Synthesis of Distributed Control Circuits for Dynamic Scheduling across Multiple Dataflow Graphs

Sayuri Ota Nagisa Ishiura

School of Science and Technology, Kwansei Gakuin University
2-1 Gakuen, Sanda, Hyogo, 669-1337, Japan

***Abstract*—This article presents a method for synthesizing circuits with distributed control from CDFGs (control dataflow graphs). The distributed control attempts to harness a datapath with multiple FSMs (finite state machines) to adjust execution timing of operations dynamically, by which wasteful waits caused by variable latency units are reduced. Although Shimizu and Nakano proposed distributed control schemes which allowed dynamic scheduling across multiple DFGs (dataflow graphs), they just presented example controllers which were manually designed. This article shows a formulation to make the multiple FSMs work in ensemble based on the Nakano's scheme, along with some restrictions on CDFGs to allow automatic synthesis. Experiments on two CDFGs with various bindings demonstrates that the execution cycles are reduced by 8.5% in the best case at the cost of 13% increase on the circuit size on the average.**

I. Introduction

As the scale and the complexity of the hardware implementable on VLSIs grow, more and more efforts for designing such circuits are needed. Moreover, there are strong demands to reduce time to market. High-level synthesis [1] 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 depending on operands or their environment. With the static scheduling, wasteful waits occur when the units take less latencies than in the worst case.

In some early attempts of runtime operation scheduling [2] a finite state machine (FSM) to control the datapath would end up with enormous amount of states. However, a distributed control scheme, where the datapath is controlled by multiple FSMs, enables runtime scheduling with reasonable amount of hardware. Different schemes for distributed controllers have been proposed by Del Barrio [3], Pilato [4], etc.

While these methods originally dealt with a case where computation was expressed with a single DFG, Shimizu extended the Del Barrio's distributed control to handle multiple DFGs [5]. This enables dynamic operation motion across dataflow graphs (DFGs) which brought about the same effect as trace/loop scheduling. While Shimizu's formulation allowed only two DFGs to be executed in parallel [5], Nakano extended the control scheme so that more than three DFGs may be executed simultaneously [6]. However, the effects of these control methods have been demonstrated based on manual design for only a limited number of instances.

In this paper, a method for automatically generating RTL designs based on the Nakano's distributed control scheme from given CDFGs. The state transition scheme and the logic for the necessary signals are formulated. Especially, the logic to control data dependency across DFGs are derived based on reachability analysis of DFGs in a given CDFG.

Experiments on two CDFGs with various bindings demonstrates that the execution cycles are reduced by 8.5% in the best case. The synthesized circuits are on average about 13% larger than those based on conventional centralized controllers, while the critical path delay stays the same.

II. Distributed control

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

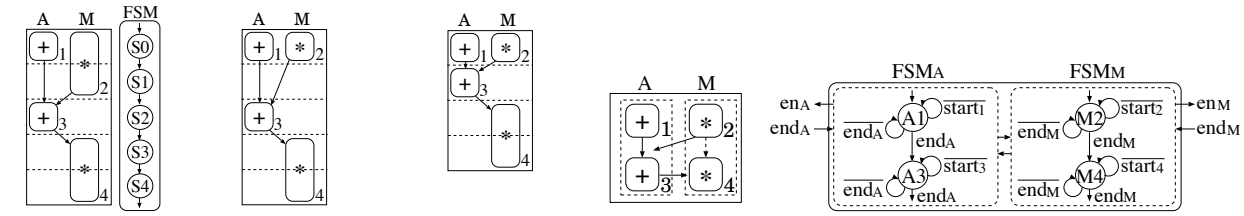
In the Del Barrio's method [3], dynamic adjustment of scheduling (Fig. 1 (c)) is realized by controlling units with separate FSMs. For example, in Fig. 1 (d), FSM_A and FSM_M control units A and M, respectively. Unit u start execution when FSM_u sets $\mathrm{en}_u = 0$, and notifies the completion by setting $\mathrm{end}_u = 1$. Each operation i is assigned a state S_i. FSM F_u waits for $start_i = 1$ at S_i, which means that all the operations that i depends on are finished. FSM_u waits for the completion signal end_u, before it moves to the next state.

Shimizu extended the Del Barrio's method to handle multiple DFGs connected with conditional and unconditional transitions [5], which was further extended by Nakano [6]. For example, in a CDFG in Fig. 2 (a) suppose DFG1, DFG2, and DFG0 are executed in this order. In Nakano's formulation, as shown in (b), operations in DFG2 and even those in DFG0 may start execution before the other operations in DFG1 are finished. This enables dynamic operation motion across DFGs, which may drastically reduce total execution cycles under the existence of variable latency operations.

III. Automatic synthesis of distributed controller

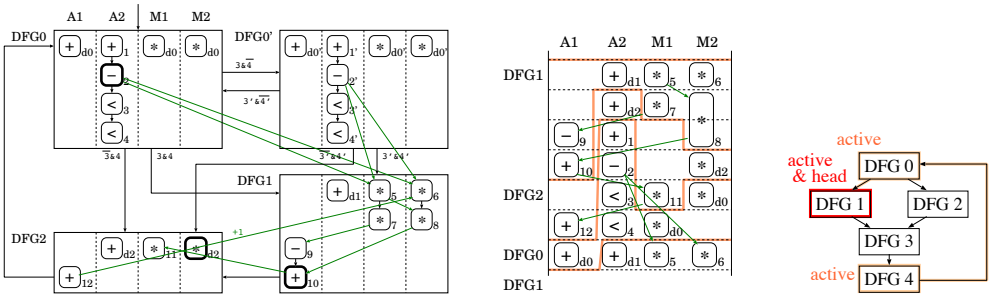
A. Control scheme

Fig. 3 shows an example CDFG handled in this paper. A CDFG consists of DFGs (d0, d1, d2) and transitions among DFGs depicted by arrows whose



(a) static scheduling (b) early completion (c) dynamic scheduling (d) Del Barrio's distributed control

Fig. 1. Handling of variable latency operations [6].



(a) CDFG of "bicubic" (b) trace (c) active-head scheme

Fig. 2. Extended distributed control on benchmark "biucbic" [6].

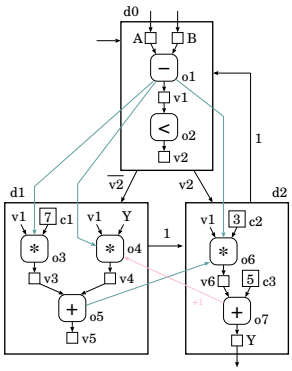


Fig. 3. CDFG (control dataflow graph)

labels indicate the transition conditions. Each DFG consists of operations, values, and dependency among them (depicted by circles, squares, and arrows, respectively). Dependency may be defined across DFGs (green arrows) and across iterations (an orange arrow).

In this paper, we handle a problem of generating RTL designs from a given CDFG, a datapath configuration, scheduling, and binding (assignment of the operations and the values to the functional units and the registers, respectively). In classical high-level synthesis, scheduling refers to assignment of the operations in each DFG to exact cycle steps. In our distributed control, however, since the execution timing of operations are dynamically adjusted, scheduling just defines the order of the operations executed by each functional unit.

We assume in this paper the following restrictions for input CDFGs, to reduce difficulty in inter-DFG dependency control.

1) Only single level loops are assumed; namely, loops should not be nested.
2) Loops may contain conditional branches and each loop may end at different DFGs, but must begin at an identical DFG.
3) The conditions for branches from a DFG must be computed in the DFG.
4) In given scheduling and binding, each functional unit executes at least one operation in each DFG. If it is not the case, dummy states (in which no operation is executed) are inserted to meet the condition.

(1) FSMs

In the distributed control scheme in this paper, each functional unit u in datapath is ruled by a dedicated FSM F_u. Each state of F_u is in charge of controlling execution of individual operation o by u. F_u waits at s_o until o is ready to execute (all the operands have been computed), and then sends the start signal to u. F_u remains at s_o until it receives the completion signal from u, and then transitions to the next state. In this paper we call the state s_o is in execution when the operation o is in execution. When unit u is scheduled to execute $o_0, o_1, \cdots, o_k$ in this order in a DFG, then FSM F_u transitions s_{o_0} through s_{o_k}.

(2) Completion flags and readiness conditions

The readiness of the operations is handled by providing completion flag $E(s_o)$ for each state s_o of operation o, whose value is determined as follows:

- Initially $E(s_o) = 0$.
- $E(s_o)$ is set (to 1) as soon as the operation o is finished.
- $E(s_o)$ is reset (to 0) as soon as all the operations in the DFG that o belongs to are finished.

Signal $r(s_o)$ represents that the operation o is ready for execution, whose value is defined as:

$r(s_o) = 1$ iff $E(s_p) = 1$ for every operation p that o depends on.

(3) Transition to the next DFG

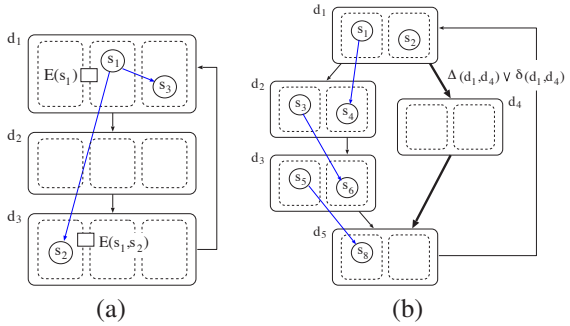


Fig. 4. Inter-DFG dependency

FSM F_u transitions from the last state in a DFG d to the first state in DFG d' only if the following two conditions are met.

1) Branch condition from d to d' is established.
2) While a DFG is being executed for a certain iteration, the same DFG must not be executed for the next iteration.

The first condition is formulated by a signal $\delta(d, d')$ and a flag $\Delta(d, d')$. Intuitively, $\delta(d, d')$ means that the branch condition from d to d' becomes true at the current cycle (the current clock period), and $\Delta(d, d')$ means that branch condition is being true before the start of the current cycle. Their values are defined as follows where $C(d, d')$ is the branch condition.

- $\delta(d, d') = 1$ iff $E(s_o) = 1$ for every operation that $C(d, d')$ depends on, and $C(d, d') = 1$.
- Initially $\Delta(d, d') = 0$.
- $\Delta(d, d')$ is set when $\delta(d, d') = 1$
- $\Delta(d, d')$ is reset when all the operations in DFG d are finished.

The second condition, which we denote $p(d, d')$, is formulated using two flags $A(d)$ (meaning d is active) and $H(d)$ (meaning d is the head most among active DFGs) defined for each DFG d. Let $e(d)$ be a signal that means all the operations in DFG d finishes in the current cycle.

1) For the starting DFG d_0, initially $A(d_0) = H(d_0) = 1$. For the other DFG d, initially $A(d_0) = H(d_0) = 0$.
2) $A(d')$ is set when there is a transition from a state in some DFG d to a state in d'.
3) $A(d)$ is reset when $e(d) = 1$.
4) $H(d')$ is set and $H(d)$ is reset, when $A(d) = 1$ and $A(d') = 0$ and there is a transition from a state in some DFG d to a state in d'.

The value of $p(d, d')$ is expressed as follows:

$$p(d, d') = \overline{H(d)} \vee \overline{A(d')} \vee e(d')$$

Let s be the FMS F_u's last state in DFG d and s be the FMS F_u's first state in a DFG d'. Then F_u transitions from s to s' when

$$(e(s) \vee E(s)) \wedge (\delta(d, d') \vee \Delta(d, d')) \wedge p(d, d_1).$$

F_u stays in s until the condition above is satisfied.

(4) Inter-DFG dependency

For inter-DFG dependency, such as $s_1 \rightarrow s_2$ in Fig. 4 (a), the end signals $E(s)$ do not work well. For example, in Fig. 4 (a), if s_2 waits for $E(s_1)$ signal which will be reset as soon as DFG1 finishes, s_2 can never start execution. If reset of $E(s_1)$ is postponed until the end of s_2, then s_3 in the next iteration starts without waiting for the completion of s_1 in the next iteration.

To solve this problem, we introduce $E(s_1, s_2)$ for every inter-DFG dependency from s_1 to s_2. It is set when s_1 is finished and reset when s_2 is reset.

Note that s_1 and s_2 in $E(s_1, s_2)$ are not always executed. For example, in Fig. 4 (b), suppose branch from DFG d_1 to DFG d_4 is taken. If s_8 would waits for $E(s_5, s_8)$, s_8 could never start execution, because s_5 is not executed and $E(s_5, s_8)$ will never be set. Moreover, $E(s_1, s_4)$ would not reset in this iteration. Then if branch from d_1 to d_2 might be taken in the next iteration, s_4 would start execution without waiting for the completion of s_1.

This problem is resolved by setting/resetting the end signal $E(s_1, s_2)$ when directions of branches are determined:

- If the DFG of s_1 will never be executed in the current iteration, then $E(s_1, s_2)$ is set.
- If the DFG of s_2 will never be executed in the current iteration, then $E(s_1, s_2)$ is reset.
- If both the DFGs of s_1 and of s_2 will never be executed in the current iteration, then the initial value is set to $E(s_1, s_2)$.

For example, in Fig. 4 (b), branch from d_1 to d_4 is determined when $\Delta(d_1, d_4) \vee \delta(d_1, d_4)$ becomes true, and at this point, it is also determined that d_2 and d_3 will not be executed in this iteration. Thus, $E(s_1, s_4) = 0$, $E(s_3, s_6) = 0$, and $E(s_5, s_8) = 1$ are enforced.

B. Automatic generation of distributed controller

The FSMs and the necessary signals to control functional units can be generated according to the formulation in the previous subsubsection.

To handle inter-DFG dependency (described in III A (4)), DFGs which will not be executed in the iteration must be identified every time the branch conditions are updated. For this purpose, reachability analysis of the given CDFG is done beforehand. First, the CDFG is traversed starting from the start DFG d_0 so that loop edges are identified and removed. Next, the set of DFGs reachable from d, denoted as $R(d)$, is computed for each DFG d. Then, for every conditional branches from d to d', the set of DFG $\rho(d, d')$ is computed which consists of DFGs that become newly unreachable if branches from d to d' is taken:

$$\rho(d, d') = R(d) - R(d') - \{d\}$$

Using the $\rho(d, d')$, updating of completion flag $E(s_1, s_2)$ is overwritten. Let $d(s)$ be the DFG that state s belongs to. The, every time a conditional branch from DFG d_1 to d_2 is known to be taken, the flags are updated as follows:

- When $d(s_1) \in \rho(d_1, d_2)$,
  - if $d(s_2) \in R(d_2)$ then set $E(s_1, s_2) = 1$
  - else set $E(s_1, s_2) = E_0(s_1, s_2)$

TABLE I
EXPERIMENTAL RESULTS.

CDFG	binding	Execution cycles (for 128 iterations)					Logic synthesis results			
		CC		DC (w/o SE)			CC (w SE)		DC (w/o SE)	
		w/o SE	w SE	$d_M = 1.0$	1.5	2.0	LUTs (FFs)	delay [ns]	LUTs (FFs)	delay [ns]
bicubic	A2M2	5,477	5,272	4,979	5,464	5,930	1,056 (373)	8.304	1,180 (501)	8.213
	A2M2'	5,247	5,042	4,612	5,097	5,563	1,245 (375)	8.005	1,128 (496)	8.605
	A1M2	5,931	5,726	6,589	7,245	7,882	1,110 (373)	8.183	1,152 (488)	7.778
	A1M1	6,677	6,677	6,906	7,854	8,833	954 (356)	8.238	954 (501)	8.383
m-lerp	A3M2	6,034	5,028	5,527	5,895	6,284	956 (499)	8.247	1,463 (598)	7.378
	A2M2	5,544	5,531	4,784	5,152	5,541	1,327 (549)	8.156	1,538 (647)	8.660
	A2M1	6,815	6,556	4,784	5,738	6,812	1,104 (562)	8.557	1,419 (631)	8.722

CC: centralized control (conventional; with trace and loop scheduling) DC: distributed control (proposed)
w SE: with speculative execution w/o SE: with speculative execution d_M : average cycles for multiplication

- When $d(s_2) \in \rho(d_1, d_2)$,
  - if $d(s_1) \in R(d_2)$ then set $E(s_1, s_2) = 0$
  - else set $E(s_1, s_2) = E_0(s_1, s_2)$

Note that the conditions for updating flags are determined statically during synthesis; there is no need for the synthesized hardware to compute the reachability.

IV. EXPERIMENTAL RESULTS

Based on the proposed method, a tool is implemented which generates an logic synthesizable RTL model in Verilog HDL from a CDFG specification.

Experiments have been conducted on two benchmark CDFGs. It was assumed that multipliers took either 1 or 2 cycles, while all the other operations completed in 1 cycle. Several operation bindings were tested for each benchmarks. As for value bindings, as many registers as the values in a given CDFG were allocated and each value was assigned to a dedicated register.

Experimental results are summarized in TABLE I. Binding "AnMm" implies that n ALUs and m multipliers were used, where A2M2 and A2M2' used the same number of units in different ways.

"Execution cycles" are the number of clock cycles spent for 128 loop iterations. We assumed the conditional branches were taken randomly at the same probabilities. "CC" indicates the conventional centralized control and "DC" the proposed distributed control. The CC was based on the loop scheduling and trace scheduling assuming multiplication always took 2 cycles. Furthermore, two versions were tested for CC; without speculative execution (w/o SE) and with speculative execution (w SE). On the other hand, only without SE was tested, for its formulation is out of the scope of this paper. d_M for DC is the average cycles for multiplication; $d_M = 1.0$ means that all the multiplications were completed in 1 cycle, $d_M = 1.5$ that the half of the multiplications took 2 cycles, and $d_M = 2.0$ that the all the multiplications took 2 cycles.

When $d_M = 2.0$, the proposed the DC was slower than the CC (even w/o SE). This is mainly due to the insertion of the dummy states. When $d_M = 1.0$, the DC was faster than the CC (w SE), with the execution cycle reduced by 8.5% at the best case. The performance of the DC depended largely on the bindings.

"Logic synthesis results" are based on Xilinx's logic synthesizer Vivado (2016.4) targeting an FPGA model Artix–7 (xc7a100tcsg324-3). The number of the look-up tables (LUTs) and flip-flops (FFs) for the DC were larger than for the CC by about 13%. This is because the DC needs more FFs for multiple FSMs and more sophisticated control logic. However, the critical path delay was almost the same (0.1% smaller than the CC on the average). This is because the control logic depends on smaller number of FFs in the DC.

V. CONCLUSION

We have presented a method of automatically generating RTL designs based on the distributed control from given CDFG data, and have conducted experiments to examine pros and cons of the distributed control scheme. The speed-up by dynamic scheduling depended largely on the binding. The circuit size was about 13% larger on average but the critical path delay stayed the same. It is future work to find optimal bindings automatically and to incorporate speculative execution into the distributed control scheme.

Acknowledgment

Authors would like to express our thanks to Ms. Miho Shimizu and Mr. Yuuki Oosako who were with Kwansei Gakuin University, and Ms. Wakako Nakano and all the members of Ishiura Laboratory of Kwansei Gakuin University, for their advice and discussion regarding this work. This work was partly supported by JSPS KAKENHI under Grant No. 16K00088.

REFERENCES

[1] D. D. Gajski, N. D. Dutt, A. C-H Wu, and S. Y-L Lin: *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers (1992).
[2] Y. Toda, N. Ishiura, and K. Sone: "Static scheduling of dynamic execution for high-level synthesis," in *Proc. SASIMI 2009,* pp. 107–112 (Mar. 2009).
[3] A. A. Del Barrio, S. Ogrenci M., M. C. Molina, J. M. Mendías, and R. Hermida: "A distributed controller for managing speculative functional units in high-level synthesis," in *Proc. DATE 2011,* pp. 350–363 (Mar. 2011).
[4] C. Pilato, V. Giovanni Castellana, S. Lovergine, and F. Ferrandi: "A runtime adaptive controller for supporting hardware components with variable latency," in *Proc. AHS-2011,* pp. 153–160 (June 2011).
[5] M. Shimizu and N. Ishiura: "Extending distributed control for high-level synthesis beyond borders of basic blocks," in *Proc. SASIMI 2016,* pp. 172–177 (Oct. 2016).
[6] W. Nakano and N. Ishiura: "Extended distributed control for dynamic scheduling across dataflow graphs" (short paper), in *Proc. SASIMI 2018,* pp. 35–36 (Mar. 2018).