(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3611295号
(P3611295)
(45)発行日　平成17年1月19日(2005.1.19)　(24)登録日　平成16年10月29日(2004.10.29)

(51)Int.Cl.[7]　F I
G06F 9/46　G06F 9/46　340F
G06F 12/00　G06F 12/00　572A
G06F 12/00　591

請求項の数 9　(全 15 頁)

(21)出願番号　特願2000-64718(P2000-64718)
(22)出願日　平成12年3月9日(2000.3.9)
(65)公開番号　特開2001-265611(P2001-265611A)
(43)公開日　平成13年9月28日(2001.9.28)
審査請求日　平成12年10月20日(2000.10.20)

前置審査

(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　ＭＡＳＣＨＩＮＥＳ　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
(74)代理人　100086243
弁理士　坂口　博
(72)発明者　河内谷　清久仁
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内

最終頁に続く

(54)【発明の名称】コンピュータシステム、メモリ管理方法及び記憶媒体

(57)【特許請求の範囲】
【請求項１】
プログラムを複数の実行主体に分けて実行すると共に、複数の当該実行主体がメモリ装置に格納されているデータを共有してそれぞれアクセスするようなデータ処理環境を有するコンピュータシステムにおいて、
前記メモリ装置に格納されている前記データは、特定の単一の実行主体のみから直接アクセスされる状態である局所性を有するか否かを示すフラグデータを有し、かつ当該データ自身が生成された際に、当該データを生成した実行主体に対して前記局所性を有することを示すように前記フラグデータをセットし、他の実行主体からアクセス可能な状態に変化させられる際に、その状態変化に先立って前記局所性を解除するように前記フラグデータを変更し、
前記実行主体は、アクセスしようとする前記データの前記フラグデータを調べ、当該データが自実行主体を対象とする前記局所性を有することが当該フラグデータにより示されている場合は、他の実行主体からのアクセスを拒否するロック処理を行わずに当該データにアクセスし、当該データが自実行主体を対象とする前記局所性を有さないことが当該フラグデータにより示されている場合は、当該ロック処理を行った後に当該データにアクセスすることを特徴とするコンピュータシステム。
【請求項２】
前記実行主体は、前記メモリ装置に格納されている前記データの中から、自実行主体を対象とする前記局所性を有することが前記フラグデータにより示されているデータであって

、自実行主体が参照ポインタを持たないデータを検出し、自由に使用できる記憶領域として解放することを特徴とする請求項１に記載のコンピュータシステム。

【請求項３】

複数のスレッドがオブジェクトを共有してそれぞれアクセスするようなデータ処理環境を有するコンピュータシステムにおいて、

前記オブジェクトは、当該オブジェクトを生成した特定のスレッドのみから直接アクセスされる状態である局所性を有するか否かを示すフラグデータを有し、かつ当該オブジェクト自身が生成された際に、当該オブジェクトを生成したスレッドに対して前記局所性を有することを示すように前記フラグデータをセットし、他のスレッドまたはオブジェクトからアクセス可能な状態に変化させられる際に、その状態変化に先立って前記局所性を解除するように前記フラグデータを変更し、

前記スレッドは、アクセスしようとする前記オブジェクトにおける前記フラグデータを調べ、当該オブジェクトが自スレッドを対象とする前記局所性を有することが当該フラグデータにより示されている場合は、他のスレッドまたは前記オブジェクトからのアクセスを拒否するロック処理を行わずに当該オブジェクトにアクセスし、当該オブジェクトが自スレッドを対象とする前記局所性を有さないことが当該フラグデータにより示されている場合は、当該ロック処理を行った後に当該オブジェクトにアクセスすることを特徴とするコンピュータシステム。

【請求項４】

前記スレッドは、自スレッドを対象とする前記局所性を有することが当該フラグデータにより示されているオブジェクトであって、自スレッドが参照ポインタを持たないオブジェクトを検出し、メモリ装置上における自由に使用できる記憶領域として解放することを特徴とする請求項３に記載のコンピュータシステム。

【請求項５】

プログラムを複数の実行主体に分けて実行すると共に、複数の当該実行主体がオブジェクトを共有してそれぞれアクセスするようなデータ処理環境におけるメモリ管理方法であって、

前記オブジェクトを生成する際に、当該オブジェクトを生成した特定の実行主体のみからアクセスされる状態である局所性を有することを示すフラグデータをセットするステップと、

前記特定の実行主体に対して前記局所性を有する前記オブジェクトが、当該特定の実行主体によって他の実行主体からアクセス可能な状態に変化させられる際に、その状態変化に先立って前記局所性を解除するように前記フラグデータを変更するステップと、

所定の前記実行主体が所定の前記オブジェクトにアクセスしようとする際に、当該オブジェクトの前記フラグデータを調べ、当該オブジェクトが自実行主体を対象とする前記局所性を有することが当該フラグデータにより示されている場合は、他の実行主体またはオブジェクトからのアクセスを拒否するロック処理を行わずに当該オブジェクトにアクセスし、当該オブジェクトが自実行主体を対象とする前記局所性を有さないことが当該フラグデータにより示されている場合は、当該ロック処理を行った後に当該オブジェクトにアクセスするステップとを含むことを特徴とするメモリ管理方法。

【請求項６】

前記オブジェクトの前記局所性を解除する前記フラグデータを変更するステップは、

前記オブジェクトが前記局所性を有していた際に当該局所性の対象である前記特定の実行主体からのアクセスにおいて省略されたロック処理を実行するステップを含むことを特徴とする請求項５に記載のメモリ管理方法。

【請求項７】

所定の実行主体が、所定のオブジェクトの前記フラグデータを調べ、自実行主体を対象とする前記局所性を有することが当該フラグデータにより示されているオブジェクトであって、自実行主体が参照ポインタを持たないオブジェクトを検出するステップと、

検出されたオブジェクトをメモリ装置上における自由に使用できる記憶領域として解放す

るステップとをさらに含むことを特徴とする請求項５に記載のメモリ管理方法。
【請求項８】
コンピュータに実行させるプログラムを当該コンピュータの入力手段が読取可能に記憶した記憶媒体において、
前記プログラムは、
オブジェクトを生成する際に、当該オブジェクトを生成した特定の実行主体のみからアクセスされる状態である局所性を有することを示すフラグデータをセットする処理と、
前記特定の実行主体に対して前記局所性を有する前記オブジェクトが、当該特定の実行主体によって他の実行主体からアクセス可能な状態に変化させられる際に、その状態変化に先立って前記局所性を解除するように前記フラグデータを変更する処理と、
所定の前記実行主体が所定の前記オブジェクトにアクセスしようとする際に、当該オブジェクトの前記フラグデータを調べ、当該オブジェクトが自実行主体を対象とする前記局所性を有することが当該フラグデータにより示されている場合は、他の実行主体またはオブジェクトからのアクセスを拒否するロック処理を行わずに当該オブジェクトにアクセスし、当該オブジェクトが自実行主体を対象とする前記局所性を有さないことが当該フラグデータにより示されている場合は、当該ロック処理を行った後に当該オブジェクトにアクセスする処理とを前記コンピュータに実行させることを特徴とする記憶媒体。
【請求項９】
所定の実行主体が、所定のオブジェクトの前記フラグデータを調べ、自実行主体を対象とする前記局所性を有することが当該フラグデータにより示されているオブジェクトであって、自実行主体が参照ポインタを持たないオブジェクトを検出する処理と、
検出されたオブジェクトをメモリ装置上における自由に使用できる記憶領域として解放する処理とを前記コンピュータにさらに実行させることを特徴とする請求項８に記載の記憶媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、プログラムを複数の実行主体（スレッド）に分けて実行すると共に、当該複数の実行主体がそれぞれ共有されたデータにアクセスするようなデータ処理環境を有するコンピュータシステムに関する。
【０００２】
【従来の技術】
コンピュータシステムにおいて、プログラムを複数の実行主体（スレッド）に分けて実行すると共に、当該複数の実行主体がそれぞれ共有されたデータにアクセスするようなデータ処理環境を考える。この種の処理環境では、所定の実行主体が所定のデータにアクセスする場合、当該データを他の実行主体が使用しないようにするロック（同期）処理を行う。このため、反対に、アクセスしようとするデータが他の実行主体によりアクセスされ、ロック（同期）されている場合は、アンロック（解放）されるまで待つこととなる。
【０００３】
このような処理環境では、特定の単一の実行主体以外からはアクセスされないデータが存在する場合がある。しかし、そのようなデータへのアクセスにおいても、当該実行主体がアクセスする場合にはロック（同期）処理が行われていた。すなわち、他の実行主体からアクセス要求される可能性のない場合にまで、ロック（同期）処理を行うという無駄な動作が行われていた。
【０００４】
また、この種の処理環境では、データを格納したメモリ装置の記憶領域において、いずれの実行主体からもアクセスされなくなったデータが残ってしまい、記憶領域が不必要に減少してしまう場合がある。そこで、必要に応じて、メモリ装置の記憶領域からそのような使用されなくなったデータをまとめて除去し、使用可能な記憶領域を増やすガーベジ・コレクションを行っている。

従来、ガーベジ・コレクションの手法としては、実行主体によるプログラムの実行を一度全て停止させ、記憶領域にある全てのデータの中からいずれの実行主体からもアクセスされなくなったデータを検索し除去する方法や、実行主体によるプログラムの実行と並行して少しずつ各データへのアクセスの有無を調べ、いずれの実行主体からもアクセスされなくなったデータを検索し除去する方法が採られている。

【０００５】

この種のデータ処理環境の例として、Ｊａｖａ言語で記述されたプログラムの動作環境（以下、Ｊａｖａの動作環境と称す）がある。Ｊａｖａ言語では、プログラムの実行主体であるスレッドが、メモリ装置の記憶領域に設定されたオブジェクトプール上に生成されるオブジェクトにアクセスすることにより、全ての処理が実行される。したがって、Ｊａｖａの動作環境では、ロック（同期）処理やガーベジ・コレクションが頻繁に発生する。

そのため、ロック（同期）処理を高速化する技術や、ガーベジ・コレクションに要する時間を短縮する技術の開発が望まれている。

【０００６】

【発明が解決しようとする課題】

上述したように、複数の実行主体（スレッド）が共有されたデータ（オブジェクト）にアクセスするようなデータ処理環境では、特定の単一の実行主体からのみアクセスが行われ、他の実行主体からアクセス要求される可能性のないデータに対しても、アクセスする際にロック（同期）処理を行うという無駄な動作が行われていたため、システムの全体性能を低下させていた。

【０００７】

また、ガーベジ・コレクションにおいては、実行主体によるプログラムの実行を一度全て停止させて不要なデータを除去する場合は、当該処理を行う際にシステム上の他の一切の処理を中断しなくてはならないため、ユーザにとって動作が煩雑となっていた。

【０００８】

一方、実行主体によるプログラムの実行と並行して不要なデータを除去する処理を実行する場合は、通常の処理を実行しながら、合間にデータの検索及び除去を実行するため、システムの全体性能を低下させてしまう。

【０００９】

そこで本発明は、単一の実行主体からのみアクセスが行われ、他の実行主体からアクセス要求される可能性のないデータに対しては、アクセスする際にロック（同期）処理を省略することにより、システムの負荷を軽減し、全体的な性能を向上させることを目的とする。

【００１０】

また本発明は、通常の処理と並行して行うガーベジ・コレクションの効率化を図ることにより、システムの全体性能の低下を軽減することを他の目的とする。

【００１１】

【課題を解決するための手段】

かかる目的のもと、本発明は、プログラムを複数の実行主体に分けて実行すると共に、この複数の実行主体がメモリ装置に格納されているデータを共有してそれぞれアクセスするようなデータ処理環境を有するコンピュータシステムにおいて、メモリ装置に格納されているデータは、単一の実行主体のみから直接アクセスされる状態である局所性を有するか否かを示すフラグデータを有し、実行主体は、アクセスしようとするデータにおけるこのフラグデータが、この実行主体に対して局所性を有することを示す場合は、他の実行主体からのアクセスを拒否するロック処理を行わずにこのデータにアクセスし、このフラグデータが、この実行主体に対して局所性を有さないことを示す場合は、ロック処理を行った後にこのデータにアクセスすることを特徴としている。

自実行主体に対して局所性を有するデータにアクセスする際に、ロック処理を省略することにより、実行主体の負荷を軽減し、システム全体の性能を向上させることが可能となる。

【００１２】
ここで、この実行主体は、メモリ装置に格納されているデータの中から、フラグデータがこの実行主体に対して局所性を有することを示しているデータであって、この実行主体が参照ポインタを持たないデータを検出し、自由に使用できる記憶領域として解放することを特徴としている。
このような構成とすれば、実行主体から参照されないデータを消去して記憶領域を解放するための処理を、所定の実行主体においてローカルに実行できるため、他の実行主体による通常の処理を停止することなく、並行して行うことができる点で好ましい。
【００１３】
また、本発明は、複数のスレッドがオブジェクトを共有してそれぞれアクセスするようなデータ処理環境を有するコンピュータシステムにおいて、オブジェクトは、単一のスレッドのみから直接アクセスされる状態である局所性を有するか否かを示すフラグデータを有し、スレッドは、アクセスしようとするオブジェクトにおけるフラグデータが、このスレッドに対して局所性を有することを示す場合は、他のスレッドまたはオブジェクトからのアクセスを拒否するロック処理を行わずにこのオブジェクトにアクセスし、このフラグデータが、このスレッドに対して局所性を有さないことを示す場合は、ロック処理を行った後にこのオブジェクトにアクセスすることを特徴としている。
【００１４】
ここで、このオブジェクトは、生成された際に、このオブジェクトを生成したスレッドに対して局所性を有することを示すようにフラグデータをセットし、他のスレッドまたはオブジェクトからアクセス可能な状態に変化させられる際に、その状態変化に先立って前記フラグデータにて示される局所性を解除することを特徴としている。すなわち、全てのオブジェクトは初期的にはいずれかのスレッドに対して局所性を有し、所定の状況が発生した場合に、当該スレッドによるアクセスに関する状態変化に応じてこの局所性を解除する。なお、オブジェクトの状態を変化させるべき状況が発生したことをスレッドが検出する方法としては、例えばこのオブジェクトが他のオブジェクトもしくはグローバルなデータに代入された場合とすることができる。さらにこのとき、代入先のオブジェクトが局所的かどうかまで総括的にチェックして解除するようにしても良い。
【００１５】
さらにこのスレッドは、フラグデータがこのスレッドに対して局所性を有することを示しているオブジェクトであって、このスレッドが参照ポインタを持たないオブジェクトを検出し、メモリ装置上における自由に使用できる記憶領域として解放することを特徴としている。
【００１６】
また、本発明は、プログラムを複数の実行主体に分けて実行すると共に、この複数の実行主体がオブジェクトを共有してそれぞれアクセスするようなデータ処理環境におけるメモリ管理方法であって、オブジェクトを生成する際に、このオブジェクトを生成した実行主体のみからアクセスされる状態である局所性を有することを示すフラグデータをセットするステップと、所定の前記実行主体に対して局所性を有する前記オブジェクトが、当該実行主体によって他の実行主体からアクセス可能な状態に変化させられる際に、その状態変化に先立って前記フラグデータにて示される局所性を解除するステップと、実行主体が所定のオブジェクトにアクセスしようとする際に、このオブジェクトのフラグデータが、この実行主体に対して局所性を有することを示す場合は、他の実行主体またはオブジェクトからのアクセスを拒否するロック処理を行わずにこのオブジェクトにアクセスし、このフラグデータが、この実行主体に対して局所性を有さないことを示す場合は、このロック処理を行った後に当該オブジェクトにアクセスするステップとを含むことを特徴としている。
自実行主体に対して局所性を有するオブジェクトにアクセスする際に、ロック処理を省略することにより、システム全体の処理速度を高速化させることが可能となる。
【００１７】

ここで、このオブジェクトのフラグデータにて示される局所性を解除するステップは、このオブジェクトが所定の実行主体に対して局所性を有していた際にこの実行主体からのアクセスにおいて省略されたロック処理を実行するステップを含むことを特徴とする。
このような構成とすれば、所定のオブジェクトにおいて局所性が解除された場合に、ロック処理を省略したことによって処理の整合性が崩れることを回避することができる点でより優れている。
省略されたロック処理を実行するためには、このロック処理が省略されたことを認識することが必要であるが、これは、実行主体の実行スタックを参照したり、ロック処理を省略するたびに省略した回数をカウントしておくことにより可能である。

【００１８】

さらにまた、本発明は、プログラムを複数の実行主体に分けて実行すると共に、この複数の実行主体がオブジェクトを共有してそれぞれアクセスするようなデータ処理環境におけるメモリ管理方法であって、オブジェクトを生成する際に、このオブジェクトを生成した実行主体のみからアクセスされる状態である局所性を有することを示すフラグデータをセットするステップと、実行主体が、このフラグデータがこの実行主体に対して局所性を有することを示しているオブジェクトであって、この実行主体が参照ポインタを持たないオブジェクトを検出するステップと、検出されたオブジェクトをメモリ装置上における自由に使用できる記憶領域として解放するステップとを含むことを特徴としている。
このような構成とすれば、ガーベジ・コレクションを、所定の実行主体においてローカルに実行できるため、他の実行主体による通常の処理を停止することなく、並行して行うことができる点で好ましい。

【００１９】

また、本発明は、コンピュータに実行させるプログラムをこのコンピュータの入力手段が読取可能に記憶した記憶媒体において、このプログラムは、オブジェクトを生成する際に、このオブジェクトを生成した実行主体のみからアクセスされる状態である局所性を有することを示すフラグデータをセットする処理と、所定の前記実行主体に対して局所性を有する前記オブジェクトが、当該実行主体によって他の実行主体からアクセス可能な状態に変化させられる際に、その状態変化に先立って前記フラグデータにて示される局所性を解除する処理と、実行主体が所定のオブジェクトにアクセスしようとする際に、このオブジェクトのフラグデータが、この実行主体に対して局所性を有することを示す場合は、他の実行主体またはオブジェクトからのアクセスを拒否するロック処理を行わずにこのオブジェクトにアクセスし、このフラグデータが、このスレッドに対して局所性を有さないことを示す場合は、ロック処理を行った後にこのオブジェクトにアクセスする処理とをこのコンピュータに実行させることを特徴としている。
このような構成とすれば、このプログラムをインストールした全てのコンピュータにおいて、自実行主体に対して局所性を有するオブジェクトにアクセスする際に、ロック処理を省略することにより、システム全体の処理速度を高速化させることが可能となる。

【００２０】

コンピュータに実行させるプログラムをこのコンピュータの入力手段が読取可能に記憶した記憶媒体において、このプログラムは、オブジェクトを生成する際に、このオブジェクトを生成した実行主体のみからアクセスされる状態である局所性を有することを示すフラグデータをセットする処理と、実行主体が、このフラグデータがこの実行主体に対して局所性を有することを示しているオブジェクトであって、この実行主体が参照ポインタを持たないオブジェクトを検出する処理と、検出されたオブジェクトをメモリ装置上における自由に使用できる記憶領域として解放する処理とを前記コンピュータに実行させることを特徴している。
このような構成とすれば、このプログラムをインストールした全てのコンピュータにおいて、ガーベジ・コレクションを、所定の実行主体においてローカルに実行できるため、他の実行主体による通常の処理を停止することなく、並行して行うことが可能となる。

【００２１】

さらに、本発明は、コンピュータに、オブジェクトを生成する際に、このオブジェクトを生成した実行主体のみからアクセスされる状態である局所性を有することを示すフラグデータをセットする処理と、所定の前記実行主体に対して局所性を有する前記オブジェクトが、当該実行主体によって他の実行主体からアクセス可能な状態に変化させられる際に、その状態変化に先立って前記フラグデータにて示される局所性を解除する処理と、実行主体が所定のオブジェクトにアクセスしようとする際に、このオブジェクトのフラグデータが、この実行主体に対して局所性を有することを示す場合は、他の実行主体またはオブジェクトからのアクセスを拒否するロック処理を行わずにこのオブジェクトにアクセスし、このフラグデータが、このスレッドに対して局所性を有さないことを示す場合は、ロック処理を行った後にこのオブジェクトにアクセスする処理とを実行させるプログラムを記憶する記憶手段と、この記憶手段からこのプログラムを読み出してこのプログラムを送信する送信手段とを備えたことを特徴としている。
このような構成とすれば、このプログラムをダウンロードして実行する全てのコンピュータにおいて、自実行主体に対して局所性を有するオブジェクトにアクセスする際に、ロック処理を省略することにより、システム全体の処理速度を高速化させることが可能となる。

【００２２】
さらにまた、本発明は、コンピュータに、オブジェクトを生成する際に、このオブジェクトを生成した実行主体のみからアクセスされる状態である局所性を有することを示すフラグデータをセットする処理と、実行主体が、このフラグデータがこの実行主体に対して局所性を有することを示しているオブジェクトであって、この実行主体が参照ポインタを持たないオブジェクトを検出する処理と、検出されたオブジェクトをメモリ装置上における自由に使用できる記憶領域として解放する処理とを実行させるプログラムを記憶する記憶手段と、この記憶手段からこのプログラムを読み出してこのプログラムを送信する送信手段とを備えたことを特徴としている。
このような構成とすれば、このプログラムをダウンロードして実行する全てのコンピュータにおいて、ガーベジ・コレクションを、所定の実行主体においてローカルに実行できるため、他の実行主体による通常の処理を停止することなく、並行して行うことが可能となる。

【００２３】
【発明の実施の形態】
以下、添付図面に示す実施の形態に基づいてこの発明を詳細に説明する。
図１は、本実施の形態におけるメモリ管理を実現するコンピュータシステムの全体構成を説明する図である。本実施の形態のコンピュータシステムは、プログラムを複数の実行主体であるスレッドに分けて実行すると共に、複数の当該スレッドがメモリ装置に格納されているオブジェクトを共有してそれぞれアクセスするようなデータ処理環境を有する。以下の説明では、特に、そのようなデータ処理環境として、Ｊａｖａ言語で記述されたプログラムの動作環境（Ｊａｖａの動作環境）を有するコンピュータシステムを例として説明する。

【００２４】
図１において、符号１１０はオブジェクトプールであり、メモリ装置上においてオブジェクトを格納する記憶領域である。符号１２１～１２４はオブジェクトである。符号１３１～１３３はスレッドである。符号１４１はスレッド１３１からオブジェクト１２１への参照ポインタ、符号１４２はスレッド１３２からオブジェクト１２２への参照ポインタ、符号１４３はスレッド１３３からオブジェクト１２３への参照ポインタである。また、符号１４４はオブジェクト１２１からオブジェクト１２２への参照ポインタ、符号１４５はオブジェクト１２１からオブジェクト１２４への参照ポインタである。符号１５１～１５４はスレッド局所フラグである。

【００２５】
通常の動作において、スレッド１３１～１３３は、各々の処理の必要に応じて、オブジェ

クトプール１１０にオブジェクトを生成し、使用する。図１を参照すると、スレッド１３１はオブジェクト１２１を参照すると共に、オブジェクト１２１を介してオブジェクト１２２及びオブジェクト１２４を参照する。同様に、スレッド１３２はオブジェクト１２２を参照し、スレッド１３３はオブジェクト１２３を参照する。
この状態をオブジェクト側から見ると、オブジェクト１２１はスレッド１３１のみから直接参照される。またオブジェクト１２２は、スレッド１３２から直接参照されると共に、オブジェクト１２１を介してスレッド１３１からも参照される。オブジェクト１２３はスレッド１３３のみから直接参照される。オブジェクト１２４は、オブジェクト１２１を介してスレッド１３１のみから参照される。

【００２６】

ここで、オブジェクトが単一のスレッドのみから直接参照されている状態を、以下の説明では、スレッド局所性を有すると呼ぶ。
すなわち、図１の例では、スレッド１３１のみから直接参照されているオブジェクト１２１と、スレッド１３３のみから直接参照されているオブジェクト１２３とがスレッド局所性を有する。なお、オブジェクト１２４は、スレッド１３１のみから参照されているのであるが、オブジェクト１２１を介して参照されているので、スレッド局所性を有さない。
オブジェクト１２４のように、他のオブジェクトから参照されているオブジェクトを、スレッド局所性を有さないこととするのは、このようなオブジェクトが実際に単一のスレッドのみから参照されているのか、当該他のオブジェクトを介して複数のスレッドから参照されているのかを判別する作業が繁雑となるからである。

【００２７】

各オブジェクト１２１、１２２、１２３、１２４は、自オブジェクトがスレッド局所性を有するかどうかを示すスレッド局所フラグ１５１、１５２、１５３、１５４を持つ。本実施の形態では、オブジェクト１２１、１２２、１２３、１２４がスレッド局所性を有する場合にスレッド局所性フラグ１５１、１５２、１５３、１５４を立て（オン）、スレッド局所性を有さない場合にスレッド局所性フラグ１５１、１５２、１５３、１５４を倒す（オフ）こととする。

【００２８】

スレッド１３１、１３２、１３３は、オブジェクトを生成する際に、当該オブジェクトのスレッド局所フラグをオンにセットする。生成されたオブジェクトは、当該オブジェクトを生成したスレッド１３１、１３２、１３３によって他のスレッドまたは他のオブジェクトから参照可能な状態に変化させられる際に、その状態変化に先立って自オブジェクトのスレッド局所フラグをオフにする。スレッド１３１、１３２、１３３が、自スレッドに対してスレッド局所性を有するオブジェクトを、他のスレッドまたは他のオブジェクトから参照可能な状態に変化させるべき状況が発生したことを検出する方法としては種々の方法が考えられる。例えば、当該オブジェクトが他のオブジェクトもしくはグローバルなデータに代入された場合に、状態を変化させることとすることができる。さらにこのとき、代入先のオブジェクトが局所的かどうかまで総括的にチェックして解除するようにしても良い。
なお、一度オフにしたスレッド局所フラグを再びオンにすることはない。実際には、所定のスレッドに対してスレッド局所性を有していたオブジェクトが、一時的に他のスレッドから参照された後、当該参照ポインタが消去されて再びスレッド局所性を回復する場合がある。しかし、このような場合分けを行うと、スレッド局所性を判断するための処理が複雑となるので、スレッド局所フラグはオフにしたままとする。

【００２９】

図２は、スレッド１３１、１３２、１３３がオブジェクト１２１、１２２、１２３を生成する動作を説明するフローチャートである。
図２を参照すると、スレッド１３１、１３２、１３３は、まず、オブジェクトプール１１０の所定の領域をオブジェクト領域として確保し、初期化する（ステップ２０１）。次に、当該オブジェクト領域にスレッド局所フラグ１５１、１５２、１５３を設定し、オンに

セットする（ステップ２０２）。そして、生成されたオブジェクト１２１、１２２、１２３への参照ポインタをスレッド１３１、１３２、１３３内に格納する（ステップ２０３）。

【００３０】

このようにして、オブジェクト１２１、１２２、１２３は、生成された時点では全てスレッド局所フラグ１５１、１５２、１５３がオンとなっている。図１は、この後、オブジェクト１２２がスレッド局所性を失い、スレッド局所フラグ１５２がオフとなった状態を示している。なお、図１の例では、オブジェクト１２２はオブジェクト１２１から参照可能となっている。

また、図１に示すオブジェクト１２４は、いずれかのスレッド１３１、１３２、１３３により生成された後、スレッド局所性を失ってスレッド局所フラグ１５４がオフとなり、さらにその後、オブジェクト１２１から参照可能な状態となり、他の参照ポインタが消去されてオブジェクト１２１からの参照のみが残った状態である。

【００３１】

Ｊａｖａのように複数のスレッドがオブジェクトを共有してそれぞれアクセスするようなデータ処理環境では、所定のスレッドが所定のオブジェクトを参照している間は、他のスレッドが当該オブジェクトを参照できないようにする必要がある。そこで、各スレッドは、オブジェクトを参照する際にロック（同期）処理を行って、他のスレッドが当該オブジェクトにアクセスすることを拒絶するようにする。他のスレッドのロック（同期）処理によってオブジェクトへのアクセスを拒絶されたスレッドは、当該オブジェクトがアンロック（解放）されるまで当該オブジェクトの参照を待ち合わせることとなる。

【００３２】

しかしながら、本実施の形態におけるスレッド局所性を有するオブジェクトは、当該オブジェクトを参照する主体が単一のスレッドに限定されている。したがって、スレッドが自スレッドに対してスレッド局所性を有するオブジェクトを参照する場合に関しては、ロック（同期）処理を省略することができる。

【００３３】

図３は、スレッド１３１、１３２、１３３がオブジェクト１２１、１２２、１２３、１２４を参照する際のロック（同期）処理に関わる動作を説明するフローチャートである。

図３を参照すると、スレッド１３１、１３２、１３３は、まず、参照しようとするオブジェクト１２１、１２２、１２３、１２４のスレッド局所フラグ１５１、１５２、１５３、１５４がオンにセットされているかどうかを調べる（ステップ３０１）。そして、スレッド局所フラグ１５１、１５２、１５３、１５４がオンであれば、その後の処理を省略して当該オブジェクト１２１、１２２、１２３、１２４を参照する。ただし、このとき、必要に応じてロック（同期）処理の省略に伴う処理を実行する（ステップ３０５）。スレッド１３１がオブジェクト１２１を参照する場合、及びスレッド１３３がオブジェクト１２３を参照する場合がこれに該当する。ロック（同期）処理の省略に伴う処理の内容については後述する。

【００３４】

これに対し、スレッド局所フラグ１５１、１５２、１５３、１５４がオフである場合は、通常通り、ロック（同期）処理を行う。すなわち、オブジェクト１２１、１２２、１２３、１２４を参照しようとするスレッド１３１、１３２、１３３は、まず、当該オブジェクト１２１、１２２、１２３、１２４が他のスレッド１３１、１３２、１３３によりロックされているかどうかを調べる（ステップ３０２）。そして、ロックされている場合は、アンロックされるまで待つ（ステップ３０４）。一方、当該オブジェクト１２１、１２２、１２３、１２４が他のスレッド１３１、１３２、１３３によりロックされていない場合（アンロックされた場合を含む）は、当該オブジェクト１２１、１２２、１２３、１２４に対してロック（同期）処理を行い（ステップ３０３）、参照する。スレッド１３２がオブジェクト１２２を参照する場合、及びスレッド１３１がオブジェクト１２１を介してオブジェクト１２２またはオブジェクト１２４を参照する場合がこれに該当する。

【００３５】
本実施の形態において、スレッドがオブジェクトを参照する際のロック（同期）処理を省略した場合は、当該オブジェクトを参照した後に行うアンロック（解放）処理も省略できる。
図４は、スレッド１３１、１３２、１３３がオブジェクト１２１、１２２、１２３、１２４を参照した後のアンロック（解放）処理に関わる動作を説明するフローチャートである。
図４を参照すると、スレッド１３１、１３２、１３３は、まず、参照したオブジェクト１２１、１２２、１２３、１２４のスレッド局所フラグ１５１、１５２、１５３、１５４がオンにセットされているかどうかを調べる（ステップ４０１）。そして、スレッド局所フラグ１５１、１５２、１５３、１５４がオンであれば、その後の処理を省略して当該オブジェクト１２１、１２２、１２３、１２４の参照を終了する。ただし、このとき、必要に応じてアンロック（解放）処理の省略に伴う処理を実行する（ステップ４０３）。ロック（同期）処理の省略の場合と同様に、スレッド１３１がオブジェクト１２１を参照した場合、及びスレッド１３３がオブジェクト１２３を参照した場合がこれに該当する。
一方、スレッド局所フラグ１５１、１５２、１５３、１５４がオフである場合は、通常通り、アンロック（解放）処理を行う（ステップ４０２）。
【００３６】
次に、オブジェクト１２１、１２３のように、スレッド局所フラグがオンにセットされており、特定のスレッドに対してスレッド局所性を有するオブジェクトが、当該スレッド局所性の対象ではない他のスレッドまたはオブジェクトに参照可能な状態に変更される際の動作を説明する。図５は、この際の動作を説明するフローチャートである。
【００３７】
スレッド１３１、１３２、１３３が、自スレッドに対してスレッド局所性を有するオブジェクト１２１、１２２、１２３、１２４に関して、他のスレッドまたは他のオブジェクトから参照可能な状態に変化させるべき状況が発生したことを検出したものとする。上述したように、このような状況を検出する方法としては種々の方法を採ることができるが、ここでは、当該オブジェクトが他のオブジェクトもしくはグローバルなデータに代入された場合に、状態を変化させるべき状況が発生したと認識することとする。
【００３８】
この場合、まず、スレッド１３１、１３２、１３３は、まず、アクセスしようとするオブジェクト１２１、１２２、１２３、１２４のスレッド局所フラグ１５１、１５２、１５３、１５４がオンにセットされているかどうかを調べる（ステップ５０１）。そして、スレッド局所フラグ１５１、１５２、１５３、１５４がオフであれば、何ら特別の処理を行うことなく、オブジェクト１２１、１２２、１２３、１２４への参照ポインタを当該オブジェクトを代入しようとした他のオブジェクトやデータに代入する（ステップ５０４）。オブジェクト１２２、１２４が他のオブジェクトまたはグローバルなデータに代入された場合がこれに該当する。
【００３９】
これに対し、スレッド局所フラグ１５１、１５２、１５３、１５４がオンである場合、スレッド１３１、１３２、１３３は、まず、当該オブジェクト１２１、１２２、１２３、１２４に対して、当該スレッド局所フラグ１５１、１５２、１５３、１５４をオフにするように指示する。これに応じて、オブジェクト１２１、１２２、１２３、１２４が当該スレッド局所フラグ１５１、１５２、１５３、１５４をオフにすると（ステップ５０２）、当該オブジェクト１２１、１２２、１２３、１２４のスレッド局所性が失われることとなるが、これに伴って、以下の処理を行うことが必要となる。
【００４０】
例として、スレッド１３３に対してスレッド局所性を有するオブジェクト１２３がスレッド局所性を解除される場合について考える。すなわち、オブジェクト１２３のスレッド局所性が解除された後、他のスレッドまたは他のオブジェクトがオブジェクト１２３をロッ

クしようとする場合、当該オブジェクト１２３がスレッド局所性の対象であったスレッド１３３によりロックされていたならば、スレッド１３２はオブジェクト１２３がアンロック（解放）されるまで参照を待たなければならない。しかし、スレッド１３２が参照しようとするまで、オブジェクト１２３はスレッド１３３に対してスレッド局所性を有していたため、スレッド１３３による参照の際にはロック（同期）処理及びアンロック（解放）処理が省略されている。したがって、スレッド１３２による参照に先立って、省略されていたロック（同期）処理及びアンロック（解放）処理を実行して、処理における整合性を取る必要がある。

【００４１】

したがって、スレッド局所フラグ１５１、１５２、１５３、１５４をオフにした後、当該オブジェクト１２１、１２２、１２３、１２４におけるスレッド局所性の対象であったスレッド１３１、１３２、１３３は、省略されていた当該オブジェクト１２１、１２２、１２３、１２４におけるロック（同期）処理及びアンロック（解放）処理を実行する（ステップ５０３）。そして、整合性がとれた後に、オブジェクト１２１、１２２、１２３、１２４への参照ポインタを当該オブジェクトを代入しようとした他のオブジェクトやデータに代入する（ステップ５０４）。

なお、この整合性確保のための処理は、オブジェクト１２１、１２２、１２３、１２４がスレッド局所性を失う直前に、当該オブジェクト１２１、１２２、１２３、１２４におけるスレッド局所性の対象であったスレッド１３１、１３２、１３３により実行されるため、Ｒａｃｉｎｇ等の問題は発生しない。

【００４２】

ステップ５０３で省略されていたロック（同期）処理及びアンロック（解放）処理を実行する場合、新規にアクセスしようとする要求が発生した時点で省略されているロック（同期）処理及びアンロック（解放）処理を認識することが必要となる。この認識を行う手法は任意であるが、ここでは次の二つの方法を例示する。

【００４３】

第１の手法は、スレッド局所性の対象であったスレッドの実行スタックをスキャンして、省略されているロック（同期）処理及びアンロック（解放）処理を認識する方法である。この場合、オブジェクトが参照されるたびに、スレッドの実行スタックには参照履歴が自動的に格納されるので、図３のフローチャートにおけるロック（同期）処理の省略に伴う処理（ステップ３０５）及び図４のフローチャートにおけるアンロック（解放）処理の省略に伴う処理（ステップ４０３）は実体的な処理が存在しないこととなる。しかし、オブジェクトにおけるスレッド局所性が失われる際に、スレッドの実行スタックをスキャンすることによるオーバーヘッドが生じる。

【００４４】

第２の手法は、ロック（同期）処理及びアンロック（解放）処理が省略された際に、省略された回数をカウントしておき、オブジェクトのスレッド局所性が失われる際に、当該カウント値に基づいて省略された処理を実行する方法である。この場合、図３のフローチャートにおけるロック（同期）処理の省略に伴う処理（ステップ３０５）及び図４のフローチャートにおけるアンロック（解放）処理の省略に伴う処理（ステップ４０３）の内容が、省略された回数をカウントする処理となる。この手法によれば、オブジェクトにおけるスレッド局所性が失われる際に、スレッドの実行スタックをスキャンすることによるオーバーヘッドは生じない。また、ステップ３０５、４０３で実行されるカウント処理は、省略されたロック（同期）処理及びアンロック（解放）処理よりもはるかに軽いため、システムに対する負担を大幅に軽減することができる。

【００４５】

上述したように、複数のスレッドがオブジェクトを共有してそれぞれアクセスするようなデータ処理環境では、オブジェクトプールに、いずれのスレッドからも参照されないオブジェクトが生じる場合がある。すなわち、スレッドは、種々の処理を実行するためにオブジェクトを生成するが、その後、当該オブジェクトを参照する必要が無くなり、参照ポイ

ンタが消去されると、当該オブジェクトは全く処理の実行に関与しない状態でオブジェクトプールに放置される。したがって、このような放置されたオブジェクトを効率的に回収し、空き領域として使用するガーベジ・コレクションを行うことが重要となる。

【００４６】

このような放置されたオブジェクトを回収するには、オブジェクトがいずれのスレッドからも参照されていないことを確認しなければならない。したがって、一般に、全てのスレッドの参照ポインタ群をたどって、参照の有無を調べることが必要となる。

しかし、本実施の形態は、特定のスレッドに対してスレッド局所性を有するかどうかを示すスレッド局所フラグをオブジェクトに付加しているため、スレッド局所フラグがオンにセットされているオブジェクトでは、当該スレッド局所性の対象であるスレッドだけが当該オブジェクトを参照することが保証されている。したがって、当該スレッド内に当該オブジェクトへの参照ポインタが残っているかどうかを調べるだけで、当該オブジェクトを回収できるかどうかを判断することができる。

【００４７】

図６は、スレッド局所フラグを用いたガーベジ・コレクションの手順の一例を説明するフローチャートである。

図６を参照すると、ガーベジ・コレクションを実行しようとするスレッドは、自スレッドに対してスレッド局所性を有するオブジェクトを一つ選択する（ステップ６０１）。そして、当該オブジェクトへの参照ポインタが自スレッド内にあるかどうかを調べ（ステップ６０３）、参照ポインタがない場合には当該オブジェクトを回収する（ステップ６０４）。

以上の動作を繰り返して行い、自スレッドに対して局所性を有する全てのオブジェクトについて調べたならば、処理を終了する（ステップ６０２）。

【００４８】

以上のようにして、所定のスレッドとの関係においてスレッド局所フラグをオンにセットしていたオブジェクトのうちで、当該スレッドから参照されない（すなわち、いずれのスレッドからも参照されない）オブジェクトを回収することができる。この処理は、スレッド自身のスタックを１段スキャンするだけで所定のオブジェクトを回収できるかどうかを判断でき、オブジェクト内部の参照ポインタのトレースを必要としないため、極めて高速に実行することができる。また、この処理は、所定のスレッドがローカルに実行するため、他のスレッドによる通常の処理を停止させることなく、並行して実行することができる。

【００４９】

なお、本実施の形態によるスレッド局所フラグを用いたガーベジ・コレクションを、各スレッドにおいて順次実行したとしても、スレッド局所フラグがオフであっていずれのスレッドからも参照されないようなオブジェクトは回収することができない。そのようなオブジェクトは、全てのスレッドによる処理を一度停止させ、オブジェクトプール中の全てのオブジェクトの中からいずれのスレッドからも参照されないオブジェクトを検索するといった、従来の方法により、ガーベジ・コレクションを実行すれば良い。すなわち、本実施の形態によるスレッド局所フラグを用いたガーベジ・コレクションは、従来の技術によるガーベジ・コレクションと併用することが可能である。

【００５０】

【発明の効果】

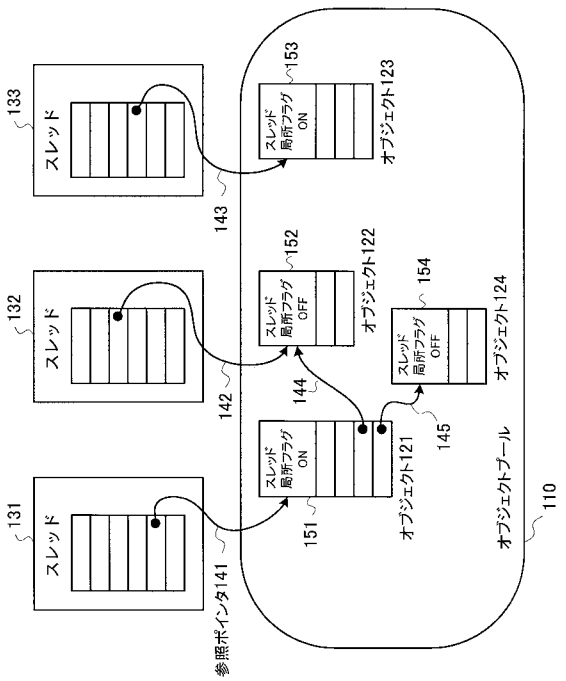
以上説明したように、本発明によれば、単一の実行主体からのみアクセスが行われ、他の実行主体からアクセス要求される可能性のないデータに対しては、アクセスする際にロック（同期）処理を省略することにより、システムの負荷を軽減し、全体的な性能を向上させることができる。

【００５１】

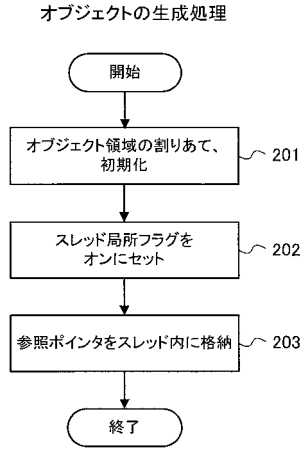
また、通常の処理と並行して行うガーベジ・コレクションの効率化を図ることにより、システムの全体性能の低下を軽減することができる。

【図面の簡単な説明】
【図１】本実施の形態におけるメモリ管理を実現するコンピュータシステムの全体構成を説明する図である。
【図２】本実施の形態におけるスレッドがオブジェクトを生成する動作を説明するフローチャートである。
【図３】本実施の形態におけるスレッドがオブジェクトを参照する際のロック（同期）処理に関わる動作を説明するフローチャートである。
【図４】本実施の形態におけるスレッドがオブジェクトを参照した後のアンロック（解放）処理に関わる動作を説明するフローチャートである。
【図５】本実施の形態における特定のスレッドに対してスレッド局所性を有するオブジェクトがスレッド局所性を解除される際の動作を説明するフローチャートである。
【図６】本実施の形態におけるスレッド局所フラグを用いたガーベジ・コレクションの手順の一例を説明するフローチャートである。
【符号の説明】
１１０…オブジェクトプール、１２１、１２２、１２３、１２４…オブジェクト、１３１、１３２、１３３…スレッド、１４１、１４２、１４３、１４４、１４５…参照ポインタ、１５１、１５２、１５３、１５４…スレッド局所フラグ

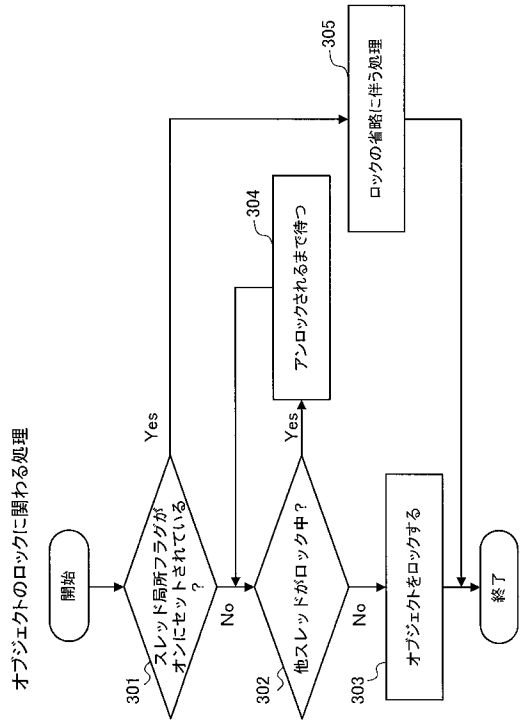
【図１】



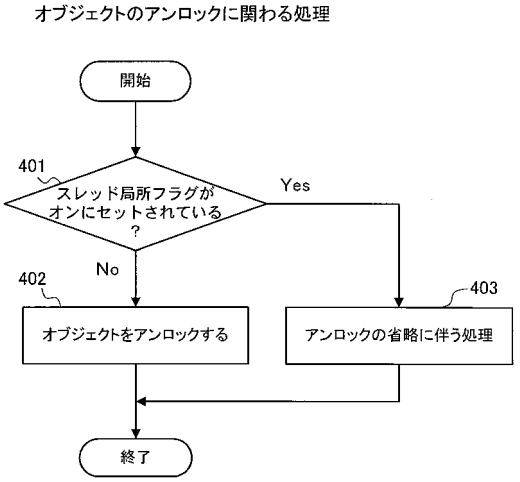
【図２】



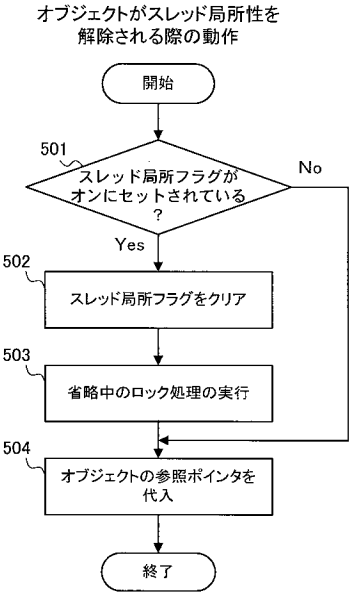
【図３】


オブジェクトのロックに関わる処理
開始
スレッド局所フラグがオンにセットされている？
Yes
No
301
他スレッドがロック中？
Yes
No
302
オブジェクトをロックする
303
アンロックされるまで待つ
304
ロックの省略に伴う処理
305
終了

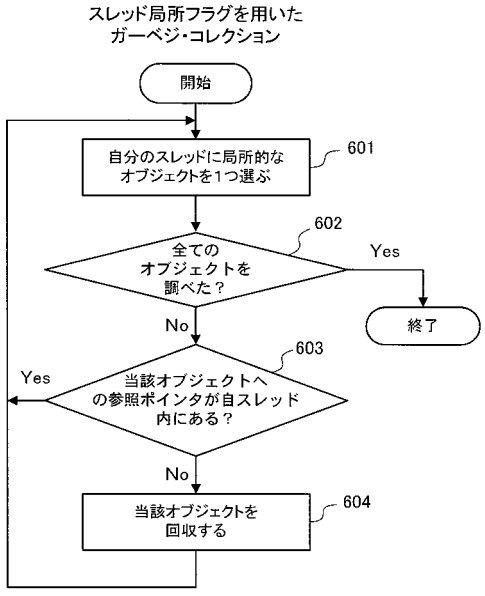
【図４】


オブジェクトのアンロックに関わる処理
開始
401
スレッド局所フラグがオンにセットされている？
Yes
No
402
オブジェクトをアンロックする
403
アンロックの省略に伴う処理
終了

【図５】


オブジェクトがスレッド局所性を解除される際の動作
開始
501
スレッド局所フラグがオンにセットされている？
No
Yes
502
スレッド局所フラグをクリア
503
省略中のロック処理の実行
504
オブジェクトの参照ポインタを代入
終了

【図６】


スレッド局所フラグを用いたガーベジ・コレクション
開始
自分のスレッドに局所的なオブジェクトを1つ選ぶ
601
602
全てのオブジェクトを調べた？
Yes
終了
No
603
Yes
当該オブジェクトへの参照ポインタが自スレッド内にある？
No
当該オブジェクトを回収する
604

フロントページの続き

(72)発明者　小野寺　民也
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内

審査官　間野　裕一

(56)参考文献　特開平０８－００６８０５（ＪＰ，Ａ）
特開平１１－２１２８０８（ＪＰ，Ａ）
鎌田十三郎他，超並列計算機上の高効率な大域的ガーベジコレクション，情報処理学会研究報告，社団法人情報処理学会，１９９３年　8月２０日，第９３巻，第７３号，第１２１－１２８頁，９３－ＰＲＧ－１３－１６

(58)調査した分野(Int.Cl.[7]，ＤＢ名)
G06F　9/46- 9/54
G06F 12/00
G06F 15/16-15/177