Template Generation and Selection Algorithms

Yuanqing Guo Gerard J.M. Smit Hajo Broersma Paul M. Heysters
University of Twente, Faculty of EEMCS
P.O. Box 217, 7500AE Enschede, The Netherlands
E-mail: {yguo, smit, broersma, heysters}@cs.utwente.nl

Abstract

The availability of high-level design entry tooling is crucial for the viability of any reconfigurable SoC architecture. This paper presents a template generation method to extract functional equivalent structures, i.e. templates, from a control data flow graph. By inspecting the graph the algorithm generates all the possible templates and the corresponding matches. Using unique serial numbers and circle numbers the algorithm can find all distinct templates with multiple outputs. The template selection algorithm shows how this information can be used in compilers for reconfigurable systems. The objective of the template selection algorithm is to find an efficient cover for an application graph with a minimal number of distinct templates and minimal number of matches.

1. Introduction

In the CHAMELEON/GECKO[1]project a heterogeneous system-on-chip (SoC) for handheld multimedia devices is being designed [9]. It contains a general-purpose processor (i.e. an ARM core), a fine-grained reconfigurable part (consisting out of FPGA tiles) and a course-grained reconfigurable part. The latter comprises several MONTIUM processor tiles. The hardware organization within a tile is very regular and resembles a very long instruction word (VLIW) architecture. The five identical arithmetic and logic units (ALU1$\cdots$ALU5) in a tile can exploit spatial concurrency to enhance performance. An ALU has two 16-bit outputs, which are connected to the interconnect. The ALU is entirely combinatorial and consequentially there are no pipeline registers within the ALU. The diagram of the MONTIUM ALU in Fig. 1 identifies two different levels in the ALU.

[1] This research is supported by PROGram for Research on Embedded Systems & Software (PROGRESS) of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affaires and the technology foundation STW.

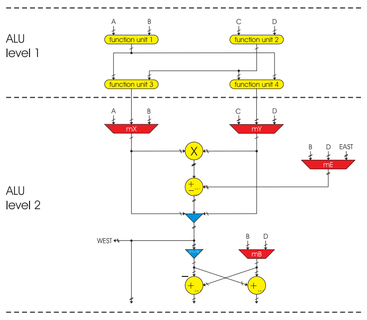

Fig. 1. MONTIUM ALU

The availability of high-level design entry tooling is critical for the viability of any reconfigurable architecture. A C compiler for the MONTIUM architecture is currently being implemented. In our toolset the input language C is first translated into a Control Data Flow Graph (CDFG). Next, the primitive operations are partitioned into clusters, such that each cluster can be executed by an MONTIUM-tile within one clock cycle. After applying graph clustering, scheduling and allocation transformations on this minimized graph, the configurations for the MONTIUM can be generated.

In this paper, we focus on the template generation algorithm. By analyzing the control data flow graphs, our algorithm can indicate which clusters of operations are the most frequently used. This information will guide compilers in selecting suitable configurations. Although, our primary target architecture is the MONTIUM processor, we believe that this technique can also be used for designing logic circuits or field programmable gate arrays (FPGA).

2. Related work

There have been published many related research efforts in the areas of high-level synthesis and FPGA logic synthesis.

In [3][5], a template library is assumed to be available and the template matching is the focus of their work. [1][7] give some methods to generate templates. The drawback of [1][7] is that the generated templates are highly dependent on the choice of the initial template. The heuristic algorithm in [6] generates and maps templates simultaneously, but cannot avoid ill-fated decisions. The algorithms in [2][4] provide all templates of a CDFG. The complete set of tree templates and single-PO (single principle output) templates are generated in [4] and all the single-sink templates are found by the configuration profiling tool in [2]. In this paper, we will present an algorithm that can find the complete set of templates with multiple outputs.

3. Definitions on CDFGs

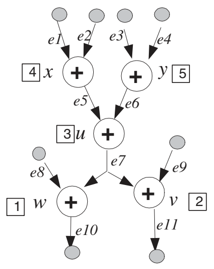

Fig. 2. A small CDFG

A hydragraph $G = (N_G, P_G, A_G)$ consists of two finite non-empty sets of **nodes** N_G and **ports** P_G and a set A_G of so-called **hydra-arcs**; a hydra-arc $a = (t_a, H_a)$ has one **tail** $t_a \in N_G \cup P_G$ and a non-empty set of **heads** $H_a \subset N_G \cup P_G$. In our applications, N_G represents the operations of a CDFG, P_G represents the inputs and outputs of the CDFG, while the hydra-arc (t_a, H_a) either reflects that an input is used by an operation (if $t_a \in P_G$), or that an output of the operation represented by $t_a \in N_G$ is input of the operations represented by H_a, or that this output is just an output of the CDFG (if H_a contains a port of P_G).

See the example in Fig. 2: The operation of each node is a basic computation such as addition (in this case), multiplication, or subtraction. Hydra-arcs are directed from their tail to their heads. Because an operand might be input for more than one operation, a hydra-arc is allowed to have multiple heads although it always has only one tail. The hydra-arc $e7$ in Fig. 2, for instance, has two heads, w and v. The CDFG communicates with external systems through its ports represented by small grey circles in Fig. 2.

Two distinct nodes u and v from N_G are called **neighbors** if there exists a hydra-arc (t, H) with $\{u, v\} \subset H \cup \{t\}$. These nodes are called **connected within a hydragraph** G if there exists a sequence $u_0, \ldots, u_k$ of nodes from N_G such that $u_0 = u$, $u_k = v$, and u_i and u_{i+1} are neighbors for all $i \in \{0, \ldots, k-1\}$. If u and v are connected within G, then the smallest k for which such a sequence exists is called the **distance of u and v within the hydragraph** G, denoted by $\mathrm{Dis}(u, v|G)$; the distance is 0 if $u = v$.

We call u and v are **connected within a subset** $S \subset N_G$, if there exists a sequence $u_0, \ldots, u_k$ of nodes from S such that $u_0 = u$, $u_k = v$, and u_i and u_{i+1} are neighbors for all $i \in \{0, \ldots, k-1\}$. Correspondingly, the **distance within a subset** S is defined, denoted by $\mathrm{Dis}(u, v|S)$.

A subset S of the nodes of a hydragraph is called **connected** if all pairs of distinct elements from S are connected within S. A hydragraph is called **connected** if all pairs of distinct elements from N_G are connected within G, i.e., if N_G is a connected set.

Let $S \subset N_G$ be a non-empty connected set of nodes of the hydragraph G. Then S generates a connected hydragraph in the following natural way:
For every $v \in S$ consider the following two types of hydra-arcs of G related to v:

- (t_v, H_v), so hydra-arcs with tail v: if $H_v \not\subset S$, we introduce a new port p_v and replace (t_v, H_v) by $(t_v, (H_v \cap S) \cup \{p_v\})$; otherwise, we keep (t_v, H_v) as it is.
- (t_u, H_u) with $v \in H_u$, so hydra-arcs for which v is one of the heads: if $t_u \notin S$, we introduce a new port t'_u and replace (t_u, H_u) by $(t'_u, H_u \cap S)$; otherwise we keep (t_u, H_u) as it is.

Doing so for all hydra-arcs, e.g. starting from the sources in S, we obtain a unique hydragraph which we will refer to as the **template** generated by S in G. We denote it by $T_G[S]$ and say that S is a **match** of the template $T_G[S]$. In the sequel we will only consider connected templates without always stating this explicitly. Templates (matches) with i nodes are called **i-templates** (**i-matches**). Correspondingly, a node subset S with i nodes are called **i-node subset**.

A template will be mapped onto one MONTIUM ALU and executed within one clock cycle. The ports of a template represent intermediates that need to be stored for a while during the execution.

For example, in Fig. 3 we see two templates of the CDFG from Fig. 2: the left one is generated by the set $\{x\}$, the right one by $\{v, w\}$. Compared with the original CDFG from Fig. 2, in the left one, the newly added port is a head for hydra-arc $e5$, while in the right one the newly added port is a tail for hydra-arc $e7$.

4. A template generation algorithm

Given a CDFG G, the objective of the template generation algorithm is to find all the i templates with $1 \leq i \leq$ *maxsize* and their corresponding matches from G. The

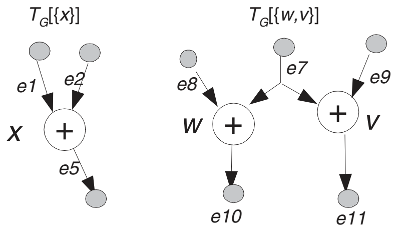

Fig. 3. Two templates of the CDFG from Fig. 2

templates generation procedure is:

1 Starting with the 1-node subsets, generate a set of connected i-node subsets by adding a neighbor node to the $(i-1)$-node subsets.

2 For all i-node subsets, consider their generated i-templates. Choose the set of nonisomorphic i-templates and list all matches of each of them. Templates that cannot be executed by one MONTIUM ALU within one clock cycle are discarded.

3 Repeat the above steps until all templates and matches up to *maxsize* nodes have been generated.

An i-node subsets can be obtained by different $(i-1)$-node subsets, which will result in unnecessarily many computations. To avoid this, we use a clever labelling of the nodes during the generation process:

- Each hydragraph node is given a unique serial number (see the numbers enclosed in small boxes in Fig. 2).

- A leading node is defined within each node subset S, which is the one with the smallest serial number.

- Within a subset S, each graph node $n \in S$ is given a **circle number**, denoted by Cir($n|S$), which is the distance between the leading node and *n* within S, i.e., Cir($n|S$)=Dis(S.LeadingNode, $n|S$).

If a $(i-1)$-node subset S and one of its neighbor node *Nei* satisfy the following conditions, $S' = S\cap$ {*Nei*} will be considered as a i-node subset, otherwise S' is thrown away.

1 S.LeadingNode.Serial<*Nei*.Serial;
2 Dis(S.LeadingNode, *Nei*$|S'$) is not smaller than Cir($n|S$) for any $n \in S$;
3 For every *n* which satisfies $n \in S$ and Cir($n|S$) = Dis(S.LeadingNode,*Nei*$|S'$), *n*.Serial< *Nei*.Serial.

Tab. 1. Multiple copies of a match are filtered out.

1-matches	2-matches	3-matches	4-matches
$\{\boldsymbol{x}\}$	~~$\{\boldsymbol{x},\underline{u}\}$~~ 1		
$\{\boldsymbol{y}\}$	~~$\{\boldsymbol{y},\underline{u}\}$~~ 1		
$\{\boldsymbol{u}\}$	~~$\{\boldsymbol{u},\underline{w}\}$~~ 1		
	~~$\{\boldsymbol{u},\underline{v}\}$~~ 1		
	$\{\boldsymbol{u},\underline{x}\}$	$\{\boldsymbol{u},x,\underline{y}\}$	~~$\{\boldsymbol{u},x,y,\underline{w}\}$~~ 1
			~~$\{\boldsymbol{u},x,y,\underline{v}\}$~~ 1
		~~$\{\boldsymbol{u},x,\underline{w}\}$~~ 1	
		~~$\{\boldsymbol{u},x,\underline{v}\}$~~ 1	
	$\{\boldsymbol{u},\underline{y}\}$	~~$\{\boldsymbol{u},y,\underline{x}\}$~~ 3	
		~~$\{\boldsymbol{u},y,\underline{w}\}$~~ 1	
		~~$\{\boldsymbol{u},y,\underline{v}\}$~~ 1	
$\{\boldsymbol{v}\}$	$\{\boldsymbol{v},\underline{u}\}$	$\{\boldsymbol{v},u,\underline{x}\}$	$\{\boldsymbol{v},u,x,\underline{y}\}$
			~~$\{\boldsymbol{v},u,x,\underline{w}\}$~~ 1
		$\{\boldsymbol{v},u,\underline{y}\}$	~~$\{\boldsymbol{v},u,y,\underline{x}\}$~~ 3
			~~$\{\boldsymbol{v},u,y,\underline{w}\}$~~ 1
		~~$\{\boldsymbol{v},u,\underline{w}\}$~~ 1	
	~~$\{\boldsymbol{v},\underline{w}\}$~~ 1		
$\{\boldsymbol{w}\}$	$\{\boldsymbol{w},\underline{u}\}$	$\{\boldsymbol{w},u,\underline{x}\}$	$\{\boldsymbol{w},u,x,\underline{y}\}$
			~~$\{\boldsymbol{w},u,x,\underline{v}\}$~~ 2
		$\{\boldsymbol{w},u,\underline{y}\}$	~~$\{\boldsymbol{w},u,y,\underline{x}\}$~~ 3
			~~$\{\boldsymbol{w},u,y,\underline{v}\}$~~ 2
		~~$\{\boldsymbol{w},u,\underline{v}\}$~~ 3	
	$\{\boldsymbol{w},\underline{v}\}$	$\{\boldsymbol{w},v,\underline{u}\}$	$\{\boldsymbol{w},v,u,\underline{x}\}$
			$\{\boldsymbol{w},v,u,\underline{y}\}$

In Table 1, the procedure of finding all the i-node subsets from $(i-1)$-node subsets of Fig. 2 is given. The symbols in bold are the names of the leading nodes and the newly added nodes are underlined. The node subsets that do not satisfy the conditions are discarded. The numbers next to the discarded node subsets indicate which of the above three conditions is violated.

4.1. A template selection algorithm

Given a CDFG G and a set of templates $\Omega = \{T_1, T_2, \cdots, T_p\}$, this template selection algorithm is to find a cover $C(G,\Omega) = \{S_1, S_2, \cdots, S_n\}$ of G using $m(\leq p)$ templates, such that the number of distinct templates m and the total number of subgraphs n are minimized.

Since the generated set of templates and matches can be quite large, the template and match selection problem is computational intensive. We adopt a heuristic to do so [6][7].

For each template T, an objective function is defined as:

$$g(T) = g(w, s).$$

where, w is the number of nodes of the template, s is the number of non-overlapped matches for the template.

The heuristic approach at each round selects a template T_i with the maximum objective function. The matches corresponding to the selected templates are selected matches. The procedure goes until the selected matches can cover the whole CDFG G.

5. Experiments

We used the template generation and selection algorithms on the CDFG of 4-point FFT shown in Fig. 4. The objective function is $g(T) = g(w, s) = w^{1.2} \cdot s$. Among all the templates, ❶❷ have the highest weight function and then ❸. The graph is completely covered by them. This result is the same as our manual solution. The same templates are chosen for n-point FFT ($n = 2^d$).

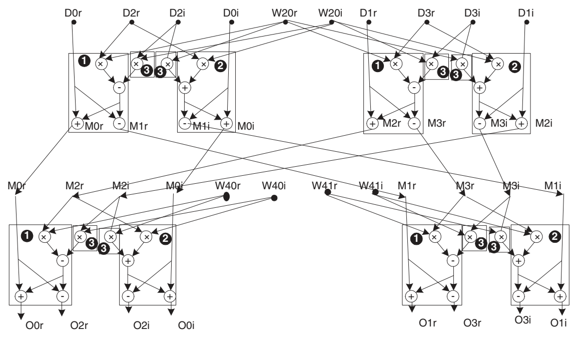

Fig. 4. The CDFG for 4-bit FFT

6. Conclusion

The central problem for template generation algorithms is how to generate and count all the subgraphs of a CDFG. The methods employed in [4] and [2] can only count the subgraphs of specific shapes (tree shape, single output or single sink) and as a result, templates with multiple outputs cannot be generated. In the MONTIUM architecture, each ALU has three outputs, so these algorithms cannot be used.

The major contributions of this paper are the algorithms developed to find all possible templates and the corresponding matches. The template generated are not limited by the shape of templates, i.e. the templates with multiple outputs or multiple sinks [2]. This algorithm is applicable both to control data flow graphs and general netlists in circuit design.

Using the templates and corresponding matches, an efficient cover for the graph can be found, which tries to minimize the number of distinct templates that are used while minimizing the number of instances of each template.

References

[1] Srinivasa R. Arikati, Ravi Varadarajan, "A Signature Based approach to Regularity Extraction", *Proc. of Internaltional Conference on Computer-Aided Design (ICCAD)*, 1997, pp.542-545.

[2] Srihari Cadambi, and Seth Copen Goldstein, "CPR: A Configuration Profiling Tool", *IEEE Symposium on FPGAs for Custom Computing Machines,* 1999.

[3] Timothy J.Callahan, Philip Chong, Andre DeHon, and John Wawrzynek, "Fast Module Mapping and Placement for Datapaths in FPGAs", *Proc. of International Sysp. of Field Programmable Gate Arrays,* 1998.

[4] Amit Chowdhary, Sudhakar Kale, Phani Saripella, Naresh Sehgal, Rajesh Gupta, "A General Approach for Regularity Extraction in Datapath Circuits", *Proc. of Internaltional Conference on Computer-Aided Design (ICCAD)* San Jose, CA, 1998, pp.332-338.

[5] Miguel R. Corazao, Marwan A. Khalaf, Lisa M.Guerra, Miodrag Potkonjak and Jan M. Rabaey, "Performance Optimization Using Templete mapping for Datapath-Intensive High-Level Synthesis", *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems*, vol.15, No.8, August 1996, pp.877-888.

[6] Ryan Kastner, Seda Ogrenci-Memik, Elaheh Bozorgzadeh and Majid Sarrafzadeh, "Instruction Generation for Hybrid Reconfigurable Systems", *Proc. of International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, November, 2001.

[7] D. Sreenivasa Rao, and Fadi J. Kurdahi, "On Clustering For Maximal Regularity Extraction", *IEEE Transactions on Computer-Aided Design*, vol.12, No.8,August,1993, pp.1198-1208.

[8] Michel A.J. Rosien, Yuanqing Guo, Gerard J.M. Smit, Thijs Krol, "Mapping Applications to an FPFA Tile", *Proc. of Date03*, Munich, March, 2003

[9] Gerard J.M. Smit, Paul J.M. Havinga, Lodewijk T. Smit, Paul M. Heysters, Michel A.J. Rosien, "Dynamic Reconfiguration in Mobile Systems", *Proc. of FPL2002,* Montpellier France, pp 171-181, September 2002.