2003－ARC－155　(6)
2003／11／28

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

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の構造は高度化，複雑化している．また組込み機器の設計においても，その複雑さは急激に増大し，システムの大部分が高度なマイクロプロセッサとその上で動作する組み込みソフトウェアによって構成されるようになっている．このような高度なプロセッサの研究・開発や，それを組み込んだ機器のハードウェア・ソフトウェア協調設計においては，その機能や性能を検証するための 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を時間軸上において分割し，並列にシミュレーションをおこない，それらを統合する手法で高速化をはかる．この分割シミュレーションで問題となるのは，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，分岐予測ミス時にパイプラインが空あるいはそれに近い状態となることを利用し，分岐予測ミス点を分割点とする．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，区間終了時の状態をチェックすることにより，高い確率で正しいシミュレーションを行なうことができる．SPEC95 を用いた予備評価の結果，重複区間を 1000 命令とすれば不一致確率が 1/30000 以下となることが確かめられた．

High Speed Simulation of High Performance Processor by Parallel processing

Toru Takasaki,[†] Takashi Nakada[†] and Hiroshi Nakashima[†]

Microprocessor simulation is indispensable not only for hardware systems design but also for software development of co-designed embedded systems. In both design fields, cycle accurate (or clock level) simulation of highly sophisticated microprocessor is required. However, existing simulators of out-of-order processors run programs thousands or times slower than actual hardware. The ultimate goal of our research is to develop a fast and accurate simulator which is capable of microarchitectural modeling and system level simulation.

Parallel simulation is to make target machine state consistent at the divided point. Our solution is to divide the simulation at branch mispredictions at which the pipeline of the target machine is completely flushed or nearly empty. Thus after a short period, it is expected the pipeline state of the simulation starting from the divided point becomes consistent with its predecessor that overlappingly simulates the period. Our experience with SPEC CPU 95 proved the correctness of our proposal showing very small probability of pipeline state inconsistency, less than one out of 30,000, if the overlap period is 1,000 instructions.

1. はじめに

集積回路技術の進歩とともに，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:slowdown) 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 ~ 数十倍の時間をようするため，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である．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

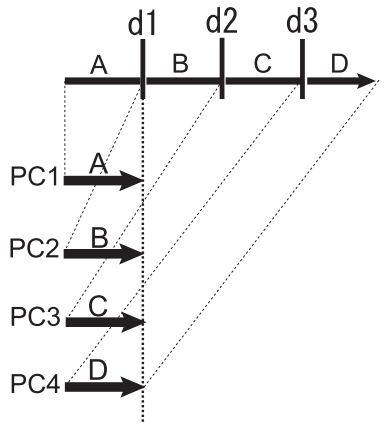

図 1 時間軸によるシミュレーションの分割

行時間の多くを占めているかが分かる．

そこで本研究では，シミュレーションを時間軸上で分割し，各々の分割区間を並列にシミュレーションすることにより，大幅な高速化が達成可能な方式を考案した．ここで問題となる分割点におけるマシン状態の整合性は，命令レベルシミュレーションを完全に重複しておこなうことと，パイプラインシミュレーションを一定区間だけ重複しておこなうことにより解決する．また重複区間終了時の状態が整合する確率を高めるため，分岐予測ミスを分割点にする．以下本報告では，2 章で時間軸分割について説明し，3 章で高速化の原理，4 章で並列シミュレーション方法について述べる．

2. 時間軸分割

高速化のためにシミュレーションを時間軸上において多数に分割し，それら分割された部分を並列に実行し結果を統合する．時間軸分割のイメージを**図 1** に示す．

図中の最も上の矢印が，通常のシミュレーションを表し，左から右にむかってシミュレーションが進んでいる．時間軸上の分割とは，図中の矢印上にある分割点 d1, d2, d3 でシミュレーション過程を分割することである．

シミュレーションは，分割点 d1, d2, d3 により，矢印上に示す A, B, C ,D のシミュレーション区間に分割される．分割された区間 A, B, C, D は PC クラスタの PC1, PC2, PC3, PC4 によって，並列にシミュレーションされる．

時間軸上でシミュレーションを分割すると，分割箇所前後でのシミュレーション過程のつながりを切り離してしまうことになる．分割したシミュレーションを統合して全体のシミュレーション結果とするには，分割箇所の前後でアーキテクチャ状態が一致していることが必要である．そこで，分割して並列実行した各シミュレーション結果において，アーキテクチャ状態の一致を導く方法を次章で説明する．

3. 高速化の原理

3.1 分割・統合による高速化

シミュレーションを時間軸で分割して，それらを同時に並列実行すればシミュレーション時間を格段に短くできる．例えば，n 区間に分割して同時に実行すれば n 分の 1 の時間で全体のシミュレーション区間が終了する．残りの作業は分割した各シミュレーション結果の統合である．

並列実行されたシミュレーション結果を統合するためには，各分割箇所の前後でそれぞれのアーキテクチャ状態が同じになっている必要がある．

そのためには，分割箇所から開始されるシミュレーションのアーキテクチャ状態と，通常の分割をおこなわないシミュレーションのアーキテクチャ状態とが同じになるような方法で分割シミュレーションをおこなわなければならない．シミュレーションは，メモリ・レジスタ，分岐予測，キャッシュ，パイプラインシミュレーションに分けて考えられる．

- 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．
- 分岐予測シミュレーション
  分岐予測は過去のすべての実行に依存している．しかし，分岐予測のシミュレーション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．
- 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．
- 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．

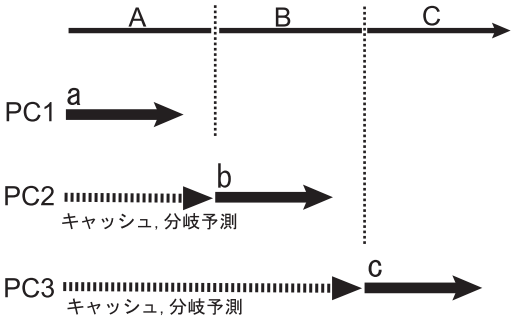

図 2 キャッシュ・分岐予測シミュレーション

3.2 キャッシュ・分岐予測シミュレーション

それぞれのアーキテクチャ状態が同じになるように分割箇所までキャッシュと分岐予測シミュレーションをおこなう．これは，時間軸上において途中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シミュレーションの情報を与えるためである．1 章で述べたように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．よってキャッシュと分岐予測については論理動作によるシミュレーションをおこなう．キャッシュと分岐予測シミュレーションは，実行時間が短いので，それぞれ分割されたシミュレーションは，ほぼ同時に開始することができる．**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を示している．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 の区間に分割されている．分割された区間の B, C は時間軸上において途中から開始するシミュレーションである．分割箇所でのアーキテクチャ状態の一致のはかるために PC2 と PC3 はそれぞれ，図の点線矢印で表されるように，キャッシュと分岐予測シミュレーションを分割箇所までおこなう．シミュレーション区間 B, C を担当する PC2，PC3 はキャッシュ・分岐予測シミュレーションに引続いて詳細なシミュレーションを開始する．論理動作によるキャッシュシミュレーション，分岐予測シミュレーションの実行時間は，詳細なシミュレーションの実行時間に比べて小さいので，図中の a, b, c 点で示される各区間の詳細なシミュレーション開始時刻は，ほぼ同じであり，同時に並列実行することができる．

3.3 パイプラインシミュレーション

分割したシミュレーションを統合するために，シミュレーション過程におけるパイプラインの状態を知る必要がある．しかし，パイプラインの状態は命令スケジューラによる詳細なシミュレーションをおこなわなければ知ることができない．

パイプラインにはその効果を最大限に引き出すため，次々に命令が送り込まれている．しかし，条件分岐命令があると，条件が成立するかしないかという結果を待たなければ次の実行を行うことができず，パイプラインが停止してしまうことがある．そこでパイプラインでは分岐予測機構をもちいて，条件分岐の結果を待たずに，命令フェッチと実行を続けておこなっている．パイプラインでは分岐予測命令も，他の命令と同様に処理をおこなうため，分岐予測に続く命令列もすでにパイプライン中に投入され実行が開始されている．したがって，予測がヒットした場合は，それらの処理を続けることができるが，分岐の予測がはずれた場合は後に続くパイプラインの内容をフラッシュして，命令フェッチからやり直したり，予測を誤って実行した命令に依存する内容をパイプラインステージから消去したりして対処をおこなっている．

そのため分岐予測がはずれた場合のパイプラインの状態は，シミュレーション過程のどの点においても，似通った状態にあると考えられる．分岐予測が外れた場合にパイプライン状態をフラッシュするタイプのプロセッサであれば，シミュレーション過程において分岐予測ミスが発生するたびに何度もパイプライン状態が空になっているはずである．またシミュレーション開始時においてはパイプラインが空であるのは明らかである．よって分岐予測ミスが発生した箇所を時間軸分割の場所とすれば，パイプラインの状態は同じである．

また分岐予測ミス発生時に，誤った予測により実行した後続の命令のみを消去するタイプのプロセッサにおいても，各シミュレーションをわずかに重複して実行することでパイプライン状態が一致する場合があると考えられる．

3.4 パイプライン状態の一致

プロセッサが分岐予測ミス発生時に，パイプライン上から誤って実行した後続の命令のみを消去する場合においても，パイプライン状態は疎であると考えられ，パイプラインがフラッシュされた時の空の状態に近い．したがって，分岐予測ミス発生時点でパイプライン状態が同じでなくてもお互いにシミュレーションが少し進んだ所では一致している可能性が高い．

そこで，シミュレーションをわずかにオーバーラップさせることによってパイプライン状態の一致をはかる．**図 3** を用いて説明する．

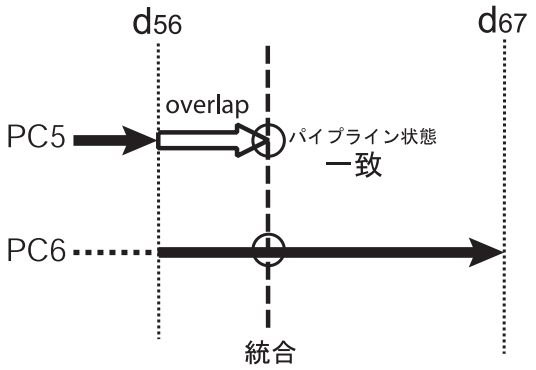

図 3 オーバーラップによるパイプライン状態の一致

図 3 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 は分岐予測ミス点であり分割点である d_{56} まで，PC6 は d_{56} からキャッシュと分岐予測シミュレーションに引き続いて詳細なシミュレーションをしている (黒矢印)．

PC5 が分割点 d_{56} までシミュレーションをおこなってきた時，PC6 は同じように次の分割点 d_{67} に到達しようとしている．そのため，PC5 が分割点からオーバーラップをかける PC6 のシミュレーション部分はすでに実行が終わっている．そこで，PC6 はオーバーラップ区間終了時のパイプライン状態をあらかじめ保存しておき，PC5 はオーバーラップ区間を含むすべてのシミュレーションが終了した時点で，PC6 が保存した状態を自分の状態を比較する．比較した結果，パイプラインの状態が同じになっていればその場所で統合できる．

3.5 予 備 評 価

予備評価として，シミュレーションのオーバーラップをおこなった場合にパイプライン状態が一致するかどうかを実験した．

実験では，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なシミュレーションのパイプラインを空にした．詳細実行の過程で分岐予測ミスが発生したら，フェッチをストップさせることで，パイプラインを空にする方法を用いた．そのようにして，再現した分割シミュレーションと通常のシミュレーションのパイプライン状態が同じになるかを調べた．

並列シミュレーションにおいてのパイプラインの状態の比較箇所は，分岐予測ミス発生から 1000 命令後にした．

実験には，PentiumIII(866MHz, 512MB)，SimpleScalar Tool Set Version 3.0 を用いた．

表 1 1000 命令後のパイプライン状態の不一致数

SPEC95	不一致数/採取数	全ミス数
101.tomcatv	0/649840	8957513
102.swim	0/273309	808877
103.su2cor	24/809285	39978745
104.hydro2d	0/333865	2958101
107.mgrid	0/791712	10139012
110.applu	0/355927	4357922
125.turb3d	0/807332	85932879
141.apsi	1/736708	6045396
145.fpppp	0/69107	348235
146.wave5	0/853435	2531592
099.go	1/413542	16040589
124.m88ksim	0/45452	334249
126.gcc	0/410479	29681944
129.compress	0/31683	632492
130.li	0/171681	3148673
132.ijpeg	0/372779	9857402
134.perl	0/347855	29559906
147.vortex	0/458287	6861667

評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．評価プログラムには SPEC95 を用いた．

結果を**表 1** に示す．表には左からプログラム名，パイプラインの状態の不一致数/分岐予測ミス採取数，各プログラムにおける全分岐予測ミス数を示す．

全てのプログラムにおいて，分岐予測ミスから 1000 命令後のパイプライン状態がほぼ一致していることわかった．よって，分岐予測ミス点を分割点とした並列シミュレーションが可能であると言える．

また，プログラム全体のシミュレーション時間からみれば 1000 命令分のシミュレーション時間は，極めて小さく無視できる．

3.5.1 パイプライン状態の不一致

実験結果により，分岐予測ミス箇所から 1000 命令後にはパイプライン状態がほぼ一致していることがわかったが，そうならない場合も考えられる．

シミュレーションのオーバーラップによっても，パイプライン状態が一致しなった場合は，オーバーラップの対象になっているシミュレーション区間の結果は使えない．

そこで，オーバーラップをおこなった PC が引き続いてシミュレーションをする．引き続きシミュレーションをおこなうのは，次の分割点での統合をねらうからである．パイプライン状態が一致しなかった場合は，一致しなかった分割点の次の分割点において，ふたたび，パイプライン状態が同じになっているかを比較する．

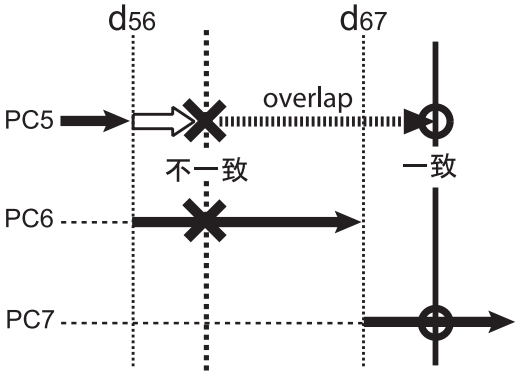

図 4 パイプライン状態の不一致

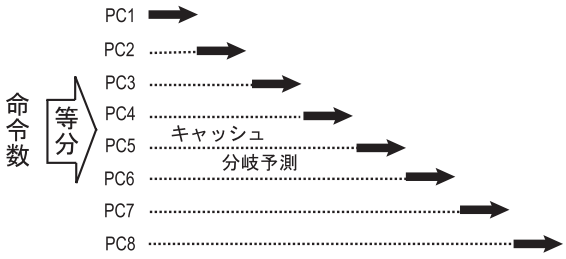

図 5 並列シミュレーション

こうすると，結果が使えない分割区間を再度シミュレーションすることになるので，時間が余分にかかってしまうが，全体のシミュレーション結果を得ることができる．また，パイプライン状態が不一致になる確率は表 1 から極めて小さいと考えられる．

パイプライン状態が不一致の場合の対処法を**図 4** を用いて説明する．

PC5，PC6，PC7 はそれぞれ時間軸上の分割点 d_{56}, d_{67} で区切られた区間を並列にシミュレーションをおこなっている．シミュレーション統合のために PC5 が PC6 に対してオーバラップ (白矢印) をおこなっても，パイプラインの状態が一致しなかった場合は，PC5 が引続き，PC6 と PC7 の分割点 d_{67} までシミュレーションをおこなう．そして，再度 PC7 に対してオーバーラップをおこないパイプライン状態の一致をはかる（点線矢印)．

シミュレーション結果が使えない PC6 のシミュレーション区間を，PC5 が代わって再度シミュレーションをおこなう時間が不一致によるペナルティとなる．

4. 並列シミュレーション

n 台の PC を用いて並列シミュレーションをおこなう場合，実行される命令数をほぼ等分とすることにより負荷の均衡をはかる．したがって分割点は 1/n 個の命令数を実行した直後に生じる分岐予測ミスになる．

例えば，1600 億命令あるプログラムについて，16 台で並列にシミュレーションをおこなわせたい場合，1 台が担当する命令数を 100 億命令とし 100 億命令分実行したら，その直後に発生する分岐予測ミス命令を分割点にする．そのためにまず命令エミュレーションをおこない，プログラムの全命令数を知る．その結果により分割点を決定し並列シミュレーションをはじめる．また分割数を決めるための命令エミュレーションは論理動作によるので，実行時間は短い．

図 5 で手順を説明する．図は 8 分割による並列シミュレーションを表している．はじめに，命令エミュレーションをおこないプログラムの全命令数を知る．並列シミュレーションをおこないたいユーザーは，何台の PC を用いるかを設定する．何並列でシミュレーションにおこなうかにより，1 台あたりが実行する命令数が決定したら，各 PC は一斉にシミュレーションを開始する．PC1 は割り当てられた命令数直後の分岐予測ミス命令まで，PC2〜PC8 は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で命令数をカウントしながら，自分が割り当てられたシミュレーション区間の開始箇所まで進み，詳細なシミュレーションをはじめる．たとえば，PC6 は自分の担当するシミュレーション区間までの 500 億命令分をキャッシュと分岐予測シミュレーションをおこない進み，それに引き続いて詳細なシミュレーションを開始する．

また，PC クラスタによる並列化台数とシミュレーションの分割数が同じ場合は，パイプライン状態が不一致になった場合，台数分の 1 の区間，たとえば 4 並列の場合では，4 分の 1 の区間が使うことができないシミュレーション結果となる．そうなると，パイプラインが不一致のときのペナルティが大きい．

そこで，**図 6** に示すように各 PC が一度に詳細なシミュレーションをおこなう区間を小さくすることも考えている．図 6 はシミュレーションを時間軸で 8 分割し，4 台の PC で並列シミュレーションをおこなっている． PC ごとに示してある矢印について，黒矢印は時間軸で分割した詳細なシミュレーションを，点線矢印はキャッシュと分岐予測シミュレーションを表している．各 PC は担当の分割箇所まで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おこない詳

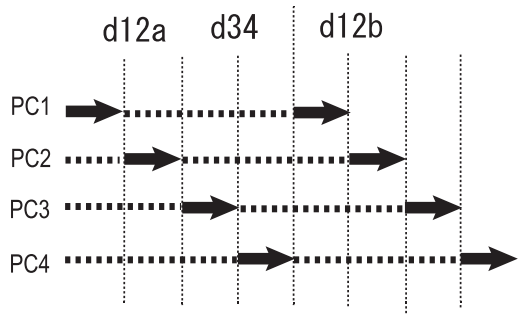

図 6 細分割並列シミュレーション

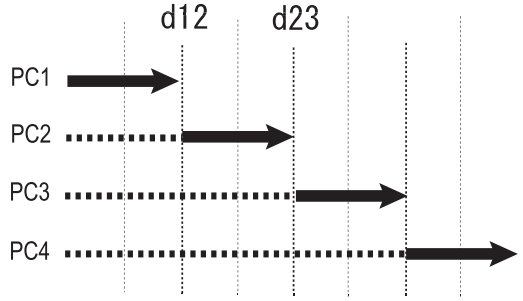

図 7 4 分割並列シミュレーション

細な実行を開始するが，割り当てられた命令数分の詳細なシミュレーションを終えたら，ふたたび，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に戻り，次の分割区間まで進む．こうすることで，パイプライン状態の不一致が発生した場合に，前の区間を担当している PC が引き続きシミュレーションをおこなう区間が小さくなるので，ペナルティとしての時間の遅れを小さくできる．シミュレーションの時間軸による分割数を増やせば，もちろんパイプラインが不一致になる確率は高くなるが，パイプライン状態の不一致箇所によっては，不一致による再実行が並列におこなわれて不一致箇所の増加によるコストを隠蔽することができる．

例えば**図 7** は，4 分割で並列にシミュレーションをおこなっている様子である．PC1 は分割点 d_{12} においてパイプライン状態が一致しなかった場合に d_{23} まで引き続きシミュレーションをおこなう．よって並列シミュレーションに要する時間は，各 PC のうちで最も長くかかったシミュレーションである PC1 に要する時間である．図中の黒矢印で表現すると矢印 2 つ分ということになる．つまり 1 台でのシミュレーションの約 1/2 の時間を要することになる．

一方，図 6 において 8 分割したシミュレーションが，4 分割したシミュレーションにくらべて，パイプライン状態が不一致になる確率が 2 倍になったとして考えてみる．分割点 d_{12a} と d_{34} でパイプラインの不一致が発生した場合を考えると，PC1 と PC3 は，ほぼ同時に不一致解消のための再実行をおこなうので，その分，時間が短縮される．全体のシミュレーションにかかる時間を図中の黒矢印で表現すると矢印 3 つ分になる．これは図 7 に照らしあわせてみれば矢印 1.5 個分に相当する．よってパイプライン状態の不一致が発生したときのペナルティは小さくなっている．図 6 の分割点 d_{12a} と d_{12b} でパイプライン状態の不一致が発生した場合は，不一致解消のための再実行を同時におこなえないので，時間短縮はできない．シミュレーション時間は矢印 4 つ分になり，すなわち，図 7 の矢印 2 つ分と同じである．

5. おわりに

本論文では、シミュレーションを時間軸で分割し並列にシミュレーションをおこない高速化をはかる手法について述べた．手法としては，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なシミュレーション開始することと，各シミュレーションをわずかにオーバーラップさせることによって，パイプラインを一致させる方法を用いた．また，予備評価をおこなうことにより、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・評価し，本方式の有効性を実証する予定である．

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) Burger, D.and Austin, T.M.: The SimpleScalar Tool Set, Version 2.0 (1997)
2) 中田 尚, 大野 和彦, 中島 浩: 高性能マイクロプロセッサの高速シミュレーション (2003), 先進的計算基盤システムシンポジウム SACSIS'03
3) 富田 眞治, 中島 浩: コンピュータハードウェア 昭晃堂 (1995)