C-013

スレッドレベル投機的実行に関する考察
A Consideration on Thread-Level Speculative Execution

斎藤 史子† 山名 早人‡
Fumiko SAITO Hayato YAMANA

1. はじめに

近年の命令レベル並列性の向上に伴い，さらなる並列性を引き出すためにスレッドレベル並列性が注目されている[2]．本稿は，その中でもスレッドレベルの投機的実行に着目した．スレッドレベル投機的実行には，load 命令を対象とした提案[3, 5, 8]と分岐命令を対象とした提案[1, 4]がある．load 命令を対象とした投機的実行と分岐命令を対象とした投機的実行は，同時に適用することが可能である．

load 命令を対象とした投機的実行は，delinquent load 命令（キャッシュ・ミスしやすい load 命令）と依存関係のある命令列をヘルパー・スレッドとして先行実行することで，キャッシュ・ミスを削減する．load 命令を対象とした手法は，ハードウェアを追加する実現方法[3, 5]とソフトウェアに変更を加える実現方法[8]が提案されている．

分岐命令を対象とした投機的実行は，難予測分岐を対象に分岐判定と依存関係のある命令列をヘルパー・スレッドとして先行実行することで，予測精度を改善する．分岐命令を対象とした手法は，ハードウェアを追加する実現方法[4]とハードウェアとソフトウェア両方に変更を加える実現方法[1]が提案されている．

近年，メモリ・アクセスによる遅延が深刻化し，キャッシュ・ミス・ペナルティが増大している．また，load 命令を対象としたスレッドレベル投機的実行の方が盛んに行われている．これらのことから，本稿では，load 命令を対象としたスレッドレベル投機的実行を中心にとり扱い，load 命令を対象とした代表的な研究とその動向についてまとめる．

2 節は，シミュレーション環境におけるスレッドレベル投機的実行に関する提案の調査，3 節は，実機におけるスレッドレベル投機的実行の適用，4 節は同期オーバヘッドに関する課題，5 節はまとめである．

2. スレッドレベル投機的実行に関する研究

load命令を対象としたSimultaneous Multithreading (SMT)環境におけるスレッドレベル投機的実行は，ハードウェアによる実現方法[3, 5]とソフトウェアによる実現方法[8]に大別できる．表1に，本稿で紹介する手法の比較を示す．

2.1 ハードウェアによる実現

ハードウェアによる実現方法として，Tullsen らの提案する Dynamic Speculative Precomputation(Dynamic SP)[5]と Roth らの提案する Data-Driven Multithreading(DDMT)[3]が挙げられる．

2.1.1 Dynamic SP[5]

Tullsenらの提案するDynamic SP(Dynamic Speculative Precomputation)では，ハードウェアが動的にスレッドを生成する点に特徴がある．

Dynamic SP の実現には，Delinquent Load Identification Table(DLIT)，Slice Information Table(SIT)，Retired Instruction Buffer(RIB)が必要になる．DLIT は，プログラムの実行中に delinquent load 命令(PC: Program Counter)を記録する．RIB は，trace を記録し，DLIT によって判断された delinquent load 命令を対象とするヘルパー・スレッドに含まれる命令列を生成する．SIT は，実行可能なヘルパー・スレッドの候補(trigger 命令[1]の PC)を登録し，trigger 命令が実行される場合に helper スレッドを起動する．Dynamic SP には，ヘルパー・スレッドがメイン・スレッドからしか生成されないという制限がある．

2.1.2 Data-Driven Multithreading

Rothらの提案するDDMT(Data-Driven Multithreading)[3]では，ヘルパー・スレッドによる先行実行結果を格納したレジスタを利用する点に特徴がある．

DDMTは，Data-Driven Thread Cache(DDTC)とIntegration Table(IT)を追加したSMTプロセッサである．

DDTCは，ヘルパー・スレッドを構成する命令のための命令キャッシュである．ヘルパー・スレッドの命令キャッシュ(DDTC)をメイン・スレッドの命令キャッシュと別に準備することによって，メイン・スレッドの命令キャッシュは，ヘルパー・スレッド実行による影響を受けずに済む．ITは，ヘルパー・スレッドがROB(Reorder Buffer)に登録されないため，ヘルパー・スレッドが利用中，もしくは，実行結果を保持するレジスタを解放しないように管理する．実行されるヘルパー・スレッドは，PTHSELと呼ばれる条件式の集合[7]によって，load命令の遅延に応じて静的決定される．

近年，DDMT は，CMP(Chip Multiprocessor)を前提とした，性能向上だけではなく消費電力に注目した研究[10]も行われている．

2.2 ソフトウェアによる実現

ソフトウェアによる実現方法[8]では，プロファイルから得られるキャッシュ・ミス情報やコンパイラの分析によって delinquent load 命令を特定し，ヘルパー・スレッドを生成する．スレッドを区別するために，ヘルパー・スレッド起動命令の追加など ISA(Instruction Set Architecture)に変更を加える必要がある．Software による実現では，helper スレッドがさらに helper スレッドを生成することもある．

† 早稲田大学大学院理工学研究科
‡ 早稲田大学理工学術院

[1] Helper スレッドの先頭の命令

表1：Speculative Computation

手法	Dynamic SP	DDMT	software 実装
スレッド生成	動的	静的	静的
load 命令遅延解決法	プリフェッチ	レジスタ値再利用	プリフェッチ
変更箇所	追加 HW(DLIT, SIT)	追加 HW(DDTC, IT)	ISA の拡張

3. スレッドレベル投機的実行の実機評価

2節に紹介した文献3, 5, 8, 10)では，シミュレーションによる評価しかされていない．2004年頃からhelperスレッドを実機に実装した評価も行われるようになってきている．ここでは，hyperthreadingが実装されたintelのプロセッサへの適用について紹介する．

3.1 Pentium 4[9)]

Kim らは，2.66GHz Pentium4 にヘルパー・スレッドを実装した．Kim らの評価では，SPECint2000 mcf において，ヘルパー・スレッドは約 50%のキャッシュ・ミスに適用されたにもかかわらず，2.7%しか性能が向上しなかった．同期オーバヘッドの大きい Windows API を利用せずに，スレッド同期処理用のハードウェア[12)]を導入することで，8.5%性能向上する見込みがある．

3.2 Xeon[11)]

筆者らは，2.4GHZ Xeonにおいてヘルパー・スレッドを実装した．筆者らの評価では，SPECint2000 181.mcf(ref 入力)において，SPECint2000 mcfは，Windows APIを使用したにもかかわらず，3.26%の性能向上を達成した．

4. 同期オーバヘッド削減の課題

SMT環境でのヘルパー・スレッド実装の際にはスレッド間の同期オーバヘッドが深刻な課題となる．データ依存を起因とするスレッド間の同期オーバヘッドを削減するだけでも，SPECint95, SPECint2000に含まれるプログラムで，平均31.76%[2]処理性能が向上するという報告[6)]がある．また，毎イタレーションごとの同期回数を削減するだけで，27.1%[3]性能向上するという報告[11)]もある．このように，同期オーバヘッドの与える影響は大きい．

SMT環境でのヘルパー・スレッドによる性能向上を実現するには，同期オーバヘッド削減に関する問題を解決することが不可避であると考えられる．

5. まとめ

本稿では，SMT 環境におけるスレッドレベル投機的実行に関する文献調査を行った．既存の SMT プロセッサ(Intel Pentium4, Xeon)による評価の結果，スレッド間の同期オーバヘッドによる影響を受け，キャッシュ・ミス削減の割に処理性能が向上しないことが判った．スレッド間のデータ通信や同期オーバヘッドを解決することで，30%以上の性能向上が期待できると考えられる．

[2] シミュレーションによる数値

[3] 実機評価による数値

謝辞

本研究の一部は，21 世紀 COE プログラム「プロダクティブ ICT アカデミア」，平成 17 年度科学研究費補助金基盤研究 (B)「ヘルパースレッドを用いたマルチスレッディングプロセッサのための高速化技術研究」，平成 17 年度早稲田大学特定課題研究助成費「SMT 環境におけるスレッドレベル制御依存／データ依存緩和手法」によるものである．

参考文献

1) Chappell, R. S., J. Stack, et al.: "Simultaneous Subordinate Multithreading (SSMT)", *Proc. of 26th ISCA* (1999)
2) Oplinger, J. T., D. L. Heine, M. S. Lam: "In Search of Speculative Thread-Level Parallelism", *Proc. of 8th PACT* (1999)
3) Roth, A. and G. S. Sohi: "Speculative Data Driven Multithreading", Proc. of 7th HPCA (2001)
4) Zilles, C. and G. Sohi: "Execution-based Prediction using Speculative Slices", Proc. of 28th ISCA (2001)
5) Collins, J. D., D. M. Tullsen, et al.: "Dynamic Speculative Precomputaion", *Proc. of MICRO-34* (2001)
6) Steffan, J. G., C. B. Colohan, el al.: "Improving Value Communication for Thread-Level Speculation", *Proc. of 6th HPCA* (2002)
7) Roth, A. and G. S. Sohi: "A Quantitative Framework for Automated Pre-Execution Thread Selection", *Proc. of MICRO-35* (2002)
8) Kim, D. and D. Yaung: "A Study of Source-Level Compiler Algorithms for Automatic Construction of Pre-Execution Code", *ACM Trans. on Computer Systems*, Vol.22, No.3 (2004)
9) Kim, D., S.- W. Liao, et al.: "Physical Experimentation with Prefetching Helper-Threaded Processors", *Proc. of 2nd CGO* (2004)
10) Renau, J., K. Strauss, et al.: "Thread-Level Speculation on a CMP can be Energy Efficient", *Proc. of 19th ISCA* (2005)
11) 本田，斎藤，山名："ハイパースレッディング環境における投機的スレッド間の同期手法"，情処研報 (2004-ARC-161) (2005)
12) Tullsen, T. M., J. L. Lo, et al.: "Supporting Fine-Grained Synchronization on a Simultaneous Multithreading Processor", *Proc. of 5th HPCA* (1999)