(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4896376号
(P4896376)
(45)発行日　平成24年3月14日(2012.3.14)　(24)登録日　平成24年1月6日(2012.1.6)

(51)Int.Cl.　　F I
G06F 9/38 (2006.01)　　G06F 9/38 370C
G06F 9/50 (2006.01)　　G06F 9/46 462A

請求項の数 9　外国語出願　(全 60 頁)

(21)出願番号　特願2004-42173(P2004-42173)
(22)出願日　平成16年2月18日(2004.2.18)
(65)公開番号　特開2004-252983(P2004-252983A)
(43)公開日　平成16年9月9日(2004.9.9)
審査請求日　平成19年2月15日(2007.2.15)
(31)優先権主張番号　60/448,402
(32)優先日　平成15年2月18日(2003.2.18)
(33)優先権主張国　米国(US)
(31)優先権主張番号　60/448,399
(32)優先日　平成15年2月18日(2003.2.18)
(33)優先権主張国　米国(US)
(31)優先権主張番号　60/448,400
(32)優先日　平成15年2月18日(2003.2.18)
(33)優先権主張国　米国(US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　110001243
特許業務法人　谷・阿部特許事務所
(74)復代理人　100115624
弁理士　濱中　淳宏
(74)復代理人　100115635
弁理士　窪田　郁大
(72)発明者　アヌジ　ビー．ゴサリア
アメリカ合衆国　９８０５２　ワシントン州　レッドモンド　ノースイースト　１１０　ウェイ　１７７１１

最終頁に続く

(54)【発明の名称】コプロセッサの性能を強化するシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
タスクに対するすべての必要なメモリリソースが前記タスクの処理を始める前に使用可能であるかどうかを示すことによってコプロセッサ内の動作を能率的にする方法であって、
コンピュータのプロセッサが、前記タスクに関連するメモリリソースをコプロセッサ可読メモリに格納することにより、コプロセッサ内の処理に関するタスクであって、前記コプロセッサによりアクセス可能な直接メモリアクセス（ＤＭＡ）バッファに含まれたコマンドに対応するタスクを準備するステップと、
前記プロセッサが、すべての必要なメモリリソースが前記コプロセッサ可読メモリ内にあるかどうかを判定するために前記メモリリソースにアクセスするステップと、
前記プロセッサが、すべての必要なメモリリソースが前記コプロセッサ可読メモリ内にあるかどうかを記録するステップであって、前記記録は、前記タスクに関連するインジケータメモリリソースを生成するステップと、
前記コプロセッサが、前記ＤＭＡバッファ及び前記インジケータメモリリソースに基づいて、前記タスクの処理の始めに前記インジケータメモリリソースを処理するステップであって、前記インジケータメモリリソースが、必要なメモリリソースが前記コプロセッサ可読メモリ内にないことを示す場合に、第１のページフォールトが生成され、前記コプロセッサは前記タスクの処理を停止するステップと、
すべての必要なメモリリソースを後の時にコプロセッサ可読メモリ内の位置に格納する

ことができるように、前記コプロセッサが処理を停止したタスクのリストを保持するステップと、

前記コプロセッサのリングバッファ内の前記ＤＭＡバッファに対する無効な指示、または前記ＤＭＡバッファ内の無効なコマンドを参照するコンテキストへのコンテキスト切り替えが発生するときに、前記コプロセッサが第２のページフォールトを生成するステップと

を備えることを特徴とする方法。

【請求項２】

前記コプロセッサはＧＰＵであることを特徴とする請求項１に記載の方法。

【請求項３】

前記コプロセッサは、前記インジケータメモリリソースを処理するステップにおいて前記コプロセッサがページフォールトを生成したために前記タスクの処理を停止することを特徴とする請求項１に記載の方法。

【請求項４】

前記後の時は、前記タスクのリストのタスクの優先順位に基づいて決定されることを特徴とする請求項１に記載の方法。

【請求項５】

すべてのタスクを最終的に処理できることを保証するために、前記タスクのリストの１つまたは複数のタスクの優先順位を上げる周期的優先順位ブーストをさらに備えることを特徴とする請求項４に記載の方法。

【請求項６】

タスクに対するすべての必要なメモリリソースが使用可能であることを保証することにより、コプロセッサにおける動作を能率的にする方法であって、

前記タスクの処理を始める前に、

コンピュータのプロセッサが、前記タスクに関連するグラフィックス処理ユニット（ＧＰＵ）によりアクセス可能なメモリ空間内のメモリリソースをＧＰＵ可読メモリ内に格納することにより、前記ＧＰＵにおける処理のためのタスクであって、前記コプロセッサによりアクセス可能な直接メモリアクセス（ＤＭＡ）バッファに含まれたコマンドに対応するタスクを準備し、

前記プロセッサが、すべての必要なメモリリソースが前記ＧＰＵ可読メモリ内にあるかどうかを判定するために前記メモリリソースにアクセスし、かつ

前記プロセッサが、すべての必要なメモリリソースが前記ＧＰＵ可読メモリ内にあるかどうかを記録し、前記記録は前記タスクに関連するインジケータメモリリソースを生成するステップと、

前記コプロセッサが、前記タスクの処理を始める前に前記インジケータメモリリソースを処理するステップであって、前記インジケータメモリリソースが、必要なメモリリソースが前記ＧＰＵ可読メモリ内にないことを示す場合に、第１のページフォールトが生成され、前記タスクは処理されず、前記コプロセッサが処理を停止したタスクを含むタスクのリストが保持されるステップと、

前記コプロセッサのリングバッファ内の前記ＤＭＡバッファに対する無効な指示、または前記ＤＭＡバッファ内の無効なコマンドを参照するタスクに対してタスク切り替えが発生したとき、前記コプロセッサが第２のページフォールトを生成するステップと

を備えることを特徴とする方法。

【請求項７】

前記プロセッサが、前記リストを使用して、必要とされるメモリリソースを後の時に前記ＧＰＵ可読メモリ内に格納するステップを更に備えることを特徴とする請求項６に記載の方法。

【請求項８】

前記後の時は、前記タスクのリストのタスクの優先順位に基づいて決定されることを特徴とする請求項７に記載の方法。

【請求項９】
すべてのタスクを最終的に処理できることを保証するために、前記タスクのリストの１つまたは複数のタスクの優先順位を上げる周期的優先順位ブーストをさらに備えることを特徴とする請求項８に記載の方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、コンピュータプロセッサに関し、具体的には、コプロセッサの処理をスケジューリングするハードウェアおよびソフトウェアに関する。
【背景技術】
【０００２】
多くのコンピュータシステムに、現在、コプロセッサ、たとえばグラフィックス処理ユニット（ＧＰＵ）が含まれる。コプロセッサは、マイクロプロセッサなどの中央処理装置（ＣＰＵ）と共にシステムのマザーボード上にある場合があり、別々のグラフィックスカードにある場合がある。コプロセッサは、その処理タスクを実行する際に、しばしば、補助メモリ、たとえばビデオメモリにアクセスする。現在のコプロセッサは、しばしば、ゲームおよびコンピュータ支援設計（ＣＡＤ）などのアプリケーションをサポート（支援）するために３次元グラフィックス計算を実行するように最適化される。現在のコンピュータシステムおよびコプロセッサは、単一のグラフィック集中型アプリケーションを実行するときに適当に動作するが、複数のグラフィック集中型アプリケーションを実行するときに、問題が生じる場合がある。
【０００３】
これの理由の１つが、通常のコプロセッサが、作業負荷を効率的にスケジューリングする能力を有しないことである。現在のコプロセッサによって、通常は、協力的マルチタスキングが実施されるが、これは、現在コプロセッサを制御しているアプリケーションが、他のアプリケーションに制御を与えなければならないタイプのマルチタスキングである。アプリケーションは、制御を放棄できない場合に、効果的にコプロセッサを「独占する」ことができる。単一のグラフィック集中型プログラムを実行するときには、これは大きな問題ではなかったが、コプロセッサを独占するという問題は、複数のアプリケーションがコプロセッサの使用を試みるときに深刻になる。
【０００４】
動作の間で処理を配分するという問題は、複数の動作の洗練されたスケジューリングが必要になるＣＰＵに関して対処されてきたが、コプロセッサのスケジューリングは、有効に対処されなかった。これは、コプロセッサが、一般に、現在のシステムにおいて、計算的に重く時間がかかる動作をＣＰＵからそらし、ＣＰＵに他の機能のためのより多くの処理時間を与えるためのリソースとみなされるからである。そのような計算的に重い動作は、しばしば、グラフィックス動作であり、これは、かなりの処理能力を必要とすることがわかっている。
【０００５】
コプロセッサのためにタスクがスケジューリングされる場合に生じる問題の１つが、コプロセッサ「長時間停止状態（ｓｔａｒｖａｔｉｏｎ）」の可能性である。長時間停止状態は、コプロセッサがビジーでなく、したがって、コンピュータシステムの処理リソースが効果的に使用されていないときに発生する。
【発明の開示】
【発明が解決しようとする課題】
【０００６】
上記および他の理由から、コプロセッサ長時間停止状態を最小にすると同時に他のスケジューリング効率を可能にすることによってコプロセッサの性能を強化するシステムおよび方法が望ましい。
【課題を解決するための手段】

【０００７】
本発明は、コプロセッサ「長時間停止状態」を最小にし、より高い効率および威力のためにコプロセッサでの処理を効果的にスケジューリングする、連繋してまたは個別に使用することができるさまざまな技法を提供する。これに関して、実行リストが、中央処理装置（「ＣＰＵ」）によって提供され、ページフォールとまたはタスク完了などの切り替えイベントの発生時に、コプロセッサが、ＣＰＵ介入を待たずに、あるタスクから次のタスクへ即座に切り替えられるようになる。実行リストに加え、「サーフェスフォールト」と称する方法によって、コプロセッサが、サーフェスの途中のかなりの処理リソースが費やされた後の場所ではなく、サーフェスのレンダリングなどの大きいタスクの初めにフォールトを発生できるようになる。さらに、ＤＭＡ制御命令すなわち「フェンス」、「トラップ」、および「コンテキスト切り替え許可／禁止」が設けられ、これらを処理ストリームに挿入して、コプロセッサに、コプロセッサの効率および威力を強化するタスクを実行させることができる。これらの命令は、下で詳細に説明するように、高水準同期化オブジェクトの構築にも使用することができる。最後に、ディスプレイのベース参照をある位置から別の位置に切り替え、これによってディスプレイサーフェス全体を変更することができる「フリップ」技法を説明する。これらの技法を有益に使用することができる状況を、本発明の上記および他の態様のさらなる説明に加えて、下で示す。

【発明を実施するための最良の形態】

【０００８】
本発明によって達成される複数の改善を、図１および図２の比較を介して概念的に示すことができる。図１に、コプロセッサに関するタスクスケジューリングの従来技術の手法を示す。さまざまなアプリケーション、たとえばアプリケーション１、アプリケーション２、およびアプリケーション３からアクセスできるバッファが設けられる。アプリケーションは、コプロセッサに関するタスクをバッファにロードすることができ、これらのタスクを、前にサブミットされたタスクが完了した後に、コプロセッサによって処理することができる。図からわかるように、この手法では、コプロセッサの潜在的な「独占（ｈｏｇｇｉｎｇ）」が未解決のままである。図１では、アプリケーション１（ＡＰＰ．１）が、コプロセッサを独占している。ＡＰＰ．１は、コプロセッサが８つのタスクを扱うことを要求するが、他の２つのアプリケーションは、組み合わせても３つのタスクだけの操作を要求している。複数のアプリケーションがコプロセッサを必要とする、これに似た状況では、図２に示されたものなどのシステムによって、改善された機能性を提供することができる。

【０００９】
図２に、本発明によるシステムおよび方法を示すが、これによって、各アプリケーション、たとえばアプリケーション１、アプリケーション２、およびアプリケーション３が、それ自体のバッファすなわち、図２の「第１バッファ」を維持することができる。これらのバッファ（下では「コマンドバッファ」と称する）は、スケジューリングプロセスにサブミットされ、スケジューリングプロセスは、さまざまなタスクをコプロセッサに渡すときを決定することができる。図２に示すように、スケジューリングプロセスは、この例では、タスクを「第２バッファ」に挿入している。単純にするために、図２の「第２バッファ」を、単一のバッファとして図示する。しかし、実際には、図２の「第２バッファ」の機能を実行するために、複数のバッファが必要になる場合がある。図２の第２バッファによって、アプリケーション１がコプロセッサリソースを独占できないように、コプロセッサに渡されるタスクが分割されている。スケジューリングプロセスは、アプリケーション１のコプロセッサの第１タスクを許可し、次にアプリケーション２、次にアプリケーション３、次に再びアプリケーション１を許可している。

【００１０】
図２に概念的に示されたシステムおよび方法の実施形態は、図２に示されたものより複雑であるが、本明細書で開示される改善は、一般に、図２に示された基本概念をサポートする方向に向けられる。本発明の実施形態のより詳細な説明に移るが、参照を簡単にする

ために、下記の用語定義を示す。

【００１１】

コマンドバッファ　－　ユーザモードドライバによって構築されるバッファ。このバッファは、レンダリングアプリケーションのコンテキスト内で割り当てられる通常のページング可能メモリとすることができる。

【００１２】

ＤＭＡバッファ　－　「直接メモリアクセス」バッファ。カーネルモードドライバによって構築されるバッファ。このバッファは、コマンドバッファの概念に基づくものとすることができる。一般に、このバッファは、カーネルページング可能メモリから割り当てられ、カーネルだけに可視である。これに関して、ページをコプロセッサが読み取れるようになる前に、ページをロックし、アパーチャを介してマッピングすることができる。

【００１３】

ページングバッファ　－　カーネルモードドライバによって構築されるバッファ。このバッファは、特定のＤＭＡバッファに必要なメモリリソースのページイン、追出し、および移動に使用することができる。ページングバッファは、対応するＤＭＡバッファの前に即座に実行されるように構成することができる。

【００１４】

リングバッファ　－　これは、コプロセッサコンテキスト固有のバッファである。ＤＭＡバッファへの指示を、このバッファに挿入することができる。これに関して、コプロセッサは、そのようなリングバッファから、実行されるコマンドをフェッチすることができる。リングバッファには、一般に、ＤＭＡバッファからのコマンド読取を開始し、ＤＭＡバッファが完全に処理されたならばリングバッファに戻るようにコプロセッサに指示するリダイレクション命令が含まれる。

【００１５】

補助メモリ　－　一般に、コプロセッサによる使用専用であり、物理システムメモリの一部である必要がないメモリ。これは、たとえば、グラフィックスカードにあるローカルビデオメモリとすることができる。これは、システムメモリアパーチャを介してマッピングされたメモリなど、他のコプロセッサ可読メモリとすることもできる。このメモリは、通常は、統合型またはＵＭＡ式のグラフィックスデバイスには存在しない。このメモリは、ＧＡＲＴ様ページテーブルベースのアパーチャを介してアクセスされない。

【００１６】

システムメモリアパーチャ　－　これは、物理システムメモリのサブセットである。これは、ＧＡＲＴ様ページテーブルベースのアパーチャを介してコプロセッサに可視とすることができる。ＣＰＵは、システムメモリアパーチャと独立に物理システムメモリにアクセス可能とすることができる。概念的に類似する例が、アパーチャを介してアクセスされる、Ａｃｃｅｌｅｒａｔｅｄ　Ｇｒａｐｈｉｃｓ　Ｐｏｒｔ（「ＡＧＰ」）メモリ、Ｐｅｒｉｐｈｅｒａｌ　Ｃｏｍｐｏｎｅｎｔ　Ｉｎｔｅｒｃｏｎｎｅｃｔ（「ＰＣＩ」）Ｅｘｐｒｅｓｓメモリ、またはＵｎｉｆｉｅｄ　Ｍｅｍｏｒｙ　Ａｒｃｈｉｔｅｃｔｕｒｅ（「ＵＭＡ」）メモリである。

【００１７】

本発明のさまざまな実施形態のより詳細な図を、図３に示す。図３は、概念的に図２に示された機能を提供するために組み合わせることができるさまざまなソフトウェアオブジェクトおよびハードウェアオブジェクトの概略図である。図３に、下で説明する一連のシーケンシャルステップが示されている。これらのステップは、本発明を明瞭に説明し、使用可能にするためにシーケンシャルに提示されるものであって、本発明の実践に必要なシーケンスを表すものと解釈するべきではない。順序は、当技術分野で既知のまたは将来に開発される実践に従って変更することができる。下の説明は、図３のシステムおよび方法の概要から始まり、図３のいくつかの態様のより詳細な説明に進む。

【００１８】

図３のステップ１は、アプリケーションプログラムインターフェース（「ＡＰＩ」）へ

のアプリケーション呼出しを表す。アプリケーションは、ユーザにとってのソフトウェアを構成するファイルの任意の組とすることができる。ＡＰＩは、通常は、オペレーティングシステムカーネルと通信するのにアプリケーションによって使用される言語およびメッセージフォーマットであるが、データベース管理システム（ＤＢＭＳ）などの他の制御プログラムとの通信のフォーマットまたは通信プロトコルも指す。本発明と共に使用される例示的なＡＰＩの１つが、ＭＩＣＲＯＳＯＦＴ（登録商標）社によって開発されたＤｉｒｅｃｔ３Ｄ　Ｒｕｎｔｉｍｅ　ＡＰＩである。

【００１９】

ステップ２は、ＡＰＩからユーザモードドライバへの呼出しを表す。ユーザモードドライバは、一般に、ソフトウェアまたはハードウェアのいずれかとすることができる周辺サブルーチンへソフトウェアシステム（しばしばオペレーティングシステムである）をリンクすることができるプログラムルーチン（またはハードウェア）である。ここで、ユーザモードドライバは、ステップ１からの元の呼出しに対応するＡＰＩパラメータを含めることができる、ＡＰＩからの呼出しを受け取る。ステップ３は、ユーザモードドライバによって生成されるレンダリングコマンドの、コマンドバッファへの累積を表す。バッファは、中間リポジトリとしての使用のために予約されたメモリの領域である。データエリアおよびプロセッサまたはコプロセッサなどの２つの位置の間での処理のための転送を待っている間に、データを一時的にバッファに保持することができる。ユーザモードドライバによって生成されるコマンドバッファ内容の詳細は、下でさらに説明するように、ハードウェア固有ＤＭＡバッファへの変換を容易にするように選択することができる。また、コマンドバッファを定義する際に、「テクスチャ」または「頂点バッファ」などのメモリリソースへの直接メモリ参照を省略することが有用である場合がある。その代わりに、独立ハードウェアベンダ（「ＩＨＶ」）が、任意選択としてハンドルを含むコマンドバッファを定義することができ、カーネルインターフェースによって、メモリリソースが作成される時にコマンドバッファへのメモリ参照を提供することができる。

【００２０】

ステップ４は、コマンドバッファのフラッシュを表す。「フラッシュ」は、単純に、累積されたレンダリングコマンドについてコマンドバッファを空にすることを指す。レンダリングコマンドは、図３に示されたコプロセッサカーネルに渡すために、図示のようにＡＰＩに送り返すことができる。フラッシュは、コマンドバッファが満杯になり、着信レンダリングコマンドの空きが必要であるため、即時処理を必要とする高優先順位レンダリングコマンドがコマンドバッファに存在するため、などを含むがこれに制限されない理由で行うことができる。

【００２１】

ステップ５は、ＡＰＩによる、コプロセッサカーネルへの累積されたコマンドバッファのフラッシュを表す。カーネルは、一般に、オペレーティングシステムの中核部分すなわち、任意選択としてメモリ、ファイル、および周辺デバイスを管理し、アプリケーションを起動でき、システムリソースを割り当てることができる部分として知られる。コプロセッサカーネルを、プライマリシステムカーネル、または別々のコプロセッサ固有のカーネル、あるいは、たとえば、ＭＩＣＲＯＳＯＦＴ（登録商標）ＤｉｒｅｃｔＸ　Ｋｅｒｎｅｌ（「ＤＸＧ」）などの特定のタイプのカーネルを含む、任意のタイプのカーネルとすることができることを諒解されたい。

【００２２】

ステップ６は、コプロセッサカーネルへのコマンドバッファのサブミットを表す。コプロセッサカーネルは、コマンドバッファをカーネルモードドライバに向けることができる。カーネルモードドライバは、一般に、ユーザモードドライバに関して上で説明したドライバとすることができるが、カーネルモードドライバは、名前が示すように、カーネルモードで動作することができる。これに関して、カーネルモードドライバは、コマンドバッファをＤＭＡバッファに変換する責任を負うことができる。ＩＨＶは、コマンドバッファの正しい確認およびカーネルモードで割り当てられたＤＭＡバッファへの正しいコピーを

確実にする適当な機構を提供することを検討することができる。ＤＭＡバッファは、最終的にコプロセッサ向けのコマンドの集合であり、したがって、コプロセッサおよびサポートするハードウェアと正しくインターフェースしなければならないという点で、ハードウェア固有とすることができる。

【００２３】

図３を横切る、ユーザモードとカーネルモードを分離する水平線に留意されたい。この線によって示されるように、本発明は、システムのセキュリティのために実施される、コンピュータメモリ割り当ての従来のレイアウト内で動作することができる。ユーザモードは、非特権的メモリであり、アプリケーションによってアクセスできる。その一方で、カーネルモードは、特権的であり、アプリケーションによってアクセスできない。カーネルモードで割り当てられたＤＭＡバッファは、理論的にはあらゆるメモリ空間にマッピングできるが、アプリケーションのプライベートプロセス空間へのマッピングが、セキュリティリスクにつながる可能性があることに留意されたい。これは、アプリケーションのプライベートプロセス空間内のスレッドによって参照される仮想アドレスの内容を、修正できるからである。言い換えると、ＤＭＡバッファの内容を、確認された時とハードウェアによって処理される時の間に修正することができる。

【００２４】

ステップ７に示されているように、カーネルモードドライバによって、ＤＭＡバッファによって使用されるメモリリソースのリストを作ることもできる。これは、コマンドバッファの確認の一部として達成することができる。このリストには、たとえば、リストのさまざまなメモリリソースのカーネルハンドル、およびメモリリソースが参照されるバッファ位置を含めることができる。このリストに、リストされたメモリリソースの期待されるコンテキスト状態も含めることができる。これによって、現在のハードウェア状態（たとえば、「現在のレンダターゲット」、「現在のＺバッファ」など）の一部であるメモリリソースを、ＤＭＡバッファの先頭に再プログラムされるリストの一部とすることができる。というのは、コプロセッサにサブミットされた最後のＤＭＡバッファ以降に位置を変更された可能性があるからである。

【００２５】

ステップ８は、メモリリソースリストと共に、ＤＭＡバッファをコプロセッサカーネルに送ることを表す。コプロセッサカーネルは、ステップ９に示されているようにコプロセッサスケジューラにＤＭＡバッファをサブミットすることができ、ステップ１０に示されているようにユーザモードに戻ることができる。

【００２６】

コプロセッサスケジューラは、一般に、コプロセッサのタスク（コプロセッサに送られるさまざまなＤＭＡバッファおよび他の作業で具体化される）のフローをスケジューリングする責任を負う。コプロセッサスケジューラの機能性は、潜在的に非常に広範囲であり、この説明には、コプロセッサスケジューラが実行できる多数の潜在的な機能が含まれる。コプロセッサスケジューラを、コプロセッサスケジューラまたは単にスケジューラのいずれかと呼称することができる。さまざまな実施形態で、図３に示されているように、スケジューラによって、ＤＭＡバッファをコプロセッサにサブミットする前に、１つまたは複数の機能を実行することができる。ステップ１１ａは、スケジューラの一機能が、処理の準備ができたＤＭＡバッファのサブミットであることを動的に示す。

【００２７】

ステップ１１ｂは、準備済みＤＭＡバッファのリストに追加されるか、次に実行されるとスケジューラが決定したＤＭＡバッファの選択を表す。これに関して、スケジューラは、ＤＭＡバッファを準備スレッドに渡すことができる。準備スレッドは、本明細書で使用される用語として、一般に、正しいメモリリソースがＤＭＡバッファの処理を引き受けるようにする機能を提供する。第１に、準備スレッドは、補助メモリマネージャプロセス（図示せず）を呼び出して、現在補助メモリにないすべての必要なメモリオブジェクト（グラフィックスに関しては「サーフェス」）をページングするのに十分な位置を判定する（

これがステップ１２である）。用語「補助メモリ」が、コプロセッサによる使用に割り当てられたメモリを指すことに留意されたい。ＧＰＵコプロセッサの場合に、補助メモリを、しばしば、「ビデオメモリ」と呼ぶ。

【００２８】

ＤＭＡバッファによって要求されるメモリリソースの一部が、使用可能な補助メモリに同時にはおさまらない可能性がある。補助メモリマネージャが、様々な理由からこの点で補助メモリにすべてのサーフェスを持ち込めない可能性がある。これが発生する場合には、さらなる処理を行って、補助メモリにより多くの空きを設けるか、その代わりにまたは空きの作成と組み合わせて、ＤＭＡバッファを複数のフラグメントに分割することができる。この場合に、準備スレッドによって、ドライバ事前定義分割点を使用して、バッファを分割し、このより小さいＤＭＡバッファが必要とするメモリリソースのサブセットを突き止めるのを試みることができる。

【００２９】

ＤＭＡバッファに十分な補助メモリが突き止められたならば、準備スレッドによって、ステップ１３に示されているように、カーネルモードドライバを呼び出すことができる。当業者が諒解するように、これは、ステップ６、７、および８に関して述べたカーネルモードドライバとするか、別々のカーネルモードドライバとすることができる。

【００３０】

ステップ１４には、カーネルモードドライバによって、処理を待っているＤＭＡバッファのページングバッファを構築できることが示されている。カーネルモードドライバによって、準備スレッドからの処理コマンドに基づいてこのページングバッファを構築することができる。ページングバッファは、上で定義したように、メモリリソースのページングのためのバッファである。「ページング」は、マッピングハードウェアを使用してメモリのブロック（ページ）の物理アドレスを変更することを指す。ページングバッファは、一般的に言って、メモリリソースをそれに割り当てられた位置に移動するコプロセッサ命令を含むＤＭＡバッファである。ページングバッファは、ＤＭＡバッファが必要とするメモリリソースを正しいメモリ位置に移す機能をサービスし、このメモリ位置から、必要なときにコプロセッサによってそのリソースにアクセスすることができる。ページングバッファが正しく生成された場合に、特定のコプロセッサタスクに関する必要なすべてのメモリリソース（すなわちＤＭＡバッファ）の位置がわかる。

【００３１】

ステップ１５は、ページングバッファが生成されたことの準備スレッドへの通知を表す。ステップ１６は、ページングバッファの準備ができたことのスケジューラへのシグナルを表す。スケジューラは、この時点で、次のＤＭＡバッファの処理の準備ができていると仮定することができ、あるいは、処理のためにコプロセッサに送る前に、ＤＭＡバッファに対するさらなる準備動作を行うために進行することができる。たとえば、元のＤＭＡバッファの作成以降にメモリ位置が変更されている可能性があるので、スケジューラは、この点でカーネルモードドライバをもう一度呼び出して、メモリリソースの実際の位置を用いてＤＭＡバッファをパッチできるようにすることができる。最後に、スケジューラは、ページングバッファ（存在する場合に）およびＤＭＡバッファの両方を、処理のためにコプロセッサ（および他の任意の補助ハードウェア）にサブミットすることができる。

【００３２】

上で説明したステップ１から１６は、ハードウェア、ソフトウェア、およびその組合せを介して実施することができる。これに関して、図４（Ａ）および４（Ｂ）に、擬似アルゴリズムの形式で図３のステップを全般的に示す。図４（Ａ）および４（Ｂ）は、本発明に関して実践することができる潜在的な擬似アルゴリズムステップの網羅的なリストではなく、図４（Ａ）および４（Ｂ）のすべてのステップが本発明の実践に必要と解釈するべきではない。そうではなく、図４（Ａ）および４（Ｂ）は、本発明の教示における暗示的なリストである。

【００３３】

図３に関して提供した上の説明は、本発明のさまざまな実施形態の説明である。しかし、上で説明した本発明の実施形態に関して、複数の利益が発見された。この説明の残りは、さまざまな改善を可能にし、本発明の実践で生じる可能性がある困難を克服するためのものである。

【００３４】

スケジューリングの考慮事項

前に定義した動作（上のステップ１から１６を参照されたい）の一部またはすべてを、ＤＭＡバッファがハードウェアにサブミットされる前に行うことができる。しかし、これらの動作の一部は、ＤＭＡバッファがハードウェアにサブミットされるまで実行が困難である場合がある。たとえば、メモリリソースの位置は、ＤＭＡバッファがコプロセッサにサブミットされる直前まで、判定が困難である可能性がある。これは、補助メモリリソースが、コプロセッサで実行されるときに各ＤＭＡバッファと共に移動される可能性があるからである。

【００３５】

上のステップ１から１６に含まれる動作の一部は、時間がかかり、したがって、たとえばスケジューラが次に実行するタスクを選択した後など、割込み時に行うことができない場合がある。同様に、まさに時間がかかるので、コプロセッサが他のタスクで忙しい間は、中央処理装置（「ＣＰＵ」）で実行することが有益である。これによって、コプロセッサ長時間停止状態（ｓｔａｒｖａｔｉｏｎ）が最小限になる。コプロセッサ長時間停止状態は、単に、コプロセッサが処理機能を処理せずに費やされる時間を指す。この問題に応答して、スケジューラと共に「ワーカースレッド」を利用することが有益である可能性がある。ワーカースレッドによって、時間のかかるセットアップ作業の一部を処理するのに役立つ機能を実行することができる。ワーカースレッドは、本発明の他のプロセスに関する動作の例について図４（Ｂ）の擬似アルゴリズムに追加された。

【００３６】

このスケジューリングの考慮事項に加え、図３のシステムで任意の所与の時に、動作中のＤＭＡバッファ（すなわち、コプロセッサによって現在処理されつつあるＤＭＡバッファ）、準備されつつあるＤＭＡバッファ、および準備される準備ができているＤＭＡバッファのリストが存在する可能性がある。スケジューラへのサブミットの際に、新しいＤＭＡバッファを、レディキュー（ｒｅａｄｙ　ｑｕｅｕｅ）に挿入し、その優先順位に応じて適当に順序付けることができる。しかし、新しいＤＭＡバッファが、スケジューラにサブミットされるときに、コプロセッサの次のタスクとして選択されたＤＭＡバッファを先取りできない場合に、本発明のさまざまな実施形態によって、機能性を高めることができる。これの理由は、ＤＭＡバッファの準備に、補助メモリとの間でのメモリリソースのページングが伴う可能性があるからである。したがって、処理に関して選択された次のＤＭＡバッファの先取りによって、補助メモリマネージャの永続状態の変化がもたらされる可能性がある。準備されるタスクが先取りされる可能性がある場合には、新たに選ばれたＤＭＡバッファの準備に起因する、補助メモリマネージャの永続状態に対する変更を元に戻すことがもたらされる可能性がある。ＤＭＡバッファタスクの動作の途中での補助メモリに対する変更を元に戻すことは、普通ではない可能性があり、潜在的に頻繁なコプロセッサ長時間停止状態につながる可能性がある。

【００３７】

ＤＭＡバッファの分割

コマンドバッファが、ＡＰＩによってコプロセッサカーネルにサブミットされるときに、カーネルモードドライバは、ハードウェア固有のＤＭＡバッファおよびそのＤＭＡバッファを実行するのに必要なメモリリソースのリストを生成する役割を担うことができる。特定のＤＭＡバッファフォーマットが、ＩＨＶによって定義される場合があるが、ソフトウェアプロバイダは、自身がカーネルモードドライバのリソースリストのフォーマットを定義する作業を知ることができる。

【００３８】

メモリリソースリストによって、ＤＭＡバッファによって使用できる異なるメモリリソースのタイムライン（ｔｉｍｅｌｉｎｅ）情報を提供することができる。スケジューラは、このメモリリソースリストを使用して、ＤＭＡバッファがコプロセッサ上で実行される前にすべての必要なメモリリソースをページインし、ＤＭＡバッファによって同時に大量のリソースが使用されるときなど、必要な場合にＤＭＡバッファを分割することができる。

【００３９】

ＤＭＡバッファが、スケジューラによって分割される場合に、カーネルモードドライバは、メモリリソースリスト内でタイムライン情報を供給することによって、これを容易にする。これは、ドライバに、ＤＭＡバッファ内の「オフセット」を指定させることによって行うことができる。オフセットは、そのオフセットでのメモリリソースの使用法を指定するメモリリソース識別子を挿入することによって、メモリリソースがプログラムされているときに送ることができる。メモリリソースは、ＤＭＡバッファ内に複数回現れることができるので、同一のメモリリソースが、メモリリソースリストに複数回現れる場合がある。ＤＭＡバッファ内のメモリリソースへの参照のそれぞれによって、１つのエントリがリソースリストに追加される。

【００４０】

本来、ハンドル／オフセットリストは、ＤＭＡバッファを分割する必要があるメモリリソースに関する十分な情報をスケジューラに与えるのに十分でない可能性がある。ＤＭＡバッファで特定のメモリリソースが必要なときを正確に知るために、スケジューラが、メモリリソースが別のリソースによって置換されるときに関する情報を必要とする場合もある。たとえば、ＤＭＡバッファの先頭で第１テクスチャステージに使用される第１のテクスチャ、テクスチャＡが、途中で第２のテクスチャ、テクスチャＢによって置換され、その後、ＤＭＡバッファの最後でテクスチャＡに戻される。スケジューラは、この追加情報を使用して、より少ないメモリリソースを使用する量でＤＭＡバッファを分割することができる。しかし、上で説明したシナリオでは、テクスチャＢも、第１テクスチャステージでプログラミングされる可能性があり、この場合には、テクスチャＡと同時に使用されており、ＤＭＡバッファの別々のサブセットに分離するべきではない。

【００４１】

上で説明した洗練された方法でＤＭＡバッファを分割するのに必要な「微細な粒子（ｆｉｎｅｒ　ｇｒａｉｎ）」の時間情報を達成するために、スケジューラによって、ＤＭＡバッファ全体のメモリリソースの使用状況に関する情報を使用することができる。これは、一実施形態では、カーネルモードドライバによって、メモリリソースリスト内のエントリごとにリソース識別子を供給することによって達成することができる。リソース識別子は、単純に、特定のメモリリソースがどのように使用されるかを表す整数値である。たとえば、０の値によって、メモリリソースがレンダターゲットとして使用されることを示すことができ、１の値によって、リソースがＺバッファとして使用されることを示すことができる。この情報を用いて、スケジューラは、テクスチャＢによってテクスチャＡが置換されるか（たとえば、両方が同一のリソース識別子を有する）、またはテクスチャＢがテクスチャＡと同時に使用される（たとえば、ＡおよびＢが異なるリソース識別子を有する）かどうかを判定することができる。リソース識別子の実際の値およびその意味は、ＩＨＶが定義するか、ソフトウェアアーキテクチャで供給することができる。リソース識別子として使用される値が、０ベースになることを保証し、ドライバ初期化の時にそのドライバが使用する最大のリソース識別子の値をドライバが指定することが有用になる可能性がある。

【００４２】

図５に、ＤＭＡバッファ内で使用されるメモリリソースのタイムラインを定義するために供給される情報をスケジューラがどのように使用できるかを示す。スケジューラは、タイムラインの使用に進んで、バッファ分割点を定義することができる。一般に、ＤＭＡバッファが、現在のメモリリソース（すなわち、前のＤＭＡバッファの末尾で最新であった

メモリリソース）の、「セットアップ」または識別プロセスを開始しなければならないことに留意することが重要である可能性がある。これの理由は、前のＤＭＡバッファが実行されてから後に、メモリリソースが移動され、したがって、再プログラムが必要である場合があるからである。メモリリソースは、ＤＭＡバッファが処理のためにスケジューリングされる時までに、再プログラムされる必要がある場合がある。

【００４３】

図５に示されたメモリリソースリストに、任意の数のフィールドを含めることができる。下の表に、有用なフィールドの非網羅的リストを示す。

【００４４】

【表１】

Handle	メモリリソースのハンドル。
ResourceId	任意選択としてリソースを使用する方法を指定するリソース識別子。
Offset	メモリリソースをプログラムできる場所のDMAバッファ内のオフセット。スケジューラは、メモリ制約のゆえにバッファを分割する必要がある場合に、その点までDMAバッファを実行するようにドライバに要求することができる。したがって、このオフセットによって、DMAバッファの有効な分割点を供給することができる。
SegmentHint	最適の性能を提供するためにドライバが特定の割り当てに使用することを望むセグメントを指定する。これによって、割り当てに関する現在のドライバプリファレンスを置換することができる。
BankHint	カーネルモードドライバが割り当てをページングするヒント付きセグメント内のバンクを指定する。これによって、割り当てに関する現在のドライバプリファレンスを置換することができる。
SegmentId	メモリリソースを保持するセグメントのセグメント識別子。これは、ページング中に書き込むことができる。
PhysicalAddress	セグメント内のメモリリソースの物理アドレスを指定する。これは、ページング中に書き込まれる。

【００４５】

ページング

一般に、ＤＭＡバッファによって参照されるメモリリソースは、コプロセッサによる実行のためにＤＭＡバッファがサブミットされる前に、メモリ内に持ち込まれる。参照されるメモリリソースをメモリ内に持ち込むことを、リソースのページングと称する。ページングに、上で説明した準備ワーカースレッドと、カーネルモードドライバなどのドライバの間の対話を含めることができる。準備ワーカースレッドと補助メモリマネージャの間の動きを示す擬似アルゴリズムについて、図６を参照されたい。

【００４６】

ページングステップは、通常は、処理についてＤＭＡバッファが選択され、特定のＤＭＡバッファに関するリソースのリストが生成されているときに行われる。ページングは、メモリリソースを補助メモリに入れる方法と、補助メモリのどこにそれを置くかを決定するために行われる。

【００４７】

ページング処理を、補助メモリマネージャによって処理することができる。補助メモリマネージャは、特定の割り当ての作成時にカーネルモードドライバによって任意選択として供給されるヒントを使用することができる。このヒントは、メモリリソースに関してメ

モリ内の適当な位置を見つけるために作成されたものである。

【００４８】

メモリリソースのページングに関連する複数の問題がある。すべてのリソースを持ち込むのに使用可能な十分な空き補助メモリがない場合があり、その場合には、現在メモリ内にあるリソースを追い出すことができる。補助メモリ内の他のオブジェクトを追い出した後であっても、ＤＭＡバッファ用のメモリが不十分である場合がある。その場合には、ＤＭＡバッファを複数のより小さい部分に分割することができ、これによって、必要なメモリリソースが減る。

【００４９】

ページング中に、補助メモリマネージャは、適当な位置にメモリリソースを置くのに使用することができるコマンドのリストを構築することができる。そのコマンドのリストは、たとえば下記の動作から構築することができる。

１）追出し：別のリソースのための空きを作るために、特定のメモリリソースを現在のセグメントからシステムメモリに移動する；

２）ページイン：特定のメモリリソースをシステムメモリから補助メモリ内の空き位置に持ち込む；

３）再配置：特定のメモリリソースをある補助メモリ位置から別の補助メモリ位置に移動する。

【００５０】

補助メモリマネージャは、メモリ配置問題を解決するために、これらの動作のどれでも使用することを許可されることができる。この非網羅的なコマンドリストは、ページング動作中に補助メモリマネージャによって生成し、後にスケジューラによってページングバッファを生成するのに使用することができる。補助メモリマネージャは、再配置、追出し、またはページイン、あるいは他の方法で移動されるまたは他の方法で変更されるメモリリソースについてコマンドリスト内のエントリを生成することができる。これに関して、本発明のさまざまな実施形態によって、コマンドリスト内に下記のフィールドを設けることができる。

【００５１】

【表２】

Handle	再配置されるメモリリソースのハンドル。
SegmentId	メモリリソースが現在ロードされているセグメントのセグメント識別子。
PhysAddress	メモリリソースの現在のセグメント内の現在の物理アドレス。
NewSegmentId	リソースが移動される先のセグメントのセグメント識別子。
NewPhysAddress	リソースが移動される先の新しいセグメント内の新しい物理アドレス。

【００５２】

ページングバッファ生成

上で説明したコマンドラインを使用して、スケジューラによってページングバッファを生成して、コマンドを実行することができる。本発明に関連して使用されるページングバッファのさまざまな実施形態は、図７に示されているように実施することができる。

【００５３】

図７からわかるように、コマンドの中には、実行の前に前処理を必要とするものと、前処理なしで処理できるものがある。前処理は、ワーカースレッドを含む複数の方法で行うことができる。コマンドの前処理で、ページングバッファの一部が処理されるまで待つ必要がある場合があることに留意されたい。図７に示されたモデルでは、ワーカースレッド

によって、ページングバッファを準備し、ページングバッファに関するＣＰＵ前処理を処理する。ＣＰＵ前処理が、ページングバッファでの動作の前に必要なときには、ワーカースレッドによって、コプロセッサでのページングバッファに対する動作がブロックされる。その後、ワーカースレッドによって、ページングバッファを再始動する前にＣＰＵ要求がサブミットされて、動作が完了する。

【００５４】

したがって、コマンドリスト内のコマンドごとに、下記のアクションが適当である可能性がある：

ページングバッファ生成時の前処理；

ページングバッファ内の同期ポイントでのＣＰＵ処理；

メモリリソースを移動する「Ｂｌｉｔ」コマンド；

ページングバッファが完了した後のＣＰＵ作業の後処理。

【００５５】

可能なアクションの上のリストに関して、ページングバッファ自体に、ＣＰＵが作業を処理している間にコプロセッサに停止するように要求するコマンドを含めることができる。そのような、割込みを生成し、コプロセッサを停止させるコマンドを、本明細書では「ブロッキングフェンス」と称する。ページングバッファ内のどのコマンドにも、その前または後にブロッキングフェンスを置くことができる。割込みは望ましくないので、ＣＰＵがコプロセッサに割り込む回数は、ポスト動作フェンスをバッファの末尾に集めることによって減らされる。ポスト動作フェンス（または「ポストフェンス」）がバッファの末尾の前に必要である場合は、スケジューラによって検出され、ポストフェンスが実行済みであることを必要とするコマンドのプリ動作フェンス（または「プリフェンス」）にマージされる。

【００５６】

補助メモリの一貫性を維持するために、ページングバッファの処理で外部割込みを許可しないことが有利になる場合がある。したがって、ページングバッファが実行を完了する前にカンタム（ｑｕａｎｔｕｍ）が満了する場合には、ページングバッファを、完了までコプロセッサの制御下に留まらせることができる。

【００５７】

ページングバッファ内のフェンスの処理を含む、ワーカースレッド内で行うことができるイベントのチェーンを表す擬似アルゴリズムについて、図８を参照されたい。図８に関して、下の表に、コマンドリスト内に現れる可能性がある一般化されたコマンドと、前処理、ページングバッファ生成、および生成され得る終了フェンスに関するコマンドのタイプのありそうな分岐のリストを示す。下の表は、役に立つ例示としてのみ提供され、可能なコマンドのタイプまたはこれらのコマンドに関して発生し得るアクションの網羅的なリストであることを意図されたものではない。

【００５８】

【表３】

補助メモリから別の補助メモリ位置への移動	前処理： なし。 ページングバッファ内： 転送がハードウェアで行われる場合 ドライバが、ページングバッファにblitを追加することができる。 転送がソフトウェアで行われる場合 現在のページングバッファをフラッシュする。フラッシュされたならば、CPUでの転送に進む。 ページングバッファの終了フェンス内： なし。
補助メモリからアパーチャへの移動	前処理： 移動される補助メモリリソースを所有するプロセスにアタッチする； そのシステムメモリバッファをMmProbeAndLockし、ロックされたページのMDLを得る； ページのMmProbeAndLockに失敗する場合 、ソフトウェアでblitを処理する； プロセスからアンアタッチされる； 割り当てられたアパーチャ位置が、現在ビジーでなく、コマンドリスト内で現在のコマンドの前のコマンドがない場合に、そのアパーチャ範囲を操作する。 生成したMDLを用いてアパーチャをプログラムする。 アパーチャがプログラムされたことに留意する。 ページングバッファ内： アパーチャが前処理ステージでプログラムされなかった場合に現在のページングバッファをフラッシュする。フラッシュの後に、MDLをアパーチャにプログラムする。ページングバッファの処理を継続する。 転送がハードウェアで行われる場合にドライバが、ページングバッファにblitを追加する。 転送がソフトウェアで行われる場合に； 現在のページングバッファをフラッシュする。フラッシュの後に、CPUを使用してメモリを転送する。ページングバッファの処理を継続する。 ページングバッファの終了フェンス内： なし。

【００５９】

【表４】

アパーチャから補助メモリへの移動	前処理： 移動される補助メモリリソースを所有するプロセスにアタッチする； システムメモリバッファをMmProbeAndLockし、ロックされたページのMDLを得る； ページのMmProbeAndLockに失敗する場合にソフトウェアでblitを処理する； プロセスからアンアタッチする； 割り当てられたアパーチャ位置が、現在ビジーでなく、コマンドリスト内で現在のコマンドの前にコマンドがない場合に、そのアパーチャ範囲を操作する。 生成したMDLを用いてアパーチャをプログラムする。 アパーチャがプログラムされたことに留意する。 ページングバッファ内： アパーチャが前処理ステージでプログラムされなかった場合に現在のページングバッファをフラッシュする。フラッシュの後に、MDLをアパーチャにプログラムする。ページングバッファの処理を継続する。 転送がハードウェアで行われる場合にドライバが、ページングバッファにblitを追加する。 転送がソフトウェアで行われる場合に： 現在のページングバッファをフラッシュする。フラッシュの後に、CPUを使用してメモリを転送する。ページングバッファの処理を継続する。 ページングバッファの終了フェンス内： アパーチャ範囲が、まだバッファ内で別の動作によって再生されていない場合にアパーチャ範囲をアンマップする； サーフェスを所有するプロセスからアタッチする； システムメモリバッファをMmUnlockする； プロセスからアンアタッチする。
補助メモリからの追出し	ビデオからアパーチャへの移動と同一の処理。ページングバッファの終了フェンスでアパーチャ範囲がアンマップされることが異なる。

【００６０】

【表５】

アパーチャからの追出し	前処理： 　アパーチャ範囲がビジーでない場合にアパーチャ範囲をアンマップする； 　サーフェスを所有するプロセスにアタッチする； 　システムメモリバッファをMmUnlockする； 　プロセスからアンアタッチする。 ページングバッファ内： 　なし。 ページングバッファの終了フェンス内： 　アパーチャ範囲が、まだ前の動作によってアンマップされていない場合 　アパーチャ範囲をアンマップする； 　サーフェスを所有するプロセスにアタッチする； 　システムメモリバッファをMmUnlockする； 　プロセスからアンアタッチする。

【００６１】

本明細書で提示するスケジューリングモデルが、コプロセッサをビジーに保つためにかなりの量の自明でないＣＰＵ処理を必要とする可能性があることに留意されたい。この作業は、少なくとも部分的に、現在存在するコプロセッサハードウェアの機能によって必要とされる。将来のグラフィックスハードウェアは、より強力なメモリ仮想化およびコプロセッサスケジューリングを有するように設計される可能性がある。これに関して、複数の長所に到達しており、これらの長所も本発明に関して開示される。ハードウェア機能のそれぞれについて、改善の動機および上で説明したスケジューリングモデルに対する影響を説明する。いくつかの改善は、特定の実施手法に基づいて提示される。これらの手法のすべてが必ずしも将来のモデルでサポートされるのではないが、特定の手法が実践される場合に、そのときに、改善を実施手法に適合させる基準を提供する方法で、さまざまな改善を本明細書で説明する。

【００６２】

割込み可能なハードウェア

コプロセッサスケジューリングの信頼性を高めるために、コプロセッサによって、ＤＭＡバッファ全体より微細な粒度での割込みをサポートすることができる。たとえば、コプロセッサおよびサポートするハードウェアによって、三角形の処理の前または後だけではなく、三角形の処理の中での割込みをサポートすることができる。

【００６３】

そのような割込み可能ハードウェアのさまざまな実施形態で、好ましい設計手法は、コプロセッサコンテキストの補助メモリへの自動的な保存および復元を介してコプロセッサの潜在的に完全な仮想化を提供することとすることができる。各コプロセッサコンテキストは、制限ではなく例として、プライベートアドレス空間、ＤＭＡバッファが累積されるプライベートリングバッファ、およびコプロセッサコンテキストが実行中でないときにハードウェアの状態が保存されるメモリのプライベート部分を有することができる。この設定でのコンテキスト切り替えをサポートするために、スケジューラによって、保存されたコンテキストの補助メモリ内の物理アドレスをメモリマップ式レジスタを介してコプロセッサに与えることができる。コプロセッサは、そのコプロセッサコンテキストをロードし、すべてのメモリリソースが有効であることを検証し、リングバッファに累積されたＤＭＡバッファを実行し、必要なリソースに出会うときにそのリソースをフォールトにする。

【００６４】

上記に関して、さらに、カーネルモードドライバによって、実行されていないコプロセ

サコンテキストの状態を照会することを可能にすることができる。これは、「実行リスト」イベントトレース（下で説明する）を使用することによるか、照会手段によって、保存されたコンテキストを検査することによって行うことができる。これに関して、ドライバによって、（１）コプロセッサがもっとも最近に特定のコンテキストから切り替えられた理由（たとえば、空、新しい実行リスト、ページフォールト）、（２）ハードウェアによって使用中のメモリリソースのリスト（サーフェスレベルフォールトがサポートされる場合）、（３）フォールトアドレス（ページレベルフォールトがサポートされる場合、および（４）特定のコンテキストを実行していたコプロセサクロックサイクル数などの有用な情報を判定することができる。

【００６５】

さらに、カーネルモードドライバによって、現在実行されていないコンテキストのリングに、新しいＤＭＡバッファを挿入できるようにすることができる。保存されたコンテキスト内で、リングの位置、そのコンテキストに保管されたページテーブルまたは他の物理メモリ参照を修正できることも可能である。そのような修正は、たとえば、メモリ内のリソースの移動に続いて必要になる可能性がある。

【００６６】

コプロセッサコンテキストごとの仮想アドレス空間

上で説明した基本的なスケジューリングモデルの複雑さの一部は、コプロセッサコンテキストが共通のコプロセッサアドレス空間を共有している可能性があるという事実に起因する。このアドレス空間を仮想化することによって、よりスマートなシステムを提供することができる。アドレス空間の仮想化では、補助メモリマネージャによって、メモリを移動することができ、リソースを補助メモリから完全に追い出すことさえできる。これは、リソースに関する実際にコプロセッサから可視のアドレスを、その寿命の間に変更できることを意味する。したがって、ユーザモードで構築されるコマンドバッファは、コマンドバッファが実行のためにスケジューリングされるまでアドレスが未知である可能性があるので、そのアドレスによって割り当てを直接に参照することができない。

【００６７】

たとえば、上で説明した基本スケジューリングモデルの下記の要素を、コプロセッサコンテキストごとのアドレス空間の使用を介して除去することができる。

１）ハンドルを実際のメモリ位置に置換することによるコマンドバッファのパッチ
２）メモリアクセスに関するコマンドバッファの確認
３）カーネルモードでのメモリリソースリストの構築
４）別々のコマンドバッファおよびＤＭＡバッファの作成
５）割り込まれたＤＭＡバッファのリソースの割込み前の位置への移動

【００６８】

プロセッサコンテキストごとの仮想アドレス空間を提供する際に、特定のコプロセッサコンテキスト内での割り当てによって、そのコンテキストのアドレス空間内の独自のアドレスを得ることができる。このアドレスは、割り当ての寿命の間に変更する必要がない。したがって、コマンドバッファによって、このアドレスを直接に参照することができ、パッチは不要である。コマンドバッファを確認し、ＤＭＡバッファにコピーする必要もなくなる。ＤＭＡバッファでのメモリ参照は、コプロセッサの仮想アドレス空間内であり、そのアドレス空間が、実際にすべてのコプロセッサコンテキストについてプライベートなので、有効性に関してメモリ参照を確認する必要はなく、したがって、アプリケーションに可視でないＤＭＡバッファ内のコマンドバッファの確認された内容を隠蔽する必要はない。割り当てまたは追い出された割り当てによって占められていないアドレス空間（ハンドルまたは実アドレス）は、ハードウェアによってダミーページにリダイレクトされるか、アクセスフォールトを引き起こすことができる。これによって、コンテキストが、アクセスすると思われないメモリにアクセスできないので、カーネルモードメモリのセキュリティが保たれる。

【００６９】

コプロセッサコンテキストごとの仮想アドレス空間の長所の一部は、次の通りである。各割り当てが、割り当て時にコプロセッサに可視のアドレス（またはハンドル）を得る。コマンドバッファがなく、ＤＭＡバッファは、ユーザモードドライバに直接に可視であり、ユーザモードドライバによって書き込まれる。ＤＭＡバッファは、それが使用する割り当てのアドレス（またはハンドル）を直接に参照する。ページングに使用されるリソースリストは、ユーザモードドライバによって構築される。

【００７０】

図３および対応する説明に示された本発明のさまざまな実施形態のモデルを想起されたい。このモデルを、割込み可能ハードウェアおよび／またはプロセッサコンテキストごとの仮想アドレス空間を使用してさらに改善することができる。これに関して、次の節で、本発明の追加の長所によってさらに改善されることを除いて図３の概念に似た概念を説明する。

【００７１】

サーフェスの割り当ておよび割り当て解除

高度なモデルで、カーネルモードのビデオメモリマネージャ「ＶｉｄＭｍ」などの補助メモリマネージャによって、コプロセッサコンテキストに関する仮想アドレス空間を提供でき、さまざまなコプロセッサコンテキストの間で、これらがメモリの公平な分け前を得られるように物理メモリを管理することができる。基本モデルの割り当て方式に対するこの改善のさまざまな実施形態を、図９に示す。図９には、当業者に馴染みのある用語を使用して本発明の実施形態を示す。というのは、これが、当技術分野で認識された概念に対応するからである。たとえば、「ＶｉｄＭｍ」は、ビデオメモリマネージャであり、「サンク（Ｔｈｕｎｋ）インターフェース」は、サンクインターフェースである。しかし、この用語は、本発明をより明瞭に説明するのに使用されるが、本発明を制限する意図の表明と解釈するべきではないことに留意されたい。したがって、「ＶｉｄＭｍ」は、任意の補助メモリに関するメモリマネージャとすることができ、「サンクインターフェース」は、任意の適当なインターフェースとすることができる。

【００７２】

図９に関して、高度なモデルを用いると、ＤＭＡバッファをアプリケーションのアドレス空間に直接にマッピングできるようになり、これによって、任意選択として、ＤＭＡバッファをユーザモードドライバから直接にアクセス可能にすることができる。ユーザモードドライバは、アクセスする必要がある各メモリリソースの永久仮想アドレスまたはハンドルを使用して（したがってパッチが不要になる）レンダリングプリミティブをＤＭＡバッファに直接にバッチ化する。ユーザモードドライバによって、ＤＭＡバッファによって使用されるメモリリソースのリストも構築され、したがって、補助メモリマネージャは、ＤＭＡバッファがスケジューリングされる前にこれらのリソースを補助メモリバッファに持ち込むことができる。悪意のあるアプリケーションがリソースリストを修正する場合に、リソースの正しい組が、正しくページインされなくなる。これによって必ずしもメモリ保護モデルが壊されないことに留意されたい。というのは、有効なメモリを参照していないアドレス空間の範囲が、ダミーメモリページを参照することまたはハードウェアにフォールトにすることのいずれかのために必要になる可能性があり、指定されたコプロセッサコンテキストの実行を停止させるからである。どちらの場合でも、壊れたリソースリストによって、コプロセッサコンテキストが別のコンテキストのメモリにアクセスできるようになることがもたらされることはない。

【００７３】

高度なモデルでは、ユーザモードドライバが、ＤＭＡバッファをカーネルモードドライバにサブミットし、カーネルモードドライバが、ＤＭＡバッファをスケジューラにサブミットする。メモリマネージャにリソースリスト内のリソースをページングするように要求した後に、スケジューラは、ＤＭＡバッファをそのままハードウェアに送る。

【００７４】

高度なモデルでのスケジューリング

高度なモデルでのスケジューリングは、基本モデルでのスケジューリングに非常に似ている。ＤＭＡバッファがコプロセッサにサブミットされる前にＤＭＡバッファを準備するワーカースレッドが依然としてある。しかし、高度なモデルでワーカースレッドによって達成できる作業は、ページング動作だけに制限される必要がある。

【００７５】

基本モデルのスケジューリングおよび高度なモデルのスケジューリングの実施形態について、図１０および図１１を参照されたい。これから明らかにするように、高度なモデルは、２つのスケジューリングオプションを有する。デマンドフォールトなしでスケジューリングするときには、準備フェーズを実施することができる。しかし、高度なモードでデマンドフォールトを使用するときには、準備フェーズは不要である。

【００７６】

さらに、図１２（Ａ）、１２（Ｂ）に、高度なスケジューリングモデルを実施することができる擬似コードを示す流れ図を示す。

【００７７】

高度なモデルでのページング

高度なモデルでのページングは、基本モデルのページングと異なる。高度なモデルでは、ページングされる割り当てのアドレスが、既に知られており、メモリマネージャは、単に、それを有効にする必要がある。リソースリスト内の割り当てを有効にするために、メモリマネージャは、空いている物理補助メモリの範囲を見つけ、ドライバに、ページテーブルまたはハンドルをその範囲にマッピングするように要求する必要がある。必要であれば、物理メモリの範囲が連続するページのセットになることを要求することができる。

【００７８】

割り当てを有効にするのに使用可能な物理ビデオメモリが不十分な場合には、本明細書でＶｉｄＭｍと称する補助メモリマネージャは、追出しについて、現在有効な割り当てをマークすることができる。割り当てが追い出されるときに、そのコンテキストが、システムメモリ（まだシステムメモリ内になかったと仮定して）に転送され、その後、その仮想アドレスまたはハンドルが、無効にされる。

【００７９】

仮想アドレス空間

当技術分野で既知または将来に開発される、仮想アドレス空間の提供に関する技法のいずれかを、本発明と共に使用することができる。そのようなアドレス空間を使用できる方法を示すために、共通の仮想アドレス空間技法を使用する２つの例を、本明細書で示す。コプロセッサの仮想アドレス空間を作成する複数の方法があり、当業者が本明細書の例から外挿できることを理解されたい。これに関して、可変長フラットページテーブルおよびマルチレベルページテーブルを使用する仮想アドレス空間を、本明細書で説明する。

【００８０】

可変長フラットページテーブル。可変長フラットページテーブルと連合しての本発明の使用を、図１３に示す。この方法では、コプロセッサのアドレス空間が、フラットページテーブルの使用を介して仮想化される。仮想アドレス空間を、事前定義のメモリ量、たとえば４ＫＢのページに分割することができる。仮想アドレス空間内のページごとに、１つのページテーブルが設けられ、このページテーブルには、関連する物理メモリの物理アドレスおよび位置（たとえば、Ａｃｃｅｌｅｒａｔｅｄ Ｇｒａｐｈｉｃｓ Ｐｏｒｔ（ＡＧＰ）、Ｐｅｒｉｐｈｅｒａｌ Ｃｏｍｐｏｎｅｎｔ Ｉｎｔｅｒｃｏｎｎｅｃｔ（ＰＣＩ）、またはビデオ）を指定するための識別子、たとえば６４ビットエントリが含まれる。一実施形態では、コプロセッサによってサポートされるページサイズが、任意ではなく、コプロセッサページテーブルによってシステムメモリページを参照できるようにするために、４ＫＢでなければならない。さらに、この実施形態では、コプロセッサページテーブルによって、同一のアドレス空間から、ローカルビデオメモリとシステムメモリの両方をアドレッシングできなければならない。コプロセッサは、単一のサーフェスに属するすべてのページを単一のタイプのメモリにマッピングすることを要求することができる。た

とえば、コプロセッサは、特定のレンダターゲットに属するすべてのページを、ローカルビデオメモリにマッピングすることを要求することができる。しかし、サーフェスをさまざまな物理メモリタイプ（ＡＧＰ、ローカルビデオなど）にマッピングするページテーブルエントリが、ページテーブルに存在することができる。

【００８１】

ＰＣＩアダプタおよびＡＧＰアダプタについて、各ページテーブルエントリの例示的実施形態に、３２ビットが含まれ、４ＧＢ物理アドレス空間全体をコプロセッサに可視にすることができる。ＰＣＩ－Ｅｘｐｒｅｓｓタイプのアダプタを使用する実施形態について、コプロセッサは、６４ビットアドレッシングサイクルをサポートすることができる。各ページテーブルエントリに、４０ビット以上を含めて、それぞれテラバイト単位のメモリをアドレッシングすることができる。マザーボード上の４０ビット以上の物理アドレス信号線を利用する６４ビットシステムを実施する実施形態では、対応するビデオアダプタがこのアドレス空間全体をアドレッシングできない場合に、性能ペナルティを経験する場合がある。したがって、６４ビットのフルサポートが推奨される。

【００８２】

フラットページテーブル法は、ページディレクトリがなく巨大なページテーブルだけがあることを除いて、現在ＩＮＴＥＬ（登録商標）社の８０８６（ｘ８６）ファミリＣＰＵで使用可能な仮想化機構に似ている。

【００８３】

有効な割り当てに関連しない仮想アドレスを、ダミーページにリダイレクトして、悪意のあるＤＭＡバッファが、コプロセッサに、アクセスするべきではないメモリを強制的にアクセスさせることを防ぐことができる。ハードウェアによって、エントリが有効であることを指定する有効ビットを、ページテーブルエントリのそれぞれで実施することができる。

【００８４】

ページテーブルは、関連するコプロセッサコンテキストがコプロセッサ上で現在実行中でないときに、再配置可能とすることができる。コンテキストが実行中でないときに、ＶｉｄＭｍによって、ページテーブルをシステムメモリに追い出すことができる。コンテキストが、再実行の準備ができたときに、ページテーブルをビデオメモリの、潜在的に異なる位置に戻すことができる。ドライバは、保存されたコプロセッサコンテキスト内のページテーブルの位置を更新することもできる。

【００８５】

この実施形態で、すべてのメモリアクセスを、コプロセッサ仮想アドレスを介して行うことができる。しかし、これは、本発明がそのようなアクセスを必要とすることを意味するのではない。ある要素を、他の方法でアクセスすることができ、他の方法でアクセスされる場合により高い機能性を提供することができる。仮想アドレス方式から除外できるいくつかの項目の例が、下記である。

１）ページテーブル自体を、物理アドレスを介して参照することができる。

２）陰極線管（ＣＲＴ）を、連続するメモリ範囲について物理アドレスにプログラムすることができる。

３）仮想印刷エンジン（ＶＰＥ）によって、物理アドレスに直接にＤＭＡを実行することができる。

４）オーバーレイは、物理アドレスから直接に読み取ることができる。

５）コプロセッサコンテキストを、物理アドレスを介して参照することができる。

６）プライマリリングバッファを、物理アドレスを介して参照することができる。

【００８６】

コンテキスト切り替え中に、コプロセッサによって、復元されるコンテキストによって使用される仮想アドレスを再変換することができる。これによって、これらのアドレスがコンテキスト切り替えの前と同一の物理ページを参照するという潜在的に誤った仮定をコプロセッサにさせるのではなく、メモリリソースが正しいプレーンに配置されることが保

証される。また、本発明のさまざまな実施形態に関して、単一のページテーブル内または複数のページテーブルにまたがる複数のエントリが、同一の物理ページを参照できるようにすることが有益であることにも留意されたい。

【００８７】

さまざまな実施形態で、コプロセッサによって、ページテーブルの現在のサイズを与えるリミットレジスタ（ｌｉｍｉｔ　ｒｅｇｉｓｔｅｒ）を実施することができる。ページテーブルの末尾を超えるメモリ参照のすべてが、コプロセッサによって無効なアクセスとみなされ、それなりに処理される。ページテーブルは、２のべきで拡張可能であり、一実施形態で、少なくとも２ＧＢのアドレス空間（２ＭＢのページテーブル）をサポートすることができる。

【００８８】

コプロセッサコンテキストに関連する仮想アドレス空間が断片化された場合に、ＡＰＩ、たとえばＭＩＣＲＯＳＯＦＴ（登録商標）Ｄｉｒｅｃｔ３Ｄランタイムによって、ガーベジコレクションを実行して、アドレス空間および関連するページテーブルのサイズを減らすことができる。上位仮想アドレスでの割り当てが、削除され、低位アドレスに再割り当てされる。

【００８９】

本発明と共に可変長フラットページテーブルを使用して仮想アドレス空間を実施することの長所および短所は、当業者に明白である。要約すると、フラットページテーブルを使用することの長所の１つが、物理メモリに対する１レベルの間接参照だけがあることである。もう１つの長所は、ページの不連続の組を用いてページングを解決できることである。しかし、短所もある。たとえば、コプロセッサが動作中であるときに、一般に、ページテーブル全体がメモリ内に存在する必要がある。また、ページテーブルによって、かなりの量のメモリが消費される可能性がある。ページテーブルは、一般にメモリ内のページの連続する組を必要とするので、配置に不便である可能性がある。

【００９０】

マルチレベルページテーブル。マルチレベルページテーブルと共の本発明の使用を、図１４に示す。マルチレベルページテーブルは、全般的に、可変長フラットページテーブルに類似するものとすることができるが、マルチレベルページテーブルでは、仮想アドレスのインデックス部分が、複数のテーブルにまたがって分割される。たとえば、さまざまな実施形態で、３２ビットアドレス空間を使用することができる。この情況では、ハードウェアが、２レベル間接参照を有することが要求される場合がある。間接参照の第１レベルを、ページディレクトリと呼び、第２レベルを、ページテーブルと呼ぶ。コプロセッサが特定のコンテキストを実行しているときに、リソースリスト内の割り当てが必要とするそのコンテキストのページディレクトリおよびページテーブルだけが、メモリ内に存在する必要がある。

【００９１】

本発明と共に複数レベルページテーブルを設けることの長所の１つが、ページの不連続の組を用いてページングを解決できることであることを諒解されたい。また、割り当てによって、システムメモリおよびローカルビデオメモリからのページを混合することができ、使用中のページディレクトリおよびページテーブルだけがメモリに存在する必要があり、ページディレクトリおよびページテーブルが、それぞれ１ページだけを必要とする（複数ページの連続的な割り当ては不要である）。しかし、これらの長所にもかかわらず、メモリへのアクセスが２つの間接参照を必要とするという短所が残っている。

【００９２】

サーフェスレベルフォールト

コプロセッサコンテキストごとの仮想アドレス空間の追加によって、高度なスケジューリングモデルが、適度に良好に動作し、一般に、特にメモリ圧迫がほとんどまたはまったくないときに、大きいＣＰＵオーバーヘッドを必要としない。ほとんどの場合に、ＤＭＡバッファをスケジューラにサブミットできるときに、そのバッファによって参照されるリ

ソースが、既にメモリ内に存在し、したがって、ＤＭＡバッファは、ページングスレッドによるページングをまったく必要としない。しかし、スケジューリングに関して、タイムキーピング（ｔｉｍｅ　ｋｅｅｐｉｎｇ）の精度を高めることによって、モデルをさらに改善することができる。

【００９３】

本発明を実施する際に出会う問題の１つが、特定のＤＭＡバッファの実行にどれだけの時間を要するかを前もって知ることが可能でない場合があることである。これは、準備される次のＤＭＡバッファに関するスケジューラによる潜在的に悪い選択をもたらす可能性がある。現在のコンテキスト以上の優先順位の他のコンテキストがない場合、またはその優先順位の他のすべてのコンテキストが空である場合に、スケジューラは、現在のコンテキストから次のＤＭＡバッファを選択することができる。そうでない場合には、スケジューラは、現在のコンテキスト以上の優先順位を有する次のコンテキストから次のＤＭＡバッファを選択することができる。しかし、その選択が、正確である保証はない。次に高い優先順位のコンテキストからＤＭＡバッファを選択するときに、スケジューラは、現在のコンテキストのＤＭＡバッファが、１カンタムより長く実行されると仮定することができる。そのとおりでない場合には、スケジューラは、あまりにも早くそのハードウェアコンテキストから切り替える可能性がある。現在のＤＭＡバッファが、１カンタム未満だけ実行される場合には、スケジューラは、現在のコンテキストから次のＤＭＡバッファを選択するべきであった（これによってコプロセッサの使用効率が最大になるからである）。

【００９４】

メモリ圧迫がほとんどまたはまったくないときに、一般に、次のＤＭＡバッファの潜在的な候補の両方が、既にすべてのリソースをメモリ内に有する可能性があり、したがって、どちらのバッファもページングを必要としない可能性が高い。そのシナリオでは、スケジューラが、最初のＤＭＡバッファのカンタムが終了するときに誤りを理解し、即座に考えを変え、コプロセッサに正しいＤＭＡバッファを与えることができる。

【００９５】

しかし、メモリ圧迫の下では、このモデルは安定性が下がる可能性がある。次のＤＭＡバッファの「サイズ決定」が、滑らかな動作を保証する際の有利なステップになる可能性がある。メモリ圧迫の下で、前に説明したシナリオでは、おそらく、次のＤＭＡバッファの２つの潜在的な候補の１つが、あるページングを必要とし、したがって、準備スレッドに送られる。その場合に、スケジューラが、最後に「考えを変え」、この２つのＤＭＡバッファを入れ替えることが賢明である。しかし、そのような変更を行うことができ、そのような実践が、本発明の説明の範囲から出ないことに留意されたい。たとえば、次のＤＭＡバッファの準備が完了し、他の潜在的なＤＭＡバッファ候補がページングを必要としないシナリオでは、ＤＭＡバッファを交換することができる。これは、補助メモリマネージャによる共有可能な割り当てのある特別なサポートを暗示する可能性がある。

【００９６】

上で説明した潜在的なタイムキーピングエラー自体は、それほど悪くはなく、後続カンタム中に、コンテキストに失われた処理時間を与えることによって対処することができる。また、ほとんどの場合に、ＤＭＡバッファに、複数のコプロセッサカンタムにわたる実行に十分なコマンドが含まれ、したがって、各コンテキストがそのすべてのカンタムを得ることができる。しかし、メモリ圧迫の下では、補助メモリマネージャが、各コンテキストの作業セットを減らすために、ＤＭＡバッファをより小さいバッファに分割することを強制される（上で説明したように）可能性がある。そのようなＤＭＡバッファの分割によって、ＤＭＡバッファのサイズが減り、それに対応して、上で説明したカンタム化の問題が増える。

【００９７】

メモリ圧迫の下で生じる可能性があるもう１つの問題は、潜在的に、ＤＭＡバッファによって実際に使用される、ページインされるより多くのメモリがあるので、このモデルによって人工的に余分の圧迫が引き起こされる可能性があることである。ページインされた

余分のメモリのすべてが、潜在的に、次のカンタムの前に追い出され、もう一度ページインされる必要が生じる。これは、ページングアクティビティが既に高いときに、さらに増加したページングアクティビティにつながる可能性がある。基本モデルおよび高度なモデルでは、補助メモリマネージャが、適当な追出しポリシを選択することによって、ページングの増加の問題に対処することができる。たとえば、軽いメモリ圧迫の下で、各コンテキストは、その作業セット内に適度な量のメモリを有する可能性が高い。他のコンテキストからメモリを追い出す前に、補助メモリマネージャは、まず現在のコンテキストからメモリを追い出すことと、使用可能な作業セットにおさまるようにＤＭＡバッファを分割することを試みる。特定のコンテキストのＤＭＡバッファが、その最小サイズに分割されたならば、補助メモリマネージャは、別のコンテキストからメモリを追い出さざるを得なくなる可能性がある。

【００９８】

この問題を解決するのに好ましい手法の１つが、コプロセッサが必要とするメモリのデマンドフォールト（ｄｅｍａｎｄ　ｆａｕｌｔｉｎｇ）を可能にすることである。この方法で、コプロセッサが必要とするメモリのサブセットだけがメモリに存在する必要があることを保証することができる。

【００９９】

高度なモデルについて提案されるフォールトのレベルは、サーフェス粒度のレベルである。しかし、任意のレベルのフォールトが、本発明と共に使用するのに適当である可能性があることを理解されたい。また、ページテーブルハードウェアの場合に、ハードウェアは、割り当てが有効であるかどうかを判定するために、割り当ての最初のページの状態だけを調べることができることに留意されたい。というのは、補助メモリマネージャが、割り当て全体を同時にメモリに持ち込めるからである。

【０１００】

さまざまな実施形態で、ハードウェアによって、下記のいずれかのときにページフォールトを生成することができる。

１）無効なリングバッファまたはＤＭＡバッファを参照しているコンテキストへのコンテキスト切り替えが発生する。

２）プリミティブが描かれようとしており、必要なメモリリソースの一部が存在しない（たとえば、頂点シェーダコード、頂点バッファ、テクスチャ）。

【０１０１】

第２の情況で、ハードウェアが、すべての三角形をレンダリングする前に、現在のメモリリソースを再サンプリングする必要がある場合があることに留意されたい。補助メモリマネージャが、コプロセッサの動作中を含めて、任意の時に仮想アドレスまたはハンドルを無効化する可能性がある。ハードウェアが、現在使用しているメモリリソースのすべてについて照会できるようにする可能性があることも期待される。補助メモリマネージャは、その情報を使用して、特定の割り当てがハードウェアによって使用中であるときを判定することができる。補助メモリマネージャは、割り当てが、コプロセッサによって現在使用中のリソースのリストに現れる場合に、その仮想アドレスまたはハンドルを無効化した後に、コプロセッサがその割り当てにアクセスできないので、その割り当てを安全に追い出せると仮定する可能性がある。それを行う試みによって、ページフォールトが引き起こされる可能性がある。

【０１０２】

本発明に関連するサーフェスレベルフォールトの使用のさらなる説明を、サーフェスレベルフォールトモデルの以下のより詳細な説明によって提供する。以下のモデルは、ある実施形態の例であり、本発明の潜在的な使用または本明細書で提供されるスケジューリングモデルのコンテキストの外部の他の応用例に関するサーフェスレベルフォールトの概念の潜在的な使用に対する制限と解釈するべきではない。

【０１０３】

第１に、メモリリソースに関する割り当て方式は、本明細書のコプロセッサコンテキス

トごとの仮想アドレス空間の節で説明したものと同一とすることができる。詳細についてはその節を参照されたい。

【０１０４】

第２に、ＤＭＡバッファのレンダリングコマンド方式およびリソースリストも、本明細書のコプロセッサコンテキストごとの仮想アドレス空間の節で説明したものと同一とすることができる。このモデルでは、グラフィックハードウェアがサーフェルレベルフォールトをサポートする場合であっても、リソースリストが必要である。補助メモリマネージャ（本明細書では「ＶｉｄＭｍ」）は、リソースリストを使用して、メモリ割り当てに関する使用状況情報を獲得する。その使用状況情報を用いて、ＶｉｄＭｍが、メモリ内に空間を作る必要があるときに、追出しの候補を決定することができる。

【０１０５】

サーフェスレベルフォールトの追加に関して、リソースリストに関するセキュリティの懸念事項はなく、したがって、ユーザモードでリソースリストを構築することができる。悪意のあるアプリケーションが、リソースリストに無効なデータを入れた場合に、発生し得る最悪の事態は、悪意のあるアプリケーションの性能が悪くなることである。ＶｉｄＭｍは、追出しの候補に関する非論理的な選択を行う可能性があり、これによって、そのアプリケーションに関する余分なページングアクティビティがもたらされる。

【０１０６】

サーフェスのデマンドフォールトを用いるスケジューリングモデルは、多くの点で、サーフェスレベルフォールトを使用しないモデルと異なる可能性がある。一般に、レディリスト内のプロセスは、コプロセッサに直接にサブミットでき、準備フェーズが不要である。スケジューラは、ページフォールトの解決を必要とするコンテキストの、専用のリストおよびページングスレッドを維持することができる。ページング動作に使用される、ＶｉｄＭｍ固有のコプロセッサコンテキストがある。最後に、コンテキストにサブミットされるＤＭＡバッファは、単一の作業アイテムを形成するように連結される。

【０１０７】

このモデルでは、準備フェーズを除去することができる。スケジューラは、あるコンテキストから別のコンテキストに直接に切り替えるようにコプロセッサに要求することができ、すべてのコンテキストがいつでも実行の準備ができていると仮定することができる。切り替え先のコンテキストの、一部のメモリリソースがメモリ内に存在しない場合には、ハードウェアがフォールトすることができ、コンテキストが、リスト（図１５のインページリストなど）に追加され、したがって、ページングスレッドが、そのフォールトの解決について作業を開始することができる。

【０１０８】

このモデルに関してスケジューラによって維持される例示的なプロセスのリストを、図１５に示す。図１５を参照すると、フォールトが発生するときに、フォールトを引き起こしたコンテキストを、インページリストに追加することができる。その後、ページングスレッドによって、そのフォールトが解決される。ページングスレッドによって、最初に解決される、フォールトした最高の優先順位のコンテキストを選択することができる。周期的な優先順位ブーストを使用して、低い優先順位のコンテキストが、そのフォールトを解決されるのに十分に高い優先順位を最終的に得ることを保証することができる。フォールトが、インページワーカースレッドによって解決されている間に、スケジューラによって、コプロセッサでの実行の準備ができているコンテキストをスケジューリングすることができる。コプロセッサが働いている間に、インページワーカースレッドは、ドライバを呼び出して割り当てをアドレスにマッピングするかアンマップすることによって、ビデオメモリを操作することができる。

【０１０９】

コプロセッサによって現在使用されている割り当てが、無効化される可能性がある。コプロセッサがその割り当てに次にアクセスしようとする時に、フォールトが発生しなければならない。しかし、コプロセッサは、任意の時に即座にフォールトすることができない

ので（たとえば、一部のコプロセッサは、三角形の間で現在の割り当ての状態を再サンプリングするだけである）、コプロセッサが、無効化された後のある時に割り当てを使用することを必要とする可能性がある。

【０１１０】

それが起こらないようにするために、ＶｉｄＭｍによって、次のコンテキスト切り替えまで、割り当てのメモリが、仮想アドレスまたはハンドルが無効化される場合であっても、有効のままになることを保証することができる。これは、ページングに起因するメモリ転送を、ＶｉｄＭｍ専用のコプロセッサコンテキストで行わせることによって達成することができる。メモリ転送が、別々のコンテキストで行われるので、メモリのコンテキストが変更される前にコンテキストが切り替えられることを確信することができる。システムメモリを参照する仮想アドレスまたはハンドルについて、追出し中のメモリ転送はない。その場合に、ＶｉｄＭｍによって、コプロセッサコンテキストがＶｉｄＭｍの専用コンテキストに切り替えられるまで、システムメモリをピン止めすることによってシステムメモリが有効なままになることを保証することができる。

【０１１１】

ＶｉｄＭｍ専用コプロセッサコンテキストは、システムメモリとビデオメモリの間のメモリ転送を行うのにＶｉｄＭｍによって使用される通常のコプロセッサコンテキストである。ＶｉｄＭｍコンテキストは、インページリスト内の最高優先順位のアイテムの優先順位をとる可変優先順位コンテキストである。すべてのページング動作を単一のコンテキストに直列化させることによって、ＶｉｄＭｍの同期化モデルが単純になる。

【０１１２】

このモデルでのもう１つの興味深い相違は、特定のコンテキストについてサブミットされるＤＭＡバッファのすべを連結して、単一のタスクを形成できる方法である。前のモデルでは、各ＤＭＡバッファによって、１つの作業アイテムが形成され、各コンテキストによって、これらの作業アイテムのリストが維持される。スケジューラは、必ずしもコンテキストをスケジューリングせず、コンテキストに関連する特定の作業アイテムをスケジューリングする（かつ、その準備を開始する）。作業アイテムが完了の機会を得る前に、スケジューラが、次の作業アイテムを選択しなければならない。各作業アイテムは、サブミットされる前に準備されなければならず、したがって、スケジューラは、次の作業アイテムをどれにしなければならないかを前もって知らなければならないが、これは常に可能ではない。

【０１１３】

サーフェスレベルフォールトを用いると、ＤＭＡバッファは準備の必要がない。このゆえに、スケジューラは、コンテキストを作業アイテムの集合とみなす必要がない。そうではなく、スケジューラは、実際にコンテキストをスケジューリングし、コンテキストは、コプロセッサの制御を得たならば、コプロセッサの制御を持ち続けることができる。いくつかの、例えば下記のイベントによって、プロセッサのコンテキスト制御を停止させることができる。

１）コプロセッサが、現在キューイングされているすべてのコマンドを完了する

２）コプロセッサが、無効なメモリアクセスによって引き起こされるページフォールトを生成する

３）スケジューラが、異なるコンテキストへの切り替えを要求する

４）コプロセッサが、ＤＭＡストリーム内の無効なコマンドに続いて無効動作割込みを生成する

【０１１４】

図１６に、上記による、本発明のさまざまな実施形態を示す図が与えられている。図１６を参照すると、左右に、第１のコンテキストの挿入から第２のコンテキストの挿入までの、同一のハードウェア環境での進行が示されている。左側では、スケジューラが、特定のＤＭＡバッファをコプロセッサコンテキスト＃１のリングに挿入するようにカーネルドライバに要求する。リングは、ドライバによって修正され、コプロセッサの末尾が更新さ

れて、新しい値を参照するようになる。コプロセッサコンテキスト＃１へのＤＭＡバッファの挿入は、コプロセッサコンテキスト＃１固有のロックの保護の下で行われる。したがって、他のスレッドは、他のコプロセッサコンテキストのリングにＤＭＡバッファを挿入することができる。

【０１１５】

右側では、スケジューラが、コプロセッサコンテキスト＃２のリングに特定のＤＭＡバッファを挿入するようにカーネルモードドライバに要求する。しかし、このリングは既に満杯であり、したがって、スレッドＢは、リングに空きができるまでブロックされる。スレッドＢが待つという事実によって、スレッドＡそれ自体のリングに新しいＤＭＡバッファを挿入できなくなるのではないことに留意されたい。

【０１１６】

このモデルでは、各コンテキストが、実行されるＤＭＡバッファの諸部分へのリダイレクトを含めることができるそれ自体のＤＭＡリングを有する。サブミット時に、スケジューラは、サブミットされたＤＭＡバッファをそのコンテキストのリングに追加することを試みることができる。リングが既に満杯である場合には、スケジューラは、もう１つのサブミットに十分なスペースがリングにできるまで待つことができる。この待機によって、サブミットされる特定のコンテキストへのさらなるサブミットだけがブロックされることに留意されたい。他のコンテキストへのサブミットは、ブロックされない。言い換えると、複数のスレッドが、それ自体のコンテキストに並列に作業アイテムを追加することができる。

【０１１７】

新しいＤＭＡバッファを、実行中のコンテキストのキューに追加することができるので、コプロセッサは、コンテキストが空であることを報告する割込みを生成する前に、キューの末尾を再サンプリングすることができる。もちろん、コプロセッサがキューをサンプリングした直後にＤＭＡバッファがキューに追加される可能性がある。しかし、割込み生成の直前のキューの末尾のサンプリングによって、これが発生する可能性が減り、スケジューリングの精度が高まる。スケジューラは、コンテキストが空であることを通知されたときに、実際にそうであるかどうかを知るために、ドライバに照会する。ドライバが、現在キューに入れられている、まだ処理されていないコマンドがあるかどうかを判定するために、保存されたコプロセッサコンテキストにアクセスすることができなければならない。図１７に、このモデルを記述した擬似コードアルゴリズムを示す。

【０１１８】

後で詳細に説明するように、限定ＤＭＡバッファ対特権的ＤＭＡバッファという概念を導入して、システムセキュリティを危険にさらさずにユーザモードでＤＭＡバッファを直接に構築できるようにすると同時に、カーネルモードドライバが特権的コマンドを含むＤＭＡバファを構築できるようにする。

【０１１９】

このモデルによって表されるさまざまな実施形態を、本明細書の後の節で詳細に説明する限定メモリ対特権的メモリという概念と共に使用することができる。さしあたり、前に提示したメモリ仮想化モデルで、限定ＤＭＡバッファ対特権的ＤＭＡバッファによってアクセスできるメモリの間に区別がなく、すべての仮想メモリがアクセス可能なので、このモデルで問題が生じる可能性があることに留意されたい。これは、ページテーブルまたはリングバッファなどのメモリリソースを、コプロセッサ仮想アドレス空間を介して適当に可視にすることができないことを意味する。というのは、それによって、悪意のあるアプリケーションがページテーブルまたはリングバッファを上書きできるようになるからである。このゆえに、あるタイプのリソースについて物理アドレッシングをサポートし、他のタイプのリソースについて仮想アドレッシングをサポートするハードウェアを設計することができる。

【０１２０】

この問題に対する異なる手法が、特権的メモリという概念を追加することである。さま

ざまな実施形態で、特権的メモリに、特権的ＤＭＡバッファからのみアクセスでき、コプロセッサは、限定ＤＭＡバッファから特権的メモリ位置へのアクセスが試みられる場合に、ページフォールトを引き起こすことができる。その一方で、特権的ＤＭＡバッファからは、特権的メモリおよび非特権的メモリの両方に無差別にアクセスすることができる。特権的メモリをサポートするために、ハードウェアが、ハンドルごとに（ハンドルベース仮想化の場合に）またはページごとに（ページテーブルベースの仮想化の場合に）メモリが特権的であるかどうかを指定する機構を有しなければならない。

【０１２１】

特権的メモリをサポートするために、ページテーブルを有し、サーフェスレベルフォールトをサポートするコプロセッサは、もはや、メモリリソースのベースアドレスだけに基づいてフォールトすることができない。コプロセッサは、現在のリソースに含まれるすべてのページテーブルエントリを調べ、そのすべてが正しい保護ビットをセットされていることを確認しなければならない。メモリリソースの最初のページだけを検査することによって、潜在的に、悪意のあるアプリケーションが、限定ＤＭＡバッファで指定した限定メモリベースアドレスに続く特権的メモリにアクセスできるようになる。

【０１２２】

実行リスト

前に提示したデマンドフォールトモデルは、複数のイベントをシグナリングするために割込みを激しく使用する可能性がある。ページフォールトなど、これらのイベントの一部は、メモリ圧迫の下で非常に頻繁に発生する可能性がある。割込みが発生する時と、コプロセッサがＣＰＵによって新しいタスクを与えられる時の間に、コプロセッサが長時間停止状態を生じる可能性がある。割込み待ち時間を隠蔽し、コプロセッサを動作状態に保つために、実行リストという概念が導入された。

【０１２３】

実行リストは、単に、ＣＰＵの介入なしにコプロセッサによって実行することができるコプロセッサコンテキストのリストである。コンテキストは、与えられた順序で、または本発明の実践に好都合であることがわかった任意の順序で実行することができる。コプロセッサは、たとえば下記など、本発明と共に実施できるさまざまな理由のいずれかについて、実行リスト内のあるコンテキストから次のコンテキストに切り替えることができる。

１）現在のコンテキストが空である、すなわち、行うべきことが残っていない。

２）現在のコンテキストが、ページフォールトを生成した。

３）現在のコンテキストが、一般保護フォールト（コプロセッサによってサポートされる場合に）を生成した。

４）コプロセッサが、新しい実行リストに切り替えることを要求された。

【０１２４】

さまざまな実施形態で、コプロセッサは、実行リスト内のあるアイテムから次のアイテムに切り替えるときに、ＣＰＵに割り込むが、停止せず、リスト内の次のアイテムにコンテキストを切り替えることができ、その実行を開始することができる。実行リストの先頭は、スケジューラが最初に実行を試みることができるコンテキストとすることができ、実行リストの他の要素は、部分的に、割込み待ち時間中にコプロセッサを動作状態に保つために置くことができる。ＣＰＵは、コプロセッサがリストの先頭から他に切り替えたことを知らせる割込みを受け取るや否や、新しい実行リストを構築し、コプロセッサに送ることができる。

【０１２５】

コプロセッサは、リストの先頭から切り替えるときに、生成した割込みがＣＰＵに進む間に、実行リストの次のコンテキストの実行を開始することができる。ＣＰＵが生成する新しい実行リストの先頭は、コプロセッサが切り替えた先のコンテキストと異なる場合がある。その場合には、コプロセッサは、もう一度切り替える必要があり、そのコンテキストで有用な作業を行う時間がない場合がある。

【０１２６】

しかし、コンテキスト優先順位が、最後の実行リストが構築されてから変更されていないので、ＣＰＵによって構築される新しい実行リストの先頭のコンテキストを、前の実行リストの第２要素と同一にすることができる。その場合には、コプロセッサは、すでに正しいコンテキストの処理を意外に早く開始している。

【０１２７】

実行リストの概念を表す図を、図１８に示す。実行リストが、本発明のさまざまな実施形態に含まれるときに、スケジューラの実行中コンテキストを、現在の実行リストによって置換することができる。保留中実行リストと称する第２の実行リストを導入して、実行リスト切り替えの同期化を単純にする。現在の実行リストは、ハードウェアが現在実行しているとスケジューラが仮定することができるコンテキストのリストであり、保留中実行リストは、スケジューラがある実行リストから別の実行リストにハードウェアを変更することを望むときに使用される過渡的な実行リストである。スケジューラは、新しい実行リストに切り替えることを望むときに、保留中実行リストを構築し、それに切り替えるようにコプロセッサに要求する。スケジューラが、コプロセッサが新しい実行リストの実行を開始したことの確認をコプロセッサから（割込みを介して）受け取ったならば、保留中実行リストが、新しい現在の実行リストになり、保留中実行リストを空にすることができる。

【０１２８】

保留中実行リストが空であるときに、ハードウェアは、現在の実行リストのコンテキストを実行しているか、アイドルである可能性がある。保留中実行リストが空でないときには、スケジューラは、遷移が行われたことの確認をコプロセッサから受け取るまで、ハードウェアが現在どの実行リストを実行しているかを知らない可能性がある。

【０１２９】

ある種のイベントは、スケジューラが実行リストの優先順位を付けなおすことを必要とする場合がある。たとえば、ページフォールトが解決され、高い優先順位のコプロセッサコンテキストの実行の準備ができる場合がある。そのようなイベントの同期化を単純にするために、スケジューラが従うことのできる一般的なルールは、前のイベントによってサブミットされた保留中実行リストがまだない場合に限って、新しい実行リスト（保留中実行リスト）をサブミットすることである。ある保留中リストの別の保留中リストへの置換の試みは、同期化が難しい。というのは、リストが、既にコプロセッサに与えられており、したがって、遷移がいつでも発生する可能性があり、スケジューラが、その事実の後でなければ通知されないからである。

【０１３０】

後者の場合に、実行リストの優先順位の付けなおしを、コンテキスト切り替えハンドラに委譲することができる。将来のある点で、保留中リストから実行中のリストへの遷移をシグナリングするためにハンドラを呼び出すことができ、その時点で、ハンドラが、優先順位が変更された場合に、新しい実行リストを生成して、ハードウェアに送ることができる。

【０１３１】

実行リスト切り替えの同期化。ある実行リストモデルでは、グラフィックスハードウェアが、コンテキストを切り替えるときに割込みを生成することができる。割込みの引渡および処理は、瞬間的ではないので、ＣＰＵが実際に割り込まれる前に複数の割込みが生成される可能性がある。同期化が正しく行われない場合に、スケジューラが混乱し、不正なスケジューリング判断を行う可能性がある。

【０１３２】

区別するようにスケジューラに指示することができる２つのクリティカルなイベントが、第１に、コプロセッサが実行リストの先頭から他に切り替えるときと、第２に、コプロセッサが保留中実行リストに切り替えるときである。このイベントを区別することは、各コンテキスト切り替えにおける単純な割込みからの情報だけでは困難になる可能性がある。この点をさらに示すために、次の例を検討されたい：コプロセッサが、現在、コンテキ

スト１－３－５－２からなる実行リストＡを実行しており、スケジューラが、コンテキスト４－１－３－２からなる実行リストＢに変更することを望む。下記の２つのシナリオが発生する可能性がある：

シナリオ＃１

コプロセッサが、現在、実行リストＡ（１－３－５－２）を実行している。

コンテキスト４に関するコマンドがサブミットされるが、コンテキスト４は、アイドルであり、コンテキスト１より高い優先順位である。実行リストＢ（４－１－３－２）が生成され、スケジューラが、実行リストＢをコプロセッサにサブミットする。

コンテキスト＃１は、コプロセッサが、実行リストＢからのコンテキスト＃４に遷移するまで実行される。

コプロセッサが、遷移をシグナリングする割込みを生成する。

コプロセッサが、コンテキスト＃４から＃１に遷移し、その後、ＣＰＵが割り込まれる前に＃３に遷移する。

ＣＰＵが、割り込まれ、コンテキスト切り替えハンドラが呼び出される。

ドライバが、現在のコプロセッサコンテキストをサンプリングするが、これは＃３である。

【０１３３】

シナリオ＃２

コプロセッサが、現在、実行リストＡ（１－３－５－２）を実行している。

コンテキスト４に関するコマンドがサブミットされるが、コンテキスト４は、アイドルであり、コンテキスト１より高い優先順位である。スケジューラが、実行リストＢをコプロセッサにサブミットする。

スケジューラが、実行リストＢを構築するのにビジーである間に、コプロセッサがコンテキスト＃３に遷移する。

コプロセッサが、コンテキスト＃３への遷移をシグナリングする割込みを生成する。

ＣＰＵが、割り込まれ、コンテキスト切り替えハンドラが呼び出される。

ドライバが現在のコプロセッサコンテキストをサンプリングするが、これは＃３である。

【０１３４】

どちらの場合でも、コンテキスト切り替え割込みの時の現在実行されているコンテキストは、＃３である。しかし、スケジューラは、追加情報がなければ、この２つのシナリオを区別できないことに留意されたい。第１のシナリオでは、コプロセッサが、実行リストＢの先頭から切り替え、したがって、スケジューラは、実行リストＣを生成し、それに切り替えるようにコプロセッサに要求する必要がある。しかし、第２のシナリオでは、第２の実行リストがまだ開始されておらず、したがって、スケジューラは待たなければならない。

【０１３５】

上の例から、コンテキスト切り替え割込みだけでは、スケジューリングモデルで実行リストを正しくサポートするのに十分でない場合があることが示される。このシナリオを区別するために、さらなる情報が必要である。次の節で、この問題に対処するいくつかの方法を、そのような問題に対処するのに有用にすることができるハードウェアサポートと共に詳細に説明する。

【０１３６】

２要素実行リスト。この同期化手法は、コプロセッサがある追加機能をサポートすることを必要とする。２要素実行リストの実施形態に関してサポートできる特徴の中に、下記がある。

１）２要素の実行リスト。

２）各コンテキスト切り替え（コンテキストＸからＸへのにせのコンテキスト切り替えを含む）に割込みを生成する能力。

３）ＶｉｄＭｍが任意の時に現在実行されているコプロセッサコンテキストを照会する

方法。

４）割込みの前に、進行中のコプロセッサコンテキストをメモリに保存すること。

５）スケジューラがコンテキスト切り替えの理由を判定できるようにするためにＣＰＵによってコンテキストが可読になる方法でのコプロセッサコンテキストの保存。

【０１３７】

ハードウェアを使用して上の機能をサポートすることができるが、スケジューラが通常のコンテキスト切り替えと実行リスト切り替えを区別できるようにするのに、そのような特殊なハードウェアが必要ではないことに留意されたい。代わりに、スケジューラは、実行リストを構成するときに単純なルールの組を必ず守ることによって、この２つのイベントを区別することができる。特定のルールは、本発明のさまざまな実施形態について変化する可能性があるが、この機能をもたらす例示的なルールは、第１に、現在の実行リストの最初のコンテキストが、新しい保留中実行リストに現れることができないことと、第２に、現在の実行リストの第２のコンテキストが、新しい保留中実行リストの先頭でない場合に、新しい保留中実行リストにこれを含めてはならないことである。下に、この２つの例示的ルールに従うときに、スケジューラが、あるコンテキストから別のコンテキストへの遷移中に行う仮定の表を示す。下の表では、実行リストＡが、コンテキスト１－２からなり、第２の実行リストＢが、コンテキスト２－３からなり、第３の実行リストＣが、コンテキスト３－４からなる。

【０１３８】

【表６】

AからBへの遷移	
CPUが割り込まれるときの現在のコンテキスト番号	意味/行われるアクション
1	グリッチ。割込みを無視する。 このグリッチは、実行リスト切り替えの際にXから1への遷移と誤って解釈された、実行リスト（X、1）から（1*、Y）への前の切り替えによって引き起こされたものである。実際の遷移は、Xから1、その後に1から1*である。現在の割込みは、遷移1から1*に関し、無視することができる（コプロセッサは、スケジューラが1-Xから1-Yへの遷移を検出できるように、この割込みを生成することを要求される）。
2	実行リスト切り替えが発生した。 これは、必ずしも真ではなく、前のグリッチにつながる可能性がある。現在の遷移が、実際に1*-2*である場合には、CPUが、遷移2*-2**または2*-3**についてもう一度割り込まれる。保留中実行リスト（B）が、現在の実行リストになり、保留中実行リストが、空にされる。スケジューラは、コンテキスト1*からのコンテキスト切り替え（たとえばページフォールト）を処理する必要がある。
3	実行リスト切り替えが発生し、第2リストの先頭が既に完了している。 実行リストBが終了した。保留中実行リスト（B）が、現在の実行リストになる。新しい保留中実行リストが、スケジューラによって構築され、コプロセッサに送られる。スケジューラは、コンテキスト1*および2**からのコンテキスト切り替え（たとえばページフォールト）を処理する必要がある。

【０１３９】

【表７】

AからCへの遷移	
CPUが割り込まれるときの現在のコンテキスト番号	意味/行われるアクション
1	グリッチ。割込みを無視する。 このグリッチは、実行リスト切り替えの際にXから1への遷移と誤って解釈された、実行リスト（X、1）から（1*、Y）への前の切り替えによって引き起こされたものである。実際の遷移は、Xから1、その後に1から1*である。現在の割込みは、遷移1から1*に関し、無視することができる（コプロセッサは、スケジューラが1-Xから1-Yへの遷移を検出できるように、この割込みを生成することを要求される）。
2	現在の実行リスト内でのコンテキスト切り替え。 コプロセッサが、コンテキスト2*に切り替える。スケジューラは、1*からのコンテキスト切り替え（たとえばページフォールト）を処理する必要があるが、それ以外では、実行リストに関して何もしない。
3	実行リスト切り替えが行われた。 保留中実行リスト（C）が現在の実行リストになり、保留中実行リストが空にされる。スケジューラは、1*からのコンテキスト切り替え（たとえばページフォールト）を処理する必要がある。コンテキスト2*が実行されたかどうかは未知であり、再スケジューリングされる。
4	実行リスト切り替えが行われ、第2リストの先頭が既に完了している。 実行リストCが終了した。保留中実行リスト（C）が現在の実行リストになる（ハードウェアはアイドルであるが）。新しい保留中実行リストが、スケジューラによって構築され、コプロセッサに送られる。コンテキスト2*が実行されたかどうかは未知であり、再スケジューリングされる。

【０１４０】

実行リストを実施するこの方法は、おそらく最も単純であり、必ずしもかなりの追加ハードウェアサポートを必要としない。しかし、上の表の実行リストが、サイズを制限され（２を超えるサイズの拡張は、実用的でなくなる可能性がある）、クリティカルでないいくつかの情報が、コンテキスト切り替え中に失われる可能性があることに留意されたい。たとえば、スケジューラが、必ずＡからＣへの遷移でコンテキスト＃２が実行されたかどうかを知るとは限らない。それが実行され、ページフォールトを引き起こしたが、別のコンテキスト切り替えによって割込みを隠される可能性がある。その場合に、スケジューラは、コンテキスト＃２がフォールトを生成したことを知らず、再スケジューリングする。

【０１４１】

スケジューリングイベントのコプロセッサトレース。実行リストは、スケジューリング情報のあるヒストリ情報がハードウェアによってスケジューラに供給されるときに、簡単にサイズＮまで拡張することができる。単純な割込みの問題の１つが、複数の割込みが一緒に強制される可能性があることであり、何がおきて割込みが引き起こされたかを正確に

判定できない可能性があることである。これは、本発明の方法と共に、ハードウェア機能によって対処することができる。より具体的には、コンテキスト切り替えヒストリを、スケジューラによって可読の指定されたシステムメモリ位置に書き込むことができるハードウェアを実施することによって対処することができる。本発明のこの態様を説明するために、下記のシナリオを検討されたい。

【０１４２】

１）スケジューラが、実行リストＡ（１－２－３－４－５）をスケジューリングする。

２）コンテキスト＃１のタイムカンタムが満了し、スケジューラが、新しい実行リストＢ（２－３－４－５－１）を送る。

３）ＣＰＵでカンタム満了を処理している間に、コプロセッサが、コンテキスト＃１が空になるのでコンテキスト＃１について完了し、したがってコンテキスト＃２に遷移する。コプロセッサは、このイベントについてコンテキスト切り替え割込みを生成した。

４）コプロセッサが、新しい実行リストに関する通知をＣＰＵから受け取り、したがって、それに遷移した。コプロセッサが、このイベントに関するコンテキスト切り替え割込みを生成した。

５）新しい実行リストのコンテキスト＃２のレンダリングコマンドを処理している間に、コプロセッサが、ページフォールトに出会い、したがって、コンテキスト＃３に遷移した。コプロセッサが、このイベントに関するコンテキスト切り替え割込みを生成した。

６）コンテキスト＃３が、ページフォールトを発生し、したがって、コプロセッサが、コンテキスト＃４に切り替えた。コプロセッサが、このイベントに関するコンテキスト切り替え割込みを生成した。

７）ＣＰＵが、最終的にコンテキスト切り替えについて割り込まれる。最初の割込みが送出されて以来、４つのコンテキスト切り替えが実際に発生した。

【０１４３】

図１９に、上のシナリオでのハードウェアヒストリ機構の動作を示す。そのようなヒストリ機構をサポートするために、下記のタスクを実行できるようにハードウェアを構成することができる。これらのタスクは、制限ではなく例として提供される。

【０１４４】

１）ヒストリバッファのベースアドレスを指定する。コプロセッサごとに単一のヒストリバッファを設けることができる。好ましい実施形態では、これを、ＰＣＩメモリまたはＡＧＰメモリのいずれかのシステムメモリ位置とすることができる。これを、オペレーティングシステムによって４ＫＢ境界に整列することができる。ＰＣＩ　ｅｘｐｒｅｓｓシステムについて、このバッファへのアクセスは、スヌープサイクルを用いて実施することができ、したがって、システムメモリバッファを、より効率的なＣＰＵ読取のためにキャッシュ可能にすることができることが好ましい。

【０１４５】

２）ヒストリバッファのサイズを指定する。ヒストリバッファは、実行リストのサイズの少なくとも２倍の長さとすることができる。これは、現在の実行リストおよび保留中実行リストの両方が、割込みが発生する前に完了する、ワーストケースシナリオを処理するのに十分なスペースがバッファ内にあることを保証するためである。

【０１４６】

３）コプロセッサ書込ポインタを指定するが、これはヒストリバッファに書き込まれた最後のイベントの直後のアドレスとすることができる。ＶｉｄＭｍは、コプロセッサが稼動している時を含めて、いつでもこのポインタを照会できるものとすることができる。スケジューラが常にコヒーレントなデータを得ることを保証するために、ポインタが更新される前に、ヒストリバッファ内のデータをメモリに正しくフラッシュすることができる。

【０１４７】

さまざまな実施形態によって、ユーザモードで構築されるＤＭＡバッファから不可視になるようにヒストリバッファを構成することができる。ヒストリバッファが、限定ＤＭＡバッファから可視の場合には、悪意のあるアプリケーションが、ヒストリバッファを上書

きし、スケジューラを壊し、おそらくはシステムクラッシュまたはさらに悪い結果をもたらすことができる。このゆえに、これらの実施形態のヒストリバッファは、物理アドレスを介してハードウェアによって参照される、または特権的ＤＭＡバッファだけに可視の仮想アドレスを介して参照されるのいずれかとすることができる。これらの実施形態では、コプロセッサが、ＣＰＵ介入なしでヒストリバッファの末尾でラップアラウンドすることを要求することができる。

【０１４８】

上で説明した実施形態による実行リストによって、コプロセッサが同一の理由で同一のコンテキストで複数回フォールトできるようするための必要のすべてが除去されるわけでないことに留意されたい。これの理由の１つが、スケジューラが、一般に、コプロセッサが現在の実行リストの実行でビジーである間に新しい実行リストを構築することである。スケジューラは、前の実行リストに既に存在するコンテキストを新しい実行リストに含める必要がある場合があるので、繰り返されるコンテキストの状態が、構築される実行リストに置かれた時と、実行リストがコプロセッサにサブミットされた時の間に変化する可能性があることがありえる。

【０１４９】

限定ＤＭＡ対特権的ＤＭＡ

高度なスケジューリングモデルでのメモリ保護の導入に伴って、コプロセッサに送られるＤＭＡバッファが、主として、実行中のアプリケーションのプロセス内部のユーザモードドライバによって構築される可能性がある。これらのＤＭＡバッファは、アプリケーションのプロセス内でマッピングされる可能性があり、ユーザモードドライバは、これらに直接に書き込むことができるが、カーネルドライバは、これを確認することができない。ＤＭＡバッファは、誤ってその仮想アドレスにアクセスするアプリケーションによって、または悪意のあるアプリケーションによって故意に、書き込まれる可能性がある。ドライバモデルがセキュアであることを可能にするために、すなわち、アプリケーションが、するべきではないリソースにアクセスできなくするために、ユーザモードで構築されるＤＭＡバッファを、行うことが許可されるものに関して制限することができる。具体的に言うと、ユーザモードで構築されるＤＭＡバッファは、下記の例示的な方法で制限された機能性を有することができる。

【０１５０】

１）仮想アドレスへの参照だけを含めることができ、物理アドレスへの参照はまったく（フェンスを含めて）含めることができない。

２）現在のディスプレイ（たとえば、ＣＲＴ、随意アクセス制御（ＤＡＣ）、Ｔｅｃｈｎｉｃａｌ Ｄｏｃｕｍｅｎｔ Ｍａｎａｇｅｍｅｎｔ Ｓｙｓｔｅｍ（ＴＤＭＳ）、Ｔｅｌｅｖｉｓｉｏｎ－Ｏｕｔ Ｐｏｒｔ（ＴＶ－ＯＵＴ）、Ｉｎｔｅｒ－Ｉｎｔｅｇｒａｔｅｄ Ｃｉｒｃｕｉｔ（Ｉ２Ｃ）バス）に影響する命令を含めることを許可されることができない。

３）アダプタ全般（たとえば位相ロックループ（ＰＬＬ））に影響する命令を含めることができない。

４）限られた電源管理および／または構成スペースを有することができる。

５）コンテキスト切り替えを防ぐ命令を含めることを許可されることができない。

【０１５１】

ユーザモードで構築されるＤＭＡバッファでプログラム可能なレジスタの正確な組は、ハードウェアによって変化する可能性が高い。しかし、ハードウェアにかかわらずに、レジスタが一般的な規則に従うことができる、すなわち、そのようなＤＭＡバッファは、リソースおよびフェンスへの仮想アドレス参照を使用するレンダリング動作だけを許可しなければならない。強化されたセキュリティをもたらすために、そのようなＤＭＡバッファに、アプリケーションがアクセスするべきではないメモリ、またはある潜在的に破滅的で回復不能な方法でハードウェアに影響することができるメモリをアプリケーションが使用することを許可しないことを要求することができる。

【０１５２】
ユーザモードで構築されるＤＭＡバッファが、ある機能性にアクセスできなくするために、複数の手法をコプロセッサ内で実施することができる。これらの手法は、機能性の性質および機能性がアプリケーションのコプロセッサコンテキストストリームにキューイングされる必要があるかどうかに非常に依存する。ある種の特権的動作は、一般に、ユーザモードで構築されるＤＭＡバッファ（たとえばアプリケーションレンダリング）およびカーネルモードで構築されるＤＭＡバッファ（たとえばキューイングされたフリップ）の両方を含むコプロセッサコンテキストストリーム内でキューイングされることを必要とする。

【０１５３】
キューイングされる必要がない機能性。ほとんどの特権的機能性は、アプリケーションコプロセッサコンテキストストリームでキューイングされることを必要としない。下記などの機能性は、キューイングされることを必要としない。
１）ＣＲＴタイミングのプログラミング。
２）ＤＡＣのルックアップテーブルの更新（ＤＡＣ　ＬＵＴのプログラミングが、特権的機能性である必要が絶対にないことに留意されたい。というのは、どのアプリケーションでも、望むならば、プライマリスクリーンにレンダリングすることができ、ルックアップテーブル（ＬＵＴ）の再プログラミングによって、そうでなければアクセスできない情報へのアクセスをアプリケーションがユーザに与えられるようにはならないからである。
３）ディスプレイ出力のプログラミング（ＴＤＭＳ、ＴＶ－ＯＵＴ、…）
４）子デバイス／モニタとの通信（Ｉ２Ｃ、…）
５）クロックのプログラミング（ＰＬＬ）
６）コプロセッサの電源状態の変更
７）コプロセッサの構成（構成スペース、ＢＩＯＳ、…）

【０１５４】
この機能性は、通常は、アプリケーションレンダリングストリームと完全に独立のシステムイベント（たとえば、ブート、解像度変更、ｐｎｐ検出、電源管理）の後に必要になる。したがって、この機能性は、特定のアプリケーションのコプロセッサコンテキストでキューイングされることを必要としない。この機能性は、特定のシステムイベントが発生しているときに、ユーザモードドライバからの介入を一切伴わずに、カーネルモードドライバ自体によって使用することができる。

【０１５５】
そのような機能性について、ＩＨＶは、メモリマップ式入出力（ＭＭＩＯ）だけを介して基礎になるレジスタのすべてをアクセス可能にすることを決定することができる。レジスタは、一般に、カーネル空間だけにマッピングされるので、アプリケーションまたはユーザモードドライバがそれにアクセスすることを不可能にすることができ、したがって、この機能性が有効に保護される。

【０１５６】
もう１つの手法は、コプロセッサコンテキストごとの特権レベルを実施することである。この手法では、コンテキストに、行えることを制限されるものと、そうでないものがある。そのシナリオでは、ユーザモードで構築されるアプリケーションのＤＭＡバッファが、限定コンテキストにキューイングされる。その一方で、カーネルモードドライバは、特権的コンテキストを使用して、特権的機能性をサブミットすることができる。

【０１５７】
キューイングされることを必要とする機能性。ユーザモードで構築されるＤＭＡバッファに挿入できるコマンドは限られているので、コプロセッサが限定ＤＭＡバッファ（前の条件を遵守するＤＭＡバッファ）および特権的ＤＭＡバッファの両方をサポートすることを要求するように、高度なモデルを実施することができる。特権的ＤＭＡバッファは、コプロセッサコンテキストのレンダリングストリームに沿った特権的機能性のキューイングを可能にするために必要である。

【０１５８】
特権的ＤＭＡバッファには、非特権的ＤＭＡバッファに見られる命令のどれでも含めることができる。本発明のさまざまな好ましい実施形態によって、下記（後の節で詳細に説明する）を少なくとも可能にする特権的ＤＭＡバッファを実施することができる。
１）特権フェンスの挿入
２）フリップ命令の挿入
３）「コンテキスト切り替えなし」領域の挿入
【０１５９】
さらに、特権的ＤＭＡバッファによって、ＩＨＶが望むすべてのハードウェアレジスタをプログラムすることができ、必要な場合に仮想メモリおよび物理メモリの両方にアクセスすることができる。特権的ＤＭＡバッファは、ユーザモードで構成することができず、可視にすることもできない。信頼されるカーネルコンポーネントだけが、特権的ＤＭＡバッファにアクセスでき、これを構築することができる。
【０１６０】
次の節に、特権的ＤＭＡバッファを実施する可能な３つの方法を示すが、次の節は、本発明を実施できるさまざまな方法を制限せずに、特権的ＤＭＡバッファの実施という概念を説明することを意図されたものである。
【０１６１】
１．ＤＭＡバッファは、カーネルモードでのみ構築される
特殊なハードウェアサポートを必要としない特権的ＤＭＡバッファをサポートする方法の１つが、ハードウェアに送られる実際のＤＭＡバッファをカーネルモードで構築することを要求することである。このシナリオでは、ユーザモードドライバが、ＤＭＡバッファによく似たコマンドバッファを構築し、カーネルモードドライバにサブミットする。カーネルモードドライバは、このコマンドバッファを確認し、カーネルモードでのみ可視のＤＭＡバッファにコピーする。確認中に、カーネルモードドライバは、特権的命令が存在しないことを検証する。これは、基本モデルで要求される確認に似ているが、メモリが仮想化されるので、メモリアクセスに関する確認は不要である。
【０１６２】
２．リングへの直接の特権的コマンドの挿入
おそらく、特権的ＤＭＡチャネルをサポートする最も簡単なハードウェア手法は、特権的コマンドをコプロセッサコンテキストリングに直接に挿入することである。リング自体は、既に特権的チャネルであり、カーネルモードからのみアクセス可能である。これを、図２０の図に示す。
【０１６３】
３．間接参照を介する特権の指定
コプロセッサ内で限定ＤＭＡバッファ対特権的ＤＭＡバッファをサポートする異なる手法を、図２１に示す。これを参照して、開始アドレスおよび終了アドレスの両方をＤＷＯＲＤに整列できることに留意されたい。アドレスの未使用ビットは、フラグの指定に再利用することができる。開始アドレスの第１ビットによって、リダイレクトされるＤＭＡバッファが特権的ＤＭＡバッファであることを指定することができる。セキュリティを強化するために、特権的ＤＭＡバッファによって、補助メモリ内の物理アドレスを参照することができる。限定ＤＭＡバッファによって、コプロセッサコンテキスト仮想アドレス空間内の仮想アドレスを参照することができる。
【０１６４】
この手法では、間接参照コマンドの１ビットを、リングバッファに挿入することができる。このビットによって、実行されるＤＭＡバッファが特権的ＤＭＡバッファであるか否かが示される。これは、リングバッファ自体を、コプロセッサによって物理アドレスを使用して参照することができ、コプロセッサ仮想アドレス空間内で不可視にすることができることを意味する。プライマリリングバッファをコプロセッサ仮想アドレス空間内で可視にすることを許可することによって、悪意のあるアプリケーションが、プライマリリング

バッファを上書きできるようになり、特権レベルでコマンドを実行できるようになり、これは、実質上、ほとんどのコンピューティング環境でのセキュリティ侵害と等しい。これに関して、特権的ＤＭＡバッファを、物理アドレスを介して参照し、限定ＤＭＡバッファのように仮想アドレスで可視でなくすることができる。

【０１６５】

ＤＭＡ制御命令

スケジューラおよび補助メモリマネージャが、コプロセッサコンテキストの進行を追跡し、そのコンテキストのＤＭＡストリーム内の命令のフローを制御するために、ＤＭＡストリーム内で下記の例示的命令をサポートするようにコプロセッサを構成することができる。

１）フェンス（限定および特権的の両方）

２）トラップ

３）コンテキスト切り替えの許可／禁止

【０１６６】

フェンス。フェンスは、ＤＭＡストリームに挿入することができる、あるデータ（たとえば６４ビットのデータ）およびアドレスの両方を含む命令とすることができる。この命令が、コプロセッサによってストリームから読み取られるときに、コプロセッサが、フェンスに関連するデータを指定されたアドレスに書き込む。コプロセッサは、フェンスのデータをメモリに書き込む前に、フェンス命令に先立つプリミティブからのピクセルが、回収され、既にメモリに正しく書き込まれていることを保証しなければならない。これが、コプロセッサがパイプライン全体を停止させることを必要とすることを意味しないことに留意されたい。フェンス命令に続くプリミティブは、コプロセッサがフェンスの前の命令の最後のピクセルが回収されるのを待っている間に実行することができる。

【０１６７】

上の説明にあてはまるフェンスのすべてを、本発明と共に使用することができるが、特に２タイプのフェンスすなわち、通常のフェンスおよび特権的フェンスを本明細書でさらに説明する。

【０１６８】

通常のフェンスは、ユーザモードドライバによって作成されたＤＭＡバッファに挿入できるフェンスである。ＤＭＡバッファの内容は、ユーザモードから来るので、信頼されない。したがって、そのようなＤＭＡバッファ内のフェンスは、そのコプロセッサコンテキストのアドレス空間の仮想アドレスを参照することができるが、物理アドレスを参照することはできない。そのような仮想アドレスへのアクセスが、コプロセッサによってアクセスされる他の仮想アドレスと同一のメモリ確認機構によって束縛されることは言うまでもない。

【０１６９】

特権的フェンスは、カーネルモードで作成された（カーネルモードでのみ可視の）ＤＭＡバッファに挿入することだけができるフェンスである。そのようなフェンスは、システムのセキュリティを高めるために、メモリ内の物理アドレスを参照することができる。フェンスのターゲットアドレスが、コプロセッサコンテキストのアドレス空間内で可視である場合には、悪意のあるアプリケーションが、そのメモリ位置に対してグラフィック動作を実行でき、したがって、カーネルモードコードが受け取ることを期待する内容をオーバーライドすることができる。潜在的なセキュリティ問題に対するもう１つの解決策が、非特権的ＤＭＡバッファから仮想アドレスにアクセスできるかどうかを示す、ＰＴＥ内の特権ビットを有することである。しかし、上の第１の手法は、早期のハードウェア生成についてより単純と思われる。

【０１７０】

特権的ＤＭＡバッファに、通常のフェンスと特権的フェンスの両方を含めることができることに留意されたい。しかし、特権的ＤＭＡバッファに通常のフェンスが含まれるときに、ＤＭＡバッファを生成するカーネルコンポーネントに、それが挿入したフェンスが絶

対に可視にならないことがわかっている。

【０１７１】

ＩＨＶは、フラッシュを必要とする内部バッファの数を最小にするために、余分なタイプのフェンスをサポートすると決定することができる。下記のタイプのフェンスは、この目的のためにサポートすることができるフェンスの例である（すべてのタイプについて、特権的および非特権的の両方をサポートしなければならないことに留意されたい）：

１．書込フェンス

書込フェンスは、前に説明したタイプのフェンスであり、唯一の必要なフェンスタイプである。書込フェンスによって、フェンス命令の前に処理されるすべてのメモリ書込が、グローバルに可視になる（すなわち、それらがキャッシュからフラッシュされ、メモリコントローラから肯定応答が受け取られている）ことが保証される。

【０１７２】

２．読取フェンス

読取フェンスは、書込フェンスに似た、より軽いタイプのフェンスである。読取フェンスによって、フェンスの前のレンダリング動作に関するメモリ読取のすべてが終了することが保証されるが、一部の書込が、まだ未解決である可能性がある。読取フェンスがサポートされる場合には、スケジューラは、これを使用して、非レンダターゲット割り当ての寿命を制御する。

【０１７３】

３．トップオブパイプ（ｔｏｐ－ｏｆ－ｐｉｐｅ）フェンス

トップオブパイプフェンスは、非常に軽量のフェンスである。トップオブパイプフェンスのサポートは、任意選択である。トップオブパイプフェンスによって、ＤＭＡバッファ内でフェンス命令の前の最後のバイトが、コプロセッサによって読み取られた（必ずしも処理されていない）ことだけが保証される。コプロセッサは、フェンスが処理された後に（そのＤＭＡバッファンの内容がもはや有効でなくなるので）、ＤＭＡバッファのうちでトップオブパイプフェンスに先立つ部分を再読み取りしない場合がある。サポートされる場合に、このタイプのフェンスは、ＤＭＡバッファの寿命を制御するのにスケジューラによって使用される。

【０１７４】

トラップ。トラップは、本発明のさまざまな実施形態で実施することができる。トラップは、コプロセッサによって処理されるときにＣＰＵ割込みを生成できる、ＤＭＡバッファに挿入された命令とすることができる。コプロセッサがＣＰＵに割り込む前に、トラップ命令に先立つプリミティブからのすべてのピクセルが、回収され、メモリに正しく書き込まれていることを保証する（フェンス命令からのメモリ書込を含むことができる動作）ことが賢明である。これが、コプロセッサがパイプライン全体を停止させる必要があることを意味しないことに留意されたい。トラップ命令に続くプリミティブは、コプロセッサがトラップの前の命令の最後のピクセルが回収されるのを待っている間に実行することができる。

【０１７５】

トラップ命令は、特権的命令である必要はなく、ユーザモードドライバによって直接に構築されるものを含むすべてのＤＭＡバッファに挿入することができる。

【０１７６】

コンテキスト切り替えの許可／禁止。サブトライアングル（ｓｕｂ－ｔｒｉａｎｇｌｅ）割込みをサポートするハードウェアについて、コンテキスト切り替えを許可または禁止する命令を設けることができる。コンテキスト切り替えが禁止されている間は、コプロセッサは、一般に、現在のコプロセッサコンテキストから切り替えてはならない。コプロセッサは、ＣＰＵによって新しい実行リストが提供される場合に現在の実行リストの情報を更新することを要求される場合があるが、コプロセッサは、コンテキスト切り替えが再度許可されるまで、新しい実行リストへのコンテキスト切り替えを延期することができる。ＯＳは、コンテキスト切り替えが禁止されているときに下記のルールが残ることを保証す

ることができる。
１）特権的ＤＭＡバッファだけが処理される。
２）コンテキスト切り替え命令がＤＭＡストリームに存在しない。
３）ＤＭＡストリームの命令が使い果たされない。
４）ページフォールトが発生しない（ページレベルフォールトがサポートされる場合）。

【０１７７】
多くのコンピュータシステムで、コンテキスト切り替えの禁止および許可は、特権的ＤＭＡバッファ内にのみ存在することができる特権的命令である。これらの命令の使用のシナリオでは、割り込まれる可能性なしで、スケジューラがスクリーンに現れる動作（すなわちプレゼンテーションブリット）をスケジューリングできるようになる。そのような動作で割り込まれることは、容易に目につく時間期間にわたる画面上の可視のアーチファクトにつながる可能性がある。

【０１７８】
コプロセッサが、ＤＭＡバッファ内で不測のエラーに出会う場合に、コンテキスト切り替えが禁止されていても、このＤＭＡバッファからコンテキストを切り替えることができることに留意されたい。カーネルモードで構築されるＤＭＡバッファだけに、割込み不能部分を含めることができるので、不測のエラーは、ドライバのバグまたはハードウェアのバグの結果である。コプロセッサが、これらのシナリオでコンテキストを切り替えない場合には、ディスプレイウォッチドッグがハングを把握し、システムを回復するためにコプロセッサをリセットする。

【０１７９】
任意選択の制御命令。スケジューラは、上で説明した単純な制御命令を用いて高水準同期化プリミティブを構築することができるが、その結果を、さらに効率的にすることができる。多くのコンピュータシステムで、コプロセッサコンテキストは、同期化オブジェクトの所有権を得る前にＣＰＵによって割り込まれる。同期化オブジェクトが高い頻度で取られ、解放される場合に、これが問題になる可能性がある。より効率的な同期化プリミティブを有するために、スケジューラは、コプロセサから特殊な命令を受け取ることができる。具体的に言うと、「ウェイト」命令および「シグナル」命令を適当な時に送出するようにコプロセッサを構成することができる。

【０１８０】
ウェイト命令は、指定されたカウンタの値を検査できることをコプロセッサに知らせるためにＤＭＡストリームに挿入される。カウンタが非ゼロの場合には、コプロセッサは、カウンタを減分し、現在のコプロセッサコンテキストの実行を継続する。カウンタがゼロの場合には、コプロセッサは、現在のコプロセッサコンテキストの命令ポインタをウェイト命令の前にリセットし、実行リストの次のコンテキストに切り替える。コプロセッサコンテキストが、ウェイト命令で停止する必要があり、後に再スケジューリングされるときに、コプロセッサは、そのウェイト命令を再実行することができる。というのは、ウェイト条件がまだ満足されていない可能性があるからである。

【０１８１】
ウェイト命令は、１つのパラメータすなわち、比較／減分されるメモリ位置を指定する仮想アドレスを有することだけを必要とする。カウンタは、少なくとも３２ビットとすることができ、任意の有効な仮想アドレスとすることができる。好ましい実施形態では、ウェイト命令を割込み不能とすることができる、すなわち、新しい実行リストがコプロセッサに与えられる場合に、コプロセッサは、ウェイト命令の前またはそれを実行した後のいずれかに新しい実行リストに切り替えることができる。ウェイト命令は、限定ＤＭＡバッファおよび特権的ＤＭＡバッファの両方に挿入することができる。

【０１８２】
シグナル命令は、カウンタの値を更新できることをコプロセッサに知らせるためにＤＭＡストリームに挿入することができる。コプロセッサは、カウンタの値を１つ増分するこ

とができる。コプロセッサは、加算中の潜在的なオーバーフローを無視することができる。その代わりに、コプロセッサが、ソフトウェアバグを追跡するのを助けるために、ストリーム内のエラーとしてオーバーフローを報告することができる。

【０１８３】

シグナル命令は、１つのパラメータすなわち、更新されなければならないカウンタの仮想アドレスだけを有することを必要とする。カウンタサイズは、ウェイト命令のカウンタサイズと一致するようにすることができ、好ましい実施形態では、少なくとも３２ビットである。シグナル命令は、限定ＤＭＡバッファおよび特権的ＤＭＡバッファの両方に挿入することができる。

【０１８４】

フリップ

全画面アプリケーションが、パイプライン内のバブルなしでシームレスに動作できるようにするために、コプロセッサによって、フリップ（すなわち、ディスプレイのベースアドレスの変更）をキューイングする命令を提供することができる。ディスプレイサーフェスは、一般に、物理モデルから連続的に割り当てられ、仮想アドレスではなく物理アドレスを使用してＣＲＴＣによって参照される。したがって、フリップ命令を使用して、ＣＲＴＣを、表示される新しい物理アドレスにプログラムすることができる。これは、物理アドレスであって仮想アドレスではないので、不良アプリケーションが、潜在的に、補助メモリのうちで別のアプリケーションまたはユーザに属する部分（秘密が含まれる可能性がある）を表示するようにＣＲＴＣをプログラムすることができる。このため、フリップ命令を実施して、宛先が確認されてからカーネルモデルドライバによってのみＤＭＡストリームに挿入される特権的命令であることを保証することによって、ほとんどのコンピュータシステムのセキュリティを保護することができる。

【０１８５】

フリップ機能と共に使用される本発明のさまざまな好ましい実施形態で、少なくとも２つのタイプのフリップ、すなわち、即時フリップおよびディスプレイリフレッシュと同期したフリップがサポートされる。コプロセッサは、即時フリップを処理するときに、可視のティアリング（ｔｅａｒｉｎｇ）が引き起こされる場合であっても、ディスプレイのベースアドレスを即座に更新することができる。コプロセッサは、同期式フリップを処理するときに、新しいベースアドレスをラッチすることができるが、次の垂直同期期間まで更新を延期する。垂直同期期間の間にコプロセッサによって複数の同期式フリップが処理される場合に、コプロセッサは、最新のフリップだけをラッチし、前のフリップを無視することができる。

【０１８６】

同期式フリップを処理するときに、コプロセッサがグラフィックスパイプラインを停止させないように、さまざまな実施形態を構成することができる。ＯＳは、現在可視のサーフェスに描画するレンダリングコマンドがリングバッファにキューイングされないことを保証する。ここで、下でさらに説明する「最適化されたフリップ」の情況で、これらの要件なしで他の実施形態を構成できることに留意されたい。

【０１８７】

どのサーフェスが現在可視であるかを判定するために、ドライバは、まず、特定のキューイングされたフリップが行われたときを判定することができ、そのイベントについてスケジューラに通知する、すなわち、ディスプレイベースアドレスが変更された後にスケジューラに通知することができる。即時フリップについて、フリップが行われたときを判定するのは簡単である。というのは、ＤＭＡストリームからのフリップ命令の読取を、ディスプレイサーフェスの更新と同一のイベントとみなすことができるからである。ＤＭＡストリーム内でフリップ命令の後にフェンスおよび割込みを挿入して、特定のフリップが読み取られたことをスケジューラに通知することができる。

【０１８８】

同期式フリップの場合には、どのサーフェスが現在可視であるかの判定が、より困難で

ある。コプロセッサは、まず、ＤＭＡストリームからフリップ命令を読み取るが、後に、次の垂直同期期間にディスプレイサーフェスを更新する。その時にコプロセッサを停止させる必要をなくすために、ディスプレイサーフェス変更が有効になるときをスケジューラに通知する機構を設けることができる。

【０１８９】

本発明と共に使用されるそのような通知の機構を設計する複数の方法がある。１つの潜在的に単純な手法を、図２２に示す。図２２に、現在のディスプレイサーフェスに関してコプロセッサに照会する方法を示す。図示の実施形態では、この機能を、ＭＭＩＯレジスタによって提供されるものと考えることができる。図２２のシステムは、最新の「ラッチ式ディスプレイサーフェス」ではなく、レジスタによって実際のディスプレイサーフェスが読み取られるときに、より高い信頼性につながる設計である。最新のラッチされたディスプレイサーフェスの照会は、コプロセッサが別のキューイングされたフリップを処理する、競争状態をもたらす可能性があり、これは画面のティアリングにつながる可能性がある。フリップ命令は、適当な技法を使用して生成することができる。本発明との互換性に関する唯一の一般的な要件は、実施される解像度によって、フリップが、有効になるまで肯定応答されないことを保証することである。

【０１９０】

フリップのキューイング。最大限の性能を実現するために、高度なスケジューリングモデルを修正して、モニタを所有するアプリケーションのレンダリングストリームにフリップ動作をキューイングすることができる。ｎバッファリングを行うときに、スケジューラは、ｎ－１個までのフリップをＤＭＡストリームにキューイングすることを許可でき、ｎ番目のフリップが挿入されようとするときにブロックすることができる。

【０１９１】

これが意味するのは、ダブルバッファリングにおいて、スケジューラが、アプリケーションが１つのフリップをキューイングするのを許可することができ、コプロセッサが現在のフレームのレンダリングおよびそのフリップの処理／肯定応答を終了する間に次のフレームのＤＭＡバッファの準備を継続させることができることである。これは、アプリケーションが、次のフレームのＤＭＡバッファの準備を終了し、第２のフリップをサブミットする時まで、第１のフリップがコプロセッサによって肯定応答されるまでアプリケーションをブロックできることも意味する。

【０１９２】

即時フリップが、スケジューラによって使用されるときに、フリップをキューイングする機構は、上で説明したように働く。しかし、同期式フリップを使用するときには、スケジューラは、フリップｎ－１を超えてキューイングされるＤＭＡバッファも特別に処理することができる。実際、フリップを超えるＤＭＡバッファは、一般に、現在可視のサーフェスにレンダリングされる。ほとんどのシステムで、現在キューイングされているフリップの数がｎ－２以下になるまで、これらのＤＭＡバッファを処理しないことが賢明である。

【０１９３】

この問題に対処する最も単純な手法は、ｎ－１個ではなく、ｎ－２個のフリップだけをキューイングできるようにすることである。しかし、この解決策は、ダブルバッファリングの場合に、フリップをキューイングできず、したがって、対応するフリップが処理されるまで、各フレームが完了した後にアプリケーションをブロックする必要があることも意味する。

【０１９４】

この設定での好ましい手法を、図２３に示す。図からわかるように、ｎ－１個のフリップのキューイングが、許可される。フリップｎ－１の後にキューイングされたＤＭＡバッファが実行されないようにするために、スケジューラは、そのコプロセッサコンテキストについて仮想リングバッファにＤＭＡバッファを累積することができる。スケジューラは、現在キューイングされているフリップの数がｎ－２に減るまで待って、これらのフリッ

プをそのコプロセッサコンテキストの実際のリングにサブミットする。

【０１９５】

同時に複数のアプリケーションが動作しているときに、コプロセッサは、図２３に示されているように停止する必要がない場合がある。コプロセッサは、一般に、特定のコプロセッサコンテキストからのＤＭＡバッファの処理を停止するが、スケジューラは、実行される他のコプロセッサコンテキストをスケジューリングすることができ、効果的にコプロセッサをビジーに保つ。しかし、単一のアプリケーションが実行されているとき、たとえば、全画面ゲームをプレイしているときに、コプロセッサは、これらのインターバル中に停止する可能性がある。次の節で、サポートされる場合に停止時間を減らすためにスケジューラによって使用される機構を説明する。

【０１９６】

最適化されたフリップ。全画面アプリケーションについて最適化することを試みて、コプロセッサが停止するのに要する時間を最小限に減らすことが望まれる。図２３を参照して、コプロセッサが、少なくとも２つの理由で停止する可能性があることを観察されたい：その理由は、第１に、フレームが完了したが、システムがフリップのためにｖｓｙｎｃを待っているため、第２に、フリップが完了したが、システムがＣＰＵに通知するための割込みを待っているため、である。

【０１９７】

第１の理由に起因する停止を減らすために、より多くのバッファを、フリップのチェーンに追加することができる。たとえば、ダブルバッファからトリプルバッファに移ることによって、そのような停止が大幅に減る。しかし、これを行うことは、必ずしもドライバの制御下ではなく、不当なメモリ消費がもたらされる場合がある。

【０１９８】

第２の理由に起因する停止を減らすために、この停止の必要を完全に除去するためにコプロセッサ機構を追加することが可能である。コプロセッサは、前にキューイングされたフリップが処理されるまでコプロセッサを停止させるウェイトオンフリップ命令を提供することができる。そのような命令がサポートされるときに、スケジューラは、フリップをキューイングするために全画面アプリケーションについてその命令を使用することができ、ＣＰＵは、各フリップの後でＤＭＡストリームを再開する必要がなくなる。

【０１９９】

高水準の同期化オブジェクト

前に定義した制御命令を使用して、スケジューラによって、クリティカルセクションおよびミューテクスなどの高水準の同期化オブジェクトを構築することができる。スケジューラは、待機の条件が満たされたならば、ＣＰＵによって明示的に再スケジューリングされるまで、選択されたＤＭＡバッファを実行されないように保護することによって、そのような同期化プリミティブを実施することができる。オブジェクトの待機は、フェンスなど、スケジューラによって実施することができる。論理的にフェンスに続くＤＭＡバッファを、スケジューラによってキューイングすることができるが、待機条件が満たされるまで、コプロセッサコンテキストのリングにサブミットすることはできない。オブジェクトを待つようになったならば、シグナリングされるまで、スケジューラによってコプロセッサコンテキストをその特定のオブジェクトのウェイトリストに移動することができる。コプロセッサコンテキストＤＭＡストリームに、割込みコマンドが続くフェンスを挿入することによって、オブジェクトにシグナリングすることができる。そのような割込みを受け取るときに、スケジューラは、どのオブジェクトがシグナリングされているかを識別し、待っているコプロセッサコンテキストをレディキューに戻さなければならないかどうかを判定することができる。コプロセッサコンテキストをレディキューに戻すときに、スケジューラは、リングからしまわれたＤＭＡバッファを挿入する。

【０２００】

たとえば、アプリケーションが、生産者（プロデューサ）と消費者（コンシューマ）の間で共有されるサーフェスを有し、アプリケーションが、リソースへのアクセスを同期化

し、その結果、コンシューマがレンダリング中に必ず有効な内容を使用するようにする必要がある、本発明の実施形態を検討されたい。このシナリオを同期化する１つの潜在的な方法を、図２４に示す。

【０２０１】

図２４に移ると、スケジューラ側で、同期化を、たとえば下記のカーネルサンクスを介して実施することができ、このカーネルサンクは、任意の組合せでまたは他のアクションと組み合わせて実施することができる。

１）ＣｒｅａｔｅＳｙｎｃｈｒｏｎｉｚａｔｉｏｎＯｂｊｅｃｔ：同期化オブジェクトのカーネル追跡構造体を作成する。後続の待機／解放／削除呼出しで使用することができるオブジェクトへのハンドルをユーザモードに返す。

２）ＤｅｌｅｔｅＳｙｎｃｈｒｏｎｉｚａｔｉｏｎＯｂｊｅｃｔ：前に作成されたオブジェクトを破棄する。

３）ＷａｉｔＯｎＳｉｎｇｌｅＯｂｊｅｃｔ／ＷａｉｔＯｎＭｕｌｔｉｐｌｅＯｂｊｅｃｔ：現在のコプロセッサコンテキストのＤＭＡストリームに同期化待機イベントを挿入する。待たれているオブジェクトへの参照と共にイベントをスケジューライベントヒストリに挿入する。

４）ＲｅｌｅａｓｅＯｂｊｅｃｔ／ＳｉｇｎａｌＯｂｊｅｃｔ：現在のコプロセッサコンテキストのＤＭＡストリームに同期化シグナルイベントを挿入する（フェンス／割込み）。解放されるかシグナリングされるオブジェクトへの参照と共にイベントをスケジューライベントヒストリに挿入する。

【０２０２】

図２４の例をミューテクスに適用すると、コプロセッサが、ＤＭＡストリーム内の同期化イベントを処理したならば、スケジューラは、下記のアクションを実行することができ、これらのアクションも、任意の組合せでまたは他のアクションと組み合わせて実施することができる。

１）待機時に：ミューテクスの状態を検査する。ミューテクスが現在とられていない場合には、ミューテクスをとり、コプロセッサスレッドをスケジューラのレディキューに戻す。ミューテクスがすでに取られている場合には、コプロセッサスレッドをそのミューテクスの待機キューに入れる。

２）シグナリング時に：他のコプロセッサスレッドがミューテクスを待っているかどうかを検査する。他のいくつかのスレッドが待っている場合には、リストの最初の待っているスレッドをとり、スケジューラのレディリストに戻す。待っているスレッドがない場合には、ミューテクスをとられていない状態に戻す。

【０２０３】

この機構を使用して、スケジューラが構築することができる。たとえば、スケジューラによって構築できる下記のタイプの同期化プリミティブを検討されたい。

ミューテクス：１時に１つのコプロセッサスレッドだけが、共有リソースへのアクセス権を有することができる

セマフォ：指定された数のコプロセッサスレッドが、同時に共有リソースへのアクセス権を有することができる。

通知イベント：複数のコプロセサスレッドが、別のコプロセッサスレッドからのシグナルを待つことができる。

【０２０４】

いくつかのシナリオで、アプリケーションを構成して、コプロセッサがレンダリング命令の処理を終えたときの通知を要求することができる。これをサポートするために、スケジューラは、ドライバが、それがサブミットしているＤＭＡバッファに関する通知を要求できるようにすることができる。ドライバは、サブミット時にＣＰＵ同期化イベントを指定することができ、このイベントは、サブミットされたＤＭＡバッファについてコプロセッサが終了した後にシグナリングすることができる。スケジューラは、所与のコプロセッサコンテキストのリングに所与のＤＭＡバッファを挿入することができ、その後、リング

にユーザモードコプロセッサイベント通知（フェンスとそれに続く割込み）を追加することができる。コプロセッサイベントがコプロセッサによって処理されるときに、スケジューラは、関連するＣＰＵ同期化イベントをシグナリングすることができる。

【０２０５】

スケジューライベントヒストリバッファ

スケジューラは、上で説明した同期化機構を複数の目的に使用することができる。割込みによってコプロセッサが停止しないので、ＣＰＵは、通知のサブセットだけを見る必要があり、したがって、一部の通知を、一緒に圧搾することができる。ＤＭＡバッファ内のすべての通知に正しく応答するために、スケジューラは、挿入されたイベントのヒストリを、これらのイベントを処理するのに必要なすべてのパラメータと共に維持することができる。

【０２０６】

イベントヒストリバッファは、単に、スケジューラ処理を必要とし、そのコンテキストのＤＭＡストリームに挿入されたすべてのイベントを追跡するイベント情報構造のコプロセッサコンテキストごとの配列とすることができる。スケジューラフェンスが、スケジューラによってイベントを同期化するのに使用されるフェンスであることに留意されたい。コプロセッサコンテキストごとに１つのフェンスを設けることができ、セキュリティを保つために、フェンスを、特権的命令を介してのみ更新が許可されるようにすることができる。いずれの場合でも、そのようなイベントは、フェンス命令およびそれに続く割込み命令としてＤＭＡストリームに挿入することができる。

【０２０７】

各フェンス割込み時に、スケジューラは、まず、現在のフェンスを判定し、次に、イベントヒストリバッファを調べて、どのイベントが発生したかを判定することができる。この判定は、関連するフェンスに基づいて行うことができる。スケジューラは、フェンス割込みの処理に進むことができる。図２５に、イベントヒストリバッファのさまざまな実施形態を示す。

【０２０８】

任意の数のイベントをサポートすることができる。下の表に、現在サポートされているイベントの一部を示すが、この表は、潜在的にサポートされるイベントの数またはタイプを制限することを意図されたものではない。

【０２０９】

【表８】

イベントのタイプ	説明およびパラメータ
DMAバッファの終り	このイベントは、DMAバッファの終りに挿入される。このイベントがスケジューラによって処理されるときに、関連するDMAバッファが、そのプロセスのDMAバッファプールに戻される。 パラメータ：プールに解放される必要があるDMAバッファへのハンドル。
同期化オブジェクトを待つ	このイベントは、コプロセッサスレッドが、イベントの状況を検査し、潜在的にそれを待つ必要があるときに挿入される。スケジューラは、このイベントを処理するときに、待機条件が既に満足されているかどうかを検査し、そうである場合には、停止したばかりのコプロセッサスレッドを再スケジューリングする。待機条件が満足されない場合には、コプロセッサスレッドが、待機状態にされ、同期化オブジェクトの待機キューに追加される。 パラメータ：待たれるオブジェクトへのハンドル。
シグナル同期化オブジェクト	このイベントは、コプロセッサスレッドが、通知オブジェクトをシグナリングするか同期化オブジェクトを解放する必要があるときに挿入される。スケジューラは、このイベントを処理するときに、オブジェクトの状況を変更し、潜在的に、イベントを待っていたコプロセッサスレッドをウェイクアップする。 パラメータ：解放されるオブジェクトへのハンドル。
ユーザモードイベント通知	このイベントは、ユーザモードドライバが、レンダリング完了の通知を要求するときに挿入される。スケジューラは、このイベントを処理するときに、関連するイベントをシグナリングする。 パラメータ：シグナリングされるイベント。

【０２１０】

プログラマブルＰＣＩアパーチャ

現在のコプロセッサは、ＰＣＩ仕様によって許可される限度に非常に近いところまでＰＣＩアパーチャを公開している。将来の世代のコプロセッサは、アパーチャを介して公開できるものより多くのオンボードの補助メモリを有する。したがって、将来に、すべての補助メモリが同時にＰＣＩアパーチャを介して可視になると仮定することができなくなる。

【０２１１】

この制限を回避できる複数の方法がある。コプロセッサコンテキストごとの仮想アドレス空間をサポートする高度なスケジューリングモデルに好ましい方法は、補助メモリのどこにでも４ＫＢ粒度でリダイレクトできるＰＣＩアパーチャを使用することである。これを図２６に示す。

【０２１２】

図２６に示された、ＰＣＩアパーチャページテーブルは、コプロセッサページテーブルと独立とすることができる。コプロセッサ自体があるコンテキストから別のコンテキストに切り替えつつある間に、複数のＣＰＵプロセスが動作し、ＰＣＩアパーチャの一部にアクセスしている可能性がある。ＰＣＩアパーチャのページテーブルは、すべてのプロセッ

サコンテキストの間の共有リソースであり、補助メモリから割り当てられる。ドライバは、マップ／アンマップＤＤＩを提供して、補助メモリマネージャＶｉｄＭｍが、動作中のアプリケーションの間でＰＣＩアパーチャアドレス空間を管理できるようにすることができる。ＰＣＩアパーチャのページテーブルを、コプロセッサによって、物理アドレスを使用して参照することができる。

【０２１３】

ＰＣＩアパーチャが、アドレス空間をローカル補助メモリだけにリダイレクトするように構成される可能性があることに留意されたい。ＶｉｄＭｍが、そのアパーチャを介するのではなく、必ずシステムメモリを直接にマッピングするので、アドレス空間をシステムメモリにリダイレクトする必要はない。

【０２１４】

ページレベルフォールト

前に説明したサーフェスレベルフォールトは、ほとんどの場合に良好に働くことができるが、改善できるシナリオがある。たとえば、サーフェスレベルフォールトを使用すると、非常に大きいデータセットを使用するあるアプリケーションが、同時にメモリ内にデータセット全体を置くことができず、したがって正しく機能しない場合がある。高度なモデルで実施できるこれに対する解決策が、ページレベルフォールト機構である。

【０２１５】

ページレベルフォールトを用いると、モデルは、前の節で説明したものと同様に働く。主要な相違は、ページフォールトがＶｉｄＭｍに報告され、ＶｉｄＭｍによって処理される方法にある。サーフェスレベルフォールトでは、コプロセッサが、進行するために必要とするリソースのリスト全体を指定する必要がある（あるリソースのページングが別の必要なリソースを追い出すことを意味する無限ループを除去するために）場合があるが、ページレベルフォールトに対し、コプロセッサが仮想アドレスのリストを公開する必要がない。ページレベルフォールトでは、コプロセッサは、フォールトした仮想アドレスを報告するだけでよい。ＶｉｄＭｍは、このアドレスがどの割り当ての一部であるかを見つけ、この特定のページだけが常駐にされる必要があるかどうか、またはあるプリフェッチが必要であるかどうかを判断することができる。複数のページが、単一ピクセルによって要求されるときには、複数のフォールトがその単一ピクセルに関して生成される可能性がある。また、そのピクセルによって要求されるページが、別のページがページインされるときに追い出される可能性がある。しかし、アプリケーションの作業セットが、１ピクセルに必要になる可能性があるページの最大個数より十分に大きい限り、ページフォールトを介するループの可能性は、非常に低い。

【０２１６】

最後に、本明細書に記載のさまざまな技法を、ハードウェアまたはソフトウェア、あるいは、適当な場合にこの両方の組合せに関して実施できることを理解されたい。したがって、本発明の方法および装置、あるいはそのある態様または部分が、フロッピ（Ｒ）ディスク、ＣＤ－ＲＯＭ、ハードドライブ、または他のマシン可読記憶媒体などの有形の媒体で実施されるプログラムコード（すなわち命令）の形式をとることができ、プログラムコードが、コンピュータなどのマシンにロードされ、これによって実行されるときに、そのマシンが、本発明を実施する装置になる。プログラム可能コンピュータでのプログラムコード実行の場合に、コンピューティングデバイスに、一般に、プロセッサ、プロセッサによって読取可能な記憶媒体（揮発性および不揮発性のメモリおよび／または記憶要素を含む）、少なくとも１つの入力デバイス、および少なくとも１つの出力デバイスが含まれる。たとえばデータ処理ＡＰＩ、再利用可能なコントロール、および類似物など、本発明のユーザインターフェース技法を実施または使用することができる１つまたは複数のプログラムは、コンピュータシステムと通信するための、高水準手続き型プログラミング言語または高水準オブジェクト指向プログラミング言語で実施されることが好ましい。しかし、望まれる場合に、プログラムをアセンブリ言語またはマシン語で実施することができる。どの場合でも、言語は、コンパイルされる言語または解釈されるものとすることができ、

ハードウェア実施形態と組み合わされる。

【０２１７】

例示的実施形態では、独立型コンピュータシステムに関する本発明の使用に言及したが、本発明は、それに制限されるのではなく、ネットワーク環境または分散コンピューティング環境など、任意のコンピューティング環境に関して実施することができる。さらに、本発明は、複数の処理チップまたはデバイスの中またはこれにまたがって実施することができ、記憶は、同様に複数のデバイスにまたがって行うことができる。そのようなデバイスに、パーソナルコンピュータ、ネットワークサーバ、ハンドヘルドデバイス、スーパーコンピュータ、あるいは、自動車または航空機などの他のシステムに一体化されたコンピュータを含めることができる。

【０２１８】

したがって、本発明は、単一の実施形態に制限されるのではなく、請求項による幅および範囲において解釈されなければならない。

【図面の簡単な説明】

【０２１９】

【図１】コプロセッサに関する処理をスケジューリングする従来技術の手法を示す概念図である。

【図２】本発明によるコプロセッサスケジューリングの改善を示す例示的な図である。

【図３】概念的に図２に示されたスケジューリングの改善をもたらすのに用いられるコンピューティングコンポーネントを示すより詳細な図である。

【図４（Ａ）】図３のステップを機能シーケンスに組み合わせるさまざまな非制限的な可能な方法を示す擬似コードアルゴリズムを示す図である。

【図４（Ｂ）】図３のステップを機能シーケンスに組み合わせるさまざまな非制限的な可能な方法を示す擬似コードアルゴリズムを示す図である。

【図５】スケジューラが、提供される情報を使用して、本発明による直接メモリアクセス（ＤＭＡ）バッファ内で使用されるメモリリソースのタイムラインを定義する方法を示す図である。

【図６】本発明による準備ワーカースレッドと補助メモリマネージャの間の動きを示すアルゴリズムを示す図である。

【図７】本発明による、ページングバッファを準備し、ページングバッファに関するＣＰＵ前処理を処理するワーカースレッドを示すページングバッファの準備を示す例示的な図である。

【図８】本発明による、ページングバッファ内のフェンスの処理を含む、ワーカースレッド内で行うことができるイベントのチェーンを表すアルゴリズムを示す図である。

【図９】コプロセッサコンテキストの仮想アドレス空間を提供でき、さまざまなコプロセッサコンテキストの間で物理メモリを管理してコプロセッサコンテキストがメモリの公平な分け前を得られるようにすることができる、カーネルモードの補助メモリマネージャ「ＶｉｄＭｍ」を示す図である。

【図１０】本発明による基本的なスケジューリングモデルを示す図である。

【図１１】本発明による高度なスケジューリングモデルを示す図である。

【図１２（Ａ）】高度なスケジューリングモデルを実施することができるアクションのシーケンスを例示的に示す図である。

【図１２（Ｂ）】高度なスケジューリングモデルを実施することができるアクションのシーケンスを例示的に示す図である。

【図１３】可変長フラットページテーブルと共に本発明の使用を示す図である。

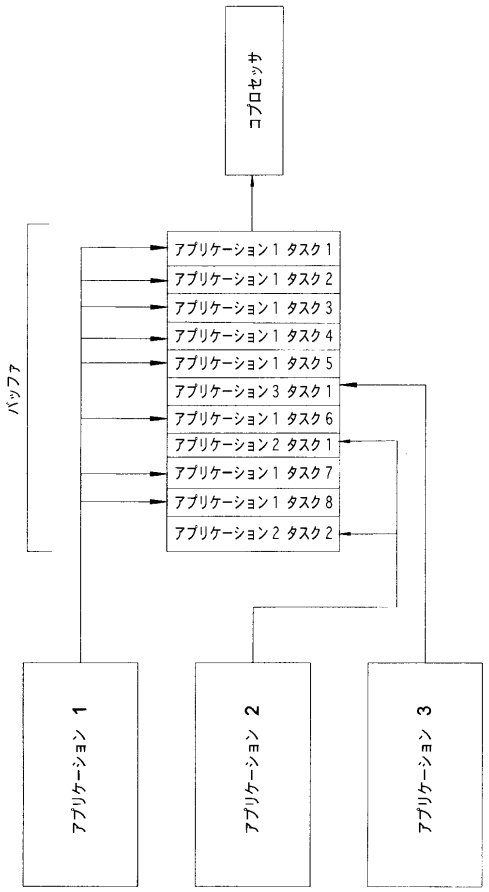
【図１４】マルチレベルページテーブルと共に本発明の使用を示す図である。

【図１５】サーフェスレベルフォールトをサポートする高度なスケジューリングモデルと共にスケジューラによって維持される例示的処理を示す図である。

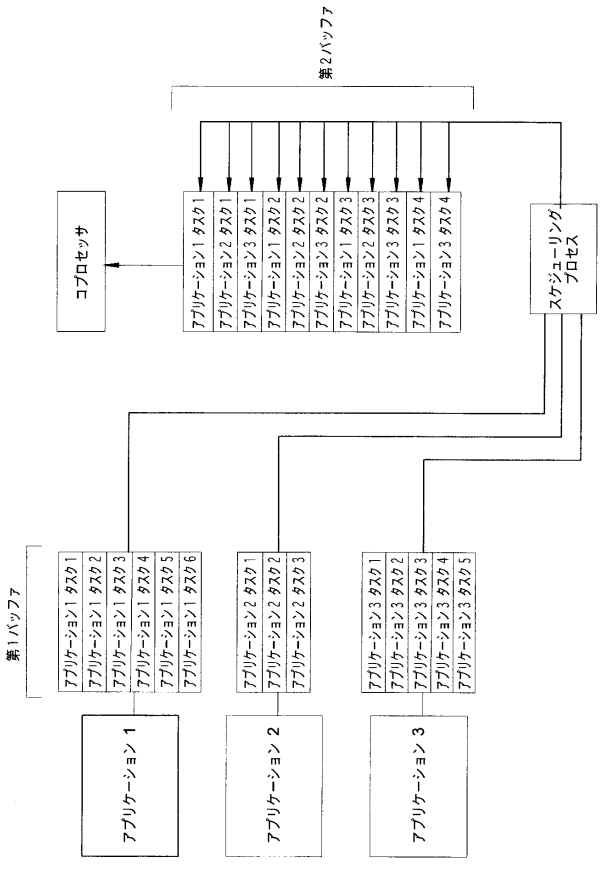
【図１６】本発明と共にサーフェスレベルフォールトが実施されるときに同時に処理することができる、それぞれがそれ自体のＤＭＡリングを有する複数のコンテキストを示す図

である。
【図１７（Ａ）】図１６の構成要素と共に本発明の動作を示す、有用であることを示すことができるさまざまな追加の特徴を含む擬似コードアルゴリズムを示す図である。
【図１７（Ｂ）】図１６の構成要素と共に本発明の動作を示す、有用であることを示すことができるさまざまな追加の特徴を含む擬似コードアルゴリズムを示す図である。
【図１７（Ｃ）】図１６の構成要素と共に本発明の動作を示す、有用であることを示すことができるさまざまな追加の特徴を含む擬似コードアルゴリズムを示す図である。
【図１８】本発明による実行リストの使用を概念的に著す図である。
【図１９】本発明に従って使用されるスケジューラによって読取可能な指定されたシステムメモリ位置にコンテキスト切り替えヒストリを書き込むことができるハードウェアの動作を示す図である。
【図２０】特権的コマンドをコプロセッサコンテキストリングに直接に挿入することによって特権的ＤＭＡチャネルをサポートするハードウェア手法を示す図である。
【図２１】間接参照コマンドのビットがリングバッファに挿入される、コプロセッサでの限定ＤＭＡバッファ対特権的ＤＭＡバッファのサポートの手法を示す図である。
【図２２】現在のディスプレイサーフェスに関してコプロセッサに照会する方法を示す図である。
【図２３】即時フリップが本発明と共に使用されるときのフリップのキューイングに好ましい手法を示す図である。
【図２４】２つまたはそれ以上のプロセッサがレンダリング中に有効な内容を使用できることを保証するためにリソースへのアクセスを同期化する例示的な技法を示す図である。
【図２５】イベントヒストリバッファのさまざまな実施形態を示す図である。
【図２６】補助メモリ内のどこにでもリダイレクトできるＰＣＩアパーチャを使用する、コプロセッサコンテキストごとの仮想アドレス空間をサポートする好ましい方法を示す図である。

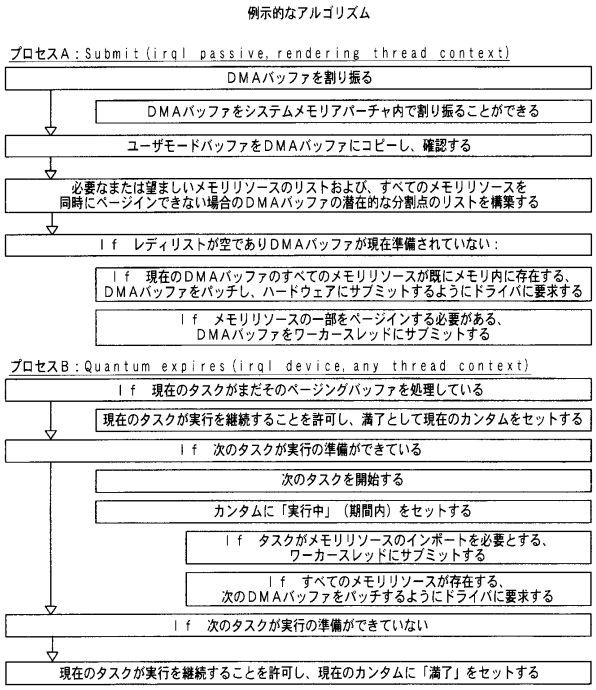
【図１】



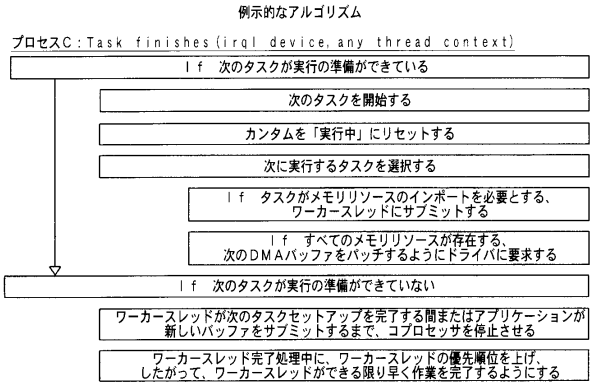
【図２】



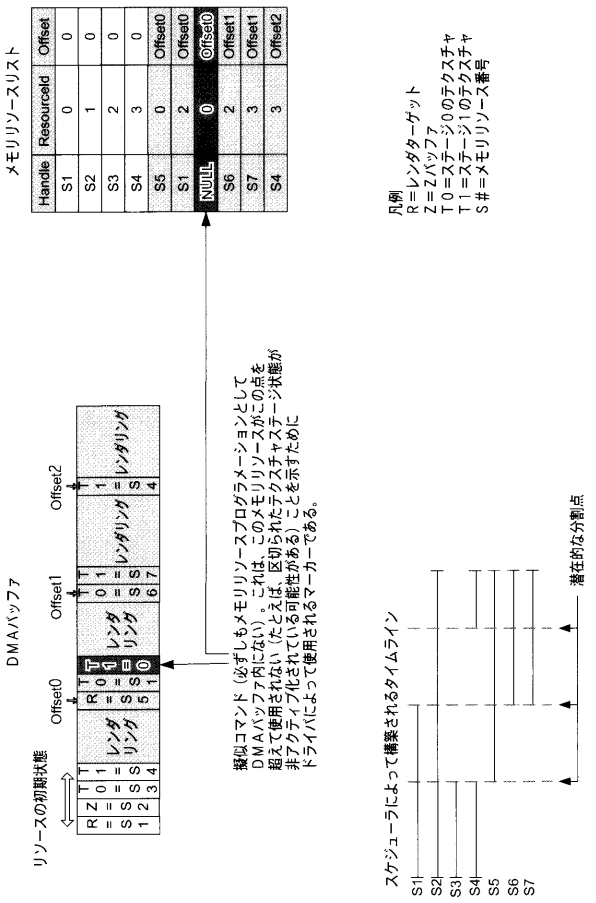
【図４（Ａ）】


例示的なアルゴリズム
プロセスA：Submit（irql passive, rendering thread context）
DMAバッファを割り振る
DMAバッファをシステムメモリアパーチャ内で割り振ることができる
ユーザモードバッファをDMAバッファにコピーし、確認する
必要なまたは望ましいメモリリソースのリストおよび、すべてのメモリリソースを同時にページインできない場合のDMAバッファの潜在的な分割点のリストを構築する
If レディリストが空でありDMAバッファが現在準備されていない：
If 現在のDMAバッファのすべてのメモリリソースが既にメモリ内に存在する、DMAバッファをパッチし、ハードウェアにサブミットするようにドライバに要求する
If メモリリソースの一部をページインする必要がある、DMAバッファをワーカースレッドにサブミットする
プロセスB：Quantum expires（irql device, any thread context）
If 現在のタスクがまだそのページングバッファを処理している
現在のタスクが実行を継続することを許可し、満了として現在のカンタムをセットする
If 次のタスクが実行の準備ができている
次のタスクを開始する
カンタムに「実行中」（期間内）をセットする
If タスクがメモリリソースのインポートを必要とする、ワーカースレッドにサブミットする
If すべてのメモリリソースが存在する、次のDMAバッファをパッチするようにドライバに要求する
If 次のタスクが実行の準備ができていない
現在のタスクが実行を継続することを許可し、現在のカンタムに「満了」をセットする

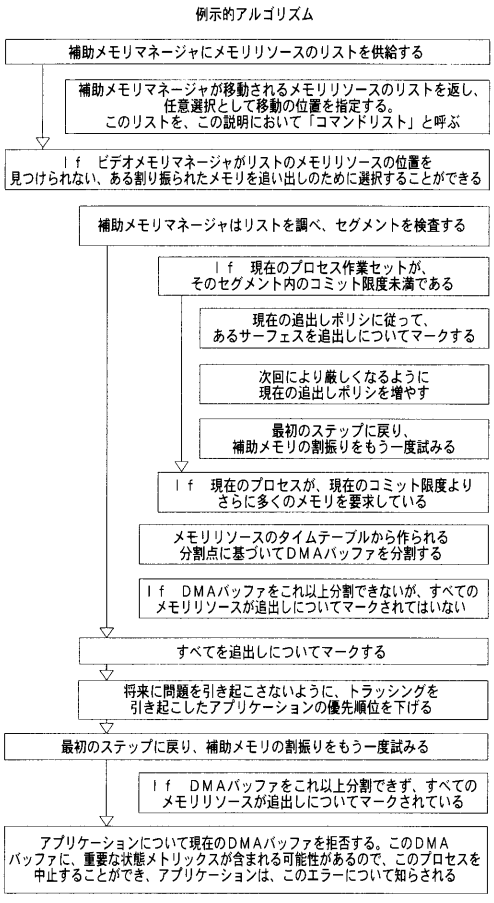
【図４（Ｂ）】


例示的なアルゴリズム
プロセスC：Task finishes（irql device, any thread context）
If 次のタスクが実行の準備ができている
次のタスクを開始する
カンタムを「実行中」にリセットする
次に実行するタスクを選択する
If タスクがメモリリソースのインポートを必要とする、ワーカースレッドにサブミットする
If すべてのメモリリソースが存在する、次のDMAバッファをパッチするようにドライバに要求する
If 次のタスクが実行の準備ができていない
ワーカースレッドが次のタスクセットアップを完了する間またはアプリケーションが新しいバッファをサブミットするまで、コプロセッサを停止させる
ワーカースレッド完了処理中に、ワーカースレッドの優先順位を上げ、したがって、ワーカースレッドができる限り早く作業を完了するようにする

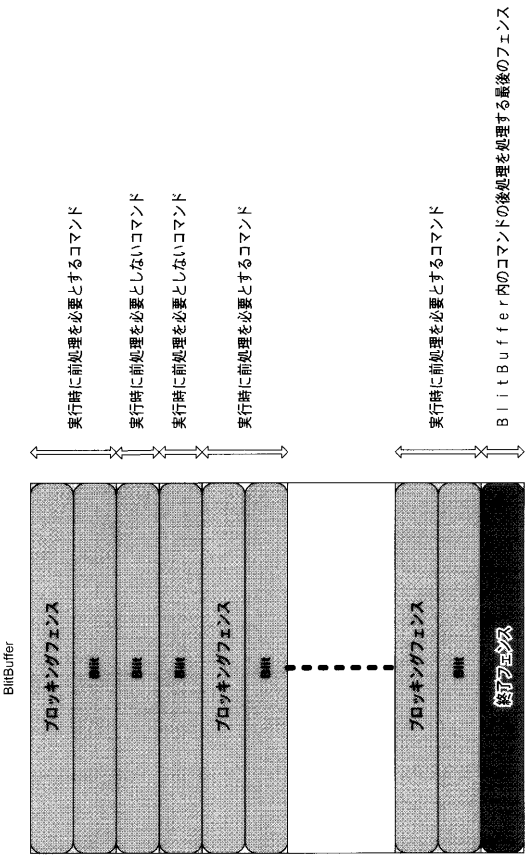
【図５】


メモリリソースリスト
Handle
Resourceld
Offset
S1
S2
S3
S4
S5
S1
S6
S7
S4
NULL
Offset0
Offset1
Offset2
凡例
R＝レンダーゲット
Z＝Zバッファ
T 0＝ステージ0のテクスチャ
T 1＝ステージ1のテクスチャ
S #＝メモリリソース番号
DMAバッファ
リソースの初期状態
レンダリング
擬似コマンド（必ずしもメモリリソースプログラメーションとしてDMAバッファ内にない）。これは、このメモリリソースがこの点を超えて使用されない（たとえば、区切れたテクスチャステージ状態が非アクティブ化されている可能性がある）ことを示すためにドライバによって使用されるマーカーである。
スケジューラによって構築されるタイムライン
潜在的な分割点
S11
S2
S3
S4
S5
S6
S7

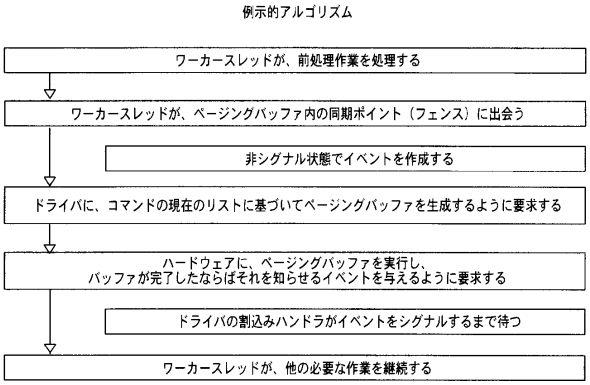
【図６】


例示的アルゴリズム
補助メモリマネージャにメモリリソースのリストを供給する
補助メモリマネージャが移動されるメモリリソースのリストを返し、任意選択として移動の位置を指定する。このリストを、この説明において「コマンドリスト」と呼ぶ
If ビデオメモリマネージャがリストのメモリリソースの位置を見つけられない、ある割り振られたメモリを追い出しのために選択することができる
補助メモリマネージャはリストを調べ、セグメントを検査する
If 現在のプロセス作業セットが、そのセグメント内のコミット限度未満である
現在の追出しポリシに従って、あるサーフェスを追出しについてマークする
次回により厳しくなるように現在の追出しポリシを増やす
最初のステップに戻り、補助メモリの割振りをもう一度試みる
If 現在のプロセスが、現在のコミット限度よりさらに多くのメモリを要求している
メモリリソースのタイムテーブルから作られる分割点に基づいてDMAバッファを分割する
If DMAバッファをこれ以上分割できないが、すべてのメモリリソースが追出しについてマークされてはいない
すべてを追出しについてマークする
将来に問題を引き起こさないように、トラッシングを引き起こしたアプリケーションの優先順位を下げる
最初のステップに戻り、補助メモリの割振りをもう一度試みる
If DMAバッファをこれ以上分割できず、すべてのメモリリソースが追出しについてマークされている
アプリケーションについて現在のDMAバッファを拒否する。このDMAバッファに、重要な状態メトリックスが含まれる可能性があるので、このプロセスを中止することができ、アプリケーションは、このエラーについて知らされる

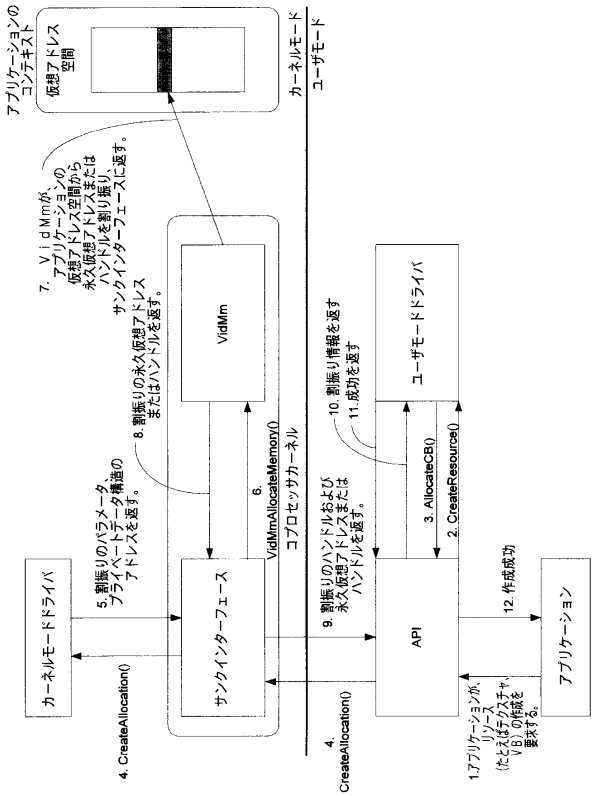
【図７】


BitBuffer
ブロッキングフェンス
待つフェンス
実行前に前処理を必要とするコマンド
実行前に前処理を必要としないコマンド
実行前に前処理を必要としないコマンド
実行前に前処理を必要とするコマンド
実行前に前処理を必要とするコマンド
ＢｌｉｔＢｕｆｆｅｒ内のコマンドの後処理を処理する最後のフェンス

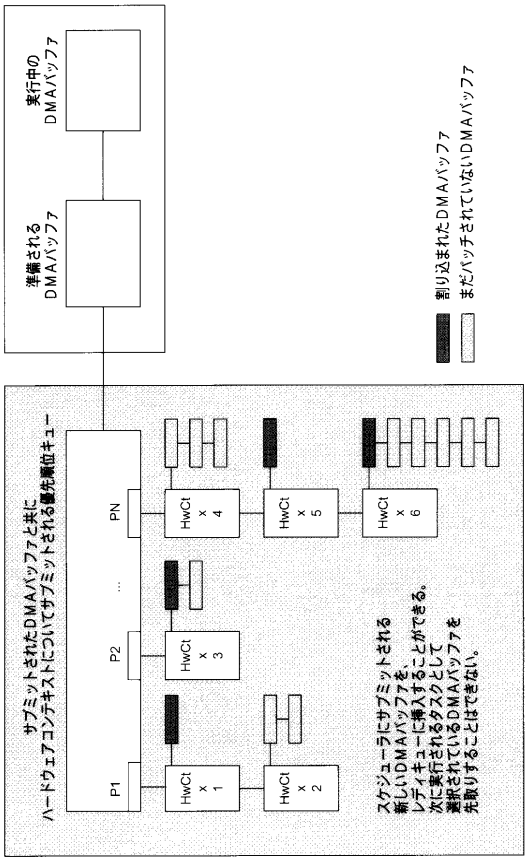
【図８】


例示的アルゴリズム
ワーカースレッドが、前処理作業を処理する
ワーカースレッドが、ページングバッファ内の同期ポイント（フェンス）に出会う
非シグナル状態でイベントを作成する
ドライバに、コマンドの現在のリストに基づいてページングバッファを生成するように要求する
ハードウェアに、ページングバッファを実行し、
バッファが完了したならばそれを知らせるイベントを与えるように要求する
ドライバの割込みハンドラがイベントをシグナルするまで待つ
ワーカースレッドが、他の必要な作業を継続する

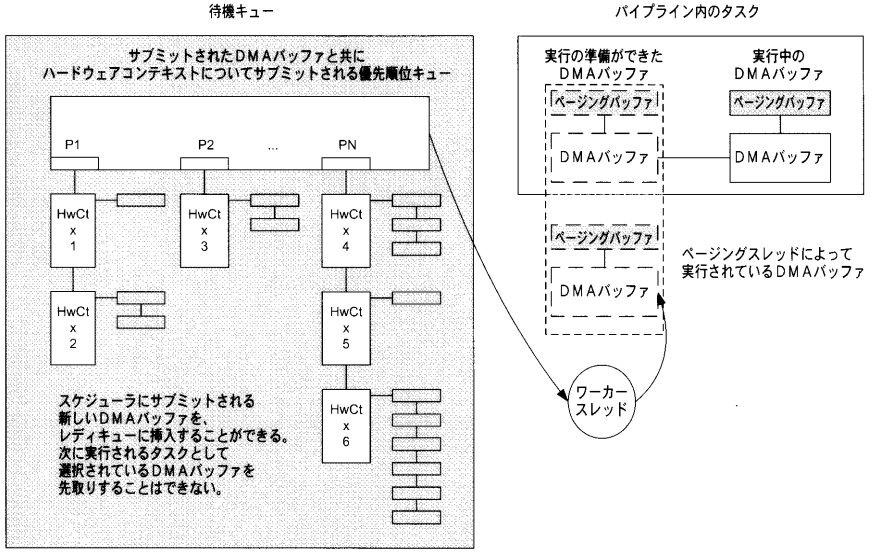
【図９】


アプリケーションのコンテキスト
仮想アドレス空間
カーネルモード
ユーザモード
コプロセッサカーネル
カーネルモードドライバ
サンクインターフェース
VidMm
ユーザモードドライバ
API
アプリケーション
4. CreateAllocation()
5. 割振りのパラメータ、プライベートデータ構造のアドレスを返す。
6. VidMmAllocateMemory()
7. VidMmが、アプリケーションの仮想アドレス空間から永久仮想アドレスまたはハンドルを割り振り、サンクインターフェースに返す。
8. 割振りの永久仮想アドレスまたはハンドルを返す。
9. 割振りのハンドルおよび永久仮想アドレスまたはハンドルを返す。
10. 割振り情報を返す
11. 成功を返す
2. CreateResource()
3. AllocateCB()
12. 作成成功
1. アプリケーションが、リソース（たとえばテクスチャ、VB）の作成を要求する。
4. CreateAllocation()

【図１０】


パイプライン内のタスク
準備されるDMAバッファ
実行中のDMAバッファ
レディキュー
サブミットされたDMAバッファと共にハードウェアコンテキストについてサブミットされる優先順位キュー
P1
P2
PN
HwCtx 1
HwCtx 2
HwCtx 3
HwCtx 4
HwCtx 5
HwCtx 6
スケジューラにサブミットされる新しいDMAバッファを、レディキューに挿入することができる。次に実行されるタスクとして選択されているDMAバッファを先取りすることはできない。
割り込まれたDMAバッファ
まだパッチされていないDMAバッファ

【図１１】



【図１２（Ａ）】

プロセスA：Submit（IRQL passive, rendering thread context）
　If DMAバッファが、準備されていないかまたは実行の準備ができていない
　If 現在のDMAバッファのすべてのメモリリソースが、既にメモリ内に存在する
　　If コプロセッサがアイドルである、DMAバッファをコプロセッサに与える
　　Else DMAバッファを実行の準備ができたスロットに挿入する
　　If メモリリソースのページインが必要、DMAバッファをページングスレッドにサブミットする
　　　Else DMAバッファを現在のコンテキストのリストの末尾に挿入する

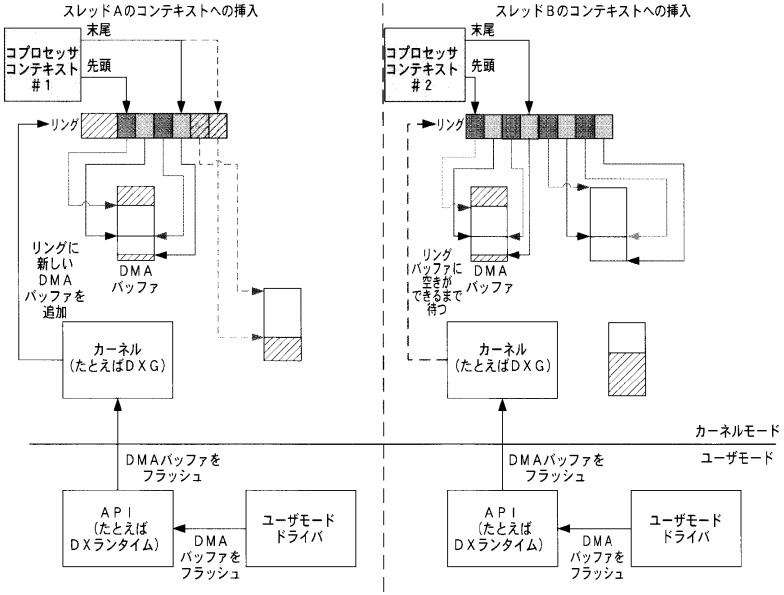
プロセスB：Quantum expires（IRQL device, any thread context）
　If 現在のタスクがまだページングバッファを処理している
　現在のタスクの動作継続を許可する
　現在のカンタムに満了をセットする
Else If 次のDMAバッファが実行の準備ができている
　現在のコンテキストの現在の優先順位をその基本優先順位にリセットする
　現在のコンテキストをその優先順位のキューの末尾に移動する
次のDMAバッファをコプロセッサにサブミットする
カンタムを実行中（満了していない）としてリセットする
　次に実行されるDMAバッファを選択する
　　If DMAバッファがページングを必要とする、ページングスレッドにサブミットする
　　Else すべてのメモリリソースが既に存在する；DMAバッファをレディスロットに挿入する
　Else 次のタスクが実行の準備ができていない
　　現在のタスクの動作継続を許可する
　　現在のカンタムに満了をセットする

プロセスC：Task finishes（IRQL device, any thread context）
　If 次のDMAバッファが実行の準備ができている
　次のDMAバッファをコプロセッサにサブミットする
　カンタムを実行中（満了していない）としてリセットする
　次に実行されるDMAバッファを選択する
　　If DMAバッファがページングを必要とする、ページングスレッドにサブミットする
　　Else すべてのメモリリソースが既に存在する；DMAバッファをレディスロットに挿入する
　　Else 次のタスクが実行の準備ができていない
　　If ページングスレッドが現在次のDMAバッファを処理している、ワーカースレッドの優先順位を一時的に上げ、
　　できる限り早く作業を終了するようにする

【図１２（Ｂ）】

プロセスD：Paging thread（IRQL passive, system thread）
　現在の追出しポリシに最初のポリシをセットする
　メモリマネージャに、リソースリストをページインするように要求する
　If すべてのリソースが成功裡にページインされた
　　ページングバッファおよびDMAバッファを実行の準備ができたスロットに移動する
　　If 現在のDMAバッファのカンタムが満了している
　　次のDMAバッファをコプロセッサにサブミットする
　　カンタムを実行中（満了していない）としてリセットする
　　次に実行されるDMAバッファを選択する
　　　If DMAバッファがページングを必要とする、ページングスレッドにサブミットする
　　　Else すべてのメモリリソースが既に存在する；DMAバッファをレディスロットに挿入する
　Else If ページングバッファが満杯なのでメモリマネージャが失敗した
　　現在のDMAバッファのカンタムが終わるか終了するまで待つ
　　ページングバッファをコプロセッサにサブミットする
　　ページングバッファが終わるのを待つ
　　戻って、メモリマネージャに、リソースリストの残りをページインするように要求する
　　Else If 十分な使用可能なリソースがないのでメモリマネージャが失敗した
　If 最後の追出しポリシを過ぎた
　リソース移動を元に戻すか、またはページングバッファを実行する
　　DMAバッファを拒否する
　　終了
　　Else If 現在の追出しポリシが、アプリケーション干渉を超える
　　If DMAバッファがまだ分割されていない
　　現在ページインされているリソースに最も近い点でバッファを分割する
　　If これ以上リソースが必要ではない
　　ページングバッファおよび分割されたDMAバッファを実行の準備ができたスロットに移動する
　　残りのDMAバッファをコンテキストのレディキューの先頭に移動する
　If 現在のDMAバッファのカンタムが満了した
　次のDMAバッファをコプロセッサにサブミットする
　カンタムを実行中（満了していない）としてリセットする
　　次に実行されるDMAバッファを選択する
　　If DMAバッファがページングを必要とする、ページングスレッドにサブミットする
　　　Else すべてのメモリリソースが既に存在する；DMAバッファをレディスロットに挿入する
　　VidMmに、現在のポリシを使用して追出しの候補をマークするように要求する
　　If VidMmが、現在のポリシでメモリをマークできないというエラーを返す
　　追出しポリシを増やす
　　追出しポリシ検査の先頭に戻る
　　　Else メモリがマークされている
　　リソースのページインの試みに戻る

【図１６】



【図１７（Ａ）】

プロセスＡ：Ｓｕｂｍｉｔ（IRQL passive, rendering thread context, coprocessor Context mutex held）
　ＶｉＤＭＭ＿ｌｏｃｋを獲得する
　与えられたリソースのリストを処理し、このプロセスの割振りに関する使用状況情報を更新する
　ＶｉＤＭＭ＿ｌｏｃｋを解放する
　スケジューラロックをとる
　ドライバを呼び出して、現在のＤＭＡバッファをリングに挿入する
　Ｉｆ　ドライバが成功した
　　Ｉｆ　コンテキストがアイドルであった
　　　レディリストのコンテキストを、現在の優先順位に関するキューの末尾に挿入する
　　Ｉｆ　保留中のコンテキスト転送がなく、現在のコンテキストが現在のコンテキスト未満の優先順位である
　　　ドライバを呼び出して、このコンテキストへのコンテキスト切り替えを行う
　　　コンテキスト切り替えが保留中であることをシグナリングする
　スケジューラロックを解放する
　Ｉｆ　ドライバが失敗した、リングが満杯であった
　　空きが使用可能になるときにシグナリングされるイベントを待つ
　　待った後に、スケジューラロックの獲得に戻る
　Ｉｆ　別のサブミットに十分な空きがＤＭＡバッファに残っている
　　現在のＤＭＡバッファでユーザモードに戻る
　コンテキストのプールから新しいＤＭＡバッファを獲得する
　Ｉｆ　この時にＤＭＡプールに別のバッファがない
　　ＤＭＡバッファがプールに挿入されたときにシグナリングされるイベントを待つ
　　待機が終わったときに、新しいＤＭＡバッファの獲得に戻り、それを試みる
　新しいＤＭＡバッファをユーザモードに返す

プロセスＢ：Ｃｏｎｔｅｘ ｓｗｉｔｈ ｄｏｎｅ（IRQL ｄｅｖｉｃｅ, ａｎｙ ｔｈｒｅａｄ ｃｏｎｔｅｘｔ）
　スケジューラロックをとる
　　Ｉｆ　より高い優先順位のコンテキストが実行の準備ができている
　　　ドライバを呼び出して、より高い優先順位のコンテキストへのコンテキスト切り替えを行う
　　Ｅｌｓｅ
　　　コンテキスト切り替えが現在保留中でないことをシグナリングする
　スケジューラロックを解放する

プロセスＣ：Ｑｕａｎｔｕｍ ｅｘｐｉｒｅｓ（IRQL ｄｅｖｉｃｅ, ａｎｙ ｔｈｒｅａｄ ｃｏｎｔｅｘｔ）
　スケジューラロックをとる
　コンテキストの現在の優先順位を、そのベース優先順位にリセットする
　コンテキストを、現在の優先順位のキューの末尾に挿入する
　Ｉｆ　現在コンテキスト切り替えが保留中でない
　　最も高い優先順位のコンテキストへのコンテキスト切り替えを行うようにドライバに要求する
　スケジューラロックを解放する

【図１７（Ｂ）】

プロセスＤ：Ｔａｓｋ ｆｉｎｉｓｈｅｓ（IRQL ｄｅｖｉｃｅ, ａｎｙ ｔｈｒｅａｄ ｃｏｎｔｅｘｔ）
　スケジューラロックをとる
　コンテキストが実際に空であるかどうかをドライバにたずねる
　Ｉｆ　コンテキストが実際に空である
　　コンテキストの現在の優先順位をそのベース優先順位にリセットする
　　コンテキストをアイドルリストに挿入する
　Ｉｆ　コンテキストが実際に空ではない
　　Ｉｆ　コンテキスト切り替えが現在保留中でない
　　　最も高い優先順位のコンテキストへのコンテキスト切り替えを行うようにドライバに要求する
　スケジューラロックを解放する

プロセスＥ：Ｐａｇｅ Ｆａｕｌｔ（IRQL ｄｅｖｉｃｅ, ａｎｙ ｔｈｒｅａｄ ｃｏｎｔｅｘｔ）
　スケジューラロックをとる
　コンテキストをレディリストから除去する
　コンテキストをアトミックオペレーションとしてページリストに挿入する
　Ｉｆ　インページスレッドが現在スリープしている
　　ワーカースレッドをウェイクアップし、シグナリングするためにＤＰＣをキューに入れる
　Ｉｆ　コンテキスト切り替えが現在保留中でない
　　最も高い優先順位のコンテキストへのコンテキスト切り替えを行うようにドライバに要求する
　スケジューラロックを解放する

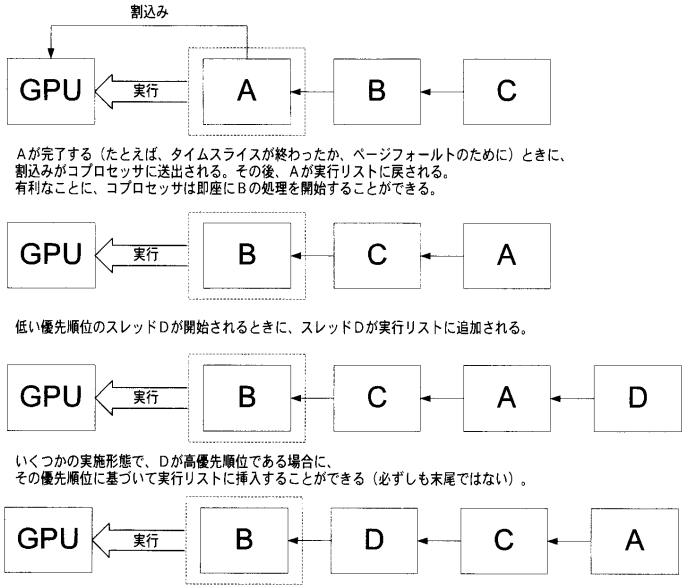
プロセスＦ：Ｆａｕｌｔ ｒｅｓｏｌｖｅｄ（IRQL ｄｅｖｉｃｅ, ａｎｙ ｔｈｒｅａｄ ｃｏｎｔｅｘｔ）
　スケジューラロックをとる
　コンテキストをインページリストから除去する
　コンテキストを、現在の優先順位のレディリストに挿入する
　Ｉｆ　コンテキスト切り替えが現在保留中でなく、現在のコンテキストが現在実行中のコンテキストより高い優先順位である
　　最も高い優先順位のコンテキストへのコンテキスト切り替えを行うようにドライバに要求する
　スケジューラロックを解放する

【図１７（Ｃ）】

プロセスＧ：Ｉｎ ｐａｇｅ ｗｏｒｋｅｒ ｔｈｒｅａｄ
　インページキューのコンテキストのリストを全部調べる。最高の優先順位のコンテキストを選択する
　コンテキストについて進行するのに必要なリソースのリストをドライバに要求する
　ＶｉＤＭＭ＿ｌｏｃｋをとる
　進行に必要な割振りのそれぞれの位置を見つける
　追い出される割振りの仮想アドレスまたはハンドルを無効化する
　必要な割振りを選択された点に持ち込むのに必要なメモリ転送コマンドによってＤＭＡバッファを満たすようにドライバに要求する
　ＶｉＤＭＭ＿ｌｏｃｋを解放する
　通常のコプロセッサコンテキストとしてＶｉｄＭｍコンテキストをサブミットする
　Ｉｆ　コンテキストのリストが空である、アイテムが追加されるまでスリープする
　ループの先頭に戻る

プロセスＨ：Ｐｅｒｉｏｄｉｃ ｔｉｍｅｒ（ｐａｓｓｉｖｅ ｌｅｖｅｌ, ｓｙｓｔｅｍ ｔｈｒｅａｄ ｃｏｎｔｅｘｔ）
　スケジューラロックをとる
　各コンテキストの現在の優先順位を増分する
　スケジューラロックを解放する

【図１８】

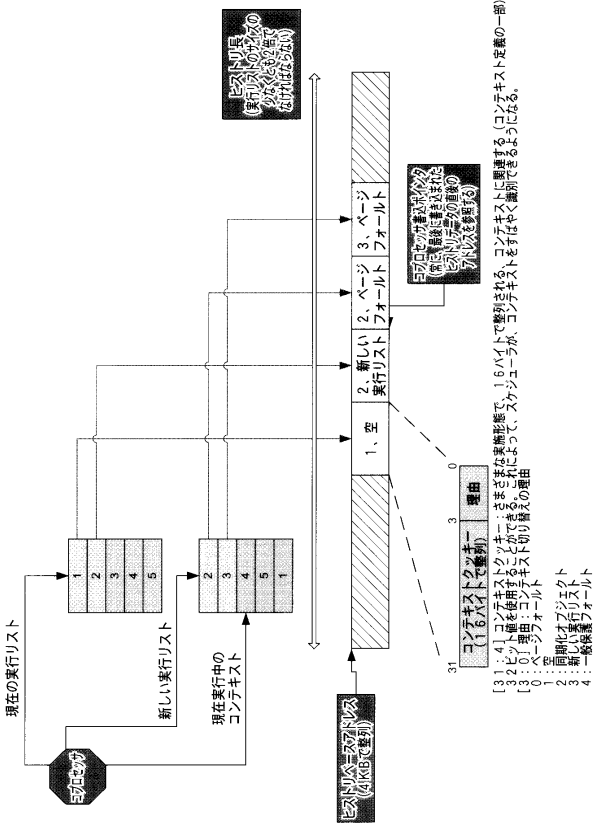


Ａが完了する（たとえば、タイムスライスが終わったか、ページフォールトのために）ときに、
割込みがコプロセッサに送出される。その後、Ａが実行リストに戻される。
有利なことに、コプロセッサは即座にＢの処理を開始することができる。

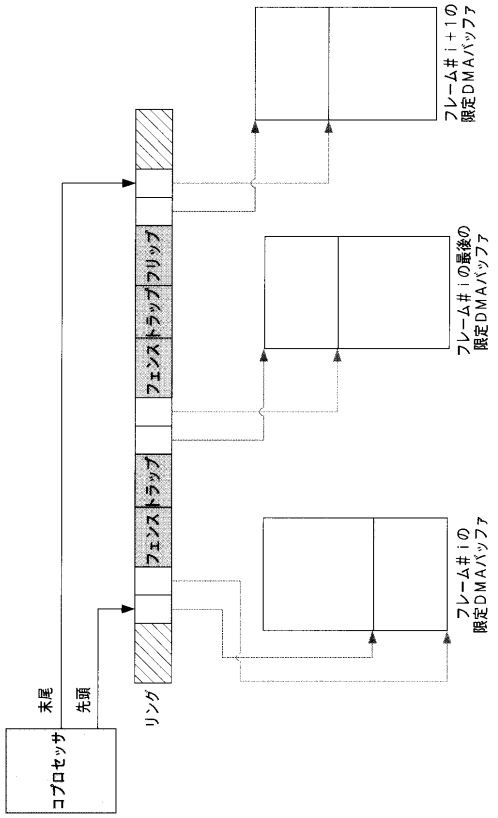
低い優先順位のスレッドＤが開始されるときに、スレッドＤが実行リストに追加される。

いくつかの実施形態で、Ｄが高優先順位である場合に、
その優先順位に基づいて実行リストに挿入することができる（必ずしも末尾ではない）。

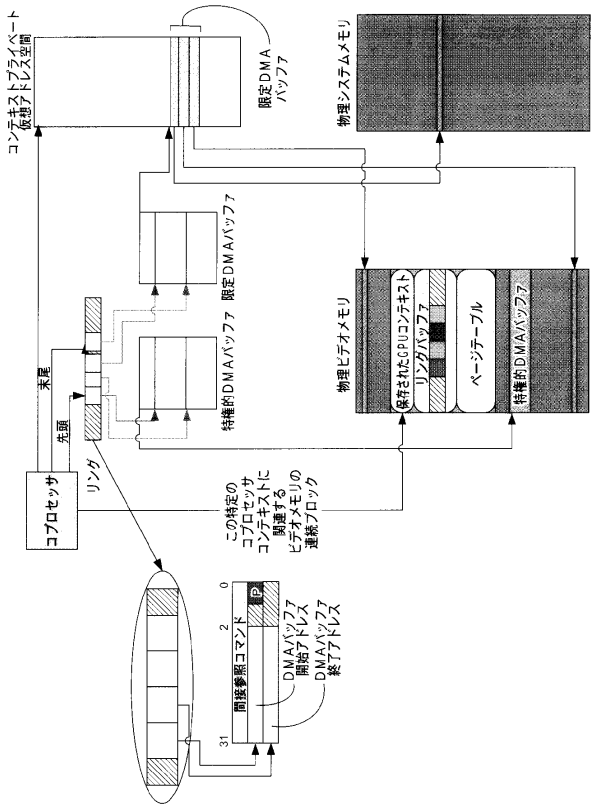
【図１９】



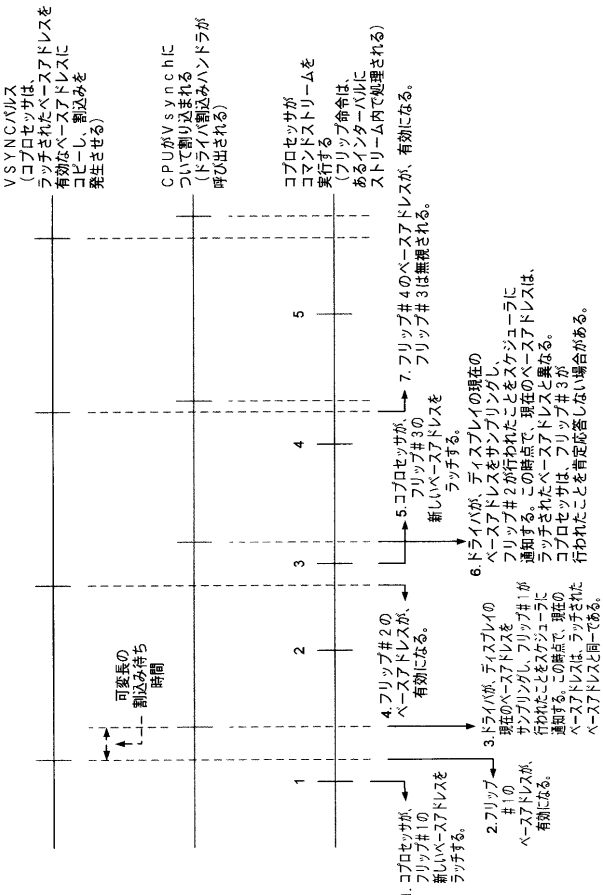
【図２０】



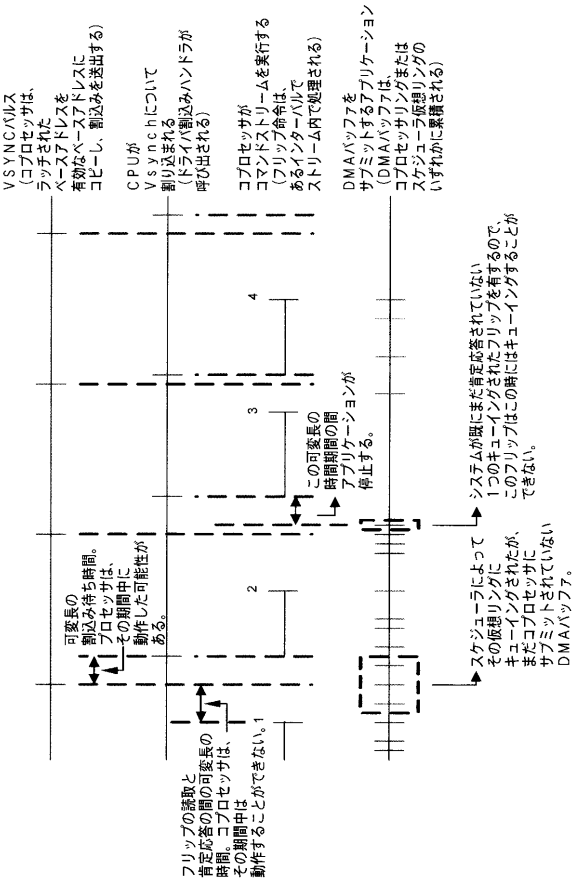
【図２１】



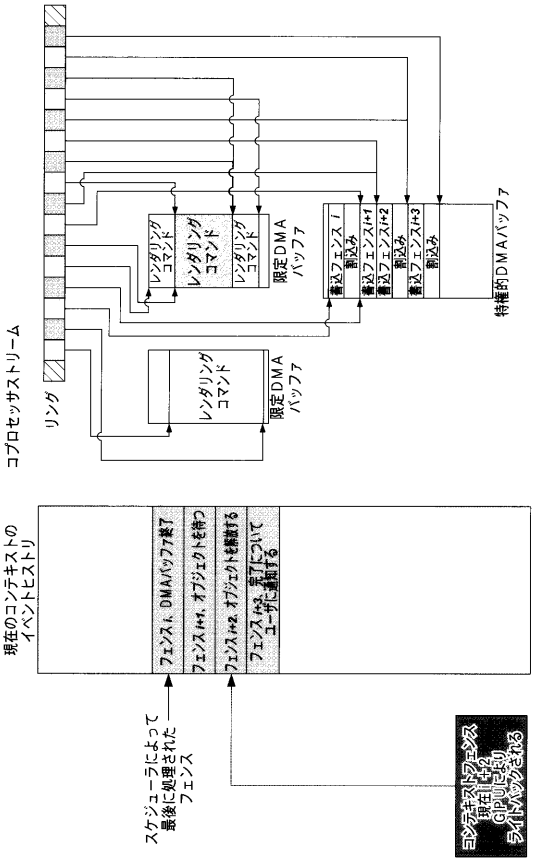
【図２２】



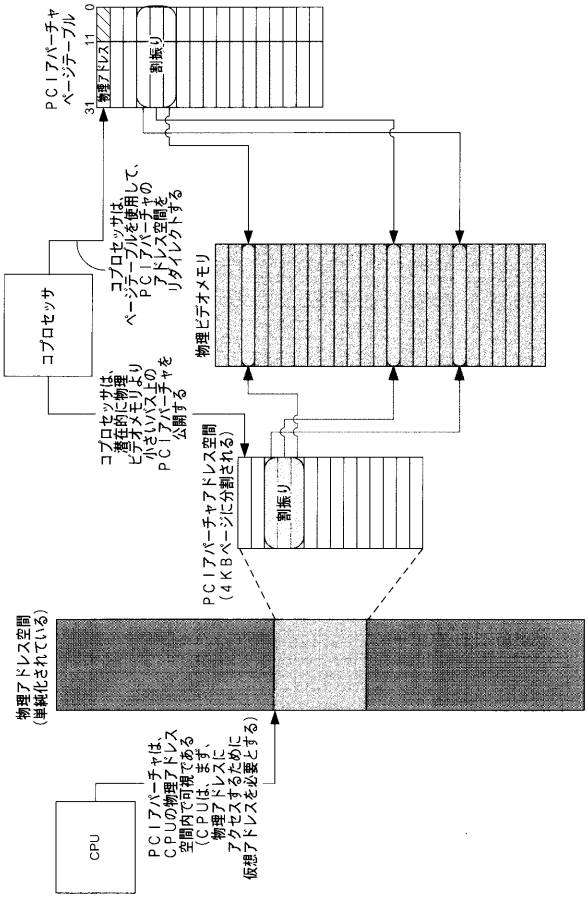
【図２３】


VSYNCパルス
（コプロセッサは、
ラッチされた
ベースアドレスを
有効なベースアドレスに
コピーし、割込みを送出する）
CPUが
VsynchについてI
割り込まれる
（ドライバ割込みハンドラが
呼び出される）
コプロセッサが
コマンドストリームを実行する
（フリップ命令は、
あるインターバルで
ストリーム内で処理される）
DMAバッファを
サブミットするアプリケーション
（DMAバッファは、
コプロセッサリングまたは
スケジューラ仮想リングの
いずれかに累積される）
可変長の
割込み待ち時間。
プロセッサは、
その期間中に
動作した可能性が
ある。
この可変長の
時間期間の間
アプリケーションが
停止する。
スケジューラによって
その仮想リングに
キューイングされたが、
まだコプロセッサに
サブミットされていない
DMAバッファ。
システムが既にまだ肯定応答されていない
1つのキューイングされたフリップを有するので、
このフリップはこの時にはキューイングすることが
できない。
フリップの読取と
肯定応答の間の可変長の
時間、コプロセッサは、
その期間中は
動作することができない。1
2
3
4

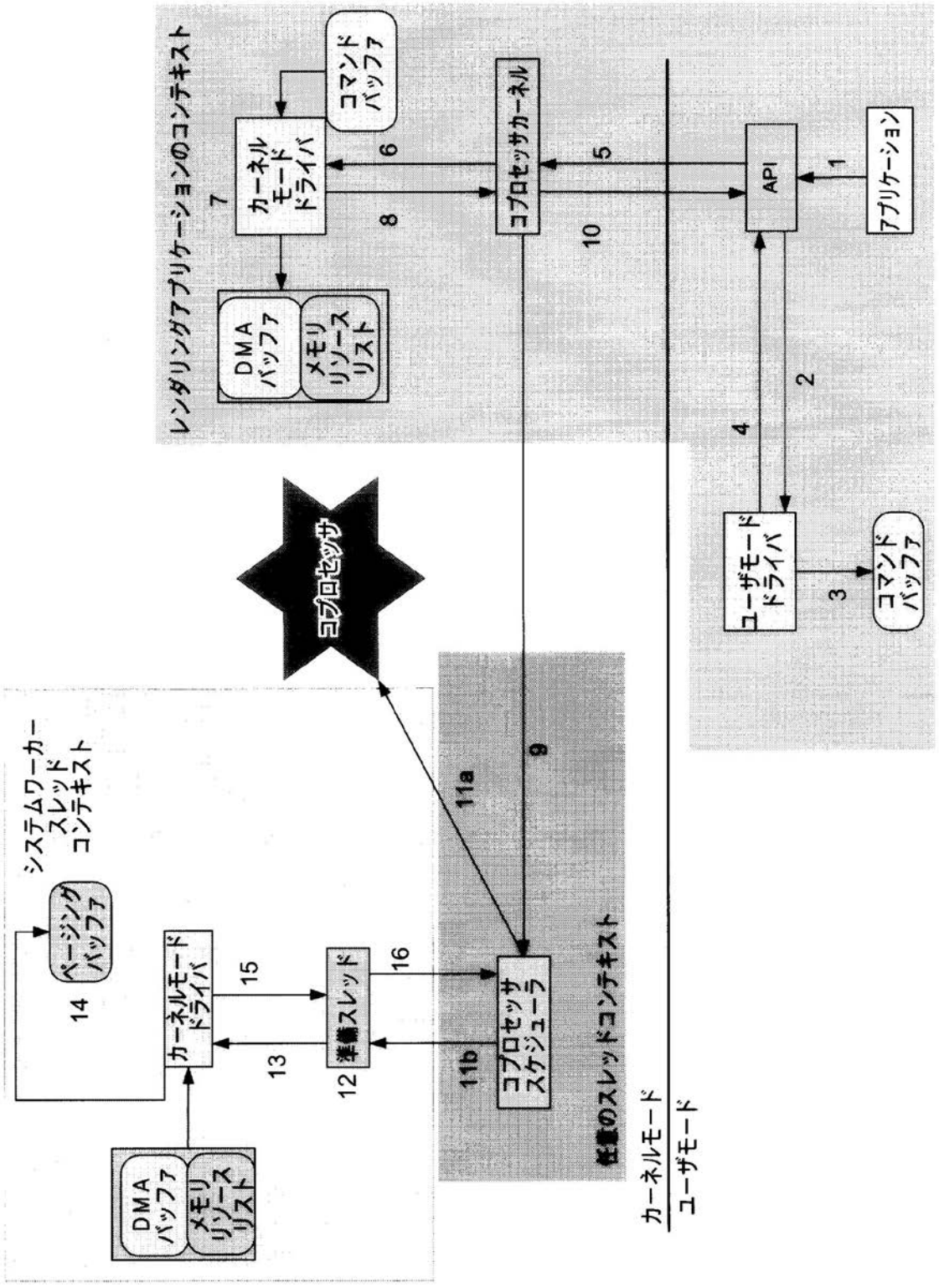
【図２５】


コプロセッサストリーム
リング
レンダリング
コマンド
限定DMA
バッファ
レンダリング
コマンド
限定DMA
バッファ
レンダリング
コマンド
限定DMA
バッファ
割込フェンスi
割込フェンスi+1
割込フェンスi+2
割込フェンスi+3
割込み
割込み
割込み
特権的DMAバッファ
現在のコンテキストの
イベントヒストリ
フェンスi：DMAバッファ終了
フェンスi+1：オブジェクトを待つ
フェンスi+2：オブジェクトを解放する
フェンスi+3：終了について
ユーザに通知する
スケジューラによって
最後に処理された
フェンス
コンテキストIDx
現在はCPUにより
ライトバックされる

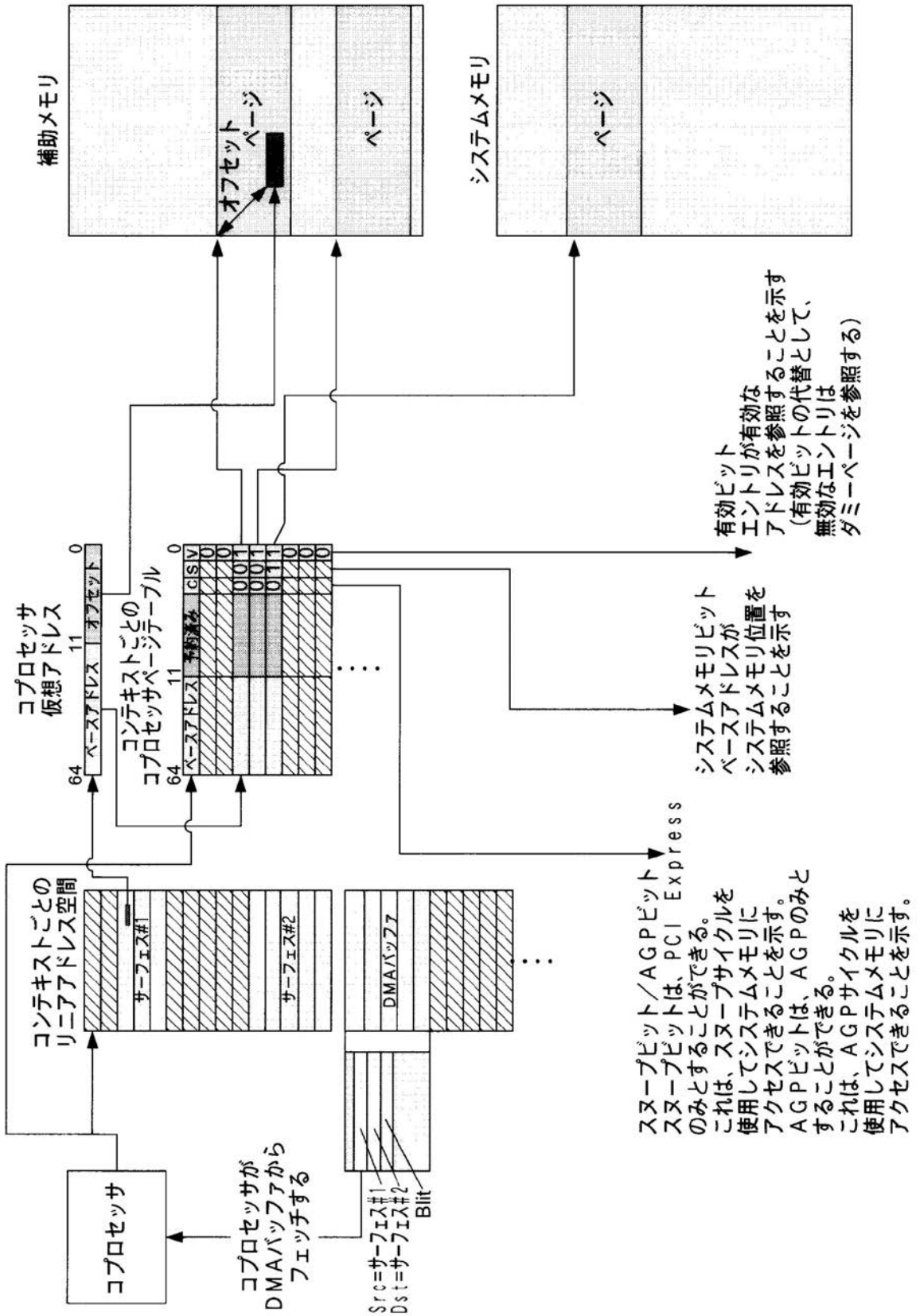
【図２６】


PCIアパーチャ
ページテーブル
0
11
31
物理アドレス
割振り
コプロセッサは、
ページテーブルを使用して、
PCIアパーチャの
アドレス空間を
リダイレクトする
コプロセッサ
コプロセッサは、
潜在的に物理
ビデオメモリより
小さいバスよの
PCIアパーチャを
公開する
物理ビデオメモリ
PCIアパーチャアドレス空間
（4KBページに分割される）
割振り
物理アドレス空間
（単純化されている）
PCIアパーチャは、
CPUの物理アドレス
空間内で可視である
（CPUは、まず、
物理アドレスに
アクセスするために
仮想アドレスを必要とする）
CPU

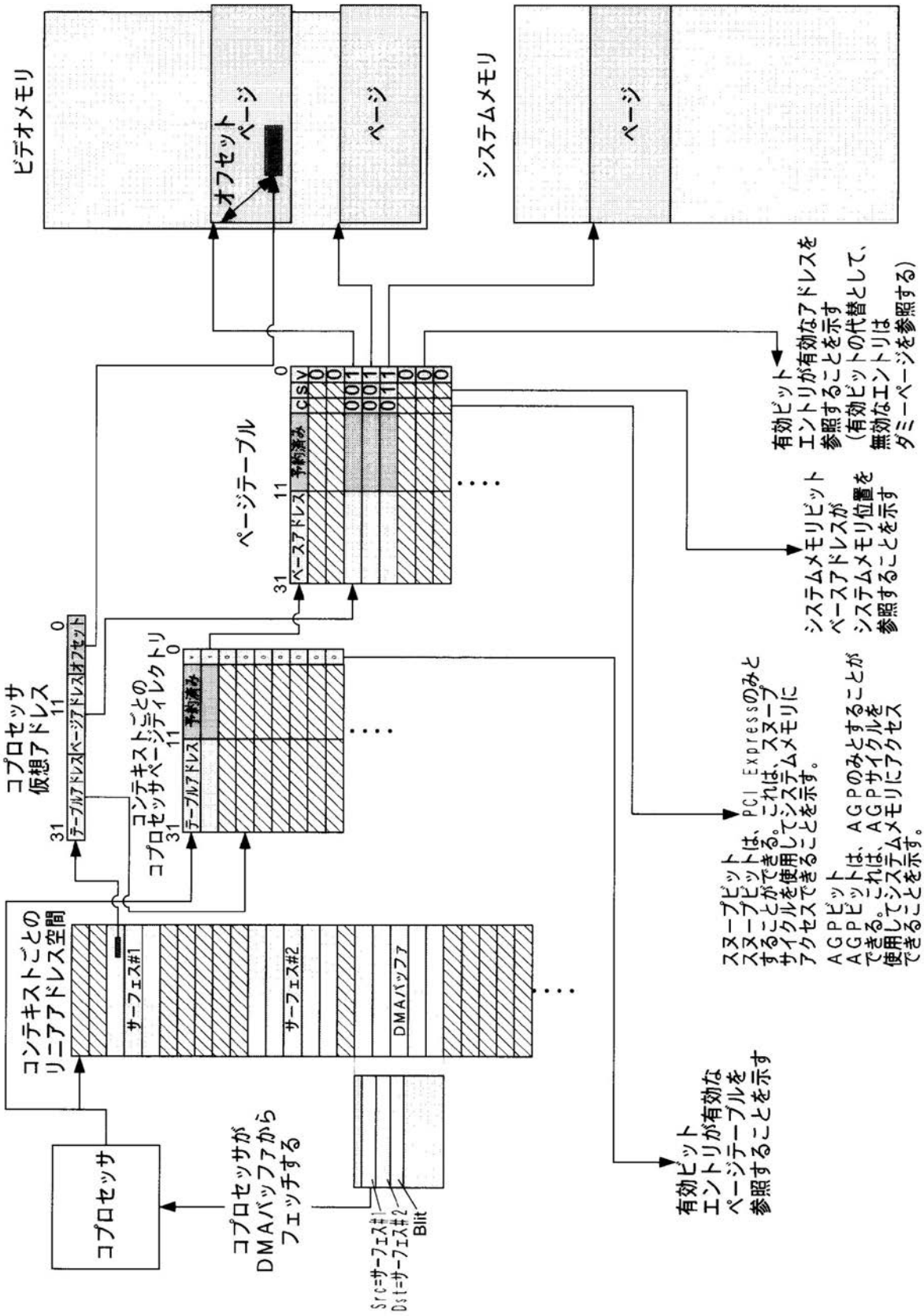
【図３】


レンダリングアプリケーションのコンテキスト
アプリケーション
API
ユーザモードドライバ
コマンドバッファ
カーネルモードドライバ
コマンドバッファ
DMAバッファ
メモリリソースリスト
コプロセッサカーネル
コプロセッサ
システムワーカースレッドコンテキスト
ページングバッファ
カーネルモードドライバ
DMAバッファ
メモリリソースリスト
準備スレッド
コプロセッサスケジューラ
低優先度のスレッドコンテキスト
カーネルモード
ユーザモード
1
2
3
4
5
6
7
8
9
10
11a
11b
12
13
14
15
16

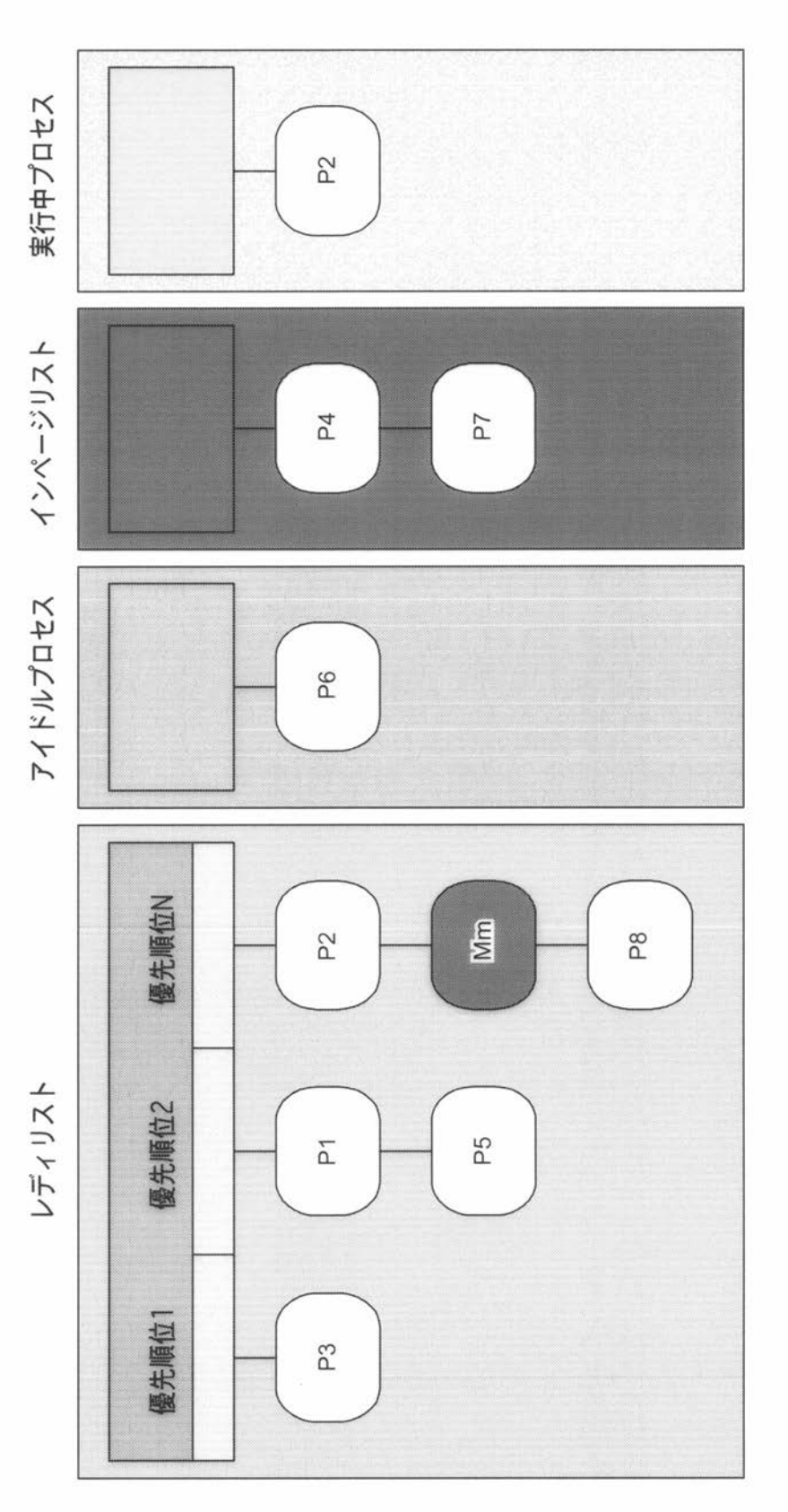
【図１３】


補助メモリ
オフセット
ページ
システムメモリ
ページ
コンテキストごとの
リニアアドレス空間
コプロセッサ
仮想アドレス
ベースアドレス
オフセット
コンテキストごと
コプロセッサページテーブル
ベースアドレス
予約済み
GS
V
サーフェス#1
サーフェス#2
DMAバッファ
有効ビット
エントリが有効な
アドレスを参照することを示す
（有効ビットの代替として、
無効なエントリは
ダミーページを参照する）
システムメモリビット
ベースアドレスが
システムメモリ位置を
参照することを示す
スヌープビット／AGPビット
スヌープビットは、PCI Express
のみとすることができる。
これは、スヌープサイクルを
使用してシステムメモリに
アクセスできることを示す。
AGPビットは、AGPのみと
することができる。
これは、AGPサイクルを
使用してシステムメモリに
アクセスできることを示す。
コプロセッサ
コプロセッサが
DMAバッファから
フェッチする
Src=サーフェス#1
Dst=サーフェス#2
Blit

【図１４】


ビデオメモリ
オフセット
ページ
ページ
システムメモリ
ページ
ページテーブル
ベースアドレス
予約済み
GIS
V
有効ビット
エントリが有効なアドレスを
参照することを示す
（有効ビットの代替として、
無効なエントリは
ダミーページを参照する）
システムメモリビット
ベースアドレスが
システムメモリ位置を
参照することを示す
コンテキストごとの
コプロセッサページディレクトリ
テーブルアドレス
予約済み
コプロセッサ
仮想アドレス
テーブルアドレス
ページアドレスオフセット
コンテキストごとの
リニアアドレス空間
サーフェス#1
サーフェス#2
DMAバッファ
スヌープビット
スヌープビットは、PCI Expressのみと
スヌープができる。これは、スヌープ
サイクルを使用してシステムメモリに
アクセスできることを示す。
AGPビット
AGPビットは、AGPのみとすることが
できる。これは、AGPサイクルを
使用してシステムメモリにアクセス
できることを示す。
有効ビット
エントリが有効な
ページテーブルを
参照することを示す
コプロセッサ
コプロセッサが
DMAバッファから
フェッチする
Src=サーフェス#1
Dst=サーフェス#2
Blit

【図１５】


実行中プロセス
P2
インページリスト
P4
P7
アイドルプロセス
P6
レディリスト
優先順位1
優先順位2
優先順位N
P3
P1
P5
P2
Mm
P8

【図２４】

コプロセッサスレッドA
擬似コード：
//共有サーフェスへの排他的アクセスを得るまで待つ
//
DxAcquireMutex(gSharedMutex);
//共有サーフェスをレンダターゲットとしてセットする
//
DxSetRenderTarget(gSharedSurface);
//必要なものを共有サーフェスにレンダリングする
//
DxDrawSomething();
//レンダリングが終了したのでミューテクスを解放する
//
DxReleaseMutes(gSharedMutex);
コプロセッサストリーム：
リング
書込フェンス
特権的ＤＭＡバッファ
レンダリングコマンド
限定ＤＭＡバッファ
書込フェンス割込み
特権的ＤＭＡバッファ
コプロセッサスレッドB
擬似コード：
//共有サーフェスへの排他的アクセスを得るまで待つ
//
DxAcquireMutex(gSharedMutex);
//共有サーフェスをテクスチャとしてセットする
//
DxSetRenderTexture(gSharedSurface);
//必要なものを共有サーフェスにレンダリングする
//
DxDrawSomething();
//レンダリングが終了したのでミューテクスを解放する
//
DxReleaseMutes(gSharedMutex);
コプロセッサストリーム：
リング
書込フェンス
特権的ＤＭＡバッファ
レンダリングコマンド
限定ＤＭＡバッファ
書込フェンス割込み
特権的ＤＭＡバッファ

フロントページの続き

(31)優先権主張番号 60/474,513
(32)優先日 平成15年5月29日(2003.5.29)
(33)優先権主張国 米国(US)
(31)優先権主張番号 10/763,778
(32)優先日 平成16年1月22日(2004.1.22)
(33)優先権主張国 米国(US)

前置審査

(72)発明者 スティーブ プロノボスト
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド １５６ アベニュー ノースイースト ４８５０ ユニット １２３

審査官 三坂 敏夫

(56)参考文献 特開平０２－０１２５２３（ＪＰ，Ａ）
特開平０３－２０２９４１（ＪＰ，Ａ）
米国特許第０５１１３１８０（ＵＳ，Ａ）
特開平０４－３１１２３３（ＪＰ，Ａ）
特開平０７－２３４８２１（ＪＰ，Ａ）
特開２００２－１８３７５０（ＪＰ，Ａ）
特開２００１－９２６５７（ＪＰ，Ａ）
特開平１０－２９４８３４（ＪＰ，Ａ）
特開平５－２７４２４９（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ ９／３８
Ｇ０６Ｆ ９／５０