(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4124849号
(P4124849)

(45)発行日　平成20年7月23日(2008.7.23)　(24)登録日　平成20年5月16日(2008.5.16)

(51)Int.Cl.　　F I
G06F 15/17 (2006.01)　　G06F 15/17 635J
G06F 12/00 (2006.01)　　G06F 12/00 572A
G06F 12/08 (2006.01)　　G06F 12/08 531B

請求項の数 13　(全 27 頁)

(21)出願番号　特願平10-21033
(22)出願日　平成10年2月2日(1998.2.2)
(65)公開番号　特開平10-289214
(43)公開日　平成10年10月27日(1998.10.27)
審査請求日　平成17年1月24日(2005.1.24)
(31)優先権主張番号　08/794172
(32)優先日　平成9年2月3日(1997.2.3)
(33)優先権主張国　米国(US)

(73)特許権者　398038580
ヒューレット・パッカード・カンパニー
HEWLETT-PACKARD COMPANY
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000
(74)代理人　100059959
弁理士　中村　稔
(74)代理人　100067013
弁理士　大塚　文昭
(74)代理人　100065189
弁理士　宍戸　嘉一
(74)代理人　100096194
弁理士　竹内　英人

最終頁に続く

(54)【発明の名称】対称的マルチプロセッサのクラスタのための可変粒度型メモリ共用方法

(57)【特許請求の範囲】

【請求項１】

複数の対称的マルチ・プロセッサを含むコンピュータシステム(200)内の、対称的マルチプロセッサ(210)のメモリ(212)内に格納されたデータへのアクセスを共用するための、ソフトウェアにより実行される方法であって、

上記各対称的マルチ・プロセッサが、複数のプロセッサ(211)、アドレスを有するメモリ、及び、入力／出力インターフェース(214)、を含み、それらが互いにバス(213)によって接続され、

上記入力／出力インターフェースが、ネットワーク(220)によって、対称的マルチ・プロセッサを互いに接続し、

上記方法は、当該対称的マルチプロセッサのうちの１つによって実行される以下の、

メモリの１組のアドレスを、共用データ(550)を記憶するための仮想共用アドレスとして指定し；

上記仮想共用アドレスの一部分を、共用データ構造体(551)を記憶するために、いずれかのプロセッサで実行されるプログラム(310)によりアクセスできる１つ以上のブロックとして割り当て、特定の割り当てられるブロックのサイズは、共用データ構造体のサイズと共に変化し、各ブロックは、整数本のライン(552)を含み、そして各ラインは、所定のバイト数の共用データを含み；

複数の共用状態エントリー(545)を含む共用状態テーブル(541)を維持し、１つ以上のブロックの各ラインごとに１つの共用テーブルエントリーがあり、各共用状態エントリーは

、ラインの考えられる状態を指示し、該考えられる状態は、無効、共用、排他的及び保留であり；

複数の対称的マルチプロセッサの各プロセッサごとにプライベート状態テーブル(542)を維持し、各プライベート状態テーブルは、複数のプライベート状態エントリー(545)を有し、特定のプライベート状態テーブルのプライベート状態テーブルエントリーは、それに関連する特定のプロセッサによりアクセスされる特定のラインの考えられる状態を指示し；

共用データ構造体の特定のブロックのディレクトリ情報をホームプロセッサのメモリに記憶し、ディレクトリ情報は、特定のブロックのサイズを含み；

データが使用できるかどうかチェックするために共用データをアクセスする命令においてプログラム(310)を実行可能にし；そして

共用データをアクセスするための要求を発している１つのプロセッサからアクセス要求を受信するのに応答して、特定のラインを含む特定のブロックと、その特定のブロックのサイズとを、ネットワークを経て、上記要求を発しているプロセッサへ送信し、可変サイズのブロックに記憶された共用データ構造体をプロセッサがネットワークを経て交換できるようにする、

という段階を備え、

上記実行可能化が、

アナライザ(320)で、プログラム(310)を手順(301)に細分化(breaking)し、当該手順(301)を、基本実行ブロック(302)に細分化し、当該基本実行ブロックが命令の組からなり、当該命令の組は、もし、当該組の最初のものが実行されるならば、実行される命令の組であり、

上記基本実行ブロック、及び、データ、及び、実行フロー(303)を分析して、メモリ(212)の共用された部分内の割当てられたアドレスへメモリアドレスを割当て、そこへのアクセスを実行する、命令を特定(locate)し、

オプティマイザ・モジュール(330)で、命令をプログラム(310)内に挿入して、アクセスが、整合的な(coherent)やり方でで実行されることを保証するために、データが利用可能か否かをチェックし、

イメージ・ジェネレータ(340)で、チェックするための上記命令で実行可能なプログラム(351)、及び、ミス・ハンドリング・プロトコル手順（352）のための手順、及び、メッセージ・パッシング・ライブラリ(353)を含む、修正されたマシン実行可能なイメージ(350)を生成する、

ステップを含む点において特徴付けられる方法。

【請求項２】

ホームプロセッサにより維持されたディレクトリ(1300)にディレクトリ情報を記憶し、ディレクトリは、共用データ構造体(551)の１つ以上のブロックの各ライン(552)ごとにエントリー(1301)を含み、各エントリーは、ラインを含む特定ブロックのサイズ(1315)を含む請求項１に記載の方法。

【請求項３】

特定ブロックの各ライン(552)のエントリー(1301)に、上記プロセッサ(211)のうちの制御しているプロセッサの識別(1310)を維持し、この制御しているプロセッサは特定のラインを含む特定のブロックの排他的コピーを最後に有する請求項２に記載の方法。

【請求項４】

各プロセッサ(211)ごとに１つのビット(1321)を含むビットベクトル（1320）をエントリー(1301)に維持し、各ビットは、対応するプロセッサが特定ブロックの共用コピーを有するかどうか指示する請求項３に記載の方法。

【請求項５】

プログラム(310)の実行中に共用データ構造体(551)に対して割り当てられた１つ以上のブロックのサイズを動的に変更する請求項１に記載の方法。

【請求項６】

共用テーブルエントリー(545)の１つを変更する前に共用状態テーブルをロックするステップ、及び、更に、
１つあるいはそれより多いブロックのサイズをダイナミックに変更する前に、無効化するために、１つあるいはそれより多いブロックの各ライン(552)の状態を設定するステップを含む、
請求項１に記載の方法。
【請求項７】
プライベート状態テーブルに関連したプロセッサ(211)のみによりプライベート状態テーブル(512)の１つを変更する請求項６に記載の方法。
【請求項８】
特定のプロセッサに関連したプライベート状態テーブル(542)の状態をダウングレードするときに、特定の対称的マルチプロセッサ(210)の特定の１つのプロセッサ(211)から、その特定の対称的マルチプロセッサの他のプロセッサへメッセージを選択的に送信する請求項７に記載の方法。
【請求項９】
第１の共用データ構造体(1421)の１つ以上のブロックのラインの数は、第２のデータ構造体(1431)の１つ以上のブロックのラインの数とは異なる請求項１に記載の方法。
【請求項１０】
あるプログラム(1411)の第１データ構造体(1421)の１つのラインにおけるバイトの数は、別のプログラム(1441)の第２データ構造体(1451)の１つのラインにおけるバイトの数とは異なる請求項１に記載の方法。
【請求項１１】
ネットワーク(220)と、
上記ネットワークによって相互接続され各々が複数のプロセッサ(211)を含む複数の対称的マルチプロセッサ(210)と、
各対称的マルチプロセッサに対するアドレスのレイアウトを有するメモリ(212)であって、各メモリアドレスは、共用データ(550)を記憶する指定された組の仮想共用アドレスを有し、上記仮想共用アドレスの一部分は、上記プロセッサのうちのいずれかにおいて実行されるプログラム(310)によってアクセスしうる１つ以上のブロックとして共用データ構造体(551)を記憶し、特定の割り当てられるブロックのサイズは、上記共用データ構造体のサイズと共に変化し、各ブロックは、整数本のライン(552)を含み、各ラインは、所定のバイト数の共用データを含み、
及び、
データが利用可能か否かをチェックするために、上記共用されたデータにアクセスする命令において、プログラム（351）を実行可能にするための手段(320)、
を含むシステムであって、
上記レイアウトは、
ｉ）複数の共用状態エントリー(545)を含む共用状態テーブル（541）であって、１つ以上のエントリーの各ラインごとに１つの共用エントリーがあり、各共用エントリーは、ラインの考えられる状態を指示し、該考えられる状態は、無効、共用、排他的及び保留であるような共用状態テーブルと、
ii）上記複数の対称的マルチプロセッサの各プロセッサごとのプライベート状態テーブル(542)であって、各プライベート状態テーブルは、複数のプライベート状態エントリー(545)を有し、特定のプライベート状態テーブルのプライベート状態エントリーは、それに関連する特定のプロセッサによりアクセスされる特定のラインの考えられる状態を指示するようなプライベート状態テーブルと、
を含み、
上記共用データは、データが使用できるかどうかチェックするためにアクセスされ、可変サイズのブロックに記憶された共用データ構造体を交換するため要求を発しているプロセッサへ１つの他のプロセッサから上記ネットワークを経て特定のブロックが送られ、

上記実行可能とするための手段は、
プログラム(310)を、手順(301)に細分化(breaking)し、当該手順(301)を、基本実行ブロック(302)に細分化するためのアナライザ・モジュール（320）であって、基本実行ブロックが命令の組からなり、当該命令の組は、もし、当該組の最初のものが実行されるならば、実行されるものであり、
アナライザ・モジュールが、基本実行ブロック、及び、データ、及び、実行フロー(303)を分析して、メモリ(212)の共用された部分内の割り当てられたアドレスへメモリ・アドレスを割り当て、そこへのアクセスを実行する命令を特定(locate)するためのものでもあり、
アクセスが整合的なやり方で実行されることを保証するために、データが利用可能か否かチェックするために、命令をプログラム(310)内に挿入するためのオプティマイザ・モジュール（330）、
チェックするための上記命令で実行可能なプログラム(351)、及び、ミス・ハンドリング・プロトコル手順(352)のための手順、及び、メッセージ・パッシング・ライブラリ(353)を含む、修正されたマシン実行可能なイメージ(350)を生成するためのイメージ・ジェネレータ(340)、
を含む点において特徴付けられる、システム。
【請求項１２】
上記プライベート状態テーブル(540)は、排他的テーブル(1000)を含む請求項１１に記載のシステム。
【請求項１３】
上記排他的テーブル(1000)は、共用部分（1001）及びプライベート部分(1002)を含む請求項１２に記載のシステム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、一般に、対称的マルチプロセッサに係り、より詳細には、対称的マルチプロセッサ間にデータを共用する方法に係る。
【０００２】
【従来の技術】
分散型コンピュータシステムは、典型的に、通信ネットワークによって互いに接続された多数のコンピュータを備えている。ある分散型コンピュータシステムにおいては、ネットワーク化されたコンピュータが共用データにアクセスすることができる。非常に多数のコンピュータがネットワーク化される場合には、分散型システムは、「かたまり的に(massively)」並列であると考えられる。かたまり的に並列なコンピュータは、１つの効果として、複雑な計算問題を適度な時間内に解くことができる。
【０００３】
このようなシステムにおいては、コンピュータのメモリが集合的に分散型共用メモリ（ＤＳＭ）として知られている。分散型共用メモリに記憶されたデータがコヒレントな仕方でアクセスされるよう確保することが問題となる。コヒレントとは、ある意味では、一度に１つのプロセッサしかデータの任意の部分を修正できないことを意味し、その他の点では、システムの状態が非決定論的である。
【０００４】
図１は、複数のコンピュータ１１０を含む典型的な分散型共用メモリシステム１００を示す。各コンピュータ１１０は、バス１０４によって互いに接続された単一プロセッサ１０１と、メモリ１０２と、入力／出力（Ｉ／Ｏ）インターフェイス１０３とを備えている。これらコンピュータは、ネットワーク１２０により互いに接続される。ここでは、コンピュータ１１０のメモリ１０２が共用メモリを構成する。
【０００５】
最近、分散型共用メモリシステムは、対称的なマルチプロセッサ（ＳＭＰ）のクラスタ

として構成されている。ＳＭＰシステムでは、共用メモリをハードウェアで効率的に実施することができる。というのは、プロセッサが対称的で、例えば、その構造及び動作が同一であり、そして単一の共用プロセッサバス上で動作するからである。ＳＭＰシステムは、４個又は８個のプロセッサの場合に良好な価格／性能比を有する。しかしながら、特別に設計されたバスのために、１２個又は１６個のプロセッサを越えてＳＭＰシステムのサイズを拡張することは困難である。

【０００６】

ネットワークで接続された対称的マルチプロセッサを使用して大規模な分散型共用メモリシステムを構成することが望まれる。その目標とするところは、プロセスがメモリを効率的に共用できるようにして、第１のＳＭＰで実行されている１つのプロセスにより第２のＳＭＰに取り付けられたメモリからフェッチされたデータを、その第１のＳＭＰで実行されている全てのプロセスに直ちに使用できるようにすることである。

【０００７】

ほとんどの既存の分散型共用メモリシステムにおいては、仮想メモリ（ページング）ハードウェアのロジックは、一般に、プロセスが実行されているローカルＳＭＰのメモリに記憶されていない共用データをアクセスするようにそのプロセスが試みた場合に信号を発する。データがローカルで入手できない場合には、ページ欠陥ハンドラーの機能が、リモートプロセッサで実行されているプロセスにメッセージを通信するソフトウェアルーチンに置き換えられる。

【０００８】

【発明が解決しようとする課題】

この解決策に伴う主たる問題は、典型的な仮想メモリのページ単位が４Ｋ又は８Ｋバイトであるために、データコヒレント性が大きな（粗い）サイズの量でしか与えられないことである。このサイズは、多数のプロセスによりアクセスされる非常に小さいサイズのデータ単位、例えば、３２又は６４バイトとは一致しない。粗いページサイズの粒度にすると、ネットワークトラフィックが増加し、システム性能が低下することになる。

【０００９】

更に、同じＳＭＰで実行される多数のプロセスは、一般的に、共用データに関する状態情報を共用する。それ故、競合状態のおそれが生じる。競合状態は、システムの状態が、どのプロセスが最初に完了するかに依存するときに生じる。例えば、多数のプロセスが同じアドレスにデータを書き込むことができる場合に、そのアドレスから読み取られるデータは、プロセスの実行順序に依存することになる。この順序は、ランタイム条件に基づいて変化する。競合状態は、ロックやフラグのようなインライン同期チェックをプロセスに付加することにより回避できる。しかしながら、明確な同期は、オーバーヘッドコストを増大すると共に、システムの実施を不可能にする。

【００１０】

対称的マルチプロセッサ間のデータ転送の単位を、アクセスされるデータ構造体のサイズに基づいて変更できるのが望ましい。大きなデータ構造体に対するコヒレント性の制御は、データを転送する時間を償還できるように大きなデータ単位の転送を許さねばならない。小さなデータ構造体に対するコヒレント性は、小さなデータ単位の転送を許さねばならない。又、偽の共用を受ける大きなデータ構造体に対しコヒレント性の小さな単位を使用することもできねばならない。偽の共用とは、異なるプロセスによりアクセスされた独立したデータエレメントがコヒレントなデータ単位に記憶されるときに生じる状態である。

【００１１】

【課題を解決するための手段】

ソフトウェアで実施される本発明の方法は、可変サイズのデータ量を使用する分散型共用メモリシステムを用いる対称的マルチプロセッサ間にデータを共用できるようにする。分散型共用メモリシステムにおいては、対称的マルチプロセッサがネットワークにより互いに接続される。各々の対称的マルチプロセッサは、複数の同一のプロセッサと、アドレ

スを有するメモリと、対称的マルチプロセッサをネットワークを経て相互接続するための入力／出力インターフェイスとを備えている。

【００１２】

本発明は、その広い形態においては、コンピュータシステムの対称的マルチプロセッサのメモリに記憶されたデータへのアクセスを共用するための請求項１に記載の方法に係る。

【００１３】

以下に説明するように、メモリの１組のアドレスが、集合的に、共用データを記憶するための仮想共用アドレスと称される。共用データは、対称的マルチプロセッサのいずれかのプロセッサにおいてプロセスとして実行されるプログラムの命令によりアクセスすることができる。仮想共用アドレスの一部分は、プロセスにより使用される共用データ構造体を記憶するために１つ以上のブロックとして割り当てられる。データがフェッチされ、そして個々のブロックのレベルにコヒレントに保たれる。

【００１４】

本発明の好ましい実施形態では、特定の割り当てられるブロックのサイズは、特定の共用データ構造体に対して変更することができる。各ブロックは、整数のラインを含み、各ラインは、所定のバイト数の共用データを含む。

特定のブロックのディレクトリ情報が、「ホーム」プロセッサと称するプロセッサのメモリのディレクトリに記憶される。割り当てられたブロックは、ラウンドロビン式に種々のプロセッサに指定される。ディレクトリ情報は、特定ブロックのサイズと、ブロックを最後に変更したプロセッサの識別と、ブロックのコピーを有する全てのプロセッサの識別とを含む。

【００１５】

実行の前に、好ましくは、プログラムを統計学的に分析して、ロード及び記憶命令のようなメモリアクセス命令を探索する。プログラムに付加的な命令を追加することによりプログラムが実行可能化される。付加的な命令は、ロード及び記憶命令のターゲットアドレスが共用データ構造体の特定のラインをアクセスするかどうかそしてターゲットアドレスのデータが有効状態を有するかどうか調べるために動的にチェックすることができる。

【００１６】

データが無効である場合には、アクセス要求が発生される。要求を発するプロセッサからのアクセス要求を受け取るのに応答して、特定のラインを含む特定のブロック及びその特定のブロックのサイズが、その要求を発するプロセッサへ送られる。ブロックは、ネットワークを経て送られる。これにより、対称的マルチプロセッサは、可変サイズのブロックに記憶された共用データ構造体をネットワークを経て交換することができる。

【００１７】

【発明の実施の形態】

以下、添付図面を参照し、本発明の好ましい実施形態を一例として詳細に説明する。

システムの概要

図２は、本発明を利用することのできる対称的マルチプロセッサ（ＳＭＰ）の分散型共用メモリ（ＤＳＭ）コンピュータシステム２００を示している。ＤＳＭ－ＳＭＰシステム２００は、ネットワーク２２０により互いに接続された複数のＳＭＰシステム２１０を備えている。各ＳＭＰシステム２１０は、プロセッサバス２０９により互いに接続された２つ、４つ、８つ又はそれ以上の対称的プロセッサ２１１を備えている。更に、各ＳＭＰ２１０は、システムバス２１３により対称的プロセッサ２１１に接続されたメモリ（Ｍ）２１２及び入力／出力インターフェイス（Ｉ／Ｏ）２１４を含むことができる。

【００１８】

メモリ２１２は、ダイナミックランダムアクセスメモリ（ＤＲＡＭ）である。メモリ２１２は、データの空間的及び時間的ローカル性の利点を取り入れるために高速ハードウェアキャッシュを含んでもよい。頻繁に使用されるデータは、キャッシュに記憶されることが多い。

メモリ２１２は、プログラム２１５及びデータ構造体２１６を記憶する。メモリ２１２のアドレスの幾つかは、集合的に、１組の共用仮想アドレスと称することができる。データ構造体の幾つかは、共用データを含むことができる。共用データは、仮想アドレスを用いていずれかのＳＭＰ２１０のいずれかのプロセッサ２１１で実行されているいずれかのプロセスによりアクセスすることができる。

【００１９】

バス２０９及び２１３は、データ、アドレス及び制御ラインを用いてＳＭＰ２１０の要素を接続する。ネットワーク２２０は、対称的マルチプロセッサ２１０間にメッセージを通信するためのネットワークプロトコル、例えば、非同期転送モード（ＡＴＭ）又はＦＤＤＩプロトコルを使用する。或いは又、ネットワーク２２０は、デジタル・イクイップメント社により製造されたメモリチャンネルのような高性能のクラスターネットワークの形態であってもよい。

【００２０】

一般的なシステムオペレーション

ＳＭＰ－ＤＳＭシステム２００の動作中に、プログラム２１５の命令がプロセッサ２１１により実行スレッド又はプロセスとして実行される。命令は、ロード及び記憶命令を用いてデータ構造体２１６をアクセスする。いずれのプロセッサ２１１で実行されるいずれのプログラム２１５も、いずれのメモリ２１２に記憶されたいずれの共用データ構造体２１６もアクセスできることが望まれる。

【００２１】

実行可能化

好ましくは、以下に説明するように、プログラム２１５は、実行の前に実行可能化される。実行可能化(instrumentation) とは、プログラム２１５のアクセス命令（ロード及び記憶）を静的に位置決めするプロセスである。又、実行可能化は、メモリ２１１の部分を割り当て及び割り当て解除する命令も位置決めする。

【００２２】

命令が位置決めされると、付加的な命令、例えば、ミスチェックコードをアクセス命令の前にプログラムに挿入し、メモリアクセスを正しく実行するよう確保することができる。ミスチェックコードは、付加的な命令を実行するのに必要なオーバーヘッドの量を減少するように最適化される。割り当て及び割り当て解除命令のために挿入される付加的な命令は、割り当てられているブロックのサイズのようなコヒレント性の制御情報を維持する。

【００２３】

上記のように、プログラム２１５は、分散型メモリ２１２の幾つかのアドレスを共用メモリとして見ることができる。共用メモリの特定のターゲットアドレスについては、命令がデータのローカルコピーをアクセスするか、又はデータのコピーを要求するために別のプロセッサにメッセージを送らねばならない。

【００２４】

アクセス状態

いずれのＳＭＰについても、共用メモリに記憶されるデータは、無効又は有効の２つの考えられる状態をもつことができる。有効状態は、共用又は排他的なサブ状態をもつことができる。データの状態が無効である場合には、データへのアクセスが許されない。状態が共用される場合には、ローカルコピーが存在し、他のＳＭＰも、コピーをもつことができる。状態が排他的である場合には、１つのＳＭＰのみがデータの唯一の有効コピーを有し、他のＳＭＰはデータにアクセスできない。更に、以下に述べるように、データは、遷移状態、即ち「ペンディング」状態にある。

データの状態は、ネットワーク２２０を経て通信されるコヒレント性制御メッセージにより維持される。これらのメッセージは、実行可能化されたプログラムのミスチェックコードによりコールされた手順によって発生される。

【００２５】

データは、それが共用状態又は排他的状態を有する場合だけローカルＳＭＰのメモリから直接ロードすることができる。データは、状態が排他的である場合だけローカルメモリに記憶することができる。プロセッサが無効状態にあるデータをロードするように試みる場合、又はプロセッサが無効又は共用状態にあるデータを記憶するように試みる場合には、通信が必要とされる。通信を必要とするこれらのアクセスを「ミス(miss)」と称する。

メモリ２１２のアドレスは、共用データを記憶するように動的に割り当てることができる。幾つかのアドレスは、ローカルプロセッサ上で実行されているプロセスのみによりアクセスされたプライベートデータを記憶するように静的に割り当てることができる。幾つかのアドレスをプライベートデータとして指定することによりオーバーヘッドを減少することができる。というのは、ローカルプロセッサによるプライベートデータへのアクセスは、ミスに対してチェックする必要がないからである。

【００２６】

ハードウェア制御式の共用メモリシステムの場合のように、メモリ２１２のアドレスは、割り当て可能なブロックへと仕切られる。ブロック内の全てのデータは、コヒレント単位としてアクセスされる。システム２００の特徴として、ブロックは、異なる範囲のアドレスに対して可変サイズをもつことができる。以下に述べるミスチェックコードを簡単化するために、可変サイズのブロックは、「ライン(line)」と称する固定サイズ範囲のアドレスへと更に仕切られる。

状態情報は、ラインごとのベースで状態テーブルに維持される。ラインのサイズは、特定のプログラム２１５が実行可能化されるときに予め定められ、通常、３２、６４又は１２８バイトである。１つのブロックは、整数本のラインを含むことができる。

【００２７】

システム２００の動作中に、メモリアクセス命令を実行する前に、ミスチェックコードは、ターゲットアドレスがプライベートメモリ内にあるかどうか決定する。ターゲットアドレスがプライベートメモリ内にある場合には、ミスチェックコードは、即座に完了することができる。というのは、プライベートデータは、常にローカルプロセッサによりアクセスできるからである。さもなくば、ミスチェックコードは、特定ブロックのどのラインが命令のターゲットアドレスを含むか計算し、そしてそのラインがアクセスに対して正しい状態にあるかどうか決定する。状態が正しくない場合には、ミスハンドラーが呼び出され、リモートＳＭＰのメモリからデータをフェッチする。

【００２８】

実行可能化プロセス

図３は、付加的な命令に必要とされるオーバーヘッドの量を減少するようにプログラムを実行可能化するのに使用できるプロセス３００のフローチャートである。更に、プロセス３００は、対称的マルチプロセッサによりアクセスされる可変サイズのデータ量に対してコヒレント性の制御を受け入れる。プロセス３００は、アナライザモジュール３２０、オプチマイザモジュール３３０、及び実行可能なイメージジェネレータ３４０を含む。

【００２９】

マシンで実行可能なプログラム３１０は、アナライザモジュール３２０へ送られる。アナライザ３２０はプログラム３１０を手順３０１に分断し、そして手順３０１を基本的実行ブロック３０２に分断する。基本的ブロック３０２は、第１の命令が実行される場合に全てが実行される１組の命令として定義される。手順及び基本的ブロックの命令は、プログラムコール及び流れグラフ３０３を形成するように分析される。

【００３０】

グラフ３０３は、プログラム３１０のデータ及び実行の流れを決定するために使用することができる。基本的ブロック及びグラフ３０３は、メモリアドレスを割り当ててその割り当てたアドレスへのアクセスを実行する命令を位置決めするために分析される。命令がメモリ２１２の共用部分をアクセスする場合には、そのアクセスがコヒレント的に実行されるよう確保するためにミスチェックコードが挿入される。

【００３１】

ミスチェックコードは、以下に詳細に述べるように、オプチマイザモジュール３３０により挿入される。プログラム３１０が実行可能化された後に、イメージジェネレータ３４０は、変更されたマシン実行可能なイメージ３５０を発生する。この変更されたイメージ３５０は、ミスチェックコードを伴う実行可能化されたプログラム３５１と、ミスハンドリングプロトコル手順３５２と、メッセージパスライブラリ３５３とを含む。

図４は、図３のオプチマイザモジュール３３０により実行されるステップを示す。これらのステップは、メモリの仕切り４１０、レジスタの分析４２０、コードのスケジューリング４３０、ロードチェック分析４４０、及びバッチ処理４５０のステップを含む。

【００３２】

メモリレイアウト

図５は、図２のメモリ２１２に対するアドレスの割り当てを示す。アドレスは図５の下部から上部へと増加する。アドレスは、スタック５１０、プログラムテキスト５２０、静的に割り当てられたプライベートデータ５３０、状態テーブル５４０、及び動的に割り当てられた共用データ５５０に対して指定される。

動作中に、スタック５１０により使用されるアドレスは、スタックオーバーフローエリア５０５に向かって減少する。テキストスペース５２０は、実行可能な命令、例えば図３のイメージ３５０を記憶するのに使用される。テキストに指定されるアドレスは、テキストオーバーフローエリア５２５に向かって増加する。

【００３３】

プライベートデータセクション５３０のアドレスは、単一のローカルプロセッサにより排他的に使用されるデータ構造体、例えば、共用されないデータを記憶するのに使用される。メモリのこの部分のアドレスは、特定のプログラムが実行のためにロードされるときに静的に割り当てられる。

【００３４】

状態テーブル

状態テーブル５４０は、共用状態テーブル５４１、プライベート状態テーブル５４２及び排他テーブル１０００を含む。排他テーブル１０００は、共用部分１００１及びプライベート部分１００２も含む。

共用及びプライベート状態テーブル５４１及び５４２は、各々、割り当てられたアドレスの各ラインごとに１バイトの共用及びプライベート状態エントリー５４５を含む。状態エントリー５４５のビットは、対応するデータラインの種々の状態を指示するのに使用できる。１つ以上のデータラインによりブロックが構成される。

【００３５】

好ましい実施形態によれば、特定のＳＭＰ２１０の全てのプロセッサ２１１は同じデータを共用することができる。それ故、状態テーブルエントリー５４５はＳＭＰ２１０の全てのプロセッサについて共用される。これは、ブロック、例えば、１つ以上のデータラインがリモートＳＭＰからフェッチされ、そしてブロックの状態が無効から共用又は排他的へと変化するときに、ＳＭＰの共用メモリハードウェアがデータの状態を確認し、そしてＳＭＰのプロセッサ２１１が新たなデータをアクセスできることを意味する。

特定のＳＭＰの２つ以上のプロセッサが状態テーブルエントリーをアクセスするよう同時に試みることがあるので、エントリーにアクセスがなされる前にエントリーがロックされる。コードに挿入されたミスチェックが状態テーブルエントリーへのアクセスを必要とすることもある。しかしながら、この場合には、オーバーヘッドを減少するためにエントリーはロックされない。むしろ、各プロセッサは、オーバーヘッドの追加を伴うことなくインラインコードによりアクセスすることのできる対応するプライベート状態テーブル５４２を維持する。

【００３６】

プロセッサのプライベート状態テーブル５４２のエントリーは、２つの異なるメカニズムにより更新される。

プロセッサが無効データをアクセスするよう試みる場合には、ミス状態が生じそしてデ

ータはリモートＳＭＰからフェッチされる。受信の際に、データの状態が有効となる。これは、今やデータが使用できるので状態の「アップグレード」と称されるが、従来はこのようにならない。しかしながら、データは、同じＳＭＰ２１０の他のプロセッサのプライベート状態テーブルにおいて無効とマークされたままとなる。

【００３７】

これらの他のプロセッサの１つがここでデータをアクセスするよう試みた場合に、他のプロセッサは、そのプライベート状態テーブル５４２において依然として無効状態を見ることになる。他のプロセッサは、共用状態テーブル５４０にロックを収集し、そしてローカルＳＭＰに対してデータが有効であることを決定すると共に、それに応じてそのプライベート状態テーブル５４２を更新する。データへのその後のアクセスは、共用状態テーブル５４０をアクセスする必要なく行うことができる。

データの状態を無効に戻すことが必要であり、例えば、別のＳＭＰのプロセッサがデータを必要とする場合には、データの状態が「ダウングレード」される。

この場合に、要求を受け取るプロセッサは、ローカルＳＭＰで動作している他のプロセッサに内部メッセージを選択的に送信し、それらのプライベート状態テーブル５４２に維持された状態をダウングレードできるようにする。ラインの「ダウングレード」は、全てのプロセッサがそれらのプライベート状態テーブルを切り換えるまで完了されない。

【００３８】

無効化要求を受け取るプロセッサがローカルＳＭＰの全てのプロセッサの全てのプライベート状態テーブルを直接的に切り換えるべきである場合に競合状態が生じることに注意されたい。例えば、第１のプロセッサは、記憶に対してインラインチェックを行う間に有効状態を見るが、第２のプロセッサは、第１のプロセスが修正されたデータを記憶する機会を得る前にデータの状態を無効へとダウングレードするときに、競合状態が生じる。

【００３９】

競合状態を回避する１つの方法は、インラインミスチェックコードと共に状態テーブルロックを得ることである。しかしながら、この解決策は、ロッキングのためにオーバーヘッドを増加する。これは、デジタル・イクイップメント社により製造されたアルファプロセッサのように弛緩メモリモードをもつプロセッサの場合に特に言えることである。従って、競合状態を効率的に回避するためには、プライベート状態テーブルの使用が重要となる。

プライベート状態テーブル５４２の使用は、ミスチェックコードにおける競合状態を回避するだけでなく、ＳＭＰ２１０内のデータの状態をダウングレードしながらも、通信する必要のあるメッセージの数を減少する。例えば、ローカルプロセッサが、ローカルＳＭＰ内の有効なデータを決してアクセスしない場合は、そのプライベート状態テーブルを更新する必要はない。

【００４０】

共用データ

共用データ５５０のアドレスは、プログラムによりその実行の間に動的に割り当てられる。１つの効果として、共用データ５５０のアドレスは、可変サイズのブロック５５１において割り当てることができる。これらブロックは、ライン５５２へと更に仕切られる。

図５に示すレイアウトでは、全てのアクセス命令を実行可能化する必要はない。例えば、プログラムスタック５１０に記憶されるデータは、共用されない。それ故、スタックポインタレジスタ（ＳＰ）をベースとして使用する命令は、ミスチェックコードの適用を必要としない。又、プライベートデータポインタレジスタ（ＰＲ）を用いてプライベートデータ５３０をアクセスする命令は、実行可能化する必要がない。

【００４１】

レジスタの使用

図３のアナライザモジュール３２０は、グラフ３０３及びデータ流分析を使用して汎用レジスタの内容を追跡し、レジスタに記憶された値がＳＰレジスタに基づくアドレスから導出されたかＰＲレジスタに基づくアドレスから導出されたかを決定する。従って、導出

されたアドレスを経てスタック又はプライベートデータをアクセスする命令は、実行可能化される必要がない。又、アナライザ３２０は、ミスチェックコードを適用する必要があるときに空いているレジスタを位置決めすることができ、これは、ミスチェックコードにより使用されるレジスタをセーブ及び回復する必要を排除する。

【００４２】

プライベート状態テーブル５４０を各プロセッサのプライベートアドレススペースのアドレス０ｘ２０００００００００でスタートすることにより、ターゲットアクセスアドレスのシフトが、プライベート状態テーブル５４０における対応エントリー５４５のアドレスを直接的に発生できる。図５に示されたアドレスのレイアウトは、６４ビットのアドレス能力をもつプロセッサに対するものであるが、レイアウト５００は、３２ビット及び他のアドレス能力を有するプロセッサ用に変更できることを理解されたい。

【００４３】

最適化されたミスチェックコード

図６は、図５のメモリレイアウトに対して最適化されたミスチェックコード６００を示している。アクセスのためのターゲットアドレスは、命令６０１により決定することができる。しかしながら、例えば、既に実行されたロード又は記憶命令により、ターゲットベースアドレスがレジスタに既に確立されている場合には、ターゲットベースアドレスをロードする命令６０１が必要とされない。

【００４４】

シフト命令６０２は、ターゲットアドレスが共用データエリア５５０内にあるかどうか決定する。そうでない場合には、分岐命令６０３に直接進み、元の記憶命令が実行される。シフト命令６０４は、ターゲットアドレスを含むラインに対応する状態テーブルのエントリーのアドレスを発生する。状態「ＥＸＣＬＵＳＩＶＥ（排他的）」の値をゼロにすることにより、定数値との比較の必要性が排除される。むしろ、ミスに対してチェックするために、簡単な分岐命令６０７を実行することができる。命令６０５－６０６は、状態テーブルエントリーを検索する。ミスハンドリングコード６０８は、ミスの場合に実行され、そして元の記憶命令は、６０９において実行される。

ミスチェックコード６００は、ターゲットアドレスが共用データエリアにない場合に、３つの命令の実行を必要とするだけである。共用データアクセスの場合には、７つの命令を実行することが必要である。

【００４５】

コードスケジューリング

図４のステップ４３０において、命令スケジューリング技術を使用し、ミスチェックコード６００により使用されるオーバーヘッドの量を更に減少することができる。パイプライン式及びスーパースケーラーである近代的なプロセッサにおいては、追加されるミスチェックコードは、多くの場合に、最小のパイプライン遅延しか導入せず、且つ単一のプロセッササイクル中に多数の命令が発生される可能性を最大にするように構成できる。

【００４６】

例えば、あるプロセッサでは、シフト動作の結果を使用できるまでに１サイクルの遅延しか生じない。それ故、図６の第２のシフト命令６０４が前進されて、第１のシフト命令７０２から生じる遅延スロットを占有する場合には、再配置された第２シフト７０３と、ｌｄｑ＿ｕ命令７０５との間の停動が排除される。これは、コード７００がコード６００より少数のマシンサイクルで完了できることを意味する。コード６００の場合と同様に、多くの場合に命令７０１の必要性を排除できることに注意されたい。

【００４７】

多重発生プロセッサのオーバーヘッドコストを更に減少するために、ミスチェックコード７００の命令は、それらが元の実行可能なコードにおいてパイプラインの停動の間に発生されるか又は実行可能なイメージの命令と同時に発生されるように配置することができる。最初の３つの命令７０１－７０３の実行は、レジスタ（ｒ１及びｒ２）が空き状態に保たれる限り、命令の基本的ブロックにおいて前進されることに注意されたい。実際に、

多くの場合に、３つ全部の命令は、命令を実行する付加的なオーバーヘッドを完全に隠すに充分なほど前進させることができる。それ故、図７に示すようにコードを配列することが有効であることは明らかである。

【００４８】

記憶チェック

このミスチェックコードは、アクセス命令が記憶命令７１０であるときに更に最適化することができる。この場合に、最初の３つの命令７０１－７０３が記憶命令７１０の前に配置される。残りの命令７０４－７０７は、記憶命令７１０の後に配置される。この配置は、記憶されるべき値をプログラムが計算する間に待ち時間の長い命令が記憶命令７１０の直前にある場合に効果的である。この場合に、記憶命令７１０は、値が得られるまで停動しなければならない。それ故、前進された命令の実行に関連するオーバーヘッドが完全に隠される。

【００４９】

ロードチェック

図８及び９に示すように、ミスチェックコードのオーバーヘッドを更に減少するために、ロード命令によりロードされるデータを分析することができる。ラインのデータが無効になるときに、「フラグ」８０１が、そのラインに関連した全てのアドレス８１０－８１１に記憶される。フラグ８０１は、例えば、０ｘＦＦＦＦＦＦ０３（－２５３）である。従って、状態テーブルエントリーを経てラインの状態を決定するのではなく、ほとんど全ての場合に、ロードされたデータから状態を決定することができる。

【００５０】

例えば、ターゲットアドレスのデータは、ステップ８２０において、ロード命令９０１に関連される。ステップ８３０において、フラグの補数８４０、例えば２５３が加算される。ステップ８５０において、メモリからロードされたデータが無効状態を指示するおそれがあるかどうか調べるチェックが行われる。もしそうであれば、ミスコード８７０に進み、さもなくば、ステップ８６０のノー・ミスに続く。ミスが推定される場合には、状態テーブル５４０のラインに対してエントリーをチェックすることによりミスコード８７０を確認することができる。

これは、プログラムがフラグに等しいデータを実際に使用するという稀な場合を考慮する。

【００５１】

フラグは、単一の命令９０２を用いて無効データをチェックできるように選択される。ほとんどの定数を使用できることが考えられる。ゼロの値を用いて無効状態を指示する場合には、簡単な分岐命令で充分であることに注意されたい。しかしながら、ゼロ又は他の小さい整数、例えば、－１、０、＋１を使用する場合には、ミスチェックコードの測定されるオーバーヘッドが、多数の偽のミスの取り扱いにより増加すると思われる。実際に、フラグ０ｘＦＦＦＦＦＦ０３を使用するときには、偽のミスが生じることは稀であり、それ故、図９に示す最適化されたミスチェックコード９００は、ロード命令、例えば２つの命令に対してミスチェックコードを相当に減少する。

【００５２】

オーバーヘッドを減少するのに加えて、フラグ技術は、他の効果も有する。主たる効果は、ロードアクセスが有効である場合に、状態テーブルを検査する必要が排除されることである。又、ターゲットアドレス及び状態チェックからの「フラグ」データのロードは、原子的に行われる。この原子性は、同じＳＭＰの別のプロセッサにおいて生じることのある同じアドレスに対するロード命令とプロトコル動作との間の考えられる競合状態を排除する。

【００５３】

又、フラグチェック技術は、フローティングポイントロードアクセス命令にも使用できる。この場合には、ミスチェックコードは、ターゲットアドレスのデータをフローティングポイントレジスタにロードした後に、フローティングポイント加算及び比較を行う。し

かしながら、あるプロセッサにおいては、フローティングポイント命令が長い関連遅延をもつことがある。それ故、フローティングポイントミスコードは、同じターゲットアドレスに対して整数ロードを挿入しそして図８及び９について述べたフラグチェックを行うことにより最適化することができる。付加的なロード命令があっても、この技術は、状態テーブルのエントリーをチェックする場合より依然として効率的である。

【００５４】

或いは又、フローティングポイントデータは、このような動作がその基礎となるプロセッサにおいて使用できる場合にはフローティングポイントレジスタから整数レジスタへ直接転送することもできる。

命令のスケジューリングをロードミスコードチェックのために図９の命令に適用できることも理解されたい。好ましい実施形態では、図４のスケジューリングステップ４３０は、ロードの値を使用すべきときにパイプラインの停動を回避するために命令９０２及び９０３の実行を遅延するよう試みる。

【００５５】

状態テーブル５４０からエントリーをロードするときには、キャッシュのミスは、ミスチェックコードに対するオーバーヘッドの増加の１つの潜在的な要因となる。プログラムが良好な空間的ローカル性を有する場合には、プログラムは、多数のハードウェアキャッシュミスを経験しない。６４バイトラインを使用する場合には、状態テーブルに必要なメモリがそれに対応するラインのメモリの１／６４のみとなる。しかしながら、プログラムが良好な空間的ローカル性をもたない場合には、データのキャッシュミス及び状態テーブルのキャッシュミスが生じ易い。

【００５６】

排他テーブル

図１０は、共用の排他テーブル１００１を示す。各プロセッサごとに１つづつある図５のプライベートな排他テーブル１００２は、構造的に同様である。排他テーブル１０００の目的は、記憶命令に対して状態テーブルエントリーをロードするミスチェックコードにより生じるハードウェアキャッシュのミスを減少することである。排他テーブル１００１は、各対応するラインごとに１ビットづつのビットエントリー１０１０を有する。対応するラインが排他的状態を有する場合には、ビットが論理１にセットされ、さもなくば、ビットが論理０にセットされる。

【００５７】

状態テーブル５４０のエントリー５４５をチェックするのではなく、記憶ミスチェックコードは、排他テーブル１０００のビット１０１０を検査して、対応するラインが排他的状態を有するかどうか決定する。ラインが排他的状態をもたない場合には、記憶を直ちに実行できる。

６４バイトラインの場合には、排他テーブル１０００により使用されるメモリは、ラインにより使用されるメモリの量の１／５１２である。それ故、排他テーブル１００１を用いる記憶ミスチェックコードにより生じるハードウェアキャッシュミスの数は、状態テーブルのみを用いる場合に生じるハードウェアキャッシュミスの１／８である。記憶ミスコードチェックに排他テーブル１０００を使用することは、図８の無効フラグ８０１によって一部可能となることに注意されたい。ロードに対するロードミスチェックコードは、データが有効な場合には状態テーブル５４０にアクセスする必要がない。従って、排他テーブル１０００は、記憶命令に対するミスチェックコードによってのみアクセスされる。

【００５８】

バッチ処理

図４のバッチ最適化ステップ４５０は、データのロード及び記憶が共通のベースレジスタに対してしばしばバッチで実行されることを確認する。例えば、プログラムにおいて、データがそれらのアドレスに基づく逐次の順序でアクセス及び処理される場合がしばしばある。バッチ最適化ステップ４５０は、１本のラインのサイズ以下の範囲、例えば、６４バイト以下の範囲のターゲットアドレスをアクセスする１組の命令を検出する。このよう

な１組のロード及び記憶命令は、せいぜい、２本のすぐ隣接するライン及びある場合には１本のラインのみにおいてデータをアクセスできるだけである。

【００５９】

この場合に、ミスチェックコードは、２本のラインが正しい状態にあるかどうか決定する。もしそうであれば、その組の全てのロード及び／又は記憶命令は、付加的なチェックを必要とせずに実行することができる。又、バッチチェックは単一のラインにわたるある範囲のターゲットアドレスに対して実行できることを理解すべきである。しかしながら、２本の隣接ラインをチェックするコードは、オーバーヘッドを実質的に増加することなく単一のラインをチェックすることができる。

１つの制約として、バッチ式のロード及び記憶命令は、個別のミスチェックコード有する他のロード及び記憶と混合することができない。他のロード及び記憶により誘起されるミスは、バッチ式のロード及び記憶命令に対して不適切な結果を生じるようにラインの状態を変化させる。しかしながら、多数のベースレジスタを経てのロード及び記憶は、対応するベースレジスタを経て参照される各ラインに対して適切なミスチェックが行われる限り、バッチ処理することができる。

【００６０】

別の制約として、命令のバッチにより使用されるベースレジスタは、チェックされる範囲内のターゲットアドレスをバッチがアクセスする間に変数により変更することができない。これは、バッチに対する初期チェックを無効化する。ベースレジスタを定数により変更することができる。というのは、この場合には、バッチ式のアクセス命令を実行する前に範囲のチェックを静的に実行できるからである。

【００６１】

バッチ処理技術は、ミスチェックコードのオーバーヘッドを常に首尾良く減少する。しかしながら、この技術は、「解かれた(unrolled)」ループの命令に対して特に有用である。解かれたループは、繰り返しの円形態ではなく直線的に実行される命令を含む。従って、アクセス命令は、一般に、繰り返し中に変更されない小さな範囲のベースレジスタ内で機能する。この場合に、バッチ技術は、ほぼ常時適用することができ、非常に効果的である。

【００６２】

バッチ処理は、単一の基本的ブロックの命令に対して常に試みられるが、多数の基本的ブロックにわたるロード及び記憶命令に対してバッチ処理を実行することも考えられる。多数の基本的ブロックにわたるロード及び記憶がバッチ処理されるときには、付加的な制約が生じる。バッチ組の命令は、サブルーチンコールを含むことができない。というのは、これらコールは、被呼サブルーチンに未知のターゲットアドレスを有するロード及び記憶の実行を生じることがあるからである。又、バッチ式の命令は、ループを含むことができない。というのは、ループが繰り返される回数を、バッチの命令が実行されるまで決定できないからである。更に、条件分岐を含むバッチにおいては、分岐した実行経路の１つに生じる記憶が、全ての経路に生じなければならない。バッチ式の命令が実行されるときにはどの記憶アクセスが実行されたかを決定することしかできない。

【００６３】

バッチ処理は、多数のベースレジスタに対し、１つ以上の基本的ブロックにわたって多数のロード及び記憶を任意にバッチ処理することができる。

「グリーディ(greedy)」なバッチアルゴリズムを使用することができる。グリーディなアルゴリズムは、バッチに含ませることのできる数のロード及び記憶命令を配置することができる。このアルゴリズムは、以下に述べるように終了条件に達したときに完了する。バッチに単一のロード又は記憶命令しかない場合は、バッチ式のミスチェックコードが使用されない。

【００６４】

２つの考えられる実行経路を生じる条件分岐命令に遭遇した場合には、バッチに含むべき命令に対して両経路が検査される。２つの別々の実行経路の走査は、２つの経路の実行

が合体するときに合体される。
終了条件は、変数により変更されたベースレジスタを使用するロード又は記憶命令と；チェックされているライン以外のターゲットアドレスを有するロード又は記憶命令と；サブルーチンコールと；ループ、例えば１つ以上の命令の再実行を生じる条件分岐命令と；サブルーチンの終了に到達することと；多数の岐路の１つにおける記憶命令と；並列岐路と合体する１つの岐路の走査であって、並列岐路の走査が既に終了していることとを含む。

【００６５】
命令のバッチに対するミスチェックコード
図１１及び１２は、ある範囲のターゲットアドレス１１３０をアクセスするバッチ式ロード命令のグループに対する流れ１１００及びミスチェックコード１２００を各々示している。範囲１１３０をチェックする１つの便利な方法は、１組のアクセス命令によりアクセスされるアドレスの範囲１１３０の最初のアドレス１１１１及び最終アドレス１１２１においてミスコードチェック１１４０－１１４１を実行することである。最初と最後のアドレスは、各々、最初と最後のライン１１１０及び１１２０になければならない。命令１２０１－１２０４を参照されたい。命令１２０５及び１２０６は、無効フラグをチェックする。

【００６６】
いずれかのアドレス１１１１又は１１２１が無効である（１１５０）場合は、ミスハンドリングコード１１６０がコールされる。最初と最後の両方のアドレスが有効なデータを記憶する場合には、その組の全ての命令を、更にチェックを行わずに実行することができる。１つの効果として、エンドポイントアドレスに対するミスチェックコード１２００を互いにインターリーブして、パイプラインの停動を効率的に排除することができる。

【００６７】
メッセージパスライブラリ
図３のメッセージパスライブラリ３５３は、対称的マルチプロセッサ２１０がネットワーク２２０を経て通信できるようにするに必要な手順を備えている。例えば、ネットワーク２２０がＡＴＭプロトコルを使用する場合には、ライブラリ３５３のルーチンがＡＴＭ型のメッセージを通信する。ライブラリ３５３のルーチンは、任意のサイズのメッセージを送信及び受信することができる。更に、これらルーチンは、到来メッセージを周期的にチェックすることができる。

【００６８】
ミスハンドリングプロトコル
図３の実行可能化されたプログラム３５１にリンクされる他のコードは、ミスハンドリングプロトコルコード３５２である。このコードは、別の対称的マルチプロセッサのメモリからデータをフェッチし、データの共用コピー間にコヒレント性を維持し、そしてデータを記憶するよう試みるプロセッサがデータの排他的関係を有するよう確保することができる。

【００６９】
又、プロトコルコード３５２は、「ロック」及び「バリア」のような同期動作も実施する。コード３５２は、ミスチェックコードがロード又は記憶ミスを検出するか、或いは同期動作が要求されるときに、呼び出される。
プロトコルコード３５２は、ディレクトリベースの無効化プロトコルである。
図５の共用データ５５０の各ブロック５５１に対し、プロセッサの１つが「ホーム」プロセッサに指定される。ブロックは、ラウンドロビン式に、例えば、割り当ての順序で異なるホームプロセッサに指定することができる。ブロックは、図３のプログラム３１０の１つによって配置のヒントが供給される場合に、特定のプロセッサに明らかに指定することができる。

【００７０】
ホームプロセッサは、ブロックのアドレスに記憶されたデータを初期化する役目を果た

す。又、ホームプロセッサは、割り当てられたブロックのラインの初期状態を確立し、例えば、状態は、排他的な所有権を表すことができる。又、ホームプロセッサは、ブロックに関する初期のディレクトリ情報を形成する。

又、ディレクトリは、以下に述べるように、どのプロセッサがホームプロセッサに指定されたブロックのコピーを有するかを指示する。ホームプロセッサ以外のプロセッサがブロックのデータをアクセスしたいときには、ブロックのデータをロードしたいか記憶したいかを指示するメッセージをホームプロセッサに送信する。記憶の場合には、所有権の要求も送られる。

【００７１】

図１３に示すように、各プロセッサ２１０は、プロセッサがホームであるところのブロックに含まれたラインに関する情報を記憶できるディレクトリ１３００を維持する。又、いかなるときにも、特定のブロックの各ラインは、「制御」プロセッサを有する。ラインを制御するプロセッサは、そのラインに対して排他的な所有権を最後に有したプロセッサである。

ホームプロセッサが所有する各ブロックに対して、ディレクトリ１３００は、ブロックの各ラインに対するエントリ１３０１を有する。各エントリー１３０１は、識別（ＩＤ）１３１０と、ブロックサイズ１３１５と、ビットベクトル１３２０とを含む。ＩＤ１３１０は、どのプロセッサがブロックを現在制御するかを指示し、そしてベクトル１３２０は、ブロックのコピーを有する各プロセッサに対して１つのビット１３２１を有する。ブロック１３１５のサイズは、以下に詳細に述べるように、変化させることができる。

【００７２】

プロトコルメッセージ

プロセッサ２１１は、図２のネットワーク２２０を経て互いにメッセージを通信する。メッセージは、次の一般的形式のものである。要求メッセージは、ロード及び記憶の目的でデータのコピーを要求し、そして応答メッセージは、要求されたデータを含むことができる。データの要求は、一般に、ホームプロセッサに送られる。ホームプロセッサがデータのコピーをもたない場合には、要求が制御プロセッサへ送られる。制御プロセッサは、その要求を発生したプロセッサに直接的に応答することができる。

【００７３】

又、あるメッセージがプロセス同期に使用される。２つの形式の同期メカニズムを使用することができる。第１に、プロセッサは、特定の「バリア」アドレスに同期することができる。バリアドレスに同期するときには、バリアアドレスに到達したプロセッサは、他の全てのプロセッサもバリアアドレスに到達するまで待機する。

別の形式の同期は、ロックによるものである。「ロック」は、共用メモリの特定のアドレスにおいていずれかのプロセッサにより与えることができる。別のプロセッサは、そのロックが解除されるまで同じアドレスにロックを与えることができない。

ミスハンドリングコード３５２によりサポートされるメッセージは、以下に詳細に説明する。

【００７４】

読み取りメッセージ

読み取りメッセージは、特定プロセッサが読み取るためのデータを要求する。このメッセージは、要求されたデータと、要求を発しているプロセッサの識別とを記憶するブロックのアドレスを含む。このメッセージに応答して、要求されたデータを含む全ブロックがフェッチされる。

【００７５】

書き込みメッセージ

書き込みメッセージは、要求されたデータのアドレスと、要求を発しているプロセッサの識別とを含む。このメッセージは、要求を発しているプロセッサがデータのコピーをもたないときに、ブロックに新たなデータを記憶する目的でデータのブロックを要求する。それ故、メッセージは、データのブロックの所有権も要求する。

【００７６】
所有権メッセージ
このメッセージは、要求を発しているプロセッサがデータのコピーを有する場合にデータの所有権を要求する。このメッセージは、要求を発しているプロセッサがそのデータのコピーを変更すると決定した場合に使用される。所有権メッセージは、データのアドレスと、要求を発しているプロセッサの識別とを含む。
【００７７】
クリーンメッセージ
このメッセージは、データの（クリーンな）読み取り専用コピーの要求を通信するのに使用される。クリーンメッセージは、要求されたデータのアドレスと、バイト数と、要求を発しているプロセッサの識別とを含む。最適化として、ホームプロセッサが要求されたデータのコピーを有する場合には、要求を別のプロセッサへ送給する必要がない。
【００７８】
送給メッセージ
このメッセージは、データの書き込み可能なコピーを、データを現在制御しているプロセッサから、データを要求したプロセッサへ送給することを要求する。
送給メッセージは、要求されたデータのアドレスと、バイト数と、要求を発しているプロセッサの識別とを含む。
【００７９】
無効化メッセージ
このメッセージは、データのコピーを無効化することを要求する。無効化が完了すると、要求を発しているプロセッサへ確認が送られる。この無効化メッセージは、要求されたデータのアドレスと、無効化されるべきバイト数と、要求を発しているプロセッサの識別とを含む。
【００８０】
ダウングレードメッセージ
このメッセージは、ブロックの状態がダウングレードされるときに、ＳＭＰ内において、プライベート状態テーブルもダウングレードしなければならないプロセッサへローカル的に送られる。ダウングレードメッセージは、ダウングレードの形式と、要求されたデータのアドレスと、バイト数と、要求を発しているプロセッサの識別とを含む。ダウングレードメッセージを受け取る最後のプロセッサは、ダウングレードを開始した要求に関連した動作を完了する。
【００８１】
クリーン応答メッセージ
このメッセージは、クリーンメッセージにおいて要求された実際のデータのコピーを含む。このクリーン応答メッセージは、要求されたデータのアドレスと、バイト数と、データとを含む。
送給応答メッセージ
このメッセージは、要求されたデータの書き込み可能なコピーを含む。この送給応答メッセージは、要求されたデータのアドレスと、バイト数と、データとを含む。
【００８２】
無効化応答メッセージ
このメッセージは、データが無効化された確認である。この無効化応答メッセージは、要求されたデータのアドレスと、無効化されたバイト数とを含む。
バリア待機メッセージ
このメッセージは、全てのプロセッサが指定のバリアアドレスに到達したときに、要求を発しているプロセッサへの通知を要求する。このバリア待機メッセージは、バリアアドレスと、要求を発しているプロセッサの識別とを含む。
【００８３】
バリア終了メッセージ

このメッセージは、バリア待機メッセージの条件を満足したことを指示する。
バリア終了メッセージは、バリアアドレスを含む。

ロックメッセージ

このメッセージは、ロックの所有権を要求する。ここでの実施においては、ロックは、共用メモリの指定のアドレスにおいて作用される。そのアドレスに記憶されたデータは、ロックメッセージには何ら関係ない。ロックメッセージは、ロックに関連したアドレスを含む。

【００８４】

ロック送給メッセージ

このメッセージは、ロックされたアドレスを現在制御しているプロセッサにロック要求を送給する。このロック送給メッセージは、ロックアドレスを含む。

ロック応答メッセージ

このメッセージは、ロックされたアドレスの制御を、要求を発しているプロセッサに転送する。このロック応答メッセージは、ロックされたアドレスを含む。

【００８５】

ダーティデータ

上記したプロトコルメッセージは、「ダーティ(dirty)」データを共用できるようにする。これは、ブロックのホームプロセッサがデータのクリーンな最新のコピーをもつ必要がないことを意味する。例えば、別のプロセッサがそのデータのコピーを変更し、その後、その変更されたデータのコピーをホームプロセッサ以外のプロセッサで共用することができる。この特徴は、ホームプロセッサへの書き戻しの必要性を任意なものにする。さもなくば、プロセッサが別のプロセッサからのダーティデータのコピーを読み取るときに、ホームプロセッサへの書き戻しが必要となる。

【００８６】

ポーリング

プロセッサ２１１により発生されたメッセージを処理するのにポーリングメカニズムが使用される。例えば、ネットワーク２２０は、プロセッサが要求メッセージの応答を待機するときに到来メッセージに対してポーリングされる。これはデッドロック状態を回避する。

更に、要求に対して適度な応答時間を確保するために、プログラムは、それがファンクションコールを行うときに到来メッセージをポーリングするように実行可能化される。ネットワーク２２０が、待ち時間の短い形式のものである場合、ポーリングは、例えば、プログラム制御バックエッジごとのように、より頻繁に行うことができる。プログラム制御バックエッジは、ループを繰り返し再実行させる分岐型の命令である。それ故、バックエッジポーリングは、ループの各繰り返しごとに行われる。

【００８７】

メッセージは、割り込みメカニズムを用いてサービスすることができる。しかしながら、割り込みサービスを行うには、通常、長い処理時間を要する。というのは、割り込み時に存在する状態を先ずセーブしそしてその後に回復しなければならないからである。又、ポーリングの場合に、原子的プロトコル動作を実施するタスクは、簡単化される。

【００８８】

プロセッサ間のメッセージ送信に関連したオーバーヘッドは比較的高いので、余計なプロトコルコヒレントメッセージが最小にされる。ブロックのホームプロセッサは、現在制御しているプロセッサへ要求を送ることにより要求にサービスするよう保証するので、ディレクトリ１３００の情報を変更する全てのメッセージは、それらメッセージがホームプロセッサに到達するときに完了することができる。従って、送られた要求が満足されたことを確認するために余計なメッセージを送信する必要はない。更に、排他的要求に応答して発生される全ての無効化確認は、ホームプロセッサを経るのではなく、要求を発しているプロセッサへ直接通信される。

【００８９】

ロックアップフリーキャッシュ

プロトコル３５２は、非ブロッキングのロード及び記憶を許すハードウェア型のロックアップフリーキャッシュと実質的に同等の解除一貫性モデルも与える。

分散型共用メモリに「キャッシュ」されたデータは、無効、共用、排他的、無効保留、又は共用保留の状態のうちのいずれか１つをもつことができる。保留状態は、ラインを含むブロックの要求が未解決になっているときのラインの一時的状態である。無効保留状態は、未解決の読み取り又は書き込み要求を有するデータに対して存在する。共用保留状態は、未解決の所有権要求をもつデータに対して存在する。

【００９０】

非ブロッキングの記憶は、データに対する要求がなされた後にプロセッサが命令を処理し続けるようにすることによりサポートされる。要求が未解決である間に、プロトコルは、ブロックのローカルコピーにおいて変更されるデータのアドレスに注目する。次いで、要求されたデータのブロックが使用できるようになったときに、変更されたデータを要求されたデータと合体することができる。ロードのバッチ処理は単一のチェックに対し多数の未解決のロードを招くので、上記のロード及び記憶をバッチ処理することにより非ブロッキングロードが可能となることに注意されたい。

【００９１】

ロックアップフリー特性は、保留状態を有するデータに対してもサポートできる。保留データのアドレスにデータを記憶することは、データが記憶されるアドレスに注目しそして図３のミスハンドリングコード３５２へアドレスを通すことにより、行うことができる。

保留状態におけるブロックへの全ての記憶は、適当な状態テーブルエントリーにロックが保持される間にプロトコルルーチン内で完了される。保留記憶を行うこの方法は、同じブロックに対してプロトコル動作を後で行うプロセッサに記憶が見えるように確保するために重要である。

【００９２】

共用保留状態を有するデータのアドレスからのロードは直ちに行うことが許される。というのは、プロセッサがデータのコピーを既に有しているからである。

無効保留状態を有するブロックのデータのアドレスからのロードも、そのロードが有効データを記憶するブロックのラインのアドレスからである限り、行うことができる。図８の無効フラグ８０１を使用するために、保留ラインへの有効ロードは、迅速に行うことができる。ロードされた値は、無効フラグに等しくないので、保留ラインへの有効ロードは、直ちに行うことができる。

【００９３】

可変粒度

ここに述べるプロトコルの特徴として、単一のプログラム又は単一のデータ構造体内であっても、コヒレント性に対する可変粒度が考えられる。可変粒度が考えられるのは、例えば、バイト、ロングワード及びクオドワードのような非常に小さな粒度でデータをアクセスするソフトウェア命令により、ミスに対する全てのチェックが行われるからである。これに対し、他の分散型メモリシステムは、通常は４０９６又は８１９２バイトの仮想メモリページサイズにより決定された固定の粗い粒度のアドレスでミスチェックを行うために仮想メモリハードウェアを使用する。

【００９４】

プログラムにより使用される異なる形式のデータは、ほとんど自然に且つ効率的に可変粒度でアクセスされる。例えば、入力／出力デバイスの多量の逐次アドレスから読み取られ及びそこに書き込まれるデータのブロックは、例えば、２Ｋ又は４Ｋ等の粗い粒度で最良に取り扱われる。しかしながら、多くのプログラムは、例えば、３２、２５６、１０２４バイトのような非常に小さい範囲のアドレスに対してランダムアクセスも必要とする。

【００９５】

アプリケーションプログラム及びデータ構造体が可変アクセス粒度をもつことを許すこ

とにより、データを最も効率的な転送単位で通信できるために、性能を改善することができる。例えば、ブロックに「凝集」されたデータのように良好な空間的ローカル性を有するデータは、長い通信待ち時間を償還するために粗い粒度で搬送することができる。これに対して、「偽の共用」を受けるデータは、細かい粒度で通信することができる。

【００９６】

偽の共用とは、例えば、アレーエレメントのようなデータの独立した部分が、例えば、１つ以上のブロックのようなデータ構造体に記憶され、そして多数の対称的マルチプロセッサによってアクセスされる状態である。可変サイズのブロックは、偽の共用データの小さな独立部分を含む固定サイズの多量のデータを対称的マルチプロセッサ間に繰り返し転送する必要性を排除する。

【００９７】

従って、図３のプロセス３００は、可変粒度のデータ転送単位を処理するように最適化される。データ転送の単位、例えば、ブロックは、プログラムに対して選択された固定のラインサイズに基づきラインの整数倍となり、例えば、異なるプログラムは、異なるラインサイズ（３２、６４、１２８バイトライン）のデータをアクセスすることができる。

【００９８】

特定のデータ構造体として適当なブロックサイズを選択するために、割り当てられたサイズに基づくヒューリスティックを使用することができる。基本的なヒューリスティックは、割り当てられたデータ構造体のサイズに等しいブロックサイズから、例えば、１Ｋ又は２Ｋバイトのようなデータ構造体の所定のスレッシュホールドサイズまでを選択する。所定のスレッシュホールドサイズより大きい割り当てられたデータ構造体に対しては、粒度が単にラインのサイズとなる。ヒューリスティックの理論的解釈は、小さなデータ構造体は、アクセス時に１つの単位として転送されねばならず、一方、アレーのような大きなデータ構造体は、偽の共用を回避するために細かい粒度で通信されねばならないことである。

【００９９】

ヒューリスティックは、ブロックサイズを明確に定める特殊な割り当て命令をプログラムに挿入することにより変更することができる。割り当てられたブロックのサイズは、プログラムの正しさに影響しないので、最大性能のための適切なブロックサイズを明確に決定することができる。

図１３に示すように、データの割り当て可能な部片のブロックサイズ１３１５は、ディレクトリ１３００においてホームプロセッサにより操作される。各ラインエントリーは、対応するブロックのサイズ１３１５を含む。プロセッサは、ブロックのデータが要求を発しているプロセッサに搬送されるときにブロックのサイズが分かる。

【０１００】

プロセッサは、ブロックのサイズを知る必要がないので、サイズを動的に決定することができる。例えば、ホームプロセッサは、先ず、データ構造体を構成する全てのラインを無効化し、次いで、ディレクトリエントリー１３０１においてブロックサイズを変更することにより、全データ構造体の粒度を変更することができる。

【０１０１】

ホームプロセッサは、特定ラインのターゲットアドレスのデータに対し、例えば、読み取り、書き込み又は所有権のようなアクセス要求を受け取ると、ブロックのサイズをルックアップすることができる。次いで、ホームプロセッサは、ブロック全体を構成する正しいライン数を、要求を発しているプロセッサに送信することができる。ベクトル１３２０を用いるプロセッサによりラインの他のコピーを適当に取り扱うことができる。初期要求以外のアクセス要求に対する応答において、全てのプロトコル動作は、ブロックの全てのラインにおいて行われる。

【０１０２】

ミスチェックコードを簡単化するために、データの部片の状態がチェックされそしてラインごとのベースで維持される。しかしながら、プロトコル３５２は、ブロックの全ての

ラインが常に同じ状態となるよう確保する。それ故、インラインミスチェックコードは、可変サイズのブロックに対する状態を効率的に維持することができる。

【０１０３】

可変サイズの粒度の場合には、プロセッサは、要求されたラインを含むブロックのサイズを知らなくてもよい。例えば、プロセッサは、アドレスＡ及びアドレスＡ＋６４のデータをアクセスするよう要求する。プロセッサがブロックのサイズを知らない場合には、たとえアドレスが同じブロックにあっても、各ターゲットアドレスごとに１つづつ、６４バイトのラインサイズを仮定して２つの要求を発する。

【０１０４】

しかしながら、１つの効果として、ここに述べるプロトコルは、ラインを含む全ブロックを単一のメッセージにおいて転送する。その後、初期要求を処理するホームプロセッサは、第２の要求が必要とされないことも確認できる。これは、第２ラインの要求が完全に処理される前に、別のプロセッサが第１ラインへのアクセスを要求するときを除いて、全ての場合に言えることである。この場合に、データの現在状態を常に決定できないので、第２要求を初期要求として処理しなければならない。

【０１０５】

図１４は、可変粒度を有するデータ構造体を示す。メモリ１４０１は、第１のプロセッサ（ＰＲＯＣ１）に関連され、そしてメモリ１４０２は、第２のプロセッサ（ＰＲＯＣ２）に関連される。

第１プロセッサのメモリ１４０１内で、第１のプログラム（Ｐ１）１４１１は６４バイトのラインをもつように割り当てられたデータ構造体を有し、そして第２のプログラム（Ｐ２）１４４１は、３２バイトのラインをもつように割り当てられたデータ構造体を有する。

【０１０６】

第１のプログラム１４１１は、データ構造体１４２１及び１４３１を含む。データ構造体１４２１は、１２８バイトの１ブロック、例えば、ブロック当たり２本のラインを含む。データ構造体１４３１は、６４バイトの８ブロック、例えばブロック当たり１本のラインを有する。

第２のプログラムは、データ構造体１４５１、１４６１及び１４７１を含む。データ構造体１４５１は、各々３２バイト（１ライン）の８ブロックを含む。データ構造体１４６１は、各々１２８バイト（４ライン）の３ブロックを含む。データ構造体１４７１は、２５６バイトの１ブロック、例えば、８本のラインを含む。

【０１０７】

第２のプロセッサのメモリ１４０２は、同等のプログラム１４１２及び１４４２と、それらのデータ構造体とを含む。上記のように、プロセッサは、ブロックサイズの転送単位でデータを通信する。例えば、第１プログラム１４１１及び１４１２は、ブロック１４０３を用いてデータを転送し、そして第２プログラム１４４１及び１４４２は、ブロック１４０４を転送する。１つの効果として、ブロック１４０３及び１４０４は、異なるサイズ、例えば、可変粒度と、異なるラインサイズ、例えば、３２及び６４バイトをもつことができる。

【０１０８】

以上、特定の実施形態について本発明を詳細に説明した。本発明の範囲内で他の応用及び修正がなされ得ることが理解されよう。それ故、本発明の範囲内に含まれるあらゆる変更や修正は、特許請求の範囲に包含されるものとする。

【図面の簡単な説明】

【図１】　公知の単一プロセッサの分散型共用メモリシステムを示す図である。

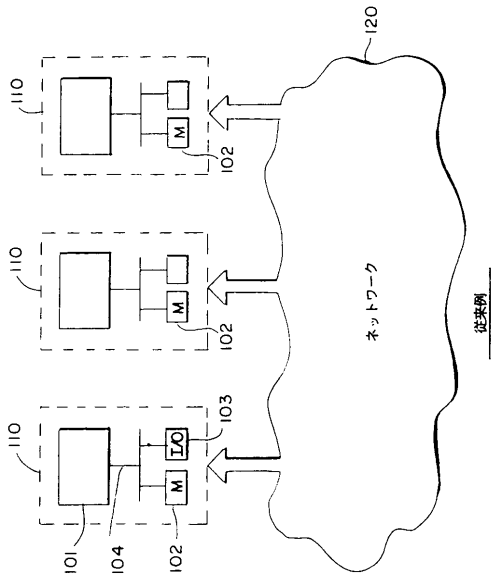
【図２】　本発明の好ましい実施形態による対称的マルチプロセッサの分散型共用メモリシステムのブロック図である。

【図３】　プログラムを実行可能化するプロセスのフローチャートである。

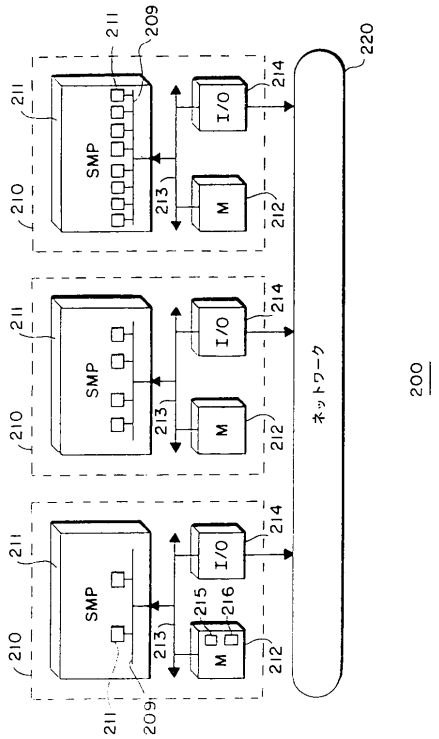
【図４】　最適化段階のブロック図である。

【図５】　メモリの仕切りのブロック図である。
【図６】　最適化された記憶ミスチェックコードを示す図である。
【図７】　最適なスケジューリングのために構成されたミスチェックコードを示す図である。
【図８】　ロードアクセスにおける無効データをチェックするプロセスのフローチャートである。
【図９】　無効フラグをチェックする命令を示す図である。
【図１０】　排他テーブルのブロック図である。
【図１１】　アクセス命令のバッチをチェックするプロセスのブロック図である。
【図１２】　図１１のプロセスを実施する命令であって、最適なスケジューリングのために構成された命令を示す図である。
【図１３】　ブロックディレクトリのブロック図である。
【図１４】　可変粒度を有するデータ構造体のブロック図である。
【符号の説明】
２００　ＤＳＭ－ＳＭＰコンピュータシステム
２０９　プロセッサバス
２１０　ＳＭＰシステム
２１１　対称的プロセッサ
２１２　メモリ
２１３　システムバス
２１４　Ｉ／Ｏインターフェイス
２１５　プログラム
２１６　共用データ構造体
２２０　ネットワーク
３００　プロセス
３０１　手順
３０２　基本的実行ブロック
３０３　プログラムコール及び流れグラフ
３１０　プログラム
３２０　アナライザモジュール
３３０　オプチマイザモジュール
３４０　イメージジェネレータ
３５０　変更されたマシン実行可能なイメージ
３５１　実行可能化されたプログラム
３５２　ミスハンドリングプロトコル手順
３５３　メッセージパスライブラリ

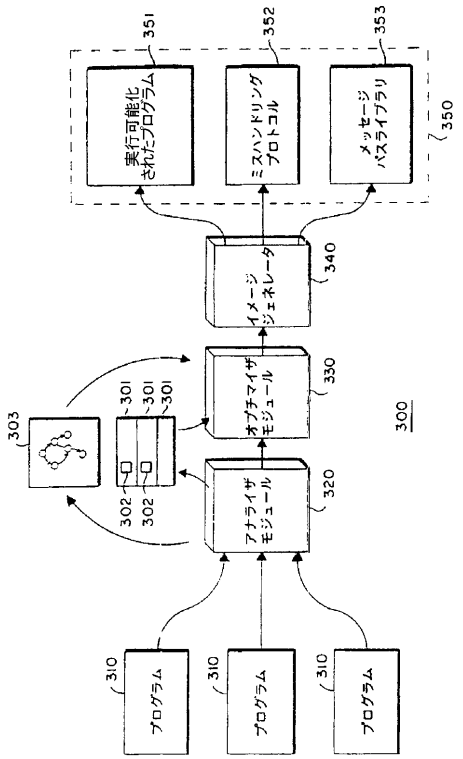
【図１】


120
110
M
102
ネットワーク
103
I/O
101
104
従来例

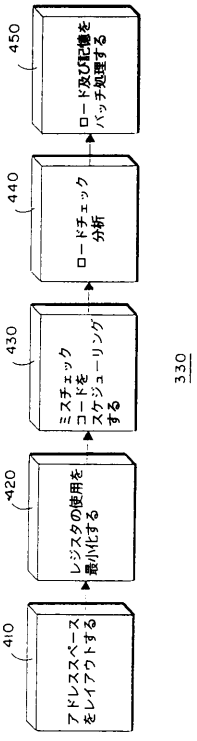
【図２】


220
211
209
SMP
210
213
212
214
I/O
M
215
216
ネットワーク
200

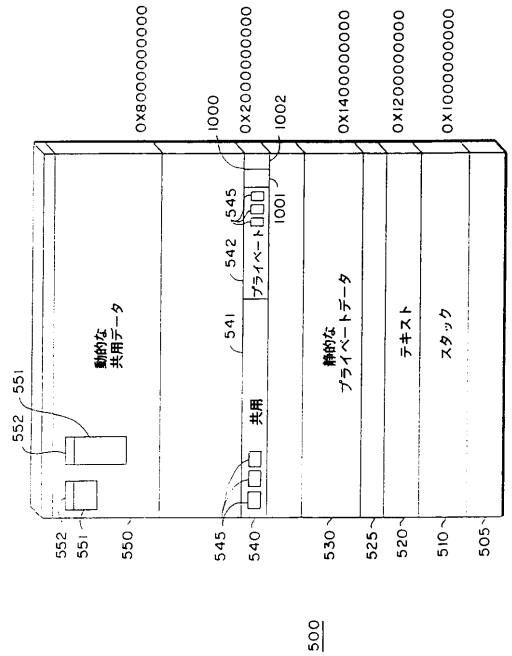
【図３】


351
352
353
実行可能化されたプログラム
ミスハンドリングプロトコル
メッセージパスライブラリ
350
イメージジェネレータ
340
オプチマイザモジュール
330
アナライザモジュール
320
303
301
302
300
310
プログラム

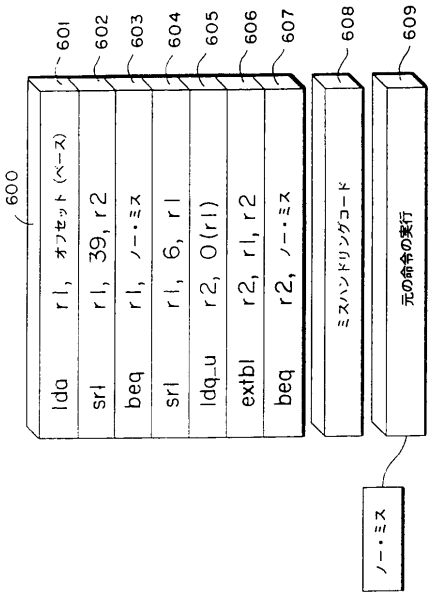
【図４】


450
ロード及び記憶をバッチ処理する
440
ロードチェック分析
430
ミスチェックコードをスケジューリングする
420
レジスタの使用を最小化する
410
アドレススペースをレイアウトする
330

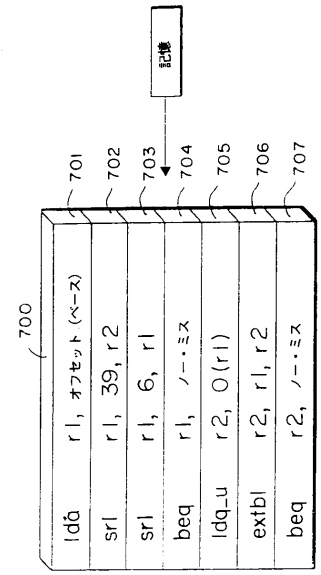
【図５】


0X8000000000
0X2000000000
0X1400000000
0X1200000000
0X1000000000
1000
1002
1001
545
542
541
551
552
動的な
共用データ
共用
プライベート
静的な
プライベートデータ
テキスト
スタック
550
540
530
525
520
510
505
500

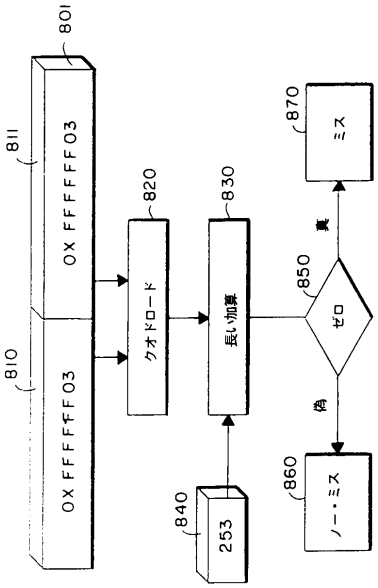
【図６】


600
601
602
603
604
605
606
607
608
609
lda r1, オフセット（ベース）
srl r1, 39, r2
beq r1, ノー・ミス
srl r1, 6, r1
ldq_u r2, 0(r1)
extbl r2, r1, r2
beq r2, ノー・ミス
ミスハンドリングコード
元の命令の実行
ノー・ミス

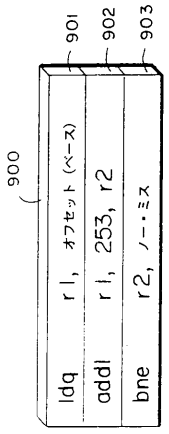
【図７】


700
701
702
703
704
705
706
707
記憶
lda r1, オフセット（ベース）
srl r1, 39, r2
srl r1, 6, r1
beq r1, ノー・ミス
ldq_u r2, 0(r1)
extbl r2, r1, r2
beq r2, ノー・ミス

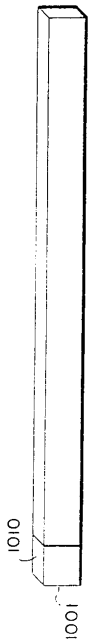
【図８】


801
811
810
820
830
840
850
860
870
0X FFFFFF03
0X FFFFFF03
253
クオドロード
長い加算
ゼロ
真
偽
ミス
ノー・ミス

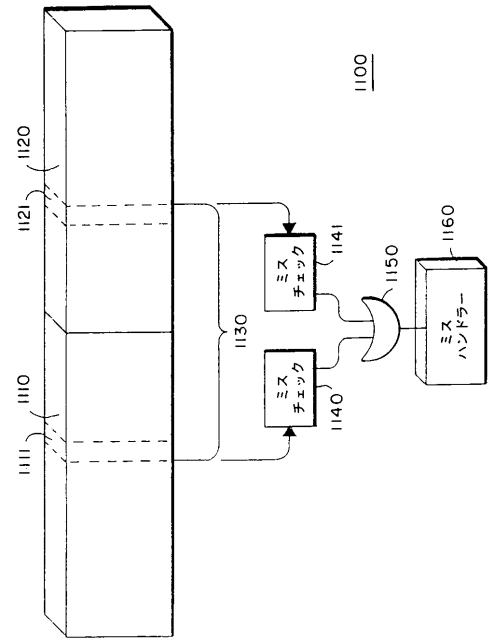
【図９】



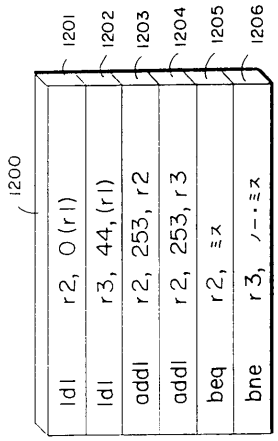
【図１０】



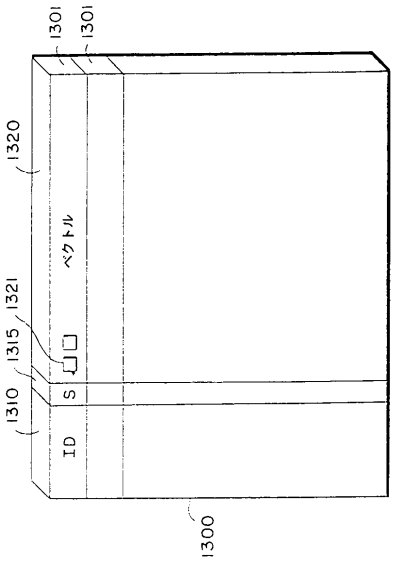
【図１１】



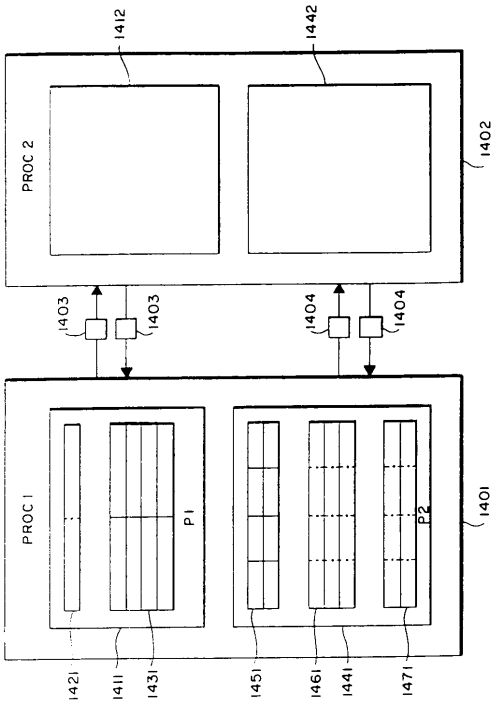
【図１２】



【図１３】


1301
1301
1320
ベクトル
1321
1315
S
1310
ID
1300

【図１４】


1412
1442
PROC 2
1402
1403
1403
1404
1404
PROC 1
P1
P2
1401
1421
1411
1431
1451
1461
1441
1471

フロントページの続き

(74)代理人 100074228
弁理士 今城 俊夫
(74)代理人 100084009
弁理士 小川 信夫
(74)代理人 100082821
弁理士 村社 厚夫
(72)発明者 ダニエル ジェイ スケイルズ
アメリカ合衆国 カリフォルニア州 ９４３０６ パロ アルト マグノリア ドライヴ ３８９８－１２
(72)発明者 コウロシュ ガーラチョルロー
アメリカ合衆国 カリフォルニア州 ９４０２５ メンロ パーク カミノ ア ロス セアロス ２２６０
(72)発明者 アンシュ アーガルワル
アメリカ合衆国 コロラド州 ８０３０１ ボールダー サーティース ストリート １６３０－２１６

審査官 石川 正二

(56)参考文献 特開平０８－０３０５６８（ＪＰ，Ａ）
特開平０４－１９５６６０（ＪＰ，Ａ）
Daniel J. Scales, Kourosh Gharachorloo, Chandramohan A. Thekkath, Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory, Proc. of the 7th Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOSV II), １９９６年
Donald Yeung, MGS: A Multigrain Shared Memory System , Proc. of the 23rd Annual Int'l Symp. on Computer Architecture (ISCA'96), １９９６年

(58)調査した分野(Int.Cl., ＤＢ名)
G06F 15/17
G06F 12/00
G06F 12/08