命令発行スロット数を考慮したVLIWプロセッサのアーキテクチャ最適化手法

大槻典正† 武内良典†
今井正治† 引地信之††

VLIWプロセッサは複数の演算を並列に実行できるため性能を高くすることが可能であるが，1つの命令中に複数の演算命令を含んでいるため命令長が長くなり，命令メモリのハードウェア・コストが大きくなるという問題がある．この問題は，演算命令を同時発行するスロット数を制限し，命令長を短縮することで解決することができる．しかし，この場合，プログラムの実行サイクル数が増加する可能性があるため，性能とハードウェア・コストを考慮したうえで，プロセッサの演算命令発行スロット数を決定する必要がある．本稿では，演算命令発行スロット数が可変なVLIWプロセッサ・モデルの提案を行い，提案モデルを用いた場合の命令スケジューリング・アルゴリズムを示す．また，ハードウェア・コストを考慮したうえで，与えられたアプリケーション・プログラムに適した演算命令発行スロット数を決定する演算命令発行スロット数最適化アルゴリズムを示す．提案手法を用いて数種のプログラムに対して試行実験を行い，適切な演算器構成と演算命令発行スロット数が選択されることを確認した．

Architecture Optimization Method of A VLIW Processor with Variable Operation Slot Count

Norimasa Ohtsuki,† Yoshinori Takeuchi,† Masaharu Imai† and Nobuyuki Hikichi††

VLIW (Very Long Instruction Word) processors are expected to have high performance processing due to its instruction level parallelism. However, long instruction words of VLIW processor require large amount of hardware cost for instruction memory. Limiting the operation slot count is one of the solutions to this problem, but the performance could become lower. Therefore, it is necessary to consider the tradeoff between the performance and the hardware cost of processors to decide the optimal number of the operation slots. In this paper, we propose a VLIW processor model with variable operation slot count and give an instruction scheduling algorithm for the model. Then, the operation slot count optimization algorithm is proposed, which finds the operation slot count to minimize the execution cycle of a given application program under a given hardware cost constraint. Experimental results for several programs show that the proposed algorithm is able to decide the optimal number of the operation slots as well as the optimal functional unit set.

1. はじめに

VLIWプロセッサ[1)]は複数の演算器に対する命令を1つの命令（VLIW命令）としてまとめて記述することで，命令の並列発行を実現している．本稿では，演算器に発行する命令を**演算命令**，複数の演算命令をまとめて並列に記述した命令を**VLIW命令**と呼ぶ．純粋なVLIWプロセッサでは，VLIW命令に含まれる演算命令の数と演算器の数は等しい．しかし，命令の依存関係によって並列実行できる命令が制限されてしまうため，多くの演算命令がNOP（No Operation）命令となる[2)]．NOP命令は演算を行わない命令なので，VLIW命令に多くのNOP命令が含まれると，任意のプログラムをスカラ型プロセッサで実行する場合に比べて，命令メモリに要するハードウェア・コストが大きくなり，命令メモリの使用効率が低下するという問題がある．この問題は，VLIW命令に含まれる演算命令の数に制限を与え，1つのVLIW命令の命令

† 大阪大学大学院基礎工学研究科情報数理系専攻
Department of Informatics and Mathematical Science, Grauate School of Engineering Science, Osaka University
†† 株式会社SRA
Software Research Associates, Inc.

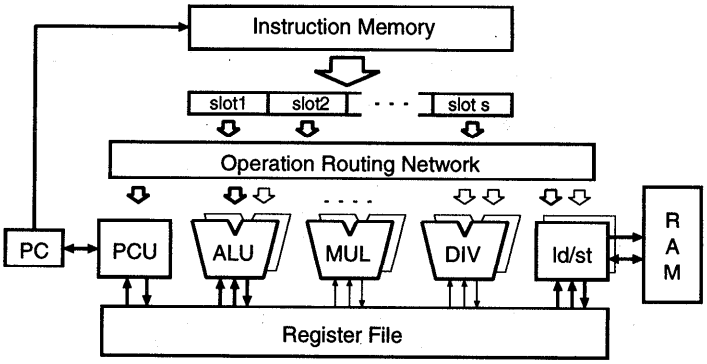

図1 VLIWプロセッサモデル
Fig. 1 VLIW processor model.

長を短縮させることで命令メモリのハードウェア・コストを削減することで緩和できる[3),4)]．フェッチされたVLIW命令は演算命令ごとに分解されて演算命令発行スロットに格納されるため，VLIW命令に含まれる演算命令数を**演算命令発行スロット数**と呼ぶ．

演算命令発行スロット数を減少させることで命令メモリのハードウェア・コストの削減が可能だが，演算命令発行スロット数が少なすぎる場合には，プログラムの実行サイクル数が増加してしまう．これは，潜在的な命令並列性の高いプログラムを実行させたときに演算命令発行スロット数が不足している場合，同時に発行可能である命令のいくつかがスロット数の制約によって発行できず，次のサイクルに発行されてしまうためである．一方，演算命令発行スロット数を増加させた場合は，実行サイクル数の低下は起こらないが，命令メモリのハードウェア・コストが増加する．したがって，VLIWプロセッサを設計する際には，実行するプログラムに応じて演算命令発行スロット数を決定しなければならない．

本稿では，演算命令発行スロット数が可変のVLIWプロセッサモデルを提案し，提案モデルを構成するうえで必要な命令発行ネットワーク機構と，命令スケジューリング・アルゴリズムを示す．また，提案モデルを用いて演算命令発行スロット数最適化問題の定式化を行い，最適化問題を解くことで，与えられた応用プログラムに応じて，プログラムの実行サイクル数が最小となるように，演算命令発行スロット数を決定することができることを示す．

本稿では，2章で演算命令発行スロット数可変なVLIWプロセッサモデルを提案し，3章で提案モデルを用いた演算命令発行スロット数最適化問題の定式化を行う．4章で提案モデルの有効性を確認するために行った実験の結果を示し，5章でまとめる．

2. 演算命令発行スロット数可変VLIWプロセッサモデル

2.1 VLIWプロセッサモデル

提案する演算命令発行スロット数可変VLIWプロセッサモデルは以下のパラメータを有する．

- 演算器実現方法
- 演算器数
- 演算命令発行スロット数

また，CPUコアの基本特性として以下の特徴を持つ．

- ハーバード方式
- ロード/ストア方式
- 汎用レジスタ方式
- パイプライン制御方式
- マルチサイクル演算方式

図1に提案するVLIWプロセッサモデルを示す．

同図で命令メモリからフェッチされたVLIW命令は各演算命令に分割され，複数の演算命令発行スロットへ格納される．各スロットの演算命令は命令発行ネットワークを経由して対応する演算器に発行される．演算命令を発行された演算器は命令の内容に従って演算を開始するが，命令を発行されなかった演算器はNOP命令を発行された場合と同様に新たな演算の開始を行わない．

提案モデルではハードウェア・インターロック機構は持たず，同じ演算器に対する複数命令の同時発行や，命令受理可能状態ではないマルチサイクル演算器に対する命令発行は禁止している．分岐命令に対しては遅延分岐法を採用する．上記の各種パイプライン・ハザードは，コンパイラのスケジューリングによって除去可能である．スケジューリング手法の詳細については2.4節で述べる．

表1 演算器のクラスと個数
Table 1 Class and the number of functional units.

クラス		個数
PCU	(PC制御ユニット)	1
ld/st	(ld/stユニット)	≥ 1
ALU	(算術論理演算器)☆	≥ 1
MUL	(乗算器)	≥ 0
DIV	(除算器)	≥ 0

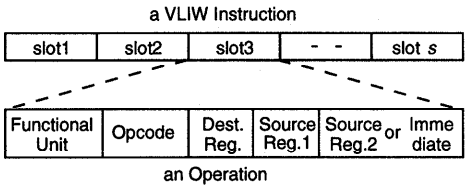

図2 命令フォーマット
Fig. 2 Instruction format.

提案モデルで使用可能な演算器のクラスと個数を表1に示す．各演算器の個数には上限を与えておらず，必要な個数だけ搭載可能である．必要最小限の処理を行うためにPCU（プログラムカウンタ制御ユニット），ld/st（ld/stユニット），ALU（算術論理演算器）は少なくとも1個は必要であるが，MUL（乗算器），DIV（除算器）はプログラム中に乗除算命令が存在しない場合に限り0個の場合も許す．

命令発行ネットワークについては2.3節で詳細を述べる．

2.2 命令フォーマット

VLIW命令と演算命令のフォーマットを図2に示す．1つのVLIW命令には複数の演算命令が格納されている．各演算命令は5つのフィールドから構成されており，第1フィールドの演算器指定フィールドにその演算命令が発行される演算器の番号が指定されている．命令を発行された演算器は，第2フィールド以降のオペコードと3つのオペランドに従って演算を開始する．

2.3 命令発行ネットワーク

命令発行ネットワーク（図3）を構成する際，すべてのスロットと演算器の間を完全結合にすることで，全スロットから全演算器に対して命令発行可能となるが，配線に要するハードウェア・コストが増加する．提案モデルでは，命令発行に制限を与えない範囲で配線数を減らし，ハードウェア・コストの削減を図る．

演算器が f 個，演算命令発行スロットが s 個であると仮定したとき，スロット m $(1 \leq m \leq s)$ からは番号 $m \sim m+(f-s)$ の演算器にしか命令を発行しない制約を与える．n $(0 \leq n \leq s)$ 個の同時発行可能命令が存在する場合におけるスロット割当てアルゴリズムを図4に示す．なお，図中の $FU(x)$ は命令 x が発行される演算器を表す．

☆ 算術論理演算にはシフト演算と比較演算も含む．

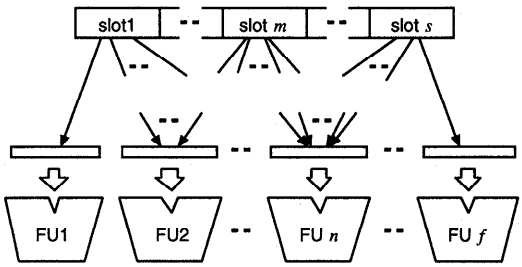

図3 命令発行ネットワーク
Fig. 3 Operation routing network.

```
全スロットを未割当て状態にする;
割当て対象の命令を実行演算器順にソート;
順に x_1, ..., x_n とする;
for i := 1 to n do
begin
  for j := 1 to s do
  begin
    if (FU(x_i) とスロット j に接続がある
        and スロット j は未割当て) then
      x_i をスロット j に割り当てる;
  end;
end;
```

図4 スロット割当てアルゴリズム
Fig. 4 Slot allocation algorithm.

図4のアルゴリズムを用いることで，s 個以下の任意の演算器に対して命令を発行することができる．このように命令発行ネットワークを構成することで，完全結合に比べて命令の並べ方に制約が生じるが，命令実行効率の劣化を生じさせることなくハードウェア・コストの削減を行うことができる．命令の並べ方に関する制約はスロット割当てアルゴリズムをコンパイラに組み込むことで吸収することができる．

このモデルを用いてスロット数 $s=1$ のネットワークを構成すると，1サイクルに1命令発行するスカラ型プロセッサとなる．これは，提案モデルがVLIW型プロセッサのみでなくスカラ型プロセッサをも表現できることを意味する．

2.4 命令スケジューリング

演算命令発行スロット数や演算器の構成を変化させた場合，VLIW命令の形式が変化するため，命令の互換性は保てない．したがって，提案モデルを用いたプロセッサのコード生成や性能評価を行うためにはリターゲット可能なコンパイラが必要であり，特に演算

命令発行スロット数の変化に対応した命令スケジューリングを行わなければならい．本研究では，資源制約スケジューリング手法として一般的に用いられるリスト・スケジューリング手法[5),6)]を，演算命令発行スロット数の変化に対応できるようにアルゴリズムの拡張を行った．本スケジューリング・アルゴリズムは基本ブロック単位でのスケジューリングのみを行っているが，広域的な最適化は現在のところ行っていない．

図5 に演算命令発行スロット数の変化に対応可能な命令スケジューリング・アルゴリズムを示す．本アルゴリズムは，基本ブロック間のデータの受け渡しも考慮したレジスタ割付けがすでに終了していることを前提としており，レジスタ割付け終了後のDFGを入力として受け取る．図5において，ready 集合$\boldsymbol{R}$は，スケジューリング可能な命令を優先度順にソートした順序集合であるとする．ここで，スケジューリング可能な命令とは，基本ブロックのデータフローグラフ（DFG）を構成した際，後続命令がない，もしくはすべての後続命令がスケジュール済みの命令である．マルチサイクル演算方式を採用している場合は，すべての後続命令がスケジュール済みとなった命令であっても，その命令を実行するために必要なサイクル数が経過してからでなければ，ready 集合に登録されない．ready 集合の要素は，先頭から順に $r_1, \cdots, r_i, \cdots$ とする．$L(r_i)$ は発行遅延であり，演算器が命令 r_i を発行された際に次の命令を発行可能となるまでのサイクル数を表す．

本アルゴリズムは一般的なリスト・スケジューリングと同様に，ready 集合内で優先度順にソートされた命令順に演算器の割当てを行う．時刻 t に割り当てられた命令は ready 集合から削除し，割当て不可能な命令は ready 集合に残す．提案手法では，演算命令を同時発行できるスロット数が s 個という制約を考慮する必要がある．時刻 t に割り当てられた演算命令数が s と等しくなったときは，スロットはすべて埋まっているため，これ以上の命令を割り当てることはできない．時刻 t に割当て可能な命令がなくなったとき，時刻 t を $t-1$ と更新し ready 集合を再構成した後，命令割当てを再開する．すべての命令が割り当てられたとき，スケジューリングが終了する．本アルゴリズムは図2，3の命令発行機構を前提としているため，演算器とスロット間の接続については考慮する必要がない．VLIW コード生成時に図4のアルゴリズムを用いることで，適切なスロットに演算命令が割り当てられた VLIW コードを生成できる．

なお，本アルゴリズムは DFG の葉の命令から根に向かってスケジューリングを行う．スケジューリング終了後の先頭命令の割当て時刻 t から，スケジューリング後の基本ブロックの実行サイクル数を知ることができる．また，本アルゴリズムをコンパイラに組み込むことで，与えられた演算命令発行スロット数に応じた VLIW コードを生成することが可能となる．

```
スケジューリング可能命令で ready 集合 R を作成;
時刻 t := 0;
スロット・カウンタ s_used := 0;
while (R ≠ φ) do
begin
  for (優先度ソートをされた順に r_i ∈ R) do
  begin
    if (演算器 ∃f ∈ FU(r_i) が
        時刻 t から t + L(r_i) − 1 まで使用可能)
    then
    begin
      命令 r_i を演算器 f の時刻 t に割当て;
      演算器 f を時刻 t から t + L(r_i) − 1 まで
          使用不可にする;
      命令 r_i を R から削除;
      s_used := s_used + 1;
      if (s_used ≥ s) then
        break;      { スロットが埋まった }
    end;
  end;
  ready 集合 R を再構成;
  t := t − 1;
  s_used := 0;
end;
```

図5 拡張リスト・スケジューリング手法
Fig. 5 Extended list scheduling method.

3. 演算命令発行スロット数最適化問題

任意のアプリケーション・プログラムが与えられた場合，そのプログラムを最も高速に実行できるプロセッサを設計することは一般に難しい[7)~10)]．本稿では演算命令発行スロット数最適化問題の定式化を行い，この問題を解くアルゴリズムを与える．提案するアルゴリズムは，与えられたハードウェア・コストに関する制約条件下で，アプリケーション・プログラムの実行サイクル数が最小となるように，演算命令発行スロット数と演算器構成を決定する．

最適化を行うためにはプロセッサのハードウェア・コストと実行サイクル数を見積る必要がある．以下では，まずハードウェア・コストと実行サイクル数の見積り方法を示す．

3.1 ハードウェア・コスト

プロセッサは，CPU コア，命令メモリ，データ・メモリから構成される．データ・メモリの量は設計者が与えると仮定するため，データ・メモリのハードウェ

ア・コストは最適化問題の対象外とする．プロセッサ全体のハードウェア・コストを，CPUコアと命令メモリのハードウェア・コストを合計して近似する．

CPUコアのハードウェア・コストは，演算器構成，演算命令発行スロット数，レジスタ数に依存する．CPUコアのハードウェア・コスト A は次の式で与えられる．

$$A(\boldsymbol{X},\boldsymbol{K},r,s)=\sum_{i=1}^{n}\{D(x_i)\times k_i\}+R(\boldsymbol{K},r)+S(\boldsymbol{K},s) \quad (1)$$

ここで，

$A(\boldsymbol{X},\boldsymbol{K},r,s)$ ： 演算器構成が $\boldsymbol{X}$，$\boldsymbol{K}$ でレジスタ数 r，演算命令発行スロット数 s の場合のCPUコアのハードウェア・コスト

$\boldsymbol{X}=\{x_1,\cdots,x_n\}$： x_i はクラス i の演算器の種類

$\boldsymbol{K}=\{k_1,\cdots,k_n\}$： k_i はクラス i の演算器の個数

n ： 演算器のクラスの数

$D(x_i)$ ： 演算器 x_i のハードウェア・コスト

$R(\boldsymbol{K},r)$ ： 演算器数が $\sum_{i=1}^{n}k_i$ でレジスタが r 個の場合のレジスタファイルのハードウェア・コスト

$S(\boldsymbol{K},s)$ ： 演算器数が $\sum_{i=1}^{n}k_i$ で演算命令発行スロット数が s の場合の命令発行ネットワークのハードウェア・コスト

である．

各演算器のハードウェア・コスト D は論理合成ツールを用いてあらかじめ測定しておいた値を使用する．レジスタファイルのハードウェア・コスト R は，全演算器のレジスタのリード/ライト・ポート数の合計とレジスタ数をパラメータとして合成を行った値を使用する．命令発行ネットワークのハードウェア・コスト S は，全演算器の個数と命令発行スロット数をパラメータとして合成を行った値を使用する．なお，命令発行ネットワークは演算命令発行スロットのハードウェア・コストを含むものとする．

命令メモリのハードウェア・コスト M は，VLIW命令の命令長とプログラムの命令数に依存する．命令メモリのハードウェア・コストは次の式で与えられる．

$$M(s,p)=C_M\times(s\times W\times p) \quad (2)$$

ここで，

$M(s,p)$： 演算命令発行スロット数 s，プログラムの命令数 p の場合の命令メモリのハードウェア・コスト

C_M ： 命令メモリ1ビットに必要なハードウェア・コスト

W ： 演算命令1スロットあたりのビット長

である．命令数 p は演算器構成 $\boldsymbol{X}$，$\boldsymbol{K}$ と，演算命令発行スロット数 s に依存し，命令スケジューリングを行うことによって知ることができる．演算命令1スロットあたりのビット長 W は固定であるが，演算器数 $\sum_{i=1}^{n}k_i$ と，レジスタ数 r に依存し，式(3)で与えられる．

$$W=W_{Op}+\lceil\log_2 f\rceil+2\lceil\log_2 r\rceil+max(\lceil\log_2 r\rceil,W_{Imm}) \quad (3)$$

ここで，

W_{Op} ： オペコードのビット幅

W_{Imm} ： 即値のビット幅

f ： 演算器数 $\sum_{i=1}^{n}k_i$

である．なお，$\lceil x\rceil$ は，x よりも小さくない最小の整数を表す．

3.2 プロセッサの実行サイクル数

アプリケーション・プログラムの実行サイクル数 T は以下の式で与えられる．

$$T(\boldsymbol{X},\boldsymbol{K},r,s)=\sum_{j=1}^{N}\{C_j\times t(B_j,\boldsymbol{X},\boldsymbol{K},r,s)\} \quad (4)$$

ここで，

$T(\boldsymbol{X},\boldsymbol{K},r,s)$： 演算器構成が $\boldsymbol{X}$，$\boldsymbol{K}$ でレジスタ数 r，演算命令発行スロット数 s の場合の応用プログラムの実行サイクル数

N ： 応用プログラムの基本ブロックの個数

C_j ： 基本ブロック B_j の実行回数

$t(B_j,\boldsymbol{X},\boldsymbol{K},r,s)$： 演算器構成 $\boldsymbol{X}$，$\boldsymbol{K}$，レジスタ数 r，演算命令発行スロット数 s における基本ブロック B_j の実行サイクル数

である．

基本ブロックの個数 N と各基本ブロックの実行回数 C_j は，アプリケーション・プログラムとデータのみに依存し，演算器構成や演算命令発行スロット数には依存しない．N と C_j は，コンパイル結果の実行プロファイルを行うことで求めることができる．

$t(B_j,\boldsymbol{X},\boldsymbol{K},r,s)$ は命令スケジューリングを行った後の実行サイクル数を表しており，プロセッサの演算器構成，演算命令発行スロット数に依存する．図5のアルゴリズムはヒューリスティックな手法であるため，スケジューリング後の実行サイクル数を数式で近似することはできない．そこで，演算器構成やスロット数を変更するごとにスケジューリングを行うことで，正確な実行サイクル数を求めている．

3.3 演算命令発行スロット数最適化問題の定式化

本最適化問題は，制約条件 (6) のもとで，目的関数 (5) の値が最小になる解 $\boldsymbol{X}$，$\boldsymbol{K}$，s を求める最適化問題として定式化できる．以下に，本最適化問題の変数，目的関数，制約条件を示す．

- 変数

  演算器構成 $\boldsymbol{X}$，$\boldsymbol{K}$

  演算命令発行スロット数 s

- 目的関数

$$T(\boldsymbol{X}, \boldsymbol{K}, r, s) \rightarrow \text{最小} \tag{5}$$

- 制約条件

$$A(\boldsymbol{X}, \boldsymbol{K}, r, s) + M(s, p) \leq A_{max} \tag{6}$$

ここで，A_{max} は設計制約として与える許容ハードウェア・コストである．

この最適化問題は，演算器構成と演算命令発行スロット数のすべての組合せについてハードウェア・コストと実行サイクル数を測定することで最適解を求めることができる．しかし，すべての組合せについて測定を行うことは計算時間の点で非現実的である．本最適化問題は組合せ最適化問題の一種であり，分枝限定法などに代表される，組合せ最適化問題の一般的な解法を適用することが可能である．

本稿では，分枝限定法を採用することで最適解導出に要する時間の短縮を図る．枝刈りを行うために各節点でハードウェア・コストおよび実行サイクル数の見積りを行うが，各演算器選択節点においてそれぞれ最悪の演算器を仮定した見積りによる枝刈りを行うことで最適な演算器構成を得ることができる[11]．

4. 評価実験

提案手法の有効性を確認するために数種の実験を行った．実験は，我々が開発中の VLIW 用ハードウェア/ソフトウェア協調設計システム PEAS-II[12]に，提案手法を実装することで実現した．また，コンパイラは GCC (GNU C Compiler)[13]に VLIW 用最適化パスを追加することで実現している．VLIW 用最適化パスは，図 5 のアルゴリズムを用いたスケジューラと VLIW コード生成部から成り，字句解析，マシン非依存最適化，レジスタ割付けは GCC のコードを利用している．

実験に用いたサンプル・プログラムは以下の 3 種である．

- 3 次 PARCOR フィルタ
- 1024 点高速フーリエ変換 (FFT)
- ADPCM

また，実験で用いた演算器を**表 2** に示す．表 2 において，演算遅延は演算器に命令を発行してから結果が得られるまでに要するサイクル数，発行遅延は演算器に命令を発行してから次の命令が発行可能になるまでに要するサイクル数である．演算遅延と発行遅延が等しい場合はノン・パイプライン演算器であり，発行遅延が演算遅延よりも短い場合はパイプライン演算器である．ハードウェア・コストは，Synopsys 社の論理合成システム Design Compiler と VLSI テクノロジー社の VSC753d (0.5 μm CMOS) ライブラリを用い，目標動作周波数 50 MHz のもとで合成を行った結果である．また，本実験では，オペコードのビット幅を 6 bit，演算器の最大個数を 16 個，即値のビット幅を 32 bit に固定しているため，式 (3) より，演算命令 1 スロットあたりのビット数 W は 48 となる．なお，本稿では，レジスタ数を 8 個とした場合の実験結果を示す．

表 2 実験で用いる演算器
Table 2 Functional units used in experiments.

タイプ	名前	演算遅延 (cycles)	発行遅延 (cycles)	HW コスト (gates)
PCU	pcu	1	1	1,624
ld/st	ld/st	3	1	1,159
ALU	alu	1	1	2,892
MUL	mul_2_2	2	2	14,427
	mul_32_8	32	8	7,613
	mul_32_16	32	16	3,944
	mul_32_32	32	32	2,183
DIV	div_21_21	21	21	17,856
	div_34_17	34	17	6,829
	div_34_34	34	34	3,841

4.1 命令発行ネットワークの評価

提案した命令発行ネットワークの構成方法が有効であることを確認するために，前述の合成ツールを用いて見積りを行った．**表 3** に，演算器数を 10 個とした場合の完全結合方式と提案方式におけるハードウェア・コストと遅延時間の見積り結果を示す．

実験結果より，平均約 67%のハードウェア・コストが削減されていることが確認できる．最もよく利用されると思われる[2]発行スロット数が 3〜4 の場合でも，約 20〜30%のハードウェア・コストが削減されている．また，配線数が減少したために，遅延時間についても短縮できていることが確認できる．

4.2 演算命令発行スロット数最適化

各サンプル・プログラムに対して，演算命令発行スロット数の最適化を行った．なお，本実験では，On-Chip DRAM を想定し，命令メモリ 1 ビットあたりのハードウェア・コスト C_M を

$$C_M = 1 \quad \text{(gates/bit)}$$

表3 命令発行ネットワーク
Table 3 Operation routing network.

スロット数	完全結合 HWコスト (gates)	完全結合 遅延時間 (ns)	提案手法 HWコスト (gates)	提案手法 遅延時間 (ns)
1	667	1.83	667	1.78
2	1,704	2.93	1,514	2.72
3	2,856	4.01	2,251	3.72
4	3,812	4.29	2,630	3.90
5	5,346	4.52	2,856	3.83
6	6,574	4.36	2,860	3.80
7	7,663	4.35	2,634	3.91
8	8,771	4.76	2,259	3.51
9	10,071	5.07	1,528	2.63
10	12,355	6.20	789	1.40

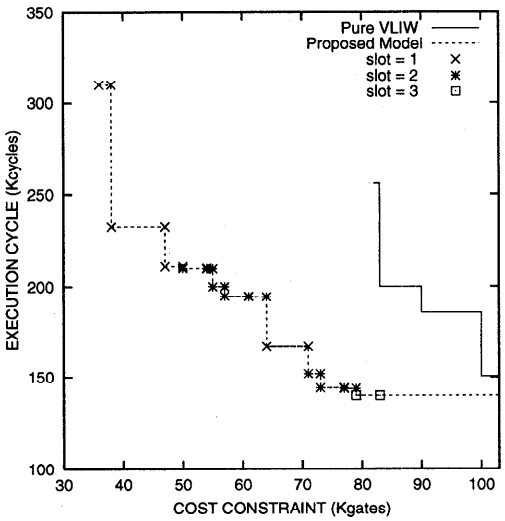

図6 コストと実行サイクル数の関係（PARCOR フィルタ）
Fig. 6 Tradeoffs between the hardware cost constraint and the execution cycles (PARCOR filter).

と仮定した．

図6〜図8に，選択されたプロセッサのハードウェア・コストと実行サイクル数の関係を示す．各図において，横軸は設計制約として与えるハードウェア・コスト，縦軸は選択されたプロセッサ上での与えられたプログラムの実行サイクル数をそれぞれ表す．実線で示された“Pure VLIW”は，演算器数と演算命令発行スロット数が1対1に対応したVLIWプロセッサである．“Proposed Model”はスロット数制約型VLIWプロセッサであり，破線で表されている．また，選択されたスロット数によって記号を変えて示してある．

実験結果より，“PureVLIW”の選択が不可能なハードウェア・コストにおいてもスロット制約型プロセッサが選択できていることがわかる．また，“PureVLIW”

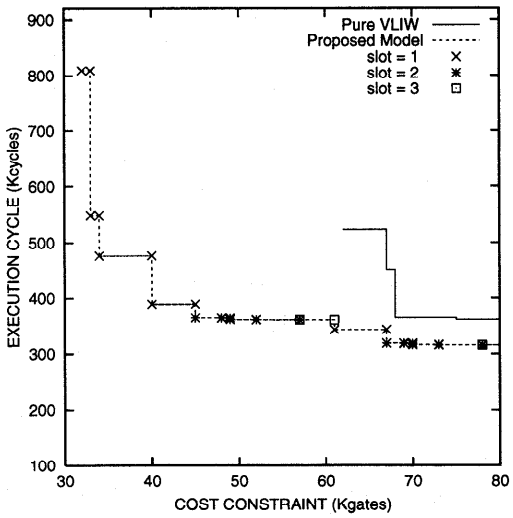

図7 コストと実行サイクル数の関係（FFT）
Fig. 7 Tradeoffs between the hardware cost constraint and the execution cycles (FFT).

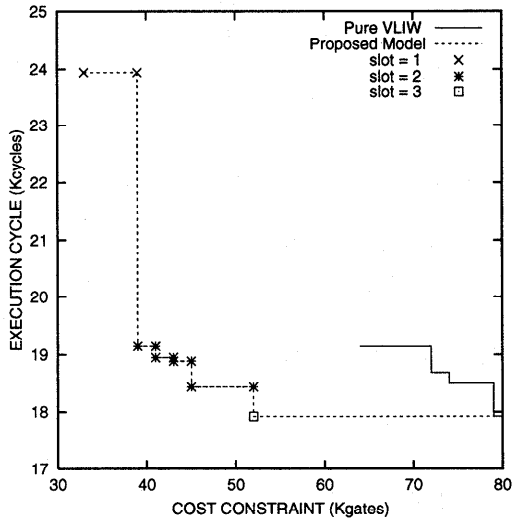

図8 コストと実行サイクル数の関係（ADPCM）
Fig. 8 Tradeoffs between the hardware cost constraint and the execution cycles (ADPCM).

が選択できる場合においては，より少ない実行サイクル数でプログラムを実行するスロット制約型プロセッサが選択されている．性能の向上率はアプリケーション・プログラムに依存するが，実験結果より提案手法が有効であることを確認することができる．

また，図6〜図8より，ハードウェア・コストの制約が厳しい場合には演算命令発行スロット数が減少していることが分かる．これは，スロット数が多い場合

表4 ハードウェアコスト制約と選択されたプロセッサの演算器構成（FFT）
Table 4 Hardware cost constraint and selected processor parameters (FFT).

HWコスト制約（gates）	スロット数	PCU	ld/st	alu	mul		div		VLIW命令数	実行サイクル数	命令メモリコスト（gates）
					32_16	32_32	21_21	34_17			
60,000	3	1	2	2	1	0	0	2	113	361,440	16,272
65,000	1	1	1	1	1	0	2	0	127	343,489	6,096
70,000	2	1	2	2	0	1	2	0	105	315,857	10,080
80,000	3	1	2	2	1	0	2	0	104	315,360	14,976

には必要な命令メモリの量が増加するが，そのためのハードウェアを確保できず，スロット数を減らして命令メモリのハードウェア・コストを節約しているためである．ハードウェア・コストの制約を緩くした場合には，一般に演算命令発行スロット数は増える傾向にある．これは，スロット数とともに増加する命令メモリのためのハードウェア・コストを確保できるためである．しかし，図7の65 Kgate付近のように，ハードウェア・コストの制約は緩いけれども，スロット数は1が選択されるような例外的な場合も存在する．

表4に，FFTをアプリケーション・プログラムとし，ハードウェア・コスト制約を60Kゲートから80Kゲートまで変化させた場合の，選択されたプロセッサの演算器構成と演算命令発行スロット数，および実行サイクル数とコンパイル後のVLIW命令数，命令メモリのコストを示す．提案手法を用いることにより，制約を60Kゲートから65Kゲートに緩めた場合，スロット数と演算器の個数を減らしたうえで，高速な除算器を2個搭載した最適構成が選択できている．さらに制約を緩めた場合は，再びスロット数が増加している．また，制約を緩めるごとに実行サイクル数とVLIW命令数が減少しているが，命令メモリのコストはVLIW命令数とスロット数によって増減していることが確認できる．このような最適化を人手で行うことは困難であるが，提案手法を用いることで，最適な演算器構成とスロット数を容易に求めることができる．また，実験結果より，制約条件によっては演算命令発行スロット数が1であるスカラ型プロセッサも選択されていることが確認できる．

4.3 最適解探索時間の評価

本稿では最適化手法として分枝限定法を採用した．アプリケーション・プログラムとしてはPARCORフィルタを用い，複数のハードウェアコスト制約のもとで最適化を行った．**表5**に最適解を得るまでに探索した節点数，および探索時間を示す．分枝限定法を用いずに探索を行った場合，25,200個の節点を探索する必要があったが，分枝限定法を用いることで探索節点数が大幅に減少し，それにともない探索時間も短縮されていることが知られる．

表5 探索節点数および探索時間（PARCORフィルタ）
Table 5 Number of visited nodes and search time (PARCOR filter).

HWコスト制約（gates）	探索節点数	探索時間（s）
40,000	34	0.050
45,000	39	0.047
50,000	58	0.073
55,000	68	0.078
60,000	66	0.088
65,000	66	0.086
70,000	70	0.091
75,000	72	0.085
80,000	62	0.062
85,000	62	0.070
90,000	61	0.082
search all combination	25,200	52.761

5. おわりに

1つの命令長が長いVLIWプロセッサを構成する際には，命令メモリのハードウェア・コストを考慮したうえで，演算命令発行スロット数を決定しなければならない．演算命令発行スロット数を減少させるとハードウェア・コストを削減することができるが，プログラムの実行時間が増加する可能性があり，トレードオフが生じる．本稿では，演算命令発行スロット数が可変のVLIWプロセッサのモデル化を行い，演算命令発行スロット数最適化問題の定式化を行った．

提案手法を，開発中のハードウェア/ソフトウェア協調設計システムPEAS-IIに組み込んで実験を行った結果，演算命令発行スロット数を制限することで，純粋なVLIW型プロセッサよりもプログラムを高速に実行するプロセッサが選択できることが確認できた．

今後の課題としては，動作周波数[14]やレジスタ数[12]等を最適化パラメータとして拡張し，最適化アルゴリズムを開発することが考えられる．

謝辞 本研究を進めるにあたり，貴重なコメントをいただいた（株）半導体理工学研究センターの小澤時典博士，（株）日立製作所の堀田正生博士，松下電器産

業（株）の村岡道明氏，および大阪大学VLSIシステム設計研究室，情報解析学研究室の諸兄に深謝する．なお，本研究の一部は（株）半導体理工学研究センターとの共同研究による．

参考文献

1) Fisher, J.A.: Very Long Instruction Word Architectures and the ELI-512, *Proc. 10th Annual Symposium on Computer Architectures*, pp.140–150, ACM (1983).
2) Wall, D.W.: Limits of Instruction-Level Parallelism, *Proc. 4th Intl. Conf. on Arch. Support for Programming Languages and Operating Systems*, pp.176–189 (1991).
3) Slavenburg, G., Rathnam, S. and Dijkstra, H.: The Trimedia TM-1 PCI VLIW Media Processor, *Proc. Hot Chips VIII*, Stanford, CA, pp.171–177 (1996).
4) Gray, J., Naylor, A., Abnous, A. and Bagherzadeh, N.: VIPER: A VLIW Integer Microprocessor, *IEEE Journal of Solid-State Circuits*, Vol.28, No.12, pp.1377–1382 (1993).
5) Johnson, M.: *Superscalar Microprocessor Design*, Prentice-Hall (1991).
6) Gajski, D.D., Dutt, N.D., Wu, A. C-H, and Lin, S. Y-L: *HIGH-LEVEL SYNTHESIS*, Kluwer Academic Publishers (1992).
7) Sato, J., Alomary, Y., Honma, Y., Nakata, T., Shiomi, A. and Imai, M.: PEAS-I: A Hardware/Software Codesign System for ASIP Developent, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Science*, Vol.E77-A, No.3, pp.483–491 (1994).
8) Akaboshi, H. and Yasuura, H.: COACH: A Computer Aided Design Tool for Computer Architects, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Science*, Vol.E76-A, No.10, pp.1760–1769 (1993).
9) Gupta, R. and De Micheli, G: Hardware-Software Cosynthesis for Digital Systems, *IEEE Design & Test*, pp.29–41 (1993).
10) Ernst, R., Henkel, J. and Benner, T.: Hardware-Software Cosynthesis for Microcontrollers, *IEEE Design & Test*, pp.64–75 (1993).
11) 大槻典正，武内良典，今井正治，浜口清治，柏原敏伸，引地信之：VLIWプロセッサ自動生成における演算器構成最適化の一手法，信学技報，VLD97-92, pp.101–108 (1997).
12) 大槻典正，武内良典，今井正治，浜口清治，柏原敏伸，引地信之：VLIWプロセッサのためのHW/SW協調設計システムPEAS-II，DAシンポジウム'98論文集，pp.53–58 (1998).
13) Stallman, R.: *Using and Porting GNU C Compiler*, Free Software Foundation (1995).
14) Shinohara, K., Ohtsuki, N., Takeuchi, Y. and Imai, M.: A Performance Optimization Method for Application Specific Instruction Set Processors with Consideration of Clock Frequency, *SASIMI'98*, pp.163–170 (1998).

(平成10年9月21日受付)
(平成11年2月 8 日採録)

大槻　典正（学生会員）

1997年大阪大学基礎工学部情報工学科卒業．現在，同大学大学院基礎工学研究科情報数理系専攻博士前期課程在学中．VLIWプロセッサ，命令レベル並列コンパイラに興味を持つ．電子情報通信学会会員．

武内　良典（正会員）

1987年東京工業大学工学部電気・電子工学科卒業．1992年同大学大学院博士課程修了．博士（工学）．同年東京農工大学工学部助手．現在大阪大学大学院基礎工学研究科情報数理系専攻講師．ディジタル信号処理，VLSI設計およびVLSI CADの研究に従事．電子情報通信学会，IEEE各会員．

今井　正治（正会員）

1974年名古屋大学工学部電気工学科卒業．1979年同大学大学院博士後期課程修了（工学博士）．同年豊橋技術科学大学奉職．1994年同教授．1996年大阪大学大学院基礎工学研究科情報数理系専攻教授．その間，1984年から1985年にかけて米国サウスカロライナ大学工学部電気計算機工学科客員助教授（文部省在外研究員）．これまで組合せ最適化アルゴリズム，ハードウェア/ソフトウェア協調設計等の研究に従事．1991年より日本電子機械工業会およびIEEE/DASCにおいてEDA標準化作業に従事．現在，情報処理学会設計自動化研究会主査．IEEE，ACM，電子情報通信学会，人工知能学会各会員．

引地　信之（正会員）

1983年豊橋技術科学大学情報工学課程修了．1983年（株）SRAに入社．ソフトウェア開発環境にかかわる研究開発に従事．GNUプロジェクト参加．コンパイラ，OS，CPUアーキテクチャに興味を持つ．電子情報通信学会会員．