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 SPXK5 の条件実行を考慮した 最適コードスケジューリング

山本　哲也† 　石浦菜岐佐† 　久村　孝寛†† 　池川　将夫†† 　今井　正治†††

† 関西学院大学 理工学部　〒 669-1337 兵庫県三田市学園 2-1
†† 日本電気株式会社 システム IP コア研究所　〒 211-8666 神奈川県川崎市中原区下沼部 1753
††† 大阪大学大学院情報科学研究科　〒 565-0871 吹田市山田丘 1-5

あらまし　本稿では, ディジタル信号処理プロセッサ SPXK5 に対し, 詳細なデータパス構成および命令の条件実行まで考慮した最適コードスケジューリングの手法を提案する. SPXK5 は画像処理や通信におけるディジタル信号処理を効率良く実行するための命令セットを有する VLIW (Very Long Instruction Word) 型 DSP で, 1 実行パケットで最大 4 演算を並列に実行できる. 本稿のコードスケジューリングは, SPXK5 のアセンブリプログラムの基本ブロックを表すデータフローグラフ (DFG) を入力として, 実行パケット, 演算器, レジスタ, フォワーディング等に関する制約を考慮して, 0-1 線形整数計画法によって実行サイクル数を厳密に最小化するコードを求める. また, 条件実行演算のスケジューリングは条件の合併・分割, 条件の排他性も考慮した最適化を行う. 準ブール充足可能判定ソルバー PBS を用いて解を求めた結果, 20 演算程度の DFG に対して現実的な時間内で最適解を求めることができた.
キーワード　VLIW 型 DSP, SPXK5, コードスケジューリング, コード最適化, 条件実行

Optimum Code Scheduling for VLIW DSP SPXK5 considering Conditional Execution

Tetsuya YAMAMOTO†, Nagisa ISHIURA†, Takahiro KUMURA††, Masao IKEKAWA††, and Masaharu IMAI†††

† Kwansei Gakuin University,　Gakuen 2–1, Sanda, Hyogo, 669–1337, Japan
†† NEC Corporation,　Simonumabe 1753, Nakahara-ku, Kawasaki, Kanagawa, 211-8666, Japan
††† Osaka University,　Yamadaoka, Suita, Osaka, 565–0871, Japan

Abstract　This article presents an optimum code scheduling method for digital signal processor SPXK5 taking account of its architectural details and conditional execution. SPXK5 is a VLIW (very long instruction word) processor dedicated for digital signal processing application which can execute up to four operations per execution packet. The proposed scheduling method takes a DFG (dataflow graph) representing a basic block as input and attempts to find the optimum code, in terms of the execution cycles, by solving 0-1 integer linear programming taking all the architectural issues into account such as the structural constraints regarding the pipeline, the size of the execution packet, the capacity of the register files, etc. It also exploits grouping/ungrouping and exclusiveness of predicated operations to optimize the code. In a preliminary experiment an implemented scheduler using a pseudo Boolean satisfiability solver PBS successfully found the optimum scheduling for DFGs with about 20 operations within a feasible CPU time.
Key words　VLIW DSP, SPXK5, code scheduling, code optimization, condition execution

1. はじめに

近年, 音声・画像データを扱う組込みシステムには高いメディア処理性能が求められているが, 厳しい電力・コストの制限から, プロセッサが利用可能な資源には強い制約が課せられる. ディジタル信号処理プロセッサ (DSP) は、デジタル信号の処理に特化したプロセッサであり, 携帯電話や家電, オーディオ機器等様々な機器に組み込まれて利用されている.

SPXK5 [1] は NEC エレクトロニクス社の DSP で, ビデオ/オーディオ信号の MPEG4 エンコード/デコード処理を効率的に実行できる機能を備えおり, 携帯電話, PDA, デジタルビデオカメラなどの携帯機器の VLSI にコアとして搭載されている. SPXK5 は VLIW 型アーキテクチャを持ち, 最大 4 並列の演算の実行が可能である.

VLIW プロセッサの性能を引き出し, 高速化・低消費電力化を達成するためには, 効率の良いコードを生成することが課題となる. VLIW プロセッサのコードスケジューリングに関する研究は [6]〜[8] をはじめ数多く行われているが, 効率の良いコードの生成を行うためには個々のプロセッサの構造まで考慮に入れる必要がある. [3] は, Texas Instruments 社製 TMS320C62x に対し, 最適コードスケジューリングを求める手法を提案している.

本論文では [2] [3] の考え方を基に, SPXK5 に対して最適コードスケジューリングを求める方法を提案する. プログラムの基本ブロックを表すデータフローグラフ (DFG) を入力として, 実行パケットの制約, 条件付実行命令等の SPXK5 特有の問題や転送演算の挿入まで考慮したスケジューリングを 0-1 整数計画問題として定式化する.

提案手法を実装した結果, 約 20 演算程度までの基本ブロックに対して解が数十秒で得ることができた.

以下, 第 2 章では SPXK5 の詳細と, スケジューリング時に考慮すべき点について述べ , 第 3 章で提案手法の概要, 第 4 章ではスケジューリング問題の定式化を示す. 第 5 章では実装・実験結果を示し, 第 6 章でまとめと今後の課題を述べる.

2. SPXK5

2.1 データパス構成

図 2.1 に SPXK5 のデータパス構成の概要を示す. SPXK5 は 7 つの演算ユニットとレジスタ群からなる.

演算器には次の 4 種類がある.

- 算術論理演算ユニット (ALU) × 2 個

加減算, シフト演算, 論理演算を行うユニットで, これらの演算を以下では ALU 演算と呼ぶ.

- 積和演算ユニット (MAC) × 2 個

16 ビット× 16 ビット乗算, 40/16 ビット乗塁算 (16 ビット× 16 ビット乗算の結果を 40 ビット精度で累算する) を行うユニットで, これらの演算を以下では MAC 演算と呼ぶ.

- ロード/ストア・ユニット (DAU) × 2 個

メモリアクセス演算を行うユニットで, これらの演算を以下では DAU 演算と呼ぶ.

- システム・ユニット (SCU) × 1 個

分岐演算, ループ制御, 条件付実行演算等を行うユニットで, これらの演算を以下では SYS 演算と呼ぶ.

SPXK5 は 17 種類のレジスタを持つ. 8 個ある汎用レジスタ (R) は 40 ビットのデータ幅を持ち, 命令によっては下位 16bit, 中位 16bit および上位 8bit を個別に命令の対象とすることができる. DP, DN, DM, DBASE はアドレッシングに用いられる. DP は 32 ビット幅のアドレスレジスタで, DN はその DP に対応するオフセットであり, それぞれ 8 個搭載している. 2 つある DM も DP の更新の際に用いられる. DBASE は直接アドレッシングを行う際に使用されるベースレジスタである. システムレジスタはループやスタック, 割り込みの制御等で使用される.

SPXK5 の命令には, 16bit 命令と即値を扱う 32bit 命令がある. 並列実行する命令の塊を実行パケットと呼び, 実行パケットの大きさは最大で 64bit であるので, SPXK5 では最大 4 つの演算を並列に実行することが可能である.

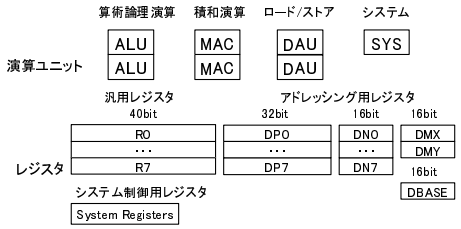


図 1　SPXK5 のデータパス構成

2.2 条件実行

SPXK5 の条件実行は TMS32062x や ARM とは異なり, 条件判定を行う「COND 演算」とその真偽に依存して実行される演算を同じ実行パケットで並列に実行する. また, 条件実行が可能な演算は, ALU 演算および MOVE 演算に限定される.

図 2 は条件実行を含んだアセンブリコードとそのスケジューリング例である. アセンブリの演算はコンマで, 実行パケットはセミコロンで区切る. 例えば (a) の左側の 2 行目の「if (r4>0), r0=r5;」は, COND 演算「if (r4>0)」が真の場合に「r0=r5」が実行されることを表す.

図 2(a) において, 左側の 3 〜4 行目の演算は同じ条件で実行されるが, ALU 演算は 1 実行パケットで 2 つまで実行できるので, 右側の 3 行目のように並列化できる. 逆に, 図 2(b) の例のように条件実行される実行パケットを分割した方がサイクル数が短くなることもある. 左側の 1 行目で 2 つの ALU 演算を条件実行しているが, パケットが 64 となるためこれ以上他の演算を同時に実行することはできない. しかし右側の 1 〜2 行目のように 2 演算を 2 つの実行パケットに分けると, 他の積和演算 (ALU 演算以外は条件の修飾を受けない) を並列実行できるようになる.

また, 条件の排他性を利用することによりスケジューリングを最適化することもできる. 図 2(c) 左側 2 〜3 行目はいずれも r2 に書き込みを行っている. 通常同一レジスタへの書き込みは直列化しなければならないが, 2 つの条件は排他と判定できるのでその制約を解除でき, 右側のようにスケジューリングすることができる.

3. 提案スケジューリング手法の概要

3.1 コードスケジューリング問題

コードスケジューリングには, スーパーブロック, ハイパーブロックを対象とするものもあるが, 本稿ではプログラムの各基本ブロックを表すデータフローグラフ (DFG) を入力として, サイクル数が最小になるように演算の開始サイクルを決定する

```
1: r0=r1, dp0=dp1+dp2;
2: if (r4>0), r0=r5;
3: if (r4<0), r0=r3+r6;
4: if (r4<0), r2=r1;
```

→

```
1: r0=r1, dp0=dp1+dp2;
2: if (r4>0), r0=r5;
3: if (r4<0), r0=r3+r6, r2=r1;
```

(a) 同じ条件の演算の並列化

```
1: if(r4!=0), r5=r1+r2, r6=r1+10;
2: r0=r0+r1L*r1H;
3: r0=r0+r2L*r2H;
4: r0=r0+r3L*r3H;
```

→

```
1: if(r4!=0), r5=r1+r2, r0=r0+r1L*r1H;
2: if(r4!=0), r6=r1+10, r0=r0+r2L*r2H;
3: r0=r0+r3L*r3H;
```

(b) 同じ条件の演算の直列化

```
1: if (r1<0), r0=r4,    dp2=dp5+dp3;
2: if (r1>0), r2=r5+10;
3: if (r1<0), r2=r5+r4;
```

→

```
1: if (r1<0), r0=r4, dp2=dp5+dp3, r2=r5+r4;
2: if (r1>0), r2=r5+10;
```

(c) 条件の排他性の利用

図 2　条件実行命令のスケジューリング例

問題を扱う. ただしレジスタファイルの容量を厳密に扱うものとし, 必要に応じた転送演算の挿入まで考慮してスケジューリングを行う(注1).

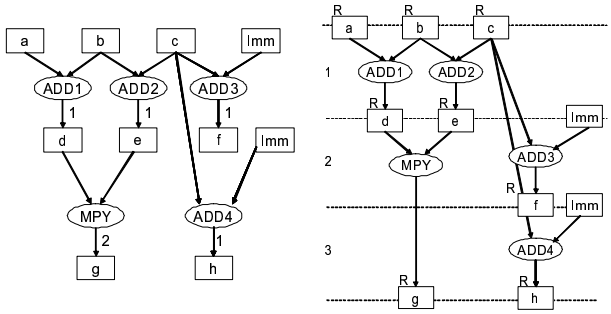
図 3 にスケジューリングの例を示す. 図 3 の (a) はアセンブリから抽出したデータフローグラフの例であり, 円形の節点は演算, 長方形の節点は値を表す. 値 a, b, c は入力値であり, これ以前の基本ブロックで計算された値である. 値 f, g, h は出力値であり, 以降の基本ブロックで利用される値である. また, imm は即値データを表す. オペランドと演算の間にあるデータ依存関係を表す枝を依存枝と呼び, 実線で表す. 各枝には遅延を定義する. 値から演算への依存枝には遅延 0 を, 演算から値への枝にはその演算の遅延 (ADD は 1, MPY は 2) を設定する. (b) は (a) に対してスケジューリングを行った例である. 演算が行われる時刻, それぞれの値が格納される記憶要素が決定されている.

スケジューリングを行う際には, 種々の制約を考慮しなければならない. 例えば図 3 の ADD3 という演算は, 同時に実行できる ALU 演算は 2 つまでという制約のため, ADD1, ADD2 と同時に実行することができない. ADD4 は ADD3 と ADD4 の命令語長が 32 ビットなので, 実行パケットの 64 ビット以内に収まらず, 2 サイクル目には実行できない.

3.2　0-1 整数計画問題による定式化

本手法では, レジスタファイルの記憶容量やデータ転送演算まで考慮したスケジューリングを行う. 転送演算の挿入を許すと DFG は動的に変化することになり, 単純な資源制約スケジューリングでは解を得る事ができなくなる. そこで [2] では, DFG とデータパス情報から実行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 (FSM) を生成し, 初期状態から最終状態への最短経路を探索することにより, 最小サイクル数でのスケジューリングを求めている. 本手法ではこの考え方を基にして,

(注1)：各値に対するレジスタ割当てを陽には行わないが, レジスタファイルの容量を厳密に考慮しているため, スケジューリング後に各値を格納するレジスタは必ず決定できる.



(a) データフローグラフ (DFG)　　(b) スケジューリング

図 3　SPXK5 のコードスケジューリング例

SPXK5 特有の制約を考慮した 0-1 線形計画法によってコードスケジューリング問題を解く.

3.3　汎用レジスタの扱い

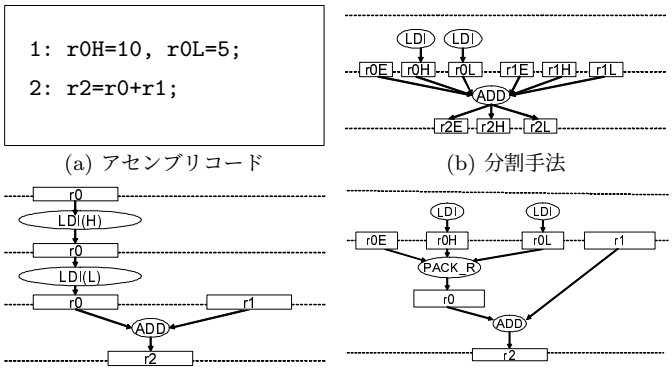
SPXK5 の汎用レジスタは, 40 ビットのレジスタとしてもアクセスできるが, 上位, 中位, 下位の 8 ビット, 16 ビットの部分レジスタに独立にアクセスしたり, 上中位, 中下位の 24 ビット, 32 ビットにアクセスすることが可能であり, その扱いが課題となる.

図 4 は, 汎用レジスタの部分レジスタに対するアクセスの例 ((a)) と, その DFG での表現 ((b)〜(d)) である.

(b) は汎用レジスタを E, H, L の三種類の別々のレジスタとして扱う方法である. この方法では r2=01+r1 という演算を行うには, 加算演算は 6 入力 3 出力となり, 各値ごとに転送演算や依存関係が存在することになるため, 変数, 式の数が増加してしまう.

(c) は, E や HL などを全て R として扱う方法である. 例えば H 部分に値を書き込む時には, R 全体を出力先と考える. 変数の数は抑制できるが, 同じ R の別部分に同時に書き込むスケジューリングが不可能になる. 図 4 では R の H, L 部の両方に即値設定を行っているが, (c) は DFG 上では H も L も同じ R として扱っているため 1 サイクルに 1 回ずつ設定することになり, 最適なスケジューリング結果が得られない.

これら問題を解決する手段として, 本稿では (d) の部分レジスタのサブクラスと仮想的なレジスタクラスの変換演算を用いる方法を提案する. 即ち, 汎用レジスタには R, REH, RHL, RE, RH, RL の 6 種類のサブクラスが存在し, 必要なサブクラス間の変換を遅延 0 の仮想的な演算で行う. 例えば (d) の ADD 演算では r0 が入力として必要なので, r0H, r0L, r0E を入力として r0 を出力する仮想的な演算「PACK_R」を挿入する. この方法では (b) に比べ変数の数が少なくなるため, 計算量を抑制しつつ最適解を求めることが可能となる.



(a) アセンブリコード (b) 分割手法

(c) 統合手法 (d) 提案手法

図 4　汎用レジスタの部分レジスタアクセスの扱い

3.4　条件実行の扱い

図 5 に条件実行の DFG の例を示す. 破線はデータ依存関係以外の制約による順序関係を表す「順序枝」である. COND 演算と ADD 演算の間に双方向の遅延 0 の順序枝を設定することによって, これらの演算を同時に実行することを表現する. SPXK5 では, COND 演算と同じ実行パケットの ALU 演算は全て条件実行される. そのため COND 演算と遅延 0 の順序枝で繋がっていない ALU 演算は, COND 演算と同一サイクルで実行してはならない. また, MAC 演算や DAU 演算は条件実行不可である.

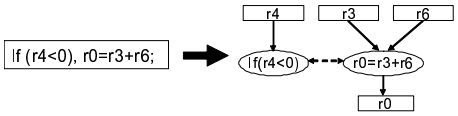
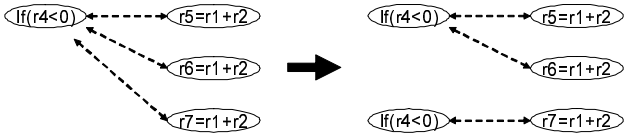


図 5　条件実行命令の DFG における表現

条件実行演算の最適スケジューリングを行うためには, 図 6 の (a), (b) のように, スケジューリングの前後で 1 つの条件によって修飾される演算数の変化を伴う変換を扱う必要がある.

本手法では, このようなスケジューリングに対応するために, アセンブリコードから DFG を生成する段階で以下の処理を行う. DFG 上の COND 演算は, この段階では各条件実行演算と長さ 0 の順序枝で繋ぐ. 図 6 の (a) のように同じ COND 演算 (オペランドの値が更新されないもの) は必ず 1 つの節点で表すものとする. DFG 中の演算に対しては, 「演算は 1 度のみ実行する」という制約 (詳細は 4. の式 (13)) を課すが, COND はその対象から外し, COND 演算の数の最小化を目的関数に加えることによって, COND 演算は必要最低限の数だけ実行される



(a) DFG における表現 (b) 実行回数の最適化

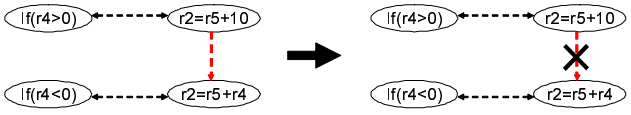
図 6　COND 演算の扱い

ようにする (図 6).

また, 同じレジスタに結果を出力する演算が複数ある場合, コードに現れる順番を保ってスケジューリングしなければならない. これは, 正しい動作を保障するための順序枝を設定することによって実現するが, 各演算の実行条件が排他的であるなら, 順序枝は設定せずスケジューリングの自由度を高めることができる.

条件の排他性は DFG 構築時に判定する. COND 演算は汎用レジスタと 0 の比較および, 汎用レジスタの E 部分に符号以外のビットが入っているかの判定のみなので, 排他性は容易に判定できる.

図 2 の (c) の左側の 2 行目と 3 行目の条件実行演算は, 通常であれば図 7 の (a) のように順序枝を設定するが, 条件が排他と判定できるので, 図 7 の (b) のように順序枝を削除する. この結果, (c) の右側のようなスケジューリングが可能となる.



(a) 排他性を考慮する場合 (b) 排他性を考慮しない場合

図 7　条件の排他性の考慮

4.　スケジューリング問題の定式化

本節では, スケジューリング問題の定式化を示す. まず, 定式化に現れる集合, 要素の定義を以下の様に行う.

- 演算を表す節点の集合を F とする. ALU 演算の集合を F_A, MAC 演算の集合を F_M, DAU 演算の集合を F_D, SYS 演算の集合を F_S とし, $F = F_A \cup F_M \cup F_D \cup F_S$ となる. $l : F \to \{16, 32\}$ は演算 $f \in F$ の命令長であり, 演算 f が 16bit 命令のときは $l(f) = 16$, 32bit 命令の時は $l(f) = 32$ となる.
- 記憶要素の種類の集合を N とする. $N = \{n_{r40}, n_{r24}, n_{r32}, n_{r8}, n_{r16}, n_{dp}, n_{dn}, n_{dm}, n_{db}, n_m\}$ である. n_{r40} は汎用レジスタ R, n_{r24} は REH, n_{r32} は RHL, n_{r8} は RE, n_{r16} は RH 又は RL である. n_{dp} はデータポインタ, n_{dn} はインデックスレジスタ,n_{dm}はモジュロレジスタ,n_{db} はベースレジスタ, n_m はメモリを表す. $c : N \to \mathcal{N}$ は記憶要素の容量を表し, $c(n_{r16}) = 16, c(n_{r8}) = c(n_{dp}) = c(n_{dn}) = 8, c(n_{dm}) = 2, c(n_{db}) = 1, c(n_m) = \infty$ である.
- 値を表す節点の集合を V とする. $A : V \to \{2^N - \phi\}$ は $v(\in V)$ が格納される記憶要素の集合を表す. $V_I(\subset V)$ は入力に用いる値を表す節点の集合, $I(\subset A) : V \to \{2^N - \phi\}$ は $v(\in V_I)$ が格納される記憶要素の集合とする.

同様に $V_O(\subset V)$ を出力に用いる値を表す節点の集合,

$O(\subset A) : V \to \{2^N - \phi\}$ は $v(\in V_O)$ が格納される記憶要素の集合である.

- 同一の記憶要素に格納する値の集合を $G(\subseteqq 2^V)$ と表す.
- $E_D(\subseteqq F \times V \cup V \times F)$ を値と演算間のデータ依存関係を表す枝の集合とする. $d : E_D \to \mathcal{N}$ は $e(\in E_D)$ に定義される遅延, $o : E_D \to N$ は演算 $f(\in F)$ を行う時, 値 $v(\in V)$ が格納されるべき記憶要素の集合を表す.
- $E_S(\subseteqq F \times F)$ は演算間の順序関係を表す枝の集合とする. $d : E_S \to \mathcal{N}$ は $e(\in E_S)$ に定義される遅延を表す.
- $s_{m,n}$ を記憶要素 $m(\in N)$ から記憶要素 $n(\in N)$ への転送命令, $d(s_{m,n})$ を $s_{m,n}$ に定義される遅延とする.
- T は t_{max} を実行サイクル数の上限とするサイクル数 t の集合 $T = \{0, 1, ..., t_{max}\}$ である.

定式化では次の 3 種類の変数を用いる. 変数の定義は以下の通りである.

- $x_{f,t}$ は t サイクル目に演算 $f(\in F)$ が行われると 1, そうでないと 0 をとる変数である.
- $r_{v,n,t}$ は t サイクル目に値 $v(\in V)$ が記憶要素 $n(\in N)$ に格納されていると 1, そうでないと 0 を取る変数である.
- $m_{v,s_{m,n},t}$ は t サイクル目に値 $v(\in V)$ が記憶要素 $n(\in N)$ から $m(\in N)$ に転送されると 1, そうでないと 0 をとる変数である.

次に SPXK5 のコードスケジューリングに必要な制約条件を示す.

(a) 入力制約: 時刻 0 に入力値 v が存在することを表す. 下記式 (1) で時刻 0 に記憶要素 n に格納されている値を表し, 式 (2) で入力値が格納されてない記憶要素を表す. また, 式 (3) では時刻 0 には入力値以外の値は存在しないことを表している.

$$\forall v \in V_I : \quad \sum_{n \in I(n)} r_{v,n,0} = 1, \tag{1}$$

$$\forall v \in V_I, \forall n \in A(v) - I(v) : r_{v,n,0} = 0, \tag{2}$$

$$\forall v \in V - V_I, \forall n \in N : r_{v,n,0} = 0. \tag{3}$$

(b) 依存制約: 演算と変数間の依存関係を表す. 式 (4) は演算 f が行われるための条件を表す. 演算を時刻 t に行うためには, 入力となる値が $t - d(v, f)$ 時に適切な記憶要素に格納されていなければならない.

$$\forall t \in T, \forall (v, f) \in E_D : \tag{4}$$
$$x_{f,t} \to \bigwedge_{(v,f) \in E_D} r_{v,o((v,f)),t-d(v,f)}.$$

式 (5) は記憶要素 n に値 v が格納されているための条件である. 時刻 t に値 v が記憶要素 n に存在するためには, 演算 f か転送演算 $s(m, n)$ が行われるか, 時刻 $t - 1$ に v が n に存在していなければならない.

$$\forall t \in T, \forall (f, v) \in E_D, \forall n \in A(v) : \tag{5}$$
$$r_{v,n,t} \to x_{f,t-d((f,v))} \vee m_{v,s_{m,n},t-d(s_{m,n})} \vee r_{v,n,t-1}.$$

式 (6) で転送演算が実行される条件を示す. 時刻 t に値 v を記憶要素 m から n に転送する. そのとき, 記憶要素 m に値 v が存在しなければ実行できない.

$$\forall t \in T, \forall v \in V, \forall n, m \in N : \quad m_{v,s_{m,n},t} \to r_{v,m,t}. \tag{6}$$

(c) 資源制約: 使用可能な資源の制約を表す. 式 (7) は実行パケットの制約を表す. 同時に実行する演算のビット数の合計値は 64 以下でなければならない.

$$\forall t \in T : \sum_{f \in F} l(f) x_{f,t} \leqq 64. \tag{7}$$

式 (8) は演算器の数の制約を表す. t サイクル時に同時に使用する演算は ALU, MAC, DAU 演算は 2 つ, SYS 演算は 1 つまででなければならない.

$$\forall t \in T : \quad \sum_{f \in F_A} x_{f,t} \leqq 2, \sum_{f \in F_M} x_{f,t} \leqq 2, \tag{8}$$
$$\sum_{f \in F_D} x_{f,t} \leqq 2, \sum_{f \in F_S} x_{f,t} \leqq 1.$$

(d) 記憶容量制約: レジスタファイルと主記憶の容量を表す. ただし, n_{40} は $n_8 \times 1, n_{16} \times 2$, n_{24} は $n_8 \times 1, n_{16} \times 1$, r_{32} は $n_{16} \times 2$ だけそれぞれ一度に使用するものとする.

$$\forall t \in T, \forall n \in N - \{n_{40}, n_{24}, n_{32}\} : \sum_{v \in V} r_{v,n,t} \leqq c(n). \tag{9}$$

(e) 順序制約: 演算間の距離を表す. 式 (10) で演算 f_1 が演算 f_2 より $d(f_1, f_2)$ サイクル以上離れて行われることを表し, 式 (11) によって条件実行を行わない演算は *cond* 演算と実行時間を離すことを表す.

$$\forall t \in T, \forall (f_1, f_2) \in E_S : \tag{10}$$
$$x_{f_2,t} \to \bigvee_{i=0}^{t-d(f_1,f_2)} x_{f_1,t-d(f_1,f_2)-i}.$$

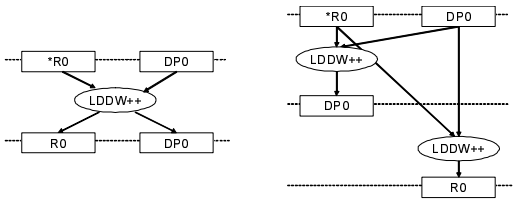
$$\forall t \in T, \forall \neg((f_{cond}, f) \in E_S), f \in F_A : \tag{11}$$
$$x_{f_{cond},t} \to \overline{x_{f,t}}.$$

(f) 代入制約: 演算結果を格納するレジスタの制約を表す. ソースオペランドとデスティネーションオペランドが同じ場合, 演算の結果を格納するときはソースとなった値は上書きされ消えなければならない. 次式によって, 同じレジスタに 2 重に値を格納しないようにする.

$$\forall t \in T, \forall (v_1, v_2) \in G, \forall (v_1, f), (f, v_2) \in E_D : \tag{12}$$
$$r_{v_1,o((v_1,f)),t} \wedge r_{v_2,o((f,v_2)),t} \to \overline{x_{f,t-d((f,v_2))}}.$$

(g) 回数制約: DFG 中の COND 演算以外の演算は全て 1 度だけ行われる. オートインクリメントロード/ストア演算等の多値出力演算を実行する場合, 図 8 の (b) のように 1 回の演算で全ての値が出力されない状態を防ぐためである. F_{cond} は COND 命令の集合とする.

$$\forall f \in F - F_{cond} : \sum_{t \in T} x_{f,t} = 1. \tag{13}$$



(a) 正しいスケジューリング　(b) 誤ったスケジューリング

図 8　多値出力演算のスケジューリング

(h)　出力制約: 実行可能解を得るための条件を表す. 実行可能解を得るには, 全ての出力値 v が t_{max} サイクル目に記憶要素 O_v に存在することが条件となる.

$$\forall v \in V_O : \quad \sum_{n \in O(n)} r_{v,n,t_{max}} = 1. \tag{14}$$

式 (15) は目的関数を表す. t サイクル目に出力値が全て何れかの記憶要素に存在するとして, $t_{max} - t$ の値を最大にすることで解を求める.

$$maxmize \sum_{t \in T} (\bigwedge_{v \in V_O} (\bigvee_{n \in O(v)} r_{v,n,t})). \tag{15}$$

5. 実装と実験結果

前章の定式化に基づき, SPXK5 のコードスケジューリングを行うシステムを実装した. システムは次に示す 4 つのフェーズからなる.

（1） スケジューリングの対象となるアセンブリコードを入力とし, その DFG を生成する.

（2） DFG から制約式のデータを生成する.

（3） 生成した式をソルバーへ入力し, 解を出力する.

（4） スケジューラの出力結果を解析し, アセンブリコードとして出力する.

今回, 命令セットのデータと DFG のデータ, 制約式生成スクリプト, 計算結果解析スクリプトの実装には Perl 5.8.7-5 を, ソルバーには準ブール充足可能性判定を行う PBS (v2.1 for Win) [4] を使用した.

スケジューリングの実験結果を表 1 に示す. 表 1 の一番左は入力に用いた DFG で, in_pr10 は 10 次元, in_pr15 は 15 次元ベクトル同士の内積計算の DFG, matrix2 は 2 次元, matrix3 は 3 次元正方行列同士の積計算の DFG である. matrix2_3 は 3 つの 2 次元正方行列の積計算を行う DFG である. その DFG の演算節点の数, この DFG のスケジューリング得た最小サイクル数, 計算時間を表している. 実験は CPU が Intel Core 2 2.00 GHz, メモリは 1 GB, Cygwin ver.1.5.24 上で行った.

この実験結果では, matrix3 以外の DFG (演算数 16 以下) では, 実行サイクル数を 2 秒以内に厳密に最小化するスケジューリングが得られた. 3 次元正方行列の積の計算では, 値の数がレジスタ容量より多いためスピル/リロードが多く発生する. この DFG 以外ではスピル/リロードは発生していないため, これが求解を困難にしている一因と考えられる. 信号処理でよく使われる DFT や DCT の計算では, 50 以上の演算を持つ基本ブロックもしばしば見らるため, この問題を解決することが 1 つの課題であると考える.

表 1　実 験 結 果

DFG	演算数	最小サイクル数	計算時間 [秒]
in_pr10	10	11	1.88
in_pr15	15	16	0.64
matrix2	8	8	0.44
matrix3	27	N/A	$>$ 1000.00
matrix2_3	16	13	0.55

6. む す び

本論文では SPXK5 を対象とした最適コードスケジューリング手法を提案した. 本手法では, [2] の考え方に基づき, SPXK5 の特性, 条件実行と条件の排他性まで考慮し, 基本ブロックに対する最適スケジューリングを求められるようにした. 今後は, スケジューリング可能な演算の数を増やすために, 速度の向上が重大な課題である. また, よりスケジューリングの効果を発揮させるために, ループスケジューリングにも取り組む必要がある.

謝辞

本研究に関してご議論・ご助言頂いた大阪大学の武内良典准教授, 稗田拓路氏, 渡辺愛子氏, NEC システム IP コア研究所の小林悠記氏, ならびに関西学院大学石浦研究室の諸氏に感謝します.

文　献

[1] T. Kumura, M. Ikekawa, M. Yoshida, and I. Kuroda, “VLIW DSP for mobile applications,” *IEEE Signal Processing Magazine*, vol. 19, no. 4, pp. 10–21 (July 2002).

[2] 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, ”情報処理学会論文誌, vol. 44, no. 5, pp. 1202–1205 (May 2003).

[3] 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).

[4] F. A. Aloul, A. Ramini, 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).

[5] David A. Patterson and John L. Hennessy: *Computer Organization & Design*：*The Hardware/software Interface*, Morgan Kaufmann (1997).

[6] 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).

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

[8] 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). in *Proc. 4th International Workshop on Software and Compilers for Embedded Systems* (Sept. 1999).