拡張されたSnoopy Spin Waitと階層化されたElastic Barrier

6B－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

並列処理を効率良く行うためには、並列アクティビティ間の通信同期のオーバヘッドを極力削減する必要がある。筆者らはこの目的のためにSnoopy Spin Wait (SS-Wait) と命名したソフトウェア技巧 [1, 2] と簡易なハードウェア機構を用いるElastic Barrierと命名した同期機構 [3, 4] を従来から提案してきた。本稿ではより多くのマシンや使用ケースに適合するように、一般化かつ拡張されたSnoopy Spin Waitの概念を定義する。また、大規模並列計算機の軽い同期機構として使用可能な階層化されたElastic Barrierを提案する。

2 拡張されたSnoopy Spin Wait

2.1 従来のSnoopy Spin Wait

マルチプロセッサ（特に共有メモリ型マルチプロセッサ）において、オーバヘッドの少ないソフトウェアによる同期方式としてSpin Waitが頻繁に用いられる。しかし、マルチユーザ／マルチジョブといったプリエンプションが行われる環境でSpin Wait同期を使用した場合に、実プロセッサの再スケジューリングが行われないかぎり同期が成立しないという状況になるケースがある。このケースではSpin Waitで同期を待つプロセッサは無駄に実プロセッサ資源を浪費することになる。一つ具体例を挙げると、排他制御が行われている資源の鍵（ロック）を取ったスレッド（またはプロセス）が運悪くプリエンプトされた場合、この時点でロックを取ろうとするスレッドは無駄にSpinループを回ることになる。この種のプロセッサ資源の浪費を防止する新しいSpin Wait方式としてSnoopy Spin Wait (SS-Wait) は提案された。SS-Waitでは実プロセッサ資源のスレッドへの割り当てをSpinループ内でチェックして、他のスレッドの割り当て状況のため同期が当分成立しないと判断される場合はそのスレッドは実行権を自ら放棄する。放棄といっても無条件に放棄するのではなく、通常は同一ユーザまたは同一ジョブ内の他のスレッド（特定可能ならば該当同期条件を成立させるスレッド）に実行権を委譲する。また、オーバヘッド低減のため、スケジューラはユーザレベルスケジューラとカーネルレベルスケジューラの二階層構成とし、SS-Waitによるコンテクスト切替えはカーネルを呼び出さずにユーザレベルスケジューラで行われる。

2.2 想定状況の拡張

旧論文 [1, 2] では主に共有メモリ型マルチプロセッサ上での共有変数を使ったSpin Waitが議論の中心になっているが、議論を共有メモリ上の共有変数に限定する必要はなく、プロセッサ間の何らかの同期を支援する資源があり、その資源の状態の変化をSpin Waitで待つ場合は、SS-Wait方式をSpin Waitの代わりに使用することができる。つまり、プロセッサ間の任意の同期支援ハードウェアの状態フラグやレジスタも共有メモリの各ロケーション同様にSS-Waitの対象となりうる。また、旧論文ではプロセッサ間の同期に議論が限定されていたが、I/Oやメモリシステムとプロセッサの間の同期にも適用可能である。つまり、高性能化のためユーザによるI/Oの直接制御が行われる状況で、そのI/Oコントローラの状態変化をSpin Waitで検出して同期する場合や、メモリシステムでキャッシュへのプリフェッチを行う状況で、データの到着をSpin Waitで確認する場合等にSS-Waitが適用可能である。後者については次節で解説する。

2.3 メモリ資源の割り当て状況によるSS-Wait

NUMA (Non-Uniform Memory Access) である大規模並列計算機においてキャッシュへのプリフェッチがレイテンシ隠蔽技法として注目されている。しかし、プログラムの性質やマシンのレイテンシの大きさによっては実際のloadの発効の十分前にプリフェッチ命令を発効できず、レイテンシを隠しきれない場合がある。こういったケースではloadのデータ待ちのプロセッサはコンテクストを切替えて他のスレッドを実行した方が有利になる。SS-Waitでこの判断を行うには、キャッシュにデータがすでに到着しているか確認する手段が装備される必要がある。この手段を使って、キャッシュにデータが到着しているか確認してから、データをloadすることになるので、プロセッサの命令セットを変更しない場合は若干オーバヘッドが増大する[1]。キャッシュにデータが到着していない場合は短い距離のブランチを引き起こすload命令を命令セットに追加できれば、この命令を使用することでプリフェッチが事前に終了した場合のオーバヘッドを増大させないで済む。キャッシュにデータが未到着の場合はプロセッサの割り当て状況等も考慮に入れて、プロセッサのコンテクストを切替えるかどうか判断を行う。

2.4 Snoopy Spin Waitの新定義

資源の割り当て状況によって同期地点で実行中のスレッド（またはプロセス）が自らコンテクストを切替えるオプションをもつSpin Wait方式であり、割り当て状況のチェックは同期未成立時に同期成立チェックの間に行われる。なお、コンシステンシキャッシュ等を持たないシステムでは、資源割り当て状況のチェックの時間コストが同期変数（または同期通信やI/Oのハードウェア機構の状態レジスタ）へのアクセス頻度を軽減するので、ホットスポットを緩和する効果も期待できる。

2.5 関連研究

最近、バリア同期の場合のSS-WaitがPPOPP'93[5]で発表されたが、これは以前の論文 [1] で想定したSS-Waitの使用ケースにまさに含まれており、筆者らにとってまったく新奇性がない。なお、user activation thread[6] のアイデアもその論文に提示済みである。また、共有変数を使った排他制御のSS-Waitに関しては、NTTの高橋氏の論文 [7] に同じアイデアを見ることができる。

3 階層化されたElastic Barrier

3.1 Elastic Barrierの概説（新解釈版）

Elastic Barrierはバリア概念の拡張、バリアのelastic動作、一般の先行関係のバリア同期への置き換え、静的並列コード最適化との適合性といった様々な側面をもった機構であり、従来 [3, 4] は説明を解り易くするために実装例や使用例と不可分に構成や動作の解説がなされてきた。ここではElastic Barrierに関して知識のある読者を対象に、実装法や使用例によらない解説を試みる。

Elastic Barrierはハードウェアによるバリア同期機構の実装が容易で、共有変数を利用したソフトウェアに比べて圧倒的に高速であることに着目して考案された軽い同期機構である。プロセッサ単位のバリアであるため、予めグループとして登録されたプロセッサによってバリアが張られる。このため、プロセッサ台数指定のスケジューリングが行われることを前提として、プログラムは対応する本数の命令流（シュレッド: shred）を持つ実行コードに変換されている。この実行コードは、台数固定で実行されるデータパラレルの数値計算コードであったり、基本ブロック内の命令レベル細粒度タスクを最適化しながら固定数のシュレッドに割り付けたコードであったりする。

システムがプログラム実行時に動的な外乱要素[2]が起こらないように構成されていれば、バリア同期や先行関係に基づく同期は完全に除去できる。しかし、キャッシュの利用は高性能化には欠くことができず、マルチユーザ／マルチジョブ環境では通信路の競合はコンパイル時に予測

*Extended Snoopy Spin Wait and Hierarchical Elastic Barrier, Takashi MATSUMOTO and Kei HIRAKI, University of Tokyo, {tm,hiraki}@is.s.u-tokyo.ac.jp

[1] レイテンシが非常に大きい場合はこの方式でも有効性がある。

[2] 静的な解析によって予見できないオーバヘッド、例えば、キャッシュミスや通信路の競合によるオーバヘッド。

できない。さらに、コンパイラによる最適化技術は飛躍的に進歩しているが、静的な解析能力には解析時間的にも技術的にも限度がある。これらの要因による動的なオーバヘッドが生じる環境下で、先行関係を保証してプログラムを正しく実行するためにElastic Barrierが用いられる。

Elastic Barrierではバリア同期をプログラム中の一点におけるグループ内の全プロセッサの待ち合わせとして捉え、シュレッドごとにnon-blocking操作である次のバリア同期を許可する操作とバリア未成立時にはblockingされるバリア同期成立をチェックする操作（RREQ）に分離して扱っている。そして、これらの操作がシュレッド内に埋め込まれ、任意の多対多の組み合わせのプロセッサ間同期を可能にしている。つまり、ハードウェアとして簡易なバリア機構で済ますために、あるバリア同期に参加する必要がないプロセッサがグループ内に存在する場合、対応するダミーのバリア同期許可操作がそのプロセッサの実行するシュレッドにも挿入されている。

バリア操作の挿入位置を最適化する[3]ことで、前述のような外乱によるオーバヘッドを吸収軽減する。また、次回のバリア同期を許可する操作は対応するRREQを必ずシュレッド内に持つPREQと、持たないAPRVの二種類が用意されている。PREQとRREQの対応関係は実行時にシュレッド内での出現順に対応が取られる。つまり、三番目のRREQにプロセッサが出会った場合、プロセッサがすでに実行した三番目のPREQに対応するバリアが成立していたら、プロセッサは同期待ちを行うことなく実行を継続する[4]。APRVの使用は、先行関係の先行側やダミーでバリア同期に参加するプロセッサに不要な待ちを生じる可能性を排除し、シュレッド内の同期挿入位置の算出コストを低減する。また、同期操作コストがゼロでない場合は、挿入する同期操作の数の大幅削減によるオーバヘッド削減にも効果がある。

そして、Elastic Barrierの名前の由来にもなっているように、Elastic Barrierのコントローラ内のカウンタ等のハードウェアによってnon-blockingの同期操作を多重に発効することができる。つまり、プロセッサがAPRVやPREQを発効した直後で、まだ対応するバリアが成立していなくても、そのプロセッサは次のAPRVやPREQを発効することができる。先行するバリアがすべて成立次第、コントローラによってプロセッサの動作とは独立に、ペンディングされた同期操作に対応したバリアを張るためのハードウェア操作が行われる。PREQを多重に前もって発効しておくことにより、複数のRREQに対応するバリア成立の履歴をコントローラ内のカウンタに蓄積することができる。RREQは対応する成立の履歴がなくなるまで、blockされることなく実行される。

3.2 階層化された実装方式

従来の論文では比較的小規模の並列システムへの実装法を示していたため、プロセッサのグループを構成する方法に制約を設けず、ビットベクタとしてシステムの全プロセッサ台数に対応するビット幅のグループ選択レジスタをプロセッサごとに持ち、その各ビットに対応するプロセッサ台数本の同期信号線（バリア同期許可でactivateされ、グループのプロセッサの同期線すべてがactivateされると同期成立を検出し、信号線はnagateされる。）をバスとしてプロセッサごとのコントローラが結合する構成を採用していた。しかし、大規模並列システムでは、プロセッサ台数に比例したハードウェア量でも非現実的であり、また単純かつ大規模なバス構成や完全な集中管理構成も実現は難しい。そこで、グループの構成に制約が生じるが、階層構造を取り入れてハードウェア量を削減する必要がある。実際問題として、NUMAまたは分散メモリ構成の大規模並列システムにおいて、システム全域にわたってまばらに複数のプロセッサが選ばれて一つの仕事を行うという事態は、通信コスト的にも非常に不利であるため考えにくく、完全に自由な組み合わせでグループ構成可能であることのメリットはないと考えられる。

階層化構造の機構は基本的にハードウェアによるバリアのcombining treeと多重発効した同期コマンドを制御する同期コントローラから構成される（図1(a)）。図は四進木構成のcombining treeを使用した機構の構成の一例を示す。treeの各階層にはcombiningとプロセッサのグループ分けを実現するために階層コントローラ（図1(b)）が存在し、その内部にグループの組合せを規定する階層グループレジスタが複数組（図では2組）設けられている。ここでは、グループ分けの制約として、各階

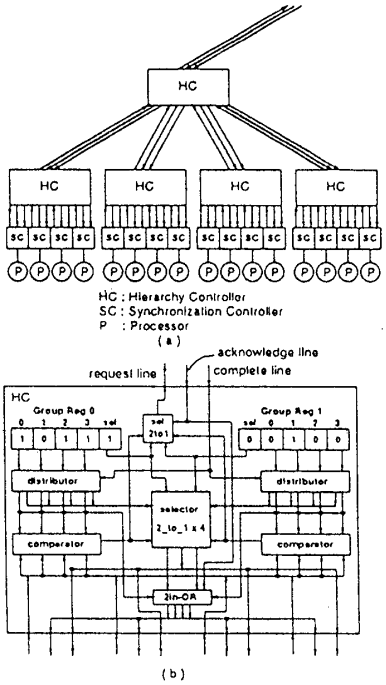

図1: 階層化されたElastic Barrier

層から上層へは多くとも1グループのバリア要求しか伝えられないという制限が存在するものと仮定する。言い替えると、複数のグループのバリア要求が1つの階層コントローラに下層から集まる場合は、多くとも一つのグループを除いてグループ内のバリアのコンバイニングがその階層で終了する（その階層を越えて接続されるプロセッサとはグループを組まない）必要がある。階層間は上層でのバリアを要求するrequest線と次のrequestの受付を許可するacknowledge線と上層から下層へバリアの成立を伝えるcomplete線の3本で接続されている。acknowledgeによってrequest要求のflow制御を行うことで、requestに対応したバリアの成立を示すcompleteが同期コントローラに返る前に、次のバリア要求をcombining treeに対して行うことができる。combining treeのこの能力とElastic Barrierの持つ同期操作の多重発効能力とが組み合わされ、パイプライン的に複数のバリアがtree上で同時に処理できる。このため、バリアを張るレイテンシはcombining treeの段数に応じて増大するが、バリアを多重発効する場合のスループットは減少せず、多重発効によってレイテンシの増大を隠せる可能性がある。このことから、大規模並列システムでは同期操作を多重発効できるElastic Barrierは多重発効できないFuzzy Barrier[8]と比較すると、性能的にかなり有利であることが判る。

参考文献

[1] 松本 尚: マルチプロセッサ上の同期機構とプロセッサスケジューリングに関する考察. 計算機アーキテクチャ研究会報告 No.79-1, 情報処理学会, pp.1-8 (Nov. 1989).

[2] 松本, 根岸, 渦原, 森山: 粒度に基づいた並列計算の分類法とマルチプロセッサの資源管理法について. 日本ソフトウェア科学会第7大会論文集, pp.133-136 (Oct. 1990).

[3] 松本 尚: 細粒度並列実行支援機構, 計算機アーキテクチャ研究会報告 No.77-12, 情報処理学会, pp.91-98 (Jul. 1989).

[4] 松本 尚: Elastic Barrier: 一般化されたバリア型同期機構. 情報処理学会論文誌 Vol.32 No.7, pp.886-896 (1991).

[5] L.Contothanassis and R.W.Wisniewski: Using Scheduler Information to Achieve Optimal Barrier Synchronization Performance. *Proc. 4th ACM PPOPP*, pp.64-72 (May 1993).

[6] T.E.Anderson, et al.: Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism. *ACM Trans. of Computer Systems*, Vol.10, No.1, pp.53-79 (1992).

[7] 高橋 直久: メモリ共有型マルチプロセッサにおける小粒度プロセスの相互排除機構. 情処第39回全国大会論文集, *4P-2*, pp.1213-1214 (Oct. 1989).

[8] R.Gupta: The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors. *Proc. 3rd ACM ASPLOS*, pp.54-63 (Apr. 1989).

[3]静的見積り時にRREQが同期待ちなしに成立する外乱の大きさが極大になるようにバリア操作の挿入位置を調整する。

[4]簡単のため、RREQは必ず対応するPREQを持つことをここでは仮定している。