社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

高位合成における分散制御のデータフローグラフ境界を越えた拡張

清水 美帆† 石浦菜岐佐†

† 関西学院大学 理工学部 〒 669–1337 兵庫県三田市学園 2-1

あらまし 本稿では, 不定サイクル演算を含むデータフローグラフの効率的な実行時スケジューリングを可能にする分散制御を, 複数のデータフローグラフへ拡張する方法を提案する. 従来の高位合成では, 演算のサイクル数は固定として静的にスケジューリングを決定するが, 実際には演算のサイクル数が実行時の状況によって変動する不定サイクル演算器が存在する. このような演算器を含む回路を効率的に制御する手法として, Del Barrio, Pilato, 山下らが複数の状態機械によって演算器の制御を行う分散制御方式を提案しているが, これらはいずれも単一のデータフローグラフの制御を対象としたものであった. 本稿では, Del Barrio の分散制御方式を複数のデータフローグラフの制御に拡張する手法を提案する. 本手法は, データフローグラフの境界を越えて演算の動的なスケジューリングを可能にするものであり, 従来の集中制御方式におけるループスケジューリングやトレーススケジューリングに比べ, 実行に必要な総サイクル数を削減できる.

キーワード 高位合成, 不定サイクル演算, 分散制御, 動的スケジューリング

Extending Distributed Control for High-Level Synthesis beyond Boundaries of Dataflow Graphs

Miho SHIMIZU† and Nagisa ISHIURA†

† Kwansei Gakuin University, 2-1 Gakuen, Sanda, Hyogo, 669–1337, Japan

Abstract This paper proposes an extension of distributed control, which enables efficient run-time scheduling of variable latency operations, to multiple dataflow graphs. Conventional high-level synthesis methods determine the execution schedule of operations statically assuming that their latencies are fixed. However, actual circuits contain so-called variable latency units whose execution cycles may vary depending on various run-time factors. Although Del Barrio, Pilato, and Yamashita have proposed distributed control methods which can efficiently control circuits with such units, they only handles a single dataflow graphs. Our method extends the Del Barrio's distributed control to handle multiple dataflow graphs. It enables dynamic scheduling of operations beyond the boundaries of dataflow graphs which results in fewer execution cycles than those by conventional centralized control with loop scheduling and trace scheduling.

Key words high-level synthesis, variable latency operation, distributed controller, dynamic scheduling

1. はじめに

近年の集積回路技術に伴って, ハードウェアとして実装されるシステムの規模や機能は益々増大している. 一方で, 製品サイクルは短縮化の傾向にあり, ハードウェアの開発期間短縮に対する要求は益々厳しくなっている. ハードウェアの設計を効率化する手法の一つとして, プログラミング言語等の動作記述からハードウェアの設計記述を自動生成する高位合成技術 [1] の研究が進められている.

従来の高位合成手法では, 演算の実行に要するサイクル数は固定として静的なスケジューリングが行われていた. しかし, 実際の演算には, オペランドや実行時の状況等に依存してサイクル数が変化する「不定サイクル演算」が存在する. 通常, 実行にかかるサイクル数を最大と想定してスケジューリングを行うが, 演算がそれよりも早く完了した場合には無駄な待ち時間が発生することになる.

これに対し, [2] では制御を実行時に変更する可変スケジューリング手法を提案している. 演算器からの完了信号に基づいてスケジューリングを動的に変更するものであるが, 状態数が著しく増加するという問題があった. この問題を解決する手法として, 複数の状態機械を設けて演算の制御を行う分散制御が挙げられる. 分散制御は, 制御回路の増大を抑制しつつ, 不定サイ

クル演算を含む回路を効率的に制御することができる．分散制御の方式としては, Del Barrio の方式 [3], Pilato の方式 [4], 山下の方式 [5] が提案されている．

しかし, これらの分散制御方式はいずれも, 単一のデータフローグラフを対象としたものに限られており, 複数のデータフローグラフについての議論は成されていない．ソフトウェアとして開発された C 言語などによるプログラムからハードウェアを合成するような応用では, 複数のデータフローグラフによって構成される制御構造を扱えるが必要となる．

そこで本稿では, 不定サイクル演算に対応した分散制御方式を複数データフローグラフに拡張する手法を提案する．本手法は Del Barrio の手法を拡張したものであるが, データフローグラフの境界を超えた演算の動的スケジューリングを可能にすることにより, トレーススケジューリングやループスケジューリングに匹敵する性能を出すことを狙いとしている．

2 つのベンチマークを用いて評価実験を行ったところ, 提案手法は従来の集中制御方式に比べ, 約 24%の回路規模の増加で, 実行に必要な総サイクル数を大幅に削減することができた．

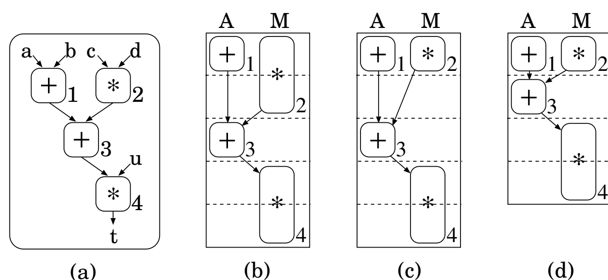


図 1　不定サイクル演算を含む回路の制御

2. 不定サイクル演算と分散制御方式

2.1 不定サイクル演算

データパス中の演算器が実行に要するサイクル数は, オペランドや演算器の状態に依存して変動することがある．例えば, 加減算の繰り返しによる乗算器や除算器では, 乗数の一部分や中間結果の被除数が 0 になれば, 計算を省略してサイクル数を短縮できる．メモリアクセスに要するサイクル数は, アドレスの系列に依存して変動する．また, 経年劣化等による遅延の変動によって演算に要するサイクル数が変わることもある．このような演算器は不定サイクル演算器 (variable-latency unit/component) と呼ばれる．

実行に 1 サイクルを要する加算器 A と, 実行に 1～2 サイクルを要する乗算器 M を用いて図 1 (a) のデータフローグラフ (以下 DFG と略する) の計算を実行する場合を考える．従来の高位合成手法では, 乗算には 2 サイクルを要するものとして (b) のようにスケジューリングを行う．しかし, 演算 2 が 1 サイクルで完了した場合, (c) のように余分な待ち時間が発生することになる．

文献 [2] では, 演算器からの完了信号を用いて演算の開始タイミングを動的に制御する手法を提案しており, (d) のような制御を行うことができる．しかし, 状態数が著しく増加し, 制御回路の規模が増大してしまう．

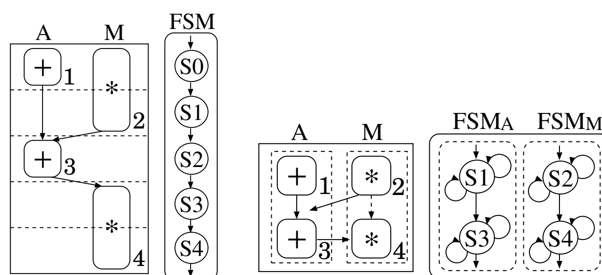
2.2 分散制御方式

制御回路の規模を増大させることなく不定サイクル演算に対応した動的な制御を実現する手法として, 近年, 複数の状態機械を用いて演算の制御を行う分散制御方式が提案されている．

Del Barrio の方式 [3] では, 演算器毎に状態機械を設けて演算の実行タイミングを制御する．演算を実行する演算器と演算を実行する順序は設計時に決定するが, 演算を実行するタイミングは実行時に調整される．Pilato の方式 [4] では, 実行の可否を表す状態変数を各演算に割り当てて制御を行う．演算を実行する演算器は設計時に決定するが, 実行のタイミングと順序は動的に変更される．山下 [5] の方式では, 演算の実行順序やタイミングだけでなく, 演算を実行する演算器も実行時にハードウェアが決定する．

これらの分散制御方式は, 従来の集中制御方式における状態機械の符号を演算器毎に分割し, それぞれの演算を独立に制御し易くしたものと見ることもできる．

しかし, これらの方式は, いずれも単一の DFG を対象にしており, ソフトウェアとして開発された C 言語などのプログラムからハードウェアを合成するような応用では, 複数の DFG から構成されるため, 複数の DFG の場合の制御が重要となる．



(a) 集中制御方式　(b) Del Barrio の分散制御方式

図 2　集中制御と分散制御

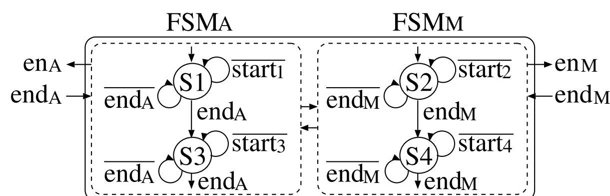


図 3　Del Barrio の分散制御方式

2.3 Del Barrio の分散制御方式

Del Barrio の分散制御方式は, 1 つの演算器に 1 つの状態機械を割り当てて演算の実行を制御する手法である．従来の集中制御方式では, 図 2 の (a) のように 1 つの状態機械 (FSM) で回路全体を制御する．これに対し, Del Barrio の分散制御方式では, (b) のように演算器 A と M をそれぞれ別の状態機械 FSM_A と FSM_M で制御する．これにより, 演算の実行タイミングが独立に変更できるようになる．

図 2 (b) の FSM を詳細化したものを図 3 に示す．なお, 以下の定式化は文献 [3] とは異なるが, 本質的には同じものである．DFG 中の各演算 i に対して 1 つの状態 S_i を使用する．状態 S_i

は DFG 中の演算 i の実行を制御する. まず, S_i では $start_i$ が 1 になるのを待つ. $start_i$ は演算 i へのデータ依存がある全ての演算が終了しているとき 1 になる.

$$start_1 = 1$$
$$start_2 = 1$$
$$start_3 = (s_2 \wedge end_M) \vee Done_2$$
$$start_4 = (s_3 \wedge end_A) \vee Done_3$$

ただし, s_i は FSM が状態 S_i にあることを表し, end_u は演算器 u からの完了信号を表す. $Done_i$ は演算 i が完了していることを表し, 初期値を 0 とて, 次のように更新される.

if $(s_1 \wedge end_A)$ $\{Done_1 <= 1\}$
if $(s_2 \wedge end_M)$ $\{Done_2 <= 1\}$
if $(s_3 \wedge end_A)$ $\{Done_3 <= 1\}$
if $(s_4 \wedge end_M)$ $\{Done_4 <= 1\}$

状態 S_i で $start_i$ が 1 になると, 各 FSM_u は演算器に開始信号 en_u を出力する.

$$en_A = ((s_1 \wedge start_1) \vee (s_3 \wedge start_3)) \wedge \overline{Exec_A}$$
$$en_M = ((s_2 \wedge start_2) \vee (s_4 \wedge start_4)) \wedge \overline{Exec_M}$$

ただし, $Exec_u$ は演算器 u が演算中であることを表し, 次のように定義される.

if (end_A) $\{Exec_A <= 0\}$
else if (en_A) $\{Exec_A <= 1\}$
if (end_M) $\{Exec_M <= 0\}$
else if (en_M) $\{Exec_M <= 1\}$

Del Barrio の分散制御方式は, Pilato や山下の方式に比べると, 実行時に演算の順序や, 演算を実行する演算器を動的に決定することはできないが, 最も簡潔であり, 回路規模や遅延も最も小さい.

Del Barrio はこの方式を, 単一の DFG のループに拡張し, あるアイテレーションの全演算の完了を待たずに次のアイテレーションの演算を実行できるようにしている. しかし, 複数の DFG からなる制御は扱っておらず, 条件分岐が発生する場合を扱うことはできない.

3. データフローグラフの境界を超えた分散制御

3.1 概 要

本稿では, Del Barrio の分散制御方式を複数の DFG が扱えるように拡張する. この際, DFG の境界を越えて演算を動的にスケジューリングできるようにする.

従来の集中制御では, 1 つの DFG の演算がすべて終了してから次の DFG に遷移する. 分散制御でも同様にすれば複数の DFG の制御が可能であるが, 不定サイクル演算の動的スケジューリングを行っている場合には全ての演算が同時に終了するわけではないので, DFG の末尾で待ち合わせが生じることになる. この無駄をなくすために, 本稿で提案する手法では, データ依存関係が満たされていれば, ある DFG の全演算の完了を待たずに, 次の DFG の演算の実行を開始できるようにする. 例えば, 図 4 において, 回路中の 2 つの演算器が 2 つの FSM で制御されているとき, (a) において, $S0$ と $S3$ の演算が完了し, かつ, DFG1 から DFG3 への遷移が確定しているとする. このような場合, 本稿の制御方式では, DFG1 の全演算の完了を待たずに DFG3 の $S11$ の演算の実行を開始する.

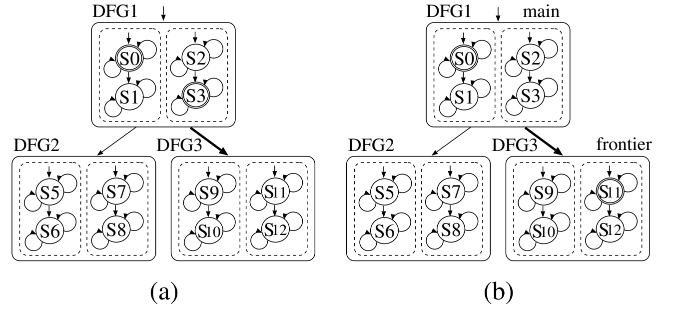


図 4 データフローグラフ境界を越えた分散制御

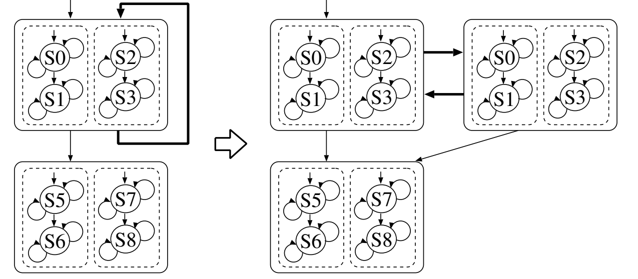


図 5 自己ループの除去

ただし, 本稿では, 同時に演算を行えるのは 2 つの DFG までであり, 3 つの DFG にまたがる演算の実行は行わないものとする. 1 つの DFG をループしている場合には, 図 5 のようにその DFG のコピーを作成して自己ループを除去する. また, 各状態機械は各 DFG に必ず状態を持つものとし, ない場合には演算を行わない状態を挿入する. 以下本稿では, 演算が行われている 1 つ目の DFG をメイン, 2 つ目の DFG をフロンティアと呼ぶ. 図 4 (b) では, DFG1 がメインで, DFG3 がフロンティアである.

3.2 定 式 化

演算器の集合を U とする. 演算器 $u \in U$ の現サイクルにおける出力を out_u, 完了信号を end_u とする. DFG の集合を D とする. DFG $d \in D$ における $u \in U$ の動作を制御する FSM を $F_{d,u}$, その状態の集合を $S_{d,u}$ とする. $F_{d,u}$ の最後の状態を $f_{d,u}$ とする.

DFG d と状態 $s \in S_{d,u}$ に対して, $final(d)$ と $Done(s)$ を次のように定義する. $final(d)$ は現サイクルで DFG d の実行が終了することを意味し, $Done(s)$ は状態 s における演算の実行が前のサイクル以前に完了していることを意味する. $Done_0(s)$ は $Done(s)$ の初期値を, $Done'(s)$ は $Done(s)$ の次のサイクルの値を表す.

$$final(d) = \bigwedge_{u \in U} (Done(f_{d,u}) \vee (f_{d,u} \wedge end_u))$$
$$Done_0(s) = 0$$
$$Done'(s) = \begin{cases} 0 & (final(d) = 1 \text{ のとき; リセット}) \\ Done(s) \vee (s \wedge end_u) & (\text{それ以外}) \end{cases}$$

$Trans(e, d)$ は DFG e から d に遷移することが現サイクルで確定していることを意味する. d が e の唯一の後続 DFG であれば, $Trans(e, d)$ は常に 1 とする. e の後続 DFG が複数あり, 遷移を決定する演算が状態 s に演算器 u で行われるとき, その初期値 $Trans_0(e, d)$ と, 次のサイクルの値 $Trans'(e, d)$

は次のように表せる．

$$Trans_0(e,d)=0$$

$$Trans'(e,d)$$

$$=\begin{cases}0 \quad (final(e)=1\text{ のとき; リセット})\\ Trans(e,d)\vee(s\wedge end_u\wedge out_u)\quad(\text{それ以外})\end{cases}$$

$Active(d)$ は DFG d がメインとして実行されていることを意味する．その初期値 $Active_0(d)$ と，次のサイクルの値 $Active'(d)$ は，d の前の DFG の集合を $prev(d)$ とすると，次のように表せる．

$$Active_0(d)=\begin{cases}1 \quad (d\text{ が全体の先頭 DFG のとき})\\ 0 \quad (\text{それ以外})\end{cases}$$

$$Active'(d)$$

$$=\begin{cases}0 \quad (final(d)=1\text{ のとき; リセット})\\ Active(d)\vee\\ (\bigvee_{e\in prev(d)}(final(e)=1\wedge Trans(e,d)=1))\quad(\text{それ以外})\end{cases}$$

$Frontier(d)$ は DFG d がフロンティアとして実行されていることを意味する．その初期値 $Frontier_0(d)$ と，次のサイクルの値 $Frontier'(d)$ は，d の前の DFG の集合を $prev(d)$ とすると，次のように表せる．

$$Frontier_0(d)=0$$

$$Frontier'(d)$$

$$=\begin{cases}0 \quad (Active'(d)=1\text{ のとき})\\ Frontier(d)\vee\\ (\bigvee_{e\in prev(d)}(Active'(e)=1\wedge Trans(e,d)=1)\quad(\text{それ以外})\end{cases}$$

DFG d 内の演算は，$Active(d)=1$ または $Frontier(d)=1$ のとき，かつそのときに限り実行される．それ以外の制御は，2.3 節の Del Barrio の方式と同様である．

4. 実験結果

2 つのベンチマークについて提案手法に基づく回路を Verilog HDL で設計し，従来の制御手法と比較する実験を行った．以下のベンチマークでは，乗算にかかるサイクル数は 1〜2 とし，それ以外の演算のサイクル数は全て 1 とした．

4.1 ベンチマーク CDFG

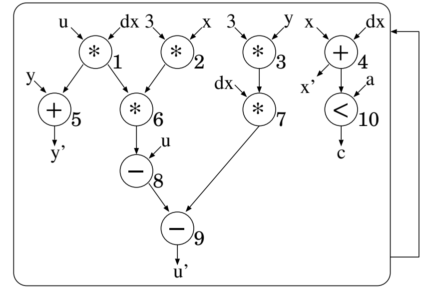
(1) diffeq

図 6(a) の DFG を加算器 A1, A2, 乗算器 M1, M2 で実行する．分散制御のバインディングとスケジューリングの例を図 6(b) に示す．自己ループを除去するために，DFG のコピーを作成している．図 6(c) は比較実験のために，従来の集中制御においてループスケジューリングを行った結果である．

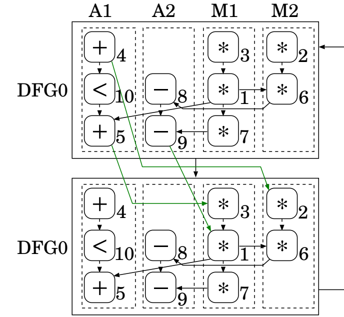
(2) iprod64

図 7(a) は 64 ビットデータの内積を計算する回路である．64 ビットの乗算は，32 ビットの乗算 4 回とその加算により行う．この際，乗数の上位または下位 32 ビットが 0 であった場合には，それに対する乗算と加算を省略する．この計算を A (ALU), AS (ALU+シフタ), M (乗算器), L (ロードストア), C (データ転送), および E (比較回路) で実行するものとする．

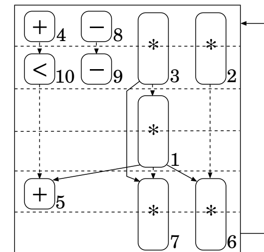
この DFG に対する分散制御のスケジューリング例を図 7(b) に示す．DFG0 のコピーを作成して，自己ループを除去している．提案手法では，各状態機械が各 DFG に少なくとも 1 つの



(a) データフローグラフ



(b) 分散制御によるスケジューリング



(c) ループスケジューリング (集中制御)

図 6　ベンチマーク diffeq

表 1　実験結果 (サイクル数)

	集中制御	分散制御 (提案)		
		$r=1.0$	$r=0.5$	$r=0.0$
diffeq	770	770	632	513
iprod64	710	740	541	422

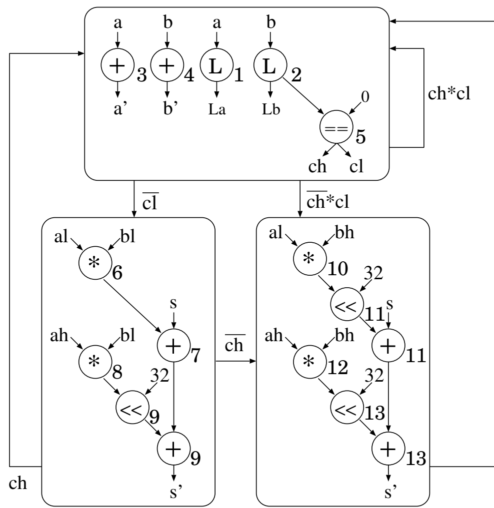
ループ回数: 128
乗算のサイクル数: 1〜2
r : 乗算のサイクル数が 2 サイクルの割合
集中制御はトレース/ループスケジューリング

状態を持つ必要があるため，状態のない DFG がある場合にはダミー状態 (図中破線で描いている状態) を追加する．この状態は演算を行わず，1 サイクルで次状態へ遷移可能なものである．

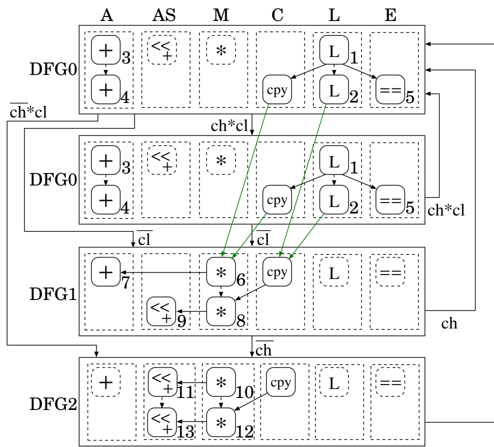
図 7(c) は比較実験のために，従来の集中制御においてトレーススケジューリングとループスケジューリングを行った結果である．

4.2 サイクル数

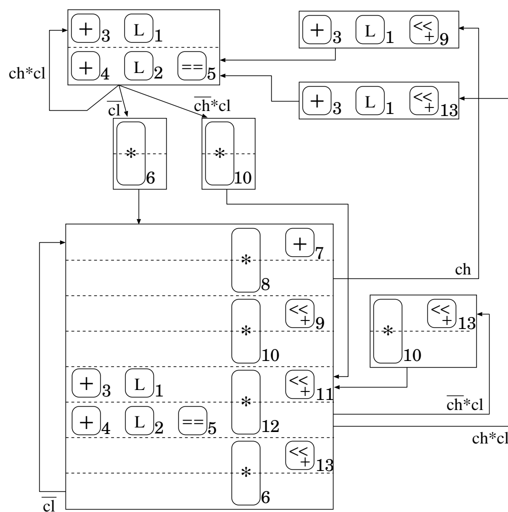
2 つのベンチマークについてサイクル数の比較を行った結果を表 1 に示す．ループ回数は 128 とした．表中の「集中制御」は図 6(c)，図 7(c) のループスケジューリング/トレーススケジューリングに基づく結果であり，乗算は 2 サイクルとしてスケジューリングを行っている．「分散制御」の r は乗算のサイクル数が 2 になる割合であり，$r=1.0$ は全ての乗算が 2 サイ



(a) データフローグラフ



(b) 分散制御によるスケジューリング



(c) トレーススケジューリング (集中制御)

図 7　ベンチマーク iprod64

クルで実行される場合, $r = 0.0$ は全ての乗算が 1 サイクルで完了する場合を表す. iprod の変数 b の上位ビット , 下位ビットがそれぞれ 0 である確率は 50%とした.

diffeq において, $r = 1.0$ の場合, 提案手法はループスケジューリングと同じサイクル数を得ることができた. $r = 0.5, r = 0.0$

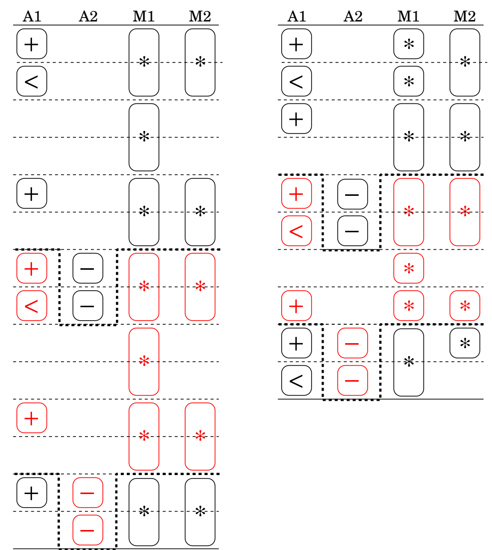


図 8　diffeq の実行例

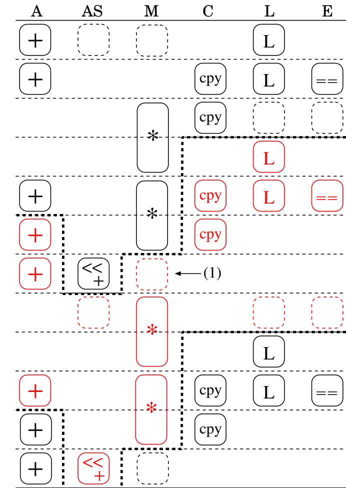


図 9　iprod64 の実行例

の場合には, 提案手法は集中制御に比べてサイクル数を大幅に削減できている.

iprod において, $r = 1.0$ の場合はトレーススケジューリングと比べてサイクル数が増加している. これは, 提案手法では演算を行わない状態が挿入されるためであり , 1 つ先の DFG までしか先行して制御できないことに起因する. しかし , $r = 0.5, r = 0.0$ の場合には, 提案手法は集中制御と比べてサイクル数を削減できている.

図 8 は提案手法による diffeq の実行例である. (a) は全ての乗算が 2 サイクルで行われた場合である. 演算器 A2 の演算が行われるとき, 次の DFG の演算を開始できるため, この例では 1 アイテレーションあたり 6 サイクルで実行できる. これは, ループスケジューリングと等価である. (b) は乗算 12 個のうち 6 個が 1 サイクルで完了した場合の実行例である. 実行可能であれば次の演算を開始できるため, サイクル数を大きく削減できる.

図 9 は提案手法による iprod64 の実行例である. $r = 1.0$ の場合を表しており , この例でもトレーススケジューリングとほ

表 2　実験結果 (回路規模)

	集中制御		分散制御 (提案)	
	slices (FFs)	delay[ns]	slices (FFs)	delay[ns]
diffeq	567 (331)	15.044	702 (441)	14.878
iprod64	724 (464)	14.492	890 (556)	15.871

ぼ同様の結果を得ることができる．しかし, 提案手法では (1) のような演算を行わない状態を持っている場合があり, トレーススケジューリングよりサイクル数がわずかに大きくなる．ただし, 1 サイクルで完了する乗算がある場合, (1) の状態への遷移を早めることができるため, トレーススケジューリングと等価, もしくはそれ以上の性能を得ることができる．

4.3 回路規模

2 つのベンチマークの論理合成結果を表 2 に示す．論理合成は Xilinx ISE (14.7) を使用し, FPGA (Spartan–3E) をターゲットに行った．状態の符号化にはワンホット符号を用いた [6].

slices は回路面積, FFs はフリップフロップ数, delay は回路遅延を示している．回路規模はいずれの回路も約 24%増加したが, 遅延はほぼ同じであった．これより, 提案手法は現実的な回路規模と遅延で実装可能であると言える．

5. 考　察

提案手法は, トレーススケジューリングやループスケジューリングと同様に, DFG の境界を越えて演算をスケジューリングさせることができる．これに加え, 本手法では, 演算のサイクル数の変化に合わせて演算のスケジューリングを変更することが可能である．ただし, 演算を移動させることができるのは 1 つ先の DFG までに限定されるので, この点で従来法に劣る場合がある．

なお, トレーススケジューリングおよびループスケジューリングの結果に対して Del Barrio の分散制御法を適用すれば, DFG 境界を越えて演算を移動しつつ, サイクル数の変動に合わせた動的なスケジューリングを実現することが可能である．この手法は, トレーススケジューリングで演算を 2 つ以上の DFG にわたって移動させることができるため, 想定したトレースの実行では本手法より優れる場合がある．しかし, それ以外のトレースが実行された場合には, 動的にスケジューリングを変更できる本手法の方が優れる場合がある．

また, 本手法が有効になるのは, DFG の最終サイクルより前に分岐先が決定している場合に限られる．この点に関しては, 分岐予測や投機的実行の導入により高速化が可能になると考える．

6. む す び

本稿では, 不定サイクル演算を含む DFG の効率的な実行時スケジューリングを可能にする分散制御を, 複数の DFG へ拡張する方法を提案した．2 つのベンチマークを用いて評価実験を行い, 提案手法は従来の集中制御方式に比べ, 約 24%の回路規模の増加でサイクル数を大幅に削減することができた．

今後の課題としては, Del Barrio 以外の分散制御手法の拡張, および分岐予測や投機的実行の導入が挙げられる．

謝　辞

本稿の研究に関して多くの御助言をいただきました京都高度技術研究所の神原弘之氏, 立命館大学の冨山宏之教授, 元立命館大学の中谷嵩之氏に感謝いたします．また, 本研究に関してご協力, ご討議頂いた関西学院大学石浦研究室の諸氏に感謝いたします．

文　献

[1] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin: *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers (1992).

[2] Yuki Toda, Nagisa Ishiura, and Kousuke Sone: “Static scheduling of dynamic execution for high-level synthesis,” in *Proc. SASIMI 2009,* pp. 107–112 (March 2009).

[3] Alberto A. Del Barrio, Seda Ogrenci Memik, María C. Molina, José M. Mendías, and Román Hermida: “A Distributed Controller for Managing Speculative Functional Units in High-Level Synthesis,” in *IEEE Council on Electronic Design Automation (DATE 2011),* pp. 350–363 (March 2011).

[4] Christian Pilato, Vito Giovanni Castellana, Silvia Lovergine, and Fabrizio Ferrandi: “A runtime adaptive controller for supporting hardware components with variable latency,” in *Proceedings of 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2011),* pp. 153–160 (June 2011).

[5] 山下真司, 石浦菜岐佐: “不定サイクル演算に対応した分散制御における動的演算バインディング,” 電子情報通信学会技術研究報告, VLD2013–128 (Jan. 2014).

[6] 清水美帆, 石浦菜岐佐: “不定サイクル演算に対応した分散制御のための状態符号化の検討,” 電子情報通信学会ソサイエティ大会, A-3-14 (Sept. 2014).