(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5901835号
(P5901835)
(45)発行日　平成28年4月13日(2016.4.13)　(24)登録日　平成28年3月18日(2016.3.18)

(51)Int.Cl.		F I		
G06F 9/52	(2006.01)	G06F	9/46	472Z
G06F 9/46	(2006.01)	G06F	9/46	430
G06F 12/00	(2006.01)	G06F	12/00	518A

請求項の数 20　(全 31 頁)

(21)出願番号　特願2015-501363(P2015-501363)
(86)(22)出願日　平成26年1月21日(2014.1.21)
(86)国際出願番号　PCT/JP2014/051051
(87)国際公開番号　WO2014/129247
(87)国際公開日　平成26年8月28日(2014.8.28)
審査請求日　平成27年11月27日(2015.11.27)
(31)優先権主張番号　特願2013-33433(P2013-33433)
(32)優先日　平成25年2月22日(2013.2.22)
(33)優先権主張国　日本国(JP)

早期審査対象出願

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
INTERNATIONAL BUSINESS MACHINES CORPORATION
アメリカ合衆国10504　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
New Orchard Road, Armonk, New York 10504, United States of America
(74)代理人　100108501
弁理士　上野　剛史

最終頁に続く

(54)【発明の名称】アボート削減方法、アボート削減装置、及びアボート削減プログラム

(57)【特許請求の範囲】
【請求項１】
ハードウェアトランザクショナルメモリを用いたプログラム実行においてアボートの回数を削減するためのコンピュータによる方法であって、
（ａ）前記コンピュータが、トランザクションブロックの実行中におけるランタイムヘルパーの呼び出しに応答して、前記ランタイムヘルパーを実行するステップと、
（ｂ）前記ランタイムヘルパーの呼び出しに起因したアボートに応答して、前記コンピュータが、アボート・ハンドラを実行するステップと、
（ｃ）ステップ（ｂ）の前記アボート・ハンドラの実行の後、前記コンピュータが、前記トランザクションブロックに対応する非トランザクションパスを実行するステップとを含み、
前記ランタイムヘルパーの実行は、ランタイムヘルパーの種別を示すID情報を前記アボート・ハンドラに渡すための処理を含み、
前記アボート・ハンドラの実行は、アボートの原因となった前記ランタイムヘルパーの前記ID情報を取得し、特定の種別のランタイムヘルパーに対して前記トランザクションブロックを無効化する処理を含む、
アボート削減方法。
【請求項２】
前記特定の種別のランタイムヘルパーに対して前記トランザクションブロックを無効化する処理は、非トランザクションパスにおいても実行不可能な第１の種別のランタイムヘ

ルパーに対し無条件で前記トランザクションブロックを無効化する処理と、非トランザクションパスにおいては実行可能な第２の種別のランタイムヘルパーに対し、一度の前記非トランザクションパスを実行中に前記第２の種別のランタイムヘルパーが呼び出されないことを条件として前記トランザクションブロックを無効化する処理とを含む、請求項１に記載のアボート削減方法。

【請求項３】

前記第１の種別のランタイムヘルパーは、実行時コード修正を含み、前記第２の種別のランタイムヘルパーは、実行時コンパイルとクラスロードとを含む、請求項２に記載のアボート削減方法。

【請求項４】

（ｄ）条件付きで前記トランザクションブロックが無効化された場合において、前記コンピュータが、プログラムが定常状態に達したことに応答して前記トランザクションブロックを再有効化するステップを更に含む、請求項３に記載のアボート削減方法。

【請求項５】

前記ランタイムヘルパーの実行は、前記ランタイムヘルパーが前記第２の種別のランタイムヘルパーである場合に、その処理内容を、非トランザクション書き込み命令を用いて記録し、及び、前記ID情報を引数としてトランザクションアボート命令を発行する処理を含み、前記アボート・ハンドラの実行は、前記第２の種別のランタイムヘルパーに対して前記トランザクションブロックを無効化する場合に、前記記録した処理内容を表に登録し、及び、（ｅ）前記コンピュータが、前記表に登録された処理内容が実行されたことに応答して、前記トランザクションブロックを再有効化するステップを更に含む、請求項３に記載のアボート削減方法。

【請求項６】

前記第２の種別のランタイムヘルパーが実行時コンパイルである場合、前記処理内容は、コンパイルすべきメソッドの識別情報であり、前記第２の種別のランタイムヘルパーがクラスロードである場合、前記処理内容は、ロードすべきクラスの識別子である請求項５に記載のアボート削減方法。

【請求項７】

前記アボート・ハンドラは、スレッドローカルヒープ割付と、非同期ガーベジコレクションチェックとを、前記特定の種別以外のランタイムヘルパーとして判断する、請求項３に記載のアボート削減方法。

【請求項８】

（ｆ）前記コンピュータが、プログラムの実行開始において全てのトランザクションブロックを無効化するステップと、（ｇ）前記コンピュータが、前記プログラムの実行開始から所定の期間が経過することに応答して、前記全てのトランザクションブロックを有効化するステップとを更に含む、請求項１に記載のアボート削減方法。

【請求項９】

前記ランタイムヘルパーの実行は、トランザクションアボート命令を発行する処理を含み、前記ID情報をアボート・ハンドラに渡すための処理は、前記ID情報を前記トランザクションアボート命令の引数とする処理を含む、請求項１に記載のアボート削減方法。

【請求項１０】

前記ID情報をアボート・ハンドラに渡すための処理は、非トランザクション書き込み命令を用いて前記ID情報を記録する処理を含み、該処理の後前記ランタイムヘルパーは本来のランタイムヘルパーの処理を行う、請求項１に記載のアボート削減方法。

【請求項１１】

前記ランタイムヘルパーの実行は、該ランタイムヘルパーが前記特定の種別と異なる他の種別のランタイムヘルパーであることを条件に、その実行に必要な情報を、非トランザクション書き込み命令を用いて記録する処理を含み、前記アボート・ハンドラの実行は、前記他の種別のランタイムヘルパーに対し、記録された前記必要な情報を用いてその実行を行う処理を含み、及び、該処理の後ステップ（ｃ）の代わりに、（ｈ）前記コンピュー

タが、前記トランザクションブロックを再実行するステップを含む、請求項１に記載のアボート削減方法。
【請求項１２】
前記ランタイムヘルパーの実行は、該ランタイムヘルパーが前記特定の種別と異なる他の種別のランタイムヘルパーであることを条件に、その実行に必要な情報を、非トランザクション書き込み命令を用いて記録する処理と、その実行をキャンセルする処理とを含み、（ｉ）前記コンピュータが、前記トランザクションブロックを実行後、前記他の種別のランタイムヘルパーに対し、記録された前記必要な情報を用いてその実行処理を行うステップを更に含む、請求項１に記載のアボート削減方法。
【請求項１３】
前記特定の種別のランタイムヘルパーはクラスロードを含み、前記他の種別のランタイムヘルパーは、実行時コンパイルであり、前記実行に必要な情報はコンパイルすべきメソッドの識別情報である、請求項１１又は１２に記載のアボート削減方法。
【請求項１４】
前記特定の種別のランタイムヘルパーはクラスロードを含み、前記他の種別のランタイムヘルパーは、実行時コード修正であり、前記実行に必要な情報は実行時コード修正されるべきアドレス及びデータである、請求項１１又は１２に記載のアボート削減方法。
【請求項１５】
前記ランタイムヘルパーが実行時コンパイルである場合に、前記ランタイムヘルパーの実行は、コンパイル対象のコードに含まれるトランザクションブロック内の実行時コード修正の呼び出しを、トランザクションアボート命令で置換し、該トランザクションアボート命令のアドレスから前記実行時コード修正に必要な情報への対応表を作成する処理を含み、（ｊ）前記コンピュータが、置換された前記トランザクションアボート命令の実行に応答してアボート・ハンドラを実行した後に、前記トランザクションブロックを再実行するステップを更に含み、前記アボート・ハンドラの実行は、前記トランザクションアボート命令のアドレスから前記対応表を引いて必要な情報を取得することにより前記実行時コード修正を行い、前記トランザクションアボート命令をｎｏｐ命令で上書きする処理を含む、請求項１に記載のアボート削減方法。
【請求項１６】
前記アボート・ハンドラの実行は、前記ランタイムヘルパーが実行時コード修正であることを条件に、アボートが起きた命令アドレスを含むページを書き込み禁止にし、及び、アボートの原因が書込み保護の例外であることを条件に、前記書込み禁止を無効化し、書込み保護が起きた位置の書き込み命令を識別することにより実行時コード修正されるべきアドレスとデータを取得して、前記実行時コード修正を行い、（ｋ）前記コンピュータが、前記アボート・ハンドラによる書き込み禁止処理又は前記実行時コード修正の後ステップ（ｃ）の代わりに、前記トランザクションブロックを再実行するステップを更に含む、請求項１に記載のアボート削減方法。
【請求項１７】
コンピュータに、請求項１乃至１６のいずれかに一項に記載のアボート削減方法の各ステップを実行させるためのアボート削減プログラム。
【請求項１８】
請求項１乃至１６のいずれかに一項に記載のアボート削減方法の各ステップを実行するように適合された手段を備える、アボート削減装置。
【請求項１９】
ハードウェアトランザクショナルメモリを用いたプログラム実行においてアボートの回数を削減するための装置であって、
実行部と、
複数のランタイムヘルパーと、
アボート・ハンドラとを含み、
前記ランタイムヘルパーの各々は、前記実行部によるトランザクションブロックの実

行中に呼び出されることに応答して、ランタイムヘルパーの種別を示すID情報を前記アボート・ハンドラに渡すための処理を行い、

前記アボート・ハンドラは、アボートの原因となった前記ランタイムヘルパーの前記ID情報を取得し、特定の種別のランタイムヘルパーに対して前記トランザクションブロックを無効化する処理を行い、及び、

前記実行部は、前記ランタイムヘルパーの呼び出しに起因した前記アボート・ハンドラの処理の後に、前記トランザクションブロックに対応する非トランザクションパスを実行する、アボート削減装置。

【請求項２０】

前記特定の種別のランタイムヘルパーに対して前記トランザクションブロックを無効化する処理は、非トランザクションパスにおいても実行不可能な第１の種別のランタイムヘルパーに対し無条件で前　記トランザクションブロックを無効化する処理と、非トランザクションパスにおいては実行可能な第２の種別のランタイムヘルパーに対し、一度の前記非トランザクションパスを実行中に前記第２の種別のランタイムヘルパーが呼び出されないことを条件として前記トランザクションブロックを無効する処理とを含み、前記実行部は、条件付きで前記トランザクションブロックが無効化された場合に、プログラムが定常状態に達したことに応答して前記トランザクションブロックを再有効化する、請求項１９に記載のアボート削減装置。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、ハードウェアトランザクショナルメモリ（Hardware Transactional Memory：HTM）を用いたプログラム実行におけるアボート回数を削減するための技術に関し、より詳細には、トランザクションブロックを実行中に所定の機能が呼び出されることにより引き起こされるアボートの回数を削減するための技術に関する。

【背景技術】

【０００２】

マルチコア・プロセッサでは、並列に実行されるスレッドがデータを共有して扱う共有メモリ型の並列プログラムが用いられることが多い。このとき、アクセスが競合しないようにする手法として、HTMと呼ばれる手法がある。HTMは、ハードウェアとして実装されるトランザクショナルメモリである。

【０００３】

HTMでは、実行しているスレッドの各々がそのコピーをローカルにもち、処理が終了した時点で共有リソースの値が他のスレッドにより書き換えられていないことを条件にローカルにもつコピーに対する処理結果を一気に書き込む。この処理はコミットと呼ばれる。

共有リソースの値が他のスレッドにより書き換えられている場合、処理そのものが廃棄される。トランザクションはいつでも中止されることがあり（アボートという）、アボートされると、そのトランザクションがそれまでに行った共有リソースへの変更は取り消される（ロールバックという）。なお、トランザクションとは、１つのスレッドで実行される一連の不可分な処理をまとめたものである。

【０００４】

最近では、HTMを搭載した商用マシンも市場に現れている。Java（登録商標）仮想マシン（VM）などのマネージドランタイム環境では、ロック除去（Lock elision）や部分インライン展開（partial inlining）、投機的チェック除去（speculative check elimination）、HTMベース並列ライブラリ（HTM-basedconcurrent library）など、種々の最適化のためにHTMが利用される。マネージドランタイム環境は、JITコンパイラやメモリ・アロケーター等の様々なランタイムヘルパーに依存している。ここで、ランタイムヘルパーとは、インタープリタとJITコンパイルされた実行コードには含まれないが、実行時にインタープリタとJITコンパイルされたコードから呼び出される仮想マシンの機能をいう。しかしながらこれらランタイムヘルパーの中には、トランザクション実行中に呼び出され実行

されるとアボートを引き起こすものもある。
【０００５】
アボートに対処する従来技術として、特許文献１～３及び非特許文献１～３が存在する。特許文献１及び非特許文献２は、HTMのハードウェアが提供するアボート理由の情報に応じてアボート・ハンドラの中で適切な処理を行う技術を開示する。また、特許文献２は、あるトランザクションを実行中に一時的にトランザクションを中断して任意の処理を行い、その後トランザクションを開始する技術を開示する。特許文献３は、複数のトランザクショナルメモリの実装を実行時に切り替える技術を開示する。
【０００６】
また非特許文献１は、トランザクションがアボートした場合に、複数回リトライした後非トランザクションパスを実行し、その後再びトランザクションを実行する技術を開示する。非特許文献３は、トランザクション同士の衝突が原因でアボートした場合において、トランザクションの再実行を遅延させる技術を開示する。
【先行技術文献】
【特許文献】
【０００７】
【特許文献１】米国特許出願公開第２０１０／１３１９５３号明細書
【特許文献２】米国特許出願公開第２０１０／３３２８０７号明細書
【特許文献３】米国特許出願公開第２００９／１７２３０６号明細書
【非特許文献】
【０００８】
【非特許文献１】Dave Dice, Yossi Lev, Mark Moir, Dan Nussbaum, ”Early Experience with a Commercial Hardware Transactional Memory Implementation”, In Sun Microsystems Technical Report TR-2009-180, 2009
【非特許文献２】Amy Wang, Peng Wu, Matthew Gaudet, Jose Nelson Amaral, Martin Ohmacht, Christopher Barton, Raul Silvera Maged Michael,”Evaluation of Blue Gene/Q Hardware Support for Transactional Memories”,In PACT 2012
【非特許文献３】Mohammad Ansari, Mikiel Lujan, Chris Kirkham, Kim Jarvis, Christos Kotselidis, lan Watson, “Steal-on-abort:Dynamic Transaction Reordering to Reduce Conflicts inTransactional Memory”, 4th Int’l ACM Sigplan Conferenceon High Performance Embedded Architectures and Compilers (HiPEAC’09),ACMPress, pp. 4-18, 2009.
【発明の概要】
【発明が解決しようとする課題】
【０００９】
しかしながら、上述したランタイムヘルパーの呼び出しによって引き起こされるアボートは、単純にトランザクションをリトライしただけでは解決しない。
【００１０】
例えばJITコンパイラは、あるメソッドが所定回数実行された場合に呼び出されるものである。しかしその呼び出しがトランザクション中になされると、JITコンパイラが使用する作業領域のサイズが、ハードウェアによりサポートされるトランザクションサイズを超えるため、アボートが起きる。
【００１１】
また、シンボル解決（Symbol resolution）又はコード修正（codepatching）は、実行時にメソッド内のシンボルをオンデマンドで解決し、結果としてシンボルを参照するコードを修正するものである。そのためシンボル解決がトランザクション中に実行されると、アボートが起きる。
【００１２】
また、スレッドローカルヒープ（thread-local heap: TLH）割り付けは、しばしばプログラムによって呼び出され、ヒープからTLHを割り付け、これをゼロでクリアするもので

ある。しかしTLH割り付けがトランザクション中になされると、ゼロクリーニング（zero-clearing）が上述したトランザクションサイズの上限を超えてオーバーフローするため、アボートが起きる。

【００１３】

また、非同期ガーベジコレクション（GC）チェックの実行は、stop-the-world GCによって設定されたフラグを定期的にチェックする各アプリケーションがスリープすることによって開始される。しかしながらトランザクション中におけるスリープのためのシステムコール呼び出しは許されないため、トランザクション中に上記フラグの設定が検出されると、アボートが起きる。

【００１４】

また、クラスロードは、プログラムの実行中にオンデマンドでクラスをロードするものである。もしトランザクション中にクラスロードが起きると、I/Oがトランザクションをアボートする。

【００１５】

このようにランタイムヘルパーによって引き起こされるアボートの原因は様々であり、かつ単純にトランザクションをリトライするだけではアボートし続けるため、従来技術によって解決することは困難である。

【００１６】

特許文献１及び非特許文献２の技術は、HTMのハードウェアが提供するアボート理由を利用するが、例えばトランザクション中に非同期GCチェック又はクラスロードが呼ばれたとすると、アボートの理由はハードウェアの観点からはどちらもシステムコール呼び出しである。そのため特許文献１及び非特許文献２の技術では、両者の区別がつかずそれぞれに対し異なる適切な処置を取ることができない。

【００１７】

また、特許文献２の技術によれば、サスペンド命令及びレジューム命令でアボートを引き起こし得るランタイムヘルパーを囲むことによりアボートを避けることが可能である。しかしながら特許文献２の技術の利用は、サスペンド命令及びレジューム命令がハードウェアによりサポートされていることが前提となるため、サポートのないハードウェアについては依然として問題が残る。

【００１８】

また特許文献３の技術によれば、複数のトランザクショナルメモリの実装を実行時に切り替えることが必要となる。しかし切り替えのタイミングとして特許文献３において記載されているのはパフォーマンス、現在のワークロード、及びリソースの配置のみであるため、アボートを引き起こし得るランタイムヘルパーの種別に応じて適切な処理を行うことは出来ない。

【００１９】

また、非特許文献１の技術は、所定回数のアボートを条件として非トランザクションパスを実行し、その後再度トランザクションを実行する手法である。しかしアボートを引き起こし得るランタイムヘルパーの種別によっては所定回数のアボートが無駄であったり、非トランザクションパスの実行によっても問題が解決しなかったりする。従って、ランタイムヘルパーの種別に応じて適切な処理を行うことが必要であるが、そのための手法は非特許文献１には記載されていない。

【００２０】

また、非特許文献３の技術は、トランザクション同士の衝突が原因でアボートした場合にトランザクションの再実行をしばらく遅らせる手法である。従って、ランタイムヘルパーの呼び出しに起因するアボートに対して該技術を適用してもアボートが解消されることはない。

【００２１】

この発明は、上記の問題点を解決するためになされたものであって、トランザクションブロックを実行中にランタイムヘルパーが呼び出されることに起因するアボートの回数を

削減するための技術を提供することを目的とする。
【課題を解決するための手段】
【００２２】
上記課題を解決するために、本発明の１態様によれば、以下のような、ハードウェアトランザクショナルメモリを用いたプログラム実行においてアボートの回数を削減するためのコンピュータによる方法が提供される。そのアボート削減方法は、（ａ）前記コンピュータが、トランザクションブロックの実行中におけるランタイムヘルパーの呼び出しに応答して、前記ランタイムヘルパーを実行するステップと、（ｂ）前記ランタイムヘルパーの呼び出しに起因したアボートに応答して、前記コンピュータが、アボート・ハンドラを実行するステップと、（ｃ）ステップ（ｂ）の前記アボート・ハンドラの実行の後、前記コンピュータが、前記トランザクションブロックに対応する非トランザクションパスを実行するステップとを含む。そして、前記ランタイムヘルパーの実行は、ランタイムヘルパーの種別を示すID情報を前記アボート・ハンドラに渡すための処理を含み、前記アボート・ハンドラの実行は、アボートの原因となった前記ランタイムヘルパーの前記ID情報を取得し、特定の種別のランタイムヘルパーに対して前記トランザクションブロックを無効化する処理を含む。なお、上述したように、ランタイムヘルパーとは、インタープリタとJITコンパイルされた実行コードには含まれないが、実行時にインタープリタとJITコンパイルされたコードから呼び出される仮想マシンの機能をいう。また、トランザクションブロックとは、トランザクション開始命令とトランザクション終了命令で囲まれたプログラム領域であり、ハードウェアトランザクショナルメモリを用いたプログラム実行において、トランザクションとして実行されるプログラム領域である。
【００２３】
好ましくは、前記特定の種別のランタイムヘルパーに対して前記トランザクションブロックを無効化する処理は、非トランザクションパスにおいても実行不可能な第１の種別のランタイムヘルパーに対し無条件で前記トランザクションブロックを無効化する処理と、非トランザクションパスにおいては実行可能な第２の種別のランタイムヘルパーに対し、一度の前記非トランザクションパスを実行中に前記第２の種別のランタイムヘルパーが呼び出されないことを条件として前記トランザクションブロックを無効化する処理とを含む。
【００２４】
ここで、第１の種別のランタイムヘルパーは、実行時コード修正を含み、前記第２の種別のランタイムヘルパーは、実行時コンパイルとクラスロードとを含む。
【００２５】
より好ましくは、上記アボート削減方法は、（ｄ）条件付きで前記トランザクションブロックが無効化された場合において、前記コンピュータが、プログラムが定常状態に達したことに応答して前記トランザクションブロックを再有効化するステップを更に含む。
【００２６】
上記トランザクションブロックの再有効化の構成に代えて以下の構成を採用してもよい。即ち、前記ランタイムヘルパーの実行は、前記ランタイムヘルパーが前記第２の種別のランタイムヘルパーである場合に、その処理内容を、非トランザクション書き込み命令を用いて記録し、及び、前記ID情報を引数としてトランザクションアボート命令を発行する処理を含む。また、前記アボート・ハンドラの実行は、前記第２の種別のランタイムヘルパーに対して前記トランザクションブロックを無効化する場合に、前記記録した処理内容を表に登録する。そして、上記アボート削減方法は、（ｅ）前記コンピュータが、前記表に登録された処理内容が実行されたことに応答して、前記トランザクションブロックを再有効化するステップを更に含む。
【００２７】
ここで、前記第２の種別のランタイムヘルパーが実行時コンパイルである場合、前記処理内容は、コンパイルすべきメソッドの識別情報であり、前記第２の種別のランタイムヘルパーがクラスロードである場合、前記処理内容は、ロードすべきクラスの識別子である

。

【００２８】

また好ましくは、前記アボート・ハンドラは、スレッドローカルヒープ割付と、非同期ガーベジコレクションチェックとを、前記特定の種別以外のランタイムヘルパーと判断する。

【００２９】

また好ましくは、上記アボート削減方法は、（ｆ）前記コンピュータが、プログラムの実行開始において全てのトランザクションブロックを無効化するステップと、（ｇ）前記コンピュータが、前記プログラムの実行開始から所定の期間が経過することに応答して、前記全てのトランザクションブロックを有効化するステップとを更に含む。

【００３０】

また好ましくは、前記ランタイムヘルパーの実行は、トランザクションアボート命令を発行する処理を含み、前記ID情報をアボート・ハンドラに渡すための処理は、前記ID情報を前記トランザクションアボート命令の引数とする処理を含む。これに代えて、前記ID情報をアボート・ハンドラに渡すための処理は、非トランザクション書き込み命令を用いて前記ID情報を記録する処理を含み、該処理の後前記ランタイムヘルパーは本来のランタイムヘルパーの処理を行ってもよい。

【００３１】

また好ましくは、前記ランタイムヘルパーの実行は、該ランタイムヘルパーが前記特定の種別と異なる他の種別のランタイムヘルパーであることを条件に、その実行に必要な情報を、非トランザクション書き込み命令を用いて記録する処理を含む。また、前記アボート・ハンドラの実行は、前記他の種別のランタイムヘルパーに対し、記録された前記必要な情報を用いてその実行を行う処理を含む。そして、該処理の後ステップ（ｃ）の代わりに、（ｈ）前記コンピュータが、前記トランザクションブロックを再実行するステップを含む。

【００３２】

上記ランタイムヘルパーの実行遅延のための構成に代えて以下の構成を採用してもよい。即ち、前記ランタイムヘルパーの実行は、該ランタイムヘルパーが前記特定の種別と異なる他の種別のランタイムヘルパーであることを条件に、その実行に必要な情報を、非トランザクション書き込み命令を用いて記録する処理と、その実行をキャンセルする処理とを含む。そして、上記アボート削減方法は、（ｉ）前記コンピュータが、前記トランザクションブロックを実行後、前記他の種別のランタイムヘルパーに対し、記録された前記必要な情報を用いてその実行処理を行うステップを更に含む。

【００３３】

ここで、前記特定の種別のランタイムヘルパーはクラスロードを含み、前記他の種別のランタイムヘルパーは、実行時コンパイルであり、前記実行に必要な情報はコンパイルすべきメソッドの識別情報である。また、前記特定の種別のランタイムヘルパーはクラスロードを含み、前記他の種別のランタイムヘルパーは、実行時コード修正であり、前記実行に必要な情報は実行時コード修正されるべきアドレス及びデータである。

【００３４】

また好ましくは、前記ランタイムヘルパーが実行時コンパイルである場合に、前記ランタイムヘルパーの実行は、コンパイル対象のコードに含まれるトランザクションブロック内の実行時コード修正の呼び出しを、トランザクションアボート命令で置換し、該トランザクションアボート命令のアドレスから前記実行時コード修正に必要な情報への対応表を作成する処理を含む。そして上記アボート削減方法は、（ｊ）前記コンピュータが、置換された前記トランザクションアボート命令の実行に応答してアボート・ハンドラを実行した後に、前記トランザクションブロックを再実行するステップを更に含む。ここで前記アボート・ハンドラの実行は、前記トランザクションアボート命令のアドレスから前記対応表を引いて必要な情報を取得することにより前記実行時コード修正を行い、前記トランザクションアボート命令をｎｏｐ命令で上書きする処理を含む。

【００３５】
上記コンパイラによる実行時コード修正の置換構成に代えて、以下の構成を採用してもよい。即ち、
前記アボート・ハンドラの実行は、前記ランタイムヘルパーが実行時コード修正であることを条件に、アボートが起きた命令アドレスを含むページを書き込み禁止にし、及び、アボートの原因が書込み保護の例外であることを条件に、前記書込み禁止を無効化し、書込み保護が起きた位置の書き込み命令を識別することにより実行時コード修正されるべきアドレスとデータを取得して、前記実行時コード修正を行う処理をふくむ。そして上記アボート削減方法は、（ｋ）前記コンピュータが、前記アボート・ハンドラによる書き込み禁止処理又は前記実行時コード修正の後ステップ（ｃ）の代わりに、前記トランザクションブロックを再実行するステップを更に含む。
【００３６】
以上、アボート削減方法として本発明を説明したが、本発明は、上記説明したアボート削減方法の各ステップをコンピュータに実行させるためのアボート削減プログラムとして把握することもできる。また、そのようなアボート削減方法の各ステップを実行するように適合された手段を備えるアボート削減装置として把握することもできる。
【発明の効果】
【００３７】
本発明によれば、ランタイムヘルパーの実行において、ランタイムヘルパーの種別を示すID情報をアボート・ハンドラに渡すための処理が行われるので、アボート・ハンドラにおいてランタイムヘルパーの種別に応じた処理が最初から可能となり、特定の種別のランタイムヘルパーに対しては、トランザクションブロックを無効化する処理を行うことができる。結果として、トランザクションブロックを実行中にランタイムヘルパーが呼び出されることに起因するアボートの回数を、無駄なく効率的に削減することができる。本発明のその他の効果については、各実施の形態の記載から理解される。
【図面の簡単な説明】
【００３８】
【図１】本発明の実施の形態に係るアボート削減装置を実現するのに好適なコンピュータ・システム１００のハードウェア構成の一例を示した図である。
【図２】図１に示すコンピュータ・システム１００のソフトウェアの構成の一例を示す図である。
【図３】本発明の実施の形態に係るアボート削減処理全体の流れの一例を示すフローチャートである。
【図４】ランタイムヘルパーによる処理の流れの一例を示すフローチャートである。
【図５】図５（ａ）は、アボート・ハンドラによる処理の前半の処理の流れの一例を示すフローチャートである。図５（ｂ）は、アボート・ハンドラによる処理の後半の処理の流れの一例を示すフローチャートである。
【図６】非トランザクション・パスの実行処理の流れの一例を示すフローチャートである。
【図７】図７（ａ）は、トランザクションブロックの有効化処理の流れの一例を示すフローチャートである。図７（ｂ）は、トランザクションブロックの有効化処理の流れの他の例を示すフローチャートである。
【図８】ランタイムヘルパーによる処理の流れの他の例を示すフローチャートである。
【図９】ランタイムヘルパーによる処理の流れの他の例を示すフローチャートである。
【図１０】アボート・ハンドラによる処理の一部の流れの他の例を示すフローチャートである。
【図１１】ＪＩＴコンパイラによる処理の流れの一例を示すフローチャートである。
【図１２】アボート・ハンドラによる処理の一部の流れの他の例を示すフローチャートである。
【図１３】アボート・ハンドラによる処理の一部の流れの他の例を示すフローチャートで

ある。
【図１４】ランタイムヘルパーによる処理の流れの他の例を示すフローチャートである。
【図１５】アボート・ハンドラによる処理の一部の流れの他の例を示すフローチャートである。
【図１６】トランザクションブロックの有効化処理の流れの他の例を示すフローチャートである。
【図１７】図１７（ａ）は、従来技術と本発明とでスレッド数に応じたスループットを比較した実験結果を示す図である。図１７（ｂ）は、従来技術と本発明とでスレッド数に応じたスループットを比較した実験結果を示す図である。
【発明を実施するための形態】
【００３９】
以下、本発明を実施するための形態を図面に基づいて詳細に説明するが、以下の実施形態は特許請求の範囲にかかる発明を限定するものではなく、また実施形態の中で説明されている特徴の組み合わせの全てが発明の解決手段に必須であるとは限らない。なお、実施の形態の説明の全体を通じて同じ要素には同じ番号を付している。
【００４０】
図１は、本発明を実施するのに好適なコンピュータ・システム１００のハードウェア構成の一例を示す。コンピュータ・システム１００は、バス１０６に接続されたメインCPU（中央処理装置）１０２とメイン・メモリ１０４を含んでいる。CPU１０２は、並列処理が可能なマルチプロセッサとされ、HTM１０２ａを含んで構成される。HTM１０２ａは、ハードウェアとして実装されるトランザクショナルメモリで、上述したように、実行しているスレッドの各々がそのコピーをローカルにもち、処理が終了した時点でその共有リソースの値が変更されていないことを確認し、処理結果を一気に書き込む。また、HTM１０２ａは、他のスレッドにより書き換えられている場合は、それらの値が変更されているため、処理そのものを廃棄してやり直すことができる。更にHTM１０２ａによるトランザクションは、いつでもアボートすることがあり、アボートすると、そのトランザクションがそれまでに行った共有データへの変更は取り消され、ロールバックする。なお、トランザクションとは、１つのスレッドで実行される一連の不可分な処理をまとめたものである。
【００４１】
バス１０６には、ディスプレイ・コントローラ１０８を介して、ディスプレイ１１０、例えば液晶ディスプレイ（ＬＣＤ）が接続されうる。ディスプレイ１１０は、コンピュータの管理のために、通信回線を介してネットワークに接続されたコンピュータについての情報と、そのコンピュータ上で動作中のソフトウェアについての情報を、適当なグラフィック・インタフェースで表示するために使用される。
【００４２】
バス１０６にはまた、ＳＡＴＡ又はＩＤＥコントローラ１１２を介して、ディスク１１４、例えばシリコン・ディスク又はハードディスクが接続されうる。バス１０６にはまた、ＳＡＴＡ又はＩＤＥコントローラ１１２を介して、任意的に、ドライブ１１６、例えばＣＤ、ＤＶＤまたはＢＤドライブが接続されうる。バス１０６にはさらに、任意的に、キーボード・マウスコントローラ１１８又はＵＳＢバス（図示せず）を介して、キーボード１２０及びマウス１２２が接続されうるが、本発明を実施する上では必要ない。
【００４３】
ディスク１１４には、オペレーティング・システム、Java（登録商標）仮想マシン（VM）等の仮想マシンを提供するプログラム、その他のプログラム及びデータが、メイン・メモリ１０４にロード可能なように記憶されている。
【００４４】
本発明の実施形態によるアボート削減プログラムは、仮想マシンを提供するプログラムの一部、即ち、インタープリタ、コンパイル済みコード実行部、アボート・ハンドラ、及び複数のランタイムヘルパーを、それぞれ後述する機能を提供可能なように修正することによって実装可能である。

【００４５】
上記コンピュータ・プログラムは圧縮し、また複数に分割して複数の媒体に記録することもできる。ドライブ１１６は、必要に応じて、ＣＤ－ＲＯＭ、ＤＶＤ－ＲＯＭまたはＢＤからプログラムをディスク１１４にインストールするために使用されうる。
【００４６】
通信インタフェース１２６は、例えばイーサネット（登録商標）・プロトコルに従う。通信インタフェース１２６は、通信コントローラ１２４を介してバス１０６に接続され、コンピュータ・システム１００を通信回線１２８に物理的に接続する役割を担い、コンピュータ・システム１００のオペレーティング・システムの通信機能のＴＣＰ／ＩＰ通信プロトコルに対して、ネットワーク・インタフェース層を提供する。なお、通信回線は、有線ＬＡＮ環境に基づくもの、又は、無線ＬＡＮ環境、例えば、ＩＥＥＥ８０２．１１ａ／ｂ／ｇ／ｎなどのＷｉ－Ｆｉ規格に基づくものであってもよい。
【００４７】
以上から、本発明の実施態様において使用されるコンピュータ・システム１００は、本発明の実施の形態に係るアボート削減プログラムを格納し、それを実行することが出来る装置であってよく、ＰＣ、サーバ、ワークステーション等をアボート削減装置とすることが可能である。なお、上記説明した構成要素は例示であり、そのすべての構成要素が本発明の必須構成要素となるわけではない。
【００４８】
図２は、図１に示すコンピュータ・システム１００のソフトウェアの構成の一例を示す図である。CPU１０２は、ディスク１１４からJava（登録商標）仮想マシン（VM）等の仮想マシン、オペレーティング・システムをメイン・メモリ１０４に読み出し実行することにより、仮想マシン２０６、オペレーティング・システム２０２をメイン・メモリ１０４に展開する。オペレーティング・システム２０２は、ＣＰＵ１０２やメモリの管理など、コンピュータ・システム１００が有する基本的な機能を提供するソフトウェアである。
【００４９】
仮想マシン２０６は、バイトコードの低速実行（Interpret）、およびコンパイル済みコードの実行を行うエミュレータであり、コードの実行を管理し、仮想マシン上で実行されるアプリケーションに対して様々なサービスを提供する。仮想マシン２０６は、インタープリタ２１０と、コンパイル済みコード実行部２１２と、ディスパッチャ２１４と、複数のランタイムヘルパーを含むランタイムヘルパー群２１８と、アボート・ハンドラ２０４を含んで構成される。
【００５０】
ディスパッチャ２１４は、ランタイムヘルパー群２１８に含まれるJITコンパイラ２２０が生成したコンパイル済みコードを保存するメモリ領域であるコードキャッシュ２１６を参照して、次に実行するバイトコードアドレスから始まるコンパイル済みコードがコードキャッシュ２１６に保存されている否かを判定する。インタープリタ２１０は、コンパイル済みコードが存在しない場合に、処理対象のバイトコードを低速に実行する。コンパイル済みコード実行部２１２は、コンパイル済みコードが存在する場合、コードキャッシュ２１６からコンパイル済みコードを取得して実行する。
【００５１】
上述したように、本発明においてCPU１０２は、並列処理が可能なマルチプロセッサとされ、HTM１０２ａを含んで構成される。従って、インタープリタ２１０は、実行対象のコードにトランザクション開始命令とトランザクション終了命令とで囲まれたトランザクションブロックが存在する場合、該領域を、HTM１０２ａを利用してトランザクションとして実行する。またJITコンパイルされたコードにトランザクションブロックが含まれる場合、コンパイル済みコード実行部２１２は該領域を、HTM１０２ａを利用してトランザクションとして実行する。
【００５２】
インタープリタ２１０及びコンパイル済みコード実行部２１２はまた、トランザクショ

ンが失敗する場合、それぞれ所定の条件に従って非トランザクションパス（non-transactional path）を実行する。ここで、非トランザクションパスとは、トランザクションブロックの実行が何度もアボートする場合にプログラムを先に進めるため通常実行される、トランザクションブロックと意味的に同じ処理をいう。非トランザクションパスは、トランザクションブロック毎に一つ存在する。両者は意味的に同じであるが、非トランザクションパスの方が実行は遅い。なお、プログラム中にトランザクションブロックを書く場合、非トランザクションパスはプログラマが書く場合もあればコンパイラなどがトランザクションブロックのコードから自動的に生成する場合もある。仮想マシンなどシステムが自動的にトランザクションブロックを作る場合、非トランザクションパスもそのシステムが同時に生成するのが一般的である。本実施形態に係る仮想マシン２０６もまた、プログラム中にトランザクションブロックが存在する場合、及びトランザクションブロックを自動で作る場合、対応する非トランザクションパスを同時に生成するものとする。なお、以下では、インタープリタ２１０及びコンパイル済みコード実行部２１２をまとめて、実行部２０８（特許請求の範囲の請求項１９における実行部に相当）という。

【００５３】

ランタイムヘルパー２１８は複数のランタイムヘルパーを含む。ランタイムヘルパーは、実行時にインタープリタやJITコンパイルされたコードから呼び出される仮想マシンの機能である。ランタイムヘルパーの中には、トランザクション中に実行されるとトランザクションをアボートさせる可能性のあるものも存在し、本明細書ではそのようなランタイムヘルパーを、アボート性ランタイムヘルパー（abort-prone runtime helper）２１９と呼ぶ。アボート性ランタイムヘルパー２１９であるか否かはHTMの実装にも依存するが、一例として、JITコンパイラ２２０、クラスローダー２２２、非同期ガーベジコレクター（garbage collector：GC）２２４、TLHアロケータ２２６、シンボル・リゾルバー２２８を挙げることができる。本発明は、アボート性ランタイムヘルパー２１９によって引き起こされるアボートの回数を削減することを目的とする。そこでまずは上に挙げた個々のアボート性ランタイムヘルパーについて説明する。

【００５４】

JITコンパイラ２２０は、あるメソッドが所定回数実行された場合に呼び出されるものである。しかしその呼び出しがトランザクション中になされると、JITコンパイラ２２０が使用する作業領域のサイズがハードウェアによりサポートされるトランザクションサイズを超えるため、アボートが起きる。そこで考えられる解決方法は、一度だけ非トランザクションパスを実行し、その間にJITコンパイル対象のメソッドが実行されることを期待することである。もし、非トランザクションパスを実行中にJITコンパイル対象のメソッドが実行されれば、JITコンパイラ２２０はトランザクションブロックの外で呼び出され、成功裏に実行される。一方、非トランザクションパスを実行中にJITコンパイル対象のメソッドが実行されない場合は、トランザクションブロックを無効化し、プログラムが定常状態になるのを待って再度トランザクションブロックを有効化する。

【００５５】

クラスローダー２２２は、プログラムの実行中にオンデマンドでクラスをロードするものである。もしトランザクション中にクラスロードが起きると、I/Oがトランザクションをアボートする。そこで考えられる解決方法は、一度だけ非トランザクションパスを実行し、その非トランザクションパスが同じクラスを使用することを期待することである。もし、非トランザクションパスを実行中に同じクラスが使用されれば、クラスローダー２２２はトランザクションブロックの外で呼び出され、成功裏に実行される。一方、非トランザクションパスを実行中に同じクラスが使用されない場合は、トランザクションブロックを無効化し、プログラムが定常状態になるのを待って再度トランザクションブロックを有効化する。

【００５６】

非同期GC２２４は、stop-the-worldGCによって設定されたフラグを定期的にチェックする各アプリケーションがスリープすることによって呼び出される。しかしながらトランザ

クション中におけるスリープのためのシステムコール呼び出しは許されないため、トランザクション中に上記フラグの設定が検出されると、アボートが起きる。そこで考えられる解決方法は、非トランザクションパスを実行することである。非トランザクションパスを実行することで、次にトランザクションに入る前に、現在のスレッドが別の非同期GCチェックポイントに到達し、これによってトランザクションブロックの外で非同期GC２２４が呼び出されることが期待できる。

【００５７】

TLHアロケータ２２６は、しばしばプログラムによって呼び出され、ヒープからTLHを割り付け、これをゼロでクリアするものである。しかしTLH割り付けがトランザクション中になされると、ゼロクリーニングがトランザクションサイズの上限を超えてオーバーフローするため、アボートが起きる。そこで考えられる解決方法は、非トランザクションパスを実行することである。非トランザクションパスを実行することで、次にトランザクションに入る前に、現在のスレッドが新しいオブジェクトを割り付け、これによってトランザクションブロックの外でTLHアロケータ２２６が呼び出されることが期待できる。

【００５８】

シンボル・リゾルバー２２８又はコード修正は、実行時にメソッド内のシンボルをオンデマンドで解決し、結果としてシンボルを参照するコードを修正するものである。そのためシンボル解決がトランザクション中に実行されると、アボートが起きる。しかしこの場合、非トランザクションパスを実行しても、トランザクションパス上のコードが修正されることはないので、問題解決にはならない。従って考えられる解決方法は直ちにトランザクションブロックを無効化して、無駄にアボートが起きないようにすることである。

【００５９】

このように、アボート性ランタイムヘルパー２１９によって引き起こされるアボートの原因は様々であり、アボートの回数を削減する解決策もそれぞれ異なる。そこで本発明の実施形態に係るアボート性ランタイムヘルパー２１９はその種類に関わらず、トランザクションブロックの実行中において呼び出されると、そのアボート性ランタイムヘルパー２１９の種別を示すID情報をアボート・ハンドラ２０４に渡す。好ましくは、アボート性ランタイムヘルパー２１９は、トランザクションブロックの実行中において呼び出されると、本来の処理を行うことなくそのID情報をアボート原因を示す引数としてトランザクションアボート命令を発行する。

【００６０】

トランザクションアボート命令が実行されることにより呼び出されるアボート・ハンドラ２０４は、アボートの原因となったアボート性ランタイムヘルパー２１９のID情報を取得し、ID情報が特定の種別のアボート性ランタイムヘルパー２１９であることを示す場合に、トランザクションブロックを無効化する。好ましくは、アボート・ハンドラ２０４は、非トランザクションパスにおいても実行不可能な第１の種別のランタイムヘルパーに対し無条件でトランザクションブロックを無効化する。また、アボート・ハンドラ２０４は、非トランザクションパスにおいては実行可能な第２の種別のランタイムヘルパーに対し、一度だけ非トランザクションパスを実行し、その実行中に第２の種別のランタイムヘルパーが呼び出されなかった場合にトランザクションブロックを無効化する。ここで、第１の種別のランタイムヘルパーは、実行時コード修正を含み、第２の種別のランタイムヘルパーは、JITコンパイルとクラスロードとを含む。なお、アボート・ハンドラ２０４は、TLH割り付けと、非同期GCチェックとを、特定の種別以外のランタイムヘルパーとして判断し、これらに対しては何もしない。

【００６１】

アボート・ハンドラ２０４による処理の後は、実行部２０８が、アボートが起きたトランザクションブロックに対応する非トランザクションパスを実行する。また、アボート・ハンドラ２０４によって条件付きでトランザクションブロックが無効化された場合、実行部２０８は、プログラムが定常状態に達したことに応答してトランザクションブロックを再有効化する。これは、プログラムが定常状態おいては、アボートの原因となったクラス

がロードされ、また、アボートの原因となったメソッドがコンパイルされていることを期待するからである。このようにアボートの原因が解決されるまでの間のトランザクションブロックを無効化した状態を、本明細書においては一時的な非トランザクションモードと呼ぶ。なお、プログラムが定常状態に達したか否かの判定は、十分な数のクラスがロードされたか否か、又は十分な数のメソッドがコンパイルされたか否か、或いはその両方であるか否かによって判定してよく、例えば、以下のような方法が挙げられる。
・新しいクラスがロードされない時間、又は新しいメソッドがコンパイルされない時間、或いはその両方が10秒間以上続いた
・クラスが新たに1000個ロードされた、又はメソッドが新たに1000個コンパイルされた、或いはその両方である
・1秒間にロードされるクラス数、又は1秒間にコンパイルされるメソッド数、あるいはその両方を監視して、10個/秒未満になった
なお、10秒、1000個などの数は一例であり、予備実験で最適な値を予め求めておくのが好ましい。

【００６２】

上述したアボート削減の方法は、アボート削減のベースの手法として以下に記載する４つの観点から拡張することが可能である。

A.一時的な非トランザクションモードに入るタイミング

A1. 一時的な非トランザクションモードへ入るタイミングをトランザクションごとに決定する（ベースの手法）

A2. プログラム実行開始直後のウォーミングアップにおいて一時的な非トランザクションモードに積極的に入っておく（その後プログラムが安定状態となった後にA1.の手法を採用）

B.アボート原因の識別方法

B1. アボート性ランタイムヘルパー２１９のID情報をアボートの原因を示す情報とし、トランザクションアボート命令の引数とする（ベースの手法）

B2. 非トランザクション書き込み命令を用いて、アボート性ランタイムヘルパー２１９のID情報を現在実行中のスレッドのランタイムヘルパーの記録領域に記録する

B3. 非トランザクション書き込み命令を用いて、アボート性ランタイムヘルパー２１９のID情報のみならず、アボート性ランタイムヘルパー２１９の処理に必要な情報をも現在実行中のスレッドのランタイムヘルパーの記録領域に記録する

B4. トランザクションアボート命令の命令アドレスからコード修正に必要な情報への対応表を作成する

B5. ページ保護機能を利用する

なお、非トランザクション書き込み命令とは通常の書き込み命令を意味し、トランザクションがアボートしてもロールバックされることはない書き込み命令をいう。

C.アボート原因の解決方法

C1. 非トランザクションパスの実行、又はトランザクションブロックの無効化、或いはその両方（ベースの手法）

C2. アボート性ランタイムヘルパー２１９の処理を遅延させ、アボート・ハンドラ２０４の実行中に行う

D.一時的な非トランザクションモードから抜け出すタイミング

D1. プログラムが定常状態に達するのまで待つ（ベースの手法）

D2. アボート・ハンドラ２０４内での遅延処理が終わるまで待つ

D3. アボートにより実行されなかったクラスロードの全てのクラスがロードされ、又はコンパイルされなかった全てのメソッドがコンパイルされ、或いはその両方がなされるまで待つ

【００６３】

ベースの手法を第１の実施形態とし、上述した拡張方法を組み合わせてなし得る他の５つの実施形態を第２の実施形態～第６の実施の形態として以下に説明する。

【００６４】

第２の実施形態（A1+B2+C1+D1）

あるランタイムヘルパーがトランザクションを実行中に呼び出された場合にアボートを引き起こすか否かはHTMの実装に依存する。例えば、JITコンパイルは、非常に大きいトランザクションサイズをサポートする実装上ではトランザクションのアボートを引き起こすことはない。このような実装において、アボート性ランタイムヘルパー２１９がトランザクション中に呼び出される度トランザクションアボート命令を発行すると、パフォーマンスの低下を招いてしまう。

【００６５】

そこで第２の実施形態では、トランザクションアボート命令を発行する代わりに、アボート性ランタイムヘルパー２１９は、その本来の処理が実行されたことを、非トランザクション書き込み命令を用いて現在実行中のスレッドの記憶領域に記録する。アボート・ハンドラ２０４は、そのような記録が存在するか否かを判定し、存在する場合にその記録に基づきアボートの原因を決定する。ここでアボート性ランタイムヘルパー２１９が記録領域に記録すべき情報は、一例としてそれ自体のID情報であってよい。

【００６６】

第３の実施形態（A1+B3+C2+D2）

ベースの第１の実施形態では、アボート性ランタイムヘルパー２１９がコード修正の場合、直ちにトランザクションブロックを無効化する。アボート性ランタイムヘルパー２１９がJITコンパイラ２２０である場合も、１度の非トランザクションパスの実行においてJITコンパイラ２２０が呼び出されないことを条件にトランザクションブロックを無効化する。上述したように非トランザクションパスの実行はトランザクションブロックの実行よりも遅いので、トランザクションブロックを無効化するとパフォーマンスが低下する。

【００６７】

そこで第３の実施の形態では、トランザクション中に呼び出されるアボート性ランタイムヘルパー２１９がJITコンパイラ２２０やシンボル・リゾルバー２２８である場合、その実行を遅延させ、アボート・ハンドラ２０４内で実行させる。JITコンパイラ２２０やシンボル・リゾルバー２２８はプログラマから見える副作用を持たず（programexecution semantics）、その実行順序は厳しく定義されるものではないため、このように実行を遅延させても問題はない。その一方で、クラスローダー２２２は、プログラマから見える副作用を持つ（programmer-visiblesemantics）ためこのような遅延は許されない。

【００６８】

トランザクションアボート命令を発行する代わりに実行を遅延させるため、JITコンパイラ２２０やシンボル・リゾルバー２２８であるアボート性ランタイムヘルパー２１９は、そのID情報と共に本来の処理に必要な情報を、非トランザクション書き込み命令を用いて現在実行中のスレッドの記憶領域に記録する。アボート・ハンドラ２０４は、そのような記録が存在するか否かを判定し、存在する場合にその記録に基づきJITコンパイル又はコード修正を行う。ここでJITコンパイラ２２０が記録領域に記録すべき情報は、コンパイルすべきメソッドの識別情報である。また、シンボル・リゾルバー２２８が記録領域に記録すべき情報は、実行時コード修正されるべきアドレス及びデータである。アボート・ハンドラ２０４による上記処理が終了した後は、実行部２０８がトランザクションブロックを再実行する。

【００６９】

第４の実施形態（A1+B4+C2+D2）

第３の実施の形態では非トランザクション書き込み命令を利用した。しかしながら非トランザクション書き込み命令をサポートしないマシンも存在する。

【００７０】

そこで第３の実施の形態では、非トランザクション書き込み命令を用いる代わりに、JITコンパイラ２２０による本来の処理に以下に説明する処理を追加する。即ち、JITコンパイラ２２０は、コンパイル対象のコードに含まれるトランザクションブロック内の実行時

コード修正の呼び出しを、トランザクションアボート命令で置換し、該トランザクションアボート命令のアドレスから実行時コード修正に必要な情報への対応表を作成する。ここで実行時コード修正に必要な情報とは、実行時コード修正されるべきアドレス及びデータである。

【００７１】

置換されたトランザクションアボート命令の実行により呼び出されたアボート・ハンドラ２０４は、トランザクションアボート命令のアドレスから上記対応表を引いて必要な情報を取得する。そして、アボート・ハンドラ２０４は、取得した情報に基づいて実行時コード修正を行い、上記トランザクションアボート命令をｎｏｐ命令で上書きする。アボート・ハンドラ２０４による上記処理が終了した後は、実行部２０８がトランザクションブロックを再実行する。

【００７２】

第５の実施形態（A1+B5+C2+D2）

コード修正は、シンボル解決だけでなく、JITコンパイルや多相的インライン・メソッド・キャッシュ（polymorphic inline methodcaching）など、JITコンパイルされたコードにおいてよく利用される。しかしながらこれまで述べてきたように、コード修正はトランザクションを実行中に呼び出されるとアボートを引き起こす。第３の実施形態のようにアボート・ハンドラ２０４の処理が開始されるまでその実行を遅延させることもできるが、ソフトウェアエンジニアリングの観点からすれば、マネージドランタイムにおいて、全てのコード修正の位置で遅延処理のためのロジックを挿入することは現実的ではない。

【００７３】

そこで、第５の実施形態では、全てのコード修正の位置で遅延処理のためのロジックを挿入することなしにコード修正の処理を遅延させる。そのためにまず、アボート・ハンドラ２０４は、アボートを引き起こしたアボート性ランタイムヘルパー２１９が実行時コード修正であることを条件に、アボートが起きた命令アドレスを含むページを書き込み禁止にする。その後実行部２０８が、トランザクションブロックを再実行する。そして、次に呼び出されたアボート・ハンドラ２０４は、アボートの原因が書込み保護の例外であることを条件に、先ほどの書込み禁止を無効化し、書込み保護が起きた位置の書き込み命令を識別することにより実行時コード修正されるべきアドレスとデータを取得して、実行時コード修正を行う。その後実行部２０８が、トランザクションブロックを再再実行する。

【００７４】

第６の実施の形態（A1+B1+C1+D3）

ベースの第１の実施形態では、JITコンパイラ２２０又はクラスローダー２２２のトランザクション中の呼び出しに起因してトランザクションブロックを無効化した場合、プログラムが定常状態になるのを待ってトランザクションブロックを再有効化する。しかしながら、プログラムの定常状態において、原因となったメソッドがJITコンパイルされ、又は原因となったクラスがロードされているという保証はない。

【００７５】

そこで第６の実施形態では、原因となったメソッドがJITコンパイルされ、又は原因となったクラスがロードされたことを確認した後にトランザクションブロックを有効化するようにする。そのために、第６の実施形態では、トランザクション実行中に呼び出されたJITコンパイラ２２０及びクラスローダー２２２は、その処理内容を、非トランザクション書き込み命令を用いて現在実行中のスレッドの記憶領域に記録し、及び、ID情報を引数としてトランザクションアボート命令を発行する。ここで処理内容は、JITコンパイラ２２０の場合コンパイルすべきメソッドの識別情報であり、クラスローダー２２２の場合、ロードすべきクラスの識別子である。

【００７６】

トランザクションアボート命令により呼び出されたアボート・ハンドラ２０４は、JITコンパイラ２２０及びクラスローダー２２２に対してトランザクションブロックを無効化する場合に、現在実行中のスレッドの記憶領域に記録された情報を読み出してグローバル

表に登録する。その後実行部２０８は、グローバル表に登録された処理内容が実行されたことに応答して、トランザクションブロックを再有効化する。

【００７７】

次に図３～図１６を参照して、本発明のアボート削減処理の流れを説明する。まず、図３～図７を参照して、第１の実施形態によるアボート削減処理の流れを説明する。図３は、本発明の実施の形態に係るアボート削減処理全体の流れの一例を示すフローチャートである。図４は、アボート性ランタイムヘルパー２１９による処理の流れの一例を示すフローチャートである。図５（ａ）及び（ｂ）は、アボート・ハンドラ２０４による処理の一部の流れの一例を示すフローチャートである。図６は、非トランザクション・パスの実行処理の流れの一例を示すフローチャートである。図７（ａ）及び（ｂ）は、トランザクションブロックの有効化処理の流れの一例を示すフローチャートである。

【００７８】

図３に示すフローチャートは、トランザクションブロックの実行により開始し、実行部２０８は、実行しようとする現在のトランザクションブロックが無効化されているか否かを判定する（ステップ３００）。現在のトランザクションブロックが無効化されていない場合（ステップ３００：ＮＯ）、続いて実行部２０８は、現在のトランザクションブロックのリトライ回数をカウントするためのリトライ・カウンタを正の整数で初期化する（ステップ３０２）。実行部２０８はまた、現在のトランザクションブロックの実行回数をカウントするための実行カウンタを１増やす（ステップ３０４）。そして実行部２０８は、トランザクションを開始し、実行する（ステップ３０６、ステップ３０８）。なお、リトライ・カウンタはスレッドごと、実行カウンタと後述するアボート・カウンタは、トランザクションブロックごと保持するものとする。なお、実行カウンタとアボート・カウンタの初期化は、対応するトランザクションブロックを含むコードがロードされた際に初期化される。

【００７９】

続いて実行部２０８は、トランザクション内でアボート性ランタイムヘルパー２１９が呼び出されたか否かを判定する（ステップ３１０）。アボート性ランタイムヘルパー２１９が呼び出された場合（ステップ３１０：ＹＥＳ）、処理はステップ３１２へ進み、呼び出されたアボート性ランタイムヘルパー２１９により処理がなされる。アボート性ランタイムヘルパー２１９による処理の詳細は、図４を参照して後述する。アボート性ランタイムヘルパー２１９による処理の後、又は、アボート性ランタイムヘルパー２１９が呼び出されなかった場合（ステップ３１０：ＮＯ）、処理はステップ３１４へ進み、実行部２０８は、トランザクション内でアボートが起きたか否かを判定する。

【００８０】

トランザクション内でアボートが起きた場合（ステップ３１４：ＹＥＳ）、処理はステップ３１６へ進み、アボート・ハンドラ２０４による処理がなされる。アボート・ハンドラ２０４による処理の詳細は、図５を参照して後述する。一方トランザクション内でアボートが起きなかった場合（ステップ３１４：ＮＯ）、処理はステップ３２０へ進み、実行部２０８はトランザクションを終了して（ステップ３２０）、処理を終了する。

【００８１】

ステップ３１６におけるアボート・ハンドラ２０４による処理は２つの結果を有し、結果が２を示す場合、処理はステップ３０６へ戻って、現在のトランザクションブロックの実行を再度試みる。一方、アボート・ハンドラ２０４による処理の結果が１を示す場合、又はステップ３００において現在のトランザクションブロックが無効化されていると判定した場合（ステップ３００：ＹＥＳ）、処理はステップ３１８へ進み、実行部は非トランザクションパスを実行する。非トランザクションパス実行処理の詳細は、図６を参照して後述する。非トランザクションパス実行処理の後、処理は終了する。

【００８２】

図４に示すフローチャートは、図３に示すフローチャートのステップ３１２におけるアボート性ランタイムヘルパー２１９による処理の詳細を示す。処理はステップ４００で開

始し、アボート性ランタイムヘルパー２１９は、そのアボート性ランタイムヘルパー２１９の種別を示すＩＤをアボートの原因を示す引数として、トランザクションアボート命令を発行する。その後アボート性ランタイムヘルパー２１９は処理を終了する。なお、図４に示すフローチャートの処理は、アボート性ランタイムヘルパー２１９がトランザクションを実行中に呼び出された場合に実行されるものであり、トランザクションが実行中でない場合には、アボート性ランタイムヘルパー２１９はそれぞれ本来の処理を行うことに留意されたい。

【００８３】

図５（ａ）に示すフローチャートは、図３に示すフローチャートのステップ３１６におけるアボート・ハンドラ２０４による処理の前半を示す。処理はステップ５００で開始し、アボート・ハンドラ２０４は、アボートの原因がTLH割り付け又は非同期GCチェックであるか否かを判定する。アボートの原因がTLH割り付け又は非同期GCチェックのいずれかである場合（ステップ５００：ＹＥＳ）、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。一方、アボートの原因がTLH割り付け又は非同期GCチェックのいずれでもない場合（ステップ５００：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がシンボル解決であるか否かを判定する（ステップ５０２）。アボートの原因がシンボル解決である場合（ステップ５０２：ＹＥＳ）、アボート・ハンドラ２０４は直ちに現在のトランザクションブロックを無効化し（ステップ５１０）、１を示す結果を残して処理を終了する。

【００８４】

アボートの原因がシンボル解決でない場合（ステップ５０２：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がJITコンパイル又はクラスロードであるか否かを判定する（ステップ５０４）。アボートの原因がJITコンパイル又はクラスロードのいずれでもない場合（ステップ５０４：ＮＯ）、処理は図５（ｂ）に示すフローチャートのステップ５２０に続く。一方、アボートの原因がJITコンパイル又はクラスロードのいずれかである場合（ステップ５０４：ＹＥＳ）、続いてアボート・ハンドラ２０４は、現在のトランザクションブロックについてのnon-tx-onceフラグが設定されているか否かを判定する（ステップ５０６）。

【００８５】

ここでnon-tx-onceフラグとは、トランザクションブロックごとに割り当てるフラグであり、後述するスレッドごとに割り当てるnon-tx-executionフラグと共に用いるフラグである。本実施例ではこれらフラグを用いた制御により、一度だけ非トランザクションパスを実行し、その間にアボートの原因となったJITコンパイル又はクラスロードが実行されないことを条件として、現在のトランザクションブロックを無効化する処理を実現する。なお、non-tx-onceフラグの設定は、図６を参照して後述する非トランザクションパスの実行においてなされ、その設定は、非トランザクションパスの実行中にJITコンパイル又はクラスロードが起きなかったことを示す。

【００８６】

図５（ａ）のフローチャートのステップ５０６に戻って、non-tx-onceフラグが設定されていない場合（ステップ５０６：ＮＯ）、続いてアボート・ハンドラ２０４は現在実行されているスレッドのnon-tx-executionフラグを設定する（ステップ５０８）。このように、non-tx-executionフラグは、アボートの原因がJITコンパイル又はクラスロードであったこと示す。一方、non-tx-onceフラグが設定されていた場合（ステップ５０６：ＹＥＳ）、アボート・ハンドラ２０４は現在のトランザクションブロックを無効化する（ステップ５１０）。これは、上述したように、non-tx-onceフラグの設定は、一度だけ実行した非トランザクションパスの実行中にJITコンパイル又はクラスロードが起きなかったことを示すためである。ステップ５０８又はステップ５１０の後、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。

【００８７】

図５（ｂ）に示すフローチャートは、図５（ａ）に示すフローチャートのステップ５０

４における判定がＮＯであった場合に実行される後の後半の処理の流れを示す。処理はステップ５２０で開始し、アボート・ハンドラ２０４はリトライ・カウンタの値が０より大きいか否かを判定する。リトライ・カウンタの値が０より大きい場合（ステップ５２０：ＹＥＳ）、アボート・ハンドラ２０４はリトライ・カウンタの値を１減らして（ステップ５２２）、２を示す結果を残して処理を終了する。

【００８８】

一方、リトライ・カウンタの値が０以下の場合（ステップ５２２：ＮＯ）、アボート・ハンドラ２０４は現在のトランザクションブロックのアボート・カウンタを１増やす（ステップ５２４）。続いてアボート・ハンドラ２０４は、アボート・カウンタの値を実行カウンタの値で割ることにより求められるアボート率が所定の閾値より小さいか否かを判定する（ステップ５２６）。アボート率が所定の閾値より小さい場合（ステップ５２６：ＹＥＳ）、アボート・ハンドラ２０４は、１を示す結果を残して処理を終了する。一方、アボート率が所定の閾値以上である場合（ステップ５２６：ＮＯ）、アボート・ハンドラ２０４は、現在のトランザクションブロックを無効化し（ステップ５２８）、１を示す結果を残して処理を終了する。

【００８９】

図６に示すフローチャートは、図３に示すフローチャートのステップ３１８の非トランザクションパス実行の処理の詳細を示す。処理はステップ６００で開始し、実行部２０８は、非トランザクションパスの実行を開始する。続いて実行部２０８は、非トランザクションパスの実行中にJITコンパイル又はクラスロードがトリガーされた場合、現在実行中のスレッドのnon-tx-executionフラグが設定されていることを条件として、非トランザクションパスの実行と同期してJITコンパイル又はクラスロードを実行し、その後現在実行中のスレッドのnon-tx-executionフラグをリセットする（ステップ６０２）。続いて実行部２０８は、非トランザクションパスの実行を終了する（ステップ６０４）。

【００９０】

続いて実行部２０８は、現在実行中のスレッドのnon-tx-executionフラグが設定されているか否かを判定する（ステップ６０６）。現在実行中のスレッドのnon-tx-executionフラグが設定されていない場合（ステップ６０６：ＮＯ）、実行部２０８は処理を終了する。一方、現在実行中のスレッドのnon-tx-executionフラグが設定されている場合（ステップ６０６：ＹＥＳ）、実行部２０８は、現在のトランザクションブロックに割り当てられているnon-tx-onceフラグを設定し（ステップ６０８）、その後、現在実行中のスレッドのnon-tx-executionフラグをリセットする（ステップ６１０）。その後実行部２０８は処理を終了する。

【００９１】

図７（ａ）に示すフローチャートは、実行部２０８により定期的に実行される処理である。処理はステップ７００で開始し、実行部２０８は、十分な数のクラスがロードされ、かつ、十分な数のメソッドがコンパイルされるまで待機する。続いて実行部２０８は、JITコンパイル又はクラスロードが原因で無効化された全てのトランザクションブロックを再度有効化し、対応する実行カウンタ、アボート・カウンタ、及びnon-tx-onceフラグを全てリセットする（ステップ７０２）。そして実行部２０８は処理を終了する。

【００９２】

図７（ｂ）は、第１の実施形態においてA1の構成に加えてA2の構成を採用した場合に、実行部２０８によりプログラム実行開始時に新たに実行される処理の流れを示すフローチャートである。処理はステップ７１０で開始し、実行部２０８は、実行しようとするプログラムのコードに含まれる全てのトランザクションブロックを無効化する。続いて実行部２０８は、十分な数のクラスがロードされ、かつ、十分な数のメソッドがコンパイルされるまで待機する（ステップ７１２）。続いて実行部２０８は、実行しようとするプログラムのコード内の全てのトランザクションブロックを再度有効化し、対応する実行カウンタ、アボート・カウンタ、及びnon-tx-onceフラグを全てリセットする（ステップ７１４）。そして実行部２０８は処理を終了する。

【００９３】
次に図３、図５～図８を参照して、第２の実施形態によるアボート削減処理の流れを説明する。但し、図８に示すフローチャートを除いた残りの図のフローチャートについては、第１の実施形態に関連して既に説明済みであるためここでは説明を省略する。図８は、図３に示すフローチャートのステップ３１２におけるアボート性ランタイムヘルパー２１９による処理の流れの他の例を示すフローチャートである。
【００９４】
図８に示すフローチャートは、ステップ８００で開始し、アボート性ランタイムヘルパー２１９は、非トランザクション書き込み命令を用いて、そのランタイムヘルパーの種別を示すＩＤを現在実行中のスレッドの記憶領域に記録する。続いてアボート性ランタイムヘルパー２１９は、その本来の処理を実行する（ステップ８０２）。続いてアボート性ランタイムヘルパー２１９は、現在実行中のトランザクションがアボートしたか否かを判定する（ステップ８０４）。
【００９５】
現在実行中のトランザクションがアボートした場合（ステップ８０４：ＹＥＳ）、アボート性ランタイムヘルパー２１９は何もせずに処理を終了する。一方、現在実行中のトランザクションがアボートしていない場合（ステップ８０４：ＮＯ）、アボート性ランタイムヘルパー２１９は現在実行中のスレッドの記憶領域に記録した情報をクリアして（ステップ８０６）、処理を終了する。なお、第２の実施形態においてアボート・ハンドラ２０４は、図５のフローチャートに示す処理の最初に現在実行中のスレッドの記憶領域に記録された情報を読み取るものとする。
【００９６】
次に図３、図５（ｂ）、図６、図７、図９、及び図１０を参照して、第３の実施形態によるアボート削減処理の流れを説明する。但し、図９及び図１０に示すフローチャートを除いた残りの図のフローチャートについては、第１の実施形態に関連して既に説明済みであるためここでは説明を省略する。但し、第３の実施形態においては、JITコンパイルが原因でトランザクションブロックが無効化されることはないため、図７（ａ）、（ｂ）にそれぞれ示すフローチャートのステップ７００、７０２、７１２及び７１４におけるJITコンパイルについての記載は削除する必要があることに留意されたい。図９は、図３に示すフローチャートのステップ３１２におけるアボート性ランタイムヘルパー２１９による処理の流れの他の例を示すフローチャートである。図１０は、図３に示すフローチャートのステップ３１６におけるアボート・ハンドラ２０４による前半の処理の流れの他の例を示すフローチャートである。
【００９７】
図９に示すフローチャートは、ステップ９００で開始し、アボート性ランタイムヘルパー２１９は、その処理内容がシンボル解決であるか否かを判定する。その処理内容がシンボル解決である場合（ステップ９００：ＹＥＳ）、アボート性ランタイムヘルパー２１９は非トランザクション書き込み命令を用いて、現在実行中のスレッドの記憶領域に、修正すべきコードのアドレスとデータとを書き込む（ステップ９０２）。
【００９８】
一方、その処理内容がシンボル解決でない場合（ステップ９００：ＮＯ）、続いてアボート性ランタイムヘルパー２１９は、その処理内容がJITコンパイルであるか否かを判定する（ステップ９０４）。処理内容がJITコンパイルである場合（ステップ９０４：ＹＥＳ）、アボート性ランタイムヘルパー２１９は非トランザクション書き込み命令を用いて、現在実行中のスレッドの記憶領域に、JITコンパイルすべきメソッドのＩＤを書き込む（ステップ９０２）。ステップ９０４において処理内容がJITコンパイルでない場合（ステップ９０４：ＮＯ）、ステップ９０２、又はステップ９０６の後処理はステップ９０８へ進み、アボート性ランタイムヘルパー２１９は、そのアボート性ランタイムヘルパー２１９の種別を示すＩＤをアボートの原因を示す引数として、トランザクションアボート命令を発行する。その後アボート性ランタイムヘルパー２１９は処理を終了する。

【００９９】
図１０に示すフローチャートは、ステップ１０００で開始し、アボート・ハンドラ２０４は、アボートの原因がシンボル解決であるか否かを判定する。アボートの原因がシンボル解決である場合（ステップ１０００：ＹＥＳ）、アボート・ハンドラ２０４は、現在実行中のスレッドの記憶領域に記録されたデータで、同じく記録されたコードのアドレスを修正し（ステップ１００２）、２を示す結果を残して処理を終了する。
【０１００】
一方、アボートの原因がシンボル解決でない場合（ステップ１０００：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がJITコンパイルであるか否かを判定する（ステップ１００４）。アボートの原因がJITコンパイルである場合（ステップ１００４：ＹＥＳ）、アボート・ハンドラ２０４は、現在実行中のスレッドの記憶領域に記録されたＩＤで識別されるメソッドをJITコンパイルし（ステップ１００）、２を示す結果を残して処理を終了する。
【０１０１】
一方、アボートの原因がJITコンパイルでない場合（ステップ１００４：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がTLH割り付け又は非同期GCチェックであるか否かを判定する（ステップ１００８）。アボートの原因がTLH割り付け又は非同期GCチェックのいずれかである場合（ステップ１００８：ＹＥＳ）、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。
【０１０２】
一方、アボートの原因がTLH割り付け又は非同期GCチェックのいずれでもない場合（ステップ１００８：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がクラスロードであるか否かを判定する（ステップ１０１０）。アボートの原因がクラスロードでない場合（ステップ１０１０：ＮＯ）、処理は図５（ｂ）に示すフローチャートのステップ５２０に続く。一方、アボートの原因がクラスロードである場合（ステップ１０１０：ＹＥＳ）、続いてアボート・ハンドラ２０４は、現在のトランザクションブロックについてのnon-tx-onceフラグが設定されているか否かを判定する（ステップ１０１２）。
【０１０３】
non-tx-onceフラグが設定されていない場合（ステップ１０１２：ＮＯ）、続いてアボート・ハンドラ２０４は現在実行されているスレッドのnon-tx-executionフラグを設定する（ステップ１０１４）。一方、non-tx-onceフラグが設定されていた場合（ステップ１０１２：ＹＥＳ）、アボート・ハンドラ２０４は現在のトランザクションブロックを無効化する（ステップ１０１６）。ステップ１０１４又はステップ１０１６の後、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。
【０１０４】
次に図３、図４、図５（ｂ）、図６、図７、図１１、図１２を参照して、第４の実施形態によるアボート削減処理の流れを説明する。但し、図１１及び１２に示すフローチャートを除いた残りの図のフローチャートについては、第１の実施形態に関連して既に説明済みであるためここでは説明を省略する。図１１は、JITコンパイラ２２０による処理の流れの一例を示すフローチャートである。図１２は、図３に示すフローチャートのステップ３１６におけるアボート・ハンドラ２０４による前半の処理の流れの他の例を示すフローチャートである。
【０１０５】
図１１に示すフローチャートは、あるメソッドが所定回数実行されJITコンパイラ２２０が呼び出されることにより開始し、JITコンパイラ２２０はコンパイル対象のコード内にトランザクションブロックがあるか否かを判定する（ステップ１１００）。コンパイル対象のコード内にトランザクションブロックがある場合（ステップ１１００：ＹＥＳ）、続いてJITコンパイラ２２０は、見つかったトランザクションブロック内に、コード修正の呼び出しがあるか否かを判定する（ステップ１１０２）。
【０１０６】

見つかったトランザクションブロック内にコード修正の呼び出しがある場合（ステップ１１０２：ＹＥＳ）、JITコンパイラ２２０は、見つかったコード修正の呼び出しをトランザクションアボート命令で置換し、かつ、アボート命令の命令アドレスから置換したコード修正の実行に必要な情報への対応表を作成する（ステップ１１０４）。ここでコード修正の実行に必要な情報は、実行時コード修正されるべきアドレス及びデータである。

【０１０７】

ステップ１１００においてコンパイル対象のコード内にトランザクションブロックがない場合（ステップ１１００：ＮＯ）、ステップ１１０２において見つかったトランザクションブロック内にコード修正の呼び出しがない場合（ステップ１１０２：ＮＯ）、又はステップ１１０４から、処理はステップ１１０６へ進み、JITコンパイラ２２０は通常のJITコンパイル処理を行い、その後処理を終了する。

【０１０８】

図１２に示すフローチャートは、ステップ１２００で開始し、アボート・ハンドラ２０４は、アボートの原因がシンボル解決であるか否かを判定する。該判定は、アボートが起きた命令アドレスで、JITコンパイラ２２０により作成された対応表を検索し、ヒットするか否かを判定することにより行うことができる。ヒットした場合アボートの原因がシンボル解決であると判定できる。アボートの原因がシンボル解決である場合（ステップ１２００：ＹＥＳ）、アボート・ハンドラ２０４は、アボートが起きた命令アドレスからJITコンパイラ２２０により作成された対応表を引いて必要な情報を取得し、取得した情報を用いてコード修正を実行し、更に何もしないという意味であるnop命令にアボート命令を置換する（ステップ１２０２）。その後アボート・ハンドラ２０４は、２を示す結果を残して処理を終了する。

【０１０９】

一方、アボートの原因がシンボル解決でない場合（ステップ１２００：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がTLH割り付け又は非同期GCチェックであるか否かを判定する（ステップ１２０４）。アボートの原因がTLH割り付け又は非同期GCチェックのいずれかである場合（ステップ１２０４：ＹＥＳ）、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。

【０１１０】

一方、アボートの原因がTLH割り付け又は非同期GCチェックのいずれでもない場合（ステップ１２０４：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がJITコンパイル又はクラスロードであるか否かを判定する（ステップ１２０６）。アボートの原因がJITコンパイル又はクラスロードのいずれでもない場合（ステップ１２０６：ＮＯ）、処理は図５（ｂ）に示すフローチャートのステップ５２０に続く。一方、アボートの原因がJITコンパイル又はクラスロードのいずれかである場合（ステップ１２０６：ＹＥＳ）、続いてアボート・ハンドラ２０４は、現在のトランザクションブロックについてのnon-tx-onceフラグが設定されているか否かを判定する（ステップ１２０８）。

【０１１１】

non-tx-onceフラグが設定されていない場合（ステップ１２０８：ＮＯ）、続いてアボート・ハンドラ２０４は現在実行されているスレッドのnon-tx-executionフラグを設定する（ステップ１２１０）。一方、non-tx-onceフラグが設定されていた場合（ステップ１２０８：ＹＥＳ）、アボート・ハンドラ２０４は現在のトランザクションブロックを無効化する（ステップ１２１２）。ステップ１２１０又はステップ１２１２の後、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。

【０１１２】

次に図３、図４、図５（ｂ）、図６、図７、図１３を参照して、第５の実施形態によるアボート削減処理の流れを説明する。但し、図１３に示すフローチャートを除いた残りの図のフローチャートについては、第１の実施形態に関連して既に説明済みであるためここでは説明を省略する。図１３は、図３に示すフローチャートのステップ３１６におけるアボート・ハンドラ２０４による前半の処理の流れの他の例を示すフローチャートである。

【０１１３】
図１３に示すフローチャートは、ステップ１３００で開始し、アボート・ハンドラ２０４は、アボートの原因がコード修正であるか否かを判定する。アボートの原因がコード修正である場合（ステップ１３００：ＹＥＳ）、アボート・ハンドラ２０４は、アボートが起きた命令アドレスを含むページを書き込み保護する（ステップ１３０２）。その後アボート・ハンドラ２０４は、２を示す結果を残して処理を終了する。
【０１１４】
一方、アボートの原因がコード修正でない場合（ステップ１３００：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因が書込み保護の例外であるか否かを判定する（ステップ１３０４）。アボートの原因が書込み保護の例外である場合（ステップ１３０４：ＹＥＳ）、アボート・ハンドラ２０４は、書込み保護を無効化し、書込み保護が起きた場所の書き込み命令を識別することにより実行時コード修正されるべきアドレスとデータを取得し、実行時コード修正を行う（ステップ１３０６）。その後アボート・ハンドラ２０４は、２を示す結果を残して処理を終了する。
【０１１５】
一方、アボートの原因が書込み保護の例外でない場合（ステップ１３０４：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がTLH割り付け又は非同期GCチェックであるか否かを判定する（ステップ１３０８）。アボートの原因がTLH割り付け又は非同期GCチェックのいずれかである場合（ステップ１３０８：ＹＥＳ）、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。
【０１１６】
一方、アボートの原因がTLH割り付け又は非同期GCチェックのいずれでもない場合（ステップ１３０８：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がJITコンパイル又はクラスロードであるか否かを判定する（ステップ１３１０）。アボートの原因がJITコンパイル又はクラスロードのいずれでもない場合（ステップ１１３１０：ＮＯ）、処理は図５（ｂ）に示すフローチャートのステップ５２０に続く。一方、アボートの原因がJITコンパイル又はクラスロードのいずれかである場合（ステップ１３１０：ＹＥＳ）、続いてアボート・ハンドラ２０４は、現在のトランザクションブロックについてのnon-tx-onceフラグが設定されているか否かを判定する（ステップ１３１２）。
【０１１７】
non-tx-onceフラグが設定されていない場合（ステップ１３１２：ＮＯ）、続いてアボート・ハンドラ２０４は現在実行されているスレッドのnon-tx-executionフラグを設定する。一方、non-tx-onceフラグが設定されていた場合（ステップ１３１２：ＹＥＳ）、アボート・ハンドラ２０４は現在のトランザクションブロックを無効化する（ステップ１３１６）。ステップ１３１４又はステップ１３１６の後、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。
【０１１８】
次に図３、図５（ｂ）、図６、図１４、図１５、図１６を参照して、第６の実施形態によるアボート削減処理の流れを説明する。但し、図１４、図１５及び１６に示すフローチャートを除いた残りの図のフローチャートについては、第１の実施形態に関連して既に説明済みであるためここでは説明を省略する。図１４は、図３に示すフローチャートのステップ３１２におけるアボート性ランタイムヘルパー２１９による処理の流れの他の例を示すフローチャートである。図１５は、図３に示すフローチャートのステップ３１６におけるアボート・ハンドラ２０４による前半の処理の流れの他の例を示すフローチャートである。図１６は、トランザクションブロックの有効化処理の流れの他の例を示すフローチャートである。
【０１１９】
図１４に示すフローチャートは、ステップ１４００で開始し、アボート性ランタイムヘルパー２１９は、その処理内容がJITコンパイル又はクラスロードであるか否かを判定する。その処理内容がJITコンパイル又はクラスロードのいずれかである場合（ステップ１

４００：ＹＥＳ）、アボート性ランタイムヘルパー２１９は非トランザクション書き込み命令を用いて、現在実行中のスレッドの記憶領域に、JITコンパイルすべきメソッドのＩＤ又はロードすべきクラスのＩＤを書き込む（ステップ１４０２）。

【０１２０】

一方、その処理内容がJITコンパイル又はクラスロードのいずれでもない場合（ステップ１４００：ＮＯ）、続いてアボート性ランタイムヘルパー２１９は、そのアボート性ランタイムヘルパー２１９の種別を示すＩＤをアボートの原因を示す引数として、トランザクションアボート命令を発行する（ステップ１４０４）。その後アボート性ランタイムヘルパー２１９は処理を終了する。

【０１２１】

図１５に示すフローチャートは、ステップ１５００で開始し、アボート・ハンドラ２０４は、アボートの原因がTLH割り付け又は非同期GCチェックであるか否かを判定する。アボートの原因がTLH割り付け又は非同期GCチェックのいずれかである場合（ステップ１５００：ＹＥＳ）、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。一方、アボートの原因がTLH割り付け又は非同期GCチェックのいずれでもない場合（ステップ１５００：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がシンボル解決であるか否かを判定する（ステップ１５０２）。アボートの原因がシンボル解決である場合（ステップ１５０２：ＮＯ）、アボート・ハンドラ２０４は直ちに現在のトランザクションブロックを無効化し（ステップ１５０４）、１を示す結果を残して処理を終了する。

【０１２２】

アボートの原因がシンボル解決でない場合（ステップ１５０２：ＮＯ）、続いてアボート・ハンドラ２０４は、アボートの原因がJITコンパイル又はクラスロードであるか否かを判定する（ステップ１５０６）。アボートの原因がJITコンパイル又はクラスロードのいずれでもない場合（ステップ１５０６：ＮＯ）、処理は図５（ｂ）に示すフローチャートのステップ５２０に続く。一方、アボートの原因がJITコンパイル又はクラスロードのいずれかである場合（ステップ１５０６：ＹＥＳ）、続いてアボート・ハンドラ２０４は、現在のトランザクションブロックについてのnon-tx-onceフラグが設定されているか否かを判定する（ステップ１５０８）。

【０１２３】

non-tx-onceフラグが設定されていない場合（ステップ１５０８：ＮＯ）、続いてアボート・ハンドラ２０４は現在実行されているスレッドのnon-tx-executionフラグを設定する（ステップ１５１０）。一方、non-tx-onceフラグが設定されていた場合（ステップ１５０８：ＹＥＳ）、アボート・ハンドラ２０４は、現在実行中のスレッドの記憶領域に記録されたメソッドＩＤ又はクラスＩＤを、グローバル・テーブルに書き込む（ステップ１５１２）。続いてアボート・ハンドラ２０４は現在のトランザクションブロックを無効化する（ステップ１５１４）。ステップ１５１０又はステップ１５１４の後、アボート・ハンドラ２０４は１を示す結果を残して処理を終了する。

【０１２４】

図１６に示すフローチャートは、ステップ１６００で開始し、実行部２０８は、グローバル・テーブル内の全てのメソッドと全てのクラスとが、それぞれJITコンパイル又はロードされるまで待機する。続いて実行部２０８は、JITコンパイル又はクラスロードが原因で無効化された全てのトランザクションブロックを再度有効化し、対応する実行カウンタ、アボート・カウンタ、及びnon-tx-onceフラグを全てリセットする（ステップ７０２）。そして実行部２０８は処理を終了する。

【０１２５】

次に図１７を参照して実験結果について説明する。図１７（ａ）は、カウンタを加算するマイクロベンチマークを用いて本発明のアボート削減方法を適用した４コアのzEC12マシンの性能を評価した結果を示す。また、図１７（ｂ）は、HTMを用いたJava(登録商標)のConcurrentHashMapを用いて本発明のアボート削減方法を適用した１６コアのzEC12マシ

ンの性能を評価した結果を示す。いずれのグラフも横軸はスレッド数を示し、縦軸は、スループットを示す。なお、スループットは、１スレッド時の従来技術を適用した場合のスループット値を１としている。

【０１２６】

適用した本発明のアボート削減方法は、上述した第３の実施形態によるアボート削減方法である。また、比較した従来技術は、非特許文献１に記載される手法であり、トランザクションがアボートした場合に何度か再実行し、それでもアボートが起こる場合には非トランザクションパスを実行してから再度トランザクションを実行するという手法である。図１７（ａ）に示す実験結果では、本発明を利用しない場合はパフォーマンスが８９％低下している。図１７（ｂ）はに示す実験結果では、本発明を利用しない場合はパフォーマンスが１３％低下している。これは本発明を利用しない場合、トランザクション中でJITコンパイルと実行時コード修正が起きてトランザクションがアボートし続けるからである。

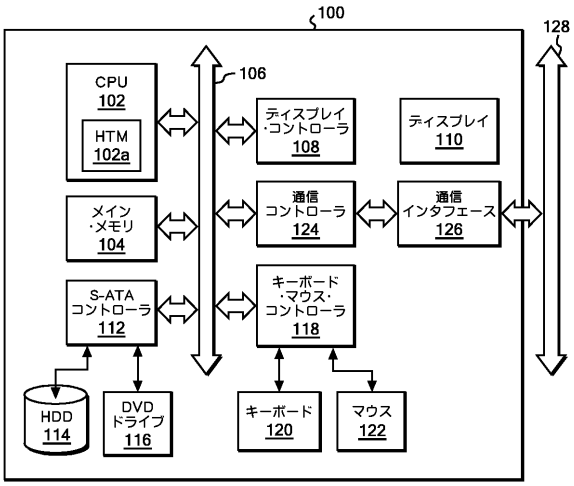
【０１２７】

以上、実施形態を用いて本発明の説明をしたが、本発明の技術範囲は上記実施形態に記載の範囲には限定されない。上記の実施形態に、種々の変更または改良を加えることが可能であることが当業者に明らかである。例えば、第３の実施形態では、トランザクション中に呼び出されるアボート性ランタイムヘルパー２１９がJITコンパイラ２２０やシンボル・リゾルバー２２８である場合、その実行を遅延させ、アボート・ハンドラ２０４内で実行させた。しかしながら同様の方法で実行に必要な情報を記録しておき、その後これをアボート・ハンドラ内ではなくアボート・ハンドラによる処理の後実行部２０９により実行させてもよい。従って、そのような変更または改良を加えた形態も当然に本発明の技術的範囲に含まれる。

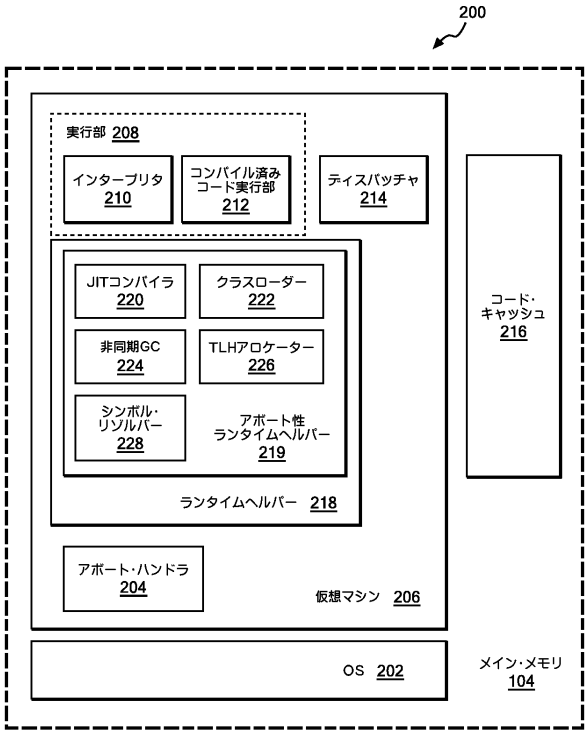
【０１２８】

なお、特許請求の範囲、明細書、及び図面中において示した装置、システム、プログラム、及び方法における動作、手順、ステップ、及び段階等の各処理の実行順序は、特段「より前に」、「先立って」等と明示しておらず、また、前の処理の出力を後の処理で用いるのでない限り任意の順序で実現しうることに留意すべきである。また、前の処理の出力を後の処理で用いる場合でも、前の処理と後の処理の間に他の処理が入ることは可能である場合があること、又は間に他の処理が入るように記載されていても前の処理を後の処理の直前に行うよう変更することも可能である場合があることも留意されたい。特許請求の範囲、明細書、及び図面中の動作フローに関して、便宜上「まず、」、「次に、」、「続いて、」等を用いて説明したとしても、この順で実施することが必須であることを必ずしも意味するとは限らない。

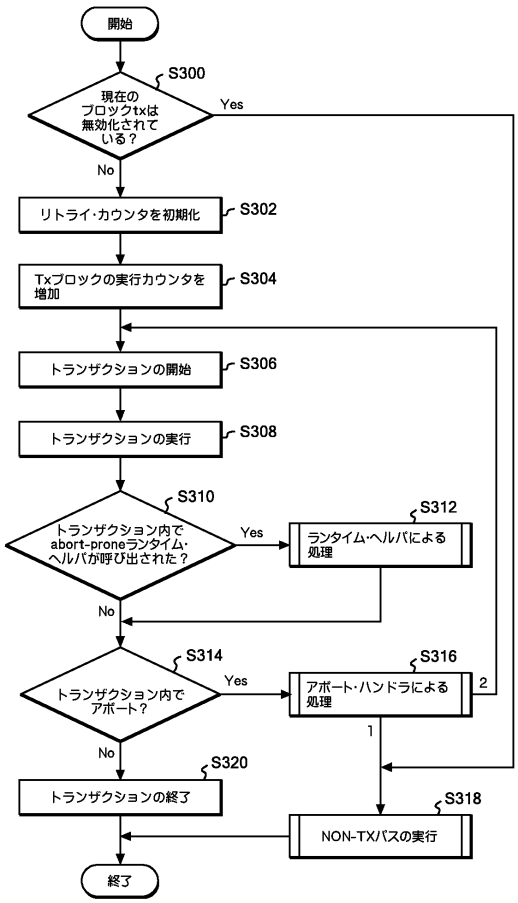
【図１】


100
128
106
CPU
102
HTM
102a
ディスプレイ
・コントローラ
108
ディスプレイ
110
メイン
・メモリ
104
通信
コントローラ
124
通信
インタフェース
126
S-ATA
コントローラ
112
キーボード
・マウス・
コントローラ
118
HDD
114
DVD
ドライブ
116
キーボード
120
マウス
122

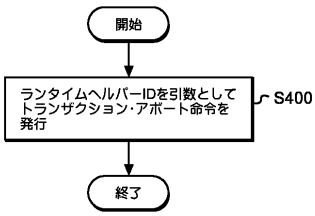
【図２】


200
実行部 208
インタープリタ
210
コンパイル済み
コード実行部
212
ディスパッチャ
214
JITコンパイラ
220
クラスローダー
222
非同期GC
224
TLHアロケーター
226
シンボル・
リゾルバー
228
アボート性
ランタイムヘルパー
219
ランタイムヘルパー 218
コード・
キャッシュ
216
アボート・ハンドラ
204
仮想マシン 206
OS 202
メイン・メモリ
104

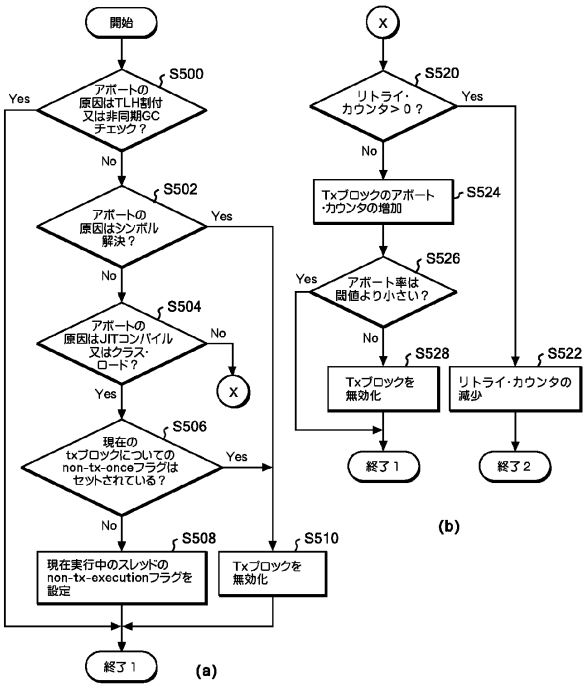
【図３】


開始
S300
現在の
ブロックtxは
無効化されて
いる？
Yes
No
リトライ・カウンタを初期化
S302
Txブロックの実行カウンタを
増加
S304
トランザクションの開始
S306
トランザクションの実行
S308
S310
トランザクション内で
abort-proneランタイム・
ヘルパが呼び出された？
Yes
S312
ランタイム・ヘルパによる
処理
No
S314
トランザクション内で
アボート？
Yes
S316
アボート・ハンドラによる
処理
2
1
No
S320
トランザクションの終了
S318
NON-TXパスの実行
終了

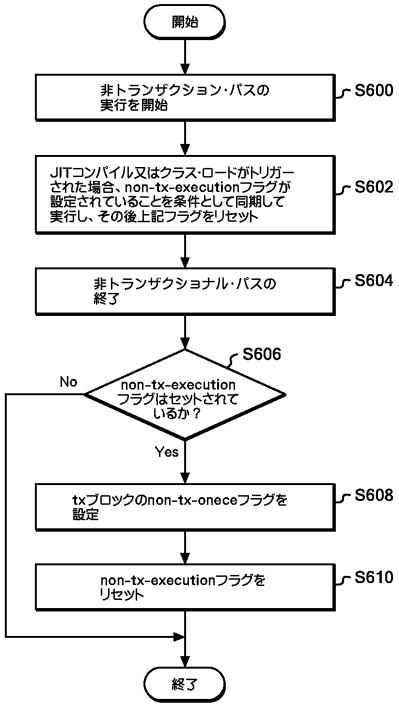
【図４】


開始
ランタイムヘルパーIDを引数として
トランザクション・アボート命令を
発行
S400
終了

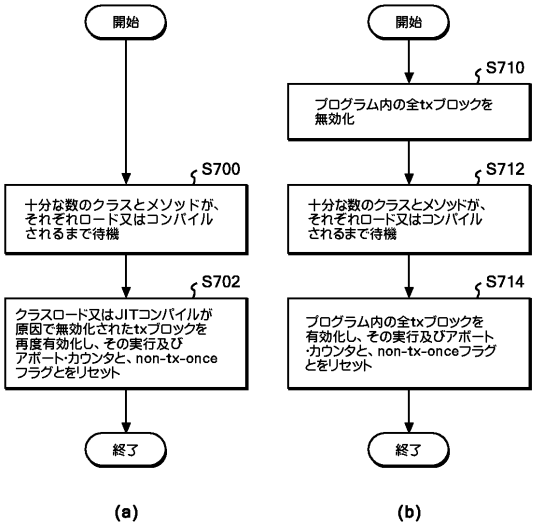
【図５】


開始
S500
アボートの原因はTLH割付又は非同期GCチェック？
Yes
No
S502
アボートの原因はシンボル解決？
Yes
No
S504
アボートの原因はJITコンパイル又はクラス・ロード？
No
Yes
X
S506
現在のtxブロックについてのnon-tx-onceフラグはセットされている？
Yes
No
S508
現在実行中のスレッドのnon-tx-executionフラグを設定
S510
Txブロックを無効化
終了１
(a)
X
S520
リトライ・カウンタ＞０？
Yes
No
S524
Txブロックのアボート・カウンタの増加
S526
アボート率は閾値より小さい？
Yes
No
S528
Txブロックを無効化
S522
リトライ・カウンタの減少
終了１
終了２
(b)

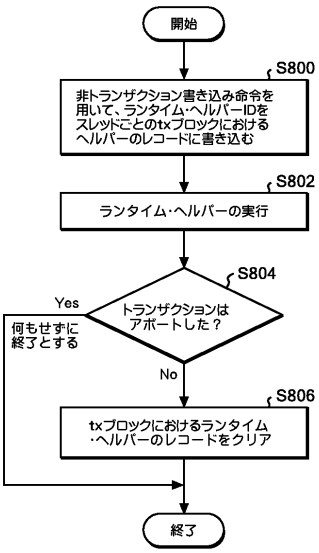
【図６】


開始
非トランザクション・パスの実行を開始
S600
JITコンパイル又はクラス・ロードがトリガーされた場合、non-tx-executionフラグが設定されていることを条件として同期して実行し、その後上記フラグをリセット
S602
非トランザクショナル・パスの終了
S604
S606
non-tx-executionフラグはセットされているか？
No
Yes
txブロックのnon-tx-oneceフラグを設定
S608
non-tx-executionフラグをリセット
S610
終了

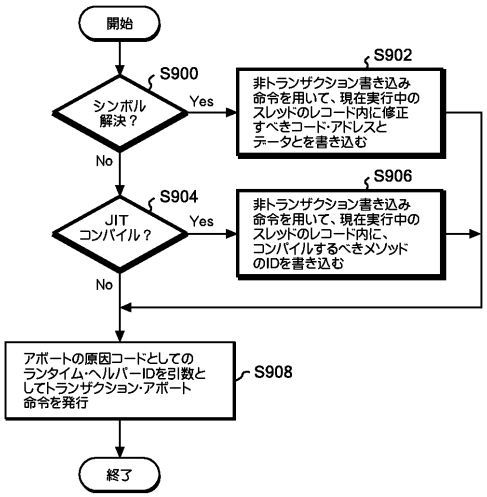
【図７】


開始
S700
十分な数のクラスとメソッドが、それぞれロード又はコンパイルされるまで待機
S702
クラスロード又はJITコンパイルが原因で無効化されたtxブロックを再度有効化し、その実行及びアボート・カウンタと、non-tx-onceフラグとをリセット
終了
(a)
開始
S710
プログラム内の全txブロックを無効化
S712
十分な数のクラスとメソッドが、それぞれロード又はコンパイルされるまで待機
S714
プログラム内の全txブロックを有効化し、その実行及びアボート・カウンタと、non-tx-onceフラグとをリセット
終了
(b)

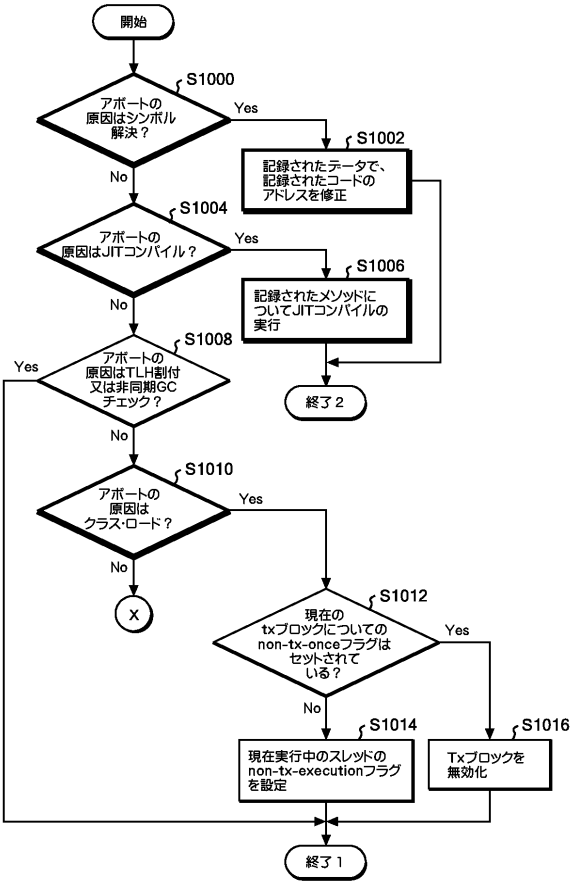
【図８】


開始
S800
非トランザクション書き込み命令を用いて、ランタイム・ヘルパーIDをスレッドごとのtxブロックにおけるヘルパーのレコードに書き込む
S802
ランタイム・ヘルパーの実行
S804
Yes
トランザクションはアボートした？
何もせずに終了とする
No
S806
txブロックにおけるランタイム・ヘルパーのレコードをクリア
終了

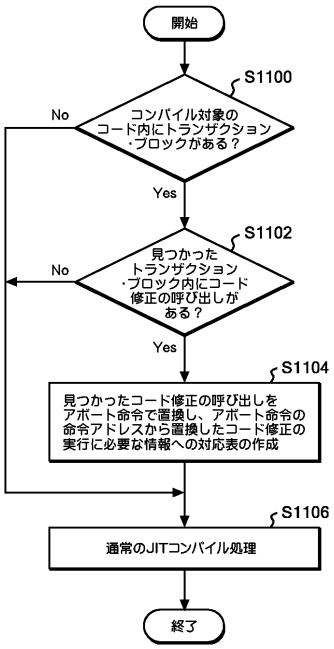
【図９】



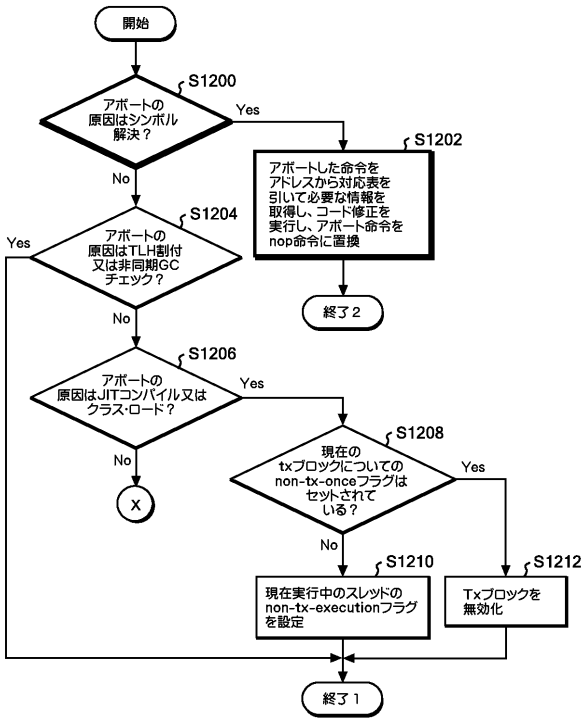
【図１０】



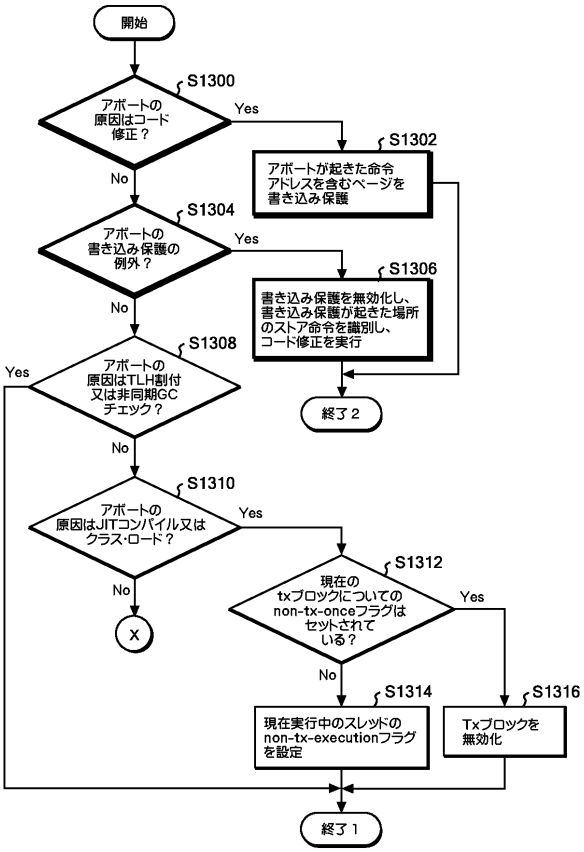
【図１１】



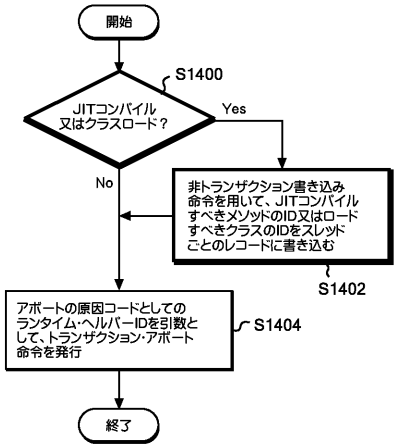
【図１２】



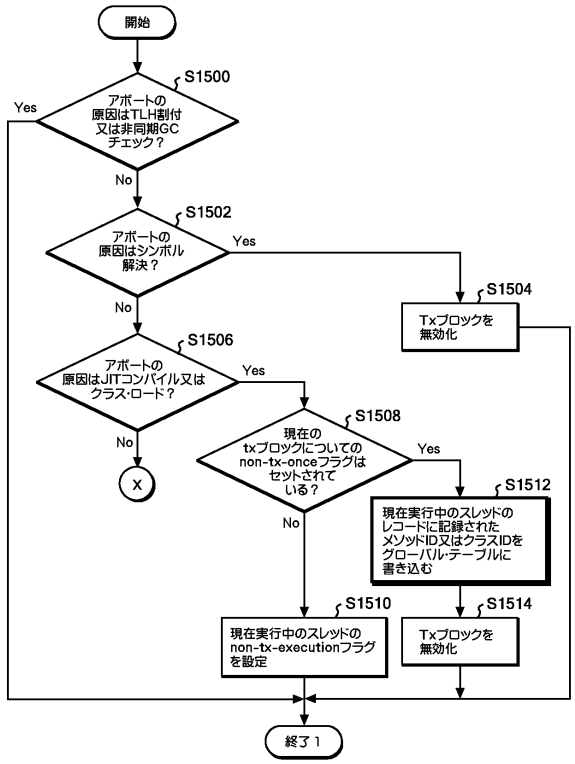
【図１３】



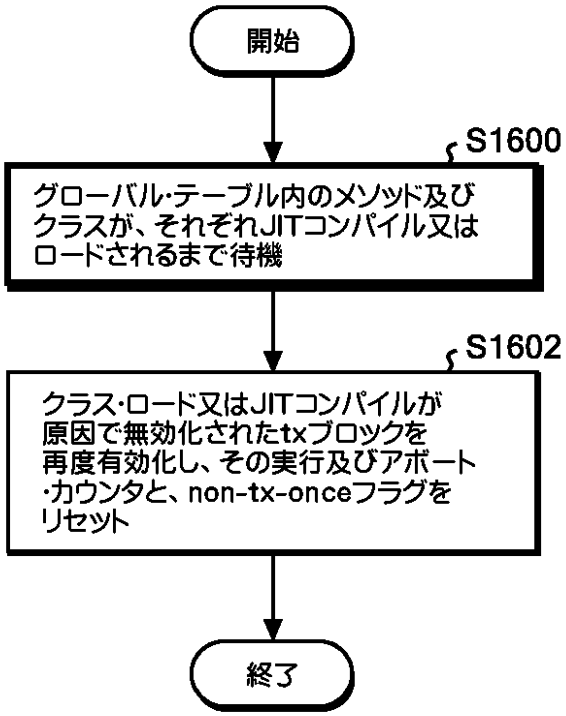
【図１４】



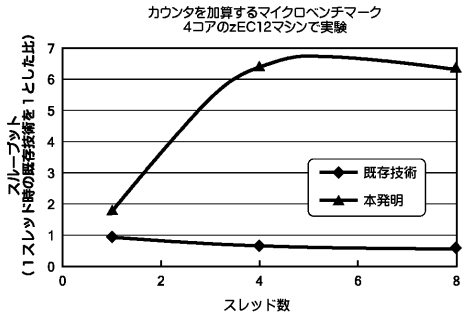
【図１５】



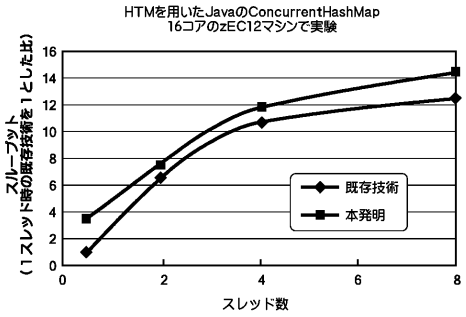
【図１６】



【図１７】


カウンタを加算するマイクロベンチマーク
4コアのzEC12マシンで実験
スループット
（1スレッド時の既存技術を1とした比）
既存技術
本発明
スレッド数

(a)


HTMを用いたJavaのConcurrentHashMap
16コアのzEC12マシンで実験
スループット
（1スレッド時の既存技術を1とした比）
既存技術
本発明
スレッド数

(b)

フロントページの続き

(74)代理人 100112690
弁理士 太佐 種一
(72)発明者 大平 怜
東京都江東区豊洲五丁目６番５２号 ＮＢＦ豊洲キャナルフロント 日本アイ・ビー・エム株式会社 東京基礎研究所内
(72)発明者 仲池 卓也
東京都江東区豊洲五丁目６番５２号 ＮＢＦ豊洲キャナルフロント 日本アイ・ビー・エム株式会社 東京基礎研究所内
(72)発明者 ホセ・ジー・カスタノス
アメリカ合衆国１０５９８ニューヨーク州ヨークタウン・ハイツ ルート１３４ キチャワン・ロード１１０１番地
(72)発明者 ペン・ウー
アメリカ合衆国１０５９８ニューヨーク州ヨークタウン・ハイツ ルート１３４ キチャワン・ロード１１０１番地

審査官 篠塚 隆

(56)参考文献 国際公開第２０１１／０８１７０４（WO，Ａ２）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ９／４６－９／５４
Ｇ０６Ｆ１２／００