Extended Distributed Control for Dynamic Scheduling across Dataflow Graphs

Wakako NAKANO  Nagisa ISHIURA
School of Science and Technology, Kwansei Gakuin University
2-1 Gakuen, Sanda, Hyogo, 669–1337, Japan

I. INTRODUCTION

While the scale and the complexity of the hardware implemented in a chip are growing rapidly, there is a strong demand to reduce time to market. High-level synthesis is considered to be one of the effective means to expedite hardware design.

In traditional high-level synthesis methods, operations are scheduled statically assuming that functional units take the same number of clock cycles for the same operations. However, in actual datapaths, functional units may exhibit different latencies for the same operations. There had been some attempts to adjust operation scheduling in runtime [1], a finite state machine (FSM) to control the datapath would end up with enormous amount of states. On the other hand, a distributed control scheme, where the datapath is controlled by multiple FSMs, enables run-time adjustment of operation scheduling with reasonable hardware cost. Different schemes for distributed controllers have been proposed by Del Barrio [2], Pilato [3], etc.

While these methods originally dealt with a case where computation is expressed with a single DFG, Shimizu extended the Del Barrio’s distributed control to handle multiple DFGs [4]. This enables dynamic operation motion across dataflow graphs (DFGs) which brought about the same effect as trace/loop scheduling. However, it posed a restriction that only two DFGs may be executed in parallel.

To address this issue, we propose an improved formulation of the distributed control where more than two DFGs can be executed in parallel. A preliminary experiment demonstrates that execution cycles can be reduced by this extension.

II. VARIABLE LATENCY UNITS AND DISTRIBUTED CONTROL

Functional units in datapaths may exhibit different latencies for the same operation, depending on operand values, states of the units, etc. Memories and iteration-based multipliers/dividers are such examples. In traditional high-level synthesis where operations are scheduled statically (Fig. 1 (a)), operation 3 cannot start early even when operation 2 finishes in a cycle (Fig. 1 (b)).

In the Del Barrio’s method [2], dynamic adjustment of scheduling (Fig. 1 (c)) is realized by controlling units with separate FSMs. For example, in Fig. 1 (d), FSM_A and FSM_M control units A and M, respectively. Unit u starts execution when FSM_u sets en_u = 0, and notifies the completion by setting end_u = 1. Each operation i is assigned a state S_i. At S_i, the FSM waits for start_i = 1, which means that all the operations i depends on are finished. For example, at S_3, \text{start}_3 = (\text{start}_2 \lor \text{end}_M) \lor \text{Done}_2, where \text{Done}_2 expresses the completion of operation i. After setting en_u = 1, FSM_u waits for the completion signal \text{end}_u, and then moves to the next state.

Shimizu extended the Del Barrio’s method to handle multiple DFGs connected with conditional and unconditional transitions [4], which enables dynamic operation motion across DFGs. For example, in the benchmark in Fig. 2 (a), A2 can execute operations 1 in DFG0 of the next iteration, while operations 12, 11, and d2 in DFG2 are still under execution.

However, the formulation in [4] posed a restriction that only one DFG ahead of the current DFG may be executed, i.e. at most two DFGs may be executed at the same time. A self loop is eliminated by duplicating the DFG (DFG0’ in Fig. 2 (a) is a copy of DFG0). It also imposed that at least one state must exist per unit per DFG. States d0, d0’, d1, and d2 are dummy states inserted to satisfy this condition.

III. EXTENDED DISTRIBUTED CONTROL

A. Overview

There are cases where the “2 DFG restriction” in [4] limits the performance of resulting circuits. For example, in a exe-
cution trace in Fig. 2 (b). A2 could execute operations 1 and 2 just after d2, but A2 must wait for the completion of operation 10 to start operation 1, due to the 2 DFG restriction.

In this paper, the formulation is updated so that operations in more than 2 DFGs can be executed at the same time. With this formulation, the trace in Fig. 2 (b) will be improved to (c).

### B. Formulation

In the previous formulation, state transition was controlled by defining a main DFG (currently being executed) and a frontier DFG (one DFG ahead of the main DFG) as illustrated in Fig. 2 (d). Transition into a state is allowed only if it is in the main or frontier DFGs.

In this paper, new predicates `active` and `head` are introduced. Intuitively, as illustrated in Fig. 2 (e), `active(d)` means that some operation in DFG `d` is being executed, and `head(d)` means that DFG `d` is the forefront of the active DFGs. Let `last(d)` means that all the operations in the DFG `d` finishes in the current cycle. Then, transition from a state in DFG `d` to a state in DFG `d'` is allowed only if transition from `d` to `d'` is fixed, and

\[
\text{head}(d) \quad \text{or} \quad \text{active}(d') \quad \text{or} \quad \text{last}(d').
\]

This allows execution of DFGs as long as the head DFG does not overrun the “tail” of the active DFGs.

Predicates `active(d)` and `head(d)` are defined operationally. Initially, `active(d) = head(d) = 0` for all DFGs `d` and `active(d0) = head(d0) = 1` for the start DFG `d0`. `active(d)` is set when there is a transition from a state in some DFG `dp`, to a state in `d`. When `last(d) = 1`, `active(d)` is reset in the next cycle. When there is a transition from a state in some DFG `dp`, to a state in `d` where `head(dp) = 1`, `head(d)` is set and `head(dp)` is reset.

### IV. Experimental Results

Preliminary experiment has been conducted by manually designing RTL circuits in Verilog HDL for the same benchmarks as in [4], and synthesizing them by Xilinx Vivado (2016.4) targeting Artix-7 (xc7a100tcsg324-3). It is assumed that multiplication takes 1 or 2 cycles depending on operands, and all the other operations take 1 cycle. TABLE I summarizes the results. Rows “DC” and “XDC” are for the previous [4] and the proposed methods, respectively. “#cycle” are the number of execution cycles for 128 iteration, where `r` is the probability that the multiplication takes two cycles. We can observe substantial reduction of execution cycles on bicubic. On the other hand, there is no change on m-lerp in which there is no chance of executing more than 3 DFGs, because of loop carried data dependence across DFGs. “#LUT” (LUT count) and “delay” (critical path delay) are almost the same as the previous method.

### V. Conclusion

This paper has proposed a new formulation of the distributed control in which operations in more than two DFGs may be executed in parallel. A preliminary experiment shows that our new method can reduce execution cycles when data dependency does not prohibit parallel execution of multiple DFGs. We are now working on automatic RTL generation from CDFG and scheduling/binding based on this formulation.

**Acknowledgements**—Authors would like to express their appreciation to Dr. H. Kanbara (ASTEM/RI), Prof. H. Tomiyama (Ritsumeikan Univ.), and Mr. T. Nakatani (formerly with Ritsumeikan Univ.) for their valuable comments. We would also like to thank to the members of Ishiura Lab. of Kwansei Gakuin Univ. for their cooperation. This work was partly supported by JSPS KAKENHI Grant #16K00088.

**References**


