

## LETTER

# An Efficient Method for System-Level Exploration of Global Optimum in a Parameterized ASIP Design

Yeong-Geol KIM<sup>†a)</sup>, *Student Member and* Tag-Gon KIM<sup>†</sup>, *Nonmember*

**SUMMARY** This paper proposes an efficient method for design space exploration of the global optimum configuration for parameterized ASIPs. The method not only guarantees the optimum configuration, but also provides robust speedup for a wide range of processor architectures such as SoC, ASIC as well as ASIP. The optimization procedure within this method takes a two-steps approach. Firstly, design parameters are partitioned into clusters of inter-dependent parameters using parameter dependency information. Secondly, parameters are optimized for each cluster, the results of which are merged for global optimum. In such optimization, inferior configurations are extensively pruned with a detailed optimality mapping between dependent parameters. Experimental results with mediabench applications show an optimization speedup of 4.1 times faster than the previous work on average, which is significant improvement for practical use.

**key words:** *design space exploration, global optimization, parameterized ASIP*

## 1. Introduction

The embedded systems market is rapidly growing due to the increasing need for product personalization and rapid growth of non-computing domains, such as medical instrumentation and imaging, and information appliances. Since processor customization yields lower area, power and cost, application specific instruction-set processor (ASIP) can be a significant component of embedded systems. The problem is that development of an embedded system consisting of such an ASIP may result in delay for time-to-market because of large overheads in designing instruction set, special hardware modules, developing compiler, debugger, and so on. Therefore, frameworks for supporting rapid development of ASIP are crucial for embedded systems design.

A recent innovation addressing this problem is an embedded soft core, a general-purpose processor which has parameterizable components such as Tensilica [4], ArcCores [5] and HP [6]. Although this parameterized ASIP approach can reduce the design overhead significantly, an exhaustive search of the optimal values of a large number of parameters is still very time-

consuming. For example, full search of 20 parameters, each with three alternatives, leads to about 35 billion configurations. This kind of problem is well known as the combinatorial explosion problem in the global optimization research field [3]. Previous work towards finding globally (sub)optimal configurations for parameterized architectures include [1], [2]. The goal of [1] is to find a sub-optimal configuration for energy-delay product of a memory hierarchy without performing an exhaustive analysis of a parameters space. It applied *separate* optimization for an instruction cache and a data cache which were assumed to be almost independent of each other. Although this paper reported a large optimization speedup with respect to full search, the found solution has about 10% distance from the true optimum. The main restriction of this work is that it was only applied to a simple and modular cache sub-system. Therefore, the quality of the found solution is questionable if the method is applied to more complex design parameters in a practical situation. Furthermore, this approach did not give any information about how close the found solution is to the true global optimum, which is a burden to designer in making decision.

On the contrary, [2] provided correct pareto-optimal configurations by exploiting parameter dependency. This paper showed a significant speedup against the full search when many architectural parameters are independent, as in the case of simple parameterized system-on-chip design. However, if parameters are highly interdependent as in the case of ASIP, this approach simply reduces to almost a full-search algorithm, as will be shown by our experimental results.

These two previous works [1], [2] were selected as a base of our framework on which the proposed algorithm is built to solve both problems simultaneously—guarantee of optimum configuration as well as robust speedup, for a wide range of processor architectures. More specifically, our framework proposes a *Pair-wise Optimality Mapping* (POM) between parameters, with which inferior configurations are extensively pruned, thereby achieving large speedup in finding the global optimum configuration. The rest of this paper is organized as follows. Section 2 presents an overview of the proposed approach, while Sect. 3 describes the details of the algorithm. After showing the experimental results in Sect. 4, the conclusion follows in Sect. 5.

Manuscript received February 28, 2003.

Manuscript revised June 20, 2003.

Final manuscript received August 21, 2003.

<sup>†</sup>The authors are with the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea.

a) E-mail: ygkim@smslab.kaist.ac.kr

## 2. Overview of the Proposed Approach

Before describing the details of the proposed optimization scheme, let us review the key ideas of parameter dependency [2] and separate optimization concept [1].

### 2.1 Parameter Dependency

A parameter  $p_j$  is dependent on the parameter  $p_i$ , if and only if changing the value of  $p_i$  affects the optimal value of  $p_j$ . Otherwise, a parameter  $p_j$  is independent from the parameter  $p_i$ . Figure 1 shows an example of parameter  $p_2$  being dependent upon  $p_1$  (left side), and one that  $p_4$  being independent from  $p_3$  (right side). Every mapping  $(v_i, v_j)$  in this figure represents that the optimal value of  $p_j$ , given the value of  $p_i$  as  $v_i$ , is  $v_j$ . The Parameter dependency graph, abbreviated as *PD-graph*, is a directed graph in which a node represents the parameter while an edge represents the dependency between two parameters. Therefore, this graph is a collection of parameter dependencies for all combinations of parameters.

As is described in [2], we have the same assumption that the designer of the parameterized architecture determines the interdependencies among the parameters. If he or she cannot determine some of the parameter dependencies, then it can be conservatively assumed as dependent *or* determined through exhaustive simulation or sampling. In spite of the large overhead of making PD-graph, such characterization is a one-time effort.

### 2.2 Separate Optimization

Following the definition of the parameter dependency, we can determine the optimization sequence for a given PD-graph.

- An edge from node  $p_i$  to  $p_j$  indicates that the optimal configuration of  $p_j$  should be calculated only after the optimal configuration of  $p_i$  is computed.
- More generally, a path from  $p_i$  to  $p_j$  indicates that the optimal configuration of all the nodes from  $p_i$  to  $p_j$ , residing on the path, is calculated, *in that order*.
- If there is an edge from  $p_i$  to  $p_j$  and edge from  $p_j$  to  $p_i$ , the optimal configuration of parameters  $p_i$  and  $p_j$  must be calculated *simultaneously*.
- More generally, a path from  $p_i$  to  $p_j$  and back to  $p_i$ , which forms a cycle, indicates that the optimal configuration of all the parameters on the cycle needs to be calculated simultaneously.

The first two cases show the example of the separate optimization by which the number of calculations is reduced, while the last two show the full-search case.

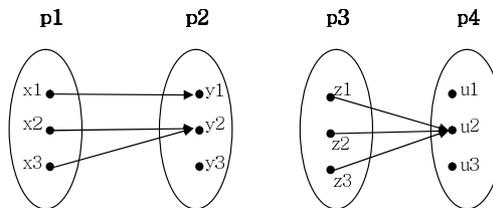


Fig. 1 Parameter dependency.

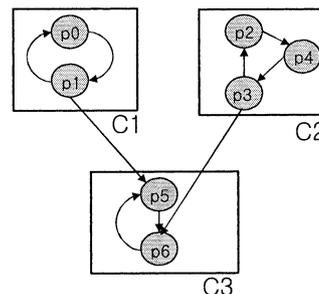


Fig. 2 Clustering PD-graph for separate optimization.

### 2.3 Proposed Idea for More Speedup

To apply the separate optimization scheme for a given PD-graph, a *cluster* is defined as maximal strongly connected sub-components of a PD-graph. After all the clusters of the PD-graph are found, these are topologically ordered for correct optimization sequence as shown in Fig. 2.

Equation (1) shows the speedup obtained from applying the separate optimization scheme.

$$Speedup_{PDO} = \frac{\prod_{i=0}^{\#clusters} F_i}{\sum_{i=0}^{\#clusters} F_i} \quad (1)$$

In this equation, the term  $F_i$  denotes the number of simulations required for *full-search* of all configurations within cluster  $c_i$ . This is the speedup provided by the previous work [2], called as *PD-Parameter Dependency based Optimization*, against simple full-search over all parameters of the PD-graph without clustering it. The problem in terms of the achievable speedup with this *PDO* approach is that the speedup rapidly decreases as the number of clusters decreases, i.e., each cluster has lots of parameters, since the term  $F_i$  rapidly increases. Therefore, when the target architecture has highly inter-dependent parameters, we cannot expect large speedup with this approach.

To enhance the speedup, the proposed scheme seeks to reduce the  $F_i$  term in the divider by finding out inferior configurations and pruning those within cluster. More importantly, this is done without loss of guarantees of finding global optimum configuration.

### 3. POM-Based Optimization Algorithm

As previously described, the proposed approach also uses the parameter dependency information and cluster-level separate optimization scheme. In addition to these, we introduce a *lower-level* mapping information, called as POM, which tells us *in which way* two parameters are dependent. Note that the parameter dependency is *higher-level* information compared to the POM since it only tells us whether two parameters are dependent or not.

The POM from parameter  $p_i$  to  $p_j$ , notated as  $pom_{i,j}$ , is a collection of  $pom_{i,j}(x)$  that maps the optimal value of  $p_j$  when the value of  $p_i$  is  $x$ . To see Fig. 1 again, we can see  $pom_{1,2}(x_1) = y_1$ ,  $pom_{1,2}(x_2) = y_1$ ,  $pom_{1,2}(x_3) = y_2$ .

The proposed algorithm is made of a repeating sequence of two major steps, one of which is POM construction process and the other is configuration pruning process. These two processes are described in the next two subsections.

#### 3.1 Pruning by POM

Once obtained, the POM information can be exploited to efficiently remove the configurations from the search space that are guaranteed to be inferior to other configuration(s). This removing process is called *pruning*.

Figure 3 shows the partial ordering of configurations caused by the  $pom_{1,2}(x_1) = y_1$  when the PD-graph corresponding to Fig. 1 is given. As shown in the PD-graph, since parameter  $p_2$  is independent from  $p_3$  and  $p_4$ , values of  $p_3$  and  $p_4$  cannot affect the optimal value of  $p_2$ , by the definition of parameter dependency. Therefore, we can tell that the configurations  $(x_1, y_2, z_*, u_*)$  and  $(x_1, y_3, z_*, u_*)$  are inferior to the configurations  $(x_1, y_1, z_*, u_*)$ . Meanwhile, since we cannot tell which is superior/inferior to the other, they remain incomparable. After all, all the configurations  $\{(x_1, y_2, z_*, u_*)\} \cup \{(x_1, y_3, z_*, u_*)\}$  can be pruned. More detailed information for the POM elements and its associated pruned configurations are shown in Table 1.

It is evident that the global optimum value can never be pruned since any pruned configuration is inferior to at least one other configuration. By the definition of global optimum configuration, it is not infe-

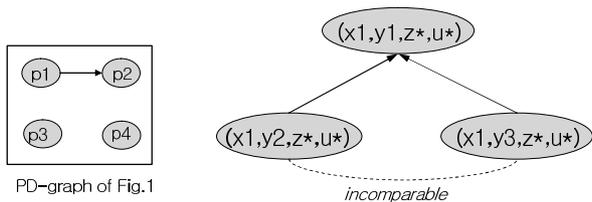


Fig. 3 Partially ordered configurations by POM.

rior to any other configuration. In this sense, we can say that the proposed pruning is *safe* in obtaining the global optimum.

#### 3.2 POM Construction Algorithm

Since a cluster can contain lots of edges representing dependencies between parameters, there arise the three following questions about constructing POM corresponding to each edge.

- Should we construct all of the POMs belonging to the given cluster?
- If not, which of the POMs should be constructed and which of them not?
- And, in what sequence should the POMs be constructed for maximum speedup?

The answers to these questions can be clearly described by defining a *POM-cluster*. A POM-cluster for a parameter  $p$  is defined to be a sub-cluster consisting of all the parameters affecting  $p$  and the parameter  $p$  itself (Refer to Fig. 5). Since one parameter has its own POM-cluster, a cluster containing  $n$  parameters produces  $n$  different POM-clusters for each parameter, as shown in Fig. 4. Each such POM-cluster is obtained by projecting the cluster onto the parameter.

Note that these POM-clusters are sorted by increasing order of expected cost, which is defined to be the number of *new* simulations to construct the POMs belonging to the POM-cluster. The reason for

Table 1 POM elements and pruned configurations.

POM elements	Pruned configurations
$x_1 \rightarrow y_1$	$\{(x_1, y_2, z_*, u_*)\} \cup \{(x_1, y_3, z_*, u_*)\}$
$x_2 \rightarrow y_2$	$\{(x_2, y_1, z_*, u_*)\} \cup \{(x_2, y_3, z_*, u_*)\}$
$x_3 \rightarrow y_2$	$\{(x_3, y_1, z_*, u_*)\} \cup \{(x_3, y_3, z_*, u_*)\}$

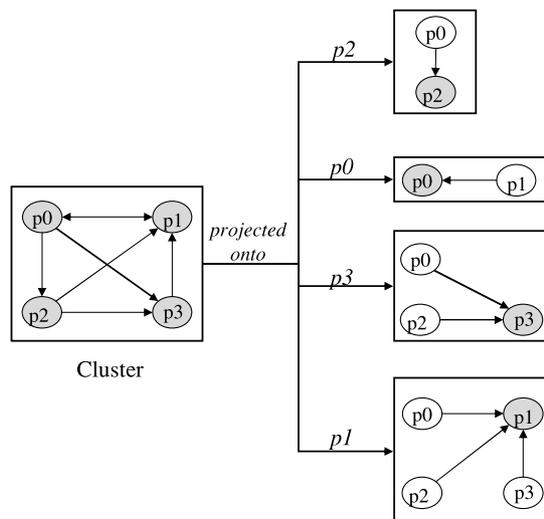


Fig. 4 POM-cluster: Basic unit for POM construction and pruning.

this sorting is that some POM-clusters can have excessive construction cost causing negative effect on the overall speedup. To avoid this kind of undesirable situation, the POM construction algorithm checks stopping condition before applying construction process to every POM-cluster.

Meanwhile, in constructing the POMs, the parameters outside of POM-cluster can be fixed to some arbitrary value since these parameters cannot change the optimal value of the destined parameter of the POM-cluster. For example, in constructing the  $pom_{0,2}$  of the first POM-cluster of Fig. 4,  $p_1$  and  $p_3$  can be fixed to some arbitrary value because  $p_2$  is independent from these parameters.

### 3.2.1 Stopping Condition for POM Construction

Intuitively speaking, the POM construction algorithm stops immediately when the accompanying cost of the POM-cluster is expected to be larger than the benefit of pruning the configurations. There are two following greedy conditions, meeting one of which stops the constructing process.

- expected cost  $\geq$  number of currently unpruned configurations
- expected cost  $\geq$  expected number of configurations to be *newly* pruned

The reason to stop for the first condition is that it requires a smaller number of simulations to simply full-search the remained configurations than construct-and-prune POM-cluster. Similarly, the second condition says that there is no need to construct the POMs if the expected number of configurations to be newly pruned is less than the expected POM construction cost. Note that the term ‘*newly pruned*’ means that a configuration pruned by the POM can be already pruned by previous POM. Now we describe how to obtain the expected cost and the number of configurations to be newly pruned.

### 3.2.2 Calculation of Expected Values

By the definition, any POM-cluster of PD-graph has a canonical structure as shown in Fig. 5. That is, a POM-cluster of size  $n + 1$  has a single destination parameter  $p_d$  which is dependent on  $n$  distinct source parameters

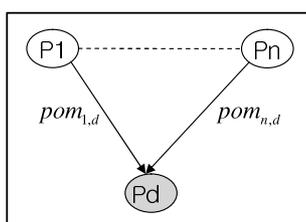


Fig. 5 Canonical structure of a POM-cluster.

$p_1, \dots, p_n$ . Clearly, all of the POM-clusters of Fig. 4 have the canonical structure. This canonical structure implies that we have to full-search over POM-cluster to obtain the  $n$  POMs. Actually, the number of simulations required is smaller than full-search, because any configuration visited before can be reused without simulation.

Fortunately, we can calculate the expected cost of the POM-cluster *exactly* since it is equal to the number of simulations required to full-search minus the number of simulation reused, i.e., already evaluated configurations. It is easy to see that the number of simulations required to full-search within the POM-cluster is calculated analytically, by multiplying the number of alternatives over all parameters. Moreover, the number of simulation reuse can also be exactly calculated by comparing each of the possible configuration with already-visited configurations.

Expectation of the number of pruned configurations is rather complex and it is not possible to exactly estimate it because of the unpredictable nature of the *POM* mapping which varies with application and data. However, a simple probabilistic approach provided a good estimate, which will be shown in Sect. 4. Note that, although accurate calculation of it can be important for obtaining maximum speedup achievable, it is not so critical as the POM-cluster cost expectation, since the expected value is used only for the second stopping condition. That is, the first stopping condition prevents the proposed algorithm getting in the worst case of doing harmful POM-construction for the POM-cluster with very large cost.

The implemented scheme is that for every  $pom_{i,j}$  of the POM-cluster, we assign equal probability to the mapping  $pom_{i,j}(v_i) = v_j$  for all possible values of  $v_j \in P_j$ . Then, by averaging the number of pruned configurations for all cases of mapping, the expected value is obtained.

### 3.3 Cooperating with Other Optimization Scheme

The POM construction/pruning method is so general that it can be suited for various optimization problems of parameterized design. One of the advantages of the proposed approach is that it can be used with another optimization techniques to make a further speedup in the whole ASIP optimization process. In other words, proposed method can be used to complement other optimization scheme.

Since POM, once obtained, can be used to find the inferior configurations, the proposed POM-based optimization scheme can be used as a *design space filter*. Especially, a set of POMs with small construction overheads would be beneficial for the search space reduction. For example, [7] developed a processor selector which first reduces the domain of search by restricting various architectural parameter ranges based on certain

gross characteristics of the processors, such as branch-delay, presence/absence of multiply-accumulate operation, depth of pipeline and memory bandwidths. The output is a smaller subset of processors. In this case, it would be possible to apply the POM-based optimization scheme by taking this output as an input for further reduction of design space.

#### 4. Experimental Results

We used the SimpleScalar simulator framework [8] to check the correctness and evaluate the speedup of the proposed algorithm. The SimpleScalar simulator supports parameterized superscalar architecture. Therefore, we used it as a research vehicle to verify the proposed algorithm in exploring superscalar-type parameterized ASIP. This experiment used the energy-delay product [10] as a performance metric because it reveals the trade-off between the most important factors of embedded applications characterized by low-power and high-performance requirements. Since the SimpleScalar framework does not yet provide the power estimation capability, we use the Wattch framework [9] for power/energy estimation. We selected Wattch among the well-known power-estimation environments because it is well integrated into the SimpleScalar framework.

Since considering whole parametric design space of the SimpleScalar simulator is not possible for the experiment with the available computation facility, we selected ten important parameters affecting the performance characteristics of energy and delay.

These parameters with their admissible range of values are shown in Table 2. Figure 6 shows the corresponding PD-graph, which is given as an input to the proposed algorithm. The dotted box in this figure shows the largest cluster consisting of eight parameters, while other two clusters have only one parameter. The speedup of PDO against full-search is exactly obtained with the formula described in Eq. (1).

$$Speedup_{PDO} = 2 * 2 * 2916 / (2 + 2 + 2916) = 3.99$$

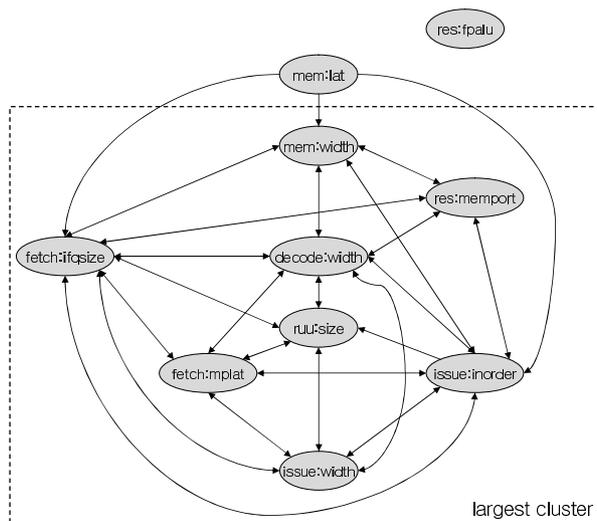
Table 3 shows the full search results for three benchmarks of a Mediabench suite [13] with the selected parameters of Table 2. The second row of this

**Table 2** Selected SimpleScalar parameters.

Parameter	Range
issue:inorder	false or true
fetch:ifqsize	2, 4, 8
decode:width	2, 4, 8
issue:width	2, 4, 8
fetch:mplat	1, 2, 3
ruu:size	4, 8, 16
res:fpalu	1, 2
mem:width	8, 16
mem:lat	(9,1) or (9,2)
res:mempport	1, 2, 4
Num. of alternatives	11664

table shows the minimum, average and maximum values of ED product. We can observe that the maximum values are about three to five times greater than the minimum values. The rest of the rows show the optimal values for each parameter which results in the minimum value.

Table 4 shows the resultant statistics after the proposed optimization algorithm was applied. Of course, we have found the global optimum configuration for all benchmarks. The first four rows show the statistics obtained for the largest cluster in Fig. 6, while the last two show the overall speedup against full-search and previous work, respectively. The first row shows the number of simulations conducted for constructing POMs belonging to the applied POM-clusters whose



**Fig. 6** Input PD-graph of ten parameters.

**Table 3** Full search results.

Benchmark	ADPCM	EPIC	PEGWIT
ED product (min/max)	9.0/48.6	5.5/26.4	21.0/72.0
issue:inorder	TRUE	TRUE	FALSE
fetch:ifqsize	4	4	2
decode:width	4	2	2
issue:width	4	4	4
fetch:mplat	2	2	2
ruu:size	16	8	8
res:fpalu	2	2	2
mem:width	16	16	16
mem:lat	(9,1)	(9,1)	(9,1)
res:mempport	4	4	4

**Table 4** Summary of optimization result.

	ADPCM	EPIC	PEGWIT
POM construction cost	108	576	576
# pruned configuratoins	2241	2833	2802
Pruning ratio(%)	76.9	97.2	96.1
# applied POM-clusters	2	5	5
Speedup against full-search	14.82	17.59	16.81
Speedup against PDO [2]	3.71	4.4	4.21

number is shown in the fourth row. With the cost of these simulations, we have effectively pruned lots of inferior configurations as shown in the second row. The third row shows the pruning ratio which is equal to the number of pruned configurations divided by the number of total configurations, averaging 90.1%.

Although the presented example considered parameters related to the micro-architectural features only, the proposed framework can be applied to more general case of allowing *instruction set* variation as well. As previously mentioned, the target ASIP is assumed to be specialized from the extendible core. Core-based design allows to derive ASIP from a common base architecture with low non-recurring engineering cost. In most such designs, all the central characteristics of core hardware are parameterized, whereas the base instruction set is fixed. There are two ways of instruction set extension: one is a selection from pre-defined extensions set while the other is a user-defined extension. Although there would be no limit to the number of such *application-specific* instructions in theory, common practices show that only about 10 to 20 application-specific instructions are made available for selection, from which less than 10 instructions are typically chosen. For example, there are 14 extension instructions in the ARCTangent-A5 core [5], 12 optional instructions in the MetaCore [11], and five user-defined instructions in the MDSP-II [12]. This implies that the design space caused by the instruction set variation is solvable in the reasonable time.

Meanwhile, the quality of instruction set for ASIP largely depends on that of special instructions for target applications. Since the special instructions are implemented as *special functional units*, the instruction set exploration problem can be abstracted to that of selecting best special functional units from possible alternatives. Through this abstraction, we can concentrate on optimizing *micro-architecture level* parameters encompassing various core components and special functional units in a unified way. In this situation, the dependencies between special FUs and core components may vary depending on the nature of target application and how much the special FUs share a common functionality and resources. One of our on-going research is to modify the simplescalar environment to incorporate special functional units to measure resultant speedup.

## 5. Conclusions

This paper proposed an efficient global optimization scheme that guarantees the global optimum configuration for a parameterized ASIP, as well as providing robust speedup characteristic for a wide range of processor architectures. The proposed framework introduced the POM in addition to the parameter dependency model. The combination of the application-

independent global information of the PD-graph and the application-specific optimality mapping of the POM provided the double speedup effects, that is, a speedup from separate optimization scheme at inter-cluster level, and speedup from POM-based pruning scheme at intra-cluster level. Experimental results with selected media-bench applications with the SimpleScalar simulator showed 4.1 times speedup on average against the previous work. Further works include experiments on an architecture where parameters are less inter-dependent than the presented example, such as VLIW-ASIP or SoC, with which more speedup is expected. Another work to do is to convert the problem from finding the optimum configuration to quickly finding configurations that satisfy the pre-specified design constraints for cycle counts, area, power, and so on.

## References

- [1] W. Fornaciari, D. Sciuto, C. Silvano, and V. Zaccaria, "A design framework to efficiently explore energy-delay trade-offs," Proc. CODES'01, pp.260-265, Copenhagen, Denmark, April 2001.
- [2] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.10, no.4, pp.416-422, Aug. 2002.
- [3] M.C. Fu, "Simulation optimization," Proc. WSC'01, pp.53-61, Arlington, VA, Dec. 2001.
- [4] J. Turley, "Tensilica CPU bends to designers' will," Microprocessor Report, vol.13, no.3, p.12, March 1999.
- [5] "Technical Summary of the ARC Core," ARC Cores Ltd, 2000. At <http://www.arccores.com>
- [6] P. Faraboschi, G. Brown, J.A. Fisher, G. Desoli, and F. Homewood, "Lx: A technology platform for customizable VLIW embedded processing," Proc. ISCA'00, pp.203-213, Vancouver, British Columbia, Canada, June 2000.
- [7] T.V.K. Gupta and P. Sharma, "Processor evaluation in an embedded systems design environment," Proc. International Conference on VLSI Design, pp.98-103, Calcutta, India, Jan. 2000.
- [8] D. Burger and T.M. Austin, "The SimpleScalar tool set, Version 2.0," Comput. Archit. News, pp.13-25, June 1997.
- [9] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," Proc. ISCA'00, pp.83-94, Vancouver, BC, June 2000.
- [10] 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. Very Large Scale Integr. (VLSI) Syst., vol.8, no.2, pp.129-137, April 2000.
- [11] J.-H. Yang, B.-W. Kim, and S.-J. Nam, "MetaCore: An application-specific programmable DSP development system," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.34, no.3, pp.173-183, 1999.
- [12] B.W. Kim, J.H. Yang, and C.M. Kyung, "MDSP-II: A 16-bit DSP with mobile communication accelerator," IEEE J. Solid-State Circuits, vol.34, no.3, pp.397-404, 1999.
- [13] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," Proc. MICRO'97, Research Triangle Park, NC, pp.330-335, Dec. 1997.