A Clustering ILP Model for Fast Instruction Selection in Embedded Applicated Specific Processor Design

Kang Zhao and Jinian Bian
EDA Lab, Dept. of Computer Science & Technology
Tsinghua Univ., Beijing 100084, China, +86-10-62785564
zhao-k04@mails.tsinghua.edu.cn, bianjn@tsinghua.edu.cn

***Abstract*—Recently to face the challenges of high efficiency and time-to-market pressure, instruction set auto-customization for application-specific instruction-set processors (ASIP) has become an attractive technology. In this process how to select an optimized instruction set in a short time is crucial to the whole synthesis of ASIP. To address this issue, we formulate the instruction set selection problem with a clustering integer linear programming (ILP) model under multi-constrains, and present a fast cutting plane algorithm to resolve this problem. Finally, the proposed model and algorithm are evaluated through a case study on FIR filter applications.**

***Keywords*—Instruction selection, cutting plane algorithm, ILP, ASIP.**

I. INTRODUCTION

In the era of high-performance, low-power and cost-effective system, automatic customization to the application requirements is the key to achieve fast turn-around time in a competitive market [1]. Application Specific Instruction-set Processor (ASIP) is a processor designed for a set of particular applications, which provides a good tradeoff between efficiency and flexibility [2]. To face the challenges of high efficiency and time-to-market pressure, customizable ASIP has become an attractive technology.

In ASIP design, specific instruction set customization plays an important role in the whole process. Selecting an optimal instruction set is crucial to enhancing the performance of ASIP. However, due to the complexity of the problem, selecting a high-performance instruction set is more difficult.

Recently, a large body of work has been done on instruction selection. Alauddin Alomary describes a formal method that selects the instruction set of ASIP to maximize the chip performance under constrains of chip area and power consumption [3], in which instruction set selection problem is first formulated with integer linear programming (ILP) and resolved by using a branch-and-bound algorithm. Similarly, in the CECS of University of California, Jong-eun Lee and Nikil D.Dutt also formulate the instruction selection problem with ILP [4]. To reduce the complexity, they divide the instruction set into three parts and present an effective heuristic algorithm. In [5] novel algorithms addressing application specific instruction compilation are presented and the problem is formulated as a graph covering under multi-constrains, but in essential the instruction selection in the whole process is also settled with integer linear programming.

However, there are still two intrinsic limitations for those techniques mentioned above. First, the instruction selection problem is formulated with ILP without considering the inherent functional relations between instructions. Thus, it is relatively difficult to find a fast method to resolve this problem. Second, because the exploration space of this problem is very large, the proposed algorithms are most time-consuming. To increase the execution speed, several heuristic algorithms are even presented in those papers.

To solve this problem, we proposed a novel fast method. First, we formulate the instruction selection problem with a clustering ILP model under multi-constrains, which considers the functional relations between instructions adequately. Second, a fast cutting plane algorithm is presented to settle the clustering ILP model, which is demonstrated to be efficient through a case study on FIR applications.

The rest of the paper is organized as follows. In Section 2, the background of instruction selection problem is first given. And then, to formulate this selection problem, in Section 3 a clustering ILP model is described in detail. In Section 4, a fast cutting plane algorithm based on ILP is then proposed to solve the selection problem. Finally, experimental results obtained from a case study on FIR applications are presented in Section 5. The last section draws the conclusion.

II. BAKEGROUND

Instruction selection process plays an important role in the whole instruction set synthesis for ASIP design.

Figure 1 is the instruction set synthesis framework we utilize, which starts with the behavior specification of applications and gradually moves the design to lower levels of implementation. First the application benchmarks with C language are compiled into an intermediate specification HCDFG (Hierarchal Control Data Flow Graph). And then the initial instruction set is defined via functional mapping from operations in HCDFG to a pre-designed instruction library. To get an instruction set with better performance, instruction selection process is utilized to extract a best sub-set from the initial instruction set. Finally, through a partial tuning we can achieve a further improvement and get the final instruction set.

1-4244-0387-1/06/$20.00 ©2006 IEEE

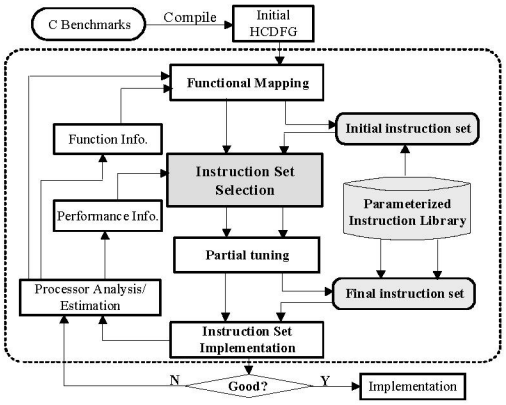

Figure 1. Framework of instruction set synthesis

In this paper we will focus on the formulation and solutions for the instruction selection process from the initial instruction set to the final instruction set.

III. A CLUSTERING ILP MODEL FOR INSTRUCTION SELECTION

Instruction selection is an optimization problem. As explained previously, it accepts the initial instruction set from functional mapping process, which only ensures the functionality completeness of the instruction set and results in many redundancies to be filtrated. In the initial instruction set, there may be functional overlapping between two instructions. So selecting an instruction set with both complete functionality and low performance is necessary.

For an optimization problem, it is the first task to model the behavior with mathematical tools. Generally an optimization problem contains a design space and a metric measuring the quality of trial solutions. The space is designed as a three constrains with execution time, hardware area and power consumption. Without loss of generality, we assume $I = \{u_1, u_2, \ldots, u_n\}$ is the initial instruction set and $I^* = \{v_1, v_2, \ldots, v_t\}$ is the optimized instruction set, so we have:

$$|I| = n \tag{1}$$

$$|I^*| = t \tag{2}$$

$$|I^*| \le |I| \Rightarrow t \le n \tag{3}$$

We define the constraint of the maximum hardware area A_{max} and the area account function $area(u_i)$ on the instruction u_i, then we have this inequation:

$$\sum_{i=1}^{n} w_i \times area(u_i) \le A_{\max} \tag{4}$$

$$w_i = 0 \ or \ 1 \quad 0 \le i \le n$$

where w_i is a binary variable which is '1' if instruction u_i is selected and '0' if not selected. Then the instruction selection problem can be formulated with an integer linear programming as follows:

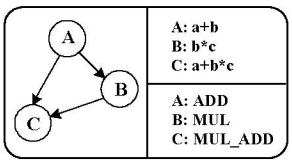

Figure 2. An example of functional dependency between instructions

Notations:

- I : Initial instruction set
- I^*: Selected instruction set
- G : Mapping relationship from one set to another
- T : Total execution time
- P : Power consumption

Problem: Given an initial instruction set I, and constrains (3) and (4) mentioned above, generate a mapping G from I to the final instruction set I^*, so that the each function containing in I is covered and the total execution time T and power consumption P is minimized.

However, because the design space of all instruction set is very large and difficult to conceptualize, its scope is intentionally limited according to the specific applications and experimental requirements. Therefore, we utilize an idea of functional cluster and make an improved alteration on the ILP formulation. The instruction subset with same functionality is then used as the minimal unit instead of only one instruction.

We assume $F = \{F^1, F^2, \ldots, F^m\}$ is the functionality set and it satisfies that arbitrary two elements have no intersections, which can be formulated as $F^i \cap F^j = \phi$ ($i \ne j$ and $1 \le i, j \le m$). Based on this assumption, we define u^i_j is the j^{th} instruction in F^i:

$$\begin{aligned}
&F = \left\{F^1 \cup F^2 \cup \cdots \cup F^m\right\} \quad |F| = m \\
&F^1 = \left\{u^1_1, u^1_2, u^1_3, \cdots, u^1_{n_1}\right\} \quad |F^1| = n_1 \\
&F^2 = \left\{u^2_1, u^2_2, u^2_3, \cdots, u^2_{n_2}\right\} \quad |F^2| = n_2 \\
&\quad \cdots\cdots \\
&F^m = \left\{u^m_1, u^m_2, u^m_3, \cdots, u^m_{n_m}\right\} \quad |F^m| = n_m \\
&F^i \cap F^j = \Phi \quad (i \ne j,\ i, j \in [1, m])
\end{aligned} \tag{5}$$

Actually, there are functional dependencies among those instructions, especially between the atom instructions and the complicated instructions. For example, as shown in Figure 2, *MUL* function can be achieved with *ADD* instruction through software techniques, and *MUL_ADD* instruction also can be obtained through the coupling of *ADD* and *MUL*. Whether using the instruction or the software techniques through other instructions is decided by the statistical analysis and the special application constrains. Accordingly we define the function $U(F)$ as the functional mapping and preprocessing result on the set F.

Here, we use a new binary variable x^i_j instead of w_i. When the instruction u^i_j is selected, the value of x^i_j is 1. So we have the following relations:

$$t=\sum_{i=1}^{|U(F)|}\sum_{j=1}^{n_i}x_j^i \quad x_j^i=0\ or\ 1 \quad |U(F)|\le m \tag{6}$$

Finally, for the sake of functionality completeness of the final instruction set, at least one instruction should be selected in each F^i. With this additional constrain, we formulate the instruction selection problem with a clustering ILP model as follows:

$$Min:\sum_{i=1}^{|U(F)|}\sum_{j=1}^{n_i}x_j^i\times f_j^i\times\{q\cdot time(u_j^i)+(1-q)power(u_j^i)\}$$

$$Subject:\begin{cases}\sum_{i=1}^{|U(F)|}\sum_{j=1}^{n_i}x_j^i\times area(u_j^i)\le A_{\max}\\ t\le n\Rightarrow\sum_{i=1}^{|U(F)|}\sum_{j=1}^{n_i}x_j^i\le n\\ \forall i\left(1\le i\le|U(F)|\ \wedge\ \sum_{j=1}^{n_i}x_j^i\ge 1\right)\end{cases} \tag{7}$$

$$x_j^i=0\ or\ 1$$

Where f_j^i is the using account of the instruction u_j^i in the benchmarks, q represents how much proportion timing occupies compared to power, and $time(u_j^i)$ and $power(u_j^i)$ stand for the calculation functions for the execution time and power consumption of instruction u_j^i.

Analysis: For the previous ILP definition, the exploration space is obviously 2^n. However, due to the clustering constraint, the design space of the improved ILP is reduced greatly:

$$\because n=n_1+n_2+\cdots+n_m$$

$$\therefore C_{n_1}^1 2^{n_1-1}\times C_{n_2}^1 2^{n_2-1}\times\cdots\times C_{n_{|U(F)|}}^1 2^{n_{|U(F)|}-1}$$

$$=(n_1\cdot n_2\cdots n_{|U(F)|})2^{n_1-1+n_2-1+\cdots+n_{|U(F)|}-1}$$

$$\le(n_1\cdot n_2\cdots n_m)2^{n_1-1+n_2-1+\cdots+n_m-1}$$

$$=(n_1\cdot n_2\cdots n_m)2^{n-m}$$

$$\therefore[2^n-(n_1\cdot n_2\cdots n_m)2^{n-m}]/2^n=1-(n_1\cdot n_2\cdots n_m)/2^n \tag{8}$$

Therefore, the exploration space has been reduced by a percentage of $(1-(n_1n_2\cdots n_m)/2^m)$.

IV. Fast Cutting Plane Algorithm

We have formulated the instruction selection with a clustering ILP model. However, because the exploration space of instructions is very large, ordinary solutions are complicated and time-consuming to this problem. To increase the execution rate of instruction selection, we present a fast solution based on the Gomory cutting plane algorithm.

First, we broaden the integral constrains and get the corresponding linear programming problem P_0, as shown below. So we can solve P through P_0, which is just the motivation of cutting plane algorithm. According to the analysis of the difference and relationship between P and P_0, we have those conclusions:

- The area of the feasible results for problem P is the subset of the one for problem P_0.
- If there is no feasible result for P_0, no results for P exist.
- If the feasible result for P_0 is integral, so it is also for P.

$$P\begin{cases}\text{Min } c^Tx\\ \text{s.t. } Ax=b\\ x\ge 0\\ x=0,1\end{cases}\Rightarrow P_0\begin{cases}\text{Min } c^Tx\\ \text{s.t. } Ax=b\\ x\ge 0\\ 0\le x\le 1\end{cases}$$

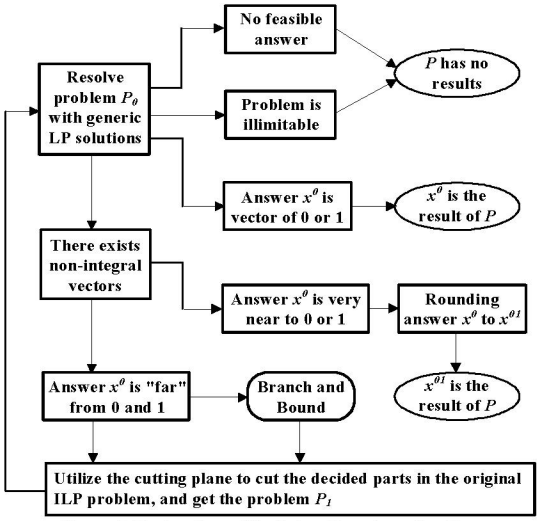

Figure 3. Design flow of the fast cutting plane algorithm

Then, the novel algorithm will be explained in detail. Its main differences from Gomory cutting plane algorithm is that it utilizes rounding method to increase the execution speed and uses the idea of branch and bound to avoid the accidental instances, as shown in Figure 3.

After the ILP problem P is converted to a LP problem P_0, we can solve it with appropriate algorithms and tools for LP problem. If there is no feasible result or P_0 is illimitable, according to the conclusions mentioned above, problem P has no feasible results; if one answer is just integer 0 or 1, it will also be the feasible result of P; if the answer is very near to 0 or 1 and the gap is under a sufferable horizon, we will round it as the feasible result of P; for most non-integral results, our strategy is to cut the useless space of variables which have been confirmed to be feasible results of P, and get a new LP problem P_1, then go back to the original step and solve the problem P_1. In this process, if all the results of problem P_0 are non-integral, the whole program will be trapped into an endless loop. Therefore, for this instance we utilize the idea of branch and bound, confirm an arbitrary variable to be 0 and 1, run respectively and make a final comparison with the feasible results.

For a general ILP problem, it will be very fast when resolved without integer constrains. Therefore, based on the LP problem, the new algorithm limits the exploration space through cutting the confirmed feasible results and speeds up the execution rate.

V. A Case Study on FIR Applications and Experimental Results

To evaluate the proposed model and algorithms, we select a set of FIR filter applications as the test bench, which is a DSP core named *cf_fir* [6] and is programmed with C. This project contains five benchmarks, and each file has three parameters. For example, *fir*(12,16,10) represents that the filter order of delay taps is 12, the input precision is 16 and the coefficient precision is 10. In the following content we will continue to use this notation.

Table 1. Pre-profiling results for FIR benchmarks

	+	*	-	/	<<	>>	&	||	=	>=	>	!=	branch	loop
fir(7,16,16)	140	85	96	20	6	62	79	13	970	40	199	24	128	115
fir(12,16,10)	205	184	167	31	26	101	252	17	1512	60	333	33	202	191
fir(24,8,8)	175	80	259	56	26	111	359	2	2230	108	426	30	372	312
fir(24,16,16)	446	255	309	56	26	209	505	30	3123	108	672	58	372	362
fir(33,16,16)	608	345	420	74	36	287	687	39	4247	144	924	76	500	491
Percentage	**6.8%**	**3.8%**	**4.7%**	**0.8%**	**0.4%**	**3.2%**	**7.7%**	**0.4%**	**47.8%**	**1.6%**	**10.4%**	**0.9%**	**5.6%**	**5.5%**

First, we compile those benchmarks into an intermediate specification and then make a statistical analysis on it. Part of the profiling results on the FIR benchmarks are summarized in Table 1. These statistics illustrate the utilized numbers of operations and the lowest row represents the utilization percents for each operation. Figure 4 also shows those relations with illustration. From it we can find that addition and multiply are the most frequently used functions in data operations, and *SRL* and *AND* are two functions in logic operations. Therefore, the complicated instruction *MUL_ADD* is adopted to speed up the execution rate.

Then we utilize the software GLPK which contains a free package for the large scale linear programming problems and evaluate our instruction selection algorithm. Here three small instruction libraries are used to do experiment, and the final results are summarized in Table 2. The first column represents selected instruction numbers, total delay, module area and leakage power for selected instructions. The lowest row compares the execution time with the previous method and the proposed method. From the final experimental results we can find that by using the proposed clustering ILP model it achieves a distinct reduction on the execution time.

VI. Conclusion

In this paper, we first formulate the instruction set selection problem with a clustering ILP model and then present a corresponding fast cutting plane algorithm to achieve a fast instruction selection. Finally, through a case study on FIR applications the feasibility of the proposed model is verified.

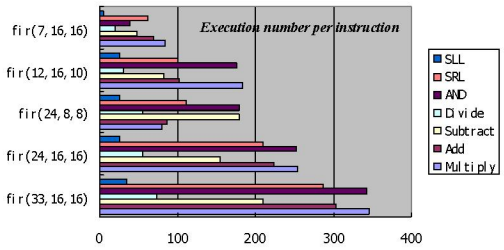

Figure 4. Compare for utilization frequencies of integer instructions

Table 2. The final results with proposed model and method

Instruction libraries		Small Lib.	Bigger Lib.	Extended Lib.
Selected instructions		44 / 129	63 / 180	91 / 261
Total delay		526.300049	909.500000	1239.200195
Total module area		349.000000	583.000000	703.000000
Total leakage power		3539.000000	3138.500000	2328.000000
Execution Time	Previous	3	9	28
	Proposed	< 1	2	7
	Improved	> 66.7%	77.8%	75.0%

Future work will focus on the automated coupling of the complex instructions. This issue will be the emphasis in the future research.

References

[1] P. Biswas, V. Choudhary, K. Atasu, L. Pozzi, P. Ienne and N. Dutt, "Introduction of Local Memory Elements in Instruction Set Extensions", Design Automation Conference (DAC '04), June 2004.

[2] Manoj Kumar Jain, M. Balakrishnan, Anshul Kumar, "ASIP Design Methodologies: Survey and Issues", Proceedings of the 14th International Conference on VLSI Design (VLSID '01), p.76, 2001.

[3] Alomary,A.; Nakata,T.; Honma, Y.; Imai,M.; Hikichi, N. "An ASIP instruction set optimization algorithm with functional module sharing constraint". Proc. ICCAD-93, 7-1, Nov. 1993

[4] J. Lee, K. Choi, N. Dutt, "Automatic Instruction Set Design Through Efficient Instruction Encoding for Application-Specific Processors". TR 02-23, August 8, 2003.

[5] Jason Cong, Yiping Fan, Guoling Han, Zhiru Zhang, "Application-Specific Instruction Generation for Configurable Processor Architectures". Twelfth International Symposium on Field Programmable Gate Arrays, Page 183-189, 2004.

[6] Free open source IP cores and chip design: http://www.opencores.org/projects.cgi/web/cf_fir/overview