Elastic Memory Consistency Models

5K－3

松本 尚　平木 敬
東京大学 大学院理学系研究科 情報科学専攻*

1 はじめに

近年、並列計算機が実用化されるに従って、性能向上の観点から、いくつかの新しいメモリアクセス順序モデル（コンシステンシモデル）が提案された。store bufferを持つことが可能な緩和されたメモリコンシステンシモデルであるProcessor Consistency (PC)[1]モデルに比べて、近年提案されたメモリコンシステンシモデルはプロセッサが発行するメモリアクセスに対する順序制約がさらに緩和され、リモートメモリアクセスレイテンシの隠蔽効果が大きくなっている。本稿では、PCモデル以上の自由度を持つコンシステンシモデルを議論の対象とする。プロセッサ間の同期や通信に関連するメモリアクセスに対しては、いかなるコンシステンシモデルにおいても、単一のプロセッサから発行されたメモリアクセス間に適切な順序関係の導入が必要となる。ハードウェア的にメモリアクセスの単一性順次性が保証されない場合には、acknowledge (Ack)をメモリアクセス要求元へ返送することによって、先行するメモリアクセスの完了を検知する方式が一般的である。そして、このAckを利用して先行するメモリアクセスの完了を管理することによって、先行するメモリアクセスと順序制約のあるメモリアクセスの間の実行順序を調整する。アクセス完了の計数管理が過度に複雑にならないように、代表的なメモリコンシステンシモデルは、メモリバリアと呼ばれる先行するメモリアクセスがある時点ですべて（場合によってはメモリアクセスの種類別に）完了したことを確認する手段を用いて実装される。緩和されたメモリモデルを使用した場合でも、メモリバリアを張るタイミングにおいてリモートメモリアクセスレイテンシによるコストが健在化する可能性があり、大規模並列計算機においてはこのコストが数百clockにも及ぶことがある。従来型のメモリバリアによるレイテンシコストの健在化を防止するために、メモリバリアをelastic動作可能に拡張することで、新しいタイプのコンシステンシモデルを提案する。また、メモリバリアをMemory-Based Memory Barrier に拡張することで、さらに緩和されたメモリコンシステンシモデルを提案する。

2 生産者消費者同期におけるバブルと解決法

話を単純にするために、load系メモリアクセスがblockingタイプであるPartial Store Order モデル (PSO)[2]と呼ばれるメモリコンシステンシモデルを例に挙げて説明する。PSOではストアバリア (STBAR)と呼ばれるstoreに対するメモリバリアが使用され、STBAR命令以降のstore命令は先行するstore命令がすべて完了するまで、外部からはその実行が観測されない。つまり、STBAR以降のstoreはそれ以前にプロセッサの外部に発行されたstoreトランザクションに対応する数のAckが返るまでプロセッサの外部に発行されない。ただし、STBAR以降のstore命令もnonblocking命令であり、store bufferが溢れない限りstore bufferに格納され、命令パイプラインはプログラムの実行を継続する。また、store発行元プロセッサでのstoreトランザクションの外部への発行順序は命令列中のstore命令の出現順と一致する。

PSOモデルに従ったプロセッサでローカルなデータを使った計算結果をupdate系プロトコルを使用してリモートのキャッシュに計算済みフラグと共に書き込む（生産者消費者同期）ループを想定する。ナイーブに考えるとプログラムは以下のようになる。

```
loop: load r1, LA[i]
      load r2, LB[i]
      add  r3, r1, r2   ; r3 = r1 + r2
      store r3, GC[i]
      STBAR
      store 1, GF[i]    ; flag set
      add  i, i, 4      ; i = i + 4
      cmp  i, imax      ; repeat until(i==imax)
      bnz  loop
```

loadはローカルにヒットすることを仮定しているので、store bufferの量がループ回数よりも小さく、リモートストアのコストが他のコストより大きい場合、このループの実行コストは配列GC[i]のストアコストのループ回数倍程度掛かってしまう。しかし、プログラムをよく見ると複数のループイテレーションのリモートストアがパイプライン的にオーバラップして実行できるはずである。そこで、ループを二重ループにして、STBARの回数を複数（例えば4回）のGC[i]のストアにつき一回に間引いて、まとめてフラグGF[i]のストアを行うことで性能が改善できる。しかし、これをフラグセット直前のメモリアクセスおよびSTBARの命令列で書くと

```
      store r3, GC[i+3]
      STBAR
      store 1, GF[i]
      store 1, GF[i+1]
      store 1, GF[i+2]
      store 1, GF[i+3]
```

となり、リモートストア直後にSTBARと別のリモートストアが行われるので、リモートアクセスレイテンシが隠蔽されずに実行コストに反映する。STBARを間引く割合を高めれば、ループ全体に占めるオーバヘッドの割合はどんどん小さくなるが、結果の代入後のフラグセットが必要以上に遅くなる可能性がある。

前述のプログラムの問題点をストアバリアをelastic動作させることで解決することができる。具体的には、プログラムを以下のように変形する。ただし、ここではイテレーション二回分の計算時間でリモートアクセスレイテンシを隠蔽出来ることが静的に判っているとする。

```
      calculation for 1
      store r3, GC[1]
      SB_PREQ[1]
      calculation for 2
      store r3, GC[2]
      SB_PREQ[2]
      i = 3;
loop: calculation for i
      store r3, GC[i]
      SB_RREQ[i-2]
      store 1, GF[i-2]
      SB_PREQ[i]
      calculation for i+1
      store r3, GC[i+1]
      SB_RREQ[i-1]
      store 1, GF[i-1]
      SB_PREQ[i+1]
      i = i + 2;
      repeat loop until(i<=imax)
      CLEAR_SB_COUNTER
```

ここで、SB_PREQ命令はこの時点でストアバリアを張ることを示すが、nonblocking命令でありプロセッサのパイプライン（ここではstore buffer）は停止しない。SB_RREQ命令を実行した時点で対応するSB_PREQに対するストアバリアが成立していなければ、後続のstoreが外部に発行されない（store bufferがstallする）。また、このプログラムは判り易くするために、故意に2イテレーション分ループをunrollした形で示されている。なお、元のイテレーション4回分の計算時間がリモートストアコストの隠蔽に必要であれば、SB_PREQが4個先行するようにループを変形する。

*Elastic Memory Consistency Models. Takashi MATSUMOTO and Kei HIRAKI, University of Tokyo, tm@is.s.u-tokyo.ac.jp

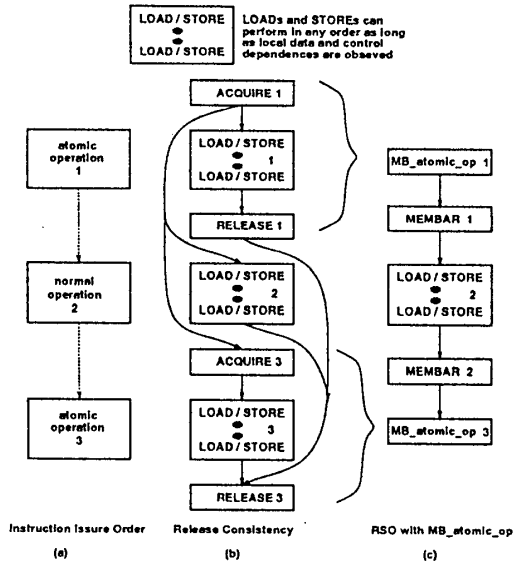

図 1: 不可分処理の実行順序制約

ストアバリアに関しては、各ノードから外部に発行されるstoreトランザクションに対応するAck到着の時間的順序が保証されているわけではない。このため、何番目のSB_PREQまで成立したかを管理するためには、どのストアバリアに対するAckであるか、storeトランザクションの発行元ノードで識別管理可能である必要がある。実装上、この管理はプロセッサとは別のRemote Access Controller (RAC)で行えばよく、プロセッサは単に何回目のストアバリアまで達成されたかをRACからの信号で把握できればよい。このため、プロセッサとRACの間のインタフェースは極めて単純で済む。プロセッサにはSB_COUNTERと呼ばれるカウンタが実装され、RACからのストアバリア成立信号でインクリメントされ、SB_RREQの実行でデクリメントされる。SB_RREQの実行時にカウンタの値がゼロであるとstore bufferがそのSB_RREQに後続するstore命令に対してstallする。

このelastic動作可能にしたメモリバリア (PSOモデルではストアバリア) をElastic Memory Barrier (EMB) と呼び、EMBで拡張したPSOメモリコンシステンシモデルをElastic PSO (EPSO) モデルと呼ぶ。

3 メモリベース不可分操作を伴う場合

不可分操作を行う場合、大規模な並列計算機では実際の演算コストよりもデータのリモートアクセスコストの方が大きい。このため、不可分操作の競合が頻発する状況では、不可分操作に必要なデータを不可分操作単位で1ノードに集めて、データを持つノードに実行すべき不可分操作を依頼するメモリベースの不可分操作[3] (MB_atomic_op) を行った方が不可分操作のスループットが高く全体的な効率が良い。本稿では不可分操作はこのメモリベース不可分操作として実現されるものとする。このメモリベース不可分操作がnonblockingのリモートメモリアクセスとして発行可能であり、メモリバリア (MEMBAR) で終了が検知可能な実装になっていると仮定する。なお、本小節では議論をより一般化するために、storeのみではなくloadもnonblockingに多重発行できるRelaxed Memory Order (RMO) モデル[2]を仮定する。

図1(a)のような処理を考えて、Release Consistencyモデル (RC)[4]による実行 (図(b)) と同じ実行順序を保証するためには、図(c)のようにバブルが生じるMEMBARを挿入する必要がある。メモリバリアがelastic動作可能であれば、無関係な複数の不可分操作列を静的に図2のように組み合わせることで、メモリベース不可分操作のレイテンシを隠蔽することができる。図2中では、記号A, B, Cで記述された依存関係をまったく持たない3本のスレッドが一本の命令列に静的に混合されている。MEMBAR_PREQおよびMEMBAR_RREQは前出のelastic動作可能なストアバリア (SB_PREQとSB_RREQ) をnonblocking load命令にも適用できるメモリバリアに置き換えたものである。RMOモデルをElastic Memory Barrierで緩和したメモリモデルをElastic RMO (ERMO) モデルと呼ぶ。

無関係な不可分操作列を静的に複数組み合わせるプログラム変換はかなり難しい処理である。結局、静的なマルチスレッディングによるレイテンシ隠蔽を行っていることになるので、より動的にマルチスレッディングを行う方法を検討する。図1(c)においてMEMBARを発行した時に現在のコンテクストをメモリ上の特定の場所で一時的に寝かせて、他のコンテクストをランキューから取り出して実行することにする。そして、メモリベース不可分操作やリモートのload/storeに対するAckがそのコンテクスト格納場所へのメモリベースシグナル[3]として働くようにする (複数のリモートアクセスを発行している場合はすべてに対するAckが届くまでスレッド再起動の条件が成立したと見做さない)。これにより不可分操作を含むリモートアクセスが終了後はリモートアクセスを起動したスレッドは再びランキューに格納されプロセッサの割当を待つことになる。これはAckの回収をメモリのアドレスIDを使ってスレッドごとに事実上任意個同時に行えることを利用したものであり、コンテクストのキューイング機能を持ったメモリベースのメモリバリア (Memory-Based Memory Barrier: MB_MEMBAR) と考えることができる。なお、メモリベース不可分操作を使わないRCモデルを細粒度スレッドによるマルチスレッディングによる実行モデルであると考えれば、このMB_MEMBARを用いて実装可能である。また、事実上任意個のバリアを同時に張れるため、RCモデルよりもきめ細かい依存グラフに対応することができる。言い換えるとコンパイル時に静的に最低限満たすべき依存関係を調べて、MB_MEMBARを利用したマクロデータフロー実行形態を採用することで、プログラムからより多くの並列度を抽出することができる。この緩和されたメモリモデルをFully Relaxed Memory Order (FRMO) モデルと呼ぶ。

4 おわりに

Elastic Memory Barrierを利用したEPSOモデルとERMOモデルを提案し、Memory-Based Memory Barrierを利用したFRMOモデルを提案した。詳細は省略するが、モデル間の自由度の高さはFRMO, RC, ERMO, EPSO, PCの順で、FRMOの自由度がもっとも高い。なお、STOやRMO以外のメモリバリアベースの既存メモリコンシステンシモデルも、これらの新しいメモリバリアスキームを用いてより制約を緩和する方向に拡張可能である。ただし、これらの拡張が意味のある拡張であるかどうかは別問題であり、実装コストと効果のトレードオフを見極める必要がある。

参考文献

[1] James R. Goodman: Cache Consistency and Sequential Consistency. Technical Report No. 61, SCI Committee (March 1989).

[2] David L. Weaver and Tom Germond: *The SPARC Architecture Manual Version 9*. PTR Prentice Hall, New Jersey (1994).

[3] 松本尚, 平木敬: 超並列計算機上の共有メモリアーキテクチャ. 信技報, CPSY 92-26, pp.47-55 (August 1992).

[4] K. Gharachorloo, et al.: Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. *Proc. 17th Int. Symp. on Computer Architecture*, pp.15-26 (June 1990).

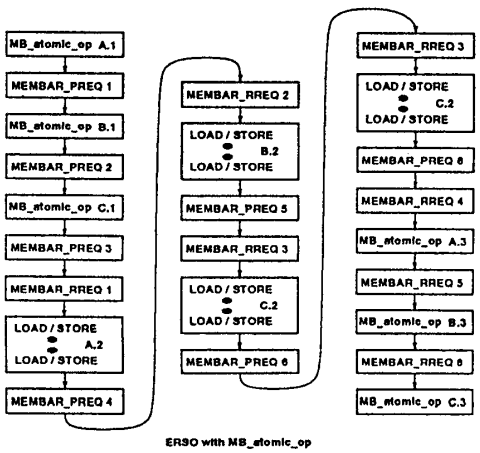

図 2: 複数スレッドの静的混合実行