論　文

パス選択によるソフトウェアパイプライニング

中西知嘉子[†]　安藤　秀樹[††]　原　哲也[†]　中屋　雅夫[†]

Software Pipelining with Path Selection

Chikako NAKANISHI[†], Hideki ANDO[††], Tetsuya HARA[†], and Masao NAKAYA[†]

あらまし　ソフトウェアパイプライニングは，VLIWやスーパスカラプロセッサにおける効果的なスケジューリング技術である．従来のアルゴリズムでは，ループの全制御パスをパイプライン化するため，実行頻度の低い制御パスによる資源占有や長いデータ依存パスによって，性能向上が制限されていた．本論文では，ループの中の実行頻度の高いパスを選択してパイプライン化するアルゴリズムを提案する．本アルゴリズムは，実行頻度の高いパスを最適にパイプライン化できるだけでなく，従来パイプライン化が困難であった最内ループでないループに対してもパイプライン化できる可能性を与える．非数値計算応用のベンチマークプログラムを用いて，従来のアルゴリズムとの比較評価を行った結果，本アルゴリズムの高い有効性を確認できた．

キーワード　ソフトウェアパイプライニング，モジュロスケジューリング，プレディケート実行

1. まえがき

ソフトウェアパイプライニングは，VLIWやスーパスカラプロセッサにおける効果的なループ最適化技術である．ソフトウェアパイプライニングは，複数のイタレーションの実行をソフトウェア的にパイプライン化し，ループ全体の実行時間を短くする．パイプライン化されたコードは，プロローグ，定常状態，エピローグからなり，繰返し回数が多いループでは多くの実行時間が定常状態で費やされる．そのため，ソフトウェアパイプライニングアルゴリズムの目標は，定常状態のサイクル数が小さくなるようにスケジュールすることである．

これまでに，多数のソフトウェアパイプライニングアルゴリズムが提案されている．その中の一つであるモジュロスケジューリング[1]は，先行制約，資源制約を同時に満たしつつ，小さな定常状態を得ることのできるアルゴリズムである．モジュロスケジューリングの欠点は，基本的には，ループボディが単一の基本ブロックからなるループを最適化するアルゴリズムであり，条件分岐命令を含むループには，適用が困難であることである．条件分岐命令は，特に，非数値計算応用では，多くのループに含まれている．そのため，条件分岐命令を含むループを最適にスケジュールする技術は重要である．

これまで，条件分岐命令を含むループにモジュロスケジューリングを適用するためには，スケジューリング前に，ループボディをストレートラインコードに変換することで対処してきた．例えば，ハイアラキカルリダクション（HR）[1]は，ifブロック全体を一つの複合命令とし，他の命令と同等にモジュロスケジューリングを適用する．また，プレディケート実行を用いる方法（PE）[2]，エンハンストモジュロスケジューリング（EMS）[3]は，if変換によって条件分岐命令を取り除く．プレディケートとは，その命令の制御依存情報である．PEでは命令にこれを付加し，ハードウェアが実行時に評価する．真と評価された命令のみ実行される．一方，EMSでは，条件ごとに異なるコードを用意し，どのコードを実行するかを選択する．

これらのアルゴリズムは，ループ内のすべての制御パス上の命令を，同時にスケジュールする．そのため，実行サイクルの長い方のパスで定常状態のサイクルが決まってしまい，短い方のパスは必ずしも最適にスケジュールされない．従って，短い方のパスが頻繁に実行されるループでは，ペナルティが大きくなってしまう．これに対してEMSにおいて，実行サイクルの短い方のパスで，定常状態のサイクル数を決める方

† 三菱電機株式会社，システムLSI開発研究所，伊丹市
System LSI Laboratory Mistubishi Electric Corporation, 4-1 Mizuhara, Itami-shi, 664 Japan
†† 名古屋大学大学院工学研究科電子情報学専攻，名古屋市
Department of Information Electronics, School of Engineering, Nagoya University, Furo, Chikusa, Nagoya-shi, 464-01 Japan

法がある[4]．しかし，この方法は，逆に実行サイクルの長い方のパスが最適にスケジュールされるとは限らない．そのため，長い方のパスが頻繁に実行される場合のペナルティが大きい．

本論文では，プレディケーティング[5]と呼ぶプレディケート実行と投機的実行を合わせて支援するハードウェア機構をもつVLIWに適し，実行頻度の高いパスを選択してソフトウェアパイプライニングを行うアルゴリズムを提案する．本アルゴリズムの利点は，実行頻度の高いパスを最適にパイプライン化できるという点である．すなわち，実行頻度の低いパスによって最適化が妨げられることがない．更に，以下の二つの利点がある．パスの選択は，単一のパスを選択するトレーススケジューリング[6]とは異なり，複数のパスの選択を行う．そのため，分岐方向に著しい偏りがない場合，複数のパスの命令をバランスよくスケジュールでき，実行時にどちらのパスが実行されても高い性能が得られる．また，パスの選択によって，従来パイプライン化が困難であった最内ループでないループに対してもパイプライン化できる可能性を与える．

本論文では，まず，2.で本アルゴリズムを適用するプレディケーティングについて説明する．3.では，ソフトウェアパイプライニングのアルゴリズムについて述べる．4.では，従来の技術との比較を行い，5.でPE，EMS，本アルゴリズムの性能比較を行う．そして，6.でまとめる．

2. プレディケーティング

本章では，プレディケーティング[5]と呼ぶ，本アルゴリズムがターゲットにしているアーキテクチャについて簡単に述べる（詳しくは，文献[5]を参照）．

プレディケーティングは，コンパイラによる投機的命令移動[7]を支援するアーキテクチャ上の機構である．投機的命令移動とは，分岐を越えた命令移動のことを言う．投機的に移動された命令は，実行によりレジスタやメモリのデータを上書きしプログラムの意味を破壊する可能性があるほか，不要に例外を発生し，実行の不正な停止，あるいは，性能の大きな低下を起こす可能性がある．プレディケーティングは，これらの二つの問題に対して対処する．コンパイラは，投機的命令移動に関する問題の解決をプレディケーティングに任せることができるため，投機的命令移動を自由に行うことができる．

プレディケーティングでは，命令に制御依存情報であるプレディケートをエンコードする．投機的に移動された命令を実行した場合，その実行結果をプレディケートと共にバッファリングする．後に，制御依存が解消した時点で投機的実行結果の有効化あるいは無効化を行う．この制御をバッファリングされたプレディケートを用いて行う．

命令は次のような形式をもっている．

プレディケート？オペレーション

プレディケートは，その命令が依存する分岐条件の論理式である．命令の意味は，「プレディケートが真であるときのみ，オペレーションの結果が有効となる」である．

分岐条件の値は，CCR（コンディションコードレジスタ）に保持される．CCRは複数のエントリをもち，各エントリは異なる分岐条件に対応し，以下の3値のいずれかの値を記憶する．

- 真：対応する分岐条件が真に定義されている．
- 偽：対応する分岐条件が偽に定義されている．
- 未定義：対応する分岐条件が未定義である．

各命令のプレディケートは，CCRのエントリの値を参照することによって評価される．例えば，

c1＆c2 ？ add s9，s10，s11

は，CCRの第1エントリ（c1）と第2エントリ（c2）がともに真であるときに，add s9，s10，s11の実行結果が有効化されることを示す．CCRの内容にかかわらず，常に，実行結果が有効化されることを示すプレディケートは，alwと表す．

ハードウェアでは，命令の発行時点で，まず，プレディケートの評価を行う．プレディケートが参照するCCRのすべてのエントリの値が定義されており，プレディケートの値が真であれば，この命令の実行結果は有効であるので，通常のマシンの命令と同様に実行を行い，実行結果をレジスタファイルに書き込む．プレディケートの値が偽であれば，既に，命令の実行結果が無効となることがわかっているので，実行せずに無効化する．これらの動作は，従来のプレディケート実行[5]と同様である．

プレディケーティングでは，更に，命令の発行時点でプレディケートが参照するCCRのエントリの値が未定義であるために，プレディケートの値の真偽が不明な場合でも，命令を実行する．但し，この場合の実行結果は，シャドウ構造に書き込む．シャドウ構造は，レジスタファイルとストアバッファに用意する．シャドウ構造には，プレディケートを記憶するフィー

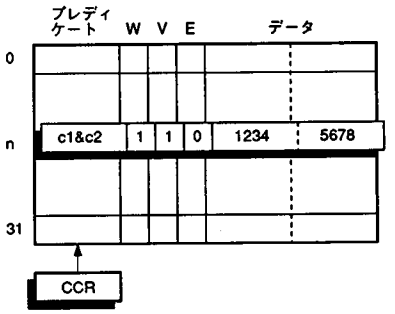


図 1 プレディケート付きレジスタファイル
Fig. 1 Predicated register file.

ルドを用意し，書込みの際には，同時にその命令のプレディケートも書き込む．更に，プレディケートフィールドに，CCR を参照しプレディケートの値を評価するハードウェアをもたせる．プレディケートはそのハードウェアによって毎サイクル評価され，評価結果が真になればシャドウ構造の内容を有効化する．偽になれば破棄する．

CCR の全エントリは，制御移行を行うジャンプ命令の実行によって「未定義」にハードウェアによって設定される．コンパイラは，**3.1** で述べる領域ごとにスケジューリングを行う．プレディケーティングの機能を利用するために，投機的命令移動は原則的には，領域内基本ブロック間および領域の次のブロックから領域内のブロックのみに制限する [8]．また，**3.2** で述べるように，領域間を制御が移動するときのみジャンプ命令が実行される．このため，CCR の全エントリは，ジャンプ命令の実行によって「未定義」にハードウェアにより設定する．

図 1 にレジスタファイルの構成を示す．二つのデータフィールドの一方が通常のレジスタであり，他方がシャドウレジスタである．どちらがシャドウレジスタであるかをフラグ W が示す．フラグ V は，シャドウレジスタが有効な値を保持していることを示す．フラグ E は，当該エントリに書込みを行った命令が投機的例外を起こしたことを示す．格納されたプレディケートが真と評価された場合，フラグ W を反転することによってシャドウレジスタの値を有効化する．偽と評価された場合，フラグ V をリセットすることによってシャドウレジスタの値を無効化する．

フラグ E は，例外処理のためのフィールドである．フラグ E がセットされているエントリのプレディケートが真となった場合（例外の有効化）は，まず，シャドウ構造のすべての値を破棄する．その後，命令の再実行を行い，マシン状態の再構築を行う．命令の再実行は，例外が有効化された点において，実行結果がまだ有効化されていない命令のみ選択して行う．有効化された例外は，再実行時に再発生し，その時点で処理される（詳細については [5] を参照）．

3. ソフトウェアパイプライニング

一般的に，投機的命令移動を行うことは，ソフトウェアパイプライニングにとって，大きな利益があると考えられる．ソフトウェアパイプライニングの目的は，複数のイタレーションを同時に実行できるよう命令の移動を行うことによって，より小さな定常状態を得ることである．これは，繰返し回数が静的に決定されるループに関しては，ループを繰り返すためのジャンプ命令（以下，ループバックのジャンプ命令と呼ぶ）を越える投機的な命令移動が必要ないので，比較的容易であるが，そうでないループでは，これが必要となり，困難であった（例えば，レジスタリネーミングの際の新たなコピー命令の挿入が必要）．これに対して，プレディケーティングを利用すれば，コンパイラは投機的命令移動により生じる問題の解決をハードウェアに任せることができるので，コンパイラはこの問題から開放される．

本章で提案するアルゴリズムは，プレディケーティングを搭載する VLIW におけるソフトウェアパイプライニングのアルゴリズムである．本アルゴリズムは，上記理由でプレディケーティングを利用するだけでなく，実行頻度の高いパスを選択することにより，投機的命令移動の機会を多くし，より積極的に利用するものである．選択したパスを最適化するには，そのパスに含まれる基本ブロックの境界を越える命令移動が必須であるが，プレディケーティングを利用することによりこれを行う．

本アルゴリズムは，以下に示す 5 ステップよりなる．

1. 領域を形成する．
2. プレディケートコードを作成する．
3. 制約グラフを作成する．
4. モジュロスケジューリングを行う．
5. 命令コードを作成する．

本章では以下，各々のステップについて例を用いて説明する．

3.1 領域の形成

本アルゴリズムでは，コンパイラは，ループを構成する制御フローグラフ（CFG）の中から，プロファイルを利用して，実行頻度の高い基本ブロックを選択し，その集合に対してスケジューリングを行う．この集合を領域（region）[5]と呼ぶ．領域に含むブロックを選択後，制御の合流点を越える命令移動に伴うブックキーピングをなくすため，領域の先頭以外から入る制御エッジ（サイドエッジと呼ぶ）がある場合，そのエッジが入るブロック以降をすべて複写し，サイドエッジを除く．すなわち，領域の先頭ブロック（ヘッダブロック）が，領域内のすべての他のブロックを支配（dominate）[9]するように領域を変換する．以下に，領域形成のアルゴリズムを示す．

（1） ループの入り口の基本ブロックをヘッダブロックとし，領域とする．

（2） 領域から外に向かってCFGのエッジをたどり，ヘッダから到達する確率が最も高いブロック（次ブロック）を領域に加え，領域を拡張していく．但し，次ブロックが以下の条件を満たす場合は領域に加えない．

- 現在拡張している領域に含まれている．
- プロシージャコールを含む．
- スケジュール済みである．
- 拡張中の領域に含まれる分岐を含むブロックの数がCCRのエントリ数と等しく(注1)，次ブロックが分岐を含む．
- 領域内のブロックから次ブロックへの分岐確率が基準として定めた値（最小分岐確率）以下である．

（3） 領域拡張後，領域のヘッダが領域内のすべてのブロックを支配できるよう，必要ならばブロックを複写し，領域を形成する．

領域の形成後，領域がループをなしている場合は，その領域をソフトウェアパイプライニングによりスケジューリングする．それ以外の場合は，ソフトウェアパイプライニングによるスケジューリングは行わない．

例を使って説明する．図2（a）は，領域形成前のCFGである．各ノードは，基本ブロックを表し，エッジに付加した数字は，分岐確率を表す．CCRのエントリ数を4，最小分岐確率を0.2とする．領域形成のアルゴリズムに従うと，まず（1）により，ブロックAが領域となる．次に（2）により，ブロックD，そしてブロックEが領域に加えられる．ブロックBは，

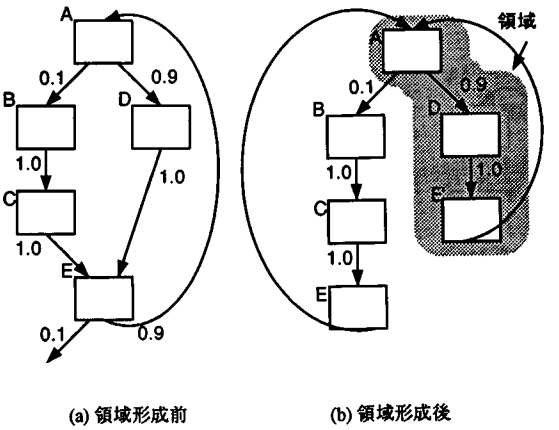


図2 領域 （a）領域形成前，（b）領域形成後
Fig. 2 Region (a) Before region selection, (b) After region selection.

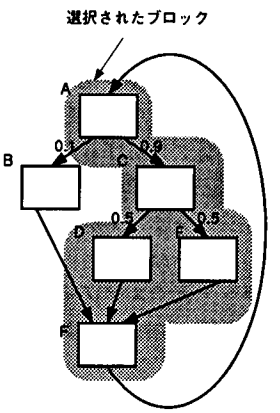
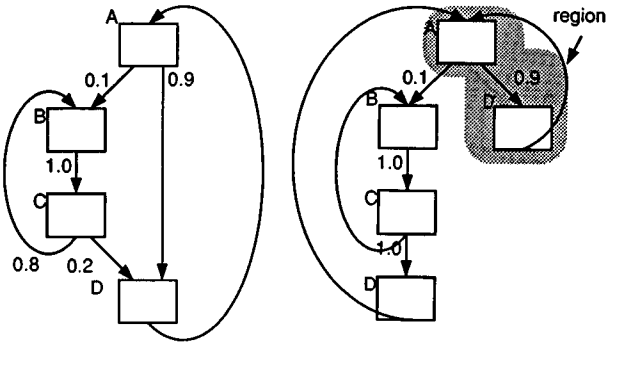


図3 ブロックの選択
Fig. 3 Selection of basic blocks.

AからBへの分岐確率が0.2以下なので加えない．領域拡張の結果，ブロックA，D，Eが選択される．ブロックEは，領域外のブロックCからのパスがあるので，（3）により，ブロックEを複写する．領域形成後のCFGを図2（b）に示す．ハッチングした部分が，本アルゴリズムを適用する領域である．

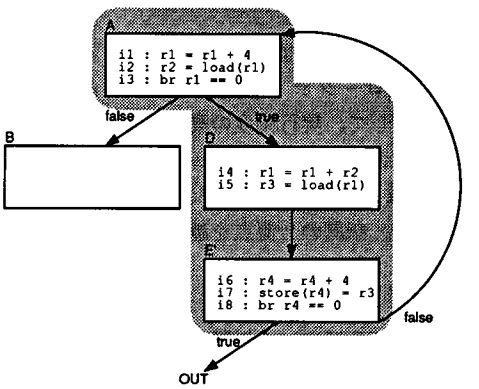
本方式には，特徴が三つある．一つは，実行頻度の高いパスが最適にスケジュールできることである．なぜなら，実行頻度の高いパスだけが選択され，スケジュールの対象となるからである．そのため，実行頻度の低い命令によって資源が無駄に消費され，実行頻度

(注1)：プレディケーティングでは，CCRのエントリ数と等しい数の分岐を越えて命令を移動できる．

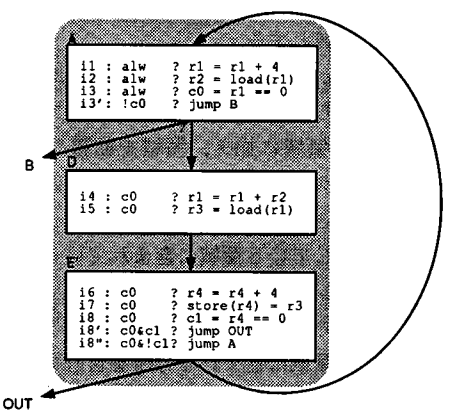


(a) 領域形成前　　(b) 領域形成後

図 4　外側ループ
Fig. 4　Outer loop.



(a)　領域と命令



(b)　プレディケート・コード

図 5　命令コード　(a)領域と命令，(b)プレディケートコード
Fig. 5　Code (a) Region and instructions (b) Predicated code.

の高いパスの最適化が妨げられることが防げる．例えば，図3の場合，ブロックA，C，D，E，Fが選択され，スケジュールの対象となる．実行頻度の低いブロックBはスケジューリングの対象とならないため，ブロックB内の命令が，実行頻度の高いパス上の命令のスケジューリングを妨げることはない．

二つめの特徴は，実行頻度に偏りがない場合も適度に効率良いスケジューリングができることである．なぜなら，複数のパスが領域に含まれるため，複数のパスの命令をバランスよくスケジューリングできるからである．例えば，図3の場合，異なったパス上のブロックD，Eが選択され，両ブロックの命令が同時にスケジュールされる．そのため，ACDF，ACEFの両方のパスが適度に最適化されたコードが生成でき，実行時に，ACDFのパスが実行されても，ACEFのパスが実行されても，高い性能が得られる．

もう一つの特徴は，領域内にループが存在すれば，最内ループ[9]，最内ループでないループ（これを，以下，外側ループと呼ぶ）の区別なくスケジューリングが可能であることである．例えば，図4(a)に領域形成前のCFG，図4(b)に領域形成後のCFGを表す．図4(a)において，ループADは外側ループである．しかし，領域形成でブロックD′が生成されることによって，領域内ではループAD′は，最内ループとみなせる．そのため，ソフトウェアパイプライニングの適用が可能となる．仮に，A，D′のパスが頻繁に実行されるとすると，性能は飛躍的に向上する．

3.2　プレディケートコード

領域の形成後，領域内の命令をストレートラインコードに変換する．具体的には，命令の制御依存情報であるプレディケートを命令に付加する．

また，プレディケーティングでは，分岐命令は，領域外への分岐，若しくは，ループバックの場合のみ必要となる．変換前のジャンプ命令は，その命令が領域内のブロックへのジャンプであれば消去される．また，変換前の分岐命令は，分岐条件を保持するCCRのセット命令と，二つのジャンプ命令に分解される．分解後のジャンプ命令が領域内のブロックへのジャンプであれば消去される．

例を使って変換方法を説明する．図5(a)に，図2

の例で形成した領域と各ブロック内の命令を示す．まず，基本ブロックA内の分岐命令以外の命令は，必ず実行されるのでalwというプレディケートを付ける．分岐命令i3は，CCRの第1エントリc0のセット命令i3とブロックBへのジャンプ命令i3′とブロックDへのジャンプ命令i3″に分解する．ブロックBへは，c0が偽のときに分岐するので，命令i3′には，!c0というプレディケートを付ける．また，ブロックDは，領域内のブロックなので，ジャンプ命令i3″は消去する．次に，基本ブロックD，E′内の命令にプレディケートを付ける．D，E′内の命令は，c0が真のときに実行されるので，プレディケートはc0となる．分岐命令i8は，CCRの第2エントリc1のセット命令i8と，ループからでるためのジャンプ命令i8′と，ループバックのためのジャンプ命令i8″に分解される．c1が偽のときループが成立するので，i8′には，c0&c1，i8″には，c0&!c1というプレディケートを付ける．図5(b)に変換後のコードを示す．変換後のコードをプレディケートコードと呼ぶ．

3.3 制約グラフの作成

作成したプレディケートコードのデータ依存を計算し，スケジューリングの制約を表すグラフ（制約グラフ[10]）を作成する．2.で述べたように，プレディケーティングでは，投機的実行結果をプレディケート付きでシャドウ構造に書き込むことによって制御依存を解消できる．この機能の適用可能範囲は，領域内と領域の次の基本ブロックの命令である．従って，ソフトウェアパイプライニングにおいては，現イタレーションの命令を投機的実行する場合と，次のイタレーションのループの入り口のブロックの命令を現イタレーションの命令と同時実行する場合において，制御依存がない．従って，CCRの定義とこれらの命令のプレディケートによる参照の間には依存関係はなく，制約エッジは付加しない．次のイタレーションのループの入り口のブロック以降のブロックの命令を，現イタレーションの命令と同時実行する場合は，プレディケーティングの機能が使用できないので制御依存が生じる．しかし，これに対しては，レジスタリネーミングによって解消するものとし，この場合もCCRの定義とプレディケートによる参照の間に制約エッジを付加しない．

また，領域を出る前に，領域のヘッダからその出口までのパス上の命令は，実行されるようスケジューリングされなければならない．このために，制約グラフ

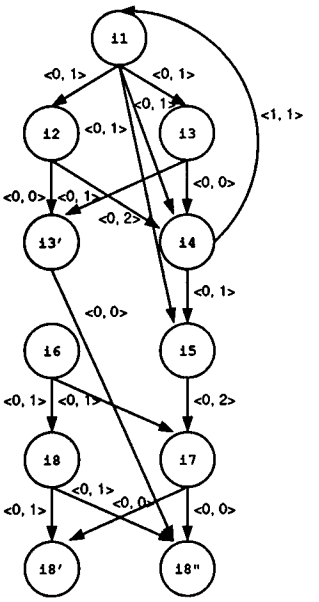


図6 制約グラフ
Fig. 6 Constraint graph.

において，リーフの命令より，その命令が属するブロックから制御が到達する領域の出口のブロックにあるジャンプ命令に対して，重み0のエッジを加える．

図6は図5(b)のプレディケートコードの制約グラフである．本例では，グラフの見やすさのため，真の依存関係にある命令間において生じるイタレーション間の逆依存のエッジの表記は省略してある．各エッジは，〈最小イタレーション間隔，遅延〉のラベル付けをする．遅延とは，エッジの両端のノード間で，依存関係を満たすために必要な最小サイクル数であり，最小イタレーション間隔とは，エッジの両端のノード間で，依存関係が生じるまでのイタレーションの回数を表す．例えば，ノードuからノードvへのエッジのラベルが$\langle p, d\rangle$の場合，pイタレーション後のノードvは，ノードuの実行が開始されてからdサイクル後でなければ実行を開始できない．

例えば，命令i2は，同じイタレーションの命令i1の結果を参照する．よって，i1からi2へのエッジには，〈0, 1〉のラベルが付く．また，命令i1は，前のイタレーションの命令i4の結果を参照する．よって，i4からi1へのエッジには，〈1, 1〉のラベルが付く(注2)．また，ジャンプ命令i3′は，命令i2の実行と

(注2)：i1，i4の命令のレイテンシは1．

同じサイクルかそれ以降のサイクルで，かつ，c0に書込みを行う命令i3の実行後に，実行されなければならない．よって，命令i2から命令i3′へのエッジには，〈0, 0〉のラベルが付き，命令i3から命令i3′へのエッジには，〈0, 1〉のラベルが付く．

3.4 モジュロスケジューリング

作成した制約グラフに対して，モジュロスケジューリングを適用する．一般に，モジュロスケジューリングは，

（1） ソフトウェアパイプライニングをするための開始間隔の下限と上限を求め，

（2） 開始間隔を(1)で求めた下限とし，すべての命令が資源制約，データ依存制約を満たすようにスケジュールを行う．もし，スケジュールができなければ，開始間隔を増やし，スケジュールを繰り返す．

という手順で行う．

本節では，本アルゴリズムにおけるモジュロスケジューリングの概要を示す．

（1） 開始間隔の下限，上限の決定

開始間隔（II：Initiation Interval）の下限 S_{min} は，1イタレーションの実行に最小限必要な資源数より決定される最小開始間隔（S_R）と，イタレーション間の依存関係より決定される最小開始間隔（S_D）の最大値で決定される．

本アルゴリズムでは，S_R は，領域内のすべての命令を考慮し，S_D は，異なるイタレーションに属する同一命令間の満たすべき条件とした．命令 v が要求する資源 k の数を $\rho_v(k)$，資源 k の数を R_k とすると，開始間隔の下限 S_{min} は，

$$S_{min}=\max(S_R, S_D)$$

となる．但し，

$$S_R=\max_{k\in R}\{\lceil(\sum_{v\in V}\rho_v(k))/R_k\rceil\}$$

$$S_D=\max_{u\in V}\{\lceil d(u,u)/p(u,u)\rceil\}$$

である．ここで，V は，領域内のすべての命令の集合，R はすべての資源の集合とする．

開始間隔の上限 S_{max} は，領域に対してリストスケジューリングを行って得られるサイクル数とした．S_{max} は，パイプライン化しない場合の最小の開始間隔であり，S_{max} より大きな開始間隔でパイプライン化することは，意味がないためである．

（2） 命令のスケジューリング

開始間隔の初期値を $s=S_{min}$ として，$s<S_{max}$ の範囲で s を1ずつ増やし，スケジュールが成功するまで，モジュロスケジューリングを行う．

命令のスケジューリングは，依存制約を満たす，最も高い優先度をもつ命令からスケジュールを行う．本アルゴリズムでは，優先度を2段階で評価する．第1段階では，制約グラフにおけるノードの高さ（リーフへの最大パス長）を評価する．もしも高さが等しいノードが複数あった場合は，第2段階として，スケジュール可能なサイクルの範囲内で命令が利用できる資源の数を評価する．

従来は，スケジューリングの優先度として，スケジュール可能な範囲のみを用いていた[1],[3]．本アルゴリズムで2段階評価を行った理由を以下で説明する．非数値計算応用では，ループの繰返し回数が動的に決まる場合が多い．ループの繰返し回数が動的に決まる場合，後続のイタレーションの命令は投機的に実行することになる．この場合，1イタレーションの実行に必要なサイクル数が長ければ，多くのイタレーションを並列に実行することになり，投機的に実行される命令数が増加する．これを可能にするには，複数のシャドウ構造を用意するか，レジスタリネーミングを行う必要がある．前者はハードウェア量を増加させ，後者は命令数と必要レジスタ数を増加させる．第1段階で制約グラフにおけるノードの高さを評価した理由は，1イタレーションの実行に必要なサイクル数を短くし，投機的に実行する命令数を少なくするためである．また，第2段階でスケジュール可能な範囲でなく，資源の空き数を評価した理由は，資源制約の厳しい命令を優先的にスケジュールするためである．一般に，メモリユニットは，ALUに比べて高価であるなどの理由により，命令の種類によって利用できる資源の数が異なるマシン（非均質マシン）が多く実現されている．そのため，最適なスケジュール結果が得られる可能性を高くするためには，利用できる資源数が少ない，資源制約の厳しい命令を優先的にスケジュールすることが必要であると考えた．

以下に，ある命令 u の利用可能な資源 k の空き数 $e_k(u)$ の求め方を説明する．$e_k(u)$ は，スケジューリング制約によって得られる命令 u のスケジュール可能な範囲内の各サイクルにおける，既にスケジュールされている命令によって占有されていない資源 k の数の和とする．

まず，命令 u のスケジュール可能な範囲を求める．スケジュール可能な範囲は，基本的には，Lamのconnected algorithm[1]を使用して求める．以下に，

cycle mod *II*	slot1	slot2	slot3	slot4
1	i4	i3		
2	i6	i5	*i1*	i3'
3	i8	*i2*		
4	i7	i8'	i8"	

図 7　モジュロ資源予約テーブル
Fig. 7　Modulo resource reservation table.

簡単に説明する．

命令 v がサイクル $cycle(v)$ にスケジュールされた場合，制約グラフにおいて v からのパスをもつ命令 u のスケジュール可能な範囲の上限 $Ub(u)$，下限 $Lb(u)$ は，以下の式によって再定義される．

$$Ub(u)=\min(Ub(u), cycle(v)-d(u,v)+s*p(u,v)) \quad \text{for } u \in P2$$

$$Lb(u)=\max(Lb(u), cycle(v)+d(v,u)-s*p(v,u)) \quad \text{for } u \in P1$$

但し，$P1$ は，制約グラフにおいて v からのパスをもつ命令の集合，$P2$ は，v へのパスをもつ命令の集合とする．また，$Ub(u)$，$Lb(u)$ の初期値は，各々 ∞，0 とする．

次に，既にスケジュールされている命令が占有している資源の数を求める．ソフトウェアパイプライニングでは開始間隔 Π ごとにイタレーションが開始される．仮に $\Pi=s$ とした場合，サイクル c にスケジュールされた命令は，s サイクルごとに資源を占有することになる．よって，サイクル c にスケジュールされた命令 v が資源 k を占有する場合，任意のサイクル x で命令 v が要求する資源 k の数は，以下の式で表せる．

$\rho_v(x,k)=1,\ x=(c \bmod s)+t*s \quad t=0,1,2,\cdots$

$\rho_v(x,k)=0$, 上記以外

よって，$e_k(u)$ は，以下の式で求められる．

$$e_k(u)=\sum_{Lb(u)\le i\le Ub(u)}(R_k-\sum_{v\in sched}\rho_v(i,k))$$

但し，*Sched* は，スケジュール済みの命令の集合である．

命令のスケジューリングは，依存制約を満たす最も遅いサイクルにスケジュールしている．これは，レジスタのライフタイムを短くし，レジスタ競合を少なくするためである．

図 6 の制約グラフをスケジュールしたときのモジュロ資源予約テーブルを図 7 に示す．モジュロ資源予約テーブルは，開始間隔数の行と資源の列で構成され，スケジューリングにおける資源の競合を管理する．スロット i のサイクル t に命令をスケジュールするためには，モジュロ資源予約テーブルの $t \bmod \Pi$ 行の *slot* i 列で資源競合が生じない必要がある．この例は，開始間隔 Π を 4，四つの ALU(slot 1-4)，四つの分岐ユニット(slot 1-4)，二つのロードユニット(slot 1-2)，一つのストアユニット（shot 1）があるマシンモデルを想定した場合のモジュロ資源予約テーブルである．また，この例では，次のイタレーションの命令を斜体で示してある．

	slot1	slot2	slot3	slot4
1	c0 ? i4	alw ? i3		
2	c0 ? i6	c0 ? i5	*c0&!c1 ? i1*	alw ? i3'
3	c0 ? i8	*c0&!c1 ? i2*		
4	c0 ? i7	c0&c1 ? i8'	c0&!c1 ? i8"	

図 8　定常状態の命令コード
Fig. 8　Code in a steady state.

3.5　命令コード生成

モジュロ資源予約テーブルから，命令コードを生成する．図 8 に，図 7 のモジュロ資源予約テーブルから生成した，定常状態の命令コードを示す．同じ行にある命令が同時に実行される命令である．空欄には nop 命令が入るとする．簡単化のため，各命令のプレディケートと命令番号のみ示す．

次イタレーションの命令には，ループが成立したとき真となる条件とプレディケートコード作成時に付加されたプレディケートの論理積を新たなプレディケートとして付加する．例えば，次のイタレーションの命令 i1 と i2 には，コード生成時のプレディケート alw とループが成立したとき真となる条件 c0 & !c1 の論理積が，新たなプレディケートとなる．このプレディケートにより，ループが繰り返されなかった場合，これらの命令は，ハードウェアで無効化される．

4.　他の研究との比較

この章では，条件分岐を含むループのソフトウェアパイプライニングに関する研究について簡単に説明し，本アルゴリズムと比較を行う．また，パスの選択を行うスケジューリング技術で，プレディケート実行を支援するハードウェアをもつマシンをターゲットにした研究についても簡単に触れておく．

条件分岐を含むループのソフトウェアパイプライニ

ングに関する研究には，モジュロスケジューリングを利用するものとして，HR [1]，PE [2]，EMS [3] などがある．また，モジュロスケジューリングを利用しないものとして，パイプラインスケジューリング (PS) [11]，などがある．

PS は，命令を 1 個ずつバックエッジを越えて前のイタレーションに移動させてパイプライン化を行う．しかし，モジュロスケジューリングと異なり，最適なスケジュール結果を得るためには，後続のイタレーションのどの命令をどこまで移動させればよいかを決める手段がないという欠点がある．一方，モジュロスケジューリングは，ループ内のすべての命令の制御を考慮してスケジューリングを行えるため，最適なスケジュール結果が得られる可能性が高い．

HR は，if ブロック全体を一つの複合命令とみなし，モジュロスケジューリングを適用する．複合命令は，各パスを各々リストスケジューリングによってスケジュールしたときの実行時間の長い方のパスの長さをレイテンシとする．必要資源数は，二つのパスの論理和をとる．HR は，ハードウェアサポートが必要ないという利点がある．しかし，複合命令の必要資源数が複雑なパターンになり，最適な開始間隔を見つける障害になるという欠点と，実行時間の長い方のパスで定常状態のサイクル数が決まってしまうため，短い方のパスが最適にスケジュールされるとは限らず，短い方のパスが頻繁に実行される場合のペナルティが大きいという欠点がある．

PE は，プレディケートを評価することにより命令を有効化または，無効化する．PE では，ループ内のすべてのパス上の命令を同時にスケジュールする．命令の有効，無効化は実行時に行われるので，すべての命令が資源を占有するとして，スケジュールを行わなければならない．そのため，異なったパス上の命令間で，資源競合が起こりやすく，各パスが最適にスケジュールされないという欠点がある．そのため，分岐に偏りがある場合には，ほとんど実行されないパス上の命令の影響により，性能が低下してしまう．

EMS は，分岐条件ごとに異なるコードを用意し，どのコードを実行するかを実行時に選択する．スケジュールはループ内のすべてのパス上の命令を同時に行うが，PE とは異なり，パスごとにコードを生成するので，異なるパス上の命令は資源競合が生じないとする．そのため，不必要な資源競合が生じないという点で PE より優れている．しかし，PE と同様，ループ内のすべてのパス上の命令を同時にスケジュールするため，実行時間の長い方のパスで定常状態のサイクル数が決まってしまい，短い方のパスが最適にスケジュールされないという問題点は解決されていない．

本アルゴリズムは，実行頻度の高いパスを選んで，そのパス上の命令をモジュロスケジューリングによってスケジュールする．実行頻度の高いパスを選択してスケジュールするため，実行頻度の低いパス上の命令の影響を受けずにスケジュールでき，実行頻度の高いパスが最適にスケジュールできる．更に，分岐方向に偏りのない場合は，複数のパスが選択され，複数のパス上の命令をバランスよくスケジュールできるため，どのパスが実行されても適度に最適化されたコードを生成できる．

パスの選択を行うスケジューリング技術で，プレディケート実行を支援するハードウェアをもつマシンをターゲットにした研究の一つに，ハイパブロックスケジューリング [12] がある．ハイパブロックスケジューリングは，本アルゴリズムと同様，複数のパスを選択し，スケジューリングの対象とする．ループに対しては，ループアンローリングを適用している．しかし，ループアンローリングには，コード量が大きくなり，メモリシステムの性能を低下させるという欠点がある．一方，ソフトウェアパイプライニングは，コード量の増加の少ないため，この欠点を緩和できる．しかし，この論文ではコード量の増加の少ないソフトウェアパイプライニングの適用法については述べられていない．

5. 評　　価

5.1 評価項目

以下の項目について評価した．

(1) パス選択による効果

(2) プレディケーティングによる効果

(1)の評価では，本アルゴリズムとパス選択を行わない二つのアルゴリズム PE，EMS との性能の比較を行った．PE は，異なるパス上の命令も資源競合が生じると仮定し，EMS は，異なるパス上の命令は資源競合が生じないと仮定してスケジューリングを行った．EMS では，コード生成を実際に行っていないため，各コードへの分岐命令を入れていない．よって，性能評価結果は，文献 [3] よりは少し高い性能を示していると考えられる．この項目の評価では，ハードウェアは，プレディケーティングを備える VLIW とし

表 1　マシンモデル
Table 1　Machine models.

最大命令発行数	ALU数	分岐ユニット数	ロードユニット数	ストアユニット数
2	2	2	2	1
4	4	4	2	1
8	8	8	8	8

た．

（2）の評価では，本手法，PE，EMS の三つに対し，ハードウェアにプレディケーティングによる投機的実行の支援があるかどうかで，比較を行った．但し，PE の場合は，プレディケート実行機構が必要なので，これは備えると仮定した．

5.2　ベンチマーク

ベンチマークに用いたプログラムは，espresso，eqntott，li，compress の 4 個の SPEC ベンチマーク，awk，grep，nroff の 3 個の UNIX ユーティリティ，スタンフォードベンチマークである．

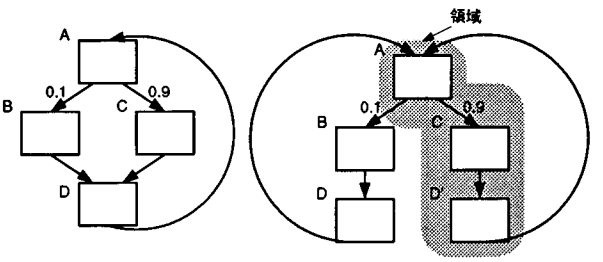
5.3　マシンモデル

表 1 に示す最大命令発行数が 2，4，8 の三つの VLIW について評価した．最大発行命令数が 2 のモデルは，資源の少ない非均質モデルの性能を評価するため，最大発行命令数が 8 のモデルは，十分に資源がある場合の評価を行うため，最大発行命令数が 4 のモデルは，平均的な非均質モデルの評価を行うために設定した．

命令のレイテンシは，MIPS R 3000 の命令のレイテンシとほぼ等しくロード命令が 2 サイクルであり，その他の命令はすべて 1 サイクルとした．プレディケーティングを備える場合，CCR は 4 個のエントリをもつとし，最大 4 分岐を越える投機的実行が可能とした．

5.4　評価方法

評価はパイプライン化の性能のみに注目するため，パイプライン化が行われたループのみを対象とした．性能向上率は，パイプライン化しなかったときにループを実行するのに必要なサイクル数の和を，各アルゴリズムで得られたループの定常状態のサイクル数の和で割ったものとした．評価にループの実行回数の項目を含めなかったのは，ループによって繰返し回数に偏りがあるため，一部の繰返し回数の多いループの特性によって評価結果が左右されるのを避けるためである．



(a)　領域形成前　　(b)　領域形成後

図 9　イタレーションの実行サイクル数
Fig. 9　Execution cycles for an iteration.

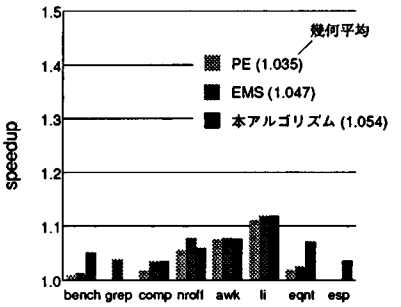
PE，EMS は，ループ内のすべてのパスを同時にスケジュールするので，実行時にどのパスを通っても定常状態のサイクル数で実行される．しかし，本アルゴリズムは，分岐確率によってパスを選択するため，すべてのパスがパイプライン化されるわけではない．そのため，実行時にどのパスを通るかでサイクル数が異なる．よって，本アルゴリズムでは，各パスの実行に必要なサイクル数の分岐確率による相加平均をループの 1 イタレーションを実行するのに必要なサイクル数として，定常状態のサイクル数の代わりに用いている．ループの 1 イタレーションの実行に必要なサイクル数 cyc_{loop} を以下の式で表す．

$$cyc_{loop}=\Pi^*(1-p_{out})+cyc_{out}{}^*p_{out}$$

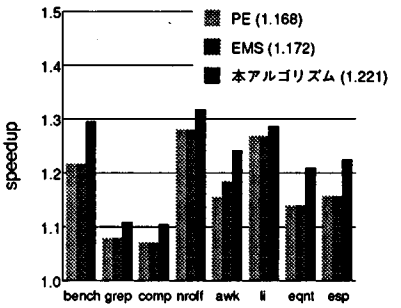
但し，Π をパイプライン化されたパスの定常状態，p_{out}，cyc_{out} はそれぞれ，パイプライン化されなかったパスへの分岐確率，そのパスを実行するために必要なサイクル数とする．

例えば，図 9（a）を領域形成前の CFG，図 9（b）を領域形成後の CFG，ハッチングした部分を領域とした場合，ABCD で構成されるループの 1 イタレーションを実行するのに必要なサイクル数は，パイプライン化されたパスの定常状態の開始間隔を $\Pi_{ACD'}$，ブロック A から B への分岐確率を 0.1，ブロック ABD の実行に必要なサイクル数を cyc_{ABD} とすると，$\Pi_{ACD'}$ *0.9＋cyc_{ABD}*0.1 となる．

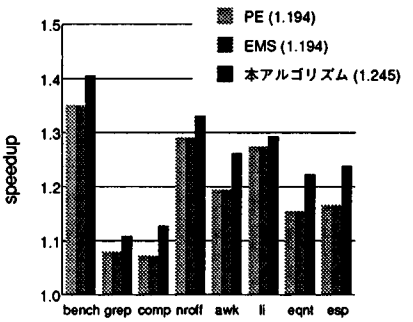
また，本アルゴリズムではパイプライン化の対象となるが，PE，EMS では対象とならない外側ループがある．このループについては，PE，EMS では，そのパスをリストスケジューリングでスケジュールしたときのサイクル数を，定常状態のサイクル数の代わりとして用いている．



(a) 2命令発行



(b) 4命令発行



(c) 8命令発行

図 10 ハードウェア資源の異なるモデルに対する性能評価結果(a) 2命令発行 (b) 4命令発行 (c) 8命令発行
Fig. 10 Speedup with various amount of resources. (a) 2 issue (b) 4 issue (c) 8 ssue

5.5 評価結果

図10に，最大命令発行数2，4，8のモデルにおける性能向上率を示す．凡例の（ ）内は，性能向上率の幾何平均である．

最大命令発行数2のマシンモデルは，少ない資源のモデルを想定したものである．資源が少ない場合は，複数のパスを同時にスケジュールするPEや，本アルゴリズムは資源競合が起こりやすい．grep，nroffにおいてEMSの性能がよいのは，このためである．しかし，本アルゴリズムは，実行頻度によって選択的にパスを選択するため，必ずしも複数のパスが選択されるわけではない．他のアルゴリズムよりも本アルゴリズムの性能が良いのは，複数のパスを選択したことによる性能低下を，実行頻度の高いパスを最適にスケジュールできたことと，外側ループがパイプライン化できたことによる性能向上が上回った結果である．

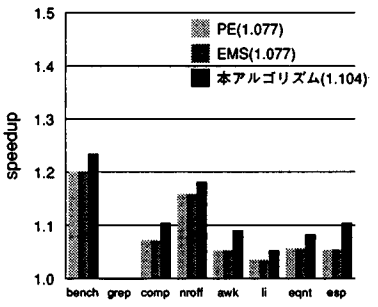


図 11 プレディケーティングを搭載しないマシンでの性能向上
Fig. 11 Speedup without predicating.

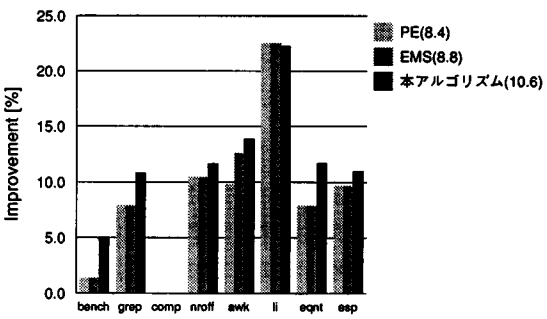


図 12 プレディケーティングの性能への寄与
Fig. 12 Improvement with predicating.

最大命令発行数4，8のマシンモデルの場合は，最大命令発行数が2のモデルよりも資源競合が緩和される．すべてのプログラムにおいて本アルゴリズムの性能がよいのは，複数パスの選択による性能低下が最大命令発行数2のモデルよりも少なくなり，実行頻度の高いパスを最適にスケジュールできたことと，外側ループがスケジュールできたことの効果が現れたためである．最大命令発行数が増加するほど，資源競合は緩和される．これは，最大命令発行数8の場合に，PEとEMSの性能が等しくなっていることからもわかる．資源競合が緩和されるため，最大命令発行数が増加するほど本アルゴリズムの効果が顕著に表れてい

る．

図11に，最大命令発行数4でプレディケーティングを搭載しないマシンでの性能向上率を示す．更に，図12に，プレディケーティングの性能への寄与を示す．図12の値は，プレディケーティングを搭載しないマシンの性能（図11）を基準とした．搭載したマシンの性能向上（図10(b)）の割合(%)である．

図10(b)，図11よりわかるように，プレディケーティングを搭載したマシンでの性能は，本アルゴリズムを適用することにより，PE，EMSに比べて平均でそれぞれ0.053(1.221−1.168)，0.049(1.221−1.172)向上するのに対して，プレディケーティングを搭載しない場合は，0.027(1.104−1.077)しか向上しない．また，プレディケーティングの性能への寄与は，平均で，PEで8.4%，EMSで8.8%であるのに対して，本アルゴリズムでは10.6%と最も大きい．bench，grep，eqntottでは特に顕著な違いが見られる．このような違いは，本アルゴリズムでは，パスを選択しそれを最適化するという手法をとっていることによるものである．パス選択により，制御依存以外の制約（資源制約と実行頻度の低いパスにより生じる制約）が緩和されるので，制御依存制約の緩和が性能に大きく影響する．すなわち，選択したパス内での制御依存が除かれ，パス内で自由に投機的命令移動できることが，他のアルゴリズムに比べて，非常に重要となる．このため，本アルゴリズムは，プレディケーティングを搭載したマシンにおいて特に効果を発揮することができる．

6. むすび

本論文では，プレディケーティングを備えるマシンにおいて，実行頻度の高いパスを選んで，スケジュールを行うソフトウェアパイプライニングのためのアルゴリズムを提案した．本アルゴリズムは，ループの中の実行頻度の高いパスを選択してスケジュールを行うため，実行頻度の低いパス上の命令によって，資源が無駄に消費されることがなく，実行頻度の高いパスの最適化が妨げられることがない．分岐に偏りがない場合には，複数の命令をバランスよくスケジュールできるため，どのパスが実行されても高い性能が達成できる．また，更に，従来のパイプライン化が困難であった外側ループに対しても，パイプライン化できる可能性を与えることができる．

従来の技術との性能を比較した結果，最大命令発行数が少ない場合も，多い場合も良い性能が得られていることが確認できた．また，プレディケーティングを搭載することによる性能向上は，本アルゴリズムを適用する場合の方が，従来のアルゴリズムを適用する場合よりも大きく，本アルゴリズムとプレディケーティングを組み合わせることによって，高い性能が達成できることがわかった．

今後の課題としては，逆依存関係を解決するためのレジスタリネーミングや，ループアンローリングを行うことにより，より性能を向上する工夫を行っていく予定である．また，非数値計算応用のプログラムにおいて，ループの性能向上を得るためには，何が必要であるかを検討していきたいと思っている．

文　献

[1] M. S. Lam, "Software pipelining: An effective scheduling technique for VLIW machines," in Proc. SIGPRAN'88 Conf. Program. Lang. Des. Implement, pp. 318-328, June 1988.

[2] J. C. Dehnert, P. Y. Hsu, and J. P. Bratt, "Overlapped loop support in the Cydra 5," in Proc. 3rd Inter. Conf. Architectural Support for Programming Languages and Operating Systems, pp. 26-38, April 1989.

[3] N. J. Warter, G. E. Haab, and J. W. Bockhaus, "Enhanced module scheduling for loops with conditional branches," in Proc. MICRO-25, pp. 170-179, Dec. 1992.

[4] 山下義行, 中田育男, "ループ中に条件分岐を含む場合の最適なソフトウェアパイプライニング," 並列処理シンポジウム JSPP'94 論文集, pp. 17-24, May 1994.

[5] H. Ando, C. Nakanishi, T. Hara, and M. Nakaya, "Unconstrained speculative execution with predicated state buffering," in Proc. 22nd Annu. Inter. Symp. on Computer Architecture, pp. 126-137, June 1995.

[6] J. A. Fisher, "Trace scheduling: a technique for global microcode compaction," IEEE Transactions on Computer, vol. C-30, no. 7, pp. 478-490, July 1981.

[7] M. D. Smith, M. A. Horowitz, and M. S. Lam, "Efficient superscalar performance through boosting," in Proc. Fifth Int. Conf. on Architectural Support for Programming Languages and Operating Systems, pp. 248-259, Oct. 1992.

[8] 安藤秀樹, 中西知嘉子, 原　哲也, 中屋雅夫, "非数値計算応用におけるプレディケート実行向け命令スケジューリング," 並列処理シンポジウム JSPP'96 論文集, pp. 65-72, June 1996.

[9] A. V. Aho, R. Sethi, and J. D. Ullman," Compilers: principles, techniques, and tools," Addison-Weslay Publishing Company, Reading, Massachusetts, 1986.

[10] T. Gross, "Code optimization of pipeline constraints," Technical Report No. 83-255, Stanford University, Stanford, CA, 94305, Dec. 1983.

[11] K. Ebcioglu, "A compilation technique for software

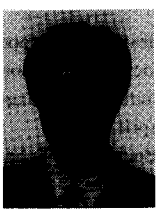
pipelining of loops with conditional jumps," in Proc. 20th Annu. Workshop on Microprogram, pp. 69-79, 1987.

[12] S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann, "Effective compiler support for predicated execution using the hyperblock," in Proc. MICRO-25, pp. 45-54, Dec. 1992.

(平成8年4月5日受付，12月26日再受付)

中西知嘉子

1988阪大・基礎工・情報卒．1988三菱電機(株)LSI研究所に入社．第5世代コンピュータプロジェクトの推論マシン用プロセッサの設計に従事．計算機アーキテクチャ，コンパイラの研究に従事．1997より信号処理プロセッサの開発に従事．

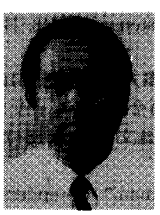


安藤　秀樹　(正員)

1981阪大・工・電子卒．1983同大大学院修士課程了．京都大学工学博士．1983三菱電機(株)LSI研究所に入社．ISDN用ディジタル信号処理LSI，第5世代コンピュータプロジェクトの推論マシン用プロセッサの設計に従事．1991 Stanford大学客員研究員．1997名古屋大学大学院工学研究科電子情報学専攻・講師．計算機アーキテクチャ，コンパイラの研究に従事．

原　哲也　(正員)

1989九大・工・情報卒．1991同大大学院総合理工学研究科情報システム学専攻修士課程了．同年三菱電機(株)LSI研究所に入社し，細粒度並列処理アーキテクチャの研究に従事．1996より信号処理プロセッサの開発に従事．現在，同社マイコン・ASIC事業統括部に所属．



中屋　雅夫　(正員)

1974早大・理工・電子通信卒．1976同大大学院修士課程了．1988工博（早稲田大学）．1976三菱電機(株)入社．以来，高速MOSゲートアレイ，ECLゲートアレイ，MOS A/D，D/Aコンバータ，三次元回路素子，通信用LSI，ISDN用LSIの開発を経て，現在，並列処理プロセッサ，ATM-LAN用LSI等のシステムLSIの研究開発に従事．1993年度本会論文賞受賞．現在，三菱電機(株)システムLSI開発部．IEEE，情報処理学会各会員．