2011 年 10 月 31 日 情報システム基盤学合同輪講発表資料

Fast Thread Migration via Cache Working Set Prediction

著者： Jeffery A. Brown, Leo Porter, Dean M. Tullsen
出典： *the 17th International Symposium on High-Performance Computer Architecture*, pp 193-204, 2011.
発表者： 本多・近藤研究室　1153018　橋本崇浩

1　はじめに

近年，1 つのチップ上に複数のコアを持つチップマルチプロセッサ *(CMP)* の発展に伴い，スレッドレベル並列性の抽出が着目されている．なぜなら，複数のスレッドを同時に扱うことで，高い演算性能を獲得できるからである．

スレッドレベル並列性を向上させる手法の 1 つとして，スレッドを他のコアに移行する**スレッドマイグレーション***(TM)* が有効である．例えば 1 つのコアで複数のスレッドを実行している時，*TM* を行うことであるコアにおけるスレッドによる占有時間を削減し，遊休状態の他のコアを有効活用することができる．しかし *TM* は，その直後においてキャッシュミス (**コールドミス**) を頻発させ，性能低下を引き起こす．これは *TM* 先のコアのキャッシュには *TM* 前のキャッシュにあったデータがないことに起因する．そこで本論文では，コールドミスを防ぐために，そのスレッドが将来的に必要とするワーキングセットを予測し，*TM* 先のコアでプリフェッチする手法を提案している．

2　関連研究

TM 直後に生じるコールドミスへの対応策はいくつかある [1, 2]．例えば，*TM* と同時にキャッシュ内データをコピーすることで，コールドミスの発生を抑制できる．しかしコピーするデータが多い場合，もしくは *TM* の頻度が高い場合，そのオーバヘッドは大きくなりコールドミス抑制への効果を十分に得られない．また，コピーするデータが少ない場合，キャッシュミスをあまり削減することができない．すなわち，キャッシュ内データをコピーするだけではコールドミスを抑制するのは困難である．

これに対して本論文が提案する手法では，ロード命令などによってスレッドがあるデータを要求する前に，あらかじめそのデータをキャッシュにロードするプリフェッチを利用する．プリフェッチに必要な情報のみを TM 先のコアへ転送する．そのため，キャッシュ内データを転送するよりもオーバヘッドが小さいという利点を持つ．

3　提案手法

提案手法ではまず，メモリや命令キャッシュへのアクセスパターンなどを常に *Memory Logger Table* へ記録する．そして *TM* 時に，記録しておいた情報を移送先のコアへ転送し，移送先のコアでスレッドが再開するまでの間にプリフェッチを行う．

これらは *capture*，*summarize*，*apply* の 3 つのステージで構成されている．一連の動作は *TM* 実行中の間に行われるので，演算性能に悪影響を与えない．

3.1　capture

このステージではメモリや命令キャッシュなどへのアクセス命令を常にチェックし，それらを選択的に *Memory Logger Tables (MLT)* へ記録する．*MLT* を図 1 に示す．

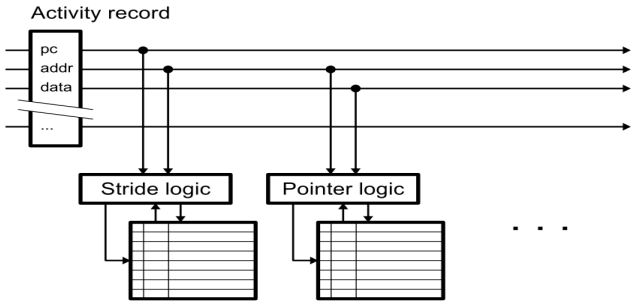

図 1: Memory Logger Tables (MLT)

MLT は 32 個のエントリを持つ連想メモリで構成されており，アクセス命令の一部をインデックスとして使用している．*MLT* にはいくつか種類があり，それらは様々なアクセスパターンを記録する．その一部を以下に示す．

- 一定の間隔でアドレスが変化するような場合，例えば配列への順次アクセスなど，そのアドレス差を記録する *(StridePC)*.
- ポインタの流れを記録する *(Pointer, Pointer-Chase)*. 例えば，ポインタベースのリスト構造など.
- 実際には *MLT* を使用せず，単にスタックポインタや PC を記録する (*SPWindow, PCWindow*).
- 命令，データキャッシュによって最近最もアクセスされたブロックを記録する ({*Inst,Data*}*-MRU*).

3.2　summarize

あるコアが他のコアへの *TM* シグナルを出した時，*MLT* に格納されているデータを抽出し，それを他のコアに転送する．抽出したデータは *TM* アドレス，ストライド，プ

リフェッチの量で構成されており，これらのデータは移送先のコアでプリフェッチに使用する．通常 *TM* は数 10〜数 100 サイクルの時間を要する．*summarize* ステージでは，この時間を活用して TM 先のコアへ抽出したデータを転送する．

3.3 apply

summarize ステージから送られてきた情報を *TM* 先のコアで受信したら，*apply* ステージではその解読を行う．スレッドを再開するためには *TM* 前のレジスタ状態をリロードする必要がある．このステージではこの間にプリフェッチを開始し，コールドミスを防ぐ．レジスタ状態のロードが完了次第そのスレッドは動作を開始する．

4 性能評価

CPU シミュレータ *SMTSIM* 上で，Spec2000 を用いて提案手法の性能評価を行った．*TM* は一定の時間間隔で行われ，すべてのキャッシュは空の状態から始まる．

4.1 キャッシュのコピーとプリフェッチ

まず最初に，キャッシュ内データのコピーやプリフェッチによる支援なしの *TM* と，提案手法に基づいて命令・データキャッシュへのプリフェッチを行う *TM* との比較を行う．図 2 に結果を示す．

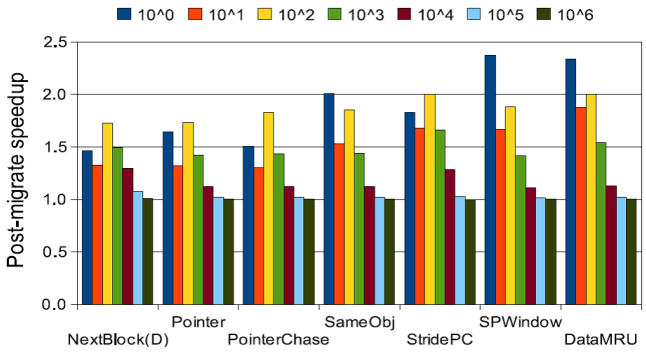

図 2: 命令コミット間隔ごとの TM 後の性能向上

縦軸が通常の *TM* のみを行った場合と比べて，プリフェッチ手法によりどの程度スループット向上を達成できているかを表している．横軸はプリフェッチに使用する各 *MLT* である．また，図にあるバーが各 *MLT* のコミット間隔を表している．コミット間隔とは，*TM* 後にスレッドが再開してから次の $1^0, 10^1, ...10^6$ 命令をコミットするまでの時間間隔である．

図 2 から，コミット間隔が短い時，すなわちスレッドが再開直後の時，大きくスループットが向上していることが分かる．特に，*SPWindow* や *DataMRU* は *TM* 直後に有効であることが分かる．これはスタックポインタや PC，最もアクセスされるブロックなどが *TM* 直後においてはタイムリーなデータだからである．

4.2 投機的マルチスレッディング

本論文では，実行時間が短いスレッドや *TM* が頻繁に起きる状況にも対応できることを目標としている．その環境の一例として，本論文の提案手法を投機的マルチスレッディング環境 (*SpMT*) に適用し，その評価を行った．*SpMT* は実行中のスレッドからメモリアクセスに関連する命令を抜き出し，投機スレッドを生成する．そして，その投機スレッドがメモリレイテンシのために時間のかかるロード命令を，実行中のスレッドの代わりに前もって，投機実行するものである．投機スレッドは実行時間の短いスレッド (平均 59 命令数程度) であり，数多く生成される特徴を持つ．

SpMT に提案手法を適用した結果を図 3 に示す．これは投機スレッドによってシングルスレッドのアプリケーションがどの程度スループットが向上したかを表している．図のバーは左から通常の *SpMT*(none)，命令・データキャッシュへのプリフェッチ (I+D Combo)，DataMRU のみ使用 (DataMRU)，次の 100 命令によって使用されるブロックをプリフェッチ (Oracle(D)) である．

通常の *SpMT* だけでもスループットが約 10%程度向上するが，提案手法を用いることによってさらに向上することがわかった．また，*SpMT* 環境では先の性能評価と異なり，DataMRU のみを用いたほうが I+D Combo よりも有効であることがわかった．

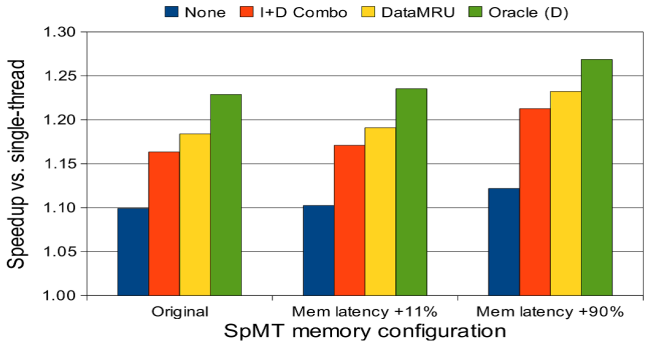

図 3: 提案手法による SpMT 環境における性能向上

5 おわりに

本論文では，*TM* によって生じるキャッシュミス削減するために，将来的に必要となるワーキングセットを予測してプリフェッチする手法を提案し，その評価を行った．評価の結果，提案手法は命令数の少ないスレッドに対して特に有効であり，スループットを 2 倍以上向上可能であることもわかった．また *SpMT* においても，提案手法を用いることでパフォーマンス向上できることがわかった．

参考文献

[1] J.A.Brown and D. M. Tullsen, The shared-thread mutiprocessor, In 21st international Conference on Supercomputing, pages 73-82, June 2008.

[2] Theofanis Constantinou, Yiannakis Sazeides, Pierre Michaud, Performance Implications Of Single Thread Migration On A Chip Multi-core. 2005.