3J-1

同期付き共有変数キャッシュを搭載したオンチップマルチプロセッサの効果的なプリフェッチ命令の挿入

佐藤　芳紀　　山脇　彰　　岩根　雅彦

九州工業大学 工学部

1. はじめに

オンチップマルチプロセッサに搭載される TSVM (Tagged Shared Variable Memory)キャッシュ [1]は同期が必要な共有変数(同期共有変数，SV)をキャッシングし，同期通信をそれに対する一貫性制御と同時に実行してより効率的な並列処理の実行を図る．TSVM キャッシュがキャッシュミスすることで同期通信が遅れないように，TSVM キャッシュはプリフェッチ命令を持ち [2]，本稿では TSVM キャッシュの特性を考慮したプリフェッチ命令の挿入法について検討する．

2. オンチップマルチプロセッサ MOC/TSVM

図 1 に示す TSVM キャッシュを持つオンチップマルチプロセッサ MOC/TSVM は CPU コアと命令キャッシュ(IC)，データキャッシュから構成される．さらにデータキャッシュは一般変数をキャッシングする一般変数キャッシュ(GVC)と同期共有変数をキャッシングし，同期通信をそれに対する一貫性制御によって実現する TSVM キャッシュ(TC)から構成される．各プロセッサはシステムバス(SYSB)と TSVM 一貫性制御バス(TCB)へ接続される．SYSB と TCB はアービタによって調停される．

同期共有変数は同期情報の 1 つとしてワードの full/empty を表すカウンタを持つ．カウンタが 0 ならばそのワードは empty であることを示し，非 0 ならば full であることを意味する．empty なワードに対するロード命令や，full なワードへのストア命令はブロックされる．ワードのストア命令で指定されるカウンタ値はロード命令ごとにデクリメントされ，通信の完了時に使用されたワードは empty となり，暗黙的に再利用可能となる．

マルチスレッド実行環境において， TSVM キャッシュのエントリはタスクやスレッド毎に各エントリを動的に保護するためタスク ID(TID)，スレッド ID(THID)，変数 ID(SVID) の連結であるタグ(TAG)によって指定される．同期共有変数はメモリ上では変数に加え，タグと同期情報からなる構造体として扱われる．このため，TSVM キャッシュはタグとメモリ上の同期共有変数を対応付けるタグアドレス変換機構を持つ．タグアドレス変換は，メモリ上に存在する変換テーブル SVDT(SV Descriptor Table) と SVT(SV Table)を参照して行う．SVDT は SVT， SVT は SV 領域のベースアドレスを格納する．タグアドレス変換では，まず TSVM キャッシュが持っている SVDT のベースアドレスと TID によって SVDT を参照する．次に，SVDT から読み出した SV 領域のベースアドレスと THID によって SVT を参照する．そして，SVT から読み出した SV 領域のベースアドレスと SVID によって，メモリ上の同期共有変数のアドレスが決定する．

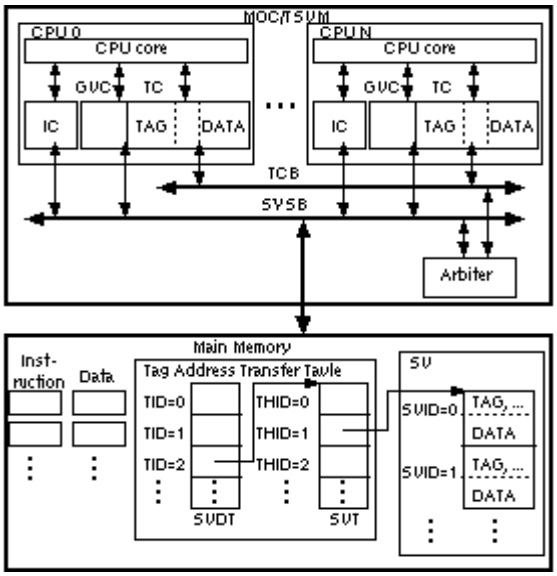

図 1．MOC/TSVM の構成

3. プリフェッチ命令

TSVM キャッシュアクセス時にキャッシュミスが発生した場合，必要とする同期共有変数をロードするために TSVM キャッシュはシステムバスの使用権を獲得した後，メモリ上のタグアドレス変換テーブルを参照して対応する同期共有変数のアドレスを得，その後メモリから同期共有変数の構造体をロードする．プリフェッチは，キャッシュミスが発生するであろうデータをメモリから予めキャッシュにロードし，キャッシュミスによるメモリレイテンシを隠蔽または軽減する．MOC/TSVM のプリフェッチ命令は並列処理における同期通信の形態に柔軟に適応できるように，PFTSVM(TAG，NUM，STRD)；のフォーマットでロードする同期共有変数の個数と間隔を指定できる．CPU がプリフェッチ命令を実行すると，指定された TAG，NUM，STRD に従い TAG を同期共有変数の先頭タグとして TSVM キャッシュを一致検索し，システムバスの使用権を獲得後，タグアドレス変換でメモリ上の対応する同期共有変数を指定し，STRD 間隔で NUM 個の同期共有変数をロードする．

4. プリフェッチ命令の挿入法

同期共有変数が多量に扱われ，プリフェッチ命令が必要となる例の 1 つとして Doacross ループプログラムが挙げられる．そこで，図 2 のような前方依存のループ伝播依存があり，依存距離が 1 の Doacross ループに対するプリフェッチ命令の挿入方法を提案する．

```
for(i=1; i<N; i++){
  ......
  x[i] = ...
  ...  = ... x[i-1] ...
  ......
}
```

図 2. 検討する Doacross ループ

図 2 に対し，従来よく用いられている並列化法は 1 つのイタレーションを単位として CPU に割当てる．プリフェッチ命令を挿入する場合は，次のイタレーションで必要となる同期共有変数のプリフェッチ命令を挿入する．また，最初のイタレーション実行に必要な同期共有変数のプリフェッチのためのプロローグ部を追加し，ループはカーネル部とプリフェッチ命令を実行する必要のない最後のイタレーションを実行するエピローグ部とにわける．プロセッサ数を p として並列化し，プリフェッチ命令を挿入したプログラムを図 3 に示す．図 3 において，pid は各 CPU が固有に持つ ID を示しており，$x は x が同期共有変数であることを示す．同期共有変数へのストア時に指定するワードの full/empty を示すカウンタ値 CN は，($x, CN)の形式で表した．

図 3 に示した方法に加え，1 イタレーションの演算が終了するまでにプリフェッチを完了させるためには，一般的にループが展開される．ループ展開数を 2 とした場合のカーネル部を図 4 に示す．

ただし，図 4 に示したループ展開の例で 1 つのプロセッサ内で同期共有変数$x の同一要素の定義，参照がなされる箇所については，プロセッサ間で同期通信する必要がなくなる．そこで，提案手法では同期通信の必要がなくなった$x については，同期共有変数に割当てず一般配列変数 x に置き換える．その結果，プリフェッチすべき変数はループ展開数によらず常に 2 変数のみとなる．図 4 に対し提案手法を適用した例を図 5 に示す．

5. 実験および考察

図 1 に示した MOC/TSVM の構成のもとで，クロック精度のシミュレーションで提案手法を評価した．実験環境ではプリフェッチ命令の実行時間はプリフェッチのオーバヘッドを含め 3 クロックを要する．さらに TSVM キャッシュの一致検索に 1 クロック，タグアドレス変換に 1 クロック，1 ワードの同期共有変数の読出しに 4 クロックを要する．評価プログラムは図 6 の Doacross ループを並列度 4 で並列化したものを用いた．1 イタレーションの演算が終了するまでにプリフェッチを完了させるため，ループ展開数 2 でループを展開し提案手法を適用した．比較対照として，提案手法では 1 ラインを 1 ワードとしてプリフェッチするのに対し，従来の 1 ライン複数ワードとしてプリフェッチする場合についてもプログラムを実行した．ここでは，1 ラインを 4 ワードとした．

プログラムを実行した結果，提案手法ではシリアル実行時と比較して 4.09 倍の速度向上を得た．一方，比較対照とした 1 ライン複数ワードでのプリフェッチによる結果は，シリアル実行と比較して 3.00 倍となった．よって，提案手法は 1 ライン複数ワードでのプリフェッチに対しても 1.37 倍の速度向上が得られた．これは，提案手法によって TSVM キャッシュの平均ミスレイテンシ

```
PFTSVM(&($x[pid-1]), 2, 1);          ] prolog
for(i=pid; i<N-p; i+=p){             ]
  PFTSVM(&($x[i+p-1]), 2, 1);        |
  ......                             |
  ($x[i], 1) = ...                   | kernel
  ... = ... $x[i-1]...               |
  ......                             |
}                                    ]
......                               ] epilog
```

図 3. 並列化とプリフェッチ命令の挿入

```
for(i=2*(pid-1)+1; i<N-(p*2); i+=p*2){
  PFTSVM(&($x[i+(p*2)-1]), 3, 1);
  ......
  ($x[i], 1) = ...
  ($x[i+1], 1) =...
  ... = ... $x[i-1] ...
  ... = ... $x[i] ...
  ......
}
```

図 4. Doacross ループ展開

```
for(i=2*(pid-1)+1; i<N-(p*2); i+=p*2){
  PFTSVM(&($x[i+(p*2)-1]), 2, 2);
  ......
  x[i] = ...
  ($x[i+1], 1) = ...
  ... = ... $x[i-1] ...
  ... = ... x[i] ...
  ......
}
```

図 5. 提案手法の適用

```
for(i=1; i<121; i++){
  x[i] = 6 * y[i];
  z[i]  = z[i] + x[i-1];
}
```

図 6. 評価に使用した Doacross ループプログラム

が削減されたことによる．プリフェッチ命令を挿入せずプログラムを実行した場合，350.75 クロックの平均ミスレイテンシが発生した．これに対し提案手法の場合，平均ミスレイテンシは 19.25 クロックまで減少した． 一方，1 ライン複数ワードでのプリフェッチを行った場合の平均ミスレイテンシは 247.75 クロックであった．

6. むすび

オンチップマルチプロセッサ MOC/TSVM において，提案手法を用いて Doacross ループを実行したところ，高い速度向上が得られた．

今後は，より多くの応用プログラムに対しプリフェッチ命令の挿入法を検討・評価する．

参 考 文 献

1) 小関大介:TSVM キャッシュとその予備的評価，九州工業大学大学院平成 15 年度修士論文

2) 有村雅彦:TSVM キャッシュへのプリフェッチ機構の導入とその評価，九州工業大学大学院平成 16 年度修士論文

3) S.P.Vanderwiel and D.J.Lilja:Data Prefetch Mechanisms,ACM Comput. Surv. 32(2):174-199(2000)