


Figure 10. The distribution of the sensitivity for the set of the architectural parameters ordered by their mean sensitivity.

GSM ENCODER AND DECODER

European GSM 06.10 standard speech transcoders based on residual pulse excitation/long term prediction coding at 13 kbit/s.

JPEG ENCODER AND DECODER

Standardized lossy compression method for full-color and gray-scale images.

MPEG ENCODER

This is an implementation of an MPEG2 encoder, a standard for high-quality digital video compression, based on the inverse discrete cosine transform.

PEGWIT

Public key encryption and authentication application based on the elliptic curves theory.

Figure 11 provides the behavior of the sensitivity-based optimizer in the energy-delay space. As shown, the algorithm starts from a point characterized by high energy and delay values and tends to optimize energy and delay simultaneously. As a matter of fact, it reaches first a best delay configuration and, from this point, it begins to consider all the points of the pareto curve until it arrives to the best energy-delay configuration.

Table 2 shows the best configurations estimated by the sensitivity optimizer with respect to those found with a full search over the entire design space. The table shows also the percentage errors on the energy-delay product between the real and the estimated optimum. Note that for all benchmarks but three, the configuration found by the heuristic coincides with the full search one. For the remaining benchmarks, the error on the ED is always less than 2%.

Figure 12 reports the speedup in terms of number of simulations needed by the sensitivity optimizer. As the figure shows, the average ratio between the number of simulations performed by the full search and the number of simulations performed by the sensitivity optimizer is always two orders of magnitude larger. As an example, for a generic application requiring a simulation time of two minutes for each single simulation step, the full search optimization would have required 13 days approximately, while our methodology finds a near optimum solution in less than 4 hours.

6. Conclusions and Future Work

This paper addresses the problem of design space exploration of system architectures where both performance and power consumption are the most relevant constraints. In particular, an exploration methodology to reduce simulation time dramatically while preserving acceptable design accuracy has been proposed and experimentally assessed by considering the design of the memory subsystem of a real-world embedded system. Experimental results have shown a speed-up in simulation time of 2 orders of magnitude and a distance from the optimal configuration always in range of 2%. Furthermore, this methodology allows the designer to save analysis time since the number of configuration to be compared is significantly reduced. Work is in progress to validate this methodology on a broader range of processor architectures.

Beyond the evaluation of the impact of the variation of cache-related parameters from the power-performance joint perspective, the next step of our work aims at estimating the

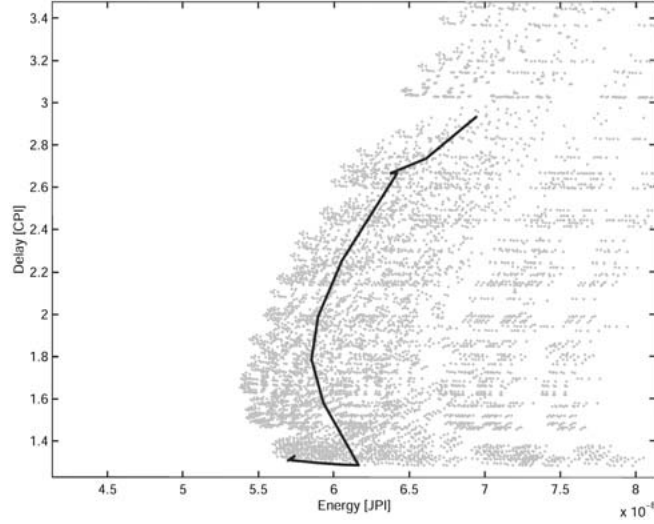


Figure 11. The path followed by the sensitivity-based algorithm when optimizing the jpeg encoder application on the **microSPARC** system.

Table 2. Comparison Between the Full Search Optimal Configurations (a_{opt}) and the Configurations Derived with the Sensitivity Optimizer (\hat{a}_{opt})

Mediabench	a_{opt}	$ED(a_{opt})$	\hat{a}_{opt}	$ED(\hat{a}_{opt})$	error
adpcmdec	[4096,4,4,8192,4,2]	5.47658e-08	[4096,4,4,8192,4,2]	5.47658e-08	0.00%
adpcmenc	[4096,4,4,8192,4,2]	5.41120e-08	[4096,4,4,8192,4,2]	5.41120e-08	0.00%
epic	[2048,8,2,32768,4,4]	1.03661e-07	[2048,8,2,16384,4,2]	1.03737e-07	0.07%
gsmdec	[8192,8,8,16384,32,2]	4.91169e-08	[8192,8,8,32768,32,1]	4.99385e-08	1.67%
gsmenc	[16384,16,2,8192,16,1]	5.45314e-08	[16384,16,2,8192,16,1]	5.45314e-08	0.00%
jpegdec	[4096,4,4,32768,16,4]	6.46782e-08	[4096,4,4,32768,16,4]	6.46782e-08	0.00%
jpegenc	[4096,8,4,65536,8,1]	7.42115e-08	[4096,8,4,65536,8,1]	7.42115e-08	0.00%
mpegenc	[2048,4,4,4096,4,4]	5.19611e-08	[4096,16,2,8192,8,4]	5.24329e-08	0.91%
pegwit	[8192,16,2,65536,4,1]	9.05444e-08	[8192,16,2,65536,4,1]	9.05444e-08	0.00%
unepic	[2048,8,2,65536,8,4]	2.71852e-07	[2048,8,2,32768,4,2]	2.72454e-07	0.22%

effects of power-aware compiler optimizations on the Energy*Delay metric at the system-level.

Other directions of our work are related to the evaluation of the influence of the variations of other system-level parameters such as the number of processors in the system, the processor architecture itself, and the bus topologies in multi-cluster architectures.

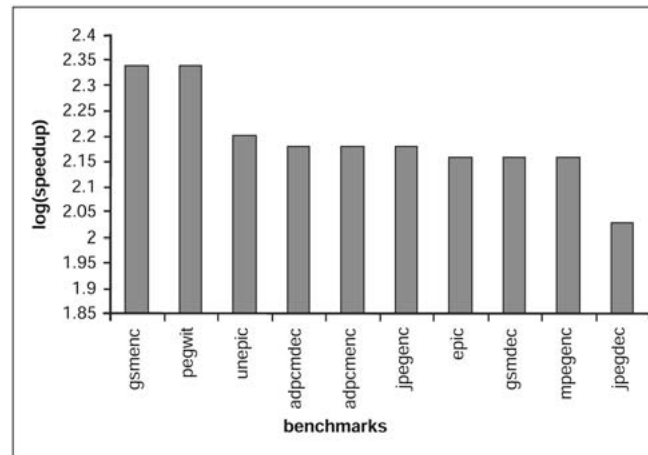


Figure 12. The optimization time speedup obtained by the sensitivity optimizer is always greater than two orders of magnitude.

References

1. Bahar, R. I., G. Albera, and S. Manne. Power and Performance Tradeoffs Using Various Caching Strategies. In *ISLPED-98: ACM/IEEE Int. Symposium on Low Power Electronics and Design*, Monterey, CA.
2. Bellas, N., I. N. Hajj, D. Polychronopoulos, and G. Stamoulis. Architectural and Compiler Techniques for Energy Reduction in High-Performance Microprocessors. *IEEE Transactions on Very Large Scale of Integration (VLSI) Systems*, vol. 8, no. 3, 2000.
3. Benini, L., A. Macii, E. Macii, and M. Poncino. Increasing Energy Efficiency of Embedded Systems by Application-Specific Memory Hierarchy Generation. *Design and Test of Computers*, vol. 17, no. 2, pp. 74–85, 2000.
4. Brockmeyer, E., L. Nachtergaele, F. Catthoor, J. Bormans, and H. D. Man. Low Power Memory Storage and Transfer Organization for the MPEG-4 Full Pel Motion Estimation on a Multi Media Processor. *IEEE Trans. on Multi-Media*, vol. 1, no. 2, pp. 202–216, 1999.
5. Brooks, D., V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *ISCA 2000: 2000 International Symposium on Computer Architecture*. Vancouver BC, Canada, pp. 83–94, 2000.
6. Catthoor, F., S. W. E. D. Greef, F. Franssen, L. Nachtergaele, and H. D. Man. System-Level Transformations for Low Power Data Transfer and Storage. In: R.B.A. Chandrakasan (ed.): *Low Power CMOS Design*. IEEE Press, 1998a, pp. 609–618.
7. Catthoor, F., S. Wuytack, E.- D. Greef, F. Balasa, and P. Slock. *System Exploration for Custom Low Power Data Storage and Transfer*. New York: Marcel Dekker, Inc., 1998b.
8. Cmelik, B. and D. Keppel. Shade: A Fast Instruction-Set Simulator for Execution Profiling. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1994.

9. Conte, T. M., 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, 2000.
10. Danckaert, K., K. Masselos, F. Catthoor, H. D. Man, and C. Goutis. Strategy for Power Efficient Design of Parallel Systems. *IEEE Trans. on VLSI Systems*, vol. 7, no. 2, pp. 258–265, 1999.
11. W. Fornaciari, M. Polentarutti, D. Sciuto, and C. Silvano. Power Optimization of System-Level Address Buses based on Software Profiling. In *CODES-2000: 8th Int. Workshop on Hardware/Software Co-Design*. San Diego, CA, 2000.
12. Fornaciari, W., D. Sciuto, C. Silvano, and V. Zaccaria. A Design Framework to Efficiently Explore Energy-Delay Tradeoffs. In *Proceedings of CODES-2001: Ninth International Symposium on Hardware/Software Codesign*. 2001b, pp. 260–265.
13. Fornaciari, W., D. Sciuto, C. Silvano, and V. Zaccaria. Fast System-Level Exploration of Memory Architectures Driven by Energy-Delay Metrics. In *Proceedings of ISCAS-2001: International Symposium on Circuits and Systems*, vol. IV, 2001a, pp. 502–506.
14. Givargis, T. D., F. Vahid, and J. Henkel. Evaluating Power Consumption of Parameterized Cache and Bus Architectures in System-on-a-Chip Designs. *IEEE Transactions on Very Large Scale of Integration (VLSI) Systems*, vol. 9, no. 4, 2001a.
15. Givargis, T. D., F. Vahid, and J. Henkel. System-Level Exploration for Pareto-Optimal Configurations in Parameterized Systems-on-a-Chip. *ICCAD 2001*, pp. 25–30, 2001b.
16. Gonzales, R., and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, 1996.
17. P. Hicks, M. Walnock, and R. M. Owens. Analysis of Power Consumption in Memory Hierarchies. In *ISLPED-97: ACM/IEEE Int. Symposium on Low Power Electronics and Design*. Monterey, CA, 1997, pp. 239–242.
18. Kamble, M. B. and K. Ghose. Analytical Energy Dissipation Models for Low Power Caches. In *ISLPED-97: ACM/IEEE Int. Symposium on Low Power Electronics and Design*, 1997.
19. Keutzer, K., S. Malik, A. R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System Level Design: Orthogonalization of Concerns and Platform-Based Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, 2000.
20. J. K. Kin, M. Gupta, and W. H. Mangione-Smith. Filtering Memory References to Increase Energy Efficiency. *IEEE Trans. on Computers*, vol. 49, no. 1, 2000.
21. Lee, C., M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating Multimedia and Communication Systems. In *Micro 30*, 1997a.
22. Lee, T., V. Tiwari, and S. Malik. 1997b, “Power analysis and minimization techniques for embedded DSP software,” 1997b.
23. Y. Li, and J. Henkel. A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems. In *DAC-35: ACM/IEEE Design Automation Conference*, 1998.
24. NEC: 16M-bit Synchronous DRAM Data Sheet, Doc. No. M12939EJ3V0DS00, 1998.”
25. Panda, P., and N. Dutt. Low-Power Memory Mapping Through Reducing Address Bus Activity. *IEEE Trans. on VLSI Systems*. vol. 7, no. 3, pp. 309–320, 1999.
26. Russell, J., and M. Jacome. Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors. In *International Conference on Computer Design: VLSI in Computers and Processors*. pp. 328–333, 1998.
27. Sami, M., D. Sciuto, C. Silvano, and V. Zaccaria. Power Exploration for Embedded VLIW Architectures. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD-2000)*. IEEE Computer Society Press. Los Alamitos, California, 2000, pp. 498–503.
28. Shiue, W.-T., and C. Chakrabarti. Power Estimation of System-Level Buses for Microprocessor-Based Architectures: A Case Study. In *DAC99: Design Automation Conference*. New Orleans, LA, 1999.
29. C. L. Su, and A. M. Despain. Cache Design Trade-offs for Power and Performance Optimization: A Case Study. In *ISLPED-95: ACM/IEEE Int. Symposium on Low Power Electronics and Design*, 1995.

30. Udayanarayanan, S., and C. Chakrabarti. Energy-Efficient Code Generation for DSP56000 Family. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED-00)*. ACM Press, N.Y., 2000, pp. 247–249.
31. N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye. Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower. In *ISCA 2000: 2000 International Symposium on Computer Architecture*. Vancouver BC, Canada, 2000.
32. Wilton, S. E., and N. Jouppi. An Enhanced Access and Cycle Time Model for On-Chip Caches. Technical Report 93/5, Digital Equipment Corporation Western Research Lab, 1994.