Selecting Profitable Custom Instructions for Area–Time-Efficient Realization on Reconfigurable Architectures

Siew-Kei Lam, *Member, IEEE*, Thambipillai Srikanthan, *Senior Member, IEEE*, and Christopher T. Clarke

***Abstract*—Profitable custom instructions provide higher performance for a given reconfigurable area. Hence, choosing profitable custom instructions that are also area–time efficient is essential if design constraints must be met by field-programmable-gate-array (FPGA)-based reconfigurable processors. In this paper, we propose a framework for FPGA-based reconfigurable processors in order to rapidly identify a reduced set of profitable custom instructions without the need for actual hardware synthesis. The proposed framework is capable of estimating the area utilization and latencies of custom instructions on lookup-table-based commercial FPGAs. Simulations based on 15 applications from benchmark suites show that the proposed method provides, on average, an area reduction of over 29% for loss of mere 1.3% in compute performance. Our evaluations also confirm that the proposed framework is superior to an existing area-optimization approach that relies on exploiting the regularity of custom instruction data paths. In particular, an average area–time product gain of over 59% was achieved by deploying a reduced set of custom instructions obtained using the proposed framework.**

***Index Terms*—Custom instruction selection, embedded systems, high-level estimation, reconfigurable hardware.**

I. Introduction

As THE nonrecurring engineering costs of application-specific integrated circuits continue to outweigh the per unit cost of field-programmable gate arrays (FPGAs) for high-volume applications, FPGAs have progressively dominated the integrated-circuit market [1]. FPGA technology can now be found in a multitude of application domains that are subjected to tight market and technological constraints [2]. This includes applications for industrial electrical control systems which have increasing level of performance expectations coupled with stringent hardware-cost constraints [3].

It has been projected that by 2010, more than 40% of all FPGA designs will contain a microprocessor [4]. Platforms, which consist of a microprocessor core that is coupled with a reconfigurable functional unit (RFU), are defined as *reconfigurable processors*. These reconfigurable processors offer the possibility of extending the basic instruction set of the microprocessor by introducing custom functional units on the RFU. Commercially available reconfigurable processors include the Altera Nios II [5], Xilinx MicroBlaze [6], and Stretch [7] processors. As opposed to loosely coupled schemes where data are communicated between the microprocessor and RFU through a shared memory, the tightly coupled scheme employs internal register files for data transfer. Hence, the communication overhead does not pose a major bottleneck to the system performance in tightly coupled schemes [8] and is often ignored during performance evaluation.

Due to its promising ability to overcome technological and market challenges, reconfigurable processors will play an important role in future embedded system-on-a-chip platforms. In order to cater to the demands for cost efficiency, reconfigurable processors in the near future are likely to offer a larger range of programmable logic capacity to suit various applications or application domains. The cost of reconfigurable-processor-based solutions will be dominated by the hardware cost (as opposed to design costs) due to the flexibility of the system. The market tendency for product differentiation will place a greater importance on lowering the hardware cost for FPGA-based systems. Given a reconfigurable processor with a fixed RFU logic capacity, a set of custom instructions that can lead to area–time-efficient realizations must be determined rapidly in order to meet the tight time-to-market (TTM) pressures.

This paper is an extension of the work presented in [9], which rapidly estimates the area–time measures of custom instructions on FPGA to facilitate the selection of a reduced set of custom instructions that can lead to good area–time solutions. In this paper, we have further reinforced the benefits of the framework by employing a larger application set in the experiments. In particular, we have used fifteen applications from widely used MiBench [10] and MediaBench [11] benchmark suites. In addition, while the area costs presented in [9] are based on estimated values, we provide actual implementation results of the selected custom instructions in this paper. This eliminates any ambiguity in the results that may arise due to estimation errors. Based on the application sets considered, we confirm that the proposed approach can achieve significant area–time gain over the existing high-level area-optimization strategy presented in [12]. Finally, in this paper, we also compare the proposed approach with the case where the area-optimization strategy operates on the same reduced set of custom instructions. This study was not reported in [9]. Contrary to expectations, the experimental results show that the proposed approach (which does not incorporate advanced area-optimization techniques) can still lead to significant area–time gains.

Manuscript received October 27, 2008; revised December 30, 2008. First published March 16, 2009; current version published September 16, 2009.

S.-K. Lam and T. Srikanthan are with the Centre for High Performance Embedded Systems, Nanyang Technological University, Singapore 637553 (e-mail: assklam@ntu.edu.sg; astsrikan@ntu.edu.sg).

C. T. Clarke is with the Department of Electronic and Electrical Engineering, University of Bath, BA2 7AY Bath, U.K. (e-mail: c.t.clarke@bath.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2009.2017091

The remainder of this paper is organized as follows. In the following section, we discuss existing approaches for custom instruction selection and a commonly used strategy for high-level area optimization. Section II provides an overview of the proposed framework, and Section III discusses the essential steps in the framework. Experimental results are provided in Section IV to demonstrate the benefits of the proposed technique for selecting area–time-efficient custom instructions. This paper concludes in Section V.

A. Related Work

Instruction set customization is defined as a process to automatically generate custom instructions from an application in order to meet certain design objectives. An existing work in instruction set customization generally consists of two steps: 1) custom instruction identification and 2) custom instruction selection. *Custom instruction identification* can be loosely described as a process of detecting a subgraph from the application data-flow graph (DFG) to form a single custom instruction in order to maximize some metric (typically performance). This step generates a set of custom instruction candidates or templates, which will be evaluated for custom instruction implementation. *Custom instruction selection* evaluates the templates in terms of their performance, area, or power and selects a subset of them that meets the design constraints. In this paper, we focus on custom instruction selection.

Heuristics are often used for custom instruction selection due to the complexity of the problem. In [13], a covering algorithm was presented to select a minimal set of templates (custom instruction candidates) that maximizes the number of covered nodes. The authors analyzed the tradeoff between the number of templates and the percentage of node coverage. It was observed that increasing the number of templates in the covering algorithm will lead to a notable increase in the number of covered nodes only up to a certain point. This implies that selecting a larger number of custom instruction candidates may not necessarily lead to a better performance gain. Although this is an interesting observation, the work in [13], however, has not studied the effect on the hardware area–time when varying number of templates is selected.

Rapid design exploration must be undertaken to facilitate effective custom instruction selection without delaying the short TTM requirements for embedded systems. This can be achieved with the presence of a fast and accurate method to estimate the performance-cost mapping of custom instructions on hardware. Previously reported design flows for instruction set customization do not incorporate an effective technique for area–time estimation that takes into account the architectural constraints of commercial FPGAs. For example, the estimation process in [14]–[16] is achieved by summing up the area–time values of the custom instruction operations that are obtained from standard-cell libraries. The work in [17] uses a similar strategy to approximate the throughput of each instruction on FPGA. These strategies do not lend themselves well toward FPGA estimations as they do not take into consideration FPGA optimization strategies that maximize the resource utilization of the programmable logic structures. Approaches presented in [18] and [19] incorporate a hardware synthesis flow to select custom instructions of a given application in hardware, which hampers the efficiency of design exploration.

High-level area–time estimation is an essential step to facilitate rapid design exploration for FPGA implementations. High-level estimation is directly performed on the behavioral/algorithmic representations of an application. It is worth mentioning that high-level estimation techniques differ from existing technology mapping approaches (see, e.g., [20]) as the latter relies on the availability of gate-level representations of the applications. For example, the high-level estimation technique presented in [21] is based on a set of formulas that models the components and corresponding FPGA hardware area of the DFG operations. The approach reported credible results with a maximum error of 10%, and the estimation can be achieved on the order of milliseconds.

Area minimization can be performed at various levels of design abstractions. In contrast to gate-level synthesis which operates on the available logic gates of a target architecture library, high-level synthesis operates on the primitive operations that are derived from the behavioral/algorithmic representations. Designs at the higher level of abstractions are less confined to the physical architecture, and hence, optimizations at this level usually lead to high-quality results. Existing high-level area-minimization approaches often aim to maximize resource sharing of the data paths. Resource-sharing-based strategies combine highly regular custom instruction data paths into a single merged data path that can implement the original data paths in a time-multiplexed manner. The work in [12] finds a maximal unique set of data-path structures that can cover all the selected custom instructions. This is achieved by combining the larger custom instruction subgraphs with smaller subgraphs that are subsumed by it. Similar concepts have been employed in [14] to maximize the area utilization of custom instructions.

B. Main Contribution

In this paper, we present a framework that enables rapid selection of custom instructions for reconfigurable processors in an architecture-aware manner. A clustering strategy is used to estimate the FPGA area–time implementation of custom instructions from the intermediate representation (IR) of ANSI C applications, without the need for lengthy hardware synthesis. The proposed clustering strategy maps the custom instructions onto the RFU to maximize the utilization of the reconfigurable resources. The proposed high-level estimation strategy targets reconfigurable structures that are similar to commercially available technologies (i.e., Xilinx devices) and hence can be readily integrated with existing hardware synthesis tools. This enables rapid selection of a reduced set of custom instructions that leads to high area efficiency without compromising heavily on the performance gain. Based on the application set considered, we show that the proposed approach can achieve significant area–time gain over an existing high-level area-optimization strategy [12].

II. Overview of Proposed Framework

Fig. 1 shows an overview of the proposed framework. We have relied upon the Trimaran compiler infrastructure [22] to generate the IR of the applications in the form of DFGs.

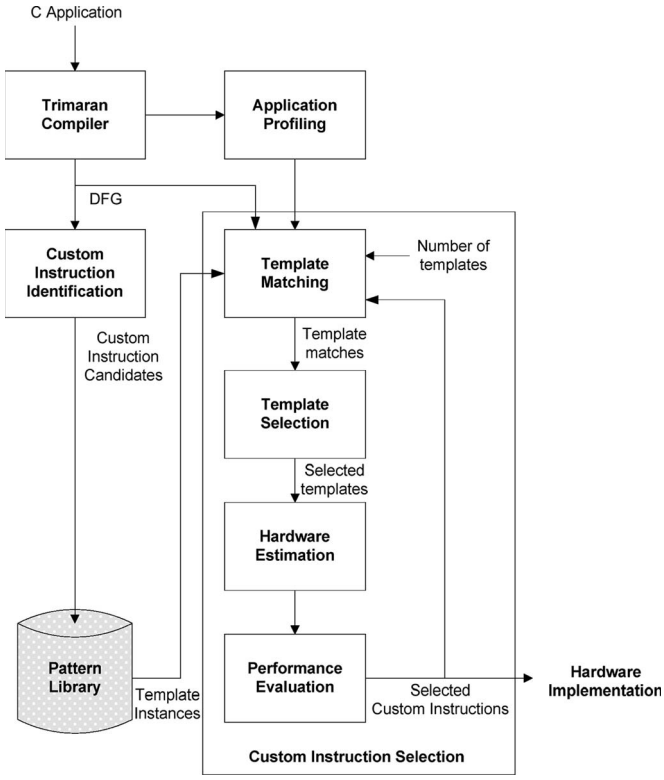


Fig. 1. Overview of the proposed framework.

In the custom instruction identification stage, a template enumeration method is combined with graph isomorphism to identify unique template instances from the Trimaran IR. These template instances form a set of potential custom instruction candidates to be mapped on the RFU and must satisfy a set of architectural constraints. Details of the custom instruction identification stage can be found in [12] and will not be elaborated here. Note that custom instruction identification and application profiling are performed only once for each application. The template instances are stored in the pattern library. The custom instruction selection stage iteratively selects custom instructions from the template instances based on a heuristic that aims to maximize the gain of the custom instructions.

III. Custom Instruction Selection

We will briefly describe the following three main steps in the custom instruction selection stage: 1) template matching; 2) template selection; and 3) hardware estimation.

A. Template Matching

A heuristic is used to first identify a set of template instances for template matching, which account for the performance gain and area utilization of the custom instruction in hardware. Each template instance t is assigned a gain as shown in (1), where the speedup obtained by mapping t on hardware is calculated as shown in (2). $T_{\mathrm{SW}}(t)$ denotes the number of clock cycles taken for the template t to run on a processor. $T_{\mathrm{HW}}(t)$ denotes the number of clock cycles of t in hardware, and we estimate this by the length of the critical path in the custom instruction subgraph. $Template\ size(t)$ denotes the size of the template t and is estimated by the number of primitive operations of t

$$Gain(t) = \frac{Speedup(t)}{Template\ size(t)} \tag{1}$$

$$Speedup(t) = \frac{T_{\mathrm{SW}}(t)}{T_{\mathrm{HW}}(t)}. \tag{2}$$

The template instances are then sorted in decreasing gain, and varying range of template instances (each range includes instances with highest gain) is iteratively selected for template matching. The template matching problem can be described as follows: Given an application DFG that is represented as a directed labeled graph $G_d(V, E)$ and a set of template instances, where each template instance is a directed graph $T_i(V, E)$, find every subgraph of G_d that is isomorphic to T_i. This problem is essentially equivalent to the subgraph isomorphism problem. We have used the vflib graph-matching library [23] to identify all the matches in the DFG for the template instances.

B. Template Selection

We have adopted the technique presented in [24] for template selection, which is based on the conflict graph approach. A conflict graph is an undirected graph $G_u = (V, E)$, where each vertex $v \in V$ is a match that is a member of the template set associated with T_i for $1 \leq i \leq n$ and n is the number of template instances with the highest gain as described in (1). We denote the template set associated with T_i as S_i. An edge $e \in E$ between two matches v_x and v_y signifies that the matches have one or more nodes in common.

The algorithm first constructs a conflict graph from the template matches and then iteratively computes the maximum independent set (MIS) of each template set to select the custom instructions. The MIS of template set S_i, denoted as MIS_i, is defined as the largest subset of vertices in S_i that are mutually nonadjacent. The adjacent matches of the MIS within each template set are temporarily removed.

An objective function computed in each iteration that gives preference to the selection of larger matches is employed. The algorithm proceeds to select MIS_i with the largest objective function, and the matches corresponding to the selected MIS are chosen as custom instructions. These matches and their adjacent neighbors are then permanently removed from the conflict graph. The rest of the matches are restored, and the algorithm repeats until the conflict graph is empty.

C. Reduced Template Selection Process

In this section, we describe how the proposed framework in Fig. 1 can be used for selecting a reduced set of custom instructions with the help of an example.

Fig. 2(a) shows a DFG and the corresponding template instances $(P_1, P_2, \ldots, P_6)$ that have been identified in the custom instruction identification stage. Note that only integer operations are considered in template instances. Other architectural constraints that are imposed on the template instances are reported in [12]. For simplicity, we have only shown six enumerated template instances in this example. The template instances are stored in the pattern library [Fig. 2(b)], and their frequency of occurrences based on an input data set is

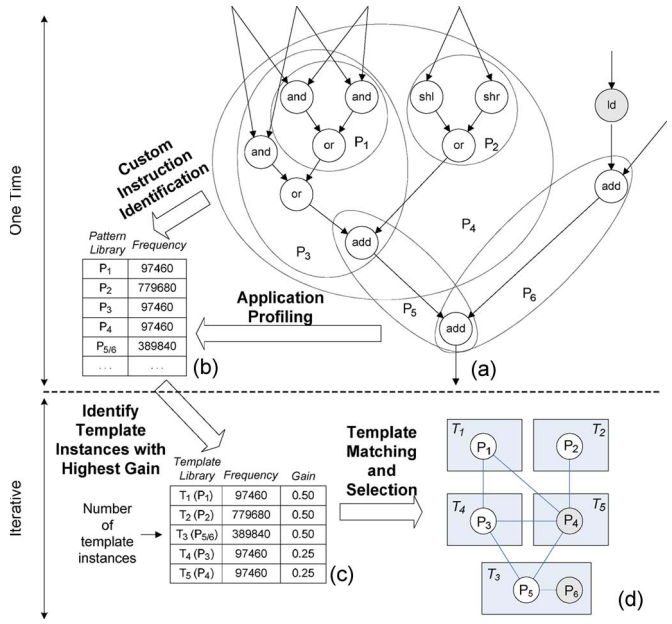


Fig. 2. Example of reduced template selection process.

obtained from application profiling. As shown in Fig. 2, custom instruction identification and application profiling are computed only once for each application. The template instances in the pattern library are then sorted in decreasing gain as described in Section III-A, and varying range of template instances (each range includes the instances with highest gain) is iteratively selected and stored in the template library for template matching and template selection. In the iteration example in Fig. 2(c), five templates $(T_1, T_2, \ldots, T_5)$ that correspond to instances $(P_1, P_2, \ldots, P_6)$ are selected for template matching. Fig. 2(d) shows an example of a conflict graph, where the template instances P_4 and P_6 are selected as custom instructions.

In the first iteration, the full range of template instances is used in template matching and selection, and the performance of the selection is evaluated using the estimated hardware measures. The range of template instances used in template matching and selection is progressively reduced in subsequent iterations. This is repeated until the estimated performance of the selected custom instructions is notably lesser than the previous iteration. When this occurs, the selected custom instructions in the previous iteration will be implemented onto the RFU using commercially available FPGA implementation tools (i.e., Xilinx ISE [25]).

D. Hardware Estimation

This step estimates the area costs and critical path delays of the selected custom instructions when they are realized with the logic elements of the RFU. In this paper, we target programmable logic elements similar to those found in the Xilinx Virtex device [26]. The proposed estimation technique partitions the custom instructions into a set of clusters such that the clusters can be efficiently mapped onto the lookup table (LUT) and carry-look-ahead structure of the FPGA logic elements. The area and critical path delays can then be estimated in terms of the number of clusters, which corresponds to the number of FPGA logic elements. Unlike previously reported hardware estimation technique, the proposed technique incorporates strategies to maximize the resource utilization of the FPGA. The details of the hardware estimation process can be found in [27]. It is noteworthy that the hardware area–time results using the proposed estimation technique have been shown to be within 8% of those obtained using hardware synthesis. In addition, the hardware estimation process can be achieved on the order of milliseconds.

IV. Experimental Results

In this section, we provide experimental results for fifteen applications from the MiBench and MediaBench benchmark suites to demonstrate the area–time benefits of the proposed approach over other commonly used techniques. In particular, we compare the area–time results of the proposed technique, which selects a reduced set of custom instructions [denoted as RT (*Reduced Templates without Data-path Merging*)], with the following approaches.

1) AT (*All Templates Without Data-Path Merging*): Custom instruction selection based on the full range of template instances without data-path merging.
2) AMT (*All Templates With Data-Path Merging*): Custom instructions selected from the full range of template instances are subjected to the area-optimization strategy in [12], which merges custom instructions in order to maximize resource sharing.
3) RMT (*Reduced Templates With Data-Path Merging*): Selected custom instructions using the proposed approach RT are further subjected to the area-optimization method in [12].

The performance estimation of custom instructions obtained using these approaches is reported in terms of software clock-cycle savings (SCS) for application A, which is computed as shown in (3), where t_i's for $i = 1$ to n represent the n custom instructions selected for the application A, $T_{\mathrm{SW}}(t_i)$ denotes the number of operations in custom instructions t_i (we assume that each operation takes one software clock cycle), $F(t_i)$ is the execution frequency of the custom instruction t_i in application A, and $T_{\mathrm{HW}}(t_i)$ is the critical path of t_i in hardware (in terms of the number of clusters). We assume that the execution time of a cluster is equivalent to one software clock cycle. This enables the reconfigurable processor to run using a global clock where multicycle custom instruction implementation [5] is allowed

$$SCS(A) = \sum_{i=1}^{n} \{F(t_i) \cdot (T_{\mathrm{SW}}(t_i) - T_{\mathrm{HW}}(t_i))\}. \quad (3)$$

The custom instructions obtained with the approaches RT, AT, AMT, and RMT have been designed in VHDL, implemented using Xilinx ISE (version 9.1.01i) [25], and targeted to the Virtex-4 FPGA device [26], which incorporate logic elements with four-input LUTs. The smallest device (i.e., xc4vlx40ff1148-10) for the Virtex-4 family that can support the number of required I/O pins has been chosen as the target device in the experiments.

The area–time product is obtained by multiplying the clock-cycle savings with the inverse of the area (in terms of the number of slices).

TABLE I
PERFORMANCE COMPARISON BETWEEN AMT, AT, RMT, AND RT

Application	Performance (Software Clock Savings)				Speedup of AT over AMT (x)	Gain of RT over RMT (%)	Gain of RT over AT (%)
	AMT	AT	RMT	RT			
Adpcm Dec	1587982	2956846	2955476	2955476	1.86	0.00	-0.05
Adpcm Enc	1379816	2748680	1377078	2745942	1.99	99.40	-0.10
Aes	68041755	259846640	127608943	259721264	3.82	103.53	-0.05
Basicmath	6372864	6372864	6372864	6372864	1.00	0.00	0.00
Bitcount	4725000	4725000	4650000	4650000	1.00	0.00	-1.59
Blowfish Dec	4190206	6553954	4190174	6553938	1.56	56.41	0.00
Cjpeg	473146	1022671	277811	930195	2.16	234.83	-9.04
CRC32	106444800	106444800	106444800	106444800	1.00	0.00	0.00
Dijkstra	7649098	7649198	7572200	7572200	1.00	0.00	-1.01
FFT	118797	118797	114688	114688	1.00	0.00	-3.46
Patricia	232869	232869	232869	232869	1.00	0.00	0.00
Pegwit	6208	38249	6208	38249	6.16	516.12	0.00
Rijndael Dec	5009554	9375580	9063509	9063509	1.87	0.00	-3.33
Rijndael Enc	4989733	9355551	9355454	9355455	1.87	0.00	0.00
Sha	2460865	2850705	2845832	2845832	1.16	0.00	-0.17

TABLE II
AREA COMPARISON BETWEEN AMT, AT, RMT, AND RT

Application	Area (Number of Slices)				Area Increase of AT over AMT (%)	Area Reduction of RT over RMT (%)	Area Reduction of RT over AT (%)
	AMT	AT	RMT	RT			
Adpcm Dec	115	141	93	93	22.61	0.00	34.04
Adpcm Enc	158	170	158	171	7.59	-8.23	-0.59
Aes	4613	5918	4647	5884	28.29	-26.62	0.57
Basicmath	16	16	16	16	0.00	0.00	0.00
Bitcount	160	160	96	125	0.00	-30.21	21.88
Blowfish Dec	217	235	221	219	8.29	0.90	6.81
Cjpeg	9326	9378	1057	1086	0.56	-2.74	88.42
CRC32	15	15	15	15	0.00	0.00	0.00
Dijkstra	1315	1635	303	303	24.33	0.00	81.47
FFT	130	130	63	63	0.00	0.00	51.54
Patricia	132	132	48	48	0.00	0.00	63.64
Pegwit	2860	3023	2860	3023	5.70	-5.70	0.00
Rijndael Dec	241	306	227	223	26.97	1.76	27.12
Rijndael Enc	1037	1102	546	557	6.27	-2.01	49.46
Sha	299	285	224	224	-4.68	0.00	21.40

A. *Area–Time Comparison of AMT With AT*

Tables I and II show the performance and area of custom instructions obtained using the AMT and AT approaches. For the AMT approach, area optimization is performed on the selected custom instructions after template matching and selection. It can be observed in column 6 of Table I that custom instructions that have undergone area optimization (i.e., AMT) can suffer from notable performance degradation in a majority of the applications. In particular, the AT approach has an average of $1.9\times$ speedup over the AMT approach. The reason for this is that area optimization introduces multiplexers in the merged custom instruction data paths (see [12]), which leads to an increase in the critical path delay of the custom instructions [i.e., $T_{\mathrm{HW}}(t_i)$ in (3)].

As shown in column 6 of Table II, the area-optimization approach AMT exhibits higher area efficiency than AT for a number of applications (maximum area reduction of over 20% can be achieved). However, the average area reduction achieved by AMT is only about 6.9%. Interestingly, in certain applications (e.g., Sha), the AT approach has slightly better area efficiency than the AMT approach. This can be explained by the fact that the area-optimization strategy in [12] inherently introduces multiplexers in the data paths to facilitate resource sharing. These multiplexers will contribute to additional area requirements for realizing the custom instructions on FPGA. Hence, an effective area reduction can only be achieved if area savings due to resource sharing outweigh the cost that is introduced to facilitate resource sharing.

Fig. 3 compares the area–time product between the various techniques. It can be observed that AT outperforms AMT in terms of area–time product for most of the cases considered. In particular, AT achieves an average area–time product gain of about 73% over AMT.

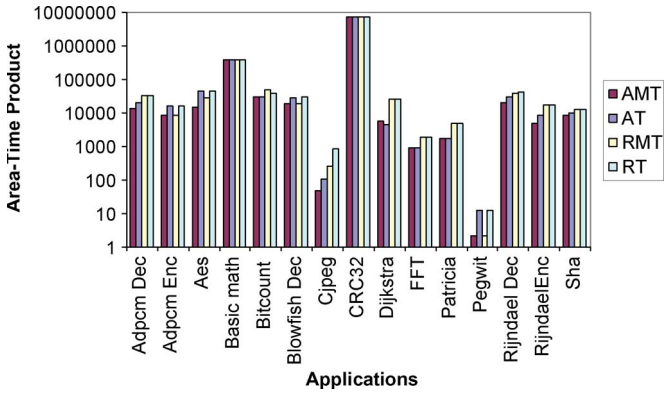


Fig. 3. Area–time product comparison between AMT, AT, RMT, and RT.

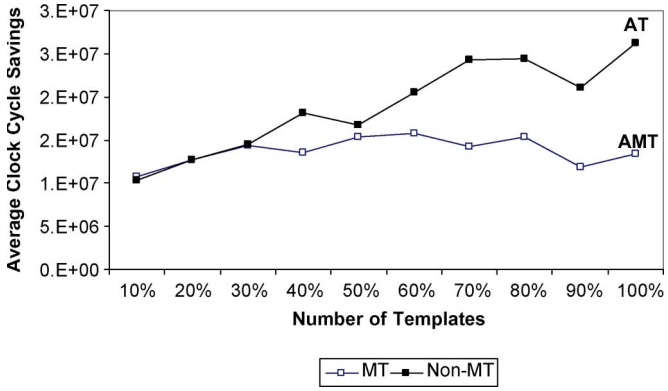


Fig. 4. Performance comparison with existing area-optimization method.

Fig. 4 compares the average performance of custom instructions (for the 15 applications) that have not undergone data-path merging (indicated as nonmerged templates or non-MT) with the average performance of custom instructions that have undergone data-path merging (indicated as merged templates or MT). The experiment reports the average performance for varying number of templates used (from 10% to 100%) during custom instruction selection.

It can be observed that the average performance of non-MT tends to increase (in a nonmonotonic fashion) with increasing number of templates used for custom instruction selection. It is interesting to note that the opposite applies for MT. In particular, the average performance of MT decreases (in a non-monotonic fashion) when more templates are used for custom instruction selection. The increase in the critical path delay due to the introduction of multiplexers in MT becomes more prominent when more template instances are used for custom instruction selection as there are more opportunities to merge the custom instruction data paths for area minimization. Hence, the effective clock-cycle savings of MT decrease with the increase in the number of templates used for custom instruction selection.

B. Selecting Reduced Set of Profitable Custom Instructions

Fig. 5 shows the estimated performance for the 15 applications using the proposed method (RT). It is evident that increasing the number of templates for custom instruction selection will not lead to any notable gain after a certain point (marked as RT in the plots) for almost all the applications (except for CRC32 and Pegwit). These observations imply that it is possible to reduce the number of custom instructions for mapping onto the RFU without compromising heavily on the performance gain.

C. Area–Time Comparison of RMT With RT

In the following experiments, we will investigate whether the proposed method can still lead to higher gains when compared to the case where the same reduced set of templates is subjected to area optimization. We denote the latter approach as RMT, where the selected custom instructions of RT are further subjected to area optimization using the method discussed in [12]. In both cases, the number of template instances used for custom instruction selection is shown in Fig. 5 (i.e., 40% for Adpcm Dec, 70% for Adpcm Enc, 70% for AES, etc.).

Table I shows the performance of the custom instructions obtained using RMT and the proposed RT approach. It can be observed in column 7 of Table I that even though both methods are based on the same set of custom instructions, the proposed method RT can still achieve significant performance gain over RMT for certain applications. In particular, the speedups that are achieved by RT over RMT are about 2, 2, 1.6, 3.4, and 6 times for Adpcm Enc, Aes, Blowfish Dec, Cjpeg, and Pegwit, respectively. The average performance gain of RT over RMT is over 67%. The reason for the performance degradation in RMT is due to the increase in critical path delay as a result of the area-optimization technique which introduces multiplexers in the custom instruction data paths.

While it is expected that the RMT approach will lead to more area efficiency than RT, it is interesting to note from column 7 of Table II that the average area reduction of RMT is only about 3.9%. The reason for this is twofold. First, since only a reduced set of custom instructions is subjected to area optimization, there are less opportunities to merge custom instructions in order to reduce the area utilization. Hence, it can be observed that in most of the applications considered, the areas of RMT and RT are comparable. Second, the area-optimization strategy in [12] inherently introduces multiplexers in the data paths to facilitate resource sharing. These multiplexers will contribute to additional area requirements for realizing the custom instructions on FPGA.

Next, we compare the area–time product between RMT and RT. It can be observed from Fig. 3 that RMT has a higher area–time product than RT in only one application (i.e., Bitcount). On the other hand, RT exhibits significant area–time gains over RMT in a number of applications (e.g., Adpcm Enc, Aes, Blowfish Dec, Cjpeg, and Pegwit). In particular, the average area–time product gain of RT over RMT is over 59%. This set of results demonstrates that the proposed method can still achieve significant area–time gains over the existing area-optimization approach for the same reduced set of custom instructions.

D. Area–Time Comparison of AT With RT

In the previous experiments, we have established that existing area-optimization approaches that commonly rely on data-path merging methods may not lead to area–time-efficient realizations. In particular, we have shown that AT outperforms

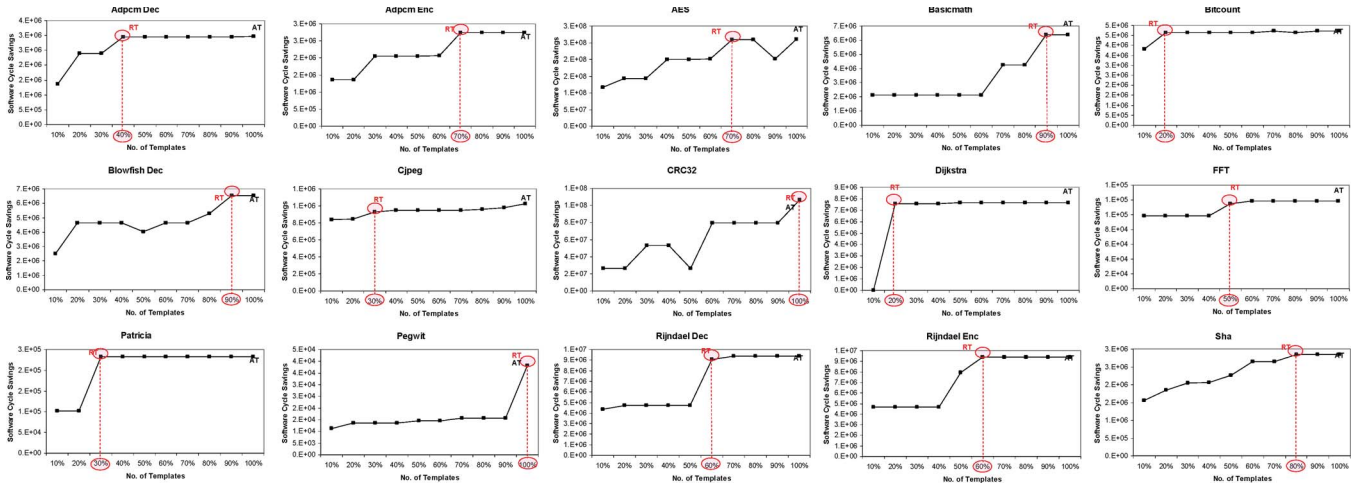


Fig. 5. Performance estimation with varying number of templates.

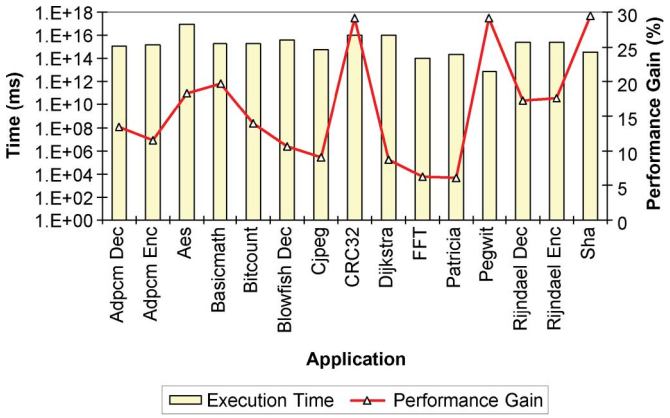


Fig. 6. Performance gain over base processor.

AMT in terms of area–time product by as much as 73%. In addition, we have shown that the proposed method (RT) has an area–time product gain over RMT, which employs the same reduced set of custom instructions, of over 59%. In this section, we will compare the area–time results of the proposed technique (RT) with AT.

Tables I and II show the performance and area of the custom instructions obtained using the AT and the proposed RT approach. The number of template instances used in each application for RT is shown in Fig. 5. It can be observed from column 8 of Table I that the average performance loss of RT is only less than 1.3% compared to AT. When compared to the case when all template instances are used for custom instruction selection (i.e., AT), it is evident from column 8 of Table II that the proposed method (i.e., RT) can lead to significant area reduction (i.e., average of 29% area reduction). This is due to the fact that RT leads to the selection of significantly lesser number of custom instructions in most of the applications when compared to AT.

Finally, it can be observed from Fig. 3 that when compared to the case when all template instances are used for custom instruction selection (AT), the proposed method (RT) has higher area–time product gain for most of the applications (and comparable with the remaining ones). In particular, the average area–time product gain of the proposed approach is about 2.1 times that of AT.

E. Performance Gain Over Base Processor

Fig. 6 shows the execution time and percentage performance gain of a custom processor with custom instructions obtained using the RT approach over the base processor implementation. We assumed that the base processor is a soft-core processor with the area-optimized configuration in [28]. The performance gain is obtained by computing the percentage execution time savings of the custom processor over the total execution time of the base processor (calculated based on the profiling results of [22]). We assumed that each operation in the base processor utilizes one clock cycle. The execution time savings of the custom processor are computed using (3), where $T_{\mathrm{SW}}(t_i)$ and $T_{\mathrm{HW}}(t_i)$ are substituted with the software latency of the custom instructions t_i and the hardware delay of t_i (measured in the FPGA tool [25]), respectively. It can be observed from Fig. 6 that the custom processor can achieve a notable average gain of over 16% and a maximum gain of about 30%. It is noteworthy that this performance gain is achieved in an area-efficient manner using the proposed strategy.

V. Conclusion

A design exploration framework for reconfigurable processors has been proposed for the rapid selection of a reduced set of profitable custom instructions. The framework incorporates a clustering strategy to facilitate rapid area–time estimation of custom instruction implementations on FPGA. Unlike existing estimation strategies that do not incorporate architecture-aware strategies, the proposed estimation method takes into consideration FPGA optimization strategies to maximize the resource utilization of the programmable logic structures. The proposed technique leads to rapid selection of custom instructions as it does not require hardware synthesis. We employed LUT-based FPGAs such as that found in Xilinx devices only to facilitate comparisons, and as such, we do not attempt to overcome the inherent architectural deficiencies. Experimental results show that the number of candidates for custom instruction selection can be significantly reduced with marginal degradation in resulting performance gain. Our investigation also reveals that the proposed method leads to higher area–time gains than that is possible with existing area-optimization approaches that suffer from undesirable critical path delay in the resulting data paths

due to resource sharing. Finally, the notable savings in area can be readily traded to increase performance or to reduce power consumption.

REFERENCES

[1] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP J. Embedded Syst.*, vol. 2006, no. 1, pp. 1–19, Jan. 2006.

[2] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of FPGAs," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1810–1823, Aug. 2007.

[3] E. Monmasson and M. N. Cirstea, "FPGA design methodology for industrial control systems—A review," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1824–1842, Aug. 2007.

[4] W. Marx and V. Aggarwal, "FPGAs are everywhere—In design, test & control," *NI Developer Zone*, Jun. 2008. [Online]. Available: http://zone.ni.com/-devzone/cda/pub/p/id/401

[5] *Altera: NIOS II Processors*. [Online]. Available: http://www.altera.com/products/ip/processors/-nios2/ni2-index.html

[6] *Xilinx Platform FPGAs*. [Online]. Available: http://www.xilinx.com

[7] J. M. Arnold, "S5: The architecture and development flow of a software configurable processor," in *Proc. IEEE Int. Conf. Field-Programmable Technol.*, Dec. 2005, pp. 121–128.

[8] F. Barat, R. Lauwereins, and G. Deconinck, "Reconfigurable instruction set processors from a hardware/software perspective," *IEEE Trans. Softw. Eng.*, vol. 28, no. 9, pp. 847–862, Sep. 2002.

[9] S. K. Lam and T. Srikanthan, "Selection of area-time efficient custom instructions for FPGA realization," in *Proc. IEEE Int. Symp. Ind. Electron.*, Jun./Jul. 2008, pp. 1704–1709.

[10] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE Int. Workshop Workload Characterization*, Dec. 2001, pp. 3–14.

[11] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. 13th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 1997, pp. 330–335.

[12] S. K. Lam, T. Srikanthan, and C. T. Clarke, "Rapid generation of custom instructions using predefined dataflow structures," *Microprocess. Microsyst.—Special Issue FPGA-Based Reconfigurable Computing*, vol. 30, no. 6, pp. 355–366, Sep. 2006.

[13] R. Kastner, A. Kaplan, S. O. Memik, and E. Bozorgzadeh, "Instruction generation for hybrid reconfigurable systems," *ACM Trans. Des. Autom. Embedded Syst.*, vol. 7, no. 4, pp. 605–627, Oct. 2002.

[14] N. T. Clark, H. Zhong, and S. A. Mahlke, "Automated custom instruction generation for domain-specific processor acceleration," *IEEE Trans. Comput.*, vol. 54, no. 10, pp. 1258–1270, Oct. 2005.

[15] L. Pozzi, K. Atasu, and P. Ienne, "Exact and approximate algorithms for the extension of embedded processor instruction sets," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 7, pp. 1209–1229, Jul. 2006.

[16] P. Yu and T. Mitra, "Characterizing embedded applications for instruction-set extensible processors," in *Proc. 41st IEEE/ACM Des. Autom. Conf.*, Jun. 2004, pp. 723–728.

[17] J. Cong, Y. Fan, G. Han, and Z. Zhang, "Application-specific instruction generation for configurable processor architectures," in *Proc. ACM/SIGDA 12th Int. Symp. Field Programmable Gate Arrays*, Feb. 2004, pp. 183–189.

[18] B. Kastrup, A. Bink, and J. Hoogerbrugge, "ConCISe: A compiler-driven CPLD-based instruction set accelerator," in *Proc. 7th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach.*, Apr. 1999, pp. 92–101.

[19] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha, "Custom-instruction synthesis for extensible-processor platforms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 2, pp. 216–228, Feb. 2004.

[20] D. Chen and J. Cong, "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, Nov. 2004, pp. 752–759.

[21] D. Kulkarni, W. A. Najjar, R. Rinker, and F. J. Kurdahi, "Compile-time area estimation for LUT-based FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 1, pp. 104–122, Jan. 2006.

[22] *Trimaran: An Infrastructure for Research in Instruction-Level Parallelism*. [Online]. Available: http://www.trimaran.org

[23] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "Performance evaluation of the VF graph matching algorithm," in *Proc. Int. Conf. Image Anal. Process.*, Sep. 1999, pp. 1172–1177.

[24] Y. Guo, G. J. M. Smit, H. Broersma, and P. M. Heysters, "A graph covering algorithm for a coarse grain reconfigurable system," in *Proc. ACM SIGPLAN Conf. Language, Compiler, Tool Embedded Syst.*, Jun. 2003, pp. 199–208.

[25] *Xilinx ISE Foundation*. [Online]. Available: http://www.xilinx.com/ise/logic_design_prod/foundation.htm

[26] *Virtex-4 FPGA User Guide*, Jun. 17, 2008, San Jose, CA: Xilinx. UG070, Ver. v2.5.

[27] S. K. Lam and T. Srikanthan, "Estimating area costs of custom instructions for FPGA-based reconfigurable processors," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Architectures Processors*, Jul. 2007, pp. 89–94.

[28] D. Mattson and M. Christensson, "Evaluation of synthesizable CPU cores," M.S. thesis, Chalmers Univ. Technol., Gothenburg, Sweden, 2004.

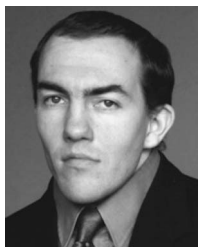
Siew-Kei Lam (M'03) received the B.A.Sc. (Hons.) degree and the M.Eng. degree in computer engineering from Nanyang Technological University (NTU), Singapore.

Since 1994, has been with NTU, where he is currently a Research Associate with the Centre for High Performance Embedded Systems and has worked on a number of challenging projects that involved the porting of complex algorithms in VLSI. He is also familiar with rapid prototyping and application-specific integrated-circuit design flow methodologies. His research interests include embedded system design algorithms and methodologies, algorithms-to-architectural translations, and high-speed arithmetic units.

Thambipillai Srikanthan (SM'92) received the B.Sc. (Hons.) degree in computer and control systems and the Ph.D. degree in system modeling and information systems engineering from Coventry University, Coventry, U.K.

Since 1991, he has been with Nanyang Technological University, Singapore, where he is currently a Full Professor, the Director of a 100 strong Centre for High Performance Embedded Systems (CHiPES), and the Director of the Intelligent Devices and Systems cluster. He founded CHiPES in 1998 and elevated it to a university-level research center in February 2000. He has several years of university experience. His research interests include design methodologies for complex embedded systems, architectural translations of compute-intensive algorithms, and computer arithmetic and high-speed techniques for image processing and dynamic routing. He has published more than 250 technical papers, including 60 journals in IEEE TRANSACTIONS, *IEE Proceedings*, and other reputed international journals. His services as a Key Consultant to embedded systems industry, both locally and internationally, are continually being sought for.

Dr. Srikanthan was the recipient of the Public Administration Medal (Bronze) at the 2006 National Day in recognition of his contributions to education in Singapore.



Christopher T. Clarke received the B.Eng. degree in engineering electronics and the Ph.D. degree in computer science from the University of Warwick, Coventry, U.K., in 1989 and 1994, respectively.

From 1994 to 1997, he was a Lecturer with Nanyang Technological University, Singapore, where he was the Cofounder of the Centre for High Performance Embedded Systems. Since then, he has spent time in industry, both as an in-house Engineering Manager and independent Consultant for U.K. silicon design houses, system integrators, and multinationals such as Philips Semiconductors. Since March 2003, he has been with the Microelectronics and Optoelectronics Research Group, Department of Electronic and Electrical Engineering, University of Bath, Bath, U.K. He has been involved with many European union funded research projects including PEPS, CIRCE, SENS, and IMANE.

Dr. Clarke is a member of the Centre for Advanced Sensor Technologies.