An Efficient Method for Custom Instruction Mapping Under the Safety Assurance of Code Generation

Yongping Luo
Dept. of Computer Engineering
Hunan University
Changsha, China
luoyongping@hnu.edu.cn

Yuchun Ma
EDA Lab of Computer Science
Tsinghua University
Beijing, China
myc@mail.tsinghua.edu.cn

Qiang Wu
Dept. of Computer Engineering
Hunan University
Changsha, China
wuqiang2000@gmail.com

***Abstract*—In the work flow of embedded systems, processor design is the critical technology to directly affect the performance of whole system. In this process, instruction mapping is responsible for identifying the portions of target application program which match with custom instruction (CI) and implementing code generation for extension processor. However, traditional instruction mapping approaches would not consider the problem of mapping safety so that the mapping result is not reliable. In addition, as target applications get more complex, the drawback of large time cost makes embedded system less efficient. In this paper, we build safety check to ensure that the matched subgraphs can really be executed by hardware accelerator. And we propose an efficient matching method according the logic information of application program to reduce matching time and search space. For overlapping matched subgraphs, we also build up Maximal Weight Independent Set-based model to obtain better speedup of extension processor and the experiment results show the superiority of our method.**

Keywords-pattern matching; custom instruction; extension processor

I. INTRODUCTION

The improvement of embedded application program's complexity makes the traditional embedded system meet the greater challenge from their power, chip area and performance goals. Since the general-purpose processor is more limited in computational efficiency for large subject application, application specific integrated circuits(ASICs) are used as cushions to the conflicts in computational speed demands. But ASICs are custom hardware modules for execution of the portions which demand for acceleration in application program and their functionality can not be changed once modules are integrated into system. Thus, the cost of the system may be expensive for handing different applications.

In order to accommodate specification changes, application specific instruction set processor (ASIP) provides a good designing scheme to balance the performance demands and flexibility. During the design process, custom instructions (CIs) are generated by analyzing application program, so the computation intensive portions of application can be executed by custom function units (CFUs) which augment the instruction set of ASIP. Here, ASIP can invoke CFUs to implement acceleration through dynamic mapping mechanism. Thus, the system can modify the software tool chain to select corresponding configuration scheme for different applications. This reduces the hardware cost of the system and the design time consuming.

However, the process of CI design is a hard work especially for the large application programs. A key step in this process is instruction mapping which has two main phase to handing. The first phase is pattern matching that it identifies the portions of application program which match with custom instruction. Pattern matching is regarded as NP-complete problem and the matching time increase exponentially when the application is too large. The second phase is instruction covering which selects final portions of application to maximize the speedup of accelerator. This problem is also difficult because the number of final selected matched subgraphs should be minimized. Clearly, instruction mapping is a challenge work for researches.

Many researchers had studied the utility of custom instructions. For pattern matching, The approach by Clark et al. [1] uses vflib graph matching library to regard pattern matching as a subgraph isomorphism problem. But the matching time is too long when dealing with large applications. A tree matching approach is presented to tackle subgraph mapping in [2]. In this approach, the matching process is very quick because the date flow graph (DFG) of subject application is partitioned into trees. The drawback of this method is that the matching result is suboptimal since many DFGs are not trees.

Here, we classify the subgraph isomorphism and tree matching as structural matching. Unlike structural matching, a method based on symbolic algebra is presented in [4] that it performs functional matching through polynomial expression which is from the decomposition of application. This technique can not hand bit-wise operations and the matching time is also exponential in the worst case. Similar methods are presented in [3] [7].

For instruction covering, unate covering [5] is used to minimize the number of execution cycle on accelerator. As this method dose not consider the situation of overlapping subgraphs, it loses more opportunities for speedup. Work by Cong [6] attacked the same problem by using a binate covering formulation and the overlapping subgraphs are allowed. But the risks of mapping confusion for accelerator are increased during the code generation phase.

Above all, most of previous works are ineffective in reducing the matching search space and matching time. Furthermore, traditional approaches neglect the safety of instruction covering so that many matched subgraphs can not really be executed by processor. In this paper, we give consideration to system computation efficiency and performance under the premise of mapping safety.

Our contributions of this paper as follows:

- We introduce control data flow graph (CDFG) as intermediary representation of application program. According the information of basic block (BB) in CDFG, we propose a set of strategies to reduce the matching time.
- In order to ensure the safety of instruction covering, we adopt maximal weight independent set-based model to hand overlapping subgraphs and build safety check mechanism for avoiding the mistakes of code generation.

II. Problem Statement

During the process of code generation, most of the previous works are focused on how to improve the performance of accelerator. These works include matching more subgraphs in subject application and selecting matched subgraphs which can minimize the compute cycle or execution time. Nevertheless, the safety of mapping, which is related to the correctness of compute result of application program is neglected. The problem of mapping safety mainly involves considering the control information between nodes, maintaining the same data dependence before and after instruction covering, and avoiding mistake when dealing with complex mapping such as for the situation of overlapping matched subgraphs.

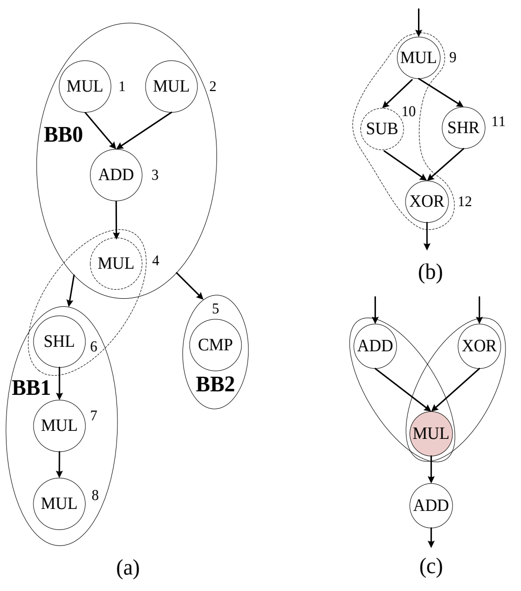


Figure 1. Some examples of the mapping safety issue

Three cases which are relevant to unsafe mapping are shown in Fig. 1. DFG is generally used as the representation of application program. But it lacks the logic information of application program such as conditional statement so that the nodes of matched subgraph may be included in different basic blocks (BBs) as shown in Fig. 1 (a). Such matched subgraph is hard to be mapped into accelerator because basic block (BB) is the smallest logic execution sequence. The unsafe mapping of the second case is occurred when the matched subgraph is replaced by custom instruction. For instance of Fig. 1 (b), the data dependence becomes cyclic after matched subgraph 9-10-12 is replaced by a CI node, which would cause destruction of data dependence. The third case is about how to handing overlapping subgraphs during instruction covering. Overlapping subgraphs(as shown in Fig. 1 (c)) can provide more opportunities to obtain high speedup, but in the meanwhile it is easy to cause mapping confusion during code generation. So the key of this problem is that how to obtain higher performance and there are no overlapping subgraphs which are finally mapped to processor.

In this work, our goal is to guarantee the accuracy of compute result of subject program and the normal execution on accelerator. In addition, we use some techniques to improve the efficiency of matching process and the speedup performance of accelerator.

III. Pattern Matching

Let *G(V, E)* be a subject graph from application program, P_{set} be a CI pattern set, pattern matching is to find all subgraphs in *G(V, E)* which match with each CI of P_{set}. As mentioned before, the search space of traditional matching approach is too large and the safety of matching is neglected. To solve the problem, we introduce CDFG to obtain the information of BB such as the number of instruction nodes and iteration times in each BB.

A. Matching based on BB

During the pattern matching, we adopt CDFG as the representation of application program for analyzing the whole structure information of program. A CDFG contains many BBs and the dependence of instruction nodes in each BB can be represented as DFG. Through CDFG, we can obtain the execution sequence between BBs which is important for the problem of mapping safety. Here, we chose BB as the elementary matching search unit because it is coarse-grained to improve the efficiency of matching process and is more safe for instruction covering. In order to reduce the matching search space, we classify the BBs into two groups according the number of instruction nodes in BB and CI. One group is big BB and the other is small BB.

To better describe the classification of BB, we denote the number of instruction nodes of BB as N_{BB} and N_{CI} is the number of instructions of CI. If $N_{BB} \geq N_{CI}$, then we classify the BB as big BB, otherwise, the BB is small BB. As BB is the smallest execution unit, there is no need to do matching work for all small BBs. Such as in Fig. 2, N_{CI} of CI pattern P_{set-1} is 4 so that we prune the search space of the basic blocks BB2 and BB3 which the N_{BB} is respectively 1 and 2. Here, BB2 and BB3 are classified as small BB and BB1 is big BB. Through the classification of BB, the matching search space can be obviously reduced.

B. Comparing the Performance for Overlapping Subgraphs

For overlapping subgraphs in big BB, we compare the performance gain between overlapping subgraphs in advance. The performance gain is calculated through (1) and (2).

$$C_{sav} = f_{BB} * (C_{sw} - C_{hw}) \tag{1}$$

$$C_{sw} = \sum_{i \in CI} Cycle(I_i) \tag{2}$$

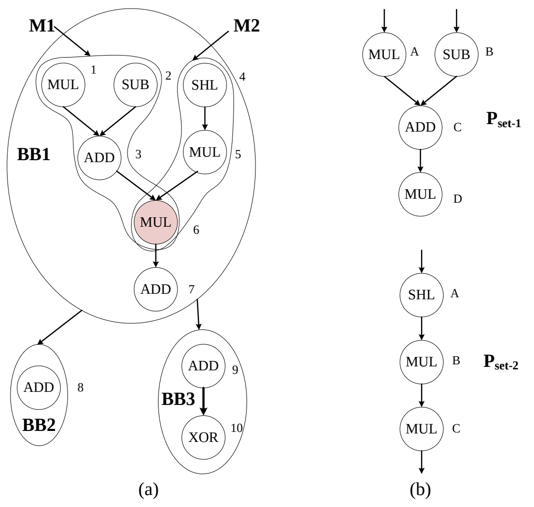


Figure 2. (a) Subject Graph (b) CI Pattern

Cycle(b$_i$) is the number of execution cycles for a basic instruction. C_{sw} is the sum number of software execution cycles for all primitive instructions in subgraph. C_{hw} is the number of execution cycles for critical path when the subgraph is regard as a custom instruction. f_{BB} is the execution times of BB. C_{sav} is the number of saving cycles when the subgraph is implemented by hardware.

In Fig. 2, assuming that the matching work of P_{set-1} have been completed (*M1* is identified), it finds the same instruction operation(MUL operation) between *M1* and P_{set-2} during the matching process of P_{set-2}. Thus, we compare the performance between P_{set-1} and P_{set-2}. If $C_{sav}(P_{set-2}) \leq C_{sav}(P_{set-1})$, we will no longer do the matching work of P_{set-2} at the neighborhood of nodes 1 and 6 although subgraph *M2* match with P_{set-2}. The reason is that we only select one subgraph to be mapped to processor for mapping safety. Therefore by comparing the performance gain between overlapping matched subgraph in advance, the matching search space can be reduced.

IV. Instruction Covering

A. *Safety Check for Matched Subgraph*

After the work of pattern matching, the matched subgraphs are replaced by CIs. During the process of replacement, most of previous works neglect the safety of instruction covering. Here, we describe the detail of the process of safety check for Fig. 3 which is from Fig. 1(b).

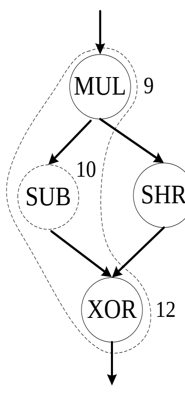
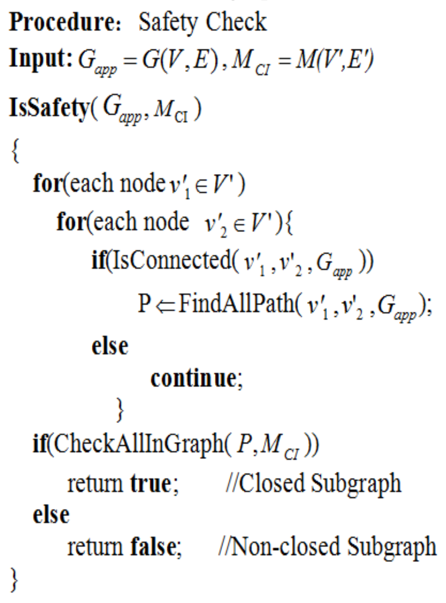


Figure 3. An example from Fig. 1(b)

The safe matched subgraph is a closed subgraph that all nodes in the paths between two arbitrary nodes are contained within subgraph. Otherwise, it is a non-closed subgraph. For example, node 11 is in the path between nodes 9 and 12, but it is not included in subgraph 9-10-12. So we call subgraph 9-10-12 a non-closed subgraph and it is unsafe to be mapped to processor.

Fig. 4 shows the algorithm of safety check for matched subgraph. We find all paths between two arbitrary nodes firstly and then check whether the nodes of all paths is in matched subgraph.



```
Procedure: Safety Check
Input: $G_{app} = G(V,E), M_{CI} = M(V',E')$
IsSafety( $G_{app}, M_{CI}$ )
{
  for(each node $v'_1 \in V'$ )
    for(each node $v'_2 \in V'$ ){
      if(IsConnected( $v'_1, v'_2, G_{app}$ ))
          P $\Leftarrow$ FindAllPath( $v'_1, v'_2, G_{app}$ );
      else
          continue;
    }
  if(CheckAllInGraph( $P, M_{CI}$ ))
      return true;      //Closed Subgraph
  else
      return false;     //Non-closed Subgraph
}
```

Figure 4. Algorithm of Safety Check

B. *Instruction Covering*

When the unsafe matched subgraphs have been pruned, it is important to determine which matched subgraphs can be finally mapped to accelerator. But as mentioned previously, the mapping safety of overlapping subgraphs and high performance should simultaneously be considered.

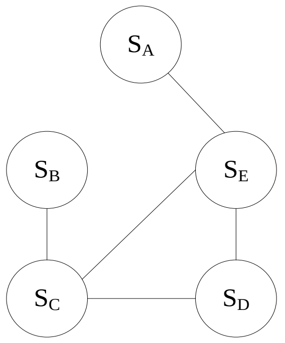


Figure 5. The model of independent set

To solve the problem, we build up maximal weight independent set-based model. The model of independent set is shown in Fig. 5. The graph vertex (such as S_A) represents matched subgraph. There is an edge between vertexes if matched subgraph is overlapped. The weight of each vertex is the number of saving cycles which is calculated by (1) and (2).

V. Experiment

We implemented our proposed approach with C++ for evaluating its effectiveness. There are two inputs to our framework: the C application programs which from Trimaran [8] and a CI pattern set which includes 30 CIs. Trimaran is a high-level synthesis tool and we can obtain CDFG or DFG from applications through the compilation of Trimaran. Our machine configuration is: Inter(R) Core(TM) i3 CPU, 2GB memory.

Fig. 6 shows the number of unsafe matched subgraphs in each benchmark. Although the minimum number of unsafe pattern is only 4 in *dag*, the compute result may be incorrect. In other benchmarks, the number of unsafe instances are much higher than *dag*. So safety check is an essential step during instruction mapping.

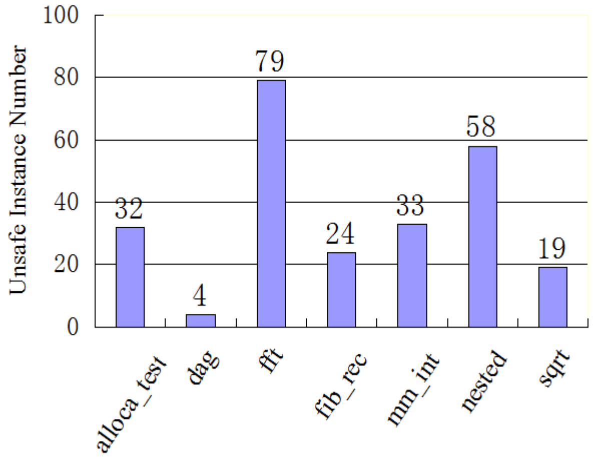


Figure 6. The Number of Unsafe Matched Subgraphs

To verify the effectiveness of our matching algorithm, we compare the matching time when intermediary representation of application program is DFG and CDFG respectively. It is evident from Figure 7 that our method (CDFG) take less time than the traditional approach using DFG as representation. Especially in *fft* and *fib_rec*, the matching time is reduced nearly by half.

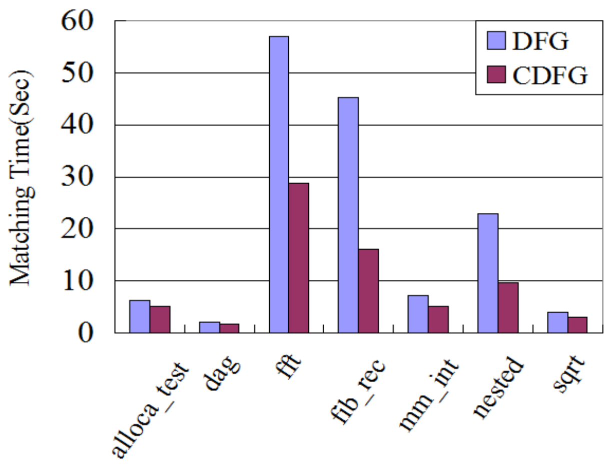


Figure 7. Comparison of Matching Time

We also compare the performance gain when overlapping subgraphs are considered or not during matching process. In the method of direct replace, there are no overlapping subgraphs during matching. The saving cycle are shown in Table I. "#Final" is the number of matched subgraphs which are finally mapped to processor. "Saving" is the saving cycles after replacement. The improvement of our method is shown in the column of "Inc". From Table I, it can obtain more performance gain by adopting MWIS replace.

TABLE I. Saving cycle of Two Methods

Benchmark	MWIS Replace		Direct Replace		Inc
	#Final	Saving	#Final	Saving	
alloca_test	3	18771	1	2694	6.97x
dag	11	24891	10	20618	1.21x
fft	20	144281	12	71727	2.01x
fib_rec	10	122570	7	103285	1.18x
mm_int	10	82448	8	43955	1.88x
nested	4	17434	3	13936	1.25x
sqrt	9	54357	7	33636	1.62x

VI. Conclusion

In this paper, we introduce CDFG and independent set to ensure the safety of instruction mapping. And as some strategies based on the structure information of BB are adopted in our method, the matching search space is reduced. In addition, the performance gain is improved through maximal weight independent set-based model.

References

[1] N. T. Clark, H. Zhong, and S. A. Mahlke. Automated custom instruction generation for domain-specific processor acceleration.IEEE Transactions on Computers, 54(10):1258–1270, 2005.

[2] A. Aho, M. Ganapathi, and S. Tijang. Code generation using tree pattern matching and dynamic programming.ACM TOPLAS, 11(4):491–516, Oct. 1989.

[3] N. Arora, K. Chandramohan, N. Pothineni, A. Kumar. Instruction Selection in ASIP Synthesis Using Functional Matching. In Proceeding. IEEE International Conference on VLSI Design, 2010.

[4] A. Peymandoust, L. Pozzi, P. Ienne, and G. D. Micheli. Automatic instruction set extension and utilization for embedded processors. In Proceedings. IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 2003, pages 108– 118, 2003.

[5] N.Clark, A.Hormati, S.Mahlke, S.Yehia. Scalable Subgraph Mapping for Acyclic Computation Accelerators. Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems. New York: ACM, pp. 147~157, 2006.

[6] Cong J, Fan Y, Han G, and Zhang Z, Application-specific Instruction Generation for Configurable Processor Architectures. FPGA, pp.183- 189, 2004.

[7] N. Cheung, S. Parameswaran, J. Henkel, and J. Chan. Mince: matching instructions using combinational equivalence for extensible processor. In Conference on Design, Automation and Test in Europe, pages 1020–1025, 2004.

[8] http://www.trimaran.org.