計算機アーキテクチャ 130－9
（1998．8．5）

コンパイラによるソフトウェア・バイパス制御方式とその評価

新井正樹† 安里彰†
小沢年弘† 木村康則†

ソフトウェア・バイパス制御方式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プロセッサアーキテクチャである．ソフトウェア・バイパス制御方式を採用したアーキテクチャでは，レジスタ割り当て時に特有の最適化が可能になるけれども，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のバイパスのタイミング調整を行わなければならない．本論文では，コンパイラによるソフトウェア・バイパス制御方式とその評価について述べる．

Evaluation of Software Bypass Control Mechanism by Compiler

Masaki Arai,† Akira Asato,† Toshihiro Ozawa†
and Yasunori Kimura†

The processor with software bypass control mechanism doesn't need a hardware bypass control circuit it makes hardware design simple. 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 must manage the data on the pipeline registers and explicitly write which bypass result or register is used in assembly language. This mechanism gives the compiler more opportunities for code optimizations in the register allocation path. But the compiler must adjust bypass timing at join basic blocks when instruction scheduling. In this paper, we show how to control the bypass by the compiler and evaluate the effect of it.

1. はじめに

ソフトウェア・バイパス制御方式は，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方式である．つまり，各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．ジオメトリプロセッサ Procyon[1] ではソフトウェア・バイパス制御方式を採用している．

ソフトウェア・バイパス制御方式のプロセッサでは，変数の生存区間の概念が一般のプロセッサと異なる．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，コンパイラによる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．

ソフトウェア・バイパス制御方式をもつプロセッサの欠点として，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，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，という問題がある．

本論文では，コンパイラによるソフトウェア・バイパス制御方式とそ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

ソフトウェア・バイパス制御方式は，ハードウェアを簡素化するという目的と，バイパスのタイミングを考える必要があるということから，VLIW アーキテクチャと相性が良い．したがって，以下の説明ならびに評価ではソフトウェア・バイパス制御方式を採用した VLIW アーキテクチャのプロセッサモデルを使用した．

以下の説明に使用する，プロセッサの命令の種類とパイプラインステージの関係を表 1 に示す．表 1 で，D はデコードステージを，R は演算結果の値をレジスタファイルから読むことができるステージを意味する．その他のステージ名が－以外のステージ名はバイパスへの出力を意味する．入力オペランドはすべてデコードステージで読むものとする．

ソフトウェア・バイパス制御方式をもつプロセッサ

† 新情報富士通研
RWCP Multi-Processor Computing Fujitsu Laboratory

表 1　評価用マシンモデルのパイプラインステージ
Table 1　Pipeline of instructions

cycle	1	2	3	4	5
整数	D	E	N	W	R
整数比較	D	-	R		
浮動小数点	D	-	F-E2	F-W	R
浮動小数点比較	D	-	-	R	
浮動小数点 move	D	F-E1	F-E2	F-W	R
ロード	D	-	-	LD	R
分岐	D				

```
(1): nop; add  %r0@R, %r1@R, %r2; nop; nop;
(2): nop; addi %r2@E, 1024,  %r3; nop; nop;
(3): nop; add  %r2@N, %r3@E, %r4; nop; nop;
```

図 1　ソフトウェア・バイパス制御方式をもつプロセッサのアセンブリコード例
Fig. 1　Example of assembly codes of software bypass control

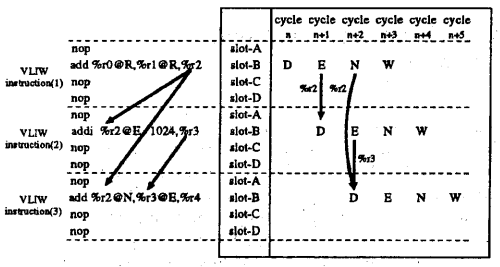

図 2　実行時のパイプラインの流れ
Fig. 2　Pipeline chart of execution

のため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．図 1 では最後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．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．

ソフトウェア・バイパス制御方式をもつプロセッサでは，アセンブリ言語プログラマあるいはコンパイラが，各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．

3. ソフトウェア・バイパス制御方式の利点と欠点

一般のバイパス制御方式をもつプロセッサとソフトウェア・バイパス制御方式をもつプロセッサのハードウェアを比較すると，ソフトウェア・バイパス制御方式をもつプロセッサの方が，実行時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決定するためのハードウェアが不要になる分，簡単になる．

また，ソフトウェア・バイパス制御方式をもつプロセッサでは，バイパス制御をソフトウェア側で行うようにしたために，コンパイラによるレジスタ割り当て時に特有の最適化が可能になる．この最適化によって，演算パイプラインのパイプラインレジスタをハードウェアレジスタとして利用できる．この最適化手法については，第 4 節で述べる．

ソフトウェア・バイパス制御方式をもつプロセッサの欠点として，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，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，という問題がある．また，レジスタ割り当て時にどのようにスピルコードを生成したらよいかという問題がある．これらの問題については，第 5 節で述べる．

最後にソフトウェア・バイパス制御方式を利用した最適化の効果と制御フローグラフの合流点のオーバーヘッドに関して評価を行う．

4. 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．

ソフトウェア・バイパス制御方式をもつプロセッサでは，仮想レジスタの生存区間の概念が一般のプロセッサと異なる．ソフトウェア・バイパス制御方式をもつプロセッサでは，仮想レジスタの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，その仮想レジス

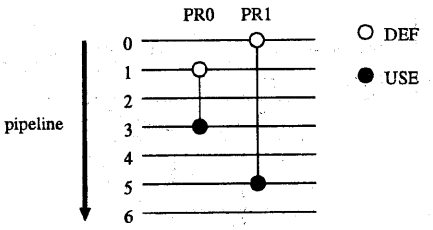

図 3 生存区間解析の結果
Fig. 3 Result of live range analysis

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，生存区間解析の結果が図3であるような仮想レジスタ PR0 と PR1 について考える．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．また，PR0 をレジスタ割り当て時に無視することができるために PR1 の隣接頂点数が 1 減少する．すなわち，ソフトウェア・バイパス制御方式をもつプロセッサでは，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．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) 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．
(4) 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．

5. バイパスのタイミング調整の問題

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開発したコンパイラでの対処方法を以下に述べる．

ソフトウェア・バイパス制御方式をもつプロセッサでは，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，

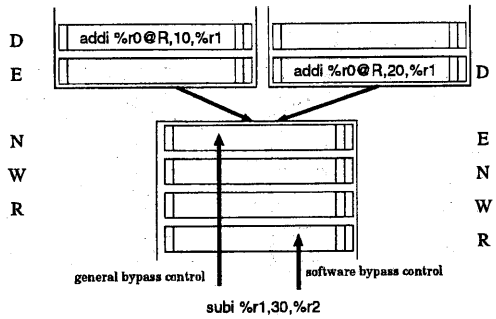

図 4 合流点でのバイパスのタイミング調整
Fig. 4 Bypass timing control at join basic block

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スケジューリングの結果を参照できない場合がある．我々の開発したコンパイラでは，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．

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スケジューリングの結果を参照し，合流点の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の出口で生きているレジスタの値を使用する命令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 4 に示す．図 4 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 の値を使用する命令 subi をスケジュールしようとしている．一般のバイパス制御方式では，命令 subi は基本ブロックの入口にスケジュールすることが可能であるけれども，ソフトウェア・バイパス制御方式ではバイパスのタイミングが合っていないために，値を確実にレジスタファイルから読むことが可能な 3 サイクル遅れた位置に命令 subi をスケジュールしなければならない．我々のコンパイラでは，このような場合，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 命令を生成して，バイパスのタイミングを調整する．レジスタ間 move 命令によるバイパスのタイミング調整の例を図 5 に示す．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．

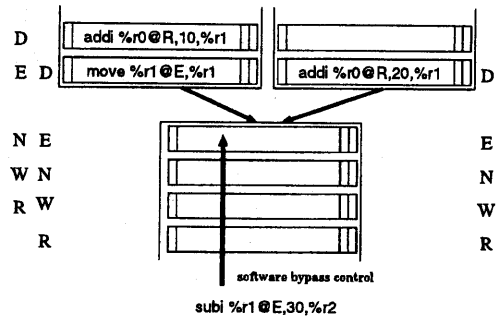

図 5 move 命令を使用した合流点でのバイパスのタイミング調整
Fig. 5 Bypass timing control using move instruction

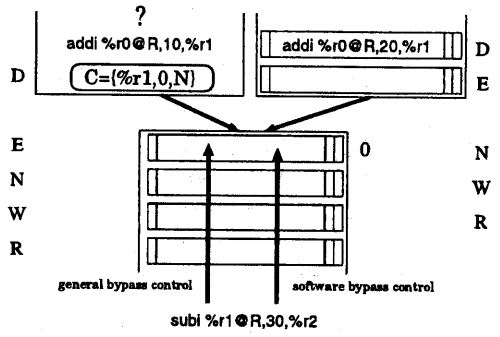

図 6 合流点でのバイパスのタイミング調整
Fig. 6 Bypass timing control at join basic block

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

先行基本ブロックに命令スケジューリングが終了していない基本ブロックが存在する場合には，その先行基本ブロックの状態を後続基本ブロックのためのスケジューリング制約をもつ場合ともたない場合のふたつに場合分けしてタイミング調整を行う．

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後続基本ブロックのためのスケジューリング制約をもたない場合は，その先行基本ブロックは任意のタイミング調整が可能であると仮定して，すでにスケジューリングが終了している先行基本ブロックのスケジューリングの結果を参照し，合流点の基本ブロックのバイパスのタイミング調整を行う．バイパスのタイミング調整が終了した後，タイミング調整の結果の情報を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に，その基本ブロックのスケジューリング時に満たさなければならないスケジューリング制約として登録する．

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すでに他の後続基本ブロックのためのスケジューリング制約をもつ場合は，その制約も同時に満たすように現在の基本ブロックのバイパスのタイミング調整を行う．

先行基本ブロックにすでに存在する制約を満たすことが出来ない場合や，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．バイパスのタイミング調整のための制約をもつ基本ブロックの命令スケジューリングを行う場合は，必要ならば後続基本ブロックのためのタイミング調整用の move 命令を生成するか，あるいは nop 命令を挿入することによって基本ブロック長を延ばして，制約を満たす．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のバイパスのタイミング調整の例を図 6 に示す．図 6 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 の値を使用する命令 subi をスケジュールしようとしている．スケジューリングがまだ終了していない先行基本ブロックへ，現在スケジューリングを行っている基本ブロックの入口で，バイパス N にレジスタ %r1 の値を出力することを強制する制約 C を登録する．この制約によって，命令 subi は基本ブロックの入口にスケジュールすることが可能になる．このような制約を利用することによって，制御フローグラフのエッジの実行頻度に偏りがある場合でも，実行頻度の高いパスのバイパスのタイミング調整を優先的に行うことが可能になる．

6. 評　　価

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効果と制御フローグラフの合流点のオーバーヘッドに関して評価を行った．評価用のベンチマークプログラムとして SPECint92 の 4 つのプログラムと clinpack の単精度版 (SP) と倍精度版 (DP) をループアンローリングしたもの (UNROLL) としないもの (ROLL) を使用した．評価用のマシンモデルとして，以下を仮定した．

- 命令セット SPARC v8 ベース．
- 分岐のディレイスロットなし．
- 2 並列 VLIW と 4 並列 VLIW．
- 命令の種類とパイプラインステージの関係は表 1 を仮定した．

並列度を 2，4 にした場合の各命令スロットと命令の種類の関係は以下の通りとした．

並列度 2：

- 整数演算命令（スロット A, B）
- 浮動小数点演算命令（スロット A, B）
- ロードストア命令（スロット A）
- 分岐命令（スロット B）

並列度 4：

- 整数演算命令（スロット A, B, C, D）

- 浮動小数点演算命令（スロット A, B, C, D）
- ロードストア命令（スロット C）
- 分岐命令（スロット D）

6.1 ソフトウェア・バイパス制御方式を利用した最適化の評価

ソフトウェア・バイパス制御方式を利用した最適化の評価を行った．整数レジスタと浮動小数点レジスタの数をそれぞれ 8 個ずつにした場合と 16 個ずつにした場合に，ソフトウェア・バイパス制御方式を採用したアーキテクチャと一般のバイパス制御方式のアーキテクチャで各ベンチマークプログラムの総実行サイクル数がどのように変化するかを調査した．評価に使用したコンパイラでは，以下の手順で命令スケジューリングとレジスタ割り当てを行った．

(1) 命令スケジューリングを行う．
(2) 命令スケジューリングの結果を VLIW 命令に変換する．
(3) VLIW 命令の状態でレジスタ割り当てを行う．
(4) もし，すべての仮想レジスタにハードウェアレジスタを割り当てることができれば終了する．
(5) もし，スピルコードが必要ならば VLIW 命令を逐次命令に戻し，レジスタ割り当てパスでハードウェアレジスタを割り当てることが出来なかった仮想レジスタのためのスピルコードを挿入し，(1) に戻る．

命令スケジューリングパスでは，基本ブロック単位に命令スケジューリングを行う．各基本ブロックのスケジューリングの前後に，可能ならばすでにスケジューリングが終了した基本ブロックの空きスロットに命令を移動する．ソフトウェア・バイパス制御方式を採用したアーキテクチャモデルでは，各基本ブロックの命令スケジューリングを行う前に基本ブロックの入口での上流の基本ブロックによる影響を考慮する．今回の評価では，制御フローグラフの合流点でバイパスのタイミング調整を行う場合，直前の基本ブロックだけを調整のための変更の対象範囲とした．評価結果を**表 2** に示す．表 2 で，soft/hard はソフトウェア・バイパス制御方式を採用した場合に一般のバイパス制御方式に比べて，各ベンチマークプログラムの総実行サイクル数がどれだけ増加したかを表す．表 2 からソフトウェア・バイパス制御方式を利用した最適化の効果は，レジスタ数を制限してもほとんどないことがわかる．これは，ソフトウェア・バイパス制御方式を利用した最適化によって，レジスタ干渉グラフから生存区間が演算パイプラインの長さ以下であるレジスタを削除して，生成されるスピルコード量を減らしたけれども，残りのスピルコードが性能にクリティカルな部分には存在しないためと，VLIW プロセッサモデルで評価を行ったためにスピルコードによる性能低下が隠されたためである．表 2 で，ソフトウェア・バイパス制御方式を採用したアーキテクチャが一般のバイパス制御方式のアーキテクチャと比べて，総実行サイクル数が増加しているのは，ソフトウェア・バイパス制御方式を利用した最適化の効果がほとんどなく，次に述べる制御フローグラフの合流点のオーバーヘッドによるものである．

表 2 ソフトウェア・バイパス制御方式を利用した最適化の評価
Table 2 Evaluation using software bypass control mechanism

プログラム (データファイル)	並列度	使用可能なレジスタ数	
		8 soft/hard	16 soft/hard
008.espresso	2	1.16	1.16
(input.ref/bca.in)	4	1.17	1.17
022.li	2	1.19	1.19
(input.short/li-input.lsp)	4	1.19	1.19
023.eqntott	2	1.09	1.09
(input.short/ex0.eqn)	4	1.10	1.10
026.compress	2	1.19	1.18
(input.ref/in)	4	1.19	1.20
clinpack (DP:UNROLL)	2	1.02	1.02
	4	1.02	1.02
clinpack (DP:ROLL)	2	1.04	1.04
	4	1.05	1.05
clinpack (SP:UNROLL)	2	1.02	1.02
	4	1.02	1.02
clinpack (SP:ROLL)	2	1.04	1.04
	4	1.05	1.05

6.2 制御フローグラフの合流点のオーバーヘッドの評価

ソフトウェア・バイパス制御方式をもつプロセッサでの，制御フローグラフの合流点のオーバーヘッドについて調査した．各ベンチマークプログラムに対して，一般のバイパス制御方式とソフトウェア・バイパス制御方式で，並列度が 2 と 4 の場合に各ベンチマークプログラムの総実行サイクル数がどのように変化するかを調査した．バイパスのタイミング調整のオーバーヘッドのみを評価するために，同じレジスタ割り当てを行った逐次命令列を，それぞれのマシンモデルに合わせた命令スケジューリングをすることで比較を行った．命令スケジューリングのアルゴリズムは基本ブロック内のみで命令移動を行う局所命令スケジューリングを用い，ソフトウェア・バイパス制御方式をもつプロセッサモデルに対しては，各基本ブロックの命令スケジューリングを行う前に先行基本ブロックの影響を考慮した．制御フローグラフの合流点でのバイパスのタイミング調整では，第 6.1 節と同様に直前の基本ブロックだけを調整のための変更の対象範囲とした．評価結果を**表 3** に示す．表 3 で，soft/hard はソフトウェア・バイパス制御方式を採用した場合に一般のバイパス制御方式に比べて，各ベンチマークプログラムの総実行サイクル数がどれだけ増加するかを表す．表 3 からソフトウェア・バイパス制御方式を採用した場合には，SPECint92 のプログラムに対しては，制

表 3 制御フローグラフの合流点のオーバーヘッドの評価
Table 3 Problems at join basic blocks

プログラム (データファイル)	並列度	soft/hard
008.espresso	2	1.12
(input.ref/bca.in)	4	1.13
022.li	2	1.21
(input.short/li-input.lsp)	4	1.20
023.eqntott	2	1.11
(input.short/ex0.eqn)	4	1.11
026.compress	2	1.18
(input.ref/in)	4	1.19
clinpack (DP:UNROLL)	2	1.02
	4	1.03
clinpack (DP:ROLL)	2	1.05
	4	1.06
clinpack (SP:UNROLL)	2	1.02
	4	1.03
clinpack (SP:ROLL)	2	1.05
	4	1.06

御フローグラフの合流点でのタイミング調整のために，一般のバイパス制御方式に比べて約 10 % から 20 % の性能低下を引き起こしていることがわかる．これは，SPECint92 のプログラムの基本ブロックの長さが短いために合流点でのバイパスのタイミング調整のオーバーヘッドが大きく影響しているためである．それに対して，clinpack では，基本ブロックの長さが長いために，合流点でのバイパスのタイミング調整のオーバーヘッドによる性能低下は小さいことがわかる．

7. まとめ

コンパイラによるソフトウェア・バイパス制御方式について述べ，その評価を行った．一般のバイパス制御方式をもつプロセッサとソフトウェア・バイパス制御方式をもつプロセッサのハードウェアを比較すると，ソフトウェア・バイパス制御方式をもつプロセッサの方が，実行時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決定するためのハードウェアが不要になる分，簡単になる．

また，ソフトウェア・バイパス制御方式をもつプロセッサでは，バイパス制御をソフトウェア側で行うようにしたために，コンパイラによるレジスタ割り当て時に特有の最適化が可能になる．この最適化によって，演算パイプラインのパイプラインレジスタをハードウェアレジスタとして利用できる．この最適化の効果を評価した結果，今回使用したベンチマークプログラムに対しては，実行性能にはほとんど効果がないことがわかった．これは，ソフトウェア・バイパス制御方式を利用した最適化によって，レジスタ干渉グラフから生存区間が演算パイプラインの長さ以下であるレジスタを削除して，生成されるスピルコード量を減らしたけれども，残りのスピルコードが性能にクリティカルな部分には存在しなかったためと，VLIW プロセッサモデルで評価を行ったためにスピルコードによる性能低下が隠されたためである．

ソフトウェア・バイパス制御方式をもつプロセッサの欠点として，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，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．このオーバーヘッドによる各ベンチマークプログラムの総実行サイクル数の増加は，基本ブロック長の短いプログラムに対しては，10 % から 20 % であり，基本ブロック長の長いプログラムに対しては，5 % 程度であることがわかった．このことから，ソフトウェア・バイパス制御方式をもつプロセッサで，基本ブロック長の短いプログラムを実行する場合には，制御フローグラフの合流点で，今回評価に用いたよりも強力なバイパスのタイミング調整手法が必要であるとわかった．

8. おわりに

今後は，ソフトウェア・バイパス制御方式の利点が生きると考えられる数値演算あるいはメディア系アプリケーションや，ループ展開やソフトウェアパイプライニングなどの最適化との関係について調査を行う予定である．そのために，ソフトウェア・バイパス制御方式をもつプロセッサのための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も検討する予定である．

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取締役，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 のアーキテクチャ, 情報処理学会研究報告 97-ARC-126 (1997).
2) 新井 正樹 他. ジオメトリプロセサ Procyon の評価, 情報処理学会研究報告 97-ARC-126 (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).