社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－SLDM－118 (21)
2005／1／26

単一チップ・マルチプロセッサSKYにおける投機的スレッド実行の性能評価

上村井明夫† 小林良太郎† 安藤秀樹† 島田俊夫†

† 名古屋大学大学院工学研究科

E-mail: †{kamimrai,kobayasi,shimada}@shimada.nuee.nagoya-u.ac.jp, ††ando@cse.nuee.nagoya-u.ac.jp

あらまし 我々は，マルチスレッド実行により性能を向上させる単一チップ・マルチプロセッサSKYを提案してきた．特に，SKYでは非数値計算プログラムのような粒度の大きなスレッドレベル並列性 (TLP: Thread-Level Parallelism) が少ないプログラムを，効率よく実行することができる．SKYはほぼ同一コストのスーパスカラ・プロセッサに対して性能向上を達成しているが，十分にTLPを利用しているとは言えない．本論文では，制御依存とデータ依存による制約を緩和するための投機手法を，SKYに導入したときの性能について評価を行う．また，データ依存を緩和する技術である値予測を考慮したスレッド分割についても評価を行う．評価の結果，投機手法を用いない場合に対してSPECint2000で平均12.7%，MediaBenchで平均12.3%の性能向上を得ることができた．しかし，値予測を考慮したスレッド分割では，従来のスレッド分割と比べてほとんど性能向上は得られなかった．

キーワード マルチプロセッサ，投機的スレッド実行，スレッド分割

Performance Evaluation of Speculative Thread Execution in the Single-Chip Multiprocessor SKY

Akio KAMIMURAI†, Ryotaro KOBAYASHI†, Hideki ANDO†, and Toshio SHIMADA†

† Graduate School of Engineering, Nagoya University

E-mail: †{kamimrai,kobayasi,shimada}@shimada.nuee.nagoya-u.ac.jp, ††ando@cse.nuee.nagoya-u.ac.jp

Abstract We have proposed multi-processor architecture, called SKY, which efficiently executes multiple threads in parallel. In particular, the SKY efficiently executes programs which have fine-grain thread-level parallelism(TLP), such as non-numerical programs. Although the SKY achieves performance improvement over the superscalar processor which has about as much cost as the SKY, the SKY does not exploit TLP effectively enough. This paper evaluates speculative thread execution on the SKY to mitigate the constraint of control and data dependences. We also introduce a new thread partitioning technique which considers value prediction which is used to mitigate the constraints of data dependences, and evaluate it. Our results show that the speculative thread execution achieves performance improvements by 12.7% in SPECint2000 or by 12.3% in MediaBench over conventional non-speculative thread execution. However, our new thread partioning technique achieves little improvement compared with a conventional thread partioning technique.

Key words multi-processor, speculative thread execution, thread partioning

1. はじめに

スーパスカラ・プロセッサに代わるアプローチとして，近年，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 (**CMP**: Chip MultiProcessor) が注目を集めている．その背景には，単一制御流において利用可能な命令レベル並列性 (**ILP**: Instruction-Level Parallelism) が限界に近づきつつあるなど，スーパスカラ・プロセッサに対する悲観的観測に加え，半導体回路技術の進歩にともない，複数のプロセッサの単一チップへの集積化が実現可能なことがある．発表されている商用のCMP [3]～[6], [11], [12] では，複数のプログラムの同時実行や，特定の処理の高速化 [3] を目的としている．

これに対し，1つのプログラムをスレッドに分割し並列実行することで性能を向上させる研究が行われている

[7], [9], [14], [17], [18]. CMP ではプロセッサ間通信のレイテンシを減少させることができるという特徴があるため，粒度の小さなスレッドレベル並列性（**TLP**: Thread-Level Parallelism）を利用することができる．このため，特に，粒度の大きな TLP が少ない非数値計算プログラムの高速化が期待できる．

細粒度の TLP を効率よく利用するためには，CMP の特徴を活かした同期／通信機構が必要である．そこで，レジスタ値を直接通信し，同期を行う機構 [17], [18] が提案された．これは，マルチプロセッサで一般的なメモリを介して同期／通信を行う機構に比べ，オーバヘッドを大幅に減少させることができる．しかし，この同期／通信機構にはまだ改善の余地がある．なぜならば，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からである．これは，要素プロセッサとしてアウト・オブ・オーダ実行のスーパスカラ・プロセッサを前提とした場合，ILP の利用を阻害しているといえる．

そこで我々は，細粒度の TLP を効率よく利用することができる CMP アーキテクチャとして **SKY** を提案した [10]．SKY は，同期が成立しなかった命令が現れても，その命令とデータ依存関係にない後続命令の実行を停止させることはない．これを**ノンブロッキング**な同期という．この機構により，スレッド間に存在する並列性を十分に引き出すことができる．

SKY に関するこれまでの評価で，SKY はほぼ同一コストのスーパスカラ・プロセッサに対して性能向上を達成しているが，十分に TLP を利用しているとは言えない．これは，特に非数値計算プログラムでは制御構造が複雑であるため制御依存による制約が厳しいことと，データ依存関係が多いためスレッド間データ依存による制約が厳しいことが原因であると考えられる．

制御依存による制約とは，SKY のスレッド生成命令やスレッド間通信命令を，それに先行する全ての分岐命令の結果が確定するまで実行することができないという制約である．分岐命令の多いプログラムではこれは TLP の利用を制限する大きな要因になると考えられる．

スレッド間データ依存による制約とは，プログラムをスレッドに分割した際にできるスレッド間のデータ依存による制約である．データ依存関係の多いプログラムではスレッド間のデータ依存が多く生じる．そのため，スレッド間の依存関係を解決するための待ち合わせ時間が長くなり，性能に大きな影響を与えると考えられる．

本論文では，SKY において，制御依存とスレッド間データ依存を緩和するために投機的にスレッドを実行する技術について説明し，その評価を行う．また，データ依存を緩和する技術である値予測の効果を考慮して作成したスレッド分割の評価も行う．

2 章では SKY の概要について述べる．3 章では投機的スレッド実行を行うための技術について説明する．4 章では値予測の効果を考慮した利得計算手法について説明する．5 章で評価を行い，6 章でまとめる．

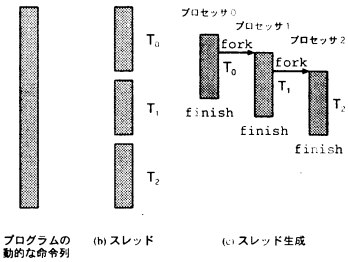

図 1　SKY のマルチスレッド・モデル

2. SKY の概要

本章では，SKY のマルチスレッド・モデル，ハードウェア構成 [10] およびコンパイラ [8] について述べる．はじめにマルチスレッド・モデルについて述べる．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 のマルチスレッド・モデルは通常のマルチスレッド・モデルと比べて次に示す制約を課している．これは，**Multiscalar** [17] や **MUSCAT** [18] と同様のモデルである．

- 各スレッドは，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，逐次実行の順において自分の直後のスレッドを生成する．

図 1(a) に逐次実行命令列を，図 1(b) にこれに対する SKY におけるスレッド分割の様子を，図 1(c) にスレッドを並列に実行する様子を示す．

図 1(c) 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．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．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．また `fork` 命令を挿入する位置を**フォーク点**，`finish` 命令を挿入する位置を**子の開始点** と呼ぶ．

次にハードウェア構成について述べる．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，`send` と呼ぶ専用命令を使用して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**の同期と呼ぶ同期機構を導入している．この機構は，命令ウィンドウ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り，ノンブロッキングな同期を実現している．この機構を導入することで，プロセッサは ILP を効率よく利用することができる．

最後にコンパイラについて述べる．コンパイラが行うスレッド分割の処理について，全体の流れを図 2 に示す．最初にスレッド分割に必要な，プロファイル情報等を得るために前処理を行う．次に実際のスレッド分割を行う．スレッド分割は具体的には，フォーク点 f，子の開始点 c，レジスタ通信集合

```
前処理;
各関数について{
  基本ブロックの組の集合 B を求める;
  B の各要素について{
    フォーク点 f, 子の開始点 c を求める;
    レジスタ通信集合 Comm を求める;
    利得 g を求める;
    利得 g があるならば分割情報の集合 F に加える;
  }
  分割情報の集合 F の中から選択する;
}
命令挿入;
```

図 2　コンパイラの処理の流れ

$Comm$，**利得** g を求めることである．これらの値及び集合の組 $(f,c,Comm,g)$ を**分割情報**と呼び，その集合を F とする．以下に F を求める手順を説明する．

まず，ある関数の中で**制御等価** [16] な基本ブロックの組の集合 B を抽出する．次に抽出されたすべての基本ブロックの組について，フォーク点 f，子の開始点 c を求める．次にレジスタ通信集合 $Comm$ を求める．各レジスタに関する send 命令の挿入位置は，送るべきレジスタ値が子の開始点に**到達する** [1] ことが決定する点とする．最後に利得 g を求める．これは注目しているスレッドを選択することによって期待される性能向上の見積りを示す．SKY では，データ依存関係にある命令間の距離に着目して利得を計算している [8]．命令 a と命令 b の距離とは，命令 a から命令 b に至るまでの全ての経路について，それぞれの経路に存在する命令数をその経路を通る確率で重み付けした期待値である．スレッド間のデータ依存関係にある命令間の距離を計算し，その最小値を利得としている．もし注目しているスレッドにスレッド間のデータ依存がなければ，フォーク点から子の開始点までの距離が利得となる．計算された利得に，プロファイルから見積もったスレッドの実行回数を掛け，それをスレッドの利得とする．

このようにして F を作った後，F の中から SKY のマルチスレッド・モデルにおいて利得が最大となるような分割情報の組合せを選ぶ．その後，選択されたスレッドの分割情報にしたがって fork，finish，send の各命令を挿入する．

SKY におけるスレッドへの分割方法では，性能向上の見積もりである利得が重要である．なぜならば，利得を正確に計算しなければ，性能向上に寄与するスレッド分割を選択できないためである．4 章では，データ依存による制約を緩和する手法である値予測を用いたときの従来の利得計算手法の問題点を述べ，値予測を考慮した利得計算手法について説明する．

3.　投機的スレッド実行を行うための技術

本章では，SKY において投機的なスレッド実行を行うための技術について説明する．はじめに，制御依存を緩和する技術について説明する．次にスレッド間データ依存を緩和する技術について説明する．

3.1　制御依存を緩和する技術

従来の SKY では，fork 命令と send 命令を，先行する全ての分岐命令の結果が判明した後に実行している．このような fork 命令，send 命令の実行方法を，本論文では**非投機 fork**，**非投機 send** とよぶ．制御依存が多く存在するプログラムでは，非投機 fork，非投機 send は制御依存による制約として，TLP の利用を制限する要因になると考えられる．

この制約を緩和するために，fork 命令と send 命令を，先行する分岐命令の結果が判明する前に投機的に実行する．本論文では，fork 命令と send 命令を投機的に実行することをそれぞれ，**投機 fork**，**投機 send** とよぶ．

本節では，まず fork 命令を投機的に実行する技術である投機 fork について説明する．次に，send 命令を投機的に実行する技術である投機 send について説明する．

3.1.1　投機 fork

投機 fork は，fork 命令を，先行する分岐命令の結果が判明する前に投機的に実行し，子スレッドを生成する技術である．投機的に子スレッドを生成することで，より多くの TLP を利用できるようになると考えられる．fork 命令の投機にミスした場合は，間違ってスレッドを生成してしまうため，そのスレッドを破棄する．

3.1.2　投機 send

投機 send は，送信するレジスタ値が得られたら，先行する分岐命令の結果が判明する前に send 命令を投機的に実行し，レジスタ値を子スレッドに送信する技術である．投機的にレジスタ値を送信することで，スレッド間レジスタ依存の解消が早くなり，子スレッドのストール時間が短くなると考えられる．send 命令の投機に失敗した場合は，送信されたレジスタ値は間違った値である．そのため，子スレッドにおいて，間違った値を使う命令，および，その命令に依存する命令の実行結果は間違っている．そのため，子スレッドの状態を回復する必要がある．状態を回復する方法としては，次の 2 つの方法がある．1 つは，子スレッドを破棄することで状態を回復する方法である．この方法は，特別な機構を必要としないが，スレッド破棄により大きなペナルティを被る可能性がある．もう 1 つは，投機に失敗した send 命令に依存した命令のみを選択的に再実行する方法である．この方法はスレッドを破棄する方法に比べて複雑な機構が必要となるが，ペナルティが非常に小さい．

本論文では，投機 send のミスから回復する方法として，投機に失敗した send 命令に依存した命令のみを選択的に再実行する方法を用いている．

3.2　スレッド間データ依存を緩和する技術

SKY では，先行スレッドで定義されたレジスタ値を後続スレッドで使用するとき，send 命令を用いて，必要な値を子スレッドに送信している．親スレッドから送信されてくるレジスタ値に依存する命令は，レジスタ値を受信し，スレッド間レジスタ依存が解消するまで実行することができない．このように，スレッド間のレジスタ依存関係にある命令が，値を必要とする前にその値を受信していないときには，その依存関係は TLP を制限する要因になる．

また，先行スレッドでストアした値を後続スレッドでロードするとき，それらの命令はスレッド間のメモリ依存関係にあるといえる．スレッド間レジスタ依存と同様に，スレッド間メモリ依存も TLP を制限する要因になる．

これらの制限を緩和するために，スレッド間データ依存に対して値予測を適用し，先行スレッドから値を受信する前に，その値を必要とする命令を投機的に実行する．本節では，値予測を SKY に適用する方法について説明する．

3.2.1 値 予 測

スーパスカラ・プロセッサにおいて値予測を行う機構として，これまでにいくつかの機構が提案されている [13], [15], [19]．本論文では，ストライド値予測と 2 レベル値予測のハイブリッド値予測器を使用する [19]．

スレッド間レジスタ依存については，受信するレジスタ値，つまり，命令のソース・オペランドの値を予測する．また，スレッド間のレジスタ依存関係にある命令のうち TLP の利用を制限する命令に対してのみ値予測を行う．スレッド間のレジスタ依存関係にある命令が，レジスタ値を必要とする前に値を受信するのであれば，その依存によって TLP の利用を制限することはない．本論文では，命令がレジスタ・フェッチを行う際に値を受信していなければ，そのスレッド間レジスタ依存は TLP の利用を制限しているとした．

スレッド間メモリ依存については，ロード命令のロード値を予測する．本論文では，メモリ依存関係にあるロード命令とストア命令が理想的にわかるとして，スレッド間レジスタ依存と同様，TLP の利用を制限するスレッド間メモリ依存に対してのみ値予測を行っている．

値予測に失敗した場合の回復方法として，スレッドを破棄する方法と，予測ミスした命令とその命令に依存した命令のみを選択的に再実行する方法がある．本論文では，値予測ミスから回復する方法として，投機 `send` の場合と同様に，予測した命令とその命令に依存した命令のみを選択的に再実行する方法を用いてる．

4. 値予測を考慮したスレッド分割

2 章で説明したように，従来の SKY のスレッド分割手法では，スレッド間のデータ依存関係にある命令間の距離に着目して，性能向上の見積りである利得を計算している．

例えば，注目しているスレッドでは，命令 `a` と命令 `b` がスレッド間レジスタ依存関係にあり，命令 `c` と命令 `d` がスレッド間メモリ依存関係にあるとする．本論文では，命令 `a` と命令 `b` のようにスレッド間のレジスタ依存関係にある命令間の距離をレジスタ依存距離，命令 `c` と命令 `d` のようにスレッド間のメモリ依存関係にある命令間の距離をメモリ依存距離とよぶ．注目しているスレッドのフォーク距離を fd，スレッド間の全てのレジスタ依存距離の中で最も短いレジスタ依存距離を rd，スレッド間の全てのメモリ依存距離の中で最も短いメモリ依存距離を md としたとき，fd，rd，md を計算しその中で最も小さい値を利得とする．

前章で説明した値予測を行う場合，従来の利得計算手法では次のような問題がある．従来の利得計算手法では，スレッド間データ依存の依存距離が短いスレッドは利得が小さくなる．そのため，TLP の利用を制限するようなスレッド間データ依存が存在するスレッドの利得は小さく，最終的にそのスレッドは選択されない．つまり，従来の利得計算手法では，TLP の利用を制限するスレッド間データ依存を含むスレッドをできる限り選択しないようにしている．

しかし，TLP の利用を制限するようなスレッド間データ依存が存在しても，値予測を行うことでその依存を解消できる場合があると考えられる．値予測で解消できるかもしれないスレッド間データ依存によって，スレッドの利得が制限されることは問題である．

上記の問題を解決するためには，レジスタ依存距離やメモリ依存距離の短いスレッド間データ依存を含むスレッドであっても，値予測を行うことでその依存が解消でき，性能が向上するならば，そのスレッドを選択できるようにする必要がある．そのために，利得計算においてデータ依存距離を計算する方法を変更する．

まず，全命令の各レジスタについて値予測精度のプロファイルをとる．例えば，命令 `b` で使用するレジスタが `r1` と `r2` であれば，`r1` の値予測ヒット率と `r2` の値予測ヒット率のプロファイルをとる．そして，以下の式を用いて，注目しているスレッドの全てのレジスタ依存距離に対して値予測精度で重み付けしたレジスタ依存距離 rd_{vpred} を計算する．

$$rd_{\mathrm{vpred}_i} = fd \times p + rd_i \times (1 - p)$$

ここで，fd はフォーク距離，rd_i はレジスタ i の値予測を行わない場合のレジスタ依存距離，p はスレッド間の依存関係にあるレジスタの値予測ヒット率である．同様に，メモリ依存距離についても以下の式を用いて値予測精度で重み付けしたメモリ依存距離 md_{vpred} を計算する．

$$md_{\mathrm{vpred}_i} = fd \times p + md_i \times (1 - p)$$

ここで，md_i はレジスタ i の値予測を行わない場合のメモリ依存距離である．値予測精度で重み付けをした rd_{vpred_i}，md_{vpred_i} の中で最も小さい値を，それぞれ値予測精度で重み付けしたレジスタ依存距離，メモリ依存距離とする．

このように，依存距離を値予測精度で重み付けすることによって，値予測が高い確率でヒットする依存に利得が制限されないようにする．データ依存距離の計算方法を変更することで，値予測を行うことで性能が向上するようなスレッドを選択できるようになると考えられる．

5. 評 価

本章ではまず，評価環境について述べる．次に，3 章で説明した投機手法を用いた場合の評価を行う．最後に，4 章で説明した値予測を考慮したスレッド分割の評価を行う．

5.1 評価環境

ベンチマーク・プログラムとして，SPECint2000 の bzip2, gcc, gzip, mcf, parser, vortex, vpr を，MediaBench の adpcm, epic, g721, jpeg, mpeg2, pegwit を使用した．MediaBench の各ベンチマークにはそれぞれエンコードとデコードがあり，ベンチマーク名の最後にそれぞれ "_enc", "_dec" を付けて表している．ベンチマーク・プログラムのバイナリは，GNU GCC Version 2.7.2.3(コンパイル・オプション: -O6 -funroll-loops) を用いて作成した．評価はトレース駆動シミュレータを用いて行った．トレースは SimpleScalar Tool Set Version 3.0 [2] を利用して採取した．

表 1　SKY の基本モデル

(a) プロセッサ

命令フェッチ幅	8 命令
命令デコード幅	8 命令
命令発行幅	8 命令
命令コミット幅	8 命令
命令ウィンドウ	64 エントリ
ROB	128 エントリ
LSQ	128 エントリ

(b) 共有資源

分岐予測機構	1024 エントリ，履歴長 4 の PAp 分岐予測ミスペナルティ 4 サイクル
命令キャッシュ	完全
データキャッシュ	完全
メモリ曖昧性検出機構	理想
値予測機構	128K エントリのハイブリッド値予測器

表 1 に SKY の基本モデルを示す．性能比較における基準プロセッサは，SKY を構成する 1 つのプロセッサとした．評価では，SKY のプロセッサ数は 8 とした．

評価したモデルを以下に示す．評価は 3 章で説明した投機 `fork`，投機 `send`，値予測およびそれらを組み合わせて用いた場合について行っている．

- base モデル: 投機手法を用いないモデル
- s モデル: 投機 `send` を用いるモデル
- f モデル: 投機 `fork` を用いるモデル
- v モデル: 値予測を用いるモデル
- fs モデル: 投機 `fork`，投機 `send` を用いるモデル
- fv モデル: 投機 `fork`，値予測を用いるモデル
- fsv モデル: 投機 `fork`，投機 `send`，値予測を用いるモデル

5.2　投機的スレッド実行の評価

図 3 に投機手法を単独で用いた場合の評価結果，図 4 に投機手法を組み合わせて用いた場合の評価結果を示す．図 3，4 において，縦軸は IPC を，横軸はベンチマーク名である．各ベンチマークにはそれぞれ 5 本の棒グラフがあり，図 3 では左から，基準プロセッサの IPC，base モデル，s モデル，f モデル，v モデルであり，図 4 では左から，基準プロセッサの IPC，base モデル，fs モデル，fv モデル，fsv モデルである．

投機手法を単独で用いた場合は，base モデルに対して s モデル，f モデル，v モデルそれぞれ平均で，SPECint2000 では 1.0%，2.8%，2.8%，MediaBench では 0.3%，6.0%，3.2%の性能向上しか得られなかった．投機 `send` や値予測を行いスレッド間データ依存関係の解消が早くなっても，性能は大きく向上していないと言える．これは，SKY のコンパイラではスレッド間データ依存に着目して，距離の短い依存をできる限り含まないようにスレッドを分割しているためであると考えられる．また，投機 `fork` を行いスレッドの生成が早くなっても，親スレッドから必要な値が送られてこないため，大きな性能向上は得られなかったと考えられる．

投機手法を組み合わせて用いた場合は，base モデルに対して SPECint2000 では fs モデル，fv モデル，fsv モデルそれぞれ平均で 8.5%，6.8%，12.7%，MediaBench では fs モデル，fv モデル，fsv モデルそれぞれ平均で 8.5%，9.2%，12.3%性能が向上した．この結果から，投機 `fork` に投機 `send` や値予測を組み合わせることにより，大きな性能向上を達成することができることがわかる．

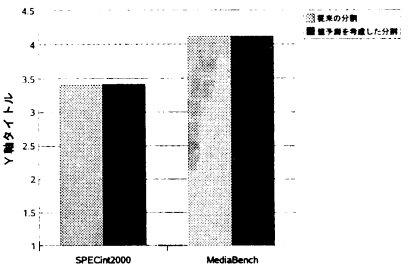

図 5　値予測を考慮したスレッド分割の評価

表 2　新たに選択されたスレッド

ベンチマーク	全スレッド数	新しいスレッド
bzip2	137	1
gcc	2323	107
gzip	101	1
mcf	33	1
parser	402	22
vortex	784	12
vpr	636	13

5.3　値予測を考慮したスレッド分割の評価

本節では，4 章で説明した値予測を考慮したスレッド分割について評価を行う．図 5 に評価結果を示す．評価では投機手法として値予測のみを行っている．この図において，縦軸は IPC，横軸は右側が SPECint2000，左側が MediaBench の平均である．2 本の棒グラフは，左側が従来のスレッド分割，右側が値予測を考慮したスレッド分割である．

図 5 からわかるように，値予測を考慮して利得計算を行いスレッド分割を行っても性能はほとんど向上しなかった．そこで，利得計算を変更したことによって，従来のスレッド分割には存在しなかったスレッドの数を調査した．結果を表 2 に示す．

表 2 から，全スレッド数に対して，従来のスレッド分割には存在しなかったスレッドの数は非常に少ないと言える．このことから，従来のスレッド分割と値予測を考慮したスレッド分割の性能がほぼ同じである原因は，値予測を考慮しても最終的に選択されるスレッドがほとんど同じためであると考えられる．

6.　ま と め

本論文では，単一チップ・マルチプロセッサ SKY において，制御依存とデータ依存を緩和する技術である投機 `fork`，投機 `send`，値予測について評価を行った．また，データ依存を緩和する技術である値予測を考慮したスレッド分割についても評価を行った．評価の結果，複数の投機手法を同時に用いることで，投機を行わない場合に対して，SPECint2000 で平均

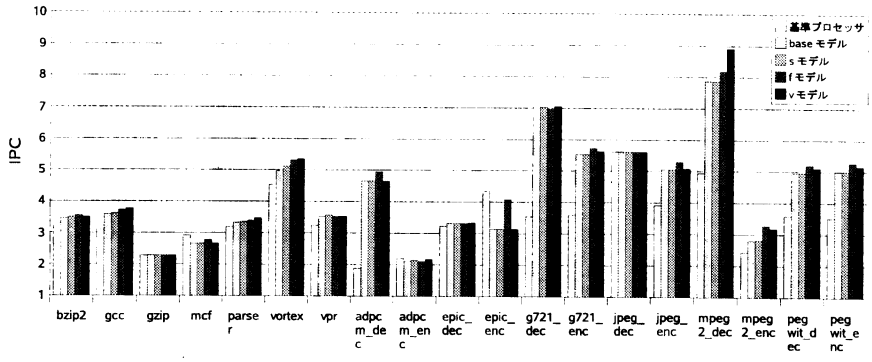

図 3　各投機手法の評価

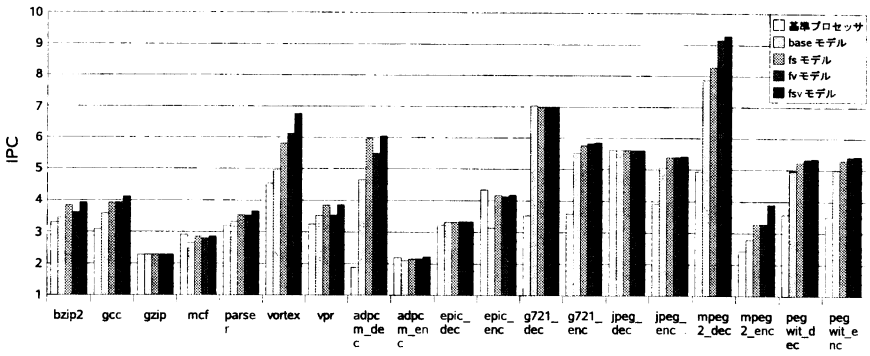

図 4　投機手法を組み合わせたときの評価

12.7%，MediaBench で平均 12.3%の性能向上を達成した．しかし，値予測を考慮したスレッド分割手法では，従来の手法に対してほとんど性能が向上しないことがわかった．

謝辞　本研究の一部は，文部科学省科学研究費補助金基盤研究（C）課題番号 15500036，文部科学省 21 世紀 COE プログラム，財団法人栢森情報科学振興財団研究助成の支援により行った．

文　献

[1] A. V. Aho, et al., *Compilers: Principles, Techniques and Tools*, Addison-Wesley Publishing Company, 1986.

[2] D. Burger, et al., "T.M.: The SimpleScalar Tool Set Version2.0,"Technical Report CS-TR-97-1342, University of Wisconsin, 1997.

[3] L. A. Barroso et al., "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," In *Proc. ISCA-27*, pp.282-293, June 2000.

[4] K. Diefendorff, "Power4 Focuses on Memory Bandwidth: IBM Confronts IA-64, Says ISA Not Important," *Microprocessor Report*, Vol. 13, No. 13, Oct. 1999.

[5] P. N. Glaskowsky, "IBM Previews Power5," *Microprocessor Report*, 9/8/03-02, Sep. 2003.

[6] P. N. Glaskowsky, "IBM Raises Curtain on Power5: More Details Disclosed at Microprocessor Forum," *Microprocessor Report*, 10/14/03-01, Oct. 2003.

[7] L. Hammond et al., "Data Speculation Support for a Chip Multiprocessor," In *Proc. ASPLOS-VIII*, pp.58-69, Oct. 1998.

[8] 岩田充晃ほか，"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 98-ARC-128, pp.127-132, 1998 年 3 月．

[9] 木村啓二ほか，"シングルチップマルチプロセッサ上での近細粒度並列処理，" 情報処理学会論文誌，Vol.40, No.5, pp.1924-1933, 1999 年 5 月．

[10] 小林良太郎ほか，"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 年 2 月．

[11] K. Krewell, "UltraSPARC IV Mirrors Predecessor: Sun Builds Dual-Core Chip in 130nm," In *Microprocessor Report*, 11/10/03-02, Nov. 2003.

[12] K. Krewell, "Fujitsu Makes SPARC See Double: SPARC64 VI Uses Process Shrink to Double Cores," In *Microprocessor Report*, 11/24/03-01, Nov. 2003.

[13] M. H. Lipasti, et al., "Value Locality and Load Value Prediction," In *Proc. ASPLOS-VII*, pp.138-147, Oct. 1996

[14] K. Olukotun et al., "The Case for a Single-Chip Multiprocessor," In *Proc. ASPLOS-VII*, pp.2-11, Oct. 1996.

[15] Y. Sazeides, et al., "The Predictability of Data Values," In *Proc. MICRO-30*, pp. 248-258, Dec. 1997.

[16] M. D. Smith et al., "Efficient Superscalar Performance Through Boosting," In *Proc ASPLOS-V*, pp.248-259, Oct. 1992.

[17] G. S. Sohi et al., "Multiscalar Processors," In *Proc. ISCA-22*, pp.414-425, June 1995.

[18] 鳥居淳ほか，"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 年 6 月．

[19] K. Wang, et al., "Highly Accurate Data Value Prediction using Hybrid Predictors," In *Proc. MICRO-30*, pp. 281-290, Dec. 1997.