計算機アーキテクチャ 120－4
（1996. 10. 31）

マルチスーパースカラパイプラインによるブロック並列実行方式

朝生良教[†] 高田 滋[†] 森 俊一郎[†]
清水健士[†] 林 達也[†]

スーパースカラプロセッサの性能向上のための手法として動的分岐予測と併せた投機的実行が実現されている．この場合分岐予測の確度を高める事が重要であるが，とりわけ整数系のプログラムでは容易ではなく早晩限界を迎えると考える．これを打開する方式としてブロック並列実行方式を提案している．これは分岐により区切られる基本ブロックどうしを複数のスーパースカラパイプラインにより同時並列に実行するものである．分岐を越える並列実行が可能であり，また分岐予測の必要もない．本稿では簡単にはなり得ぬブロック並列実行方式の，より現実的な構成とその動的スケジューリング機構に関して提案を行なう．

A Method of Block Parallel Execution with Multi Superscalar Pipelines

Yoshinori Asoh,[†] Shigeru Takada,[†] Shunichiro Mori,[†]
Kenji Shimizu[†] and Tatsuya Hayashi[†]

Speculative executions based on the dynamic branch prediction are widely realized to improve the performance of superscalar processors. It is not easy to increase the correctness of branch prediction much higher than now, especially in the integer intensive programs. We have proposed formerly a method of block parallel execution to overcome this situation. Revised version of this method is described in this paper, in which two adjacent basic blocks are executed concurrently using two superscalar pipelines.

1. はじめに

バイナリ互換性を維持するとの点に依拠しスーパースカラプロセッサの研究を行なっている．スーパースカラパイプラインを採用した最近のプロセッサは分岐ペナルティを軽減する手法として，分岐先バッファ(BTB) 等を用いた分岐予測による命令フェッチの先行制御から更に実行の先行制御 (投機的実行) までも行なっており効果を発揮していると思われる．しかし2つの分岐方向の予断による二者択一である以上，選択を誤ればやり直す事は避けられない．また分岐予測の確度には限界があり，また整数系のプログラムでは予測外れも多くそのための損失が深刻な性能向上の阻害要因となっていると考える．この問題を解消し更なる命令レベルの並列性を抽出する手法として本方式は位置付けられる．それはスーパースカラパイプラインによる基本ブロックの実行と同時並列に別のスーパースカラパイプラインでその分岐先の基本ブロックも実行するというものである．故にブロック並列実行方式と命名し研究を行なっている[1)2)]．ブロック並列実行は分岐命令により開始され，また開始毎にスーパースカラパイプラインの同期を取るものであるが，理想的にはスーパースカラパイプラインの倍の性能を発揮し得る．

以後はまず基本となるブロック並列実行方式について概説し (第2節)，現実的な構成とその動的スケジューリング機構を提示し (第3節)，問題点と今後の展望を述べる (第4節)．

† 名古屋工業大学電気情報工学科
Department of Electrical and Computer Engineering, Nagoya Institute of Technology

2. ブロック並列実行方式

2.1 その概念

2.1.1 ブロックの概念

ノイマン型コンピュータに基づくプログラムとは演算，転送，分岐の3種からなる命令が逐次順序付けされ，プログラムはその流れを基本ブロックを単位に区切る事ができる．なお以後はこの基本ブロックをブロックと略す．ブロックとは分岐命令を含まない一連の命令群であり，その入口と出口では命令実行の流れ(命令流) が切り替わり得る．ブロックの出口には必ず分岐命令が存在するが，入口には必ずしも存在しない．(条件) 分岐命令は成立と成立と不成立の2つの分岐方

向を持つ．以上のブロックの概念を例示すると図1のようになる．

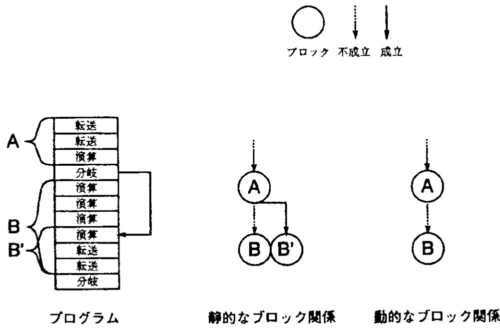


図1 ブロックの概念

2.1.2 ブロック並列実行の概念

ブロック並列実行とは，あるブロックとその分岐先のブロックをそれぞれ異なるスーパースカラパイプラインで同時並列に実行するものである．この概念を図示すると図2のようになる．

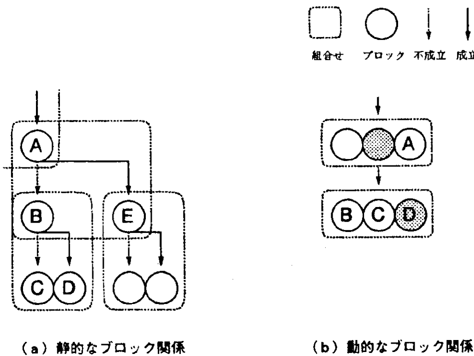


図2 ブロック並列実行の概念

図2(a) にコンパイル時に把握されるブロック関係の例を示す．破線で囲まれる3つのブロックが並列実行の組合せであり，コンパイラはあらゆる組合せを網羅する．次に図2(b) に実行時に定まるブロック関係の例を示す．なおこれは図2(a) の例を並列実行したものである．ここでBCDの組合せに注目すると，Bを当該実行ブロック，Bの不成立の分岐先であるC，成立の分岐先であるDを投機的実行ブロックと呼ぶ事にする．また図でDが黒いのはBがCに分岐した結果Dの実行は無意味となった事を表す．つまり3つのブロックの内有効となるのは常に2つである．

当該実行ブロックに対して投機的実行ブロックの実行は後続すべきものである．即ち本来時系列性のあるブロックどうしを並列に実行する本方式では多くの問題を内包する．以下にそれらをまとめる．

- 多重化による多量のハードウェアを必要とし，また制御論理が複雑化する．
- 並列実行されるブロック間のデータ (フロー) 依存のためにデータハザードが頻発する．
- 分岐先ブロックを予めフェッチしていても分岐ペナルティの軽減が重要である．

更に，本質的に次のような問題がある．

当該実行ブロックと投機的実行ブロックを並列に実行する事は，即ちブロック間で命令フェッチから発行，実行，完了の全ての段階において完全に順外 (out-of-order) である事を意味する．よってブロック並列実行では投機的実行ブロックが当該実行ブロックとのデータ依存関係を知りようがないという問題が考えられる．具体的には投機的実行ブロックにおいて，ソースオペランドの当該実行ブロックへのデータ依存の有無と，更に依存がある場合はデータハザードの有無が判断不能という状態が起こる．これは依存情報の欠落によるハザードであり，情報ハザードと呼んでいる．情報ハザードの発生は投機的実行ブロックの全ての命令の発行を困難にし，またハードウェアのみでは容易に解消できない問題である．なお情報ハザードが完全に解消されるのは，当該実行ブロックの全命令の発行が完了する時である．

2.2 その実現機構

まず各ブロックを実行するスーパースカラパイプラインは4命令を順当 (in-order) に発行するものとし，それぞれは並列実行の組合せに固定的に対応する．また命令キャッシュは同時に命令フェッチできるようマルチポートとする．

2.2.1 実行開始方法

ブロック並列実行のため各ブロックの実行を各スーパースカラパイプラインに同時に開始させるため専用の分岐命令を用意する．この分岐命令はPC相対のオフセット3つを有する．そしてブロック当りこの分岐命令を2つ備えて成立と不成立のそれぞれの場合の並列実行を開始させるものとする．

2.2.2 分岐ペナルティ軽減機構

投機的実行の有効の是非と新たな並列実行開始のため分岐方向の早期決定と分岐先アドレスの早期生成が必要である．よってコンパイラによる分岐命令の前方移動とその際に埋め込まれるカウントを利用したハードウェアによる適正かつ早期の分岐を実現する事で対処するものとしている．

2.2.3 動的スケジューリング機構

まず情報ハザードへの対処のためのアーキテクチャ上の支援を説明する．投機的実行ブロックの命令の発行に際し，当該実行ブロックの全命令の発行終了時点の情報が得られないために発生する情報ハザードは，動的なブロック間のデータ依存関係の未知に起因しハードウェアのみでは対処し難い．よってソフトウェアの支援により依存情報を与えるものとする．

与えるべき情報はフラグの形を取り，投機的実行ブロックの命令に関しては依存し得るソースオペランドを，当該実行ブロックの命令に関しては依存され得るターゲットオペランドをそれぞれ特定するものとする．このために全命令語に3(ソース×2，ターゲット×1)ビットのフラグ領域を確保する．なお，それぞれ依存ビット，提供ビットと呼ぶ事にする．これら依存情報は確実ではあり得ないため☆レジスタ操作に限り補う情報として，当該実行ブロックで変更されるターゲットオペランドを特定するフラグ列を考えている．これはアーキテクチャで規定されたレジスタ数分のフラグ列でレジスタ変更ビット列と呼び分岐命令に付加するものとしている．以上が情報ハザードに対処するために必要なソフトウェアの支援である．

次に動的スケジューリング機構について説明する．これにはデータハザードを解消し順外実行を許すTomasuloアルゴリズムを用いる．なぜならブロック並列実行ではデータハザードが頻発する事が予想されるからである．このためTomasuloアルゴリズムの，命令発行を滞らせずに待機させ，データの放送によりデータが揃い次第一斉に実行可能という特徴は非常に有用である．Tomasuloアルゴリズムの基本構成は，レジスタファイル(RF:resister file)に対応する管理情報を記憶するレジスタ状態(RC:resiter condition)，発行された命令を待機させる予約ステーション(RS:reservation station)とロードバッファ(LB:load buffer)，ストアバッファ(SB:store buffer)，および放送のための共通データバス(CDB:common data bus)から成る．全体の構成はこれらを3つ用意し共通データバスを結合したものとなる．投機的実行側のレジスタファイルとレジスタ状態は投機的実行結果のバッファリングを行い，並列実行の完了後に当該実行側に書き戻される．投機的実行ブロックでは自己生成したデータ以外はソースオペランドを当該実行側から参照する．

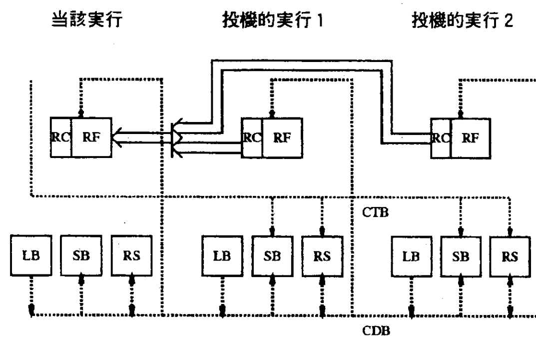


図3　基本構成

☆ 異なる当該実行ブロックが存在する場合は依存ビットが，メモリ操作の場合はどちらもが誤る可能性がある．

Tomasuloアルゴリズムではデータの生成遅延によるデータハザードを，データのタグ(機能ユニット番号)を利用して発行する事で回避するが，ブロック並列実行では投機的実行側の参照において当該実行側のタグの決定遅延という問題が発生する．これはソフトウェアにより与えられた依存情報で特定は可能だが回避は不能である．よって投機的実行側ではタグのタグ，つまり仮のタグを用いて発行し，タグの決定時に放送する事で回避するものとする．このタグの放送のために新たに共通タグバス(CTB:common tag bus)を設ける．なおタグの決定は必ずデータの生成よりも早いため放送が遅らされなければ，タグの獲得が遅れてデータの放送を取り落とす事はない．なお仮のタグとはソースオペランドのレジスタ番号である．この変更に伴って，投機的実行側の予約ステーションとストアバッファには仮のタグの記憶を示すフラグと仮のタグの記憶領域を必要とする．図3に以上の構成を示す．

3. 現実的構成

3.1 基本的な変更

現行方式は以下のような点において現実性を欠いている．

- 3つのスーパースカラパイプラインの内，常に1つが遊ぶため効率が悪い．
- ブロック当り3つのオフセットを持つ2組みの分岐命令を要しコード量が増大する．
- 依存情報であるレジスタ変更ビット列が大きくコード量が増大する．
- 常に分岐命令により並列実行を開始するため命令の先読みが無駄になる．

よってこれらを解消するため現実的な構成に変更する．具体的には図4に示すように並列実行の組合せを大幅に減らし，またレジスタ変更ビット列も扱わないものとする．これにより図5に例示するように分岐命令の語長が現実的な長さに納まり得る．これらの変更により性能は低下するが，それはループでなく不成立のブロックのブロック長が長い場合にほぼ限られるため深刻ではないと判断した．

3.2 レジスタ操作に関する動的スケジューリング機構

現行方式のレジスタ操作に関する動的スケジューリング機構は，各スーパースカラパイプラインにTomasuloアルゴリズムを適用して全体の統合のために拡張したものであった．これはオペランドの依存関係がブロック内に終始する場合は各々がTomasuloアルゴリズムに従い，異なるブロックに及ぶ場合にはそちらの状態を確認するために参照を必要とする．ただし異なるブロックとは投機的実行ブロックに対する当該実行ブロックであって，その逆は該当しない．つまり参照は一方通行である．この構成では動的スケジューリン

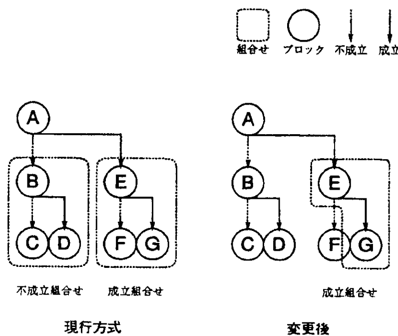
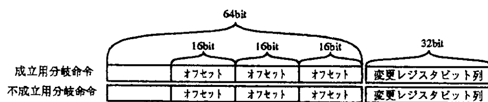
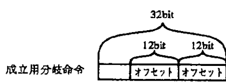


図4　並列実行の組合せ



（a）現行方式の分岐命令



（b）変更後の分岐命令

図5　分岐命令の語長

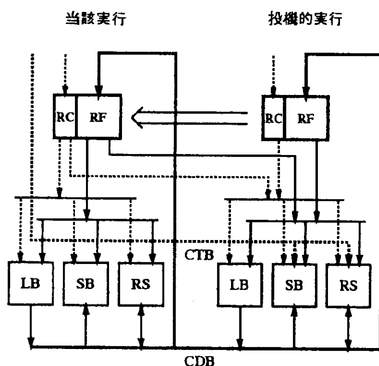


図6　現行方式の動的スケジューリング機構

グ機構における命令発行機構の発行ネットワークが本数分のスーパースカラ度(つまり2本×4命令発行で8)に等しい規模となってしまうと予想される．また当該実行側のレジスタ状態及びレジスタファイルの読み出しポート数も倍の規模になる．図6に現行方式の動的スケジューリング機構を示す．

現行方式ではコストが高くつき過ぎる．各スーパースカラパイプラインはその独立性が高くなければならない．極論すれば，単純にスーパースカラパイプラインを並べた程度の規模で済む事が理想となる．このためにスケジューリングのためのやり取りを命令発行単位ではなく，投機的実行側の結果書き戻しと併せてブロック並列実行を単位として一括で行うものとする．この相互書き戻しによりブロック並列実行開始時に投機的実行側は当該実行側と等しい情報とデータをレジスタ状態とレジスタファイルとに有する事になる．これで投機的実行側が参照を要するものは純粋に当該実行側で生成されるものに限られ，依存のない命令の当該実行側への参照を回避できる．表1にレジスタ状態並びにレジスタファイル間の相互書き戻し規則を示す．

	当該実行	投機的実行	相互書き戻し後
変更の有無	n	N	変更なし
	n	Y	Y
	y	N	y
	y	Y	Y

表1　相互書き戻し規則

回避できぬ参照とはソースオペランドに対応したタグかデータを取得するためであり，それらはいつ生成されるか不定である．ならばむしろ当該実行側が生成し次第提供する方が効率が良い．よって投機的実行側の参照から当該実行側の提供へと変更する．提供すべき内容はタグであり，当該実行側が命令発行時に決定されたタグを自己のみならず投機的実行側のレジスタ状態に書き通す．タグの書き通しを行なう必要があるのは当該実行ブロックの提供ビットの立っている命令である．また書き通しを行なうのはそのレジスタに投機的実行側が書き込みを行なっていない場合である．行なっていれば元々必要がないか，あるいはそのタグを必要とする命令は既に仮のタグを用いて発行済みなので問題はない．この書き込みの有無を判断するために投機的実行側のレジスタ状態にフラグを設ける．書き通しが行なわれた場合，投機的実行側は依存ビットの有無によらず問題なくタグを取得して良い．この書き通しの有無を判断するために同じく投機的実行側のレジスタ状態にフラグを設ける．書き通しのために考えられる問題はアクセス競合である．当該実行側の書き通しと投機的実行側の書き込みが衝突する場合は書き通しを捨てれば良いが，書き通しと読み出しの衝突では必ず書き通してから読み出されなければならない．なぜなら書き通しと同時に共通タグバスにタグが放送されているため，どちらも取り落とせば致命的な誤りを引き起こすからである．

以上の変更により低コストとなりながらも現行方式に劣らぬスケジューリングが可能である．図7に変更後の動的スケジューリング機構を示す．

3.3　メモリ操作の動的スケジューリング機構

これまでメモリ操作に関する動的スケジューリング機構は未だ具体的な決定を見なかった．なぜならメモリ操作では以下の点に注意を要したからである．

- アドレス検索のために連想的参照を必要とする．

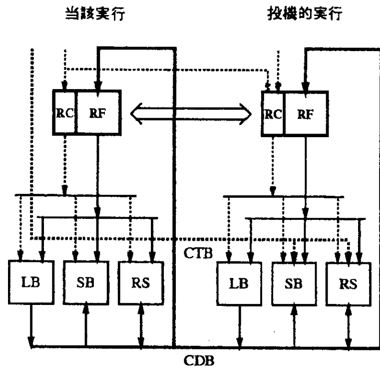


図7　変更後の動的スケジューリング機構

- 正確な割り込みの保証のためにストアの順序が守られねばならない.
- 依存情報の確度の低さにより無駄な待ちが発生し得る.

またこれまで前提としていた完全に独立したロードバッファとストアバッファという構成ではアドレスの決定が命令発行時に完了する必要があるため無理があり，更にプログラム上の順序が失われたり，キャッシュのポート数が増えるといった問題もあった．よって以下に挙げる要項を踏まえて新方式を考案した．

- フォワーディングによるロードの可急的速やかな処理.
- ストアバッファ間の書き戻しによるストア順の整合性の保証.
- 先行命令の実行終了 (確定) を待つ，ストアの確実な逐次処理.
- デュアルポートキャッシュによる二命令の並列処理.

その構成を図 8に示す．これはアドレス決定を逐次的に行なうプリアドレスバッファ (PB:preaddress buffer)，ストア命令を実行可能となるまで待機させるストアバッファ(SB)，正当なデータの到着までロード命令を待機させるロードバッファ(LB) からなる．プリアドレスバッファと当該実行側ストアバッファ，ロードバッファには一世代前の並列実行と区別するための境界ポインタを有する．プロセッサ内のキャッシュは一次キャッシュと二次キャッシュに階層化したものを想定している．小規模の一次キャッシュをデュアルポートとしてバッファとキャッシュ間の転送帯域幅を確保する．一次キャッシュと大規模な二次キャッシュ間の転送帯域幅はライン単位で一括して行なう事で確保できると考える．この構成の要点は投機的実行側のロード命令の，当該実行側からのフォワーディングを当該実行側の提供パスと投機的実行側の参照パスの双方を用意することで実現した事である．

両実行側に共通の挙動は次のようである．命令デコード後にメモリ操作命令のアドレスレジスタの内

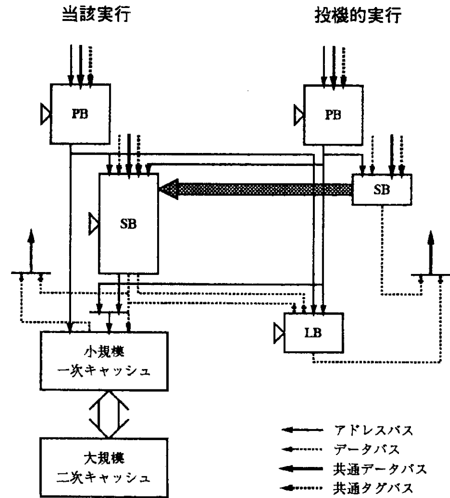


図8　新方式の構成

容またはそのタグとオフセットが順当にプリアドレスバッファに格納される．ストアの場合，ストアすべきデータまたはタグはストアバッファに直接格納される．プリアドレスバッファからは1クロックに1命令がアドレス決定されて送出されTLBミスはこの時発見される．またロードではキャッシュに対してそのアドレスを送出すると同時にストアバッファにも送出し，一致するアドレスがあるならキャッシュからのロードを中断し最新のデータをフォワーディングする．ストアでは一旦ストアバッファに順当に格納される．

当該実行側ではストアアドレスがストアバッファへ格納される時に投機的実行側のロードバッファにアドレスとデータまたはタグを提供する．このロードバッファへの提供は例外なく行なわれるが，提供ビットの立つストア命令の場合はその旨を伝える必要がある．ストアバッファは先行命令が確定され，かつバスが未使用で命令ある限りストアを実行する．

投機的実行側ではロード命令もストア命令も一旦各バッファに蓄えられる．ストアバッファはストア能力を有さずただ蓄える．ロードバッファは3つある入力パスの内，最も新しい正当なデータを取捨選択するために存在する．ロード命令ではロードバッファへの登録と同時に両ストアバッファとキャッシュへアドレスを送出する．投機的実行側のストアバッファでフォワーディング可能ならそのデータを共通データバスへ放送してそのロードは終了する．当該実行側からのフォワーディングデータはキャッシュからのロードより優先されるが提供ビットが立っていなければ直ちに送出はせずロードバッファに保持する．またロードデータが届いても依存ビットの立っているロード命令は，当該実行側の提供ビットの立つストア命令からの提供か両プリアドレスバッファからの全命令の送出の完了を

待たねばならない．

投機的実行側のストアバッファは当該実行側のストアバッファへの書き戻しを必要とする．その時期は両プリアドレスバッファ中の同世代の命令が全て送出された時である．そして書き戻しは当該実行側ストアバッファの，次に書き込まれるべき位置に投機的実行側ストアバッファの書き込まれた分だけが行なわれる．なお投機的実行側のストアバッファが溢れた場合は，投機的実行側プリアドレスバッファからの送出を停止して当該実行側のプリアドレスバッファの送出完了と有効の確定を待ち，その後ストアバッファの書き戻しを行なってから再開するものとする．

以上より各バッファの特徴を総括する．振舞いについてはロードバッファ以外はいずれもキューとして，ロードバッファは空きを詰めるキューとしてそれぞれ振舞うものとする．所要エントリ数を見積もると，ブロック当りにロード命令とストア命令が均等に n 命令ずつ存在すると仮定する場合，プリアドレスバッファが 2 ブロック分で $4n$，当該実行側ストアバッファが $2n$ 以上，投機的実行側ストアバッファが n 以上，投機的実行側ロードバッファが $2n$ 程度が妥当であろうと思われる．

3.4 正確な割り込みを保証する回復機構

順外実行かつ順外完了を行なうプロセッサでは，例外処理への移行前と復帰後で完全にプロセッサ状態が保持される，いわゆる正確な割り込みの保証のためには何らかの回復（リカバリ）機構を設ける必要がある．この機構として，順外完了に因由するのであるから完了結果を順当に並べ蓄えて順当に確定させるというリオーダバッファ (RB:reorder buffer) が有用と考える．なおリオーダバッファについての説明は割愛する．このリオーダバッファにフューチャファイル (FF:future file) を組み合わせた回復機構☆をブロック並列実行方式に適用すると図 9 のようになる．なお，これは試みに検討したものでありまだ改良の余地が多々ある．

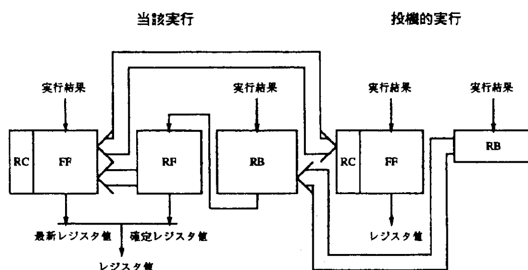


図 9　ブロック並列実行方式における回復機構

この構成の特徴は投機的実行側リオーダバッファが命令確定機能を持たずただ登録し，当該実行側リオーダバッファへ書き戻すものである事である．また例外処理から復帰する際には確定値を有するレジスタファイルの内容を当該実行側フューチャファイルへ複写する必要がある☆☆．よってリオーダバッファ間の書き戻しとレジスタファイル，フューチャファイルの複写という非常にコストの高い実現手法を採る必要がある．

4. おわりに

ブロック並列実行方式の再検討を行なった．並列実行の組合せを変更する事でより現実的な方式を目指し，動的スケジューリング機構に関して検討と考案を行った．これで大まかではあるがプロセッサ全体の構成が定まったため，正確で多様なシミュレーションを行いたい．

ブロック並列実行方式は 2 レベルの投機的実行であるがこのレベルを増やすのは容易ではない．またブロック並列実行の更に投機的実行は困難であり，今後は分岐予測等による分岐ペナルティの軽減と，命令流を単一化しブロックを大きくする条件付実行等の方式の導入を検討する必要があると考える．

またブロック並列実行方式はハードウェアで動的にスケジューリングするものではあるがより高い性能を追求するのであればソフトウェアの支援は必須である．その支援で最も優先されるべきは依存情報の確度の向上である．特にメモリ操作において依存ビットと提供ビットという信憑性の乏しい依存情報では無駄な待ちが多発し得ると考えられる．これらの確度を上げるためにはコンパイラがプログラムの意味解析を行なって，あり得る事とあり得ぬ事を確実に判断する必要がある．特にループに関してこの意味解析の方法を研究する事が今後の課題である．

参考文献

1) 朝生 良教, 柳瀬 正俊, 桐山 佳隆, 林 達也, “スーパースカラパイプラインによるブロック並列実行方式”, 情報処理学会研究報告 94-ARC-106-1,Jun 1994
2) 高田 滋, 朝生 良教, 桐山 佳隆, 林 達也, “スーパースカラパイプラインによるブロック並列実行方式”, 第 51 回全国大会論文誌 4P-2,Sep 1995
3) Mike Johnson(村上 和彰 訳), “Superscalar Microprocessor Design”, PrenticeHall, Inc(日経 BP 出版センター),Aug 1994
4) 安里 彰, 志村 浩也, “ スーパスカラプロセサにおけるリカバリー方式”, 情報処理学会研究報告 93-ARC-101-10,Aug 1993

☆ 書き戻しを行う都合上リオーダバッファのみでは実現が困難である．

☆☆ 相互書き戻しを行う都合上フューチャファイルがレジスタファイルよりも古い値を保持していては問題がある．