計算機アーキテクチャ 119－22
（1996. 8. 28）

エラスティックメモリコンシステンシモデルのシミュレーション評価

大津金光　松本尚　平木敬

大規模分散共有メモリシステムでのリモートアクセスによるレイテンシの大きさは性能低下の要因であり、解消すべき問題である。メモリアクセスのレイテンシ隠蔽の手法としてバッファリング、パイプライン化が挙げられるが、これはメモリコンシステンシモデルに課される制限の緩和によって実現される。**Weak Consistency** モデルや **Release Consistency** モデルなどの緩和されたメモリコンシステンシモデルでは、メモリアクセスの順序制御のためにメモリバリアと呼ばれる機構が必要である。当研究室ではこのメモリバリアをエラスティック動作させることで既存のモデルの拡張行なったエラスティックメモリコンシステンシモデルの提案を行なってきた。本稿では **Elastic Memory Consistency** モデルおよびその実装に関して説明を行ない、各メモリコンシステンシモデルにおける性能を測定するための実行駆動型のシミュレータ環境に関して述べる。ベンチマークプログラムで各モデル間の性能比較を行ない、エラスティックメモリコンシステンシモデルの有効性を示す。

An Evaluation on Elastic Memory Consistency Model

KANEMITSU OOTSU , TAKASHI MATSUMOTO and KEI HIRAKI

A great cost of latencies caused by remote accesses on large-scale distributed shared memory parallel computer systems highly degrades the whole performance of the system, and is difficult problem to be solved. Towards this problem, recent studies proposed various relaxed memory consistency models in order to allow buffering and pipelining of memory accesses. In relaxed memory consistency models, such as Weak Consistency model and Release Consistency model, the mechanism called 'memory barrier', which preserves the ordering of memory accesses, should be provided. We have proposed the 'Elastic Memory Consistency Model', which is extended by introducing the elastic activity of 'memory barrier' into the usual models. In this paper, 'Elastic Memory Consistency Model' and an implementation are explained. And performance comparison between the models with simple benchmark shows the effectiveness of 'Elastic Memory Consistency Model'.

1. はじめに

大規模分散共有メモリシステムでは、ノードローカルに閉じないメモリアクセスはレイテンシが大きく、性能低下の要因となる。従って、メモリアクセスのレイテンシを隠蔽するためのメカニズムが必要かつ重要となる。メモリアクセスのレイテンシを隠蔽するために **Sequential Consistency (SC)** モデルよりも制限を緩和したメモリコンシステンシモデルが従来様々に提案・研究されてきた。

図1で従来提案されてきた各種メモリコンシステンシモデルを示す。**Sequential Consistency (SC)** モデル [1] は全てのロード・ストアに関してメモリアクセスの処理が先行するメモリアクセスを追い越してはいけないという最も制限の厳しいモデルであるが、**Processor Consistency (PC)** [2] モデルはストアバッファを持つことでロードの処理がストアの処理を追い越すことが可能となったメモリコンシステンシモデルである。**Weak Consistency (WC)** [3] モデルはPCモデルよりも制限の緩いコンシステンシモデルで、ロード・ストアによるメモリアクセスの発行と完了に関する制限が緩和されている。**WC** モデルにおいて既に発行したメモリアクセスの完了を保証するための手段としてメモリバリア (**MEMBAR**) と呼ばれるメカニズムが提供されており、これを利用してロード・ストア間の先行順序関係を守ることになる。最新のマイクロプロセッサでは **WC** モデルに分類されるメモリコンシステンシモデルが既に実装されている。

WC モデルに分類されるモデルとして **Partial Store Ordering (PSO)** モデルと **Relaxed Memory Ordering (RMO)** モデルがある [4]。**PSO** モデルは **PC** モデルに加えてストアによるメモリアクセスが発行順に処理が完了しなくてもよいという点で制限が緩和されたモデルである。**PSO** モデルにおいては既

† 東京大学 大学院 理学系研究科 情報科学専攻
Department of Information Science, Faculty of Science, the University of Tokyo

に発行したストアの処理が完了したことを保証するためにストアバリア (**STBAR**) が使用される。**Relaxed Memory Ordering (RMO)** モデルは **PSO** モデルの拡張であり、ロードの処理が先行するメモリアクセス処理を追い越してもよいモデルである。**RMO** モデルにおいては全てのロード・ストアが発行順に処理が完了する保証がないため、メモリアクセスの先行関係を保証したい場合はメモリバリア (**MEMBAR**) を使用する必要がある。

Release Consistency (RC) モデル [5] は **WC** モデルを拡張したモデルとしてとらえることができる。**RC** モデルではメモリバリアを **acquire** と **release** という 2 つの機能単位に分割することでさらにメモリアクセスに課される制限を緩和したモデルである。**acquire** はデータセットに対するアクセス権を獲得するための操作 (**lock**) であり、**release** はそのアクセス権を解放する操作 (**unlock**) である。データへのアクセス権という情報を用いることで **WC** モデルではオーバラップすることが不可能だった同期操作 (メモリバリア) を挟んでのメモリアクセスのオーバラップが可能となる。図 1において、同期操作 **release A** と **acquire B** が **acquire A** より先行しない、かつ **release B** よりも先行するという関係を守っている限りは各ロード・ストアによるメモリアクセスはいかなる順序で完了してもよい。

以上の各メモリコンシステンシモデルに加えて、当研究室ではこれまでにメモリバリアをエラスティック動作可能に拡張した **Elastic Memory Barrier** を導入することで得られる新しいタイプのメモリコンシステンシモデル、及びメモリバリアを **Memory-Based Memory Barrier** に拡張することで、さらに制約を緩和したメモリコンシステンシモデル [6] の提案・研究を行なってきた。**WC** モデルに **Elastic Memory Barrier** を導入したモデルは **WC** モデルと **RC** モデルの間を埋めるモデルとして、**Memory-Based Memory Barrier** を導入したモデルは **RC** よりも緩和されたモデルとしてとらえることができる。

以上の各メモリコンシステンシモデルにおいて自由度の高いモデルになるほどシステムの性能向上の可能性は大きいが、反面実装コストおよび、実装の方法によってはオーバヘッドが大きくなる。従って、コスト対効果の面からどのメモリコンシステンシモデルが最も適しているかを見極めるためには各メモリコンシステンシモデルにおける定量的な性能評価を行なうことが必要かつ極めて重要である。本稿では Elastic Memory Consistency モデルと従来型のメモリコンシステンシモデルとの定量的な性能比較を行なうための実行駆動型シミュレーション環境の説明を行ない、ベンチマークプログラムを用いての各モデルでの性能評価の結果を示す。

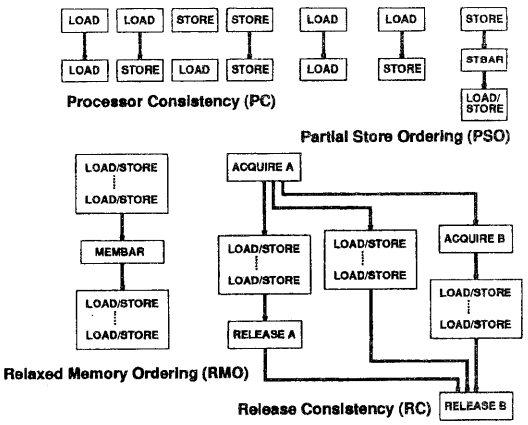

図1 Memory Consistency Models

2. Elastic Memory Consistency モデル

ここで先ず Elastic Memory Consistency モデルについて述べる。簡単のために **PSO** モデルを例にとって説明を行なう。

図 2の **(a)** に **PSO** モデルでのプログラムの一例を示す。これは生産者・消費者プログラムの生産者側のコードであり、データ **A,B,C** (各プロセッサ間で共有) を更新した後、更新したというフラグを **Flag** に設定するという操作をループで回すというものである。このコードにおいて、データ **A,B,C** の消費者側での更新が完了していることを保証するためにフラグをセットする前にストアバリアを張る必要がある。このとき、リモートアクセスを生じるなどによりストアのアクセスコストが大きく、ストアバッファのエントリ数が十分でない場合はストアバリアの完了までプロセッサは処理を先に進めることが出来ないという状況が発生する可能性がある。この場合当然のことながらリモートストアのレイテンシがまともに性能低下に反映されてしまう。

この問題に対してメモリバリアをエラスティックに動作させるという方法が非常に有効である。**Elastic Memory Barrier** は 2 つの操作 **PreREQuest (PREQ)** と **RealREQuest (RREQ)** から構成される。**PREQ** はメモリバリアを張る条件が揃った場合に **RREQ** に先行して使用し、**RREQ** は真にメモリバリア成立を要求する場合に使用する。**PREQ** はノンブロッキング操作であり、同時に共存可能なメモリバリア数を越えない限り、プロセッサのパイプラインは停止することはない。**RREQ** は対応するメモリバリアが既に完了していればプロセッサのパイプラインは停止することなく処理を進めるが、完了していなければそれ以降のロード・ストア命令はストールする。

図 2の **(b)** に **EPSO** モデルを適用して改良を施したプログラム例を示す。図中で **SB_PREQ,**

SB_RREQ がそれぞれストアバリア **STBAR** をエラスティック動作可能にするための 2 つの操作に対応する。1イテレーション分先行してデータ **A,B,C** を更新した上で **SB_PREQ** を発行しておき、次のイテレーションで先の **SB_PREQ** に対応する **SB_RREQ** を発行してから対応するフラグをセットするというプログラムの変更を施している。こうすることでデータ **A,B,C** へのアクセスによって生じるレイテンシを 1 イテレーション分の計算にかかる時間だけ隠すことが可能となる。データアクセスのレイテンシが隠れるのに十分なイテレーション回数分だけメモリアクセスを先行させれば完全にデータアクセスのレイテンシは隠蔽可能である。

以上のようにメモリバリアをエラスティック動作可能に拡張した **Elastic Memory Barrier** を導入することで **PSO** モデルを拡張したものを **Elastic PSO (EPSO)** モデルと呼ぶ。同様に **RMO** モデルを拡張したものを **Elastic RMO (ERMO)** モデルと呼ぶ。

```
L1:                          L0:
    ...                          STORE A[0]
    STORE A[i]                   STORE B[0]
    STORE B[i]                   STORE C[0]
    STORE C[i]                   SB_PREQ 0
    STBAR                    L1:
    STORE Flag[i]                ...
    ...                          STORE A[i]
    LOOP L1                      STORE B[i]
                                 STORE C[i]
                                 SB_PREQ i
                                 SB_RREQ i-1
                                 STORE Flag[i-1]
                                 ...
                                 LOOP L1
```

(a) PSO モデル　　(b) EPSO モデル

図2 Elastic Memory Consistency モデル

図 3に Elastic Memory Barrier の実装法の一例を示す。Elastic Memory Consistency モデルでは複数のメモリバリアの処理が同一時刻に共存しうるため、メモリバリアの張る空間に色を持たせて、重複するメモリバリア空間をその色で区別する。各メモリアクセスには対応するメモリバリア空間の色を付随させることで、そのメモリアクセスがどのメモリバリア空間に属しているかを区別する。

Sync Buffer は各メモリバリアに関する情報を保持しておく FIFO キューである。Sync Buffer の各エントリは対応するメモリバリア空間内に現在何個のメモリアクセスが発行され処理中であるかを計数するカウンタになっている*。このカウンタは対応するメモリバリア空間に属するメモリアクセスがロードストアバッファ内に投入される際にインクリメントされ、メモリアクセスのリクエスト先から返送されてくるアクノレッジ **(ACK)** が返ってきた時にデクリメントされる。(ただしインクリメントが起こるのは Sync Buffer の最後尾のエントリのみである。これは最後尾以外のエントリに対応するメモリバリア空間へのロードストアはもはや発生しないからである) それぞれのエントリに関してカウンタの値が 0 であることは既に対応するメモリバリア空間に属しているメモリアクセスは全て完了したことを意味する。この Sync Buffer のエントリ数が同時に展開可能なメモリバリア空間の最大多重度数となる。**COLOR** はメモリバリア空間の色情報を管理し、メモリアクセスがロードストアバッファに投入される際に参照される。**COLOR** の実装はカウンタで行なえる。Sync Buffer 内の各エントリがそれぞれ固有の色を持てさえすれば良いので $\lceil \log_2$ (Sync Buffer のエントリ数) $\rceil$ ビット幅のカウンタを用意すれば良い。**MB_CNT** は既にメモリバリアの完了した数を計数するカウンタである。

エラスティックメモリバリアの動作を説明すると次のようになる。プロセッサがロードストアを発行、メモリアクセスがロードストアバッファに投入される際に、**COLOR** が保持している現在のメモリバリア空間の色が付加される。それと同時に Sync Buffer 内の最後尾のエントリ (現在のメモリバリア空間に対応する) のカウンタをインクリメントする。ロードストアバッファから取り出され処理されたメモリアクセスリクエストに対する ACK がリクエスト先から返送されて来たときに対応する色のカウンタの値がデクリメントされる。Sync Buffer の先頭のエントリが 0 であった場合には、その色のメモリバリアは完了したと判定され、Sync Buffer から取り除かれると同時に MB_CNT の値をインクリメントする。(ただし、この場合 Sync Buffer 内のエントリ数が 1 個であった場合は除く) 同時に **COLOR** に対して色の使用の解放を通知する必要があるが、前述の通りにカウンタで実装している場合、その必要はない。MB_PREQ 操作が発行された場合、**COLOR** は使用されていない新しい色を用意すると同時に Sync Buffer 内に新しくエントリが投入される。(このエントリの色が現在のメモリバリア空間の色ということになる) この際、新しいエントリはカウンタの値を 0 に初期化された状態で投入される。MB_PREQ 操作を行なう際、Sync Buffer が満杯である場合は Sync Buffer に空きエントリができるまでこの操作はストールさせられる。MB_RREQ 操作が発行された場合、MB_CNT の値が 0 であればそれはまだメモリバリアが完了して

* $\lceil \log_2$ (同時にペンディングにできるメモリアクセスの数) $\rceil$ ビット幅のカウンタであれば良い。

いないことを意味するため、以後のロードストアはストールさせられる。MB_CNTが非0であった場合はプロセッサはストールすることなく処理を進めることができる。MB_RREQ操作時にMB_CNTの値だけでメモリバリアの完了を判定した場合、必ず対応するMB_PREQを先行して発行する必要が出てくるが☆、同時にロードストアバッファが空であるかどうかを判定に加えればMB_RREQを単独で使うことが可能となる。(MB_RREQ操作だけを使用する場合はエラスティック拡張をしてない従来モデルの動作を行なう)

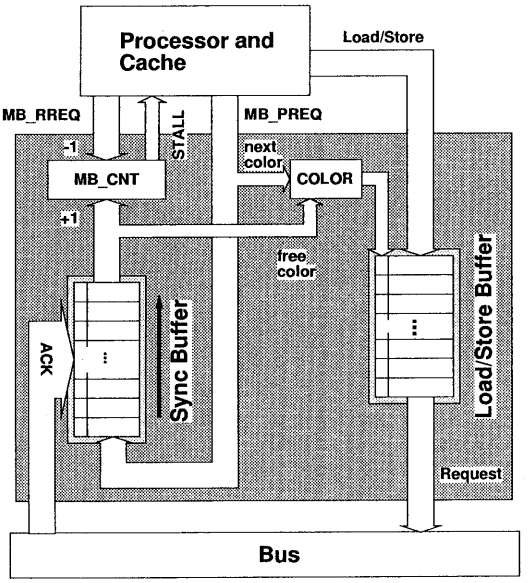

図3 Elastic Memory Barrierの実装例

3. シミュレーション環境

Elastic Memory Consistencyモデルと従来のモデルとの定量的な性能比較を行なうために我々は実行駆動型のシミュレーション環境の構築を行なった。本節ではシミュレーション環境について述べる。

シミュレーションを行なう計算機のシステム構成を図4に示す。システムは複数のプロセッサを1本の共有バスで接続した集中共有メモリマシンである。各構成要素の詳細は以下の通りである。

- プロセッサ
  命令セットはMIPS3 [7]をベースにして一部MIPS4 [8]相当の命令(プリフェッチ命令など)と独自の命令拡張(エラスティックバリア同期命令、エラスティックメモリバリア関連)を行なっている。パイプラインの構成はMIPS R4000/4400と同等のスーパーパイプラインを基本としてパイプラインの動作状態を精密にシミュレートしている。(ただし、浮動小数点演算パイプラインのタイミングは多少異なる)モデルとして用いたプロセッサはブロッキングタイプのメモリアクセスしかサポートしておらず、メモリアクセスが発生すると同時にパイプラインが停止するが、メモリコンシステンシモデルによってはノンブロッキングのロードまたはストアを実現する必要があるためにこの点は現実のものと異なる。ロード・ストアバッファのエントリ数は最小0(この場合メモリアクセスの度にパイプラインが停止する)から自由に設定可能となっている。後述のシミュレーションではエントリ数を8に設定している。
- キャッシュメモリ
  キャッシュメモリは小容量で高速な1次キャッシュと大容量ではあるがアクセス速度が劣る2次キャッシュの2レベル多階層キャッシュを持つ。1次キャッシュは命令・データ分離のダイレクトマップまたは2-wayセットアソシアティブ方式のどちらかが選択可能である。2次キャッシュは命令・データ混在のダイレクトマップ方式または2-wayセットアソシアティブ方式から選択可能である。後述のシミュレーションにおいては1次キャッシュは容量32KBのダイレクトマップ方式、2次キャッシュは容量1MBのダイレクトマップ方式に設定している。キャッシュのブロックサイズは1次、2次キャッシュともに32バイトである。1次キャッシュでヒットする場合には1クロックでアクセスが完了する。1次キャッシュでミス、2次キャッシュでヒットした場合は4クロックでアクセスを完了する。キャッシュの書き込み方式はライトバック方式である。書き込み時にキャッシュミスを発生した場合は該当ブロックのfillを行なった後に書き込みを行なうfetch-on-write方式で処理される。
- メモリシステム
  スヌープキャッシュによりキャッシュのコヒーレンスを維持している。ストア時にインバリデートを行なうかアップデートを行なうかの選択をページ単位で指定することが可能となっており、データセットの性質によって切り替えて使用することで性能向上を図ることができる。
- システムバス
  スプリットトランザクションバスであり各リクエストには必ずリプライが返送される。バススヌーピングにより既にリクエスト発行が完了してリプライ待ちのキャッシュブロックと競合するリクエストはキャンセルされ、後で改めて再発行することになる。データバス幅は64ビットであり、1クロックで8バイトのデータを転送可能である。

☆ これはSync Bufferのエントリ数が1個の場合はカウンタの値が0でもSync Bufferから取り除かれずMB_CNTもインクリメントされないからである。

従ってキャッシュの1ブロック32バイトの転送にバスクロックで4クロックを必要とする。データのブロック（32バイト）転送を伴わないトランザクションは2クロック、ブロック転送を伴うトランザクションは5クロックで完了する。バスの調停はラウンドロビン方式で行なわれる。2次キャッシュおよびメインメモリはバスクロックと同じクロックで動作しており、プロセッサ内部のパイプラインクロックの任意数分の1速度の設定で動作させることができる。後述のシミュレーションではバスクロックはパイプラインクロックの2分の1の速度で動作する。

アプリケーションプログラムはC言語によって記述される。この際、gccの拡張機能を使用することでコンパイラが直接吐かない命令（メモリバリア命令など）の埋め込みやデータの配置のユーザ指定などが可能になっている。Cで記述されたソースコードはgccを利用したクロスコンパイラ、一部使用可能命令の拡張を行なったアセンブラ、リンカを経由してUNIX a.out形式のファイルに変換される。その後、このa.outファイルを一度オリジナルのアセンブルソース形式に逆アセンブル後、オリジナルのアセンブラを用いて最終的にシミュレータに入力可能なオブジェクトファイルが生成される。オリジナルアセンブル形式のソースコードに変換できた段階でプロセッサ台数、プロセッサIDの割り付け、各データセクションのキャッシュアルゴリズム属性などの様々なパラメータの変更が可能である。

4. シミュレーション評価

前節で説明したシミュレーション環境を用いてElastic Memory Consistencyモデルと他のモデルとの定量的な性能比較を行なった。比較を行なったモデルは**SC, PSO, RMO, EPSO, ERMO**の5つである。ベンチマークプログラムとしては図2に示した生産者・消費者型のデータ通信を行なうものを使用した。生産者側プロセッサは1個のダブルワード（64ビット）データのストア完了後に対応するフラグの更新を行ない、消費者側のプロセッサは生産者側プロセッサのストアしたデータのロードを行なう。フラグ部分とデータ部分は別のキャッシュブロックに載るように配置した。キャッシュブロックは32バイトであるため、1キャッシュブロック上でフラグは8個、データは4個格納されることになる。各プロセッサはこれを10000回実行した。プログラムは2台（生産者1台、消費者1台）で実行した。ストア動作のレイテンシ隠蔽のためにループの1イテレーション分先行してストアを発行しておくものと2イテレーション分、3イテレーション分先行して発行しておくものの3つのバージョンに関して性能比較を行なった。データ領域にはインバリデートプロトコルを使用した。

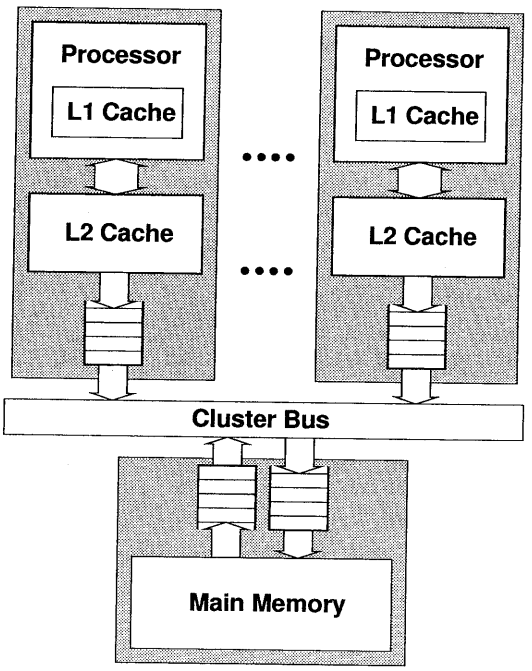

図4　シミュレータの構成

以上の設定でシミュレーションを行なった結果が図5である。括弧内の数字は前倒しに実行したイテレーション回数を示している。図において実行時間は**SC, PSO, RMO, EPSO, ERMO**モデルの順に短くなっていく。**PSO**と**RMO**、または**EPSO**と**ERMO**の差はプログラム実行初期のバスアービトレーションのタイミングの差のみで本質的には等しい時間で実行を完了している。エラスティックモデル以外のモデルでは先行して実行するイテレーション数が増えるにしたがい、実行時間が短くなる。高速化する度合は表1に示される通りバス上で処理されるインバリデートリクエストの数で説明できる。例えば、**PSO**モデルでは**INV (1)**と**INV (2)**ではインバリデートリクエストの数が変化しないため、実行時間に差はないが、**INV (3)**ではインバリデートリクエストの数が半分以下になるために大きく実行時間が減少している。一方、**EPSO**モデルと**ERMO**モデルでは先行イテレーション数を大きくしても高速化はせず、その差は小さくはあるが逆に結果が悪くなっている。この2つのモデルに関してはほとんどインバリデートリクエストが発生しないことから消費者側プロセッサがロードを実行しているキャッシュブロックへのデータストアを既に生産者側プロセッサは全て完了しているものと推測される。そのためバス上で同時に処理中になっているリクエストの数が多くなり、バスの混雑度が増したために、結果的にレイテンシが増大したことが原因である。つまりこのベンチマークプログラムをエラスティックモデルで実行

する場合は1イテレーション分先行して実行することでほぼ高速化の上限に達することを示している。

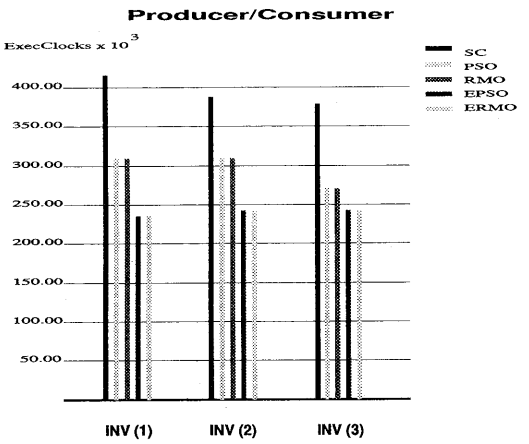

図5　各モデルの実行時間による比較

先行イテレーション数	1	2	3
SC	7505	6254	5787
PSO	2821	2821	1259
RMO	2821	2821	1259
EPSO	8	5	5
ERMO	8	5	5

表1　インバリデートリクエストの数

5. おわりに

メモリバリアを Elastic Memory Barrier に拡張を行なうことで従来のメモリコンシステンシモデルを拡張した Elastic Memory Consistency モデルおよびその一実装例に関して説明を行なった。性能比較を行なうためのシミュレーション環境について述べ、本シミュレーション環境で単純なプログラムを走らせてのベンチマークにより既存の各種メモリコンシステンシモデルとの定量的な性能比較を行なった。その結果、Elastic Memory Consistency モデルの有効性が示された。

参考文献

1) Lamport, L.: How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs, *IEEE Transactions on Computers*, Vol. C-28, No. 9, pp. 241–248 (1979).
2) Goodman, J. R.: Cache Consistency and Sequential Consistency, Technical Report 1006, Compuer Sciences University of Wisconsin, Madison (1991).
3) Dubois, M., Scheurich, C. and Briggs, F.: Memory Access Buffering in Multiprocessors, *International Symposium on Computer Architecture*, pp. 434–442 (1986).
4) Weaver, D. L. and Germond, T.: *The SPARC Architecture Manual Version 9*, Prentice Hall (1994).
5) Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A. and Hennessy, J.: Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, *International Symposium on Computer Architecture*, pp. 15–26 (1990).
6) 松本 尚平木 敬: Elastic Memory Consistency Models, 第49回（平成6年後期）情報処理学会全国大会 講演論文集, Vol. 6, pp. 5–6 (1994).
7) MIPS Computer Systems, Inc.: *MIPS R4000 User's Manual* (1992).
8) MIPS Technologies, Inc.: *MIPS IV Instruction Set* (1995).