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.

VLIW型DSPのコード最適化のためのサイクル分割スケジューリング

益井　勇気†　　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
E-mail: †{y-masui,ishiura}@ksc.kwansei.ac.jp

あらまし　本稿では, VLIW 型 DSP のコードスケジューリングを高速に行う一手法として, サイクル分割スケジューリングを提案する. クラスタ型 VLIW DSP に対してレジスタファイルの容量, データ転送演算の挿入, オペランドの非対称性まで考慮したスケジューリング問題の厳密解法が提案されているが, コードが大規模になると現実的な時間で解を得ることができない. そこで, 本手法では, コード全体を一度にスケジューリングするのではなく, コードを先頭から一定サイクル数づつスケジューリングする. 解の厳密最適性は保証されないが, 各回のスケジューリング処理の計算時間を抑制できるため, より規模の大きいコードに対しても現実時間で解を求めることが可能になる.
キーワード　クラスタ型 VLIW DSP, コード最適化, TMS320C62x, サイクル分割スケジューリング

Cycle Partitioned Scheduling for Code Optimization of VLIW DSP

Yuki MASUI† and Nagisa ISHIURA†

† Kwansei Gakuin University,　Gakuen 2–1, Sanda, Hyogo, 669–1337, Japan
E-mail: †{y-masui,ishiura}@ksc.kwansei.ac.jp

Abstract　This paper proposes a *cycle partitioned* scheduling method for code optimization of VLIW DSPs. The previously proposed optimum code scheduling method for VLIW DSPs, which takes into account the capacity of registerfiles, insertion of data transfer operations, and operand asymmetry of functional units, required such an enormous computation cost that it can not handle large scale codes within a practical amount of time. Instead of processing a whole code at a time, our scheduler builds up entire scheduling by repeating computation for a fixed amount of cycles. This curbs the computation cost for each stage and allows optimization of the larger codes within feasible time, though the optimalty of the solution may not be guaranteed.
Key words　clutered VLIW DSP, code optimization, TMS320C62x, cycle partitioned scheduling

1. はじめに

近年, ディジタル映像・音響機器の急速な普及に伴い, 動画配信等のさまざまなサービスが提供され, これらの機器に要求されるディジタル信号処理の計算量が増大している. 高性能な汎用プロセッサにより要求性能の達成は可能であっても, 消費電力やコストが大きな問題となる. 専用ハードウェアは最も電力性能比に優れるが, 設計のコストが大きく, また実装後の修正や仕様変更が難しい. DSP (Digital Signal Processor) はディジタル信号処理を高速に低消費電力で実行することを目的に最適化されたプロセッサであり, 性能電力比と開発の容易さのトレードオフにおける 1 つの解を与えるものである.

DSP には, 用途に応じて様々なアーキテクチャを持つものがあるが, 特に高い処理性能を求める場合には VLIW (Very Long Instruction Word) 型 DSP が利用される. VLIW DSP は静的スケジューリングに基づく並列演算により優れた電力性能比を達成するが, その性能を引き出すためには, 効率のよい命令スケジューリングを行うことが課題となる.

VLIW 型 DSP のコードスケジューリングに関する研究は [1], [2], [3] など多数存在するが, DSP は特殊なデータパス構成を持つものが多いため, 実際の DSP のコード生成, 最適化に際しては, 個々の DSP の持つ特殊な制約まで考慮しなければならない.

[4] では Texas Instruments 社製 TMS320C62x [5] (以下 C62x) をモデルとしたクラスタ型 VLIW プロセッサに対して, Simulated Annealing を利用したスケジューリングの高速近似解法を提案している. しかしレジスタファイルの容量制約やユニットのオペランド非対称性まで考慮した定式化は行えていないため, これをそのまま実際のコード生成に適用することはできない. また必要に応じてレジスタバンク間の転送演算の挿入を行うが, 1 つの演算に対して転送は高々1 回しか行えないという制限もある.

[6] では [7], [8] の考え方に基づき, C62x のデータパスの詳細やレジスタ制約まで考慮したコードスケジューリング問題を定式化している. PBSAT (pseudo Boolean satisfiability) [9] により厳密な最適解を求めているが, 大規模なコードに対しては, 現実時間で解を導くことが出来ないという問題がある.

そこで本研究では, [6] に基づいて, 一定サイクルずつスケジューリングを行う, 分割スケジューリング手法を提案する. コード全体を一度にスケジューリングするのではなく, コードの先頭から一定サイクルづつスケジューリングを行うことにより, 計算時間を抑制する. この手法を C62x に適用した結果, [6] では解けなかったいくつかの問題を解くことが出来た.

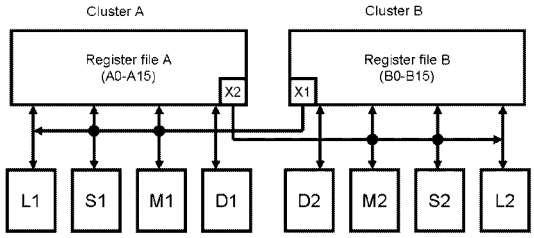

図 1　TMS320C62x のデータパス

2. TMS320C62x

本稿では具体的な VLIW 型 DSP として Texas Instruments 社製 TMS320C62x [5] (以下 C62x) を対象に, コードスケジューリング問題を考える. 図 1 に C62x のデータパスを示す. C62x は 2 つのクラスタ (A, B) を持ち, 各クラスタはレジスタファイル (RF) と 4 つの演算器 (L, S, M, D) から成る.

演算器は合計 8 個あり, 最大で 8 演算が同時に実行できるが, 演算器毎に実行可能な演算が異なる. 表 1 に C62x の演算の一部に対して, 実行サイクル数と使用可能演算器を載せる. 実行サイクル数は各演算の遅延サイクルであり, 使用演算器にかかわらず決まる.

レジスタの読み込みは演算の 1 サイクル目, 書き込みは 1〜6 サイクル目で行う. 演算に必要なレジスタ, 演算器, パスの資源は 1 サイクル目にだけ使用する.

各演算器のオペランドには基本的に同じクラスタの RF を指定するが, 「クロスパス」を使用すれば反対側のクラスタの RF をオペランドとして指定出来る. クロスパスには RF B から RF A に転送する X1, RF A から RF B に転送する X2 の 2 つがあり, それぞれ 1 サイクルに 1 回ずつ使用出来る.

クロスパスと即値の使用条件は演算器毎に異なる. 演算器 L では両方のオペランドでクロスパス, 即値が使用できるが, 演算器 M, S は第 1 オペランドでしかクロスパス, 即値を使用できない. 演算器 D は第 1 オペランドのみ即値を使用できるがクロスパスは使用できない.

本研究では C62x のコードに対して演算の並列化, 資源割り当て, 転送演算の挿入を行う。ただし演算の変換は行わないものとする.

表 1　TMS320C62x の演算

演算の種類	命令	実行サイクル数	使用可能演算器
加算	ADD	1	L, S, D
減算	SUB	1	L, S, D
乗算	MPY	2	M
左シフト	SHL	1	S
右シフト	SHR	1	S
ゼロ演算	ZERO	1	L, S, D
分岐	B	6	S
レジスタ間転送	MV	1	L, S
ロード	LDW/LDH/LDB	5	D
ストア	STW/STH/STB	3	D

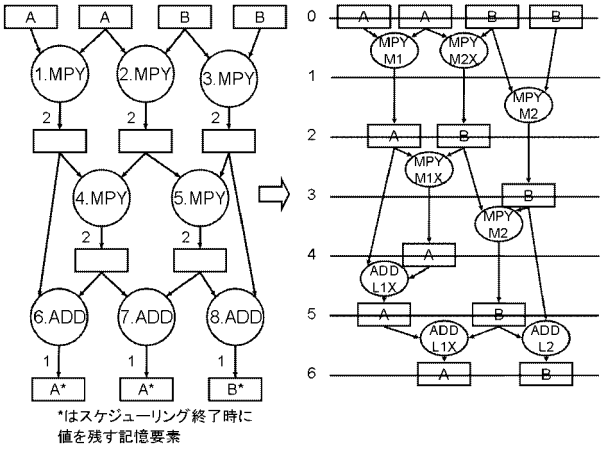

図 2　コード生成の例

表 2　命令パターンの情報の例

演算		$I(p)$	ADD
ユニット		$U(p)$	{L1}
クロスパス		$X(p)$	{X1}
オペランド	1	$m(p,1)$	A
	2	$m(p,2)$	B
	3	$m(p,3)$	A

3. コードスケジューリング問題

3.1 最適コードスケジューリング [6]

本稿で扱うコード生成問題は, 「依存グラフ」と, 「命令パターン集合」に対し, 実行サイクル数が最小となるように転送演算を含めた各演算の実行開始サイクルと資源割り当てを決定する問題である. 転送演算は, クラスタ間でのレジスタ転送演算や, レジスタが足りないときのスピル/リロードであり, 必要に応じて自動的に挿入する. 「依存グラフ」は分岐演算を含まない基本ブロックを表したグラフであり, 「命令パターン」はプロセッサで実行可能な命令の情報を表す. 図 2 にコードスケジューリングの概要を示す。左側の依存グラフから, 各種制約を満たし, 実行サイクル数が最小となるように, 各演算節点の実行開始サイクルと使用資源を右図のように決定する.

基本ブロック中の演算を表す節点の集合を F, 値を表わす節点の集合を V, データ依存枝の集合を E とする. 依存グラフ

$G=(V\cup F, E)$ は非巡回の有向2部グラフ ($E\subseteqq V\times F\cup F\times V$) である. 各枝 $e\in E$ にはオペランド番号 $o(e)$ と整数値の遅延 $d(e)$ が定義されている. データを格納する記憶要素の集合を M とし, $m\in M$ の容量を $C(m)$ とする. C62x では $M=\{A,B,MM\}$ (それぞれ RF A, RF B, メモリを表す) である. 各節点には $i(v)\subseteqq M$, $o(v)\subseteqq M$ が定義されている. $m\in i(v)$ はサイクル 0 に v の値が記憶要素 m に与えられることを示す. $m\in o(v)$ は v が記憶要素 m に得られる必要があることを示し, すべての節点 v が $o(v)$ に示された記憶要素に値が格納されたときスケジューリングが完了する.

命令パターン集合 P の要素 p には, p が使用する演算器の集合 $U(p)\subseteqq\{L1,S1,M1,D1,L2,S2,M2,D2\}$, クロスパスの集合 $X(p)\subseteqq\{X1,X2\}$ が定義されている. また命令パターン p において i 番目のオペランドの値を格納する記憶要素 $m(p,i)$ が定義されている. 演算 $f\in F$ の結果を記憶要素 m に格納する命令パターンの集合を $P_{f,m}$ とする. 表 2 に命令パターンの例を示す. この命令パターンは演算器 L1, クロスパス X1 を使用することにより, RF A の値と RF B の値の和を RF A に出力することを表す.

S をスピル, リロード, レジスタ間転送演算の集合, $d(s)$ を転送演算 s の遅延とする. $P_{s,n,m}$ を転送演算 s で記憶要素 n から記憶要素 m へデータを転送する命令パターンの集合とする.

上記で記したコード生成問題を解くために [7] の考え方が利用できる. [7] では依存グラフとデータパス情報から可能な全てのスケジューリングを表現する有限状態機械 (以下 FSM) を生成し, 初期状態から最終状態に至る最短経路 (実行サイクル数が最小のスケジューリングに相当する) を求めるというものである. この手法は, 記憶容量の制約やデータ転送の挿入まで扱うことができる.

FSM における状態と状態遷移を表すために 3 種類の変数 $\alpha_{t,v,m}$, $\xi_{t,f,p}$, $\tau_{t,v,p}$ を使用する. 変数 $\alpha_{t,v,m}$ は, サイクル t に節点 v が記憶要素 m に存在すれば 1, そうでなければ 0 をとる. $\xi_{t,f,p}$ はサイクル t に節点 f の演算を命令パターン p で実行すれば 1, そうでなければ 0 をとる. $\tau_{t,v,p}$ はサイクル t に節点 v を命令パターン p で転送すれば 1, そうでなければ 0 をとる. ただし $t\leqq -1$ のとき $\alpha_{t,v,m}=0$, $\xi_{t,f,p}=0$, $\tau_{t,v,p}=0$ とする. t_{max} を実行サイクル数の上限とし, 実行サイクルの集合を $T=\{0,1,2,\cdots,t_{max}\}$ とする.

最適スケジューリングの制約条件は次の通りである.

(1) 入力制約: サイクル 0 に節点 v が記憶要素 $i(v)$ に与えられていることを表す.

$$\forall v\in V:\quad \alpha_{0,v,m}=1 \quad \text{iff} \quad m\in i(v).$$

(2) 依存制約: 演算と変数の間のデータ依存関係を表す.

1. サイクル t に節点 v の値が記憶要素 m に存在するための必要条件を表す.

$\forall t\in T-\{0\},\forall(f,v)\in E,\forall s\in S,\forall n\in M,\forall m\in M:$

$$\alpha_{t,v,m}\to\alpha_{t-1,v,m}\vee\Big(\bigwedge_{(f,v)\in E}\Big(\bigvee_{p\in P_{f,m}}\xi_{t-d(f,v),f,p}\Big)\Big)\vee\Big(\bigvee_{p\in P_{s,n,m}}\tau_{t-d(s),v,p}\Big).$$

2. サイクル t に演算 f を命令パターン p で実行するための必要条件を表す.

$\forall t\in T-\{t_{max}\},\forall f\in F,\forall p\in P_{f,m}:$

$$\xi_{t,f,p}\to\bigwedge_{(v,f)\in E}\alpha_{t-d(v,f),v,m(p,o(e))}.$$

3. サイクル t に節点 v を命令パターン p で記憶要素 n から記憶要素 m に転送するための必要条件を表す.

$\forall t\in T-\{t_{max}\},\forall v\in V,\forall s\in S,\forall m,n\in M,\forall p\in P_{s,n,m}:$

$$\tau_{t,v,p}\to\alpha_{t,v,n}.$$

(3) 資源制約: 同一サイクルで同じ資源を複数の命令パターンで使用できないことを表す.

サイクル t でユニットまたはパス r を使用する変数の集合を $B_r^t=\{\xi_{t,f,p},\tau_{t,v,p}\mid t\in T-\{t_{max}\},f\in F,U(p)=r\vee X(p)=r\}$ とする.

$$\forall t\in T-\{t_{max}\},\forall r\in U(p)\cup X(p):\qquad\sum_{\beta\in B_r^t}\beta\leqq 1.$$

(4) 記憶容量制約: レジスタファイルと主記憶の容量の制約を表わす.

$$\forall t\in T,\forall m\in M:\qquad\sum_{v\in V}\alpha_{t,v,m}\leqq C(m).$$

(5) 代入制約: 全ての演算は 1 度しか実行しないことを表す.

$$\forall f\in F:\qquad\sum_{t\in T-\{t_{max}\},p\in P}\xi_{t,f,p}\leqq 1.$$

(6) 目的関数: 実行サイクル数の最小化を表す.

$$\text{maximize}\ \sum_{t\in T}\Big(\bigwedge_{v\in V}\Big(\bigwedge_{m\in o(v)}\alpha_{t,v,m}\Big)\Big)$$

3.2 分割スケジューリング

前節の最適スケジューリング問題は NP 完全と考えられるため, 依存グラフの規模が大きいと現実時間で解けないという問題がある. そこで本稿では, コード全体を一度にスケジューリングするのではなく, コードを先頭から計算が可能なサイクル数分づつスケジューリングしていく, 分割スケジューリング手法を提案する. まず 0 から $t_{max}-1$ サイクル目までのスケジューリングを行う. これで全体のスケジューリングが完了しないときは, スケジューリングする演算数の最大化を目指す. 以後 t_{max} ～ $2t_{max}-1$ サイクル, $2t_{max}$ ～ $3t_{max}-1$ サイクル, $\cdots$ と, t_{max} サイクルずつのスケジューリングを全体のスケジューリングが完了するまで繰り返す. 図 3 に分割スケジューリングの概要を示す. この例は左図の依存グラフを $t_{max}=4$ で分割スケジューリングした結果である. 分割により, 解の最適性は必ずしも保証されないが, 問題が小さくなり高速に解くことが出来る.

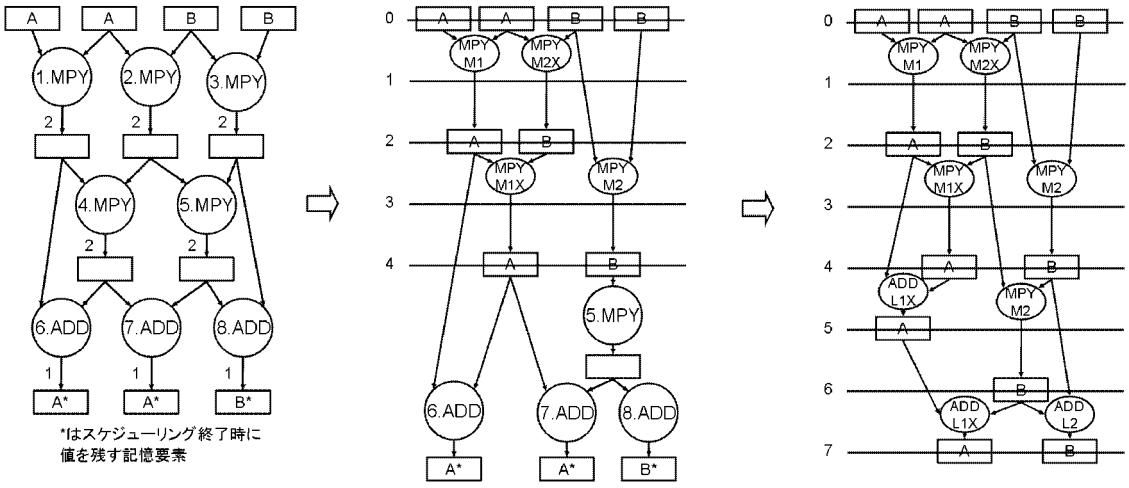

図 3　分割スケジューリングの例

以下, i 回目のスケジューリング $(i-1).t_{max}$ サイクル〜$i.t_{max}$ サイクルまでのスケジューリングを 0〜t_{max} サイクルに置き換え, 各スケジューリングにおける定式化を示す.

使用する変数に c_f, c_v を追加する. c_f は演算節点 f の演算をスケジューリング済みならば 1, そうでなければ 0 をとる. c_v は節点 v が次回以降のスケジューリングで必要ないなら 1, そうでなければ 0 をとる.

依存グラフの演算, 値を表す節点に $c(f) \in \{0,1\}, c(v) \in \{0,1\}$ を定義する. $c(f), c(v)$ は, 1 回目のスケジューリングでは $c(f)=0, c(v)=0$ であり, 2 回目以降のスケジューリングでは 1 回前のスケジューリングの c_f, c_v の値である. また各節点には $i(v)$ ではなく, $\hat{i}(v) = \{(t,m) \mid 1 \leqq t \leqq t_{max}, m \in M\}$ を定義する. $(t,m) \in \hat{i}(v)$ は節点 v の値がサイクル t に記憶要素 m に存在することを意味する. $\hat{i}(v)$ は 1 回目のスケジューリングでは $\hat{i}(v) = \{(0,m) \mid m \in i(v)\}$ である. 2 回目以降のスケジューリングではその前の回のスケジューリングの変数 ξ, α を元に決まり

$$\hat{i}(v) = \{(t,m) \mid \bigwedge_{(f,v)\in E} (\bigvee_{p\in P_{f,m}} (\xi_{t_{max}-d(f,v)+t,f,p})) = 1\}$$

$$\cup\{(0,m) \mid \alpha_{t_{max},v,m} = 1\}$$

となる.

各スケジューリングにおける制約式は最適コードスケジューリングの定式化と同じである. 異なる部分を以下に示す.

(1') 入力制約:

各スケジューリングの入力となる $\hat{i}(v)$ によって各値の存在変数 α は以下の制約を満たさなければならない.

$$\forall v \in V: \quad \alpha_{0,v,m} = 1 \quad \text{iff} \quad (0,m) \in \hat{i}(v).$$

(2') 依存制約:

1'. $\forall t \in T - \{0\}, \forall (f,v) \in E, \forall s \in S, \forall n \in M, \forall m \in M$:

$$\alpha_{t,v,m} \to \alpha_{t-1,v,m}$$

$$\vee(\bigwedge_{(f,v)\in E} (\bigvee_{p\in P_{f,m}} \xi_{t-d(f,v),f,p}))$$

$$\vee(\bigvee_{p\in P_{s,n,m}} \tau_{t-d(s),v,p}).$$

$$\vee(t,m) \in \hat{i}(v)$$

v に値が格納されるため条件を示している. 第 1 項は前のサイクルに値がすでに格納されていることを表す. 第 2 項は親の演算 f の結果が出力されたことを表す. 第 3 項は転送演算により値が転送されたことを表す. 第 4 項はその前の回のスケジューリングで親の演算 f が実行されてことを表す.

2'. $\forall t \in T, \forall f \in F, \forall p \in P_{f,m}$:

$$\xi_{t,f,p} \to (\bigwedge_{(v,f)\in E} \alpha_{t-d(v,f),v,m(p,o(e))}) \wedge \overline{c(f)}$$

サイクル t に演算 f を命令パターン p で実行するための必要条件に, 全体のスケジューリングで演算 f を 1 回しか実行できない制約を追加している.

(7) 完了制約

$$c_f \leftrightarrow c(f) \vee \sum_{t\in T, p\in P} \xi_{t,f,p} \geqq 1.$$

$$c_v \leftrightarrow c(v) \vee ((\bigwedge_{(v,f)\in E} c_f) \wedge (o(v) = \phi))$$

各節点の演算は全体のスケジューリングで 1 度しか実行してはならないので, 演算節点 f の開始サイクルが定められたときに c_f が 1 となる. 各節点の値は子の演算がすべて実行されたときその値を保持する必要がなくなるので, 子の演算節点の開始サイクルがすべて定められたときに c_v が 1 となる. ただし $o(v) \neq \phi$ のとき, 節点 v の値を全体のスケジューリング終了時

まで保持する必要があるため最終スケジューリングまで c_v は 0 となる.

(8) 存在制約: まだ完了してない値節点の値を消滅させないための制約.

$$\overline{c_v} \rightarrow (\bigvee_{m\in M} \alpha_{t_{max},v,m}) \vee (\bigwedge_{(f,v)\in E} \overline{c_f}) \vee \bigwedge_{(f,v)\in E} (\bigvee_{p\in P_f, t\in T} (\xi_{t-d(f,v),f,p} \wedge t \geqq t_{max} - d(f,v))$$

v がまだ必要な値であるときは右辺の 3 項のいづれかを満たさなければならない. 第 1 項では v の値がレジスタ, もしくはメモリに格納されていることを表す. 第 2 項では v の親 f が計算されていないことを表す. 第 3 項では v の親 f が現在計算中で次回のスケジューリングで v の値が得られることを表す.

(6') 目的関数:

$$\text{maximize } \beta \sum_{t\in T} (\bigwedge_{v\in V} (\bigwedge_{m\in v(o)} \alpha_{t,v,m})) \quad + \sum_{t\in T, f\in F, p\in P} \xi_{t,f,p}.$$

ここで β は十分大きい数とする. 第 1 項により t_{max} サイクルで実行可能の実行サイクル数を最小化する. 第 2 項により t_{max} サイクルで実行不可能のときに演算の実行数を最大化する.

分割スケジューリングでは 1 回のスケジューリングではスケジューリングの対象にならない節点が存在する. そこで, 各節点の ASAP 値 $a(f), a(v)$ を求め, $a(f) > t_{max}, a(v) > t_{max}$ を満たす節点 f, v には変数 $\alpha_{t,v,m}$, $\xi_{t,f,p}$, $\tau_{t,v,p}$ を割り当てないことにより変数, 式の数を減らすことができる. また必要のなくなった節点 $c(f) = 1, c(v) = 1$ を満たす f, v に関する変数も削減することができる.

4. 実装と実験結果

前節の定式化に基づき, TMS320C62x のコードスケジューリングプログラムを実装した. ソルバーには PBSAT (PBS Ver2.1 for Win) [9] を用いた. PBSAT は SAT の CNF に準ブール制約（一次不等式）を追加したもので, 解探索を高速に行うことができる. [9] の性能評価では, PBSAT は SAT に比べて 10 倍以上高速であることが示されている.

[6] と提案手法の比較を行った実験の結果を表 3 に示す. 「四則演算 1〜3」は演算数, 演算の種類, 枝の張り方などを変化させた依存グラフである. 「内積」は 2 つのベクトル値の内積を求めるものであり, データはメモリよりロードする. 「2×2 行列の乗算」は, 全データがレジスタ上にある場合の依存グラフである. 「4×4 行列の乗算」は, データをメモリよりロードする.

「演算数」は各依存グラフの演算節点の数, 「サイクル数」は生成されたコードの実行に要するサイクル数, 「CPU」は解を得るのに要した時間 (Cygwin ver.2.05b.0, Intel Celeron 1.83GHz, メモリ 1GB) である. 「>1000」は 1000 秒で解が得られなかったことを表わす. 「NA」は最適スケジューリング, 分割スケジューリングともに解が得られなかったことを表す. 最適スケジューリングが得られなかった依存グラフの「*」を付したサイクル数は, 最適解の理論値である. 分割スケジューリングでは $t_{max} = 5$ としてスケジューリングを行った. 「四則演算 3」と「内積 (100 次元)」では, 最適スケジューリングでは 1000 秒で解を得られなかったが, 分割スケジューリングでは数十秒で解くことが出来た. また結果が得られたすべての依存グラフで, 最適スケジューリングと同じサイクル数の解を求めることが出来ている. ただし 4×4 行列の乗算では分割スケジューリングでも解を得ることは出来なかった. 各節点の ASAP 値が小さいため, 変数と式の数が増えたこと, スピル/リロードの挿入が必要なことが計算量を増加させたと考えられる.

表 3　TMS320C62x のコードスケジューリング結果

	演算数	最適解 [6]		提案手法 ($t_{max} = 5$)	
		サイクル数	CPU[s]	サイクル数	CPU[s]
四則演算 1	10	13	1.4	13	2.6
四則演算 2	70	24	7.2	24	6.3
四則演算 3	80	*77	>1000	77	23.0
内積 (30 次元)	121	40	18.3	40	8.6
内積 (100 次元)	401	*110	>1000	110	31.7
2×2 行列の乗算	12	6	1.2	6	1.8
4 つの 2×2 行列の乗算	36	15	3.0	15	3.2
4×4 行列の乗算	160	NA	>1000	NA	>1000

* 最適解の理論値

5. むすび

本稿では VLIW 型 DSP のコード生成のための分割スケジューリング手法を提案した. スケジューリングを分割することにより, 計算時間の抑制を確認することが出来た. しかし現時点では, 分割スケジューリングでも解を得ることの出来ないコードが存在する. 種々のコードによる評価実験を行い, その結果に基づいて分割法の改善を試みることが今後の課題である.

謝　辞

本研究において共にコードスケジューリングの研究を行い, サポート頂いた小林涼氏 (現在古野電気株式会社) に感謝します. また, 貴重なご助言を頂いた日本電気株式会社の池川将夫氏, 久村孝寛氏, 大阪大学今井正治教授はじめ今井研究室の皆様に感謝します. ご助言, ご討論いただいた野垣内聡氏, 山本哲也氏をはじめ関西学院大学石浦研究室の関係諸氏に感謝します.

文　献

[1] G. Desoli: "Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach," Hewlett-Packard Laboratories, Technical Report, HPL-98-13 (Jan. 1998).

[2] A. Roemer and G. Fettweis: "Flow Graph Based Parallel Code Generation," in *Proc. 4th International Workshop on Software and Compilers for Embedded Systems* (Sept. 1999).

[3] D. Kastner and S. Winkel: "ILP-based Instruction Scheduling for IA-64," in *Proc. Workshop on Languages, Compilers and Tools for Embedded Systems*, vol. 36, no. 8, pp. 145–154

(Aug. 2001).
[4] R. Leupers: “Instruction Scheduling for Clustered VLIW DSPs,” in *Proc. 2000 International Conference on Parallel Architectures and Compilation Techniques,* pp. 291–300 (Oct. 2000).
[5] Texas Instruments: *TMS320C6000 Optimizing Compiler User’s Guide* (Mar. 2000).
[6] 小林, 石浦, 益井: “準ブール充足可能性判定によるクラスタ型 VLIW DSP の最適コードスケジューリング,” 信学技報 VLD2006-94 (Jan. 2007).
[7] 瀬戸, 藤田: “有限状態機械 (FSM) とシンボリック状態探索を利用したコード生成手法,” 情処論, vol. 43, no. 5, pp. 1235–1251 (May 2002).
[8] 瀬戸, 藤田, 浅田: “充足可能性判定を利用した最適コード生成手法,” 情処論, vol. 44, no. 5, pp. 1202–1205 (May 2003).
[9] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah: “Generic ILP versus Specialized 0-1 ILP: An Update,” in *Proc. International Conference on Computer-Aided Design*, pp. 450–457 (Nov. 2002).
[10] Texas Instruments: *TMS320C67x/67x+ DSP CPU and Instruction Set Reference Guide* (May 2005).