(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4276028号
(P4276028)

(45)発行日　平成21年6月10日(2009.6.10)　(24)登録日　平成21年3月13日(2009.3.13)

(51)Int.Cl.　F I
G06F 9/52 (2006.01)　G06F 9/46 475A
G06F 12/08 (2006.01)　G06F 12/08 531B
G06F 12/08 551J
G06F 12/08 559Z
G06F 12/08 565

請求項の数 4　(全 13 頁)

(21)出願番号　特願2003-300510 (P2003-300510)
(22)出願日　平成15年8月25日 (2003.8.25)
(65)公開番号　特開2005-71109 (P2005-71109A)
(43)公開日　平成17年3月17日 (2005.3.17)
審査請求日　平成18年3月22日 (2006.3.22)

(73)特許権者　000005108
株式会社日立製作所
東京都千代田区丸の内一丁目６番６号
(74)代理人　100075513
弁理士　後藤　政喜
(74)代理人　100084537
弁理士　松田　嘉夫
(74)代理人　100114236
弁理士　藤井　正弘
(72)発明者　中村　友洋
東京都国分寺市東恋ヶ窪一丁目２８０番地
株式会社日立製作所　中央研究所内
(72)発明者　助川　直伸
東京都国分寺市東恋ヶ窪一丁目２８０番地
株式会社日立製作所　中央研究所内

最終頁に続く

(54)【発明の名称】マルチプロセッサシステムの同期方法

(57)【特許請求の範囲】

【請求項１】

複数のプロセッサでプログラムを並列的に実行するためのバリア同期を行うマルチプロセッサシステムの同期方法であって、

前記マルチプロセッサシステムは、各プロセッサが共有可能な共有メモリ領域を設け、

この共有メモリ領域内に各プロセッサ毎に個別のメモリ領域を割り当て、

このメモリ領域には各プロセッサがプログラム中のバリア同期をとる部分の実行回数に応じた値を格納するカウンタを設け、

各プロセッサは、プログラム中のバリア同期をとる部分を実行したときに、実行回数に応じた値を定める数列から実行回数に対応する値を求め、この値を自プロセッサに割り当てられた前記個別のメモリ領域のカウンタに書き込み、

他のプロセッサに割り当てられたメモリ領域すべてに同一の値または前記数列において該実行回数に１を加えた回数に応じた値が書き込まれたことを判定した後に、後のプログラムの実行を行い、

前記プロセッサはキャッシュメモリを有し、前記各プロセッサに個別に割り当てるメモリ領域のメモリ上のアドレスの間隔を、該プロセッサに搭載されるキャッシュメモリ間のコヒーレンスを保つための管理における処理単位以上に大きくすることを特徴とするマルチプロセッサシステムの同期方法。

【請求項２】

前記プログラム中のバリア同期をとる部分の実行回数に応じて一意の値を定める数列は

、一定の実行回数毎に同一の数列を繰り返すことを特徴とする請求項１に記載のマルチプロセッサシステムの同期方法。
【請求項３】
前記数列は、少なくとも現在のバリア同期が実行中であることを示す値と、現在のバリア同期が終了したことを示す値と、次のバリア同期が終了したことを示す値の少なくとも３つの値を備えることを特徴とする請求項２に記載のマルチプロセッサシステムの同期方法。
【請求項４】
前記プロセッサは、他のプロセッサとのキャッシュメモリ間のコヒーレンスを保つ処理を実行することで前記キャッシュメモリ内の前記メモリ領域を更新することを特徴とする請求項１に記載のマルチプロセッサシステムの同期方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、マルチプロセッサ・システムに関し、特に、共有メモリ型マルチプロセッサ・システムにおけるバリア同期処理に関し、専用ハードウェアを必要とせずに簡易なプログラムコードでバリア同期を実現するコンピュータの制御方法に関する。
【背景技術】
【０００２】
各プロセッサがメモリを共有する共有メモリ型マルチプロセッサ・システムにおいて、並列プログラムを実行する際に各プロセッサ間のバリア同期をとる必要がある。
【０００３】
従来は、このバリア同期を行う場合には、コヒーレンスを保つためにロック処理と呼ばれる処理を行ってきた。
【０００４】
この処理は、データに対する排他制御を行う処理で、複数のプロセッサがロック変数と呼ばれる変数に対して排他的に読み書きできるようにし、このロック変数の状態によってバリア同期の成立・不成立を判断する手法を取っていた。しかし、この排他的な読み書きを実現するためには、プロセッサに用意されているテストアンドセット命令のような処理時間のかかる命令を実行する必要がある。さらに、排他的な処理であるため、バリア同期を行うプロセッサの数が増えるに従って、処理時間も大きく増大してしまう欠点がある。
【０００５】
なお、ロック変数を用いたバリア同期の方法については、「コンピュータの構成と設計」（パターソン＆ヘネシー　著／日経ＢＰ社　1996年4月　刊）の第５５９頁に記載されている。
【０００６】
また、各プロセッサにローカルメモリを備えるとともに、各プロセッサで共有する共有メモリにカウンタを格納し、このカウンタにより各プロセッサの同期を取るものが知られている（特許文献１）。これは、一つのプロセッサが親となり、他のプロセッサが子となって、親と子の間で共有メモリ上のカウンタの値に基づいて同期を取るものである。
【特許文献１】特開平９－３０５５４６号
【発明の開示】
【発明が解決しようとする課題】
【０００７】
上記前者の従来例のロック処理による排他制御によってマルチプロセッサ間でバリア同期処理を行うことは時間がかかる処理である。
【０００８】
特に頻繁にバリア同期をとる必要がある場合には、ロック処理の影響により並列処理の効率が大幅に低下し、場合によっては逐次処理から高速化しない場合がある。
【０００９】

この原因としては、ロック処理が常にメインメモリ上でのデータ読み出し・書き込み処理となり、１回のテストアンドセット命令の実行に多大な時間がかかる、という問題がある。

【００１０】

さらに、ロック処理に伴うメインメモリ上でのデータ読み出し・書き込みが、１つの同一のアドレスに対して行われることによりメモリアクセス性能の低下を引き起こすことがある、という問題があった。

【００１１】

また、上記後者の従来例では、一つのプロセッサが親となり、他のプロセッサを子として親子間の同期を保証することはできるものの、共有メモリのカウンタは親子間の同期のみに利用されるため、子のプロセッサ間では同期を保証することができず、並列処理などに適用した場合では高速化が行えない場合がある、という問題があった。

【００１２】

そこで本発明は、上記問題点に鑑みてなされたもので、ロック処理のような時間のかかる処理を行うことなく、また、特別なハードウェア機構の追加をせずに高速にマルチプロセッサ間でのバリア同期を行うことを目的とする。

【課題を解決するための手段】

【００１３】

本発明は、共有メモリ上にバリア同期をとる各プロセッサの実行が完了した同期ポイントを示す同期フラグ領域（カウンタ）を割り当て、ソフトウェアによりこの同期フラグ領域を実行状態に応じて更新し、各プロセッサはバリア同期に参加する他のプロセッサの同期フラグ領域同士を比較することでバリア同期処理を行う。

【００１４】

同期フラグは、各プロセッサが同期ポイントに到達すると更新されるフラグで、各プロセッサは他のすべてのプロセッサの同期フラグが自分の同期フラグと一致もしくは１つ先の同期ポイントに到達した際の同期フラグの状態と一致するか否かでバリア同期成立を判断する。

【００１５】

さらに、共有メモリにプロセッサに搭載したキャッシュメモリを用い、同期フラグはキャッシュコヒーレンス管理単位であるキャッシュライン毎に１プロセッサ分を割り当てることで、同期フラグ更新のたびに各プロセッサのキャッシュがフラッシュされることなく、各キャッシュラインは同期１回あたり１度ずつフラッシュされるだけでよいため、バリア同期処理の高速化を実現できる。

【発明の効果】

【００１６】

したがって、本発明は、マルチプロセッサシステムにおける並列処理性能を安価に向上させるために、マルチプロセッサ間での同期処理に対して、次の２つの効果が得られる。

【００１７】

まず第１は、特別なハードウェアを用意せずにソフトウェアで様々な規模のバリア同期処理の実現ができる。

【００１８】

次に、第２は、キャッシュ構成を意識した最適化により、バリア同期処理の高速化を実現できる。

【００１９】

上記において、様々な規模のバリア同期処理の実現とは、例えば、バリア同期に参加するプロセッサの数が非常に大きくなっても、同期フラグ領域はメインメモリ上に確保しているので、必要なだけの領域を確保でき、さらに具体的なバリア同期処理においても、ソフトウェアによる制御で行えば、必要な回数だけ他のプロセッサとの同期フラグのチェックを増加させることで対応ができる。

【００２０】

また上記のキャッシュ構成を意識した最適化とは、例えば、別キャッシュライン方式であり、キャッシュやコヒーレンス制御単位などのシステム構成に応じてソフトウェアを調整することによりバリア同期処理の高速化が可能である。

【発明を実施するための最良の形態】

【００２１】

以下、本発明の一実施形態を添付図面に基づいて説明する。

【００２２】

図１は、本発明を適用する共有メモリ型マルチプロセッサシステムを示す。

【００２３】

図１において、複数のプロセッサ１（図中ＣＰＵコア）にはそれぞれキャッシュメモリ２が搭載され、各プロセッサ１とメモリ４（メインメモリ）を接続する共有バス３、複数のプロセッサ１で共有されるメインメモリ４から構成される。なお、プロセッサ１はＣＰＵ＃１～＃ＮのＮ個で構成した場合を示す。

【００２４】

本実施形態ではメインメモリ４の一部の領域を、バリア同期の際に使用する同期フラグ領域５とする。同期フラグ領域５はメインメモリ４内の任意の場所でよく、特別な領域を用意する必要はない。また、バリア同期をとる部分（バリアポイントまたはタイミング）は、マルチプロセッサシステムで実行されるプログラム中に予め設定されたものである。

【００２５】

同期フラグ領域５には、各プロセッサ（ＣＰＵ＃１～＃Ｎ）毎に専用のカウンタとしての同期フラグ（ＦＬＡＧ＃１～＃Ｎ）を用意し、各プロセッサ１がバリア同期を行う毎に、同期フラグＦＬＡＧ＃１～＃Ｎは加算など所定の演算操作によりそれぞれ更新される。

【００２６】

同期フラグ領域５は通常のメインメモリ４上の一部であるので、場合によってＣＰＵ１のキャッシュメモリ２にキャッシングされている。その場合にはキャッシュメモリ２のコヒーレンスを保つための機構が必要である。

【００２７】

本発明によるバリア同期処理の処理フローチャートを図２に示す。

【００２８】

各プロセッサ１は、プログラムコード中のバリア同期ポイントに到達すると現在のそのバリアポイントに応じた値およびその次のバリアポイントの値を計算し（図中６）、現在のそのバリアポイントに応じた値を、自プロセッサ（例えば、ＣＰＵ＃１）に割り当てられた同期フラグ（例えば、ＦＬＡＧ＃１）へ保存する（７）。各プロセッサ１が同一のバリアポイントに到達した際に計算されるバリアポイントに応じた値は同一である。

【００２９】

次に、他の１つのプロセッサ（例えばＣＰＵ＃２）の同期フラグ（ＦＬＡＧ＃２）を読み出し（８）、その同期フラグＦＬＡＧ＃２の値と自ＣＰＵが保存した同期フラグＦＬＡＧ＃１の値と比較をする。この２つの値が同一であった場合、その２つのＣＰＵ＃１、＃２は同一のバリアポイントに到達したと判断できるため、この２つのＣＰＵ間でのバリア同期が成立したと言える。また、他のＣＰＵ＃２の同期フラグ＃２の値が、自ＣＰＵ＃１が次のバリアポイントに到達した場合に同期フラグに保存する値と同一である場合もバリア同期が成立したと判断する（９）。この理由については後ほど説明する。

【００３０】

このいずれでもない場合（１２）には、再び他の１つのＣＰＵの同期フラグを読み出し（６）同様の比較を条件が成立するまで繰り返す。

【００３１】

上記（９）の条件が成立したら、バリア同期に参加するすべてのＣＰＵ＃１～＃Ｎの条件が成立したかをチェックする（１０）。まだチェックの済んでいないＣＰＵが有る場合（１４）には、同期フラグを読み出すＣＰＵを次のＣＰＵ（例えば、ＣＰＵ＃３）にする（１３）。これをすべてのＣＰＵの条件が成立するまで繰り返す。そしてすべてのＣＰＵ

＃２～＃Ｎの条件が成立すればバリア同期成立（１１）と判定する。そして、次の処理を開始する。

【００３２】

バリア同期が成立していても２つのＣＰＵの同期フラグの値が同一でない場合がある。図３に示す例は、２つのＣＰＵの同期フラグの値が同一である場合のみバリア同期が成立と判断する方式ではデッドロックの発生する場合があることを示したものである。

【００３３】

図３は、縦方向に時間をとり、プロセッサ１の内、ＣＰＵ＃１とＣＰＵ＃２の２ＣＰＵがバリア同期[１]とバリア同期[２]を取る場合の、時間軸方向での処理の進行状況と同期フラグの状態を示した模式図である。

【００３４】

ＣＰＵ＃１は並列実行部[１]（図中１５）の処理を終えると、バリア同期[１]の処理に入り、自ＣＰＵの同期フラグＦＬＡＧ＃１をＡからＢに書きかえる（１６）。そしてバリア同期相手のＣＰＵ＃２が同じ同期フラグＦＬＡＧ＃２の値がＢであるかをチェックする（１７）。

【００３５】

一方、ＣＰＵ＃２も同様に並列実行部[１]（２２）、バリア同期[１]での自ＣＰＵの同期フラグＦＬＡＧ＃２のＡからＢへの書き換え（２３）、そしてバリア同期相手のＣＰＵ＃１との同期フラグＦＬＡＧ＃１の比較（２４）をほぼ同時に行う。

【００３６】

この場合、ＣＰＵ＃１およびＣＰＵ＃２における同期フラグのチェック（１７、２４）で、いずれも同期フラグＦＬＡＧ＃１、＃２がＢであることから、バリア同期[１]が成立する（３１）。

【００３７】

次に、同様に並列実行部[２]がＣＰＵ＃１およびＣＰＵ＃２で実行され（１８、２５）、バリア同期[２]の処理でそれぞれのＣＰＵの同期フラグＦＬＡＧ＃１、＃２がＢからＣに書きかえられた（１９、２６）場合を考える。

【００３８】

このタイミングで、ＣＰＵ＃１が割込み処理（２０）に入った場合、ＣＰＵ＃２ではＣＰＵ＃１の同期フラグを読み出してＣであるため、自ＣＰＵの同期フラグＦＬＡＧ＃２と同一と判断して（２７）バリア同期[２]を成立とする（３２）。よって、ＣＰＵ＃２はバリア同期の次の並列実行部[３]（２８）の実行を行い、次のバリア同期[３]の処理に移る。すると、ＣＰＵ＃２の同期フラグＦＬＡＧ＃２はＣからＤに書きかえられてしまう（２９）。

【００３９】

このタイミング以降で、ＣＰＵ＃１が割込み処理（２０）から復帰し、バリア同期[２]の処理の続きを行うことを考える。この時ＣＰＵ＃１はＣＰＵ＃２の同期フラグを読み出して、自ＣＰＵの同期フラグＦＬＡＧ＃１の値Ｃと同一であるか否かの比較を行うが、すでにバリア同期[２]は成立したものだと判断してしまったＣＰＵ＃２の同期フラグＦＬＡＧ＃２はバリア同期[３]の処理によってＤに書きかえられてしまっているため、ＣＰＵ＃１は自ＣＰＵの同期フラグＦＬＡＧ＃１と同一のＣを読み出すことができない。

【００４０】

この状態は、ＣＰＵ＃１、ＣＰＵ＃２共にお互いがお互いの同期フラグの値の更新を待っていることになるので（２１、３０）、永遠に解決されることがないデッドロック状態（３３）となる。

【００４１】

このようなデッドロック状態（３３）を回避するには、同期フラグＦＬＡＧ＃１、＃２の比較において、他のＣＰＵの同期フラグの値が、自ＣＰＵが次のバリアポイントに到達した場合に同期フラグに保存する値と同一である場合もバリア同期が成立したと判断すればよい。

【００４２】
つまり、ＣＰＵ＃１のバリア同期[２]の処理におけるＣＰＵ＃２の同期フラグＦＬＡＧ＃２との比較（２１）においては、ＣＰＵ＃１の現在の同期フラグＦＬＡＧ＃１の値ＣとＣＰＵ＃２の同期フラグＦＬＡＧ＃２の値が一致する場合だけでなく、ＣＰＵ＃１が次のバリアポイントに到達した場合に同期フラグＦＬＡＧ＃１に保存する値Ｄと一致した場合もバリア同期成立とすれば、デッドロック状態（３３）とならない。
【００４３】
上記図２の処理（９）においては、上記のように、次のバリアポイントに到達した場合に同期フラグに保存する値と一致した場合もバリア同期成立とすることで、上述のようなデッドロックを回避して、マルチプロセッサシステム（または並列計算機）における並列処理を円滑に進めることができる。
【００４４】
こうして、本発明によれば、バリア同期に参加するプロセッサ１の数が非常に大きくなっても、同期フラグ領域５はメインメモリ４上に確保しているので、必要なだけの領域を確保でき、特別なハードウェアを用意することなくソフトウェアで様々な規模のバリア同期処理の実現ができる。
【００４５】
これにより、並列処理を行う際に、オーバーヘッドの大きな部分を占めるプロセッサ間のバリア同期処理を高速化することを可能にし、並列化オーバーヘッドを削減してマルチプロセッサシステムの並列処理を高速化できるのである。加えて、専用ハードウェアを用意することがなく、コスト・柔軟性（並列規模や多重実行への対応）などの点でソフトウェアによる方法に優位性がある。
【００４６】
なお、並列処理（または並列計算機）の性能を決定する大きな要因として、次の２点が挙げられる。
・並列化が容易であること（既存のプログラムの構造を変えなくても並列化できること）。
・並列実行時にスケーラブルな性能（並列度に応じた性能が得られること）。
【００４７】
そして、これを実現するためには、高性能な自動並列化コンパイラが必要で、そこで特に重要となるのは次の２点である。
・並列化に伴うオーバーヘッドの影響を低減すること。
・できるだけ並列実行される部分を増やすこと。
【００４８】
並列実行部分を増やすには、小さいループ（演算部分が少ない）も並列化する必要があるが、並列化オーバーヘッドの影響を押さえるには、並列実行部の演算部分がオーバーヘッドに比べて十分に大きい必要があり、この根本的解決策は、並列化オーバーヘッド自体の低減である。本発明は、この並列化オーバーヘッドの大きな部分を占めるプロセッサ間のバリア同期処理を高速化することで並列化オーバーヘッドの低減を実現することに関する。専用ハードウェアを新設する方法もあるが、コスト・柔軟性（並列規模や多重実行への対応）などの点でソフトウェアによる方法に優位性がある。
【００４９】
図４は、第２の実施形態を示し、前記第１実施形態の同期フラグ領域５をキャッシュメモリ上に移したもので、その他の構成は前記第１実施形態と同様である。
【００５０】
複数のプロセッサ１を構成する各ＣＰＵ＃１～＃Ｎのキャッシュメモリ２には、同期フラグＦＬＡＧ＃１～＃Ｎを保存する同期フラグ領域３４が設定される。なお、同期フラグＦＬＡＧ＃１～＃Ｎを保存するためのキャッシュメモリは、既設のキャッシュメモリ２内に確保しても良いし、別途同期フラグＦＬＡＧ＃１～＃Ｎを保存するためのキャッシュメモリを新設しても良い。

【００５１】
プロセッサ１のキャッシュメモリ２に同期フラグ領域３４を設定することで、プロセッサ１のメモリアクセス・レイテンシを削減し、同期フラグＦＬＡＧ＃１～＃Ｎの読み書きが高速化され、上記図２に示した処理フローの処理時間が短縮される。
【００５２】
この場合、各ＣＰＵ＃１～＃Ｎ毎に設けたキャッシュメモリ２毎に全ＣＰＵの同期フラグ領域をミラーリングして保持してもよいし、自ＣＰＵの同期フラグ領域のみを保持し、他ＣＰＵの同期フラグを読み出す場合には、共有バス３経由で読み出すようにしてもよい。
【００５３】
本発明が上記従来例のロック変数を用いたバリア同期処理と異なるのは、従来例ではロック変数を複数プロセッサで奪い合い、同時には１つのプロセッサのみが処理を可能としているため、複数のプロセッサの処理が逐次的に行われていくのに対し、本発明では、図２に示した処理フローを各プロセッサが並列的に行っていくことが可能である点である。そのため、バリア同期に参加するプロセッサの数が多くなった場合の処理時間の増大が、従来例に比べて少ない利点がある。
【００５４】
図５は、上記図４に示した同期フラグ領域３４のメモリ上への確保の方法を模式的に２通り示した図である。図５でグレーの領域は同期フラグ領域３４のメモリ空間を示し、その横幅はキャッシュラインサイズである。１つのキャッシュラインは横一列分であるとする。
【００５５】
同一キャッシュライン方式（図中３５）は、１つのキャッシュラインＬｉｎｅ＃１（３７）にＮ個のプロセッサの同期フラグを確保する方式である。図５ではＮ個の同期フラグ（図中フラグ[１]～[Ｎ]）をＬｉｎｅ＃１（３７）内で連続する位置に確保しているが、この確保する順序や場所はＬｉｎｅ＃１内で同期フラグ同士で重ならない限りどのように確保してもよい。
【００５６】
一方、別キャッシュライン方式（図中３６）は、１つのキャッシュラインには１つのプロセッサの同期フラグ（図中フラグ[１]～[Ｎ]）のみを確保する方式で、Ｎ個のプロセッサの同期フラグＦＬＡＧ＃１～＃ＮはＮ個のキャッシュラインＬｉｎｅ＃１～＃Ｎ（３７～３９）に分散して確保する。
【００５７】
それぞれのキャッシュラインにおいて、同期フラグを確保する位置は、図５の例ではキャッシュライン毎にずらしてあるが、すべてキャッシュラインの先頭に配置するなど、それぞれのキャッシュラインの中のいずれかの位置に確保する場合を含む。またＮ個のキャッシュライン３７～３９はメモリ空間上で連続している領域である必要はない。
【００５８】
図６は、プロセッサ１としてＣＰＵ＃１、ＣＰＵ＃２、ＣＰＵ＃３の３つのＣＰＵ間で本発明による方式でバリア同期処理を行った場合の処理順序の例を、縦方向を時間軸として模式的に示した図である。
【００５９】
同一キャッシュライン方式（図中３５）でも別キャッシュライン方式（図中３６）でも３つのＣＰＵは同時にバリア同期処理を開始したとする。
【００６０】
同一キャッシュライン方式（３５）では、まずＣＰＵ＃１がＬｉｎｅ＃１（図５参照）上のＣＰＵ＃１の同期フラグ領域３７を更新する。ＣＰＵ＃２、ＣＰＵ＃３もほぼ同時にＬｉｎｅ＃１上の自ＣＰＵの同期フラグ領域を更新しようとするが、ＣＰＵ＃１が同一のキャッシュラインであるＬｉｎｅ＃１に対するｓｔｏｒｅを行ったため、ＣＰＵ＃２、ＣＰＵ＃３の処理はＣＰＵ間でのキャッシュコヒーレンスを維持するための機構によりキャ

ンセルされる。なお、各ＣＰＵ＃１～＃Ｎは、キャッシュメモリ２のコヒーレンスを維持する機構を備えるものである。

【００６１】

次に、ＣＰＵ＃２がＬｉｎｅ＃１上の自ＣＰＵの同期フラグ領域を更新する。先ほどと同様にしてＣＰＵ＃３もほぼ同時にＬｉｎｅ＃１上の自ＣＰＵの同期フラグ領域を更新しようとするが、再度キャンセルされる。そのため、ＣＰＵ＃２の更新処理が終了してからＣＰＵ＃３はＬｉｎｅ＃１上の自ＣＰＵの同期フラグ領域を更新することになる。ここまでの処理は図２に示した処理フローにおける同期フラグの更新処理（６、７）の部分である。

【００６２】

次に、同期フラグのチェックの処理（８～１４）をＣＰＵ＃１、ＣＰＵ＃２、ＣＰＵ＃３でそれぞれ行う。この例ではＣＰＵ＃３はＬｉｎｅ＃１がすでに自ＣＰＵのキャッシュ上にありＣＰＵ＃１およびＣＰＵ＃２の同期フラグの値とＣＰＵ＃３の同期フラグを比較することでバリア同期成立を判定することができる。

【００６３】

一方、ＣＰＵ＃１とＣＰＵ＃２は、フラグ更新後にそれぞれ自ＣＰＵの同期フラグと他ＣＰＵの同期フラグの値の比較を行うものの、ＣＰＵ＃３がＣＰＵ＃３の同期フラグを更新し、その更新されたＬｉｎｅ＃１が転送（４３）されてくるまでは、同期フラグの比較においてバリア同期成立の条件を満たさないため、バリア同期成立（４４）のタイミングは、ＣＰＵ＃３がバリア同期成立を判定した頃とほぼ同時となる。そのため、同一キャッシュライン方式（３５）は同期フラグの更新が結果的に逐次的にしか行えず、バリア同期処理にかかる時間がバリア同期に参加するプロセッサの数にほぼ比例して増大する。

【００６４】

一方、別キャッシュライン方式（３６）では、各ＣＰＵが更新するそれぞれのＣＰＵの同期フラグ領域が別々のキャッシュラインＬｉｎｅ＃１～＃Ｎとなっているため、ＣＰＵ＃１、ＣＰＵ＃２、ＣＰＵ＃３いずれもほぼ同時に同期フラグの更新が可能である。

【００６５】

次に、同期フラグＦＬＡＧ＃１～＃Ｎのチェックの処理では、他のＣＰＵの同期フラグがあるキャッシュラインが自ＣＰＵのキャッシュラインに転送されてフラッシュした後、このキャッシュラインの同期フラグとの比較を行うことでバリア同期成立の判定を行う。よって、別キャッシュライン方式（３６）は同期フラグの更新が各ＣＰＵ間で並列的に行えるため、バリア同期処理にかかる時間はバリア同期に参加するプロセッサの数が増大しても大きくは増加しない。キャッシュラインの転送回数が増加するが、プロセッサの数の増大に応じて転送ラインの数を増加させることで処理時間を増加させないようにすることが可能である。

【００６６】

図６による説明では、キャッシュラインごとのコヒーレンス制御を前提として説明をしたが、キャッシュラインサイズによらず、コヒーレンス制御単位（キャッシュライン単位）以上のサイズの領域当たり１つの容量で、各プロセッサの同期フラグ領域を確保すれば別キャッシュライン方式と同様の並列的な同期フラグ領域の更新が可能である。つまり、第１の実施形態においては、各プロセッサ１へ個別に割り当てるメモリ領域（同期フラグ領域５）のメモリ上のアドレスの間隔を、プロセッサ１に搭載されるキャッシュメモリ２間でコヒーレンスを保つための管理における処理単位（管理単位）以上に大きくすればよい。

【００６７】

次に、バリアポイントにおいて、同期フラグ領域に保存する値に関しては、各プロセッサが同一のバリアポイントに到達した際に同一の値になれば問題ないが、例えば図７に示す数列５０を用いると、同期フラグの比較処理を高速化することが可能である。

【００６８】

図７に示したバリア同期実行回数を示す数列５０の例では、８ビットの同期フラグを仮

定しバリア同期実行回数をＭとして、２の（Ｍｍｏｄ８）乗までを同期フラグに書き込む値とする。つまり、８ビットの同期フラグにおいて、いずれか１ビットのみが常に１であり、他は０である状態で、バリア同期実行毎に１であるビットが左に１ビットずつシフトしていき、８ビット目の次は１ビット目に戻る操作を行う数列である。換言すれば、バリア同期実行毎に１であるビットを左に１ビットずつローテートする操作である。

【００６９】

このように数列５０が循環してよい理由は、バリア同期処理が正しく行われている場合には、各プロセッサのバリア同期の実行回数に１以上の差が生じることがないためである。つまり、他のプロセッサ１に先行して次のバリアポイントに達したプロセッサ１は、他のプロセッサ１が同じバリアポイントに達するまで後続の命令（プログラム）を実行することができないためである。

【００７０】

図７に示したバリア同期実行回数を示す数列５０の例が、同期フラグの比較処理に適している理由は次の通りである。

【００７１】

上記図２に示したバリア同期処理の処理フローチャートにおいて、他のプロセッサの同期フラグとの比較処理が、値が同一であるか、または、自プロセッサの同期フラグを１ビット左にローテートしたものと同一であるかの確認となり、演算処理が簡単となるため、処理時間が短縮できる。

【００７２】

なお、上記数列５０は、現在のバリア同期処理が実行中であることを示す値と、現在のバリア同期処理が終了したことを示す値と、次のバリア同期処理が終了したことを示す値の少なくとも３つの値を備え、循環して利用可能な数列であれば良く、好ましくは、数値的に連続する数列や、ビット配列の順に連続する数列である。

【００７３】

以上のように、各ＣＰＵ＃１～＃Ｎのキャッシュメモリ２のコヒーレンスを維持する機構を備える場合には、共有メモリにキャッシュメモリ２を用い、同期フラグＦＡＬＧ＃１～＃Ｎはキャッシュコヒーレンスの管理単位であるキャッシュライン毎に１プロセッサ（ＣＰＵ）分を割り当てることで、同期フラグ更新のたびに各プロセッサのキャッシュがフラッシュされることなく、各キャッシュラインは同期１回あたり１度ずつフラッシュされるだけでよいため高速化される。

【００７４】

そして、マルチプロセッサシステムで並列処理を行う場合に、並列化オーバーヘッドの大きな部分を占めるプロセッサ間のバリア同期処理を高速化することで並列化オーバーヘッドの低減を実現することが可能となり、特に、専用ハードウェアを必要とすることなく、コスト・柔軟性（並列規模や多重実行への対応）などの点でソフトウェアによる方法に優位性がある。

【産業上の利用可能性】

【００７５】

以上のように、本発明に係るマルチプロセッサシステムの同期方法は、並列処理を行うマルチプロセッサ、もしくはその並列処理のためのコンパイラに適用することができる。

【図面の簡単な説明】

【００７６】

【図１】本発明の一実施形態を示し、バリア同期処理を行うマルチプロセッサシステムの構成図を示す。

【図２】バリア同期処理の一例を示すフローチャート。

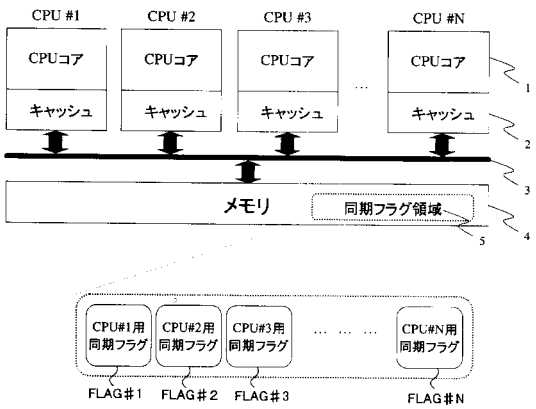
【図３】同じく、２つのＣＰＵによるバリア同期処理の一例を示すタイムチャート。

【図４】第２の実施形態を示し、バリア同期処理を行うマルチプロセッサシステムの構成図を示す。

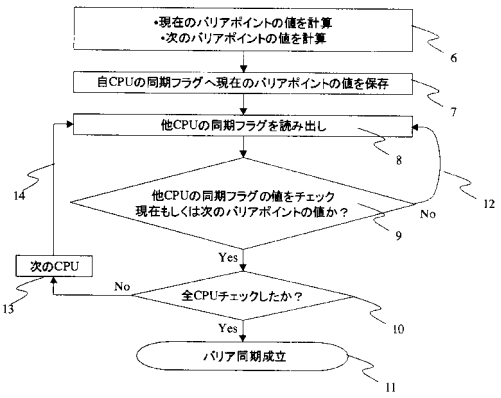
【図５】キャッシュメモリの同期フラグ領域のマッピングを示す説明図で、図中上部が同

一キャッシュライン方式を、図中下部が別キャッシュライン方式を示す。
【図６】３つのＣＰＵによるバリア同期処理のタイムチャートを示し、図中左半分が同一キャッシュライン方式を、図中右半分が別キャッシュライン方式を示す。
【図７】バリア同期処理の際に更新される値の一例を示す説明図。
【符号の説明】
【００７７】
１　プロセッサ
２　キャッシュメモリ
３　共有バス
４　メインメモリ
５、３４　同期フラグ領域

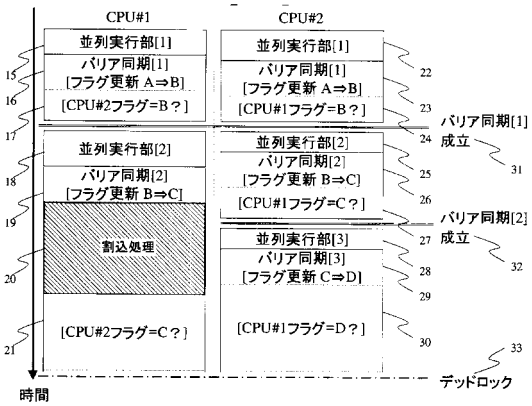
【図１】



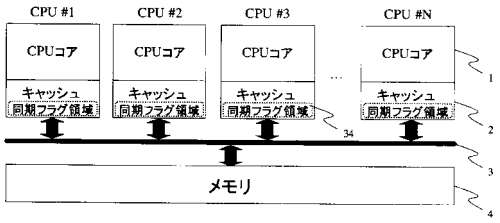
【図２】



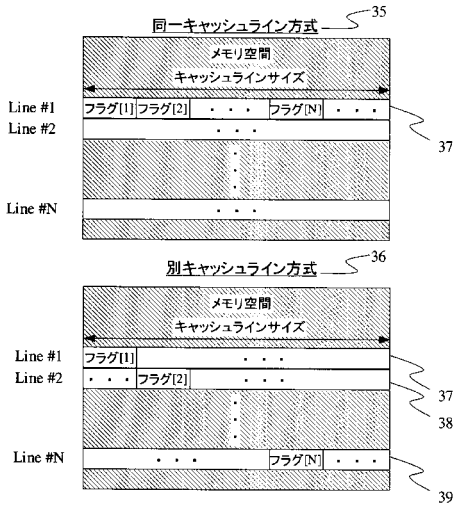
【図３】


CPU#1
CPU#2
並列実行部[1]
バリア同期[1]
[フラグ更新 A⇒B]
[CPU#2フラグ=B？]
[CPU#1フラグ=B？]
バリア同期[1]
成立
並列実行部[2]
バリア同期[2]
[フラグ更新 B⇒C]
[CPU#1フラグ=C？]
バリア同期[2]
成立
割込処理
並列実行部[3]
バリア同期[3]
[フラグ更新 C⇒D]
[CPU#2フラグ=C？]
[CPU#1フラグ=D？]
デッドロック
時間

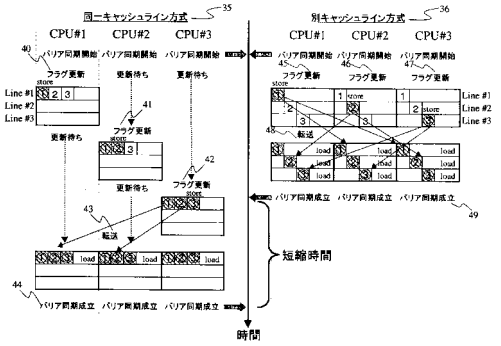
【図４】


CPU #1
CPU #2
CPU #3
CPU #N
CPUコア
キャッシュ
同期フラグ領域
メモリ

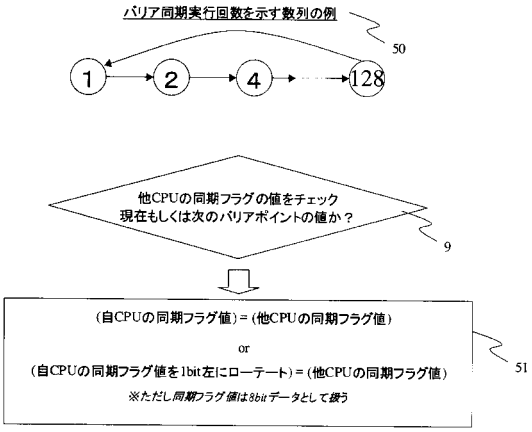
【図５】


同一キャッシュライン方式
メモリ空間
キャッシュラインサイズ
Line #1
Line #2
Line #N
フラグ[1]
フラグ[2]
フラグ[N]
別キャッシュライン方式
メモリ空間
キャッシュラインサイズ
Line #1
Line #2
Line #N
フラグ[1]
フラグ[2]
フラグ[N]

【図６】


同一キャッシュライン方式
別キャッシュライン方式
CPU#1
CPU#2
CPU#3
バリア同期開始
フラグ更新
更新待ち
転送
バリア同期成立
短縮時間
時間

【図７】


バリア同期実行回数を示す数列の例
50
1
2
4
128
他CPUの同期フラグの値をチェック
現在もしくは次のバリアポイントの値か？
9
（自CPUの同期フラグ値）＝（他CPUの同期フラグ値）
or
（自CPUの同期フラグ値を1bit左にローテート）＝（他CPUの同期フラグ値）
※ただし同期フラグ値は8bitデータとして扱う
51

フロントページの続き

審査官　鈴木　修治

(56)参考文献　特開２００１－０４３２０３（ＪＰ，Ａ）
特開２００２－３４２１６３（ＪＰ，Ａ）
特開２００３－０３００４９（ＪＰ，Ａ）
特開平０５－０３５６９７（ＪＰ，Ａ）
特開平０６－２４３１１０（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　９／４６－９／５４
Ｇ０６Ｆ　１２／０８