Des Autom Embed Syst
DOI 10.1007/s10617-013-9117-2

A new merit function for custom instruction selection under an area budget constraint

Mehdi Kamal · Amir Yazdanbakhsh · Hamid Noori · Ali Afzali-Kusha · Massoud Pedram

Received: 25 September 2012 / Accepted: 17 July 2013

Abstract This paper presents a new merit function for custom instruction selection phase of the design flow of application-specific instruction-set processors (ASIPs) in the presence of an area budget constraint. In contrast to nearly all of the previously proposed approaches where ratio of the ASIP speed to layout area is used as a merit function to select the candidate custom instructions (CIs), we show that a merit function based on normalized cycle saving and area function can result in better CI selections in terms of the achievable speedup under a given area budget for both greedy and branch-and-bound techniques. The efficacy of the proposed approach is assessed by comparing the results of using the proposed and conventional merit functions for different benchmarks. The comparison points toward an average (maximum) speed enhancement of 3.65 % (27.4 %) for the proposed merit function compared to the conventional merit functions.

Keywords Application-specific instruction-set processors · Custom instruction · Custom instruction selection · Merit function

M. Kamal · A. Yazdanbakhsh · H. Noori · A. Afzali-Kusha (✉)
School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran
e-mail: afzali@ut.ac.ir

M. Kamal
e-mail: mehdikamal@ut.ac.ir

Present address:
A. Yazdanbakhsh
Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI, USA
e-mail: yazdanbakhsh@wisc.edu

Present address:
H. Noori
Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
e-mail: hnoori@um.ac.ir

M. Pedram
Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
e-mail: pedram@usc.edu

Published online: 17 September 2013

1 Introduction

Performance and flexibility are two major factors that should be considered in designing embedded systems. Although application specific ICs (ASICs) offer high performance, they are expensive and lack programmability. On the other hand, general purpose embedded processors are fully flexible and cheaper compared to ASICs. They, however, fail to meet the required performance and power consumption of embedded applications. Extending the instruction set of a base processor with custom instructions (CIs) is an approach that provides both high performance and flexibility. Higher performance is achieved by extending the instruction set architecture (ISA) of the base processor with frequently-executed patterns of instructions i.e., the CIs [1–3].

Another important advantage of customized processors is programmability. This feature brings the possibility to meet modest changes that may be required by a target application after the embedded system has been designed. Examples of such changes include bug fixes, functional additions due to adoption of an industry standard (such as the case of the video processing) [3]. Each CI needs a custom functional unit (CFU) to be executed, which results in area overhead. Hence, the ISA extension methods struggle to efficiently implement the best set of CIs considering the available area budget so as to achieve the maximum application speedup.

The main challenge in the design of customized processors is to provide a complete chain of tools. These tools should be able to automatically extend the instruction set of the base processor, and preferably generate the software development tools such as compiler, assembler, and linker. There are two main steps in the design flow of customized processors: identification and selection of CIs. These steps are costly and time-consuming. Many commercial and academic approaches have been introduced to automate these phases (see, e.g., [1–20]). The tools identify and select the set of CIs under various micro-architectural constraints such as power and area budget.

In the design flow, first the data flow graphs (DFGs) of basic blocks of the application are constructed. Then, in the identification phase, all convex sub-graphs of the DFGs, which meet the defined constraints (e.g., I/O constraints), are enumerated. These sub-graphs correspond to CIs. Since there are some limitations, e.g., on the unused op-codes and layout area, in the selection phase, an optimal subset of the CIs is chosen from a pool of identified CIs in the identification phase based on their merit values. The selected CIs need custom functional units (CFUs) to be executed, which are put in parallel with the ALU of the base processor. It has been proved that the selection phase is an NP-complete problem [1]. The merit values of the ISA extension could be determined based on the objectives of improving the application speed and layout area. Considering both of these objectives is typically achieved by using the area budget as efficiently as possible while reaching the shortest execution time. To solve the problem under a given area constraint, various exact, greedy and heuristic methods have been proposed [1, 4–7].

The proposed approaches which consider both cycle saving and area in the selection phase, use cycle saving per area overhead (denoted by CSPA in this paper) as an objective function to optimize the area utilization for specified area budget (see, e.g., [1, 5]). On the other hand, those who only concentrate on the cycle saving, use the merit function only based on the cycle saving. This approach (denoted by CyS in this paper) has been widely used when the area constraint is not defined [6, 20]. In this paper, we propose a different merit (objective) function for the selection phase of CIs under area constraint which utilizes the area more efficiently to achieve higher application speedup (performance). Please note that the focus of this paper is to propose a merit function which may be used in non-optimal

(e.g., greedy) CI selection algorithms for choosing the better CIs from an identified CI set. The performance of the proposed merit function is compared with those of some existing merit functions (i.e., CSPA and CyS) for the same set of identified CIs.

The structure of the paper is organized as follows. Section 2 describes the ISA extension design flow and problem statement for the proposed method. In Sect. 3, related works in the area of CI selection are briefly reviewed. The proposed merit function is stated in Sect. 4 while the results are discussed in Sect. 5. Finally, Sect. 6 concludes the paper.

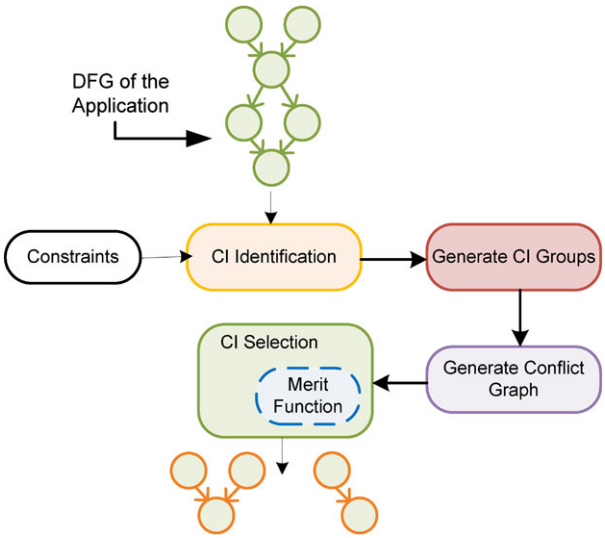
2 ISA extension design flow and problem statement

We first explain the overall flow of the ISA extension [1, 6, 10, 18], which is shown in Fig. 1. The flow starts by the CI identification phase. In this phase, all sub-graphs of the input application DFG that meet the specified constraints (e.g., propagation delay, Convexity) are identified.

Note that a subgraph is architecturally feasible if it is convex. Given a graph G, a subgraph S is convex if, for every two nodes u and v in S, there does not exist any path from u to v ($u \rightarrow v$) involving a node w in G which is not in S. Figure 2 shows an example of a DFG with two subgraphs. The subgraph ABD is not convex while the subgraph CD is convex. The reason for the non-convexity of the subgraph ABD is that there is the path $A \rightarrow C \rightarrow D$ which includes the node C not belonging to this subgraph.

Among the identified sub-graphs (CIs), there may be similar sub-graphs based on the functional and structural isomorphism. These CIs can be executed on one CFU, which results in less area overhead and better area utilization. Such similar sub-graphs are identified and placed into CI *groups* [6, 7]. If a node of the DFG is included in two CIs, these two CIs are considered to overlap with one another. Because any node in the DFG of the input application must be uniquely executed by one CI, all CIs that have overlaps with a selected CI are subsequently removed from the list of the identified CIs in the selection phase. Based on the overlap between the CIs, a conflict graph is constructed. The nodes in the conflict graph are CIs and an edge between two nodes shows that these CIs have overlap. Removing

Fig. 1 The ISA extension flow



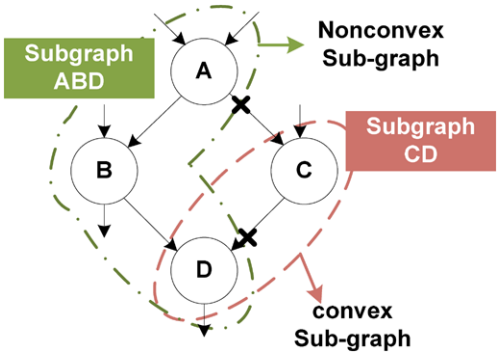


Fig. 2 Examples of a DFG and two subgraphs

```
1:Find the Parameters of CIs such as CS and Area
2:While (the Candidate CI List is not empty)
3:    Calculate the merit of the CI groups
4:    Select the best CI group and remove it from the list
5:    Remove the conflicting CIs from the remaining CI groups based on the conflict graph
6:END
```

Fig. 3 The pseudo code of the greedy algorithm for the CI selection phase

conflicting CIs causes changes in the members of CI groups which, consequently, results in changes in the overall speedup of each CI group.

There are two general approaches for the CI selection which are branch-and-bound [6] and greedy approaches (see, e.g., [4–7]). The former is an exact method which attempts to find the optimum CI set by searching the whole search space. The method becomes intractable for many real applications. One remedy to this problem is to prune the search space[4, 7]. The latter, which is much faster than the branch-and-bound technique, may not be able to find the optimal set. In this work, we consider both methods but with more emphasis on greedy approaches. Note that the computational complexity of the instruction set extension flow is determined by both the identification and selection phases.

The pseudo code for the greedy selection algorithm is given in Fig. 3. The main loop, which iterates by checking the number of CIs in a *Candidate* CI *List* (list of identified CIs), is terminated when the list is empty. In each iteration, the merit value of each CI group, which shows the eligibility of the CI to be selected, is calculated.

To find the ability of each CI in reducing the number of the cycles of the application, the *Cycle Saving* (CS) parameter is defined for each CI. Normally, *Cycle Saving* is the main objective in the ISA extension. The CS may be computed as

$$\mathrm{CS}_i = freq \times \left(\#\mathrm{CI}_i \cdot SW - \mathrm{CI}_i \cdot IO\_Penalty - ceil\left(\frac{\mathrm{CI}_i \cdot Critical\ Path\ Delay}{Clock\ Period} \right) \right) \quad (1)$$

where CS_i is the *Cycle Saving* of the ith CI (CI_i), *freq* is the execution frequency of the basic block to which CI_i belongs, and $\#\mathrm{CI}_i \cdot SW$ is the number of clock cycles of the base processor that the CI needs in order to be executed. $\mathrm{CI}_i \cdot IO\_Penalty$ denotes the number of extra accesses for reading/writing data to/from the register file (when the number of I/O ports of the CI is larger than number of read/write ports of the register file). Note that if the number of the I/O ports of the CI is smaller or equal than the I/O ports of the register

file, no extra cycles is needed and $\mathrm{CI}_i \cdot \mathit{IO\_Penalty}$ is equal to 0. Otherwise, $\mathrm{CI}_i \cdot \mathit{IO\_Penalty}$ will be larger than 0. The last fractional term calculates the number of clock cycles needed to execute the CI_i on the CFU (note that CIs can be multi-cycle as well as single cycle operations). In the fractional term, $\mathrm{CI}_i \cdot \mathit{CriticalPathDelay}$ denotes the propagation delay of the critical path of the CI where as *Clock Period* is the desired clock period for the extended processor.

Whether the design flow includes the selection and identification phases together or separate, when the area constraint is defined as a constraint in the ISA extension, we can formulate the instruction selection problem as

$$\text{Maximize} \sum_{i=1}^{|\mathit{Selected}\ \mathrm{CI}\ \mathit{Groups}|} \mathrm{CS}_{\mathrm{CI}\ \mathit{Group}_i} \tag{2}$$

while

$$\sum_{i=1}^{|\mathit{Selected}\ \mathrm{CI}\ \mathit{Groups}|} \mathit{Area}_{\mathrm{CI}\ \mathit{Group}_i} < \mathit{Area}_{\mathit{Constraint}} \tag{3}$$

Equations (2) and (3) are the objective function and its constraint used in the CI selection process which is an optimization problem. To select CIs, one should use either these equations in an optimal (exact) optimization algorithm or a merit function in a non-optimal optimization algorithm (e.g., greedy). In the non-optimal selection, when we wish to implement the CFU while considering the area overhead of selected CIs, the merit function, for example, may be specified as *Cycle Saving/Area* (see, e.g., [1, 5]). In this case, the merit function of the jth CI group is modified as

$$\mathit{Merit}_j = \frac{\sum_{i=1}^{|\mathrm{CI}\ \mathit{group}_j|} \mathrm{CS}_i}{\mathit{Area}_j} \tag{4}$$

where *Area* denotes the area of the jth CI group. It is obvious that in the cases where the area budget constraint is not considered, the merit function may be defined as only the cycle saving. Hence, in this case, the merit function of the jth CI group is formulated as

$$\mathit{Merit}_j = \sum_{i=1}^{|\mathrm{CI}\ \mathit{group}_j|} \mathrm{CS}_i \tag{5}$$

The best CI group is selected based on one of these merit functions (Fig. 3, line 4). Finally, the CIs that are adjacent to the selected CIs in the conflict graph are removed.

The main contribution of this work is to propose a new merit function for the selection phase of CIs which is an optimization problem. This merit function, which will be explained later, mostly provides better solutions in terms of cycle saving within a specified area budget in comparison to the existing merit functions.

3 Related works

As mentioned above, a key part of the ISA extension is the selection phase where the best CIs are selected from the pool of the candidate CIs based on their merit function values. The objective of the selection phase is to select the CIs to achieve the highest speedup

with and without considering a predefined area budget. Several research efforts have been devoted to the problem of CI selection. In [6], exact and approximate algorithms for solving the coverage and recurrence problems of candidate extensions are presented. The authors describe an optimal search technique that uses a branch-and-bound algorithm in conjunction with a greedy method. The use of the branch-and-bound technique, which is an exhaustive approach, is not practical in real applications with large DFG sizes. Hence, the authors also present a method that uses branch-and-bound in combination with a greedy approach to speed up finding the solution while not being trapped in a local optimum. The technique does not consider any area budget.

In [7], the design flow starts by partitioning the DFG of the application into several sub-graphs where each is considered as a candidate CI. Next, the set of candidate CIs is pruned through sub-graph isomorphism check, which reduces the CI selection search space. In the final phase, the CIs are selected by solving a unate covering problem. The unate covering algorithm is, however, a branch-and-bound technique which it is too expensive for real applications. To make the branch-and-bound technique practical, the authors prune search space using two techniques. One technique is based on a predefined maximum number of selected CIs, while the other one is based on the speedup achieved by the CIs (e.g., they use the CI speedup as the merit function). It should be noted that the second pruning method is only applicable in a design flow where the merit function considers only the cycle saving.

In [4], a technique for selecting CIs for the multi-issue processors is presented. The method, which relies on a branch-and bound algorithm, makes use of a few pruning techniques to ensure that the proposed solution approach remains tractable. The pruning methods are useful only for multi-issue processors. In [20], a complete design flow of the ISA extension was described. In the selection phase, a quadric optimization problem was used to select the best CIs. The authors mapped the CI selection problem to the Partitioned Boolean Quadric Problem (PBQP). The merit function invoked for this problem was based on the CS shown in Eq. (1).

An automatic framework for designing a customized processor that deals with the instruction set identification and CI selection is described in [1]. In this work the CI selection is done based on the greedy approach while the CSPA is used as a merit function. In the work presented in [5], a CI selection method is presented for extensible processors that are realized on FPGA devices. The CI selection is performed online and therefore the authors focus on fast and accurate estimation of the area and cycle saving for selecting and implementing the CIs. The CSPA is used as a merit function in the proposed algorithm. In [21], for the selection phase, the use of the integer linear programming (ILP) and greedy algorithms were proposed. For the latter approach, the CSPA and CyS merit functions are invoked.

In [22], a complete survey on ISA extension methods was reported. The survey showed that most of the prior works considered the speedup as the main objective, and used approaches such as Integer Linear Programming, Branch-and-Bound, and heuristic to select the CIs. Note that, the merit function could be defined based on terms such as the frequency of execution, number of distinct templates, and number of uncovered nodes. All these terms somehow correspond to the cycle saving of the selected CIs.

In [23], a linear programming relaxation technique to solve the binate covering was proposed. The goal of this technique was to select the sub-graphs that cover most parts of the input DFG. This method was not able to detect the recurrence of a template in different input DFGs. A method to select the CIs based on the symbolic arithmetic was proposed in [24]. By using the symbolic algebra, the design flow mapped the candidate CIs on the symbolic representation of the application. Based on this representation, the speed enhancement of the CIs was estimated. The main disadvantage of this method is that it is only applicable to the cases of CIs with a single output (i.e., MISO).

Table 1 Comparison between the works reviewed in this section

Reference	Optimal selection	Non-optimal selection	Considering cycle saving (CyS)	Considering cycle saving and area (CSPA)	Considering the CIs recurrence	Architecture/ hardware dependent
[6]	×	×	×		×	
[7]	×		×			
[4]	×		×			×
[20]	×		×			
[1]		×		×	×	
[5]		×		×		×
[21]		×	×	×		
[23]	×					
[24]	×					×
[25]		×		×	×	×

The method proposed in [25] selects the CIs for reconfigurable instruction set processors. Due to the area budget limitation of FPGAs, the policy of the proposed method was based on reducing the number of distinct templates. This way the area required for implementing the selected CIs was reduced while the speed enhancement decreased. The other technique which has been considered for reducing the area of CFU is resource sharing (see, e.g., [26, 27]). In this technique, after selecting the CIs, the data paths of the CIs were combined to reach smaller areas. This, however, increases the latency. In [15], the authors describe the CoWare tool which uses the description of the processor based on the LISA ADL language and generates the hardware HDL code of the processor as well as its software packages. The tool, however, cannot be used for finding the best CIs as is required for extensible processors. After finding the custom function unit (CFU), one may use CoWare to generate the hardware description and software packages of the extensible processors. Finally Table 1 shows the difference between the methods studied in this section.

In this paper, we describe a more appropriate merit function to increase the application speedup while considering an area budget constraint. This merit function improves the results of the greedy approach. In addition, we show that, by using this merit function, the search space of an exact algorithm (e.g., branch-and-bound in this work), may be pruned better compared to the case of conventional merit functions. Note that the proposed merit function can be utilized in any selection method which uses the merit value as the parameter to select the CIs. Also, combining this method with the resource sharing technique may result in smaller area usages.

4 Proposed merit function

We start this section by an example. Figure 4 shows the CS distribution of two sets of identified CIs. In one case, the average CS of the set of identified CIs is high while in the second case it is low. Now consider two selected CIs A and B, which have equal CSs. In the case of A, the CS of the CI is close to the average CS of the set of identified CIs, whereas in the case of B, the CS of the CI is higher than the average of the corresponding set of identified CIs. Hence, for the set of CIs in Fig. 4(a), the probability of finding another CI that has a similar (or even higher) CS but a smaller area than CI "A" is more than that for the

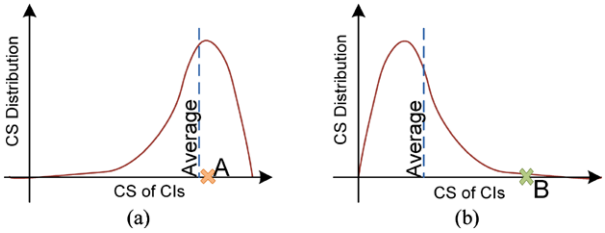


Fig. 4 Two different distribution functions for the CSs of two sets of identified CIs. (**a**) with high average CS, (**b**) with low average CS

set of CIs in Fig. 4(b) and CI "B". Therefore, the importance of CS component of the merit function in the selection process should be less for "A" compared to that for "B". Therefore, one should find a merit function which provides a higher merit value for the CI B.

To achieve this goal, the merit function may use the normalized CS instead of the absolute speedup (which is the same for the CIs "A" and "B"). Therefore, we suggest using $\frac{\mathrm{CS}_i}{\overline{\mathrm{CS}}}$ instead of CS_i, where the $\overline{\mathrm{CS}}$ is the average cycle saving of all the identified CIs and CS_i is the cycle saving of the ith CI. Note that, when a CI cycle saving is close to (larger than) the average cycle saving, the ratio $\frac{\mathrm{CS}_i}{\overline{\mathrm{CS}}}$ is close to (larger than) one and, hence, it provides a lower (higher) bias toward selecting the CI. In the case where there is a large number of CIs with the cycle savings around the average value, one has several options of selecting CI with the CS around this value and, hence, no need to put much emphasis on selecting a CI such as "A". For this type of CIs, the area overhead component of the merit function should play a more important role in the selection process. For the distributions with a small number of CIs with the cycle savings much larger than the average value, the merit function should emphasize on selecting these CIs. For these CIs, the relative importance of the area should be lower. Using a similar reasoning for the relative importance of the area in selecting CIs, we suggest using the ratio $\frac{A_i}{\overline{A}}$ in the merit function instead of A_i, where $\overline{A}$ is the average of the areas of all the identified CIs and A_i is the area of the ith CI. Since lower area consumption is desired, the area ratio should have a negative sign in the merit function. The above discussion, inspired our proposed merit function for the ith CI based on the normalized speedup and area as

$$Merit_{\mathrm{i}} = \frac{\mathrm{CS}_i}{\overline{\mathrm{CS}}} - \frac{A_i}{\overline{A}} \tag{6}$$

Using the average values in the merit function enables decision making process somehow aware of the overall merit values of other CIs. This does not exist in the conventional CSPA approach. Using CS–A also has the same problem as that of the SPA merit function. It should be noted, the CS and A terms in Eq. (6) could be normalized to the maximum values or the absolute values. Also, we are able to control the importance of each parameter in selecting the CIs by giving different weights to each ratio (α for the speed ratio and $(1-\alpha)$ for the area ratio).

If the selection is based on CI groups instead of individuals CIs, Eq. (6) changes to

$$Merit_{\mathrm{i}} = \frac{\sum_{j=1}^{|\mathrm{CI}\ group_i|} \mathrm{CS}_j}{\overline{\mathrm{CS}}} - \frac{A_i}{\overline{A}} \tag{7}$$

where the sigma operator shows the summation of the CSs of all the CIs in the ith CI group. We call this proposed merit function as Difference of Normalized Cycle Saving and Area (DNCSA). The intuition behind the proposed merit function is to consider the normalized cycle saving and area of each CI using a compact function. The goodness and badness

of each parameter in the merit function is determined by dividing them to their average values. The average value, which is obtained by averaging the corresponding parameters of all the CIs, somehow indicate that the overall value of that parameter for these CIs. Dividing the parameter of each CI to the average value shows the smallness or largeness of that parameter in a relative sense. For example, when the ratio of the speedup value of a CI to the average speedup value is smaller (larger) than 1, it means that the speedup value of this CI is relatively small (large) in the candidate set.

Finally, the runtime overhead of the proposed merit function is $O(n)$ where n is the number of the CIs in the candidate CI list. This originates from calculating the average of cycle savings for all CIs. By using this average value, we calculate the merit value for each CI group. During the selection process, in the worst case, when the selected CI group has conflict with the rest of the CI groups, the conflicting CIs must be removed from each of these CI groups and their merit values must be recalculated. Also, note that the complexity of updating the conflict graph is $O(n)$. In each iteration of the greedy selection algorithm, a set of CIs which belongs to a CI group is selected, and some CIs due to the conflict with the selected CIs will be removed from the candidate set. Hence, during the greedy selection phase, each CI only is met one time by the selection algorithm. It means that each CI during the selection phase is labeled as the selected or removed. Hence, the complexity of the greedy selection algorithm is $O(n)$ because it meets each CI only one time. In the best case, if the selected CIs have no conflict with the other CIs, the merit value of the CIs will not be changed and, hence, there is not any runtime overhead for recalculating the merit value of the CI group. In this case, the complexity is $O(n)$. In the worst case, the merit value of half of the CIs must be recalculated. This may be justified by assuming that half of the CIs are removed due the conflict with the selected CIs and each of these belongs to a different CI group. In this case the runtime of the greedy selection algorithm based on the proposed merit function is given by

$$\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\cdots=n\left(\sum_{i=1}^{\log n}2^{-i}\right)=O(n) \tag{8}$$

The same complexities exist for the greedy selection algorithms based on the CyS and CSPA merit functions.

5 Results and discussion

In this section, first we describe the simulation setup used to obtain the results and then discuss the results.

5.1 Simulation setup

To assess the efficacy of the DNCSA merit function, we developed a framework which is shown in Fig. 5. The flow starts with codes written in C/C++. The main parts of the framework are Profiling and Flow Graph Generation, Custom Instruction Identification/Classification, and Custom Instruction Selection.

GCC [28] is exploited to generate a 3-address representation, called GIMPLE, and control flow graph (CFG) of the benchmarks. GIMPLE representation converts complex C/C++ codes into an intermediate representation with no more than three addresses. The generated CFG is utilized to determine the basic blocks of the application. With the help of

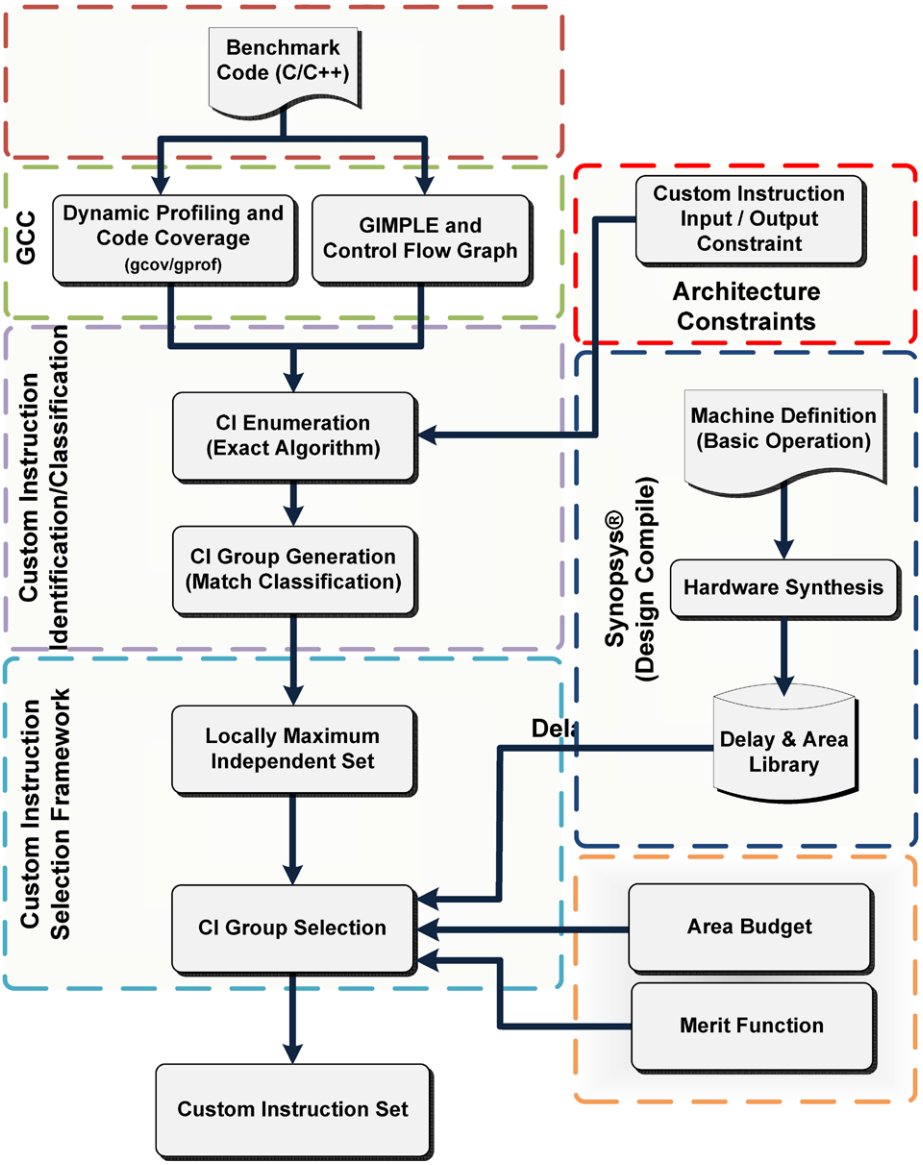


Fig. 5 Developed framework used to obtain results in this work

GCC profiling tools, (GCOV and GPROF [28]) the dynamic profiling and code coverage of each basic block is calculated to consider the dynamic behavior of the applications. In the next step the Multiple-Cut identification algorithm proposed in [2] is utilized to identify all the feasible CIs, considering input/output constraints. Once all the valid CIs are identified, the graph isomorphism algorithm [7] is applied to classify structural and functional equivalent CIs into the CI groups. Since there should not be any common nodes between any CIs, a heuristic algorithm is utilized to locally find the maximum independent set in each CI group. Finally, a CI selection algorithm is applied to the locally conflict-free CI groups

based on a specified merit function. We use both greedy and branch-and-bound techniques to evaluate our proposed merit function. For the branch-and-bound technique, we use the method proposed in [7]. The pruning technique proposed in [7] is not applicable when an area budget is given. Hence, to control the worst-case complexity of the branch-and-bound technique, we restrict the number of the CI groups by sending only a small number of the best CI groups (chosen based on their merit values) to the selection phase. It should be mentioned that this is not the only way of making the computation time of the branch-and-bound technique practical. As an example, one could run the branch-and-bound algorithm only for a specific time (e.g., one hour) and then use the best CI sets obtained up to that point. Hence, before the branch-and-bound selection, some CI groups are removed from the candidate set based on their merit values.

To be able to compute the speedup and area of CIs, we synthesize the basic operations using a 45 nm technology [29], and save them as a library. The library is used as an input to our framework. The area of each CI is normalized to the area of a NAND gate (Table 2). The template selection part explores all the templates and iteratively selects the template that has the maximum merit and removes all the matches that have at least one common node with the selected template (based on the conflict graph). The selection process attempts to find the CIs with the maximum CS while meeting the specified area constraint.

We have assumed an in-order processor where the instructions were fetched sequentially. Also, the instructions of the processor are reported in Table 2. Also, we have considered 2 cycles for the memory access and the I/O constraint was $4/4$. Also, the access to memory is serial without any concurrent access.

For the evaluation of the proposed technique, we examined different applications. They included *IP-Sec* and *MD5* as the representatives of the encryption benchmarks in the packet processing domain [30], *adpcm* and *bitcounter* from MiBench [31] in the domains of telecommunication and automotive, respectively, and finally, $G721$ benchmarks selected from Mediabench [32]. It should be noted that the input of the identification phase is the DFG of the benchmarks.

5.2 Results

In order to assume (define) reasonable area budgets while evaluating our proposed merit function, firstly, we need to measure the maximum area needed by CFUs for each exam-

Table 2 Basic operations and their areas

Primitives name (32-bits)	Area (normalized to the two inputs NAND gates)
ADD/SUB	965
REL/LES/GRT	295
OR/AND	48
EQT	138
SLCT	80
XOR	68
SHADD	1146
MUL	20189

Table 3 Speedup, area, and number of selected CIs in different benchmarks without any area constraint

Application name	Application code size (lines)	Application node count*	Area	Speedup	|CI groups candidate list|	Number of selected CI groups
IP-Sec	230	294	20116	2.04	14953	21
MD5	284	523	48335	2.23	29366	40
bitcounter	652	163	12421	2.02	27543	14
Adpcm	282	83	7350	1.37	44	12
G721decode	365	373	28625	1.34	609	30
G721encode	370	368	25905	1.33	686	30

*Without considering the forbidden nodes

ined application. Therefore, the developed framework (Fig. 5) is applied on six benchmarks without considering area budget. To show the performance improvement using the selected CIs, we use the speedup parameter that is obtained from

$$Speedup = \left(\frac{\text{ART}}{\text{ART} - \sum_{\forall Selected\ \text{CIs}} \text{CS}_i} \right) \tag{9}$$

where ART (application run-time) is the number of clock cycles to run the application on the base processor, and the CS_i is calculated based on Eq. (1). Note that to find the *ART* the access to memory for load and store instructions was assumed 2 cycles, while for other instructions, 1 clock cycle was assumed.

Table 3 reports the speedup and the maximum area used by the selected CIs for each application without considering any area constraint when the greedy approach is used in selection. Note that, in this paper, the area is expressed in terms of equivalent two input NAND gates. To evaluate the efficiency of the proposed merit function more precisely, we categorize the results in three aspects of speedup, area, and conflict. In Sects. 5.2.1 and 5.2.2, we investigate the effectiveness of the proposed merit function for both greedy and branch-and-bound selection algorithms.

In the DNCSA, the importance of CS and area in selecting the CIs may be controlled by giving different weights to each ratio. Figure 6 shows the impact of these weights on the speedup value. For this study, the considered area budgets were categorized into three different groups of small, medium, and large where each group contained the appropriate area budgets (e.g., for IP-Sec, small: 500, 1000, 3000, medium: 5000, 7000, 9000, large: 11000, 13000). To efficiently present the results, we have shown the average speedup values in each area group for different applications in the figure. The results for the areas considered in our study show that the ratio between the highest and lowest speedup values for the small, medium, and large areas are 1.02, 1.05, and 1.07 which are not very high. Also, note that when the area budget is small, higher speedup values are achieved when the importance of the area is more ($\alpha < 0.5$). For large area budgets, higher α values lead to higher speed gain. For the medium size budgets, when both ratios have the same weight of 0.5, the maximum speedup is attained. Note that, in the case of the other two area groups, this value of α also leads to speedup close to the maximum and, hence, for the rest of the paper, we consider $\alpha = 0.5$.

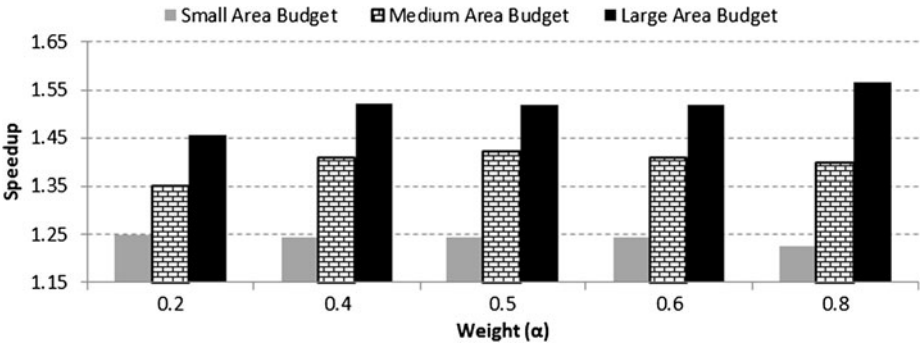


Fig. 6 Speedup values for different α values

5.2.1 Speedup

In this step, the maximum area needed for CIs of each application is determined, and based on that, the CI selection is performed under different area budgets. In Fig. 7, the speedup for each application under various area budgets is depicted. The CIs were selected based on the greedy approach when the merit functions were based on the DNCSA, CyS, and CSPA merit functions. Additionally, to evaluate the efficacy of the greedy selection method that enhanced with the proposed merit function compared to optimal techniques, in the case of three small benchmarks (i.e., *G721decode*, *G721encode* and *adpcm*), the speedups of the selected CFUs which are obtained by the ILP approach are reported in Fig. 7. Note that For other benchmarks, the problem size became too large for the ILP solver. The results show that the proposed merit function results in shorter execution time for all the applications. Also, the figure shows that for most of the benchmarks, by increasing the area budget, the speedup difference between the CyS and DNCSA merit function is reduced. This originates from this fact that the CyS select the CIs with higher speedups without considering the area usage. In most cases, the CIs with higher CSs have larger areas. When the area budget is small, selecting each large area CI limits the options for selecting the following CIs. Enlarging the area budget, however, increases the options for selecting the next CIs making the speedup of the CyS and DNCSA merit functions close to each other. In the *adpcm* and *bitcounter* benchmarks, when the area budget increases, for some area budget (e.g., in *adpcm* when the area budget is 8000), the speedup decreases. This reduction is due to selecting a large CI that reduces the remaining area budget in such a way that limits the selection of the next CIs. By increasing the area budget further, the problem is removed.

In the *MD5* and *IP-Sec* applications, the DNCSA merit function results in better CIs in terms of speedup under different area budgets compared to the CSPA merit function. In the case of *G721-encoder* and *G721-decode*, the selection of CIs based on the CSPA merit function yields higher speedup compared to the case of the DNCSA for the first area budget constraint (1000 units) while for the others better CIs are obtained when the DNCSA is used. In the case of *bitcounter*, the proposed merit function reaches higher speedups for all the area budgets except for the area budget of 3000 units.

For the *adpcm* application, in all area budgets, the DNCSA merit function leads to a slightly (about 1 %) higher speedup. The reason for obtaining trivial improvement over CSPA merit function is that the frequently executed sequences in *adpcm* belong to a very small loop that limits the number of identified CIs (Table 3). Furthermore, all of the identified CIs are distributed in few different cycle saving values (refer to Fig. 8(a)) and, hence,

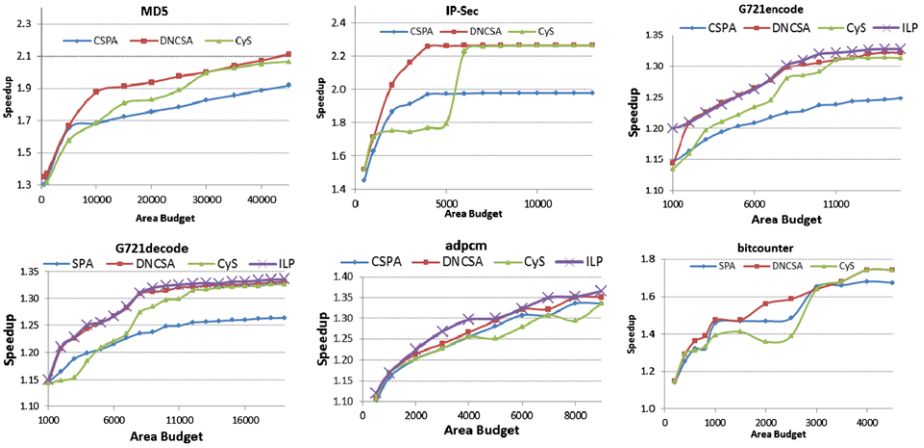


Fig. 7 Speedup under different area budgets (Greedy selection)

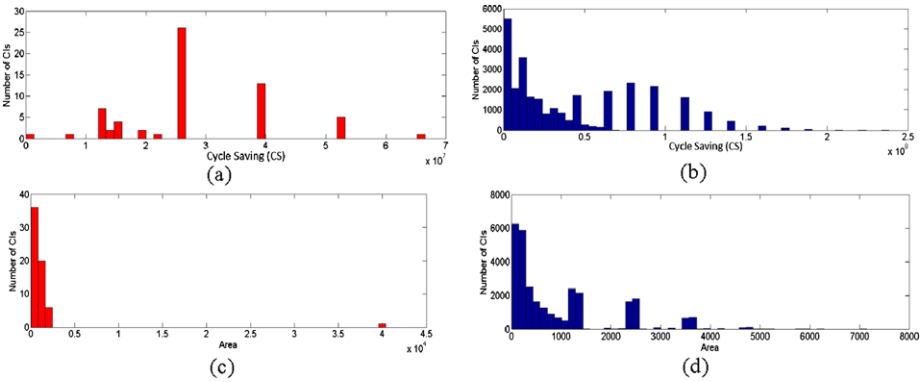


Fig. 8 The cycle saving distribution functions of the CIs in (**a**) adpcm, (**b**) IP-Sec and area distribution functions in (**c**) adpcm, (**d**) IP-Sec. The cycle savings of CIs are calculated based on Eq. (1)

the $\frac{\text{CS}}{\text{CS}}$ values are the same for most of CIs limiting the options for the CI selection. This limits the efficacy of the proposed merit function for this benchmark. The proposed merit function yields better results when CIs with a variety of speedup options are available in the candidate set. Figure 8 shows the cycle saving and area distributions for the two *adpcm* and *IP-Sec* benchmarks. As is evident in this figure, the CS and area options of the identified CIs for *IP-Sec* are more than those for *adpcm* (30 versus 603).

The comparison of the speedups obtained by the proposed merit function and the ILP approach indicates that, on average, the speedups of DNCSA is about 1 % smaller than those of the ILP for the worst case (i.e., the *adpcm* benchmark). This suggests the high efficacy of the proposed merit function.

Figure 9 shows the average of speedup improvement when DNCSA merit function is used instead of the CSPA and CyS merit functions. The minimum (maximum) speedup improvement compared with the CSPA case is 1 % (12.6 %) which belongs to *adpcm* (*IP-Sec*). Also, the results indicate that, the DNCSA merit function has the maximum improvement of ∼6.8 % compared to the CyS merit function which occurs for the case of *IP-sec* while

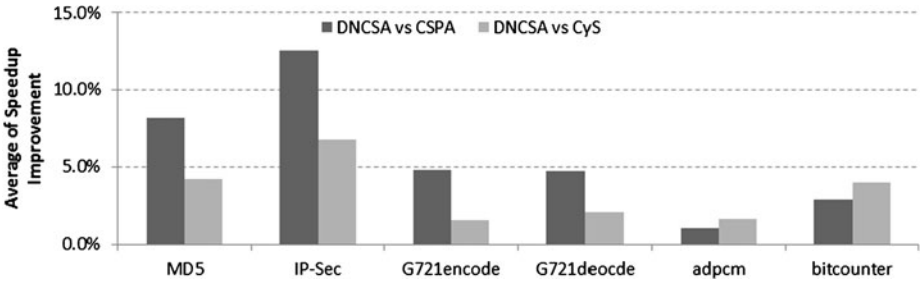


Fig. 9 Average of speedup improvement of the DNCSA merit function compared to the CSPA and CyS merit functions for different area constraints

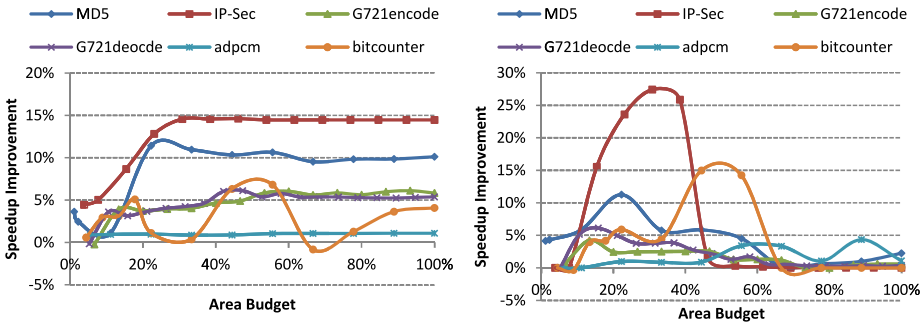


Fig. 10 Speedup improvement of the DNCSA merit function compares to the (**a**) CSPA and (**b**) CyS merit functions for different area constraints

the minimum improvement belongs to the *G721encode* which is ∼1.5 %. On average, the proposed merit function improves speedup by 5.7 % (3.3 %) in comparison with the CSPA (CyS) merit function case.

Figure 10 shows the speedup improvements of the DNCSA merit function with respect to CSPA and CyS merit functions for different area budgets. The horizontal axis in both figures shows the ratio of the considered area budget to the maximum area budget (see the horizontal axes in Fig. 7). In Fig. 10(a), for *adpcm*, the improvement is constant for all the area budgets. On the other hand, the speedup improvements in other five benchmarks are variable for different area budgets. In *IP-sec*, *G721-encode*, and *G721-decode* for smaller area budgets the speedup improvements are increased and then become constant. In *MD5* and *bitcounter*, the speedup improvements fluctuate but finally they saturate to constant values. This may be justified by noting the fact that in the case of small area budgets, the DNCSA selects CIs with higher cycle savings compared to those of the CSPA. By increasing the area budget, in addition to the CIs that were selected in the case of small area budgets, some other CIs were selected which are the same for both DNCSA and CSPA. Therefore, the speedup difference between the DNCSA and CSPA becomes saturated for large area budges. Note that in the case of *adpcm*, the results indicate that most of the selected CIs for most of the budgets in the two cases of the DNCSA and CSPA are the same leading to a constant speedup difference for these areas. Hence, one may conclude that the speedup enhancement obtained by DNCSA merit function depends on the area budget while becoming stable at some points. This situation is similar for CyS merit function where, in most cases,

its speedup approaches that of the DNCSA merit function when the area budget increases (>70 %). The maximum difference between these two methods is when the area budgets are between the 20 % and 60 %. As explained before, by increasing the area budget, the importance of the area decreases giving rise to the selection of the same CIs for both merit functions. Thus, the difference between the DNCSA and CyS approaches zero as the area increases.

We should mention that we studied the impact of the I/O constraint on the efficacy of the proposed merit function. In this study, we considered the I/O constraints of 3/3 and 2/2 in addition to 4/4 which was considered before. The results show that, expect for the case of the *MD5* (3/3), *bitcounter* (3/3), and *G721encode* (2/2), the DNCSA outperforms the other two merit functions (the two numbers inside the parenthesis next to the name of the benchmark specify the I/O constraint). The results show the in the case of the I/O constraint of the 3/3, the average speedup of the DNCSA is about 7.6 % (6.1 %) higher than the case of the CyS (CSPA) objective function. Also, in the case where the I/O constraint is 2/2, the DNCSA results in on average 8.8 % (7.8 %) higher speedup compared to CyS (CSPA).

To investigate the efficacy of the proposed method in the branch-and-bound technique, we selected 50 best CI groups as the candidates CIs using the pruning technique (and then the branch-and-bound technique was utilized to select the optimal CI set). In addition, the number of optimally selected CI groups was limited to five. Setting this constraint is reasonable because, in practice, the number of CI groups may not be an arbitrary number due to, e.g., baseline processor architecture constraint. The number five is an example which is used in our study. Other numbers in this order also may be used. Of course the speedup improves by increasing this number.

We ran the pruning algorithm based on the DNCSA, CyS, and CSPA merit functions for each benchmark. The results which are reported in Fig. 11 which show that, for most cases, the DNCSA merit function yields higher speedups. However, when the area budget is small, in some cases, e.g., *MD5* and *IP-Sec*, the CSPA merit function outperforms the DNCSA merit function. When the area budget is small, pruning the set of CI groups based on the DNCSA merit function causes large CI groups to remain in the set and the small CI groups to be removed from the set. Therefore, the DNCSA merit function has fewer choices of the CIs for filling the allocated area. This is not the case that is valid for the CSPA where the merit function does not eliminate small area choices. Note that this did not occur for the greedy approach in which we did not have to prune the search space due to much higher efficiency

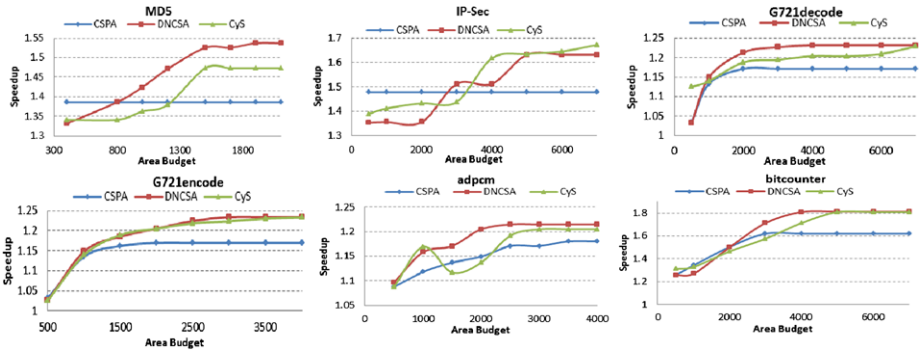


Fig. 11 Speedup under different area budgets (branch-and-bound selection)

of the greedy technique. For the *MD5* and *IP-Sec* benchmarks, the CSPA merit function resulted in the same speedups for all area budgets by selecting the same set of CIs. Hence, the CSPA merit function is not able to use the area budget efficiently. In the case of DNCSA merit function, higher speedups are achieved by choosing better CIs. Contrary to the greedy algorithm, in the branch-and-bound technique, the DNCSA merit function outperforms the CSPA merit function for the *adpcm* benchmark. This originates from the fact that there are many CIs with small area budgets and large cycle savings in the case of the *adpcm* benchmark. The way that the proposed merit function considers both the area and cycle savings caused that more high cycle savings CIs remain in the candidate set. In the branch-and-bound technique, DNCSA merit function results in 3.2 % better performance on average for all the area budgets in comparison to the CSPA merit function for *adpcm* benchmarks. When the CIs are pruned based on the CyS merit function, the probability of keeping larger CIs in the candidate set is increased. Hence, in some cases, such as *G721decode* and *MD5*, the speedup of the CyS merit function is not as good as that of the DNCSA merit function.

In some cases, for small area budgets, the CyS merit function yields a higher speedup. This due to the fact that when the identified CIs are pruned based on the CyS merit function, higher speedup CIs with small areas may be kept in the candidate set. This is different from the DNCSA merit function case where the area is also considered in the merit function and, hence, the remained CIs with small areas may not have necessarily as high CSs as those remained in the CyS merit function case. For the *IP-sec* benchmark, the CyS merit function outperforms the DNCSA merit function for most of the area budgets. This may justified by noting that most of the candidate CIs for this *IP-sec* benchmark have small area budgets including those with high cycle savings (see Fig. 8(d)). On the other hand, the CyS merit function has a tendency toward selecting CIs with high cycle savings. Therefore, in this case, the CyS merit function could perform better. In the case of *G721encode*, for most of the area budgets, due to selecting the same CIs, both methods reach a speedup value. Finally, similar to the case of the greedy approach, when the area budget increases, the speedup of these two methods approach each other.

To investigate the impact of pruning the CI groups on the speedup of the selected CIs, we evaluated these techniques while the maximum number of the CI group candidates was set to 80. Figure 12 depicts the average of speedup improvements when the CIs are pruned based on the DNCSA merit value compared with the two other methods while the numbers of the CI group candidates are 50 and 80, respectively. For *MD5*, the outperformance

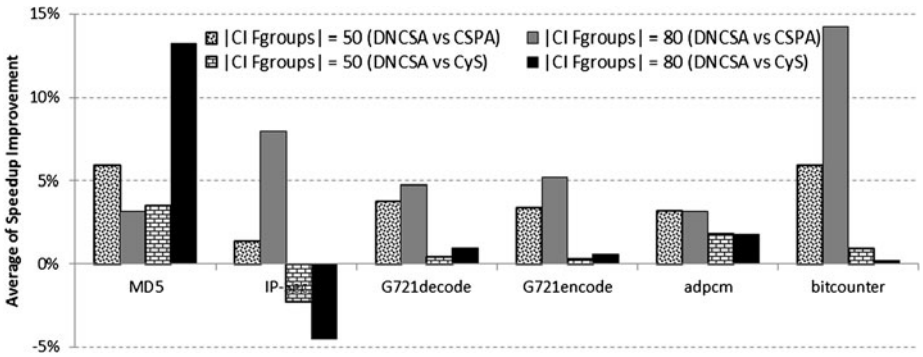


Fig. 12 Average of speedup improvement of the DNCSA merit function compared to the CSPA and CyS merit functions for different area constraints when the maximum number of the CI group candidates are 50 and 80

of DNCSA merit function with respect to CSPA merit function was reduced to ∼3.17 % when the number of the CI group candidates increased. In the *MD5* case, by increasing the candidate set, the speedup improvements of the DNCSA merit function compared to the CyS merit function is increased from ∼3.54 % to the ∼13.26 %. For *IP-Sec*, the speedup of the DNCSA merit function compared with the CSPA merit function is increased (form ∼1.35 % to 7.98 %) as the number of CI group candidate is raised. For this benchmark, the CyS merit function yields better results for both candidate set sizes and their difference is increased by increasing the candidate set size. For the cases of *G721decode* and *G721encode*, the DNCSA merit function provides slightly higher speedup compared to those of the CyS merit function, and its outperformance is increased by increasing the candidate set size. For the *bitcounter* benchmark, the DNCSA merit function selects better CIs in average compared with the CyS merit function but its speedup difference is decreased by increasing the size of the candidate set. For *adpcm*, the total number of the CI group candidates was smaller than 50 and, hence, increasing the size of the candidate set did not lead to any fixed trend in the speedup improvement.

In general, the results show that increasing the candidate size has different impacts on the benchmarks. However, in most cases, the outperformance of the proposed merit function compared to the other merit functions decreases. The reason is that increasing the candidate set enhances the chance of having the same CIs in the corresponding pruned candidate sets. Therefore, the exact selection method may select the same CIs from these sets.

5.2.2 Area

In this part, we show that the DNCSA merit function utilizes the area budget more efficiently than the CSPA and CyS merit functions. First, we start by the greedy selection algorithm. Based on the results presented in Fig. 7, for a given area constraint, the DNCSA merit function results in better speedups for all cases. Figure 13 shows the average of the area usage (utilization) which is calculated using the following equation:

$$AreaUsage_{\mathrm{AVG}} = Average\left(\left\{\forall Area\ budget, \frac{Area\ Usage}{Area\ Budget} \times 100\right\}\right) \tag{10}$$

The results show that, in all the cases, the CyS merit function has more area budget usage (∼2.1 %) compared with that of the DNCSA merit function, while its speedup is normally lower than that of the DNCSA merit function. Also, in the case of the CSPA merit functions, the proposed method results in better speedup by having just 1 % more area usage for all the applications, except for *bitcounter*. In *bitcounter*, the proposed merit function reaches

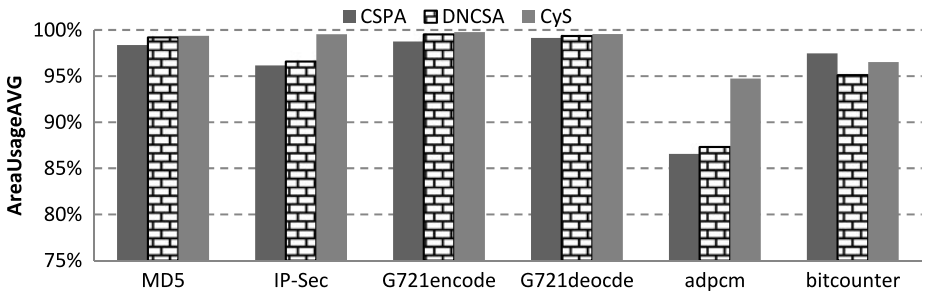


Fig. 13 Average area usage for three merit function methods in greedy selection algorithm

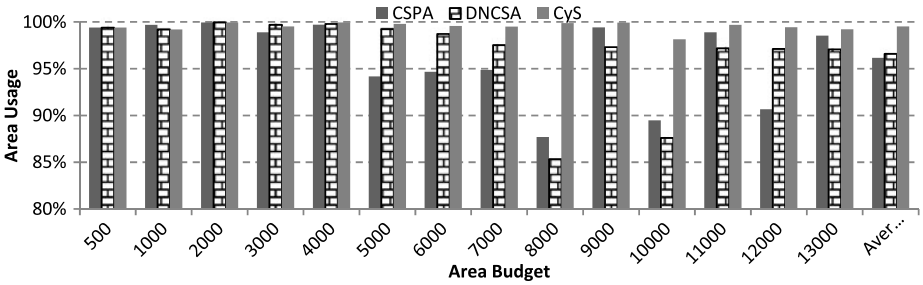


Fig. 14 Area usage under different area budgets for three merit functions for the *IP-Sec* application

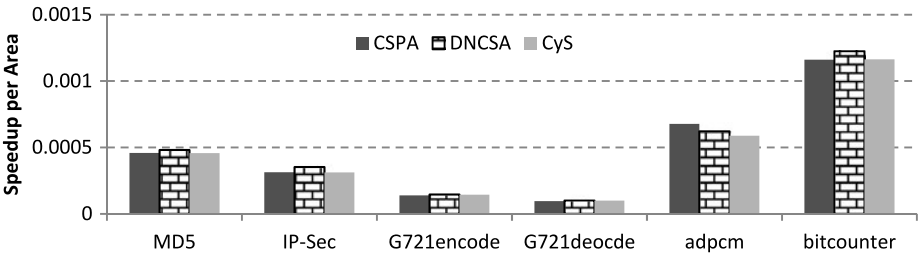


Fig. 15 The average of Speedup/Area under different area budgets for different applications while greedy selection algorithm is used

a higher speedup while using less area. Figure 14 depicts the difference between the area usages of the DNCSA merit function with those of the CyS and CSPA merit functions for different area budgets for the *IP-Sec* benchmark. For some area budgets (e.g., 4000, and 5000 units), DNCSA merit function uses more area compared with CSPA merit function while in others it uses smaller areas (e.g., 8000, and 9000 units). In all of them, DNCSA merit function reaches higher speedup leading to better CS/area values compared to the case of CSPA merit function. For this benchmark, the CyS merit function uses more areas under all the budgets except when it is 3000 where the CyS merit function has a smaller usage. For the two first budgets, the usages and speedups (Fig. 7) of the DNCSA merit function and CyS merit function are similar. Also, for the area budget greater than 10000, while the CyS merit function uses more area, the speedups of both merit functions are similar.

To illustrate the efficiency of the proposed merit function in using area, Fig. 15 shows the average of the ratio of Speedup/Area for different area budgets on each benchmark. For all the benchmarks, except for *adpcm*, our method provides higher ratios. In the *adpcm* benchmark, the CSPA merit function outperforms both the DNCSA and CyS merit functions.

After studying the impacts of the DNCSA merit function on the area usage for the greedy selection algorithm, we performed the same study for the branch-and-bound technique. Figure 16 depicts the average of the area usage of the DNCSA, CSPA, and CyS techniques when the branch-and-bound algorithm is used. As the results show, the DNCSA merit function uses more area in comparison to the CSPA merit function. By increasing the maximum number of the CI group candidates (from 50 to 80), the difference between the area usage of the CSPA and DNCSA merit functions reduces. In *adpcm*, despite the other benchmarks, the area usage of CSPA merit function is more than the area usage of DNCSA merit function. Additionally, the comparison between the area usages of the CyS and DNCSA merit functions shows that in some benchmarks (e.g., *MD5*) the DNCSA merit function uses more area

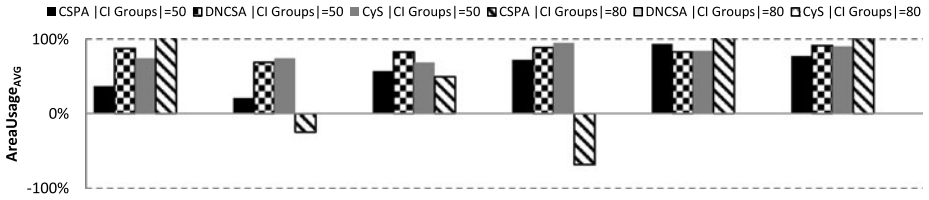


Fig. 16 Average area usage for three merit function methods in the branch-and-bound selection algorithm

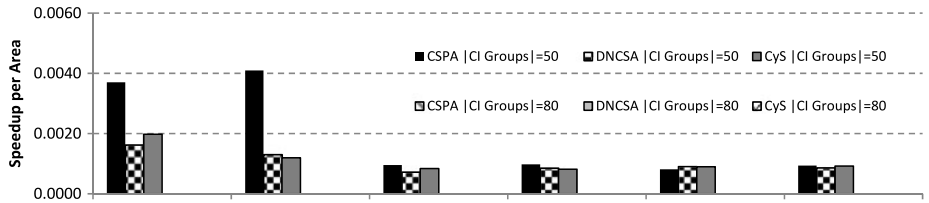


Fig. 17 The average of Speedup/Area under different area budgets for different applications while branch-and-bound selection algorithm is used

while in others (e.g., *G721decode*) the CyS merit function uses more area. By increasing the size of the candidate set, in some benchmarks (e.g., *G721encode*) the area usages of both methods become about the same.

Figure 17 shows the efficiency of the proposed merit in using area. In all benchmarks, except for *MD5* and *IP-Sec*, results for the CSPA and DNCSA merit functions are the same. In *MD5* and *IP-Sec*, the CSPA merit function has more efficiency than DNCSA merit function. However, even for these cases, the speedup obtained using DNCSA merit function is higher for the given area budget. Also, the values of Speedup/Area for both CyS and DNCSA merit functions, in most cases, are close. For the *MD5* (*IP-sec*) benchmark, CyS (DNCSA) merit function outperforms DNCSA (CyS) merit function. Note that, based on the problem formulation (Eqs. (2) and (3)), achieving a higher performance using a fixed area budget is a key parameter in designing digital systems.

6 Conclusion

This paper proposed a new merit function for the selection phase of the custom instruction in the design flow of application-specific instruction-set processors (ASIPs). Unlike the normally used merit functions which were either based on only the cycle saving (CyS) or the ratio of the cycle saving to area (CSPA), in the proposed merit function (Difference of Normalized Cycle Saving and Area (DNCSA)), the selection of each CI was performed using the difference of the normalized cycle saving and area. Since for the normalization, the average of the parameter was used, the selection was done by being somehow aware of the overall merits of other CIs. The efficacy of the proposed merit function on both greedy and branch-and-bound techniques was studied by applying it to several benchmarks from different application domains. Also, to show the effectiveness of the proposed merit function in using the area budget, the study was performed for different area budget values. The results showed that the proposed merit function provided higher speedup compared to the case of CSPA merit function. Also, for large area budgets, in most cases, the CyS and DNCSA merit

functions provided similar speedups, while for small area budgets, the DNCSA merit function outperformed the CyS merit function. On average, the speedup of the DNCSA merit function compared to the CSPA (CyS) merit function was about 4.83 % (2.48 %) more.

Appendix

In this section, we provide the motivation for the proposed merit function. For the sake of simplicity and without loss of generality, for this example, we make several simplifying assumptions which are marked by a star in the text below. Assuming that by using the exact identification algorithm of [2], all CIs that meet the defined constraints (e.g., the number of input and output) for an application are identified. In this example, there are eighteen identified CIs, and similar CIs (based on functional and structural isomorphism) are classified in seven internally-similar CI groups.

The conflict graph, the area (A) and clock saving (CS) factor of each CI group are shown in Fig. 18. Note that CS values in the graph show the clock saving of the CIs for a single iteration of some parts in an application. To make the example simple, we assume that the CIs have no intra-conflict* (i.e., the CIs within each group has no conflict), but they may have inter-conflict* (i.e., conflicts among the CIs that belong to different CI groups). Note that an edge between any two nodes in Fig. 18 signifies that all CIs of the corresponding CI groups have some conflict with each other. This also means that by selecting a CI group, all CI groups (nodes) that have a conflict with the selected group (there is a conflict edge between them) must be removed from further consideration.

Now, let us define the parameters CS and CS_{Norm} for each CI in a CI group (see Fig. 18). The parameter CS denotes the cycle saving factor for each CI in the group (in this example,

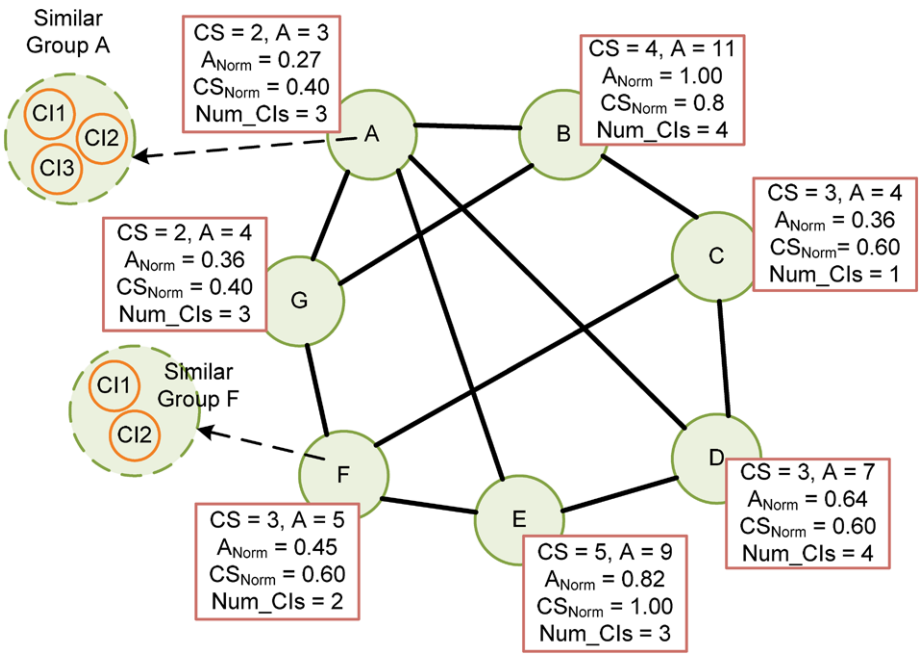


Fig. 18 The conflict graph of the example application

all the CIs of a CI group have the same cycle saving*). Note that, just for the sake simplicity, this assumption has been made only for the motivational example. The purpose of presenting this simple example is to demonstrate that the proposed merit function may improve the speedup of the extensible processors compared to the existing merit functions under the same conditions. Different combinations of cycle savings could be assumed for CIs in the group. These combinations would provide different levels of effectiveness for the proposed merit function compared to the existing merit functions. The results for the efficacy of the proposed function for different combinations of cycle savings in a group are presented in Sect. 5. The parameter CS_{Norm} is the normalized cycle saving. For the ith CI in a CI group, this value is calculated from

$$CS_{Norm,i} = \frac{CS_i}{\text{Max}(\{\forall CS_j \, j \epsilon Candidate \text{ CI } List\})} \tag{11}$$

We also define the parameters A, A_{Norm}, and Num_CIs for each CI group. Due to fact that all the CIs within a CI group use the same CFU, the parameters A and A_{Norm} denote the area usage and normalized area usage of the CFU which is used for each CI group, respectively. The parameter Num_CIs represents the number of CIs of the CI group. The parameter A_{Norm} for the ith CI in each CI group is obtained from

$$A_{Norm,i} = \frac{A_i}{\text{Max}(\{\forall A_j \, j \epsilon Candidate \text{ CI } List\})} \tag{12}$$

The normalizations, which are performed using the corresponding maximum values, give rise to the values between 0 and 1 in both cases.

One of Eqs. (4) or (5) is evaluated for all the nodes in the conflict graph and the node with the highest merit value is selected at each iteration of the selection algorithm. Then, the adjacent nodes to the selected node in the conflict graph are removed. This process continues until no node remains in the conflict graph or the area constraint is violated.

In this example, we assume that the area budget is equal to 13 units. To select the CIs from the candidate set, we used the greedy approach. Note that since the design space of this example is very small, we could have used the branch-and-bound technique to obtain the optimal CIs. The selected groups are depicted in Fig. 19. In this example, the merit values are calculated using A_{Norm}, and CS_{Norm} values of the CIs. First, we consider the case

Fig. 19 Solving the example problem with different merit functions (**a**) CSPA, (**b**) CyS, (**c**) proposed merit function (DNCSA)

of the CSPA merit function. In the first iteration, because the value of the merit function for CI group A is the highest (4.40), this node is selected. Because of the conflict with the node A, CI groups B, D, E and G are removed. After this step, the remaining area is 10 units ($13 - \text{area(A)} = 10$). In the next (last) iteration, from the remaining CI groups (C and F), the CI groups F is selected which has conflict with the group C. Hence, the group F is the last selected group. After selecting these groups, the final cycle saving may be calculated as

$$\text{CS}_{\text{CSPA}} = (\text{CS}_{Norm_A} \times Num\_\text{CIs}_A + \text{CS}_{Norm_F} \times Num\_\text{CIs}_F) \times \text{CS}_{\max} \tag{13}$$

where the $\text{CS}_{\max}$ is the maximum CS among all the identified CIs.

If the CyS merit function is used, only the group B will be selected. The CS of this group is 16 ($\sum \text{CS}_{Norm} = 3.2$) which is greater than the other CI groups. By selecting the group B, the groups A, C, and G must be removed due to conflict. Also, since the remaining area budget is small (13 – area (B) = 2), no other CI group may be selected.

In this example, we achieve a maximum cycle saving of 16, by using 11 area units (two area units are unused). However, the optimal answer to this problem is the groups E and G, which results in a cycle saving of 21 and uses the total area budget. This shows that using CSPA and CyS as the merit functions do not necessarily lead to the optimal solution. The reason is that, CIs with few primitive nodes (such as adder and shifter) and small areas have higher priority to be selected due to their larger CSPAs. On the other hand, CIs with many nodes usually have a higher cycle saving but also have many nodes (and large area) which leads to a large number of conflicts with other CIs. Using CSPA as a merit value can result in selecting CIs with few nodes and a lower CS compared to CIs with many nodes but lower CSPAs. The two consequences of using CSPA as a merit function are selecting low CS CIs with few nodes and removing CIs with many nodes (higher CSs) due to the conflicts with the previously selected CIs. Now, let us consider the case of the CyS merit function which selects the CIs with the higher CS, without considering the area budget usage in the merit function. For this case, after each CI selection, first, the available area budget will be updated and then the CIs whose areas are larger than the updated area budget will be removed from candidate set. As mentioned before, the CIs with higher CSs usually have larger areas and normally more conflict with other CIs. Both of these lead to limiting the choices available for selecting the next CI and, hence, less chance of increasing the speedup much further.

In the case of the proposed merit function, the group E which is the best CI group is selected in the first iteration (see Fig. 19(c)). After selecting this group, the groups A, D, and F are removed due to the conflict. After selecting the group E, the area budget reduces from 13 to 4 ($13 - \text{area(E)} = 4$) and, hence, in the next iteration, the selection must be done between the groups C and G. The merit value of the group G is greater than that of the group C and, hence, is selected as the better group in the second iteration. After this selection, the area budget reduces to zero terminating the selection phase. Hence, the performance gain of the proposed merit function is better than the conventional merit functions for this example.

References

1. Clark NT, Zhong H, Mahlke S (2005) Automated custom instruction generation for domain-specific processor acceleration. IEEE Trans Comput 54:1258–1270. doi:10.1109/TC.2005.156
2. Pozzi L, Atasu K, Ienne P (2006) Exact and approximate algorithms for the extension of embedded processor instruction sets. IEEE Trans Comput Aided Des 25:1209–1229. doi:10.1109/TCAD.2005.855950
3. Keutzer K, Malik S, Newton AR (2002) From ASIC to ASIP: the next design discontinuity. In: Proceedings of international conference on computer design: VLSI in computers and processors, pp 84–90. doi:10.1109/ICCD.2002.1106752

4. Lu YS, Shen L, Huang LB, Wang ZY, Xiao N (2009) Optimal subgraph covering for customisable VLIW processors. Comput Digit Tech 3:14–23. doi:10.1049/iet-cdt:20070104
5. Siew-Kei L, Srikanthan T, Clarke CT (2009) Selecting profitable custom instructions for area–time-efficient realization on reconfigurable architectures. IEEE Trans Ind Electron 56:3998–4005. doi:10.1109/TIE.2009.2017091
6. Bonzini P, Pozzi L (2008) Recurrence-aware instruction set selection for extensible embedded processors. IEEE Trans Very Large Scale Integr (VLSI) Syst 16:1259–1267. doi:10.1109/TVLSI.2008.2001863
7. Clark N, Hormati A, Mahlke S, Yehia S (2006) Scalable subgraph mapping for acyclic computation accelerators. In: Proceedings of international conference on compilers, architecture and synthesis for embedded systems, pp 147–157. doi:10.1145/1176760.1176779
8. Atasu K, Ozturan C, Dundar G, Mencer O, Luk W (2008) CHIPS: custom hardware instruction processor synthesis. IEEE Trans Comput-Aided Des Integr Circuits Syst 27(3):528–541. doi:10.1109/TCAD.2008.915536
9. Biswas P, Banerjee S, Dutt ND, Pozzi L, Ienne P (2006) ISEGEN: generation of high-quality instruction set extensions by iterative improvement. IEEE Trans Very Large Scale Integr (VLSI) Syst. 14:754–762. doi:10.1109/DATE.2005.191
10. Clark N, Zhong H, Mahlke SA (2003) Processor acceleration through automated instruction set customization. In: Proceedings of the 36th annual IEEE/ACM international symposium on microarchitecture, pp 129–141. doi:10.1109/MICRO.2003.1253189
11. Kastrup B, Bink A, Hoogerbrugge J (1999) ConCISe: a compiler-driven CPLD-based instruction set accelerator. In: Proceedings of the seventh annual IEEE symposium on field-programmable custom computing machines, pp 92–101. doi:10.1109/FPGA.1999.803671
12. Goodwin D, Petkov D (2003) Automatic generation of application specific processors. In: Proceedings of international conference on compilers, architecture and synthesis for embedded systems, pp 137–147. doi:10.1145/951710.951730
13. Yazdanbakhsh A, Salehi ME, Fakhraie SM (2010) Architecture-aware graph-covering algorithm for custom instruction selection. In: Proceedings of the 5th international conference on future information technology, pp 1–6. doi:10.1109/FUTURETECH.2010.5482719
14. Muhammad R, Apvrille L, Pacalet R (2008) Evaluation of ASIPs design with LISATek. Lecture notes in computer science, vol 5114. Springer, Berlin, pp 177–186. doi:10.1007/978-3-540-70550-5_20
15. The LISATek™ solution: automated embedded processor design and software development tool generation. http://www.coware.com/PDF/products/LISATek.pdf
16. Biswas P, Dutt N, Ienne P, Pozzi L (2006) Automatic identification of application-specific functional units with architecturally visible storage. In: Proceedings of the conference on design, automation and test in Europe (DATE), pp 212–217. doi:10.1109/DATE.2006.244088
17. Cheung N, Parameswaran S, Henkel J (2003) INSIDE: instruction Selection/Identification and design exploration for extensible processors. In: Proceedings of the international conference on computer aided design, pp 291–297. doi:10.1109/ICCAD.2003.1257681
18. Clark N, Jason B, Michael C, Mahlke S, Biles S, Flautner K (2005) An architecture framework for transparent instruction set customization in embedded processors. In: Proceedings of the 32nd annual international symposium on computer architecture, pp 272–283. doi:10.1109/ISCA.2005.9
19. Gonzalez RE (2000) XTENSA: a configurable and extensible processor. IEEE MICRO. 20(2):60–70. doi:10.1109/40.848473
20. Scharwaechter H, Kammler D, Leupers R, Ascheid G, Meyr H (2011) A retargetable framework for compiler/architecture co-development. Des Autom Embed Syst 15:1–32. doi:10.1007/s10617-011-9080-8
21. Pan Y, Mitra T (2004) Characterizing embedded applications for instruction-set extensible processors. In: Proceedings of the design automation conference (DAC), pp 723–728
22. Galuzzi C, Bertels K (2011) The instruction-set extension problem: a survey. ACM Trans Reconfigurable Technol Syst 4(18):1–28. doi:10.1145/1968502.1968509
23. Liao S, Devadas S (1997) Solving covering problems using lpr-based lower bounds. In: Proceedings of the 34th annual conference on design automation (DAC'97), pp 117–120. doi:10.1145/266021.266046
24. Peymandoust A, Pozzil L, Ienne P, Micheli GD (2003) Automatic instruction set extension and utilization for embedded processors. In: Proceedings of the 14th international conference on application-specific systems, architectures and processors (ASAP'03), pp 108–118. doi:10.1109/ASAP.2003.1212834
25. Lam S-K, Srikanthan T (2009) Rapid design of area-efficient custom instructions for reconfigurable embedded processing. J Syst Archit 55(1):1–14
26. Brisk P, Kaplan A, Sarrafzadeh M (2004) Area-efficient instruction set synthesis for reconfigurable system-on-chip designs. In: Proceedings of the 41st annual design automation conference (DAC), pp 395–400

27. Zuluaga M, Topham N (2009) Design-space exploration of resource-sharing solutions for custom instruction set extensions. IEEE Trans Comput-Aided Des Integr Circuits Syst 28(12):1788–1801. doi:10.1109/TCAD.2009.2026355
28. The GNU operating system. www.gnu.org
29. Nangate 45 nm open cell library (2008) . http://www.nangate.com
30. Ramaswamy R, Wolf T (2003) PacketBench: a tool for workload characterization of network processing. In: Proceedings of IEEE international workshop on workload characterization, pp 42–50. doi:10.1109/WWC.2003.1249056
31. Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB (2001) MiBench: a free, commercially representative embedded benchmark suite. In: Proceedings of 4th IEEE international workshop on workload characterization, pp 3–14. doi:10.1109/WWC.2001.15
32. Lee C, Potkonjak M, Mangione-Smith WH (1997) MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In: Proceedings of 30th annual IEEE/ACM international symposium on microarchitecture, pp 330–335. doi:10.1109/MICRO.1997.645830