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

信学技報 VLD2006-94/CPSY2006-65/RECONF2006-65 (2007-1)
TECHNICAL REPORT OF IEICE.

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

小林　涼†　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: †{r-kobayashi,y-masui,ishiura}@ksc.kwansei.ac.jp

あらまし　本稿では, クラスタ型 VLIW DSP TMS320C62x のデータパスの詳細やデータ転送演算の挿入まで考慮した最適コードスケジューリングを, 準ブール充足可能性判定 (以下 PBSAT) により求める方法を提案する. ユニット毎のオペランドの非対称性, レジスタファイルの記憶容量, および主記憶やレジスタファイル間のデータ転送まで考慮した最適スケジューリング問題を定式化する. PBSAT は充足可能性判定に一次不等式の制約を追加したものであり, 記憶容量制約を効率的に表現し, かつ本稿のスケジューリング問題の解を高速に求めることができる.

キーワード　準ブール充足可能性判定, クラスタ型 VLIW DSP, コードスケジューリング, TMS320C62x

Optimum Code Scheduling for Clustered VLIW DSP Using Pseudo Boolean Satisfiability

Ryo KOBAYASHI†, Yuki MASUI†, and Nagisa ISHIURA†

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

Abstract　This paper proposes a method of finding optimum code scheduling for clustered VLIW DSP TMS320C62x considering the detailed configuration of its datapath and insertion of data transfer operations using Pseudo Boolean Satisfiability (PBSAT). We formulate an optimum code scheduling problem which takes the operand assymmetry for each unit, the capacity of the registerfiles, and the data transfer operations among the main memory and the registerfiles into account. PBSAT is an extension of the Boolean satisfiability problem (SAT) which can deal with linear inequality constraints. It enable us to express the registerfile capacity constraints succinctly and to solve the scheduling problem efficiently.

Key words　Pseudo Boolean Satisfiability, clustered VLIW DSP, code scheduling, TMS320C62x

1. はじめに

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る. 特に演算の並列実行が可能な VLIW (Very Long Instruction Word) 型 DSP は, 静的スケジューリングに基づく並列演算のコード最適化により優れた電力性能比を達成するが, 演算処理の高速化を実現するために複雑なデータパス構成を持つので, 品質の高いコードをいかに生成するかが重要な課題となる.

VLIW 型 DSP のコードスケジューリングに関する一般的な研究は, [1] をはじめとして数多く行われているが, DSP は様々な制約から固有の特殊なデータパス構成を持つものが多く, 具体的な DSP のコード生成/最適化に際しては, これらの制約まで考慮したコードスケジューリングを行う必要がある. 例えば Texas Instruments 社製 TMS320C62x [2] (以下 C62x) は, 8 つある実行ユニットが全て対称ではないので, 実行できる演算だけでなく即値オペランドやクロスパスの使用条件がユニット毎に異なる. [3] では, C62x をモデルとしたクラスタ型 VLIW DSP 用スケジューリングを simulated annealing により解いているが, ユニットの非対称性や, メモリやレジスタファイル間のデータ転送用資源まで考慮したスケジューリングは行っていない.

DSP のコードスケジューリングを考える上でもう 1 つ問題となるのが, 記憶要素とその間のデータ転送の扱いである. レジスタファイルの容量を考慮すると, スピル/リロードのコードまで含めた最適スケジューリングが必要となる. さらにクラスタ型 VLIW DSP では, データのクラスタ間移動まで考慮する

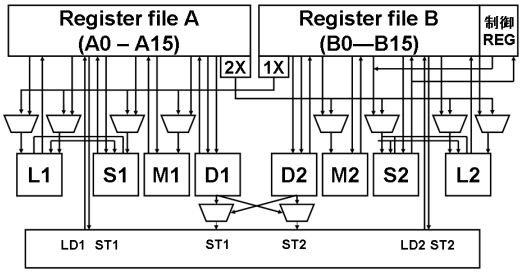
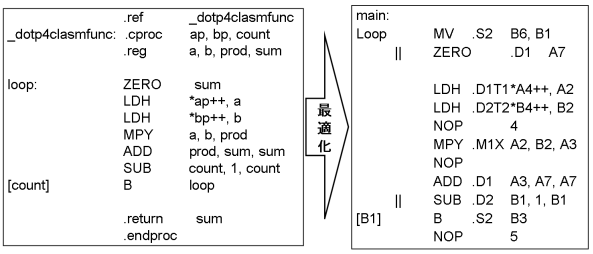


図 1　TMS320C62x のデータパス構成



(a) リニアアセンブリ　　(b) アセンブリ

図 2　リニアアセンブラ

ことが必要になる．[3] では, 依存関係のある演算のクラスタが異なる場合にクラスタ変更かデータ転送演算の挿入を行うが, データ転送演算の挿入は 1 つの演算に対して高々1 回で, レジスタファイルの記憶容量制約も考慮していない．[4] はデータ転送ではなく NOP 演算の挿入によりレジスタファイルの記憶容量制約を解決するが, データ転送演算の挿入に比べて実行サイクル数が増大する可能性がある．

記憶要素やデータ転送まで考慮したコード最適化手法としては [5] の方法がある．この手法はデータフローグラフ (以下 DFG) とデータパス情報から有限状態機械 (以下 FSM) を作り, その状態探索により最適なコードスケジューリングを求める．[5] ではこの状態探索を二分決定グラフ (以下 BDD) を用いて解くが, DFG のサイズが大きい場合やデータパスが複雑な場合に非現実的な計算時間が必要となる．[6] はこの FSM を有限時間展開し, 状態探索を和積形論理式 (CNF) の充足可能性判定 (以下 SAT) に帰着して解いている．近年の解法の進歩により, SAT は 0-1 ILP (0-1 Integer Linear Programming) よりも高速に解けるとされている [7]．しかし, SAT の節形式ではレジスタファイルの記憶容量制約の表現のサイズが大きくなるという問題があり, 大容量の汎用レジスタを持つ DSP への適用は必ずしも効率的ではない．

本稿では, [6] の考え方に基づき, C62x のデータパスの詳細やデータ転送演算の挿入まで考慮した最適コードスケジューリングを準ブール充足可能性判定 (以下 PBSAT) により求める方法を提案する．即値オペランドやクロスパスの使用条件を演算とユニットの組合わせ毎に定義できるようにし, レジスタファイルの記憶容量, および主記憶やレジスタファイル間のデータ転送まで考慮した最適コードスケジューリング問題を定式化する．PBSAT は SAT に一次不等式の制約を追加したものであり, 記憶容量制約を効率的に表現し, かつ本稿のコードスケジューリング問題の解を高速に求めることができる．

2. TMS320C62x とリニアアセンブラ

図 1 に C62x のデータパス構成を示す．C62x は 2 つのクラスタ (A, B) を持ち, 各クラスタは 1 つのレジスタファイル (RF) と 4 種類 (L, S, M, D) のユニットから成る．ユニットは合計 8 個あり, 最大で 8 演算を同時に実行できるが, ユニット毎に実行可能な演算が異なる．例えば, ロード/ストアは D ユニット, 乗算は M ユニットでしか実行できないが, 加算は L, S, D ユニットで実行できる．演算の実行サイクル数は 1〜6 で, ユニットにかかわらず演算の種類により決まる．レジスタの読み込みは演算の 1 サイクル目, レジスタへの書き込みは 1〜6 サイクル目で行う．演算に必要なユニットやパスの資源は 1 サイクル目にだけ使用する．

各ユニット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できる．クロスパスには RF B から RF A に転送する “1X” と RF A から RF B に転送する “2X” の 2 つがあり, それぞれ 1 サイクルに 1 つずつ使用できる．

クロスパスや即値の使用条件はユニット毎に異なる点に注意を要する．例えば, L ユニットは両方のオペランドでクロスパスや即値を使用できるのに対し M, S ユニットは第 1 オペランドでしかクロスパスや即値を使用できない．D ユニットは第 1 オペランドにのみ即値を使用できるがクロスパスは使用できない．さらに C62x 固有の問題としてオートインクリメント付きロード/ストア演算, 条件付実行命令, 分岐演算の扱いも考慮する必要がある．

本研究では C62x のリニアアセンブラの最適スケジューリング問題を扱う．リニアアセンブラは, 図 2(a) のようにプロセッサ上の命令と 1 対 1 で対応する演算の逐次実行のセマンティクスを持ち, 各演算の実行に必要なユニットやレジスタを指定しないアセンブリ (リニアアセンブリ) から, 図 2(b) の最適化されたアセンブリを出力するものである．コードスケジューラは演算の並列化スケジューリング, 資源割り当て, データ転送演算の挿入を行うが, 演算 (ニーモニック) の変換は行わないものとする．

3. コードスケジューリング問題

本稿で扱うコードスケジューリング問題は, プログラムの基本ブロックを表す「依存グラフ」と, プロセッサで実行可能な命令の情報を表す「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と定式化できる．レジスタファイルの容量を考慮し, 必要なデータ転送演算 (スピル, リロード, レジスタ間転送) の挿入まで含めた最適コードスケジューリングを求める．

3.1 依存グラフと命令パターン

依存グラフはプログラム中の 1 つの基本ブロックを表す．基本ブロック中の演算と値を表わす節点の集合をそれぞれ F, V, データ依存枝の集合を E_D, 順序枝の集合を E_S, 枝の集合を

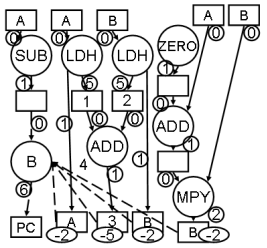


図 3　依存グラフ

表 1　命令パターン

演算		I(p)	ADD
ユニット		U(p)	L1
パス		X(p)	X1
遅延		d(p)	1
オペランド	1	m(p,1)	A
	2	m(p,2)	B
	3	m(p,3)	A

$E = E_D \cup E_S$ とする．データ依存枝はオペランドと演算の間にある直接的な依存関係を, 順序枝はその他の制約から生じる順序関係を表す．図 3 に依存グラフの例を示す．実線がデータ依存枝を, 破線が順序枝を表わす．依存グラフ $G = (V \cup F, E)$ は非巡回の有向 2 部グラフ ($E \subseteq V \times F \cup F \times V$) である．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．例えば図 3 の 4 の ADD 演算は 1 の値節点がオペランド番号 $o(e) = 1$ で遅延が 0, 2 の値節点がオペランド番号 $o(e) = 2$ で遅延が 0, 3 の値節点がオペランド番号 $o(e) = 3$ で遅延が 1 となる．データを格納する記憶要素の集合を $\mathcal{M}$ とし, $m \in \mathcal{M}$ の容量を $C(m)$ とする．C62x では $\mathcal{M} = \{A, B, M\}$ である．A, B はそれぞれ RF A, RF B, M は主記憶を表し, $C(A) = C(B) = 14$(注1), $C(M) = \infty$ とする．$V_I \subseteq V$ および $V_O \subseteq V$ はそれぞれ基本ブロックの入力節点, 出力節点の集合であり, 入力節点 v には入力が与えられる記憶要素 $i(v) \in (2^{\mathcal{M}} - \phi)$, 出力節点 v には出力を期待する記憶要素 $o(v) \in (2^{\mathcal{M}-\phi})$ がそれぞれ定義されている．例えば, $i(v) = \{A, M\}$ は入力節点 v のデータが RF A または主記憶 M に与えられていることを, $o(v) = \{A, B\}$ は出力節点 v のデータが RF A または RF B に格納されることを表す．$Q = \{(v_i, v_o) \mid v_i \in V_I, v_o \in V_O, v_i$ と v_oは同じ変数 $\}$ とする．$(v_i, v_o) \in Q$ は, ループに含まれる基本ブロックにおいて, 入力値 v_i と出力値 v_o が同一の変数であることを表す．

命令パターン集合 P の要素 p には, p が実行する演算の集合 $I(p)$, ユニットの集合 $U(p)$, パスの集合 $X(p)$, 遅延 $d(p)$ が定義されている．演算 $f \in F$ の結果を記憶要素 $m \in \mathcal{M}$ に格納する命令パターンの集合を $P_{f,m}$, 命令パターン p において j 番目のオペランドの値を格納する記憶要素を $m(p, j)$ とする．表 1 に命令パターンの例を示す．この命令は ADD 演算で, 使用ユニットは L1, パスはクロスパスの X1 を使用し, 遅延サイクル数は 1 である．オペランド 1 の記憶要素は A, オペランド 2 の記憶要素は B, オペランド 3 の記憶要素は A であることを表す．

(注1)：RF の容量は 16 だが, うち 2 つは用途が予約されているとする．

表 2　転 送 演 算

命令の種類	転送方向	ユニット	パス
スピル	RF A → M	D1	T1
スピル	RF B → M	D2	T2
リロード	M → RF A	D1	T1
リロード	M → RF B	D2	T2
レジスタ間転送	RF B → RF A	L1	X1
レジスタ間転送	RF B → RF A	S1	X1
レジスタ間転送	RF A → RF B	L2	X2
レジスタ間転送	RF A → RF B	S2	X2

S を転送演算の集合, $d(s)$ 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．転送演算の転送方向と使用するユニットとパスを表 2 に示す．

3.2　コードスケジューリング問題

t_{max} を実行サイクル数の上限とし, $T = \{1, 2, \cdots, t_{max}\}$ とする．本稿では 3 種類の 0-1 変数 $\alpha_{t,v,m}, \xi_{t,f,p}, \tau_{t,v,p}$ を用いる．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 をとるものとする．$\xi_{t,f,p}$ 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 をとるものとする．$\tau_{t,v,p}$ 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 をとるものとする．

制約条件は以下の 6 つである．

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

$$\begin{cases} \forall v \in V_I : & \sum_{m \in i(v)} \alpha_{0,v,m} = 1. \\ \forall v \in V_I, \forall m \in \mathcal{M} - i(v) : & \alpha_{0,v,m} = 0. \\ \forall v \in V - V_I, \forall m \in \mathcal{M} : & \alpha_{0,v,m} = 0. \end{cases}$$

(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．

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

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

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．

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

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

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．

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

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

(3) 資源制約: 同一サイクルで資源が高々1 つしか使用できないことを表わす. 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 $B_r^t = \{\xi_{t,f,p}, \tau_{t,v,p} \mid t \in T, f \in F, U(p) = r \vee X(p) = r\}$ とする.

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

(4) 代入制約:全ての演算は高々1 度しか実行しないことを表わす.

$$\forall f \in F \quad \forall m \in M: \quad \sum_{t \in T} \sum_{p \in P_{f,m}} \xi_{t,f,p} \quad \leqq 1.$$

(5) 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.

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

(6) 出力制約:
1. 出力節点 v の値が記憶要素 $o(v)$ に存在することを表す. $h_{t,v}$ をサイクル t で出力節点 v に値が存在すれば 1, そうでなければ 0 を表す中間変数とする.

$$\begin{cases} \forall v \in V_O: & h_{t_{max},v} = 1. \\ \forall t \in T, \forall v \in V_O: & h_{t,v} \rightarrow \bigvee_{m \in o(v)} \alpha_{t,v,m}. \end{cases}$$

2. 入力節点 $v_i \in V_I$ と出力節点 $v \in V_O$ が同じ変数ならば, 同じ記憶要素 $m \in M$ に格納しなければならない.
$\forall t \in T, \forall (v_i, v) \in Q, \forall m \in o(v)$:

$$h_{t_{max},v} \rightarrow (\alpha_{t,v,m} \rightarrow \alpha_{0,v_i,m}).$$

以上の制約を満たす $\alpha_{t,v,m}, \xi_{t,f,p}, \tau_{t,v,p}$ への割り当てを求めることにより, 実行可能なスケジューリングが得られる. 実行サイクル数の最小化は以下のように表せる.

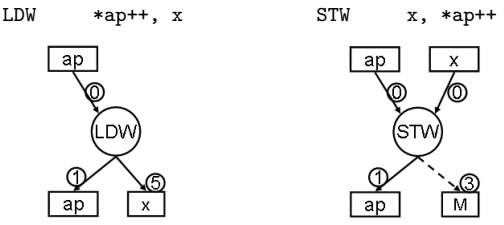
$$\text{maximize} \quad \sum_{t \in T} (\bigwedge_{v \in V_O} h_{t,v}).$$

3.3 TMS320C62x 固有の問題

3.3.1 オートインクリメント付きロード/ストア演算

C62x のほとんどの演算は 1 出力だが, オートインクリメント付きのロード/ストア演算はアドレスレジスタに対して read と write の両方を行うため, 図 4 のように多出力演算として扱う必要がある. 前節の制約式の (2) の 1 は,「ある 1 つの出力値を得るためには演算が実行されなければならない」ことを表しているので, 多出力の場合, これだけでは出力の数だけ演算が実行される可能性がある. そこで, 各演算は 1 度しか実行しないという制約を (4) 式で与えることによりこれを防ぐ.

また, アドレスレジスタは同一のレジスタであるため, 図 5 のようにオートインクリメントを含むロード/ストア演算間には, アドレスレジスタの RAW および WAR 依存を設定する必要がある.



(a) ロード演算　(b) ストア演算

図 4　オートインクリメント付きロード/ストア演算

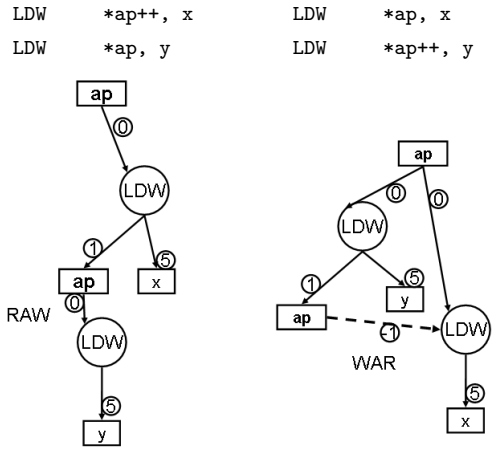
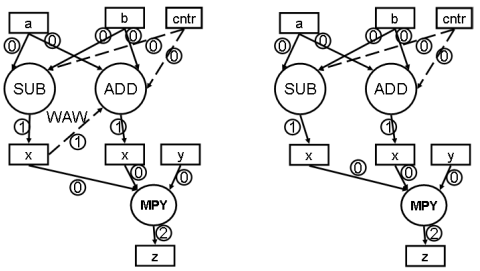


図 5　ロード演算間の依存関係

```
[cntr]    SUB   a, b, x
[!cntr]   ADD   a, b, x
          MPY   x, y, z
```

(a) 条件付実行命令



(b) 排他性を考慮しない場合　(c) 排他性を考慮した場合

図 6　条件付実行命令

3.3.2 条件付実行命令

C62x のように条件付実行命令を持つプロセッサでは, 複数の命令が同一のレジスタに値を書き込む場合注意が必要である. 図 6(a) の例を考える. SUB と ADD が同じオペランド x に結果を書き込み, MPY がそれを使用する. SUB と ADD の実行条件を意識せずに依存グラフを作成すると, x を rename して (b) の形となるので, x の WAW 依存を表わす枝が必要であり, SUB と ADD は同一サイクルにスケジューリングできない. しかし, この例のように条件が排他と判断できる場合には, WAW 依存が除去できるので (c) のように SUB と ADD を並列に実行できる.

```
LOOP:     ADD     a, b, g
          ADD     c, d,, h
          MPY     h, e, i
          SUB     cntr, 1, cntr
[cntr]    B       LOOP
```

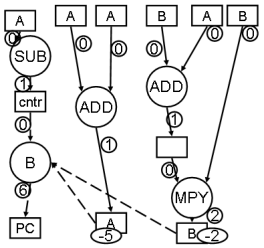


図 7　分岐演算の扱い

3.3.3　分岐演算の扱い

分岐演算は,

1) 分岐の条件判定に必要な値とのデータ依存を満たす
2) 他の全ての演算の終了後に (PC の更新が) 実行される
3) 遅延スロットが存在する

の 3 点を考慮してスケジューリングする必要がある. 例を図 8 に示す. 1) は依存グラフ作成時に設定される. 2) は全ての出力節点 (ただし, 分岐演算の PC への書き込み以外) から分岐演算に順序枝を設定する. 3) は, その順序枝に負の重みを設定することにより実現できる.

4. 実装と実験結果

4.1 準ブール充足可能性判定

3.2 (5) 式の記憶容量制約は, 各サイクル各記憶要素に対して 1 つの一次不等式で表現可能である. しかし, SAT の節形式でこれを表現した場合, 変数の数を n, 容量を k とすると, 節の数が ${}_nC_{k+1}$ となり, n が大きい場合には $O(n^k)$ となる. 中間変数を導入すれば節の数を $O(n^2)$ にできるが, n^2 個の中間変数が必要になる.

PBSAT は SAT の CNF に準ブール制約 (一次不等式) を追加したものである. SAT の BCP (Boolean Constraint Propagation), Random Restart, Backtracking 等の手法を準ブール制約の部分にも直接適用して高速に解探索を行う手法 [7] が提案されている.

4.2 実 験 結 果

3 節の最適スケジューリングを解く処理系を PBSAT のソルバ PBS Ver2.1 for Win [7] を用いて実装した. いくつかのプログラムに対して実験を行った結果を表 3 に示す. リニアアセンブリプログラムはユニット, パスの指定やレジスタファイルの指定を行わないものを入力とした. “演算数” はプログラム中で最も大きい基本ブロックの演算数を表す. “サイクル数” はアセンブリコードの実行サイクル数で, “CCS” は Code Composer Studio Ver2.0 で最適化オプション (–O1) によりリニアアセンブルを行い実行した結果である. “本手法” はスケジューリングで得た結果を元に手動で記述をしたアセンブリコードを実行した結果である. “CPU” は本手法のスケジューリングに要した

表 3　実 験 結 果

プログラム	演算数	サイクル数		CPU
		CCS	本手法	[sec]
s3_dotp	8	358	358	13.3
s5_wvec	14	559	509	37.2
s6_iir	11	1208	1208	15.5
s7_if	9	200	200	10.0

計算時間 (Cygwin ver.2.05b.0, Intel Celeron 2.6GHz, メモリ 768MB) である.

演算数が 15 程度の小規模なプログラムを数十秒で解くことができた. 4 つのプログラムのうち 3 つで CCS の生成するコードが最適と判明した. s5_wvec では本手法の実行サイクル数が CCS に比べて 50 サイクル減少した.

5. む す び

本稿では, TMS320C62x のデータパスの詳細やデータ転送まで考慮した最適コードスケジューリング問題の定式化を行い, これを PBSAT により解く手法を提案した. 本手法は基本ブロック単位で高速に最適コードスケジューリングを行うことができる.

本手法のループスケジューリングへの拡張や, 大規模なプログラムに適用するための近似解法の開発が今後の課題である.

謝　　辞

本研究に際し, リニアアセンブラの開発に支援を頂いた野垣内聡氏, CCS に関するアドバイスを頂いた井上裕介氏, 山本陽平氏をはじめ, 支援と助言を頂いた石浦研究室の諸氏に感謝します.

文　　献

[1] G. Desoli: “Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach”, Technical Report HPL-98-13, Hewlett-Packard Laboratories, (Jan. 1998).
[2] Texas Instruments: *TMS320C6000 Optimizing Compiler User's Guide* (Mar. 2000).
[3] R. Leupers: “Instruction Scheduling for Clustered VLIW DSPs,” in *Proc. International Conference on Parallel Architectures and Compilation Techniques*, pp. 54–59 (Aug. 2001).
[4] Daniel Kastner, Sebastian Winkel: “ILP-based Instruction Scheduling for IA-64,” in *Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, vol. 36, no. 8, pp. 145–154 (Aug. 2001).
[5] 瀬戸謙修, 藤田昌宏: “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,” 情報処理学会論文誌, vol. 43, no. 5, pp. 1235–1251 (May 2002).
[6] 瀬戸謙修, 藤田昌宏, 浅田邦博: “充足可能性判定を利用した最適コード生成手法,” 情報処理学会論文誌, vol. 44, no. 5, pp. 1202–1205 (May 2003).
[7] F. A. Aloul, A. Ramani, l. 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).