計算機アーキテクチャ 126－10
（1997. 10. 29）

ジオメトリプロセサ Procyon の評価

新井正樹† 西崎慎一郎†† 安里彰†
小沢年弘† 木村康則†

ジオメトリプロセサ Procyon はハードウェアを簡素化するために命令のソースオペランドの値をレジスタあるいは複数のバイパス出力のいずれから読むかをソフトウェアで指定するという特徴をもった 4 並列の VLIW プロセサである．Procyon では，各 VLIW 命令のソースオペランドの値をバイパスあるいはレジスタから読む方法を，パイプラインのタイミングを考慮してアセンブリコードで陽に指定する必要がある．本論文では，コンパイラによる Procyon のソフトウェア・バイパス制御方式とソフトウェア・バイパス制御方式の有効性の評価について述べる．

Evaluation of Geometry Processor 'Procyon'

Masaki Arai,† Shin'ichiro Nishizaki,†† Akira Asato,†
Toshihiro Ozawa† and Yasunori Kimura†

The geometry processor 'Procyon' is an implementation of a VLIW architecture which has max 4 operations in a single instruction. This processor doesn't have a hardware bypass control circuit to simplify a hardware design. Each instruction has a field to specify a register or a bypass where the input data comes as its input operands. The compiler or the hand coder of the Procyon must manage when and where pipeline results are available and explicitly write which bypass result or register is used in assembly language.

In this paper, we show how to control the bypass by the compiler and evaluate the effect of it.

1. はじめに

ジオメトリプロセサ Procyon[1] はハードウェアを簡素化するために命令のソースオペランドの値をレジスタあるいは複数のバイパス出力のいずれから読むかをソフトウェアで指定するという特徴（ソフトウェア・バイパス制御方式）をもった 4 並列の VLIW プロセサである．つまり，各 VLIW 命令のソースオペランドの値をバイパスか，あるいはレジスタから読むかを，パイプラインのタイミングを考慮してアセンブリコードに陽に指定する必要がある．

ソフトウェア・バイパス制御方式のプロサセでは，変数の生存区間の概念が一般のプロセサと異なる．変数の生存区間で，値がバイパス上に存在する範囲を特別扱いすることでレジスタ割り当て時にソフトウェア・バイパス制御方式に特有の最適化が可能になる．

VLIW プロセサの性能向上のためには，コンパイラによる命令スケジューリングが不可欠である．Procyon ではソフトウェア・バイパス制御方式のために，コンパイラによる命令スケジューリング時にどのようにバイパスについて考慮したらよいか，という問題が生じる．

```
(1): nop; add  %r2, %r0@R, %r1@R; nop; nop;
(2): nop; addi %r3, %r2@E, _A   ; nop; nop;
(3): nop; add  %r4, %r2@N, %r3@E; nop; nop;
```

図 1 Procyon のアセンブリコード例
Fig. 1 Example of assembly codes on Procyon

本論文では，コンパイラによる Procyon のソフトウェア・バイパス制御方式とソフトウェア・バイパス制御方式の有効性の評価について述べる．

2. ソフトウェア・バイパス制御方式

Procyon のアセンブリコードの例を図 1 に，その実行時のパイプラインの流れを図 2 に示す．Procyon は 4 並列の VLIW であり，ひとつの VLIW 命令は 4 つの VLIW エレメントからなる．図 1 では最初のオペランドが出力レジスタを表す．整数系命令は D, E, N, W の 4 段のパイプラインをもち，E, N, W の各パイプラインステージで結果の値をバイパスに出力する．

† 新情報富士通研
RWCP Multi-Processor Computing Fujitsu Laboratory
†† 富士通 SSL
FUJITSU SOCIAL SCIENCE LABORATORY LTD.

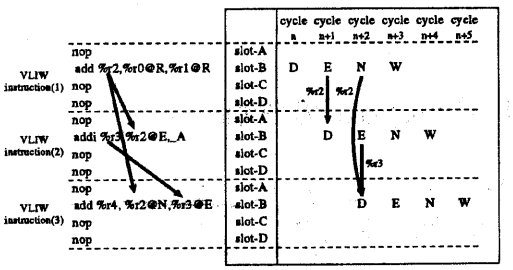

図 2 実行時のパイプラインの流れ
Fig. 2 Pipeline chart of execution

図 1 と図 2 で，VLIW 命令 (2) の%r2@E はレジスタ %r2 の値として VLIW 命令 (1) がバイパス E に出力した演算結果の値をデコードステージ D で使用することを表す．同様に，VLIW 命令 (3) の %r2@N はレジスタ %r2 の値として VLIW 命令 (1) がバイパス N に出力した演算結果の値を使用することを表す．VLIW 命令 (1) の %r0@R はレジスタ %r0 の値をバイパスではなく，レジスタから読むことを表す．

3. ソフトウェア・バイパス制御方式を利用した最適化

一般的なコンパイラは，仮想レジスタが無限個存在するという想定で中間言語を生成し，その中間言語に対して命令スケジューリングやレジスタ割り当て等の最適化を行う．

レジスタ割り当てパスは，仮想レジスタにターゲットマシンに固有のハードウェアレジスタを割り当てる．コンパイラで広く用いられているレジスタ彩色法[3]に基づくレジスタ割り当てのアルゴリズムは次のようになる．

(1) 各仮想レジスタの生存区間を求める．
(2) 各仮想レジスタの生存区間の重なりを調べ，レジスタ干渉グラフを作成する．
(3) レジスタ干渉グラフの各頂点にハードウェアレジスタを割り当てる．

例えば，図 3 の生存区間をもつ仮想レジスタ PR0, PR1, PR2, PR3 が存在する場合，そのレジスタ干渉グラフは図 4 のようになる．図 4 より，仮想レジスタ PR0, PR1, PR2, PR3 すべてにハードウェアレジスタを割り当てるためには，ハードウェアレジスタが最低 3 個必要となる．

ソフトウェア・バイパス制御方式をもつプロセサでは，仮想レジスタの生存区間の概念が一般のプロセサと異なる．ソフトウェア・バイパス制御方式を利用した最適化について，以下に述べる．

3.1 生存区間が短い仮想レジスタ

ソフトウェア・バイパス制御方式をもつプロセサでは，仮想レジスタの生存区間が演算パイプラインの

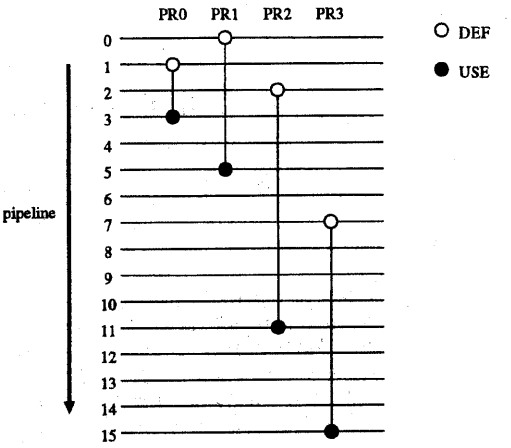

図 3 生存区間解析の結果
Fig. 3 Result of live range analysis

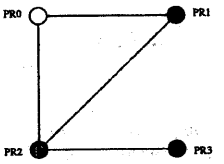

図 4 レジスタ干渉グラフ
Fig. 4 Interference graph

長さ以下である場合には，その仮想レジスタに対してハードウェアレジスタを割り当てる必要がない．これは，生存区間が演算パイプラインの長さ以下である場合には，演算結果はバイパスのみを使って使用され，レジスタファイルに書いた演算結果が後続の命令に使用されることがないためである．例えば，図 3 では仮想レジスタ PR0 の生存区間は 3 サイクルであり，これは演算パイプラインの長さ以下である．したがって，仮想レジスタ PR0 にはハードウェアレジスタを割り当てる必要がない．この結果，レジスタ干渉グラフは図 5 のようになり，最低 2 個のハードウェアレジスタで図 3 の全ての仮想レジスタにレジスタ割り当てを行うことができる．すなわち，ソフトウェア・バイパス制御方式をもつプロセサでは，コンパイラのレジスタ割り当てパスで，レジスタ干渉グラフの各頂点にハードウェアレジスタを割り当てる前に，生存区間が演算パイプラインの長さ以下である仮想レジスタに相当する頂点をレジスタ干渉グラフから削除することができる．これによって，ハードウェアレジスタの不足により生成されるスピルコードの量を減らすことができる．

3.2 レジスタ干渉の除去

一般のプロセサでは，ふたつの仮想レジスタの生存区間が重なる場合には，それらの仮想レジスタに同じハードウェアレジスタを割り当てることはできない．

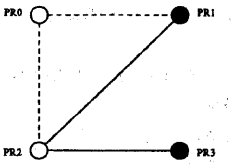

図 5 レジスタ干渉グラフ
Fig. 5 Interference graph

しかし，ソフトウェア・バイパス制御方式をもつプロセサでは，ふたつの仮想レジスタの生存区間が重なる範囲が演算パイプラインの長さ以下である場合には，それらの仮想レジスタに対して同じハードウェアレジスタを割り当てることが可能である．いま，ふたつの仮想レジスタ A と B が存在し，サイクル m で B の生存区間が開始し，サイクル n で A の生存区間が終了するとする．ここで $m < n$ とする．A と B の生存区間の重なる範囲が演算パイプラインの長さ L 以下である場合，すなわち $n - m + 1 \leq L$ である場合には，サイクル m での B の定義はサイクル n ではまだレジスタファイルに書き込まれていない．したがって，サイクル n で A の値をレジスタファイルから読むことによってレジスタ A と B に同一のハードウェアレジスタを割り当ててもプログラムの意味は保存される．例えば，図 3 では仮想レジスタ PR1 と PR2 の生存区間は 4 サイクルの重なりがあり，これは演算パイプラインの長さ以下である．サイクル 2 で PR2 に定義した値は PR1 の値を最後に使用するサイクル 5 でも，まだレジスタファイルに書き込まれていないので，サイクル 5 で PR1 の値をレジスタファイルから読むことによって，PR1 と PR2 に同じハードウェアレジスタを割り当てることができる．つまり，レジスタ干渉グラフ上での PR1 と PR2 のエッジは削除することができる．この結果，3.1 の最適化と合わせると最適化後のレジスタ干渉グラフは図 **6** のようになる．すなわち，ソフトウェア・バイパス制御方式をもつプロセサでは，コンパイラのレジスタ割り当てパスで，レジスタ干渉グラフの各頂点にハードウェアレジスタを割り当てる前に，ふたつの仮想レジスタの生存区間の重なる範囲が演算パイプラインの長さ以下である場合には，それらの仮想レジスタ間のエッジをレジスタ干渉グラフから削除することができる．これによって，ハードウェアレジスタの不足により生成されるスピルコードの量を減らすことができる．

3.3 move 命令による生存区間の分割

一般に，レジスタ間 move 命令を生成することによって，仮想レジスタの生存区間を分割することができる．VLIW 命令の空きスロットに仮想レジスタの生存区間を分割するためのレジスタ間 move 命令を生成することによって，3.1 と 3.2 で述べた最適化の機会を増やすことができる．例えば，図 3 の仮想レジスタ PR3 の生存区間を VLIW 命令の空きスロットに

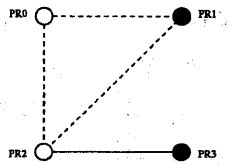

図 6 レジスタ干渉グラフ
Fig. 6 Interference graph

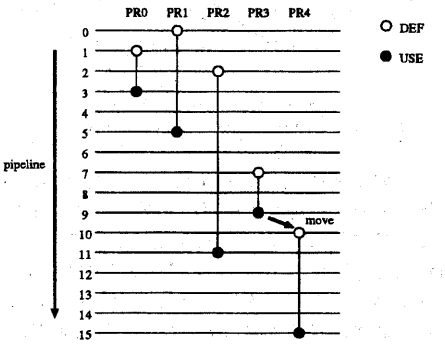

図 7 move 命令による生存区間の分割
Fig. 7 Splitting a live range using move instruction

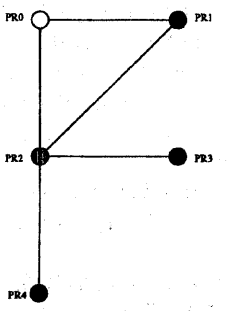

図 8 レジスタ干渉グラフ
Fig. 8 Interference graph

レジスタ間 move 命令を生成することで，サイクル 9 と 10 の間で分割することを考える．新しい仮想レジスタ PR4 を導入して PR3 の生存区間を分割した結果を図 **7** に，そのレジスタ干渉グラフを図 **8** に示す．図 7 では，PR3 の生存区間は演算パイプラインの長さ以下となり，PR2 と PR4 の生存区間の重なりも演算パイプラインの長さ以下となっている．図 8 より，一般のプロセサでは図 7 の仮想レジスタ PR0，PR1，PR2，PR3，PR4 すべてにレジスタを割り当てるためには，ハードウェアレジスタが最低 3 個必要となる．このレジスタ干渉グラフに対して，ソフトウェア・バイパス制御方式を利用した最適化を行うと，仮想レジスタ PR0 と PR3 の生存区間は演算パイプラインの長さ以下であるために 3.1 によりハードウェアレジスタを割り当てる必要がなく，また PR1 と PR2 の生

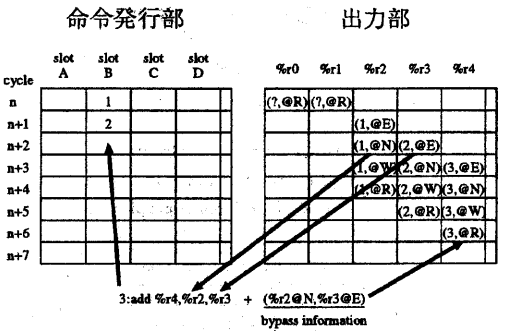

図 9 Procyon の資源予約表
Fig. 9 Resource reservation table for Procyon

存区間の干渉と，PR2 と PR4 の生存区間の干渉は生存区間の重なる範囲が演算パイプラインの長さ以下であるために 3.2 により無視することができる．結局，最低 1 個のハードウェアレジスタで図 7 の全ての仮想レジスタにレジスタ割り当てを行うことができる．

以上述べたソフトウェア・バイパス制御方式を利用したレジスタ割り当てのアルゴリズムは次のようになる．

(1) 各仮想レジスタの生存区間を求める．
(2) 各仮想レジスタの生存区間の重なりを調べ，レジスタ干渉グラフを作成する．
(3) VLIW 命令の空きスロットにレジスタ間 move 命令を生成し，仮想レジスタの生存区間を分割することによって，3.1 と 3.2 の最適化の機会を増やす．
(4) 生存区間が演算パイプラインの長さ以下である仮想レジスタをレジスタ干渉グラフから削除する．
(5) 生存区間の重なる範囲が演算パイプラインの長さ以下である仮想レジスタ間のエッジをレジスタ干渉グラフから削除する．
(6) レジスタ干渉グラフの各頂点にハードウェアレジスタを割り当てる．

4. 命令スケジューリング時の問題点

Procyon のソフトウェア・バイパス制御方式のために生じる問題点と我々のコンパイラ[2]での対処方法を以下に述べる．

4.1 命令スケジューリング時のバイパスの扱い

Procyon のソフトウェア・バイパス制御方式のために，コンパイラによる命令スケジューリング時にどのようにバイパスについて考慮したらよいか，という問題が生じる．コンパイラでは一般に VLIW プロセサの命令スケジューリングを次のように行う．

(1) 対象プログラムを VLIW エレメントの列に変換する．
(2) VLIW エレメントの命令スケジューリングを行う．
(3) 命令スケジューリングの結果から VLIW 命令を作成する．

例として，次の VLIW エレメントの列の命令スケジューリングを考える．

```
1: add  %r2, %r0, %r1
2: addi %r3, %r2, _A
3: add  %r4, %r2, %r3
```

命令スケジューリングには図 9 に示す資源予約表を使用する．資源予約表は，命令発行部と出力部からなっており，命令発行部は VLIW 命令生成に使用し，書き込み部は先行命令の定義する結果を使用できるサイクルを知るために利用する．一般の VLIW プロセサでは，資源予約表の書き込み部分には書き込みが終了するサイクルにエレメント番号を登録すれば良い．Procyon の場合には，資源予約表の書き込み部分には，エレメント番号だけでなくバイパス情報も登録する．また，エレメントを資源予約表に登録するときに，どのレジスタ・バイパスからオペランドの値を読むかを命令に付加する．例えば，図 9 でエレメント 3 は，スロット B がサイクル n + 2 で空いていて，かつソースオペランド %r2 と %r3 をそれぞれバイパス N と E から読むことができるので，サイクル n + 2 のスロット B にスケジューリングできる．エレメント 3 を資源予約表に登録するとき，エレメント 3 と使用したバイパス情報 %r2@N と %r3@E を一緒に保存し，アセンブリコード生成時に使用する．この資源予約表から，結果として図 1 の VLIW 命令列を生成することができる．

4.2 制御フローグラフの合流点

Procyon では，制御フローグラフの合流点で，複数の先行基本ブロックで定義されるレジスタの値を使用する命令のレジスタ・バイパス指定をどのように決定したらよいか，という問題が生じる．また，制御フローグラフは一般に有向循環グラフであるため，合流点の基本ブロックの命令スケジューリングを行う時に，先行基本ブロックの資源予約表を参照できない場合がある．我々のコンパイラでは，制御フローグラフの合流点の最適化のために，命令スケジューリング時に，制御フローグラフの合流点の基本ブロックの状態を次のふたつに場合分けして扱う．

4.2.1 先行基本ブロックの命令スケジューリングがすべて終了している場合

この場合には，先行基本ブロックで定義されるレジスタの値をバイパスから読むために，先行基本ブロックの資源予約表を参照し，基本ブロックの命令スケジューリングを行う．基本ブロック内に複数の先行基本ブロックが定義するレジスタの値を使用するエレメントが存在する場合には，可能ならば，先行基本ブロックの空きスロットにレジスタ間 move 命令を生成する

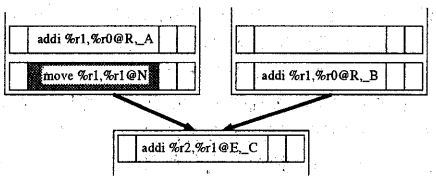

図 10　move 命令によるバイパスのタイミング調整
Fig. 10　Bypass timing control using move instruction

ことで基本ブロックの入口でのバイパスのタイミングを合わせる．バイパスのタイミング調整の例を図 10 に示す．図 10 では，合流点の基本ブロックの入口でふたつの先行基本ブロックが定義するレジスタ %r1 の値をバイパス E から読むために，先行基本ブロックの空きスロットにレジスタ間 move 命令を生成している．バイパスのタイミングを合わせることができない場合には，オペランドの値をレジスタから読むようにスケジューリングを行う．

4.2.2　命令スケジューリングが終了していない先行基本ブロックが存在する場合

この場合には，命令スケジューリングが終了していない先行基本ブロック以前の命令の出力がどのバイパスで使用できるかわからないので，レジスタへの書き込みが完了した後にそのレジスタの値を読む事を保証する必要がある．このために，命令スケジューリングが終了していない先行基本ブロック以前の基本ブロックが定義するオペランドを使用するエレメントは，レジスタへの書き込みが完了するサイクル以降に実行されるようにスケジューリングする．それ以外のレジスタに関しては 4.2.1 の方法でバイパスのタイミング調整を行う．

5.　ソフトウェア・バイパス制御方式の有効性

ソフトウェア・バイパス制御方式の有効性について，ソフトウェア・バイパス制御方式を利用した最適化の適用性と制御フローグラフの合流点の問題による弊害に関して評価を行った．評価に使用したプログラムの情報を表 1 に示す．これらのプログラムは CG 処理を行う C 言語のプログラムである．以下の評価はレジスタ割り当て前に，仮想レジスタを含む VLIW エレメントを命令スケジューリングして行った．評価に使用したコンパイラの命令スケジューリングのアルゴリズムの概要は以下の通りである．

命令スケジューリングのアルゴリズムの概要

- すべての基本ブロックのスケジューリングが終了するまで，以下を実行する．
  (1)　先行基本ブロックが存在しないか，あるいはすべての先行基本ブロックのスケジューリングが終了していて，かつまだスケジューリングが終了していない基本ブロック B を見つける．
  (2)　そのような基本ブロック B が存在すれば B のスケジューリングを行う．そのような基本ブロック B が存在しない場合は，先行基本ブロックにスケジューリングが終了した基本ブロックをもち，かつまだスケジューリングが終了していない先行基本ブロックの数が最も少ない基本ブロック B のスケジューリングを行う．

表 1　テストプログラム
Table 1　test programs

プログラム	基本ブロックの総数	VLIW エレメントの総数
P1	24	585
P2	24	537
P3	4	413
P4	58	681
P5	352	8600
P6	42	457
P7	22	595
P8	123	2314
P9	26	348
P10	419	9172

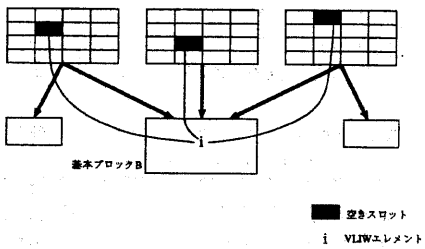

図 11　投機的命令移動
Fig. 11　Speculative code motion

基本ブロック B の命令スケジューリングのアルゴリズムは次の通りである．

(1)　基本ブロック B のすべての直前の先行基本ブロックの命令スケジューリングが終了していれば，B から直前の先行基本ブロックの空きスロットへ挿入可能な VLIW エレメントを投機的に移動する（図 11）．
(2)　基本ブロック B の入口でのバイパスのタイミング調整を行う（4.2 を参照）．
(3)　基本ブロック B の残りの命令で基本ブロック内の命令スケジューリングを行う．
(4)　基本ブロック B の後続の基本ブロックでまだスケジューリングを終了していない基本ブロックから，非投機的に移動可能で，かつ B の空きスロットに挿入可能な VLIW エレメントを移動する（図 12）．

5.1　生存区間の短い仮想レジスタ

ソフトウェア・バイパス制御方式によって，生存区間が演算パイプラインの長さ以下であるためにハード

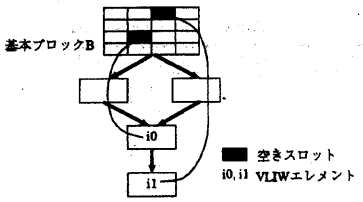

図 12 非投機的命令移動
Fig. 12 Non speculative code motion

表 2 生存区間の短い仮想レジスタ
Table 2 Short live ranges

	個数
仮想レジスタ	13436
レジスタ割り当てが不要な仮想レジスタ	4296

表 3 制御フローグラフの合流点の問題による弊害
Table 3 Problem at join basic blocks

	総数
基本ブロック	1094
合流点の基本ブロック	202
タイミング調整が必要な仮想レジスタ	1027
タイミング調整に成功した仮想レジスタ	544

ウェアレジスタを割り当てる必要がない仮想レジスタがプログラム中にどれだけ存在するかを調査した．結果を表 2 に示す．表 2 から，ソフトウェア・バイパス制御方式によって仮想レジスタの 30 % 以上に対してハードウェアレジスタを割り当てる必要がないことがわかる．

5.2 制御フローグラフの合流点の問題

制御フローグラフの合流点の問題による弊害についての調査結果を表 3 に示す．表 3 で，タイミング調整が必要な仮想レジスタは，合流点の基本ブロックの複数の先行基本ブロックの出口で生きていて，かつ合流点の基本ブロックがその値を使用する仮想レジスタを表す．この表から，合流点の基本ブロックの入口では平均 5 個の仮想レジスタのバイパスのタイミング調整が必要であり，タイミング調整が必要な仮想レジスタの 50 % 以上に対してタイミング調整が可能であるとわかる．

6. おわりに

コンパイラによるジオメトリプロセサ Procyon のソフトウェア・バイパス制御方式とソフトウェア・バイパス制御方式の有効性の評価について述べた．

ソフトウェア・バイパス制御方式をもつプロサセでは，変数の生存区間の概念が一般のプロセサと異なることを示した．このことを利用して，レジスタ割り当て時に生存区間が演算パイプラインの長さ以下の仮想レジスタをレジスタ干渉グラフから削除する最適化手法について述べ，その評価を行った．評価結果から，30 % 以上の仮想レジスタに対してハードウェアレジスタを割り当てずにすむことがわかった．また，ソフトウェア・バイパス制御方式を利用してレジスタ干渉グラフでの干渉を軽減する方法について述べ，ソフトウェア・バイパス制御方式を利用したレジスタ割り当てのアルゴリズムを示した．

ソフトウェア・バイパス制御方式のために，コンパイラによる命令スケジューリング時に，制御フローグラフの合流点で生じる問題とその解決方法について述べた．複数の先行基本ブロックで定義されるレジスタの値を使用する命令のレジスタ・バイパス指定は，可能ならば VLIW 命令の空きスロットにレジスタ間 move 命令を生成して同期を取り，それができない場合はレジスタファイルから読むことによって，対処できることを示した．評価結果より，各合流点でタイミング調整が必要となる仮想レジスタは平均 5 個であり，タイミング調整が必要な仮想レジスタの 50 % 以上に対してタイミング調整が可能であることがわかった．

以上のことから Procyon のソフトウェア・バイパス制御方式は，インタロックがないことによる問題点はコンパイラで補うことが可能であり，ハードウェアを簡素化したことよる利点とソフトウェア・バイパス制御方式を利用した最適化方式により，ハードウェアとソフトウェアの両方からの性能向上が期待できる手法であると言える．

今後はソフトウェア・バイパス制御方式を利用した広域命令スケジューリングとレジスタ割り当ての協調技法について検討することを考えている．

評価に使用したコンパイラは、主要部を UtiLisp[4] で記述した．SunOS および Solaris 上で開発し、運用している．

謝辞 日頃ご指導頂く，(株) 富士通研究所和田常任顧問，同研究所マルチメディアシステム研究所林所長，村松所長代理に感謝いたします．また，同研究所コンピュータシステム研究部の研究員諸氏に感謝いたします．

参考文献

1) 安里彰 他: ジオメトリプロセサ Procyon –概要–, 第 55 回情報処理学会全国大会 (1997).
2) 西崎慎一郎 他: ジオメトリプロセサ Procyon –コンパイラ–, 第 55 回情報処理学会全国大会 (1997).
3) G. J. Chaintin: Register Allocation & Spilling via Graph Coloring, Proceedings of the ACM Symposium on Compiler Construction(1982).
4) 田中 哲朗: SPARC の特徴を生かした UtiLisp/C の実現法, 情報処理学会論文誌, Vol.32, No.5, pp.684–690 (1991).