A Heterogeneous Dependency Graph as Intermediate Representation for Instruction Set Customization

Kang Zhao, Jinian Bian, Sheqin Dong
EDA Lab, Dept. of Computer Science & Technology, Tsinghua Univ., Beijing 100084, China, +86-10-62785564

Abstract

A heterogeneous dependency graph (HDG) defined as the intermediate representation for instruction set automated customization is presented in this paper. The main motivation of this model is to construct a unified internal specification that bridges the gap from the application benchmarks to the instruction set customization. To represent the necessary information required by the instruction set customization process, basic elements and heterogeneous structures including pipelining, parallel, branch and loop are presented in detail.

Keywords: Heterogeneous dependency graph, HDG, design automation, instruction set customization, ASIP.

1. Introduction

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 [1]. To face the challenges of high efficiency and time-to-market pressure, automatic customization of ASIP to the application requirements has become an attractive technology [2].

In ASIP synthesis, instruction set customization plays a kernel role, which consists of the generation of whole instruction set and corresponding assembly code. In order to find an efficient solution for the instruction set customization, designers must bridge the gap with an intermediate representation which can satisfy the requirements of the customization process.

There are many candidate representations which have been widely used. Control and Data Flow Graph (CDFG) has been widely used in high level synthesis [3] [4]. As an intermediate representation, CDFG performs the conversion of the behavior description into a single graph used by the scheduling and allocation algorithm in synthesis. CDFG Toolkit [5] is even produced by Design Automation Lab in Seoul National University. Furthermore, a powerful three layered CDFG including DFG, CFG and Hierarchical Task Graphs have been presented in SPARK [6], which resolves the existing problem that abstraction level is too low for the range of coarse-grain and fine-grain parallelizing compiler transformations.

However, there are two intrinsic limitations in the specifications mentioned above. First, main structures in those models are sequential, which cannot satisfy the parallel optimizations in the instruction set customization. Second, those models focus on certain application problems and their specific definitions are not suitable for instruction set customization.

To address these issues, we present a novel model named as Heterogeneous Dependency Graph (HDG), which consists of four heterogeneous structures, pipelining, parallel, branch and loop. As the unified intermediate representation, HDG satisfies the basic requirements of three sub-processes of instruction set customization, basic instruction set selection, complex instruction generation and assembly code generation.

The rest of the paper is organized as follows. In Section 2, the instruction set customization framework and corresponding requirements for the intermediate representation are presented. Then the basic elements of HDG are defined in Section 3. Section 4 presents the four main heterogeneous structures in detail. To understand it deeply, an experimental example for HDG is presented in Section 5. Finally, an analysis compared with CDFG is done to support our proposed model HDG. The last section draws the conclusion.

2. Application Requirements

Instruction set customization is a process which gradually moves the design to lower levels, as shown in Figure 1. It starts with the behavior specification of the application benchmarks written with high level languages. During instruction set customization, basic instructions are selected from a pre-designed library and complex instructions are generated by profiling.
is very necessary in the intermediate representation. To find an efficient solution for instruction set customization, those data dependency information must be represented with a proper format. Therefore, the intermediate representation should also be able to represent those heterogeneous structures.

From the analysis of the application requirements above, we present a novel model as the intermediate representation which named as HDG. In the following sections we will give the detailed definitions of HDG.

3. Basic Elements of HDG

As the unified representation for the instruction set customization, HDG consists of two basic elements, node and flow. It is defined as flows:

HDG = <V, E>

V = \{v_1, v_2, \ldots, v_m\}

E = \{e_1, e_2, \ldots, e_n\}

Where V is the set of nodes, which represent the utilized operations, states, variables, or tasks. E is the set of flows, which represent a data flow or a control token’s transmission. And to represent the hierarchical feature, we add an option to the parameters in V:

v = <id, type, in_flows, out_flows, father, child>

where father is a pointer which links to the hierarchical node v belongs to, and child points to the set of nodes which are captured in v.

V = ON \cup TN \cup CN

CN = SN \cup MN \cup FN \cup JN \cup CLN \cup EN

E = CF \cup DF

The nodes in HDG consist of three types: operation node (ON), transport node (TN) and control node (CN). ON stands for one operation or task, TN represents an event which results in a transport of data, and CN is the symbol of special structures. In addition, CN consists of select node (SN), merge node (MN), fork node (FN), join node (JN), clear node (CLN) and encase node (EN). The representations of these control nodes are shown in Figure 4. Furthermore, CF and DF represent the control flow and data flow respectively.

The difference of SN and FN is that SN sends the control token to only one output flow and FN sends...
the same token to each output flow. Similarly, MN is fired only if one token from input flows arrives, but JN waits until all the tokens from input flows arrive. CLN and EN are mainly used in pipelining structures.

4. Heterogeneous Structures

In order to represent the features for the instruction set customization, special structures in HDG are presented, including pipelining and parallel supporting assembly code optimization, and branch and loop supporting basic structures in the benchmark.

4.1. Pipelining

Pipelining is a special structure of parallel execution which is widely used to represent the characters of hardware. Recently effective hardware generation to deal with complex loops requires the ability to exploit the pipelining. Furthermore, pipelining also has an important effect on power dissipation, which has been validated by Actel Corporation [7].

HDG also supports the specification of the data storage when pipelining. Dummy registers in TNS are defined to fulfill this task. Between the pipeline stages, additional dummy registers are added into TNS to store variables in different stages. As the example shown in Figure 5(c), variable g1 stores data from operation A1 to A3, whereas g2 and g3 from operation A1 to A2, and A2 to A3 respectively. Three dummy registers in g1 and two dummy registers in g2 and g3 ensure that g1 transfer data in the second pipelining period from A1 to A2 and in the third pipelining period from A1 to A3.

4.2. Parallel

Parallel is a simple but widely used structure, which represents the feature of hardware design very well. Many input languages support this structure, such as VHDL, Verilog, SpecC, SystemC. Furthermore, in the hardware design level, parallel computation and data transmission is connatural.

Parallel structures are represented mainly with fork and join nodes in HDG. An example is shown in Figure 6(a), where A1, A2 and A3 run in parallel. They all start simultaneously when the fork node is started up. Once all of them have completed their execution, the token will be route to the join node.

4.3. Branch

Branch is one of the most widely used structures. A majority of input languages have this structure, in spite of software languages and hardware languages such as C and VHDL.

In HDG, branch structure begins with a select node and ends with the corresponding merge node, as illustrated by Figure 6(b). In this definition, the condition estimation part is not contained in the branch structure; instead, it only communicates with the branch by the data input flow of the select node, which controls the directions in the branch.

4.4. Loop

Loop structure is also a familiar structure in high level languages, which contain while and do-while two types. The graph representation for loop is shown in Figure 6(c) (d). It begins with a merge node and ends with the corresponding select node. The difference between while type and do-while type is that the positions of condition estimation node and loop body are just opposite.

```c
(1) for (j=0; j<hight; j++)
(2) offs=(width*(j<<1)+j0)+i0;
(3) for(i=0; i<width; i++)
(4) offs++;
(5) blk0[(16+i]=cur[offs]-pred[offs];
(6) blk1[(16+i]=cur[offs+width]-pred[offs+width];
(7) }
(8) }
```

Fig. 7: A section of MPEG as an example.
5. An Experimental Example

To understand the novel model deeply, we choose one section of MPEG as an example, as shown in Figure 7. Because the computation for the motion picture is very large, many optimized process should be done. Several guidelines should be followed when turning MPEG to HDG: (1) All possible parallelism operations should be exposed explicitly using the concurrent-execution constructs in HDG, such as parallel and pipelining. (2) Hierarchical nodes should be chosen with appropriate size and utilized to group related leaf operations, which are the smallest indivisible units in HDG.

![HDG representation for the MPEG example.](image)

Following the two guidelines mentioned above, we can find that the fourth statement is staggered from the fifth and sixth statements and they can be implemented with pipelining. Furthermore, the fifth and sixth statements also can be operated in parallel because there are no relations between them. The corresponding specification with HDG is shown in Figure 8. There are three hierarchical nodes and three special structures, pipelining, parallel and loop. If we adopt CDFG or other models, this character cannot be expressed. In this figure, the real lines stand for control flows and broken lines stand for data flows.

6. Advantage compared with CDFG

CDFG is a widely used model which has many same characters with HDG. For example, they are both composed with nodes and flows, and data dependency and control dependency can be represented both in these two representations.

However, there are two advantages in HDG compared with CDFG to supply necessary information for the instruction set customization:

1. HDG has pipelining and parallel structures which support the assembly code optimization process. But this heterogeneous feature does not exist in CDFG. CDFG can only represent the nonparallel structures.

2. For the generation of complex instruction set generation, the intermediate specification should be able to represent the hierarchical character. HDG solves this problem very well. As the example shown in section 5, subset of nodes can be captured into a hierarchical node. However, CDFG can only represent the fine-granularity level graph.

From the compare above, we can find that HDG is a more appropriate choice than CDFG for representing the instruction set customization process.

7. Conclusion

In this paper, a novel intermediate representation named as HDG is presented to bridge the gap between application benchmarks and the process of instruction set customization. This specification is composed of nodes, flows and ports, which have four heterogeneous structures, pipelining, parallel, branch and loop. Those definitions ensure that it satisfies the requirements of instruction set customization.

Future work will focus on the automated conversion from the benchmarks written in high level languages to this intermediate representation.

8. References


