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

2006－ARC－169 (24)
2006／8／2

時間軸分割並列マイクロプロセッサシミュレータの高速化手法

矢野聖宗[†] 中田尚[†]
津邑公暁[†,☆] 中島浩[†,☆☆]

集積回路技術の進歩に伴い，マイクロプロセッサの構造は高度化・複雑化している．高度なマイクロプロセッサの性能検証にはクロックレベルでのシミュレーションが不可欠であるが，現存するシミュレータは一般に低速であり，研究開発の大きな障害となっている．そこで我々は，シミュレーション過程を時間軸方向に分割，並列化することによるマイクロプロセッサのクロックレベルシミュレーションの高速化手法を提案している．並列シミュレーションは，分割点でのマシン状態を一致させること，もしくは分割された区間のシミュレーションの正当性をシミュレーション履歴によって検証することにより，精度を落とすことなく高速化を行う．しかし，正しいシミュレーションが実行されなかった場合には，シミュレーションやり直しのペナルティが発生する．本論文では，分岐予測ミス時のキャッシュ・TLB アクセスを論理シミュレーション実行時に部分的にシミュレートすることで，分割区間失敗回数を減らし，やり直しのペナルティを抑える方法を提案した．SPECfp95 を用いて評価した結果，16 ノード 56 分割の条件の下，従来方式と比べて最大 30%の実行時間の減少を達成することができた．また，従来最高 4.86 倍であった高速化率を 6.16 倍にまで引き上げることができた．

A Speedup Technique with Time-Division Parallel Microprocessor Simulator

Masahiro Yano,[†] Takashi Nakada,[†] Tomoaki Tsumura[†,☆]
and Hiroshi Nakashima[†,☆☆]

Microprocessor simulation is indispensable to design hardware systems. To estimate performance of highly sophisticated microprocessors, cycle accurate (or clock level) simulation is essential. However, existing simulators of out-of-order processors cost thousands times as long execution time as their targeting actual processors. The ultimate goal of our research is to develop a fast and accurate parallel simulator which is capable of microarchitectural modeling and system level simulation. We proposed a time-division parallel simulator in which each time interval is simulated in parallel with an approximated machine state at the beginning of the interval. The contribution of this paper is to improve the accuracy of the approximation of caches and TLB by partially simulating instruction sequences led by branch mispreditions. This technique reduces the number of *failures* in interval simulations, which is caused by incorrect state approximation and degrades performance due to the reexecution of the failed interval, up to 30% for SPECfp95 benchmarks simulated on a 16-node PC cluster. This improvement also achieved up to 6.16-fold speed-up that was 4.86-fold without the proposing technique.

1. はじめに

集積回路技術の進歩に伴い，マイクロプロセッサの構造は高度化・複雑化している．近い将来，組み込み機器等にも高度なマイクロプロセッサが用いられるようになると予想される．高度なマイクロプロセッサや，それらを用いた組み込み機器についての研究開発には，その機能や性能を前もって検証するためにシミュレーションが不可欠である．一般に，マイクロプロセッサのシミュレーションでは，命令の論理的な挙動だけをシミュレートする場合にはその実時間性能比（slowdown:SD）は 10～100 であるが，命令の実行順序を入れ換えて実行する out-of-order 実行や，複数の命令を同時に実行するスーパースカラ方式等をクロックレベルでシミュレーションする場合には SD は 1000～10000 となる．SD が 10000 の場合，実機では 1 分で終了する動作をシミュレートするために，約 7 日を要することになり，シミュレータの低速さが研究

† 豊橋技術科学大学
Toyohashi University of Technology
☆ 現在，名古屋工業大学
Presently with Nagoya Institute of Technology
☆☆ 現在，京都大学
Presently with Kyoto University

開発の効率化の大きな障害になっている．

さて，マイクロプロセッサの動作をシミュレートしていく過程において，ある時点でのパイプライン，キャッシュ等の状態を考えてみると，それらの状態は，その時点以前のシミュレーション過程すべてに依存しているわけではないことが分かる．たとえば，パイプラインでは分岐予測ミスが発生するたびにその状態は空，または空に近い状態となり過去との依存関係をほぼ失う．またキャッシュについて言えば，特定のセットは連想度に等しい数の異なるアドレスに対するアクセスがあれば，過去の状態に関わらず一定の状態となる．したがって，ある時点での状態や，その時点から別のある時点までの区間シミュレーションの結果は，要素と場合によっては，始めからシミュレートしなくても求められる．そのような場合には，区間ごとのシミュレーションを別々のノードで並列に行い，その結果を後で統合することでシミュレーション時間を大幅に短縮できる．

区間シミュレーション結果が正当なものであること，すなわち逐次的にシミュレートした場合と同じ結果が得られることを保証する必要ある．このためには，近似的に求めた各区間の初期状態が先行する区間の終了状態と一致するか否かを判定し，不一致であれば区間シミュレーションのやり直しを行わなければならない．

このような考え方に基づき我々は，マイクロプロセッサのシミュレーション過程を複数の区間に分割し，それらを別々のノードで実行することにより精度を全く落とさずに高速化を図る時間軸分割方式の並列シミュレーション方式を提案している．[2)]

並列シミュレーションにおいては，分割区間のやり直しのペナルティがシミュレーション時間を決定する重要な要素となる．たとえば，8ノードで8分割されたワークロードをシミュレーションする場合，分割区間失敗が1回生じると実行時間は約2倍となってしまう．そのため，分割区間の失敗数をより少なくする必要がある．そこで，本論文では分割区間失敗の主な原因である，分岐予測ミス時のキャッシュ・TLBへのアクセスを論理シミュレーションでも部分的に実施することで，分割区間失敗回数を削減し，並列シミュレーションを高速化することを目的とする．

以下，2章で時間軸分割方式の並列シミュレーション方式について述べ，3章で分割区間失敗を削減する方法について説明する．4章で評価を述べ，5章でまとめる．

2. 時分割並列シミュレータ

本章では，時分割並列シミュレータの概要を説明する．

時間軸分割による並列シミュレーションの概念図を**図1**に示す．図1では，マイクロプロセッサのシミュレーション過程を時間軸方向に4分割し，それぞれを別々のノードPC1~4で並列に実行する様子を表している．シミュレーション精度を保証するためには，それぞれの分割区間を正しくシミュレートしなければならないが，分割点でシミュレーション対象プロセッサの状態（**マシン状態**）が一致している場合には正しくシミュレートされる．

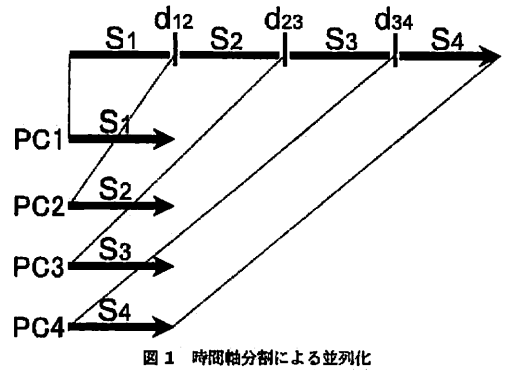

図1　時間軸分割による並列化

また，分割点d_{12}でマシン状態が多少違っていても，その違いが分割区間S_2のシミュレーションに影響を及ぼさない場合もある．このような場合には，分割区間シミュレーションの履歴の一部を保存しておけば，並列実行後に分割点でのマシン状態の違いによる影響を検証できる．検証の結果，分割区間シミュレーションに影響があった場合には，その区間を並列実行後にやり直すことで，全体の正しいシミュレーション結果が求められる．

なお，以降の説明では，高速化の対象としているクロックレベルの詳細なマイクロプロセッサシミュレーションを**詳細シミュレーション**，命令の論理的な挙動のみ（命令レベル）のシミュレーションを**論理シミュレーション**と呼ぶことにする．

2.1 マシン状態とその一致

本論文では，マシン状態としてメモリ・レジスタ・プログラムカウンタ・パイプライン・キャッシュ・TLB・分岐予測器を想定する．これらの状態が分割点で一致していれば，並列シミュレーションは正しく行われる．

以下，マシン状態のそれぞれの要素について分割点での状態一致を図る手法について述べる．

2.1.1 論理シミュレーション

メモリ・レジスタ・プログラムカウンタについては，並列シミュレーションを開始する前に，論理シミュレーションを行うことにより，分割点での状態を求めればよい．各ノードは論理シミュレーションを分割点まで行い，引続き詳細シミュレーションを行う．

シミュレート対象が単一プロセッサの場合，ロードストアを含むすべての命令はタイミング情報を必要とせず，命令レベルでシミュレートできるので，論理シ

ミュレーションによって分割点での完全な状態を求めることができる．また，論理シミュレーションはSDが小さく短時間で実行できるため，並列実行開始時刻におよぼす影響は少ない．なお論理シミュレーションでは，分割点でのキャッシュ・TLB・分岐予測器の状態を近似的に求めるために，これらの動作の命令レベルでのシミュレーション，すなわちout-of-order実行や投機実行を無視したシミュレーションも同時に行う．

2.1.2 重複実行

パイプラインについては，各ノードが一定区間，詳細シミュレーションを重複して行うことで状態一致を図る．

パイプラインは，分岐予測ミスが発生するとフラッシュされ，分岐方向および分岐先アドレスを間違えて実行した命令が消去される．パイプラインが完全にフラッシュされる場合であれば，そのたびにパイプラインは空になる．分岐方向や分岐先を間違えて実行した命令のみが消去される場合においても，分岐予測ミスのたびにパイプラインは空に近い状態となるため，予測ミスが繰り返されることによって過去との依存関係の多くを失う．したがって分割点でパイプライン状態が異なっていても，分割点の前後の区間シミュレーションを一定区間，重複して実行し，その区間で分岐予測ミスが何回か発生すれば，パイプライン状態が一致すると予想される．

2.1.3 履歴を用いた正当性検証方法

キャッシュ・TLB・分岐予測器の分割点での状態は，前述のように論理シミュレーションによって近似的に求めるが，投機実行を無視しているためその失敗に起因する状態変化が反映されていない．この近似状態と真の状態の不一致は，L2キャッシュのように状態数が大きいものでは少なからず存在し，また性能的に許容できる長さの重複実行では完全な一致を期待することはできない．

一方，これらの機構が保持する状態に部分的な差異があっても，その差異が参照されない，あるいは参照されても実行サイクルなどへの影響がないことがしばしばある．そこで，詳細シミュレーションの正当性検証は，キャッシュなどの状態の一致ではなく，詳細シミュレーション中で参照された状態に基づく動作の一致を確認することで行う．具体的には，キャッシュの連想度などにより定まる一定数の参照履歴を詳細シミュレーションの過程で保存し，これを区間終了後に前区間の末尾における正しい状態と照らし合わせ，アクセス遅延などの参照結果が全て等しければ正当であったと判定する．

2.2 並列シミュレーション

我々の研究では，精度を落とさず高速にマイクロプロセッサシミュレーションを行うことを目的としているので，分割区間シミュレーションが正しく行われなかった場合には，その区間のシミュレーションをやり直す必要がある．やり直しのためのシミュレーションは前区間を担当するノードが引続き詳細シミュレーションを行うことで対応する．

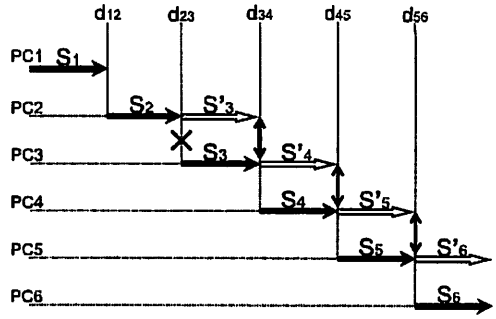

図2 並列シミュレーション（台数分割）

なお，分割区間シミュレーションが正しく行われた場合を**分割区間成功**，正しく行われなかった場合を**分割区間失敗**と呼ぶことにする．

2.2.1 並列シミュレーション方法（台数分割）

ここでは，分割数とPC台数が等しい場合の台数分割シミュレーションについて述べる．

台数分割での並列シミュレーションの様子を図2に示す．この方法ではS_3が失敗した場合に，PC2が再実行を行うのと同時に，それ以降のPCが分割区間失敗のあるなしに関わらず，次の区間をシミュレートする．そして，S'_3:S'_4，S'_4:S'_5，S'_5:S'_6の順に検証を行う．このような方法を用いると，失敗した分割区間シミュレーションの区間長が重複区間として活用され，再実行による各ノードの分割区間シミュレーションの成功率が飛躍的に増加する．例ではS_1 - S_2 - S'_3 - S'_4 - S'_5 - S'_6の区間を統合して，並列シミュレーションが終了している．

2.2.2 並列シミュレーション方法（多数分割）

ここではPC台数よりも多数に分割する多数分割シミュレーションについて述べる．多数分割では，分割区間長が短いため区間失敗時のやり直しのペナルティが小さくてすむ．一方，区間失敗の確率は必ずしも区間数に比例して増加するとは限らない．たとえば，ワークロードの特定の箇所で失敗する可能性が高い場合，区間長が多少変動しても失敗する区間数は同じであることが考えられる．そこでワークロードによっては，ペナルティが小さい多数分割が有利である可能性がある．

多数分割シミュレーションは，基本的に台数分割シミュレーションを繰り返す形で実行する．なお，始めの台数分割シミュレーションから順に第1，2，3...フェーズと呼ぶことにする．

多数分割での並列シミュレーションの様子を図3に示す．台数分割を単純に繰り返すと，フェーズの最後尾を担当するノードがやり直しに備えて次のフェーズ

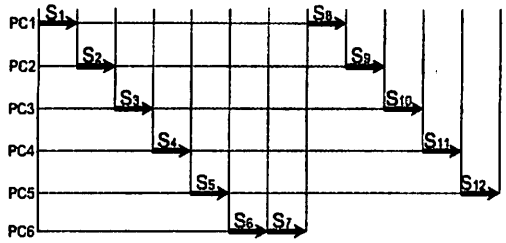

図 3 並列シミュレーション（多数分割）

に移行できない問題があるため，図 3 に示すように第 2 フェーズでは，PC6 は S_7 の 詳細シミュレーションを行うことにし，他 PC もそれに応じて詳細シミュレーション区間を後方にシフトさせる．このような方法をとることで，全ての PC が次フェーズのシミュレーションを開始することができる．また，S_7 については S_6 に引き続いて詳細シミュレーションが行われるため，分割区間シミュレーションの検証を行う必要がない．フェーズ数が増えた場合についても，同様の考え方を適用することでフェーズの境を意識することなく並列シミュレーションを実行することができる．

3. 分割区間失敗を削減する方法の提案

分割区間失敗は並列シミュレーションの性能に大きな影響を与える．本章では，分割区間失敗の削減方法について述べる．

3.1 概　　要

既存の並列シミュレータでは，1 回の分割区間失敗につき 1 区間の詳細シミュレーションをやり直すことを必要とするため，分割区間失敗を削減することは，そのまま性能の向上につながる．

そこで，論理シミュレーション時に分岐予測ミスが発生する箇所を予測し，間違った方向への命令列を一定区間実行することで，分割区間失敗を削減する．

なお，以降の説明では間違った方向への命令列のことを**ミスパス**と呼ぶことにする．

3.2 分割区間失敗の原因

分割区間失敗の主な原因は，分岐予測ミス以後からパイプラインがフラッシュされるまでの間に発生するキャッシュ・TLB へのアクセスである．たとえば，ダイレクトマッピングである命令 1 次キャッシュに対して，このようなキャッシュアクセスが発生した場合，対応するキャッシュラインにヒットする場合を除いて，キャッシュアクセスの前後でキャッシュ状態は変化する．論理シミュレーションでは，このようなキャッシュ状態の変化まではシミュレートできないので，詳細シミュレーションに切り替わった後に変化のあったキャッシュを参照すると分割区間失敗の原因となる．

3.3 分岐予測ミスの検出

2.1.1 節で説明したように，論理シミュレーションでは分岐予測器のシミュレートも同時に実行している．これを利用して，分岐予測ミスを起こす分岐命令を予測する．論理シミュレーションでは，以下の手順で詳細シミュレーションでの分岐予測器や投機実行の挙動を近似的に求める。

- 分岐予測
  命令レベルでシミュレートしている分岐予測器の動作に基づき，予測アドレスを近似的に求める．
- 予測成否の判定
  近似的に求めた予測アドレスと，実際の分岐先アドレスを比較し，一致していれば予測成功，そうでなければ失敗と判定する．
- ミスパスの実行
  分岐予測が失敗したときには，間違った分岐先アドレスからの論理シミュレーションを，次節で述べる方法により，終了条件が満たされるまで行う．

3.4 ミスパスの実行

詳細シミュレーションでは，ミスパスの実行は分岐命令が正しい分岐先を決定した時点で終了し，パイプラインに残っている分岐予測ミス以降の命令をフラッシュする．この間にフェッチされた命令により，命令キャッシュの状態は変化してしまう．さらに，ここでフェッチされた命令がロード命令であった場合は，データキャッシュに対するアクセスを発生させるかもしれない☆．

しかし，前節で述べたように論理シミュレーションでは，分岐予測ミスをすると予測した時点からミスパスの実行を行うため，何をもって実行を終了するのか定めなければならない．

そこで，ミスパスの実行を終了するための条件を以下のようにした．どちらかの条件が満たされた場合，ミスパスの実行を終了する．

- n 命令実行した場合には終了
  命令数 n だけ，ミスパスの実行を行う．
- 命令 1 次キャッシュがミスした場合には終了
  キャッシュをミスした場合には，より遠いメモリにアクセスするためのレイテンシが発生するため，詳細シミュレーションにおいては，フェッチが一時停止する．その間にミスパスの実行が終了することが考えられるため，命令 1 次キャッシュがヒットする間だけミスパスの実行を行う．

実際のミスパスがどこまで進むかを予測することはマシン構成にも依存するため，最大 n 命令で実行を終了するように設計し，n を様々に変化させて最適な値を調べることとする．ミスパス実行中に分岐命令をフェッチした場合については，分岐予測器を参照しその後の振る舞いを決定することにする．

☆ ミスパスの実行がどの程度まで進むかは，そのときのリソースの占有状況や，パイプラインの進み具合により，動的に決定される．

表 1 プロセッサの構成

命令発行幅			4	
RUU エントリ数			16	
LSQ エントリ数			8	
メモリポート数			2	
機能ユニット数	INT-ALU		4	
	INT-MUL/DIV		1	
	FP-ALU		4	
	FP-MUL/DIV		1	
分岐予測	予測方式		2bit カウンタ/2K エントリ	
	BTB		512 エントリ/4-way	
	RAS		8 エントリ	
メモリ	初期参照レイテンシ		18	
	バースト転送間隔		2	
TLB	命令		16 エントリ/4-way	
	データ		32 エントリ/4-way	
	ミスレイテンシ		30	
キャッシュ	容量	ラインサイズ	way 数	レイテンシ
L1 命令	16KB	32B	1-way	1
L1 データ	16KB	32B	4-way	1
L2 統合	256KB	64B	4-way	6

また，我々の以前の研究[2] より，データキャッシュの状態の一致率は比較的高いことが分かっているため☆，ミスパスの部分的なシミュレーションは，L1 命令，L2 統合キャッシュおよび命令 TLB を対象とするアクセスに限定した．

4. 評　　価

並列シミュレータは，SimpleScalar Tool Set Version3.0[1] を並列化することで実装されている．

シミュレーション過程の分割には，実行プログラムの命令数を利用した．すなわち，全実行命令数を N，使用する PC 台数を n，多数分割のフェーズ数を P とすると（$P=1$ ならば台数分割），一つの分割区間の命令数（分割区間長）l は $l=N/(nP)$ となる．

シミュレーション対象プロセッサモデルは，SimpleScalar のデフォルトモデルとした．主なパラメータを表 1 に示す．また，評価プログラムには SPEC CPU95 の fp を，データセットには train をそれぞれ用いた．ただし，実行時間の短い swim，applu，fpppp は対象外とした．

評価には，OS:Linux 2.4.31，CPU:Intel Xeon /2.8GHz，Mem:1GB，N/W:Gigabit Ethernet のクラスタを用いた．また，並列実行には MPI を用いた．

4.1 予備評価

4.1.1 分岐予測器の正確さ

実際の詳細シミュレーションで発生する分岐予測ミスを，論理シミュレーションでどの程度予測ができるか調査を行った．双方のシミュレーションの分岐予測ミスを起こした分岐命令を検出し，その一致率を調査したところ，95%～99%一致することがわかった．よって，論理シミュレーションでも分岐予測ミスを起こすであろう分岐命令を高い精度で発見できると言える．

4.1.2 ミスパス実行のオーバヘッド

提案手法では，論理シミュレーション時にミスパスを新たに実行するためのオーバヘッドが発生する．そこで，tomcatv，hydro2d においてミスパス実行をしなかったときに対する，ミスパス実行の命令数を 21 としたときのオーバヘッドを調査した．対象となる論理シミュレーション区間を調査した結果，tomcatv では最大 3.3%，平均 1.2%の，hydro2d では最大 2.2%，平均 0.9%のオーバヘッドが発生することがわかった．

4.2 提案手法の効果

4.2.1 分割区間失敗回数

PC 台数を 16，分割数を 56 とし，重複区間を 100 万命令としたときの分割区間失敗回数を図 4 に示す．ここでは，ミスパス実行の命令数を 0～21 まで変化させ，分割区間失敗回数がどのように推移するかを示している．対象とした 7 つの評価プログラムのうち，5 つにおいて分割区間失敗回数を削減することができた．図 4 では，効果がなかった tomcatv，hydro2d を除いて示している．tomcatv では 1 回，hydro2d では 4 回の分割区間失敗を確認することができた．

su2cor では，1 命令実行したとき 1 回分割区間失敗回数が減少し，以降，ミスパス実行命令数を 21 命令まで増やしても分割区間失敗回数の削減は見られなかった．それに対して，turb3d では，ミスパス実行命令数を 8 としたとき，分割区間失敗回数は 0 回となり，6 回の分割区間失敗を削減することに成功している．また，mgrid では 3 回の分割区間失敗を，apsi，wave5 では 5 回の分割区間失敗を削減している．wave5 では，分割区間失敗回数が単純に減少するのではなく，命令数 7～9 で一度増加している．これは，ミスパスの実行によって単調に減少していく分割区間失敗回数の他に，間違ったミスパスの実行によって増加する分割区間失敗回数があることを示している．

最も良い効果が得られた命令数は，評価プログラムによって多様であった．最短では su2cor での 1 命令から，最長で mgrid の 16 命令まで確認することができた．これは，命令発行幅の 4 倍の長さにあたることから，実際のミスパスの中でも 4 サイクル以上のフェッチがあることが予想される．

4.2.2 実行時間

最適なミスパス実行の命令数をとったときの，提案手法を用いなかった場合との実行時間比を図 5 に示す．提案手法の効果がなかった tomcatv，hydro2d においては，ミスパス実行の命令数によって実行時間が変化するので，実行時間が最長となるミスパス実行の命令数を 21 としたときの実行時間を用いて示している．

☆ 分割数を 16，重複実行を分割区間の 10%の命令数として並列シミュレーションを行うと仮定し測定した結果，キャッシュの状態一致率（全状態が完全に一致した区間数の全体に占める割合）は，平均 L1 命令:53%，L1 データ:98%，L2 統合:62%であった．

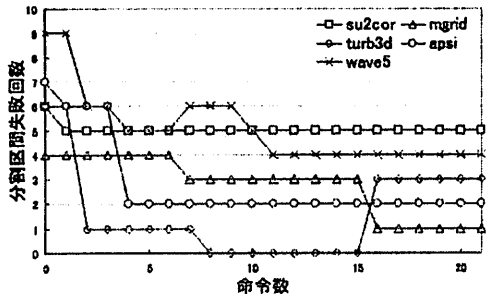

図 4　分割区間失敗回数の推移

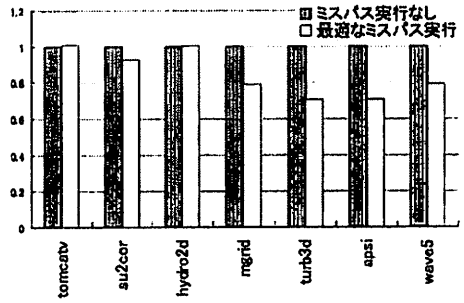

図 5　従来方式との比較

図に示すように，最も実行時間が減少したのは turb3d の 30%減少であり，次いで apsi の 29%減少となった．図 4 に示したように，分割区間失敗をより多く削減したものが，実行時間をより減少させていることがわかる．しかしながら，分割区間失敗が 5 回減少した wave5 と，3 回減少した mgrid とで 21%の減少と，同程度の実行時間の減少となっている．これは，総命令数を一定の命令数に分割することで分割区間を決定したため，各ノードでの詳細シミュレーションの実行時間に差が出たことに起因する．すなわち，同じ命令数であっても，パイプラインの混雑度などにより詳細シミュレーション時間が異なるので，ここでの実行時間差はフェーズ間で同期をとるときに，同期オーバヘッドとして現れ，全体の実行時間に影響する．

オリジナルの SimpleScalar に対する高速化率を図 6 に示す．図に示すように，mgrid では，従来 4.86 倍であった高速化率が，6.16 倍となり最も高い値を達成できた．次いで，turb3d が 5.6 倍の高速化率となった．

なお今回の評価では，ミスパスの実行命令数をベンチマークごとに最適なものとしたが，実際の応用ではたとえばターゲットのマイクロアーキテクチャを勘案した上で固定的な値とする必要がある．図 4 に示した結果から，今回使用したマイクロアーキテクチャでは実行命令数を 10 程度とするのが適切であると考えられるが，その場合の実行時間の評価や命令数の決定方法は，今後の課題である．

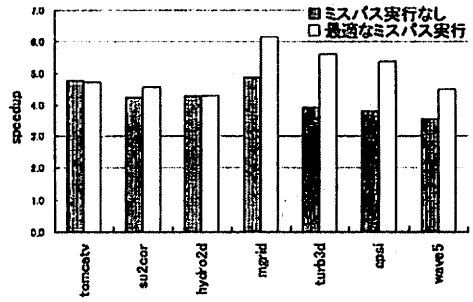

図 6　高速化率

5. おわりに

本論文では，マイクロプロセッサシミュレーション過程を時間軸分割し並列実行するシミュレータにおいて，その分割区間失敗回数を削減する手法について提案した．

提案手法を既存の時分割並列シミュレータに適用し，PC 台数を 16，分割数を 56，重複区間を 100 万命令として SPECfp95 を用いて評価を行った．その結果，従来方式と比較して最大 30%の実行時間を減少させることができた．また，従来方式では最高 4.86 倍であった高速化率を 6.16 倍にまで引き上げることができた．

今後の課題として，トレースに基づき実行時間ベースで分割区間を決定するなど，ワークロードの性質に合わせた分割方法について検討することが挙げられる．また，我々の研究グループで提案している論理シミュレーション高速化技法[3] を用いて，論理シミュレーション部分を高速化し，全体を高速化することも検討する必要がある．

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 (B)，研究課題番号 17300015，「高度情報機器開発のための高性能並列シミュレーションシステム」）による．

参考文献

1) Austin T., Larson E., and Ernst D.: SimpleScalar: An Infrastructure for Computer System Modeling, *Computer*, Vol.35, No.2, pp.59-67, (2002).
2) 高崎透，中田尚，津邑公暁，中島浩：時間軸分割並列化による高速マイクロプロセッサシミュレーション，情処論 ACS, 46-SIG12 (ACS11), pp. 84-97 (2005).
3) 中田尚，津邑公暁，中島浩：ワークロード最適化シミュレータの設計と実装，情処論 ACS, 46-SIG12 (ACS11), pp. 98-109 (2005).