不定サイクル演算に対応した分散制御における投機的実行

清水 美帆[1]　石浦 菜岐佐[1]

概要：本稿では, データフローグラフ境界を越えて動的スケジューリングを行う分散制御方式に投機的実行を導入する手法を提案する. 一般的な高位合成手法で用いられる静的スケジューリングでは, 実行時にサイクル数が変動する演算が存在する場合, 余分な待ちが発生する. このような回路を効率的に制御する手法として, 演算の実行タイミングを動的に調整する分散制御方式が提案されているが, 我々はこれを拡張し, 複数のデータフローグラフにまたがって動的な演算スケジューリングを実現する手法を提案している. これに対し本稿では, さらに分岐予測に基づく演算の動的な投機的実行を可能にする手法を提案する. 2 つの回路に対する実験では, 投機的実行を行わない分散制御に対し約 5.0%の回路規模の増加で, サイクル数を平均 15.2%削減できた.

キーワード：高位合成, 分散制御, 動的スケジューリング, 分岐予測, 投機的実行

Speculative Execution in Distributed Controllers for High-Level Synthesis

Shimizu Miho[1]　Ishiura Nagisa[1]

Abstract: This article proposes a method of incorporating speculative execution into distributed control which enables dynamic scheduling of operations beyond the boundaries of basic blocks. In the presence of variable latency units, the static scheduling scheme in conventional high-level synthesis causes wasteful waits. Distributed control enables dynamic scheduling of operations, of which we previously proposed an extension to allow operation motion across two dataflow graphs. In this article, we further introduce speculative execution based on branch prediction into our previous scheme. Experimental results on two examples showed that the execution cycles were reduced by 15.2% on average as compared with our previous method without speculative execution, while the circuit size was increased by 5.0%.

Keywords: high-level synthesis, distributed controller, dynamic scheduling, branch prediction, speculative execution

1. はじめに

近年の集積回路技術の発展に伴って, 実装可能なハードウェアの機能や規模は年々増大している. 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, C 言語等による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, 演算の中には, オペランドや演算器の状況等に依存して実行時にサイクル数が変化する「不定サイクル演算」が存在する. 通常は実行に要する最大サイクル数を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してしまう.

演算器からの完了信号に基づいて演算の実行タイミングを動的に調整すれば無駄な待ちを解消できるが, 従来の制御方式では状態機械の状態数が著しく増加し, 回路規模が

[1] 関西学院大学 理工学部
Kwansei Gakuin University, 2-1 Gakuen, Sanda, Hyogo, 669–1337, Japan

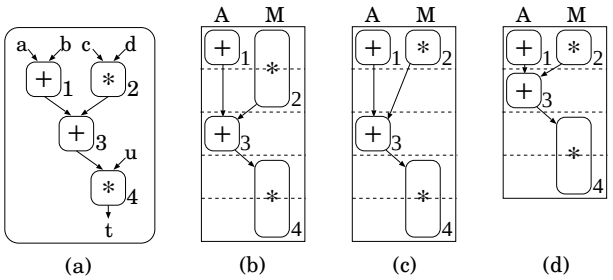


図 1 不定サイクル演算を含む回路の制御

非現実的になってしまう [2]. これに対し, 演算器毎に状態機械を設ける分散制御方式を用いれば回路規模の増加を抑制することができる. 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であった.

我々は [7] において, 複数のデータフローグラフに対応した分散制御方式を提案している. この方法では, データフローグラフ境界を越えて演算の実行タイミングを調整できる. しかし, データフローグラフの最終サイクルより前に分岐先が決定していない場合には, 動的スケジューリングの効果が見込めない.

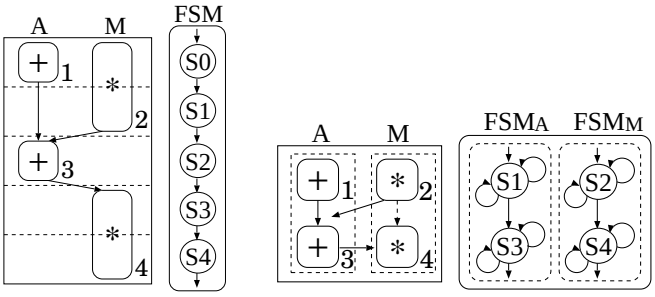
そこで本稿では, データフローグラフ境界を越えて動的スケジューリングを行う分散制御方式に投機的実行を導入する手法を提案する. 本手法では, 分岐予測により, 分岐先が決定していなくてもデータフローグラフ境界を越えた投機的実行を行う. 2 つのベンチマークを用いて評価実験を行ったところ, 提案手法は従来の集中制御や投機的実行を行わない従来方式と比べ, 約 30%の回路規模の増加で, 実行に必要な総サイクル数を大幅に削減することができた.

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. 本稿ではこのような演算器を不定サイクル演算器 (variable-latency unit) と呼ぶ.

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 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.



(a) 集中制御方式　　(b) Del Barrio の分散制御方式

図 2 DFG を制御する状態機械

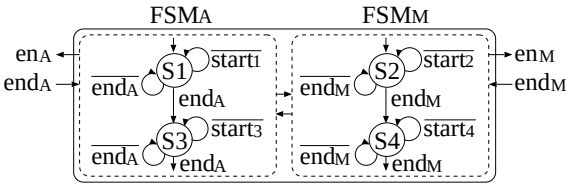


図 3 Del Barrio の分散制御方式

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 Del Barrio の分散制御方式

Del Barrio の分散制御方式 [3] 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. 図 2(a) は, 従来法である集中制御の制御例を表している. DFG 中の演算 1, 3 は演算器 A で, 演算 2, 4 は演算器 M で実行される. 有向枝はデータ依存を, 破線の有向枝は順序を示している. (a) は集中制御による制御例を表しており, FSM は状態機械を表す. (b) は Del Barrio の分散制御方式による制御例を表している. 演算 1, 3 は FSM A, 演算 2, 4 は FSM M という状態機械でそれぞれ制御される.

Del Barrio の分散制御方式の状態機械を図 3 に示す. 状態 1 では, 演算 1 の開始条件である $ready_{S1}$ が 1 になると演算 1 を実行する. 各状態には, 演算が完了したことを示すレジスタ $done$ があり, 演算 1　が完了すると $done_{S1} = 1$ となる. 演算器 A の完了信号 $Aend$ が 0 である間は演算器からの完了信号を待ち, $Aend$ が 1 になれば, $done$ を 0 から 1 に書き換え, 次の状態へ遷移する. 演算の開始条件は, 演算の依存関係から決まる. 例えば, 演算 3　は演算 2 の後に実行できるので, $ready_{S3} = S4 \vee done_{S2}$ となる. Si は状態 i にあることを示す. 演算器 A が演算中である場合は $running_A = 1$ とする. 演算器 A への開始信号は, $start_A = (running_A = 0) \wedge (S1 \wedge ready_{S1}) \vee (S3 \wedge ready_{S3})$ と表す.

これにより, 各状態機械内で演算の開始のタイミングを

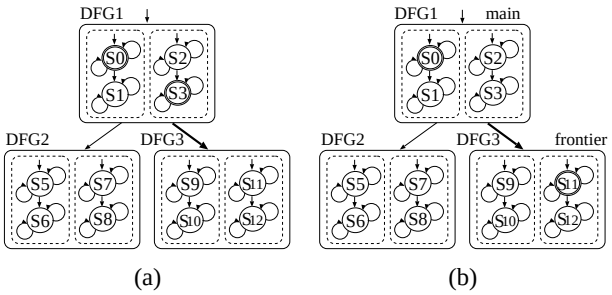


図 4 データフローグラフ境界を越えた動的スケジューリング

動的に変更することができる. ただし, 同一演算器にバインディングされている演算は, 設計時に決められた順序でしか実行できない. この 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.

2.3 データフローグラフ境界を越えた動的スケジューリング

この課題に対し我々は, 複数の DFG に対応した分散制御方式を提案している [7]. この手法は, 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するというものであり, DFG 境界を越えて演算を動的にスケジューリングすることができる. 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.

ただし, DFG のループの扱いには注意が必要である. [7] では, 同時に演算を行える DFG は 2 つ以下に限定し, DFG の自己ループは展開するという方式を採っている.

この手法では, 基本ブロックの境界を越えて演算の実行タイミングを調整できるため, 演算のサイクル数が固定の場合でも, 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ほぼ同等の高速化効果をが得られる. 演算のサイクル数が変動する場合には, スケジューリングを動的に調整できるため, 実行に要する総サイクル数を大幅に削減することができる. 変数の生存区間が長くなるため, 必要なレジスタ数やマルチプレクサが増えるが, 回路規模の増加は集中制御方式に比べて 24%程度である.

しかしこの手法は, 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のみ有効であり, それ以外の場合は従来手法のトレーススケジューリングに比べて, サイクル数が増大してしまう場合がある.

3. 分散制御への投機的実行の導入

3.1 概要

本稿では, 投機的実行の導入により, この課題を解決する手法を提案する.

本手法では, 分岐予測に基づいて, 分岐先が決まる前に DFG 境界を越えて投機的に動的スケジューリングを行う. 分岐予測が外れた場合には, それが判明した時点で正しい分岐先の DFG に状態遷移を行う.

DFG 間で依存のあるレジスタについては, 投機的実行を行っている間は書き込みを行わず, 別のレジスタに値を保持する. 分岐予測が当たった場合にはそのレジスタから書き戻しを行い, 分岐予測が外れた場合にはそのレジスタの値は破棄する.

なお, 演算を同時に実行できる DFG 数は本稿でも [7] と同様, 2 までとする.

3.2 定式化

3.2.1 データフローグラフ境界を越えた動的スケジューリング

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, DFG3 がフロンティアである.

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 o_u , 完了信号を e_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, $\gamma(s)$, $\Gamma(s)$, および $\zeta(d)$ を次のように定義する. $\Gamma(s)$ は, 状態 s の演算の実行が現サイクルの開始までに完了していることを, $\gamma(s)$ は, 状態 s の演算の実行が現サイクルの終了までに完了することを, $e(d)$ は, 現サイクルで DFG d の実行が終了することを意味する. $\sigma_{d,u}$ は $F_{d,u}$ の現サイクルでの状態を表し, $\Gamma_0(s)$ と $\Gamma'(s)$ は, それぞれ $\Gamma(s)$ の初期値と次のサイクルでの値を表す.

$$\gamma(s) = \Gamma(s) \vee ((\sigma_{d,u} = s) \wedge e_u)$$
$$e(d) = \bigwedge_{u \in U} \gamma(f_{d,u})$$
$$\Gamma_0(s) = 0$$
$$\Gamma'(s) = \textbf{if } e(d) \textbf{ then } 0 \textbf{ else } \gamma(s)$$

また, $e(d)$ が 1 になれば, 全ての $u \in U$ について, $F_{d,u}$ の次状態を初期状態とする.

$\Delta(e,d)$ は DFG e から d に遷移することが現サイクルの開始までに確定していることを, $\delta(e,d)$ は DFG e

から d に遷移することが現サイクルの終了までに確定することを表す. d が e の唯一の後続 DFG の場合は, $\Delta(e,d) = \delta(e,d) = 1$ である. e の後続 DFG が複数ある場合, 遷移は状態 t において演算器 u で行わる演算の結果が 1 のときに e から d への遷移が起きるとすると, $\delta(e,d)$ と $\Delta(e,d)$ は次のように定義される. ($\Delta_0(e,d)$ と $\Delta'(e,d)$ は, それぞれ $\Delta(s)$ の初期値と次のサイクルでの値を表す.)

$$\delta(e,d) = \Delta(e,d) \vee ((\sigma_{e,d} = t) \wedge e_u \wedge o_u)$$
$$\Delta_0(e,d) = 0$$
$$\Delta'(e,d) = \textbf{if } e(e) \textbf{ then } 0 \textbf{ else } \delta(e,d)$$

$\mathcal{M}(d)$ は DFG d がメインとして実行されていることを表す. d の前の DFG の集合を P_d とすると, $\mathcal{M}(d)$ の初期値 $\mathcal{M}_0(d)$ と次サイクルの値 $\mathcal{M}'(d)$ は次のように表せる.

$$\mathcal{M}_0(d) = \textbf{if } d \text{ が先頭 DFG } \textbf{then } 1 \textbf{ else } 0$$
$$\mathcal{M}'(d) = \textbf{if } e(d) \textbf{ then } 0 \textbf{ else } \mathcal{M}(d) \vee \bigvee_{e \in P_d} (e(e) \wedge \delta(e,d))$$

$\mathcal{F}(d)$ 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し, その初期値 $\mathcal{F}_0(d)$ と次サイクルの値 $\mathcal{F}'(d)$ は次のように表せる.

$$\mathcal{F}_0(d) = 0$$
$$\mathcal{F}'(d) = \textbf{if } \mathcal{M}'(d) \textbf{ then } 0 \textbf{ else } \mathcal{F}(d) \vee \bigvee_{e \in P_d} (\mathcal{M}'(e) \wedge \delta(e,d))$$

3.2.2 投機的実行の導入

文献 [7] では DFG d は $\mathcal{M}(d) \vee \mathcal{F}(d)$ のときそのときに限り実行されるとしていた. これに対し, 本稿では, DFG e から d への分岐が予測されていることを表す $B(e,d)$ と, DFG d が投機的に実行されることを表す $\mathcal{P}(d)$ を導入し, DFG d は $\mathcal{M}(d) \vee \mathcal{F}(d) \vee \mathcal{P}(d)$ のときそのときに限り実行されるものとする.

$B(e,d)$ は, 静的に決定されていても, 動的に決定されても良い. $\mathcal{P}(d)$ の初期値 $\mathcal{P}_0(d)$ と次サイクルの値 $\mathcal{P}'(d)$ は次のように定義される.

$$\mathcal{P}_0(d) = 0$$
$$\mathcal{P}'(d) = \textbf{if } \mathcal{M}'(d) \vee \mathcal{F}'(d) \textbf{ then } 0 \textbf{ else } \mathcal{P}(d) \vee \bigvee_{e \in P_d} (\mathcal{M}'(e) \wedge B(e,d))$$

DFG d において, 他の DFG の値に依存がある値を格納するレジスタの集合を R とする. また, d への遷移が確定するまでレジスタ $r \in R$ の値を保持する一時レジスタを t_r とし, r に値を供給する演算器を u とする. レジスタ r', t'_r の次サイクルでの値 r', t'_r は次のように定義される.

$\mathcal{P}(d) \wedge \mathcal{P}'(d)$ のとき (投機的実行)
$r' = r$
$t'_r = \textbf{if } e_u \textbf{ then } o_u \textbf{ else } t_r$
$\overline{\mathcal{P}(d)} \wedge \overline{\mathcal{P}'(d)}$ のとき (非投機的実行)
$r' = \textbf{if } e_u \textbf{ then } o_u \textbf{ else } r$
$t'_r = t_r$
$\mathcal{P}(d) \wedge \overline{\mathcal{P}'(d)}$ のとき (投機的から非投機的に遷移)
$r' = \textbf{if } e_u \textbf{ then } o_u \textbf{ else } t_r$
$t'_r = t_r$

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, 集中制御および [7] の分散制御と比較する実験を行った. 本稿の実験では, 分岐予測は静的であり, 集中制御のトレーススケジューリングと同じ方向の分岐のみを選ぶものとした. 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

いずれのベンチマークも, 条件分岐を決定する演算は DFG の最終サイクルでしか行えないものである.

(1) bicubic

図 5(a) の DFG を加算器 A1, A2, 乗算器 M1, M2 で実行する. 分散制御のスケジューリング例を図 5(b) に示す. 図 5(b) 中の実線の有効枝と番号は, 依存関係と状態の番号を示している. 自己ループを除去するために, DFG0 はコピーを作成している. 図 5(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. このスケジューリングは, 分岐予測に基づいて投機的実行を行うものである.

(2) m-lerp

図 6(a) の DFG を加算器 A1, A2, 乗算器 M1, M2 および EQ (比較回路) で実行する. 分散制御のスケジューリング例を図 6(b) に, 集中制御のループスケジューリング例を図 6(c) に示す.

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 5(c) および図 6(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. r は乗算のサイクル数が 2 となる割合であり, $r = 1.0$ は全ての乗算に 2 サイクルを要した場合, $r = 0.0$ は全ての乗算が 1 サイクルで完了した場合を表す. 分岐予測が当たる確率は 75%とした.

提案手法と投機的実行を行わない従来手法 [7] を比較すると, 提案手法では r がいずれの場合もサイクル数が削減されている. 削減率は平均 15.2%であり, これは, 投機的実行の導入による効果と考えられる. 集中制御で投機的実行を行うトレーススケジューリングと比較すると, $r = 1.0$ の場合には提案手法のほうがサイクル数が増加している. 本手法では演算の実行は 1 DFG 先までと限定されるので, こ

表 1 実験結果 (サイクル数)

	集中制御 (投機的実行)	分散制御					
		投機的実行なし [7]			提案手法 (投機的実行)		
		$r = 1.0$	$r = 0.5$	$r = 0.0$	$r = 1.0$	$r = 0.5$	$r = 0.0$
bicubic	990	1193	1047	928	1043	888	735
m-lerp	763	913	895	789	764	753	709

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, 分岐予測が当たる割合: 75%

表 2 実験結果 (回路規模)

	集中制御 (投機的実行)			分散制御					
				投機的実行なし [7]			提案手法 (投機的実行)		
	FFs	LUTs	delay [ns]	FFs	LUTs	delay [ns]	FFs	LUTs	delay [ns]
bicubic	276	802	14.62	363	945	16.64	366	985	15.50
m-lerp	428	923	14.94	562	1299	15.44	567	1373	15.03

の点でトレーススケジューリングに劣る場合がある. なお, 分岐予測が外れた際のサイクル数はトレーススケジューリングと同等である. $r = 0.5$, $r = 0.0$ の場合にはサイクル数が大幅に削減されており, これは, 動的スケジューリングによる効果と考えられる.

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]. 表中, FFs はフリップフロップ数, LUTs はルックアップテーブル数, delay は回路遅延を示している. 従来の集中制御に対し FFs 約 32.5%, LUTs 約 35.2%, 投機的実行を行わない従来手法 [7] に対し FFs 約 0.9%, LUTs 約 5.0%の増加が見られたが, 遅延はほぼ同じであった.

5. おわりに

本稿では, DFG 境界を越えて動的スケジューリングを行う分散制御方式に投機的実行を導入する手法を提案した. これにより, DFG の最終サイクルまでに分岐先が決定しない場合においても, 次の DFG の演算を開始することができる. 2 つのベンチマークを用いて評価実験を行ったところ, 提案手法は投機的実行を行わない従来手法 [7] に対し約 5.0%の回路規模の増大でサイクル数を平均 15.2%削減することができた.

今後の課題として, 回路面積の削減やバインディング/スケジューリングの自動化等が挙げられる.

謝辞 本稿の研究にあたり, ご指導, ご助言を頂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, 本研究に関してご討論, ご協力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. なお, 本研究は一部 JSPS 科研費 16K00088 の助成による.

参考文献

[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 (Mar. 2009).

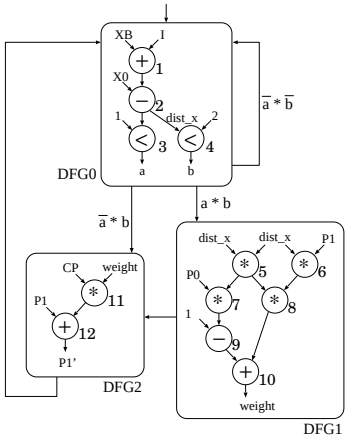
[3] Alberto A. Del Barrio, Seda Ogrenci Memik, María C. Molina, José M. Mendías, and Román Hermida: "A Distributed Controller for Managing Speculative Functional Units in High-Level Synthesis," in *Proc. (DATE 2011),* pp. 350–363 (Mar. 2011).

[4] Christian Pilato, Vito Giovanni Castellana, Silvia Lovergine, and Fabrizio Ferrandi: "A runtime adaptive controller for supporting hardware components with variable latency," in *Proc. NASA/ESA (AHS-2011),* pp. 153–160 (June 2011).

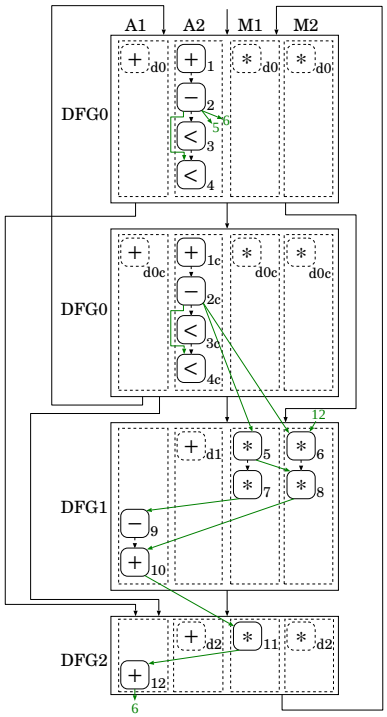
[5] 山下真司, 石浦菜岐佐: "不定サイクル演算に対応した分散制御における動的演算バインディング," 信学技報, VLD2013–128 (Jan. 2014).

[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," 信学ソ大, A-3-14 (Sept. 2014).

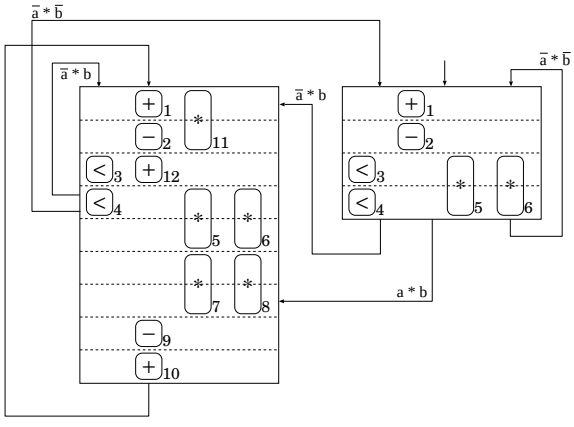
[7] 清水美帆, 石浦菜岐佐: "高位合成における分散制御のデータフローグラフ境界を越えた拡張," 信学技報, VLD2015–61 (Dec. 2015).



(a)

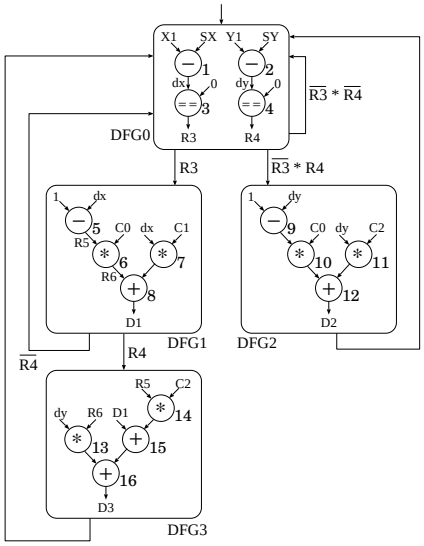


(b)

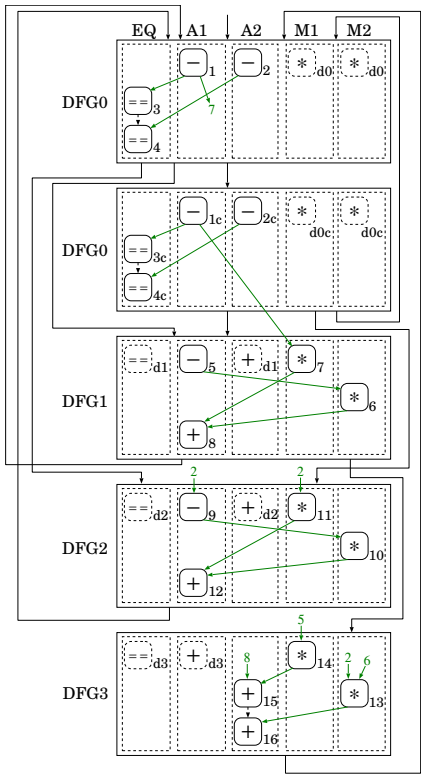


(c)

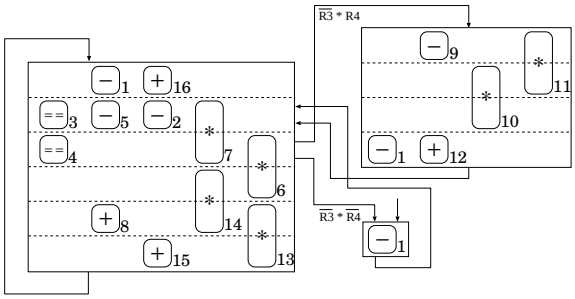
図 5 ベンチマーク bicubic



(a)



(b)



(c)

図 6 ベンチマーク m-lerp