2004－ARC－159　(16)
2004／7／31

高性能マイクロプロセッサシミュレータの並列化による高速化

高崎　透†　中田　尚†　中島　浩†

集積回路技術の進歩に伴いマイクロプロセッサは高度化・複雑化しており，性能の検証には cycle accurate なシミュレータが不可欠である．しかし，現状のシミュレータは一般に低速である．

そこで，本論文ではシミュレーション過程を時間軸上で分割し，並列シミュレーションをおこなうことで高速化を図る．シミュレーションの時分割で問題となるのは，分割点においてシミュレーション対象マシンの状態を一致させることである．本論文で提案する手法では，パイプラインが分岐予測ミス時に空あるいはそれに近い状態となることと，キャッシュの各セットは一定数のアクセスにより一定の状態となることを利用する．また分割点の後の一定区間を複数のシミュレーションノードで重複実行し，区間終了時の状態及びキャッシュの参照履歴をチェックすることにより正しいシミュレーションをおこなう．シミュレーションの分割数を 16，重複区間を分割区間の 10%と想定した場合，SPECfp95 107.mgrid では，パイプライン，キャッシュの状態が一致し，キャッシュ参照履歴は SPECfp95 において平均 82%，SPECint95 では平均 65%であることが分かった．

High Speed Simulation of High Performance Processor by Parallel processing

Toru Takasaki,† Takashi Nakada† and Hiroshi Nakashima†

Microprocessor simulation is indispensable for hardware systems design. In systems design field, cycle accurate (or clock level) simulation of highly sophisticated microprocessor is required. However, existing simulatiors of out-of-order processors run programs thousands times as slow as actual hardware. 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.

Our parallel simulation is performed by decomposing the problem along time axis so that each simulator node works on a subdivided interval. To make the out-of-order simulation for an interval correct, we perform *logical* in-order simulation for the preceding interval in advance. Moreover, two adjacent intervals are overlapped so that the machine state derived from the logical simulation matches that of the real out-of-order simulation. The correctness is verified by comparing pipeline states at the end of the overlapped interval, and by examining cache access trace at the end of the simulation interval. Our experience with SPEC CPU 95 proves the adequateness of our proposal showing that 107.mgrid is correctly simulated when we subdivide the simulation into 16 intervals with 10% overlap. It is also shown that the statistical correctness of SPECfp95 and SPECint95 are 82% and 65% in average respectively.

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のシミュレータが不可欠である．しかし，現状ではシミュレータは一般に低速であり，最も簡単なユニプロセッサのアーキテクチャシミュレーションでさえ実時間性能比 (SD) は 1000〜10000 となっている．また，高度なワークロードやマルチプロセッサを含む複雑なシステムのシミュレーションではこの数倍 ～ 数十倍の時間を必要とするため，研究・開発の効率化の大きな障害になる．

高性能マイクロプロセッサのためのアーキテクチャシミュレーションの SD が 1000〜10000 という大きな値になる要因は，命令実行順序を動的に定める命令スケジューラのシミュレーションに要する時間コストが大きいことにある．実際，命令の論理的な挙動のみのシミュレーションであれば SD は 10〜100 のオーダであり，いかにパイプラインスケジューラの計算量が実

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

行時間の多くを占めているかがわかる．

マイクロプロセッサシミュレーション過程のある時点においての，パイプライン，キャッシュ，分岐予測の状態は，過去のシミュレーション結果すべてに依存しているわけではない．たとえばパイプラインの状態は分岐予測ミス発生の度に空または空に近い状態となり，過去の状態を失う．またキャッシュの特定のセットは，連想度に等しい数の異なるアドレスによ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，ある時点でのシミュレーション対象マシンの状態を知るのに必要なのは，その時点の状態に影響を与える過去の情報である．その時点に影響を与えない区間があれば，その間は論理動作によるシミュレーションをおこなえばよい．

本研究では，命令スケジューラを含む詳細なシミュレーションを時間軸上で分割し，それぞれを並列に実行することにより高速化を図る．また，分割点の後の一定区間を複数のシミュレーションノードで重複実行し，区間終了時の状態を及びキャッシュの利用履歴をチェックすることにより正しいシミュレーションをおこなう．

2 章でシミュレーションの種類，3 章で高速化手法，4 章で並列シミュレーションの設計 5 章で実現可能性の評価について述べる．

2. シミュレーションの構成

マイクロプロセッサシミュレータはプロセッサの論理挙動のみをシミュレーションする場合と，パイプラインを含めた詳細なシミュレーションをおこなう場合とに分けられる．以後，それぞれを**論理シミュレーション**，**詳細シミュレーション**と呼ぶことにする．

詳細シミュレーションは分岐予測，キャッシュ，パイプラインのシミュレーションに分けて考えられる．

2.1 命令エミュレーション

命令エミュレーションは，命令スケジューリングをおこなう必要はなく論理動作で実行することができる．ユニプロセッサのシミュレータは，ロードストアを含むすべての命令のタイミングに依存せずに実行することができる．

2.2 分岐予測シミュレーション

分岐予測をシミュレーションするにはタイミング情報は不要である．したがって，基本的に論理動作で実行できる．

2.3 キャッシュシミュレーション

キャッシュシミュレーションでは，キャッシュへのアクセスタイミングからレイテンシを求めるが，キャッシュのアクセス順序からキャッシュのヒット/ミスを求めるだけならば，論理動作で実行できる．

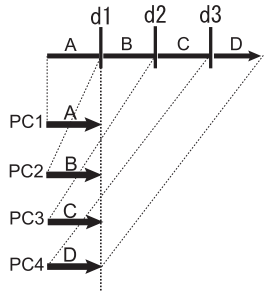

図 1 シミュレーションの時間軸分割

2.4 パイプラインシミュレーション

パイプラインシミュレーションでは命令の詳細なスケジューリング計算を行う．したがって，パイプラインシミュレーションは 詳細な実行を行う必要がある．

3. 高速化手法

3.1 マシン状態

本論文で提案するシミュレータの高速化の基本原理は，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にある．

時間軸分割 (図 1) では，分割シミュレーションを統合する際にマシン状態が一致しているかどうかが問題となる．次節でこれについて述べてゆく．

3.1.1 パイプライン

パイプラインの状態一致問題に関しては，分岐予測ミスを利用する．分岐予測ミスが発生すると，パイプラインはフラッシュされ，分岐方向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の状態は空となり，各並列シミュレーションノードで一致する．分岐方向を間違えて実行された命令のみが消去される場合においても，分岐予測ミスの度にパイプラインは空に近い状態になり過去のシミュレーション結果との依存関係を失ってゆく．よって，分割点の後の一定区間を複数のシミュレーションノードで重複実行することによりパイプライン状態の一致が予想される．また，分割点を分岐予測ミス点とすることで速やかな状態一致が期待される．

3.1.2 キャッシュ・分岐予測

キャッシュと分岐予測の状態一致問題をを解決する方法として，論理動作によるキャッシュと分岐予測シミュレーションを用いる．分割点から開始するシミュレーションにキャッシュと分岐予測の情報を与えるためである．

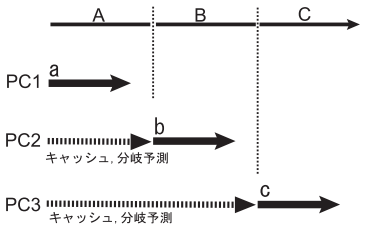

図 2 論理シミュレーションの利用

1 章で述べた通りシミュレータの実行時間の大部分は動的に実行順序を定める命令スケジューラのパイプラインシミュレーションに要する時間である．よって，論理動作によるキャッシュと分岐予測シミュレーションは詳細シミュレーションと比較して実行時間が短い．論理シミュレーションによってキャッシュ・分岐予測の情報を得れば，各分割区間シミュレーションの開始時刻をほぼ同時にできる．

図 2 において例を上げて説明する． 図中の最上部の矢印が，通常のシミュレーション過程を表している．このシミュレーション過程は，時間軸分割によって，A, B, C の区間に分割される．分割された区間の B, C は時間軸上の途中から開始するシミュレーションである．そこで，分割箇所でのマシン状態一致のために，図の点線の矢印で表されるように論理動作によるキャッシュと分岐予測シミュレーションを，PC2 と PC3 がそれぞれ分割箇所までおこない，それに引続いて詳細シミュレーションをおこなう．

この並列シミュレーションにおいて各 PC 間の時間関係を考える．論理シミュレーションは，詳細シミュレーションにくらべて実行時間が短いため，図中の a, b, c 点で示される各区間の詳細シミュレーション開始時刻を近づけることが可能である．したがって，並列シミュレーションの効果が期待できる．

3.2 論理/詳細シミュレーションの違い

前項で，分割開始箇所にキャッシュと分岐予測の情報を与えるため論理シミュレーションを用いると述べた．しかし，論理シミュレーションではパイプラインをシミュレーションしないため，シミュレーション結果に違いが生じる．以降は，キャッシュについてその違いを述べてゆく．また，説明するにあたって論理シミュレーションによるキャッシュを**論理キャッシュ**，詳細シミュレーションによるキャッシュを**詳細キャッシュ**と呼ぶことにする．

3.2.1 キャッシュ参照履歴

詳細キャッシュには，シミュレーションの過程で発生した分岐予測ミスに関係するキャッシュアクセスが反映されるが，論理キャッシュには反映されない．このことが，論理キャッシュと詳細キャッシュの違いとなる．キャッシュの違いが解消されるためには，相違のあるブロックが置換されなければならない．パイプライン状態の一致については，分岐予測ミスの度にパイプラインが空もしくは空に近い状態になるため，各シミュレーションノードの分割点の後の一定区間の重複による状態一致を見込めるが，キャッシュにおいては重複によって違いが解消されない場合も考えられる．しかし，本論文で提案する並列シミュレーションにおいて問題となるのは分割区間内におけるキャッシュの違いである．言い替えれば，論理キャッシュと詳細キャッシュがシミュレーションの分割点で一致していなくても，分割区間内においての参照履歴が一致していれば，詳細シミュレーションは正しく実行される．

キャッシュに相違があっても，分割シミュレーション区間においてキャッシュ参照履歴が同じになる場合は，

- 相違のある箇所を参照しなかった．
- 相違のある箇所を参照したが，参照前に置換されていた．

の 2 通りが考えられる．

3.2.2 キャッシュ参照履歴比較

キャッシュ参照履歴を比較するためには，分割区間シミュレーション中にキャッシュ参照履歴を記録しておく必要があり，すべての履歴を保存するには膨大なコストを要する．

しかし，キャッシュ容量が有限であることを考慮すれば，記録するべき履歴は定数回で抑えられることが分かる．なぜなら，キャッシュ参照履歴の比較で知りたいのは，区間シミュレーションごとにキャッシュのアクセス結果が一致するか否かであり，そのためには各セットについての各々ウェイの最初のアクセス結果だけを記録しておけばよいからである．すなわち各ウェイが一度ずつアクセスされてしまえば，セットの状態は過去に依存しないものとなるため，それ以後のアクセス結果が一致することは明らかである．例えば，512 セット，4 連想のキャッシュであるならば，キャッシュ参照履歴を比較するためには最大 512 × 4 回の異なるキャッシュアクセスを記録するだけでよい．

4. 並列シミュレーションの設計

並列シミュレーションは，分割区間の並列実行フェーズとそれら分割シミュレーションの結果を比較・統合する検証フェーズによって構成される．

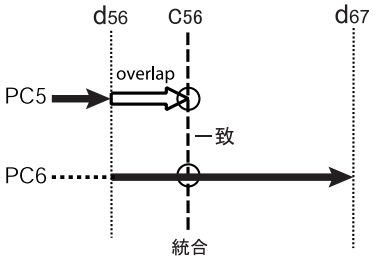

図 3 マシン状態一致によるシミュレーション統合

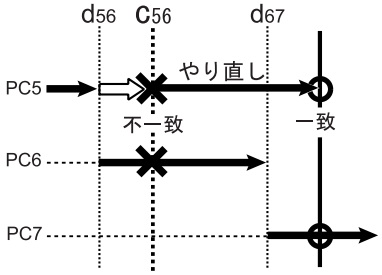

図 4 マシン状態不一致によるシミュレーションやり直し

4.1 分割シミュレーション区間の統合

分割区間の並列実行フェーズでは論理シミュレーションに引続き各 PC が分割区間と，状態一致のための分割点後の一定重複区間をシミュレーションする．その後，検証フェーズにおいてマシン状態を比較する．

並列シミュレーションの説明をするにあたって**図 3** を用いる．図は例として分割点 d_{56} における PC5 と PC6 の並列シミュレーションを表している．PC5 は分割点である d_{56} まで，PC6 は d_{56} から論理キャッシュ・分岐予測シミュレーションに引き続いて詳細シミュレーションを行っている (黒矢印)．

この図で PC5 と PC6 のシミュレーションの統合箇所となるのは，分割点 d_{56} から重複をおこなった c_{56} である．PC5 のシミュレーションが c_{56} に到達する時刻には、比較相手である PC6 の c_{56} 箇所シミュレーションはすでに終わっているので、あらかじめ PC6 の重複後の状態を保存しておくことによって状態を比較する．また，キャッシュ参照履歴を比較するために，各セットについて連想度に等しいだけの異なるアドレスに対するアクセス結果を記録しておく．マシン状態が一致もしくは，参照履歴により分割区間内の詳細シミュレーションが正しく実行されていることが確認できれば分割シミュレーション結果を統合する．キャッシュ参照履歴比較方法については，次項で述べる．

シミュレーションの重複によっても，マシン状態，キャッシュ参照履歴が不一致であった場合は，PC6 の区間の結果は使えない．この場合は，d_{56} の次の分割点の重複後での統合を目的とし PC5 が PC6 の担当区間を引続きシミュレーションする．この際， PC5 が PC6 に代わってシミュレーションをやり直す時間が状態不一致によるペナルティとなる．(**図 4**)

4.1.1 キャッシュ参照履歴比較方法

キャッシュ参照履歴は各分割区間ごとにひとつしか存在しない．(**図 5** を例に用いて説明すれば，A 区間の履歴はあるが A ’区間の履歴は存在しないということである)．よって，並列シミュレーションにおいてのキャッシュ参照履歴比較は，統合対象となる分割シミュレーション区間 A のキャッシュ参照履歴をαの詳細キャッシュに照し合せ，キャッシュを更新しながらそのヒット/ミスが一致するかどうかで検証する．例に上げた区間 A では，あるセットの履歴が e:ヒット，f:ミス，g:ミス，h:ヒットとなっているので，それをαの詳細キャッシュに照し合せて同様のヒット/ミスとなるかを検証する．この検証方法では，詳細キャッシュを用いて比較をおこなっている．

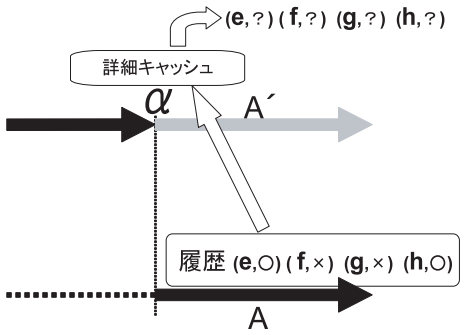

図 5 キャッシュ参照履歴の比較

しかし，分割区間シミュレーションでは，論理シミュレーションに引続き詳細シミュレーションをおこなうため，分割区間が終わった際のキャッシュが正しい詳細キャッシュなっているとは限らない．その解決方法について次項で説明する．

4.1.2 キャッシュの合成

説明のために，分割点によって区切られる区間を通常シミュレーションにおいての開始点から，分割区間 1，分割区間 2，分割区間 3... と呼ぶ．前項で述べたキャッシュ参照履歴比較方法では，前分割区間による詳細キャッシュの存在が前提となっている．図 5 が，分割区間 4 と 5 の分割点を表しているとすれば，図中のαに詳細キャッシュが存在するとは限らない．区間 4 においてキャッシュが正しくアクセスされたとしても，キャッシュの全領域がアクセスされるとは限らないからである．アクセスされずに残っている箇所の状態は，分割シミュレーション開始時の論理キャッシュ

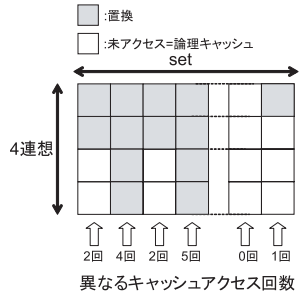

図 6 論理キャッシュの置換

の状態のままである (**図 6**)．しかし，分割区間 1 だけは詳細キャッシュが存在する．分割区間 1 は最初から詳細シミュレーションをおこなっているからである．よって分割区間 1 の詳細キャッシュを用いて分割区間 2 のキャッシュの未アクセス部分 (アクセス履歴がない部分) を補ってやれば分割区間 2 における詳細キャッシュが分かる．同様にして分割区間 2 の詳細キャッシュを用いて分割区間 3 のキャッシュの未アクセス部分を補ってやれば分割区間 3 の詳細キャッシュが分かる．こういった具合にして，キャッシュ参照履歴が正しければ次の区間の詳細キャッシュが順々に分かる．よって，比較検証フェーズを分割区間 1 から順におこなえば，キャッシュ参照履歴の比較が可能である．(**図 7**)

4.1.3 キャッシュ参照履歴比較・合成

前項，全前項で説明したキャッシュ参照履歴の比較方法と，キャッシュの合成をまとめると，以下の通りである．

- 分割区間 1 と分割区間 2 から順に，キャッシュ参照履歴を詳細キャッシュに照らし合わせキャッシュアクセスのヒット/ミスが，一致するか調べる．
- キャッシュアクセスのヒット/ミスが詳細キャッシュと一致していれば，アクセスされた箇所の論理キャッシュは詳細キャッシュと同等であると言える．分割シミュレーション区間において，全領域に対するアクセスがなかった場合，アクセスされずに残った部分は詳細キャッシュと同じかどうかは不明である．そこでこの場合は前分割区間の詳細キャッシュを用いて補うことによってキャッシュの合成をおこない現分割区間の詳細キャッシュ作る．この詳細キャッシュは次のキャッシュ履歴参照比較の際，リファレンスとして用いる．

5. 実現可能性の評価

並列シミュレーションの実現可能性について，パイプラインとキャッシュについてマシン状態を調査した．SimpleScalar Tool Set Version 3.0 を用い，評

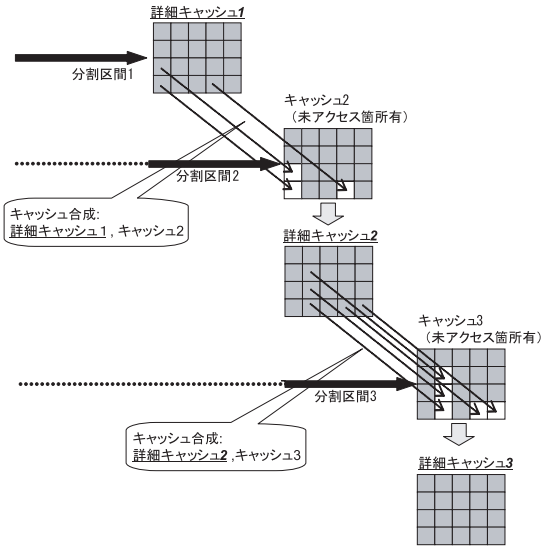

図 7 キャッシュの合成

価モデルの演算ユニット数は INT-ALU が 4, INT-MUL/DIV が 1, FP-ALU が 4, FP-MUL/DIV が 1 とし，フェッチ幅と発行幅は 4 とした．キャッシュは il1 を 512 セット 1 連想，dl1 を 128 セット 4 連想，ul2 を 1024 セット 4 連想と設定した．評価プログラムには SPEC CPU95 を用いた．

5.1 パイプライン状態

パイプラインについての予備評価として，シミュレーションの重複をおこなった場合のパイプライン状態の一致率を調査した．時間軸分割シミュレーションの開始状態を再現するために詳細シミュレーションのパイプラインを空にした．このようにして通常のシミュレーションと比較し重複後のパイプライン状態の一致率を調べた．また今回はパイプライン状態の収束が最も早いと考えられる分岐予測ミス点を分割点とした．その結果，パイプラインの不一致確率は重複区間が 1000 命令のとき，0.01%以下であることが分かった．

5.2 キャッシュ状態

任意の分割候補点約 1000 箇所について重複区間を 1000 命令としたときのキャッシュの一致状況を調べた．キャッシュの一致率を**表 1** に示す．結果から論理キャッシュは，パイプラインのようにわずかな重複では，詳細キャッシュとは一致しないことが分かる．とくに SPECint95 では論理キャッシュが詳細キャッシュと異なる傾向が強い．よって重複区間を延長することで詳細キャッシュと論理キャッシュの相違の解決を試み，また相違が完全に解決されなくても，キャッシュ参照履歴をチェックすることにより分割シミュレーションの統合を図る．

表 1 1000 命令重複のキャッシュの一致率 (%)

SPEC95	il1	dl1	ul2
101.tomcatv	0.0	75.2	66.5
102.swim	0.0	99.5	79.6
103.su2cor	57.2	91.2	9.9
104.hydro2d	60.0	96.6	88.3
107.mgrid	49.1	97.5	70.9
110.applu	96.3	81.3	74.4
125.turb3d	0.0	98.1	96.1
141.apsi	58.7	77.2	0.0
145.fpppp	83.6	82.4	0.0
146.wave5	0.0	51.1	33.8
099.go	0.5	0.7	0.0
124.m88ksim	0.0	93.2	85.4
126.gcc	0.0	4.5	0.0
129.compress	0.0	26.7	1.3
130.li	0.0	36.6	0.0
132.ijpeg	0.0	39.0	0.0
134.perl	0.0	13.5	2.4
147.vortex	3.8	49.2	0.4

表 2 16 分割 10%重複におけるキャッシュ状態 (%)

SPEC95	il1	dl1	u2	履歴
101.tomcatv	0.0	98.8	97.6	97.8
102.swim	0.6	100.0	96.6	87.1
103.su2cor	91.1	100.0	97.1	91.3
104.hydro2d	86.9	100.0	98.0	88.1
107.mgrid	100.0	100.0	100	100.0
110.applu	70.0	100.0	81.4	83.8
125.turb3d	12.5	100.0	99.9	74.1
141.apsi	87.6	100.0	0.0	47.3
145.fpppp	100.0	100.0	0.0	95.6
146.wave5	3.9	100.0	99.1	70.9
099.go	99.2	83.6	3.8	6.8
124.m88ksim	0.8	99.0	87.3	90.1
126.gcc	93.8	100.0	54.2	77.4
129.compress	0.0	81.9	15.7	10.5
130.li	0.0	100.0	0.0	97.0
132.ijpeg	100.0	100.0	86.5	90.3
134.perl	0.0	100.0	29.7	64.5
147.vortex	99.5	100.0	73.2	86.0

5.3 キャッシュ参照履歴一致率

並列シミュレーションの分割数を 16，重複を分割区間の 10%，(つまり全命令の 0.6%が重複区間にあたる) と想定しキャッシュ，キャッシュ参照履歴について調査した．キャッシュの状態については，任意の分割点約 1000 箇所．キャッシュ参照履歴については，約 90 箇所について調査した．キャッシュ参照履歴に関しては，キャッシュ状態が一致していた場合も含まれる．結果を**表 2** に示す．

SPECfp95 107.mgrid については，想定した重複後，命令 1 次キャッシュ，データ 1 次キャッシュ，2 次キャッシュが一致し，並列シミュレーションの効果が期待できることが分かった．またキャッシュ参照履歴は SPECfp95 において平均 82%，SPECint95 で平均 65%一致することが分かった．16 ノードでの並列シミュレーションの加速率は「16 ×一致率」と見積もることができるので，キャッシュ参照履歴を用いればキャッシュが不一致であっても効率的な並列シミュレーションをおこなうことができる．分岐予測器については，分割点において分岐予測方向がほぼ一致することを確認しているが，分岐予測に関しても履歴よる状態検証を適用することで，より効率的な並列シミュレーションが期待できる．

6. 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にシミュレーションをおこなうことで高速化を図る手法を述べた．手法として，以下を用いた．

- 論理動作によるキャッシュシミュレーションと分岐予測シミュレーションを分割点までおこない，それに引き続き詳細シミュレーションをおこなう．分割候補点は分岐予測ミス箇所とする．
- 各シミュレーションノードを重複させることにより分割シミュレーションにおけるマシン状態を近づける．
- キャッシュの状態比較において，キャッシュの参照履歴を用い，分割区間内でのシミュレーションが正しくおこなわれたかチェックする．

予備評価により、その実現性を示した．今後は本方式に基づく並列シミュレータを実装し，本方式の速度評価をおこなう予定である．

謝辞 本研究の一部は (株) 半導体理工学研究センター (STARC) との共同研究「SpecC によるソフトウェア記述検証システム」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) 中田 尚, 中島 浩: 高速マイクロプロセッサシミュレータ BurstScalar の設計と実装. 情報処理学会論文誌:コンピューティングシステム,Vol 45, No.ACS6,2004
3) 高崎 透, 中田 尚, 中島 浩: 高性能マイクロプロセッサシミュレータの並列化による高速化の構想. 情報処理学会研究報告, 2003-ARC-155, November 2003.