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

2004－ARC－159 (17)
2004／7／31

割込みによるマイクロプロセッサの性能劣化の予測方式

小西昌裕† 中田尚† 中島浩†

本論文では，割り込みによるプリエンプションに起因する性能悪化の最大値を高速に取得する方法を論ずる．キャッシュメモリや命令パイプライン機構を搭載するマイクロプロセッサでは，実効的な性能悪化量を得るためには割り込みの実行をシミュレートする必要があるが，割り込みが発生しうるすべての箇所においてこのようなことを行えばその実行時間は膨大なものになる．そこで，プロセッサ状態の比較を行うことで重複する実行を省略し，実行時間の大幅な短縮を図った．プラットフォームとしてSimpleScalarを用いて，まず命令パイプラインについて状態の比較による実行の省略を行った．その結果，連続する10万サイクルの命令実行に対し，各サイクルにおける割込みによる性能悪化量を1回あたり60ミリ秒で解析することができた．

Measuring Method of the Performance Deterioration Affected by Interruption

Masahiro Konishi,† Takashi Nakada† and Hiroshi Nakashima†

In this paper, we describe a method to estimate the worst-case performance of a program affected by preemption. For modern microprocessors with complicated mechanisms, such as pipeline or cache memory, it is difficult to measure worst-case performance quickly and exactly. We devised a fast measurement mechanism by recording and comparing pipeline states with and without an interruption, rather than executing the whole program for each interruption point.

We are developing a simulator for the worst-case performance estimation. As the first step, we modified the SimpleScalar microprocessor simulator to handle interruptions and to measure the performance with an interruption varying its occurrence point. We evaluated the implementation using a 9-Queen solver program to find our simulator measures performance effect of a interruption in 60 msec.

1. はじめに

自動車のエンジン制御などといった組込みシステムにおいては，割り込みにより生じるプリエンプションがもたらす性能悪化が，システムの性能見積りの際に重要な意味を持つことになる．ある処理時間中に終了しなければならない処理があるとすれば，それはどのようなタイミングで割り込みが発生しようともこれを越えてはならないということである．しかし一方で，昨今のマイクロプロセッサによく見られる命令のパイプライン機構やキャッシュメモリ，分岐予測器などが存在すれば，割り込みのタイミングによってその性能悪化の度合が大きく異なることになる．

従来の最悪性能予測技法としてクリティカルパスなどを用いるものが存在する．しかしこれらの方法では，予測結果に誤差がつきものであり，特に前述のような複雑なマイクロプロセッサであれば誤差は大きくなることが予想される．ある地点で割り込みが発生した場合に全体の性能がどれだけ低下するかを調べるもっとも確実な方法は，実際にその地点で割り込みを発生させることであるが，これを割り込みが発生しうるすべての地点で実行すれば，プログラムの実行時間に地点の数を掛けただけの実行時間が必要となり，現実的ではない．本論文では，プロセッサ状態の比較を行うことでこの実行時間を大幅に削減する方法について論ずる．2章では，割り込みによるプリエンプションの影響と状態一致による実行省略の基本的な考え方を説明する．3章ではそのような実行省略を行うための具体的設計を述べる．4章では，特にパイプラインのみに対象を絞って実行省略を実装し，5章でその結果を報告する．6章では今後の課題について，7章では本論文の総括を，それぞれ述べる．

2. 最悪性能予測

2.1 プリエンプションと性能悪化

一般的に，命令パイプラインを備えたマイクロプロセッサでは，割り込みが発生した際にパイプラインに

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

投入されている未完了の命令をどうするかが問題となる．パイプラインをフラッシュすることで未完了の命令を一旦破棄してしまう，などの対処法があるが，いずれにしても性能には悪影響を与える．また，キャッシュメモリを備えている場合は，割り込みやそれに起因するプリエンプションによってキャッシュメモリに存在しているエントリの有効性が減少する．具体的には，割り込みハンドラやプリエンプトしたプロセスがキャッシュメモリのエントリを置き換えてしまい，割り込み元に処理が戻ってきたときには有効なエントリは減少している．

以上のように，プリエンプションはプログラムの性能に悪影響を及ぼすと言うことができる．

2.2 プリエンプションの影響

今回は，プログラムを実行中にプリエンプションが発生することで，以下のようなことが起こると想定する．

- 命令パイプライン中のコミットされていない命令がフラッシュされる．フラッシュされた命令は実行完了していないので，再びプログラムに実行が戻ってきた際にパイプラインの最初のステージから実行がやり直される．
- キャッシュメモリのエントリがすべて無効化される．実際にはいくつか有効なエントリが残る可能性もあるが，今回は最悪性能を求めるため，すべて無効化されるものとする．

なお，割り込みハンドラやプリエンプトしたプロセスでどのような処理が行われるかは考えず，割り込みの発生による影響は上で述べた2点のみとする．またTLBや分岐予測器への影響についてはキャッシュと同様に考えることができるので，本論文では議論を省略する。

2.3 最悪性能の検出方法

はじめに述べた通り，実際に各割り込み候補点で割り込みを起こすことで，割り込みによる性能悪化量を検出する．従って，割り込みによる最悪性能予測は以下のようにして行われることになる．

(1) 割り込み候補点に到達するまでプログラムを実行する．
(2) 割り込み候補点に到達したら割り込み発生をシミュレートする．
(3) プログラムを最後まで実行する．
(4) 実行終了までに要した総CPUサイクル数を出力する．

これをすべての割り込み候補点に対して実行すれば，割り込み発生による最悪性能量が算出できることになる．しかしこの方法では，プログラムを割り込み候補点の数だけ繰り返し実行することになるため，膨大な時間が必要となり非現実的である．そこで，次に述べるような状態比較による実行の省略を行うことで，総実行時間を削減する．

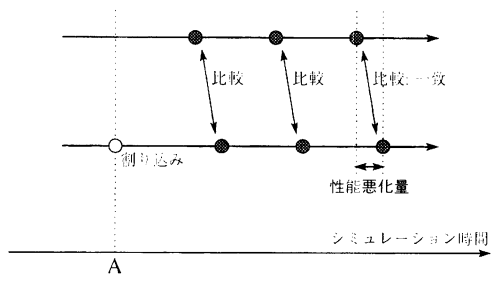


図1 状態一致による実行の省略

2.4 状態一致による実行の省略

プリエンプションが発生した直後のプロセッサ状態は，発生しなかった場合と異なっている可能性があるが，プログラムの実行を進めて行くにつれてこの差は小さくなり，やがては状態が一致すると考えられる．

プリエンプション発生後にプログラムの実行を継続し，ある地点においてプロセッサ状態がプリエンプション未発生の場合と等しくなったとすれば，これ以降もプロセッサ状態はすべて等しいままであると考えられるので，実行に必要なCPUサイクル数はプリエンプション未発生時と等しくなると考えられ，プリエンプションによる割り込みがもたらした性能悪化量は容易に算出することができる．つまり，ある地点Aでプリエンプションが発生した場合の性能悪化は，以下のようにして求められる．

(1) プリエンプションが発生しない場合のA地点以降のプロセッサ状態をすべて保存する．
(2) A地点でプリエンプションを発生させ，プロセッサ状態をプリエンプション未発生時のものと比較しながら実行を継続する．
(3) プロセッサ状態が一致したら，A地点から状態が一致した地点までに要した時間の差をとる．この差が性能悪化量である．

この様子を図1に示す．

プロセッサ状態の一致がプログラム全体の実行時間に比べてごく短い時間で起こるなら，この手法により性能悪化量を本来必要な時間に比べてごく短い時間で検出することができるということになる．

3. 設　　計

3.1 パイプラインの収束による一致

命令パイプラインはキャッシュメモリに比べ，プリエンプション発生後により早期に収束する[2]．これは，パイプラインが保持しうるデータ量がキャッシュメモリのそれと比べてごく小さいこと，分岐予測ミスが発生すると，投機的に実行されていたミスパスの命令がすべてフラッシュされ，パイプラインが空に近い状態になることが原因である．

そこで，まずプリエンプションによって影響を受けるものがパイプラインだけであるとして話を進める．

3.2 状態の比較回数削減

プリエンプションを発生させた後，パイプラインの状態を比較しながら実行を継続し，プリエンプション未発生時と状態が一致すれば実行は完了となるわけであるが，1CPU サイクルごとにパイプラインの比較を行うと比較にかかるコストが大きくなることが予想される．一方で，前述の通り，分岐予測ミスが発生するとパイプラインは空に近い状態になるのだから，分岐予測ミスが発生することでパイプラインの差異は非常に小さくなり，また，比較すべきデータが減るので比較も容易である．そこで，比較箇所を分岐予測ミス発生地点のみとすることで，比較回数の削減を図る．

3.3 プリエンプションの発生

プリエンプションの発生候補地点それぞれにおいて，以下のような処理を行う必要がある．

(1) プリエンプションの発生をシミュレートする．具体的には，命令パイプライン中の未完了命令を消去する．

(2) 保存しておいたパイプラインの状態と現在のパイプラインの状態を比較しながら実行を継続する．

(3) パイプラインの状態が一致したら，プリエンプションを発生させた地点から状態が一致した地点までの総 CPU サイクル数の差を出力する．

図 2 のように，地点 A でプリエンプションが発生した場合の性能悪化量を解析したとする．次に地点 A+1 で同様のことを行うのだが，プログラムをまた最初から地点 A+1 まで実行するのは明らかに無駄であるので，これを省略したい．

そこで，以下のような手順で各プリエンプション発生候補地点における性能悪化量を調べることにした．

(1) プログラムを通常通り実行していく．

(2) プリエンプションの発生候補地点に到達したら，現在の全プロセッサ状態を退避しておき，パイプラインをフラッシュする．

(3) 分岐予測ミスが発生したら，あらかじめ保存されているプリエンプション未実行時のパイプライン状態と現在のパイプライン状態を比較する．状態が一致したら，総 CPU サイクル数の差を求めて出力し，退避しておいた全プロセッサ状態を復帰させる．

(4) 2 に戻る．

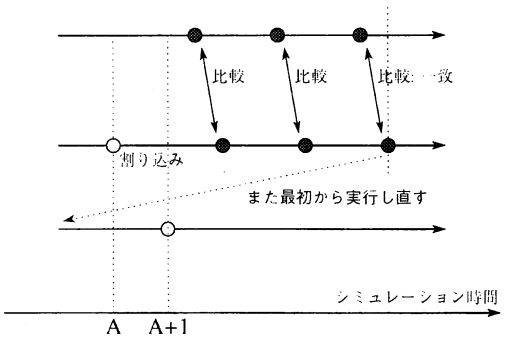


図 2　状態の退避・復帰の必要性

3.4 キャッシュメモリなどへの拡張

ここまでの議論は，プリエンプションの影響を命令パイプラインのみに限定したものであった．最悪性能予測をより実用的にするためには，これをキャッシュメモリなどにも広げて行かなくてはならない．以下，プリエンプションの対象をキャッシュメモリにまで広げた場合の考え方を説明する．

プリエンプションによってキャッシュメモリがすべて無効化されると，それがプリエンプション未発生時のキャッシュメモリの内容と一致するようになるまでには，最低でもプリエンプション前に有効だったキャッシュブロックをすべて有効にしなくてはならない．これはパイプラインと比べてかなり時間がかかると思われるので，プリエンプションが発生しない場合と発生した場合を比較するのではなく，プリエンプションが地点 A で発生した場合と地点 A+1 で発生した場合の比較を考えることにする．

地点 A でプリエンプションが発生した場合と，地点 A+1 でプリエンプションが発生した場合の，キャッシュメモリにおける違いは，地点 A で実行された命令によるメモリアクセスの結果がキャッシュメモリに反映されているか否かである（図 3）．両者にはこの他に命令パイプラインの違いも存在する可能性があるが，これはごく早期に収束すると考えられる．一方でキャッシュメモリの差異は，このキャッシュデータを

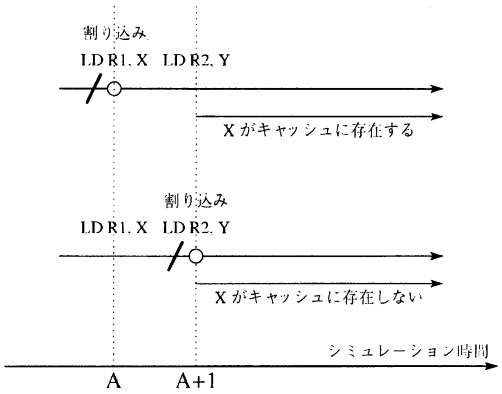


図 3　キャッシュメモリの差異

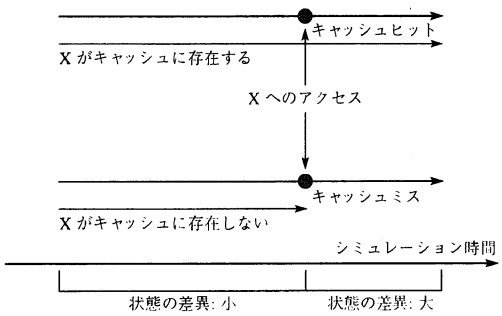
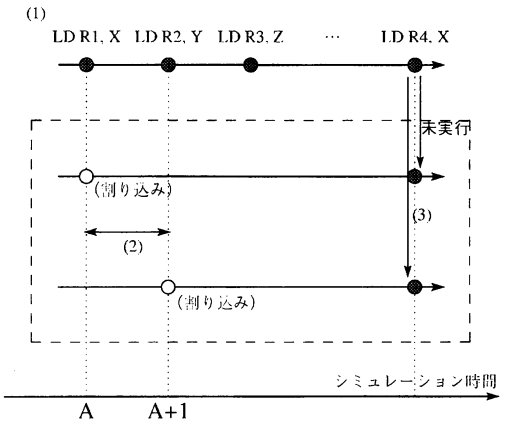


図 4　キャッシュメモリの差異の発現

利用するようなメモリアクセスが発生するまで潜在的に存在し続ける．そのようなメモリアクセスが発生すると，地点 A でプリエンプションが発生していた場合はキャッシュヒットとなるのに対し，地点 A+1 でプリエンプションが発生していた場合はキャッシュミスとなり，パイプラインが再び乱れることになる（図 **4**）．しかしながらこれでキャッシュメモリの差異は霧消し，パイプラインがじきに収束して 2 つの場合におけるプロセッサ状態は完全に一致する．

これで地点 A と地点 A+1 における性能悪化量の差が判明するので，次は地点 A+1 と地点 A+2 について同様の処理を行う．これを繰り返せば実行の省略が行える，というものである．

しかしながら，キャッシュメモリの潜在的な差異を残しながらパイプラインが一致した地点から，この潜在的な差異が発現する場所までは，どのくらいの時間がかかるのかが不明瞭である．この区間がプログラム全体に対して無視できないほど大きいこともありえるので，この手法により省略できる実行時間というのはごく限られたものになってしまう．そこで，無視できないほど長い時間がかかる場合に対応するための方法について考察する．



(1) プリエンプションなしで実行しメモリアクセスのログを保存
(2) ログからキャッシュの差異が X であると判明する
(3) 次に X にアクセスする地点がここであると判明する

図 5　メモリのアクセスログ

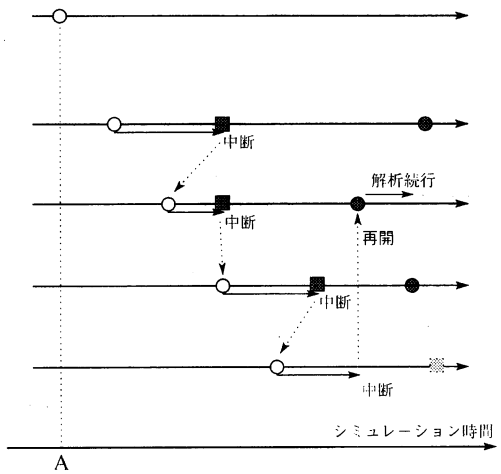


図 6　解析の中断と再開

キャッシュメモリに潜在的な差異があったとしても，それが発現するまでの間は地点 A と地点 A+1 それぞれの場合におけるプロセッサ状態には（前述の潜在的な差異を除いて）何ら差はないものと考えられる．また，地点 A と地点 A+1 それぞれでプリエンプションが発生した場合のキャッシュメモリの差は，プリエンプションを発生させずにプログラムを実行する際にメモリのアクセスログを取ることで容易に判断すること

CPU	Intel Pentium4 Xeon 2.8GHz (2CPU) (HyperThreading 有効)
主記憶	3GBytes
OS	Vine Linux 2.6r3 (Linux 2.4.20)
シミュレータ	SimpleScalar 3.0c
対象プログラム	9-Queen
総 CPU サイクル	6,476,959
割り込み候補区間	1,000,000 ~ 1,100,000

表 1　評価環境

命令発行幅		4
RUU		16
LSQ		8
メモリポート		2
機能ユニット	INT-ALU	4
	INT-MUL/DIV	1
	FP-ALU	4
	FP-MUL/DIV	1

表 2　シミュレータの設定

ができる．同様にアクセスログを参照すれば，この差異が今後どの地点で発現するかということもあらかじめ認識できる（図 **5**）．これを利用し，最初にパイプラインが一致した時点で一時的に解析を中断し，次の地点 A+2 の解析に移ることとする．解析を中断しながら次々に地点を進めていき，差異の発現が発生する直前までプログラムが進んだら中断した解析を再開する（図 **6**）．

このようにして，キャッシュメモリを考慮した最悪性能予測を高速に行うことができる．また同様の手法は TLB や分岐予測器についても適用でき，これらを考慮したより精密な性能予測も高速に行うことができる。

4. 実　　装

4.1 対象とするシミュレータ

今回はプラットフォームとして SimpleScalar Toolset 3.0c の PISA target を用いる．また評価に用いたのは *sim-outorder* である．SimpleScalar にはパイプラインをフラッシュする機能は存在しないので，これを付け加える必要があった．

4.2 状態の保存

まずプリエンプションが発生しない場合のプロセッサ状態を保存する必要がある．そこで，解析を始める前にプログラムを通常通り実行し，分岐予測ミスが発生した地点におけるプロセッサ状態を逐次保存する．今回はプロセッサ状態として命令パイプラインのみを扱うので，保存するのはパイプラインの状態だけでよい．

4.3 全プロセッサ状態の退避・復帰

前章で提案した設計では，全プロセッサ状態の退避・復帰処理が必要となる．今回の実装では，この処理を行うために fork システムコールを利用することにした．

5. 評　　価

9-Queen 問題を解くプログラムに対し，最悪性能予測を行った結果を図 **7**，表 **3**，表 **4** に示す．またこのときの環境は表 **1**，シミュレータの設定は表 **2** の通り

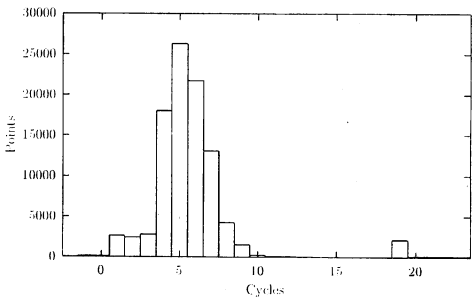


図 7　最悪性能予測の実行結果（最悪性能）

実行種別	時間 [sec]
プリエンプションなし	14.28
10 万候補点	5626

表 3　最悪性能予測の実行結果（実行時間）

プリエンプション後状態が一致するまでの比較回数	箇所数
1 回	95855
2 回	421
(2 回目までにすべての場合で状態が一致)	

表 4　最悪性能予測の実行結果（比較回数）

である．

表 3 から，最悪性能予測などを何も行わずにこれらのプログラムを実行すると 14 秒かかる一方で，今回 10 万候補点をとって実行すると 5626 秒で実行が終了している．5626/100000 = 0.05626 なので，候補点一つあたり 56 ミリ秒で解析が行えたことになる．

仮に，この解析をパイプラインの状態比較による実行省略なしに行ったとすると，各候補点ごとに残り 550 万～540 万サイクルの実行を行うことになる．650 万サイクルの実行に 14 秒かかっているので，550 万サイクルの実行には単純計算で 11.9 秒かかることになる．これを各候補点ごとに繰り返すわけであるから，11.9 × 100000 = 1190000 秒，およそ 331 時間かかることになる．今回 1.6 時間程度で実行できたことから，実行省略による高速化という目的は十分達成できてい

るということができる．

また表4から，今回の実行では，割り込み発生後1回目の分岐予測ミスでパイプラインは99%以上一致している．このことから，パイプラインはプリエンプション発生によってフラッシュされても，その後速やかに収束・一致することが確認できた．

6. 今後の課題

今回の実装はパイプラインの比較のみを行うものであり，キャッシュメモリは考慮しないことになっている．従って，まずキャッシュメモリについても比較するような実装にすることが求められるだろう．

その他，本論文で述べられている手法では，以下のような前提を必要としている．

- プリエンプションの発生候補地点としてプログラム中の連続する区間を使用する．
- プリエンプションの発生回数は1回とする．

前者はキャッシュメモリへ拡張する手法に存在する制限で，この手法では比較する2つのプリエンプション発生候補地点が離れれば離れるほどパイプラインやキャッシュメモリの差異が大きくなるため，プロセッサ状態が一致するまでに要する時間が大きくなってしまう．後者については最悪性能予測に本質的に存在する問題で，発生回数が複数回になればその発生パターンは指数的に増加していくために，計算量や必要な記憶容量もそれに比例する形で増加することになる．しかし複数の割り込み発生箇所の相互間隔が十分に大きければ，個々の割り込み発生箇所の周辺区間内の割り込み候補点に関する性能予測は，3章で述べた手法を用いれば他の箇所とは独立に行うことができる．実際，キャッシュミス回数が最悪となるような複数の割り込み発生箇所を求める研究[3]によれば，相互の間隔は数10万サイクルのオーダであり，独立した解析を行える可能性が高い．

7. まとめ

今回，SimpleScalarを改変して，連続する特定区間においてプリエンプションが発生した場合の最悪性能を調査することができた．このプリエンプションが対象としているのは命令パイプラインのみではあるものの，本来必要な時間に対して極めて少ない時間で調査を行うことができる．また，今回用いた方法をキャッシュメモリに対して適用する手法を提案した．

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) 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 (2003), 研究報告「計算機アーキテクチャ」No. 155
3) 宮本 寛史, 飯山 真一, 冨山 宏之, 高田 広章, 中島 浩: キャッシュフラッシュの最悪タイミングの探索手法 (2004), SACSIS 2004