プログラムの制御依存を緩和し並列性を引き出すプロセッサに関する研究

小林 良太郎

目　次

1　序論

プロセッサの処理能力は，プログラム内に存在する並列性を利用することで向上させることができる．近年のハイエンド・マイクロプロセッサは，複数の命令の処理過程をオーバーラップさせる命令のパイプライン処理及び命令の並列処理により，命令レベルの並列性 (ILP: Instruction-Level Parallelism) を引き出し，性能を向上させている．

プログラムから引き出すことのできる並列性は制御依存により厳しく制限される．ある分岐命令に後続する命令は，当該分岐命令の結果が得られるまで実行できないとき，当該分岐命令に制御依存しているという．制御依存は次の２つの原因により性能低下をひきおこす．第１に，分岐命令の処理を開始してからその実行結果が得られるまで後続命令のパイプライン処理がストールすることにより，プログラムの実行サイクル数が増加する．第２に，プログラムにおいて並列性を引き出すことのできる範囲が分岐命令間に制限される．マイクロプロセッサの最も一般的な応用である非数値計算プログラムにおいては，数命令に１回の頻度で分岐命令が現われるので，上記原因による性能低下は著しい．

現在，ILP を利用する商用の主なマイクロプロセッサは，この制御依存を緩和するために，投機的実行を行っている．これは，分岐命令の実行結果が得られるまえにその結果を予測し，分岐先の命令の処理を開始する技術である．これにより，予測が成功した分岐に関する制御依存を取り除くことができる．具体的には，以下の２つの効果が得られる．第１に，分岐命令によってパイプライン処理がストールする頻度を減少させることができる．高速化のためパイプライン段数，つまり，分岐によりパイプライン処理がストールするサイクル数は増加する傾向にあり，この効果はますます重要になる．第２に，プログラムにおいて並列性を引き出すことのできる範囲を，予測に失敗した分岐命令間に広げることができる．分岐予測精度を上げるに従い，上記の効果はより大きくなるので，投機的実行を用いるプロセッサにおいて分岐予測精度の向上は非常に重要である．

さらに制御依存を緩和するために，最近，投機的実行に加えて，マルチスレッド実

行を導入するアプローチが注目されている．スレッドとはプログラムの実行におけるある1部分である．マルチスレッド実行とは，1つのスレッドを複数の互いに制御の依存しないスレッドに分割し，それらを並列に実行する技術である．マルチスレッド実行では，あるスレッド内の命令の処理は別のスレッド内の分岐命令の結果に依存しないので，プログラムの連続した部分だけでなく，不連続な部分からも並列性を引き出すことができる．Lam らの研究によれば，マルチスレッド実行を導入することにより，投機的実行にのみ頼ったアプローチよりも制御依存を大幅に緩和することができるとしている．

しかし，一般的なマルチスレッド実行機構では，スレッドの並列実行に関わるオーバヘッドが大きいという問題がある．数値計算プログラムにおいては，スレッドの並列実行により豊富な並列性を引き出すことができるため，オーバヘッドは許容できる．一方，非数値計算プログラムにおいては，数値計算プログラムと異なり，不規則な制御構造と複雑なデータ依存関係が数多く存在するため，オーバヘッドが問題とならない程の並列性は引き出すことができない．著者は非数値計算プログラムにおいて高い性能を発揮するマイクロプロセッサの実現に興味を持ち研究を行っている．その実現のためには，スレッドの並列実行に関わるオーバヘッドの削減が非常に重要である．

本研究は制御依存を緩和してプログラムの並列性を引き出すプロセッサに関するものである．制御依存を緩和する技術として，投機的実行およびマルチスレッド実行に着目する．本研究の目的は大きく2つに分けることができる．1つは，投機的実行において，いかにして分岐予測精度を向上させるかを検討することである．もう1つは，マルチスレッド実行において，いかにしてスレッドの並列実行に関わるオーバヘッドを削減するかを検討することである．

以上の目的に対して，本研究において得られた成果を以下に示す

1. 分岐方向の予測精度を向上させる機構の提案

   分岐について予測する事項としては，分岐方向と分岐先アドレスの二つがある．分岐方向の予測手法は，分岐命令のアドレスと過去の分岐方向の履歴の相

関を用いるのが一般的である．しかし，予測情報を保持するテーブルにおいて競合が発生し，分岐方向の予測成功率が低下するという問題がある．この問題に対し本研究では，競合が発生しても予測精度を大きく落さない sgshare と呼ぶ予測機構を提案した．シミュレーションによる評価の結果，sgshare 予測機構は従来の機構と比較して，ハードウェア量 8K バイト時において平均で 0.45% 予測精度が向上した．また，計算機全体の性能では今日のスーパースカラマシンで平均 7.19%，命令発行幅が増えパイプライン段数が深い将来のスーパースカラマシンでは平均 14.6% 速度向上が見込めることがわかった．

2. 分岐先アドレスを予測する機構のハードウェア量を削減する手法の提案

分岐先アドレスの予測手法は，過去の分岐先アドレスを用いるのが一般的である．過去の分岐先アドレスは BTB(Branch Target Buffer: 分岐先バッファ) に保持する．分岐先アドレスの予測成功率を高めるためには，より多くの分岐について分岐先アドレスを保持する必要がある．このため，BTB にはより多くのエントリが必要となり，ハードウェア量が大きくなるという問題がある．この問題に対し本研究では，分岐先アドレス予測成功率をほとんど低下させることなく，BTB のハードウェア量を削減する手法を提案した．シミュレーションによる評価の結果，Fagin が提案したハードウェア量削減手法を適用した BTB において，提案手法によりハードウェア量を約 38% 削減することができた．

3. プログラムの並列性を効率良く利用するマルチスレッド実行機構の提案

非数値計算プログラムにおいて，スレッドの並列実行により性能を向上させるためには，スレッドの並列実行に関わるオーバヘッド，そのなかでも特に同期／通信のオーバヘッドの削減が重要である．これまでに提案された機構は，同期／通信のオーバヘッドの削減に成功しているが，スレッド内の並列性の利用が妨げられる問題があった．そこで本研究では，スレッド内の並列性とスレッド間の並列性を同時に最大限利用することを可能とする SKY と呼ぶマルチプロセッサ・アーキテクチャを提案した．シミュレーションによる評価の結果，8 命令発行の 2 つのスーパスカラ・プロセッサにより構成した SKY は，単一のスーパスカラ・プロセッサに対して，最大 46.1%，平均 21.8% の高い性能を達

成できることを確認した．

以下，本論文は次のような構成になっている．

第2章で，分岐方向を予測するテーブルにおいて競合が発生しても分岐方向の予測精度を大きく落とさない機構を提案する．第3章で，分岐先アドレスの予測成功率をほとんど低下させることなく，分岐先アドレスを予測する機構のハードウェア量を削減する手法を提案する．第4章で，スレッド間の並列性とスレッド内の並列性を同時に最大限利用することを可能とするSKYと呼ぶマルチプロセッサ・アーキテクチャを提案する．最後に第5章で，本論文をまとめる．

2　分岐方向の偏りを利用し破壊的競合を低減する分岐予測方式

本章では予測に悪影響を与える破壊的競合の削減に焦点を当て，競合が発生しても予測精度を大きく落とさない機構について議論する．この機構を **sgshare** 予測機構と呼ぶ．まず，2.1 節において，分岐方向の予測方法について述べる．2.2 節では分岐予測機構及び競合に関する過去の研究について述べる．2.3 節では競合の問題に対する著者のアプローチについて述べる．2.4 節で本アプローチに基づく予測機構として sgshare 予測機構を提案する．2.5 節では同機構の性能の評価を行い有効性を検証する．2.6 節では本機構によりプロセッサの性能がどの程度向上するかについて調査する．2.7 節で本章をまとめる．

2.1　はじめに

分岐方向の予測手法にはさまざまな機構がある．これらは全て過去の分岐方向の履歴と将来の分岐方向の間に相関があることを利用している．初期の機構[24]は，予測する分岐自身の履歴を要約するために 2 ビットの飽和型カウンタを用いた．この方式を **2 ビット・カウンタ方式 (2bc)** と呼ぶ．2bc では，カウンタを多数並べたテーブルを設け，分岐命令のアドレスをインデクスとして参照し，過去の履歴を知り予測する．

これに対して，より高い予測精度を達成するために，より多くの履歴情報を元に予測する方式が提案された[25),34),48),50]．これらの方式が新たに利用している履歴情報とは，主として履歴のパターンや予測する分岐以外の他の分岐の履歴である．これらの方式を総称して **2 レベル適応型方式**と呼ぶ．2 レベル適応型方式では，分岐命令のアドレスと新たに加えた履歴情報の組に対して，2 ビット・カウンタで構成されるテーブルのエントリを対応付ける．

2 レベル適応型方式は 2bc に比べて優れているものの，予測精度は**競合 (aliasing)** という現象によって制限されていることが最近わかってきた[51]．競合とは，異なる分岐命令アドレスと分岐履歴の組が同一カウンタに対応付けられることをいう．

競合により，異なる組に対する予測情報を区別して記録することができなくなり，予測精度が低下する．最近の研究では，カーネルの振る舞いも含めた実際的な使用環境においては競合が頻発し，その結果，予測精度が大きく低下するという問題が注目されている[10),35)]．

競合の問題に対して，分岐アドレスと分岐履歴を２ビット・カウンタ・テーブルに対応づけるマッピング関数を工夫し競合を減少させることで対応しようとした研究がある[26),35)]．このアプローチは，テーブルのエントリを有効に利用することを促進するが，これだけによる性能改善には限界がある．なぜならば，現実的なハードウェアの制約のもとで分岐命令アドレスと分岐履歴の取り得るすべての組み合わせに対応できる巨大なテーブルを保有することが不可能であるからである．つまり，競合を回避するいかに優れたマッピング手法を用いても競合は不可避である．

著者は競合の問題に対して，競合自体を削減するのではなく，競合が生じても予測精度が低下しない方策を考えた．具体的には，分岐命令アドレスと履歴の組に対し，競合が生じたとしても分岐方向の偏りが同一のものが競合するように予測機構を構成する．こうすることで，競合により予測が誤る確率を下げる．本章ではこの考えに基づき，sgshare 予測機構[28)~30)]と呼ぶ新しい機構を提案する．

2.2　関連研究

本節では最初に，２レベル適応型分岐予測方式の概要について述べる．その中でも著者の提案する sgshare の基本となる gshare について詳しく説明する．次に，予測精度を低下させる競合に関するこれまでの研究について述べる．

2.2.1　２レベル適応型分岐予測方式

図 2.1 に**２レベル適応型分岐予測方式**[25),34),48),50)]の概要を示す．この方式ではまず，各エントリが２ビットの飽和型カウンタよりなるテーブルを持つ．このテーブルのことを**パターン履歴テーブル (PHT: Pattern History Table)** と呼ぶ．分岐命令アドレスと分岐履歴の組は，各機構特有のマッピング関数により PHT のエントリに対応付けられる．分岐が実行されると，分岐履歴及び対応するカウンタを更新す

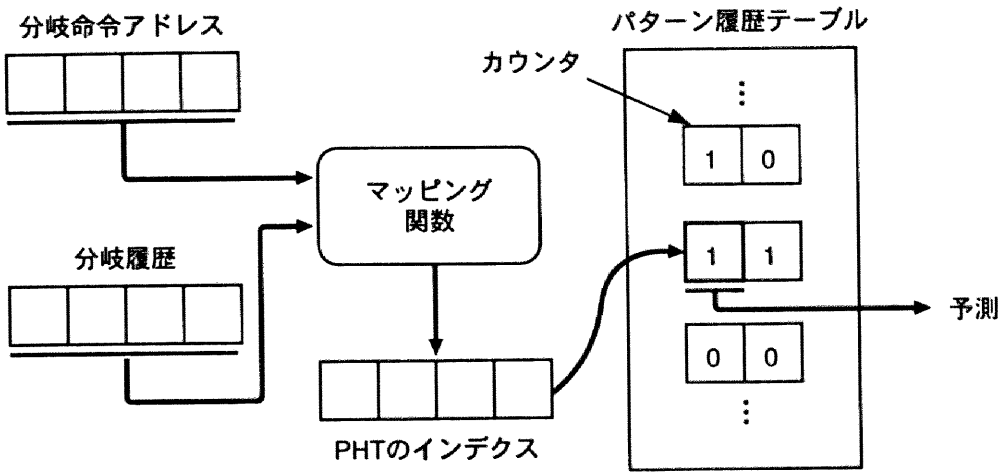


図 2.1　2 レベル適応型分岐予測方式

る．予測は対応するカウンタの値を参照することによって行う．

2 レベル予測方式では，使用する履歴や PHT へのマッピングの方法などで多くのバリエーションが考えられてきた．その中でも **gshare** [25] と呼ばれる方式は，極めて高精度な予測を行う方式として知られている．

gshare は，ある分岐の結果はその分岐以外の履歴と相関があることを利用する方式の 1 つである．図 2.2 に gshare における PHT のインデクスの生成方法を示す．履歴を記録するために m ビットのシフト・レジスタを用意する．分岐が実行されたら，シフト・レジスタを前にシフトし分岐結果を最後尾に書き込む．このようにして m ビットのシフト・レジスタに最近の過去 m 個の動的分岐の履歴を保持する．

PHT のインデクスは次のようにして生成する．分岐命令アドレスの第 $(m + n - 1)$ ビット目から第 n ビット目までの m ビットの部分と分岐履歴の m ビットの XOR を取る．これを分岐命令アドレスの下位 n ビットと結合し PHT のインデクスとする．以下，分岐命令アドレスの下位 n ビットの部分を PHT のインデクスへの**追加アドレス**と呼ぶこととする．

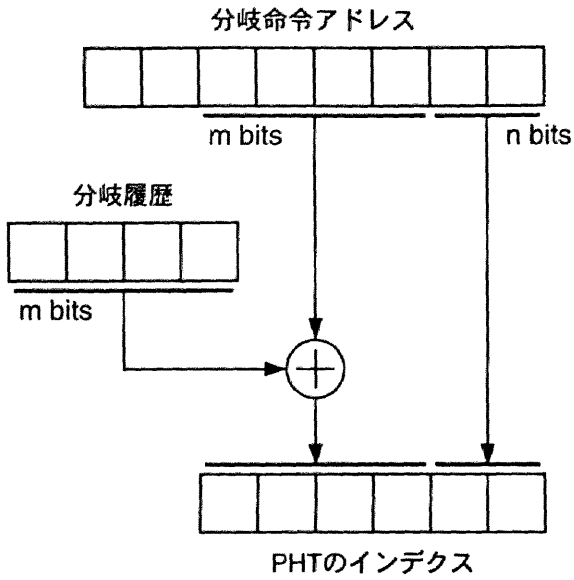


図 2.2　gshare における PHT のインデクスの生成

2.2.2　競合

分岐命令アドレスと履歴の組が，PHT の同一カウンタに対応づけられる現象を競合と呼ぶ．カーネルの振る舞いを含めた実際的な環境においては，PHT での競合が予測精度に大きな影響を与えることがわかり[10),35)]，近年この現象に対する関心が高まっている．

こうした中で，Young らは分岐予測に与える影響の観点から競合を以下の 3 種類に分類した[51)]．

- **破壊的競合 (destructive aliasing)**　競合の結果，競合がない時と異なる予測となり，かつ，その予測が誤っている場合における競合
- **建設的競合 (constructive aliasing)**　破壊的競合と同じく競合がない時と異なる予測となったが，結果的にその予測が正しかった場合における競合
- **無害な競合 (harmless aliasing)**　競合がない時と同じ予測となった場合における競合

競合は予測に対して必ず悪い影響を与えるというわけではない．上記分類では，無害な競合は影響を与えないし，建設的競合は予測精度を改善しさえする．予測精度を低下させるのは破壊的競合だけである．しかし Young らの調査では，破壊的競合の方が建設的競合に比べ圧倒的に多いという結果を得ている．つまり，競合は総合的に見れば予測に悪い影響を与えるということができる．

一方 Michaud らは，競合の生じる原因によって以下の 3 種類に分類した [26]．

- **初期競合 (compulsory aliasing)**　初めてテーブルを参照した際に発生する競合
- **容量競合 (capacity aliasing)**　テーブルの容量不足に起因する競合
- **対立競合 (conflict aliasing)**　マッピングに起因する競合

彼らはこのうち対立競合を低減する gskewed と呼ぶ方式を提案した．gskewed では，異なるマッピング関数を持つ 3 つの予測器を用意する．予測においては，これらの予測器を同時に参照し，得られた予測結果の多数決によって最終的な予測結果とする．予測器のマッピング関数として，ある 1 つの予測器で競合が生じたとしても，残りの 2 つの予測器では起きにくいものを用いる．これにより，対立競合による予測精度低下を減少させた．3 つの予測器を用いるので，与えられたコストの下では 1 つの予測器を用いる場合に比べて容量競合が増加するが，対立競合減少による効果の方が大きいという測定結果を示した．

2.3　破壊的競合の削減

競合という問題に対するこれまでのアプローチは主にマッピング関数を工夫して，競合の発生頻度を減少させるというものであった [26),35]．このアプローチでは，対立競合の削減に効果があるものの，初期競合や容量競合の発生頻度を削減することはできない．そのためこのアプローチのみでは，競合による性能低下を取り去るには不十分である．

しかし，初期競合と容量競合の出現頻度を削減することは難しい．初期競合は初めて PHT が参照される際には必ず発生するので出現を妨げることはできない．ま

た，容量競合の出現頻度を下げるには，単純にテーブルの容量を大きくするか，インデクスに用いる分岐アドレスと分岐履歴の組み合わせ総数を減少させるかである．前者はコストが増加し，後者は予測に利用する情報の減少によって予測精度が低下する．したがって，現実的なハードウェアの制約の下では競合の発生は本質的に不可避であるということができる．

このように競合の発生がある程度は避けられないものならば，より高精度な予測のためには，たとえ競合が起きても予測精度を下げない仕組みが必要である．つまり，予測機構の**競合への耐性**が必要である．このためには，より具体的にいえば，競合が破壊的となりにくい仕組みを構築する必要がある．

ところで，多くの分岐は taken か not-taken のどちらかに偏っている（例えば文献 5) 参照)．つまり，競合の多くは，互いに異なる方向に偏った分岐の間か，同じ方向に偏った分岐の間で発生する．互いに異なる方向に偏った分岐の間で競合を起こした場合，対応づけられたカウンタは taken の状態と not-taken の状態の間を激しく振動することになり正確な予測を行うことができない．これは破壊的競合である．逆に，同じ方向に偏っている分岐が競合を起こした場合には，カウンタは同じ状態へ遷移するため，競合が発生しない状態と同じく正確な予測を行うことができる．以上より，著者は，競合を引き起こす分岐の偏りの方向が同じになるような仕組みを設ければ，予測精度を悪化させる破壊的競合を削減できると考えた．

2.4　sgshare 予測機構

前節では，競合を起こす分岐の偏り方向をそろえることで，破壊的競合が削減できることを述べた．これを実現するために著者は，PHT を分岐方向が taken に偏った分岐用のテーブルと，not-taken に偏った分岐用のテーブルに分離し別々に管理する方式を提案する．以下この方式を**分離型 PHT 方式 (Separate PHT)** と呼ぶ．PHT を taken に偏った分岐用と not-taken に偏った分岐用とに分離することにより，仮に競合が発生したとしても，同じ分岐方向同士の競合，すなわち無害な競合となる可能性を高め，逆の分岐方向同士の情報の競合，つまり破壊的競合となる可能性を下げる．

gshare 予測機構にこの手法を適用したものを著者は sgshare 予測機構と呼ぶ．以下では，この予測機構の仕組みの詳細について述べる．

2.4.1　sgshare 予測機構の構成

図 2.3 に sgshare 予測機構の構成を示す．同図に示すように sgshare は，**T テーブル**，**N テーブル**，及び **分岐偏り履歴テーブル (BBHT: Branch Bias History Table)** の 3 つのテーブルで構成する．

T テーブルと N テーブルは，これまでの 2 レベル予測機構における単一の PHT を，taken に偏った分岐用のものと，not-taken に偏った分岐用のものとに分離したものである．どちらも図 2.2 に示した通常の gshare と同じく，分岐命令アドレスと分岐履歴の XOR を取ったものをテーブルのインデックスとする（図 2.3 では，簡単のため追加アドレスに関しては省略している）．

BBHT は分岐毎の分岐方向の偏り情報を保持する．各エントリはカウンタよりなり，分岐結果に応じて増減させる．BBHT より得られる偏り情報は，T テーブルと N テーブルのどちらの予測結果を使用するかを選択するために用いる．

予測結果は以下の手順で得る．まず，3 つのテーブルを同時に参照する．次に，BBHT の出力結果により T テーブルの出力か N テーブルの出力を選択し，これを最終的な予測結果とする．BBHT より taken に偏った分岐であることがわかった場合，T テーブルの出力を予測とする．逆に，not-taken に偏った分岐であることがわかった場合，N テーブルの出力を予測とする．

テーブルの更新は次のようにして行う．BBHT は，予測の際に参照されたカウンタを分岐結果に従い増減させる．PHT は，T テーブルと N テーブルの両方のカウンタが参照されているが，予測の際に BBHT が選択したテーブルのカウンタのみ更新する．これは，T テーブルに taken に偏っている分岐の情報だけを保持し，N テーブルに not-taken に偏っている分岐の情報だけを保持するようにするためである．

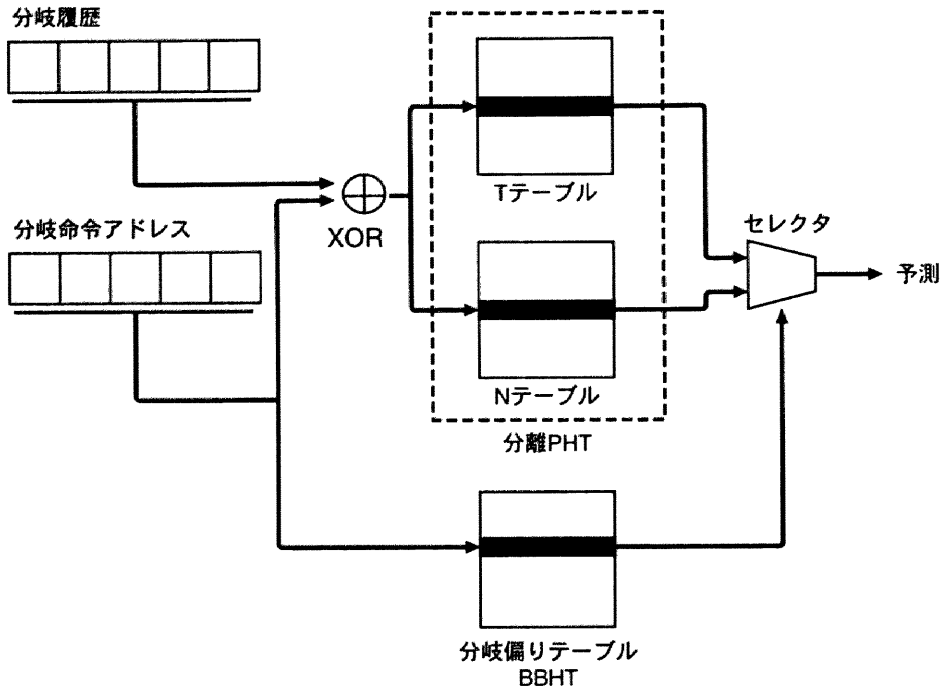


図 2.3　sgshare 予測機構

2.4.2　sgshare 予測機構のコスト配分

sgshare を実現する際，与えられたハードウェア・コストを BBHT と PHT の間で最適に配分する構成を見出す必要がある．全ての場合を尽くして測定を行い最適な構成を見つけるには，測定に非常に時間がかかる．そこで本項では，測定方法の指針を得るために，どのようにコストを配分するのが良いかを定性的に議論する．

まず著者は，BBHT には PHT に優先してコストを配分すべきと考えた．これは，BBHT は PHT での破壊的競合を抑えるための分岐のマッピングを行うので，BBHT で競合が起きると sgshare の機能が著しく低下すると考えられるからである．

次に BBHT の構成について考える．BBHT のコストは，エントリ数とカウンタ・ビット数の積によって決まる．したがって，これらの間にトレードオフが存在する．これに関しては，カウンタ・ビット数よりエントリ数の方を優先しコストを割り当てるべきと考えられる．なぜなら，先に述べたように BBHT での競合は sgshare の機能を著しく低下させると考えられるので，競合が生じないようにエント

リ数は十分でなければならないからである．

分岐方向の偏りを捕えるには，カウンタ・ベースの分岐予測の研究[24]から推測すると，2ビット程度あれば良いと考えられる．しかしコスト制約が非常に厳しく，十分なエントリ数が得られない場合は，カウンタのビット数を1ビットにする選択肢もある．逆にコスト制約が緩く，十分なエントリ数が確保できるならば，カウンタのビット数を多くし，より精度の高い分岐偏り情報を得るという選択肢も存在する．

2.5　性能評価

本節ではsgshareを評価する．最初に評価環境について述べる．次に，構成のためのパラメータを変化させ性能を評価する．

sgshareに関するパラメータを以下に示す．

- BBHTのエントリ数
- BBHTのカウンタ・ビット数
- 分岐履歴長
- 追加アドレス長

2.4.2項での議論にしたがい，まずBBHTに関するパラメータを変化させ評価し，BBHTの構成を決定する．その上で，分岐履歴と追加アドレスの長さを変化させ，従来のgshareと比較することによりPHTを分離したことの効果を評価する．

2.5.1　評価環境

トレース駆動シミュレーションにより評価を行った．ベンチマーク・プログラムとしてIBS-Ultrix[43]を用いた．表2.1に各ベンチマークの説明と，これらに含まれる動的な分岐数と静的な分岐数を示す．これらのベンチマークの各トレースはDECstation (MIPS R3000)にハードウェア・モニタをつなぐことで，OS (Ultrix 3.1)の実行トレースも含めて採取された．そのため，このベンチマークは従来用いられてき

表 2.1　IBS-Ultrix ベンチマーク・プログラム

プログラム	分岐数		説明
	動的	静的	
groff	11,568,181	5,634	nroffのGNU C++版 (version 1.09)
gs	14,288,742	10,935	Ghostscript (version 2.4.1) テキストと画像を含む1ページの表示
jpeg_play	20,926,069	6,716	xloadimage (version 3.0) 2ページのJPEG画像の表示
mpeg_play	8,109,029	4,752	mpeg_play (version 2.0) 圧縮ファイルからの85フレームの表示
nroff	21,368,201	4,480	Ultrix 3.1版nroff
real_gcc	13,940,672	16,716	GNU Cコンパイラ (version 2.6)
sdet	5,221,321	4,583	マルチプロセス性能評価 SPEC SDMベンチマークの1つ
verilog	5,692,823	3,918	Verilog-XL (version 1.6b) マイクロプロセッサのシミュレーション
video_play	5,175,630	3,977	mpeg_playの修正版 圧縮されていないファイルからの610フレームの表示

た SPEC 等から採取したアプリケーションのみの実行トレースに比べ，より実際の環境に近い実行トレースを提供できるとされている．このため，最近の分岐予測の研究で多く用いられている[10),23),26),35)]．

2.5.2　BBHT の構成

本項では，PHT に関するパラメータを固定し，BBHT のエントリ数とカウンタ・ビット数を変化させ性能に与える影響を調査する．これにより BBHT の最適なコスト配分を決定する．

図 2.4 に BBHT のエントリ数とカウンタ・ビット数を変化させた時の分岐予測ミス率の全ベンチマークでの平均を示す．PHT の履歴を 10 ビット，追加アドレスを 0 ビットとした．

図 2.4 からエントリ数が 2K 以下とそれほど多くない時には，2 ビットや 3 ビットのカウンタを用いるよりも，1 ビットのものを用いた方が性能が良いことがわかる．この理由は，ビット数が少ないほうが BBHT のウォーム・アップ時間が短いためである．つまり，あるエントリにそれまで割り当てられていた分岐とは異なる

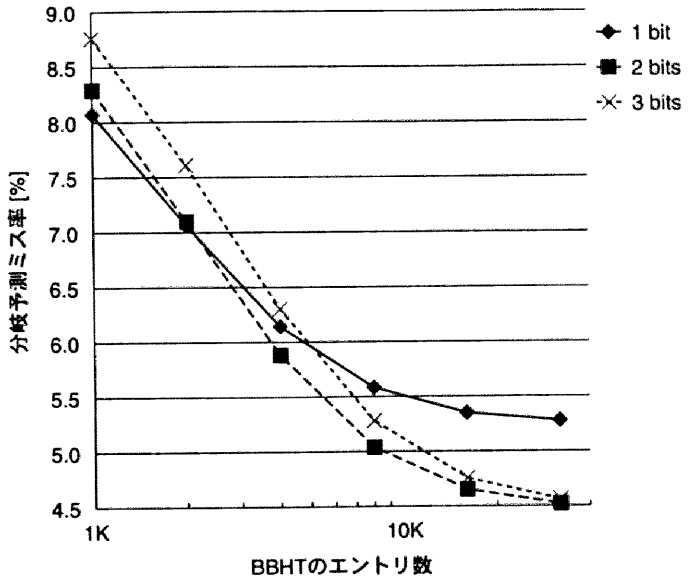


図 2.4　BBHT のエントリ数とカウンタ・ビット数の
予測精度に与える影響

分岐が割り当てられた時，そのエントリのカウンタが新しく割り当てられた分岐に関して十分な情報を蓄えるまでの時間は，ビット数が少ないほど速い．例えば 3 ビット場合，カウンタ値が 0 の時に逆の偏りを持つ分岐が新しく割り当てられたとすると，中央の値を越えるまでにはその分岐は少なくとも 4 回 ($2^3/2$ 回) 実行されなければならない．これに対して 1 ビットなら，1 回ですむ．エントリ数が 2K 以下の時は，カウンタのビット数を増加させることによる情報の確かさの向上より，競合からのウォーム・アップ時間が重要となり，ビット数が少ない方が性能が良いという結果となっている．

BBHT を 1 ビット・カウンタで構成する場合は，8K エントリ程度で性能が飽和する．この時の BBHT のハードウェア量は 1K バイトである．

図 2.4 の測定結果より，BBHT により多くのハードウェアが割けるならば，BBHT を 2 ビット・カウンタで構成し，より正確な情報を蓄えることが性能改善に貢献することがわかる．この場合には，16K エントリ程度で性能は飽和する．この時の BBHT のハードウェア量は 4K バイトである．

BBHT のカウンタ・ビット数を 3 ビットに増やしても，測定した 32K エントリまででは 2 ビットで構成したものの性能を上回ることができなかった．32K エントリの時のハードウェア量は 12K バイトにもなることから，3 ビット以上のカウンタを用いることは良い選択ではないことがわかる．

以上のことから，低コストの予測機構においては 1 ビットの BBHT を，大規模な予測機構では 2 ビットの BBHT を用いることが適当であることがわかった．以降の性能評価では，BBHT として 8K エントリで 1 ビット・カウンタの構成と，16K エントリで 2 ビット・カウンタの構成についてのみ評価する．

2.5.3　PHT 分離の効果

本項では，PHT を T テーブルと N テーブルに分離することで，予測精度と競合にどのような効果が現れるかを調査する．PHT の大きさを 32K エントリとし，以下の 3 つのモデルについて測定した．

- gshare: gshare 予測機構
- sgshare1: BBHT を 1 ビット×8K エントリで構成した sgshare 予測機構
- sgshare2: BBHT を 2 ビット×16K エントリで構成した sgshare 予測機構

履歴長と予測精度の関係

図 2.5 に，履歴長を変化させ各予測機構について予測精度を測定した結果をベンチマーク毎に示す．横軸は履歴長で，縦軸は予測ミス率である．PHT のエントリ数は 32K で固定なので，履歴長を 1 増加させた場合，追加アドレス長は 1 減らしている．

まず第 1 に，sgshare1，sgshare2 ともに，履歴長が非常に短い場合を除いて，全てのベンチマークで gshare より良い予測精度を示している．jpeg_play はわずかな履歴を用いるだけで，99% という非常に高い予測精度を得ることができるので改善の余地がほとんどない．しかしその他のベンチマークでは，各予測機構での最善の履歴長と追加アドレス長の組み合わせにおいて，gshare に比べ sgshare1 は 0.24 ～

0.71%，sgshare2 は 0.37 ～0.91% 性能が良い[1]．

どの予測機構も履歴長を延ばせば，予測に利用する情報が増加するためあるところまでは予測精度が改善する．しかし gshare の場合，あまり長くするとほとんどのベンチマークで予測精度が悪化する．これは履歴を延ばすことにより，1 つの静的分岐に対して対応付けられる PHT のエントリ数が増加することにより競合が増加するからである．履歴を長くし過ぎると，競合による負の効果が，多くの情報を予測に用いる正の効果を上回り，結果として予測精度が低下する．

この現象は real_gcc において著しい．図 2.5 を見ればわかるように履歴を 7 ビットより長くすると急速に予測精度が悪化する．real_gcc は表 2.1 に示したように他のベンチマークに比べて静的分岐数が非常に多い．そのため PHT での競合が起きやすく，競合による負の効果が顕著に現われる．

このように競合により性能が著しく低下するという性質は予測機構として望ましくない．なぜならば，プロセッサに予測機構を組み込む際，何らかの評価の下に妥当なパラメータに決めたとしても，設計段階で想定したプログラム（これは評価に使ったプログラムの平均であろうが）より静的分岐数の非常に多いプログラムが実行された場合，著しく性能が低下してしまうからである．望ましい性質は，設計段階で想定しなかったような静的分岐数の多いプログラムがたとえ実行されたとしても，予測精度が大きく下がらないことである．つまり，予測機構には競合への耐性が要求される．

図 2.5 からわかるように，sgshare は gshare と異なり競合への耐性が非常に強い．履歴を長くし競合が頻発する状況となっても，予測精度に与える負の影響は小さい．例えば，real_gcc では，gshare は履歴長を最適なものから 14 ビットに増加させると予測精度は 1.58% も悪化するが，sgshare1 では 0.54%，sgshare2 ではわずか 0.15% しか悪化しない．

1. 一般にパーセントで 2 つの数値を比較するときは，それらの比をいう。本論文でも断りのないかぎり数値をパーセントで表現するときは，比をいうこととする．ただし，予測精度の比較では，例えば，予測精度 A_1% の機構は予測精度 A_2% の機構より A_3% 良いと表現する場合は，予測精度の差 (A_1-A_2) が A_3 であることを意味する．

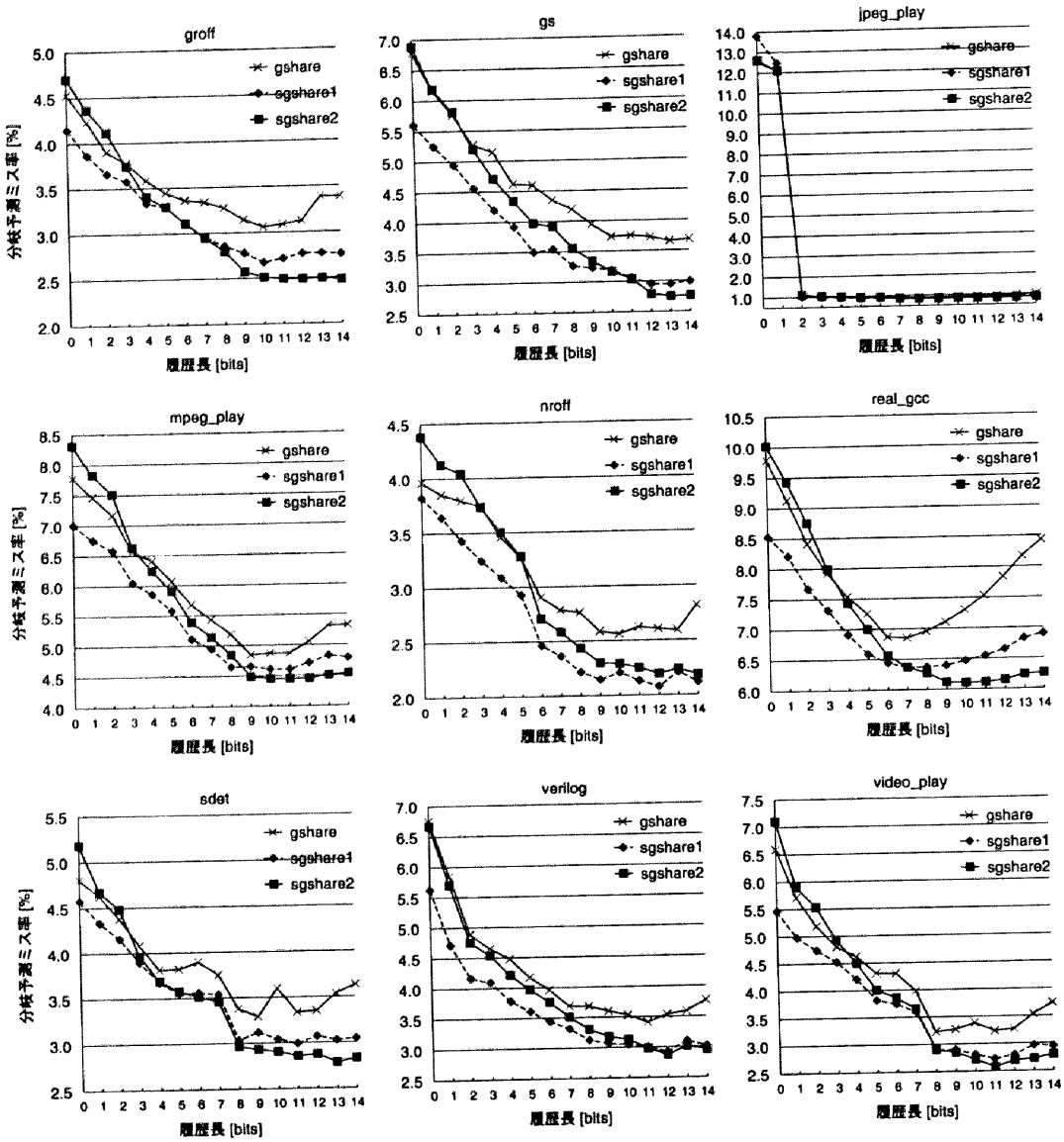


図 2.5　32K エントリの PHT での履歴長と予測精度

競合内容

図 2.6 は，各予測機構の**競合率**とその種類の内訳を示したものである．競合率とは，競合の発生回数を実行した分岐の数で割って算出したもので，競合がどのような頻度で発生したかを示す．図 2.6 の縦軸はその競合率を示す．横軸は履歴長である．0 から 14 までの各履歴長に対する 3 つの棒グラフは，左から gshare, sgshare1, sgshare2 に関するものである．各棒グラフは競合の種類で 3 つの部分に分解してい

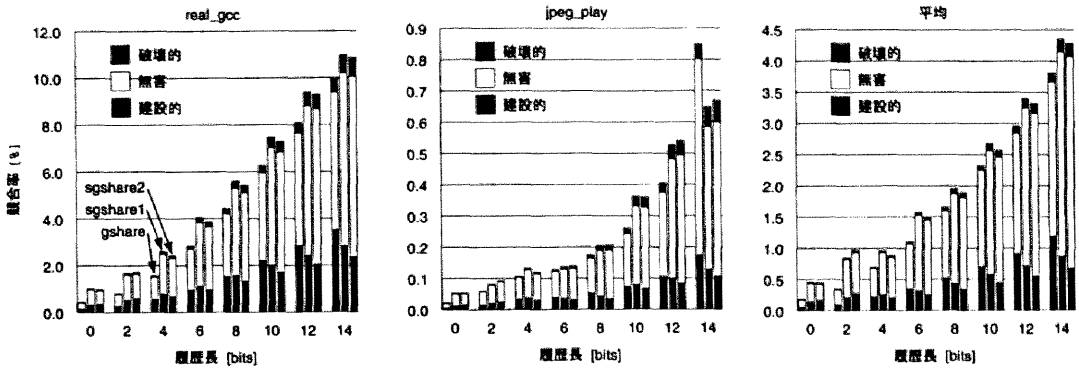


図 2.6　競合率とその内訳

る．それぞれ下から，破壊的競合，無害な競合，建設的競合の発生頻度である．競合の分類は，実際の予測機構と同一履歴長で PHT の大きさが無限で競合のない予測機構による予測結果を，実際の予測機構の予測結果と比較することにより行った．図 2.6 には，最も競合を起こす real_gcc と，最も競合を起こさない jpeg_play と，全ベンチマークの平均の 3 つの結果を示した．他のベンチマークの結果は，これらと類似した傾向を持ち，数値は real_gcc と jpeg_play 間のものとなる．

図 2.6 からわかるように，real_gcc では競合は平均の倍もの頻度で起こっている．14 履歴の時の競合率は約 10% もあり，競合が予測精度に大きく影響していることを裏付けている．一方 jpeg_play では，競合は平均の 1/5 ～1/10 しかない．14 履歴の時でも 1% もなく，競合による予測への影響は小さいことが確認できる．

履歴長にかかわらず sgshare は，競合の発生頻度そのものは，ベンチマーク平均で 0.3 ～0.5% 程度増加している[2]．これは，sgshare では BBHT の出力する分岐の偏り情報が変化することによって全く同じアドレスと分岐履歴を持つものであっても両方のテーブルへ対応づけられることがあるためである．PHT に 2 重に情報を蓄えれることがあるので，競合の総数は増加する．

競合自体は増加しているものの，予測に悪影響を与える破壊的競合については，ベンチマーク平均で見ると，履歴長 6 ビット以上ではむしろ減少している．特に履歴

2. これらの数値は競合率の差である．

が長く，競合の発生頻度が高くなった時の減少率は大きい．履歴長 14 ビットの時，sgshare1 は gshare での破壊的競合の 27.2% を，sgshare2 は 43.4% を削減している．

履歴長が 6 ビット以下と少なく競合自体があまり発生することのない場合は，sgshare では破壊的競合はわずかに増加している．しかし，図 2.5 での予測性能の結果から見てわかるように，履歴が極めて短い時は高い予測精度を達成することができないので，このような結果は重要ではない．

また，予測に好影響を与える建設的競合は，sgshare ではすべての履歴長において gshare より高い．これは競合全体の発生頻度の増加によるものである．

以上より，PHT を分割したことで破壊的競合を無害な競合に変えるという仕組みは有効に機能していることがわかる．

2.5.4　コスト性能比の評価

本項では，sgshare と gshare のコスト性能比について調査する．前項と同様に，gshare, sgshare1, sgshare2 について測定を行う．

PHT の大きさを変化させ，各 PHT に対し全ベンチマークでの平均予測精度が最も高くなる履歴と追加アドレスの配分を測定によって求めた．図 2.7 に，この最適な配分での各ベンチマークにおける予測精度を示す．ただし，jpeg_play を省略し，代わりに全ベンチマークでの平均値を載せた．jpeg_play は図 2.5 に示したように，わずかな履歴を予測に用いるだけで非常に高い予測精度を示し，コスト依存性がほとんどない．このため本研究にとっての興味はほとんどない．ただし平均の測定結果には含めている．

図 2.7 において，横軸のハードウェア・コストとは，各予測機構を構成するテーブルの大きさ (単位：K バイト) である．つまり，gshare の場合 PHT の大きさであり，sgshare1 及び sgshare2 の場合，PHT と BBHT の大きさの合計である．

図 2.7 から，ハードウェア・コストが 2K バイト以下の時には，多くのベンチマー

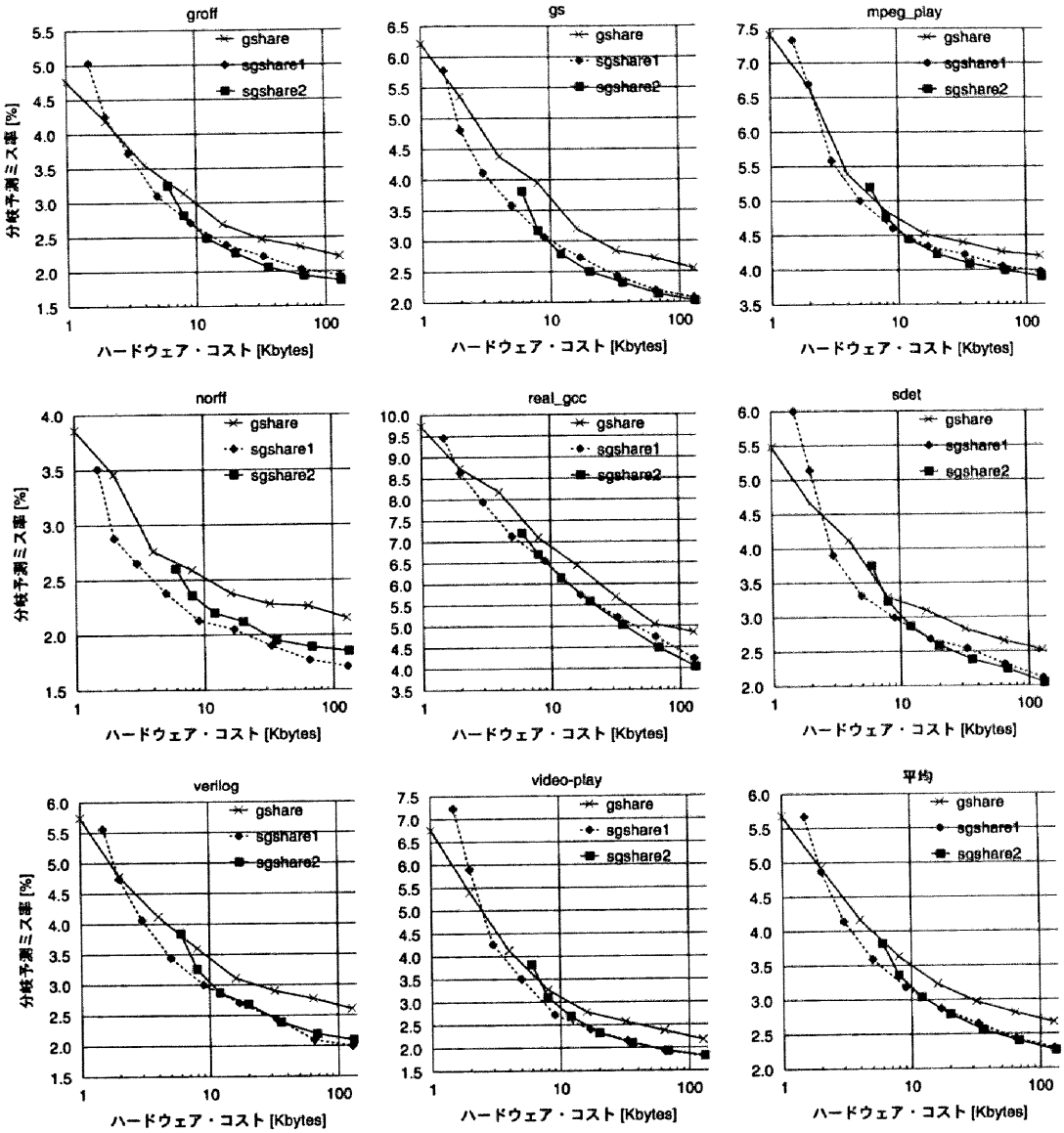


図 2.7　ハードウェア・コスト対予測精度

クで sgshare1 は同規模の gshare の性能に劣っている．同様に，ハードウェア・コストが 6K バイト (sgshare2 の測定での最少のコスト) の時には，半数のベンチマークで sgshare2 は同規模の gshare の性能に劣っている．8K エントリの 1 ビット・カウンタで構成した BBHT は 1K バイト，16K エントリの 2 ビット・カウンタで構成した場合にはこれだけで 4K バイトのハードウェア量を要する．このため，小規模な構成の時では，破壊的競合削減の効果よりも BBHT を加えたことによるコスト

増加の影響が大きく，同一コストでは予測精度が向上しない．

逆に費やすことのできるハードウェア・コストが大きくなると，BBHT を追加することによるコスト増は相対的に小さなものとなる．この場合，破壊的競合が削減された効果により同一コストにおいても予測精度が向上する．ほぼ同等のコストで比較すると，例えば 8K バイトの gshare と 9K バイトの sgshare1 では，最大 0.89%(gs)，ベンチマーク平均で 0.45% 予測精度が向上する．コストの大きなところでは sgshare2 がやや有利となり，128K バイトの gshare と 132K バイトの sgshare2 では，最大 0.82%(real_gcc)，ベンチマーク平均で 0.41% 予測精度が向上する．

表 2.2 に，図 2.7 の「平均」のグラフの各測定点における履歴長と追加アドレス長の組み合わせを示す．この表から，sgshare は同程度の予測精度を持つ gshare と比較して，追加アドレス長を短くできることがわかる．例えば，予測ミス率約 3.2% を実現するのに gshare，sgshare1 ともに 11 の履歴長を要しているが，追加アドレス長は gshare が 5 であるのに対し，sgshare1 では 2 ビット少ない 3 で同程度の予測精度を達成している．同様に予測ミス率約 3.0% を達成するには，gshare，sgshare2 ともに 12 の履歴長を要しているが，追加アドレス長は gshare では 5，sgshare2 では 2 である．これは，sgshare では破壊的競合が削減された結果，gshare と同じ履歴長を利用して同程度の予測精度を達成するために必要とする PHT のハードウェア・コストが 1/4 ～1/8 で済むことを意味する．このため前述の 2 つの例では，それぞれ 43.8%((1 - 9Kbytes/16Kbytes)*100)，62.5%((1 - 12Kbytes/32Kbytes)*100) も少ないハードウェア量で gshare と同程度の予測精度を達成している．このことから，sgshare は非常にコスト性能比の高い予測機構であることがわかる．

2.6　プロセッサの性能に与える効果

本節では，分岐予測精度の向上が現在及び将来のプロセッサにおいてどの程度性能向上に貢献するかについて評価を行う．

評価は，トレース駆動のシミュレーションを用いて行った．ベンチマークには IBS-Ultrix の 9 本のベンチマークのそれぞれ 1000 万命令のトレースを用いた．以下

表 2.2　分岐履歴と追加アドレスの配分

予測機構	PHTのインデクス		コスト [Kbytes]	予測ミス率 [%]
	履歴長	追加アドレス長		
gshare	4	8	1	5.67
	5	8	2	4.92
	9	5	4	4.17
	9	6	8	3.63
	11	5	16	3.23
	12	5	32	2.97
	12	6	64	2.81
	16	3	128	2.67
sgshare1	5	5	1.5	5.67
	6	5	2	4.87
	9	3	3	4.14
	9	4	5	3.59
	11	3	9	3.18
	11	4	17	2.87
	14	2	33	2.65
	17	0	65	2.43
	18	0	129	2.29
sgshare2	10	2	6	3.82
	11	2	8	3.36
	12	2	12	3.04
	13	2	20	2.79
	16	0	36	2.56
	17	0	68	2.40
	18	0	132	2.26

つ4つのモデルについて評価を行った.

- Current モデル：8 命令フェッチ，分岐予測ミスペナルティ 10 サイクル
- Wide モデル：32 命令フェッチ，分岐予測ミスペナルティ 10 サイクル
- Deep モデル：8 命令フェッチ，分岐予測ミスペナルティ 40 サイクル
- Wide&Deep モデル：32 命令フェッチ，分岐予測ミスペナルティ 40 サイクル

Current モデルは現在のハイエンドのプロセッサ[11),12)]に近い命令発行幅とパイプラインの深さのものを想定している．他の3つはいずれも将来におけるプロセッサを想定したものである．Wide モデルは，将来のプロセッサが命令発行幅の増加という方向へ進んだ場合のモデルである．一方 Deep モデルは，将来のプロセッサがパイプラインを深くしクロック速度を向上させる方向へ進んだ場合のモデルである．最後の Wide&Deep モデルは，パイプライン段数，命令発行幅の両方が増加する方向へ進んだ場合のモデルである．

評価した n 命令フェッチのマシンは，現在の PC から分岐予測が誤るまで最大 n 命令フェッチし，リザベーション・ステーションに登録し，実行可能になった命令から out-of-order で実行する．機能ユニットは，ALU，分岐，ロード／ストア，浮動小数点ユニット等からなり，それぞれ n 個持つ．リザベーション・ステーション，リオーダ・バッファ，キャッシュ，BTB は十分大きくし，これらによってパイプラインが停止することがないようにした．

各モデルにおいて，分岐予測機構として 8K バイトのコストの gshare を組み込んだものと，ほぼ同等のコストである 9K バイトの sgshare1 を組み込んだものとを比較した．これらの予測機構のパラメータは前節までの測定結果に基づき最適に設定した．

図 2.8 に，gshare を用いた場合に対する sgshare1 を用いた場合の各モデルの実行サイクル数による速度向上率を示す．いずれのモデルにおいても，予測精度が向上した結果，性能が向上している．jpeg_play では既に予測精度が 99% に達しており，sgshare による予測精度の向上は小さい．このため，どのモデルについても jpeg_play の性能向上は小さい．しかし他のベンチマークでは大きな性能向上を達成していることがわかる．Current モデルでは 4.0 ～13.3% 性能が向上している．パイプライン段数が深い Deep モデル，Wide&Deep モデルでは，9.8 ～27.2% もの性能向上を達成している．

9K バイトの sgshare1 の予測ミス率はベンチマーク平均で 3.2% である（図 2.7 参照）．この予測ミス率は十分に小さいように感じられるが，依然としてプロセッサ

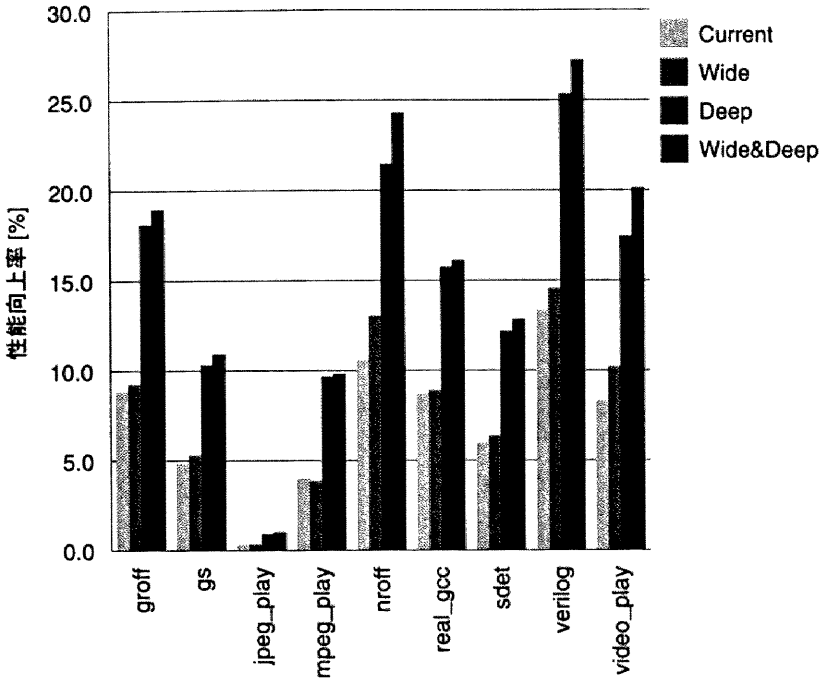


図 2.8　プロセッサ性能への効果

性能に多大なる影響を及ぼしている．表 2.3 に gshare を組み込んだ Current モデルに対し，分岐予測が 100% 成功した場合の性能向上率を示す．性能向上率は jpeg_play を除いて約 26.8 ～75.7% もある．分岐予測精度はさらなる改善が必要であることがわかる．

表 2.3　分岐予測率 100% による性能向上率

プログラム	性能向上率 [%]
groff	51.9
gs	48.9
jpeg_play	3.0
mpeg_play	26.8
nroff	61.5
real_gcc	75.7
sdet	39.9
verilog	66.5
video_play	47.6

3　2レベル表方式による分岐先バッファ

本章では分岐命令の分岐先までの距離の分布を利用し，予測成功率をほとんど低下させることなく分岐先アドレス予測機構のハードウェア量を削減する機構について議論する．この機構を**2レベル表方式のBTB**と呼ぶ．まず，3.1節において，分岐先アドレスの予測方法について述べる．3.2節では関連研究について述べる．3.3節では分岐命令アドレスと分岐先アドレスの関係をアドレスのビット列の点から考察し，それを利用したハードウェア量削減のための提案手法について述べる．3.4節では提案手法によるハードウェア量削減の評価と検討を行う．

3.1　はじめに

ILPを利用する商用の主なマイクロプロセッサでは，制御依存を緩和するために，分岐命令の実行結果が得られるまえにその結果を予測し，分岐先の命令の処理を開始する投機的実行を行っている．ある分岐命令の分岐先は，当該分岐命令をフェッチして分岐方向および分岐先アドレスを計算することにより得られるので，投機的実行を行うためには，前章で述べた分岐方向だけでなく，分岐先アドレスも予測する必要がある．分岐方向の予測手法には様々な機構があるが，分岐先アドレスの予測手法にはBTBを用いるのが一般的である[24]．BTBとは分岐先アドレスを保持するキャッシュで，分岐命令アドレスをインデクスとして当該命令の分岐先アドレスを出力する．

分岐先アドレスの予測成功率を高めるためには，エントリにおける分岐命令間の競合を避けることが必要である．このためBTBには多くのエントリ数が必要となりハードウェア量が増大するという問題がある．したがって，予測成功率を低下させずにBTBのハードウェア量を減少させるには，エントリあたりのハードウェア量を減少させることが必要である．

BTBのエントリはタグ部と分岐先アドレス部からなる．これまでに，タグ部のハードウェア量を削減するための手法はいくつか提案されている[8),9),45)]．一方，分岐先アドレス部のハードウェア量を削減するための研究[13)]はまだ多くはなされて

おらず，有効な手法の提案や詳細な評価は不十分である．

本研究の目的は，分岐先アドレス予測成功率をほとんど下げることなくBTBをより少ないハードウェア量で実現することである．本章ではこの目的を達成する2レベル表方式のBTB[46),47)]を提案する．

3.2　関連研究

まず分岐先アドレスの予測方法について述べる．つぎにBTBのタグ部と分岐先アドレス部のハードウェア量削減に関する研究について述べる．

3.2.1　分岐先アドレスの予測方法

一般的にリターン命令の分岐先アドレス，つまりリターン先アドレスの予測手法には**リターン・スタック**[16)]を用い，それ以外の分岐先アドレスの予測手法にはBTBを用いる．BTBは分岐先アドレスを保持し，分岐命令アドレスをインデクスとして当該命令の分岐先アドレスを出力する．BTBの予測成功率を高めるためには，多くのエントリ数が必要となりハードウェア量が増大するという問題がある．

リターン・スタックとはリターン先アドレスを保持するスタックである．手続き呼出し時にリターン先アドレスをプッシュし，予測時にリターン先アドレスをポップする．予測成功率を高めるために必要なリターン・スタックのサイズは小さいのでハードウェア量は問題とならない．

複数の分岐先がある間接分岐に対して，BTBではなく間接分岐専用のバッファを用いる予測手法が提案されている[6),7)]．間接分岐専用バッファは1つの間接分岐に対して複数の分岐先を保持し，分岐先アドレスと実行パスとの相関を利用して予測を行う．間接分岐専用バッファは間接分岐に関してはBTBよりも高い予測成功率を得ることができる．しかし間接分岐命令が全分岐命令数に占める割合は小さいにも関わらず，高い予測成功率を得るために必要とされるハードウェア量はBTBを上回るという問題点があるので，一般には間接分岐専用バッファは用いられていない．

これらの予測方法に対して，文献 4) では，分岐命令アドレスの上位部分と分岐命令にエンコードされた即値を連結することで分岐先アドレスを生成する方式を提案している．このような分岐命令を **Latched Branch** と呼ぶ．Latched Branch は，分岐命令をフェッチするサイクルにおいて分岐先アドレスを簡単に生成できるため，BTB を用いて分岐先アドレスを予測する必要がない．しかし，BTB を削除するために命令セットを変更しなければならない．また，連結する命令アドレスの上位部分が同一であるセグメント内でしか分岐できないという欠点がある．特に，近い分岐先でセグメント境界を越えている場合に，大きな問題となる．これらの問題点があるので，一般には Latched Branch は用いられていない．

以上をまとめると，分岐先アドレスの予測手法には BTB とリターン・スタックを用いるのが一般的である．リターン・スタックのハードウェア量は小さいが，多くのエントリ数を必要とする BTB はハードウェア量が大きいので，エントリあたりのハードウェア量を減少させる必要がある．

3.2.2　BTB のタグ部の削減

文献 8),9) では，エントリあたりのタグ・ビット数削減と分岐先アドレス予測成功率のトレードオフについて検討されている．エントリ数 512，ダイレクト・マップ方式の構成で SPECint92 によって評価した場合，タグ・ビット数が 2 の場合でもタグ・ビットを全て保持する場合の精度の 99.9% を得ることができると報告している．

CAT（Caching Address Tags）と呼ばれる機構[45] では，キャッシュのタグ部の情報と，別に用意したタグ・キャッシュの情報を組み合わせてタグ値を生成する．これによってエントリに保持するタグ部のビット数を削減させる．

3.2.3　BTB の分岐先アドレス部の削減

従来の BTB では分岐先アドレスをそのまま保持する．これに対して，**変位保持方式**[13] では分岐命令アドレスと分岐先アドレスの差分（変位）を保持する．分岐先アドレスの予測は現在の分岐命令アドレスに BTB から得られる変位を加算するこ

とによって行う．変位のほとんどはある値以下に分布していることに着目して，分岐予測成功率をほとんど低下させることなくBTBの分岐先アドレス部のハードウェア量を削減することができる．しかしこの方式では，変位がBTBに保持できない程に大きい場合は正しい分岐先アドレスを算出できないという問題点があるので，分岐先アドレス部の大幅な削減は困難である．また，加算操作が必要なため，BTB参照から予測アドレスを得るための遅延時間が長いという欠点もある．

3.3　分岐距離の分布を利用した予測手法

分岐命令アドレスと分岐先アドレスの差分を**分岐距離**という．本節では，分岐距離の分布を調べ，それを利用した分岐先アドレスの予測手法を提案する．

3.3.1　分岐距離

図3.1に分岐距離に対する動的分岐数の累積分布をSPECint95について調べた結果を示す．図において，分岐距離は2進数で表記したときに要するビット数で表し，動的分岐数は全動的分岐数に対する割合で表している．

図3.1より，分岐距離はほとんどが15ビット以下であり，20ビット以上のものは存在しないことがわかる．この分布を直接利用したものが変位保持方式である．

3.3.2　Dビット数

本項では，分岐命令および分岐先アドレスのビット列の点から分岐距離の性質を考える．

分岐命令アドレスと分岐先アドレスの間の距離として，**Dビット数**（Differentビット数の意）と呼ぶ距離を定義する．2つのアドレスのDビット数がnであるとは，これらのアドレスを1ビットごとに比較したとき，異なる値を持つビットの内，最も上位の桁が第nビット目であることをいう．例えば，010100と011110の場合，異なる値を持つビットの内，最も上位の桁は4桁目なので，Dビット数は4である．アドレス値によっては分岐距離が同じでもDビット数が異なるが，Dビット数の分布と分岐距離の分布はほぼ等しいと考えられる．

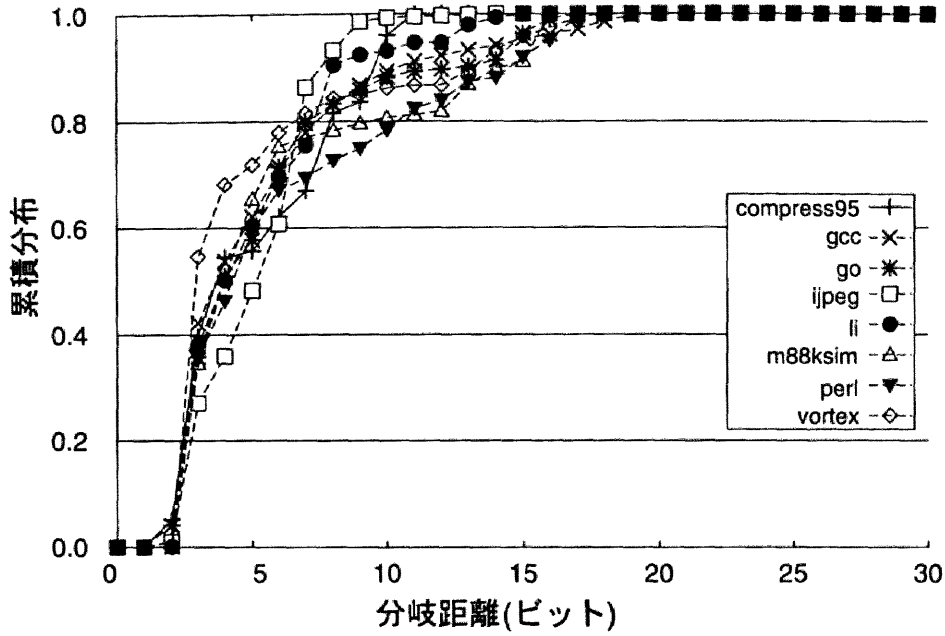


図 3.1　分岐距離の累積分布

3.3.3　PC 連結方式

前項までに示した分岐距離および D ビット数の分布より，BTB には分岐先アドレスの全てのビット列を保持する必要はなく，下位の一部のみを保持すればよいと考えられる．すなわち，分岐先アドレスの下位部分のみを BTB に保持し，BTB からの出力値と現在の分岐命令アドレスの残りの上位部分を連結することにより目的の分岐先アドレスを生成する手法である．この方式を**PC 連結方式**と呼ぶことにする．

図 3.2 に PC 連結方式の構成を示す．命令アドレスを BTB へのインデクスとする．命令アドレスの総ビット数が A ビットであることに対し，保持する分岐先アドレス情報は分岐先アドレスの下位部分の a ($a<A$) ビットだけである点が特徴である．分岐先アドレスを全て保持する従来の方式に対して，エントリあたりのハードウェア量を $(A-a)$ ビット削減することができる．

分岐先アドレス部のハードウェア量（ビット数）は以下の式で表される．

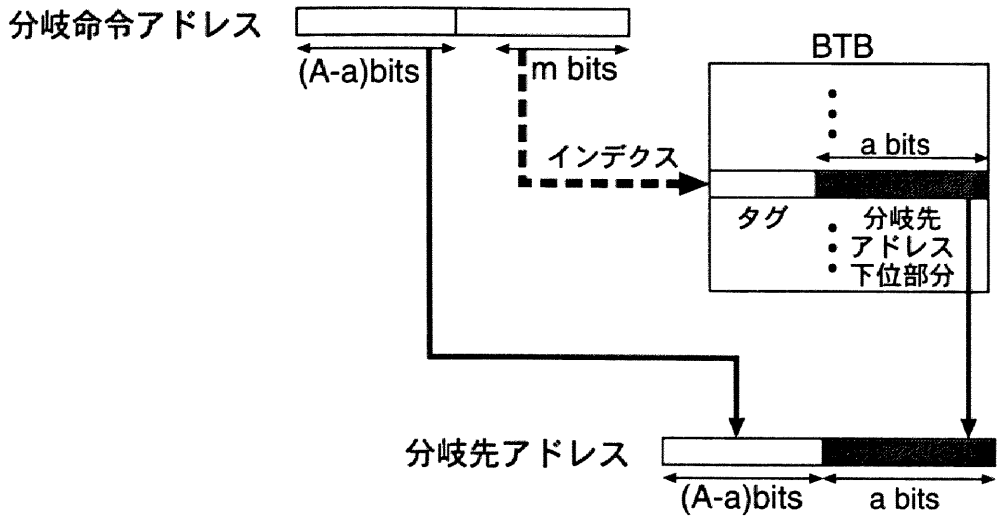


図 3.2　PC 連結方式

ハードウェア量 = エントリ数 × a

PC 連結方式は容易に BTB のハードウェア量を削減することができるが，変位保持方式と類似した問題点がある．すなわち，D ビット数が BTB の分岐先アドレス部ビット数よりも多い場合，正しい分岐先アドレスを表現できない．このため，高い予測精度を維持するためには，分岐先アドレス部のビット数を大きく削減することはできない．

3.3.4　2 レベル表方式

PC 連結方式での問題点を解決するために，著者は 2 レベル表方式を提案する．2 レベル表方式では，D ビット数が多く PC 連結方式では正しく分岐先アドレスを表現できない場合に備えて，PC 連結方式のテーブルに加えて分岐先アドレスの上位部分を保持する別のテーブルを用意する．

図 3.3 に 2 レベル表方式の構成を示す．分岐先アドレスの下位部分を保持するテーブルを第 1 テーブルと呼び，上位部分を保持するテーブルを第 2 テーブルと呼ぶことにする．分岐命令アドレスを双方のテーブルへのインデクスとする．分岐先アドレスの上位として，分岐命令アドレスの上位部分を用いるか，第 2 テーブルからの

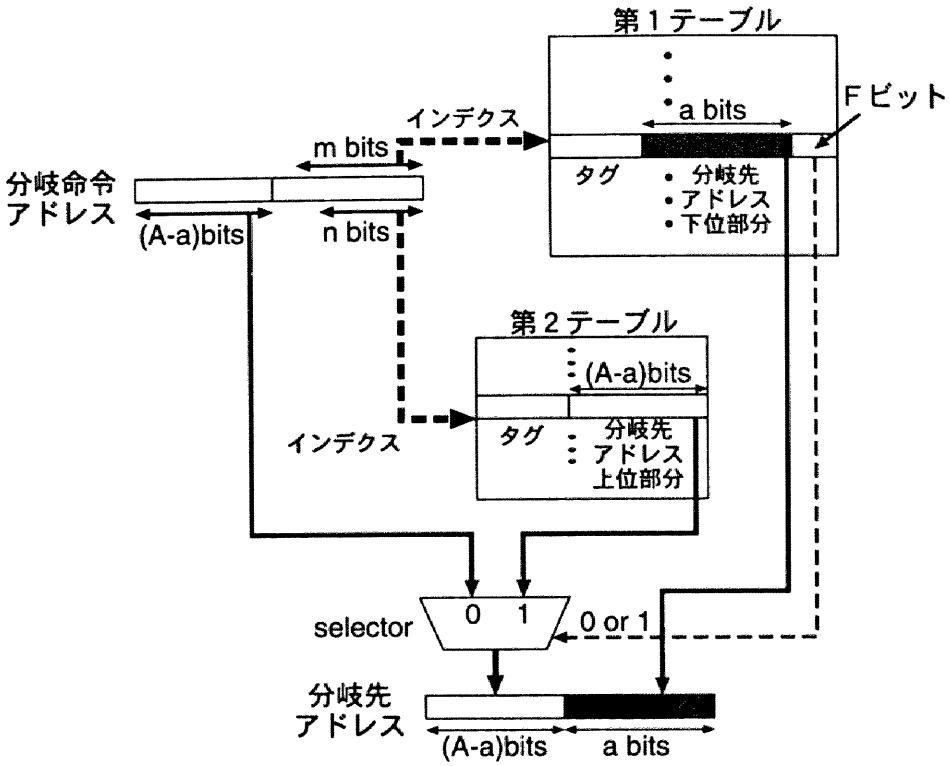


図 3.3　2レベル表方式

出力を用いるかを選択するための判定ビットを，第1テーブルの各エントリに保持する．このビットを**Fビット**（Farビットの意）と呼ぶことにする．Fビットの値が0なら分岐命令アドレスの上位を用い，1なら第2テーブルからの出力を用いる．

テーブル更新は次のように行う．まず実際に算出された分岐先アドレスと分岐命令アドレスとを比較する．Dビット数が a（第1テーブルの分岐先アドレス部ビット数）より大きくない場合，第2テーブルを使用する必要がないので，第1テーブルのみを更新し，第1テーブルの当該エントリのFビットを0にリセットする．そうでなければ，第1テーブルと第2テーブルの両方を更新し，第1テーブルの当該エントリのFビットを1にセットする．第1テーブルのエントリは常に用いられるが，第2テーブルのエントリはDビット数が a より大きい場合にのみ用いるので，第2テーブルのエントリ数は第1テーブルのエントリ数に比べて少なくてすむ．

分岐先アドレス部とFビットのハードウェア量の合計は以下の式で表される.

$$\text{ハードウェア量} = \text{第1テーブルのエントリ数} \times (a+1) + \text{第2テーブルのエントリ数} \times (A-a)$$

3.4　2レベル表方式の評価

本節では，従来方式に対するPC連結方式と2レベル表方式によるハードウェアの削減率を比較し，2レベル表方式の有効性を示す.

3.4.1　評価環境

ベンチマーク・プログラムとして，SPECint95の全8種を使用した．ベンチマーク・プログラムはNEC EWS 4800/360AD（CPUはMIPS R4400）のコンパイラで最適化し，MIPS R2000[17]用のコードを生成し，これを測定に用いた．評価はトレース駆動シミュレーションにより行った．トレースはpixie[37]を用いて採取した.

3.4.2　共通の測定条件

評価する各方式において共通する条件を以下に挙げる.

- 予測成功率とは，動的な無条件分岐とtakenの条件分岐に対し，BTBによって予測した分岐先アドレスが正しかった割合である．特に断りのない限り，全ベンチマークによる算術平均値を用いる.
- 命令アドレスは30ビットとする.
- BTBの更新は，無条件分岐の場合には常に行う．条件分岐の場合には分岐方向がtakenの場合に行う.
- リターン命令に対しては，リターン先アドレスをリターン・スタック[16]を用いて予測し，BTBを用いない．リターン・スタックの深さは十分あるとする.
- 分岐方向は全て正しく予測できるものとする.

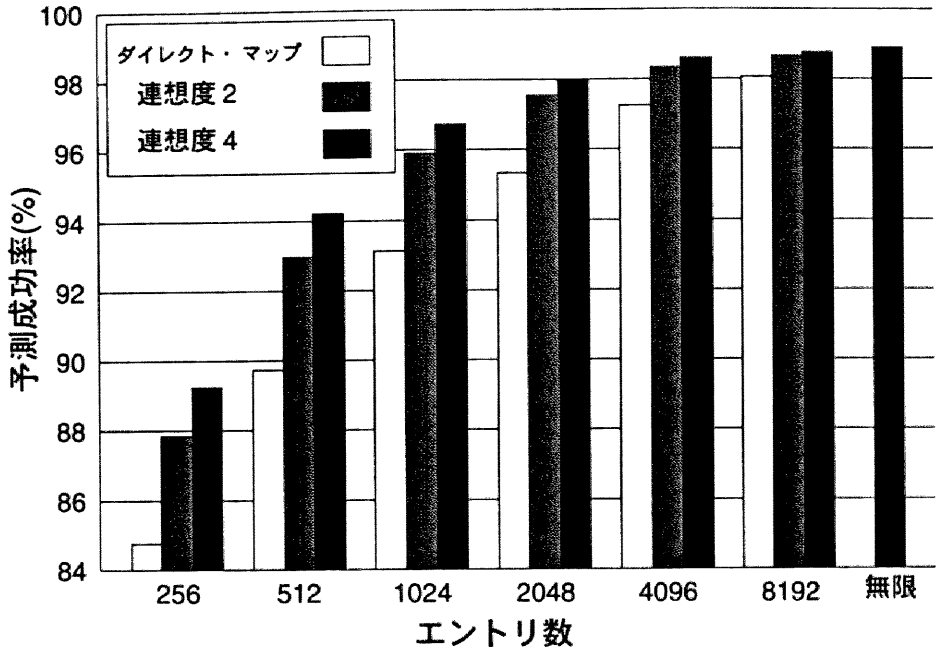


図3.4　従来方式でのエントリ数および連想度と予測成功率

- BTBのエントリの入れ替えのアルゴリズムとして，LRU（Least-Recently Used）法を用いる．

以降，**従来方式**とは，分岐先アドレス部に命令アドレスの全てのビットを保持する方式を意味するものとする．また，ある方式における**予測成功率の低下率**とは，その方式と同じエントリ数および連想度（2レベル表方式では第1テーブルのエントリ数および連想度）を持つ従来方式のBTBで得られる予測成功率と比べてどれだけ予測成功率が低下したかをいう．

3.4.3　BTBのエントリ数と連想度

本項では，PC連結方式および2レベル表方式の第1テーブルのエントリ数と連想度を定める．図3.4に従来方式のBTBでエントリ数と連想度を変化させて予測成功率を測定した結果を示す．3本で組になっている棒グラフのうち，左側がダイレクト・マップ，中央が連想度2，右側が連想度4の場合の予測成功率である．無限のエントリ数の場合の予測成功率は限界値である．

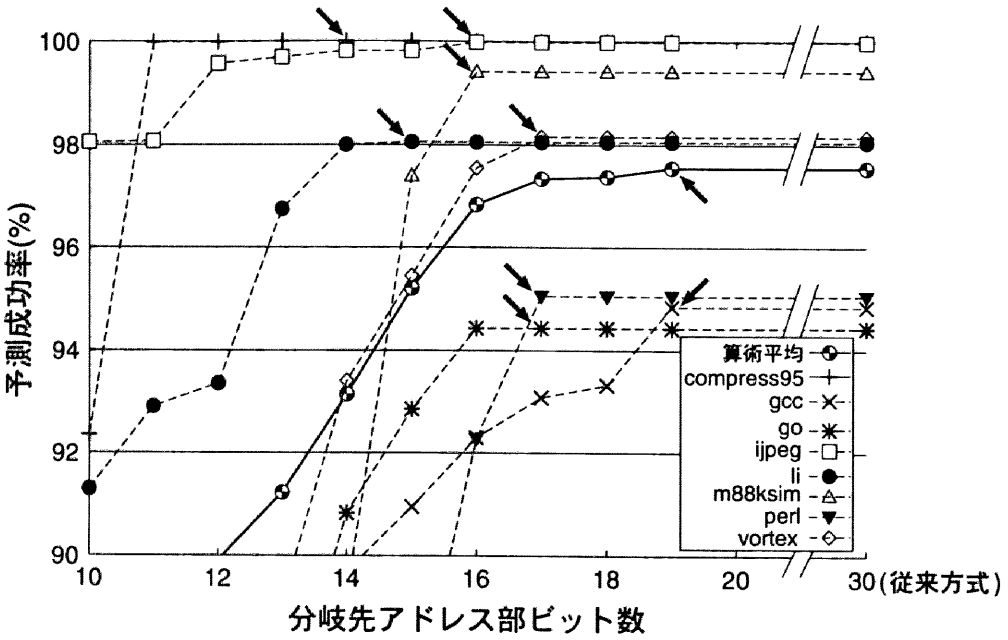


図 3.5　PC 連結方式での分岐先アドレス部ビット数と予測成功率

図 3.4 より，エントリ数 2048，連想度 2 で，限界値に対して低下は 1% 未満と限界に近い予測成功率を得ることができる．以降の評価では特に断りの無い限り PC 連結方式および 2 レベル表方式の第 1 テーブルのエントリ数を 2048，連想度を 2 とする．

3.4.4　分岐先アドレス部の削減

本項では，まず PC 連結方式について，分岐先アドレス部ビット数と予測成功率の関係を示す．その後，2 レベル表方式による分岐先アドレス部の削減を評価する．本項で示すハードウェア量とは，BTB の分岐先アドレス部および F ビットの合計とし，タグ部を含めない．測定ではタグは全てのビットを保持するものとする．タグ部を含めた場合の評価は 3.4.5 項以降で行う．

PC 連結方式の評価

図 3.5 に PC 連結方式における分岐先アドレス部ビット数と予測成功率の評価結果を，ベンチマークごとに示す．図において，矢印で示す箇所よりも分岐先アドレス

部ビット数を少なくすると予測成功率が低下していく．

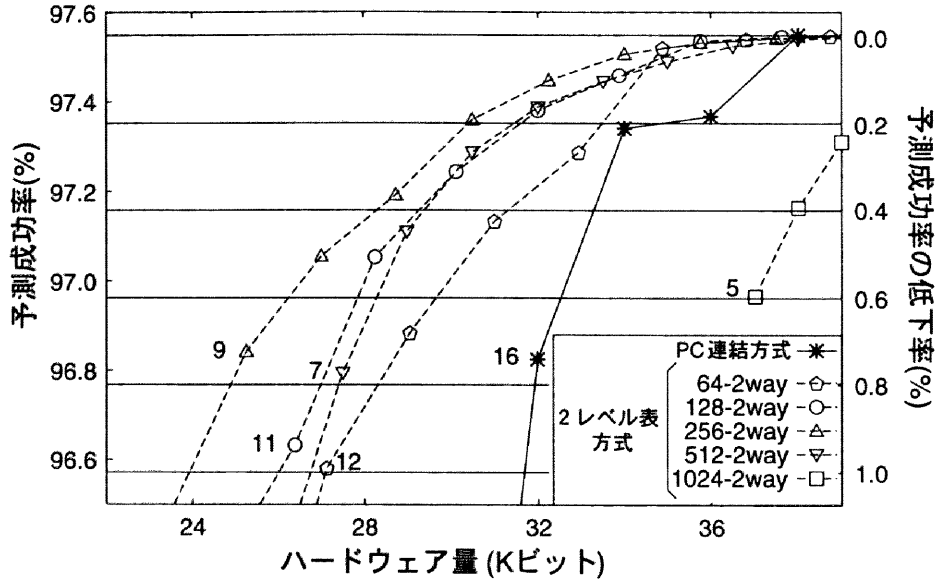
予測成功率を全く低下させないために必要な分岐先アドレス部ビット数の最小値はベンチマークごとに異なるが，あらゆるベンチマークに渡って予測成功率を全く低下させないためには，19ビット必要である．

2レベル表方式の評価

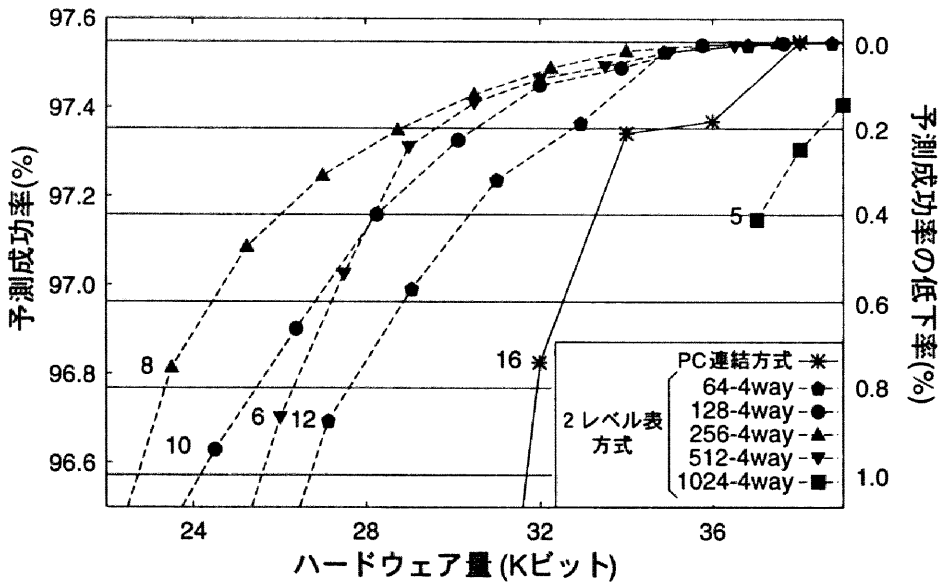
図3.6に2レベル表方式におけるハードウェア量と予測成功率の評価結果を示す．図の右の縦軸には，従来方式に対する予測成功率の低下率を併せて示す．図3.6(a)は第2テーブルの連想度を2とした場合，(b)は連想度を4とした場合である．各折れ線グラフは，第2テーブルのエントリ数が64,128,…,1024の場合の測定結果である．折れ線グラフの種類を示す*X*-*Y* wayは，第2テーブルのエントリ数*X*連想度*Y*であることを意味する．各構成について第1テーブルの分岐先アドレス部ビット数を1ビットずつ変化させて測定した．一つの折れ線グラフにおいて最も左側の測定点に付加した数字は，その点における第1テーブルの分岐先アドレス部ビット数を示し，それより右側の測定点は，右に1つ移動するごとに第1テーブルの分岐先アドレス部ビット数が1ビット多い．

図3.6より，第2テーブルのエントリ数64～512の場合，2レベル表方式は同等の予測成功率をPC連結方式より少ないハードウェア量で実現できることがわかる．また，従来方式の予測成功率からの低下率を大きく許容するほど同等の予測成功率を得るためのハードウェア量の差が大きいことがわかる．

第2テーブルのエントリ数で比較すると，連想度が2でも4でも第2テーブルのエントリ数が256の場合が同等の予測成功率を最も少ないハードウェア量で実現できることがわかる．エントリ数が256より少ない場合ではエントリ数が不足して置き換えが頻繁になされてしまうので予測成功率が低下する．これを避けるためには，第2テーブルを使用する頻度を低くするために第1テーブルの分岐先アドレス部のビット数を大きくしなくてはならず，ハードウェア量が大きくなってしまう．一方，エントリ数が256より多い場合ではエントリが十分に使用されずハードウェアが無駄に大きくなってしまっている．



(a) 第2テーブルの連想度2



(b) 第2テーブルの連想度4

図3.6　2レベル表方式でのハードウェア量と予測成功率

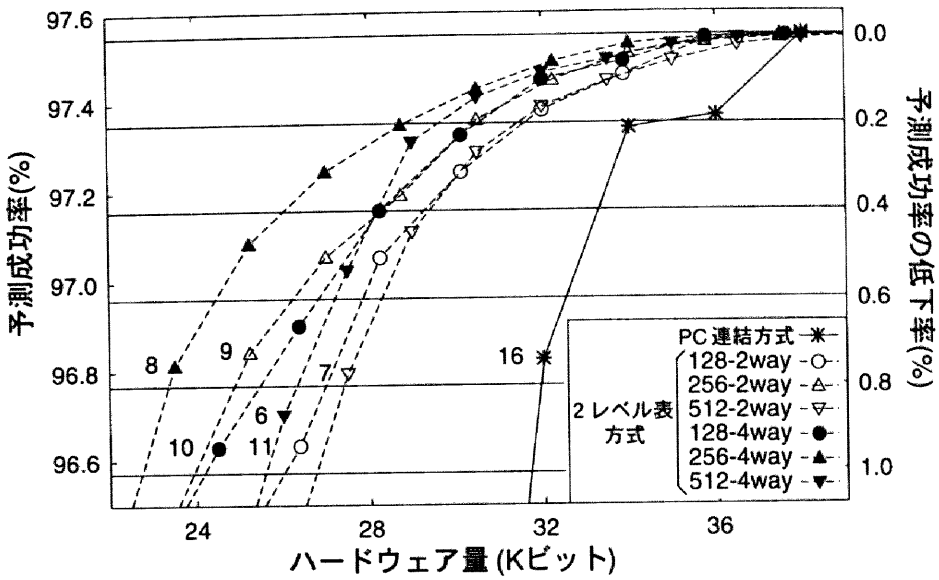


図 3.7　2レベル表方式でのハードウェア量と予測成功率(まとめ)

図 3.7 に，第 2 テーブルの連想度が 2 および 4 の場合の結果を共に示す．第 2 テーブルのエントリ数は，図 3.6 より，削減量の多い 128，256 および 512 に限定して示す．エントリ数に関わらず連想度が 2 より 4 の方が同等の予測成功率をより少ないハードウェア量で実現できることがわかる．以上より，最適な構成は，第 1 テーブルの分岐先アドレス部ビット数 12，第 2 テーブルのエントリ数 256，連想度 4 であることがわかった．このとき，従来方式に比べ予測成功率をほとんど低下させることなく（低下率 0.1%[3]）ハードウェア量を約 49%（60.0K ビットから 30.5K ビット）削減できる．

3.4.5　タグ部を含めた BTB 全体の削減

前項までに，2 レベル表方式は従来方式や PC 連結方式に比べて，同等の予測成功率をより少ない分岐先アドレス部のハードウェア量で実現できることを示した．しかし，BTB にはタグが存在するので，タグ部のハードウェア量も含めた全体のハードウェア量がどれだけ削減できるのかを調べなければならない．図 3.7 で示し

3. 分岐先アドレス予測ミスペナルティ 6 サイクル，発行幅 8 命令のスーパスカラ・プロセッサのシミュレータで評価した結果，計算機性能の低下は軽微であった．

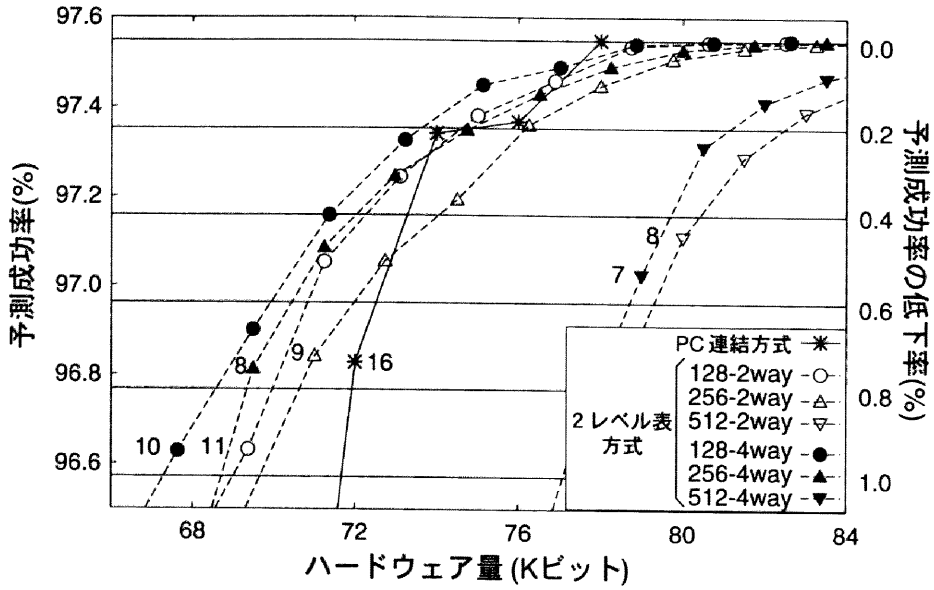


図3.8　タグ部を含めた全体ハードウェア量と予測成功率

た結果について，タグ部を含めた場合のハードウェア量と予測成功率の関係を図3.8に示す．

図3.8において，PC連結方式に対する2レベル表方式のハードウェア削減量が小さくなり，場合によってはPC連結方式よりも多くのハードウェア量を必要としてしまっている．この理由は2つある．1つは，2レベル表方式では第1テーブルのタグ部の量は変わらないが第2テーブルのタグ部が余分に必要なためである．もう1つは，タグ部のハードウェア量を含めることにより，分岐先アドレス部削減が全体に及ぼす効果が減少するためである．

この問題を解決し2レベル表方式を有効なものとするには，タグ部のハードウェア量も削減する必要がある．

Faginのタグ部削減手法

Faginによって提案されたタグ部のハードウェア量削減手法とは（以降**Faginの削減手法**と略記），単にエントリあたりのタグ・ビット数を少なくするものである[8),9)]．文献8),9)ではタグ・ビット数は数ビットで十分であると報告している．ただ

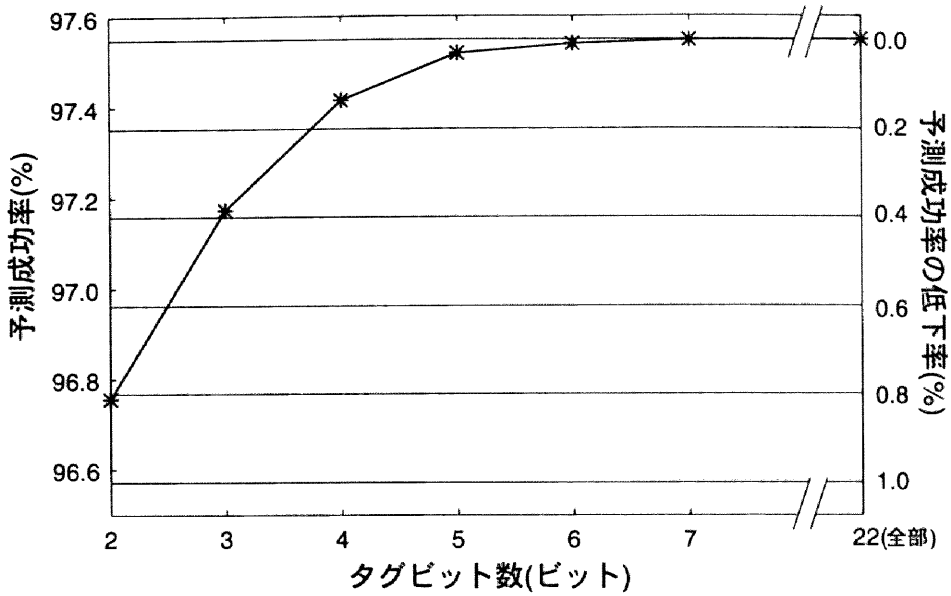


図 3.9　従来方式に対する Fagin の削減手法適用による予測成功率

し，タグ・ビット数が少なすぎると異なる分岐命令同士を同一視してしまい，誤った分岐先アドレスを出力したり BTB の内容を誤って置き換えてしまったりする．この問題は，連想度が大きい場合により顕著になることは容易に推測できる．

以下，Fagin の削減手法を 2 レベル表方式に適用し評価する．

Fagin の削減手法の適用：第 1 テーブル

2 レベル表方式の第 1 テーブルは従来方式と同様に必ず用いられる．従来方式に対して Fagin の削減手法を適用し，予測成功率への影響を調べることによって，2 レベル表方式の第 1 テーブルのタグ部の必要なビット数を見積もる．図 3.9 に従来方式でのタグ・ビット数と予測成功率の関係を示す．

図 3.9 より，タグの全てのビットを保持する場合に対して予測成功率をほとんど低下させないためには（低下率 0.1%），タグ・ビット数は 5 必要であることがわかる．よって，これ以降の評価では，従来方式および 2 レベル表方式の第 1 テーブルのタグ・ビット数を 5 とする．

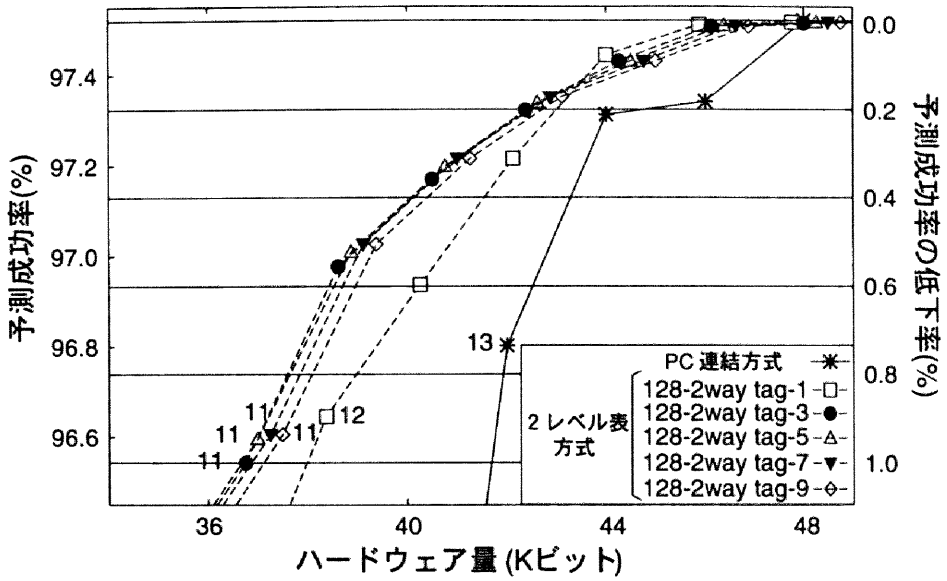
Fagin の削減手法の適用：両テーブル

3.4.4 項で得られた構成に対して Fagin の削減手法を適用させ，BTB 全体のハードウェア量と予測成功率の比較を行う．図 3.7 においてタグ部を含まないハードウェア量を比較した際，第 2 テーブルのエントリ数は 256 の場合が最適であることを示した．第 2 テーブルのエントリ数を 512 とする場合では第 2 テーブルのタグ部のハードウェア量がさらに多くなるので，第 2 テーブルのエントリ数が 256 の場合と同等の予測成功率をより少ない全体ハードウェア量で実現することは期待できない．よって，本項では第 2 テーブルのエントリ数は 128 および 256 のみを評価する．図 3.10 および図 3.11 に，タグ部も含めた全体ハードウェア量と予測成功率の関係を示す．

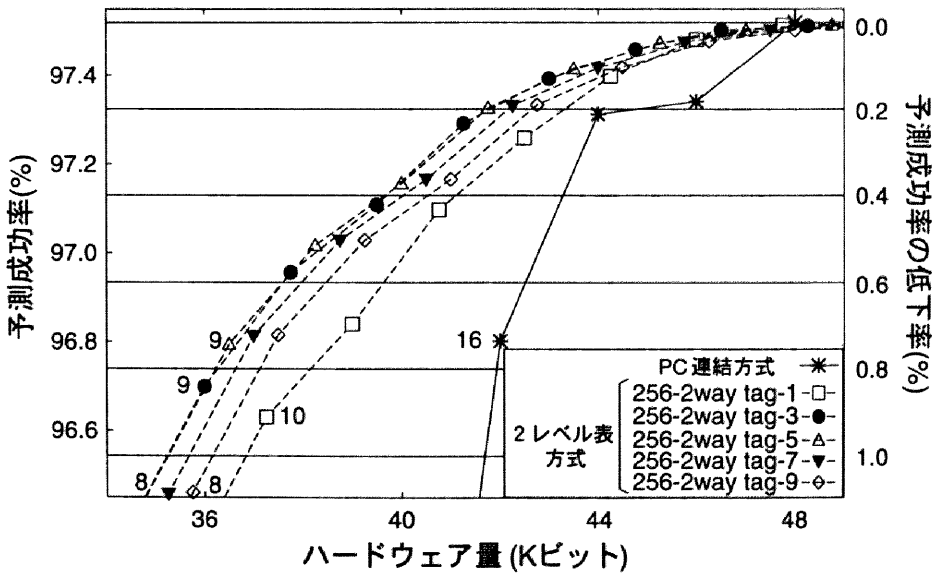
図 3.10 は第 2 テーブルの連想度が 2 の場合である．(a) は第 2 テーブルのエントリ数を 128 とした場合，(b) は 256 とした場合である．図 3.11 は第 2 テーブルの連想度が 4 の場合である．(a) は第 2 テーブルのエントリ数を 128 とした場合，(b) は 256 とした場合である．折れ線グラフの種類を示す *X*-*Y* way tag-*Z* とは第 2 テーブルのエントリ数 *X*，連想度 *Y*，タグ・ビット数 *Z* であることを意味する．図の右の縦軸の予測成功率の低下率は，Fagin の削減手法を用いた従来方式（タグ・ビット数は図 3.9 より定めた 5 ）と比べた割合を示す．

図 3.10 および図 3.11 より，タグ部を含めた全体ハードウェア量で比較しても，2 レベル表方式は従来方式と同等の予測成功率を PC 連結方式より少ないハードウェア量で実現できることがわかる．図 3.10 より，第 2 テーブルの連想度が 2 の場合において第 2 テーブルの各タグ・ビット数の場合を比較すると，エントリ数が 128 でも 256 でもタグ・ビット数が 3 あるいは 5 の場合が同等の予測成功率を少ないハードウェア量で実現できることがわかる．図 3.11 より，第 2 テーブルの連想度が 4 の場合では，タグ・ビット数が 5 あるいは 7 の場合が同等の予測成功率を少ないハードウェア量で実現できることがわかる．

図 3.12 に第 2 テーブルの連想度が 2 および 4 の場合の結果を共に示す．第 2 テーブルのタグ・ビット数は，図 3.10 および図 3.11 より，削減量の多い 5 の場合のみ

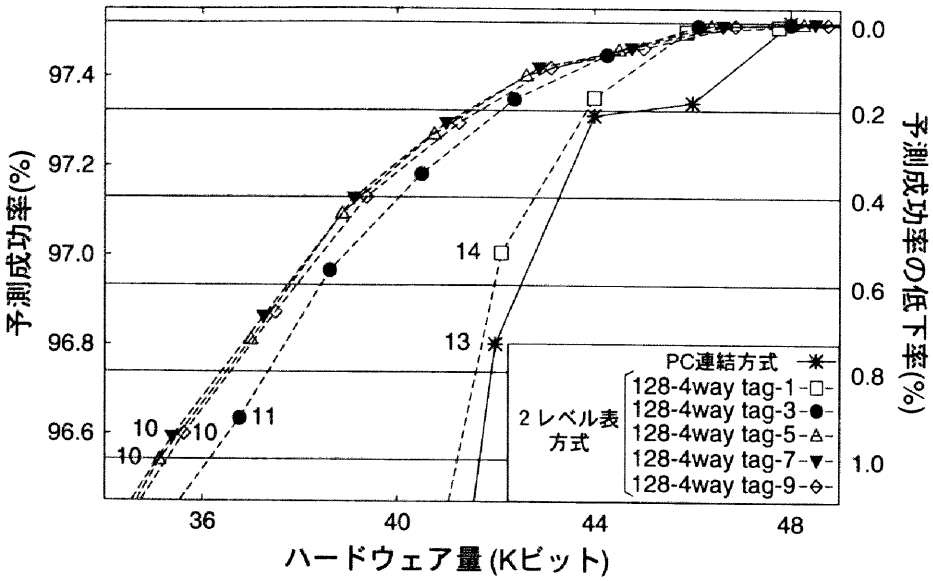


(a) 第2テーブルのエントリ数 128

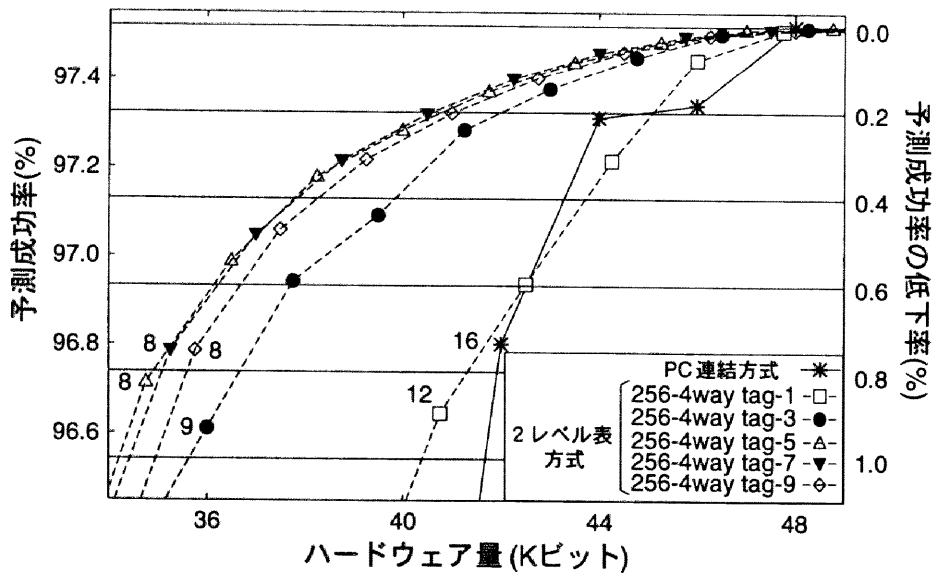


(b) 第2テーブルのエントリ数 256

図 3.10　タグ部を削減した場合の全体ハードウェア量と予測成功率
(第2テーブルの連想度2の場合)



(a) 第2テーブルのエントリ数 128



(b) 第2テーブルのエントリ数 256

図 3.11　タグ部を削減した場合の全体ハードウェア量と予測成功率
(第2テーブルの連想度4の場合)

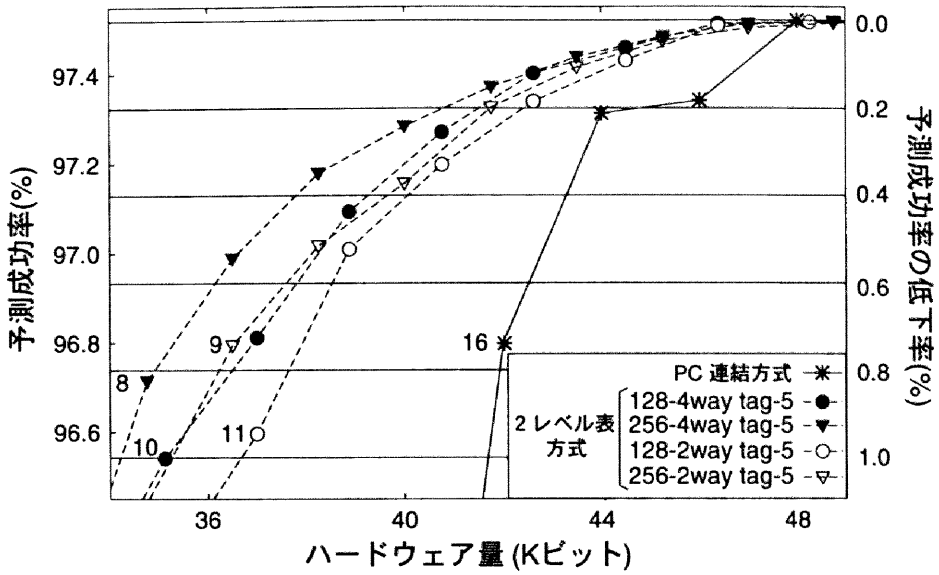


図 3.12　タグ部を削減した場合の全体ハードウェア量と予測成功率 (まとめ)

を示す．第 2 テーブルの連想度が 2 でも 4 でも第 2 テーブルのエントリ数は 256 の場合が同等の予測成功率を最も少ないハードウェア量で実現できるといえる．以上より，最適な構成は，第 1 テーブルの分岐先アドレス部ビット数 13，第 2 テーブルのエントリ数 256，連想度 4 であることがわかった．このとき，従来方式に比べ予測成功率をほとんど低下させることなく（低下率 0.1%）ハードウェア量を約 38%（70.0K ビットから 43.5K ビット）削減できる．

4　スレッド・レベル並列処理マルチプロセッサ・アーキテクチャ SKY

本章では，**スレッドレベルの並列性 (TLP: Thread-Level Parallelism)** と ILP を最大限利用する **SKY** と呼ぶマルチプロセッサ・アーキテクチャについて議論する．まず，4.1 節において，マルチプロセッサ・アーキテクチャについて述べる．4.2 節ではこの研究の背景について述べる．4.3 節で SKY のマルチスレッド実行モデルについて述べる．4.4 節で SKY アーキテクチャの提案を行う．4.5 節で性能を評価する．4.6 節で関連研究について述べる．

4.1　はじめに

これまでスーパスカラ・プロセッサは，命令の発行幅を広げ，より多くの並列性を利用することで，性能向上を図ってきた．しかし今後さらに命令発行幅を広げても，それによって得られる性能向上は小さいと考えられる．この理由は以下の 2 点である．

- 命令発行幅を広げるとハードウェアの複雑度が増加し，サイクル時間に大きな悪影響を与える [14),33)]．
- 現在のスーパスカラ・プロセッサが引き出している ILP は，単一制御流において利用可能な ILP の限界に近づきつつある [22),44)]．

スーパスカラ・プロセッサに代わるアプローチとして，最近，複数のプロセッサを単一チップに集積するマルチプロセッサの研究が行われている [18),32),39)~42)]．その背景には，上記スーパスカラ・プロセッサに対する悲観的観測に加え，半導体集積回路技術の進歩に伴い，複数のプロセッサの単一チップへの集積化が現実的になりつつあることがある．単一チップ・マルチプロセッサの長所として，以下の 3 点をあげることができる．

- ILP に加えて，TLP を利用することができる．
- 命令幅を広げるなどプロセッサの複雑度を増加させる必要がないので，サイ

クル時間への悪影響を抑えることができる.

- チップ上に相互結合網を構成することで，従来のマルチプロセッサに比べ通信レイテンシを減少させることができる.

これらにより，単一チップ・マルチプロセッサはスーパスカラ・プロセッサを越える性能を達成できる可能性がある.

これまでマルチプロセッサの応用として，多くの場合，数値計算プログラムが想定されてきた．この理由の1つは，複数のプロセッサを単一チップに集積することがこれまで不可能であったことから，マルチプロセッサが非常に多くの計算量を必要とするが，コスト対性能比に寛容な分野に適するという点である．これは一般に，数値計算の分野である．もう1つの理由は，数値計算プログラムは一般に，豊富な並列性を内在しており，マルチプロセッサを比較的容易に利用することができる点である.

これに対して，マイクロプロセッサの多くの応用は，非数値計算プログラムである．著者は，その非数値計算プログラムにおいて，従来のスーパスカラ・プロセッサでは達成することができない高い性能を発揮するマルチプロセッサの実現に向けて研究を行っている．非数値計算プログラムでは，数値計算プログラムと異なり，不規則な制御構造と複雑なデータ依存関係が数多く存在するため，粒度の大きなTLPを利用することは難しく，粒度の小さなTLPを利用することが不可欠である．細粒度のTLPでは，スレッドあたりの計算量が少ないので，スレッドを並列に実行するためのオーバヘッドを小さくすることが強く求められる．そのオーバヘッドのなかでも同期／通信のオーバヘッドの削減は，特に重要である.

一般にマルチプロセッサでは，同期／通信はメモリを介して行う．これを**メモリ・ベースの同期／通信**と呼ぶこととする．これに対して，単一チップ・マルチプロセッサでは，集積化という利点を生かし，レジスタ値を直接通信し，同期を行う機構が提案されている[2),18),39)]．これを**レジスタ・ベースの同期／通信**と呼ぶこととする．この機構はメモリ・アクセス回数を削減することができるので，メモリ・ベースの同期／通信機構に比べてオーバヘッドを大幅に減少させることができる[18)]．

これまで提案されたレジスタ・ベースの同期／通信機構は，低オーバヘッドを実現しているものの，同期が成立しなかった命令が現れたとき，その命令が受信待ちで停止するのに加え，それに後続する命令の実行も停止するという問題がある．このことを著者は，**同期による命令実行のブロッキング**と呼ぶ．後続命令の中には，受信待ちの命令とは独立の命令がある可能性がある．このような命令は，受信待ち命令とは無関係に実行可能である．したがって，ブロッキングは不効率である．ブロッキングが生じると，スレッド内の ILP の利用が妨げられ，性能が低下する．非数値計算プログラムにおいては，スレッド内の ILP は並列性の重要な源なので，命令実行のブロッキングは性能に対して大きな影響を与えると考えられる．

本章では，スレッド内の ILP 利用を阻害することなく細粒度の TLP を利用することを目的とした SKY[19)-21)] と呼ぶ単一チップ・マルチプロセッサのアーキテクチャを提案する．SKY は，リング・バスで結合された複数のスーパスカラ・プロセッサからなる．細粒度の TLP を利用するため，プロセッサ間でレジスタ値を直接通信し，同期／通信のオーバヘッドを 2 サイクルまで減少させている．また，**命令ウィンドウ・ベースの同期**と呼ぶ新しい同期機構を提案し，これを導入している．この機構は，命令ウィンドウ上で受信値に対応するタグを用いてレジスタに関する同期をとる機構である．この機構により，同期による命令実行のブロッキングが生じることはなく，プロセッサは ILP を最大限利用することができる．

4.2　背景

プログラムに内在する並列性を調べた研究が数多く存在する（たとえば，文献 3),22),44)）．その中でも，Lam らの研究[22)] は複数のスレッド実行から得られる並列性について重要なことを示唆している．この研究によれば，並列性が低いと考えられている非数値計算プログラムにおいても，基本ブロック・レベルで制御依存関係を詳細に解析し，複数の制御流の命令を同時に実行すれば，単一制御流の命令だけを実行する場合に比べて，多くの並列性を引き出すことができるとしている．これは，スレッド内の ILP とともに細粒度の TLP を利用すれば，その様なマシンの性能はスーパスカラ・プロセッサの限界を大幅に越える可能性があることを示唆して

いる．

TLP を利用する一般的な方法は，マルチプロセッサによるスレッド並列実行である．単一チップ・マルチプロセッサとスーパスカラ・プロセッサの性能を比較した研究として，Olukotun らの研究がある[32]．この研究では，命令発行幅が狭く小規模なスーパスカラ・プロセッサを複数結合したマルチプロセッサと，それとほぼ同一のチップ面積で，命令発行幅が広く規模の大きな単一のスーパスカラ・プロセッサの性能を比較している．マルチプロセッサの同期／通信機構は，従来のメモリ・ベースのものである．この研究によれば，彼らが評価に用いたプログラムの中で，並列化困難な非数値計算プログラムにおいては，単一のスーパスカラ・プロセッサに比べて単一チップ・マルチプロセッサは，10 ～30% 性能が低いという測定結果を得ている．しかし彼らは，マルチプロセッサは単純なプロセッサにより構成されているので，高いクロック速度での動作が期待でき，その結果，この程度の性能差はなくなると推測している．

単一チップ集積の利点を生かし，レジスタ・ベースの同期／通信機構を提案した研究として，**マルチスカラ・プロセッサ**[2),39)] と**マルチ ALU プロセッサ**（MAP）[18)] の研究がある．これらで提案された機構は，プロセッサ間でレジスタ値を直接通信し，full/empty ビット[36)] を用いてレジスタに関する同期をとるものである[4]．full/empty ビットはレジスタごとに存在し，対応するレジスタに格納された値が利用可能かどうかを示す．Keckler らは，共有のオンチップ・キャッシュを持つ 3 プロセッサ構成の MAP アーキテクチャにおいて，レジスタ・ベースの同期／通信機構を用いれば，メモリ・ベースの機構を用いる場合に比べて，オーバヘッドを大幅に小さくできることを示した．

これらの研究において提案されたレジスタ・ベースの同期／通信機構は，オーバヘッドを小さくできるという長所を持つが，4.1 節で述べたように，同期による命令実行のブロッキングという欠点を持つ．つまり，デコードした命令のソース・オペランドに対応する full/empty ビットが empty であった場合，その命令は full/empty

4. マルチスカラ・プロセッサでは，accum マスクと recv マスクという 2 種類のビットで実現されているが，機能は full/empty ビットと同じである．

ビットが full になるまで後続の命令のパイプライン処理を停止させてしまうという欠点を持っている．

4.3　SKY のマルチスレッド・モデル

スレッド並列実行のためのオーバヘッドを小さくするため，SKY におけるマルチスレッド・モデルは通常のモデルと比べて，次に示す制約を課している．これは，同じく非数値計算プログラムの TLP を利用する目的を持つアーキテクチャである，マルチスカラや **MUSCAT** [40),41)] と同様のモデルである．

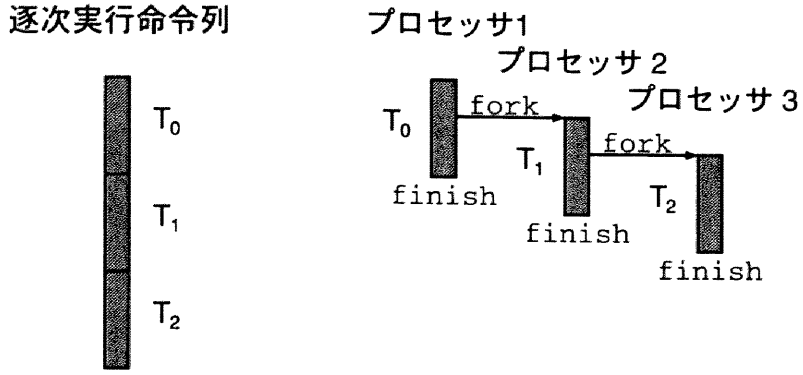
- 1 つのスレッドは，逐次実行における動的に連続する部分で構成される．
- 各スレッドは，逐次実行の順における自身の直後のスレッドを逐次生成する．

図 4.1(a) に，逐次実行命令列に対する SKY のスレッド分割を示す．同図に示すように，SKY における各スレッドは，動的な命令列における単一の連続した部分からなり，異なる複数の部分からは構成されない．したがって，スレッドに結合はなく，制御に関する同期は必要ない．

図 4.1(a) に示したスレッドを並列に実行するようすを，図 4.1(b) に示す．図 4.1(a) において，各スレッドに T_0，T_1，T_2 と逐次実行の順に名前をつける．図 4.1(b) に示すように，スレッド T_i は実行途中で，スレッド T_{i+1} を生成するという逐次生成を繰り返す．各スレッドは実行中には高々1 回しか新しいスレッドを生成しない．実行のできるだけ早い時期に新しいスレッドを生成することにより，スレッドがオーバラップして実行される．スレッドの生成は，`fork` と呼ぶ専用の命令を用い，終了は `finish` と呼ぶ専用の命令を用いて行う．これらの命令については，4.4.4 項で詳しく述べる．以下，T_{i+1} を T_i の**子スレッド**と呼び，T_i を T_{i+1} の**親スレッド**と呼ぶ．

4.4　SKY アーキテクチャ

本節では，SKY アーキテクチャについて述べる．最初に，ハードウェアの構成に



(a) スレッド分割　　(b) スレッド生成

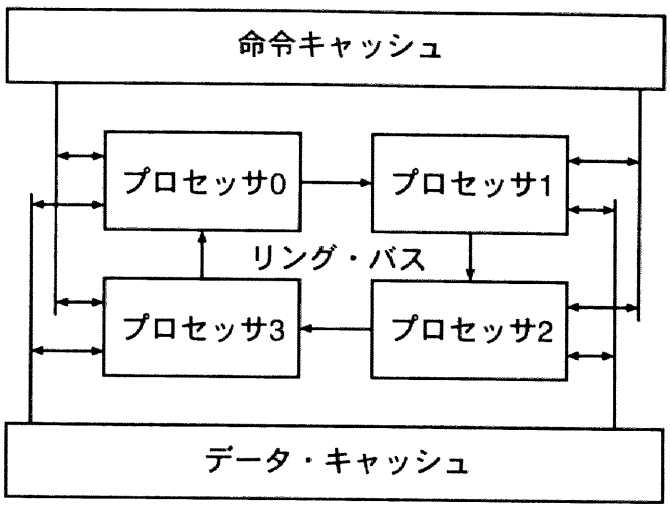
図 4.1　SKY のマルチスレッド・モデル

ついて述べる．その後，SKY の特徴である同期／通信機構について説明する．さらに，SKY 用のコンパイラについて述べ，最後に SKY 専用命令の詳細について述べる．

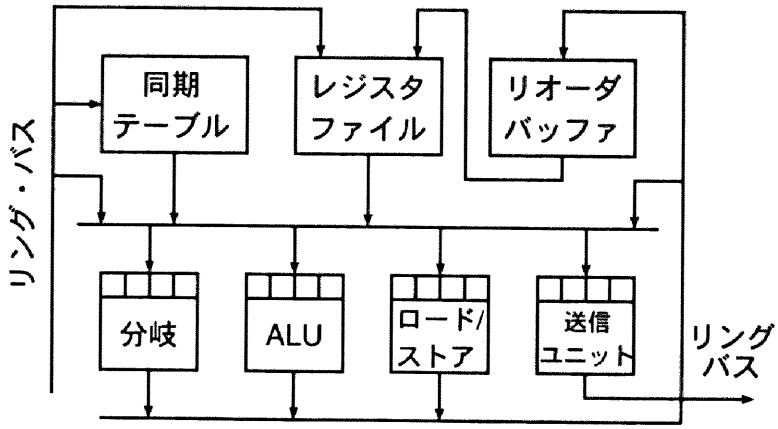
4.4.1　ハードウェア構成

図 4.2(a) に示すように，SKY は複数のスーパスカラ・プロセッサからなる．プロセッサは単方向のリング・パスで結合する．同図に示すように，プロセッサ i はプロセッサ $(i+1)\%n$ にのみデータを送り，プロセッサ $(i-1)\%n$ からのみデータを受け取る（n はプロセッサ台数で，% はモジュロ演算子を示す）．以下，プロセッサ $(i+1)\%n$ をプロセッサ i の**後続プロセッサ**と呼び，プロセッサ $(i-1)\%n$ をプロセッサ i の**先行プロセッサ**と呼ぶ．

スレッドは実行を開始してから終了するまで，1 つのプロセッサを占有する．あるプロセッサ上のスレッドは `fork` 命令を実行することによって，後続プロセッサに子スレッドを生成する．ただし，後続プロセッサが他のスレッドによって占有されているなら，`fork` 命令は無効化され，子スレッドは生成されない．4.3 節で示した SKY のスレッドの性質より，`fork` 命令を無効化しても，スレッドに分割され実行されるべきところが逐次に実行されるだけで，プログラムの意味は保持され



(a) 構成



(b) プロセッサ

図 4.2　SKY のアーキテクチャ

る．

キャッシュはプロセッサ間で共有する．SKY は少ない数のプロセッサで構成することを想定しているため，キャッシュのポート数は実現上容認できると考える．

図 4.2(b) にプロセッサ内部の構成を示す．通常のスーパスカラ・プロセッサと異なる点は，レジスタの同期／通信を支援する機構を持つ点である．レジスタの送信

は，send と呼ぶ専用の命令を送信ユニットで実行することにより行う．send 命令は，コンパイラによって挿入される．これについては 4.4.3 項で述べる．

一方，受信はハードウェアによって暗黙的に行う．このハードウェアの中心をなすものが，**同期テーブル**と呼ぶテーブルである．このテーブルの各エントリはレジスタに対応し，当該プロセッサがそのレジスタの値を受信すべきかどうかについての情報と，受信値を識別するタグを保持する．命令はレジスタ・フェッチの際，同期テーブルを参照し，参照オペランドがこれから受信する値かどうかを判断する．受信値がオペランドの場合，同期テーブルからその受信値に対応するタグを読み出し，命令ウィンドウで受信を待ち合わせる．一方，先行プロセッサからレジスタ値が送られてくると，プロセッサはそれをレジスタ・ファイルに書き込むとともに，同期テーブルより対応するタグを読み出し，命令ウィンドウに放送する．命令ウィンドウで受信を待ち合わせている命令は，放送されたタグと自身の持つタグが一致する場合，その受信値を受け取り，同期が完了する．

上記は，最も典型的な受信の動作を示したものであるが，実際には種々の場合に対応し動作が異なる．次項では，この説明を含め，同期／通信の機構の詳細について述べる．

4.4.2　同期／通信の機構

SKY の同期／通信機構は，以下の 2 つの特徴を持っている．

- 同期／通信のオーバヘッドを 2 サイクルにまで減少させている．
- ノンブロッキングの同期を実現している．

これらの特徴は，従来の単一チップ・マルチプロセッサの研究で提案あるいは論じられている同期／通信の高速化，すなわち，

- 単一チップ化による物理的な高速化[27)]
- メモリを介さずレジスタ値を直接送受信することによるアーキテクチャ上の高速化[18),39)]

といった事項に加え，新しく命令ウィンドウ上でレジスタに対する同期を行う機構を導入したことによるものである．この機構を著者は，命令ウィンドウ・ベースの同期機構と呼ぶ．

以下，命令ウィンドウ・ベースの同期機構について説明する．まず，本機構の説明を容易にするために，投機的実行を行わないプロセッサにおいて，本機構の基本的動作を説明する．次に，投機的実行を行うプロセッサに対応できるように，機構を修正する．

非投機的実行プロセッサにおける実現

最初に機構について説明し，その後，簡単な例を用いて動作を説明する．

a) 機構

同期テーブルについて説明する．同期テーブルの各エントリはレジスタに対応し，受信値を識別するタグと**R フラグ**（receive フラグの略）と呼ぶフラグを保持する．R フラグは，当該エントリに対応するレジスタ値を受信するかどうかを示す．R フラグが 1 ならば受信し，0 ならば受信しない．

先行プロセッサよりレジスタ値が送られてきた場合，そのレジスタに対応する同期テーブルのエントリを読み出す．読み出されたエントリの R フラグが 1 ならば，当該プロセッサはそのレジスタ値を受信する．受信においては，送られてきた値をレジスタ・ファイルに書き込むとともに，同期テーブルより読み出されたタグを値に付けて命令ウィンドウの全エントリに放送する．これにより，命令ウィンドウで受信値を待ち合わせている命令に対し，レジスタ・ファイルを介さず値を渡すことができる．

読み出されたエントリの R フラグが 0 ならば，受信動作は行わず，送信されてきた値は捨てられる．ここで，R フラグが 0 であっても，先行プロセッサから値が送信されて来る場合があることを注意しておく．前述のようにレジスタの送信はコンパイラがプログラム中に send 命令を挿入することによって実現しているが，静的解析の性質上コンパイラは，スレッド間の可能性のある全てのデータ依存について

send 命令を挿入するからである．実行時には，静的に存在するデータ依存の一部がオペランドの定義と参照の関係として現れるので，受信する必要のないレジスタの送信が行われることがある．

同期テーブルの R フラグは，スレッドがプロセッサに生成されたときに，全て 1 にセットする．あるエントリの R フラグは，そのエントリに対応するレジスタの値を受信するか，あるいは，スレッド内で値が命令により定義されたならば，0 にリセットされる．R フラグを参照することによって，命令が参照するオペランドについて，次の 3 つの場合を判断することができる．

(1) これから受信する値

(2) すでに受信された値

(3) 自身のプロセッサにおいて定義された値

(2) または (3) の場合，通常のスーパスカラ・プロセッサと同様の方法で，値またはそのタグを得，命令は命令ウィンドウに発行される．(1) の場合，同期テーブルよりタグを得，同じく命令ウィンドウに発行される．従来の full/empty ビットによる同期機構では，受信待ちの命令はレジスタ・フェッチのステージで停止し，後続の命令の実行をブロックするが，著者の機構では，命令は命令ウィンドウで受信値を待ち合わせるので，後続の命令の実行をブロックすることはない．

b) 簡単な例

同期／通信における典型的な動作を例を用いて説明し，著者の機構では同期／通信のオーバヘッドが 2 サイクルであること，また，同期がノンブロッキングであることを示す．

2 つのプロセッサにおける次の場合を考える．プロセッサ 1 では，命令 i1 と i2 を実行する．i1 はレジスタ r1 を生成する演算命令であり，i2 は i1 の実行結果 r1 をプロセッサ 2 に送信する send 命令である．プロセッサ 2 では，命令 i3 と i4 を実行する．i3 は i1 の実行結果 r1 を参照する命令であり，i4 はどの命令にも依存しない独立な演算命令とする．

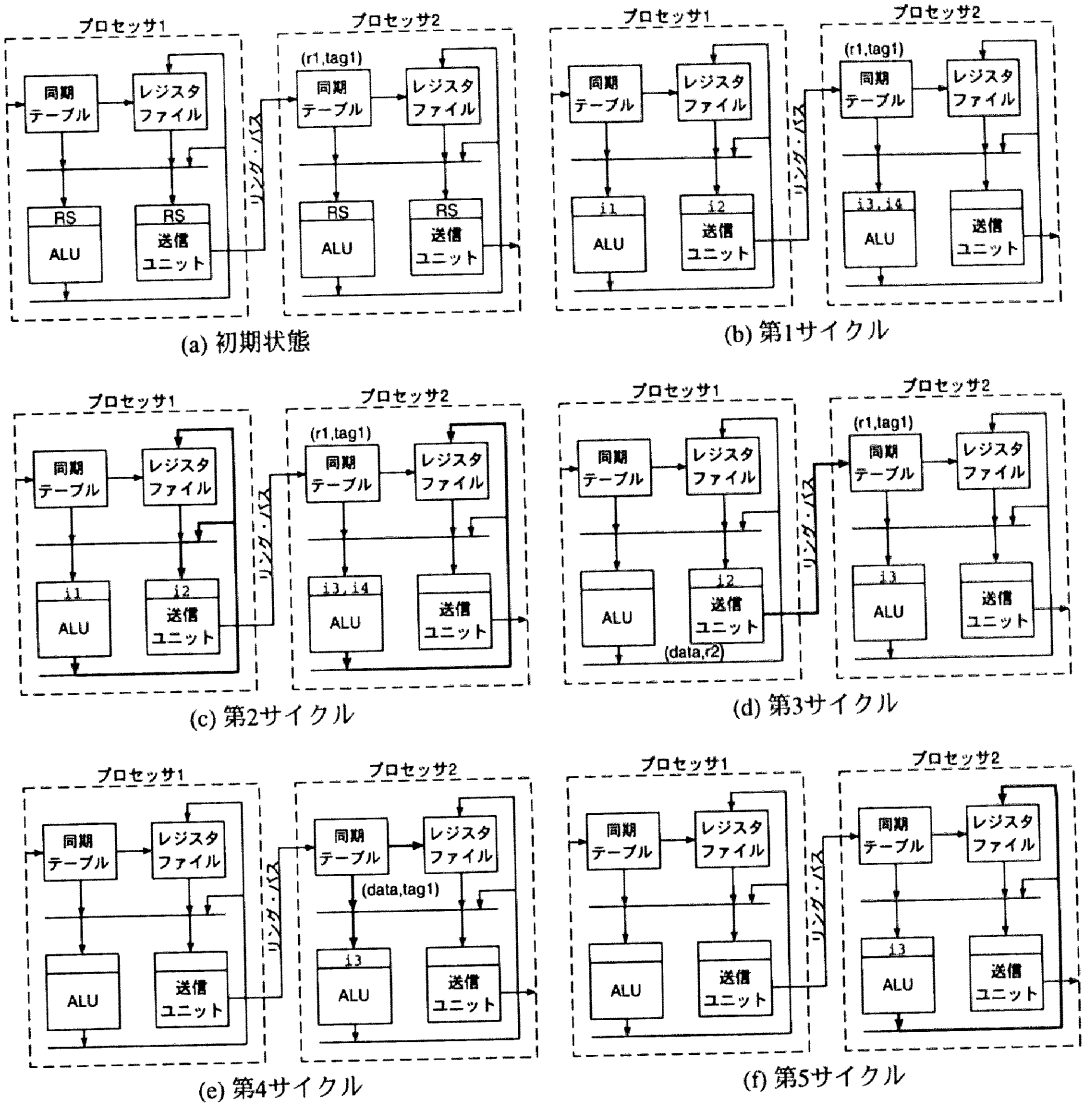


図 4.3　同期／通信の例

プロセッサのパイプラインは，命令フェッチ，命令デコード，実行，書き戻しの４ステージで構成されているとする．図 4.3(a)～(f) に実行の過程をサイクルごとに示す．図において，点線で囲んだ部分は１つのプロセッサを表す．プロセッサの内部は簡略化し，ここでの説明に関係する部分のみ示す．RS はリザベーション・ステーションを表す．

図 4.3(a) は最初の状態である．プロセッサ２の同期テーブルに (r1, tag1) とある

のは，レジスタ r1 に対応するエントリの R フラグがセットされており，タグとして tag1 を保持していることを示す．これ以外のエントリの R フラグは，プロセッサ 1 の同期テーブルを含めて全てリセットされているとする．このサイクルでは，プロセッサ 1 は命令 i1 と i2 を，プロセッサ 2 は命令 i3 と i4 をフェッチしている．

第 1 サイクル：プロセッサ 1 において，命令 i1 と i2 が同期テーブルを参照し，ソース・レジスタに対応するエントリを読み出す．読み出されたエントリの R フラグは全て 0 なので，どのソース・レジスタも受信する値ではないことがわかる．この場合，通常のスーパスカラ・プロセッサと同一の手順で，命令はリザベーション・ステーションに発行される．

プロセッサ 2 においても，命令 i3 と i4 が同期テーブルを参照し，ソース・レジスタに対応するエントリを読み出す．命令 i3 のソース・レジスタ r1 に対応する R フラグは 1 なので，r1 は受信する値であることがわかる．この場合，同期テーブルより得られるタグ tag1 を r1 に付け，命令 i3 はリザベーション・ステーションに発行される．命令 i4 については，ソース・レジスタに対応する R フラグは 0 なので，通常のスーパスカラ・プロセッサと同一の手順で発行される．

第 2 サイクル：プロセッサ 1 において，命令 i1 が実行される．その結果は，レジスタ・ファイルに送られると同時に，送信ユニットのリザベーション・ステーションで待っている send 命令 i2 にフォワーディングされる．図で太く強調した線は，結果が送られていくパスを示す（以下の説明でも同様）．同様にプロセッサ 2 においては，命令 i4 が実行される．

第 3 サイクル：プロセッサ 1 において，send 命令 i2 が送信ユニットで実行され，レジスタ r1 の値とレジスタ番号が，リング・バスを介してプロセッサ 2 に送信される．プロセッサ 2 では，送信されてきたレジスタ番号をインデクスとして同期テーブルを参照し，対応する R フラグとタグを読み出す．

第 4 サイクル：プロセッサ 2 は，前サイクルで読み出した R フラグの値が 1 なので，受信動作を行う．まず，送信されてきたレジスタ r1 の値をレジスタ・ファイ

ルに書き込み，R フラグをリセットする．同時に，r1 の値と読み出したタグ tag1 を，リザベーション・ステーションの全てのエントリに放送する．リザベーション・ステーションで待ち合わせていた命令 i3 は，自身の保持するタグと放送されたタグが一致するので，値を受け取る．

第5サイクル：命令 i3 はオペランドがそろったので実行される．

以上からわかるように，命令 i1 と i3 の間に発生する同期／通信のオーバヘッドは，第3サイクルと第4サイクルの2サイクルである．また，同期が成立していない命令 i3 は，リザベーション・ステーションで受信値を待ち合わせることにより，後続する命令 i4 の実行を停止させることはない．つまり，ノンブロッキングの同期を実現している．

投機的実行プロセッサにおける実現

投機的に命令を実行するプロセッサでは投機に失敗した場合，一般に，投機的実行結果を全て取り消しプロセッサ状態を戻した上で，分岐予測を誤った点からプログラムの実行を再開する．非投機的実行プロセッサにおいて説明した機構では，投機を失敗したとき，プロセッサ状態に影響を与える動作で，取り消すことのできないものが存在する．それは，受信に関する動作である．この動作には，送信値を受け取る動作と受け取らない破棄という動作がある．前者については，受信により書き込まれたレジスタを再実行時に上書きするので，プログラムの意味は保たれ，問題ない．これに対して破棄の場合は，プログラムの意味を保つことができなくなる．これを図4.4に示す例で説明する．

図4.4において，ブロック D にある命令 i1 に注目する．命令 i1 を含むスレッド（以下，現スレッドと呼ぶ）は，命令 i5 から始まるスレッドであり，その親スレッドは命令 i4 で終了するスレッドである．命令 i1 のソース・レジスタ r1 は，親スレッドの命令 i2，あるいは，現スレッドの i6 によって定義される．どちらの定義を参照するかは，実行時のブロック B からの制御移行に依存する．もしも命令 i5 の分岐が成立せず，制御がブロック C を経由せず D に移行すれば，命令 i1 のレジスタ r1 は，命令 i2 によって定義された値である．命令 i5 の分岐

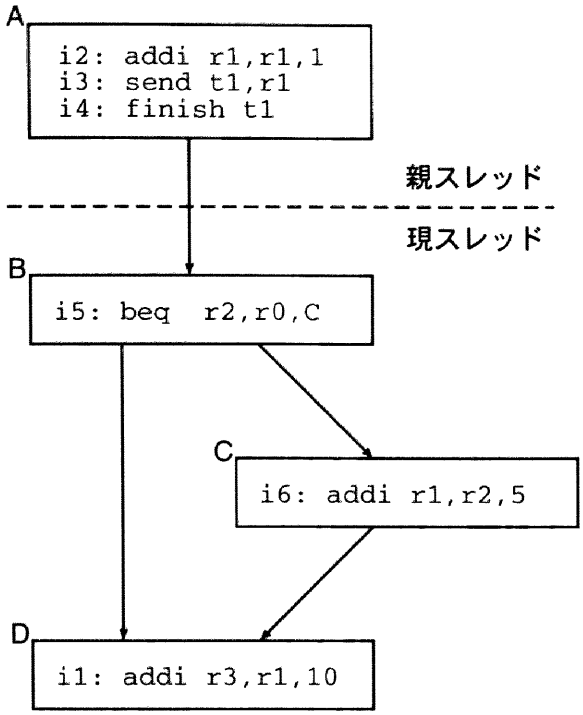


図4.4　プログラム例

が成立し，制御がBからCを経由してDに移行すれば，i6によって定義された値である．

ここで，命令i5の分岐は成立と予測され，命令i6が投機的に実行された場合を考える．この実行により，レジスタr1に対応するRフラグはリセットされる．これにより，命令i1はr1に関して先行プロセッサから送られてくる値を待つのではなく，命令i6の実行結果をオペランドとすることを認識する．このとき，先行プロセッサで命令i3が実行され，命令i2が定義したレジスタr1の値が送信されてきたとする．r1に対応するRフラグはリセットされているので，この値は受信されず破棄される．

ここで投機が失敗したとする．命令i6とi1の投機的実行結果は破棄され，ブロックBから正しい方向（ブロックDの方向）に向かって実行が再開される．これにより命令i1が再実行される．この場合，命令i1が参照すべきソース・レジスタr1のオペランドは，命令i2が定義した値である．しかしこの値は，投機的

実行時に送信されてきたが，受信されず破棄されている．そのため命令 i1 はこれを得ることができない．

この問題に対処するために，オペランド・フェッチと受信の動作において，次の2点を変更する．

1. R フラグは，対応するレジスタ値を受信したときか，あるいは，そのレジスタを定義した命令がリオーダ・バッファからリタイアするときにリセットする．

2. オペランド・フェッチにおいては，リオーダ・バッファ検索の結果，ソース・レジスタを定義する命令が見つかったときは，そのソース・レジスタの値またはタグを，R フラグの値にかかわらずフェッチする．

第1の修正は，投機の失敗時においても，親スレッドからの送信値を使用することができるようにするための措置である．定義時，つまり，実行終了したときではなく，リオーダ・バッファからリタイアするときに R フラグをリセットする点が，非投機的実行プロセッサの場合の実現方法と異なる．第2の修正は，第1の修正に対応して，オペランドあるいはそのタグを正しくフェッチするための措置である．R フラグの参照に加えて，リオーダ・バッファの検索結果を判断に加えた点が異なる．オペランド・フェッチの動作を，表 4.1 にまとめる．

以上の修正により，オペランド・フェッチと受信が投機的実行を行うプロセッサにおいても正しく行われることを説明する．いま，ある命令 i1 がソース・レジスタ r1 をフェッチしようとしている場合について考える（図 4.4 の例ではなく一般的な場合について考えていることに注意）．r1 に対応する R フラグとリオーダ・バッファの参照結果に応じて，以下の3つ場合が存在する．

1. R フラグが 0 の場合

   この場合，r1 をすでに受信したか，あるいは，r1 を定義する命令が i1 の前に少なくとも1つ存在し，すでにリタイアしていることを示している．いずれの場合も，必要とする r1 の値あるいはそのタグは，レジスタ・ファイルかリ

表 4.1　オペランド・フェッチ

ソース・レジスタを定義する命令がリオーダ・バッファにあるか?	Rフラグ	
	0	1
ない	レジスタ・ファイルより値を得る	同期テーブルよりタグを得る
ある	リオーダ・バッファより値またはタグを得る	

オーダ・バッファの中に存在する．これは，通常のスーパスカラ・プロセッサで行われている手順によって得ることができる．よって，表 4.1 の R フラグが 0 の場合の動作は正しい．

また，r1 を定義する命令のリタイアによって R フラグがリセットされた場合において，先行プロセッサから r1 の値が送られてくることがあるが，この場合，R フラグは 0 なので，それは破棄される．したがって，r1 が不正に上書きされることはなく，プログラムの意味は保たれる．

2. R フラグが 1 であり，かつ，リオーダ・バッファに r1 を定義する命令が存在しない場合

   R フラグが 1 ということより，r1 は受信しておらず，かつ，r1 を定義する命令があったとしても，まだリタイアしていないことがわかる．後者はこの場合あてはまらないので，結局，r1 を受信していない場合である．したがってこの場合，同期テーブルより r1 に対応するタグを得て，リザベーション・ステーションで受信を待つ必要があり，表 4.1 の該当する場合の動作は正しい．

   リオーダ・バッファ内の命令はプログラム順にリタイアし，現在リオーダ・バッファに r1 を定義する命令はないから，r1 に対応する R フラグは，r1 の値を受信するまでリセットされることはない．つまり，送信されてくる値を i1 が受け取ることは保証されている．

3. R フラグが 1 であり，かつ，リオーダ・バッファに r1 を定義する命令が存在する場合

リオーダ・バッファに r1 を定義する命令が存在するので，命令 i1 が参照すべき値は，リオーダ・バッファ内の r1 に関する最新の定義である．これは，リオーダ・バッファより得られ，表 4.1 の該当する場合の動作は正しい．

R フラグは 1 なので，リオーダ・バッファ内の r1 を定義する命令がリタイアするまでに，先行プロセッサから値が送られてくると，これを受信し，レジスタ・ファイルに書き込まれる．これがプログラムの意味を変えることはないことを，次の 3 点によって示す．

- r1 に対応する R フラグが 1 であるということは，現スレッドの命令によってレジスタ r1 に値が書き込まれたことは，1 度もないことを示している．つまりレジスタ r1 は現スレッドにおいては「空」である．
- 先行プロセッサから送られてくる r1 の値は，現スレッドに先行するいずれかのスレッドにおいて，r1 に対する最後の定義値である（詳細は 4.4.3 項で述べる）．
- 現プロセッサで空のレジスタが過去のスレッドでの最終値を持つようになることは，SKY のスレッドが逐次的実行における 1 部分であり，スレッドは逐次に生成されるという性質より正当である．

リオーダ・バッファ内の r1 を定義する命令は，まだ投機的状態である場合とすでにコミットされている状態（制御依存がすべて解消されている状態）という 2 つの場合がある．すでにコミットされている場合，受信したレジスタは，リオーダ・バッファ内命令によって上書きされ，プログラムの意味は保たれる．投機的状態であったが，投機に成功しコミットされた場合も同様である．一方，投機に失敗した場合，図 4.4 で示したように，命令 i1 が参照すべき値が受信値となることがある．受信値はレジスタ・ファイルに存在するので，再実行の際にはこれを得ることができ，プログラムの意味は保たれる．

以上，オペランド・フェッチと受信が，投機的実行を行うプロセッサにおいても正しく行われることを示した．

ここで述べた方法以外の解決法として，投機的実行時の送信記録を残し，先行プロ

セッサに送信を要求する方法が考えられる．この方法では，あるプロセッサで送信するレジスタは，後続のプロセッサの投機が完了し，送信を要求されることがないとわかるまで，別の命令による上書きは許されず，保持しておかなければならない．このような制御は複雑であるばかりでなく，新たに逆依存を生じさせる結果となり，性能が低下する．別の方法として，投機的実行時に送られてきた値を，別途バッファを用意し保持する方法が考えられる．この方法は実現は簡単であるが，そのようなバッファを必要とせず，制御も同程度に簡単な本機構の方が優れている．

4.4.3　コンパイラの概要

プログラムの分割は，コンパイラが行う[15),31)]．コンパイラは分割の後，スレッド間で送信すべきレジスタを求め，それを送信する命令をプログラムに挿入する．本項ではまず，プログラム分割について述べ，次にレジスタ送信命令の挿入について述べる．

プログラム分割

コンパイラは最初に，プログラム分割の候補を求める．次に，各候補についてスレッド並列実行による性能利得を計算し，利得が大きいものを採用する．本項ではまず，プログラム分割の候補の求め方について述べる．次に，スレッドの並列実行による性能利得の計算方法を述べる．

a) プログラム分割の候補

SKY においてプログラム分割とは，**フォーク点**と**子の開始点**を定めることである．フォーク点とは，スレッドを新たに生成するプログラム上の位置であり，子の開始点とは子スレッドが実行を開始する位置である．フォーク点に `fork` 命令を挿入し，子の開始点の直前に `finish` 命令を挿入する．

フォーク点と子の開始点を含む基本ブロックをそれぞれ X，Y とする．このとき，実行を開始したスレッドの制御は，その親スレッドの制御に依存してはならないので，Y は X を**後支配**[38)]しなければならない．ここで，制御フロー・グラフ[1)]において，X から出口ノードに向かうどのパスも Y を通るとき，Y は X を後支配する

という．

理想的にはＹがＸを後支配する全ての組を，プログラム分割の候補とすることが望まれる．しかし，そのような組の数はプログラム中に非常に多く存在するので，プログラムのサイズが大きくなると爆発的に増え，その結果，分割に必要な計算量が膨大となる．現実的な計算量で分割するためには何らかの条件を設け，プログラム分割の候補を少なくする必要がある．

そこでＹがＸを後支配する組の中でも，ＸがＹを**支配**[1)]する特別な組，つまり**制御等価**[38)]な組をプログラム分割の候補とすることとした．一般に，制御フロー・グラフにおいて，入口ノードからＹへ向かうどのパスもＸを通るとき，ＸがＹを支配するという．またＸがＹを支配し，同時にＹがＸを後支配するとき，組(X,Y)は制御等価であるという．Ｙが Ｘを後支配する組の数よりも，制御等価な組の数の方が少ないので，分割の際の計算量をかなり削減することができる．

b) スレッドの並列実行による性能利得

あるスレッドとその子スレッドの並列実行による**性能利得**を，２つのスレッドの実行がオーバラップする間に子スレッドで実行された命令の数と定義する．コンパイラは，フォーク点から子の開始点までの**距離**，および，データ依存関係にある命令間の距離を求め，その最小値を性能利得の推定値とする．ここで点 p_1 から点 p_2 までの距離とは，p_1 から p_2 に至る各パス上の命令数の，パスの実行確率による重み付き平均値とする．パスの実行確率は，分岐のプロファイルから得る．

レジスタ送信命令の挿入

4.3 節で述べたようにSKYでは，逐次実行の連続する一部分であるスレッドを逐次に生成していく．生成順にスレッドに番号を付け，$i < j$ のとき，T_i は T_j より後方にあるといい，T_j は T_i より前方にあるということにする．この表現を使えば，SKYではスレッド内のどの命令も，データに関して，そのスレッドより後方に位置するスレッド内の命令に依存する可能性はあるが，前方に位置するスレッド内の命令に依存することはない．したがって，あるスレッドが `send` 命令で送信しなけ

ればならないレジスタは，現スレッドで定義され，その前方のいずれかのスレッドで使用される可能性のあるレジスタである．

スレッドは（生成されるとすればそのときは）常に後続プロセッサに生成される．また，プロセッサは単方向リングで結合されているので，スレッド T_i から T_j への送信において，$j = i + 1$ の場合は，単に後続プロセッサに送信すればよい．$j > i + 1$ の場合，直接の通信路はないので，T_i と T_j の間のスレッド T_k （$i < k < j$）を経由して送信する必要がある．SKY では送信はすべて命令により明示しなければならないので，T_k では値の転送についても send 命令を挿入しなければならない．

以上まとめると，あるスレッドが send 命令によって送信しなければならないレジスタは，次の 2 つに大別できる．

1. **真の送信レジスタ**：現スレッドで定義され，その前方のいずれかのスレッドで使用される可能性のあるレジスタ
2. **転送レジスタ**：現スレッドの後方のいずれかのスレッドで定義され，その前方のいずれかのスレッドで使用されるレジスタ

どちらの場合についても，あるスレッドにおいて，子スレッドに送信しなければならないレジスタは，その前方のいずれかのスレッドで使用されるレジスタである．つまり，子スレッドの開始点で**生きている（live）**[1] レジスタを送信しなければならない．

実際に send 命令を挿入する点は，送るべきレジスタ値が子の開始点に**到達**[1] することが決定する点である．ここで，レジスタ値 r が点 p に到達するとは，r が p または p の後方のどこかで参照される可能性があることをいう．真の送信レジスタについては，スレッド内のデータフロー解析を行い，レジスタ値の到達に関する計算を行う必要がある．転送レジスタについては，送るべきレジスタ値が子スレッドの開始点に到達することは，フォーク点においてすでに決定しているので，到達が決定する点を求める必要はない．現在の著者のコンパイラは，フォーク点の直後に send 命令を挿入している．

図 4.5(a) に示すプログラム例を用いて，send 命令の挿入について説明する．このプログラムでは，i1 ～i7 と i10 と i11 が 1 つのスレッド（以下，現スレッドと呼ぶ）を構成し，i8 以後がその子スレッドを構成する．現スレッドは fork 命令 i10 によって子スレッドを生成し，finish 命令 i11 によって終了する[5]．レジスタ r2 ～r4 のみが，子スレッドの開始点に到達するとする．以下，これらのレジスタを送信する send 命令の挿入について説明する．

図 4.5(b) に send 命令挿入後のプログラムを示す．レジスタ r2 は，転送レジスタである．よって，フォーク命令 i10 の直後に，これを送信する命令 i12 を挿入する．レジスタ r3 は，真の送信レジスタである．r3 を定義する命令は i1 だけなので，i1 の直後に r3 を送信する命令 i13 を挿入する．レジスタ r4 も真の送信レジスタであるが，これを定義する命令は i2 と i4 の 2 つがある．どちらの定義が子の開始点に到達するかは，分岐 i3 によって決まる．つまり，i3 の分岐が成立するなら i2 が到達し，成立しないなら i4 が到達する．したがって，レジスタ r4 を送信する命令は，i3 の分岐先であるラベル L1 の地点（命令 i15 ）と，命令 i4 の直後（命令 i14 ）に，それぞれ挿入する．

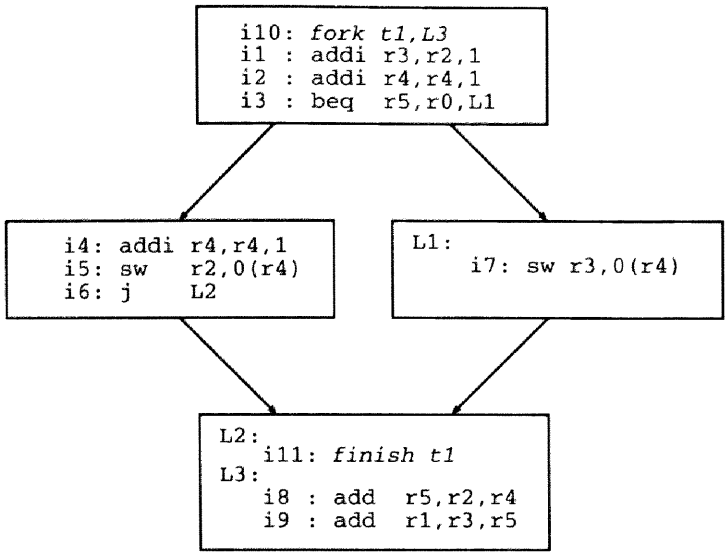
4.4.4　SKY 専用命令の詳細

SKY 専用命令はすでに述べたように，fork 命令，finish 命令，send 命令の 3 つである．各スレッドはその親スレッドの制御に依存してはならないので，これらの専用命令は投機的に実行せず，スレッド内における制御依存が解消した後に実行する．各命令の形式は以下の通りである．

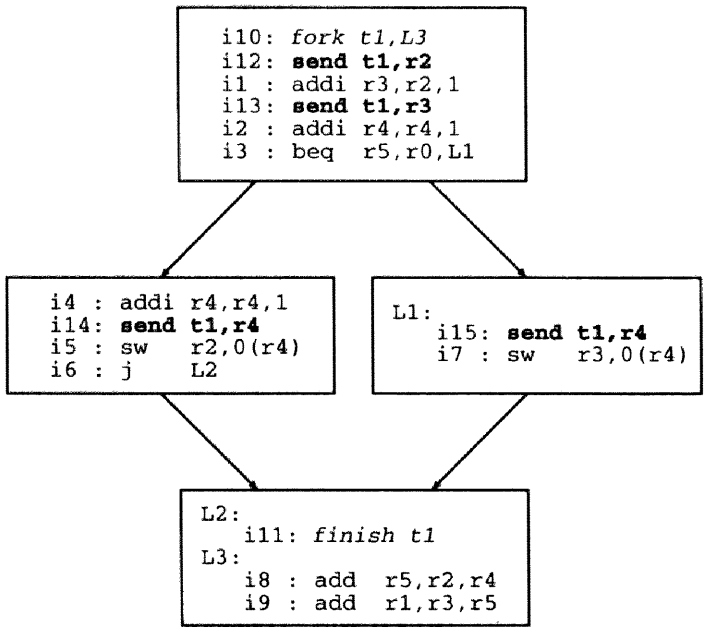
```
fork    tid,offset
finish  tid
send    tid,reg
```

以下，各命令について説明する．

5. fork 命令と finish 命令の形式，および，これから挿入する send 命令の形式については，4.4.4 項で述べる．



(a) send 命令挿入前



(b) send 命令挿入後

図 4.5　send 命令の挿入

fork 命令

fork 命令は，後続プロセッサが空いていれば，そこに子スレッドを生成する．具体的には，自身の命令アドレスと命令にエンコードされている offset を加え，後続プロセッサの PC に書き込む．同時に，自身のプロセッサの **TID レジスタ**と呼ぶ専用のレジスタに，tid を書き込む．レイテンシは 2 サイクルである．最初のサイクルで子スレッドの開始アドレスを計算し，次のサイクルで PC と TID への書き込みを行う．

TID（Thread Identifier）は，子スレッドに与えられる識別子である．あるスレッドに対しコンパイラが挿入する send 命令と finish 命令は，そのスレッドから生成されるある特定の子スレッドに対応して挿入されるものである．一方，子スレッドが生成されるかどうかは，前述のように，fork 命令が実行される際の後続プロセッサの空き状況による．したがって，実行中に現れた全ての send 命令と finish 命令を実行してよいわけではなく，実際に実行された fork 命令によって生成された子スレッドに対応する send 命令と finish 命令だけを実行する必要がある．TID はこれを認識するためのものである．

finish 命令

コンパイラは finish 命令挿入の際，対応するスレッドの TID を命令にエンコードする．実行時は，TID レジスタの内容と tid を比較し，一致すれば現スレッドを停止する．そうでなければ無効化される．

finish 命令は，命令フェッチを停止し，後続の命令を全て無効化する．そして，その finish 命令がリオーダ・バッファの先頭に来るまで待った後，現プロセッサを解放する．リオーダ・バッファ内で待つことによって，finish 命令に先行する全ての命令がリタイアすることを保証する．finish 命令のレイテンシは，命令フェッチを停止し，後続の命令を全て無効化するための 1 サイクルとリオーダ・バッファ内で待機していたサイクルを加えたものである．

send 命令

finish 命令と同様に，コンパイラは send 命令挿入の際に，対応するスレッドの TID を命令にエンコードする．実行時は，TID の内容と tid を比較し，一致すれば reg の値と番号を後続プロセッサに送る．そうでなければ無効化される．レイテンシは，4.4.2 項で述べたように 2 サイクルである．

4.5　性能評価

本節ではまず，評価環境について述べる．次に，プロセッサ数を変化させたときの SKY の性能を評価し，何が性能に大きな影響を与えているについて議論する．また，ほぼ同一規模のスーパスカラ・プロセッサとの性能比較も併せて行う．次に，命令ウィンドウ・ベースの同期機構の有効性を評価し，その後，同期／通信オーバヘッドの性能への影響を評価する．最後に，専用命令の挿入によるコードの増加について述べる．

4.5.1　評価環境

ベンチマーク・プログラムとして，SPECint95 の全 8 種を使用した．表 4.2 に各ベンチマークにおける入力セットを示す．同表に示すように，異なる 2 つの入力セットを用いた．1 つは，プログラムのプロファイルを採取するためのものであり，もう 1 つは，シミュレーションを行うためのものである．ベンチマーク・プログラムは NEC EWS 4800/360AD（MIPS R4400）のコンパイラで最適化し，MIPS R2000[17] 用のコードを生成し，これを測定に用いた．評価はトレース駆動シミュレーションにより行った．トレースは pixie[37] を用いて採取した．

表 4.3 に，SKY の基本モデルを示す．以下特に断らない限り，評価においてはこの基本モデルを使用する．性能比較における基準プロセッサは，SKY を構成する 1 つのプロセッサ，つまり 8 命令発行の単一のスーパスカラ・プロセッサとする．表 4.4 に，各ベンチマーク・プログラムにおけるこの基準プロセッサの IPC を示す．

表 4.2　ベンチマーク・プログラム

ベンチマーク	プロファイル用トレース		シミュレーション用トレース	
	入力セット	命令数	入力セット	命令数
compress95	train/test.in	42M	ref/bigtest.in (reduced to 30K elements)	113M
gcc	ref/regclass.i	141M	ref/genoutput.i	99M
go	11x11 board, level 4, ref/null.in	106M	9x9 board, level 6, train/2stone9.in	75M
ijpeg	train/vigo.ppm, 68x48 pixcels	92M	ref/specmun.ppm	437M
li	test/test.lsp	474K	train/train.lsp	187M
m88ksim	test/dhry	499M	train/dcrand	121M
perl	train/jumble.pl, train/jumble.in, train/dictionary	61M	train/scrabbl.pl, train/scrabbl.in (add three words), train/dictionary	89M
vortex	ref/persons.1k, ref/vortex.in (reduced iterations)	136M	ref/persons.1k, ref/vortex.in	89M

4.5.2　性能

図 4.6 に性能の評価結果を示す．縦軸は，基準プロセッサに対する性能向上率である．プロセッサ数が 2 と 4 の場合について性能を測定した．また，比較のため，基準プロセッサの 2 倍の命令発行幅を持ち，2 倍の資源（例えば，機能ユニット，命令ウィンドウのエントリ数など）を投入した 16 命令発行の単一のスーパスカラ・プロセッサの性能向上率についても測定した．このプロセッサのハードウェア量は，2 プロセッサ構成の SKY のハードウェア量と同程度である．

16 命令発行のスーパスカラ・プロセッサ

基準プロセッサの 2 倍の資源を投入した 16 命令発行の単一のスーパスカラ・プロセッサの性能向上率は，最大で 14.4%，平均で 7.8% しかない．このことは，ILP のみで性能向上を図ることは難しいことを示している．

2 プロセッサ構成の SKY

2 プロセッサ構成の SKY は，最大で 46.1%，平均で 21.8% の性能向上率を達成している．特に，compress95，ijpeg，m88ksim では，性能向上率は 30% 以上と大き

表 4.3　SKY の基本モデル

(a) プロセッサ

フェッチ幅	8命令
発行幅	8命令
命令ウィンドウ	64エントリ
リオーダ・バッファ	128エントリ
機能ユニット	8つの整数演算ALU, 8つの浮動小数点演算ALU, 4つのロード/ストア・ユニット, 2つの分岐ユニット, 4つの送信ユニット
分岐予測機構	深さ32のリターン・スタック[16]

(b) 共有資源

命令キャッシュ	2ポート, 各ポート8命令, 常にヒット
データ・キャッシュ	4ポート, 各ポート1つのデータ・アクセス, 常にヒット
メモリあいまい性検出機構	理想
分岐予測機構	1024エントリ, 連想度2のBTB, 1024エントリ, 履歴長4のPAp[49]
分岐予測ミスペナルティ	4サイクル

表 4.4　8 命令発行スーパスカラ・プロセッサの性能

ベンチマーク	IPC
compress95	2.73
gcc	2.07
go	1.84
ijpeg	3.63
li	2.55
m88ksim	2.04
perl	2.25
vortex	2.57

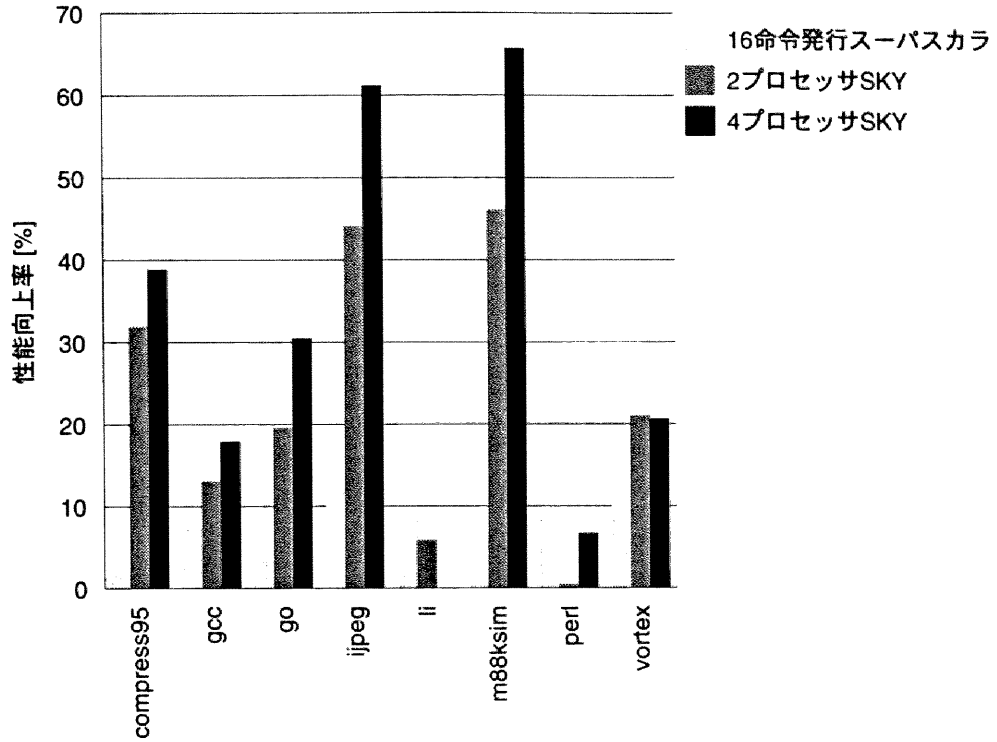


図 4.6　基準スーパスカラ・プロセッサに対する性能向上

い．この性能向上率は，同程度のハードウェア量を投入した 16 命令発行のスーパスカラ・プロセッサより最大で 37.3%，平均で 13.0% 高い．SKY を構成する 8 命令発行のスーパスカラ・プロセッサは，16 命令発行のものより大幅に単純なので，実現可能なサイクル時間には大きな差が生じる[33]．したがって，実際の性能差は IPC の差よりさらに大きく広がると考えられる．

SKY は以上述べたように，大きな性能向上を達成しているものの，プログラムによって性能向上率に大きな違いが見られる．ijpeg，m88ksim，compress95 では大きな性能向上を達成しているものの，逆に li と perl ではほとんど性能が向上していない．これらでは，16 命令発行のスーパスカラ・プロセッサより性能が低いという結果を得ている．

解析の結果，このような性能差が生じる原因には次の 2 つがあることがわかった．1 つは，コンパイラがプログラムから多くの TLP を抽出できるかどうかという点である．抽出できればスレッド並列実行により性能は向上し，そうでなければ，ス

レッドが並列に実行される状況が少なく，性能向上は押さえられる．もう1つの原因は，スレッド並列実行がスレッド内のILP利用に悪影響を与えるかどうかという点である．SKYは，同期については，命令ウィンドウ・ベースの機構によりILP利用が妨げられないよう工夫を施しているが，この他にもILP利用に大きな影響を及ぼす事項があることが評価によりわかった．以下この2点について考察する．

a) TLPの抽出

まず，コンパイラによるTLPの抽出について考察する．このために，次の2つの原因により，1サイクル当たりに命令発行スロットが平均でいくつ「空き」となったかを測定した．

- スレッド不在
- スレッド間データ依存

ここで，**空きスロット**とは，次のようなものである．1つのプロセッサの同時発行命令数は最大8である．これを，「このプロセッサは8つの発行スロットがある」ということとする．2プロセッサ構成のSKYの同時発行命令数は全部で最大16であるから，スロット数は16である．これらのスロットは実行時においては，全て有効な命令によって埋められるわけではなく，データ依存など種々の原因で埋められないことがある．命令を埋められなかったスロットのことを，空きスロットと呼ぶ．

上記2つの原因の内，スレッド不在で多くの空きスロット数が生じるのは，コンパイラがプログラムよりプロセッサ数に見合うだけの十分なTLPを抽出することができなかった場合である．スレッド間データ依存で多くの空きスロットが生じるのは，コンパイラが何らかの理由で誤ってTLPの少ないスレッド分割を行ってしまった場合である．

図4.7に評価結果を示す．縦軸は，上記2つの原因によるサイクル当たりの平均空きスロット数である．各棒グラフの下の部分は，スレッド不在による空きスロット数であり，上の部分は，スレッド間依存によるものである．同図よりまずわかるこ

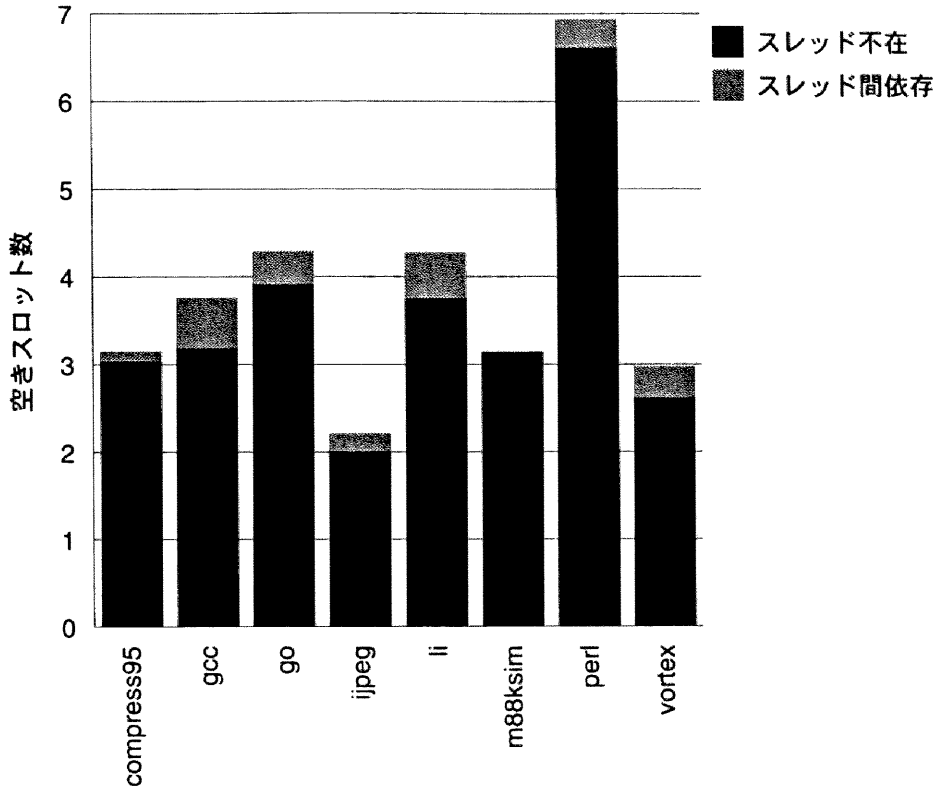


図4.7　スレッド不在とスレッド間依存によるサイクル当たりの空きスロット数
（2 プロセッサ SKY）

とは，スレッド間依存による空きスロット数は非常に少ないということである．このことよりコンパイラは，TLP の少ないスレッドへの分割は行っていないことがわかる．

次にわかることは，perl ではスレッド不在による空きスロットが非常に多いということである．perl では，全スロットのほぼ半数が，スレッド不在で空いている．つまり，ほとんどの時間 2 つのプロセッサの一方でしかスレッドは実行されていない．これが，perl で性能向上がほとんどない原因である．逆に，ijpeg ではスレッド不在による空きスロットが 2 程度と少なく，大きな性能向上を達成している原因であることがわかる．

コンパイラが十分な TLP を抽出できなかった理由は，3 つ考えられる．1 つは，著者の現在のコンパイラの能力不足である．たとえば，次の点において能力に不足が

ある．

- コンパイラは，制御等価なブロックの組に着目している．これは制御依存がないブロックの組の一部であり，解析範囲が制限されている．
- 関数ごとに解析を行っている．このため，フォーク点を含む関数の外に子スレッドの開始点を設けることはできない．

前者は，論理的には後支配の組に対し解析を行うことで解決できる．ただし，4.4.3 項で述べたように，後支配の組は制御等価の組に比べて非常に多いので，コンパイラを実用的なものにするには，計算時間を短縮する工夫が必要である．後者は，関数を展開することで解決できる．ただし，コード量が増加するので，このトレードオフを考慮した方法が必要である．これらについては，今後検討していく予定である．

コンパイラが十分な TLP を抽出できなかった 2 つめの理由は，4.3 節で述べた SKY のマルチスレッド実行における制約である．特に，スレッドの逐次生成の制約は単純化に貢献しているが，TLP の抽出には大きな制約を課している．例えば，次のようなコードを考える．

```
        fork  t1, L1;
        foo();
L1:     ...
```

このコードは，関数 foo と L1 以下のコードの間に存在する TLP を利用するものであるが，foo 内に豊富な TLP が存在しても，それを利用することができない．制約を緩和すればこの問題は解消できるが，マルチスレッド実行の複雑さが増加し，そのオーバヘッドとのトレードオフが存在する．

3 つめの理由は，プログラム中に内在する TLP がそもそも十分には存在しないということである．存在しない TLP をコンパイラは引き出すことはできない．これを証明することは非常に難しいが，今後調査の必要がある．

b) マルチスレッド実行による ILP 利用への悪影響

perl と ijpeg 以外のプログラムにおいては，スレッド不在による空きスロット数は比較的近い値を示している．しかし，図 4.6 に示したように性能向上率に大きな差が見られる．これは調査の結果，マルチスレッド実行が ILP 利用に対して悪影響を与えており，その影響の大きさによることがわかった．マルチスレッド実行が原因で ILP の利用が妨げられたとき，これを ILP を損失したということとする．ILP 損失の原因として，次の 2 つがある．

- スレッドの実行開始と終了時の命令不足
- 分岐予測精度の低下

スレッドの実行開始と終了時には，十分な命令がプロセッサに供給されないために，ILP の量が定常状態に比べて低下する．SKY では，スレッドの最初の命令がフェッチされ実行を開始するまで 3 サイクルかかり，この間プロセッサは実行する命令がない．また，`finish` 命令が発行された後，リオーダ・バッファからリタイアする十数サイクルの間発行する命令がない．このような期間は単一のスーパスカラ・プロセッサにはなく，マルチスレッド実行により生じたものであるから，ILP 損失ということができる．これらの期間のサイクル数はあまり大きくないように感じられるが，TLP の少ない非数値計算プログラムでは性能に対して大きな影響を与える．

また，マルチスレッド実行によって分岐予測精度が低下することもわかった．SKY では，分岐予測機構はプロセッサ間で共有している．静的に一つの分岐が異なるスレッドで同時に実行される場合，PAp 予測器の**分岐履歴表**（BHT: Branch History Table）には，逐次実行とは異なる順で履歴が記録される．これは例えば，ループの各イタレーションをスレッドとし，これらを同時に実行する場合，そのループに含まれる分岐に対して生じる．このような場合，逐次実行において存在した履歴パターンと将来の分岐方向の間の相関を，PAp 予測器はとらえることができなくなり，予測を誤る確率が高まる．予測を誤る確率が高まると，パイプラインへの命令供給が阻害され，ILP の利用が妨げられる．

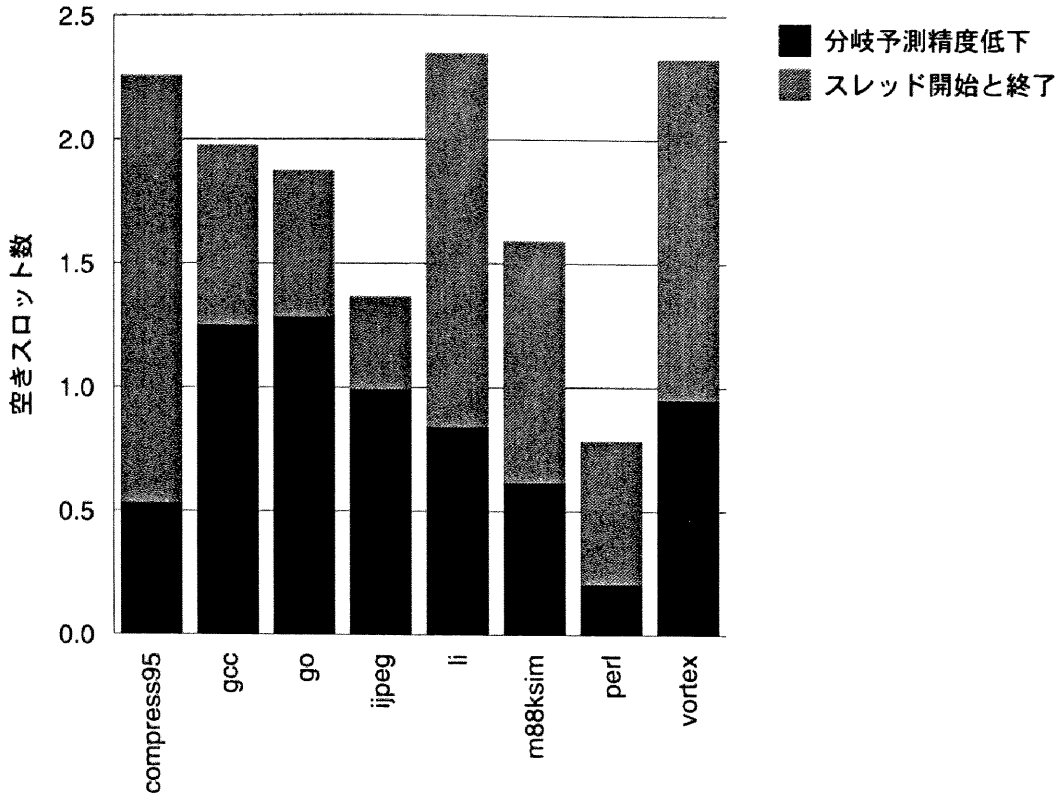


図 4.8　マルチスレッド実行によるサイクル当たりの空きスロット数

図 4.8 に，これら 2 つの原因によって生じたサイクル当たりの平均空きスロット数を示す．各棒グラフの下の部分は，分岐予測精度の低下により生じた空きスロット数であり，上の部分は，スレッドの実行開始と終了により生じた空きスロットである．同図よりわかるように，TLP の利用が少ない perl を除いて，1.5 ～2.5 ものスロットがマルチスレッド実行によって空いてしまっている．一般に，スレッド・サイズ（後に表 4.6 で示す）が小さいプログラムほど，スレッドの開始と終了による空きスロット数が大きい．これは，そういったプログラムほどスレッドの開始と終了の頻度が高いからである．一方，分岐予測精度の低下は，プログラムをスレッドにどのように分割するかによる．この解析はプログラムが大きく複雑なため，プログラムとの明確な関係をいうことは難しい．

図 4.9 に，マルチスレッド実行による ILP 損失と性能向上の間の相関を示す．横軸

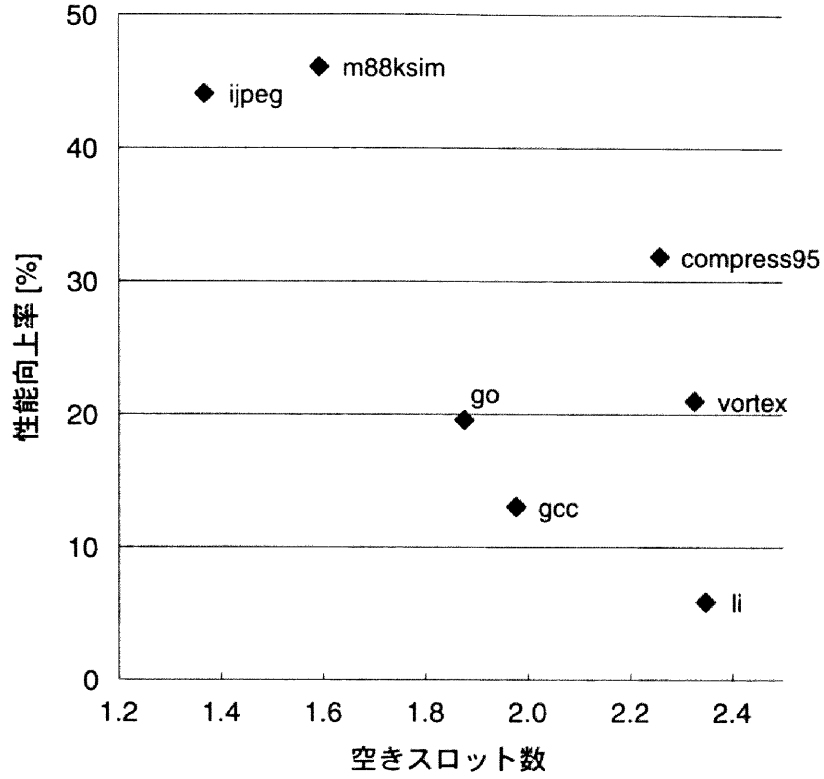


図 4.9　マルチスレッド実行による ILP 損失と性能向上率の相関

は ILP 損失による空きスロット数であり，縦軸は性能向上率である．TLP 利用の少ない perl については，この議論の対象外なのでプロットしていない．同図より，空きスロット数が多いほど性能向上率が低いという相関があることがわかる．ベンチマーク別に見ると，図 4.7 より li は TLP を比較的多く利用しているといえるが，ILP 損失が大きく，その結果，性能が向上していない．逆に go は，li と同程度 TLP を利用しているが，ILP 損失が少ないので，li より高い性能向上率を達成している．また，li と同程度の ILP 損失であっても，compress95 と vortex はより多くの TLP を利用しているので，li より高い性能向上率を達成している．m88ksim と ijpeg は，TLP 利用量が大きく，かつ，ILP 損失が少ないので，他のプログラムに比べ大きな性能向上を達成している．

スレッド実行開始と終了による ILP 損失を抑える 1 つの方法として，1 つのプロセッサ内において，`fork` 命令による新しいスレッドの開始と `finish` 命令による

スレッドの終了をオーバラップさせることが考えられる．現在プロセッサの解放は，finish 命令がリオーダ・バッファからリタイアするまで行われないが，これを finish 命令実行完了時に解放するようにすれば，新しいスレッドをそのプロセッサで開始することができる．このようにスレッドの開始と終了をオーバラップさせることができれば，開始と終了時の命令供給不足をある程度回避することができると思われる．これを実現するには，何らかのハードウェア支援が必要であり，今後の課題である．

一方，マルチスレッド実行による分岐予測精度の低下を抑えるのは，かなり困難な問題である．なぜならば，現在知られている分岐予測機構は全て，逐次実行における分岐の過去の履歴と将来の分岐方向との間の相関を利用しているが，マルチスレッド実行では，分岐が逐次に実行されないので，この相関を利用することができないからである．

4 プロセッサ構成の SKY

4 プロセッサ構成の SKY は基準プロセッサに対して，最大で 65.7%，平均で 21.8% の性能向上率を達成している（図 4.6）．しかし，2 プロセッサ構成の SKY に対して最大でも 13.4% しか性能が向上しておらず，コスト性能比が良いとはいえない．これはスレッド不在が主因である．図 4.10 に，スレッド不在とスレッド間依存によるサイクル当たりの空きスロット数を示す．どのプログラムでも，2 プロセッサ分のスロットである 16 に近いスロットが，スレッド不在で空きとなっている．2 プロセッサを越えるプロセッサを効率良く利用するには，TLP 抽出に関してさらなる検討が必要である．

4.5.3　同期機構

本項では，次の 2 つの有効性に関して同期機構を評価する．

- 同期の粒度が全レジスタでなく 1 レジスタであること
- ノンブロッキング同期

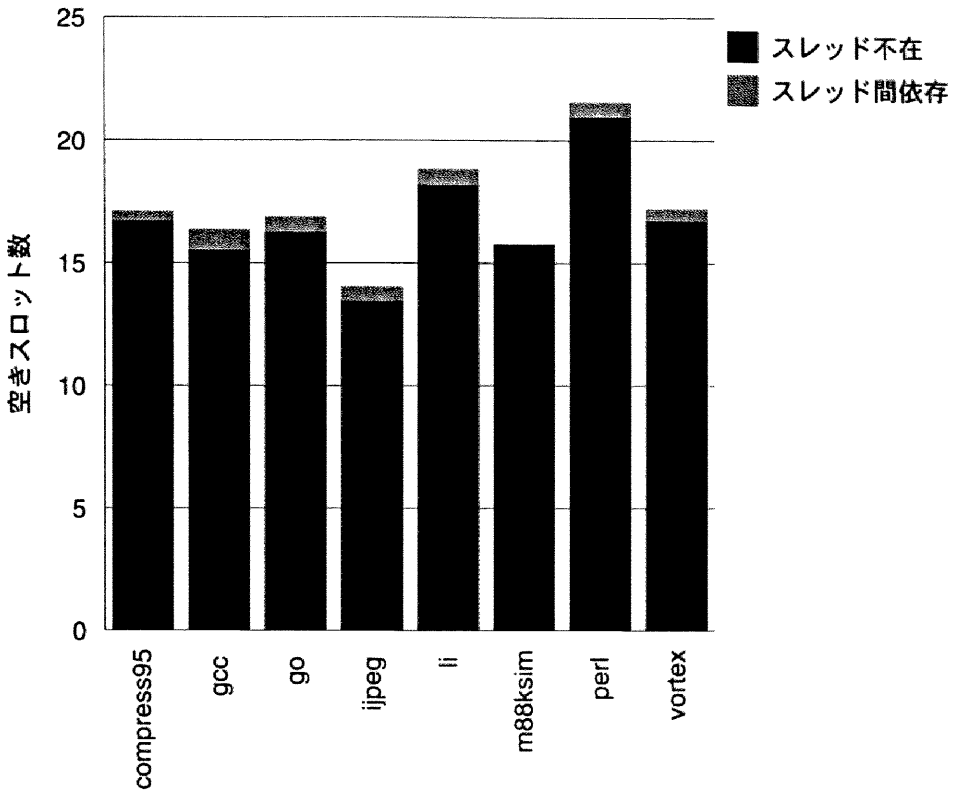


図 4.10　スレッド不在とスレッド間依存によるサイクル当たりの空きスロット数
（4 プロセッサ SKY）

2 プロセッサ構成の SKY を基本とし，以下の 3 つの同期機構を実現したモデルについて評価した．

C モデル（coarse-grain model）：このモデルでは，子スレッドの開始点に到達するレジスタ値が現スレッドにおいて全て定義されるまで子スレッドを生成しない．子スレッドを生成するときに，レジスタ・ファイルの内容を全て後続プロセッサにコピーする．レジスタ転送のバンド幅は十分にあるとし，コピーには 1 サイクルしかかからないと仮定した．これにより，一旦スレッドが生成されると，レジスタに関する同期は必要ない．このため，個々のレジスタについての同期／通信機構が必要なく単純である．しかし，通信データの粒度を全レジスタにまで粗くしたため，同期に無駄なオーバヘッドが生じている．

B モデル（blocking model）：このモデルでは，SKY と同様，個々のレジスタについ

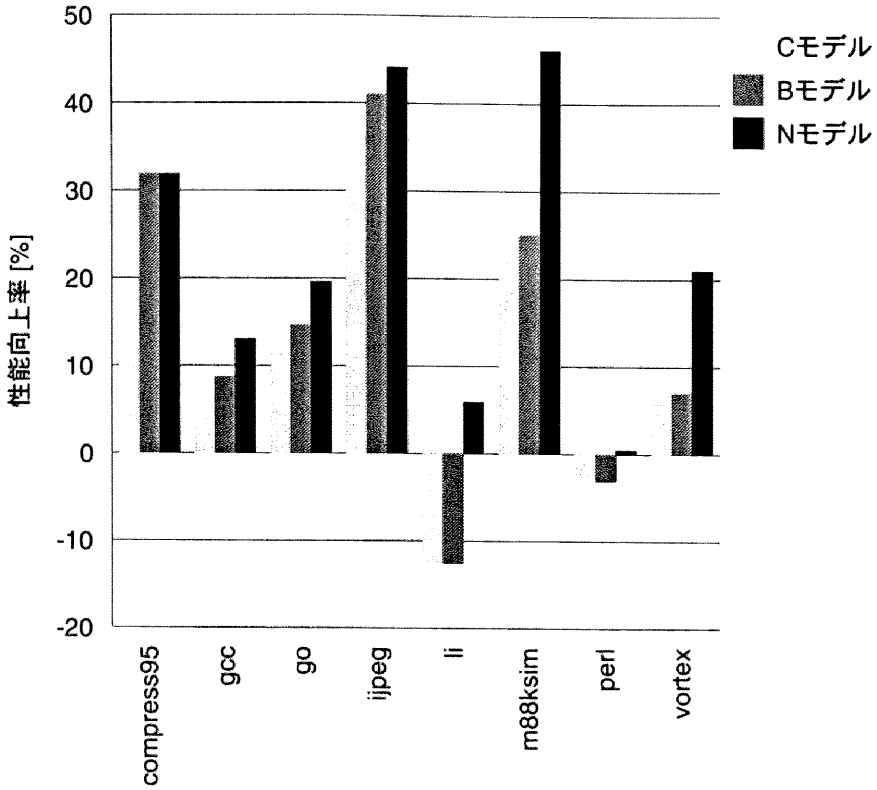


図 4.11　同期機構の評価結果

ての同期／通信機構を持つが，受信待ち命令に後続する命令の実行がブロックされる．このモデルは，full/empty ビットによる既存の同期モデル[2),18),39)]である．

N モデル（non-blocking model）：命令ウィンドウ・ベースの同期機構を有し，ブロッキングの生じない同期を実現する SKY のモデルである．

図 4.11 に評価結果を示す．縦軸は基準プロセッサに対する性能向上率である．同図よりわかるように，C モデルは，どのベンチマーク・プログラムにおいても N モデルより性能が低い．基準プロセッサに対する平均の性能向上率は，わずか 8.9% である．このことより，レジスタ通信バンド幅がたとえ十分にあったとしても，全レジスタという粗い粒度での同期は性能低下の大きな要因となることがわかる．

B モデルの性能向上率も，平均で 12.9% と低い．compress95 では N モデルに近い性能を示しているが，その他では N モデルに劣る．特に，m88ksim と vortex にお

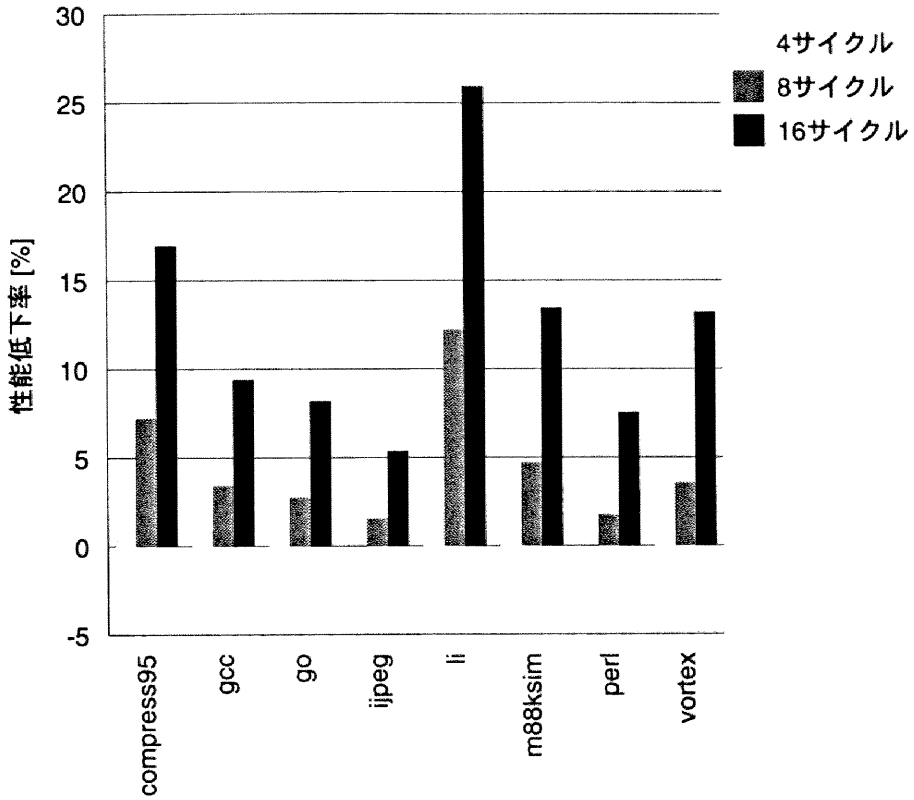


図 4.12　同期／通信のオーバヘッドによる性能低下

いて大きな違いを示している．N モデルは B モデルより最大で 16.9%，平均でも 7.8% 性能が高い．このことより，ノンブロッキング同期機構によりスレッド内 ILP の利用を妨げないことは，性能向上にとって重要であることがわかる．

4.5.4　同期／通信のオーバヘッド

同期／通信のオーバヘッドが性能に与える影響を評価する．図 4.12 に，同期／通信のオーバヘッドが 2 サイクル場合を基準とし，オーバヘッドを 4，8，16 サイクルと増加させた場合の 2 プロセッサ構成の SKY での性能低下率を示す．

図 4.12 よりわかるように，ほとんどの場合，オーバヘッドの増加にしたがい，性能が低下していく．オーバヘッドが 4 サイクルの場合，性能低下率は最大で 1.9% であり，この程度は許容できることがわかる．一方，オーバヘッドが 8 サイクルの場合，性能低下率は，平均では 4.4% とあまり大きくないが，プログラムによって

は 12.2% もの大きな性能低下が生じている．また，オーバヘッドが 16 サイクルの場合，平均で 12.6%，最大では 25.9% にもなり，性能を著しく低下させることがわかる．

なお vortex では，オーバヘッドが 4 サイクルのときの方が 2 サイクルのときより，わずかに（0.6%）性能が向上している．SKY では，fork 命令が実行されスレッドが生成されるかどうかは，後続プロセッサの空き状況によるので，同期／通信のオーバヘッドが変われば，生成されるスレッドも変わる．この効果が性能に現れたと思われる．

4.5.5　コード量と命令の分布

SKY が実行するプログラムには，通常の逐次プログラムに対して，SKY 専用の命令を追加しなければならない．命令を追加することによりコード量が増加し，メモリ・システムの性能を低下させる可能性がある．表 4.5 に，SKY のプログラムにおける専用命令の静的分布を示す．send 命令についてはさらに，4.4.3 項で述べた真の送信と転送の 2 つに分類している．同表より，コード量の増加率は，平均でわずか 4.8%，最大でも 10.5% と小さい．したがって，メモリ・システムに与える影響はほとんどないと考えられる．

表 4.6 に，2 プロセッサ構成の SKY における，1 スレッドあたりの動的命令数の平均を示す．命令は，専用命令を除く元のプログラムにおける命令と，3 つの SKY 専用命令に分類している．send 命令に関してはさらに，表 4.5 と同様，真の送信と転送に分類している．同表に示すように，スレッドあたりの動的命令数は約 100 ～1000，平均で 368 である．この数は，従来の粗粒度並列における命令数より非常に小さく，非数値計算に求められる細粒度 TLP を SKY が利用していることがわかる．また，SKY 専用命令が全命令数に占める割合は，平均 4.9%，最大でも 7.8% と多くない．また，挿入された専用命令のほとんどは send 命令であるが，これらは互いに独立な命令なので並列に実行され，これらの命令による性能低下はほとんどないと考えられる．さらに，1 スレッドあたりの転送用 send 命令の数は，1 ～4 と非常に小さいことがわかる．このことは，複数のスレッドをまたいで

表 4.5　SKY 専用命令の静的分布

ベンチマーク	fork	finish	send		合計
			真の送信	転送	
compress95	1.01%	1.01%	7.02%	1.50%	10.53%
gcc	0.72%	0.72%	5.64%	1.02%	8.10%
go	0.66%	0.66%	5.53%	1.25%	8.11%
ijpeg	0.29%	0.29%	2.34%	0.36%	3.27%
li	0.31%	0.31%	1.78%	0.38%	2.78%
m88ksim	0.02%	0.02%	0.12%	0.01%	0.15%
perl	0.32%	0.32%	2.36%	0.34%	3.35%
vortex	0.27%	0.27%	1.35%	0.24%	2.12%

生きるレジスタはほとんどなく，プロセッサが命令実行によりレジスタ転送を行う

表 4.6　スレッド当たりの動的命令数

ベンチマーク	元のコード	fork	finish	send		合計
				真の送信	転送	
compress95	193.3	1.0	1.0	11.2	1.3	207.8
gcc	256.0	1.0	1.0	7.7	4.1	269.8
go	263.6	1.0	1.0	9.4	3.0	278.0
ijpeg	1218.1	1.0	1.0	12.2	2.7	1235.0
li	128.2	1.0	1.0	7.8	1.0	139.0
m88ksim	234.7	1.0	1.0	11.9	0.9	249.5
perl	292.9	1.0	1.0	8.8	1.7	305.4
vortex	250.6	1.0	1.0	3.8	1.2	257.6

ことは，性能上ほとんど問題がないことを示している．

4.6　関連研究

Olukotun らは，スーパスカラ・プロセッサを複数持つマルチプロセッサと，同一のチップ面積でより広い命令発行バンド幅を持つスーパスカラ・プロセッサの性能の比較を行った[32]．非数値計算プログラムにおける IPC の評価では，マルチプロセッサはスーパスカラ・プロセッサより 10 ～30% 性能が低いという結果が得られた．しかし，マルチプロセッサの方がより単純なスーパスカラ・プロセッサを用いているので，高いクロック速度を実現でき，その結果，性能差はなくなると推測している．この研究は，マルチプロセッサの将来の可能性を示したといえるが，アー

キテクチャは従来のものをチップ上にそのまま実現したものである．これに対して，SKY は単一チップの利点を積極的に生かしている．

Nayfeh らは，マルチプロセッサをチップに集積する場合において，種々のメモリ・システムの性能について詳細な測定を行った[27]．彼らの測定結果によると，L1 キャッシュを共有するマルチプロセッサは，共有することにより（具体的には，キャッシュとプロセッサをつなぐクロスバー・スイッチによる），L1 キャッシュ・アクセス時間が延びるにも関わらず，細粒度並列アプリケーションではもちろん，計算量に対してより通信量の少ないアプリケーションにおいても，L2 キャッシュ共有や主記憶共有のマルチプロセッサより高い性能を示すことを示した．彼らの研究は，メモリ・システムに関する研究であり，プロセッサに関する著者の研究とは直交するものである．

Breach と Sohi らは，逐次的にスレッドを生成し，レジスタ通信を行うマルチスカラと呼ぶアーキテクチャを提案した[2),39)]．レジスタ通信は，レジスタ・ファイル間でコンパイラの支援の下に行われる．スレッドが送信すべきレジスタと受信するレジスタをコンパイラが指定する．送信側では指定されたレジスタに書き込みが起こると，後続のプロセッサのレジスタ・ファイルに値が自動的に送られる．受信側のプロセッサでは，コンパイラが指定したレジスタが未受信の場合で，そのレジスタをソース・レジスタとする命令は，実行を停止し受信を待ち合わせる．SKY は，マルチスカラの実行モデルと同様のモデルを採用しているが，通信機構がマルチスカラとは異なる．特に，マルチスカラでは同期による命令実行のブロッキングが生じるが，SKY では生じない．なお，彼らはアウト・オブ・オーダのプロセッサで構成したマルチスカラ・プロセッサの性能を評価しているが，ノンブロッキングの同期機構については提案していない．また，本章で著者が指摘した TLP 利用と ILP 利用との干渉については何も議論していない．

Keckler らは，細粒度 TLP のためにレジスタ・ベースの同期／通信機構を提案し，メモリ・ベースの同期／通信機構に比べて，大幅にオーバヘッドを削減できることを示した[18)]．送信も受信も命令により行う．受信機構としては，レジスタごとに full/empty ビットを持ち，`empty` 命令と呼ぶ命令によってこれを empty とし受信待

ちを指示する．empty であるレジスタをソース・レジスタとする命令は，実行を停止し，後続の命令の実行をブロックする．SKY は命令によりレジスタの送信を行う点でこの機構と同一であるが，同期機構がノンブロッキングである点が異なる．また，彼らの研究は数値計算プログラムに対するものであり，非数値計算プログラムにおける評価や議論については行われてない．

鳥居らは，スレッド生成時に全てのレジスタの内容を後続のプロセッサのレジスタに，非常に少ないサイクル数でコピーする特別なレジスタ・ファイルを持つ MUSCAT と呼ぶアーキテクチャを提案した[40),41)]．レジスタ・ファイル間に高いバンド幅を実現することにより，スレッド間通信の時間を小さくした．しかし，スレッド生成後に生成され後続スレッドに送信しなければならないレジスタ値については，メモリを介しての同期／通信の必要性があり，このオーバヘッドは削減されていない．

Tullen らは，単一のスーパスカラ・プロセッサにおいて，命令レベルで複数のスレッドを実行する SMT（Simultaneous Multithreading）とよぶ方式を提案した[42)]．SMT は，豊富な機能ユニットを持つスーパスカラ・プロセッサにおいて，複数のプログラムを命令レベルで同時に実行することにより，プロセッサのスループットを向上させることができる．しかし，SMT だけでは，1 つのプログラムの実行時間を短縮することはできない．

5　結論

近年のハイエンド・マイクロプロセッサはスーパスカラ方式によりILPを引き出し性能を向上させている．分岐による制御流の乱れは，こうしたプロセッサの性能を厳しく制限する．この影響を緩和するために一般に投機的実行が用いられる．今後もプロセッサの命令発行幅はより広く，パイプライン段数はより深くなる傾向にあり，分岐予測精度の向上はますます重要になる．本研究の1つの課題は，スーパスカラ・アーキテクチャにおいて分岐予測機構に着目し，マイクロプロセッサの性能を向上させることであった．

しかし，スーパスカラ方式により，より多くのILPをプログラムより引き出し高性能化を図る方法には限界が見え始めてきた．これを解決する1つの方法として，最近，ILPに加えTLPを利用する単一チップ・マルチプロセッサの研究が行われている．マイクロプロセッサのもっとも一般的な応用である非数値計算応用で高い性能を達成するためには，細粒度のTLPを低いオーバヘッドで利用することが要求される．本研究のもう1つの課題は，スーパスカラ方式の限界を超えるアーキテクチャを実現することであった．

以上を課題として行なわれた本研究によって得られた成果をまとめる．

1. 分岐方向を予測するテーブルにおいて競合が発生しても分岐方向の予測精度を大きく落とさない機構の提案

   競合はハードウェア資源の限られた2レベル分岐予測機構の性能を制限する大きな原因である．扱う分岐アドレスと分岐履歴の組み合わせ数に対し，現実的には小さなテーブルしか用意できないので，競合は本質的に不可避である．これに対して本研究では，分岐方向の偏りを考慮しPHTを利用することで破壊的競合が削減できることを示し，この考えに基づくsgshareと呼ぶ予測機構を提案した．sgshareはtakenに偏った分岐とnot-takenに偏った分岐を別々のPHTに対応づけることで破壊的競合を削減する．この結果，gshareに比べ競合による予測精度への悪影響を減少させることができ，同じ大きさのPHTでより長い履歴を活用することができる．シミュレーションにより性能評価を行った結

果，ハードウェア量8Kバイトのgshareに対して，ほぼ同等のコストでベンチマーク平均で0.45%，最大で0.89%予測精度が向上することを確認した．この予測精度の向上により，今日のスーパースカラ・プロセッサで0.3～13.3%の速度向上が，また命令発行幅が増えパイプライン段数が深くなった将来のスーパースカラ・プロセッサで0.9～27.2%速度向上が見込めることがわかった．

2. 分岐先アドレス予測成功率をほとんど低下させることなく，BTBのハードウェア量を削減する手法の提案

分岐先アドレスを予測するBTBでは，高い予測成功率を得るためには多くのエントリ数が必要となりハードウェア量が大きくなるという問題がある．本研究では，分岐距離の分布を利用してエントリあたりのハードウェア量を削減することによって，BTBのハードウェア量を削減する2レベル表方式を提案した．シミュレーションにより性能評価を行った結果，従来のBTB方式に対して分岐先アドレス予測成功率をほとんど低下させることなく，分岐先アドレス部のハードウェア量を約49%削減することができた．タグ部を含めたBTB全体のハードウェア量では削減率は約24%に低下するが，Faginが提案したタグ部のハードウェア量削減手法を適用することにより，全体ハードウェア量を約38%削減することができた．

3. レジスタ値の同期／通信機能を備え，複数のスレッドを並列に実行するSKYと呼ぶマルチプロセッサ・アーキテクチャの提案

本研究では，非数値計算プログラムに対し，マルチスレッド実行により性能を向上させるSKYと呼ぶマルチプロセッサ・アーキテクチャを提案し，詳細な評価を行った．SKYの最大の特徴は，ノンブロッキングのレジスタ同期を行う命令ウィンドウ・ベースの同期機構である．本機構は，TLP利用において，同期がILPに与える悪影響を抑制することができ，マルチプロセッサを構成するプロセッサとしてスーパスカラ・プロセッサを用いることに適している．SKY用にコンパイラを作成し，シミュレーションにより評価を行った．単一のスーパスカラ・プロセッサに対して，2プロセッサ構成のSKYは最大で46.1%，平均で21.8%の性能向上率を達成した．2プロセッサ構成のSKYと同等のハード

ウェア量を持つ命令発行幅の広い単一のスーパスカラ・プロセッサと性能を比較すると，SKY は最大で 37.3%，平均で 13.0% 高い性能が得られることを確認した．SKY を構成する命令発行幅の狭いスーパスカラ・プロセッサは，命令発行幅の広いスーパスカラ・プロセッサに比べて高速に動作するので，性能差はさらに広がると考えられる．また，ブロッキングの同期に対して，著者の提案するノンブロッキングの同期は，最大で 16.9%，平均で 7.8% の性能向上率を示した．このように SKY は高い性能と従来の機構に対する優位性を示したものの，まだ多くの課題を残している．1 つはコンパイラを改良し，プログラムに内在する TLP をより多く引き出すことが必要である．また，同期以外に TLP 利用が ILP 利用を阻害する問題が大きいことが明らかとなった．この問題を解決する方法を見いだす必要がある．これらについて今後も検討していく予定である．

謝辞

本研究を進めるにあたり，ご指導頂いた名古屋大学大学院工学研究科島田俊夫教授，平田富夫教授，高木直史教授，安藤秀樹助教授に深く感謝の意を表する．

また，本研究に多大なる協力を頂いた野口良太氏，森敦司氏，山田祐司氏，岩田充晃氏，小川行宏氏に，深く感謝する．また，いつも有益な議論，アドバイスをして下さった名古屋大学大学院工学研究科島田研究室の皆様に感謝の意を表する．

最後に本研究の一部は，文部省科学研究費補助金基盤研究 (C)「広域命令レベル並列によるマイクロプロセッサの高性能化に関する研究」(課題番号 1068034)，文部省科学研究費補助金基盤研究 (C)「分岐予測と投機的実行に関する研究」(課題番号 11680351) 及び 財団法人堀情報科学振興財団助成「広域命令レベル並列性を利用するコンピュータ・アーキテクチャとコンパイラに関する研究」 の支援により行った．

参考文献

1) Aho, A. V., Sethi, R., and Ullman, J. D.: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley Publishing Company (1986).

2) Breach, S. E., Vijaykumar, T. N., and Sohi, G. S.: The Anatomy of the Register File in a Multiscalar Processor, *Proc. MICRO-27*, pp.181-190 (1994).

3) Butler, M., Yeh, T-Y., and Patt, Y.: Single Instruction Stream Parallelism Is Greater than Two, *Proc. 18th Int. Symp. on Computer Architecture*, pp.276-286 (1991).

4) Calder, B. and Grunwald, D.: Fast & Accurate Instruction Fetch and Branch Prediction, *Proc. 21st Int. Symp. on Computer Architecture*, pp.2-11 (1994).

5) Chang, P-Y., Hao, E., Yeh, T-Y. and Patt, Y.: Branch Classification: a New Mechanism for Improving Branch Predictor Performance, *Proc. MICRO-27*, pp.22-31 (1994).

6) Driesen, K. and Hölzle, U.: Accurate Indirect Branch Prediction, *Proc. 25th Int. Symp. on Computer Architecture*, pp.167-177 (1998).

7) Driesen, K. and Hölzle, U.: The Cascaded Predictor: Economical and Adaptive Branch Target Prediction, *Proc. MICRO-31*, pp.249-258 (1998).

8) Fagin, B.: Partial Resolution in Branch Target Buffers, *Computer*, Vol.46, No.10, pp.1142-1145 (1997).

9) Fagin, B. and Russeell, K.: Partial Resolution in Branch Target Buffers, *Proc. MICRO-28*, pp.193-198 (1995).

10) Gloy, N., Young, C., Chen, J. B. and Smith, M. D.: An Analysis of Dynamic Branch Prediction Schemes on System Workloads, *Proc. 23rd Inte. Symp. on Computer Architecture*, pp.12-21, (1996).

11) Gwennap, L.: Intel's P6 Uses Decoupled Superscalar Design, Vol. 9, No. 2, pp.9-15 (1995).

12) Gwennap, L.: Digital 21264 Sets New Standard, Microprocessor Report, Vol. 10, No. 14, pp.11-16 (1996).

13) 原 哲也 , 安藤 秀樹 , 中西 知嘉子 , 中屋 雅夫 : 分岐先バッファにおける分岐先情報の削減 , *第 49 回情報処理学会全国大会論文集* , pp.1-2 (1994).

14) Hara, T., Ando, H., Nakanishi, C., and Nakaya, M.: Performance Comparison of ILP Machines with Cycle Time Evaluation, *Proc. 23rd Int. Symp. on Computer Architecture*, pp.213-224 (1996).

15) 岩田充晃 , 小林良太郎 , 安藤秀樹 , 島田俊夫 : 制御等価を利用したスレッド分割技法 , *情報処理学会研究報告* , 97-ARC-128, pp.127-132 (1998).

16) Kaeli, D.R. and Emma, P.G.:Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns, *Proc. 18th Int. Symp. on Computer Architecture*, pp.34-42 (1991).

17) Kane, G.: *MIPS RISC Architecture*, Prentice Hall, Englewood Cliffs, New Jersey (1988).

18) Keckler, S. W., Dally, W. J., Maskit, D., Carter, N. P., Chang, A., and Lee, W. S.: Exploiting Fine-Grain Thread Level Parallelism on the MIT Multi-ALU Processor, *Proc. 25th Int. Symp. on Computer Architecture*, pp.306-317 (1998).

19) 小林良太郎 , 岩田充晃 , 安藤秀樹 , 島田俊夫 : 制御依存解析と複数命令流実行を導入した投機的実行機構の提案と予備的評価 , *情報処理学会研究報告* , 97-ARC-125, pp.133-138 (1997).

20) 小林良太郎, 岩田充晃, 安藤秀樹, 島田俊夫 : 非数値計算プログラムのスレッド間命令レベル並列を利用するプロセッサ・アーキテクチャ SKY, *1998 年並列処理シンポジウム JSPP'98*, pp.87-94 (1998).

21) Kobayashi, R., Iwata, M., Ogawa, Y., Ando, H., and Shimada, T.: An On-Chip Multiprocessor Architecture with a Non-Blocking Synchronization Mechanism, *Proc. 25th Euromicro Conf.*, pp.432-440 (1999).

22) Lam, M. S. and Wilson, R. P.: Limits of Control Flow on Parallelism, *Proc. 19th Int. Symp. on Computer Architecture*, pp.46-57 (1992).

23) Lee, C-C., Chen, I-C. K. and Mudge, T. N.: The Bi-Mode Branch Predictor, *Proc. MICRO-30*, pp.4-13 (1997).

24) Lee, J.K.F. and Smith, A.J.: Branch Prediction Strategies and Branch Target Buffer Design, *Computer*, Vol.17, No.1, pp.6-22 (1984).

25) McFarling, S.: Combining Branch Predictors, *WRL Technical Note*, TN-36, Digital Equipment Corporation (1993).

26) Michaud, P., Seznee, A. and Uhlig, R.: Trading Conflict and Capacity Aliasing in Conditional Branch Predictors, *Proc. 24th Int. Symp. on Computer Architecture*, pp.292-303 (1997).

27) Nayfeh, B. A., Hammond, L., and Olukotun, K.: Evaluation of Design Alternatives for a Multiprocessor, *Proc. 23th Int. Symp. on Computer Architecture*, pp.22-24 (1996).

28) 野口良太，森敦司，小林良太郎，安藤秀樹，島田俊夫：履歴情報の競合の悪影響を緩和した分岐予測機構の予備的評価，*電気関係学会東海支部連合大会論文集*，p.317 (1997).

29) 野口良太，森敦司，小林良太郎，安藤秀樹，島田俊夫：競合による予測精度低下を緩和する分岐予測機構，*情報処理学会研究報告*，97-ARC-127, pp.63-70 (1997).

30) 野口良太，森敦司，小林良太郎，安藤秀樹，島田俊夫：分離型パターン履歴表による分岐予測機構の競合耐性の改善，*1998 年並列処理シンポジウム JSPP'98*, pp.7-14 (1998).

31) 小川行宏：マルチスレッド計算機 SKY のコンパイラにおける命令移動に関する研究，*名古屋大学電気学科，電子工学科，および電子情報学科卒業研究論文* (1997).

32) Olukotun, K., Nayfeh, B. A., Hammond, L., Wilson, K., and Chang, K.: The Case for a Single-Chip Multiprocessor, *Proc. Seventh Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.2-11 (1996).

33) Palacharla, S., Jouppi, N. P., and Smith, J. E.: Complexity-Effective Superscalar Processors, *Proc. 24th Int. Symp. on Computer Architecture*, pp.206-218 (1997).

34) Pan, S-T., So, K. and Rahmeh, J. T.: Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proc. Fifth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.76-84 (1992).

35) Sechrest, S., Lee, C-C. and Mudge, T.: Correlation and Aliasing in Dynamic Branch Predictors, *Proc. 23rd Int. Symp. on Computer Architecture*, pp.22-32 (1996).

36) Smith, B. J.: Architecture and Applications of the HEP Multiprocessor Computer System, Society of Photo-optical Instumentation Engineers, 298:241-248 (1981).

37) Smith, M. D.: Tracing with Pixie, Technical Report CSL-TR-91-497, Stanford University (1991).

38) Smith, M. D., Horowitz, M. A., and Lam, M. S.: Efficient Superscalar Performance Through Boosting, *Proc. Fifth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.248-259 (1992).

39) Sohi, G. S., Breach, S. E., and Vijaykumar, T. N.: Multiscalar Processor, *Proc. 22nd Int. Symp. on Computer Architecture*, pp.414-425 (1995).

40) 鳥居淳，近藤真己，本村真人，西直樹，小長谷明彦：On Chip Multiprocessor 指向制御並列アーキテクチャ MUSCAT の提案，*1997 年並列処理シンポジウム JSPP'97*, pp.229-236 (1997).

41) 鳥居淳，近藤真己，本村真人，池野晃久，小長谷明彦，西直樹：オンチップ制御並列プロセッサ MUSCAT の提案，*情報処理学会論文誌*，Vol. 39, No. 6, pp.1622-1631 (1998).

42) Tullsen, D., Eggers, S., and Levy, H.: Simultaneous Multithreading: Maximizing On-Chip Parallelism, *Proc. 22nd Int. Symp. on Computer Architecture*, pp.392-403 (1995).

43) Uhlig, R., Nagle, D., Mudge, T., Sechrest, S. and Emer, J.: Instruction Fetching: Coping with Code Bloat, *Proc. 22nd Int. Symp. on Computer Architecture*, pp.345-356 (1995).

44) Wall, D. W.: Limits of Instruction-Level Parallelism, *Proc. Fourth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.177-188 (1991).

45) Wang, H., Sun, T. and Yang, Q.: Minimizing Area Cost of On-Chip Cache Memories by Caching Address Tags, *Computer*, Vol.46, No.11, pp.1187-1201 (1997).

46) 山田祐司，小林良太郎，安藤秀樹，島田俊夫：分岐先アドレスの性質を利用した2レベル表による分岐先バッファの容量削減，*情報処理学会研究報告*，98-ARC-131, pp.59-64 (1998).

47) 山田祐司，小林良太郎，安藤秀樹，島田俊夫：2レベル表構成の導入による分岐先バッファの容量削減，*1999年並列処理シンポジウム JSPP'99*, pp.103-110 (1999).

48) Yeh, T-Y. and Patt, Y. N.: Two-Level Adaptive Branch Prediction, *Proc. MICRO-24*, pp.55-61 (1991).

49) Yeh, T-Y. and Patt, Y. N.: Alternative Implementation of Two-Level Adaptive Branch Prediction, *Proc. 19th Int. Symp. on Computer Architecture*, pp.124-134 (1992).

50) Yeh, T-Y. and Patt, Y. N.: A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History, *Proc. 20th Int. Symp. on Computer Architecture*, pp.257-266 (1993).

51) Young, C., Gloy, N. and Smith, M. D.: A Comparative Analysis of Schemes for Correlated Branch Prediction, *Proc. 22nd Int. Symp. on Computer Architecture*, pp.276-286 (1995).

発表論文

1) 森敦司，小林良太郎，野口良太，安藤秀樹，島田俊夫：直交性を考慮したハイブリッド分岐予測機構，*情報処理学会研究報告*，97-ARC-125, pp.115-120 (1997).

2) 小林良太郎，岩田充晃，安藤秀樹，島田俊夫：制御依存解析と複数命令流実行を導入した投機的実行機構の提案と予備的評価，*情報処理学会研究報告*，97-ARC-125, pp.133-138 (1997).

3) 野口良太，森敦司，小林良太郎，安藤秀樹，島田俊夫：競合による予測精度低下を緩和する分岐予測機構，*情報処理学会研究報告*，97-ARC-127, pp.63-70 (1997).

4) 野口良太，松崎元昭，小林良太郎，安藤秀樹，島田俊夫：遺伝的アルゴリズムを用いた分岐予測機構設計，*電子情報通信学会研究報告*，97-CPSY-107, pp.45-50 (1998).

5) 岩田充晃，小林良太郎，安藤秀樹，島田俊夫：制御等価を利用したスレッド分割技法，*情報処理学会研究報告*，97-ARC-128, pp.127-132 (1998).

6) 野口良太，森敦司，小林良太郎，安藤秀樹，島田俊夫：分離型パターン履歴表による分岐予測機構の競合耐性の改善，*1998 年並列処理シンポジウム JSPP'98*, pp.7-14 (1998).

7) 小林良太郎，岩田充晃，安藤秀樹，島田俊夫：非数値計算プログラムのスレッド間命令レベル並列を利用するプロセッサ・アーキテクチャ SKY, *1998 年並列処理シンポジウム JSPP'98*, pp.87-94 (1998).

8) 山田祐司，小林良太郎，安藤秀樹，島田俊夫：分岐先アドレスの性質を利用した 2 レベル表による分岐先バッファの容量削減，*情報処理学会研究報告*，98-ARC-131, pp.59-64 (1998).

9) 森敦司，小林良太郎，安藤秀樹，島田俊夫：動的に破壊的競合を削減する分岐予測機構に関する検討，*情報処理学会研究報告*，98-DSP-89, pp.27-34 (1998).

10) Kobayashi, R., Yamada, Y., Ando, H., and Shimada, T.: A Cost-Effective Branch Target Buffer with a Two-Level Table Organization, *Proc. Second Int. Symp. on Low-Power and High-Speed Chips*, p.267 (1999).

11) 野口良太，森敦司，小林良太郎，安藤秀樹，島田俊夫：分岐方向の偏りを利用し破壊的競合を低減する分岐予測方式，*情報処理学会論文誌*，Vol. 40, No. 5, pp.2119-2131 (1999).

12) 山田祐司，小林良太郎，安藤秀樹，島田俊夫：2 レベル表構成の導入による分岐先バッファの容量削減，*1999 年並列処理シンポジウム JSPP'99*, pp.103-110 (1999).

13) 小林良太郎，安藤秀樹，島田俊夫：データフロー・グラフの最長パスに着目したクラスタ化スーパスカラ・プロセッサにおける命令発行機構，*情報処理学会研究報告*，99-ARC-134, pp.181-186 (1999).

14) Kobayashi, R., Iwata, M., Ogawa, Y., Ando, H., and Shimada, T.: An On-Chip Multiprocessor Architecture with a Non-Blocking Synchronization Mechanism, *Proc. 25th Euromicro Conf.*, pp.432-440 (1999).

15) 野口良太，松崎元昭，小林良太郎，安藤秀樹，島田俊夫：遺伝的アルゴリズムを用いた分岐予測機構設計，*計測自動制御学会論文集*，Vol. 35, No. 11, pp.1431-1437 (1999).

16) 小林良太郎，山田祐司，安藤秀樹，島田俊夫：2 レベル表方式による分岐先バッファ，*情報処理学会論文誌*，Vol. 41, No. 5, pp.1351-1359 (2000).

13) 加納正晃，小林良太郎，安藤秀樹，島田俊夫：非数値計算プログラムにおけるスレッド・レベル並列性の限界，*情報処理学会研究報告*，2000-ARC-140, pp.55-60 (2000).

17) 小林良太郎，小川行宏，岩田充晃，安藤秀樹，島田俊夫：非数値計算応用向けスレッド・レベル並列処理マルチプロセッサ・アーキテクチャ SKY, *情報処理*

学会論文誌 , Vol. 42, No. 2, pp.349-366 (2001).