5T-5

ATTEMPTのキャッシュ コヒーレンシィプロトコル

寺沢卓也、天野英晴

慶応義塾大学理工学部

1. はじめに

メッセージパッシングは、様々な言語やOSにおいて、交信、同期の中心的手段として用いられ、その効率的な実装はシステムの性能に大きな影響を与えるようになってきた。現在まで、バス結合主記憶共有型のマルチプロセッサのための様々なキャッシュコヒーレンシィプロトコル提案されているが、これらのプロトコルにはメッセージパッシングの機能が組み込まれていない。しかし、ATTEMPTは高度な同期機構を持つので、これとキャッシュを組み合わせてメッセージパッシングを効率良く行うことができる。すなわち、同期変数やシングルワードのメッセージの転送には同期メモリを使い、ブロックメッセージ転送にはキャッシュをメッセージバッファとして用いる。本報告ではそのためのキャッシュプロトコルを提案する。

2. 設計の前提

ATTEMPTのキャッシュプロトコルを考える上での前提条件として以下の点がある。

1. ATTEMPTで用いるFuturebusは強力なブロック転送機能を持つ非同期バスで、アービトレーションも高速に行うことができる。
2. キャッシュメモリには高速のSRAMを、共有メインメモリにはアクセスタイムは遅いが大容量のDRAMを使う。

以上の点をふまえてプロトコルの説明を行う。

3. Core Keioプロトコル

ATTEMPTのキャッシュプロトコル(Keioプロトコル)はプロトコルにメッセージパッシングが組み込まれているため複雑である。そこで、まず核となるプロトコルであるCore Keioプロトコルを述べ、これにメッセージパッシングを組み込む。

他の多くのライトバック型のスヌープキャッシュ同様、Core Keioプロトコルにおいてもキャッシュブロックの状態を以下の3つの要素を用いて表現する。

- 有効か無効か (Valid or Invalid)
- 唯一か共有か (Exclusive or Shared)
- 変更されているか否か (Clean or Dirty)

Core Keioプロトコルではこれらの組み合わせにより Invalid (I), Clean-Exclusive (CE), Clean-Shared (CS), Dirty-Exclusive (DE), Dirty-Shared (DS) の5状態を用いている。この5状態の遷移を図1に示す。Core Keioプロトコルの特徴は以下の通りである。

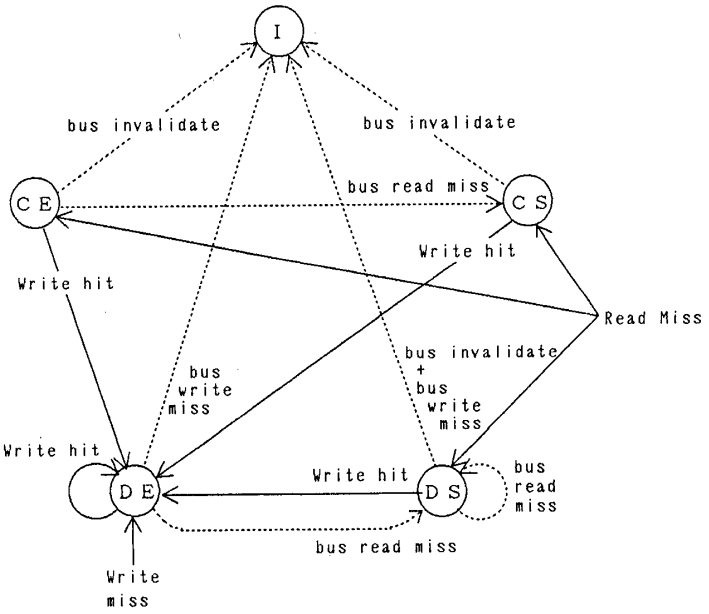

図1 Core Keioプロトコル

1. **書き込み時無効化型である。**
ライトバックプロトコルには同一コピーを持つ他のブロックに対しデータを同時転送してしまうブロードキャスト型と他のブロックを無効化してしまう無効化型がある。文献1等によるとブロードキャスト型の効率が良いことになっているが、これはすべての交信をキャッシュを介して行うことが前提になっている。ATTEMPTでは共有型と交換型に相互作用が分離され、データの交換が頻繁に行われる場合は同期機構が用いられる。このためCore Keioプロトコルではプロセスのマイグレーション等で有利な無効化型を採用した。また、この型はFuturebusのブロック転送機能を有効に利用できる利点がある。

2. **ミスしたブロックはできる限りキャッシュ間でブロック転送する。**
ATTEMPTにおいてはSRAMを用いたキャッシュの動作速度はDRAMを用いた主記憶の4、5倍である。このためCore Keioプロトコルはできる限り主記憶のアクセスを避ける方針で設計されている。ミスしたブロックを他のキャッシュが持っていた場合いずれかのキャッシュが応答し、主記憶はアクセスされない。主記憶へのライトバックが起こるのはリプレイス時に限られる。

4. Keioプロトコル

Core Keioプロトコルはキャッシュ間直接転送が積極的に用いられているため他のキャッシュ

A cache coherence protocol for ATTEMPT
Takuya TERASAWA, Hideharu AMANO
Keio University

に比べブロック単位のメッセージパッシングが効率良く行われる。しかし、それでもまだ次のような無駄が生ずる。

- メッセージ領域をキャッシュにマッピングしている場合、メッセージを送る時に送り手のキャッシュからメッセージを書き込んだブロックがリプレイスされている時にはメインメモリからロードしなければならない。
- 交信終了後、キャッシュ中のメッセージバッファブロックがリプレイスされる時にメインメモリへの書き戻しが起きてしまう。

これらの無駄を減少させるため次の2つの操作を導入する。

(a) Msgwrite (Address,data)
キャッシュブロックをメッセージバッファとして使う。ブロックはメッセージバッファとマークされる。この際、領域確保のため必要ならばブロックの書き戻しを行う (DE,DSが追い出される場合) が、データのロードは行わない。

(b) Free (Address,data)
ブロックはメッセージバッファの状態を解除され、以後、リプレイスの対象となる。リプレイスされる時はメインメモリへの書き戻しは行わない。

マークは送り手のプロセッサのキャッシュにのみ必要で、また、そのブロックがキャッシュ中にある時に限り有効である。マークされているブロックがリプレイスされる時には自動的にマークは解除される。以上の操作の導入により簡単な変更でキャッシュをメッセージ転送用バッファとして用いることができる。

5. シミュレーション

Core Keio プロトコル、Keio プロトコルを評価をするために文献1の方法を参考にシミュレーションを行った。このシミュレーションのモデルは Shared のブロックへのアクセスと、Exclusive のブロックへのアクセスの場合で異なっている。Exclusive のブロックへのアクセスは完全に確率で管理されており、プロセッサの待ち時間を計算するだけである。一方、Shared へのアクセスについてはアクセスの局所性を実現し、テーブルで各ブロックの状態を管理している。

図2にシミュレーションの結果を示す。文献1の結果とは多少異なっているが、それはメインメモリへのアクセスにかかる時間の設定が異なるためである。Keio プロトコル (メッセージパッシング対応版) では Shared ブロックへのアクセスのうち 60% をメッセージパッシングに割り当て、その 90% は同期メモリを用いている。この場合は他のプロトコルの中で最も良いといわれる Dragon と同等以上の結果が得られている。

Parameter
Reference to shared blocks (SHD)
Read reference (RD)
Hit ratio to private blocks (HIT)
Probability of modified (MD)
Percentage of Message Passing (MP)
Useful work cycle (W)
Cache cycle time
Main memory cycle time (CYRATE)
Block size (BSIZE)
Cache size (CSIZE)
Number of shared blocks (SBLKN)
Number of processors (PU)
Simulation cycle (CYCLE)
Depth of stack (DEPTH)

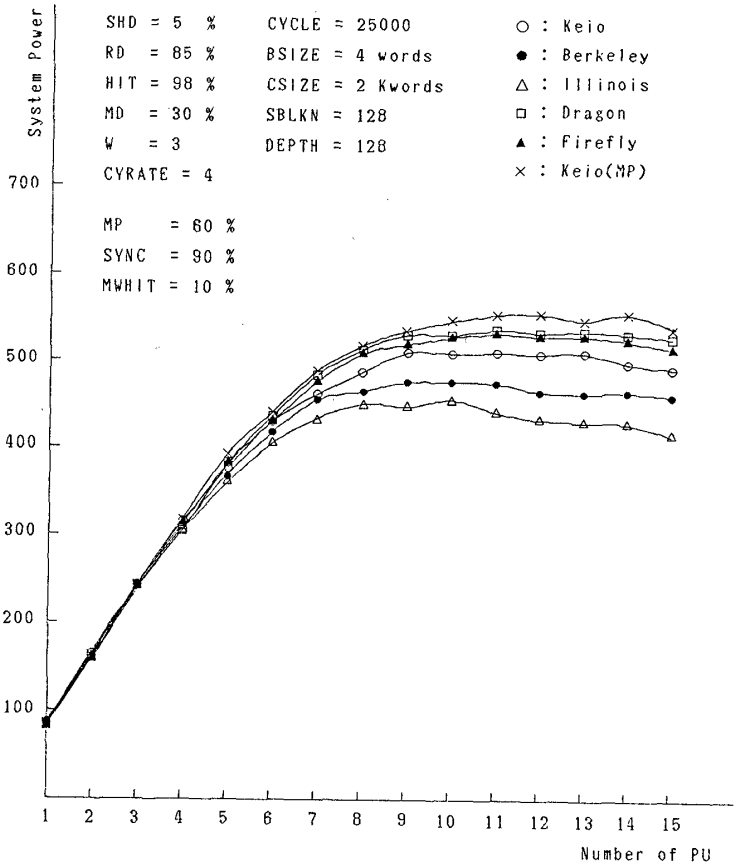

図2 シミュレーション結果
(プロセッサ数 対 性能)

6. おわりに

ATTEMPT は商用化を目指して第1バージョンを開発中であるが残念ながらこの第1バージョンのキャッシュはライトスルー型である。このキャッシュは4セットアソシアティブで 128Kbyte、ヒット時アクセスタイムは 65nsec である。第2バージョンでは Keio プロトコルが搭載される予定である。

参考文献

[1] J.Archibald and J-L,Baer, "*Cache Coherence Protocols : Evaluation Using a Multiprocessor Simulation Model*" ACM, Trans.on Comp.Sys., Vol.4, No.4, Nov. 1986