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

2006－ARC－167（34）
2006－HPC－105（34）
2006／3／1

時分割マイクロプロセッサシミュレーションにおける最適な分割数の調査

中田　尚[†]　高平　剛[†]
津邑公暁[†]　中島　浩[†]

高度な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，並列化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．並列化はシミュレーション過程を時間軸方向に分割することにより行い，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．本論文では，時間軸分割時の分割数や重複区間長がマシン状態の一致率に与える影響を調査した．その結果，重複区間長は 1000 命令程度で十分であること，分割数を増やしても失敗数の増加はわずかであり，分割数を増やすほど一致率は向上することがわかった．性能モデルを用いて 16 ノードにおける性能を予測したところ，SPECfp95 では 1000 万から 2000 万命令で分割することによりほとんどのベンチマークでほぼ 8 倍の高速化率が期待できることがわかった．

Investigation of Optimal Interval Length for Time-Division Parallel Microprocessor Simulation

Takashi Nakada,[†] Tsuyoshi Takahira,[†] Tomoaki Tsumura[†]
and Hiroshi Nakashima[†]

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 parallelization for microprocessor simulation in which the series of intervals of a workload execution are simulated in parallel. The correctness of the parallel simulation is partially kept by *logical* in-order simulation preceding each out-of-order interval and overlapping intervals. Then the correctness is fully assured by examining the initial and final states of adjacent intervals and by executing the successor again if they disagree. We investigated relation of the agreement rate to the lengths of an interval and its overlapped region to optimize them. We found that the rate is maximized with 10 to 20 million instruction intervals and overlapping execution of 1000 instructions is sufficient to keep the rate high. Then we estimated the performance of SPECfp95 simulation on 16 PC cluster using a performance model and optimized lengths to find parallel speedup will be about 8-fold.

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，out-of-order 実行や，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 となり，シミュレータの低速さが研究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

† 豊橋技術科学大学
Toyohashi University of Technology

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．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，途中からシミュレーションを開始しても求められる．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によりシミュレーション時間を大幅に短縮できる．

なお，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がある．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法を提案している[3]．

並列シミュレーションにおいては，シミュレーション対象の分割方法が性能に非常に大きな影響をおよぼす．それにもかかわらず，その分割方法についてはこれまでほとんど調査されていない．そこで，本論文では分割方法による実行速度の違いを考察し，最適な分割方法を発見することを目的とする．

以下，2 章では時分割並列化シミュレーションの概要を述べ，3 章で分割方法と高速化率の関係を説明する．4 章で最適な分割方法を考察し，5 章でまとめる．

2. 時分割並列化シミュレーション

高速化手法の基本的なアイデアは，シミュレーション過程を時間軸方向に分割し，それぞれを並列に実行することである．

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落とさないためには，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**）を比較し一致している場合には正しくシミュレートされる．

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

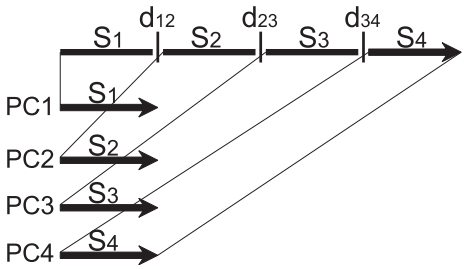

図 1 時間軸分割による並列化

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．マシン状態の比較と履歴の検証の結果，一致が確認できなかった場合には，その区間を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，本研究で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

各ノードはそれぞれの詳細シミュレーション開始地点まで論理シミュレーションを行う．論理シミュレーションでは命令エミュレーションに加え，キャッシュ・TLB と分岐予測器のシミュレーションを行う．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．

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しないので，メモリ・レジスタ・プログラムカウンタについては，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．

その他のマシン状態は論理シミュレーションのみでは完全な状態を求めることが難しい．そこで，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マシン状態の一致を図る．

2.1.2 重 複 実 行

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．分岐予測ミスのたびにパ

イプラインは空また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，重複実行で分岐予測ミスが数回発生すれば，パイプライン状態が一致すると予想される．

2.1.3 履歴による正当性の検証

キャッシュ・TLB・分岐予測器においてはパイプラインにおいての分岐予測ミスのような過去と依存関係を大幅に断ち切るイベントがないので，重複実行だけでは完全に一致するとは限らない．例えば，実行プログラムによってはあまり参照されないキャッシュブロックも存在し，場合によってはシミュレーションの終了まで一致しないブロックが存在することもある．したがって，一定区間を重複実行し，大半のキャッシュブロックを一致させた後，見切りで詳細シミュレーションを開始する．さらに，詳細シミュレーション中に発生したキャッシュ・TLB アクセスや分岐予測の履歴を用いて，相違点を参照したかどうかを調べ，シミュレーションの正当性を検証する．

ノード間で分割点におけるマシン状態が異なる場合でも，その違いが分割区間シミュレーションに影響をおよぼさないことが確認できればシミュレーションが正しく行われたことを保証できる．

履歴が正しいかどうかを検証するためには，分割点の前区間終了時の正しい状態に後区間の参照履歴を適用し，その結果が同じであるかどうかを調べる．

S_1 終了時だけは，始めから詳細シミュレーションを行うため正しいマシン状態を保持している．よって先頭の区間から順に正しいマシン状態を求め，履歴の検証を行うことができる．

2.2 並列シミュレーション

本研究では，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．そのため，状態比較と履歴の検証の結果，分割区間シミュレーションが正しく実行されなかったことが判明した場合は，分割区間シミュレーションのやり直しを行うことになる．やり直しのシミュレーションは，前区間を担当するノードが引き続き詳細シミュレーションを行うことで対応する．

以降の説明では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)

まず，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．

この場合は**図 2** のように並列シミュレーションを行う．ここで，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．なぜならば、分割点における正しい状態は，先頭の区間から順に決定し，1 回でも分割区間失敗が発生すると，それ以降の検証を正しく行うことができなくなるからである．その結果，この例では S'_3，S'_4，S'_5，S'_6 の順に検証を行う．

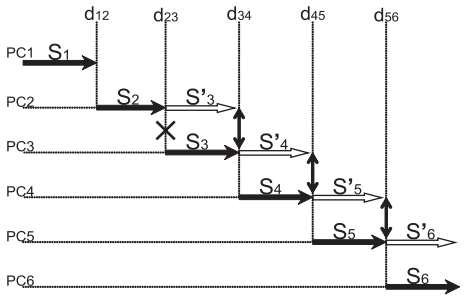

図 2 並列シミュレーション

2.2.2 並列シミュレーション方法 (多数分割)

次に，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分割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，分割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．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くり返す形で実行する．また，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．

ただし，第 2 フェーズでは**図 3** に示すように，PC6 が S_7 の詳細シミュレーションを行うことにし，他の 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．この結果，第 2 フェーズの最初の区間である S_7 の実行は，その正しい開始状態を保持している PC6 によって行われるため，確実に成功する．すなわち第 2 フェーズでは，PC6 が第 1 フェーズの PC1 の役割を果たし，PC1~5 はそれぞれ第 1 フェーズの PC2~6 の役割を果たすことになり，第 1 フェーズと同じ考え方で並列シミュレーションを実行することができる．

3. 分割区間失敗率と高速化率

本章では，分割区間失敗率と高速化率の関係を考察するために，並列シミュレータの性能モデルを考える．

逐次詳細シミュレーションの実行時間を T_S，論理シミュレーションと詳細シミュレーションの実行時間比を $R_{D/L}$，1 区間の結果検証のための状態・履歴の保

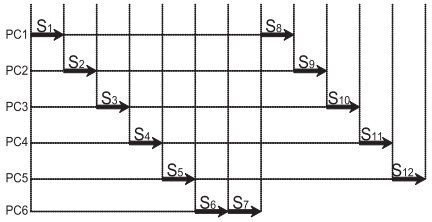

図 3 多数分割方法

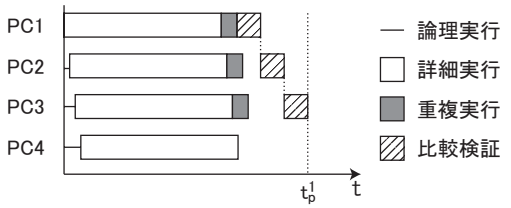

図 4 性能モデル (1)

存・比較に要する時間を T_V とし，n 台のノードで全区間数 N を並列化シミュレーションを行うとする．重複区間長は区間長に定数 α を乗じた長さとする．

並列シミュレーションは一般に複数のフェーズに分かれており，現在の設計ではフェーズ間では明示的な同期を取る．そこで，以降ではフェーズ毎の性能モデルを求め，それを必要なフェーズ数だけ合計することにより全体の性能モデルとする．

3.1 1 フェーズあたりの実行時間

はじめに，論理シミュレーションの実行速度が十分に高速な場合について考え，その後に論理シミュレーションの実行時間が無視できない場合について考える．

まず，**図 4** に示すように比較検証時間が論理シミュレーション 1 区間分の実行時間よりも長い場合について考える．このとき，逐次的にしか実行できない比較検証時間が並列実行全体の時間を規定する．全く失敗がない時の 1 フェーズあたりの並列実行時間 t_P^1 は下式により近似できる．

$$t_P^1 = T_S \frac{1+\alpha}{N} + T_V(n-1) \tag{1}$$

上式の第 1 項は第 1 区間の詳細シミュレーション区間が全体の $1/N$ であり，α だけ重複実行を行うことを意味している．また第 2 項は，以降の区間について比較検証が必要であることを示している．T_V は対象アーキテクチャによって定まりワークロードには依存しない定数と考えることができる．

次に，**図 5** に示すように比較検証時間が論理シミュレーション 1 区間分の実行時間よりも短い場合について考える．このとき，1 フェーズあたりの実行時間は最終ノードまたはその 1 つ手前のノードの実行時間に

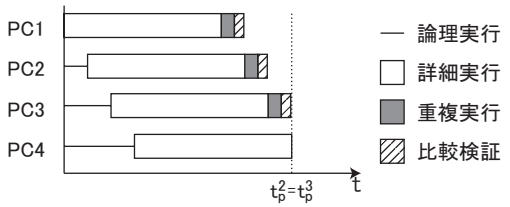

図 5 性能モデル (2)

よって決定する．これらのノードの実行を比較すると，最終ノードは論理シミュレーションの実行時間が長いが，次のノードのための重複実行時間と比較検証時間が不要であることがわかる．そして，これらの実行時間によって最終ノードとその直前ノードのどちらが並列実行時間を決定するかが決まる．図 5 の例はこれらの実行時間が等しい場合を示している．以上より，最終フェーズの実行時間は下式で近似できる．

最終ノードの直前のノードが実行時間を決定するとき

$$t_P^2 = T_S\left(\frac{1+\alpha}{N} + \frac{n-2}{NR_{D/L}}\right) + T_V \tag{2}$$

最終ノードが実行時間を決定するとき

$$t_P^3 = T_S\left(\frac{1}{N} + \frac{n-1}{NR_{D/L}}\right) \tag{3}$$

式 (1) と式 (2) の比較からこれら 2 式の閾値は，$T_V = T_S/NR_{D/L}$ であることがわかる．これは，1 区間の論理シミュレーションの実行時間を比較検証時間で隠蔽できれば，論理シミュレーションの実行時間を無視できることを示している．さらに，分割数 N が大きくなると 1 区間の論理シミュレーションが短くなるので，比較検証時間が支配的となることを示している．また，式 (2) と式 (3) の比較からこれら 2 式の閾値は，$T_V + T_S\alpha/N = T_S/NR_{D/L}$ である．

3.2 並列シミュレーションの実行時間

次に並列シミュレーション全体の性能モデルを検討する．

N 区間を n 台の PC で並列シミュレーションし k 回失敗した場合，フェーズ数 P は $P = \lceil (N-k)/n \rceil$ となる．説明を簡単にするために，分割区間失敗はすべて最終フェーズで発生すると仮定する．これにより，最終フェーズ以外の実行時間は前節の式で近似でき，最終フェーズの区間数 n_f は $n_f = N - n(P-1)$ となる．

最終フェーズで k 回失敗した場合の実行時間は下式で近似できる．

比較検証時間が長いとき

$$t_{Pf}^1 = T_S \frac{1+k+\alpha}{N} + T_V(n_f - 1) \tag{4}$$

最終ノードの直前のノードが実行時間を決定するとき

$$t^2_{Pf} = T_S\left(\frac{1+k+\alpha}{N} + \frac{n_f-k-2}{NR_{D/L}}\right) + T_V(1+k) \quad (5)$$

最終ノードが実行時間を決定するとき

$$t^3_{Pf} = T_S\left(\frac{1+k}{N} + \frac{n_f-k-1}{NR_{D/L}}\right) + T_V k \quad (6)$$

それぞれの式は式 (1) から式 (3) に対応する．ただし，$(n_f-1)=k$ のときには最終ノードの 1 つ手前のノードが存在しないため，式 (5) ではなく式 (6) となる．

以上のことから，並列シミュレーション全体の実行時間 T_P は下式で近似できる．

$$T_P = (P-1)\max_i t^i_P + \max_j t^j_{Pf} \quad (7)$$

3.3 パラメータの仮定

論理シミュレーションと詳細シミュレーションの実行速度比 $R_{D/L}$ は従来 4.2 程度であった[3]．その後，我々の研究[2] により sim-cache を平均で 8.3 倍高速化できることがわかっている．本シミュレータの論理シミュレータは sim-cache に分岐予測器を加えたものであるので，$R_{D/L}$ は 30 程度になることが予想される．比較検証時間 T_V は 0.5 秒を仮定し，ノード数 n は 16 を仮定する．

次章では，以上を基に最適な分割方法を検討する．

4. 最適な分割方法

本章では，さまざまな分割方法の予測高速化率を求め，その中から最善の分割方法を決定する．

4.1 測 定 条 件

並列シミュレータは，SimpleScalar Tool Set Version 3.0[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．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，実行時間の差を少なくするために，実行時間の長い ‘hydro2d’, ‘mgrid’, ‘su2cor’, ‘tomcatv’, ‘turb3d’ についてベンチマーク中のイタレーション数を削減し，実行時間の短い ‘fpppp’, ‘applu’ は対象外とした．この結果，逐次シミュレーションの実行時間は 2114 秒 (swim) が最短となり，その他のベンチマークは 6042 秒 (apsi) から 17290 秒 (su2cor) の範囲となった．

評価には，OS:Redhat Linux 7.2, CPU:Mobile Intel Pentium III-M/866MHz, Mem:512MB, N/W:Gigabit Ethernet のクラスタを用いた．並列実行には LAM/MPI を用いた．

前述の通り論理シミュレーションを高速化することができることはわかっている．しかし，本シミュレータへの実装がまだ完了していないため，高速化率を実際に測定することはできない．そこで，シミュレーションモジュールの実行速度には依存しない失敗回数を測定し，その結果から各モジュールが高速化された場合のシミュレーション全体の高速化率を予測することとした．

表 1 プロセッサの構成

命令発効幅			4	
RUU エントリ数			16	
LSQ エントリ数			8	
メモリポート数			2	
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	
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

多数分割方式ではフェーズの境界では比較検証を行わないので，MPI の論理プロセスを多数生成することで，仮想的に台数分割を行った．これにより，すべての区間で比較検証が行われ，より正確な比較検証失敗回数を測定することができる．

4.2 分割区間長と重複区間長

まず，分割区間長と重複区間長をそれぞれ変化させた場合に失敗率がどのように変化するかを測定した．

分割区間長が高速化率に与える影響を考える．分割区間長を短くすると，分割数も多くなる．すると比較検証回数も増加するため，必然的に検証失敗回数も増加することが考えられる．しかし，比較検証回数の増加率よりも失敗回数の増加率が小さければ，つまり失敗率が低下すればやり直しのオーバヘッドが少なくなる可能性がある．

次に，重複区間長が高速化率に与える影響を考える．重複区間長を長くすることで，一般に検証失敗回数は減少すると考えられる．その一方で重複区間は並列実行にとってオーバヘッドとなる．つまり検証失敗回数が増加しない範囲で重複区間長は短い方が良い．

‘swim’, ‘wave5’, ‘apsi’ について分割区間長と重複区間長を変化させたときの高速化率の変化を測定した．結果を**図 6** に示す．ただし，それぞれのベンチマークの全実行命令数が異なるので，分割区間長が等しくても分割数が異なり，比較検証回数も異なっている．

図 6 の結果から，分割区間長を短くし分割数を増やすほど失敗率は減少し，やり直しのオーバヘッドを削減できる可能性が高いことがわかる．つまり，比較検証のオーバヘッドが顕在化しない範囲でできる限り分

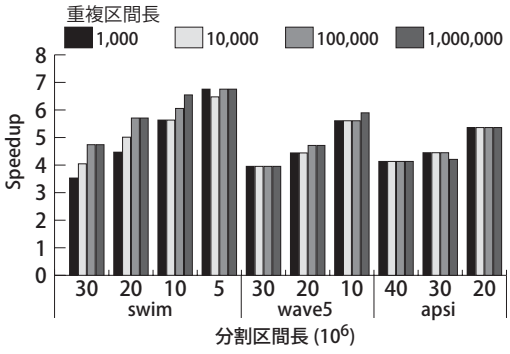

図 6 分割方法と高速化率

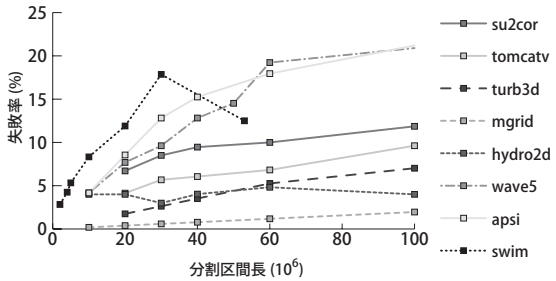

図 7 失敗率（重複 1000 命令）

割区間長を短くすれば良いといえる．

また，分割区間長が短い場合には重複区間長による失敗数の変化はわずかであるので，分割区間長を短くすれば重複区間長は 1000 命令と短くても良いことがわかる．1000 命令であれば重複のオーバヘッドは十分無視できるので，これ以上短くする必要性はない．したがって，重複区間長は 1000 命令が十分最適に近いといえる．

次に，重複区間長を 1000 命令に固定し，分割数の変化による高速化率の変化を詳しく測定した．失敗率の測定結果を**図 7** に，高速化率の予測結果を**図 8** に示す．

図 7 の結果から，ほとんどのベンチマークで分割区間長が短くなるにつれ，失敗率が低下していることがわかる．‘swim’ の 5300 万命令において失敗率が大きく低下しているが，これは分割数が 16 と少なかったため，例外的に失敗率が低下したと考えられる．また，‘hydro2d’ では失敗数が分割数にほぼ比例して増えたために，失敗率はほぼ一定になっている．

図 8 の結果より，失敗率の低下ともに高速化率が向上していることがわかる．1000 万から 2000 万命令で分割することにより，多くのベンチマークで 8 倍近くの高速化を達成している．‘swim’ では分割区間長を非常に短くしたときに高速化率が急激に低下しているが，

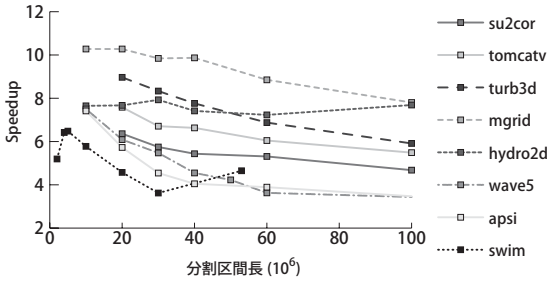

図 8 高速化率（重複 1000 命令）

これは細かく分割しすぎたために，比較検証のオーバヘッドが顕在化したためと考えられる．

5. ま と め

本論文では，シミュレーション過程を時間軸分割した並列マイクロプロセッサシミュレータの性能モデルを求め，最適な分割方法を調査した．

16 ノードの PC クラスタを仮定し，分割区間長と重複区間長を変化させたときの高速化率を予測した．SPEC CPU95 を用いて評価した結果，重複区間長は 1000 命令で十分であることがわかった．分割区間長を 1000 万から 2000 万命令とすることで，ほとんどのベンチマークでほぼ 8 倍以上の高速化が達成できることがわかり，並列化のオーバヘッドが顕在化しない範囲で細かく分割することにより，検証失敗時のオーバヘッドを削減できることがわかった．

今後の課題として，論理シミュレータの高速化を本シミュレータに実装することが緊急的なものとして挙げられる．

謝辞 本研究は，文部科学省 21 世紀 COE プログラム「インテリジェントヒューマンセンシング」および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」）の支援によって行われた．

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) 中田尚, 津邑公暁, 中島浩: ワークロード最適化によるキャッシュシミュレータの高速化, 情報処理学会研究報告 2005-ARC-164, SWoPP 2005, pp. 97–102 (2005).
3) 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性能マイクロプロセッサシミュレーション, 先進的計算基盤システムシンポジウム SACSIS2005, pp. 339–348 (2005).