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

2007－ARC－171 (17)
2007－EMB－ 3 (17)
2007／1／23

少しは組込的なアーキテクチャシミュレーション —色々やってわかったこと—

中島 浩†

筆者らは最近10年間ほど，マイクロプロセッサやシステムのアーキテクチャレベルのシミュレータ高速化について，さまざまな研究を行ってきた．これらの研究の直接的な成果は，たとえばSimpleScalarをx倍高速化したといったシミュレーション技術そのものに関するものであるが，シミュレーション対象であるマイクロプロセッサやそのワークロードについても副次的な知見がいくつか得られている．これらの知見がマイクロプロセッサや組込みシステムの技術に貢献するものであるか否かは定かではないが，これらの分野の研究を進める上で何らかのヒントになる可能性もあると考え，本報告にてその一端を紹介する．

Architecture Simulation with a Certain Level of Embedded Flavor —What We Learned from Our Experiments—

HIROSHI NAKASHIMA†

We have pursued research work on the performance improvement of architectural simulators for microprocessors and systems with them for about ten years. Besides the direct contribution of these researches to accelerate, for example, SimpleScalar by x times, we obtained byproduct insights of the behavior of the simulation targets, microprocessors and their workloads. Although it is quite unclear whether these insights are useful and/or meaningful for microprocessor and embedded technologies, they could trigger some good idea in these technology fields and thus are introduced in this report.

1. はじめに

筆者らは近年，マイクロプロセッサやそれを搭載したシステムの，アーキテクチャレベルシミュレーションに関する研究を行っている．研究の対象は，マルチプロセッサ，単体プロセッサ，および最悪割込み遅延に大別され，以下の成果を発表している．

(1) マルチプロセッサシミュレータ
ソフトウェアDSMを応用した参照フィルタリング[1),2)]による分散シミュレータShaman[3),4)]．

(2) 単体プロセッサシミュレータ
- ワークロードごとに最適化された命令レベルシミュレータの生成による高速化技術[5),6)]．
- out-of-order 命令スケジューリング計算を計算再利用により高速化したBurst-Scalar[7),8)]．
- out-of-order シミュレーションの時間軸方向の並列化による高速化技術[9),10)]

(3) 最悪割込み遅延
- メモリアクセストレースや分岐トレースの解析による最悪フラッシュタイミング解析[11)~13)]．
- out-of-order シミュレーションによる最悪割込み遅延の直接算出[14),15)]．

これらはそれぞれ，従来手法に対する大幅な高速化や，困難視されていた問題の現実的な時間での求解など，シミュレーションや最悪性能解析の技術に貢献するものであった．また，これらの直接的貢献とは別に，シミュレーションや解析の対象であるマイクロプロセッサやそのワークロードについて，副次的な知見がいくつか得られた．それらの知見は，必ずしもプロセッサアーキテクチャや組込みシステムの技術に対して直ちに貢献するものとは限らないが，技術の進歩のために何らかのヒントとなりうる可能性はある．そこで，得られた知見の一端とである下記のものについて，筆者なりの考察を若干加えつつ紹介することとする．

† 京都大学
Kyoto University

- 忘れっぽくパターンにはまるパイプライン
- 割り込みは思ったほどは悪させず
- 強力なパワーが潜む再利用

2. 忘れっぽくパターンにはまるパイプライン

out-of-order スーパスカラーのパイプラインは，非常に複雑で大規模な状態機械であり，直感的にはその挙動を把握あるいは予測することは困難に思える．しかし，SimpleScalar の時分割並列化[9),10)]や，SimpleScalar を用いた割込み遅延解析[14),15)]の高速化技法は，パイプラインが非常に忘れっぽいこと，すなわち過去の内部状態への依存が極めて薄弱であることを前提としており，実際にこの前提が成り立つことが観測されている．たとえば時分割並列化では，任意の時点でパイプラインを空にしても，高々1000 命令を実行すればその影響は消えてしまうことが確認されている．また割込み遅延解析では，ある命令の前後での割込みについて，両者のタイミングの違いによるパイプライン状態の差異は，僅か数命令の実行で解消されることも確認されている．

また BurstScalar は，ループの繰り返しごとのパイプライン状態が少数のパターンの繰り返しになることを前提に，一度行ったパイプライン状態遷移の計算を再利用することで高速化を図っているが，やはり期待通りに再利用することができている．このことは，たとえばループを何回繰り返したかといった過去の経緯にパイプライン状態が左右されにくいことを意味すると同時に，潜在的には膨大な状態空間の中のごく一部だけが発現していることも意味する．またパターンの繰り返しという現象は，割込み遅延解析においてパイプラインの微視的な差異の繰り返しという形でも観測されている．

さて，このようなパイプラインの「忘れっぽさ」や「パターンにはまりやすさ」といった「性格」が，直接マイクロアーキテクチャの設計に利用できるかというと，残念ながらそのような魔法を（少なくとも筆者の愚脳では）思いつくことはできない．すなわち，たとえば 1000 サイクル以前の状態が絶対に影響しないことがわかったとしても，入力記号数 N について $1000 \log N$ ビットでパイプラインが表現できることがわかるだけであり☆1，それで最適なパイプライン回路の設計ができるわけではない．またあるワークロードについて M 個のパイプライン状態しか出現しないとわかったからといって，$\log M$ ビットのパイプライン回路が作れるわけでもなく，仮にできたとしても他のワークロードには適用できない．

しかしパイプラインの「性格」は，短期的な記憶に基づいてその挙動を予測しやすいことを示唆しており，これに基づく最適化ができる可能性はある．たとえば昨今流行の低電力化を例に取ると，命令フェッチ／発行幅，物理レジスタ数，命令ウィンドウサイズ，機能ユニット数などのパラメータを，ワークロードの性質に応じて動的に最適化する（e.g. 非効率な部分の電源を切る）ことが考えられる．このような動的最適化にはパイプライン挙動の予測が不可欠となるが，たとえば特定のループの実行中の挙動は比較的予測しやすく，細かい制御，たとえばある命令がデコードされたら，ある機能ユニットの電源をオンにする，といった制御が可能ではないかと考えられる．また動的な傾向把握や予測のハードウェア実装が困難であったとしても，組込みシステムのように一つのプログラムが繰り返し実行されるような環境では，最適な制御シナリオをシミュレーション等による解析で作成し☆2，簡単なハードウェアでシナリオにしたがって制御する，といった方法は比較的高い実現性を有していると考えられる．

3. 割り込みは思ったほどは悪させず

組込みシステムの分野で，キャッシュ，分岐予測，out-of-order 実行といった「定跡」の技術が嫌われる理由として，これらの機構の性能予測性が悪いことが挙げられる．特に割込みが生じるとこれらの機構が依拠する局所性が失われ，性能に致命的な悪影響が生じるという一種の「迷信」があり☆3，リアルタイム性が要求されるシステムには適合しないという「風説」がある．

しかし我々の研究結果[11)~15)]によれば，1 回の割込みによるキャッシュミス増は高々1,000 のオーダ，分岐予測ミス増は高々10,000 のオーダ☆4，また実行サイクルの増加は高々100,000 のオーダに留まる．このことは，たとえば 1 GHz のプロセッサが実行しているリアルタイムプロセスが 1 ms に 1 回の割合で割込まれる

☆1 入力記号が 100 ビットで表現できるとして 100 Kbit ということになり，これ自体が全然小さくない．

☆2 したがって我々の飯の種が生じる．

☆3 「おまえの論文の Introduction にも『迷信』が書いてあるぞ」という文句が来る前に慌てて弁明しておくと，「大きなダメージを生じうる」や"significantly large delay" などの表現を用いており，また我々の研究の目的は仮に迷信であればそれを明確な根拠で打破する実用的ツールを作成することにあり，したがって十分な意義を持っている（ふぅ）．

☆4 分岐予測ミス増が多いのは，割込みが 2 ビットカウンタに対して非常に「不幸な」最悪値を設定しうるからである．

として，その「内部遅延」は10％未満に留まることを意味する．したがって「外部遅延」の要因であるリアルタイムOSのスケジューラや割込みハンドラ，さらには周期起動される高優先プロセスの実行時間を許容値に抑えることができれば，割り込まれたプロセスのリアルタイム性は十分確保できるものと考えられる．

ここで重要なポイントは，割込み内部遅延という直感的に把握困難な値を正確に見積もることであり，そのためには実用的かつ精密な性能予測ツールが不可欠である．たとえば我々の最悪遅延解析システム[15]は，割込み回数が1回には限定されているが，逐次環境で5 KIPS程度の性能，すなわち1 GIPSマシンで1秒を要するワークロードの解析を2日程度で完了する「実用性」を有しており，さらにごく単純な並列化で1桁程度の改善（2日を5時間に短縮）が可能である．割込みが複数回の場合については，内部遅延が10％程度であれば単純に割込み回数を乗じる「安全」な見積もりでも実用的上限が得られるので☆，高性能プロセッサでの「定跡」を組込みシステムに「安心して」応用できることはまず間違いないものと考えられる☆☆．

4. 強力なパワーが潜む再利用

我々の高速化手法は全て，単純な実装では実施されている計算を，何らかの方法で省略しつつ全く同じ結果を得るという技法に基づいている．パイプライン状態遷移の計算を再利用するBurstScalarはもちろんであるが，これらの省略技法の多くは再利用の一種と捉えることができる．たとえばSimpleScalarを用いた割込み遅延解析では，異なる割込み点に関する2種類の実行中に両者のパイプライン状態が一致すると，その一致関係を壊すような事象，たとえばキャッシュアクセス遅延の不一致などが生じるまでは，一方の実行のシミュレーションを省略する．この技法は，省略しない側の計算を省略する側が再利用するという，一種の再利用計算であると捉えることもできる．またこの2例では再利用の効果がかなり違い，前者では大きいとはいえ定数倍の計算量削減であるのに対し，後者では計算量のオーダが$O(N^2)$から$O(N \log N)$に削減されるという非常に大きな効果がある．

一方，再利用を行うためのプロセッサアーキテクチャや，コンパイラが一種のプログラム変換を行って再利用を行う方式について，多数の研究が我々自身のもの[16),17]も含めて行われているが，上記の再利用アルゴリズムとの間には大きなギャップがある．たとえば比較的わかりやすいBurstScalarの再利用であっても，以下の理由から自動的な再利用技術の適用はほぼ絶望的である．

(1) 多くの自動再利用技術では，レジスタやアドレスも含めたメモリ参照の一致を再利用条件としている．一方BurstScalarの主要な再利用条件であるパイプライン状態の一致は，循環バッファで表現されたキューの有効範囲の比較のためアドレスが一致しないばかりでなく，その内容も「意味的に」一致する（e.g. キューの先頭からの距離が等しいポインタ）ことを条件としており，値自体の一致がそもそも必要ではない．

(2) メモリデータも対象とする自動再利用技術では，再利用対象計算区間中で参照され，かつ区間外で定義される全てのメモリデータが一致することが再利用条件となる．一方BurstScalarでは，パイプライン状態一致から確実に一致が帰結されるメモリ上のデータについては，一致判定を省略して比較オーバヘッドを最小化している．割込み遅延解析ではこの一致判定省略をさらに進め，対象計算区間長とは独立な定数回の判定に抑えることで，計算量オーダを削減している．

(3) 上記の2項目とも関連するが，自動再利用技術では全ての入力の一致判定を一度に行って，再利用の可否を判断する．これに対しBurstScalarや割込み遅延解析では，計算区間への入力の一部分（パイプライン状態）の一致のみを再利用条件とし，他の必要条件の成立を仮定した一種の投機実行を行う．この結果，潜在的には膨大な数になる入力変数の一致判定を，それらに基づく計算の結果の一致判定に置き換えることができる．たとえばある数列の和が一致することが再利用条件であるとすると，自動再利用技術では全要素の一致を判定するため比較コストが大きく一致率も低下するのに対し，再利用「型」プログラムでは対象区間中の総和計算だけを投機的に実行してその結果を比較するため，比較が1回で済むと同時に一致率も飛躍的に向上する．

以上のように，自動再利用技術と，プログラムあるいはアルゴリズムレベルでの再利用技法には隔絶的な差があり，したがって両者の効果も隔絶している．しかし「手動」の再利用はしばしば非常に煩雑なプログラミングを必要とし，さらに再利用論理の正当性の検証が極めて困難という深刻な問題もある☆☆☆．した

☆ 「ではお前の論文で将来課題としている複数回割込みの解析は無意味か」という文句が来る前に慌てて弁明すると，安全な見積もりが悲劇的に過大な値となる可能性は常にあるため，やはり複数回割込み解析は必須である（ふぅ）．

☆☆ したがって再び我々の飯の種が生じる．

☆☆☆ たとえば割込み遅延解析では，再利用の有無によって1万倍程

がって，両者の断絶を埋めるための手段，たとえば再利用の適用やその条件をプログラマが「宣言」し，その正当性を何らかの手段で検証した上で，計算区間の入出力の保存や再利用条件判定のための検索をハードウェアあるいはソフトウェアライブラリで行うといった，半自動的な再利用技術が有望であると同時に今後の計算最適化技術の重要なテーマであると考える☆.

5. おわりに

本報告では，アーキテクチャシミュレータに関するさまざまな研究開発を通じて得られた副次的知見をいくつか紹介し，極めて主観的かつ粗雑ではあるが各々についての考察も披瀝した．これらの知見・考察が，プロセッサアーキテクチャや組込みシステムの技術進歩に貢献するかについては甚だ心もとないが，何らかのアイデアの小さな萌芽にでもなれば幸甚である．

なおこれらの技術分野に対して，確実に有意義な知見を最後に紹介する．それは，シミュレーション等を通じてプロセッサ，システム，ワークロードなどの挙動を詳細に観測することの有用性である．時間軸に（場合によっては空間軸にも）沿った観測データが，GFLOPS や IPC といった全体的・巨視的な性能値からは得ることができない，新しい知識や発見をもたらすことは疑いない．ここをこうすると性能が 10%向上する，といった研究の合間にでも，一度詳細な観測データを色々な距離・角度から眺めて，ご自身のハードウェアやソフトウェアがどのように振舞っているかを体感されることを，是非ともお勧めしたい．

参考文献

1) Imafuku, S., Ohno, K. and Nakashima, H.: Reference Filtering for Distributed Simulation of Shared Memory Multiprocessors, *SS 2001*, pp.219–226 (2001).
2) 今福　茂，大野和彦，中島　浩：共有メモリ・マルチプロセッサの分散シミュレーションのための参照フィルタ方式，情処論 HPS，Vol.42, No.SIG 9(HPS3), pp.93–105 (2001).
3) Matsuo, H., Imafuku, S., Ohno, K. and Nakashima, H.: Shaman: A Distributed Simulator for Shared Memory Multiprocessors, *MASCOTS 2002*, pp.347–355 (2002).
4) 松尾治幸，今福　茂，大野和彦，中島　浩：共有メモリマルチプロセッサの分散シミュレータ Shaman の設計と実装，情処論 HPS，Vol.44, No.SIG 1(HPS6), pp.114–127 (2003).
5) 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, Vol.46, No.SIG12 (ACS11), pp.98–109 (2005).
6) Nakada, T., Tsumura, T. and Nakashima, H.: Design and Implementation of a Workload Specific Simulator, *SS 2006*, pp.230–243 (2006).
7) 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，情処論 ACS，Vol.45, No.SIG 6(ACS6), pp.54–65 (2004).
8) Nakada, T. and Nakashima, H.: Design and Implementation of a High Speed Microprocessor Simulator BurstScalar, *MASCOTS 2004*, pp.364–372 (2004).
9) 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，Vol.46, No.SIG12 (ACS11), pp.84–97 (2005).
10) Yano, M., Takasaki, T., Nakada, T. and Nakashima, H.: An Accurate and Efficient Time-Division Parallelization of Cycle Accurate Architectural Simulators, *SS 2007* (2007). to appear.
11) Miyamoto, H., Iiyama, S., Tomiyama, H., Takada, H. and Nakashima, H.: An Efficient Search Algorithm of Worst-Case Cache Flush Timings, *RTCSA 2005*, pp.45–52 (2005).
12) 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効率的な探索手法，情処論 ACS，Vol.46, No.SIG 16 (ACS12), pp.85–94 (2005).
13) 小西昌裕，中島　浩，中田　尚，津邑公暁，高田広章：分岐予測器の最悪フラッシュタイミングの効率的解析手法，情処研報，2006-EMB-1, pp. 1–6 (2006).
14) 小西昌裕，中田　尚，津邑公暁，中島　浩：重複実行省略を用いた割り込みによるマイクロプロセッサの最悪性能予測，情処論 ACS，Vol.47, No.SIG12 (ACS15), pp.159–170 (2006).
15) Nakashima, H., Konishi, M. and Nakada, T.: An Accurate and Efficient Simulation-Based Analysis for Worst Case Interruption Delay, *CASES 2006*, pp.2–12 (2006).
16) 鈴木郁真，池内康樹，津邑公暁，中島康彦，中島　浩：再利用による GA の高速化手法，情処論 ACS，Vol.46, No.SIG 16 (ACS12), pp.129–143 (2005).
17) Tsumura, T., Ikeuchi, Y., Suzuki, I., Nakashima, H. and Nakashima, Y.: Design and Evaluation of an Auto-Memoization Processor, *PDCN 2007* (2007). to appear.

度の実行時間比が生じるため，結果の比較という最も基本的な検証すら不可能に近い．

☆ 実はこのような構想で科研費萌芽研究を行っており，やはり我々の飯の種であったりする．