Trigger-Centric Loop Mapping on CGRAs

Abstract:
A coarse-grained reconfigurable architecture (CGRA) is a promising platform based on considerations for both performance and power efficiency. One of the primary obstacles that CGRAs might face is how to accelerate loops with if–then–else (ITE) structures. A recent control paradigm for CGRAs named triggered instruction architecture (TIA) can provide an efficient scheme to accelerate loops with ITE structures. Yet common loop mapping frameworks cannot leverage this scheme autonomously. To this end, this brief makes two contributions: 1) identify and remove redundancy nodes from a data flow graph and 2) propose an integrated approach—TRMap, which consists of operations merging, Boolean operations offloading, and transformation of triggers. Our experimental results from some vital kernels extracted from SPEC2006 benchmarks and digital signal processing applications show that by using TIA scheme, TRMap is able to accelerate loops with ITE structures to an execution that is 1.38× and 1.64× faster than that achieved by a full predication scheme (FP-Choi) and a state-of-the-art method (BRMap). The proposed architecture of this paper analysis the logic size, area and power consumption using Xilinx 14.2.

Enhancement of the project:

Existing System:
Loop mapping is one of the major challenges associated with CGRA, especially those loops with if–then–else (ITE) structures. The importance of accelerating loops with ITE structures. Software pipelining is widely used in loop mapping onto CGRA and its goal is to minimize the initiation interval (II)—the number of cycles between the start of two consecutive iterations of a loop. II has the smallest possible value called the minimum II (MII). The MII is max (RecMII and ResMII), where RecMII and ResMII are the MIIs determined, respectively, by recurrences in the loop and by the available hardware resources. There are two major problems in mapping the loops with ITE structures. First, the branch paths in ITE structures carry a lot of operations, which occupy many PEs. Second, since these operations are guarded by the control-flow decisions calculated from Boolean operations (AND, OR, and NOT), many PEs are also
occupied to execute these Boolean operations on their ALUs. These two problems degrade the utilization ratio of PEs, resulting in a large II. For the first problem, merging operations in branch paths can be a good choice. The state-of-the-art method BRM leverages this merging method and shows better performance than the full predication scheme (FP-Choi). To tackle the second problem, we can offload the Boolean computation of the control-flow predicates from the PEs onto the schedulers. Since the schedulers offer possibilities for transforming the Boolean operations to triggers. We define redundancy nodes to be the Boolean operations that alter the control flow in a data flow graph (DFG). They can be executed on the scheduler and then be removed from the DFG. However, besides those Boolean operations for control-flow decisions, there are also some other Boolean operations for computing logic values which cannot be executed on the schedulers. Therefore, we need to identify the redundancy nodes in the DFG and guarantee that only the redundancy nodes are removed.

Disadvantages:

- Loop mapping is worse

Proposed System:

Basic Idea

To improve the performance of mapping loops with ITE structures onto the TIA, our approach contains three procedures:

1) Merging operations from branch paths;
2) Removing redundancy nodes from the DFG;
3) Trigger allocation and assignment.

In particular, the operations that update the same variable from branch paths must be mapped to the same PE, since only one of the paths is taken at runtime.
Fig. 1. (a) TDFG after eliminating the Boolean operator P from the DFG since P's source operands are predicates P1 and P2. (b) Valid mapping of the TDFG with II of 5.

TRMap

Based on the idea above, we propose a mapping flow for TIA named TRMap, as shown in Fig. 2. First, TRMap inputs a control flow graph (CFG) and employs a front-end technique called hyperblock to construct the DFG from multiple basic blocks of the CFG through SSA transformation. Next, our heuristic of legal merging, valid elimination, and trigger transformation (MET) is applied to minimize the DFG and get the triggered DFG (TDFG). Finally, TRMap employs modulo scheduling-based placement and routing (P&R) algorithm, e.g. REGIMap, to map the TDFG onto the TIA.
Algorithm and Time Complexity

Our MET algorithm is shown in Algorithm 1. MET starts with operations merging. m and n represents the number of branch paths and the number of the operations in the longest branch path, respectively. Operations in the j-path are stored in the set Vj (Vj ⊂ VP). MET merges multiple operations that update the same variable from n branch paths to a merged node v. If the operation from j-path is NULL, a nop operation would be put into v. All merged nodes are put into VM from our formulation and then the select operations are eliminated. After operations merging, MET starts eliminating redundancy nodes.

Trigger transformation contains two steps:

1) dependence conversion

2) control-flow decisions computation.

We present the process of trigger transformation by generating the triggered instructions in Fig. 3 from the TDFG in Fig. 1(a). The first step is used to convert the data dependence into predicate dependence. It first assigns a predicate to each node of the DFG (lines 27 and 28 in Algorithm 1), such as a0, b0, and c0 in lines 1, 3, and 5, respectively, in Fig. 3. These predicates are initialized to false and turned into true in an implicit way until their predicated operations are performed (lines 2, 4, 6, and so on in Fig. 3). Once the output of the source instruction is taken out by the last destination instruction, they must be reset (lines 8, 10, and 12 in Fig. 3). If current node has data dependence, a predicate is assigned to this node and conjuncted with existing predicates (lines 29–31 in Algorithm 1). This process is repeated iteratively until no data dependence exists. The second step (lines 32–35 in Algorithm 1) computes the predicate expressions Vr stored by the elimination procedure to get the control-flow decisions that form the triggers (lines 17, 19, 21, and 23 in Fig. 3).
Fig. 3. Triggered instructions.

Advantages:

- better loop mapping

Software implementation:

- Modelsim
- Xilinx ISE