計算機アーキテクチャ 89−23
（1991. 7. 19）

拡張MOESIモデルと並列階層キャッシュ・プロトコル

東芝 総合研究所 情報システム研究所
村谷 博文
muratani@isl.rdc.toshiba.co.jp

並列推論マシンPIM/kは並列階層 キャッシュ・メモリをアーキテクチャ上の特徴としている。PIM/kの採用するキャッシュ・コンシステンシ・プロトコルは、Berkeleyプロトコルを階層化により拡張して得られたものである。本稿では、このような階層化されたキャッシュ・コンシステンシ・プロトコルを統一的に扱うためにMOESIモデルの拡張を行う。さらに、PIM/kが採用するキャッシュ・プロトコルの問題点を指摘し、それに対する改良プロトコルを提案する。

An entended MOESI model and parallel hierarchical cache protocols

Hirofumi Muratani

Toshiba R & D center, Information and System Lab.
1 Komukaitoshiba, Saiwai-ku, Kawasaki, 210, Japan

The parallel inference machine PIM/k has a multi-level cache/bus architecture. The cache consistency protocol implemeted on PIM/k is an extension of the Berekely protocol. We discuss an possible extension of the MOESI model, which makes it possible to treat a class of hierarchical cache consistency protocols unifyingly. Furthermore we point out a problem of superfluous memory-bus transactions in the current PIM/k protocol, and propose an improved protocol, expressed by the extended MOESI model, to solve this problem.

1.はじめに

我々は、第五世代プロジェクトの一環として、並列推論マシンPIM/kを開発している[1]。このマシンのアーキテクチャ上の特徴は並列階層キャッシュ・メモリ[2]である。スヌーピング・キャッシュ方式を採用しており、データの一貫性はハードウェアが保証する。採用されているキャッシュ・コンシステンシ・プロトコルはBerkeleyプロトコル[3]を階層化して得られるものである[4]。

従来の1レベルのキャッシュ・メモリに関しては、既に数多くのキャッシュ・コンシステンシ・プロトコルが提案されており、その効果の比較に関する議論がなされてきた[5,6]。一方、並列階層キャッシュ・メモリに関しては、キャッシュ・コンシステンシ・プロトコルの提案はそれほど多くない[2,4,7]。

一方、これまでに提案された1レベルのキャッシュ・コンシステンシ・プロトコルを統一的に記述する研究も行なわれている。たとえば、Futurebus+は次世代の標準バスとして仕様の検討が進められているが、その特徴の一つは、主なキャッシュ・コンシステンシ・プロトコルをサポートしていることである[8]。この標準化にあたりSweazy and Smithにより1レベルのキャッシュ・メモリにおけるキャッシュ・コンシステンシ・プロトコルを統一的に記述できるMOESIモデルが提案された[9]。

本稿では、このMOESIモデルを拡張し、並列階層キャッシュのキャッシュ・コンシステンシ・プロトコルを記述するためのモデルを提案する。従来のMOESIモデルでは、キャッシュ・ブロックが3つの属性で特徴付けられていたのに対し、この拡張されたMOESIモデルでは、キャッシュ・ブロックは5つの属性で特徴付けられる。

以下では、2章では、MOESIモデルとその拡張について述べる。拡張としては、階層化のための拡張のほかに、read broadcastプロトコルを包含するための拡張についても述べる。3章では、拡張されたMOESIモデルを用いてPIM/kプロトコルがいかに記述されるかを説明する。さらに、この拡張されたMOESIモデルで記述される別のキャッシュ・コンシステンシ・プロトコルとしてPIM/kプロトコルの改良案について説明する。この改良案はPIM/kプロトコルにおいて発生するバス・トランザクションのうち本来不要であるものの発生を防ぐために提案されたものである。4章は本稿のまとめである。

2.プロトコルの統一モデル

2.1.MOESIモデル

ここでは、Sweazy and Smithの提案するMOESIモデル[9]について説明する。MOESIモデルでは、キャッシュ・ブロックは3つの属性で特徴付けられる:

- validity

  そのキャッシュ・ブロックのデータが有効であるか(valid)いなか(invalid)。

- exclusiveness

  そのキャッシュ・ブロックのデータが他のキャッシュ・メモリと共有されているか(sharedまたはreplicated)いなか(exclusiveまたはunique)。

- ownership

  そのキャッシュ・ブロックがそのキャッシュ・メモリから追い出されるときに主記憶に書き戻す必要のあるデータであるか(ownedまたはdirtyまたはmodified)いなか(unownedまたはcleanまたはunmodified)。

キャッシュ・ブロックのとりうる状態は、5つである:

M(Modified)	: valid &	owned	&	exclusive
O(Owned)	: valid &	owned	&	shared
E(Exclusive)	: valid &	unowned	&	exclusive
S(Shared)	: valid &	unowned	&	shared
I(Invalid)	: invalid			

コピーバック・キャッシュのキャッシュ・ブロックの状態遷移表は表1(a)と(b)で与えられる。表1(a)はプロセッサからのコマンドに対するキャッシュの応答を表している。プロセッサがライト時にキャッシュ・メモリの応答が2通りあるのは、上段がwrite broadcastタイプ、下段がwrite invalidationタイプのキャッシュ・プロトコルに対

	Read	Write
M	M	M
O	O	CS:O/M,CC,IM,BC,W M,CC,IM
E	E	M
S	S	CS:O/M,CC,IM,BC,W M,CC,IM
I	CS:S/E,CC,R	Read>Write M,CC,IM,R

表1(a)

	CC,BC?	CC,IM	CC,IM,BC
M	O,CS,DI,DK S,CS,DI	I,DI,DK I,DI	-
O	O,CS,DI,DK S,CS,DI	I,DI,DK	S,SL,CS I
E	S,CS,DI	I	-
S	S,CS,DI	I	S,SL,CS I
I	I	I	I

表1(b)

応している。一方、表1(b)はバス・コマンドに対するキャッシュの応答を表している。CC,BC?はバス・コマンドread sharedである。MまたはO状態のキャッシュはデータを返す。上段は、データ転送時に主記憶へのコピーバックを伴わない応答、下段は伴う応答を表している。CC,IMはバス・コマンドread modifiedである。MまたはO状態のキャッシュはデータを返す。CC,IM,BCは上段がバス・コマンドbroadcast write、下段がバス・コマンドinvalidationである。やはり、上段は、データ転送時に主記憶へのコピーバックを伴わない応答、下段は伴う応答を表している。図1はMOESIモデルから主なキャッシュ・プロトコルが得られることを表している。MOESIモデルをread sharedとread modified時にコピーバックを行わないようにwrite invalidation化すると、5状態のwrite invalidationタイプのプロトコルが得られる(図1中のWI MOESI プロトコル)。このWI MOESIプロトコルにおいてE状態とS状態の間の遷移を短絡除去し、両状態を新たにS状態と見なすとBerkeleyプロトコルが得られる。また、read sharedとread modified時に主記憶にコピーバックするようにwrite invalidation化すると別の5状態のwrite invalidationタイプのプロトコルが得られる(図1中のWI+ MOESIプロトコル)。このWI+ MOESIプロトコルにおいてO状態を除去するとIllinoisプロトコルが得られる。Dragonプロトコルは MOESIモデルをread shared時にコピーバックしないようにwrite broadcast化すると得られる。また、read shared時に主記憶にコピーバックしないようにwrite broadcast化するとDragonプロトコルとは別の5状態のプロトコルが得られる(図1中のWB+ MOESIプロトコル)。このWB+ MOESIプロトコルにおいてO状態を除去するとFireflyプロトコルが得られる。

MOESIモデルはこの他にもWrite-throughプロトコルやWrite-onceプロトコルも包含している。

次章では、さらに多くのプロトコルを表現できるようにMOESIモデルの拡張を行う。

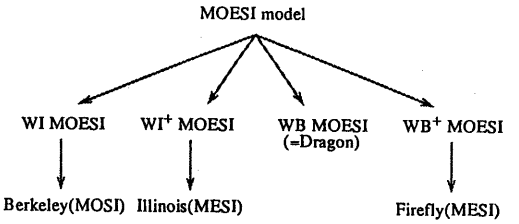

図1

2.2.拡張されたMOESIモデル

2.2.1.read broadcastへの拡張

read broadcastプロトコル[11]は複数のプロセッサがときおり更新される共有データにアクセスするような場合にinvalidationによるキャッシュ・ミスを減少させると期待されている*1。MOESIモデルを拡張してread broadcastプロトコルも統一的に記述できるようにする。

read broadcastプロトコルは、キャッシュ・メモリがバスを絶えず監視していて、read sharedに対する応答としてデータが返されるとき、そのデータのアドレス・タグを持つinvalidateされたキャッシュ・ブロックにデータを書き込む。

バス上にデータが転送されるのはread shared以外にも、キャッシュ・ブロックの置き換えによるコピーバックやbroadcast writeがある。これらのデータ転送の際にもread broadcastを行うプロトコルも提案されている[11]。

read broadcastプロトコルのプロトコル表を表2に示す。read shared(CC,BC?)、read modified(CC, IM,BC?)、copyback(BC?)の要求対象となっているアドレス・タグを持つキャッシュ・ブロックがI状態のときの左段がread broadcastタイプのプロトコルの応答を表している。そのキャッシュはデータを取り込みS状態に遷移する。右段は通常のプロトコルの動作である。

WI MOESIプロトコルをread broadcast化すると5状態のread broadcastタイプのプロトコルが得られる(図2中のRB WI MOESIプロトコル)。このプロトコルのE状態とS状態の間の遷移を短絡除去

	Flush(Copyback)	CC,BC?		CC,IM,BC		BC?	
M	I,BC?,W	O,CS,DI,DK S,CS,DI		-		M,DI,DK,CS?	
O	I,BC?,W	O,CS,DI,DK S,CS,DI		S,SL,CS I		CS:O/M,DI,DK	
E	I	S,CS,DI		-		S,CS	
S	I	S,CS,DI		S,SL,CS I		S,CS?	
I	-	S,SL,CS	I	S,SL,CS	I	S,SL,CS	I

表2

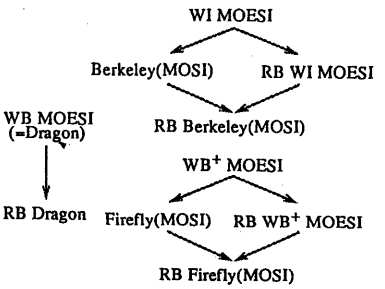

図2

*1[10]よれば、invalidationによるミスが減少しても、プロセッサ・ロックアウトのため処理効率は必ずしも上がらないと報告されている。

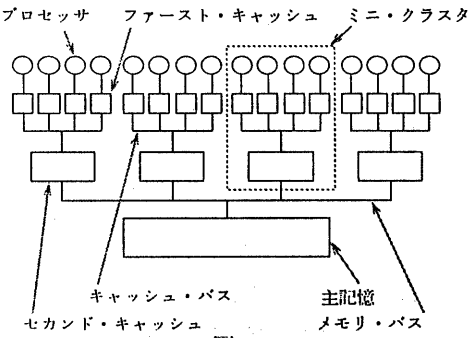

図3

すると、read broadcast化されたBerkeleyプロトコルが得られる(図2中のRB Berkeleyプロトコル)。

Dragonプロトコルや Fireflyプロトコルのような write broadcastタイプのプロトコルも read broadcast化可能である。

2.2.2.階層化への拡張

図3は並列階層キャッシュ・メモリの概略図である。MOESIモデルはキャッシュ・プロトコルを系統的に理解する際の助けとなるが、階層性(ツリー構造)を意識したモデルではない。そのためキャッシュ・ブロックの状態をMOESIモデルが与えている状態だけでは表現できないことがある。そこで、並列階層キャッシュの木構造を反映する特徴を導入する。具体的には、ownershipとexclusivenessに対して、それがクラスタ(そのキャッシュをルートとするサブ・ツリー)全体としての性質か、あるいは、そのクラスタ内の一つのキャッシュとしての性質かによって異なる2つの特徴を定義する*2。それらの属性について説明する。

- c-ownership
  そのキャッシュ・ブロックがそのクラスタから追い出されるときにクラスタ外に書き戻す必要のあるデータであるか(c-owned)いなか(c-unowned)。
- c-exclusiveness
  そのキャッシュ・ブロックのデータが他ミニ・クラスタ内のキャッシュ・ブロックと共有されているか(c-shared)いなか(c-exclusive)。
- i-ownership
  そのキャッシュ・ブロックがそのキャッシュ・メモリから追い出されるときに次のレベルのキャッシュ・メモリあるいは主記憶に書き戻す必要のあるデータであるか(i-owned)いなか(i-unowned)。
- i-exclusiveness
  そのキャッシュ・ブロックのデータがそのクラスタ内の他キャッシュ・メモリと共有されているか(i-shared)否か(i-exclusive)。

ここでは、データの更新の結果、最新のデータが自分をルートとするクラスタ内の自分以外のキャッシュにあって、そこから追い出される際には自分に書き戻す必要がある場合、最新データを持つキャッシュ内のそのキャッシュ・ブロックはi-ownedであるが、自分の中の対応するキャッシュ・ブロックはi-unownedであると定義する。

c-ownershipとc-exclusivenessはクラスタとしての属性なので、そのクラスタ内のすべてのキャッシュ・メモリがこの属性に関する情報を格納する必要はない。クラスタのルートのキャッシュで格納するのが自然と考えられる。

この拡張されたMOESIモデルでは、キャッシュ・ブロックの状態数は24+1=17と複雑なものになる。MOESIモデルの場合と同様に、これらの状態から適当な状態を選択して合理的な状態数のキャッシュ・コンシステンシ・プロトコルを作ることができる。後で説明するPIM/kプロトコルやその改良プロトコルはこの拡張MOESIモデルで記述され、それぞれ4状態、5状態のキャッシュ・プロトコルである。

ファースト・キャッシュからセカンド・キャッシュへコピーバックされたデータがセカンド・キャッシュから追い出されるとき、コピーバックされる必要がないというプロトコルは不自然である。従って、「i-ownedならばc-ownedである。」という制約があると考えてよい。これにより、許される状態数は13となる。

3レベル以上の多階層の並列階層キャッシュ・メモリにも、この拡張MOESIモデルを適用できる。しかも、レベルごと、あるいはクラスタごとに異なるキャッシュ・プロトコルを採用することも可能である。

3.階層キャッシュ・プロトコル

3.1.PIM/kのキャッシュ・プロトコル

並列推論マシンPIM/kに採用されているプロトコルは、Berkeleyプロトコルの拡張であって、次のような特徴を持つ[4]。

- 4状態キャッシュ
  ファースト・キャッシュ、セカンド・キャッシュともに4状態である。
- write-invalidation policy
- snoopy cache protocol
- copy-back cache
  ファースト・キャッシュ、セカンド・キャッシュともにcopyback cacheである。
- multi-level inclusion property
  ファースト・キャッシュ内のデータと同じアドレスのキャッシュ・ブロックがそのミニ・クラスタ内のセカンド・キャッシュに必ず存在する。
- U-bit
  セカンド・キャッシュの各キャッシュ・ブロックのデータのコピーがミニ・クラスタ内

*2validityに対して、i-validity、c-validityを区別しない理由は、multi-level inclusion property(MLI)を仮定しているためである。つまり、c-validならばi-valid。一方、i-validならばc-validは自明。よって、i-validity≡c-validity。MLIが成立たないシステムでは、この区別は意味を持つ可能性がある。

のどのファースト・キャッシュに存在するかを表す。セカンド・キャッシュのキャッシュ・ブロックの置き換えアルゴリズムで使用する。

このPIM/kプロトコルは先にのべた拡張MOESIモデルで記述できる。キャッシュ・ブロックの各状態の意味について説明する。まず、セカンド・キャッシュのキャッシュ・ブロックの状態は次のようになる。

EXC : valid & c-owned & c-exclusive & i-unowned & i-shared

NON : valid & c-owned & c-shared & i-owned & i-shared

UNO : valid & c-unowned & c-shared & i-unowned & i-shared

INV : invalid

一方、ファースト・キャッシュはそれ自身でひとつのクラスタを成していると見なすと、キャッシュ・ブロックの状態は次のようになる。

EXC : valid & c-owned & c-exclusive & i-owned & i-exclusive

NON : valid & c-owned & c-shared & i-owned & i-shared

UNO : valid & c-unowned & c-shared & i-unowned & i-shared

INV : invalid

このように定義すると、ファースト・キャッシュのキャッシュ・ブロックの状態は、c-ownership = i-ownership、c-exclusiveness = i-exclusivenessを満たす。

ファースト・キャッシュからキャッシュ・バスへのバス・トランザクションは次の通りである。

RSH : read shared

RFO : read modified

WFI : invalidation

WWI : copyback

セカンド・キャッシュからキャッシュ・バスへのバス・トランザクションは次の通りである。

FWI : read shared

FAI : read modified

セカンド・キャッシュからメモリ・バスへのバス・トランザクションは次の通りである。

RSH : read shared

RFO : read modified

WFI : invalidation

WWI : copyback

(PIM/kプロトコルの詳細については文献[4]で述べられている。)

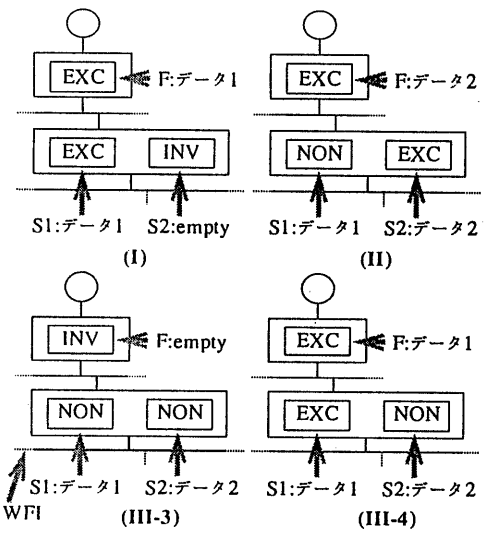

図4

3.2.PIM/kプロトコルの問題点

上に述べたPIM/kプロトコルにおいて、メモリ・バスにinvalidationのトランザクションWFIが生ずるのは次のような状況においてである。

・キャッシュ・バスのRFOがセカンド・キャッシュでNONまたはUNOにヒット

・キャッシュ・バスのWFIがセカンド・キャッシュでNONまたはUNOにヒット

このようなinvalidationがプロセッサ間でデータの共有が無い場合にも発生するという問題がある。キャッシュ間でのデータの共有が無いので、このようなinvalidationは本来不要である。この現象についてより詳細に説明する。

次のような状況を考える。あるファースト・キャッシュ内でデータ1とデータ2は同じキャッシュ・ブロックFにマップされるとする。

(I)プロセッサがあるデータ1をライトする。

(I-1)セカンド・キャッシュのブロックS1がEXCになる。

(I-2)ファースト・キャッシュのブロックFがEXCになる。

(II)プロセッサがあるデータ2をライトする。

(II-1)ファースト・キャッシュのブロックFがコピーバックされる。

(II-2)セカンド・キャッシュのブロックS1がNONになる。

(II-3)セカンド・キャッシュのブロックS2がEXCになる。

(II-4)ファースト・キャッシュのブロックFがEXCになる。

(III)プロセッサがデータ1をライトする。

(III-1)ファースト・キャッシュのブロックFがコピーバックされる。

(III-2)セカンド・キャッシュのブロックS2がNONになる。

(III-3)RFOがセカンド・キャッシュのブロックS1にNONで、ヒットしWFIが出る。

(III-4)ファースト・キャッシュのブロックFがEXCになる。

図4は、このような状況を表す図である。ファースト・キャッシュからセカンド・キャッシュへcopybackされたデータは本来c-exclusiveであるにもかかわらず、c-sharedのNON状態に遷移している。そのため、再びこのデータに対するライトのアクセスが発生した場合には、他のミニ・クラスタに対して無駄なWFIを発することになる。

バス結合のマルチプロセッサはバス・コンテンションが問題となるため通常プロセッサ間のデータ共有があまり大きくならないように使用される。上のような現象はそのような状況でも発生するものである。

セカンド・キャッシュの目的のひとつは、アクセスがファースト・キャッシュでキャッシュ・ミスした場合でもセカンド・キャッシュでのヒットによりアクセス時間を実効的に短くすることである。先のWFIはセカンド・キャッシュにデータが存在するにもかかわらずメモリ・バスにWFIを発してしまい、この目的の妨げとなる。そこで、次節では、この無駄なWFIの発生を防ぐプロトコルを提案する。

3.3.プロトコル改良案

本節では、先に述べたPIM/kプロトコルの問題点を解決するため、拡張MOESIモデルを手掛かりにプロトコルの改良を行う。

改良に際して次の制約条件を満たすプロトコルを検討する。

- メモリ・バスのWFIを減らす効果が明確である。

  ワークの性質によってWFI数が減るかもしれないし、増えるかもしれないというプロトコルではない。

- 制御が複雑になりすぎない。

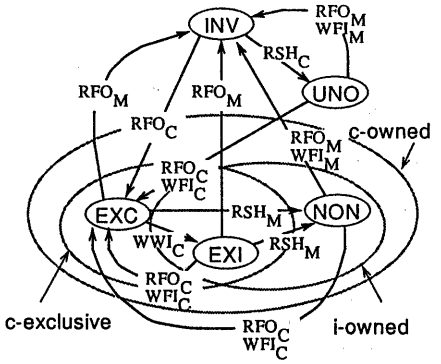

図5

WFI数を少なくすることができても、制御が複雑過ぎて容易に実現できないプロトコルは好ましくない。

- キャッシュ・バスのトランザクションは増やさない。

  メモリ・バスのトランザクショは減らしてもキャッシュ・バスのトランザクションを増やすようでは、性能の向上が必ずしも期待できない。

上の制約が満たされない場合には、あまり良い改良とは呼べないであろう。今回提案する改良案はこれらの制約を満たしている。

先に述べたようなWFIの発生は、PIM/kプロトコルには "c-exclusive & i-owned" という状態がセカンド・キャッシュにはないために、ファースト・キャッシュでキャッシュ・ブロックの置き換えが発生してセカンド・キャッシュへのcopybackが発生すると、セカンド・キャッシュは "c-exclusive & i-owned" の代わりに "c-shared,i-owned" の状態であるNONに遷移するためである。従って、キャッシュブロックの状態として第5の状態を加えることにより、このインバリデーションは防ぐことができる。

	RSH	RFO	WFI	WWI
EXI	data/NON	if(u){ WFIc data/INV}	never	never
EXC	FWIc data/NON	FAIc data/INV	never	never
NON	data	if(u){ WFIc data/INV}	if(u){ WFIc data/INV}	never
UNO	data	if(u){ WFIc data/INV}	if(u){ WFIc data/INV}	never
INV	no	no	no	no

表3

	RSH	RFO	WFI	WWI
EXI	data	data/EXC	/EXC	never
EXC	no	no	no	/EXI
NON	data	WFIm/EXC data	WFIm/EXC	never
UNO	data	WFIm/EXC data	WFIm/EXC	never
INV	RSHm/UNO	RFOm/EXC	never	never

表4

第5状態が満たすべき性質は、"c-owned & c-exclusive & i-owned (& i-shared)" である。この状態は、ミニ・クラスタ内から見ると、NONのように見えて、ミニ・クラスタ外から見ると、EXCのように見える。以下では、この状態のことをEXI(EXclusive Intrinsically)と呼ぶことにする。

表3および表4はEXI状態を加えたプロトコルのセカンド・キャッシュのプロトコル表である。表3はキャッシュ・バスからのコマンドに対する応答、表4はメモリ・バスからのコマンドに対する応答である。ファースト・キャッシュからcopybackされたデータはEXI状態に遷移する。従って、このミニ・クラスタ内で再び、このデータに対するアクセスが生じた場合には他のミニ・クラスタに対してWFIを出すことなくデータをファースト・キャッシュに返すことができる。

図5はセカンド・キャッシュのキャッシュ・ブロックの状態遷移を表す。コマンドの添字C、Mはそれがキャッシュ・バスからのコマンドかメモリ・バスからのコマンドかを表している。

次に、この第5の状態を加えることによりプロトコルが複雑になりすぎないかを検討する。

まず、本改良では、ファースト・キャッシュ側のプロトコルに変更の必要はない。さらにキャッシュ・バス、メモリ・バスともにバス・コマンドの数を増やす必要がない。つまり、改良点はセカンド・キャッシュのプロトコルのみである。しかもU-bit管理に関する本質的変更は無い

以上から本改良は、プロトコルの複雑さをほとんど増すことなく行えると言える。

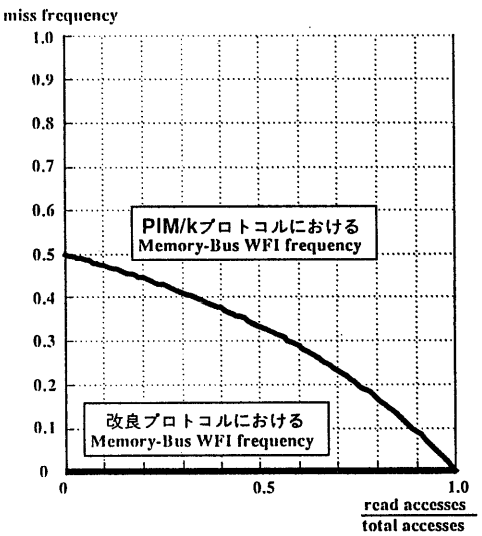

図6

3.4.改良プロトコルの効果

改良プロトコルの効果に関する考察を行なう。ある仮定のもとで、PIM/kプロトコルと改良プロトコルにおいて、どれ程WFIがメモリ・バスに発生するかを検討する。

仮定として、セカンド・キャッシュでは異なる2つのキャッシュ・ブロックに格納され、ファースト・キャッシュでは一つのキャッシュ・ブロックで衝突するような2つのデータを考える。今、プロセッサはこの2つのデータにランダムにリードまたはライトのアクセスを繰り返すとする。さらにこれらのデータは他のミニ・クラスタと共有されていないとする。このとき、メモリ・バスにWFIが発生する頻度を求めた結果が図6である。この頻度はアクセスにおけるリードとライトの比に依存し、ライトの割合が大きい方がWFIの発生率が高くなる。仮にリードの割合を0.5程度にとると、アクセスの内の約1/3はメモリ・バスにWFIを発生させることになる。改良プロトコルでは、リードとライトの比に関係なくWFIは発生しない。

ただし、上の仮定は作為的であって、現実的には、プロセッサのアクセスがすべてファースト・キャッシュで衝突するデータへのアクセスとは限らない。また、ファースト・キャッシュで衝突するデータにランダムにアクセスするという仮定も現実的とはいえない。実際の状況で、これらはWFIの発生率を小さくする方に作用する。一方、ファースト・キャッシュの1つのキャッシュ・ブロックで3つ以上のデータが衝突している場合には、WFIの発生率は大きくなる方に作用される。

システムやプログラムによっては、無駄なWFIの発生による性能低下を十分無視できる可能性も否定できない。改良プロトコルの有効性を示すためには、シミュレーションにより現実的なワークロードを用いた評価を行なう必要がある。

4.まとめ

本稿で述べたことをまとめる。

- Sweazy and SmithのMOESIモデルを並列階層キャッシュ・メモリのプロトコルのモデルに拡張した。これにより、並列階層キャッシュ・プロトコルをより統一的に理解することができる。このモデルはプロトコルの設計・改良の有効な手段となる。
- PIM/kが採用している階層キャッシュ・プロトコルは、プロセッサ間でデータの共有が無い場合でも、メモリ・バスにWFIが発生するという問題点がある。
- この無駄なWFIはセカンド・キャッシュを5状態にする改良により防ぐことができる。

今回提案したプロトコルは、従来のPIM/kプロ

トコルの極めて自然な拡張である。特に、キャッシュ・バスとメモリ・バスのコマンドの種類及び数を変えない。また、ファースト・キャッシュのバス・コマンドに対する応答も全く変えない。セカンド・キャッシュの状態が新たに1つ増え、バス・コマンドへの応答に多少の変更が加わるだけである。しかも、ひとつのシステムに両方のプロトコルのセカンド・キャッシュが共存していてもデータの一貫性は保証される。

システムの階層構造を反映するi-ownership、c-ownership、i-exclusiveness、c-exclusivenessといった概念によりプロトコルの状態に意味づけを行ったが、システムの構造と関係のない概念による意味づけも新たなプロトコルを生む可能性がある。例えば、MOESIモデルは、そのデータが更新されたデータであるかいなかをownershipにより表現している。つまり、"owned"="dirty"である。この制約をはずすことにより新しいプロトコルの可能性があるか否かは検討の余地がある*3。

並列階層キャッシュの評価において、ワークとして用いるプログラムがどのような並列処理のモデルに基づくものであるかということが評価結果に影響を与えると考えられる。我々は、KL1プログラムをワークとして用いた並列階層キャッシュ・プロトコルの評価を検討している。

謝辞

日頃、御指導いただくICOT第1研究室の方々に感謝いたします。

参考文献

[1]浅野滋博,"並列推論マシンPIM/k~ハードウェアアーキテクチャ~",情報処理学会第39回全国大会論文集(III),(1990),1772-1773.

[2]Andrew W. Wilson Jr.,"Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors", Proc. 14th International Symposium on Computer Architecture, (1987),244-252.

[3]R. Katz, S. Eggers, D. A.Wood, C. Perkins and R. G. Sheldon,"Implemeting a Cache Consistency Protocol", Proc. 12th International Symposium on Computer Architecture, (1985),276-283.

[4]浅野滋博,"二階層並列キャッシュコンシステンシプロトコル",情報処理学会計算機アーキテクチャ研究会報告80-3,(1990),17-24.

[5]James Archibold and Jean-Loup Baer,"Cache Coherence Protocols:Evaluation Using a Multiprocessor Simulation Model",ACM Trans. Comput. Syst. 4, 4(1986),273-298.

[6]Per Stenström,"A Survey of Cache Coherence Schemes for Multiprocessors", IEEE Computer 23, 6(1990),12-24.

[7]Rajeev Jog, G. S. Sohi and M. K. Vernon, "Performance Analysis of Hierarchical Cache-Consistent Multiprocessors",Performance Evaluation 9(1989),287-302.

[8]IEEE,Draft Standard:Futurebus+,Logical Layer Specifications Draft 8.2, P896.1R/D8.2(1990).

[9]P.A.Sweazey and A.J.Smith,"A Class of Compatible Cache Consistency Protocols and Support by the IEEE Futurebus", Proc. 13th International Symposium on Computer Architecture, (1986),414-423.

[10]Susan J. Eggers and Randy H. Katz,"Evaluating the Performance of Four Snooping Cache Coherency Protocols", Proc. 16th International Symposium on Computer Architecture,(1989),2-15.

[11]Larry Rudolph and Zary Segall,"Dynamic Decentralized Cache Schemes for MIMD Parallel Processors", CMU-CS-84-139(1984).

*3[9]は複数のオーナーが存在しうるプロトコルの可能性を示唆している。