(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4963018号
(P4963018)
(45)発行日　平成24年6月27日(2012.6.27)　(24)登録日　平成24年4月6日(2012.4.6)

(51)Int.Cl.　F I
G06F 9/48 (2006.01)　G06F 9/46 452C

請求項の数 10 (全 15 頁)

(21)出願番号　特願2005-235582(P2005-235582)
(22)出願日　平成17年8月15日(2005.8.15)
(65)公開番号　特開2007-52511(P2007-52511A)
(43)公開日　平成19年3月1日(2007.3.1)
審査請求日　平成20年7月22日(2008.7.22)

(73)特許権者　310021766
株式会社ソニー・コンピュータエンタテインメント
東京都港区港南１丁目７番１号
(74)代理人　100105924
弁理士　森下　賢樹
(74)代理人　100109047
弁理士　村田　雄祐
(74)代理人　100109081
弁理士　三木　友由
(74)代理人　100134256
弁理士　青木　武司

最終頁に続く

(54)【発明の名称】スケジューリング方法およびスケジューリング装置

(57)【特許請求の範囲】
【請求項１】
マルチプロセッサシステムに含まれる複数のプロセッサのいずれにおいても実行可能な実行対象となる各実行単位に、当該実行単位とは分離して保存される識別子を付与し、
各実行単位のそれぞれが実行可能な状態にあるか、あるいはあるプロセッサで実行中であって別のプロセッサからは実行不可能な状態であるかを前記実行単位の識別子に対応づけて示す実行可否情報と、各実行単位のうちの、直近に実行された実行単位の識別子を示す直近実行情報とを含む実行単位情報を保持し、
実行単位情報に基づいて、直近に実行された実行単位の識別子以外の識別子を優先的に選出する制約の下でいずれかの実行可能な実行単位の識別子を、プロセッサによって実行される実行単位の識別子として選出するとともに、実行単位情報を更新することを特徴とするスケジューリング方法。
【請求項２】
前記実行単位情報が、各プロセッサがアクセス可能なメモリに保持され、
前記選出および更新は、選出された識別子に対応する実行単位を実行するプロセッサ自身によって行われることを特徴とする請求項１記載のスケジューリング方法。
【請求項３】
実行可否情報は、各実行単位について識別子として１ビットを割り当てたビット列として保持され、
前記選出および更新は、不可分操作によって行われることを特徴とする請求項１または

２記載のスケジューリング方法。
【請求項４】
直近に実行された実行単位に対応するビットが末尾になるようにビット列をローテートし、
ローテートされたビット列の先頭から順に実行可能な実行単位のビットを検索することによって前記選出を行うことを特徴とする請求項３記載のスケジューリング方法。
【請求項５】
マルチプロセッサシステムに含まれる複数のプロセッサのいずれにおいても実行可能な実行対象となる各実行単位のそれぞれが実行可能な状態にあるか、あるいはあるプロセッサで実行中であって別のプロセッサからは実行不可能な状態であるかを、それぞれの実行単位に対して付与された識別子に対応づけて示す実行可否情報と、各実行単位のうちの、直近に実行された実行単位の識別子を示す直近実行情報とを含む実行単位情報を、前記各実行単位とは分離して保持する実行単位情報保持部と、
実行単位情報に基づいて、直近に実行された実行単位の識別子以外の識別子を優先的に選出する制約の下でいずれかの実行可能な実行単位の識別子を、プロセッサによって実行される実行単位の識別子として選出する実行単位選出部と、
前記選出に伴い、実行単位情報を更新する実行単位情報更新部とを備えることを特徴とするスケジューリング装置。
【請求項６】
実行単位情報保持部は、各プロセッサがアクセス可能なメモリに実行単位情報を保持し、
実行単位選出部および実行単位情報更新部は、選出された識別子に対応する実行単位を実行するプロセッサ自身により構成されることを特徴とする請求項５記載のスケジューリング装置。
【請求項７】
実行単位情報保持部は、実行可否情報を、各実行単位について識別子として１ビットを割り当てたビット列として保持し、
実行単位選出部および実行単位情報更新部は、不可分操作によって前記選出および更新を行うことを特徴とする請求項５または６記載のスケジューリング装置。
【請求項８】
実行単位選出部は、直近に実行された実行単位に対応するビットが末尾になるようにビット列をローテートし、
ローテートされたビット列の先頭から順に実行可能な実行単位のビットを検索することによって前記選出を行うことを特徴とする請求項７記載のスケジューリング装置。
【請求項９】
マルチプロセッサシステムに含まれる複数のプロセッサのいずれにおいても実行可能な実行対象となる各実行単位のそれぞれが実行可能な状態にあるか、あるいはあるプロセッサで実行中であって別のプロセッサからは実行不可能な状態であるかを、それぞれの実行単位に対して付与された識別子に対応づけて示す実行可否情報と、各実行単位のうちの、直近に実行された実行単位の識別子を示す直近実行情報とを含む実行単位情報を、前記各実行単位とは分離して保持する機能と、
実行単位情報に基づいて、直近に実行された実行単位の識別子以外の識別子を優先的に選出する制約の下でいずれかの実行可能な実行単位の識別子を、プロセッサによって実行される実行単位の識別子として選出するとともに、実行単位情報を更新する機能とをコンピュータに実行せしめることを特徴とするプログラム。
【請求項１０】
プログラムを記憶した記憶媒体であって、
前記プログラムは、
マルチプロセッサシステムに含まれる複数のプロセッサのいずれにおいても実行可能な実行対象となる各実行単位のそれぞれが実行可能な状態にあるか、あるいはあるプロセッ

サで実行中であって別のプロセッサからは実行不可能な状態であるかを、それぞれの実行単位に対して付与された識別子に対応づけて示す実行可否情報と、各実行単位のうちの、直近に実行された実行単位の識別子を示す直近実行情報とを含む実行単位情報を、前記各実行単位とは分離して保持する機能と、

実行単位情報に基づいて、直近に実行された実行単位の識別子以外の識別子を優先的に選出する制約の下でいずれかの実行可能な実行単位の識別子を、プロセッサによって実行される実行単位の識別子として選出するとともに、実行単位情報を更新する機能とをコンピュータに実行せしめることを特徴とする記憶媒体。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、マルチプロセッサシステムにおける並列処理の実行単位のスケジューリング方法および装置に関する。

【背景技術】

【０００２】

最近のマルチタスクをサポートするオペレーティングシステムは、複数のプロセスを同時に実行することができるマルチタスク環境を実現するとともに、これらのプロセスがプロセス内部で複数のスレッドを生成して並行処理を行うことができるマルチスレッド技術を搭載している。プロセスは実行時に固有のリソースやアドレス空間が割り当てられ、他のプロセスの領域にアクセスすることができない。これに対してスレッドは、プロセス内部で生成される実行単位であり、各スレッドはプロセス内の領域に互いに自由にアクセスすることができる。スレッドは、オペレーティングシステムがＣＰＵ（Ｃｅｎｔｒａｌ Ｐｒｏｃｅｓｓｉｎｇ Ｕｎｉｔ）の実行時間を割り当てる基本的な実行単位となる。本明細書において、スレッドの割り当てをスケジューリングという。

【０００３】

１つのシステム内に複数のプロセッサを搭載したマルチプロセッサシステムでは、並列に、あるいは協調して処理を実行して処理全体の高速化を図ることができる。通常、マルチプロセッサシステムにおいて、共有メモリにタスクキューと呼ばれる待ち行列に実行可能なスレッドが保持される。これらのスレッドはいずれかのプロセッサに割り当てられて、実行される。スレッドを割り当てる方法によって、プロセスの実行速度やメモリ消費量などの性能が変わってくるため、マルチプロセッサシステムでは、シングルプロセッサシステムにおけるスケジューリングと異なった工夫が必要となる。

【０００４】

マルチプロセッサシステムにおけるスケジューリングについて、たとえば、スレッドを管理する役割を担う管理ユニットが各プロセッサと通信し、それぞれのプロセッサに割り当てるスレッドをスケジューリングする方法が考えられる。

管理ユニットによってスケジューリングを行うこの方法では、管理ユニットとプロセッサ間の通信において、メッセージ遅延が起きることが多く、スレッドの実行を遅延させてしまうという問題がある。この問題を解決するために、各プロセッサによって自己支配的にスケジューリングする方法が考えられる。この方法では各プロセッサ上でおのおのスケジューラを実行し、共有メモリにあるタスクキューにアクセスして、実行するスレッドを選択する。

【０００５】

この方法では、各プロセッサがスケジューラを実行している間、すなわちスケジューリング中において、共有メモリ上のタスクキューが他のプロセッサによって修正されることを防ぐために、スケジューラがタスクキューをロックする。この間、割り込みを禁止するか否かはシステムの設計者にとってジレンマである。

【０００６】

スケジューリング中に割り込みを禁止しない場合には、スケジューラがタスクキューをロックしている最中に、割り込みを受け付けてしまう。この場合、割り込み処理が終了し

、ロックが開放されるまで、スケジューリングが実行されない結果になってしまい、システムの効率低下につながる。

【０００７】

一方、スケジューリング中に割り込みを禁止すると、システムの割り込み応答性能が低下してしまう問題がある。

【０００８】

これらの問題はこれまで述べたマルチプロセッサ上で動作するカーネルスケジューラだけの問題でなく、マルチプロセッサ・マルチスレッド環境でのユーザレベルスケジューラ（すなわち、各プロセッサで動作するスレッド上でスケジューラを実行することで、ユーザレベルにおいてマルチスレッドを実現する方式）の場合も同様である。

特に、ユーザレベルスケジューラを実行しているカーネルスレッドは各プロセッサ上のカーネルレベルスケジューラによってスケジューリングされ、他のカーネルスレッドによってプリエンプト（横取り）される可能性があるため、タスクキューのロックはより深刻な問題を引き起しかねない。この場合、横取りされた際にそのカーネルスレッドで動作するユーザレベルスケジューラがロックを取得していた場合には、他のプロセッサのスレッド上で動作するユーザレベルスケジューラがロックが解除されるまでスケジューリングすることができなくなる。

【発明の開示】

【発明が解決しようとする課題】

【０００９】

本発明は上記事情に鑑みてなされたものであり、その目的は、マルチプロセッサシステムにおける並列処理の実行単位をプロセッサに割り当てる順序を制御して処理効率を向上させることができるスケジューリング技術を提供することにある。

【課題を解決するための手段】

【００１０】

本発明にかかる態様は、マルチプロセッサシステムにおいて、プロセッサの実行対象となる実行単位をスケジューリングする方法に関する。このスケジューリング方法は、マルチプロセッサシステムに含まれる複数のプロセッサの実行対象となる各実行単位に識別子を付与し、各実行単位のそれぞれが実行可能な状態にあるか否かを実行単位の識別子に対応づけて示す実行可否情報と、各実行単位のうちの、直近に実行された実行単位の識別子を示す直近実行情報とを含む実行単位情報を保持する。そして、実行単位情報に基づいて、直近に実行された実行単位の識別子以外の識別子を優先的に選出する制約の下でいずれかの実行可能な実行単位の識別子を、プロセッサによって実行される実行単位の識別子として選出するとともに、実行単位情報を更新する。

【００１１】

この態様では、実行可否情報と直近実行情報を含む実行単位情報が保持されるとともに、次に実行する実行単位の選出に伴って更新される。実行単位の選出は、実行単位情報を参照してなされる。そのため、実行単位の選出にあたって、実行単位情報を参照すればよく、実行単位の実体を記憶したたとえばタスクキューをロックする必要がない。そのため、スケジューリング中に割り込み処理を受け付けるか否かのジレンマが解消される。

【００１２】

また、直近実行情報が保持され、実行単位の識別子を選出する際に、直近実行情報により示される識別子以外の識別子を優先的に選出するとともに、この選出に伴って直近実行情報を更新するようにしているので、直近に実行された実行単位がまた実行可能になっても、他の実行可能な実行単位が優先的に実行されるので、実行単位のスケジューリングにおいて重要な公平性を保つことができる。

【００１３】

本発明のこの態様は、管理ユニットによってスケジューリングするシステムに適用してもよく、各プロセッサ自身によってスケジューリングを行うシステムに適用してもよい。

また、本発明のこの態様は、各プロセッサ上で直接動作するスケジューラだけではなく

、各プロセッサ上のスケジューラが提供するスレッド上で動作するユーザレベルスケジューリングを用いるシステムに適用してもよい。

【００１４】

ここで、実行可否情報は、各実行単位について識別子として１ビットを割り当てたビット列として保持され、ビットの選出および実行単位情報の更新を不可分操作、すなわちアトミック操作によって行うようにしてもよい。

【００１５】

「不可分操作」は、これ以上分割できない最小単位の操作を意味し、マルチプロセッサシステムにおいて、他のプロセッサが行う操作との相互作用なしに実行されることが保証される操作である。

【００１６】

ビットの選出に当たっては、直近に実行された実行単位に対応するビットが末尾になるようにビット列をローテートし、ローテートされたビット列の先頭から順に実行可能な実行単位のビットを検索することによって行うようにしてもよい。

【００１７】

なお、以上の構成要素の任意の組合せ、本発明をシステム、プログラム、プログラムを記憶した記憶媒体として表現したものも、本発明の態様としては有効である。

【発明の効果】

【００１８】

本発明は、マルチプロセッサシステムにおける並列処理の実行単位をスケジューリングすることにおいて有利である。

【発明を実施するための最良の形態】

【００１９】

マルチプロセッサシステムにおいて、各プロセッサの実行単位たとえばスレッドのスケジューリング中に、スレッドを記憶したタスクーキューがほかのプロセッサにより修正されることを防ぐために、タスクキューをロックする必要があった。

【００２０】

このロックによって、スケジューリング中に、割り込みを禁止するとシステムの割り込み応答性能が低下し、割り込みを禁止しないと、割り込み処理が終わり、ロックが解除されるまでスケジューリングができないというジレンマがあった。

【００２１】

上述した問題を解決するために、本発明者は、下記の技術を提案する。

【００２２】

各スレッドに識別子を付与し、これらのスレッドのそれぞれが実行可能な状態にあるか否かをスレッドの識別子に対応づけて示す実行可否情報と、各スレッドのうちの、直近に実行されたスレッドの識別子を示す直近実行情報とを含むスレッド情報を保持する。そして、スレッド情報に基づいて、直近に実行されたスレッドの識別子以外の識別子を優先的に選出するという制約の下で、いずれかの実行可能なスレッドの識別子を、プロセッサに割り当てるスレッドの識別子として選出する。

【００２３】

ここで、スレッドの実体（以下スレッド実体という）を、プロセッサが選出された識別子に基づいて取得することができるいかなる方法で保持してもよい。たとえばスレッド実体を識別子と対応づけて保存するようにしてもよいし、各スレッド実体をメモリのそれぞれの所定の領域に保存するとともに、スレッド実体が保存された領域の開始アドレスとその識別子とを対応づけてメモリに保存するようにしてもよい。

【００２４】

この技術は、スレッド実体と、いずれのスレッドを実行するかの選出を行うために必要なスレッド情報とに分けてメモリに保存する。そのために、スケジューリングする際に当たって、スレッド実体を記憶した領域をロックする必要がなく、スケジューリング中に割り込みを禁止するか否かのジレンマを解消することができる。

【００２５】
この技術は、ユーザレベルスケジューラを実行するシステムにおいても、上記ジレンマを解消することができるとともに、ユーザレベルスケジューラゆえの問題も解決することができる。
【００２６】
たとえば、マルチプロセッサシステム、特にＯＳ（オペレーティングシステム）機能を実行できるプロセッサが限られるような非対称のマルチプロセッサシステムにおいては、各プロセッサで動作するＯＳが提供するカーネルスレッド上にユーザレベルスレッドを作成して、自己支配的にスケジューリングする方法、すなわちユーザレベルスケジューラを用いる方法は、マルチプロセッサシステムの処理効率を向上させる有効な方法であると考えられる。この方法では、ユーザレベルスケジューラを用いることに起因するスケジューリングの破綻という問題がある。
【００２７】
たとえば、ユーザレベルスケジューラを実行しているスレッドは、各プロセッサの上のカーネルスケジューラによってスケジューリングされ、ほかのカーネルスレッドによってプリエンプトされる可能性がある。そのため、プリエンプトされた際にそのスレッドで動作するユーザレベルスケジューラがロックを取得していた場合には、ほかのプロセッサのスレッド上で動作するユーザレベルスケジューラがロックが解除されるまで、スケジューリングができなくなり、プロセッサの処理効率を下げてしまう。さらに、ロックが解除されるまでの時間は、状況に依存してしまい、見積もることが困難になるため、システムの不安定を引き起こす原因にもなりえる。
【００２８】
ここで、マルチプロセッサシステムにおいて各プロセッサが自律的に動作するカーネルスケジューラを実行する従来のシステムについて考える。タスクキューにスレッド１、スレッド２、スレッド３が入っており、複数のプロセッサのうちのプロセッサＡは、タスクキューにあるスレッドを実行することができる状態になった場合を想定する。
【００２９】
プロセッサＡは、実行するスレッドを選出するために、タスクキューを一旦自分のローカルメモリにコピーし、タスクキューからたとばスレッド１を選出した後に、タスクキューからスレッド１を削除する更新処理を行って、更新されたタスクキューをメインメモリに書き戻す。タスクキューのコピーが開始するときから、タスクキューの書戻しが終了するまでの間、共有メモリ上のタスクキューが他のプロセッサによって修正されることを防ぐために、タスクキューをロックする。タスクキューがロックされている間、他のプロセッサは、タスクキューに入ったスレッドを実行できる状態になったとしても、タスクキューを利用することができないため、ロックの期間が長いほどシステムの処理効率が低下する。
【００３０】
また、プロセッサＡで動作するスケジューラは前述したように割り込みによってプリエンプトされる可能性がある。プロセッサＡはタスクキューがロックされた状態でプリエンプトされると、ほかのプロセッサは、プロセッサＡが割り込み処理から復帰し、ロックが解除されるまでスケジューリングを行うことができず、スレッド２、３を実行することが不可能となる。これでは、システムは処理効率が低下するとともに、不安定にもなりかねない。
なお、上記説明は、プロセッサＡ、・・・を、プロセッサＡ、・・・上で動作するユーザレベルスケジューラを実行するスレッド（たとえばスレッドａ、・・・）に置き換えれば、ユーザレベルスケジューラを実行するシステムにも適用することができる。具体的には、ユーザレベルスケジューラを実行するシステムにおいて、たとえばプロセッサＡ上で動作するユーザレベルスケジューラを実行するスレッドａも、割り込みによってプリエンプトされる可能性がある。スレッドａは、タスクキューがロックされた状態でプリエンプトされると、ほかのスレッドは、スレッドａが割り込み処理から復帰し、ロックが解除さ

れるまでスケジューリングを行うことができない。すなわち、ユーザレベルスケジューラを用いるシステムにおいても同じように、タスクキューのロック起因して、システムは処理効率が低下し、不安定になりかねない問題がある。

【００３１】

また、マルチプロセッサシステムにおいて、各々のプロセッサがそれぞれの処理ユニットに含まれる形で存在する。これらの処理ユニットはＰＰＵ（Ｐｏｗｅｒ　Ｐｒｏｃｅｓｓｉｎｇ　Ｕｎｉｔ）とＳＰＵ（Ｓｙｎｅｒｇｉｓｔｉｃ　Ｐｒｏｃｅｓｓｉｎｇ　Ｕｎｉｔ）とに分けることができる。ＳＰＵのすべてが同一のアーキテクチャを用いて実現されてもよく、それぞれ異なる構成を有してもよい。ＰＰＵは、ＳＰＵに対してローカルに、たとえばＳＰＵと同一のチップ、同一のパッケージ、同一の回路基板、同一の製品に位置してもよいし、ＳＰＵに対してリモートに、たとえばバスやインターネットなどの通信ネットワークを介して接続可能な異なる製品に位置してもよい。同様に、ＳＰＵは、互いにローカルにまたはリモートに位置してもよい。

【００３２】

スケジューリング禁止区間の存在と禁止時間の見積もりの困難さはマルチプロセッサシステムの処理効率を下げる要因になる。ユーザレベルスケジューラを実行するスレッドのすべてがＳＰＵのスレッド（以下ＳＰＵスレッドという）であれば、この問題を解決する方法としては、スレッドをグループ化してグループ単位でスケジューリングする方法が考えられる。ここで、図９に示すマルチプロセッサシステムを例にして説明する。

【００３３】

図９に示すマルチプロセッサシステムは、複数の処理ユニット１１０とメインメモリ１３０と有し、それらはメインバス１２０に接続されている。各処理ユニット１１０は、プロセッサ１１２、ローカルメモリ１１４、メモリ制御部１１６を有する。プロセッサ１１２は、ローカルメモリ１１４に対してデータを読み書きすることができる。メモリ制御部１１６は、他の処理ユニット１１０のプロセッサ１１２からローカルメモリ１１４のデータを参照するときのインターフェースを与えるとともに、メモリの同期、排他制御の機能を提供する。

【００３４】

ここで、メインメモリ１３０に設けられたタスクキューにＳＰＵスレッドのみが記憶されている場面について考える。この場合、タスクキュー内のスレッドが図１０に示すようにグルーピングされる。　図１０（ａ）は、３つのスレッドｔｈ１ａ、ｔｈ１ｂ、ｔｈ１ｃを含む第１スレッドグループを示す。図１０（ｂ）は、１つのスレッドｔｈ２ａを含む第２スレッドグループを示す。このようなスレッドが１つだけの場合もスレッドグループとして扱う。同様に、図１０（ｃ）は、２つのスレッドｔｈ３ａ、ｔｈ３ｂを含む第３スレッドグループを示す。図１０（ｄ）は、１つのスレッドｔｈ４ａを含む第４スレッドグループを示す。

【００３５】

これらのスレッドのスケジューリングは、同一スレッドグループに属するすべてのスレッドを同時にいずれかのプロセッサ１１２に割り当てることを条件として行われる。第１スレッドグループがプロセッサ１１２に割り当てられるときは、第１スレッドグループに属する３つのスレッドｔｈ１ａ、ｔｈ１ｂ、ｔｈ１ｃが同時にいずれかのプロセッサ１１２に割り当てることができる場合に限られる。３つのスレッドｔｈ１ａ、ｔｈ１ｂ、ｔｈ１ｃの１つまたは２つがプロセッサ１１２に割り当てられ、残りがメインメモリ１３０に退避しているという状況は作らない。

【００３６】

図１１は、スレッドグループ単位でスレッドがプロセッサ１１２に割り当てられる様子を説明する図である。同図は、プロセッサ総数４のマルチプロセッサシステムにおいて、図１０に示した４つのスレッドグループに属するスレッドのプロセッサ１１２への割り当て状態を示している。ある時刻において、第１スレッドグループに属する３つのスレッドｔｈ１ａ、ｔｈ１ｂ、ｔｈ１ｃは、それぞれ第１プロセッサ、第２プロセッサ、第３プロ

セッサに割り当てられ、第２スレッドグループに属する１つのスレッドｔｈ２ａは、第４プロセッサに割り当てられている。それ以外の第３スレッドグループに属する２つのスレッドｔｈ３ａ、ｔｈ３ｂ、および第４スレッドグループに属する１つのスレッドｔｈ４ａはメインメモリ１１２に退避されている。

【００３７】

このようなシステムによれば、1つのスレッドグループ内に所属する複数のスレッドを必ず同時にいずれかのプロセッサに割り当てるようにする。スレッドグループ内において、ロックを取ったスレッドだけがプリエンプトされることがないため、スケジューリングの禁止区間を限定することができる。

【００３８】

しかし、ＰＰＵのスレッド（以下ＰＰＵスレッドという）とＳＰＵスレッドが非同期にスケジューリングされる環境では、ＰＰＵスレッドとＳＰＵスレッドで同じタスクキューを共有した場合には下記の問題が生じうる。あるＰＰＵスレッドがタスクキューのロックを取ったまま他のＰＰＵスレッドにプリエンプトされると、ＳＰＵスレッドはこのＰＰＵスレッドが再実行されるまで待たざるを得ないため、スケジューリングの禁止区間を限定することができなくなってしまう。

【００３９】

本発明者が提案したスケジューリング技術は、スレッド情報とスレッド実体を分けて保存し、プロセッサは、いずれのスレッドを実行するかの選出を行う際、スレッド情報のみをロードすればよい。選出を終え、スレッド情報の更新（具体的には、直近に実行されたスレッドの識別子を、選出されたスレッドの識別子に変更する処理と、選出されたスレッドの識別子を該識別子の対応するスレッドが実行不可であることを示すように修正する処理）を終了すれば、更新されたスレッド情報をメインメモリにストアする。その後、プロセッサは、選出した識別子に対応するスレッド実体をコピーするが、コピーする間、他のプロセッサは、スレッド情報を利用することができる。こうすることによって、スレッド情報は１つのプロセッサに占有される時間が短いため、スレッド情報をロックしたとしても、システム全体の処理効率の低下を軽減させることができる。

【００４０】

さらに、本発明者は、実行可否情報を、各スレッドについて識別として１ビットを割り当てたビット列として保持することを提案する。こうすることによって、スレッドの選出、スレッド情報の更新などを、アトミック操作やアトミック命令を用いて行うことができ、ロック操作を伴わないすなわちロックレスのタスクキューを実現することができる。ロック操作がなければ、上述した、タスクキューのロックに起因するおのおのの問題も解消される。

【００４１】

図１は、本発明の実施形態となるマルチプロセッサシステム１００の構成を示す。マルチプロセッサシステム１００は、複数の処理ユニット１０とメインメモリ３０と有し、それらはメインバス２０に接続されている。各処理ユニット１０は、プロセッサ１２、ローカルメモリ１４、メモリ制御部１６を有する。プロセッサ１２は、ローカルメモリ１４に対してデータを読み書きすることができる。メモリ制御部１６は、他の処理ユニット１０のプロセッサ１２からローカルメモリ１４のデータを参照するときのインターフェースを与えるとともに、メモリの同期、排他制御の機能を提供する。

【００４２】

処理ユニット１０のうちのいずれか１つを、スレッドのスケジューリングに関して他の処理ユニットに対するサービスユニットの役割を担う。サービスユニットの役割としては、たとえばメインメモリ３０の割当てや、メインメモリ３０内のスレッドに関する最初の記憶に関わることなどである。メインメモリ３０の割当ては、たとえばスレッド情報に割り当てられる領域、領域の容量や、スレッドの実体に割り当てられるべきメモリ容量を決定することなどとすることができる。

【００４３】

なお、このサービスユニットは、いずれの処理ユニット１０によって担当されてもよい。

【００４４】

ある時刻において、各プロセッサ１２には１つのスレッドが動作し、マルチプロセッサシステム１００全体で並列に複数のスレッドが実行される。各プロセッサ１２において動作しているスレッドは、処理ユニット１０内のローカルメモリ１４やメモリ制御部１６内のレジスタなどのすべての資源を占有して使用することができる。

【００４５】

この状態において、処理待ちしているスレッドは、そのコンテキストがメインメモリ３０に保持される。スレッドのコンテキストは、そのスレッドが実行された処理ユニット１０内で占有するすべての資源の状態であり、スレッドがプロセッサ１２において動作しているときに各種レジスタに保持されている値の集合、ローカルメモリ１４に保持されたデータ、メモリ制御部１６の各種レジスタの内部状態などである。スレッドがプロセッサ１２上で動作してないときは、そのスレッドのコンテキストをメインメモリ３０にコピーしておき、再度プロセッサ１２によって処理可能となったときに、そのコンテキストをメインメモリ３０から読み込んで、処理を継続することができる。スレッドのコンテキストは、スレッド実体に該当する。

【００４６】

図２は、メインメモリ３０により記憶された、スレッドに関する情報を示す。これらの情報は、スレッド情報４０、スレッドアドレス情報５０と、スレッド実体６０であり、マルチプロセッサシステム１００におけるタスクキューの役割を担う。なお、これらの情報が記憶される領域は、サービスユニットによって割り当てられ、ほかの処理ユニット１０に通知される。

【００４７】

スレッド情報４０は、実行可否情報と直近実行情報とを含む。実行可否情報は、各スレッドに対してそれぞれ付与された識別子を、それぞれのスレッドが実行可能な状態にあるか否かを示す情報に対応づけたものである。図３は、スレッド情報４０の詳細を示す。

【００４８】

ビット列（ｉｓＳｃｈｅｄｕｌａｂｌｅ）は、実行可否情報であり、それに含まれる１ビットが１つのスレッドに対応し、ビット番号はスレッドの番号に対応する。なお、マルチプロセッサシステム１００は、ビット番号そのものをスレッドの番号として用いて処理の簡潔化を図る。

【００４９】

ビット列ｉｓＳｃｈｅｄｕｌａｂｌｅの各ビットの値はそのビットに対応するスレッドが実行可能状態にあるか否かを示している。ここで、ビットの値の「１」は実行可能であることを示し、「０」は実行不可であることを示すようにされている。ビット列ｉｓＳｃｈｅｄｕｌａｂｌｅに含まれるビットの数は、実行可否情報の保持用に割り当てられた領域の容量に相当し、ここでは１２８ビットとする。

【００５０】

ｌａｓｔＳｃｈｅｄｕｌｅｄは、ビット列ｉｓＳｃｈｅｄｕｌａｂｌｅに含まれる各ビットのうち、直近に実行されたビットの番号を示す整数値を取る変数である。

【００５１】

ｉｓＳｃｈｅｄｕｌａｂｌｅとｌａｓｔＳｃｈｅｄｕｌｅｄによって、図４の例のように、実行可能なスレッドの番号、直近に実行されたスレッドの番号が示される。図４の例では、ビット列ｉｓＳｃｈｅｄｕｌａｂｌｅのうち、値が「１」であるビット（図中矢印Ｂ、Ｃ、Ｄが示すビット）が３つあり、この３つのビット番号に対応するスレッドは実行可能スレッドである。値が「０」であるほかのビットについて、それらに対応するスレッドは、実行不可である。また、ｌａｓｔＳｃｈｅｄｕｌｅｄの値に等しいビット番号（図中矢印Ａが示すビットの番号）に対応するスレッドは、直近に実行されたスレッドであり、その状態は実行不可である。

【００５２】
スレッドアドレス情報５０は、各スレッドの番号ここではビット列ｉｓＳｃｈｅｄｕｌａｂｌｅに含まれるビットの番号と、その番号に対応するスレッドの実体が保存された領域の開始アドレスとを対応づけたものである。どの番号のスレッド実体を、どのアドレスの領域に保存するかについては、サービスユニットによって決められる。
【００５３】
処理待ちのスレッドがない状態において、スレッド情報４０のビット列ｉｓＳｃｈｅｄｕｌａｂｌｅの各ビットの値が「０」である。
【００５４】
処理ユニット１０の処理が進み、スレッドが生成される。生成されたスレッドは、またいずれかの処理ユニット１０によって実行される。処理待ちのスレッドが生じた際に、タスクキューが利用される。ここで、例として図４に示すタスクキューの状態を起点にして、処理ユニット１０が次に実行するスレッドを選出する処理、およびこの選出に伴うスレッド情報の更新処理を図５のフローチャートを用いて説明する。
【００５５】
図４に示す状態では、マルチプロセッサシステム１００の各処理ユニット１０は、それぞれ処理中のスレッドがあり、処理待ちしている実行可能なスレッドは３つあり、この３つのスレッドのそれぞれの実体は、メインメモリ３０に記憶されている。
【００５６】
この状態において、ある処理ユニット１０において、処理中のスレッドの処理が終了すると、この処理ユニット１０のプロセッサ１２は、次に実行するスレッドを選出するために、スレッド情報４０に含まれるビット列ｉｓＳｃｈｅｄｕｌａｂｌｅとｌａｓｔＳｃｈｅｄｕｌｅｄを、ローカルメモリ１４にロードする（Ｓ１０）。マルチプロセッサシステム１００において、処理ユニット１０は、スレッド情報４０に関わる処理をアトミックコマンドを用いて行い、ここでは、スレッド情報４０をロードするためのコマンドとして、たとえば「ｌｗａｒｘ」または「ｇｅｔｌｌａｒ」を用いる。なお、図６のＡ欄に示すビット列ｉｓＳｃｈｅｄｕｌａｂｌｅは、図４に示すビット列ｉｓＳｃｈｅｄｕｌａｂｌｅである。
【００５７】
プロセッサ１２は、ロードしたビット列ｉｓＳｃｈｅｄｕｌａｂｌｅ（図６のＡ欄に示すビット列）を、ｌａｓｔＳｃｈｅｄｕｌｅｄの値に等しい番号のビット（図６の矢印Ａ１が示すビット）が末尾になるように、矢印Ｌが示す方向、すなわち左方向にローテートする（Ｓ１４）。これによって、図６のＡ欄のビット列ｉｓＳｃｈｅｄｕｌａｂｌｅは、同図のＢ欄のビット列ｉｓＳｃｈｅｄｕｌａｂｌｅになる。図示のように、（ｌａｓｔＳｃｈｅｄｕｌｅｄ＋１）の番号のビット（矢印Ｂ１が示すビット）が、ビット列ｉｓＳｃｈｅｄｕｌａｂｌｅの先頭に位置し、ｌａｓｔＳｃｈｅｄｕｌｅｄの番号のビット（矢印Ｂ３が示すビット）がビット列ｉｓＳｃｈｅｄｕｌａｂｌｅの末尾になる。
【００５８】
プロセッサ１２は、続いてローテートされたビット列ｉｓＳｃｈｅｄｕｌａｂｌｅに対して先頭から順に、値が「１」であるビットの検索を行い、最も先に検出したビット（矢印Ｂ２が示すビット）の番号を、次に実行するスレッドの番号として得る（Ｓ１８）。ビットの検索に用いるコマンドとして、たとえばビット列ｉｓＳｃｈｅｄｕｌａｂｌｅの先頭から連続した、「０」の値を有するビットの数を数える「Ｃｏｕｎｔ　Ｌｅａｄｉｎｇ　Ｚｅｒｏ」を用いることができる。「Ｃｏｕｎｔ　Ｌｅａｄｉｎｇ　Ｚｅｒｏ」により得られた値（図６に示すＢ欄のビット列ｉｓＳｃｈｅｄｕｌａｂｌｅの例では４）に（ｌａｓｔＳｃｈｅｄｕｌｅｄ＋１）を加算して得た値は、次に実行するスレッドの番号として選出される。
【００５９】
そして、プロセッサ１２は、選出した番号のビットの値を「０」にセットするとともに、ｌａｓｔＳｃｈｅｄｕｌｅｄをこの番号にセットして、スレッド情報４０の更新を行う

（Ｓ２０）。更新されたスレッド情報４０は、メインメモリ３０にストアされる（Ｓ２４）。ここで、ビットの値の更新は、図６のＣ欄に示す１２８ビットのビット列を用いて行う。Ｃ欄のビット列は、１２８ビットを有し、ステップＳ１０において選択されたした番号と同じ番号のビット（矢印Ｃ１が示すビット）のみが「１」の値を有する。プロセッサ１２は、図６のＢ欄のビット列ｉｓＳｃｈｅｄｕｌａｂｌｅと、Ｃ欄のビット列とを、「ＡｔｏｍｉｃＡｎｄｃ」コマンドで演算することによってＢ欄のビット列ｉｓＳｃｈｅｄｕｌａｂｌｅを更新する。また、更新されたビット列ｉｓＳｃｈｅｄｕｌａｂｌｅのストアに用いるコマンドは、たとえば「ｓｔｗｃｘ」または「ｐｕｔｌｌｃ」とすることができる。

【００６０】

図６のＤ欄は、更新されたスレッド情報４０を示す。ここで、次に実行するスレッドとして選出されたビット（矢印Ｄ１が示すビット）は、直近に実行されたスレッドを示すビットとなり、その値が「１」から「０」になっている。

【００６１】

その後、プロセッサ１２は、スレッドアドレス情報５０を参照して、選出した番号のスレッドに対応するスレッド実体６０の開始アドレスを取得するとともに、この開始アドレスにより示される領域からスレッド実体６０をローカルメモリ１４にロードして処理する。

【００６２】

次に図７のフローチャートを用いて、タスクキューに実行可能となったスレッドを追加する処理について説明する。

【００６３】

処理ユニット１０のプロセッサ１２は、新たに実行可能となったスレッドをタスクキューに追加するために、スレッド情報４０に含まれるビット列ｉｓＳｃｈｅｄｕｌａｂｌｅをローカルメモリ１４にロードする（Ｓ５０）。このビット列ｉｓＳｃｈｅｄｕｌａｂｌｅはたとえば図８のＡ欄に示すビット列ｉｓＳｃｈｅｄｕｌａｂｌｅである。プロセッサ１２は、Ａ欄のビット列ｉｓＳｃｈｅｄｕｌａｂｌｅの各ビットのうちの、値が「０」であるいずれかのビット（たとえば矢印Ａ１が示すビット）の番号を、追加するスレッドのビット番号として選出する。そして、プロセッサ１２は、Ａ欄のビット列ｉｓＳｃｈｅｄｕｌａｂｌｅと、Ｂ欄に示すビット列とを「ＡｔｏｍｉｃＯｒ」コマンドで演算してＣ欄に示すビット列ｉｓＳｃｈｅｄｕｌａｂｌｅを得る（Ｓ５４）。図８のＢ欄に示すビット列は、１２８ビットを有し、プロセッサ１２により選出された番号と同じ番号のビット（矢印Ｂ１が示すビット）のみが「１」の値を有する。

【００６４】

続いて、プロセッサ１２は、ステップＳ５４により得られたビット列ｉｓＳｃｈｅｄｕｌａｂｌｅ（図８のＣ欄のビット列）をメインメモリ３０にストアして、スレッドを追加するためのスレッド情報４０の更新を終了する（Ｓ５８）。

【００６５】

プロセッサ１２は、この後、追加するスレッドの実体を、ステップＳ５４において選出したビット番号に対して割り当てられた領域にコピーして、スレッドを追加する処理を終了する。

【００６６】

このように、図１に示すマルチプロセッサシステム１００によれば、各プロセッサ１２自身によって、タスクキューからスレッドを選出する処理、タスクキューを更新する処理を行っているので、システム全体の処理効率を向上させることができる。

【００６７】

また、タスクキューを構成する際に、スレッド情報とスレッド実体とに分けて保存し、スレッドの選出、更新は、スレッド情報のみを用いて行うことができるようにしたので、より効率の良いマルチプロセッサシステムを実現している。

【００６８】

さらに、スレッド情報としてビット列ｉｓＳｃｈｅｄｕｌａｂｌｅと、ｌａｓｔＳｃｈｅｄｕｌｅｄとの２つの変数を用いることによって、スレッドの選出と更新を、アトミックコマンドで行うことを可能とした。これによって、ロックレスのタスクキューを実現している。

【００６９】

ロックレスのタスクキューの実現により、スケジューリング中に割り込みを受け付けるか否かのジレンマを解消することができる。

ここで、図１に示すマルチプロセッサシステム１００は、カーネルスケジューラのみをを用いるマルチプロセッサシステムであるが、マルチプロセッサシステム１００に用いられたスケジューリング方法は、各プロセッサ上で動作するスレッド上で実現されたユーザレベルスケジューラを用いたシステムにも適用することができる。その場合、スケジューリング中に割り込みを受け付けるか否かのジレンマを解消することができるとともに、前述した、ユーザレベルスケジューラを用いることに起因する問題も解消することができる。

【００７０】

以上、本発明を実施の形態をもとに説明した。実施の形態は例示であり、それらの各構成要素や各処理プロセスの組合せにいろいろな変形例が可能なこと、またそうした変形例も本発明の範囲にあることは当業者に理解されるところである。

【００７１】

また、図１に示す実施形態は、各プロセッサが自律的にスケジューリングを行うシステムであるが、本発明のスケジューリング方法は、このようなシステムに限らず、たとえばひとつの管理ユニットによってスケジューリングするシステムにも適用することができる。

【００７２】

また、本発明を適用したデバイスも、本発明の範囲にある。これらのデバイスには、パーソナルコンピュータやサーバなどに限らず、携帯電話、ゲーム機、モバイルコンピュータ、個人携帯情報機器（ＰＤＡ）、デジタルテレビなどが含まれる。

【図面の簡単な説明】

【００７３】

【図１】本発明にかかる実施形態のマルチプロセッサシステムを示す図である。

【図２】図１に示すマルチプロセッサシステムのタスクキューを示す図である。

【図３】図２に示すタスクキューに含まれるスレッド情報の構成を示す図である。

【図４】図３に示すスレッド情報の詳細を説明するための図である。

【図５】プロセッサによりスレッドを選出する処理を示すフローチャートである。

【図６】図５に示す処理に伴うスレッド情報の変化を示す図である。

【図７】プロセッサによりスレッドを追加する処理を示すフローチャートである。

【図８】図７に示す処理に伴うスレッド情報の変化を示す図である。

【図９】ユーザレベルスケジューラを用いるマルチプロセッサシステムの例を示す図である。

【図１０】図９に示すマルチプロセッサシステムにおけるスレッドのグルーピングを示す図である。

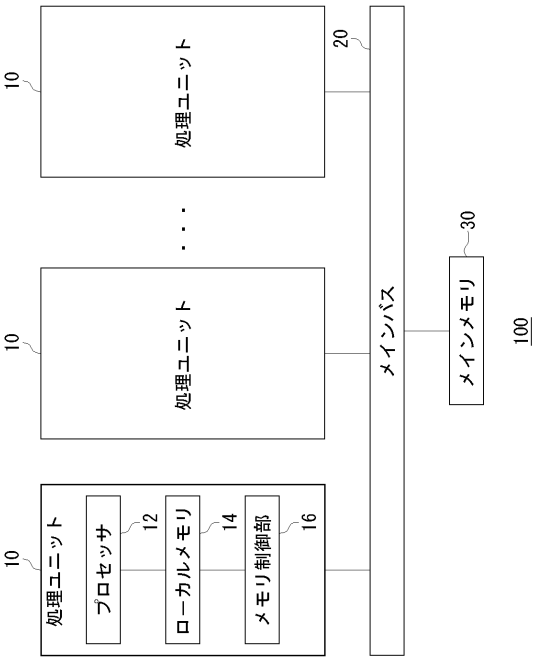
【図１１】図９に示すマルチプロセッサシステムにおけるスレッドのスケジューリングを示す図である。

【符号の説明】

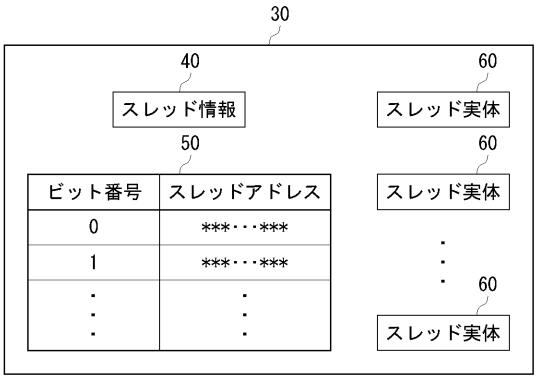
【００７４】

１０　処理ユニット、　１２　プロセッサ、　１４　ローカルメモリ、　１６　メモリ制御部、　２０　メインバス、　３０　メインメモリ、　４０　スレッド情報、　５０　スレッドアドレス情報、　６０　スレッド実体、　１００　マルチプロセッサシステム、　１１０　処理ユニット、　１１２　プロセッサ、　１１４　メモリ制御部、　１２０　メインバス、　１３０　メインメモリ。

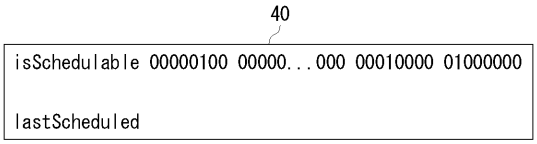
【図１】



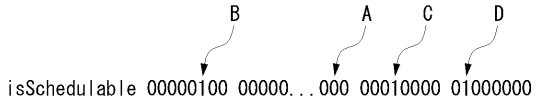
【図２】



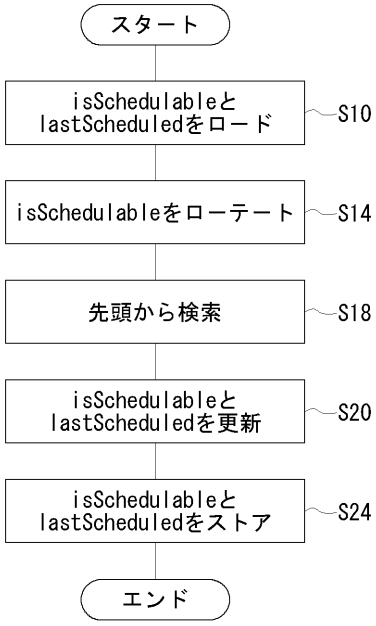
【図３】



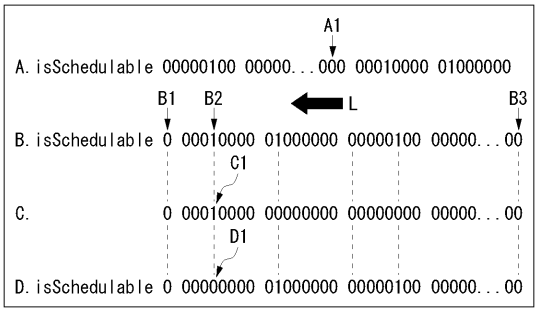
【図４】



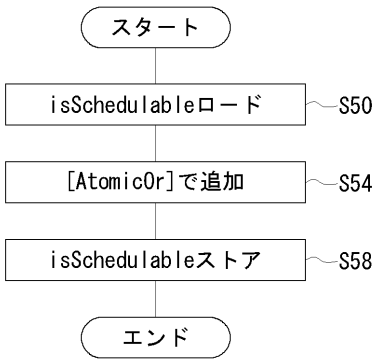
【図５】



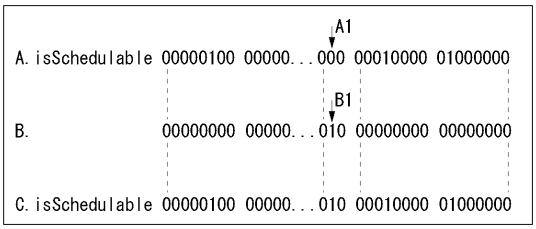
【図６】



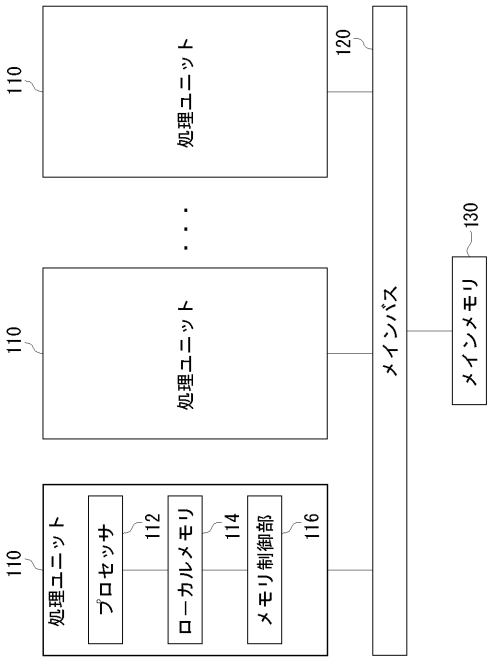
【図７】



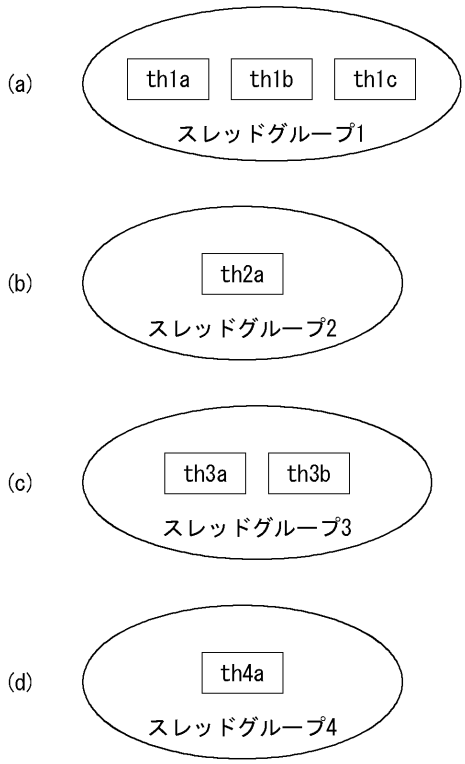
【図８】


A1
A. isSchedulable 00000100 00000...000 00010000 01000000
B1
B. 00000000 00000...010 00000000 00000000
C. isSchedulable 00000100 00000...010 00010000 01000000

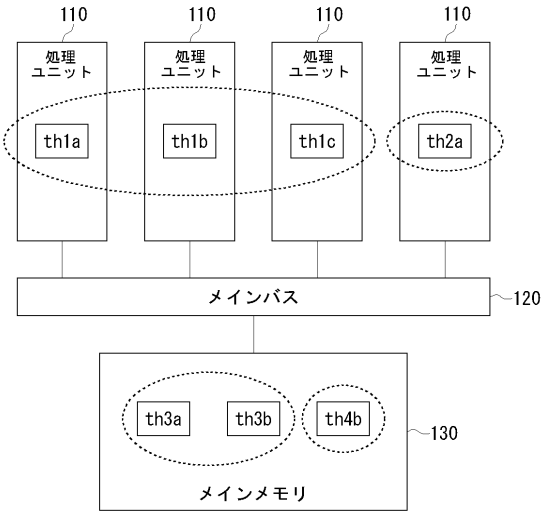
【図９】


110
処理ユニット
120
メインバス
130
メインメモリ
プロセッサ
112
ローカルメモリ
114
メモリ制御部
116

【図１０】


(a)
th1a
th1b
th1c
スレッドグループ1
(b)
th2a
スレッドグループ2
(c)
th3a
th3b
スレッドグループ3
(d)
th4a
スレッドグループ4

【図１１】


110
処理ユニット
th1a
th1b
th1c
th2a
メインバス
120
th3a
th3b
th4b
130
メインメモリ

フロントページの続き

(72)発明者　井上　敬介
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　村田　誠二
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内

審査官　鈴木　修治

(56)参考文献　特開平０２－２４２４３４（ＪＰ，Ａ）
特開平０４－０２４８２８（ＪＰ，Ａ）
特許第２８０４４７８（ＪＰ，Ｂ２）
特許第２５０５５２６（ＪＰ，Ｂ２）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４８