An Integrated Temporal Partitioning and Mapping Framework for Handling Custom Instructions on a Reconfigurable Functional Unit

Mehdipour, Farhad
Computer and IT Engineering Department, Amirkabir University of Technology

Noori, Hamid
Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University

Saheb Zamani, Morteza
Computer and IT Engineering Department, Amirkabir University of Technology

Murakami, Kazuaki
Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University

他

出版情報：Proc. of 11th Asia-Pacific Conference (ACSAC 2006), pp.219-230, 2006-09. Springer
バージョン：
権利関係：

An Integrated Temporal Partitioning and Mapping Framework for Handling Custom Instructions on a Reconfigurable Functional Unit

Farhad Mehdipour[1], Hamid Noori[2], Morteza Saheb Zamani[1], Kazuaki Murakami[2], Mehdi Sedighi[1], and Koji Inoue[2]

[1] Computer and IT Engineering Department, Amirkabir University of Technology, Tehran, Iran
{mehdipur, szamani, msedighi}@ce.aut.ac.ir
[2] Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan
noori@c.csce.kyushu-u.ac.jp,
{murakami, inoue}@i.kyushu-u.ac.jp

Abstract. Extensible processors allow customization for an application by extending the core instruction set architecture. Extracting appropriate custom instructions is an important phase for implementing an application on an extensible processor with a reconfigurable functional unit. Custom instructions (CIs) usually are extracted from critical portions of applications. This paper presents approaches for CI generation with respect to the RFU constraints to improve speedup of the extensible processor. First, our proposed RFU architecture for an adaptive dynamic extensible processor called AMBER is described. Then, an integrated temporal partitioning and mapping framework is presented to partition and map the CIs on the RFU. In this framework, a mapping aware temporal partitioning algorithm is used to generate CIs which are mappable on the RFU. Temporal partitioning iterates and modifies partitions incrementally to generate CIs. In addition, a mapping algorithm is presented which supports CIs with critical path length more than the RFU depth.

1 Introduction

Synthesis of application-specific instruction-set processors (ASIPs) has been an important design methodology for system-on-chip processors in the last decade. ASIPs have more potential to meet the high-performance demands of embedded applications, compared to general purpose processors (GPPs) but the synthesis of ASIPs traditionally involved the generation of a complete instruction set architecture for the targeted application. On the other hand, GPPs are very flexible but may not offer the necessary performance.

Another method for providing enhanced performance is application-specific instruction set extension. An important feature of this design method is extending an existing processor core with units specialized for a given domain, rather than designing a custom processor completely. By creating application-specific extensions to an instruction set, the critical portions of an application's dataflow graph (DFG) can be accelerated by using custom functional units. The nodes of these DFGs are the instructions of critical potion of applications and the edges of DFGs represent the

C. Jesshope and C. Egan (Eds.): ACSAC 2006, LNCS 4186, pp. 219 – 230, 2006.

dependency between instructions. In our method, custom instruction is a sequence of instructions that are extracted from hot basic blocks (HBBs). HBBs are basic blocks which are executed more than a predefined number of times and a basic block is a sequence of instructions that terminates by a control instruction.

Using an extensible processor with a reconfigurable functional unit proposes favorable tradeoff between efficiency and flexibility, while keeping design turnaround times much shorter. The reconfigurable part of an extensible processor executes critical portions of an application to gain better performance. It can be coarse grain or fine grain. The former, demands for less configuration memory. Also mapping of instructions on it is easier. The latter is more flexible but it is slower comparing with the coarse grain one.

Extracting CIs from applications is an important stage in accelerating application execution. Some generated CIs cannot be mapped to reconfigurable hardware because some RFU constraints, like physical constraints, cannot be considered at this stage. We call this kind of CIs *rejected CIs*. Two different strategies are used for rejected CIs. In the first case, rejected CIs are run on the base line processor, and so, this offers no speedup. As the second strategy, we suggest using approaches to recover and execute rejected CIs on the RFU rather than the base processor. To achieve this goal, two approaches are proposed. In the first approach, a CI generation tool is used to regenerate the CIs from HBBs according to the RFU constraints. As another approach, we propose a novel framework for generating CIs. This framework generates CIs in such a way that they can be executed on the RFU. Besides, it partitions rejected CIs to multiple mappable CIs. We utilize the same well-known *temporal partitioning* concept for this purpose.

In Section 2, we highlight some related work. The RFU architecture is described in Section 3. Section 4 discusses the design flow proposed for generating CIs. In Section 5, experimental results are presented and finally, Section 6 concludes the paper.

2 Related Works

Identifying optimal set of custom instruction to improve the computational efficiency of applications has received a lot of attention recently. PRISC [13] and Chimaera [17] provide compilation tools that attempt to automatically generate mappings for the reconfigurable logic. Custom instructions tend to be relatively small, due in part to the difficulty of the matching problem and the size of the programmable fabric available. DISC [16] is another system that requires CIs to be identified and programmed manually. The main focus of DISC is in the management of the loading of custom instructions.

Research in reconfigurable computing is often more in line with our goal. Researches in reconfigurable computing investigate the identification of application sections that are mapped to a reconfigurable fabric. Most of CI extraction methods attempt to identify patterns within a basic block. In [7] the authors combine template matching and generation based on the occurrence of patterns which usually led to small templates. Template matching is done based on graph isomorphism. Methods presented in [5], [8] impose further constraints by allowing multiple input-single output patterns. Arnold et al. [1] avoids the exponentially increasing of these patterns

by using an iterative technique that detects 2-operator patterns, replace their occurrences in DFG and repeats the process. Atasu et al. [2] search a full binary tree and decides at each step whether or not to include a particular instruction in a pattern. The potential exponential search space is pruned based on input/output constraints. They attempt to find maximal subgraphs of application data flow graph, but it does not take into account the underlying structure of the execution hardware. Clark et al. [4] search possibly good patterns by starting with small patterns and expanding them considering the input, output and convexity constraints [18].

The general goal of this work is presenting methods for CI generation, specifically for recovering the rejected CIs. We propose approaches for generating CIs for AMBER, an adaptive dynamic extensible processor presented in [11]. AMBER uses a coarse grain reconfigurable functional unit with fixed resources. Some of the generated CIs might be rejected because of violating RFU constraints. Rejection of CIs decreases the speedup. We do not use any pruning algorithm for making smaller CIs from rejected CIs because obviously by using bigger CIs more speedup can be obtained. Our main contribution is in using an RFU architecture-aware temporal partitioning algorithm, which iteratively attempts to partition and generate appropriate CIs. These CIs are maximal subgraphs extracted from data flow graph of non-mappable CI.

For this purpose, we use an integrated temporal partitioning and mapping framework. The idea behind temporal partitioning is that functions that are too large to fit on a programmable hardware can be partitioned into several modules which are then successively downloaded into the hardware in accordance with a predefined schedule [6]. Different algorithms have been presented for temporal partitioning. Bobda [3] proposed two methods to solve temporal partitioning problem. The first one was an enhancement of the well-known list vector space. The second method uses a spectral placement to position the modules in a three-dimensional vector space. Karthikeya et al. [6] proposed algorithms for temporal partitioning and scheduling of large designs on area constrained reconfigurable hardware. *SPARCS* [12] is an integrated partitioning and synthesis framework, which has a temporal partitioning tool to temporally divide and schedule the DFGs on a reconfigurable system. Tanougust et al. [15] attempted to find the minimum area while meeting timing constraints during temporal partitioning. In [14], Spillane and Owen focused on finding a sequence of conditions for activating an appropriate component at a particular time and optimizing successive configurations to achieve the desired trade-offs among reconfiguration time, operation speed and area.

In [9], a new design flow was proposed for the compilation of data flow graphs for a reconfigurable system. This design flow consists of temporal partitioning and physical design phases with a feedback loop. In this paper, we propose a modified version of this design flow for generating appropriate CIs as a general methodology and use is specifically for AMBER RFU. This framework attempts to take advantages of the basic design flow to generate CIs and improve target extensible processor speedup.

3 RFU Architecture

In [11] an adaptive extensible processor (AMBER) was presented which has the capability of tuning its extended instructions to the running application. For this

extensible processor, a coarse grain reconfigurable functional unit (RFU) was designed which is an array of functional units (FUs). FUs support all fixed point instructions of the base line processor except multiplication, division and load. A quantitative approach [4] was used to determine the number of inputs, outputs, nodes, routing resources and other architectural specifications. Twenty-two applications of Mibench [19] were used to provide quantitative analysis. Also, a mapping tool was developed to map CIs on the RFU. The details of RFU design and its integration with the base processor is out of the scope of this paper, therefore, for completeness we only describe the specification of the final architecture.

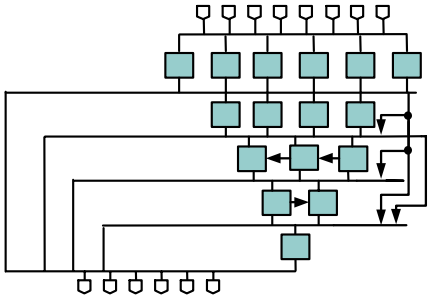

Fig. 1. Block diagram of RFU designed for AMBER.

According to the obtained results, eight inputs, six outputs and 16 FUs brought about a reasonable CI rejection rate (about 10%). Rejection rate represents the percentage of CIs that can not be mapped on the RFU according to its defined constraints. In addition, a proper topology for RFU connections was achieved based on the quantitative analysis (Fig. 1). In the proposed architecture, there are left to right connections in the 4th row and right to left connections in the 3rd row. Outputs of FUs in each row are fully connected to inputs of FUs in subsequent row. In addition, there are extra vertical connections, as in Fig. 1, between non-subsequent rows to keep the CI rejection rate low.

4 Integrated Temporal Partitioning and Mapping

Initial CIs for AMBER can be extracted from hot basic blocks of applications according to the algorithm presented in [12]. Two different approaches for generating appropriate CIs are used. Appropriate CI set means the set of CIs which satisfy the RFU primary constraints and may have the capability of being mapped successfully on the RFU. RFU *primary constraints* are the architectural constraints including the number of inputs, outputs and nodes. We used two different approaches for generating CIs. The first CI generation approach (*CIGen*) considers RFU *primary constraints* for mapping but it cannot consider all of the constraints such as routing resources constraints. For considering the physical constraints during CI generation physical design process need to be done. Therefore, for rejected CIs, *CIGen* follows a conservative method to generate appropriate CIs.

4.1 The *Integrated Framework*

Integrated Framework is the second CI generation approach that performs an integrated temporal partitioning and mapping process to generate mappable CIs. The proposed design flow is shown in Fig. 2. This design flow takes rejected CIs and attempts to partition them to appropriate CIs those have the capability of mapping on the RFU. Each CI is partitioned into two or more CIs.

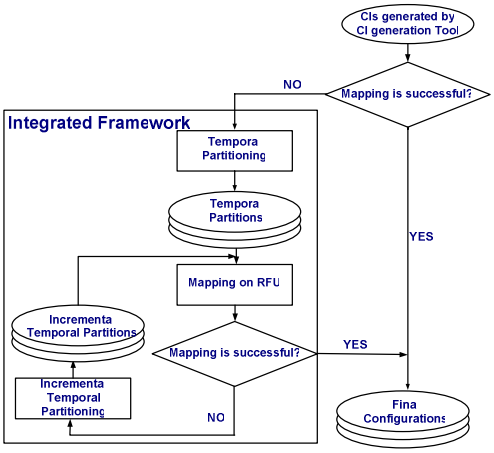

Fig. 2. Integrated temporal partitioning and mapping for supporting large CIs.

Initial temporal partitioning algorithm is done according to [9]. In this stage, RFU *primary constraints* are considered. The generated CIs are accepted and finalized if they can be mapped on the RFU. For each partition generated in the previous step, the mapping process is done and the generated CI is considered as appropriate if it can be mapped on the RFU successfully. Otherwise, an incremental temporal partitioning algorithm modifies the partition by moving some of nodes to the subsequent partition. In the next step, the mapping process is repeated. This process is done iteratively while all partitions are mapped successfully on the RFU. Fig. 3 shows an example of a rejected CI which is finally partitioned into two partitions and mapped on the RFU successfully. This framework has the following advantages:

- Reducing the number of rejected CIs: This can affect the overall performance by partitioning the rejected CIs to CIs which can be mapped on the RFU.
- Using a mapping-aware temporal partitioning process: This process attempts to prevent the rejection of CIs by modifying CIs according to the feedbacks obtained from the mapping process. In fact, only *primary constraints* of the RFU can be considered in the *CIGen* but it is unaware of such mapping information as routing resource constraints. In *Integrated Framework*, CIs are partitioned in such a way that they can be mapped on the RFU.

4.2 Incremental Temporal Partitioning Algorithm

In *Integrated Framework*, an incremental temporal partitioning process is performed iteratively until all partitions are mapped on the RFU successfully. Each partition which does not satisfy RFU constraints is modified by selecting and moving proper nodes to the subsequent partition and then a new iteration starts. An incremental temporal partitioning algorithm tries to modify partitions during the iteration process. This algorithm chooses the nodes with highest *ASAP* level first. The *ASAP* level of nodes represents their order to execute according to their dependencies [10]. In other words, a parent node should be executed before its descents because of data dependencies between them.

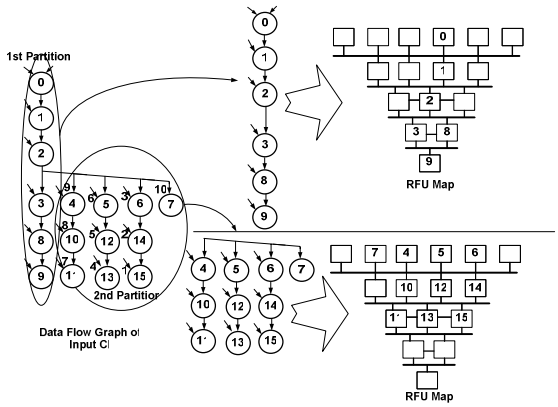

Fig. 3. An example of CI generation using the *Integrated Framework*.

All nodes in a partition are sorted according to their *ASAP* level and the node with the highest *ASAP* level is selected and moved to the subsequent partition. In Fig. 3, the order in which are selected and moved to the next partition is 15, 13, 11, 9, 14, 12, 10, 8, 3 and 7. The nodes are moved until all the generated partitions satisfy the RFU architectural constraints.

4.3 Mapping Procedure

Mapping process in the *Integrated Framework* is the same as the well-known placement problem. Mapping process can be defined as the placement of the DFG nodes on a fixed architecture RFU, to determine the appropriate positions for DFG nodes on the RFU. Assigning CI instructions or DFG nodes to FUs is done based on the priority of the nodes.

We calculated *slack* of nodes [10] to determine their priority for partitioning. Slack of each node represents its criticality. For example, slack equal to *0* means that it is on the critical path of DFG and should be scheduled with the highest priority. On the other hand, for the nodes with the same criticality, *ASAP* level of them determines their mapping order. Therefore, in the first step, *ASAP, ALAP*[1] and *slack* values of

[1] As Late As Possible.

each node in DFG are determined [9, 10]. Assigning a position for each selected node starts by determining an appropriate row for that node. Row number is set to the last row if the selected node is on a critical path with the length more than or equal to RFU depth. Otherwise, row number is selected according to *slack* and *ALAP* of the selected node and the number of un-occupied cells available in the RFU rows.

For the nodes which do not belong to any critical path with length more than the RFU depth, their starting row is set to *ALAP- slack -1*. This means that we reserve FUs of lower rows for the nodes belong to critical path. For this purpose, we prevent the occupation of FUs in the lower RFU rows by the nodes which do not belong to critical paths. Therefore, spiral shaped mapping of nodes is being possible for long length critical paths. After determining the row number, an appropriate column is determined for the selected node. Column number is determined according to the minimum connection length criterion. All non-occupied cells of the RFU in the determined row are checked to find an FU which gives the minimum connection length between the selected node and its dependent nodes positioned on the RFU.

For each row, a maximum capacity is considered to prohibit gathering many nodes in a row. Capacity of rows is determined with respect to longest critical path and the number of critical paths in the DFG. Row number is decreased and a new attempt starts if there is not any cell to assign the selected node. The pseudo code of the mapping algorithm is as follows:

Mapping Algorithm:

```
- Determine ASAP level of each node in the input DFG,
- Determine ALAP level of each node in the DFG,
- Calculate slack for each node in the DFG.
for s= 0 to Maximum slack value
   - Create List of Nodes with slack equal to s
     for all nodes in the list
   - Determine appropriate position for the selected
   node from the list
   - if the number of nodes mapped on the RFU is equal
   to the DFG node number then mapping process is
   terminated successfully
```

Determine appropriate position for a selected node:

```
if ALAP- slack >= RFUDepth
 StartRow= RFUDepth;
else
   StartRow= ALAP- slack - 1;
for Row= StartRow to 0
   -if there is un-occupied column in the selected row
   and the selected row has sufficient capacity, select
   a column with minimum connection length.
```

Referring to the RFU architecture in Fig. 1 and its routing resources, though the RFU depth is equal to 5, our mapping algorithm can map CIs whose critical path length are at most equal to 8. In Fig. 3, corresponding DFG of the first partition has a critical path longer than the RFU depth, and so it takes advantage of a spiral shaped mapping. This kind of mapping results in effective usage of routing resources (horizontal connections of the third and forth rows) and FUs.

5 Experimental Results

SimpleScalar tool set (PISA configuration)[20] and 22 applications of Mibench [19] were used for doing experiments. The base line processor of AMBER was MIPS324K with five stage pipeline, 32KB L1 data cache (1 clock cycle latency); 32KB L1 instruction cache (1 clock cycle latency) and 1MB unified L2 cache (6 clock cycle latency). RFU was implemented using Synopsys tools with Hitachi 0.18μm library. The RFU area size is 1.15$mm^{2.}$. It was assumed that the RFU has a variable latency based on the length of the longest critical path. Regarding base processor frequency (166MHz) and RFU delay, CIs with critical path length less than or equal to 5 take 1 clock cycle and CIs including critical path length more than 5 take 2 clock cycles for execution on the RFU.

Initial CIs were generated according to the method proposed in [11]. Experiments showed that the CI rejection rate with respect to RFU architectural constraints was about 10%. In 9 of the 22 applications, there was not any rejected CI, which means that

Table 1. Mibench Applications, their CI rejection rates and maximum and minimum length of Cis

App. No.	Application Name	CI Rejection % (Considering Execution Freq)	Min. CI length	Max. CI length	Min. length of Rejected CIs
1	adpcm(enc)	0	5	7	-
2	adpcm(dec)	0	5	7	-
3	**bitcounts**	**2.3**	**4**	**20**	**20**
4	**blowfish**	**43.2**	**5**	**16**	**15**
5	**blowfish (dec)**	**43.2**	**5**	**16**	**15**
6	basicmath	0	3	11	-
7	**cjpeg**	**11.7**	**5**	**59**	**11**
8	crc	0	5	5	-
9	dijkstra	0	4	9	-
10	**djpeg**	**28.8**	**4**	**48**	**8**
11	**fft**	**3.4**	**3**	**16**	**16**
12	**fft (inv)**	**3.4**	**3**	**16**	**16**
13	**gsm (dec)**	**2.8**	**5**	**14**	**14**
14	**gsm (enc)**	**6.5**	**4**	**26**	**13**
15	**lame**	**11.9**	**3**	**13**	**7**
16	patricia	0	3	6	-
17	qsort	0	5	7	-
18	**rijndael (enc)**	**40.6**	**5**	**16**	**10**
19	**rijndael (dec)**	**35.4**	**5**	**18**	**10**
20	**sha**	**1.9**	**5**	**18**	**7**
21	stringsearch	0	5	9	-
22	susan	0	6	10	-

all CIs in these applications were mapped on the RFU successfully. Rejected CIs of remaining 13 applications are as input of our Integrated Framework. Table 1 shows the applications, the percentage of rejected CIs considering the RFU constraints and execution frequency of CIs, minimum and maximum length of initial CIs and minimum length of rejected CIs. Application names with rejected CIs are shown in bold face.

As mentioned in Section 3, for generating appropriate CIs two approaches including *CIGen* and *Integrated Framework* were used. For CIs generated by *CIGen*, the mapping process was done and some of them were rejected again at the mapping stage because of the RFU violation of routing resource constraints. In this method, CIs were generated using a more conservative approach. Some of the CIs can not be supported and are rejected. Fig. 4 shows that 10 applications already have CIs which are non-mappable on RFU. These rejected CIs have to execute on the base line processor and offer no speedup.

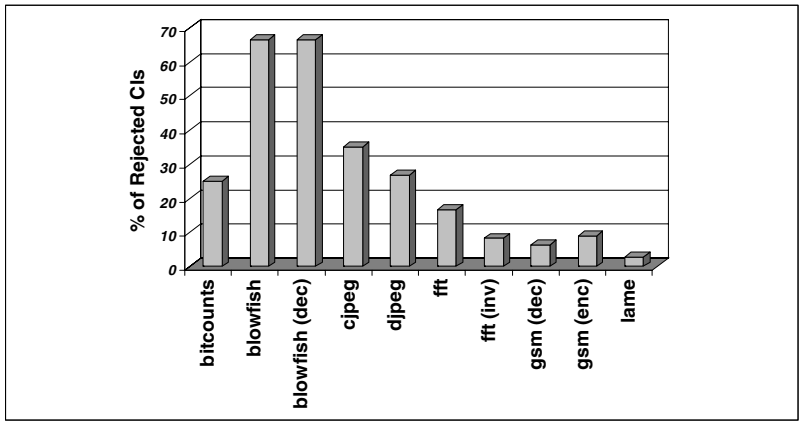

Fig. 4. Percentage of rejected CIs generated by *CIGen*

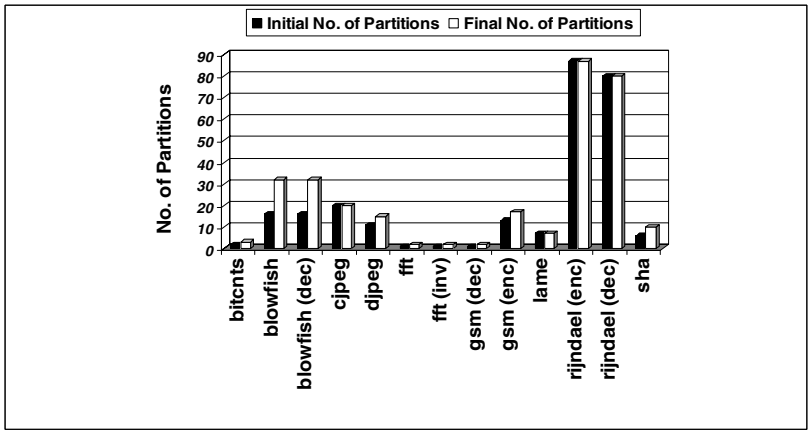

Fig. 5. Initial and final number of partitions generated by the *Integrated Framework*

In the second approach, we used the *Integrated Framework* to generate appropriate CIs. Using this approach, which iteratively generates CIs, all CIs were successfully mapped on the RFU during partitioning process. This is one of the most important advantages of the proposed design flow. Fig. 5 shows the initial and final number of partitions (CIs) generated for each application using the *Integrated Framework*. Initial number of CIs is the number of partitions generated by the temporal partitioning algorithm. In addition, the final number of partitions means the number of CIs that are generated after performing the iterative process to modify and generate appropriate CIs.

Fig. 6 shows the maximum length of the critical path for the generated CIs. According to the results obtained, for *cjpeg, fft, fft(inv), gsm(end)* and *gsm(dec)*, the mapping algorithm took advantage of spiral shape mapping to handle critical paths with length more than 5.

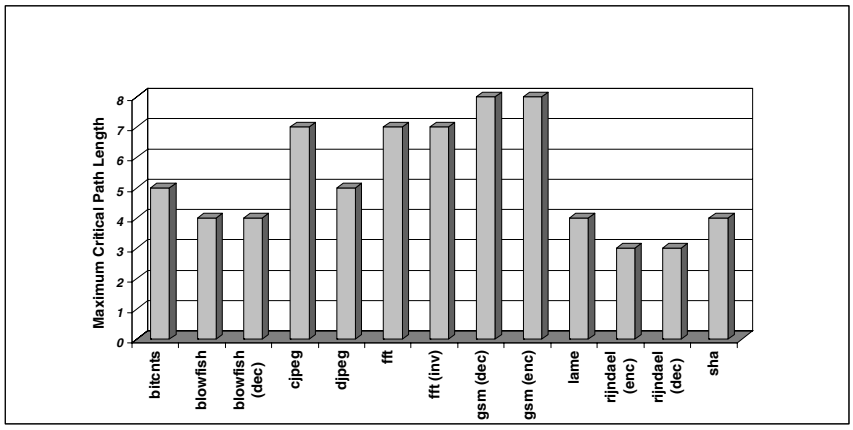

Fig. 6. Maximum critical path length for CIs generated by the *Integrated Framework*

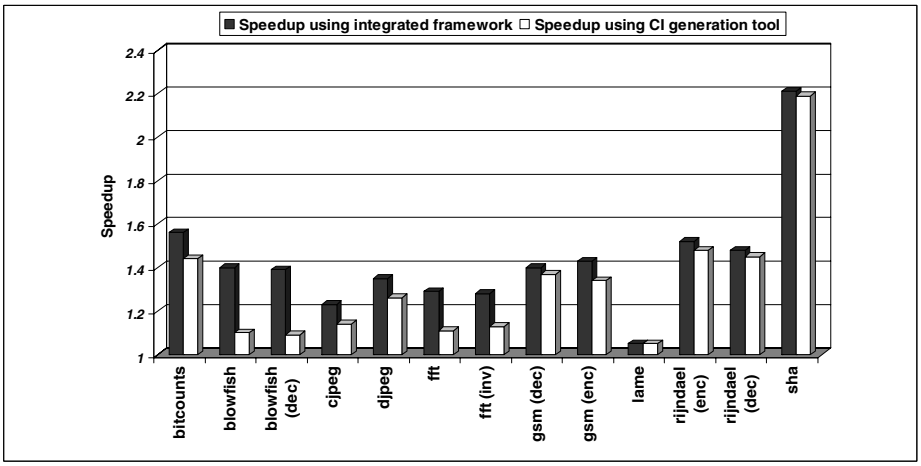

Fig. 7. Speedup comparison between *CIGen* and the *Integrated Framework*

Finally, Fig. 7 shows the speedup comparison for *CIGen* and the *Integrated Framework*. The *Integrated Framework* generated CIs all of which can be mapped on the RFU, because, temporal partitioning stage is properly aware of the mapping process result and is iteratively done according to the feedbacks obtained from the mapping phase. According to Fig. 7, speedup increases using the *Integrated Framework*. For *lame, CIGen* and the *Integrated Framework* generated similar CIs, therefore, the *Integrated Framework* does not offer more improvement for *lame* in compared to *CIGen*.

6 Conclusion

In this paper, an integrated framework was presented to address generating appropriate custom instructions and mapping them on RFU of an adaptive extensible processor. First, an RFU was presented for AMBER, a dynamic adaptive extensible processor. Some CIs of the attempted applications were rejected because of RFU primary constraints. One method for generating appropriate CIs is applying the RFU constraints to the CI generation tool and extracting the CIs which meet these constraints (*CIGen*). Using *CIGen* may still cause some generated CIs to be rejected. This approach does not have the capability of considering constraints such as routing resource constraints before mapping since it is unaware of the mapping process result. The *Integrated Framework* is the second approach which uses a mapping-aware temporal partitioning algorithm for generating appropriate CIs. In this framework, each rejected CI is partitioned to smaller partitions and iteratively modified to meet the RFU constraints. The experimental results showed that for the attempted benchmarks, the algorithm successfully mapped all CIs on the RFU. Our proposed mapping algorithm uses spiral shaped paths to cover CIs with critical paths longer than the RFU depth. Also, the *Integrated Framework* brought about more speedup enhancement comparing with *CIGen* by generating CIs which have less running time on the RFU.

Acknowledgement

The authors would like to thank *System LSI* Laboratory of Kyushu University for providing the necessary facilities and equipments. This work has been supported by Iran Telecommunication Research Center *(ITRC)*.

References

[1] Arnold, M., Corporaal, H., Designing domain-specific processors. In Proceedings of the Design, Automation and Test in Europe Conf, 2001, pp. 61-66.

[2] Atasu, K., Pozzi, L., Lenne, P., Automatic application-specific instruction-set extensions under microarchitectural constraints, 40th Design Automation Conference, 2003.

[3] Bobda, C., Synthesis of dataflow graphs for reconfigurable systems using temporal partitioning and temporal placement, Ph.D thesis, Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, 2003.

[4] Clark, N., Kudlur, M., Park, H., Mahlke, S., Flautner, K., Application-specific processing on a general-purpose core via transparent instruction set customization, In Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, 2004.
[5] Halfhill, T.R., MIPS embraces configurable technology, Microprocessor Report, 3 March 2003.
[6] Karthikeya, M., Gajjala, P., Dinesh, B., Temporal partitioning and scheduling data flow graphs for reconfigurable computer, IEEE Transactions on Computers, vol. 48, no. 6, 1999, pp.579–590.
[7] Kastner, R. Kaplan, A., Ogrenci Memik, S., Bozorgzadeh, E., Instruction generation for hybrid reconfigurable systems, ACM TODAES, vol. 7, no. 4, 2002, pp. 605-627.
[8] Lee, C., Potkonjak, M., Mangione-Smith, W.H., MediaBench: A tool for evaluating and synthesizing multimedia and communications systems, In Proceedings of the 30-th Annual Intl. Symp. On Microarchitecture, 1997, pp 330-335.
[9] Mehdipour, F., Saheb Zamani, M., Sedighi, M., An integrated temporal partitioning and physical design framework for static compilation of reconfigurable computing system, International Journal of Microprocessors and Microsystems, Elsevier, vol. 30, no. 1, Feb 2006, pp. 52-62.
[10] Micheli, G.D., Synthesis and optimization of digital circuits, McGraw-Hill, 1994.
[11] Noori, H., Murakami, K., Inoue, K., General overview of an adaptive dynamic extensible processor architecture, Workshop on Introspective Architecture (WISA'2006) , 2006.
[12] Ouaiss, I., Govindarajan, S., Srinivasan, V., Kaul M., Vemuri R., An integrated partitioning and synthesis system for dynamically reconfigurable multi-FPGA architectures, In Proceedings of the Reconfigurable Architecture Workshop, 1998, pp. 31-36.
[13] Razdan, R., Smith, M.D., A high-performance microarchitecture with hardware-programmable functional units, In Proceedings of the 27th Annual International Symposium on Microarchitecture, 1994, pp. 172-180.
[14] Spillane, J., Owen, H., Temporal partitioning for partially reconfigurable field programmable gate arrays, IPPS/SPDP Workshops, 1998, pp. 37-42.
[15] Tanougast, C., Berviller, Y., Brunet, P., Weber, S., Rabah, H., Temporal partitioning methodology optimizing FPGA resources for dynamically reconfigurable embedded real-time system, International Journal of Microprocessors and Microsystems, vol. 27, 2003, pp. 115-130.
[16] Writhlin, M., Hutchings, B., A dynamic instruction set computer, In Proceeding IEEE Symposium on Field Programmable Custom Computing Machines, IEEE Computer Society Press, 1995, pp. 99-107.
[17] Ye, Z.A., et al., Chimaera: A high-performance architecture with tightly-coupled reconfigurable functional unit, In Proceeding of 27th ISCA, 2000, pp. 225-235.
[18] Yu, P., Mitra, T., Characterizing embedded applications for instruction-set extensible processors, In Proceedings of Design and Automation Conference, 2004, pp. 723- 728.
[19] http://www.eecs.umich.edu/mibench.
[20] http://www.simplescalar.com.