(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6563485号
(P6563485)
(45)発行日　令和1年8月21日(2019.8.21)　(24)登録日　令和1年8月2日(2019.8.2)

(51)Int.Cl.　　F I
G06F 9/52 (2006.01)　　G06F 9/52 １２０Ｂ
G06F 9/46 (2006.01)　　G06F 9/46 ４１０

請求項の数 27　(全 26 頁)

(21)出願番号　特願2017-512043 (P2017-512043)
(86)(22)出願日　平成27年7月28日 (2015.7.28)
(65)公表番号　特表2017-530455 (P2017-530455A)
(43)公表日　平成29年10月12日 (2017.10.12)
(86)国際出願番号　PCT/GB2015/052177
(87)国際公開番号　WO2016/038328
(87)国際公開日　平成28年3月17日 (2016.3.17)
審査請求日　平成30年7月23日 (2018.7.23)
(31)優先権主張番号　1415834.9
(32)優先日　平成26年9月8日 (2014.9.8)
(33)優先権主張国・地域又は機関
英国 (GB)

(73)特許権者　594154428
エイアールエム　リミテッド
イギリス国　シービー１　９エヌジェイ
ケンブリッジ，チェリー　ヒントン，フルバーン　ロード　１１０
(74)代理人　110000855
特許業務法人浅村特許事務所
(72)発明者　ホルム、ルーネ
イギリス国、ケンブリッジシャー、ケンブリッジ、チェリー　ヒントン、フルボーン　ロード　１１０、エイアールエム　リミテッド　気付

最終頁に続く

(54)【発明の名称】複数のスレッドを実行するデータ処理装置における共有リソース

(57)【特許請求の範囲】
【請求項１】
複数のスレッドを実行するように構成されたデータ処理装置であって、
前記複数のスレッドの少なくとも部分集合について実行される、命令ストリーム内の１つの命令を特定する汎用プログラム・カウンタであって、前記部分集合内の各スレッドが、前記命令ストリーム内の１つの命令を特定するための関連するスレッド・プログラム・カウンタを有する汎用プログラム・カウンタと、
スレッドの前記部分集合の選択されたスレッドを選び出し、且つ、前記汎用プログラム・カウンタを前記選択されたスレッドに関連する前記スレッド・プログラム・カウンタに設定するように構成されたセレクタと、
前記選択されたスレッドを含むスレッドの前記部分集合の１つ又は複数について、前記汎用プログラム・カウンタによって特定された命令を実行するように構成されたプロセッサと
を備え、
スレッドの前記部分集合が、スレッドの前記部分集合のどれが共有リソースへの排他的アクセスを有するかを追跡する、少なくとも１つのロック・パラメータに関連しており、
前記プロセッサが、スレッドがそのスレッドについて実行された第１の命令に応答して前記共有リソースへの排他的アクセスを獲得していることを示すように前記少なくとも１つのロック・パラメータを変更し、且つ、前記スレッドがそのスレッドについて実行された第２の命令に応答して前記共有リソースへの排他的アクセスをもはや有していないこと

を示すように前記スレッドに関連する前記少なくとも１つのロック・パラメータを変更するように構成され、
前記セレクタが、前記少なくとも１つのロック・パラメータに基づいて前記選択されたスレッドを選び出すように構成される、データ処理装置。
【請求項２】
前記プロセッサが、そのスレッドについて実行される少なくとも１つのロック命令を含むロッキング・シーケンスに応答して、前記共有リソースへの排他的アクセスを、スレッドに認めるように構成される、請求項１に記載のデータ処理装置。
【請求項３】
前記ロッキング・シーケンスが、前記第１の命令を含まない、請求項２に記載のデータ処理装置。
【請求項４】
前記プロセッサが、そのスレッドについて実行される少なくとも１つのアンロック命令を含むアンロッキング・シーケンスに応答して、スレッドのための前記共有リソースへの排他的アクセスを放棄するように構成される、請求項１から３のいずれか一項に記載のデータ処理装置。
【請求項５】
前記アンロッキング・シーケンスが、前記第２の命令を含まない、請求項４に記載のデータ処理装置。
【請求項６】
前記セレクタが、前記共有リソースへの排他的アクセスを有さないとして前記少なくとも１つのロック・パラメータによって示された第２のスレッドに優先して、前記共有リソースへの排他的アクセスを有するとして前記少なくとも１つのロック・パラメータによって示された第１のスレッドを、前記選択されたスレッドとして選び出すように構成される、請求項１から５のいずれか一項に記載のデータ処理装置。
【請求項７】
前記セレクタが、スレッドの前記部分集合の各スレッドに関連する関数呼び出し深さパラメータに基づいて前記選択されたスレッドを選び出すように構成される、請求項１から６のいずれか一項に記載のデータ処理装置。
【請求項８】
前記セレクタが、スレッドの前記部分集合の各スレッドに関連する前記スレッド・プログラム・カウンタに基づいて前記選択されたスレッドを選び出すように構成される、請求項１から７のいずれか一項に記載のデータ処理装置。
【請求項９】
前記部分集合の各スレッドが、対応するロック・パラメータを有し、前記プロセッサが、スレッドがそのスレッドについて実行された第１の命令に応答して前記共有リソースへの排他的アクセスを獲得していることを示すように、前記スレッドに対応する前記ロック・パラメータを変更し、且つ、前記スレッドがそのスレッドについて実行された第２の命令に応答して前記共有リソースへの排他的アクセスをもはや有していないことを示すように、前記スレッドに対応する前記ロック・パラメータを変更するように構成される、請求項１から８のいずれか一項に記載のデータ処理装置。
【請求項１０】
前記プロセッサが、第２のスレッドが第２の共有リソースへの排他的アクセスを有している一方で、第１のスレッドに第１の共有リソースへの排他的アクセスを有することを許可するように構成される、請求項９に記載のデータ処理装置。
【請求項１１】
前記セレクタが、スレッドの前記部分集合の各スレッドの前記対応するロック・パラメータに基づいてスレッドの第１の集合を選択するように構成され、
前記セレクタが、関数呼び出し深さパラメータに基づいてスレッドの第２の集合を、スレッドの前記第１の集合の中から、選択するように構成され

前記セレクタが、スレッドの前記第２の集合の前記スレッドに関連する前記スレッド・プログラム・カウンタに基づいて、スレッドの前記第２の集合から、前記選択されたスレッドを選び出すように構成される、請求項９及び１０のいずれか一項に記載のデータ処理装置。

【請求項１２】

前記少なくとも１つのロック・パラメータが、スレッドの前記部分集合の間で共有される共有ロック・パラメータと、スレッドの前記部分集合のいずれが共有リソースへの排他的アクセスを有するかを示すロック所有者パラメータとを含む、請求項１から８のいずれか一項に記載のデータ処理装置。

【請求項１３】

前記共有ロック・パラメータが、スレッドの前記部分集合のいずれかが共有リソースへの排他的アクセスを有するかどうかを示すロック・フラグを含む、請求項１２に記載のデータ処理装置。

【請求項１４】

前記共有ロック・パラメータが、前記ロック所有者パラメータによって示された前記スレッドが排他的アクセスを有するリソースの数を示すロック・カウントを含む、請求項１２及び１３のいずれか一項に記載のデータ処理装置。

【請求項１５】

第１のスレッドが第１の共有リソースへの排他的アクセスを有する場合、前記プロセッサが、第２のスレッドが第２の共有リソースへの排他的アクセスを有することを防ぐように構成される、請求項１２から１４のいずれか一項に記載のデータ処理装置。

【請求項１６】

前記少なくとも１つのロック・パラメータがスレッドが共有リソースへの排他的アクセスを有することを示す場合、前記セレクタが、前記ロック所有者パラメータによって示された前記スレッドを前記選択されたスレッドとして選び出すように構成され、

前記少なくとも１つのロック・パラメータが共有リソースへの排他的アクセスを有するスレッドがないことを示す場合、前記セレクタが、関数呼び出し深さパラメータに基づいて、スレッドの前記部分集合の中からスレッドのさらなる部分集合を選択し、且つ、スレッドの前記さらなる部分集合の前記スレッドに関連する前記スレッド・プログラム・カウンタに基づいて、スレッドの前記さらなる部分集合の中から前記選択されたスレッドを選び出すように構成される、請求項１２から１５のいずれか一項に記載のデータ処理装置。

【請求項１７】

前記プロセッサが、前記第１の命令に応答して前記少なくとも１つのロック・パラメータをインクリメントするように構成され、

前記プロセッサが、前記第２の命令に応答して前記少なくとも１つのロック・パラメータをデクリメントするように構成される、請求項１から１６のいずれか一項に記載のデータ処理装置。

【請求項１８】

前記少なくとも１つのロック・パラメータが、スレッドが排他的アクセスを有する共有リソースの数を示す、請求項１７に記載のデータ処理装置。

【請求項１９】

スレッドの前記部分集合の前記１つ又は複数が、前記選択されたスレッドの対応するスレッド・パラメータに一致する１つ又は複数の関連するスレッド・パラメータを有する前記スレッドを含み、前記１つ又は複数の関連するスレッド・パラメータが少なくとも前記スレッド・プログラム・カウンタを含む、請求項１から１８のいずれか一項に記載のデータ処理装置。

【請求項２０】

前記命令が、マイクロ－オペレーションを含む、請求項１から１９のいずれか一項に記載のデータ処理装置。

【請求項２１】

前記第１の命令と前記第２の命令とが、少なくとも１つの所定の命令セットにおいてＮＯＯＰ命令として符号化される、請求項１から２０のいずれか一項に記載のデータ処理装置。

【請求項２２】

前記プロセッサが、前記第１及び第２の命令の少なくとも１つに応答してさらなるオペレーションを行うように構成される、請求項１から２１のいずれか一項に記載のデータ処理装置。

【請求項２３】

前記さらなるオペレーションが、前記共有リソースへの排他的アクセスを獲得するロッキング・シーケンスの一部であるオペレーションと、前記共有リソースへの排他的アクセスを放棄するアンロッキング・シーケンスの一部であるオペレーションと、前記共有リソースにアクセスするオペレーションと、前記スレッドが前記共有リソースへの排他的アクセスを有するかを決定するオペレーションと、比較及び交換オペレーションと、の１つ又は複数である、請求項２２に記載のデータ処理装置。

【請求項２４】

前記共有リソースが、メモリ内のデータを含む、請求項１から２３のいずれか一項に記載のデータ処理装置。

【請求項２５】

前記プロセッサが、前記共有リソースへの排他的アクセスを放棄する前に、前記メモリにメモリ・バリア・オペレーションを発行するように構成される、請求項２４に記載のデータ処理装置。

【請求項２６】

複数のスレッドの部分集合について命令ストリームを実行するデータ処理方法であって、前記部分集合の各スレッドが前記命令ストリーム内の１つの命令を特定する関連するスレッド・プログラム・カウンタを有し、当該データ処理方法が、

スレッドの前記部分集合の選択されたスレッドを選び出し、且つ、汎用プログラム・カウンタを前記選択されたスレッドに関連する前記スレッド・プログラム・カウンタに設定するステップであって、前記汎用プログラム・カウンタが、スレッドの前記部分集合について実行されるべき命令ストリーム内の１つの命令を特定する、ステップと、

前記選択されたスレッドを含むスレッドの前記部分集合の１つ又は複数について、前記汎用プログラム・カウンタによって特定された命令を実行するステップと

を含み、

スレッドの前記部分集合が、スレッドの前記部分集合のどれが共有リソースへの排他的アクセスを有するかを追跡する、少なくとも１つのロック・パラメータに関連しており、

前記少なくとも１つのロック・パラメータが、スレッドがそのスレッドについて実行された第１の命令に応答して前記共有リソースへの排他的アクセスを獲得していることを示すように変更され、且つ、前記スレッドがそのスレッドについて実行された第２の命令に応答して前記共有リソースへの排他的アクセスをもはや有していないことを示すように変更され、

セレクタが、前記少なくとも１つのロック・パラメータに基づいて、前記選択されたスレッドを選び出すように構成される、データ処理方法。

【請求項２７】

複数のスレッドを実行するように構成されたデータ処理装置であって、

前記複数のスレッドの少なくとも部分集合について実行される、命令ストリーム内の１つの命令を特定する汎用プログラム・カウンタ手段であって、前記部分集合内の各スレッドが、前記命令ストリーム内の１つの命令を特定するための関連するスレッド・プログラム・カウンタ手段を有する汎用プログラム・カウンタ手段と、

スレッドの前記部分集合の選択されたスレッドを選び出し、且つ、前記汎用プログラム・カウンタ手段を前記選択されたスレッドに関連する前記スレッド・プログラム・カウンタ手段に設定する選択手段と、

前記選択されたスレッドを含むスレッドの前記部分集合の１つ又は複数について、前記汎用プログラム・カウンタ手段によって特定された命令を実行する、プロセッサ手段とを備え、

スレッドの前記部分集合が、スレッドの前記部分集合のどれが共有リソースへの排他的アクセスを有するかを追跡する、少なくとも１つのロック・パラメータに関連しており、

前記プロセッサ手段が、スレッドがそのスレッドについて実行された第１の命令に応答して前記共有リソースへの排他的アクセスを獲得していることを示すように前記少なくとも１つのロック・パラメータを変更し、且つ、前記スレッドがそのスレッドについて実行された第２の命令に応答して前記共有リソースへの排他的アクセスをもはや有していないことを示すように前記少なくとも１つのロック・パラメータを変更するためのものであり、

前記選択手段が、前記少なくとも１つのロック・パラメータに基づいて前記選択されたスレッドを選び出すためのものである、データ処理装置。

【発明の詳細な説明】

【技術分野】

【０００１】

本技法は、データ処理の分野に関する。特に、本技法は、複数のスレッドが実行される、データ処理の装置及び方法を考察する。

【背景技術】

【０００２】

データ処理装置は、複数のスレッドを実行し得る。スレッドは、ロックステップ方式で進行し得る。具体的には、各スレッドは、自身のプログラム・カウンタを維持し得る。汎用プログラム・カウンタによって示された命令が、そのプログラム・カウンタが汎用プログラム・カウンタに一致するスレッドについて実行されるように、システム自体が、自身の汎用プログラム・カウンタを維持し得る。このタイプのシステムは、たとえば、単一命令複数スレッド（ＳＩＭＴ：Single Instruction Multiple Thread）システムとして知られている場合がある。したがって、各ステップにおいて、単一の命令が、複数のスレッドの少なくとも部分集合に対して実行される。一旦、命令が、スレッドの部分集合に対して実行されると、汎用プログラム・カウンタは、新しい命令を示すように変更され得る。このようなシステムでは、汎用プログラム・カウンタを、スレッドのそれぞれによって維持されるプログラム・カウンタの最小のものに一致するように設定することが好適であり得る。この方法では、最も遅れているスレッドが進行することが許され、それによって、より高いプログラム・カウンタを有するスレッドに追いつくことができる。これにより、単一の命令がスレッドのできる限り多くにおいて実行され得るように、スレッドに収束に向かわせる、すなわち、同じプログラム・カウンタ値を共有させることができる。

【０００３】

このスキームに対する変更形態は、各スレッドの関数呼び出し深さを追跡することに関係する。具体的には、スレッドが関数呼び出しを行う度に、そのスレッド用の関数呼び出し深さカウンタがインクリメントされ、スレッドが関数呼び出しから戻る度に、そのスレッド用の関数呼び出し深さカウンタがデクリメントされる。汎用プログラム・カウンタが変わろうとするとき、それは、第１に、最も高い関数呼び出し深さカウンタを有するスレッドのプログラム・カウンタに、第２に（このようなスレッドが複数ある場合）、その集合からの、最も低いプログラム・カウンタ値を有する１つ又は複数のスレッドのプログラム・カウンタに、一致するように設定される。言い換えれば、プログラム・カウンタ値は、最も高い関数呼び出し深さを有するすべてのスレッド間でのみ考慮される。したがって、このことは、プログラム・カウンタ値のみが考慮された場合に起こり得るパフォーマンスの問題又はデッドロック状態を防ぐのに役立つ。スレッドの部分集合が関数呼び出しを行うと、これにより、それらのスレッドに関連するプログラム・カウンタが劇的に増加し得て、関数それ自体から次に実行されるべき命令が、最も低いプログラム・カウンタ値を有さないようになる。実行するスレッドを関数呼び出し深さに基づいて選択することによ

って、関数は、たとえそれが最も低いプログラム・カウンタを有していない場合であっても、最初に処理され得る。

【０００４】

上述の２つの機構の両方は、実行されるコードが、排他的アクセスが要求される共有リソースを備える場合のデッドロックに陥りやすい。たとえば、そのスレッドのみの共有リソースへのアクセスを可能になるように、ロックが任意の時点で単一のスレッドによって保持され得る。

【０００５】

共有リソースにアクセスするスレッドは、その共有リソースにアクセスできない残りのスレッドよりも高いプログラム・カウンタを有し得る。結果として、残りのスレッドは実行ができるようになる一方、共有リソースへのアクセスを有するスレッドは実行ができないようになる。残りのスレッドが共有リソースにアクセスすることができなくなる一方、別のスレッドが共有リソースへのアクセスを有することから、デッドロックが起こることになる。しかしながら、共有リソースへのアクセスを有するスレッドは、決して実行が許されないかもしれない。したがって、いずれのスレッドも進行せず、システムは停止する。

【発明の概要】

【課題を解決するための手段】

【０００６】

本技法の一態様によれば、複数のスレッドを実行するように構成されたデータ処理装置であって、複数のスレッドの少なくとも部分集合について実行される、命令ストリーム内の１つの命令を特定する汎用プログラム・カウンタであって、部分集合内の各スレッドが、命令ストリーム内の１つの命令を特定するための関連するスレッド・プログラム・カウンタを有する汎用プログラム・カウンタと、スレッドの部分集合の選択されたスレッドを選び出し、且つ、汎用プログラム・カウンタを選択されたスレッドに関連するスレッド・プログラム・カウンタに設定するように構成されたセレクタと、選択されたスレッドを含むスレッドの部分集合の１つ又は複数について、汎用プログラム・カウンタによって特定された命令を実行するように構成されたプロセッサとを備え、スレッドの部分集合が、スレッドの部分集合のどれが共有リソースへの排他的アクセスを有するかを追跡する、少なくとも１つのロック・パラメータに関連しており、プロセッサが、スレッドがスレッドについて実行された第１の命令に応答して共有リソースへの排他的アクセスを獲得していることを示すように少なくとも１つのロック・パラメータを変更し、且つ、スレッドがそのスレッドについて実行された第２の命令に応答して共有リソースへの排他的アクセスをもはや有していないことを示すように少なくとも１つのロック・パラメータを変更するように構成され、セレクタが、各スレッドに関連するロック・パラメータに基づいて選択されたスレッドを選び出すように構成されるデータ処理装置が提供される。

【０００７】

本技法の別の態様によれば、複数のスレッドの部分集合について命令ストリームを実行するデータ処理方法であって、部分集合の各スレッドが命令ストリーム内の１つの命令を特定する関連するスレッド・プログラム・カウンタを有し、データ処理方法が、

スレッドの部分集合の選択されたスレッドを選び出し、且つ、汎用プログラム・カウンタを選択されたスレッドに関連するスレッド・プログラム・カウンタに設定するステップであって、汎用プログラム・カウンタが、スレッドの部分集合について実行されるべき命令ストリーム内の１つの命令を特定する、ステップと、

選択されたスレッドを含むスレッドの部分集合の１つ又は複数について、汎用プログラム・カウンタによって特定された命令を実行するステップと

を含み、

スレッドの部分集合が、スレッドの部分集合のどれが共有リソースへの排他的アクセスを有するかを追跡する、少なくとも１つのロック・パラメータに関連しており、

少なくとも１つのロック・パラメータが、スレッドがそのスレッドについて実行された

第１の命令に応答して共有リソースへの排他的アクセスを獲得していることを示すように変更され、且つ、スレッドがそのスレッドについて実行された第２の命令に応答して共有リソースへの排他的アクセスをもはや有してないことを示すように変更され、
セレクタが、少なくとも１つのロック・パラメータに基づいて、選択されたスレッドを選び出すように構成されるデータ処理方法が提供される。

【０００８】

別の態様によれば、複数のスレッドを実行するように構成されたデータ処理装置であっては、
複数のスレッドの少なくとも部分集合について実行される、命令ストリーム内の１つの命令を特定する汎用プログラム・カウンタ手段であって、部分集合内の各スレッドが、命令ストリーム内の１つの命令を特定するための関連するスレッド・プログラム・カウンタ手段を有する汎用プログラム・カウンタ手段と、
スレッドの部分集合の選択されたスレッドを選び出し、且つ、汎用プログラム・カウンタ手段を選択されたスレッドに関連するスレッド・プログラム・カウンタ手段に設定する選択手段と、
選択されたスレッドを含むスレッドの部分集合の１つ又は複数について、汎用プログラム・カウンタ手段によって特定された命令を実行する、プロセッサ手段と
を備え、
スレッドの部分集合が、スレッドの部分集合のどれが共有リソースへの排他的アクセスを有するかを追跡する、少なくとも１つのロック・パラメータに関連しており、
プロセッサ手段が、スレッドがそのスレッドについて実行された第１の命令に応答して共有リソースへの排他的アクセスを獲得していることを示すように少なくとも１つのロック・パラメータを変更し、且つ、スレッドがそのスレッドについて実行された第２の命令に応答して共有リソースへの排他的アクセスをもはや有していないことを示すように少なくとも１つのロック・パラメータを変更するためのものであり、
選択手段が、少なくとも１つのロック・パラメータに基づいて選択されたスレッドを選び出すためのものであるデータ処理装置が提供される。

【０００９】

本技法のさらなる態様、特徴、利点、及び実例の実施例は、単なる例示として、以下の図面を参照して説明されることになる。

【図面の簡単な説明】

【００１０】

【図１】一実施例によるデータ処理装置を示す図である。
【図２】異なるプログラム・カウンタ値を有する、多くの異なるスレッドの実行の一実例を示す図である。
【図３】ＳＩＭＴシステムにおいてデッドロックが起こり得るコードを示す図である。
【図４】図３のデッドロックがどのように防がれ得るかを説明する、第２のコードの実例を示す図である。
【図５】選択が、どのようにＳＩＭＴシステムのスレッド間で起こり得るかをフローチャート形式において示す図である。
【図６】スレッドを実行する方法をフローチャート形式において示す図である。
【図７】スレッドの部分集合に対して維持された状態データの別の実例を図示する図である。
【図８】第３のコードの実例を図示する図である。
【図９】図７の状態データを使用してスレッド・プログラム・カウンタを選択する方法を示すフローチャートである。

【発明を実施するための形態】

【００１１】

一実例において、スレッドの部分集合は、部分集合のいずれのスレッドが、共有リソースへの排他的アクセスを有するかを追跡するための、それに関連している少なくとも１つ

のロック・パラメータを有する。プロセッサは、少なくとも１つのロック・パラメータを変更するようにスレッドによって実行された第１の命令に応答し、それによってそのスレッドがその共有リソースへの排他的アクセスを獲得していることを示し得る。同様に、プロセッサは、少なくとも１つのロック・パラメータを変更するように第２の命令に応答し、それによってそのスレッドがその共有リソースへの排他的アクセスをもはや有してないことを示し得る。そのスレッド・プログラム・カウンタが汎用プログラム・カウンタ用の値として使用され、それによって、スレッドの部分集合に対して次にどの命令を実行するかを決定する、選択されたスレッドの選び出しにおいて、データ処理装置は、その少なくとも１つのロック・パラメータを考慮し得る。ロック・パラメータに基づいて汎用プログラム・カウンタの値を選択することによって、いずれのスレッドが、任意の特定の時点において実行され得るかを制御し、それによってデッドロック発生の可能性を減らす又は防ぐことが可能である。

【００１２】

たとえば、第１の命令は、スレッドが共有リソースへの排他的アクセスを獲得する、又は獲得している場合に実行され得、第２の命令は、スレッドが共有リソースへのアクセスを放棄する、又は放棄している場合に実行され得る。

【００１３】

プロセッサは、スレッドについて実行された少なくとも１つのロック命令を含むロッキング・シーケンスに応答して、そのスレッドの共有リソースへの排他的アクセスを認めるように構成され得る。ロッキング・シーケンスは、様々な比較、ロード、及びストアを含む、多くの異なる命令を含み得る。そのシーケンス内には、共有リソースの「所有権」を最終的に設定するロック命令があり得る。いくつかの実例では、ロック命令は、共有リソースへの排他的アクセスを有するスレッドの識別を、格納又は記録させ得る。他の実例においては、ロック命令は、いずれのスレッドがロックの所有権を有するかを明示的に特定することなく、リソースがロックされていることの表示を設定するのみかもしれない。ロッキング・シーケンスは、場合によっては、第１の命令を含まない可能性がある。言い換えれば、スレッドに共有リソースへの排他的アクセスを獲得させるロッキング・シーケンスは、スレッドが共有リソースへの排他的アクセスを獲得していることをプロセッサに示す第１の命令から分かれている可能性がある。したがって、コードは、第１の命令に先行されるか又はその後に第１の命令が続くかのいずれかの、ロック命令を含むロッキング・シーケンスを含み得る。この手法は、第１の命令をサポートしない、レガシー・システムとの後方互換性を確実にするのに有用であり得る。第１の命令がサポートされていない場合でも、ロッキング・シーケンスは、なおレガシー・システムによって通常通りに処理され得る。いくつかの実例は、第１の命令の前に、ロッキング・シーケンスを実行することができ、最初に、スレッドが共有リソースへの排他的アクセスを獲得しようと試み、成功した場合、第１の命令を実行することによって、ロック・パラメータが、適宜更新されるようにする。他の実例は、ロッキング・シーケンスの前に第１の命令を実行することができ、ロックがロッキング・シーケンスを使用して実際に実施される前に、第１の命令が、スレッドに対するロッキング特権を要求するために有効に使用されるようにする。第２の場合では、ロッキング・シーケンスへの進行は、別のスレッドがすでにロックを有しているか否かに左右される可能性のある、第１の命令が成功のうちに実行されるかに左右される可能性がある。他の実施例においては、ロッキング・シーケンスは、第１の命令を含む。言い換えれば、ロッキング・シーケンスは、共有リソースへの排他的アクセスをスレッドの１つに認め、かつ、そのスレッド用のロック・パラメータを、そのスレッドが共有リソースへの排他的アクセスを有することを示すように設定する。このように、ロッキング・シーケンス及び第１の命令を実施する様々な方法がある。

【００１４】

同様に、プロセッサは、スレッドについて実行された少なくとも１つのアンロック命令を含むアンロッキング・シーケンスに応答して、そのスレッドの共有リソースへの排他的アクセスを放棄するように構成され得る。アンロッキング・シーケンスは、特定のスレッ

ドに、共有リソースへの排他的アクセスを失わせる、たとえば、比較、ロード、又はストアを含む、多くの異なる命令を含み得る。リソースへの排他的アクセスは、たとえば、リソースに対するロックを取り除くことによって、又は、異なるスレッドに排他的アクセスを渡すことによって、放棄され得る。アンロッキング・シーケンス内には、いずれのスレッドが共有リソースへの排他的アクセスを有するかを明らかにするために、又は特定のスレッドが共有リソースへのアクセスをもはや有していないことを示すために使用されるアンロック命令があり得る。言い換えれば、アンロック命令は、特定のスレッドが共有リソースへの排他的アクセスをもはや有していないことを示すよう、変数を設定又は消去するために使用され得る。いくつかの実施例では、アンロッキング・シーケンスは、第２の命令を含まない。言い換えれば、アンロッキング・シーケンスは、そのスレッドがその共有リソースへの排他的アクセスをもはや有していないことを示すようには、スレッドに関連するロック・パラメータを変更しない。したがって、アンロッキングを行うコード内の１つ又は複数の命令と、そのスレッドが共有リソースへのアクセスをもはや有していないことを示すようにロック・パラメータを設定する、コード内の命令との間には隔たりがある。このことは、上述のように、後方互換性のために有用である。第２の命令は、アンロッキング・シーケンスの後に、実行され得る。他の実施例では、アンロッキング・シーケンスは、第２の命令を含む。

【００１５】

セレクタは、少なくとも１つのロック・パラメータによって共有リソースへの排他的アクセスを有すると示された第１のスレッドを、少なくとも１つのロック・パラメータによって共有リソースへの排他的アクセスを有さないと示された第２のスレッドに優先して、選択されたスレッドとして選び出すように構成され得る。言い換えれば、セレクタは、共有リソースへの排他的アクセスを有するスレッドが優先されるように、構成され得る。したがって、そのスレッドは、共有リソースのロックを解くために、自身の共有リソースの使用を完了することが可能であり得る。このことは、共有リソースのロックを解くことができる第１のスレッドが、実行され、それによって共有リソースのロックを解くことが許されることから、デッドロックを防ぐのに役立つ。

【００１６】

セレクタは、スレッドの部分集合内の各スレッドに関連する関数呼び出し深さパラメータに基づいて、選択されたスレッドを選び出すように構成され得る。言い換えれば、セレクタは、いずれのスレッドのスレッド・プログラム・カウンタが、汎用プログラム・カウンタを設定するために使用されるべきかを決定する際、関数深さ呼び出しパラメータ及び少なくとも１つのロック・パラメータの両方を考慮し得る。

【００１７】

セレクタは、スレッドの部分集合の各スレッドに関連するスレッド・プログラム・カウンタに基づいて、選択されたスレッドを選び出すように構成され得る。言い換えれば、セレクタは、いずれのスレッド・プログラム・カウンタを、汎用プログラム・カウンタに設定すべきかを決める際、スレッド・プログラム・カウンタ及び少なくとも１つのロック・パラメータの両方を考慮し得る。

【００１８】

選択されたスレッドの選び出しは、様々な方法において実行され得る。いくつかの実例では、スレッドは、直接に選択され得、その後、そのスレッドのスレッド・プログラム・カウンタが、汎用プログラム・カウンタとして使用され得る。他の実例では、所与のプログラム・カウンタ選択アルゴリズム（対応するスレッドを選択されたスレッドとして非明示的に選び出す）を使用して、特定のスレッド・プログラム・カウンタが選択され得る。したがって、本出願における選択されたスレッドを選び出すことへの言及は、一般に、スレッドに関連するスレッド・プログラム・カウンタを選択することを含むことが意図されている。

【００１９】

ロック・パラメータは、多くの異なる形式をとり得る。一実例では、部分集合内の各ス

レッドは、対応するロック・パラメータを有し得る。共有リソースへの排他的アクセスを獲得するスレッドが、第１の命令を実行するとき、そのスレッドのためのロック・パラメータは、そのスレッドがそのリソースへの排他的アクセスを獲得していることを示すように変更され得る。同様に、スレッドが排他的アクセスをスレッドに譲り渡すとき、そのスレッドについて実行された第２の命令に応答して、そのスレッドがその共有リソースへの排他的アクセスをもはや有していないことを示すように、対応するロック・パラメータが変更され得る。たとえば、各スレッドのためのロック・パラメータは、そのスレッドによっていくつの共有リソースが現時点でロックされているかを示し得る。この場合、プロセッサは、一度に、複数のスレッドに、異なる共有リソースへの排他的アクセスを有することを許可し得る。選択されたスレッドを選び出す際、セレクタは、スレッドの第１の集合を選び出し（たとえば、ロックされたリソースの最大数を示すロック・パラメータを有するスレッドを選び出す）、次に、スレッドの第１の集合の中から、関数呼び出し深さに基づいてスレッドの第２の集合を選び出し、さらに、スレッドの第２の集合内のスレッドに関係するスレッド・プログラム・カウンタに基づいてスレッドの第２の集合の１つを選択されたスレッドとして選び出すために、スレッドのロック・パラメータを考慮し得る。この手法は、複数のスレッドが一度に異なるリソースに対するロックを保持することができ、それらの両方が互いに並行して進行することができることから、向上したパフォーマンスをもたらす。

【００２０】

言い換えれば、セレクタは、最初にスレッドのロック・パラメータ、次に関数呼び出し深さ、最後にプログラム・カウンタを考慮するように構成され得る。この順番でパラメータを考慮することによって、最も適したスレッドからの命令を、特定の時点で実行させることが可能になり得る。セレクタは、第１の集合又は第２の集合からの選び出しを行う必要がなくてもよいことに留意されたい。たとえば、単一のスレッドが所望のロック・パラメータを有していさえすれば、そのスレッドのスレッド・プログラム・カウンタが汎用プログラム・カウンタを設定するために使用されることになる。同様に、スレッドの第１の集合が２つのスレッドを含み、それらの２つのスレッドの１つのみが所望の関数呼び出し深さを有する場合、スレッドの第２の集合は特に形成されてなくてもよく、その代わりに、所望の関数呼び出し深さを有する、スレッドの第１の集合内のスレッドのスレッド・プログラム・カウンタが、汎用プログラム・カウンタに設定され得る。言い換えれば、セレクタは上述の３つの選び出しを行うように構成され得るが、必ずしもあらゆる場合において３つの選び出しが実行される必要がなくてもよい。

【００２１】

別の実例において、少なくとも１つのロック・パラメータは、スレッドの部分集合間で共有される共有ロック・パラメータと、スレッドの部分集合のいずれが、共有リソースへの排他的アクセスを有するかを示すロック所有者パラメータを含み得る。共有ロック・パラメータは、場合によっては、ロック所有者パラメータによって示されたロック所有スレッドがいくつのリソースへの排他的アクセスを有するかを示すロック・カウントを含み得る。他の場合では、共有ロック・パラメータは、スレッドの部分集合のいずれかが、現時点で、共有リソースへの排他的アクセスを有している否かを示すロック・フラグを含み得る。この実例の場合、プロセッサは、一度に、スレッドの部分集合当たりただ１つのスレッドしか共有リソースへの排他的アクセスを有することが許されないことを保証し得る。これにより、１つのスレッドが第１の共有リソースへの排他的アクセスを有する場合、第２のスレッドは、（第２の共有リソースが第１の共有リソースとは異なる場合でも）第２の共有リソースへの排他的アクセスが許されない。この手法は、様々なスレッドが、一連のリソースをロックすることをステップ・スルーし、その後、順にロックを解放するという再帰ロックがある場合に起こる可能性のあるデッドロックに対する向上した保護を提供する。実行されるコードがこのような再帰ロックを含み得ないことがわかっている場合、各スレッドが対応するロック・パラメータを有する上述の手法は、パフォーマンスを向上させるのに好適であり得る（また、上述の手法は、先行技術に比べ、デッドロックの発生

を大幅に減らす）。しかしながら、再帰ロックが所望される場合、一度に１つのスレッドにロックを制限することは、デッドロックに対する向上した保護を提供し得る。この場合、一度にただ１つのスレッドしかロックを保持することができないことから、各スレッドにロック・パラメータを提供する必要はない。その代りに、いずれのスレッドが現時点でロックを保持しているかを示すロック所有者パラメータ・スレッド、及びスレッドの部分集合間で共有されるロック・パラメータ（たとえば、ロック所有スレッドによって保持されたスレッド数を示すロック・カウント）は、いずれのスレッドがロックを保持しているかを追跡し、プログラム・カウンタの選択が、デッドロックを回避し、前進を確実にするために、他のスレッドよりもロックを保持するスレッドを選択することに有利に働くことができるようにするのに十分である。ロック・カウントは、スレッドによって設定されたすべてのロックが、他のスレッドにロックを保持させる前に、再び放棄されることを確実にするのに有用である。

【００２２】

第２の実例においては、少なくとも１つのロック・パラメータが、スレッドが１つ又は複数の共有リソースへの排他的アクセスを有することを示す場合、セレクタは、ロック・パラメータによって示されたロック所有スレッドを選択されたスレッドとして選び出し、それによってロック所有スレッドは、最終的にはロックを解放し別のスレッドが順次ロックを得ることができるように、前進することができる。一方、ロックを保持するスレッドがない場合、選択されたスレッドの選び出しは、第１の実例の場合のように、スレッドの関数呼び出し深さ及びスレッド・プログラム・カウンタに基づくことができる。

【００２３】

上述の両方の例において、少なくとも１つのロック・パラメータは、第１の命令に応答してインクリメントされ得、第２の命令に応答してデクリメントされ得る（ロック・パラメータが、スレッドの部分集合間で共有されるか、又は単一のスレッドに固有であるかに関わらず）。ロック・パラメータは、一実例では、２つの状態、すなわち、スレッドが共有リソースへの排他的アクセスを有することを示す第１の状態と、スレッドが共有リソースへの排他的アクセスを有していないことを示す第２の状態のみを有する可能性がある。このようなパラメータは、各スレッドについて１つのビットしか必要としないため、表しやすい。したがって、各スレッドの状態を表すために、非常に小さなスペース、よって非常に小さなエネルギーしか必要とされない。他の実施例においては、ロック・パラメータは、スレッドが排他的アクセスを有する共有リソースの数を示すことができる。言い換えれば、ロック・パラメータは、スレッドが共有リソースへの排他的アクセスを獲得するとインクリメントされ、スレッドが共有リソースへの排他的アクセスを失うとデクリメントされるカウンタとして機能し得る。このようなシステムは、各スレッドに、優先権のよりよい指標を提供する。具体的には、単に１つの共有リソースへの排他的アクセスを有するスレッドよりも、いくつかの共有リソースへの排他的アクセスを有するスレッドを優先させることが望ましいものであり得る。しかしながら、この付加的な情報の格納は、より大きなスペースを必要とし、それによって、システム内で実行されることになる各スレッドに、より多くのエネルギーが充てられる必要がある。

【００２４】

プロセッサによって実行されるスレッドの部分集合の１つ又は複数は、選択されたスレッドの対応するスレッド・パラメータに一致する１つ又は複数のスレッド・パラメータを有するスレッドを含むことができる。一実例では、１つ又は複数のスレッド・パラメータは、スレッド・プログラム・カウンタのみを含み得る。したがって、実行される１つ又は複数のスレッドは、そのスレッド・プログラム・カウンタが汎用プログラム・カウンタによって特定されるのと同じ命令を特定するスレッドであり得る。言い換えれば、データ処理装置は、そのプログラム・カウンタが汎用プログラム・カウンタと同じであるそれらスレッドのすべてを実行し得る。あるいは、スレッド・パラメータは、スレッド・プログラム・カウンタに加えて、関数呼び出し深さパラメータ又はロック・パラメータの１つ又は両方をさらに含むことができ、選択されたスレッドのスレッド・パラメータに一致するス

レッド・パラメータの組合せを有するスレッドについて命令が実行される。この方法では、単一の命令が、複数のスレッドについて、同時に実行され得る。

【００２５】

データ処理装置のプロセッサによって実行される命令は、マイクロ－オペレーションであり得る。いくつかのシステムでは、複雑な命令（たとえば、ロード／ストア多重命令）は、それらの命令がプロセッサに到達する前に、マイクロ－オペレーションに分割され得る。したがって、本出願における「命令」への言及は、命令、又は命令の部分に対応するマイクロ－オペレーションのいずれかを指すと解釈されるべきである。

【００２６】

第１の命令及び第２の命令は、少なくとも１つの所定の命令セットにおいて、ＮＯＯＰ（何もしない）命令として符号化され得る。このような場合、データ処理装置によって使用される命令セットは、所定の命令セットの拡張バージョンである。言い換えれば、データ処理装置によって使用される命令セットは、所定の命令セットにおいて定義されていない命令を定義し得る。第１の命令及び第２の命令が、少なくとも１つの所定の命令セットにおいてＮＯＯＰ命令として符号化される結果として、第１の命令及び第２の命令は、それらの所定の命令セットにおいて無視され得る。データ処理装置上で動作するように書かれたコードは、第１の命令及び第２の命令が第２のデータ処理装置に単に何の影響も及ぼさないことから、所定の命令セットを実施する第２のデータ処理装置上でなお正しく機能する。結果として、データ処理装置用に書かれているコードは、レガシー・システムとの後方互換性を有すると言える。このことは、異なるシステムに対して、異なるバージョンのコードを書く必要がなく、コードの開発をより効率的にすることを意味する。

【００２７】

プロセッサは、第１及び第２の命令の少なくとも１つに応答して、さらにオペレーションを行うように構成され得る。たとえば、さらなるオペレーションは、共有リソースへの排他的アクセスを獲得するロッキング・シーケンスの一部であるオペレーション、共有リソースへの排他的アクセスを放棄するアンロッキング・シーケンスの一部であるオペレーション、共有リソースにアクセスするオペレーション、スレッドが共有リソースへの排他的アクセスを有するか否かを決定するオペレーション、並びに比較及び交換オペレーションの、１つ又は複数であり得る。これにより、スレッドのためのロック・パラメータを更新する機能は別の命令で上書きされ得、実行される必要がある命令の数が減少し得る。

【００２８】

共有リソースは、メモリ内にデータを含み得る。共有リソースの他の実例は、ハードウェア・デバイス、通信チャネル、又はコードのクリティカル・セクションであり得る。

【００２９】

プロセッサは、共有リソースへの排他的アクセスを放棄する前に、メモリにメモリ・バリア・オペレーションを発行するように構成され得る。メモリ・システムは、トランザクションがプロセッサから受信される順番とは異なる順番で、トランザクションを処理し得る。メモリ・バリア・オペレーションは、メモリがトランザクションを並べ替えることができる範囲を制御するために、プロセッサによって発行され得る。メモリ・システムによるメモリ・オペレーションの並べ替えは、メモリ・バリアを越えては禁止される。すなわち、メモリは、連続するメモリ・バリア・オペレーション間では任意の順番で、自由にオペレーションを処理できるが、メモリ・バリア・オペレーションが受信されると、メモリは、メモリ・バリア・オペレーションの後に受信されたメモリ・オペレーションの前に、メモリ・バリア・オペレーションの前に受信されたすべてのメモリ・オペレーションを処理しなければならない。メモリ・バリアが発行されないと、リソースへの排他的アクセスが放棄された後に発行されたトランザクションが、排他的アクセスが依然として持続している間にメモリ・システムによって処理され、不正確なデータ値につながる可能性がある。共有リソースへの排他的アクセスを放棄する前にメモリ・バリア・オペレーションを発行することによって、システムは、共有リソースが一度に単一のスレッドによってのみアクセスされるようにすることを確実にすることができ、メモリ内のデータの一貫性を確保

することができる。

【００３０】

図１は、一実施例によるデータ処理装置１００を示す。汎用プログラム・カウンタ１２０が、セレクタ１１０によって、特定のスレッドに関連するプログラム・カウンタ１８４の１つに設定される。次に、汎用プログラム・カウンタ１２０の値が、汎用プログラム・カウンタ１２０によって参照された命令をフェッチするフェッチ・ユニット１３０に送信される。フェッチされた命令は、命令を復号する復号ユニット１４０に渡され、復号された命令が、発行ユニット１５０に送信される。発行ユニット１５０は、スレッドの１つ又は複数について、フェッチされた命令を実行するために、プロセッサ１６０に１つ又は複数の信号を発行する。プロセッサ１６０は、１つ又は複数のスレッドについて、命令を、同時に又はほぼ同時に実行することが可能であり得る。いくつかの実施例では、プロセッサ１６０は、同じ命令が複数のスレッドについて並行して実行されるように、それぞれのスレッドを処理する並列機能ユニットを備え得る。他のシステムは、単一の機能ユニットのみを備えてもよく、それによって同じ命令が、次の命令に移る前に、スレッドの部分集合のそれぞれについて、連続して処理される。他のシステムは、いくつかの並列ユニットを備え得るが、スレッドの総数よりも少なく、それによって、スレッドの総数よりも少ないバッチにおいて、同じ命令が多くのスレッドについて処理される。

【００３１】

プロセッサ１６０はまた、メイン・メモリだけでなくキャッシュも備え得る、メモリ１７０にアクセスし得る。プロセッサはまた、データ値がその中に読み込まれ、又は格納され得るレジスタ・ファイル１８０と通信し得る。この実施例では、レジスタ・ファイル１８０は、各スレッドに対して、多くの異なるハードウェア・ユニットを備える。図１に示されているレジスタ・ファイル１８０は、たとえば、各スレッドに対して、一組のレジスタ１８２、プログラム・カウンタ１８４、関数呼び出し深さカウンタ１８６、及びロック・カウンタ１８８を備える。図１に示されているものと異なるハードウェア構成が可能であり得ることが理解されるであろう。たとえば、単一のレジスタ・バンク１８２は、１つよりも多いスレッドのためのレジスタを備え得る。さらに、示されているハードウェア構成要素のサブセットは、レジスタ・ファイル１８０の外にあり得る。

【００３２】

本明細書において説明されている実施例では、それらのスレッドの少なくとも部分集合が並行して実行される、スレッドの単一の集合のみが考察されている。しかしながら、データ処理装置１００はこのような構成に限定されないことが理解されるであろう。たとえば、データ処理装置１００は、スレッドの各群がほぼ同時に実行されることが可能である、スレッドの複数の群又は部分集合上で実行し得る。したがって、汎用プログラム・カウンタ１２０は、集合内の各汎用プログラム・カウンタが異なるスレッド群に関係付けられている、一組の汎用プログラム・カウンタを備え得る。本明細書の残りの部分はスレッドの単一の群（部分集合）のみを考察するが、説明される本技法が、複数のスレッド群がある場合には、それぞれの異なるスレッド群に適用され得ることが理解されるであろう。

【００３３】

図２は、一連の命令を実行する単一のスレッド群Ｔ＃０～Ｔ＃３１を示す。スレッドのそれぞれは、自レジスタを有し、また、実行される命令がレジスタを参照することから、同じ命令を実行する際、スレッドのそれぞれが、それぞれ異なって動作し得る。命令０は、レジスタ５及びレジスタ３内に保持されたデータ値を、加算し、レジスタ０内に格納されるようにする。ライン１の命令は、その前の和が０であった場合、スレッドの実行を、ライン３における「label」にジャンプさせる。この場合、スレッドＴ＃２によって行われた加算（０＋０）の結果、及びスレッドＴ＃４によって行われた加算（－１＋１）の結果は両方とも０であり、したがって、これら２つのスレッドの制御の流れは、「label」にジャンプする。他のスレッドはジャンプせず、その代わりに、ライン２における命令に続く。

【００３４】

結果として、スレッドＴ＃２及びＴ＃４のためのプログラム・カウンタは３に等しく、残りのスレッドのためのプログラム・カウンタは２に等しい。スレッドの収束を促進するために、データ処理装置１００のための汎用プログラム・カウンタ１２０は、すべてのスレッドの中で最も低いスレッド・プログラム・カウンタ１８４（すなわち、２）に設定されることになる。このように汎用プログラム・カウンタ１２０の値を選択することによって、それほど進行していないスレッドは進行するようにされ、したがって他のスレッドに追い付くことができ、こうしてスレッドの収束に導く。これは、それによってシステムの並列処理が向上し、すなわち、より多くのスレッドが並行して実行されるようになることから、そうなることが望ましい状態である。したがって、汎用プログラム・カウンタ１２０は、いずれかのスレッドに関連する最も低いプログラム・カウンタ値である、値２に設定される。スレッドＴ＃２及びＴ＃４は、２に等しいスレッド・プログラム・カウンタを有さないため、スレッドＴ＃２及びＴ＃４については、ライン２における命令は実行されない。残りのスレッドは、レジスタ０及びレジスタ４におけるデータ値を、掛け合わせてレジスタ６に格納されるようにする、ライン２における命令を実行する。乗算を行うと、命令を実行したスレッドのそれぞれに対するスレッド・プログラム・カウンタは、３に進む。したがって、すべてのスレッドは同じプログラム・カウンタを有し、収束が達成される。ライン３における「命令」は単にラベルであり、そうして実行はライン４に進む。この命令により、レジスタ９に格納されているメモリ・アドレスがメイン・メモリからアクセスされ、そのアドレスのデータ値がレジスタ８に格納される。図２に示されているように、スレッドのそれぞれは、異なるメモリ・アドレス値をレジスタ９に格納しており（たとえば、スレッドＴ＃０はメモリ・アドレス１００を格納している一方、スレッドＴ＃５はメモリ・アドレス２００を格納している）、スレッドのそれぞれは、異なるメモリ・アドレスにアクセスし、したがって、異なる値をそれらのそれぞれのレジスタ８に格納することになる。メモリの同じキャッシュ・ライン又は同じページをターゲットにするメモリ・アクセスを単一のメモリ・アクセスにまとめ、電力と時間を節約することが可能である。

【００３５】

図３は、それへのアクセスがロックを介して制御される共有リソースを含む、２つのスレッド上で実行されるコードを示す。ここで、共有リソースは、ライン７とライン９との間のコードのクリティカル・セクションによってアクセスされる。共有リソースは、コードのクリティカル・セクションによって要求され、かつ、リソースへの排他的アクセスが一度に１つのスレッドによって必要とされる、任意のものであり得る。たとえば、リソースは、共有データ構造、又はハードウェア・ユニット若しくはデバイスの使用であり得る。あるいは、コードのクリティカル・セクションそのものが、共有リソースと見なされ得る。この実例では、セレクタ１１０は、汎用プログラム・カウンタ１２０を、最も高い関数呼び出し深さカウンタ１８６を有するスレッドのスレッド・プログラム・カウンタ１８４に設定するように構成されている。複数のスレッドが最も高い関数呼び出し深さカウンタ１８６を有する場合、汎用プログラム・カウンタ１２０は、最も高い関数呼び出し深さカウンタ１８６を有するスレッドの中からの最も低いプログラム・カウンタ１８４に設定される。上述のように、通常の状況下では、スレッドの収束が起こるようにするために、より高いプログラム・カウンタ値を有するスレッドに優先して、最も低いプログラム・カウンタを有するスレッドが実行されるようにできることが望ましい。しかしながら、これから示されるように、プログラム・カウンタ選択のこの機構は、コードのクリティカル・セクションがある場合、デッドロックを生じさせる。

【００３６】

図３のいくつかの命令は、レジスタｘ１９内のアドレスを参照する。このアドレスは、共有リソースがスレッドによる排他的アクセスのためにロックされているか否かを示すデータ値に対応し得る。この実例では、レジスタｘ１９によって参照されたアドレスにおける値が０の場合、これは、ライン７～９におけるコードのクリティカル・セクションによって使用されるリソースについてロックが設定されていないことを示し、このアドレスに

おける値が１の場合、ロックが設定されている。しかしながら、リソースに対して排他的アクセスが認められているか否かを表す他の方法があることが理解されるであろう。スレッドがこのコード実例を実行するとき、ライン２及び５における命令を実行する異なるスレッドが、異なるメモリ・アドレスを含み得る、それらのそれぞれのレジスタの組１８２内のレジスタｘ１９の異なるバージョンにアクセスし得ることに気付かれよう。したがって、スレッドがアクセスしようとする共有リソースは、スレッド間で異なる可能性がある。一方、レジスタｘ１９が同じアドレスを含む複数のスレッドがある場合、以下に述べられるように、矛盾が潜在する。

【００３７】

最初、すべてのスレッドは、ロックステップ方式で実行されることになる。ライン０はラベルを含み、したがって実行に何の影響もない。ライン１は、データ値１をレジスタｗ１内に格納させる。ライン２における命令は、ロード排他（load exclusive）命令として知られている。これにより、データ値がアクセスされ、かつ、監視（monitor）が設定される。監視は、データ値が変更されているか否かを検出する。したがって、この実例では、レジスタｘ１９において参照されたメモリ・アドレスに格納されたデータ値がアクセスされ、レジスタｗ０に格納され、またそのメモリ・アドレスのための監視が設定される。ライン３において、レジスタｗ０に格納されたデータ値が、数０と比較される。言い換えれば、監視を設定することとは別に、ライン２及び３は、共同で、レジスタｘ１９で参照されたメモリ・アドレスに格納されたデータ値が０と等しいか否かを決定する。ライン４における命令により、データ値が等しくない場合、実行は、ライン０におけるラベルretry_lockにジャンプする。言い換えれば、レジスタｗ０に格納されたデータ値が０と等しくない（すなわち、別のスレッドが、共有リソース上にすでにロックを有している）場合、制御の流れは、ラベルretry_lockに戻り、さもなければ、制御の流れはライン５に続く。それにより、ライン２～４における命令は、ライン２におけるロード命令が実行された時点までに別のスレッドがすでにロックを有しているか否かを確認する。

【００３８】

一方、ライン５及び６における命令は、ライン２における排他的ロード命令を実行した後、コードのクリティカル・セクションを開始する前のある時点で、別のスレッドがロックを獲得しているか否かを確認する。これにより、複数のスレッドが同時にロックを獲得できないことを確実にする。ライン５は、予め設定された監視が、監視されているメモリ・アドレスが変更されていないことを示す場合にのみ、データ値を格納させるストア排他（store exclusive）命令である。この実例では、命令は、レジスタｘ１９によって参照されたメモリ・アドレスに格納されたデータ値がライン２のロード排他命令によって監視が確立されてから変更されていない場合のみ、レジスタｗ１に格納されたデータ値（すなわち１）をレジスタｘ１９によって参照されたメモリ位置に格納させる。そして、この格納が成功したか否かの場合の結果がレジスタｗ２に格納される。具体的には、格納が成功した場合、値０がｗ２に格納される。そうではなく、格納が成功しなかった場合、非ゼロ値がｗ２に格納される。当然ながら、成功又は失敗を示すために使用される特定の数字は重要ではなく、これらの数字は簡単に逆にされ得ることが当業者には理解されるであろう。別のスレッドがレジスタｘ１９によって参照されたアドレスに格納されたデータ値を変更していない場合のみストア・オペレーションが実行されることから、レジスタｘ１９の値がすべてのスレッドで同じ場合、ただ１つのスレッドがデータ値を変更することになる。もし異なるスレッドが、異なるロックされたリソースを要求するならば、異なるスレッドについて異なるアドレスがレジスタｘ１９に置かれることが可能で、それによって、１つより多いスレッドが、ライン５におけるストア命令を成功のうちに実行することが可能になる。したがって、各スレッドに対するストア排他命令の結果は、そのスレッドがロックを取得しているか否かを示す。ライン６において、レジスタｗ２に格納されたデータ値が０であるか否か（すなわち、ストア排他命令が失敗に終わったか否か）を決定するために、比較が行われる。ｗ２に格納されたデータ値が０ではない場合、制御の流れは、ライン０のラベルretry_lockに戻る。さもなければ、流れはライン７に続く。

【００３９】
ライン７～９（図３では図示せず）は、クリティカル・セクションを表す。すなわち、その前のロッキング・シーケンスによって、クリティカル・セクションは、一度に共有リソース当たり１つのスレッドのみによって実行され得る（異なるリソースを使用している場合、複数のスレッドがクリティカル・セクションを実行し得る）。同じリソースを要求する別のスレッドが入ることができるようになる前に、スレッドはクリティカル・セクション内のコードの実行を終了しなければならない。クリティカル・セクションは、データの一貫性を維持するために排他的アクセスが要求される共有位置若しくは共有データ構造にアクセスするロード／ストア命令や、一度に１つのスレッドによってのみ使用され得るハードウェア・デバイスを利用する命令などの、共有リソースを利用する任意の命令を含み得る。

【００４０】
ライン１０から、アンロッキング・シーケンスが示されている。このコードは、クリティカル・セクションが実行された後に実行される。ライン１０において、データ値０がレジスタｗ１に格納される。ライン１１において、レジスタｗ１に格納されたデータ値（すなわち０）が、このスレッドがもはやロックを有してないことを示すために、レジスタｘ１９で参照されたメモリ位置に格納される。これは、ライン２～４における命令に到達する別のスレッドが今度はロックの獲得に成功できることを意味する。

【００４１】
上述のように、クリティカル・セクションにロックを提供するこのコードは、複数のスレッドが同じロックを得ようとしている（すなわち、同じメモリ位置が、異なるスレッドのためのレジスタ・バンク１８２内のレジスタｘ１９のそれぞれのバージョンに示されている）場合、ＳＩＭＴデータ処理装置にデッドロックを生じさせる可能性がある。たとえば、図３に示されているように、スレッドＴ＃０は、ロックへのアクセスを取得し、したがってコードのライン７まで続いているスレッドであり得る。しかしながら、スレッドＴ＃１（及び、同じリソースをターゲットとする他のスレッド）は、ロックの獲得に失敗することになる。したがって、コードの最初の実行では、スレッドＴ＃０以外のスレッドは、ロックの獲得の失敗によってそれらの制御の流れがライン０に戻される前に、コードのライン６まで実行することになる。それに続く試みでは、スレッドＴ＃０がレジスタｘ１９で参照されたアドレスに値＃１を今度は格納していることになるため、スレッドＴ＃０以外のスレッドはライン４まで実行し得る。すべてのスレッドは、１の同じ関数呼び出し深さカウンタ値を有する。したがって、セレクタ１１０は、汎用プログラム・カウンタ１２０を、スレッドのそれぞれに関連する最も低いスレッド・プログラム・カウンタ１８４に設定することになる。スレッドＴ＃０以外のすべてのスレッドは０のプログラム・カウンタ値を有し、それによって、汎用プログラム・カウンタ値１２０は０に設定されることになる。したがって、スレッドＴ＃０以外のスレッドは実行が許されることになる。しかしながら、それらのスレッドは、ロックがＴ＃０によって保持されていることから、なおクリティカル・セクションへのロックの獲得に失敗することになる。したがって、ライン４において、それらのスレッドのための制御の流れはライン０に進むことになる。再度、プログラム・カウンタ値の選択がセレクタ１１０によって行われるとき、スレッドＴ＃０以外のスレッドはスレッドＴ＃０の値（７）よりも低いプログラム・カウンタ値（０）を有する。したがって、汎用プログラム・カウンタ値１２０は０に設定されることになり、スレッドＴ＃０以外のスレッドは実行が許されることになる。このプロセスは、場合によっては永遠に続くことになる。スレッドＴ＃０のみがクリティカル・セクションへのアクセスのロックを解くことができ、スレッドＴ＃０は、クリティカル・セクションが実行された後でのみ、クリティカル・セクションへのアクセスのロックを解くことができる。しかしながら、スレッドＴ＃０は、そのプログラム・カウンタ値が他のスレッドのそれよりも高いためにクリティカル・セクションを実行することができず、それによって汎用プログラム・カウンタ１２０は、決して、スレッドＴ＃０に関連するスレッド・プログラム・カウンタ１８４に設定されないことになる。したがって、この実例ではデッドロックが起

こる。進行できるスレッドはなくなり、それによりシステムは停止する。

【００４２】

図４は、このデッドロック問題のソリューションを示す。この実施例では、各スレッドにロック・カウンタ１８８が提供され、セレクタ１１０は、関数呼び出し深さ及びスレッド・プログラム・カウンタに加えてロック・カウンタに基づいて、選択されたスレッド（そのスレッド・プログラム・カウンタが汎用プログラム・カウンタに設定される）を選び出すように構成される。汎用プログラム・カウンタ１２０は、最も高いロック・カウンタ１８８を有するスレッドに関連するスレッド・プログラム・カウンタ１８４に設定される。複数のスレッドが最も高いロック・カウンタ１８８を有する場合、最も高いロック・カウンタ１８８を有するスレッドのうち、最も高い関数呼び出し深さカウンタ１８６を有するスレッドが考慮される。複数のスレッドが最も高いロック・カウンタ１８８及び最も高い関数呼び出し深さカウンタ１８６を有する場合、それらのスレッドのうち、最も低いプログラム・カウンタ値を有するスレッドが選択される。そして、汎用プログラム・カウンタが、選択されたスレッドのスレッド・プログラム・カウンタに基づいて更新され、次に、汎用プログラム・カウンタによって示された命令が、一致するスレッド・パラメータ（少なくとも一致するスレッド・プログラム・カウンタ、また任意選択で、一致する関数呼び出し深さ及びロック・パラメータ）を有する任意のスレッドについて実行されることができる。他の実施例が、（スレッドを選択するのではなく）特定のスレッド・プログラム・カウンタ、又はスレッド・プログラム・カウンタ、関数呼び出しパラメータ及びロック・パラメータの特定の組合せが直接選択される異なる機構を適用し得ることが理解されるであろう。

【００４３】

図４のコードは、２つの新しい命令が、それぞれライン６、ライン１２に加わっている、図３のコードと同じである。これらの命令は、スレッドに関連するロック・カウンタ１８８の条件付きインクリメント及びデクリメントをもたらす。具体的には、ライン６の命令「cond_inc_lock_count w2」が、ｗ２の値がゼロである場合、スレッドに関連するロック・カウンタ１８８をインクリメントさせる。図３に関して説明されたように、ｗ２の値は、値０がＳＴＸＲ命令の成功（すなわち、スレッドがロックの取得に成功したこと）を表し、１がＳＴＸＲ命令の失敗（すなわち、スレッドがロックの取得に失敗したこと）を表す、０又は１に設定される。したがって、ライン６の命令は、特定のスレッドのためのロック・カウンタ１８８を、そのスレッドがライン５の手続命令において何とかロックを取得していた場合、インクリメントさせる。ライン１２における命令「dec_lock_count」は、特定のスレッドのためのロック・カウンタ１８８をデクリメントする。この命令は、無条件である。それは、ライン８～１２におけるコードが、一度に単一のスレッド、具体的には、現時点でロックを保持し、したがってクリティカル・セクションへのアクセスを有するスレッドのみによって実行されるからである。結果として、いずれのスレッドのロック・カウンタがデクリメントされるべきであるかに関して曖昧さがない。

【００４４】

コードが実行されると、ライン６における命令の結果として、ロックを取得したスレッド（Ｔ＃０）は、そのロック・カウンタがインクリメントされる。反対に、他のスレッドは、レジスタｗ２のそれらの値が非ゼロである（ロックの取得に失敗した）ため、それらのロック・カウンタをインクリメントしない。ライン７において、スレッドＴ＃０はライン８に進むことになるが、残りのスレッドはライン０におけるラベル「retry_lock」に戻ることになる。上で説明されるように、この実施例では、セレクタ１１０は、汎用プログラム・カウンタ１２０を、最も高いロック・カウンタを有するスレッドのプログラム・カウンタに設定するように構成される。したがって、スレッドＴ＃０のプログラム・カウンタが８である場合、そのスレッドは、そのプログラム・カウンタ１８２が他のスレッドのそれよりも高くても、そのより高いロック・カウンタ１８８によって、実行を継続することができることになる。したがって、スレッドＴ＃０は実行を継続することができ、ロックを解放するライン１０及び１１のアンロッキング・シーケンスのコードをやがて実行す

ることになる。その後、スレッドＴ＃０は、そのロック・カウンタがデクリメントされるライン１２に続く。この時点で、それぞれのスレッドのロック・カウンタは０に等しく、それぞれのスレッドの関数呼び出し深さは１に等しく、よって、汎用プログラム・カウンタ１２０は、すべてのスレッドのうち最も低いスレッド・プログラム・カウンタ１８４に設定される。したがって、スレッドＴ＃０以外のスレッドは実行が許され、ライン０～７のコードを実行する。この実行中、スレッドの１つがロックを取得し、上述のように、そのロック・カウンタ１８８がインクリメントされる。このシーケンスは、すべてのスレッドがコードのクリティカル・セクションを通り過ぎるまで、繰り返されることになる。したがって、コードのクリティカル・セクションの存在にも関わらず、デッドロック状態が回避される。

【００４５】

この図４の実例では、cond_inc_lock_count及びdec_lock_countの命令は、特定のスレッドに関連するロック・カウンタ１８８に影響を及ぼすだけである。しかしながら、これらの命令が追加のオペレーションを行い得ることが理解されるであろう。たとえば、条件付きインクリメント命令は、ライン５の排他的ストア命令又はライン７の比較分岐命令のいずれかと組み合わされる可能性がある。このような場合、条件付きインクリメント命令は、比較及び分岐命令に関するオペレーションの前又はそれと並行して、且つ排他的ストア命令に関するオペレーションの後に行われる。同様に、デクリメント・ロック・カウンタ命令（decrement lock counter instruction）は、他のオペレーションと組み合わされ得る。たとえば、デクリメント・ロック・カウンタ命令は、レジスタｘ１９によって指し示されたアドレスに数値０を格納する命令と組み合わされ得る。

【００４６】

また、ロッキング・シーケンスを実施する多くの方法がある。図４の実例では、ライン２のロード排他命令と、ライン５のストア排他命令との組合せが使用されている。ロッキング機構を実施する別の方法は、比較及び交換命令（ＣＭＰＸＣＨＧ：compare and exchange instruction）の使用を通してであり得る。比較及び交換命令は、アトミック・プリミティブである。言い換えれば、この命令は実行途中で中断され得ず、その代わりに、いったん始まると完了まで及ぶ。比較及び交換命令は３つのパラメータを使用する。それらのパラメータの１つはメモリの位置である。もう１つのパラメータは比較値であり、残りのパラメータはストア値である。比較及び交換命令は、メモリ内の位置におけるデータ値が比較値と等しいか否かをテストし、そうであれば、ストア値をメモリ位置に書き込み、そのオペレーションが成功したことを示す結果を返す。メモリ位置におけるデータ値が比較値と等しくない場合、メモリ位置に何も書き込まず（メモリ位置は、その元の値のままである）、その代わりに、その結果が、オペレーションが成功しなかったことを示す。繰り返すが、このような比較及び交換命令は他のオペレーションと組み合わされ得、cond_inc_lock_count命令と組み合わされ得る。

【００４７】

メモリ・バリア・オペレーションが、dec_lock_count命令の前、且つコード内のクリティカル・セクションの後に行われ得る。データ処理装置において、メモリ・アクセス命令が、効率化のために並べ替えられ得る。しかしながら、このような並べ替えは、メモリ・バリアを越えては起こり得ない。これにより、メモリ・バリアが、コードのクリティカル・セクションの一部であるメモリ・アクセス・オペレーションが、クリティカル・セクションが完了した後に発行されるメモリ・アクセス・オペレーションに先立って、処理されることを確実にする助けとなり得る。

【００４８】

cond_inc_lock_count命令及びdec_lock_count命令はそれぞれ、所定の命令セットにおいて、cond_inc_lock_count命令及びdec_lock_count命令の符号化が、ＮＯＯＰ命令に一致するように、符号化され得る。たとえば、データ処置装置１００によって使用される命令セットは、これら２つの命令をＮＯＯＰ命令として符号化する命令セットの拡張バージョンであり得る。結果として、データ処理装置１００以外のデータ処理装置において、こ

の２つの命令は何の影響も及ぼさない。したがって、データ処理装置１００での使用のために設計されたコードは、他のデータ処理装置との後方互換性をもち得、ＳＩＭＴを使用しないデータ処理装置との後方互換性をもち得る。

【００４９】

図５は、そのスレッド・プログラム・カウンタが汎用プログラム・カウンタ１２０として使用される、選択されたスレッドを選び出す方法を図示している。当該方法は、第１の集合及び第２の集合の両方が空集合に設定される、ステップＳ２００において始まる。ステップＳ２０５において、すべてのスレッドのうちの次のスレッドが選択される。このステップはステップＳ２１０、Ｓ２１５、及びＳ２２０と共に、共同で、すべてのスレッドを通して繰り返すループを形成する。ステップＳ２１０では、ステップＳ２０５において選択されたスレッドのロック・パラメータ（すなわち、ロック・カウンタ１８８）が、すべてのスレッドの最も高いロック・パラメータと等しいか否かが決定される。このスレッドのロック・パラメータがすべてのスレッドの最も高いロック・パラメータと等しいならば、流れはステップＳ２１５に進み、さもなければ、流れはステップＳ２２０に続く。ステップＳ２１０では、データ処理装置は、すべてのスレッドの最も高いロック・パラメータを知っていると仮定される。もし、この情報がすぐに利用できない場合、最初にすべてのスレッドを通して繰り返すことによって、又は、以前のスレッドから確認された最も高いロック・パラメータの継続したカウント（running count）を保持し、以前の最も高いロック・パラメータよりも高いロック・パラメータに遭遇したときに、第１の集合にすでに加えられているスレッドを捨てることによって、決定され得る。ステップＳ２１５において選択されたスレッドが第１の集合に加えられ、その後、流れはステップＳ２２０に進む。ステップＳ２２０において、繰り返すスレッドがもっとあるか否かが決定される。もっとスレッドがある場合、流れは、次のスレッドが選択されるステップＳ２０５に戻る。もっとスレッドがない場合は、流れはステップＳ２２５に続く。これにより、ステップＳ２２０の終了までに、スレッドのいずれかのより高いロック・パラメータに等しいロック・パラメータを有するスレッドを含む、スレッドの第１の集合が決定される。ステップＳ２２５は、第１の集合にエントリがただ１つであるか否かを決定する。第１の集合にエントリがただ１つである場合、流れは、第１の集合内のエントリが選択されたスレッドとして返されるステップＳ２３０に続く。すなわち、汎用プログラム・カウンタ１２０は、第１の集合のただ１つのスレッドに関連するスレッド・プログラム・カウンタ１８４に設定される。ステップＳ２２５において、第１の集合のエントリがただ１つではない場合、流れはステップＳ２３５に続く。ステップＳ２０５～Ｓ２３０は、共同で、スレッド・プログラム・カウンタ選択の決定を、スレッドのすべてのロック・カウント・パラメータに基づかせようとする。もし、スレッドの１つがその他のスレッドのすべてより高いロック・カウンタ・パラメータを有するならば、そのスレッドは選択されたスレッドであり、汎用プログラム・カウンタ１２０は、より高いロック・カウンタ・パラメータ１８８を有するスレッドに対応するスレッド・プログラム・カウンタ１８４に設定される。さもなければ、スレッドのさらなる絞り込みが、以下に説明されるように行われる。

【００５０】

ステップＳ２３５において、第１の集合から次のスレッドが選択される。ステップＳ２３５～Ｓ２５０は、共同で、第１の集合のすべてのスレッドを通して繰り返すループを形成する。ステップＳ２４０において、選択されたスレッドの関数呼び出し深さが、第１の集合のすべてのスレッドの最も高い関数呼び出し深さに等しいかが決定される。再び、第１の集合のすべてのスレッドの最も高い関数呼び出し深さを決定することが可能であると仮定される。この情報が決定され得る１つの方法は、最初に、第１の集合のすべてのスレッドを通して繰り返すことであるか、又は上述のように継続したカウント値を維持することによる。ステップＳ２４０において、選択されたスレッドの関数呼び出し深さが、第１の集合のすべてのスレッドの最も高い関数呼び出し深さに等しいならば、流れはステップＳ２４５に続く。さもなければ、流れはステップＳ２５０に続く。ステップＳ２４５において、選択されたスレッドが第２の集合に加えられ、流れはステップＳ２５０に続く。ス

テップＳ２５０において、繰り返されるべきスレッドが第１の集合にもっとあるか否かが決定される。第１の集合にもっとスレッドがある場合、流れはステップＳ２３５に戻り、ループを継続する。さもなければ、流れは、第２の集合にエントリはただ１つであるか否かが決定されるステップＳ２５５に続く。ステップＳ２５５において、第２の集合にエントリがただ１つである場合、流れはＳ２６０に続く。ステップＳ２６０において、第２の集合の単一のエントリは、選択されたスレッドとして返される。言い換えれば、汎用プログラム・カウンタ１２０は、第２の集合の単一のスレッドに関連するスレッド・プログラム・カウンタ１８４に設定される。このような状況は、たとえば、１つより多いスレッドが最も高いロック・カウンタ・パラメータ１８８を共有しているが、それらのスレッドのただ１つが最も高い関数呼び出し深さカウンタ１８６を有する場合に、起こり得る。したがって、このようなスレッドは、汎用プログラム・カウンタ１２０をそのスレッドに関連するスレッド・プログラム・カウンタ１８４に設定することによって、実行が許される。ステップＳ２３５～Ｓ２６０は、共同で、汎用プログラム・カウンタ１２０を決定するために使用される第２の基準に関わる。

【００５１】

第２の集合にエントリがただ１つではない場合、流れは、最も低いプログラム・カウンタ１８４を有する第２の集合のエントリが選択されたスレッドとして返されるステップＳ２６５に進む。言い換えれば、汎用プログラム・カウンタ１２０は、第２の集合のすべてのスレッドの中でスレッド・プログラム・カウンタ１８４の最も低いものに設定される。したがって、ステップＳ２６５は、汎用プログラム・カウンタ１２０を決定する際の第３の基準を提供する。

【００５２】

図６は、各周期において実行されるべきスレッドがどのように選択されるかを示す。ステップＳ４００において、複数のスレッドの１つ（又は同義的に、スレッドに対応するスレッド・プログラム・カウンタの１つ）が、ロック・パラメータ（ロック・カウンタ）１８８、関数呼び出し深さカウンタ１８６及びスレッド・プログラム・カウンタ１８４に基づいて選択される。このプロセスの実例が、図５に関して示される。ステップＳ４１０において、汎用プログラム・カウンタ１２０が、選択されたスレッドのスレッド・プログラム・カウンタに設定される。言い換えれば、汎用プログラム・カウンタ１２０は、ステップＳ４００において選択されたスレッドに対応するスレッド・プログラム・カウンタ１８４に一致するように設定される。最後に、ステップＳ４２０において、関連するスレッド・パラメータが選択されたスレッドのスレッド・パラメータに一致するすべてのスレッドについて、汎用プログラム・カウンタ１２０によって特定された命令が実行される（スレッド・パラメータは、スレッド・プログラム・カウンタ１８４のみを含み得るか、又は、スレッド・プログラム・カウンタの、ロック・カウンタ１８８及び関数呼び出し深さ１８６の１つ若しくは両方との組合せを含み得る）。このプロセスは、各スレッドに関連するスレッド・プログラム・カウンタ１８４を変更させる場合があり、したがって、次にプロセスがステップＳ４００から繰り返され、異なる汎用プログラム・カウンタ値１２０を決定させ得る。

【００５３】

図４に関して述べられた実例では、プロセッサは楽観的にワープ内のすべてのスレッドについてロックの取得を試み、いずれのスレッドがロックを取得することができたかに基づいて、実行優先順位を並べ替える。

【００５４】

第２の実施例が、図７～９に関して説明される。この実施例では、一度に、群内のたった１つのスレッドが、ロックを保持することを許される。また、図４では、プロセッサが楽観的に、ＳＴＸＲ命令を含むロッキング・シーケンスを使用して、すべてのスレッドについてロックの取得を試み、次に、それが成功した場合のみ、ロック・カウントをインクリメントして、スレッドがロッキング特権を獲得したことを知らせるが、図７～９では、順番が逆にされ、最初に、ロッキング特権を要求し、それに従ってロック・パラメータを

更新するように命令が実行される。この時点で、プロセッサは、単に１つのスレッドが、ロッキング特権を要求する命令を成功のうちに実行できるようにする（他のスレッドについては、この命令の実行は失敗し、そのスレッド用のスレッド・プログラム・カウンタはインクリメントされない）ことを確実にする。ロッキング特権を獲得すると、成功したスレッドは次に、ロックを取得するために、ロッキング・シーケンスを実行する。この方法では、群内のたった１つのスレッドが、任意の時点でロックを保持することを許される。アンロッキング・シーケンスは、図４の場合と同じままであり、最初にロックを解放し、次にロッキング特権を解放する。ロックは、ロック特権を保持している間、単一のスレッドにおいて複数回、要求可能で、スレッドによって保持されたロックの数は、ロック・カウンタを使用して追跡され得る。スレッド毎に１つのロック・カウンタを有し、より高いロック・カウントを優先させるプログラム・カウンタ選択機構を備えた第１の実施例と同じ方法で、第２の実施例を実施することが可能である一方で、ロッキング特権（実行の優先権）が、一度に、群毎に単一のスレッドに与えられることから、群内のいずれかのスレッドがロッキング特権を有しているかどうかの単一の表示、このスレッドの識別、及びその群のための単一のロック・カウンタを、単に群全体について格納することにより、このスキームを簡略化することが可能である。

【００５５】

ここで、図７は、この第２の実施例で使用されるレジスタ・ファイル１８０及び状態データの一実例を示す。図７は、ロック・カウンタ１８８が各スレッドに提供されてない点で、図１のレジスタ・ファイルとは異なる。その代わりに、スレッドの群のための汎用プログラム・カウンタ１２０に加えて、多くのロック・パラメータが、そのスレッドの群の間で共有されている。

・群内のいずれかのスレッドが１つ又は複数の共有リソースをロックしているかどうかを示すロック設定フラグ５００

・群のどのスレッドが１つ又は複数の共有リソースをロックしているかを示すロック所有者パラメータ５０２

・いくつのリソースが、ロック所有者パラメータ５０２によって示されたロック所有スレッドによってロックされているかを示すロック・カウント・パラメータ５０４

いくつかの実例では、ロック設定フラグ５００は省かれてもよく、その代わりに、ロック所有者パラメータ５０２又はロック・カウント・パラメータ５０４が、スレッドがロックを保持しているか否かを知らせるために使用され得る。たとえば、ロック・カウント・パラメータがゼロである場合、これがロックを保持しているスレッドがないことの知らせであり得、ロック・カウント・パラメータが非ゼロ値を有する場合、これがスレッドがロックを保持していることの知らせであり得る。あるいは、ロック所有者パラメータ５０２は、スレッドの１つがロックを保持している場合、そのスレッドに対応する値を有することができ、ロックを保持しているスレッドがない場合、スレッドのいずれにも対応しない値を有することができ、それによって、これが、ロックが設定されているか否かを効果的に示すことができる。しかしながら、ロックが設定されているか否かを示す別個のロック設定フラグ５００を提供することは、たとえば、ロック設定フラグ５００が、ロック所有者パラメータ５０２又はロック・カウント値５０４に対応するマルチ・ビット値よりも、確認するのにより早く且つエネルギー効率がよいシングル・ビットを含み得ることから、時には有用であり得る。

【００５６】

図８は、以下のように、第２の実施例でのコードの実例を示す。

ライン０：ロッキング・シーケンスの開始を表すラベルで、プロセッサにおける何の特定の動作を引き起こさない。

ライン１：inc_lock_countは、ロッキング特権（スレッドがリソースに対して、１つ又は複数のロックを設定する権利）を要求する命令である。この命令は、群内の１つのスレッドについてのみ、成功のうちに実行され得る。これにより、この命令に引き続いて、１つのスレッドが（このスレッドのプログラム・カウンタを次の命令にインクリメントす

ることで）次の命令に進むことができる一方、他のスレッドは、この命令に失敗する（それによって、それらのスレッド・プログラム・カウンタは、それを成功のうちに実行するまで、同じ命令inc_lock_countに留まる）。inc_lock_countが成功のうちに実行されると、スレッドがロッキング特権を保持していることを示すためにロック設定フラグ５００を設定し、現在のスレッドの識別を示すためにロック所有者パラメータ５０２を設定し、さらにロック・カウント５０４をインクリメントする。

ライン２：ロッキング特権を有するスレッドがロックを得ることができなかった場合に分岐する先であるラベルretry_lock。

ライン３～５：図４のライン１～３と同じで、別のプロセス（たとえば、現在のスレッドの群の外のスレッド、又はマルチプロセッサ・システムにおける異なるプロセッサ上で実行されるプロセスなど）が、レジスタｘ１９内のアドレスによって特定されたリソースに対して、すでにロックを得ているか否かを確認する。

ライン６：ｘ１９内で特定されたアドレスに対して別のプロセスがロックを保持している場合にライン１０～１２における「バックオフ」シーケンスに分岐する、条件付き分岐。

ライン７～８：図４のライン５及び７と同じ。レジスタｘ１９内のアドレスに対してロックを設定するストア排他命令、及びストア排他命令が失敗した場合にライン２に後方分岐する条件付き分岐。図４のライン６とは異なり、図８においてロック・カウントがライン１ですでにインクリメントされていることから、ロックが成功のうちに取得された場合にロック・カウントをインクリメントする条件付き命令はない。

ライン９：ロックが成功のうちに取得されたとき、ロックされたリソースを使用して何かを行うためにライン１３に分岐する。

ライン１０：レジスタｘ１９内のアドレスによって特定されたリソースに対するロックがすでに設定されている場合に、ロッキング特権を放棄するためのバックオフ・シーケンスの開始を表すラベル「backoff_lock」。ロッキング特権を放棄することによって、レジスタｘ１９のそのバージョンにある異なるアドレスを有し得る別のスレッドが、そのロックを成功のうちに取得し、進行することが可能であり得る。

ライン１１：ロック・カウント５０４をデクリメントする。この結果によりロック・カウント５０４がゼロになるとロック設定フラグ５００も消去し、別のスレッドがロッキング特権を得ることができる。ライン１１の後、同じ群の他のスレッドがロックを使用したその処理を終了し、且つロックを解放する時間を許すために、スレッドはしばらく待機することができる。

ライン１２：ロックの取得を再度試みるために、ライン０におけるロッキング・シーケンスの開始に後方分岐する。

ライン１３：ロックが成功のうちに取得された場合に実行されるシーケンスの開始を表すラベル「lock_acquired」。これに続いて、任意の個数の命令がロックされたリソースを使用して実行され得る。

ライン１４：ロックされたリソースを使用して処理が完了すると、アンロッキング・シーケンスを開始する。

ライン１５～１７：この場合には、dec_lock_count命令が実行され、その結果としてロック・カウント５０４がゼロになるとロック設定フラグ５００が消去されることを除き、ロックを放棄する図４のライン１０～１２と同じ。

【００５７】

したがって、この実例では、デッドロックに対するさらなる保護を提供するために、一度に、同じスレッドの群内でただ１つのスレッドのみがロックを保持し得る。

【００５８】

図９は、第２の実施例を使用する場合の選択されたスレッド（そして、スレッドの群について汎用プログラム・カウンタに設定するスレッド・プログラム・カウンタ）を選び出す方法を示す。その群内でただ１つのスレッドが一度にロックを保持することができることから、プログラム・カウンタ選択プロセスの最初の部分は、より単純になる。ステップ

６００において、セレクタ１１０はロック設定フラグ５００がその群のいずれかのスレッドがロックを保持しているかどうかを示しているかを決定する。そうであれば、ステップ６０２において、セレクタ１１０は、ロック所有者パラメータ５０２によって示されたスレッドを選択されたスレッドとして選び出す。

【００５９】

一方、ロック設定フラグ５００が、その群のどのスレッドも現在、ロックを保持していないことを示している場合、ステップ６０４において、その群（部分集合）のすべてのスレッドが、スレッドの「第１の集合」の一部であると見なされる。そして当該方法は、図５のステップＳ２２５に進み、その後、方法は図５と同様に進行し、関数呼び出し深さ及びスレッド・プログラム・カウンタに基づき、選択されたスレッドを選び出す。

【００６０】

選択されたスレッドを選び出すと、図６のステップＳ４１０及びＳ４２０が、上述のように適用され得る。

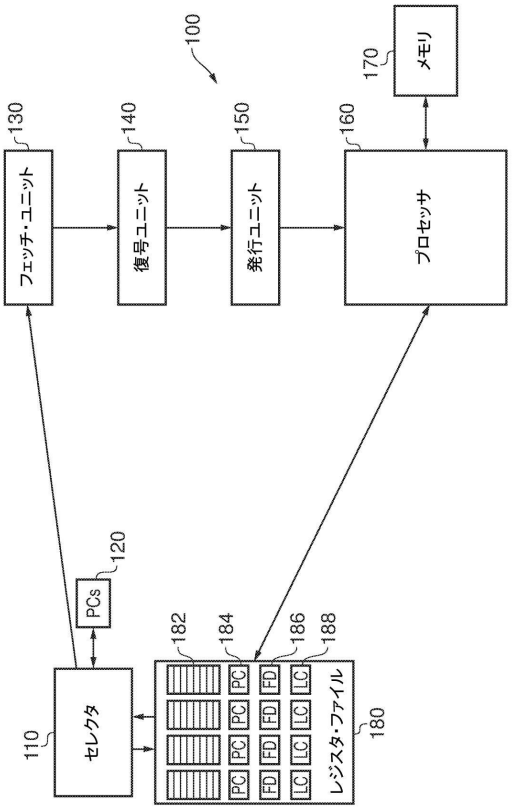
【００６１】

本出願において、「…するように構成される（…configured to）」という用語は、装置の要素が規定のオペレーションを行うことのできる構成を有することを意味するために使用される。これの文脈で、「構成（configuration）」は、ハードウェア若しくはソフトウェアの相互接続の配置又は方法を意味する。たとえば、装置は、規定のオペレーションを提供する専用のハードウェアを備え得るか、又はプロセッサ若しくは他の処理デバイスが機能を果たすようにプログラムされ得る。「するように構成される（configured to）」は、装置の要素が、規定のオペレーションを提供するために何らかの方法で変更される必要のあることを意味するものではない。

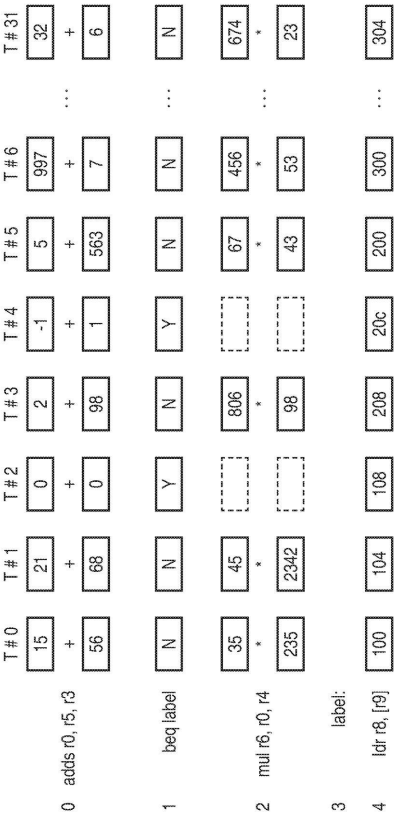
【００６２】

本発明の例示的な実施例が、本明細書において添付の図面を参照して詳細に説明されてきたが、本発明はそれらのまさにその実施例に限定されず、様々な変更及び修正が、添付の特許請求の範囲によって定められた本発明の範囲及び趣旨から逸脱することなく、当業者によって本明細書にもたらされ得ることが理解されるべきである。

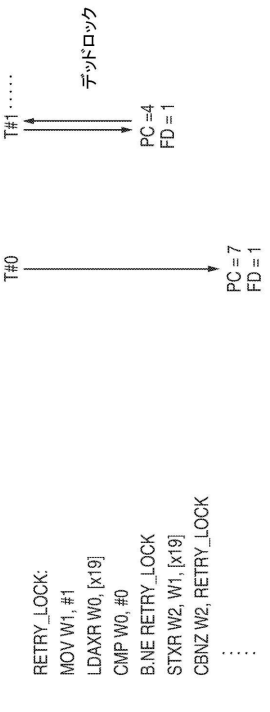
【図１】


100
フェッチ・ユニット
130
復号ユニット
140
発行ユニット
150
プロセッサ
160
メモリ
170
PCs
120
セレクタ
110
182
184
186
188
レジスタ・ファイル
180
PC
FD
LC

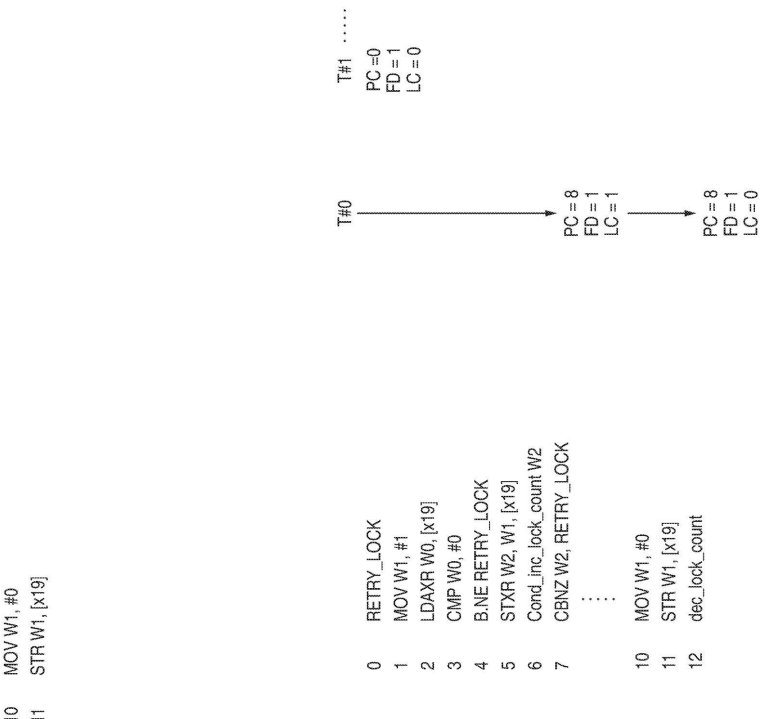
【図２】


T#0
T#1
T#2
T#3
T#4
T#5
T#6
T#31
0 adds r0, r5, r3
1 beq label
2 mul r6, r0, r4
3 label:
4 ldr r8, [r9]

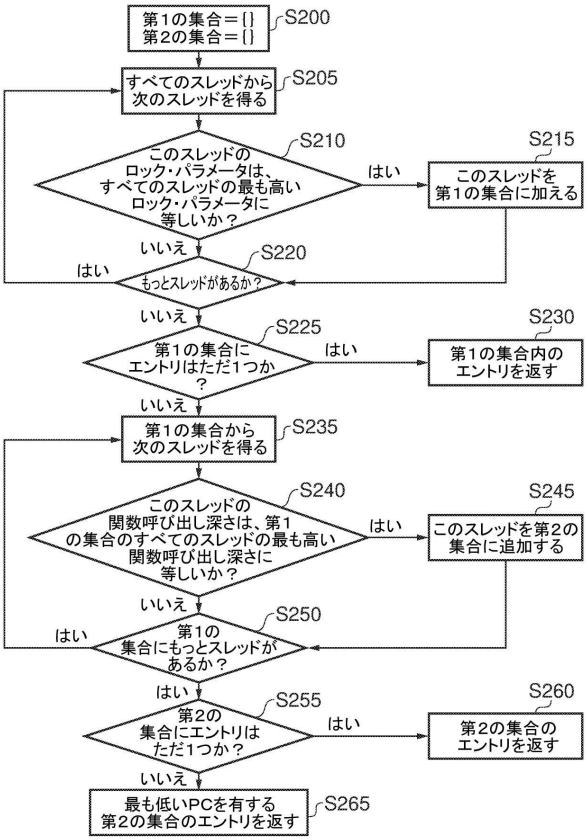
【図３】


T#0
T#1
デッドロック
PC =4
FD = 1
PC = 7
FD = 1
0 RETRY_LOCK:
1 MOV W1, #1
2 LDAXR W0, [x19]
3 CMP W0, #0
4 B.NE RETRY_LOCK
5 STXR W2, W1, [x19]
6 CBNZ W2, RETRY_LOCK
10 MOV W1, #0
11 STR W1, [x19]

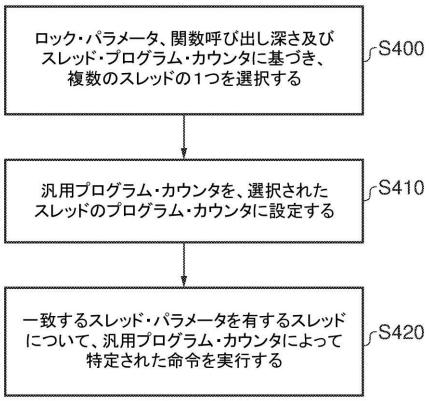
【図４】


T#0
T#1
PC =0
FD = 1
LC = 0
PC = 8
FD = 1
LC = 1
PC = 8
FD = 1
LC = 0
0 RETRY_LOCK
1 MOV W1, #1
2 LDAXR W0, [x19]
3 CMP W0, #0
4 B.NE RETRY_LOCK
5 STXR W2, W1, [x19]
6 Cond_inc_lock_count W2
7 CBNZ W2, RETRY_LOCK
10 MOV W1, #0
11 STR W1, [x19]
12 dec_lock_count

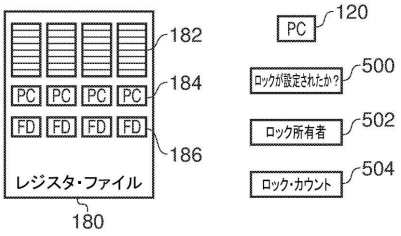
【図５】



【図６】



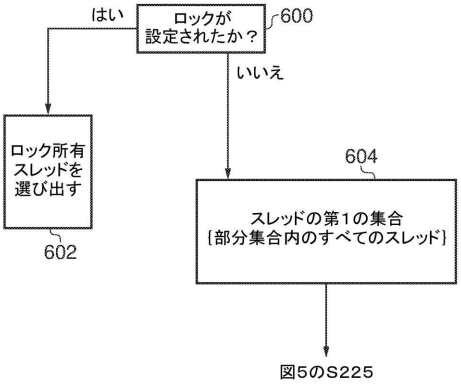
【図７】



【図８】

```
0     locking_sequence_start:
1     inc_lock_count
2     retry_lock:
3     mov w1, #1
4     ldaxr w0, [x19]
5     cmp w0, #0
6     bne backoff_lock
7     stxr w2, w1, [x19]
8     cbnz w2, retry_lock
9     br lock_acquired


10    backoff_lock:
11    dec_lock_count
      #スレッドの部分集合内の別のスレッドが異なるロックを欲し、
      進行する可能性がある場合、ロック・カウントを減らす

...#ロックを保持しているスレッドが終了しロックを解放することを期待して、
しばらく待機する

12    br locking_sequence_start
      #ロック・カウントを増やすことを含む全ロッキング・シーケンスを再開する

13    lock_acquired

...#ロックされたデータで何かを行う

14    release_lock:
15    mov w1, #0
16    str w1, [x19]
17    dec_lock_count
```

【図９】



フロントページの続き

(72)発明者　マンセル、デイヴィッド　ヘナ
イギリス国、ケンブリッジシャー、ケンブリッジ、チェリー　ヒントン、フルボーン　ロード　１０、エイアールエム　リミテッド　気付

審査官　大桃　由紀雄

(56)参考文献　米国特許出願公開第２０１４／０１８９２６０（ＵＳ，Ａ１）
特開２００９－２３０７５７（ＪＰ，Ａ）
特表２０１１－５０５６４７（ＪＰ，Ａ）
米国特許出願公開第２０１４／００７５１６５（ＵＳ，Ａ１）
米国特許第０７０１５９１３（ＵＳ，Ｂ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／５２
Ｇ０６Ｆ　　９／４６