(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5576798号
(P5576798)
(45)発行日　平成26年8月20日(2014.8.20)　(24)登録日　平成26年7月11日(2014.7.11)

(51)Int.Cl.　　F I
G06F 9/45 (2006.01)　　G06F 9/44 322A
G06F 9/46 (2006.01)　　G06F 9/46 410

請求項の数 35　(全 32 頁)

(21)出願番号　特願2010-538213 (P2010-538213)
(86)(22)出願日　平成20年12月12日 (2008.12.12)
(65)公表番号　特表2011-507112 (P2011-507112A)
(43)公表日　平成23年3月3日 (2011.3.3)
(86)国際出願番号　PCT/US2008/086711
(87)国際公開番号　WO2009/076654
(87)国際公開日　平成21年6月18日 (2009.6.18)
審査請求日　平成23年12月12日 (2011.12.12)
(31)優先権主張番号　61/013,019
(32)優先日　平成19年12月12日 (2007.12.12)
(33)優先権主張国　米国(US)

(73)特許権者　500093498
ユニバーシティ・オブ・ワシントン
アメリカ合衆国　ワシントン州　シアトル　スイート　５００　イレブンズ　アベニュー　エヌイー　４３１１
(74)代理人　100140109
弁理士　小野　新次郎
(74)代理人　100089705
弁理士　社本　一夫
(74)代理人　100075270
弁理士　小林　泰
(74)代理人　100080137
弁理士　千葉　昭男
(74)代理人　100096013
弁理士　富田　博行

最終頁に続く

(54)【発明の名称】決定論的マルチプロセッシング（DETERMINISTICMULTIPROCESSING）

(57)【特許請求の範囲】
【請求項１】
マルチプロセッシングシステムにおけるマルチスレッドアプリケーションの決定論的実行を提供するために、前記マルチスレッドアプリケーションを増補するコンピューティングシステムにおける方法であって、
２つ以上のスレッドの実行を指定するマルチスレッドアプリケーションコードにアクセスするステップと、
前記マルチスレッドアプリケーションコードに同期コードを自動的に挿入するステップであって、毎回、前記マルチスレッドアプリケーションコードが、特定の入力によって実行される度に、当該同期コードが、操作の組の中の決定論的順序を指定し、当該組の少なくとも１つが、前記マルチスレッドアプリケーションコードが実行されるとき、前記２つ以上のスレッドのうちの少なくとも１つの別のスレッドによってアクセス可能な状態に影響を与えることができる１つ又は複数の操作を含むものと、
を含む方法。
【請求項２】
前記決定論的順序が、前記２つ以上のスレッドが作成された順序である請求項１に記載の方法。
【請求項３】
前記決定論的順序がトークンの値に従って決定されており、
前記２つ以上のスレッドの各スレッドについて、前記２つ以上のスレッドのうちの少な

くとも１つによってアクセス可能な状態に影響を与えることができる操作を実行する前に、
　前記トークンの前記値を決定するために、前記同期コードを呼び出すステップと、
　前記トークンの前記決定された値が前記スレッドのスレッド識別子に一致するとき、前記操作の実行を可能にするステップと、
　前記トークンの前記決定された値が前記スレッドのスレッド識別子に一致しないとき、前記スレッドの実行を一時停止するステップと
　をさらに含む請求項１に記載の方法。
【請求項４】
　コンパイラによって実行される請求項１に記載の方法。
【請求項５】
　トランザクショナルメモリシステムを増補するためのコンピューティングシステムにおける方法において、
　トランザクショナルメモリシステムのためのコードにアクセスするステップであって、前記コードがマルチスレッドアプリケーションソースコードからコンパイルされたコードによって呼び出されるインターフェイスの１つまたは複数の実装を含み、前記マルチスレッドアプリケーションソースコードが１つまたは複数のコードブロックをアトミックブロックと宣言し、前記マルチスレッドアプリケーションソースコードが２つ以上のスレッドを指定する、ステップと、
　同期コードを含むために前記アクセスされたコードを増補するステップであって、当該同期コードが、特定の入力により前記マルチスレッドアプリケーションソースコードがマルチプロセッシングシステムによって実行される度に毎回、トランザクションがコミットするような特定の順序を指定するものであるもの、と、
　を含む方法。
【請求項６】
　メモリ操作の順序を制御するためのマルチプロセッシングシステムで実行される方法であって、
　マルチプロセッシングシステムにおいてマルチスレッドアプリケーションコードを実行するステップであって、前記マルチスレッドアプリケーションコードが複数のスレッドを指定する、ステップと、
　前記マルチスレッドアプリケーションコードの前記実行を２つ以上の量子に分割するステップであって、各量子がメモリ操作を含む決定論的数の操作を指定する、ステップと、
　前記複数のスレッドが前記２つ以上の量子を実行する決定論的順序を指定するステップと
　を含み、前記マルチスレッドアプリケーションコードが実行されるとき、メモリ操作を指定するスレッド間通信が決定論的である。
【請求項７】
　前記複数のスレッドのうちの少なくとも１つのスレッドが前記複数のスレッドの別のスレッドによってプライベートに保持されるデータをロードするとき、前記スレッド間通信が行われる請求項６に記載の方法。
【請求項８】
　スレッドが別のスレッドによってプライベートに保持されるデータをロードしようと試行するとき、前記複数のスレッドのそれぞれがその実行における決定論的ポイントに到達し、前記スレッドが実行を始めることを前記決定論的順序が指定するまで、前記スレッドの実行を一時停止するステップをさらに含む請求項７に記載の方法。
【請求項９】
　前記複数のスレッドのうちの１つのスレッドが前記１つのスレッドによってプライベートに保持されないデータを格納するとき、前記スレッド間通信が行われる請求項６に記載の方法。
【請求項１０】

スレッドが前記スレッドによってプライベートに保持されないデータを格納しようと試行するとき、前記複数のスレッドのそれぞれがその実行における決定論的ポイントに到達し、前記スレッドが実行を始めることを前記決定論的順序が指定するまで、前記スレッドの実行を一時停止するステップをさらに含む請求項９に記載の方法。

【請求項１１】

決定論的順序を指定するステップが、

マルチスレッドアプリケーションコードの中に同期コードを挿入するステップを含み、

前記挿入された同期コードが、１つ又はそれより多いロック又は共有テーブルを実行して、スレッド間通信をモニタする、

請求項６に記載の方法。

【請求項１２】

前記マルチプロセッシングシステムが、トランザクショナルメモリシステムを含み、

決定論的順序を指定するステップが、マルチスレッドアプリケーションコードの中に同期コードを挿入するステップを含み、

同期コードを挿入するステップが、各量子をトランザクション内に封入する（encapsulating）ステップを含み、

前記トランザクショナルメモリシステムが、各トランザクションを、指定された決定論的順序でコミットする、

請求項６に記載の方法。

【請求項１３】

前記決定論的順序がトークンの値に従って指定されており、

前記複数のスレッドごとに、メモリ操作を実行する前に、

前記トークンの前記値を決定するステップと、

前記トークンの前記決定された値が前記スレッドのスレッド識別子に一致するとき、前記複数のスレッドの１つおきのスレッドがその実行における決定論的ポイントに到達すると、前記メモリ操作の実行を可能にするステップと、

前記トークンの前記決定された値が前記スレッドの識別されたスレッドに一致しないとき、前記スレッドの実行を一時停止するステップと

をさらに含む請求項６に記載の方法。

【請求項１４】

マルチスレッドアプリケーションのスレッドのインターリービングを制御するためのマルチプロセッシングシステムであって、

複数のスレッドを指定するマルチスレッドアプリケーションコードと、

前記マルチスレッドアプリケーションコードを、決定論的数の操作をそれぞれ指定する２つ以上の量子に分割するための量子ビルダコンポーネントと、

前記マルチスレッドアプリケーションのスレッドが前記２つ以上の量子を実行する決定論的順序を指定するための決定論的コンポーネントと

を含み、前記マルチスレッドアプリケーションコードの複数の実行中に特定の入力が指定されたとき、各実行が前記特定の入力について同じ出力を生成する

マルチプロセッシングシステム。

【請求項１５】

前記指定された決定論的数の操作内に、制御された操作として指定された特定の操作のみを含めることによって、前記量子ビルダコンポーネントが前記マルチスレッドアプリケーションコードの実行を選択的にシリアル化する請求項１４に記載のシステム。

【請求項１６】

請求項１５に記載のシステムであって、制御された操作として指定された操作が、メモリ操作を含み、

前記複数のスレッドのうちの１つのスレッドが、前記複数のスレッドのうちの別のスレッドの状態に影響を与えることができるメモリ操作を実行するとき、前記メモリ操作が前記決定論的コンポーネントによって指定された前記決定論的順序を侵害するかどうかを前

記量子化ビルダコンポーネントが決定し、
前記メモリ操作が前記決定論的順序を侵害するとき、前記複数のスレッドの各スレッドがその実行における決定論的ポイントに到達し、前記スレッドが続行することを前記決定論的コンポーネントが指定するまで、前記マルチプロセッシングシステムが前記メモリ操作の実行を一時停止し、
前記メモリ操作が前記決定論的順序を侵害しないとき、前記マルチプロセッシングシステムが前記メモリ操作の実行を許可する、
請求項１５に記載のシステム。
【請求項１７】
前記スレッドが別のスレッドによってプライベートに保持されると見なされるデータをロードまたは格納することを前記メモリ操作が指定するとき、
前記スレッドが前記複数のスレッドによって共有されると見なされるデータを格納することを前記メモリ操作が指定するとき、又は、
前記スレッドが前記複数のスレッドのうちの任意のものによってこれまでアクセスされていないデータをロードまたは格納することを前記メモリ操作が指定するとき、
前記メモリ操作が前記決定論的順序を侵害する請求項１６に記載のシステム。
【請求項１８】
前記マルチプロセッシングシステムが各スレッドの実行を一時停止したとき、又は、各スレッドが量子の実行を終了すると、
前記複数のスレッドの各スレッドがその実行における決定論的ポイントに到達する請求項１６に記載のシステム。
【請求項１９】
同期コードを前記マルチスレッドアプリケーションコード内に挿入することによって、前記量子ビルダコンポーネントが前記マルチスレッドアプリケーションコードを２つ以上の量子に分割する請求項１４に記載のシステム。
【請求項２０】
前記挿入された同期コードが１つまたは複数のロックを含む、又は、メモリ操作を追跡するための共有テーブルを含む、請求項１９に記載のシステム。
【請求項２１】
トランザクショナルメモリシステムをさらに含み、前記挿入された同期コードが各量子をトランザクション内に封入し、各トランザクションが前記決定論的コンポーネントによって指定された前記決定論的順序でコミットされる請求項１９に記載のシステム。
【請求項２２】
前記トランザクションが並行して実行され、２つ以上の並行して実行されるトランザクション間に競合が存在するとき、前記トランザクションのうちの少なくとも１つが、前記決定論的順序に従って中止され、再開される、請求項２１に記載のシステム。
【請求項２３】
前記含まれるトランザクショナルメモリシステムがハードウェアトランザクショナルメモリシステム、ソフトウェアトランザクショナルメモリシステム、ハイブリッドハードウェアーソフトウェアトランザクショナルメモリシステム、およびトランザクショナルメモリシステムの組み合わせを含むグループから選択される請求項２１に記載のシステム。
【請求項２４】
前記決定論的順序が、前記複数のスレッドのそれぞれが作成された順序に基づいて指定される請求項１４に記載のシステム。
【請求項２５】
前記決定論的順序が、前記マルチスレッドアプリケーションコードのソフトウェア開発者によって前記決定論的コンポーネントに対して指定される請求項１４に記載のシステム。
【請求項２６】
マルチプロセッシングシステムに、マルチスレッドアプリケーションのスレッドによっ

て実行されるメモリ操作の順序を制御させることができるコードを格納するコンピュータ可読記憶媒体において、前記コードが、
マルチスレッドアプリケーションコードを複数の量子に分割するためのコードであって、各量子が決定論的な有限数のメモリ操作を指定する、コードと、
各量子を、前記マルチスレッドアプリケーションによって指定された２つ以上のスレッドのうちの１つによって決定論的にコミットされるトランザクション内に封入するためのコードと
を含み、前記マルチプロセッシングシステムがトランザクショナルメモリシステムと共に動作する
コンピュータ可読記憶媒体。
【請求項２７】
トランザクション内に封入される各量子が、前記２つ以上のスレッドが作成される順序に従って前記２つ以上のスレッドのうちの一方によって決定論的にコミットされる請求項２６に記載のコンピュータ可読記憶媒体。
【請求項２８】
請求項２６に記載のコンピュータ可読記憶媒体であって、
２つ以上の多いスレッドによってトランザクションがコミットされる前に、更に、１つのスレッドの実行を一時停止するために、
前記２つ以上の多いスレッドの各々が、その実行中に、決定論的ポイントに到達し、そして、トークンが、前記スレッドが、前記トランザクションをコミットすべきことを指定するまで、コードを格納する、コンピュータ可読記憶媒体。
【請求項２９】
各スレッドがトランザクションの実行を完了すると、各スレッドがその実行における決定論的ポイントに到達する請求項２８に記載のコンピュータ可読記憶媒体。
【請求項３０】
前記マルチスレッドアプリケーションコードが、前記マルチスレッドアプリケーションコードのソフトウェア開発者によって指定された１つまたは複数のトランザクショナルメモリブロックを含み、前記コードが前記１つまたは複数のトランザクショナルメモリブロックを増補するためにさらに使用される請求項２６に記載のコンピュータ可読記憶媒体。
【請求項３１】
操作の組の中の決定論的順序が、特定のレジスタに依存すること無しに決定される、請求項１に記載の方法。
【請求項３２】
トランザクションがコミットされる特定の順序が、特定のレジスタに依存すること無しに決定される、請求項５に記載の方法。
【請求項３３】
複数のスレッドが実行される決定論的順序が、特定のレジスタに依存すること無しに決定される、請求項６に記載の方法。
【請求項３４】
特定のレジスタに依存すること無しに、前記決定論的コンポーネントが、マルチスレッドアプリケーションのスレッドが２つ以上の量子を実行する決定論的順序を指定する、請求項１４に記載のシステム。
【請求項３５】
決定論的にコミットする各量子が、特定のレジスタに依存することが無い、請求項２６に記載のコンピュータ可読記憶媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本願発明の実施例は、例えば、決定論的マルチプロセッシングに関する。
【背景技術】

【０００２】
本出願は、参照により本明細書に組み込まれる２００７年１２月１２日出願の「ＤＥＴＥＲＭＩＮＩＳＴＩＣ　ＭＵＬＴＩＰＲＯＣＥＳＳＩＮＧ（決定論的マルチプロセッシング）」という名称の米国特許仮出願第６１／０１３，０１９号の利益を主張する。
【０００３】
マルチプロセッシングは、２つ以上の処理装置がそれぞれ１つまたは複数のプロセス（プログラムまたは命令のセット）を同時に実行する動作モードである。マルチプロセッシングシステムの目的は、処理速度を向上させることである。通常、これは、各処理装置が同じプロセスの異なる命令のセットまたは異なるスレッドを処理することによって達成される。プロセスは、１つまたは複数のスレッドを実行することができる。各スレッドは、それ自体のプログラムコンテキストを含むそれ自体のプロセッサコンテキストを有する。従来、アプリケーションがマルチプロセッシングの利益を利用するためには、ソフトウェア開発者は、アプリケーションをマルチスレッド型で書く必要がある。本明細書で使用する場合、マルチスレッドアプリケーションとは、２つ以上のスレッドを同時に実行することができるプログラムを指す。
【０００４】
マルチプロセッサまたはマルチコアシステム（まとめて「マルチプロセッシングシステム」と呼ぶ）において、マルチスレッドアプリケーションの２つ以上のスレッドは同時に実行することができ、各プロセッサまたはコアが特定のスレッドを実行する。マルチスレッドアプリケーションのスレッドが、同時実行中に、例えばメモリなどリソースを共有することは、一般的である。本明細書で使用される場合、同時実行とは、マルチスレッドアプリケーションの２つ以上のスレッドの同時の実行を指す。同時実行の結果、マルチスレッドアプリケーションの２つ以上のスレッドが同じ共有リソースを読み取り、および／または更新することができる。例えば、あるスレッドは、共有メモリロケーションの値を変更することができ、一方、別のスレッドは、共有メモリロケーションに格納された値に応じて、一連の操作を実行する。
【０００５】
従来のソフトウェア開発モデル下で、ソフトウェア開発者は、そのマルチスレッドアプリケーション内の並列スレッドを識別し、正しく同期しようと試みることにかなりの時間量を費やす。例えば、開発者は、ロック、セマフォ、バリア、または他の同期機構を明示的に使用して、共有リソースへのアクセスを制御することができる。スレッドが共有リソースにアクセスするとき、同期機構は、リソースが使用可能になるまで、これらのスレッドを一時停止することによって、他のスレッドがリソースにアクセスするのを防ぐ。同期機構を明示的に実装するソフトウェア開発者は、通常、同期コードをデバッグするのにもかなりの時間量を費やす。しかし、通常、同期エラーに起因するソフトウェアの欠陥（バグとも呼ばれる）が一時的に表面化する（すなわち、インターリーブされたスレッド操作の特定のシーケンスにのみバグが現れる可能性がある）。その結果、欠陥のあるソフトウェアは、小さな同期バグが現れる前に、何百回も正しく実行する可能性がある。
【０００６】
こうしたシステムにおけるスレッドの様々なインターリービングによって非決定論的挙動が作り出されるため、マルチプロセッシングシステムのためのソフトウェアを開発することは難しい。インターリービングとは、スレッド間の対話を含み得るスレッド操作の順序を指す。スレッド間の可能なインターリービングの数は、スレッドの数が増すにつれて、著しく増す。その結果、マルチスレッドアプリケーションは、誤り検出およびモデリングプログラムの挙動に関して、追加の問題を提示する。例えば、マルチスレッドアプリケーションに同じ入力が与えられると、マルチプロセッシングシステムは、スレッド操作を非決定論的にインターリーブし、それによって、マルチスレッドアプリケーションが実行されるたびに異なる出力を生成する。図１は、マルチプロセッシングシステムにおいて実行されるマルチスレッドアプリケーションにおける２つの可能なスレッドインターリービングの例を示す高レベル図である。図示されるように、アプリケーションは、少なくとも

２つのスレッド、スレッド１およびスレッド２を含む。アプリケーションが呼び出されると、ある時点で、スレッド１は、変数Ａの値を１に設定する（Ａ＝１）操作、次いで変数Ｂの値を変数Ａの値に設定する（Ｂ＝Ａ）操作を実行し、スレッド２は、変数Ｂの値をゼロに設定する（Ｂ＝０）操作、次いで変数Ａの値を変数Ｂの値に設定する（Ａ＝Ｂ）操作を実行する。図示されるように、スレッド１およびスレッド２の操作は、非決定論的にインターリーブされ、それによって、アプリケーションが呼び出されるたびに、異なる出力を生成する。すなわち、最初に示された呼び出し中、操作のインターリービングによって変数ＡおよびＢがそれぞれゼロに設定され、２番目に示された呼び出し中、操作のインターリービングによって変数ＡおよびＢがそれぞれ１に設定された。

【０００７】

マルチスレッド実行における非決定性は、例えば、他のプロセスが同時に実行する、オペレーティングシステムリソースの割り当てにおける差、キャッシュの状態、トランスレーションルックアサイドバッファ（「ＴＬＢ」）、バス、割り込み、および他のマクロアーキテクチャ構造など、実行環境におけるわずかな変化に起因し得る。

【発明の概要】

【発明が解決しようとする課題】

【０００８】

その結果、マルチスレッドアプリケーションを開発することは、シングルスレッドアプリケーションを開発するよりかなり難しい。

【課題を解決するための手段】

【０００９】

従来、この問題に対処するにあたっての取り組みは、以前生成されたログファイルに基づいてマルチスレッド実行を決定論的に再生することに焦点を当てていた。しかし、決定論的再生システムは、再生ログファイルの維持に伴うオーバーヘッドの結果、かなりの性能の低下を受ける。さらに、決定論的再生では、ソフトウェア開発者は、スレッドのインターリービングがどのように実行されるかを制御しない。その結果、ソフトウェアが顧客に配布される前に、操作の特定のインターリービングに起因する同期バグは、識別（およびより重要には修正）されない場合がある。非決定性は、テストカバレージを評価するのを難しくする点で、ソフトウェア開発プロセスをさらに複雑にする。良好なカバレージは、広範なプログラム入力と、広範な可能なスレッドインターリービングとを必要とする。

【００１０】

ファシリティの１つまたは複数の実施形態が、添付の図面の図に例として、かつ限定されないものとして示されている。図中、参照番号は、類似の要素を示す。

【図面の簡単な説明】

【００１１】

【図１】マルチスレッドプログラムにおける、２つの可能なスレッドインターリービングの一例を示す高レベル図である。

【図２】１つまたは複数の実施形態における、ファシリティによって実行される決定論的シリアル化プロセスのフロー図である。

【図３】１つまたは複数の実施形態における、ファシリティによって実行される決定論的選択的シリアル化プロセスのフロー図である。

【図４】１つまたは複数の実施形態における、ファシリティが実行するコンピューティングシステムのアーキテクチャ例を示す高レベルブロック図である。

【図５】１つまたは複数の実施形態における、決定論的マルチプロセッシングレイヤの様々な機能的要素を示す高レベルブロック図である。

【図６】１つまたは複数の実施形態における、マルチプロセッサコードを決定論的にするためにファシリティによって使用されるデータ構造を示す高レベルブロック図である。

【図７】１つまたは複数の実施形態における、スレッドを作成し、決定論的に実行する一例を示す高レベル図である。

【図８】１つまたは複数の実施形態における、マルチプロセッサコードを決定論的にする

ためにトランザクショナルメモリシステムを使用する一例を示す高レベルブロック図である。
【図９】１つまたは複数の実施形態における、アプリケーションを増補（augment）するためにファシリティによって実行されるプロセスを示すフロー図である。
【図１０】１つまたは複数の実施形態における、ブロックを構文解析するためにファシリティによって実行されるプロセスを示すフロー図である。
【図１１】１つまたは複数の実施形態における、マルチスレッドアプリケーションの増補された機能の制御フローグラフの一例である。
【図１２】１つまたは複数の実施形態における、決定論的マルチプロセッシング初期化関数を示すフロー図である。
【図１３】１つまたは複数の実施形態における、決定論的マルチプロセッシングコミット関数を示すフロー図である。
【発明を実施するための形態】
【００１２】
決定論的再生システムなどの従来のシステムは、マルチスレッドアプリケーションの開発における非決定論的挙動に伴う問題を適切に解決しない。さらに、既存のシステムは、マルチスレッドアプリケーションの配置における非決定論的挙動に伴う問題を低減することも、解決しようと試みることもない。したがって、マルチスレッドアプリケーションの決定論的マルチプロセッシングのためのハードウェアおよび／またはソフトウェアファシリティ（「ファシリティ」）が開発された。本明細書で使用される場合、決定論的マルチプロセッシングという用語は、マルチスレッドアプリケーションに同じ入力が与えられると、マルチスレッドアプリケーションによって同じ出力が生成される技術を指す。例えば、共有リソースへのスレッドアクセスを同期するための負担から開発者を解放することによって、ファシリティは、マルチスレッドアプリケーションを開発する処理を簡略化する。さらにファシリティは、こうしたマルチスレッドアプリケーションが配置されるとき、例えば、開発者がバグを再生し、様々なスレッドインターリービングを厳格にテストできるようにすることによって、マルチスレッドアプリケーションの信頼性を向上させる。
【００１３】
いくつかの実施形態において、ファシリティは、マルチスレッドアプリケーションの実行を決定論的な有限数の操作の組（各組は、本明細書では「量子」と呼ばれる）に分割する。量子を識別するとき、ファシリティは、例えば、通信無しのスレッド操作など、並行して実行され得る操作と、スレッド間通信、システムコールなど、決定論的な順序で実行されるべき操作とを区別することができる。次いで、ファシリティによって識別される各量子は、決定論的順序で実行される。マルチスレッドアプリケーションのスレッドによって量子が実行される順序を制御することによって、ファシリティは、マルチスレッドアプリケーションが決定論的に挙動できるようにする。すなわち、同じ入力が与えられると、マルチスレッドアプリケーションのスレッドは、その操作を決定論的にインターリーブし、それによって同じ出力を提供する。
【００１４】
いくつかの実施形態において、ファシリティは、マルチスレッドアプリケーションの実行をシリアル化する。すなわち、ファシリティは、すべてのスレッド操作のグローバルなインターリービングを制御することができる。例えば、これは、スレッド間に決定論的順序で渡されるメモリアクセストークンを確立することによって達成され得る。スレッドは、トークンの値がそのスレッドの識別子に一致するとき、トークンを「保持する」と呼ばれ得る。トークンの値がスレッドの識別子に一致しないとき、トークンの値がスレッドの識別子に一致するまで、その実行は一時停止される。トークンの値がスレッドの識別子に一致するとき、スレッドは、トークンが次のスレッドに渡される前に、決定論的な有限数の操作（すなわち量子）を実行する。例えば、決定論的順序で次のスレッドの識別子に対応するように、トークンの値を進めることによって、トークンは、次のスレッドに渡され得る。

【００１５】
図２は、１つまたは複数の実施形態における、ファシリティによって実行される決定論的シリアル化プロセス２００のフロー図である。例えば、決定論的シリアル化プロセス２００は、マルチスレッドアプリケーションがマルチプロセッシングシステム上で実行している間に実行され得る。マルチスレッドアプリケーションが実行している間、ファシリティは、スレッドごとにステップ２０５～２１５をループする。ステップ２０５で、トークンの値がスレッドの識別子に一致することをファシリティが決定した場合、ファシリティはステップ２１０に進み、そうでない場合、ファシリティは折り返してステップ２０５に戻る。すなわち、ファシリティは、トークンの値がそのスレッドの識別子に一致するまで、スレッドの実行を一時停止する。ステップ２１０で、ファシリティは、識別子がトークンに一致するスレッドが決定論的な有限数の操作（すなわち量子）を実行できるようにし、次いでファシリティは、ステップ２１５に進む。ステップ２１５で、ファシリティは、トークンの値を、決定論的な順序で次のスレッドの識別子に等しくなるように設定し、次いでファシリティは、ステップ２０５に進む。ファシリティは、アプリケーションが終了するまで、シリアル化プロセス２００をループし続けることができることに留意されたい。

【００１６】
図２および以下のフロー図のそれぞれに示されるステップは様々な方法で変更され得ることを、当業者であれば理解されたい。例えば、いくつかのステップの順序が並べ替えられてもよく、いくつかのサブステップが並行して実行されてもよく、いくつかの示されたステップが省略されてもよく、または他のステップが含まれていてもよい。

【００１７】
いくつかの実施形態において、ファシリティは、マルチスレッドアプリケーションの実行を選択的にシリアル化する。すなわち、ファシリティは、他のスレッド操作が並行して実行される間に、いくつかのスレッド操作のインターリービングを制御する（本明細書では「制御された操作」と呼ばれる）ことができる。例えば、ファシリティは、２つ以上のスレッド間の通信を伴う操作のインターリービングを制御することができる。スレッド間通信は、スレッドが別のスレッドによってプライベートに保持されるデータを読み取るとき、またはスレッドが共有データに書き込み、それをプライベート化するときに起こる。いくつかの実施形態において、スレッドが別のスレッドによってプライベートに保持されるとみなされるデータを読み取ろうと試みるとき、スレッドは、トークンの値がその識別子に一致するまで、その実行を一時停止する。同様に、いくつかの実施形態において、スレッドは、共有される、または別のスレッドによってプライベートに保持されるとみなされるデータに書き込もうと試みるとき、トークンの値がその識別子に一致し、すべての他のスレッドがその実行における決定論的ポイントに到達する（例えば、量子の実行を終了する）まで、その実行を一時停止する。その結果、ファシリティは、すべてのスレッドが、その実行での決定論的ポイントにおけるデータの状態の変化（共有からスレッドによってプライベートに保持されるまで）を観察することを確実にする。

【００１８】
いくつかの実施形態において、スレッド間通信を検出するために、ファシリティは、マルチスレッドアプリケーションのアドレス空間におけるメモリロケーションごとに、共有情報を含む共有メモリデータ構造を維持する。例えば、こうした情報は、メモリロケーションが共有である、プライベートであるなどを示すことができる。共有は、操作レベル、命令レベル、ページレベルなど、様々なレベルで起こり得ることに留意されたい。いくつかの実施形態において、スレッドは、それ自体のプライベートに保持されたデータにアクセスすることも、トークンを保持することなく共有データを読み取ることもできる。しかし、共有データに書き込むために、または別のスレッドによってプライベートとして保持されるデータを読み取るために、スレッドは、トークンを保持し、すべての他のスレッドがブロックされるまで待つ（すなわち、他のスレッドもそのトークンを待っている）。スレッドが、プライベートとみなされるメモリロケーションを読み取るとき、共有メモリデ

ータ構造は、読み取られたメモリロケーションを共有されたものとみなすべきであることを示すために更新される。スレッドがメモリロケーションに書き込むとき、メモリロケーションをそのスレッドによってプライベートに保持されているものとみなすべきであることを示すために、共有メモリデータ構造が更新される。同様に、スレッドが別のスレッドによってこれまではアクセスされていないメモリロケーションを読み取るとき、共有メモリデータ構造は、メモリロケーションをそのスレッドによってプライベートに保持されているものとみなすべきであることを示すために更新される。

【００１９】

図３は、１つまたは複数の実施形態における、ファシリティによって実行される決定論的選択的シリアル化プロセス３００のフロー図である。例えば、スレッドまたはプロセッサが、メモリ操作、システムコールなど、制御された操作を実行しようと試行すると、選択的シリアル化プロセス３００が実行され得る。ステップ３０５で、操作がシステムコールである（例えばＩ／Ｏ操作など）ことをファシリティが決定した場合、ファシリティはステップ３２５に進み、そうでない場合、ファシリティはステップ３１０に進む。ステップ３１０で、操作がスレッドによってプライベートに保持されていないメモリにアクセスするとファシリティが決定した場合、ファシリティはステップ３１５に進み、そうでない場合、ファシリティはステップ３５５に進む。ステップ３１５で、操作が共有メモリにアクセスしたことをファシリティが決定した場合、ファシリティはステップ３２０に進み、そうでない場合、ファシリティはステップ３２５に進む。ステップ３２０で、操作が格納操作であることをファシリティが決定した場合、ファシリティはステップ３２５に進み、そうでない場合、ファシリティはステップ３５５に進む。ステップ３２５で、トークンの値がスレッドの識別子に一致することをファシリティが決定した場合、ファシリティはステップ３３０に進み、そうでない場合、ファシリティは折り返してステップ３２５に戻る。すなわち、ファシリティは、トークンの値が選択されたスレッドの識別子に一致するまで、選択されたスレッドの実行を一時停止する。ステップ３３０で、マルチスレッドアプリケーションのすべてのスレッドが一時停止（またはブロック）されたことをファシリティが決定した場合、ファシリティはステップ３３５に進み、そうでない場合、ファシリティは折り返してステップ３３０に戻る。トークンを保持するスレッドが実行し得る前に、すべてのスレッドが一時停止されるのを待つことによって、ファシリティは、実行における決定論的ポイントで、すべてのスレッドが操作の実行に起因する任意の状態の変化を観察することを確実にする。ステップ３３５で、操作がシステムコールであることをファシリティが決定した場合、ファシリティはステップ３５５に進み、そうでない場合、ファシリティはステップ３４０に進む。ステップ３４０で、操作が格納操作であることをファシリティが決定した場合、ファシリティはステップ３４５に進み、そうでない場合、ファシリティはステップ３５０に進む。ステップ３４５で、ファシリティは、操作によって影響を受けるメモリロケーションを、スレッドによってプライベートに保持されているものとみなすべきであることを示すために、共有メモリデータ構造を更新し、次いで、ファシリティはステップ３５５に進む。ステップ３５０で、ファシリティは、操作によってアクセスされたメモリロケーションを共有されたものとみなすべきであることを示すために、共有メモリデータ構造を更新し、次いでファシリティはステップ３５５に進む。ステップ３５５で、ファシリティによって、スレッドは操作を始めることができ、次いでファシリティは戻る。

【００２０】

いくつかの実施形態において、ファシリティは、トランザクショナルメモリシステムと共に動作して、マルチスレッドアプリケーションの実行をシリアル化または選択的にシリアル化する。例えば、ファシリティは、トランザクショナルメモリシステムを使用して、メモリ操作の決定論的順序を侵害することになるスレッド間通信を検出することができる。すなわち、共有メモリデータ構造の代わりに、またはそれに加えて、トランザクショナルメモリシステムが使用され得る。トランザクショナルメモリシステムは、ハードウェアトランザクショナルメモリ（ＨＴＭ）システム、ソフトウェアトランザクショナルメモリ

（ＳＴＭ）システム、またはハイブリッドハードウェア－ソフトウェアトランザクショナルメモリシステム（ＨＳ－ＴＭ）とすることができることに留意されたい。トランザクショナルメモリシステムと共に動作するとき、ファシリティは、スレッドによって実行される各量子をトランザクション内に封入する。各量子をトランザクション内に封入することによって、スレッドは、アトミック的に、かつ隔離されて実行するようにみえる。その結果、トランザクションは、並行して実行され、次いで、決定論的順序に従ってコミットされ得る。通常、トランザクションは、決定論的順序を侵害することになる（本明細書では「競合」と呼ばれる）スレッド間通信を含む場合、コミットされない。競合が存在するとき、トランザクションは、中止され、再開される。

【００２１】

いくつかの実施形態において、ファシリティは、量子ビルダコンポーネント（quantum builder component）、および決定論的マルチプロセッシング（「ＤＭＰ」）コンポーネントを含む。量子ビルダコンポーネントは、マルチスレッドアプリケーションの実行を量子（すなわち、決定論的な有限数の操作の組）に分割するために使用される。いくつかの実施形態において、量子ビルダコンポーネントは、例えば通信無しのスレッド操作など、並行して実行され得る操作と、スレッド間通信、システムコールなど、決定論的な順序で実行されるべき操作（例えば、制御された操作）とを区別する。ＤＭＰコンポーネントは、決定論的順序に従って各量子が実行されることを確実にする。いくつかの実施形態において、トークンがブロックされた（例えば、別のスレッドによって保持されたロックを待つ）スレッドに進められると、ファシリティは、トークンを次のスレッドに渡し、それによって、開発者がマルチスレッドコード内に含まれる同期プリミティブのブロックに起因するライブロックを回避する。例えば、トークンがスレッド２に渡されるときにスレッド２が進むために必要とするロックをスレッド１が保持する場合、トークンは、次のスレッド（例えば、スレッド３）に渡される。トークンが決定論的順序で渡されるため、また各スレッドが量子を実行する（またはトークンを渡す）ため、量子は、決定論的にインターリーブされ、それによってコードが同じ入力で実行されるたびに同じ出力を生成し、ライブロックを防ぐ。

【００２２】

量子ビルダコンポーネントおよびＤＭＰコンポーネントは、ハードウェア、ソフトウェア、またはハードウェアおよびソフトウェアの組み合わせにおいて実装され得る。例えば、量子ビルダコンポーネントは、命令が後退するにつれてそれらをカウントし、所定の量子サイズに到達したとき、量子境界を配置することによって実装され得る。実行をシリアル化するために、ＤＭＰコンポーネントは、決定論的順序で量子境界においてプロセッサ間に渡されるトークンとして実装され得る。別の例として、実行を選択的にシリアル化するために、量子ビルダコンポーネントは、アクセスがスレッド間通信を伴うかどうか（例えば、共有データへのアクセスなど）を決定するために、メモリアクセスを監視することができる。例えば、一実施形態において、量子ビルダは、共有テーブルを実装するために、ＭＥＳＩ（「変更、排他、共有、無効」）キャッシュコヒーレンスプロトコルによって維持されるキャッシュライン状態を使用する。排他または変更状態のキャッシュラインは、プロセッサによってプライベートに保持されるものとみなされ、トークンを保持しないそれ自体のスレッドによって自由に読み取られ、または書き込まれ得る。同様に、共有状態のキャッシュラインは、トークンを保持しないそれ自体のスレッドによって自由に読み取られ得る。すべてのスレッドがその実行における決定論的ポイントにあるとき（例えば、すべてのプロセッサがブロックされたとき）、およびプロセッサが決定論的トークンを取得したとき、プロセッサは、共有状態のキャッシュラインに書き込むことができる。こうした実施形態において、各プロセッサは、それがブロックされ、および／またはブロック解除されると、ブロードキャストする。任意のプロセッサによってキャッシュに入れられないラインに対応する共有テーブルにおけるエントリの状態は、メモリに保持され、メモリコントローラによって管理することができ、こうしたエントリの状態は、キャッシュミスが処理されるときに転送され得ることに留意されたい。いくつかの実施形態において

、量子ビルダおよびＤＭＰコンポーネントは、ハードウェアトランザクショナルメモリ（ＨＴＭ）システムなどのトランザクショナルメモリ（ＴＭ）システムと共に動作して、特定のトランザクションコミット順序、すなわちトランザクション内に封入された量子の決定論的コミット順序を指定する。こうした実施形態において、ＴＭシステムは、プロセッサがトークンを保持するとき、トランザクションをコミットし、トランザクションがコミットされた後、トークンは、決定論的順序で次のプロセッサに渡される。いくつかの実施形態において、ハードウェアは複数のトークンをサポートすることができ、それによって、プロセッサ間に渡されるトークンをそれぞれ指定する複数の決定論的プロセスが同時に実行できるようになることに留意されたい。

【００２３】

いくつかの実施形態において、ファシリティは、コンパイラまたはバイナリ書き換えインフラストラクチャを使用して実装され得る。例えば、量子ビルダコンポーネントは、マルチスレッドアプリケーションコード内に同期コードを挿入し、コンパイラによって生成される制御フローグラフ（「ＣＦＧ」）において操作を追跡することによって、コンパイラを使用して量子を構築することができる。量子は、サイズが決定論的である限り、均一サイズのものである必要はないことに留意されたい。こうした同期コードは、例えば、関数呼び出しの最初および最後、およびＣＦＧ後退エッジの最後に挿入することができる。挿入されたコードは、量子サイズを追跡し、ターゲットサイズに到達したとき、ＤＭＰコンポーネントにコールバックする。例えば、実行のこうした実施形態をシリアル化するために、ＤＭＰコンポーネントは、決定論的順序でスレッド間に渡される待ち行列ロックとしてトークンを実装することができる。別の例として、実行を選択的にシリアル化するために、量子ビルダコンポーネントは、ロード操作および格納操作がＤＭＰコンポーネントへのコールバックをもたらすように、コンパイラを使用して、コードを挿入することができる。いくつかの実施形態において、ＤＭＰコンポーネントは、ソフトウェアトランザクショナルメモリ（ＳＴＭ）システムなどのトランザクショナルメモリシステムと共に動作し、および／または共有テーブルを実装する。

【００２４】

いくつかの実施形態において、スレッドによって実行される操作のインターリービングを制御するために、ファシリティは、ソースコード、ソースコードの中間表現、または実行ファイルを増補することができる。例えば、ファシリティは、１つまたは複数の決定論的マルチプロセッシング（「ＤＭＰ」）関数またはデータ構造をアプリケーションコードに挿入することによって、マルチスレッドアプリケーションコードを増補することができる。別の例として、挿入されたＤＭＰ関数は、１つまたは複数のデータ構造（例えば、共有メモリデータ構造）を維持する、ＤＭＰコンポーネントによって提供されるものなど、ランタイムシステムにコールバックすることができる。増補されたコードがマルチプロセッシングシステムによって実行されると、挿入されたＤＭＰ関数およびデータ構造は、次いで、メモリおよびＩ／Ｏ操作、システムコールなど、操作が実行される順序を制御するために使用される。スレッドがこうした動作を実行する順序を制御することによって、ファシリティは、マルチスレッドアプリケーションが決定論的に挙動できるようにする（本明細書では、「増補されたアプリケーション」と呼ばれる）。すなわち、同じ入力が与えられると、増補されたアプリケーションのスレッドは、操作の一部またはすべてを決定論的にインターリーブし、それによって同じ出力を提供することができる。ファシリティは他のスレッド操作を制御するように拡張され得ることを、当業者であれば理解されたい。

【００２５】

いくつかの実施形態において、ファシリティは、増補されたアプリケーションのスレッドによって実行される量子の決定論的実行を実施する、ＤＭＰライブラリによって提供される関数を挿入することによって、マルチスレッドアプリケーションコードを増補するコンパイラモジュールとして実装される。いくつかの実施形態において、コードが増補された後、コンパイラは、例えば、ＤＭＰライブラリに対するすべての呼び出しをインライン化するなど、コードを再度最適化する。コンパイラは本明細書では具体的に記載されない

増補されたコードへの他の最適化を実行することができることを、当業者であれば理解されたい。

【００２６】

いくつかの実施形態において、ファシリティは、本明細書では「スレッドデータ構造」と呼ばれるＤＭＰデータ構造を含み、その詳細は、図６を参照して以下でより詳しく説明される。しかし、任意の数のＤＭＰデータ構造が含まれていてよいことに留意されたい。スレッドデータ構造が複数のＤＭＰデータ構造を表し得ることにさらに留意されたい。いくつかの実施形態において、スレッドデータ構造は、実行中に増補されたアプリケーションによって作成される各スレッドに対応するスレッド識別子（「ＩＤ」）を格納する。例えば、スレッドデータ構造は、配列、リンクリスト、キュー、またはスレッドＩＤの他のデータ構造（本明細書では「スレッドコンテナ」と呼ばれる）を含み得る。

【００２７】

いくつかの実施形態において、スレッドデータ構造は、量子の実行の順序を制御するために使用され得るトークンを含む。例えば、いくつかの実施形態において、量子を実行する前に、スレッドは、トークンの現在の値がスレッドのＩＤに一致するかどうかを決定する。スレッドのＩＤがトークンの現在の値に一致するとき、スレッドは、量子を実行することができる。そうでない場合、スレッドは、トークンの現在の値がその識別子に一致するまで、量子を実行するのを待つ。

【００２８】

いくつかの実施形態において、スレッドが作成される順序は、スレッドが決定論的に実行される順序に対応する。例えば、各スレッドが作成されるとき、スレッドの対応するスレッドＩＤは、スレッドコンテナ内に順次格納され得る（例えば、最初に作成されたスレッドのスレッドＩＤは１、２番目に作成されたスレッドのスレッドＩＤは２など）。操作が実行されるとき、スレッドは、（第１のスレッドＩＤから開始して）スレッドＩＤが格納される順序に基づいてスレッドコンテナに格納されるスレッドＩＤを順次ループすることによって、トークンの値を進めるよう動作するいくつかのＤＭＰ関数を呼び出すことができる。スレッドが存在するとき、通常、スレッドの対応するＩＤがスレッドコンテナから削除されることに留意されたい。

【００２９】

いくつかの実施形態において、スレッドデータ構造は、トークンが進められる前にスレッドＩＤがトークンの現在の値に一致するスレッドによって実行され得る決定論的な有限数の（すなわち量子）制御された操作またはブロックに対応する値を格納する。制御された操作またはブロックのこの数は、本明細書では「コミットブロックサイズ」と呼ばれる。コミットブロックサイズは、１つからＮ個までの制御された操作またはブロックに及び得る。大きいコミットブロックサイズおよび小さいコミットブロックサイズには性能のトレードオフが関連することを、当業者であれば理解されたい。例えば、コミットブロックサイズが小さすぎるとき、スレッド間でのコンテキストの切り替えに伴うオーバーヘッドの結果として、増補されたアプリケーションの性能が悪化する。別の例として、コミットブロックサイズが大きすぎるとき、多くのまたはすべてのスレッドは、スレッドＩＤがトークンに一致するスレッド（およびスレッドＩＤがそのスレッドＩＤに先行するすべてのスレッド）がコミットブロックサイズによって指定された数の制御された操作を終了する、または実際に実行するのを待つのを余儀なくされ得るため、増補されたアプリケーションの性能が悪化する。少なくとも１つの実施形態において、コミットブロックサイズは、１，０００に等しい。

【００３０】

いくつかの実施形態において、コミットブロックサイズは、構成可能である。例えば、コミットブロックサイズは、増補されたアプリケーションの様々なスレッドインターリービングをプログラム的に操作し、テストするように、ソフトウェア開発者によって構成され得る。別の例として、コミットブロックサイズは、増補されたアプリケーションによって作成され得る最大数のスレッド、および／または増補されたアプリケーションが実行す

るマルチプロセッシングシステムのプロセッサまたはコアの数に基づいて、自動的に構成され得る。スレッドによって実行される制御された操作の数をカウントするために様々な技術が使用され得ることを、当業者であれば理解されたい。例えば、いくつかの実施形態において、スレッドデータ構造は、スレッドＩＤが現在のトークンＩＤに一致するスレッドによって実行された制御された操作の数に対応する値を含む。スレッドが制御された操作を実行するたびに、制御された操作の数は、増分され、コミットブロックサイズと比較される。制御された操作の数がコミットブロックサイズに等しい場合、トークンは、次のスレッドＩＤに進められ、制御された操作の数は、ゼロにリセットされる。

【００３１】

マルチスレッドアプリケーションを増補して、いくつかのスレッドの操作（例えば、制御されたスレッド操作）の順序を制御することによって、開発プロセスは、かなり簡略化される。例えば、ファシリティは、マルチスレッドアプリケーションのスレッドインターリービングを直接操作し、それによってマルチスレッドアプリケーションの実質的により良いテストカバレージを可能にできるようにするために、ソフトウェア開発者によって使用され得る。開発者は、例えばコミットブロックサイズを変更することによって、制御されたスレッド操作のインターリービングを操作することができる。別の例として、開発者は、スレッドコンテナに格納されるスレッドＩＤの順序を変更することによって、制御されたスレッド操作のインターリービングを操作することができる。いくつかの実施形態において、ファシリティによって、ソフトウェア開発者は、挿入されたコードが量子構築物に影響を与えないように、増補のために挿入されたとコードをマーク付けすることができる。

【００３２】

いくつかの実施形態において、マルチスレッドアプリケーションは、その増補された形で配置される。マルチスレッドアプリケーションを増補された形で配置することによって、アプリケーションの信頼性は、実質的に向上する。というのは、例えば、「現場での」（すなわち顧客による）増補されたアプリケーションの実行は、社内でのアプリケーションのテストに、より似たものになるからである。さらに、マルチスレッドアプリケーションがクラッシュする、または同期バグを経験するとしたら、ソフトウェア開発者は、顧客から意味のあるクラッシュ情報を集めることによって欠陥を迅速に解決することができる。すなわち、増補された形で配置されると、クラッシュに先行する顧客によって実行されるアクションは、ソフトウェア開発者がクラッシュを容易に再生することができるようになるため、意味がある。その結果、ソフトウェア開発者は、クラッシュまたは同期バグがスレッドの未知のインターリービングに関連付けられた場合より実質的に早く欠陥を解決することができる。したがって、ファシリティは、マルチスレッドアプリケーションの開発および配置の両方を向上させる。

【００３３】

いくつかの実施形態において、マルチスレッドアプリケーションが開発される、および／またはマルチスレッドアプリケーションが配置されるコンピューティングシステムは、共有メモリへのアクセスを制御するためのトランザクショナルメモリ（「ＴＭ」）システムを含む。トランザクショナルメモリシステムは、ハードウェアトランザクショナルメモリ（「ＨＴＭ」）、ソフトウェアトランザクショナルメモリ（「ＳＴＭ」）システム、またはハイブリッドハードウェア－ソフトウェア（ＨＳ－ＴＭ）システムとすることができる。両方のＴＭシステムは、当分野で知られている。ＳＴＭシステムは、プログラミングアブストラクション（programming abstraction）を提供し、それを介して、スレッドは、共有リソースをロックすることなく、または共有リソースが解放されるのを待つことなく、その一部に１つまたは複数の共有リソース（例えばメモリ）が関与し得る操作のシーケンスをアトミック的に実行する。

【００３４】

従来のＴＭシステムは、他のスレッドが何をしているかに関係なく、スレッドが共有メモリへの変更を終了するという点で「楽観的」である。これは、例えば、マルチスレッド

アプリケーションのスレッドごとにログを維持することによって達成され、トランザクションごとに、各スレッドは、その対応するログにその操作を順次記録する。例えば、ログは、メモリロケーションの数、並びにトランザクション中にスレッドが読み取り、および／または書き込む値を含み得る。トランザクションの最後に、他のスレッドが同じ共有メモリロケーションに並行してアクセスしなかった場合、スレッドは、実際に、操作のシーケンスを実行する（これは一般に「コミット」と呼ばれる）。しかし、別のスレッドが同じメモリロケーションのうちの１つまたは複数に並行してアクセスした場合、トランザクションは、中止され、再開される。すなわち、従来のＴＭシステムにおいて、同じトランザクション中に共有リソースが複数のスレッドによってアクセスされない限り、トランザクションは、並行して実行する。

【００３５】

従来のＴＭシステムに関連付けられた欠点がいくつかある。例えば、従来のＴＭシステムは、開発者がいくつかの操作、またはいくつかの操作のシーケンスをアトミックとして宣言できるようにすることによって、開発をある程度簡略化するが、従来のＴＭシステムは、マルチスレッドアプリケーションの決定論的マルチプロセッシングを提供しない。さらに、従来のＴＭシステムでは、ソフトウェア開発者は、マルチスレッドアプリケーションにおけるスレッドのインターリービングを指定し、または操作することができない。その結果、従来のＴＭシステムは、潜在的な同期バグにも苦しむ。また、ＨＴＭシステムと比較すると、ＳＴＭシステムは、ログの維持に伴うオーバーヘッド、およびトランザクションのコミットに費やされた時間の結果、パフォーマンスヒットを被る。

【００３６】

いくつかの実施形態において、ファシリティは、ＨＴＭ、ＳＴＭ、ＨＳ－ＴＭシステムなど、共有リソースへのアクセスを制御するためにトランザクショナルメモリシステムを使用するマルチスレッドアプリケーションのいくつかのスレッド操作の実行の順序を制御する。すなわち、ファシリティは、スレッドが開始する、および／またはトランザクショナルメモリシステムにおけるトランザクションをコミットする順序を制御することができる。いくつかの実施形態において、ファシリティは、ＳＴＭシステムによって提供されるアプリケーションプログラミングインターフェイス（「ＡＰＩ」）を増補する。一例として、ファシリティは、以下の表１に示されたＳＴＭ　ＡＰＩの関数を増補することができる。ファシリティのいくつかの実施形態は、表１に提供されるＳＴＭ　ＡＰＩを参照して記載されるが、ファシリティは様々なトランザクショナルメモリシステムにおいて動作し得ることを、当業者であれば理解されたい。

【００３７】

【表１】

表１

void STMBeginTransaction( ):	スレッドによって実行される新しいトランザクションを開始する
value STM Read (*addr):	共有メモリロケーションの動作タイプ・アドレス、及び／又は、現在の値についての情報をログに記録する
void STMWrite(*addr, value):	操作タイプ・アドレス、及び／又は、操作の結果としての共有メモリロケーションの現在の値についての情報をログに記録する
bool STMValidTransaction ( ):	スレッドのログに基づいて、別のスレッドが同じ共有リソースのうち１つまたは複数に並行してアクセスしたかどうかを決定する
void STMAbortTransaction( ):	スレッドによって実行されるトランザクションを中止する
bool STMCommitTransaction( ):	スレッドによって実行されるトランザクションをコミットする

【００３８】

いくつかの実施形態において、ソフトウェア開発者は、マルチスレッドアプリケーション内のアトミックブロックを手動で指定する。例えば、ソフトウェア開発者は、以下のアトミックブロックを含み得る。

【００３９】

【数１】

```
atomic {
        a = b + c;
}
```

【００４０】

コンパイル後、上記のアトミックブロック例は、以下の擬似コードによって置き換えられることになる。

【００４１】

【数２】

```
STM_Begin_Transaction( );
try {
        var_1 = STMRead(*b);
        var_2 = STMRead(*c);
        STMWrite(*a, var_1 + var_2);

        bool transaction_valid = STMValidTransaction( );
        if (!STMValidTransaction( )) {
              STMAbortTransaction( );
        }
        else if (STMValidTransaction( )) {
              bool transaction_commited = STMCommitTransaction( );
              if (!transaction_commited) {
                throw transaction_failed_to_commit;
              }
        }
  }
  catch (transaction_failed_to_commit)
  {
        …
  }
```

【００４２】

いくつかの実施形態において、トランザクションのうちの１つまたは複数（すなわち、アトミックブロック）は、ソフトウェア開発者に可視ではない。例えば、これらは、コンパイラ、ランタイム、ＴＭシステム、またはその何らかの組み合わせによって挿入され得る。いくつかの実施形態において、ブロックがソフトウェア開発者によって指定されたか、それともコンパイラ、ランタイム、またはＴＭシステムによって挿入されたかにかかわらず、アトミックブロックは、増補される。いくつかの実施形態において、スレッドがＳＴＭ　ＡＰＩの増補された関数を呼び出すと、関数は、トークンの現在の値に対応するスレッドＩＤをチェックするＤＭＰ関数に制御を転送し、これは、トランザクションを開始し、および／または決定論的にコミットするために使用される。多くの異なる技術はトランザクションをインターセプトするために使用され得ることを、当業者であれば理解されたい。例えば、いくつかのＳＴＭ　ＡＰＩは、ＡＰＩ関数の実行前および／または後に、制御をＤＭＰ関数に転送するために、フックが登録され得るコールバック機構を提供する。

【００４３】

増補されたトランザクショナルメモリシステムのトランザクションは、サイズが決定論的である。すなわち、各スレッドは、ブロックにおいて特定数の操作（本明細書では「コミットブロックサイズ」と呼ばれる）を実行し、次いでスレッドは、ＩＤがトークンの現在の値に一致するスレッドで開始して、決定論的にコミットしようと試みる。トランザクションが有効であり、スレッドＩＤがトークンに一致する場合、スレッドは、ＳＴＭ＿Ｃｏｍｍｉｔ＿Ｔｒａｎｓａｃｔｉｏｎ（）を呼び出す。トランザクションがコミットされた後、トークンは、次のスレッドＩＤに進められる。しかし、トランザクションが無効で

ある場合（例えば、スレッドがそのトランザクション中に別のスレッドによって書き込まれたロケーションから読み取ったため）、スレッドはＳＴＭ＿Ａｂｏｒｔ＿Ｔｒａｎｓａｃｔｉｏｎ（）を呼び出す。通常、スレッドＩＤがトークンに一致するスレッドがその対応するトランザクションを正常にコミットするまで、トークンは進められないことに留意されたい。

【００４４】

いくつかの実施形態において、トークンの現在の値がトランザクションを実行するスレッドのスレッドＩＤに一致しない場合、いくつかのタイプの操作は、トランザクションに即座に中止させる。例えば、トランザクションがＩ／Ｏ操作など元に戻すことができない操作を含むとき、トランザクションを実行するスレッドは、そのスレッドＩＤがトークンに一致するかどうかを決定する。そのスレッドＩＤがトークンに一致する場合、トランザクションは、続行し得る。そうでない場合、トランザクションは、自動的に中止され得る。

【００４５】

いくつかの実施形態では、中止されたスレッド以降のスレッドＩＤを有するすべてのスレッドが中止され、一方、別の実施形態では、並行のトランザクションが同じ共有リソースにアクセスしたスレッドのみが中止され、再開される。通常、スレッドＩＤがトークンに一致するスレッドがその対応するトランザクションを正常にコミットするまで、トークンは進められない。その結果、それらのトランザクションを中止しなかった中止されたスレッド以降のスレッドＩＤを有する任意のスレッドは、ＳＴＭ＿Ｃｏｍｍｉｔ＿Ｔｒａｎｓａｃｔｉｏｎ（）を呼び出す前に、トークンがそのスレッドＩＤに一致するのを待つ。

【００４６】

ＨＴＭを有するコンピューティングシステムにおいて増補されたアプリケーションが実行されると、増補されたアプリケーションは、実質的に性能のペナルティなく、決定論的に実行され得ることに留意されたい。その結果、ソフトウェア開発者および／または製造業者は、スレッドインターリービングの可能性について徹底的にテストしたことを知っているマルチスレッドアプリケーションを配布することができる。したがって、同期バグがマルチスレッドコードに残っている場合でさえ、顧客には見えない。

【００４７】

より詳しくファシリティについて説明する前に、ファシリティを実施することができる環境について検討することが有用である。図４は、１つまたは複数の実施形態における、ファシリティが実行するコンピューティングシステム４００のアーキテクチャ例を示す高レベルブロック図である。説明を曖昧にするのを回避するために、いくつかのよく知られている構造および機能は、詳細に示されても述べられてもいない。コンピューティングシステム４００は、相互接続システム４１５に結合された１つまたは複数のプロセッサ４０５およびメモリ４１０を含む。プロセッサ４０５は、コンピューティングシステム４００の中央処理装置（「ＣＰＵ」）であり、したがってその操作全体を制御する。いくつかの実施形態において、プロセッサ４０５は、メモリ４１０に格納されたソフトウェアを実行することによってこれを達成する。いくつかの実施形態において、コンピューティングシステム４００は、単一の集積回路（「ダイ」と呼ばれる）から成るパッケージ、ひとまとめにされた１つまたは複数のダイ、複数のパッケージなどに２つ以上の独立したコアを有するプロセッサ４０５を含む。いくつかの実施形態において、コンピューティングシステム４００は、単一のコアのみを有するにもかかわらず、マルチコアプロセッサとして実行することができるハイパースレッドプロセッサ４０５を含む。プロセッサ４０５は、１つまたは複数のプログラム可能な汎用または専用マイクロプロセッサ、デジタル信号プロセッサ（「ＤＰＳ」）プログラム可能コントローラ、特定用途向け集積回路（「ＡＳＩＣ」）、プログラム可能論理装置（「ＰＬＤ」）など、またはこうした装置の組み合わせとすることができ、またはそれらを含み得る。

【００４８】

図４に示される相互接続システム４１５は、適切なブリッジ、アダプタ、および／また

はコントローラによって接続される任意の１つまたは複数の個別の物理バスおよび／またはポイントツーポイント接続を表す抽象概念である。相互接続システム４１５は、例えば、システムバス、ある形の周辺機器コンポーネント相互接続（ＰＣＩ）バス、ハイパートランスポートまたは業界標準アーキテクチャ（ＩＳＡ）バス、小型コンピュータシステムインターフェイス（ＳＣＳＩ）バス、ユニバーサルシリアルバス（ＵＳＢ）、電気電子技術者協会（ＩＥＥＥ）標準１３９４バス（時として「ＦｉｒｅＷｉｒｅ」と呼ばれる）などを含み得る。

【００４９】

システムメモリ４１０は、プログラムおよびデータが使用されている間にそれらを格納するためのメモリ４２０、プログラムおよびデータを永続的に格納するためのハードドライブなどの固定記憶装置４２５、およびコンピュータ可読媒体に格納されるプログラムおよびデータを読み取るためのＣＤ－ＲＯＭまたはＤＶＤ－ＲＯＭドライブなどのコンピュータ可読媒体ドライブ４３０を含む。本明細書で使用される場合、システムメモリ４１０は、任意の形の揮発性、不揮発性、取外式、および固定式媒体、またはコンピュータ可読命令、データ構造、プログラムモジュール、およびコンピューティングシステム４００の他のデータなどの情報を格納することができるこうした媒体装置の任意の組み合わせを含む。

【００５０】

また、プロセッサ４０５には、相互接続システム４１５を介して、ネットワークアダプタ４３５、１つまたは複数の入力装置および出力装置（「Ｉ／Ｏ装置」）４４０も結合される。ネットワークアダプタ４３５は、コンピューティングシステム４００に、ネットワークを介して他のコンピューティングシステムと通信することができる機能を提供し、例えば、Ｅｔｈｅｒｎｅｔ(登録商標)アダプタとすることができる。Ｉ／Ｏ装置４４０は、コンピューティングシステム４００のユーザに、システムメモリ４１０に格納されるプログラムおよびデータにアクセスすることができる機能を提供する。例えば、Ｉ／Ｏ装置４４０は、キーボード、ポインティング装置、マイクロフォンなどの入力装置、および表示装置、スピーカ、プリンタなどの出力装置を含み得る。上述したように構成されたコンピューティングシステムは、通常、ファシリティの操作をサポートするために使用されるが、様々なタイプおよび構成の装置を使用して、様々なコンポーネントを有するファシリティが実装され得ることを、当業者であれば理解されたい。

【００５１】

図５は、１つまたは複数の実施形態における、決定論的マルチプロセッシングレイヤ５００の様々な機能的要素を示す高レベルブロック図である。決定論的マルチプロセッシングレイヤ５００はコンピューティングシステム４００によって実装される必要がないことに留意されたい。例えば、いくつかの実施形態において、決定論的マルチプロセッシングレイヤ５００は、マルチスレッド型ソフトウェアコードが入力として提供される個別のコンピューティングシステムに実装される。

【００５２】

いくつかの実施形態において、決定論的マルチプロセッシングレイヤ５００は、量子ビルダコンポーネント５０５および決定論的マルチプロセッシング（「ＤＭＰ」）コンポーネント５１０を含む。量子ビルダコンポーネント５０５は、例えば、ＤＭＰコンポーネント５１０によって提供される関数５１５～５４０のうちの１つまたは複数を使用して、マルチスレッドアプリケーション５４５のコードを増補するコンパイラモジュールとして実装され得る。ＤＭＰコンポーネント５１０によって提供される関数は様々な方法で変更され得ることを、当業者であれば理解されたい。例えば、いくつかの関数がマージまたは分割されてもよく、いくつかの関数が省略されてもよく、いくつかの関数が追加されてもよい。いくつかの実施形態において、量子ビルダコンポーネント５０５は、例えば低レベル仮想マシン（「ＬＬＶＭ」）コンパイラインフラストラクチャ内など、コンパイラインフラストラクチャ内にコンパイラパスとして実装される。一方、他の実施形態では、量子ビルダコンポーネント５０５は、マルチスレッドアプリケーションコード５４５が入力とし

て提供される個別のシステムによって実装される。

【００５３】

示された実施形態において、決定論的マルチプロセッシングレイヤ５００は、マルチスレッドアプリケーションコード４１０を受信し、および／またはそれにアクセスする。マルチスレッドアプリケーションコード５４５は１つまたは複数のコードファイルを表し得ることに留意されたい。コード５４５は、マルチスレッドアプリケーションのソースコード、マルチスレッドアプリケーションのソースコードの中間表現（「ＩＲ」）、マルチスレッドアプリケーションの実行ファイルなどとすることができる。いくつかの実施形態において、量子ビルダコンポーネント５０５は、マルチスレッドアプリケーションコード５４５内に同期コードを挿入して、コンパイラによって生成される制御フローグラフ（「ＣＦＧ」）において操作を追跡することによって、コンパイラを使用して量子を構築することができる。挿入されたコードは、量子サイズを追跡し、量子サイズに到達したとき、ＤＭＰコンポーネント５１０によって提供される１つまたは複数の関数を呼び出して、アプリケーション内のスレッドの前進を制御する。ＤＭＰコンポーネント５１０は、ランタイムシステムを提供することができ、および／またはＤＭＰ関数５１５～５４０のうちの１つまたは複数をコード５４５に挿入することができる。いくつかの実施形態において、決定論的プロセッシングレイヤ５００は、トランザクショナルメモリシステムと共に動作し、および／または共有テーブルを実装する。

【００５４】

示された実施形態において、ＤＭＰライブラリは、ＤＭＰ開始関数（「ＤＭＰ＿Ｆｕｎｃｔｉｏｎ＿Ｓｔａｒｔ（）関数５１５」）、ＤＭＰ初期化関数（「ＤＭＰ＿Ｉｎｉｔ（）関数５２０」）、ＤＭＰ格納関数（「ＤＭＰ＿Ｓｔｏｒｅ（）関数５２５」）、ＤＭＰロード関数（「ＤＭＰ＿Ｌｏａｄ（）関数５３０」）、ＤＭＰコミット関数（「ＤＭＰ＿Ｃｏｍｍｉｔ（）関数５３５」）、およびＤＭＰ終了関数（「ＤＭＰ＿Ｆｕｎｃｔｉｏｎ＿Ｅｎｄ（）関数５４０」）を含む。ＤＭＰ開始関数５１５および終了関数５４０は、アプリケーション関数が開始し、終了するときを画定するために使用され得る。ＤＭＰロード関数５３０は、ロード操作が実行される、または実行された決定論的マルチプロセッシングレイヤ５００に運ぶために使用され得る。同様に、ＤＭＰ格納関数５２５は、格納操作が実行される、または実行された決定論的マルチプロセッシングレイヤ５００に運ぶために使用され得る。ＤＭＰ格納関数５２５およびロード関数５３０は、メモリ操作の順序を制御し、それによってこうした操作の決定論的実行を実施するために使用される。ＤＭＰ初期化関数５２０およびＤＭＰコミット関数５３５は、メモリ操作の順序を制御する、またはトランザクションを開始し、または終了するために使用されるコードのブロックを画定するために使用され得る。ＤＭＰコンポーネント５１０によって提供される関数は様々な方法で変更され得ることを、当業者であれば理解されたい。例えば、いくつかの関数がマージまたは分割されてもよく、いくつかの関数が省略されてもよく、いくつかの関数が追加されてもよい。

【００５５】

いくつかの実施形態において、量子ビルダコンポーネント５０５は、以下の表２に列挙されるＤＭＰコンポーネント５１０の関数５１５～５４０を挿入する。

【００５６】

【表２】

表２

DMP_Function_Start( ) -	コード 545 に含まれる各関数の最初に挿入される
DMP_Function_End( ) -	コード 545 に含まれる各関数の最後に挿入される
DMP_Load( ) -	各ロードブロックの前に挿入される
DMP_Store( ) -	各ストアブロックの前に挿入される
DMP_Commit( ) -	任意のジャンプブロックの前に挿入される；任意の関数呼び出しの前に挿入される；任意のＯＳの呼び出しの前に挿入される；リターンブロックの前に挿入される
DMP_Init( ) -	DMP_Commit( )を含む別のブロックから各ジャンプ先ブロック（jump-to block）に挿入される；各関数呼び出し後に挿入される;各ＯＳ呼び出し後に挿入される;DMP_Function_Start( )後に挿入される

【００５７】

いくつかの実施形態において、量子ビルダコンポーネント５０５は、増補されたコードの中間表現を作成し、これは、例えば、制御フローグラフ（「ＣＦＧ」）として表され得る。図１１は、表２に従って増補されるマルチスレッドアプリケーションコード５４５の関数の制御フローグラフの一例を示す。いくつかの実施形態において、マルチスレッドアプリケーションコード５４５が増補された後、コンパイラは、例えばＤＭＰ関数５１５～５４０への呼び出しをインライン化することによって、増補されたコードを再度最適化する。コンパイラは本明細書では具体的に記載されない増補されたコードへの他の最適化を実行することができることを、当業者であれば理解されたい。

【００５８】

いくつかの実施形態において、マルチスレッドアプリケーションコード５４５は、ＳＴＭ、ＨＴＭ、またはＨＳ－ＴＭなど、トランザクショナルメモリシステムを使用して、スレッドによる共有リソースへのアクセスを制御する。こうした実施形態において、決定論的マルチプロセッシングレイヤ５００は、トランザクションがマルチスレッドアプリケーションのスレッドによってコミットされる順序を制御するために使用され得る。例えば、量子ビルダ５０５は、ＤＭＰ初期化関数５２０およびＤＭＰコミット関数５３５への呼び出しを挿入することによってトランザクションにおける各量子を包むことができる。別の例として、マルチスレッドアプリケーションコード５４５が１つまたは複数のアプリケーションレベルトランザクショナルメモリブロックを含むとき、量子ビルダコンポーネント５０５は、ソフトウェア開発者によって宣言される各アトミックブロックの前にＤＭＰ初期化関数５２０への呼び出しを挿入することによって、また命令をコミットするためのＴＭシステムへの任意の呼び出しの前にＤＭＰコミット関数５３５への呼び出しを挿入することによって、マルチスレッドアプリケーションコード５４５を増補することができる。さらに別の例として、決定論的マルチプロセッシングレイヤ５００は、ＴＭインターフェイスの関数への呼び出しをＤＭＰコンポーネント５１０の１つまたは複数の関数５１５～５４０への呼び出しで包むことによって、ＴＭシステムによって提供されるインターフェイスを増補することができる。その結果、決定論的マルチプロセッシングレイヤ５００がＴＭシステムと共に動作するとき、トランザクションは、決定論的に開始され、および／またはコミットされ得る。トランザクショナルメモリシステムがＨＴＭシステムであるとき、ＨＴＭがこうした追跡を実行する限り、ＤＭＰロード関数５３０およびＤＭＰ格納関数５２５が含まれる必要はないことに留意されたい。

【００５９】

いくつかの実施形態において、マルチスレッドアプリケーションコード５４５は、実行

可能な増補されたアプリケーション５５０にコンパイルされる。一方、他の実施形態では、増補されたアプリケーション５５０は、マシンに依存しない中間言語コードであり、これは、実行時に実行可能命令に変換される。増補後、増補されたアプリケーション５５０は、マルチプロセッシングシステム上で決定論的に実行され得る。すなわち、増補されたアプリケーション５５０に同じ入力が与えられると、マルチプロセッシングシステムは、スレッド量子を決定論的にインターリーブし、それによって、増補されたアプリケーション５５０が実行されるたびに同じ入力を生成する。図５に示されるコンポーネントは様々な方法で変更され得ることを、当業者であれば理解されたい。例えば、コンパイラなど、いくつかのコンポーネントがマージまたは分割されてもよく、いくつかのコンポーネントが省略されてもよく、いくつかのコンポーネントが追加されてもよい。

【００６０】

いくつかの実施形態において、ＤＭＰコンポーネント５１０によって提供される関数５１５～５４０は、増補されたアプリケーションのスレッド間に決定論的にトークンを渡し、または進め、それによって各スレッドの前進を決定論的に制御する責任を負う。いくつかの実施形態において、これは、スレッドデータ構造６００を使用することによって達成される。図６は、１つまたは複数の実施形態において、マルチプロセッサコードを決定論的にするためにファシリティによって使用されるスレッドデータ構造６００を示す高レベルブロック図である。いくつかの実施形態において、スレッドデータ構造６００は、スレッドコンテナ６０５を含む。スレッドコンテナは、実行中に増補されたアプリケーションによって作成されるスレッドごとにスレッドＩＤを格納する。スレッドコンテナ６０５は、配列、リンクリスト、キュー、またはスレッドＩＤの他のデータ構造として実装され得る。

【００６１】

いくつかの実施形態において、スレッドデータ構造６００は、実行中に増補されたアプリケーションのスレッドによるトランザクションまたは制御された操作の実行の順序を制御するために使用されるトークン６１０を含む。例えば、いくつかの実施形態において、制御された操作を実行する、またはトランザクションをコミットする前に、スレッドは、そのスレッドＩＤがトークン６１０の現在の値に一致するかどうかを決定する。トークン６１０の現在の値がスレッドＩＤに一致するとき、対応するスレッドは、制御された操作を実行する、またはトランザクションをコミットしようと試行することができる。そうでない場合、対応するスレッドは、トークン６１０の現在の値がそのスレッドＩＤに一致するまで待つ。

【００６２】

いくつかの実施形態において、スレッドが作成される順序は、スレッドが決定論的に実行される順序に対応する。例えば、各スレッドが作成されるとき、スレッドの対応するスレッドＩＤは、スレッドコンテナ６０５に順次格納され得る。トランザクションまたは制御された操作が実行されるとき、実行中のスレッドが、例えばＤＭＰ＿Ｃｏｍｍｉｔ（）５３５などいくつかのＤＭＰ関数を呼び出し、こうした関数は、（第１のスレッドＩＤで開始して）スレッドＩＤが格納されたシーケンスに基づいてスレッドコンテナ６０５に格納されたスレッドＩＤを順次ループすることによって、トークン６１０の値を進めるように動作する。スレッドが終了すると、スレッドの対応するＩＤはスレッドコンテナ６０５から削除されることに留意されたい。

【００６３】

いくつかの実施形態において、スレッドデータ構造は、コミットブロックサイズ６１５を格納する。コミットブロックサイズ６１５は、トークンが進められる前に、スレッドＩＤがトークン６１０の現在の値に一致するスレッドによって実行され得る予め定められた数のトランザクションまたは制御された操作を表す。コミットブロックサイズ６１５は、１つのトランザクションまたは制御された操作からＮ個のトランザクションまたは制御された操作まで及び得る。少なくとも１つの実施形態において、コミットブロックサイズ６１５は、１，０００に等しい。いくつかの実施形態において、コミットブロックサイズ６

１５は、構成可能である。例えば、コミットブロックサイズ６１５は、マルチスレッドアプリケーションの様々なスレッドインターリービングをプログラム的に操作し、テストするように、ソフトウェア開発者によって構成され得る。別の例として、コミットブロックサイズ６１５は、増補されたアプリケーションによって作成され得る最大数のスレッド、および／または増補されたアプリケーションが実行するマルチプロセッシングシステムのプロセッサまたはコアの数に基づいて、自動的に構成され得る。

【００６４】

スレッドによって実行される制御された操作の数をカウントするために様々な技術が使用され得ることを、当業者であれば理解されたい。いくつかの実施形態において、スレッドデータ構造６００は、スレッドコミットブロック６２０を含む。スレッドコミットブロック６２０は、スレッドＩＤが現在のトークンＩＤ６１０に一致するスレッドによって実行された制御された操作の数を表し得る。スレッドが制御された操作を実行するたびに、スレッドコミットブロック６２０の値は、増分され、コミットブロックサイズ６１５と比較される。スレッドコミットブロック６２０の値がコミットブロックサイズ６１５に等しい場合、トークン６０５は、次のスレッドＩＤに進められ、スレッドコミットブロック６２０の値は、ゼロにリセットされる。代替例として、スレッドコミットブロック６２０は、スレッドがその対応するトランザクションをコミットしようと試行する前に残っているブロックの数を表し得る。こうした実施形態において、スレッドコミットブロック６２０は、スレッドコンテナ６０５に格納されたスレッドＩＤを有するスレッドごとに残りのブロックの数を含み得る。次いで、スレッドは、ブロックを実行するたびに、その対応するスレッドコミットブロックを減分し、残りのブロックの数がゼロに等しいとき、そのトランザクションをコミットしようと試みる。

【００６５】

いくつかの実施形態において、スレッドデータ構造は、使用中スレッドブロック６２５を含み、これは、増補されたアプリケーションで実行中のスレッドの数を表す。いくつかの実施形態において、使用中スレッドブロック６２５は、スレッドが作成されるたびに増分される。同様に、使用中スレッドブロック６２５は、スレッドが終了するたびに減分される。一方、他の実施形態では、使用中スレッドブロック６２５は、スレッドコンテナ６０５のサイズに基づいて決定される。図６に示されるスレッドデータ構造６００は様々な方法で変更され得ることを、当業者であれば理解されたい。例えば、いくつかの部分がマージまたは分割されてもよく、いくつかの部分が省略されてもよく、いくつかの部分が追加されてもよい。

【００６６】

図７は、１つまたは複数の実施形態における、スレッドを作成し、決定論的に実行する一例を示す高レベル図である。説明を容易にするために、ある期間にわたるスレッドデータ構造６００の一部分の内容が示される。トークン値６１０によって示されるように、スレッドが作成される順序は、スレッドが決定論的に実行される順序に対応する。

【００６７】

示された例において、最初に作成されたスレッド（「スレッド１」）は、マルチスレッドアプリケーションのメインのアプリケーションスレッドを表す。説明を容易にするために、各スレッドのスレッドＩＤは、スレッドが作成された順序に等しい。すなわち、最初に作成されたスレッドのスレッドＩＤは１、２番目に作成されたスレッドのスレッドＩＤは２、３番目に作成されたスレッドのスレッドＩＤは３、などとなる。時刻T_0とT_1との間で、スレッド１が実行し、スレッド２が作成される。示された例において、スレッドの実行は、指定された数の制御された操作（例えば、コミットブロックサイズ６１５によって指定された量子）によって表される。したがって、図７に示される時間の増分は、必ずしも等しくない。各スレッドによって実行された未制御の操作の数は、異なっていてもよく、またその各実行期間中にスレッドごとに異なっていてもよいことにも留意されたい。

【００６８】

図７に戻って、スレッド１がその量子の実行を終了する前のある時点でスレッド２が作成されたため、時刻Ｔ$_0$とＴ$_1$との間の使用中スレッド６２５の数は２である。その結果、スレッド１が終了すると、トークン６１０は、スレッドコンテナ６０５に格納された次のスレッドＩＤに進められた（すなわち、スレッド２）。

【００６９】

時刻Ｔ$_1$とＴ$_2$との間で、スレッド２が実行し、次いでトークン６１０がスレッド１に戻される。時刻Ｔ$_2$とＴ$_3$との間で、スレッド１が実行し、次いでトークン６１０がスレッド２に進められる。時刻Ｔ$_3$とＴ$_4$との間で、スレッド２が実行し、次いでトークン６１０がスレッド１に戻される。

【００７０】

時刻Ｔ$_4$とＴ$_5$との間で、スレッド１が実行し、スレッド２が作成される。時刻Ｔ$_4$とＴ$_5$との間でスレッド３が作成されるが、スレッド２は、時刻Ｔ$_5$とＴ$_6$との間で実行する。これは、スレッドが作成された順序が、スレッドが実行される順序に対応するからである。その結果、時刻Ｔ$_5$とＴ$_6$との間でスレッド２が実行し、次いで、トークン６１０がスレッド３に進められる。次いで時刻Ｔ$_6$とＴ$_7$との間でスレッド３が実行し、次いでトークン６１０がスレッド１に戻される。

【００７１】

図８は、１つまたは複数の実施形態における、マルチプロセッサコードを決定論的にするためにトランザクショナルメモリシステムを使用する一例を示す高レベルブロック図である。説明を容易にするために、ある期間にわたるスレッドデータ構造６００の一部分の内容が示される。また、説明を容易にするために、スレッドＩＤがスレッド１、スレッド２、スレッド３などのようにスレッドコンテナ６０５に配列されると仮定する。ある期間にわたってトークン値６１０によって示されるように、スレッドがトランザクションをコミットする順序は、決定論的である。説明を容易にするために、トークン６１０の最初の値は、スレッド１のスレッドＩＤに対応する。示された例において、各スレッドによって実行されるトランザクションは、サイズが決定論的である。すなわち、各スレッドは、特定の数のブロックを実行する。説明を容易にするために、コミットブロックサイズ６１５は２である。

【００７２】

示されるように、時刻Ｔ$_0$において、スレッド１～３がトランザクションを開始する。スレッドがその対応するトランザクションを終了した後、スレッドは、そのトランザクションを決定論的にコミットしようと試行する。いくつかの実施形態において、各スレッドは、そのトランザクションが、スレッドにそのトランザクションをコミットさせないようにする競合をもたらしたかどうかを決定する。一方、他の実施形態では、この決定は、そのスレッドＩＤがトークン６１０の現在の値に一致するとき、スレッドによって行われる。例えば、これは、ＳＴＭＶａｌｉｄＴｒａｎｓａｃｔｉｏｎ（）を呼び出すことによって達成され得る。

【００７３】

時刻Ｔ$_1$で、トークン６１０の現在の値は、スレッド１のＩＤに一致する。したがって、示された例では、スレッド１は、そのトランザクションが、それにトランザクションをコミットさせないようにする競合をもたらしたかどうかを決定する。スレッド１およびスレッド２は、同じ共有メモリロケーション（すなわち、アドレスＡ）にアクセスしているが、スレッド１のトランザクションは有効である。これは、スレッド１がアドレスＡに値を格納し、トークン６１０がそのスレッドＩＤに一致するからである。すなわち、（スレッド１によって実行される）Ａの格納は、（スレッド２によって実行される）Ａのロードによって影響されない。その結果、スレッド１は、そのトランザクションをコミットし（例えば、ＳＴＭＣｏｍｍｉｔＴｒａｎｓａｃｔｉｏｎ（）を呼び出すことによって）、次いでトークン６１０は、次のスレッドＩＤに進められる。しかし、トークン６１０は、スレッド２のスレッドＩＤに一致した場合、スレッド１は、そのトランザクションを中止することになる。これは、スレッド１がＡを格納した後、スレッド２がＡをロードしたかも

しれないからである。トークン６１０がスレッド２のＩＤに一致すると仮定すると、スレッド１およびスレッド２は、そのトランザクションを中止することになる。この場合、スレッド２は、スレッド１の中止されたトランザクションを再開する前に、中止されたトランザクションを開始し、コミットすることになる。

【００７４】

示されるように、時刻$Ｔ_{1}$で、スレッド１は、そのトランザクションをコミットし、次いでトークン６１０は、スレッド２に進められる。しかし、スレッド２は、そのトランザクションをコミットすることができない。というのは、スレッド２は、同じトランザクション中にスレッド１によって格納された値をロードしたからである。すなわち、スレッド２は、スレッド１がＡを格納する前に、Ａをロードしたかもしれない。その結果、スレッド２は、そのトランザクションを中止し、再開しなければならない。示された例において、中止されたスレッド以降のスレッドＩＤを有するすべてのスレッドが中止される。一方、他の実施形態では、並行のトランザクションが同じ共有リソースにアクセスした以降のＩＤを有するスレッドのみが中止され、再開される。したがって、示された例では、スレッド３のトランザクションは、中止され、再開される。しかし、他の実施形態において、スレッド３のトランザクションは、中止されない。というのは、そのトランザクションは、並行のトランザクション中にスレッド２またはスレッド１によってアクセスされた共有リソースにアクセスしなかったからである。代わりに、スレッド３は、単にトークン６１０がそのスレッドＩＤに一致するのを待つことになる。スレッドＩＤがトークンに一致するスレッドが、その対応するトランザクションを正常にコミットするまで、トークン６１０は進められないことに留意されたい。

【００７５】

示されるように、時刻$Ｔ_{3}$で、スレッド２～３は、その中止されたトランザクションを再開する。時刻$Ｔ_{4}$で、トークン６１０の現在の値は、スレッド２のＩＤに一致するため、スレッド２は、その再開されたトランザクションが、それにトランザクションをコミットさせない競合をもたらしたかどうかを決定する。示された例において、スレッド２および３の再開されたトランザクションは、任意の共有メモリロケーションにアクセスしない。その結果、時刻$Ｔ_{4}$で、スレッド２は、そのトランザクションを正常にコミットし、次いでトークン６１０は、スレッド３に進められる。時刻$Ｔ_{5}$で、スレッド３は、そのトランザクションを正常にコミットし、次いでトークン６１０は、スレッド１に戻される。

【００７６】

次に、時刻$Ｔ_{6}$で、スレッド１～３は、トランザクションを開始し、プロセスは上述したように続行する。時刻$Ｔ_{6}$で、スレッド１および３の並行のトランザクションによって、スレッド３がそのトランザクションを中止し、再開することに留意されたい。しかし、スレッド１および２は、決定論的にコミットし、トークン６１０は、上述したように、スレッド３に進められる。

【００７７】

図９は、１つまたは複数の実施形態において、マルチスレッドアプリケーションコードを増補するためにファシリティによって実行されるプロセス９００を示すフロー図である。ステップ９０５～９４０で、ファシリティは、マルチスレッドアプリケーションコード５４５の各関数をループする。ステップ９０５で、ファシリティは、関数を選択し、次いでステップ９１０に進む。ステップ９１０で、ファシリティは、ＤＭＰ＿Ｆｕｎｃｔｉｏｎ＿Ｓｔａｒｔ（）関数５１５など、決定論的マルチプロセッシング起動関数を挿入し、次いでステップ９１５に進む。ステップ９１５で、ファシリティは、ＤＭＰ＿Ｉｎｉｔ（）関数５２０など、決定論的マルチプロセッシング初期化関数を挿入し、次いでステップ９２０に進む。ステップ９２０～９３０で、ファシリティは、選択されたアプリケーションの各ブロックをループする。ステップ９２０で、ファシリティは、ブロックを選択し、次いでステップ９２５に進む。ステップ９２５で、ファシリティは、構文解析ブロック関数１０００を呼び出し、次いでステップ９３０に進む。ステップ９３０で、追加のブロックが残っている場合、ファシリティはステップ９２０に進み、そうでない場合、ファシリ

ティはステップ９３５に進む。ステップ９３５で、ファシリティは、ＤＭＰ＿Ｆｕｎｃｔｉｏｎ＿Ｅｎｄ（）５４０など、決定論的プロセッシング終了関数を挿入し、次いでステップ９４０に進む。ステップ９４０で、追加の関数が残っている場合、ファシリティはステップ９０５に進み、そうでない場合、これらのステップは終了する。

【００７８】

図１０は、１つまたは複数の実施形態における、ブロックを構文解析するためにファシリティによって実行されるプロセス１０００を示すフロー図である。ステップ１００５で、ブロックがロードブロックであることをファシリティが決定した場合、ファシリティはステップ１０１０に進み、そうでない場合、ファシリティはステップ１０１５に進む。ステップ１０１０で、ファシリティは、ロードブロックの前にＤＭＰ＿Ｌｏａｄ（）関数５３０への呼び出しを挿入し、次いでファシリティは戻る。ステップ１０１５で、ブロックが格納ブロックであることをファシリティが決定した場合、ファシリティはステップ１０２０に進み、そうでない場合、ファシリティはステップ１０２５に進む。ステップ１０２０で、ファシリティは、格納ブロックの前にＤＭＰ＿Ｓｔｏｒｅ（）関数５２５への呼び出しを挿入し、次いでファシリティは戻る。ステップ１０２５で、ブロックがジャンプブロックであることをファシリティが決定した場合、ファシリティはステップ１０３０に進み、そうでない場合、ファシリティはステップ１０３５に進む。ステップ１０３０で、ファシリティは、ジャンプの前にＤＭＰ＿Ｃｏｍｍｉｔ（）関数５３５への呼び出しを挿入し、ジャンプ先ポイントでＤＭＰ＿Ｉｎｉｔ（）関数５２０への呼び出しを挿入し、次いでファシリティは戻る。ステップ１０３５で、ブロックが関数呼び出しであることをファシリティが決定した場合、ファシリティはステップ１０４０に進み、そうでない場合、ファシリティはステップ１０４５に進む。ステップ１０４０で、ファシリティは、呼び出し前にＤＭＰ＿Ｃｏｍｍｉｔ（）関数５３５への呼び出しを挿入し、呼び出し後ＤＭＰ＿Ｉｎｉｔ（）５２０への呼び出しを挿入し、次いでファシリティは戻る。ステップ１０４５で、ブロックがＩ／Ｏ呼び出しであることをファシリティが決定した場合、ファシリティは、上述したようにステップ１０４０に進み、そうでない場合、ファシリティはステップ１０５０に進む。ステップ１０５０で、ブロックが戻りブロックであることをファシリティが決定した場合、ファシリティはステップ１０５５に進み、そうでない場合、ファシリティは戻る。ステップ１０５５で、ファシリティは、戻りブロック前にＤＭＰ＿Ｃｏｍｍｉｔ（）５３５への呼び出しを挿入し、次いでファシリティは戻る。

【００７９】

図１１は、１つまたは複数の実施形態における、マルチスレッドアプリケーションの増補された関数の制御フローグラフ１１００の一例である。「制御フローグラフ」という用語は、その実行中にアプリケーションによってトラバースされ得るすべてのパスの表現を指す。グラフ１１００における各ノード１１０５～１１３０は、基本ブロック、すなわち、任意のジャンプまたはジャンプターゲットのない直線のコードを表す。ジャンプターゲットは、ブロックを開始し、ジャンプは、ブロックを終了させる。例えば、ＤＭＰ＿Ｉｎｉｔ（）関数５２０を表すブロック１１１０は、ジャンプターゲットである。ブロック１１０５は、入口ブロックを表し、そこを通ってすべての制御がフローグラフに入る。ブロック１１３０は、出口ブロックを表し、そこを通ってすべての制御フローが出る。有向辺、例えば、ブロック１１１５と１１２５との間の辺、１１２０と１１２５との間の辺、およびブロック１１１０とブロック１１１５、１１２０、および１１２５との間の辺は、制御フローにおいてジャンプを表すために使用される。

【００８０】

図１２は、１つまたは複数の実施形態における、決定論的マルチプロセッシング（「ＤＭＰ」）初期化関数１２００を示すフロー図である。例えば、ＤＭＰ初期化関数１２００は、ファシリティがトランザクショナルメモリシステムと共に動作するとき、実行され得る。ＤＭＰ初期化関数は、スレッドがトランザクションの処置を開始または続行できるように、スレッドが初期化された状態であるかどうかを決定するために実行され得る。スレッドが初期化されない（すなわち、スレッドのｉｎｉｔＳｉｔｅ変数の値がゼロに等しい

）場合、その実行は、トークンの値がスレッドＩＤに一致するまで一時停止される。スレッドが初期化された場合、スレッドは実行を続ける。

【００８１】

ステップ１２０５で、ファシリティは、スレッド開始変数（「ｉｎｉｔＳｉｔｅ」）の値がゼロに等しいことを決定した場合、ファシリティはステップ１２１０に進み、そうでない場合、ファシリティは戻る。スレッドの初期化変数は、例えば、スレッドが正常にトランザクションをコミットした後、ゼロに割り当てることができる。ステップ１２１０で、トークンの現在の値がスレッドＩＤに一致することをファシリティが決定した場合、ファシリティはステップ１２１５に進み、そうでない場合、ファシリティは折り返してステップ１２１０に戻る。すなわち、ファシリティは、スレッドＩＤがトークンの値に一致するまで、ステップ１２１０におけるスレッド実行を一時停止する。ステップ１２１５で、ファシリティは、ｉｎｉｔＳｉｔｅ変数を、スレッドがトランザクションを開始するメモリアドレスに割り当て、次いでファシリティは戻る。次いでｉｎｉｔＳｉｔｅ変数は、トランザクションをコミットできない場合、明示的なジャンプアドレスとして使用され得る。

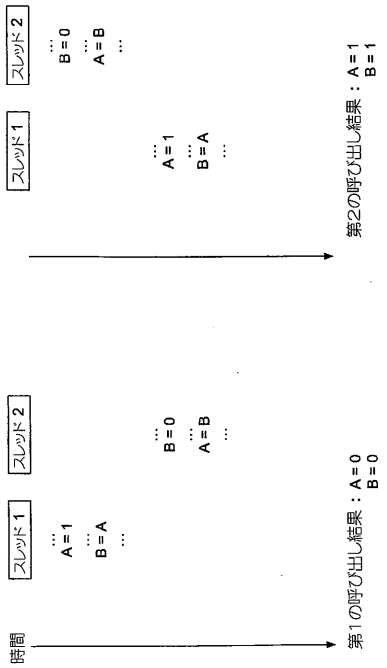
【００８２】

図１３は、１つまたは複数の実施形態における、決定論的マルチプロセッシング（「ＤＭＰ」）コミット関数１３００を示すフロー図である。例えば、ＤＭＰコミット関数１３００は、ファシリティがトランザクショナルメモリシステムと共に動作するとき、実行され得る。ステップ１３０５で、ファシリティは、コミットブロック変数の値を減分し、次いでステップ１３１０に進む。コミットブロック変数は、スレッドによって実行された操作の数をカウントするために使用される。ステップ１３１０で、コミットブロック変数の値がゼロであることをファシリティが決定した場合、ファシリティはステップ１３１５に進み、そうでない場合、ファシリティは戻る。ステップ１３１５で、ファシリティが間に競合があったことを決定した（例えば、トランザクション中に別のスレッドによって書き込まれたロケーションからスレッドが読み取ったため）場合、ファシリティはステップ１３２０に進み、そうでない場合、ファシリティはステップ１３２５に進む。ステップ１３２０で、ファシリティはトランザクションを中止する。ステップ１３２５で、ファシリティは、トランザクションをコミットし、次いでステップ１３３０に進む。ステップ１３３０で、ファシリティは、スレッドのｉｎｔｉＳｉｔｅ変数の値をゼロに割り当て、次いでステップ１３３５に進む。ステップ１３３５で、ファシリティは、コミットブロック変数の値をコミットブロックサイズに割り当てることによって、スレッドのコミットブロック変数の値をリセットし、次いで、ステップ１３４０に進む。ステップ１３４０で、ファシリティは、トークンの値を次のスレッドＩＤの値に割り当てることによって、トークンを進め、次いでファシリティは戻る。

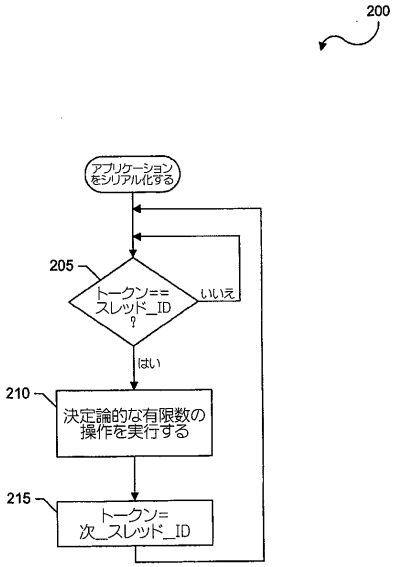
【００８３】

このように、マルチスレッドアプリケーションの決定論的マルチプロセッシングのためのファシリティについて説明した。ファシリティについて、特定の実施形態を参照して説明してきたが、ファシリティは、記載した実施形態に限定されず、添付の特許請求の範囲の意図および範囲内の修正および変更で実施することができることを理解されたい。したがって、明細書および図面は、制限的意味ではなく、例示的意味でみなされるものとする。

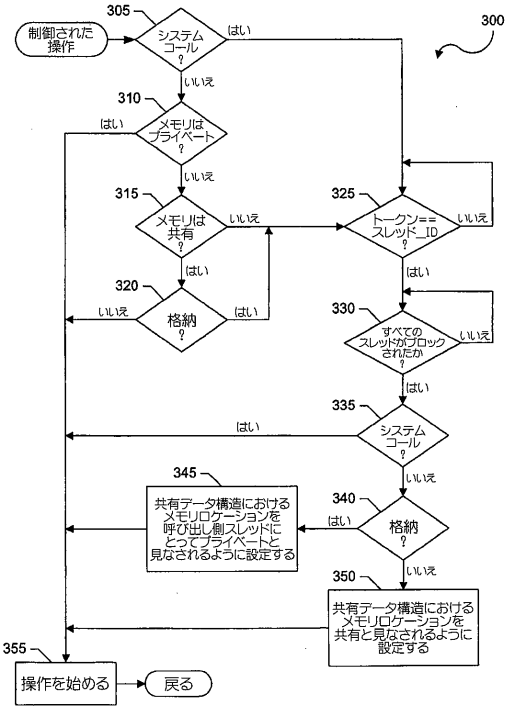
【図１】


スレッド２
スレッド１
B = 0
A = B
A = 1
B = A
第２の呼び出し結果：A = 1 B = 1
スレッド２
スレッド１
B = 0
A = B
A = 1
B = A
第１の呼び出し結果：A = 0 B = 0
時間

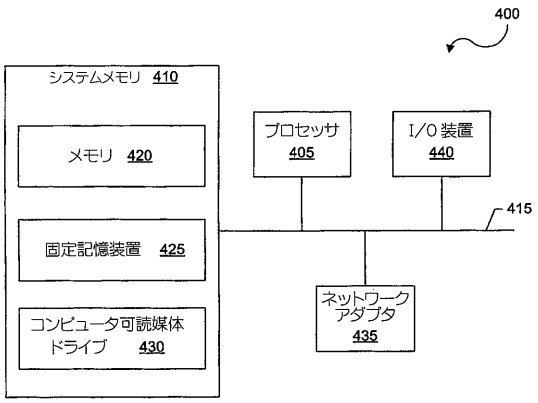
【図２】


200
アプリケーションをシリアル化する
205
トークン== スレッド_ID ?
いいえ
はい
210
決定論的な有限数の操作を実行する
215
トークン= 次_スレッド_ID

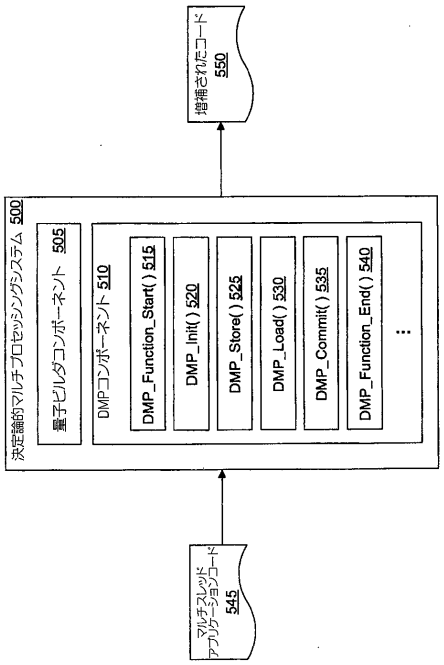
【図３】


300
制御された操作
305
システムコール ?
はい
いいえ
310
メモリはプライベート ?
はい
いいえ
315
メモリは共有 ?
いいえ
はい
320
格納 ?
いいえ
はい
325
トークン== スレッド_ID ?
いいえ
はい
330
すべてのスレッドがブロックされたか ?
いいえ
はい
335
システムコール ?
はい
いいえ
340
格納 ?
はい
いいえ
345
共有データ構造におけるメモリロケーションを呼び出し側スレッドにとってプライベートと見なされるように設定する
350
共有データ構造におけるメモリロケーションを共有と見なされるように設定する
355
操作を始める
戻る

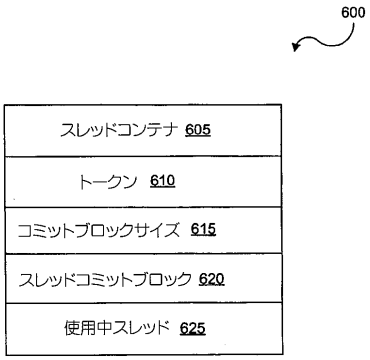
【図４】


400
システムメモリ 410
メモリ 420
固定記憶装置 425
コンピュータ可読媒体ドライブ 430
プロセッサ 405
I/O 装置 440
415
ネットワークアダプタ 435

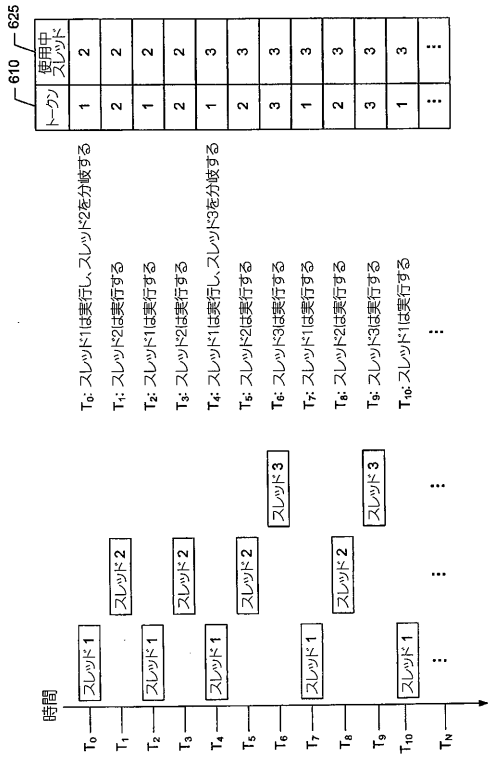
【図５】


決定論的マルチプロセッシングシステム 500
量子ビルダコンポーネント 505
DMPコンポーネント 510
DMP_Function_Start( ) 515
DMP_Init( ) 520
DMP_Store( ) 525
DMP_Load( ) 530
DMP_Commit( ) 535
DMP_Function_End( ) 540
捕捉されたコード 550
マルチスレッドアプリケーションコード 545

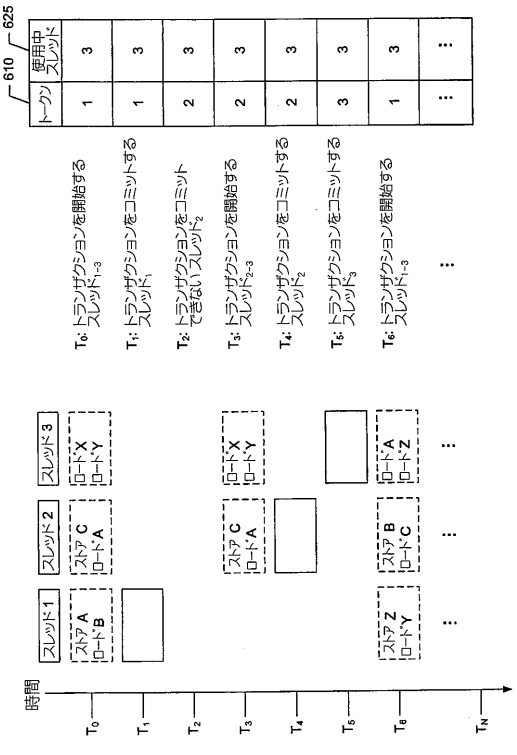
【図６】


600
スレッドコンテナ 605
トークン 610
コミットブロックサイズ 615
スレッドコミットブロック 620
使用中スレッド 625

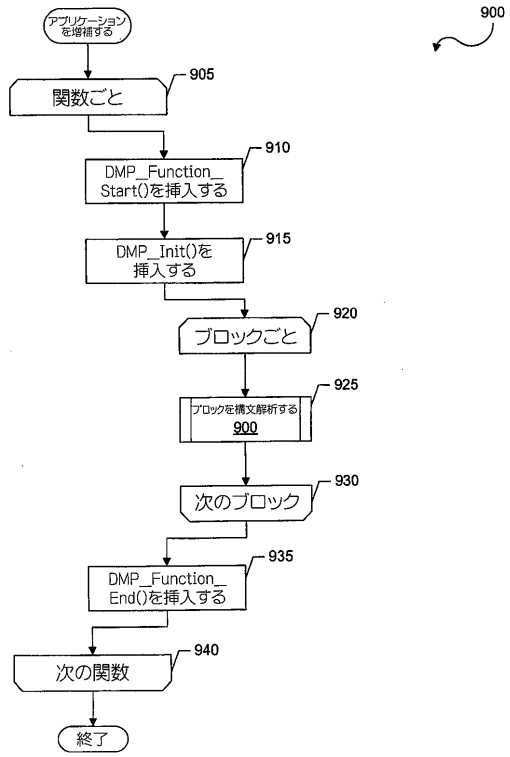
【図７】


610
625
トークン
使用中スレッド
時間
スレッド1
スレッド2
スレッド3
T0: スレッド1は実行し、スレッド2を分岐する
T1: スレッド2は実行する
T2: スレッド1は実行する
T3: スレッド2は実行する
T4: スレッド1は実行し、スレッド3を分岐する
T5: スレッド2は実行する
T6: スレッド3は実行する
T7: スレッド1は実行する
T8: スレッド2は実行する
T9: スレッド3は実行する
T10: スレッド1は実行する

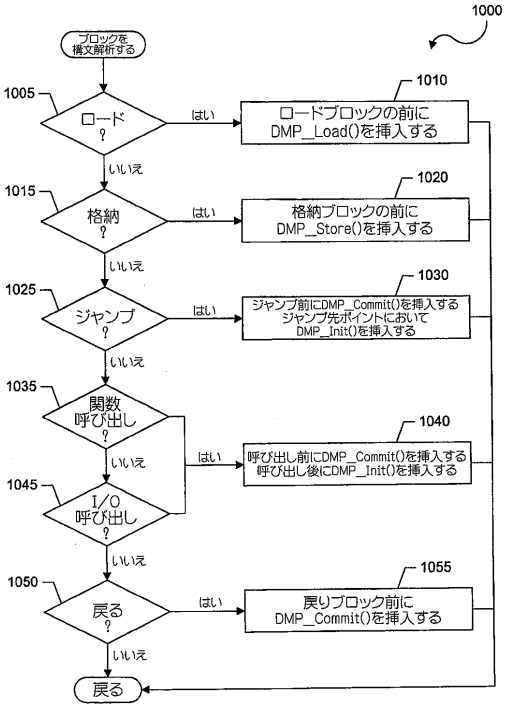
【図８】


610
625
トークン
使用中スレッド
時間
スレッド1
スレッド2
スレッド3
T0: トランザクションを開始するスレッド1-3
T1: トランザクションをコミットするスレッド1
T2: トランザクションをコミットできないスレッド2
T3: トランザクションを開始するスレッド2-3
T4: トランザクションをコミットするスレッド2
T5: トランザクションをコミットするスレッド3
T6: トランザクションを開始するスレッド1-3
ストアA ロードB
ストアC ロードA
ロードX ロードY
ストアC ロードA
ロードX ロードY
ストアB ロードC
ストアZ ロードY
ロードA ロードZ

【図９】



【図１０】



【図１１】

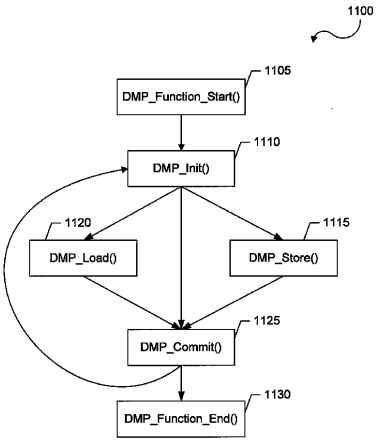
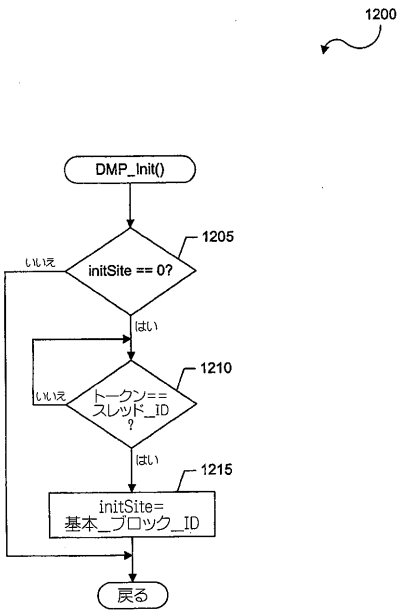
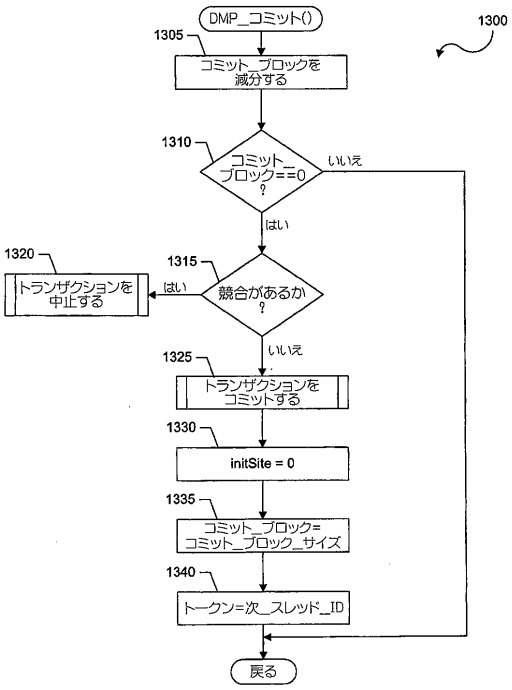


FIG. 11

【図１２】



【図１３】


DMP_コミット()
1300
1305
コミット_ブロックを減分する
1310
コミット_ブロック==0 ?
いいえ
はい
1320
トランザクションを中止する
1315
競合があるか ?
はい
いいえ
1325
トランザクションをコミットする
1330
initSite = 0
1335
コミット_ブロック=コミット_ブロック_サイズ
1340
トークン=次_スレッド_ID
戻る

フロントページの続き

(74)代理人 100119781
弁理士 中村 彰吾
(72)発明者 セゼ，ルイス・エイチ
アメリカ合衆国ワシントン州９８１３６，シアトル，ビーチ・ドライブ・サウス・ウエスト ６３０９
(72)発明者 オスキン，マーク・エイチ
アメリカ合衆国ワシントン州９８１３６，シアトル，ビーチ・ドライブ・サウス・ウエスト ６３０９

審査官 坂庭 剛史

(56)参考文献 特表２００６－５０２５０５（ＪＰ，Ａ）
国際公開第２００７／０６７３９０（WO，Ａ１）
米国特許出願公開第２００７／０１１３２３２（ＵＳ，Ａ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ ９／４５
Ｇ０６Ｆ ９／４６