High-level Synthesis for DSP Applications using Heterogeneous Functional Units *

Zili Shao[1] Qingfeng Zhuge [1] Chun Xue[1] Bin Xiao[2] Edwin H.-M. Sha[1]

[1]Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083, USA

[2]Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

Abstract— This paper addresses high level synthesis for real-time digital signal processing (DSP) architectures using heterogeneous functional units (FUs). For such special purpose architecture synthesis, an important problem is how to assign a proper FU type to each operation of a DSP application and generate a schedule in such a way that all requirements can be met and the total cost can be minimized. In the paper, we propose a two-phase approach to solve this problem. In the first phase, we propose an algorithm to assign proper FU types to applications such that the total cost can be minimized while the timing constraint is satisfied. In the second phase, based on the assignments obtained in the first phase, we propose *a minimum resource scheduling algorithm* to generate a schedule and a feasible configuration that uses as little resource as possible. The experimental results show that our approach can generate high-performance assignments and schedules with great reduction on total cost compared with the previous work.

I. Introduction

Most previous work on the synthesis of special purpose architectures for real-time DSP applications focuses on the architectures that only use homogeneous FUs (same type of operations will be processed by same type of FUs). With more and more different types of FUs available, same type of operations can be processed by heterogeneous FUs with different costs, where the cost may relate to power, reliability,etc. Therefore, an important problem arises: how to assign a proper FU type to each operation of a DSP application and generate a schedule in such a way that the requirements can be met and the total cost can be minimized. We call this problem *heterogeneous assignment and scheduling problem*.

A ILP (Integer Linear Programming) model that gives optimal solutions for the assignment problem considering heterogeneous functional units is proposed in [2]. However, the exponential run time of the algorithm limits its applicability. In [1], a heuristic approach is proposed to solve heterogeneous assignment and scheduling problem, and it can can produce a solution with one or two orders of magnitude less time compared with the previous ILP model. This approach, however, may not produce a good result in terms of the total cost, since the resource configuration is fixed in the early phase. In the circuit design field, the problem of selecting an implementation of each circuit module from a cell library is studied in [3]. Basic circuit implementation problem is only a special case of the *heterogeneous assignment problem* in which each node must have the same execution time; therefore, their solutions can not be directly applied. In [5], we prove *heterogeneous assignment problem* is NP-complete and propose several algorithms to obtain optimal (when given DFG (Data Flow Graph) is path or tree) or near-optimal solution (for general problem). However, the computation time of the algorithms for general problem is related to the size of the tree extracted from a DFG. So it takes longer time when the tree is larger. Therefore, we design a more efficient algorithm that directly works on DFGs in this paper.

We propose a two-phase approach to solve the problem. In the first phase, we propose an algorithm to solve *heterogeneous assignment problem*. In our algorithm, we first assign each node with the minimal cost type and then iteratively change the type of the node in the critical path such that the timing constraint can be satisfied with the minimal cost increase. In the second phase, based on the obtained assignment, a minimum resource scheduling algorithm is proposed to generate a schedule and a configuration. We experiment with our assignment algorithms on a set of benchmarks. We compare our algorithms with the greedy algorithm in [3] and the ILP model in [2]. The experimental results show that our algorithm gives a reduction of 23.6% on the system cost compared with the greedy algorithm. Our algorithm gives a near-optimal solution with much less time compared with the ILP model. While the given DFGs become too big for the ILP algorithm to solve, our algorithm can still efficiently give results.

The remainder of this paper is organized as follows: In Section II, we give the basic definitions and concepts. The algorithms for *heterogeneous assignment problem* are presented in Section III. The minimum resource scheduling algorithm are presented in Section IV. Experimental results and concluding remarks are provided in Section V and Section VI respectively.

II. Definitions

In our work, Data-Flow Graph (DFG) is used to model a DSP application. A DFG $G = \langle V, E, d \rangle$ is a node-weighted directed graph, where $V = \langle u_1, u_2, \cdots, u_N \rangle$ is the set of nodes, $E \subseteq V \times V$ is the edge set that defines the precedence re-

*This work is partially supported by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, NSF CCR-0309461, USA , and HK POLYU A-PF86 and COMP 4-Z077, HK.

lations among nodes in V, and $d(e)$ represents the number of delays for an edge e. A DFG may contain cycles to model a DSP application with loops. The intra-iteration precedence relation is represented by the edge without delay and the inter-iteration precedence relation is represented by the edge with delays. Given an edge, $e = u \rightarrow v$, $d(e)$ means the data used as inputs in node v are generated by node u $d(e)$ iteration before. A *static* schedule of a cyclic DFG is a repeated pattern of an execution of the corresponding loop. And a static schedule must obey the precedence relations of the *directed acyclic graph (DAG)* portion of a DFG that is obtained by removing all edges with delays from the DFG. In this paper, the DAG part of a DFG is considered when we do assignment and scheduling.

A special purpose architecture consists of different types of FUs. Assume there are M different FU types in a FU library, $P_1, P_2, \cdots, P_M$. $T(u_i)$ is used to represent the execution times of each node $u_i \in V$ for different FU types: $T(u_i)$=$\{t_1(i)$, $t_2(i), \cdots, t_M(i)\}$ where $t_j(i)$ denotes the execution time of u_i for type P_j. $C(u_i)$ is used to represent the execution costs of each node $u_i \in V$ for different FU types: $C(u_i)$=$\{c_1(i), c_2(i)$, $\cdots$, $c_M(i)\}$ where $c_j(i)$ denotes the execution cost of u_i for type P_j. An assignment for a DFG is to assign a FU type to each node. Given an assignment of a DFG, we define *the system cost* to be the summation of execution costs of all nodes because it is easy to explain and useful. Please note that our algorithms presented later will still work with straightforward revisions to deal with any function that computes the total cost such as $\sum_i c_j(i)^2$ as long as the function satisfies "associativity" property.

We define *heterogeneous assignment problem* as follows: Given M different FU types: P_1,P_2,$\cdots$,P_M, a DFG $G = \langle V, E, d\rangle$ where V=$\langle u_1$,u_2,$\cdots$,$u_N\rangle$, $T(u_i) = \{t_1(i), t_2(i), \cdots, t_M(i)\}$ and $C(u_i)$=$\{c_1(i)$, $c_2(i)$, $\cdots$, $c_M(i)\}$ for each node $u_i \in V$, and a timing constraint L, *find an assignment for G such that the system cost is minimized within L.*

III. The Heterogeneous Assignment Algorithm

The heterogeneous assignment problem is NP-complete [5]. In this section, we propose a heuristic algorithms, HAA (Heterogeneous Assignment Algorithm), to solve this problem. The basic idea is first to assign each node with the best cost type and then iteratively change the type of the node in the critical path such that the timing constraint can be satisfied with the minimal cost increase.

In Algorithm HAA, we first assign the best cost type to each node and mark the type to denote that this type has been assigned. We then find a critical path, CP, based on the current assigned types for DFG G, where a critical path is the path that has the maximum execution time among all possible paths. Next, if the execution time of the critical path is greater than TC, the given timing constraint, we reduce it by selecting a node from the critical path and changing its type. The node is selected from the critical path, since only the nodes in the critical path can influence the longest execution time of a DFG. For all other nodes that are not in the critical path, we want to keep their current types since they have been assigned the best types initially.

We select the node in the critical path base on a ratio. Ratio is calculated for each node and its each unmarked type as follows:

$$Ratio \leftarrow IncreasedCost/ReducedTime$$

Given node u_i in the critical path and its unmarked type p, ReducedTime and IncreasedCost are the corresponding reducing time and the increasing cost if u_i is changed from its current assigned type to type p. This ratio is used to represent the average increasing cost per reducing time unit. Since we want to reduce the execution time with the minimal cost increase, we pick up the node with the minimal ratio among all nodes with all possible unmarked types in the critical path. We keep the record of the node and its type that has the minimal ratio during the processing. After the node and the type have been found, we change the node to this type and mark this type as assigned. The procedure is repeated until the timing constraint is satisfied or we can not reduce the execution time of the critical path any more.

Given a DFG G, let $|E|$ and $|V|$ be the number of edges and the number of nodes, respectively. It takes $O(|E|)$ to find a critical path for a given DFG using the clock period computation algorithm in [4]. In our algorithm, in each iteration, it takes at most $O(|V| * M)$ to calculate ratios, select a node from the critical path and change its type, where M is the number of FU types. And the algorithm iterates at most $|V| * M$ times, since each type of a node is only assigned one time. Therefore, the time complexity of our algorithm is $O(|V| * M * (|E| + |V| * M))$. Considering M is a constant, our algorithm takes $O(|V| * |E| + |V|^2)$.

IV. The Minimum Resource Scheduling

In this section, we propose minimum resource scheduling algorithms to generate a schedule and a configuration. *Algorithm Lower_Bound_RC* is used to produce an initial configuration with low bound resource. And *Algorithm Min_RC_Scheduling* is used to refine the initial configuration and generate a schedule to satisfy the timing constraint.

In *Algorithm Lower_Bound_RC*, the total number of FUs for each FU type in each control step is counted in the ASAP and ALAP schedule, respectively. Then the lower bound for each FU type is obtained by the maximum value that is selected from the average resource needed in each time period. Using the lower bound of each FU as an initial configuration, *Algorithm Min_RC_Scheduling* generates a schedule that satisfies the timing constraint and get the finial configuration. In the algorithm, we first compute ALAP(v) for each node v, where ALAP(v) is the schedule step of v in the ALAP schedule. Then we use a revised list scheduling to perform scheduling. In each scheduling step, we first schedule all nodes that have reached to the deadline with additional resource if necessary and then schedule all other nodes as many as possible without increasing resource. Due to limited space, the detailed algorithms are omitted.

V. EXPERIMENTS

In this section, we experiment with our algorithms on a set of benchmarks including 4-stage lattice filter, 8-stage lattice filter, voltera filter, differential equation solver, elliptic filter and RLS-laguerre lattice filter. Three different PE types are used in the system. The execution costs and times are randomly assigned. For each benchmark, the first time constraint we use is the minimum execution time. We compare our HAA algorithm with the greedy algorithm in [3] and the ILP model in [2]. The experiments are performed on a Dell PC with a P4 2.1 G processor and 512 MB memory running Red Hat Linux 7.3. All experiments are finished in less than 1 second.

TC	GHD[3]	ILP[2]	HAA	
	cost	cost	cost	%
4-stage Lattice Filter				
31	286	**191**	**193**	**32.5%**
35	210	**159**	**162**	**22.9%**
40	201	**149**	**151**	**24.9%**
45	192	**140**	**145**	**24.5%**
50	189	**133**	**133**	**29.6%**
8-stage Lattice Filter				
55	490	**325**	**325**	**33.7%**
60	369	**291**	**296**	**19.8%**
65	350	**273**	**296**	**15.4%**
70	340	**252**	**254**	**25.3%**
75	329	**237**	**241**	**26.7%**
Voltera Filter				
37	324	**243**	**245**	**24.4%**
40	324	**214**	**219**	**32.4%**
45	233	**191**	**211**	**9.4%**
50	219	**175**	**177**	**19.2%**
55	211	**155**	**166**	**21.3%**
Differential Equation Solver				
21	132	**111**	**111**	**15.9%**
25	128	**91**	**107**	**16.4%**
30	91	**63**	**76**	**16.5%**
35	74	**54**	**60**	**18.9%**
40	74	**45**	**45**	**39.2%**
RLS-laguerre Lattice Filter				
23	204	**155**	**156**	**23.5%**
25	188	**139**	**146**	**22.3%**
30	156	**117**	**120**	**23.1%**
35	137	**104**	**104**	**24.1%**
40	112	**98**	**100**	**10.7%**
Elliptic Filter				
57	399	**318**	**320**	**19.8%**
60	395	**279**	**297**	**24.8%**
65	371	**247**	**249**	**32.9%**
70	328	**231**	**231**	**29.6%**
75	296	**215**	**215**	**27.4%**

TABLE I
COMPARISON OF THE SYSTEM COSTS FOR VARIOUS BENCHMARKS WHEN THE TIMING CONSTRAINT VARIES.

The experimental results are shown in Table I. In the table, Column "TC" presents the given timing constraint. The system costs are obtained from different algorithms: *Algorithm GHD* (Column "GHD") is the greedy algorithm from [3]; *Algorithm ILP* (Column "ILP") is the ILP model from [2]; and *Algorithm HAA* (Column "HAA") is our assignment algorithm. Columns "%" under "HAA" present the percentage of reduction on system cost compared to *Algorithm GHD*. The experimental results show that our HAA algorithm obtains the near-optimal solution compared with the optimal results obtained by the ILP model. Compared to the greedy algorithm in [3], our algorithm can greatly reduce system cost and achieves an average improvement of 23.6%.

Benchmark	Node Num	TC	ILP		HAA	
			cost	time (s)	cost	time (s)
4-Lat-IIR	78	45	482	**2**	489	**0**
8-Lat-IIR	126	55	1009	**22**	1013	**0.03**
Voltera	81	100	658	**12**	660	**0.02**
Dif.-Eq.-Slover	33	55	263	**1**	265	**0**
RLS-Laguerre	57	80	380	**360**	383	**0**
Elliptic	102	165	-	-	976	**0.05**

TABLE II
COMPARISON OF THE SYSTEM COST AND TIME FOR THE ILP ALGORITHM [2] AND OUR HAA ALGORITHM.

Although the ILP algorithm from [2] can give an optimal solution for heterogeneous assignment problem, it is NP-hard problem to solve the ILP model. Therefore, the ILP model may take very long time to get results even when a given DFG is not very big. We unfold each benchmark three times and perform the tests for the ILP algorithm and our HAA algorithm. The experimental results are shown in Table II. From the experimental results, we can see the ILP algorithm takes much bigger time to get results compared with our HAA algorithm. For the Elliptic filter, it can not give a result after 3 days. Our HAA algorithm can obtain near-optimal solution in all cases in very short time.

VI. CONCLUSION

We have proposed a two-phase approach for real-time digital signal processing applications to perform high-level synthesis of special purpose architectures using heterogeneous functional units. In the first phase, we solved *heterogeneous assignment problem*. In the second phase, we proposed *a minimum resource scheduling algorithm* to generate a schedule and a feasible configuration that uses as little resource as possible. The experimental results show that our approach can generate high-performance assignments and schedules with great reduction on total cost compared with the previous work.

REFERENCES

[1] Y.-N. Chang, C.-Y. Wang, and K. K. Parhi. Loop-list scheduling for heterogeneous functional units. In *6th Great Lakes Symposium on VLSI*, pages 2–7, March 1996.

[2] K. Ito and K. Parhi. Register minimization in cost-optimal synthesis of DSP architecture. In *Proc. of the IEEE VLSI Signal Processing Workshop*, Oct. 1995.

[3] W. N. Li, A. Lim, P. Agarwal, and S. Sahni. On the circuit implementation problem. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 12:1147–1156, Aug. 1993.

[4] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[5] Z. Shao, Q. Zhuge, Y. He, C. Xue, M. Liu and E. H.-M. Sha. Assignment and Scheduling of Real-time DSP Applications for Heterogeneous Functional Units. *18th International Parallel and Distributed Processing Symposium*, CD-ROM Proceeding, Santa Fe, Apr. 2004.