(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4209959号
(P4209959)
(45)発行日　平成21年1月14日(2009.1.14)　(24)登録日　平成20年10月31日(2008.10.31)

(51)Int.Cl.　　F I
G06F 11/34 (2006.01)　　G06F 11/34 B
G06F 15/167 (2006.01)　　G06F 15/167 610A

請求項の数 7　(全 23 頁)

(21)出願番号　特願平10-58065
(22)出願日　平成10年3月10日(1998.3.10)
(65)公開番号　特開平10-260869
(43)公開日　平成10年9月29日(1998.9.29)
審査請求日　平成16年11月19日(2004.11.19)
(31)優先権主張番号　08/812899
(32)優先日　平成9年3月10日(1997.3.10)
(33)優先権主張国　米国(US)

前置審査

(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　100084009
弁理士　小川　信夫

最終頁に続く

(54)【発明の名称】性能カウンタの高頻度サンプリング装置及び方法

(57)【特許請求の範囲】
【請求項１】
プログラムの命令を同時に実行するための複数のプロセッサを含むコンピュータシステムにおいて性能データを収集する装置であって、
前記性能データは、命令を実行する間に各プロセッサにより発生される特定事象の発生数であり、
前記装置は、
前記プロセッサの各々に複数接続された性能カウンタ、
前記プロセッサの各々に１つの、当該プロセッサで実行される割り込みハンドラー、
前記割り込みハンドラーの各々に１つのハッシュテーブルを含む第１メモリ、
オーバーフローバッファを含む第２メモリ、
ユーザバッファを含む第３メモリ、及び、
フラッシュする手段を備え、
前記性能カウンタは、前記性能データを記憶し、当該性能カウンタの特定の状態の発生時に、割り込みを発生するものであり、
前記ハッシュテーブルは、複数のチャンクに仕切られ、当該各チャンクは、複数のエントリーに仕切られたものであり、
前記割り込みハンドラーは、前記割り込みに応答して前記性能カウンタから前記性能データを読み取って、当該割り込みハンドラーに対応する前記ハッシュテーブルの前記エントリー、及び、前記オーバーフローバッファのいずれか一方に、前記性能データを記憶す

るものであり、
前記フラッシュする手段は、前記ハッシュテーブルの前記エントリー及び前記オーバーフローバッファに記憶されている前記性能データを、前記ユーザバッファへ周期的にフラッシュするものであり、
前記割り込みハンドラーは、特定の前記性能データを記憶する処理において、当該性能データの記憶先であるエントリーを含む前記チャンクが、前記フラッシュする手段によって、フラッシュ処理中である場合は、当該性能データを、前記オーバーフローバッファへ記憶することを特徴とする装置。
【請求項２】
各チャンクは、更に、
複数のライン、
当該チャンクが、前記割り込みハンドラーによって書き込み中であるか否かを示すフラグ、
当該チャンクが、前記フラッシュする手段によって、フラッシュ処理中であるか否かを示すフラグを備え、
前記各ラインは、更に、複数の前記エントリーを備え、各エントリーは、前記性能データ及び当該性能データの関連情報を記憶するための複数のフィールドを含む請求項１に記載の装置。
【請求項３】
上記各エントリーの複数のフィールドは、プロセス識別、プログラムカウンタ、プロセッサ事象識別及びプロセッサ事象カウンタを更に含む請求項２に記載の装置。
【請求項４】
前記割り込みハンドラーは、
上記プロセス識別、プログラムカウンタ、及びプロセッサ事象識別から、複数の前記ライン内の現在のエントリと関連付けられているか否かを決定するためのハッシュインデックスを発生し、
ミス信号に応答して、前記ハッシュインデックスにより指示された現在エントリーに記憶された性能データをハッシュテーブルからオーバーフローバッファへ移動し、性能カウンタから読み取った性能データで現在エントリーをオーバーライトし、
ヒット信号に応答して、現在ハッシュインデックスにより指示された現在エントリーに記憶されたプロセッサ事象カウンタを増加することを特徴とする請求項３に記載の装置。
【請求項５】
上記第２メモリのオーバーフローバッファは、二重バッファとして編成された第１及び第２バッファを備え、
各バッファは、ハッシュテーブルのエントリーの性能データを記憶するための複数のスロットを含み、
前記オーバーフローバッファは、更に、複数のプロセッサの各プロセッサごとに１つずつ存在する二重バッファに仕切られ、
アクティブな第２バッファに性能データを書き込みながら非アクティブな第１バッファから性能データを読み取ることを特徴とする請求項１に記載の装置。
【請求項６】
上記プログラムは、アプリケーション、オペレーティングシステム及び共用ライブラリープログラム成分を含み、更に、これらアプリケーション、オペレーティングシステム及び共用ライブラリープログラム成分から性能データを収集する請求項１に記載の装置。
【請求項７】
プログラムの命令を同時に実行するための複数のプロセッサを備え、当該各プロセッサには１セットの性能カウンタが接続されたコンピュータシステムにおいて性能データを収集するためのコンピュータ化された方法であって、
前記複数セットの性能カウンタに、命令を実行する間に各プロセッサにより発生される特定事象の発生数である性能データを記憶する段階と、

前記性能カウンタの特定の状態の発生時に、割り込みを発生する段階と、
前記割り込みに応答して性能カウンタのセットに記憶された性能データをサンプリングする段階と、
前記サンプリングされた性能データを、第１メモリの複数のハッシュテーブルであって、各プロセッサごとに１つ備え、複数のチャンクに仕切られ、当該各チャンクは、複数のエントリーに仕切られたものであるハッシュテーブルの前記エントリー、及び、第２メモリのオーバーフローバッファのいずれか一方に記憶する段階と、
前記ハッシュテーブルの前記エントリー及び前記オーバーフローバッファに記憶されている前記性能データを、前記ユーザバッファへ周期的にフラッシュする段階とを備え、
前記サンプリングされた性能データを記憶する段階は、特定の前記性能データの記憶先であるエントリーを含む前記チャンクが、前記フラッシュする段階によって、フラッシュ処理中である場合は、当該性能データを、前記オーバーフローバッファへ記憶することを特徴とする方法。

【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、コンピュータシステムに係り、より詳細には、コンピュータシステムにおける性能データの収集に係る。

【０００２】

【従来の技術】

動作しているコンピュータシステムにおける性能データの収集は、ハードウェア及びソフトウェアのエンジニアにより行われる頻繁で且つ非常に重要な作業である。ハードウェアのエンジニアは、新しいコンピュータハードウェアが既存のオペレーティングシステム及びアプリケーションプログラムといかに動作するかを決定するために性能データを必要とする。

【０００３】

プロセッサ、メモリ及びキャッシュのようなハードウェア構造体の特定の設計は、同じ１組のプログラムに対し著しく異なるそして時には予想し得ない使い方をする。ハードウェアの欠陥を識別し、それらを将来の設計において修正できることが重要である。性能データは、ソフトウェアがいかに効率的にハードウェアを使用するかを識別できると共に、改良されたシステムを設計する上で助けとなる。

【０００４】

ソフトウェアのエンジニアは、プログラムの重大な部分を識別することが必要である。例えば、コンパイラーの著者は、コンパイラーが命令を実行に対しいかにスケジュールするか、又はソフトウェアを最適化する入力を与えるために条件分岐の実行がいかに良好に予想されるかを見出さねばならない。同様に、オペレーティングシステムのカーナル、デバイスドライバ、及びアプリケーションソフトウェアプログラムの性能を理解することが必要である。

【０００５】

コンピュータシステムの動作環境を妨げることなくハードウェア及びソフトウェアシステムの性能を正確に監視することが問題となる。特に、何日又は何週間といった長い期間にわたって性能データが収集される場合に問題となる。多くの場合に、性能監視システムは、手作りである。システムのオペレーションが監視システムにより影響されないように保証するためには、コストのかかるハードウェア及びソフトウェア変更が必要となる。

【０００６】

コンピュータシステムの性能を監視できる１つの方法は、性能カウンタを使用することによるものである。性能カウンタは、システムにおける重大な事象の発生を「カウント」する。重大な事象は、例えば、キャッシュミス、命令の実行、Ｉ／Ｏデータ転送要求、等々を含む。性能カウンタを周期的にサンプリングすることにより、システムの性能を推論することができる。

【０００７】
【発明が解決しようとする課題】
コンピュータシステムの性能は、ソフトウェアの変更を伴わずに監視できることが望ましい。又、サンプリング率を固定又は可変とし、そしてその率を非常に高くできることが望ましい。更に、高頻度のサンプリング中に、サンプリングのオーバーヘッドを最小に保持し、性能データがシステムのオペレーションを正確に反映するのが望ましい。オーバーヘッドを低く保つことは、データアクセスを同期させる必要のあるマルチプロセッサシステムでは特に困難であり、サンプリング率は、非常に高いものとなり、例えば、毎秒５０，０００ないし１００，０００サンプルとなる。
【０００８】
【課題を解決するための手段】
本発明は、コンピュータシステムにおいて性能データを収集する装置を提供する。コンピュータシステムは、プログラムの命令を同時に実行するための複数のプロセッサを備えている。本発明の装置は、複数セットの性能カウンタを備えている。１セットの性能カウンタが各プロセッサに接続される。性能カウンタは、命令を実行する間に各プロセッサにより発生される性能データを記憶するためのものである。
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性能データをサンプリングするための請求項１に記載の装置、及び請求項９に記載のその方法に係る。
【０００９】
各プロセッサにおいて割り込みハンドラーが実行される。割り込みハンドラーは、割り込みに応答して、プロセッサの性能データをサンプリングするためのものである。第１メモリは、各割り込みハンドラーに関連したハッシュテーブルを含む。ハッシュテーブルは、プロセッサ上で実行される割り込みハンドラーによりサンプリングされた性能データを記憶する。第２メモリは、オーバーフローバッファを含み、このオーバーフローバッファは、ハッシュテーブルの一部分が非アクティブでありそしてそれらの部分がいっぱいである間に性能データを記憶するためのものである。第３メモリは、ユーザバッファを含む。更に、ハッシュテーブル及びオーバーフローバッファからユーザバッファへ性能データを周期的にフラッシュするための手段が設けられる。
【００１０】
好ましくは、以下に述べるように、第１メモリのハッシュテーブルは、多方セット連想(multi-way set-associative) キャッシュとして編成される。更に、多方セット連想キャッシュは、複数のチャンクを含み、各チャンクは、第１メモリと第３メモリとの間のデータ転送の単位である。更に、各チャンクは、複数のラインを含み、そして各チャンクには、対応するチャンクが非アクティブ及びいっぱいである場合を各々指示するためのａｃｔｉｖｅ＿ｃｈｕｎｋフラグ及びｆｌｕｓｈ＿ｃｈｕｎｋフラグが関連される。各チャンクのラインは、更に、複数のエントリーに仕切られ、そして各エントリーは、性能データを記憶するための複数のフィールドを含む。性能データのフィールドは、プロセッサ識別、プログラムカウンタ、プロセッサ事象識別、及びプロセッサ事象カウンタフィールドを含む。
【００１１】
プロセッサ識別、プログラムカウンタ及びプロセッサ事象識別からハッシュインデックスを発生するための手段が設けられるのが好都合である。ハッシュインデックスは、ヒット又はミス信号を発生するために特定のプロセッサに関連したハッシュテーブルのラインを探知するのに使用される。
【００１２】
ミス信号に応答して、現在ハッシュインデックスにより指示された現在エントリーに記憶された性能データがハッシュテーブルからオーバーフローバッファへ移動される。現在エントリーは、割り込みハンドラーによりサンプリングされた性能データでオーバーライトされる。ヒット信号の場合には、現在ハッシュインデックスにより指示された現在エントリーに記憶されたプロセッサ事象カウンタが増加される。性能データは、圧縮形態でエン

トリーに記憶される。

【００１３】

以下に示すように、第２メモリのオーバーフローバッファは、二重バッファとして編成された第１及び第２のバッファを備えている。各バッファは、ハッシュテーブルのエントリーの性能データを記憶するための複数のスロットを含む。

【００１４】

【発明の実施の形態】

本発明は、添付図面を参照した好ましい実施形態の以下の詳細な説明から容易に理解されよう。

システムの概要

図１に示すように、コンピュータシステム１００は、バス１４０により互いに接続されたプロセッサ１１０と、メモリサブシステム（メモリ）１２０と、入力／出力インターフェイス（Ｉ／Ｏ）１３０とを備えている。システム１００は、埋め込まれたシステム、ＰＣ、ワークステーション、メインフレーム、又はネットワークでリンクされたシステムのクラスターのメンバーである。

【００１５】

プロセッサ１１０は、ＣＩＳＣ又はＲＩＳＣのいずれかのアーキテクチャーを有する１つ以上の個々のプロセッサチップ１１１として構成することができる。各プロセッサ１１１には、性能カウンタ１１２のセットが組み合わされる。性能カウンタ１１２の各セットは、複数のレジスタとして実施することができる。これらレジスタは、システムにおいてシステムの性能を表す重大な事象の発生をカウントすることができる。

【００１６】

メモリ１２０は、個別の静的、動的、ランダム、逐次、揮発性、及び永続的な記憶エレメント、又はその組合せを含むことができる。幾つかのメモリは、ハイアラーキ構成とすることができる。記憶エレメントは、レジスタ、キャッシュ、ＤＲＡＭ、ディスク、テープ等である。メモリ１２０は、マシンで実行できる命令の形態のソフトウェアプログラム１２１と、命令によりアクセスされるデータ１２２とを記憶する。ソフトウェアプログラムは、オペレーティングシステム、デバイスドライバ、及びアプリケーションプログラムを含む。

【００１７】

Ｉ／Ｏ１３０は、プリンタ、ターミナル及びキーボードのような入力及び出力デバイスへのインターフェイスを含むことができる。又、Ｉ／Ｏ１３０は、他のコンピュータシステムとデータを通信するためにライン１５０を経てネットワーク（ＮＥＴ）１６０に接続することもできる。バス１４０は、通常、アドレス、データ、制御及びタイミング信号を種々の要素間に搬送するために複数のラインとして実施される。

【００１８】

オペレーションの概要

システム１００のオペレーション中に、プログラム１２１の命令は、１つ以上のプロセッサ１１１により実行される。命令は、一般に、プログラムの実行流を制御するか、又はデータ１２２をアクセス（ロード、記憶、読み取り及び書き込み）する。通常のオペレーション環境を著しく妨げることなくシステム１００の性能データを収集することが望まれる。性能データの分析は、システム１００のハードウェア及びソフトウェア要素の設計を最適化するのに使用できる。又、性能データは、パイプラインストールのようなオペレーションの問題を決定するのにも使用できる。

【００１９】

データ収集サブシステム

図２に示す１つの実施形態では、性能データ収集サブシステム２００は、種々のハードウェアデバイス２２０と、カーナルモードプログラム及びデータ構造体２３０と、ユーザモードプログラム及びデータ構造体２６０とを備えている。プログラム及びデータ２３０及び２６０は、図１のメモリ１２０に記憶することができる。

ハードウェア２２０は、複数の個々のプロセッサ２２１－２２３を含むことができる。各プロセッサには、性能カウンタ又はレジスタ２０１のセットが組み合わされる。レジスタ２０１のセットは、同じ半導体ダイ上に関連プロセッサと共通に常駐することができる。各セット２０１は、複数のレジスタ２３１－２３４を含むことができる。

【００２０】

レジスタ２３１－２３４の各々は、性能事象のカウントを記憶するために多数のビットを含むことができる。１つのカウンタに累積することのできる特定事象の全発生数は、レジスタ２３１－２３４のサイズ即ちビット数に依存する。

特定の実施形態に基づき、レジスタ２０１のセットは、イネーブル、ディスエイブル、休止、再開、リセット（クリア）、読み取り及び書き込みすることができる。典型的に、レジスタは、オーバーフロー（ある特定ビット位置の桁上げ）時に割り込み２５１－２５３を発生する。割り込みに応答して、レジスタ２０１をサンプリング（読み取り）することができる。ある実施形態においては、２の整数累乗に対応する頻度をカウントするために１つのレジスタの特定ビットの設定時に割り込みを発生することができる。

【００２１】

ハードウェア２２０のオペレーション中に、システムの種々のオペレーション特性を表す信号又は「事象」、例えば、（E_1、・・・E_M）２４１－２４４がそれに対応するレジスタを増加させる。信号送信される厳密な事象は、通常は、システムに特有のものである。レジスタを増加できる典型的な事象は、例えば、キャッシュミス、分岐予想ミス、パイプラインストール、命令発生、演算動作、プロセッササイクル、再生トラップ、変換バッファミス、Ｉ／Ｏ要求、プロセスアクティブ、等々を含むことができる。一度に１つ以上の特定の事象数をサンプリングのために選択することができる。

【００２２】

サブシステム２００のカーナルモード成分２３０は、デバイスドライバ又は割り込みハンドラー（プログラム）２３１－２３２と、テーブル２３４－２３５とを含む。各プロセッサごとに、１つのハンドラー及びそれに関連したテーブルがある。効果として、各プロセッサごとに１つのハッシュテーブルをもつと、種々のプロセス間のほとんどの処理活動を同期する必要性が排除される。更に、１つの実施形態において、全てのハンドラー２３１－２３３により共用されるオーバーフローバッファ２３８がある。バッファ２３８へのアクセスは、ロック（Ｌ）２３９により制御される。別の実施形態では、各ハンドラーは、同期する必要のある事象の数を更に減少するために専用バッファを関連させることができる。

【００２３】

カーナルモード成分２３０のオペレーション中に、割り込みがそれに対応するハンドラーをアクチベートする。ハンドラー２３１－２３３は、多数の同時実行プロセス又はスレッドとして動作することができる。ハンドラーは、レジスタから性能データを読み取り、そして以下に詳細に述べるように、それに関連したテーブル２３４－２３６の１つにデータを記憶する。いずれかのテーブルがいっぱいになるか又はアクセス不能になる場合には、オーバーフローデータをオーバーフローバッファに記憶することができる。バッファ２３８へ書き込むためには、プロセスは、先ず、データの一貫性を確保するようにロック２３９を得なければならない。

【００２４】

ユーザモード成分２６０は、デーモンプロセス２６１、１つ以上のユーザバッファ２６２、及び処理された性能データ２５１を含む。データ２５１は、ディスク２５０に記憶することもできるし、又は図１のメモリ１２０の一部分である他の形式の不揮発性記憶装置に記憶することもできる。

オペレーション中に、デーモン２６１は、ハッシュテーブル及びオーバーフローバッファをユーザバッファ２６２へ周期的にフラッシュする（空にする）ことができる。更に、デーモン２６１は、累積された性能データを処理し、例えば、システム１００の性能を分析するのに有用な実行プロファイル、ヒストグラム、及び他の統計学的データを発生するこ

ともできる。

【００２５】

ハッシュテーブル

好ましい実施形態として図３に示すように、テーブル２３４－２３６の各々はハッシュテーブル３００として構成される。ハッシュテーブルは、ハッシュインデックスによりアクセスされるデータ構造体である。典型的に、ハッシュインデックスは、ある範囲のアドレスにわたってデータをランダムに分配する傾向のある決定論的に計算されたあるアドレスである。

【００２６】

ハッシュファンクションは、良く知られている。性能データを収集するためのこの実施形態においては、ハッシュテーブルは、カーナルプロセスとユーザプロセスとの間にデータを転送するのに必要なメモリ帯域巾を減少するのに使用される。より詳細には、以下に述べるように、各ハッシュテーブルは、メモリサブシステムの帯域巾を減少するために多方セット連想キャッシュとして実施することができる。本発明の好ましい実施形態は、ハッシュテーブル３００において４方連想を使用する。

【００２７】

ハッシュテーブル３００は、複数のチャンク３０１－３０３に仕切られる。チャンクは、カーナルモード成分とユーザモード成分との間のデータ転送の単位である。ハッシュテーブル３００の各チャンクは、更に、複数のライン３１１－３１４に仕切られる。ラインとは、通常、システムのメモリ転送ロジック及びキャッシュにより効率的に取り扱われるデータ転送のコヒレントな単位である。

【００２８】

ハッシュテーブル３００の（４方）連想性は、ハードウェアキャッシュライン３１１－３１４のサイズに合致するように入念に選択され、例えば、各キャッシュラインは、６４バイトを含む。又、チャンク３０１－３０３の各々には、ａｃｔｉｖｅ＿ｃｈｕｎｋ及びｆｌｕｓｈ＿ｃｈｕｎｋフラグ３１５及び３１６が組み合わされる。ａｃｔｉｖｅ＿ｃｈｕｎｋフラグ３１５は、ハンドラーの１つがハッシュテーブルのチャンクの１つに記憶されたデータを変更（更新）するときにセットすることができる。フラグがクリアされると、それに対応するチャンクが非アクティブとなり、即ち書き込まれない。ｆｌｕｓｈ＿ｃｈｕｎｋフラグ３１６は、データがハッシュテーブルからユーザバッファにコピーされるときにセットすることができる。テーブル及びバッファの非同期の取り扱いについては、以下に詳細に述べる。

【００２９】

各ラインは、複数のセット連想エントリー３２０を含む。各エントリー３２０は、複数のフィールド３２１－３２４を含む。例えば、フィールド３２１－３２４は、各々、プロセス識別（ｐｉｄ）と、プログラムカウンタ（ｐｃ）と、事象識別（ｅｖｅｎｔ）と、カウンタ値（ｃｏｕｎｔ）を記憶する。ライン３１１－３１４のエントリー３２０に記憶されるデータは、キャッシュミスの数を減少するために高度に圧縮される。連想性をキャッシュラインのサイズに合致することにより、単一キャッシュミスの場合に、エントリーの最大数を探知することができる。サンプルデータを圧縮されたキャッシュラインとして記憶すると、メモリサブシステム１２０の帯域巾及びストレスが減少される。

【００３０】

割り込みハンドラープロセス

図４は、図２のハンドラーのプロセス４００を示す。プロセス４００は、図３のハッシュテーブル３００のラインのエントリー３２０を更新するためのものである。このプロセスは、ステップ４１０において、通常は、割り込みに応答して開始される。ステップ４２０において、ハッシュインデックスH_iを決定する。このインデックスは、ビットを合成するあるハッシュ関数f_{hash}、例えば、pid_i、pc_i及び$event_i$のこのインスタンスの排他的オア、例えば、$H_i = f_{flush}$（pdi_i、pc_i、$event_i$）により決定することができる。ステップ４３０において、テーブル３００におけるインデックス

H_i の全てのエントリーをチェックし、それらがpid_i、pc_i及び$event_i$について合致するかどうか決定する。ヒットの場合には、ステップ４４０において、カウントフィールド３２４を増加し、即ち$count_i = count_i + 1$となる。

【００３１】

ミスの場合、即ちインデックスH_iのエントリーにおいてヒットがない場合には、ステップ４５０において、インデックスH_iのエントリーを図４のオーバーフローバッファ２３８へ移動する。ステップ４６０において、ハッシュテーブルのインデックスH_iに新たなエントリーを記憶し、そしてカウントを１にセットする。ヒット又はミスのいずれの場合にも、プロセスは、ステップ４９０で完了する。

【００３２】

特定の割り込みハンドラーは、グローバルデータの多数の断片をアクセスしなければならない。グローバルデータは、ハンドラーが実行されているプロセッサに対するハッシュテーブルのポインタと、新たなエントリーに対しハッシュ関数の値でインデックスされるハッシュテーブルのラインと、オーバーフローバッファのポインタと、データ構造体の状態を制御するのに使用される多数のグローバル変数（例えば、アクティブなオーバーフローバッファへ挿入するための次のインデックス、及びハッシュテーブルにおいて次のミスに対し所与のラインのどのエントリーを退かせるかを指示するカウンタ）と、その他の種々のグローバル変数とを含む。

【００３３】

このグローバルデータは、全て、データを記憶するのに使用されるハードウェア構造体に合致するために入念にレイアウトしなければならない。例えば、６４バイトのキャッシュラインをもつプロセッサにおいては、データが単一の６４バイト構造体にパックされる。これは、このデータのいずれをアクセスするのにもせいぜい１つのキャッシュミスしか生じないよう確保する。キャッシュミスは、約百サイクル以上で生じ、そして割り込みハンドラーは、その完了のために全体的に数百サイクル以下しか消費してはならないので、キャッシュミスの数を最小にすることは、性能データ収集の影響を最小にするために重要である。

【００３４】

更に、多数のプロセッサ間に共用されたデータを含むキャッシュラインにマルチプロセッサが書き込みするのは、不経済である。グローバルデータは、異なるプロセッサに入念に複製されて、各プロセッサがそれ自身のコピーをもつようにされ、共用キャッシュラインへの書き込みの必要性が回避される。

別の実施形態においては、割り込みを取り扱う時間長さは、特定の状態に対して各々最適化された多数の異なるハンドラーを使用することにより減少することができる。例えば、以下に述べる多数のオーバーフローバッファをもつ１つの実施形態では、割り込みハンドラーが、その開始時に、ハッシュテーブルをバイパスすべきかどうかチェックする。ほとんどの時間、このチェックは、偽となる。

【００３５】

しかしながら、ハッシュテーブルをバイパスすべきかどうか調べるためにフラグをチェックしない形態の割り込みハンドラーを形成することができる。そうではなく、ハンドラーは、ハッシュテーブルをアクセスすべきであると仮定する。ハッシュテーブルをバイパスすべきかどうか指示するためにフラグを変更するときには、適当な割り込みハンドラーを指すようにシステムレベルの割り込みベクトルを変更することができる。これは、通常の場合に、ハッシュテーブルをバイパスすべきでないときに、チェックが必要でないよう確保し、従って、多数のプロセッササイクルがセーブされる。通常の場合に全ての異なるフラグ及びそれらの設定を入念に分析し、そしてその通常の場合に特殊な形態の割り込みハンドラーを使用することにより、著しい高速性能が得られる。

ハッシュテーブルに記憶されたデータを操作する間に他のプロセッサとの同期をとることについて、以下に詳細に説明する。

【００３６】

同期

ハッシュテーブル及びオーバーフローバッファへのアクセスの同期は、次のように管理される。先ず、各プロセッサごとに個別のハッシュテーブル２３４－２３６があり、従って、異なるプロセッサで実行される割り込みハンドラー２３１－２３３は、ハッシュテーブルを操作する間に互いに同期する必要がない。しかしながら、ハンドラーは、オーバーフローバッファ２３８を共用し、従って、オーバーフローバッファへのアクセスは、以下に述べるように同期することが必要となる。

【００３７】

更に、ユーザレベルデーモン２６１がカーナルのテーブル及びオーバーフローバッファからユーザバッファ２６２へ全てのサンプルデータを検索して、デーモン２６１が現在情報を有するよう確保できるようにするために、カーナルモードデバイスドライバにより手順が与えられる。ハッシュテーブル及びオーバーフローバッファから各々データを検索するために「ｆｌｕｓｈ＿ｈａｓｈ」及び「ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗ」と称する個別のルーチンが与えられる。これらのルーチンは、以下に述べるように、ハンドラー２３１－２３３と同期する必要がある。

【００３８】

ハッシュテーブル同期

所与のプロセッサのハッシュテーブルへのアクセスを同期させる必要性のあるアクティビティは、２つあり、そのプロセッサに対する割り込みハンドラーと、ｆｌｕｓｈ＿ｈａｓｈルーチンである。同期される必要のある考えられる事象のタイミングが図５に示されている。割り込みハンドラー及びｆｌｕｓｈ＿ｈａｓｈルーチンにより共用されるグローバルな変数が図６に示されている。割り込みハンドラーの擬似コードが図７に示され、そしてｆｌｕｓｈ＿ｈａｓｈルーチンの擬似コードが図８に示されている。

【００３９】

特定のプロセッサのハッシュテーブルがフラッシュされる間は、割り込みハンドラーがそのテーブルにアクセスすることができない。それ故、この時間中に、割り込みハンドラーは、サンプルデータをオーバーフローバッファ２３８に直接的に記憶する。オーバーフローバッファ２３８に直接書き込まれるサンプルの数を最小にするために、ハッシュテーブルは、そのときのチャンクがフラッシュされる。ハッシュインデックスが、フラッシュされるチャンクに入るサンプルは、オーバーフローバッファに直接書き込まれる。しかしながら、チャンクがフラッシュされるサンプルは、上記のようにハッシュテーブルに書き込まれる。

【００４０】

ａｃｔｉｖｅ＿ｃｈｕｎｋ及びｆｌｕｓｈ＿ｃｈｕｎｋフラグ３１５－３１６は、各々、割り込みハンドラー及びｆｌｕｓｈ＿ｈａｓｈルーチンによりどのチャンクが使用中であるか指示する。各フラグは、チャンクの名前を記録し、チャンクは、その第１エントリーのインデックスにより命名される。値－１は、ａｃｔｉｖｅ＿ｃｈｕｎｋ及びｆｌｕｓｈ＿ｃｈｕｎｋフラグにおいて、チャンクが使用されず、例えば、チャンクが非アクティブであることを指示するのに使用される。

【００４１】

オーバーフローバッファにサンプルデータを記憶するための次の空きスロットを決定し、そしてオーバーフローバッファの空きスロットにエントリーを書き込むための手順は、オーバーフローバッファ２３８へのアクセスを同期する説明の一部分として以下に述べる。データ形式「ｓｌｏｔ＿ｉｎｄｅｘ」についても述べる。

ｆｌｕｓｈ＿ｈａｓｈルーチンの同期は、そのルーチンが多数の高速プロセッサをもつシステムに使用されるよう意図されるので、メモリモデルが順次一貫しない場合には、部分的に繊細である。これについては、「マルチプロセスプログラムを正しく実行するマルチプロセッサコンピュータを構成する方法(How to make a multiprocessor computer that correctly executes multiprocess programs)」、ＩＥＥＥトランザクション・オン・コンピュータ、Ｃ－２８（１９７９年、９月）、第６９０－６９１ページを参照されたい。

【００４２】
オペレーションの順序がある条件のもとでしか保証されない場合には、メモリモデルがシーケンシャルに一貫しない。プロセッサは、第１順序で命令を発生しそして実行できるが、メモリサブシステムは、第２順序（メモリ順序）でアクセスを完了する。更に、メモリ順序は、推移的であり、ＡがＢの前に生じそしてＢがＣの前に生じる場合には、ＡがＣの前に生じる。オペレーションが実際に生じる順序は、次の制約に基づきメモリ順序によって決定される。
１．単一プロセッサによる単一メモリ位置でのメモリアクセスオペレーションは、アクセス命令がそのプロセッサにより発生される順序で生じる。
２．単一プロセッサによる異なるメモリ位置でのメモリアクセスオペレーションは、それらオペレーションが「メモリバリア」（ＭＢ）命令により分離されない限り、いかなる順序で発生することもできる。この場合に、メモリバリア命令の前の全てのオペレーションは、メモリバリア命令の後の全てのオペレーションの前に生じる。
３．２つのプロセッサが同じメモリ位置を１つのプロセッサ読み取りによりアクセスし、そして他のプロセッサが書き込みし、更に、読み取りオペレーションがその書き込まれる値を通知する場合には、その読み取りオペレーションが書き込みオペレーションの後に生じる。
【００４３】
メモリバリア命令は、比較的長い実行時間を必要とすることに注意されたい。それ故、頻繁に横断される実行経路におけるＭＢ命令の使用を最小にすることが望ましい。
図７に示す割り込みハンドラーにとられる特定の経路に基づき、３つのケースが考えられる。ハンドラーは、ライン７０４、７０７又はライン７０９－７２２で始まるコードを実行することができる。これらケースの各々は、以下に説明する。
【００４４】
先ず、ハンドラーが、ライン７０４で始まるコードを実行する場合には、ハンドラーは、ハッシュテーブルにアクセスする必要が全くない。サンプルは、オーバーフローバッファ２３８に直接書き込まれ、このバッファは、以下に述べるように、サンプルの損失又は二重カウントを回避するために同期される。この場合に、たとえ変数ｆｌｕｓｈ＿ｃｈｕｎｋ〔ｉ〕がｆｌｕｓｈ＿ｈａｓｈルーチン（異なるプロセッサで実行される）によりｃ以外の値にセットされた場合でも、ハンドラーは、ライン７０４を実行するためにライン７０３において判断することが考えられる。
【００４５】
このため、図１のメモリサブシステム１２０は、変数ｆｌｕｓｈ＿ｃｈｕｎｋ〔ｉ〕の新たな値が全ての他のプロセッサ上に直ちに見えるように保証しない。これは、誤った振る舞いを生じるものではなく、幾つかのサンプルがハッシュテーブルに記憶できたであろうときにそれらがオーバーフローバッファに書き込まれることを意味するに過ぎない。従って、性能は若干低下するが、データの正しさは、影響がない。
第２に、ハンドラーがライン７０７のコードを実行する場合には、高価な同期オペレーションは、実行されない。ハンドラーがライン７０７にあるときには、ハッシュテーブルの一致するエントリーが見つかり、これは、そのエントリーに対してカウントを増加する必要があることだけを意味する。しかしながら、この場合に、非常に稀であるが、単一のサンプルが失われるおそれがあることに注意されたい。
【００４６】
図５は、ハッシュテーブルアクセスの考えられるタイミングを示す。図５において、ライン５６０は、左から右へ増加する全時間を示し、ライン５７０は、１つのプロセッサ（ｃｐｕ　ｊ）の割り込みハンドラー５３１の事象のタイミングを示し、そしてライン５８０は、別のプロセッサ（ｃｐｕ　ｉ）５３２のｆｌｕｓｈ＿ｈａｓｈルーチンの事象のタイミングを示す。
割り込みハンドラー（ｃｐｕ　ｊ）５３１は、多数の動作を実行し、即ちｆｌｕｓｈ＿ｃｈｕｎｋフラグを読み取り（事象Ｃ　５７１）、ハッシュテーブルにおいてヒットを見出

し、ハッシュテーブルにおいて一致するエントリーのカウントを読み取り（事象Ｄ　５７２）、そして増加されたカウントをハッシュテーブルに書き込む（事象Ｅ　５７３）。

【００４７】

ｆｌｕｓｈ＿ｈａｓｈルーチンがオーバーフローバッファ２３８をフラッシュするように同時に実行されない場合には、増加されたカウントが正しく書き戻される。しかしながら、たとえ割り込みハンドラーがａｃｔｉｖｅ＿ｃｈｕｎｋフラグ３１５をチェックし、そしてそれが必要とされるチャンクに対してセットされたものでないと決定されても、ｆｌｕｓｈ＿ｈａｓｈルーチンが実際にそのチャンクを同時にフラッシュすることが考えられる。この稀なケースにおいては、以下に述べるように、タイミングの入念な分析により、単一のサンプルが失われることがあるが、サンプルが二重カウントされることはないことが示される。

【００４８】

図５において、ｆｌｕｓｈ＿ｈａｓｈルーチンに対して次の事象が生じる。ｆｌｕｓｈ＿ｃｈｕｎｋフラグ３１６は、チャンクが次にフラッシュされるようにセットされる（事象Ａ　５６１）。次いで、ルーチンは、割り込みハンドラーにより使用されるハッシュテーブルエントリーをユーザバッファへコピーする（事象Ｇ　５８３）。次いで、ルーチンは、ハッシュテーブルのエントリーをゼロにし（事象Ｈ　５８４）、そしてｆｌｕｓｈ＿ｃｈｕｎｋフラグ３１６をクリアすることにより完了を指示する（事象Ｉ　５８５）。

【００４９】

２つの他の事象に対する時間、即ちｆｌｕｓｈ＿ｃｈｕｎｋフラグの更新値が他の全てのプロセッサに伝播されるよう保証される（事象Ｂ　５８２）時間と、割り込みハンドラーによりハッシュテーブルに書き戻される増加カウントが至る所に伝播されるように保証される（事象Ｆ　５７４）時間とが図５に示されている。これらの時間は、特定のプロセッサ実施形態に基づくもので、アーキテクチャー仕様により予め決定されるものではないことに注意されたい。

【００５０】

事象Ｅ　５７３が事象Ｇ　５８３の前に発生する（メモリ順序で）場合には、増加カウントが事象Ｇ　５８３の時間にユーザバッファにコピーされ、そして事象Ｈ　５８４の時間にカウントがゼロにリセットされる。このケースにおいて、サンプルは、厳密に一度だけカウントされる。これは、稀である。

事象Ｅ　５７３が事象Ｈ　５８４の後に生じる場合には、エントリーのカウントがゼロにセットされた後に増加カウントがハッシュテーブルに書き戻される。これは、このエントリーの元のカウントで表されたサンプルが、事象Ｇの時間にユーザバッファへコピーされることにより一度、そしてこのエントリーがハッシュテーブルから追い立てられ即ちフラッシュされる次のときにもう一度、の２回カウントされるようにする。これは、受け入れられない。というのは、単一のハッシュテーブルエントリーは、多数のサンプルを表す大きなカウントをもつことができるからである。

【００５１】

二重カウントは、事象Ｅ　５７３が事象Ｈ　５８４の前に生じる限り起こり得ない。これは、次の変数について以下に述べる制約により保証される。

記憶された値を全てのプロセッサに伝播するのに必要な最大時間を「ｍａｘ＿ｐｒｏｐ」とする。図７のライン７０７を実行するときに割り込みルーチンに対する最大時間を「ｍａｘ＿ｉｎｔｒ」とする。同じエントリーに対し事象Ａ（５６１）から事象Ｈ（４８４）までの最小時間（即ち、ｆｌｕｓｈ＿ｃｈｕｎｋフラグ３１６がセットされるときから、時間フラグ３１６がｆｌｕｓｈ＿ｈａｓｈルーチンによってクリアされるまでの最小時間）を「ｍｉｎ＿ｆｌｕｓｈ」とする。

【００５２】

次の制約は、事象Ｅが事象Ｈの前に生じるように確保する。

（ｍａｘ＿ｉｎｔｒ＋（２＊ｍａｘ＿ｐｒｏｐ））＜ｍｉｎ＿ｆｌｕｓｈ

特定のプロセッサ実施形態におけるタイミングは、ｍａｘ＿ｐｒｏｐ及びｍａｘ＿ｉｎｔ

ｒを決定するように測定することができる。次いで、チャンクサイズは、ｍｉｎ＿ｆｌｕｓｈが充分大きくなるよう確保するに充分な大きさに選択することができる。

【００５３】

第３に、ハンドラーがライン７０９－７２２を実行する場合には、ハッシュテーブルからオーバーフローバッファへエントリーを移動し、そしてカウント１の新たなエントリーをハッシュテーブルに書き込まねばならない。

ハッシュテーブルからオーバーフローバッファへ移動されるエントリーの損失又は二重カウントを回避するために、２つのフラグ（ａｃｔｉｖｅ＿ｃｈｕｎｋ〔ｉ〕及びｆｌｕｓｈ＿ｃｈｕｎｋ〔ｉ〕）との入念な同期と、３つまでのメモリバリアオペレーションとが使用される。この同期は比較的高価であるが、これは、ハッシュテーブルにミスがあるときしか生じず、これは、比較的稀である。

【００５４】

このコードの重要な特性は、ライン７１９及び７２０がライン７０７と相互に排他的なことである。このアルゴリズムは、メモリバリア命令と共に使用される標準的な相互除外技術の変形である。この技術は、オペレーションを強制的に順序付けし、そして割り込みハンドラーがロックを待機しないように確保する。

ｆｌｕｓｈ＿ｈａｓｈルーチンが所望のチャンクで行われるまで待機するのではなく、割り込みハンドラーは、所望のチャンクが得られないときにライン７１４－７１６のハッシュテーブルを単にバイパスする。又、オーバーフローバッファがいっぱいになる非常に稀な事象においては、割り込みハンドラーが単に復帰し、サンプルを破棄する。その結果、１つの効果として、この場合には、ハンドラーがライン７０９－７２２を実行するときに、サンプルが失われることも二重カウントされることもない。

【００５５】

この解決策の正味の作用は、ハッシュテーブルにおいてヒットする通常の場合に、ハンドラーは、高価な同期オペレーションを実行せず、非常に稀なケースにおいてサンプルを失う。

ライン７１０、７１４及び７１８を最適化することができ、ライン７１１においてオーバーフローバッファにスロットを得るには、ロックを得そして解除することが必要となる。ロックを得るには、それを解除するときと同様に、メモリバッファを伴い、従って、ライン７１０のメモリバッファは排除できる。又、ロック解除がライン７１３の条件テストの後に移動される（即ち、ロック解除がライン７１４及び７１８で行われる）場合にも、ライン７１４及び７１８のメモリバリアを排除することができる。

【００５６】

オーバーフローバッファの同期

好ましい実施形態においては、オーバーフローバッファ２３８は、実際には、２つの部分に分割され、一方の部分がハンドラーによりアクセスされる間に他方の部分をフラッシュできるようにされる。この形式の技術は、「二重バッファリング」とも称される。更に、オーバーフローバッファ２３８のロック２３９は、バッファのポインタが操作される間にのみ保持され、エントリーがそれに書き込まれる間には保持されない。これは、ハンドラーがエントリーを同じオーバーフローバッファに並列に書き込みできるようにすることにより効率を改善する。

【００５７】

明瞭化のために、上記のオーバーフローバッファ２３８の説明は、あたかもそれが単一バッファであるかのように簡単化され、そしてスロットは、形式ｓｌｏｔ＿ｉｎｄｅｘの値により識別される。ここでは、バッファ及びｓｌｏｔ＿ｉｎｄｅｘがいかに表されるかを詳細に説明する。

２つのバッファが存在する。バッファに対するインデックス（ｓｌｏｔ＿ｉｎｄｅｘ）は、バッファｉｄ（０又は１）と、そのバッファのスロットのインデックスとで構成される。ハンドラー及びｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗにより共用されるグローバル変数が図９に示されており、次の空きスロットを決定しそしてオーバーフローバッファにエントリー

を書き込むためにハンドラーにより使用される手順が図１０及び１１に示されており、そしてｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンが図１２に示されている。

【００５８】

ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンは、単一のオーバーフローバッファをフラッシュする。満杯のバッファが読み取りを待機する場合には、それがフラッシュされ、さもなくば、現在部分的に満杯のバッファがフラッシュされる一方、オーバーフローが他のバッファへ送られる。

単一のロック（ｏｖｅｒｆｌｏｗ＿ｌｏｃｋ）は、図９の変数へのアクセスを同期し、そしてバッファ（インデックス、完了、ｃｕｒｒｅｎｔ＿ｏｖｅｒｆｌｏｗ及びｆｕｌｌ＿ｏｖｅｒｆｌｏｗ）を管理するのに使用される。これら変数に対する全ての更新は、ｏｖｅｒｆｌｏｗ＿ｌｏｃｋを保持する間に行われる。バッファｉについては、ｉｎｄｅｘ〔ｉ〕は、書き込まれるべき次のスロットのインデックスである。

【００５９】

エントリーは、ロックを保持せずにオーバーフローバッファに書き込まれ、むしろ、ｉｎｄｅｘ〔ｉ〕を増加することによりバッファｉにスロットが指定される。スロットを指定したプロセッサのみが、それに書き込むことが許される。プロセッサは、スロットへの書き込みを行うと、図１１について説明するように、ｃｏｍｐｌｅｔｅｄ〔ｉ〕を増加する。従って、スロットは、いかなる順序で書き込むこともできる（特定の順序で指定されたが）。

【００６０】

ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンのワイル(while) ループは、ｃｏｍｐｌｅｔｅｄ〔ｉ〕がｉｎｄｅｘ〔ｉ〕に等しくなるまで待機する。これは、指定された全てのスロットへの書き込みが完了したことを意味する。ｉｎｄｅｘ〔ｆｕｌｌ＿ｏｖｅｒｆｌｏｗ〕は、ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンがワイルループにある間は変更できないことに注意されたい。というのは、満杯のオーバーフローバッファに対してスロットが指定されないからである。又、ｃｏｍｐｌｅｔｅｄ〔ｆｕｌｌ＿ｏｖｅｒｆｌｏｗ〕は、単調に増加され、即ち読み取りオペレーションは、原子的なものであるから、ロックを保持せずにｃｏｍｐｌｅｔｅｄ〔ｆｕｌｌ＿ｏｖｅｒｆｌｏｗ〕を読み取ることができ、ｉｎｄｅｘ〔ｆｕｌｌ＿ｏｖｅｒｆｌｏｗ〕に等しい値が見えたときから、指定のスロットへの全ての書き込みを完了しなければならない。

【００６１】

図８のライン８０５のワイルループの後のメモリバリアは、満杯のオーバーフローバッファをユーザバッファにコピーするのに伴うメモリオペレーションが、ｃｏｍｐｌｅｔｅｄ〔ｆｕｌｌ＿ｏｖｅｒｆｌｏｗ〕をｉｎｄｅｘ〔ｆｕｌｌ＿ｏｖｅｒｆｌｏｗ〕に比較するテストの後に、換言すれば、バッファへの書き込みが実際に完了した後に生じるよう確保するために必要となる。

【００６２】

上記技術においては、全てのプロセッサが共通バッファへと退去し、これは、各退去のたびにプロセッサ間にある程度の整合を必要とする。実際には、一対のオーバーフローバッファがあって、その一方が満杯で、ユーザスペースへフラッシュされねばならないときに、その他方のバッファへ退去できるようにする。事象のタイミングを入念に分析することに基づく付加的な同期は、ハッシュテーブルへのアクセスを適切に同期させるよう確保する。

【００６３】

プロセッサごとのローカルオーバーフローバッファ

別の実施形態においては、プロセッサごとのハッシュテーブル及び単一の共用のオーバーフローバッファを伴う上記方法を、各プロセッサごとに付加的な小さなオーバーフローバッファをもつように拡張することができる。この方法では、割り込みハンドラーがオーバーフローバッファに書き込もうとするとき、先ず、そのローカルオーバーフローバッファをチェックする。そのバッファがいっぱいでなければ、単にそのバッファに書き込むだけ

である。そのバッファがいっぱいであれば、共用オーバーフローバッファにロックを得、そしてその全ローカルバッファを共用バッファにコピーする。これは、ロックを得る頻度、及び共用キャッシュラインに書き込む頻度を減少し、従って、マルチプロセッサの性能を改善する。

【００６４】

プロセッサごとのローカルオーバーフローバッファを伴うこの方法の変形として、ハッシュテーブルを完全に排除することによりこの方法が更に変更される。これは、上記した他の方法よりも高いメモリトラフィックを有するが、共用のオーバーフローバッファへの書き込みが依然として低い頻度であるので、マルチプロセッサの共用メモリへのアクセス及びロックのためのオーバーヘッドは依然として小さい。

【００６５】

多数のオーバーフローバッファ

別の実施形態においては、異なる同期技術を使用し、全てのプロセッサに共用される単一の二重バッファではなくて、付加的なオーバーフローバッファ、即ちプロセッサ当たり２つのバッファを使用することにより、ハッシュテーブル同期及びオーバーフローバッファアクセスのコストを低減することができる。

この技術では、各プロセッサは、専用のハッシュテーブル（前記のような）と一対の専用の（二重）オーバーフローバッファとを「所有」する。この場合に、「所有」とは、アクティブなバッファ及びハッシュテーブルの状態に対する全ての変更（１つを除く）がそのプロセッサにより行われ、上記の単一の二重バッファ技術に存在するメモリ同期オペレーションの多くを排除することを意味する。

【００６６】

第１の技術に対し２つの主たる変更がある。第１に、ハッシュテーブルをユーザスペースへとフラッシュする間に、性能カウンタ事象がアクティブなオーバーフローバッファに付加され、ハッシュテーブルがバイパスされる。これは、フラッシュオペレーション中にハッシュテーブルが変更されないよう確保するために必要な同期を排除する。第１の技術の場合と同様に、ハッシュテーブルは、チャンクにおいてフラッシュすることができ、事象をオーバーフローバッファに直接付加しなければならない頻度を減少する。

【００６７】

第２に、各プロセッサは、専用の一対のオーバーフローバッファを有し、全てのプロセッサにわたって単一のアクティブなオーバーフローバッファを共用するのに必要な同期が排除される。

以下のデータ状態は、プロセッサごとに維持される。

(A) buffer overflow[ncpus][2]	プロセッサ当たりの一対のオーバーブローバッファ
(B) int: index[ncpus][2]	書き込むべき次のスロットのインデックス
(C) int: active[ncpus]	アクティブなオーバーフローバッファのインデックス
(D) bool: allow __flip[ncpus]	フラグ：バッファをフリップしてもよいか？
(E) bool: bypass__hash[ncpus]	フラグ：ハッシュテーブルをバイパスするか？

【００６８】

ハッシュテーブルの同期

特定のプロセッサに対してハッシュテーブルをアクセスする２つのアクティビティがある。
１）割り込みハンドラーは、新たなサンプルをハッシュテーブルに記憶する；そして
２）ｆｌｕｓｈ＿ｈａｓｈルーチンは、ハッシュテーブルをユーザスペースにコピーする。
図１３に示す割り込み中にオーバーフローバッファ事象を取り扱うための擬似コードにおいては、「ｂｙｐａｓｓ＿ｈａｓｈ」変数を用いて、ハッシュテーブルへのアクセスが制御される。ライン１３０１－１３０３は、この変数が「真」にセットされた場合に、割り込みハンドラーがハッシュテーブルを完全にスキップし、そして新たなサンプルをオーバーフローバッファに直接書き込む。

【００６９】
ライン１３０５－１３１０は、割り込みハンドラーを通る他の経路を示す。新たなサンプルがハッシュテーブルのエントリーの１つに一致する場合は（ライン１３０５－１３０６）、ハンドラーは、その一致するエントリーに関連したカウントを単に増加する。さもなくば（ライン１３０８－１３１０）、ハンドラーは既存のエントリーの１つを退去のために取り上げる。このエントリーは、ハッシュテーブルから除去され、そしてオーバーフローバッファへ書き込まれる。新たなサンプルは、ハッシュテーブルの空にスロットに書き込まれる。

【００７０】
「ｆｌｕｓｈ＿ｈａｓｈ」ルーチンの擬似コードが図１４に示されている。各プロセスに対し、このルーチンは、「ｂｙｐａｓｓ＿ｈａｓｈ〔ｃｐｕ〕」フラグを真にセットし（ライン１４０４－１４０８）、ハッシュテーブルをユーザスペースにコピーし、そして「ｂｙｐａｓｓ＿ｈａｓｈ〔ｃｐｕ〕」フラグを偽にリセットする（ライン１４０６及び１４１０）。正しい同期のために、「ｆｌｕｓｈ＿ｈａｓｈ」ルーチンは、「ｃｐｕ」という番号のプロセッサにおいて「ｂｙｐａｓｓ＿ｈａｓｈ〔ｃｐｕ〕」への変更を入念に実行する。

【００７１】
「ｆｌｕｓｈ＿ｈａｓｈ」ルーチンが実行されているプロセッサが、ハッシュテーブルがコピーされるプロセッサと同じである（ライン１４０３－１４０６）場合には、このルーチンは、ローカルオペレーションにより「ｂｙｐａｓｓ＿ｈａｓｈ〔ｃｐｕ〕」フラグを単にセットしそしてクリアする。さもなくば、「ｆｌｕｓｈ＿ｈａｓｈ」ルーチンは、プロセッサ間割り込みを使用して、「ｂｙｐａｓｓ＿ｈａｓｈ〔ｃｐｕ〕」への変更を「ｃｐｕ」において実行させる。

【００７２】
割り込みハンドラー及びｆｌｕｓｈ＿ｈａｓｈルーチンは、正しく同期する。というのは、「ｂｙｐａｓｓ＿ｈａｓｈ〔ｃｐｕ〕」が読み取られそして「ｃｐｕ」という番号のプロセッサのみに書き込まれるからである。プロセッサ間割り込みは、性能カウンタのオーバーフローに対し割り込みハンドラーと同じ割り込み優先順位レベルで実行するようにセットされ、それらが互いに原子的に実行するよう確保する。メッセージを送信するような他の通信メカニズムも、同じ原子的レベルが確保される限り、プロセッサ間割り込みに代わって使用できる。

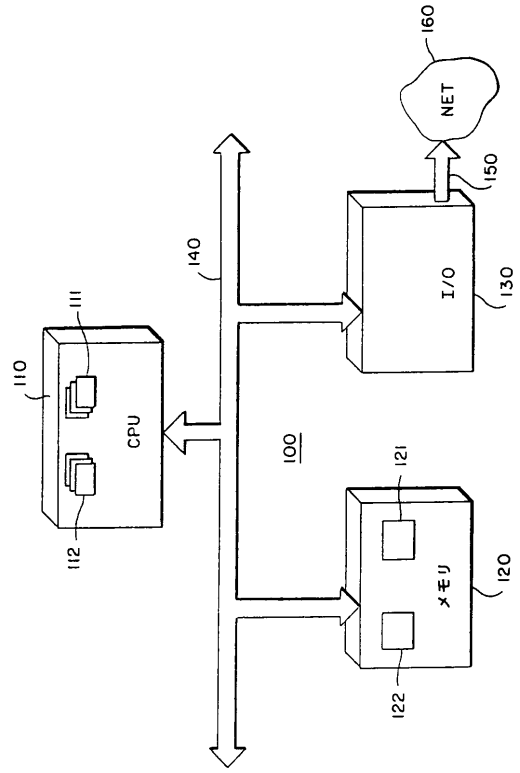
【００７３】
オーバーフローバッファの同期
特定のプロセッサに関連した一対のオーバーフローバッファを使用する２つのアクティビティがある。第１に、割り込みハンドラーは、エントリーをオーバーフローバッファに時々書き込む。第２に、ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンは、オーバーフローバッファの内容をユーザスペースへ周期的にコピーする。これら２つのアクティビティは、バッファの一方を「アクティブ」バッファとしてマークすることにより同期される。割り込みハンドラーは、「アクティブ」バッファへ書き込むことだけが許され、そしてｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンは、非アクティブバッファから読み取ることだけが許される

。
【００７４】
アクティブバッファがいっぱいであるときは、割り込みハンドラーは、古いアクティブバッファを非アクティブとマークしそして古い非アクティブバッファをアクティブとマークすることによりバッファをスイッチ（フリップ）するように試みる。フラグ「ａｌｌｏｗ＿ｆｌｉｐ〔ｃｐｕ〕」は、「ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗ」が古い非アクティブバッファをユーザスペースにコピーする間にこの切り換えが起きるのを防止するのに使用される。
【００７５】
特定のプロセッサに対しオーバーフローバッファにエントリーを追加するルーチンが図１５に示されている。アクティブバッファがいっぱいでない（ライン１５０２）場合には、このルーチンは、サンプルをアクティブバッファに単に付加するだけである。アクティブバッファがいっぱいである場合には、このルーチンは、バッファをフリップ（切り換える）ように試みる。
【００７６】
このルーチンは、先ず、フリップが許されたかどうか決定するようにチェックする（ライン１５０５）。フリップが許された場合は、バッファをフリップし、そして新たなアクティブバッファから全てのサンプルをドロップすることにより新たなアクティブバッファを書き込みのために準備する（ライン１５０８）。バッファがフリップされた後に、ルーチンは、満杯の非アクティブバッファを読み出すようにユーザレベルデーモンに通知する（ライン１５０７）。新たなサンプルが新たなアクティブバッファに追加される。フリップが許されない場合には、ルーチンが新たなサンプルをドロップし、そして復帰する（ライン１５１１）。
【００７７】
ｗｒｉｔｅ＿ｔｏ＿ｏｖｅｒｆｌｏｗルーチンは、２つのケースにおいてサンプルをドロップする。第１のケースでは、フリップ後に、新たなアクティブバッファの読み取られなかったサンプルがドロップされる（ライン１５０５）。サンプルが実際にドロップされるのは非常に稀である。というのは、これらサンプルは、最後のフリップのとき以来、ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンが読み取りのために得ることができ、そしてフリップは、非常に低い頻度で生じるからである。
【００７８】
第２のケースは、アクティブバッファがいっぱいであって、フリップが許されないときである。このケースも、非常に稀である。フリップは、ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンが非アクティブなバッファを読み出すときだけ許可されない。非アクティブなバッファは、最後のフリップのときに読み取りの準備がなされ、そしてこのシステムではフリップが生じる頻度が非常に低いので、ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンが非アクティブなバッファを依然コピーすることはほとんどあり得ない。前記ケースの両方において、いずれかのサンプルがドロップした場合には、ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗルーチンが、オーバーフローバッファの充満に応答して、ユーザレベルデーモンにより充分迅速にコールされないという指示である。
【００７９】
「ｆｌｕｓｈ＿ｏｖｅｒｆｌｏｗ」ルーチンの擬似コードが図１６に示されている。このルーチンは、非アクティブなバッファを特定のプロセッサのユーザスペースへコピーする。このルーチンは、「ａｌｌｏｗ＿ｆｌｉｐ〔ｃｐｕ〕」フラグを使用して、非アクティブなバッファがユーザスペースへコピーされる間にｗｒｉｔｅ＿ｔｏ＿ｏｖｅｒｆｌｏｗルーチンが非アクティブなバッファをアクセスするのを防止する。上記のｆｌｕｓｈ＿ｈａｓｈルーチンの場合と同様に、「ａｌｌｏｗ＿ｆｌｉｐ〔ｃｐｕ〕」への全てのアクセスが、「ｃｐｕ」という番号のプロセッサにおいて生じ、それ故、正しく同期されるように確保するために、プロセッサ間割り込みが使用される。
【００８０】

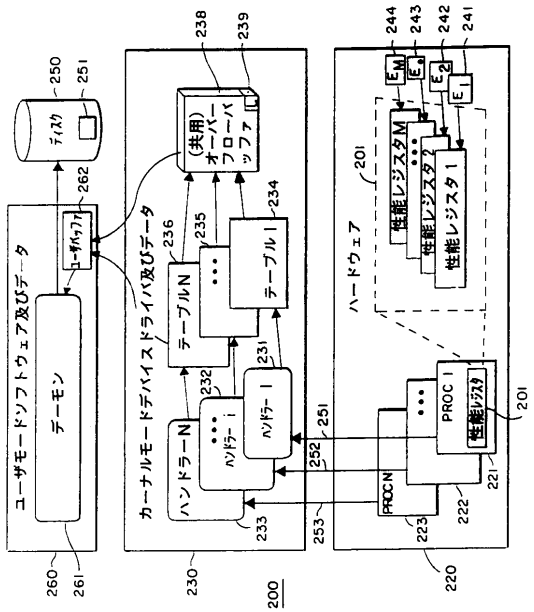
収集される情報
性能カウンタの厳密な実施に基づき、選択された命令に対してプログラムカウンタ値をサンプリングすることができる。更に、汎用レジスタを特定するオペランドを有するメモリアクセス（ロード及び記憶）及びジャンプ命令に対して、ベースアドレスも収集することができる。
【００８１】
以上に述べた性能監視システムは、カーナルソフトウェア、入力／出力デバイスドライバ、アプリケーションプログラム及び共用ライブラリーを含むコンピュータシステムのオペレーションの多くの観点で性能データを収集することができる。これは、システムの通常のオペレーションを不当に妨げることなく非常に高いレートでサンプリング割り込みを発生しそして処理することによって達成される。
当業者であれば、本発明の範囲から逸脱せずに本発明に種々の変更がなされ得ることが明らかであろう。
【図面の簡単な説明】
【図１】本発明の好ましい実施形態による性能監視サブシステムによって性能データを収集することのできるコンピュータシステムのブロック図である。
【図２】データ収集サブシステムのブロック図である。
【図３】収集された性能データを記憶するためのハッシュテーブルを示すブロック図である。
【図４】図３のハッシュテーブルを更新するための流れ線図である。
【図５】ハッシュテーブルの非同期更新を示すタイミング図である。
【図６】割り込みハンドラー及びハッシュテーブルフラッシュルーチンにより共用される変数を示す図である。
【図７】割り込みハンドラールーチンの擬似コードを示す図である。
【図８】ハッシュテーブルをフラッシュするための擬似コードを示す図である。
【図９】共用同期変数を示す図である。
【図１０】オーバーフローバッファに空きスロットを得るための擬似コードを示す図である。
【図１１】オーバーフローバッファの空きスロットにエントリーを書き込むための擬似コードを示す図である。
【図１２】オーバーフローバッファをフラッシュするための擬似コードを示す図である。
【図１３】サンプル事象割り込みの間にオーバーフローバッファ事象を取り扱うための擬似コードを示す図である。
【図１４】ハッシュテーブルをフラッシュするための擬似コードを示す図である。
【図１５】オーバーフローバッファへエントリーするルーチンのための擬似コードを示す図である。
【図１６】オーバーフローバッファをユーザバッファへフラッシュするルーチンのための擬似コードを示す図である。
【符号の説明】
１００　コンピュータシステム
１１０　プロセッサ
１１２　性能カウンタ
１２０　メモリサブシステム
１２１　プログラム
１２２　データ
１３０　入力／出力インターフェイス
１４０　バス
１６０　ネットワーク
２００　性能データ収集サブシステム
２０１　レジスタ

２２０　ハードウェアデバイス
２２１－２２３　個々のプロセッサ
２３０　カーナルモードプログラム及びデータ構造体
２３１－２３２　割り込みハンドラー
２３４－２３５　テーブル
２３８　オーバーフローバッファ
２３９　ロック
２５０　ディスク
２５１　性能データ
２６０　ユーザモードプログラム及びデータ構造体
２６１　デーモンプロセス
２６２　ユーザバッファ
３００　ハッシュテーブル
３０１－３０３　チャンク

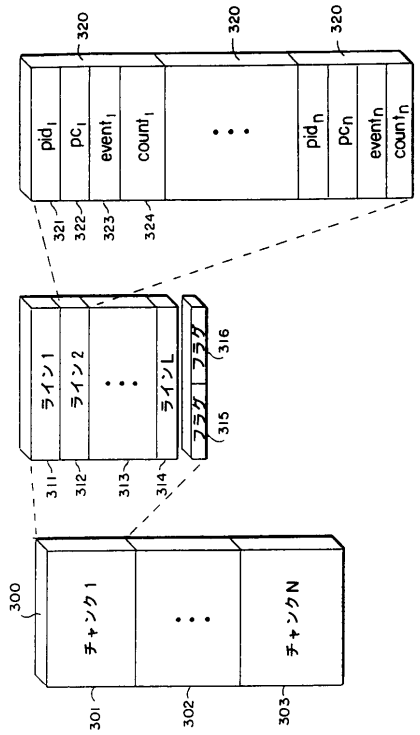
【図１】



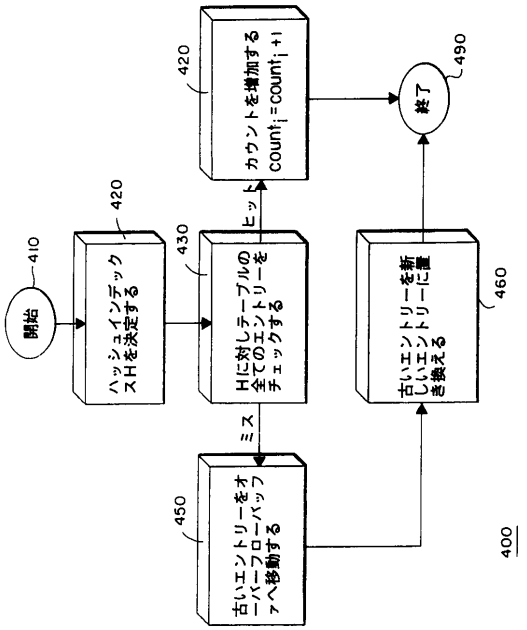
【図２】



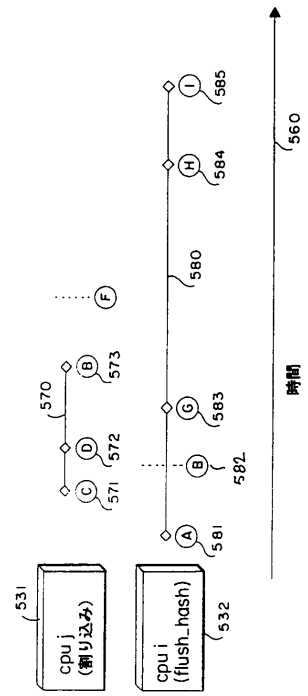
【図３】



【図４】



【図５】



【図６】

```
610  int ncpus                    マシンにおけるＣＰＵの数
620  hash_table hash[ncpus]       各ＣＰＵのハッシュテーブル
630  int flush_chunk[ncpus]       各ＣＰＵでフラッシュされるチャンク
640  int active_chunk[ncpus]      ＣＰＵハンドラーでアクセスされるチャンク
```

【図７】

```
701. int i = ... 現在ＣＰＵ... ;
702. int c = ... 使用されるべきチャンク... ;
703. if (flush_chunk[i] == c) {
704.         ... 所望のチャンクがフラッシュされる ;
                  オーバーフローバッファの次の空きスロットを決定し、
                  そのスロットにサンプルを直接書き込み、そして復帰
                  する ;
                  空きスロットがない場合には、サンプルを破棄し、
                  そして復帰...
705. }
706. もし... ハッシュヒットの場合は... 次いで
707.  ... ハッシュテーブルのエントリーに対しカウントを増加する...
708.  さもなくば
709.         active_chunk[i] = c;
710.         memory_barrier();
711.         slot_index s = ... オーバーフローバッファの次の空きスロット...
712.         もし... 空きスロットがなければ... 次いで復帰する ;
713.     もし (flush_chunk[i] == c) であれば、次いで
714.                 memory_barrier();
715.                 active_chunk[i] = -1;
716.                  ... オーバーフローバッファのスロットにサンプルを
                        書き込む...
717.  さもなくば
718.         memory_barrier();
719.                 ... オーバーフローバッファのスロットに古いエントリーを書き込む...
720.                 ... ハッシュテーブルに新しいエントリーを書き込む...
721.         memory_barrier();
722.         active_chunk[i] = -1;
```

【図８】

```
801.  for (i = 0; i < ncpus; i++) {
802.          for (c = 0; c < table_size; c+=chunk_size) {
803.                  flush_chunk[i] = c;
804.                  memory_barrier();
805.                  while active_chunk[i] = c do {}
806.                  memory_barrier();
807                   ... cpuiにおけるコピー及びゼロチャンクc...
808.                  memory_barrier();
809.                  flush_chunk[i] = -1;
810          }
811   }
```

【図９】

バッファグローバル変数		意味
910	lock overflow_lock	
920	buffer overflow[2]	一対のオーバーフローバッファ
930	int index[2] = 0	所与のバッファに書き込むための次のスロットのインデックス
940	int completed[2] = 0	所与のバッファに対して完了した書き込みの数
950	int current_overflow = 0	新たなエントリーを書き込むための現在バッファ
960	int full_overflow = -1;	満杯バッファ (又は－１の場合は非満杯)

【図１０】

```
acquire_lock(overflow_lock);
co = current_overflow;
i = index[co];
index[co] = i + 1;
if (i == ... バッファサイズ... － 1 )  {
        /* buffer full; try to make other buffer current */
        if full_overflow = -1 {
         /* other buffer is empty; make it current */
                full_overflow = co;
                current_overflow = ! co;
                ... 現在満杯のバッファをフラッシュするように
                    デーモンに通知する...
        } else { /* both buffers are full; drop this sample */
          index[co] = I;
          ... 空きスロットが使用できないという指示を戻す...
        }
}
release_lock(overflow_lock);
... return the slot_index (co, I) ...
```

【図１１】

```
overflow[co][j] = ... 書き込まれるべきエントリー... ;
acquire_lock(overflow_lock);
completed[co] = completed[co] + 1;
release_lock(overflow_lock);
```

【図１２】

```
acquire_lock(overflow_lock);
if (full_overflow == -1) {
        full_overflow = current_overflow;
        current_overflow = ! current_overflow
        }
release_lock(overflow_lock);
while (completed[full_overflow] < index[full_overflow) {}
memory_barrier();
        ... overflow[full_overflow]をユーザバッファへコピーする...
acquire_lock(overflow_lock);
completed[full_overflow] = 0;
index[full_overflow] = 0;
full_overflow = -1;
release_lock(overflow_lock);
```

【図１３】

```
1301.   interrupt(cpu, sample) {
1302.   if (bypass_hash[cpu]) {
1303.       write_to_overflow(cpu, sample)
1304.     } else {
1305.       if (... hash hit ...) {
1306.         ハッシュテーブルのエントリーのカウントを
              増加する
1307.       } else {
1308.         old := pick old entry to evict from hash table
1309.         ハッシュテーブル新たなサンプルを記憶する
1310.         write_to_overflow(cpu, old)
1311.       }
1312.     }
1313.   }
```

【図１４】

```
1401.     flush_hash() {
1402.       for each cpu {
1403.         if (cpu == current cpu) {
1404.           bypass_hash[cpu] := true
1405.           copy hash_tables[cpu] to user space
1406.           bypass_hash[cpu] := false
1407.         } else {
1408.           interprocessor_interrupt(cpu) {
          bypass_hash[cpu] := true }
1409.           copy hash_tables[cpu] to user space
1410.           interprocessor_interrupt(cpu) {
            bypass_hash[cpu] := false }
1411.         }
1412.       }
1413.     }
```

【図１５】

```
1501.   write_to_overflow(cpu, sample) {
1502.     if (active buffer is not full) {
1503.       アクティブバッファにサンプルを付加し、
            インデックスを増加する
1504.     } else {
1505.       if (allow_flip[cpu]) {
1506.         active[cpu] := 1 - active[cpu]
1507.         ユーザデーモンをウェイクアップし、満杯の
            バッファを読み取る
1508.         新たなアクティブバッファから全てのサンプルを
            ドロップする
1509.         アクティブバッファにサンプルを付加し、
            インデックスを増加する
1510.       } else {
1511.         サンプルをドロップし、復帰する
1512.       }
1513.     }
1514.   }
```

【図１６】

```
flush_overflow(cpu) {
  if (cpu == current_cpu) {
    allow_flip[cpu] := false
    ｃｐｕの非アクティブバッファをユーザスペースへコピーする
    allow_flip[cpu] := true
  } else {
    interprocessor_interrupt(cpu) { allow_flip[cpu] := false }
    ｃｐｕの非アクティブバッファをユーザスペースへコピーする
    interprocessor_interrupt(cpu) { allow_flip[cpu] := true }
  }
}
```

フロントページの続き

(72)発明者 ランス エム バーク
アメリカ合衆国 カリフォルニア州 ９４０２５ メンロ パーク ケント ２４－２
(72)発明者 サンジェイ ゲムワト
アメリカ合衆国 カリフォルニア州 ９４０４３ マウンテン ヴィュー ノース ウィスマン ロード １００－５６
(72)発明者 リチャード エル シテス
アメリカ合衆国 カリフォルニア州 ９４０２５ メンロ パーク カンポ ベーロ レーン １４５
(72)発明者 モニカ ハー ヘンツィンガー
アメリカ合衆国 カリフォルニア州 ９４０２５ メンロ パーク ラ ローマ ドライヴ ８０
(72)発明者 カール エイ ウォールドスパーガー
アメリカ合衆国 カリフォルニア州 ９４０２７ アサートン パーク ドライヴ ２７
(72)発明者 ウィリアム ウェイール
アメリカ合衆国 カリフォルニア州 ９４１１４ サン フランシスコ クリッパー ストリート ２８０

審査官 多胡 滋

(56)参考文献 特開平０５－３１３８２４（ＪＰ，Ａ）
特開平０４－３４４５４２（ＪＰ，Ａ）
特開平０４－０３３０２３（ＪＰ，Ａ）
Harold S. Stone, 高性能コンピュータアーキテクチャ, 日本, 丸善株式会社, １９８９年 ３月３０日, 第1版, ｐｐ．３１－５６

(58)調査した分野(Int.Cl., ＤＢ名)
G06F 11/34
G06F 15/167