(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7465887号
(P7465887)

(45)発行日　令和6年4月11日(2024.4.11)

(24)登録日　令和6年4月3日(2024.4.3)

(51)国際特許分類

		FI		
G06F	*9/312(2018.01)*	G06F	9/312	L
G06F	*9/34 (2018.01)*	G06F	9/34	330
G06F	*9/345(2018.01)*	G06F	9/345	A

請求項の数　23　（全33頁）

(21)出願番号　特願2021-550246(P2021-550246)
(86)(22)出願日　令和2年3月23日(2020.3.23)
(65)公表番号　特表2022-543331(P2022-543331A)
(43)公表日　令和4年10月12日(2022.10.12)
(86)国際出願番号　PCT/GB2020/050774
(87)国際公開番号　WO2021/023954
(87)国際公開日　令和3年2月11日(2021.2.11)
審査請求日　令和5年3月16日(2023.3.16)
(31)優先権主張番号　16/531,208
(32)優先日　令和1年8月5日(2019.8.5)
(33)優先権主張国・地域又は機関
米国(US)

(73)特許権者　500395107
アーム・リミテッド
グレート・ブリテン及び北部アイルランド連合王国、イングランド、 シー・ビー１、９エヌ・ジェイ、ケンブリッジ、フルボーン・ロード １１０
(74)代理人　110000855
弁理士法人浅村特許事務所
(72)発明者　ステファンズ、ニゲル ジョン
イギリス国 ケンブリッジシャー ＣＢ１ ９ＮＪ ケンブリッジ，フルボーン ロード １１０， アームリミテッド宛
(72)発明者　マンセル、デイヴィッド ヘンナ
イギリス国 ケンブリッジシャー ＣＢ１ ９ＮＪ ケンブリッジ，フルボーン ロー
最終頁に続く

(54)【発明の名称】　データ構造処理

(57)【特許請求の範囲】

【請求項１】

装置であって、

命令を復号するための命令デコーダと、

前記命令デコーダによって復号された命令に応答してデータ処理を実行する処理回路とを備え、

少なくとも１つの入力データ構造識別子及び出力データ構造識別子を指定するデータ構造処理命令に応答して、前記命令デコーダは、前記処理回路を制御して、前記出力データ構造識別子によって識別された出力データ構造を生成するために、前記少なくとも１つの入力データ構造識別子によって識別された少なくとも１つの入力データ構造上で処理動作を実行させるように構成されており、

前記少なくとも１つの入力データ構造及び前記出力データ構造は各々、複数のメモリアドレスに対応するデータの配列を含み、

前記装置は、１つ以上のデータ構造メタデータレジスタの複数のセットを含み、１つ以上のデータ構造メタデータレジスタの各セットは、対応するデータ構造識別子に関連付けられており、前記対応するデータ構造識別子によって識別された前記データ構造のための前記複数のメモリアドレスを識別するためのアドレス表示メタデータを保持するように指定される、装置。

【請求項２】

前記処理回路は、データ構造識別子の少なくともサブセットについて、データ構造識

別子の前記サブセットのうちの１つに対応するデータ構造がクリーン又はダーティであるかを、ソフトウェアが識別することを可能にする命令セットアーキテクチャに従って動作するように構成されている、請求項１に記載の装置。
【請求項３】
前記出力データ構造識別子は、前記少なくとも１つの入力データ構造識別子とは別個の識別子空間内に定義される、請求項１又は２のいずれか一項に記載の装置。
【請求項４】
少なくとも１つのダーティインジケータを保持するための少なくとも１つのアーキテクチャレジスタを備え、各ダーティインジケータは、所与のデータ構造識別子に対応するデータ構造がクリーンであるか又はダーティであるかを示す、請求項１から３のいずれか一項に記載の装置。
【請求項５】
プロセッサ状態の所定のサブセットをメモリに保存するために、コンテキスト保存トリガイベントに応答するコンテキスト保存回路を備え、プロセッサ状態の前記所定のサブセットは、クリーンデータ構造に対応する少なくとも１つのデータ構造識別子に対応する１つ以上のデータ構造メタデータレジスタのセットを含むが、前記少なくとも１つのデータ構造識別子に対応する前記データ構造を除外する、請求項１から４のいずれか一項に記載の装置。
【請求項６】
所与のデータ構造識別子に関連付けられた前記データ構造メタデータレジスタは、
前記命令デコーダ及び前記処理回路によってサポートされる非データ構造処理命令に応答してアクセス可能でもある複数の汎用レジスタの固定サブセットと、
前記複数の汎用レジスタとは別個のデータ構造メタデータレジスタの専用セットと
のうちの１つを含む、請求項１から５のいずれか一項に記載の装置。
【請求項７】
前記データ構造処理命令に応答して、前記処理回路は、所与の入力データ構造又は所与の出力データ構造の先行の値が前記処理回路に利用できない場合に、前記所与の入力データ構造又は前記所与の出力データ構造の前記先行の値を、メモリからロードするためのロード要求を生成するように構成されている、請求項１から６のいずれか一項に記載の装置。
【請求項８】
前記処理回路は、前記所与の入力データ構造が、前記メモリとは別個のデータ構造保持回路内で利用可能であるか否かをチェックするためのチェック回路を備える、請求項７に記載の装置。
【請求項９】
メモリとは別個のデータ構造保持回路を備え、
前記データ構造処理命令に応答して、前記命令デコーダは、前記処理回路を制御して、前記データ構造保持回路から前記少なくとも１つの入力データ構造を読み出させ、前記出力データ構造を前記データ構造保持回路に書き込ませる又は更新させるように構成されている、請求項１から８のいずれか一項に記載の装置。
【請求項１０】
所与のデータ構造が前記データ構造保持回路内に保持されると、前記処理回路は、前記データタが前記データ構造保持回路に書き込まれた順序とは異なる順序で、前記所与のデータ構造からデータを読み取ることができる、請求項９に記載の装置。
【請求項１１】
前記データ構造保持回路は、
アーキテクチャ的にアドレス指定可能なレジスタと、
非アーキテクチャ的にアドレス指定可能なレジスタと、
スクラッチパッドメモリと、
キャッシュと

のうちの少なくとも１つを備える、請求項９又は１０に記載の装置。
【請求項１２】
目標入力データ構造識別子又は目標出力データ構造識別子を指定するデータ構造ロード命令に応答して、前記命令デコーダは、処理回路を制御して、前記データ構造保持回路に、前記目標入力データ構造識別子又は前記目標出力データ構造識別子に対応する１つ以上のデータ構造メタデータレジスタの前記セット内に保持された前記アドレス表示メタデータから導出されたメモリアドレスからロードされたデータ構造を書き込ませるように構成されている、請求項９から１１のいずれか一項に記載の装置。
【請求項１３】
目標入力データ構造識別子又は目標出力データ構造識別子、及び１つ以上の汎用レジスタを指定するデータ構造ロード命令に応答して、前記命令デコーダは、前記処理回路を制御して、
前記データ構造保持回路に、前記１つ以上の汎用レジスタに保持された目標アドレス識別情報から導出されたメモリアドレスからロードされたデータ構造を書き込ませ、
前記目標アドレス識別情報又は前記データ構造ロード命令のプロパティに基づいて、前記目標入力データ構造識別子又は前記目標出力データ構造識別子に対応する１つ以上のデータ構造メタデータレジスタの前記セットを更新させる
ように構成されている、請求項９から１２のいずれか一項に記載の装置。
【請求項１４】
目標入力データ構造識別子を指定するロード準備命令に応答して、前記命令デコーダは、前記処理回路を制御して、
前記ロード準備命令によって暗黙的又は明示的に指定された目標アドレス識別情報と、
前記ロード準備命令のプロパティと
のうちの１つに基づいて、前記目標入力データ構造識別子に対応する１つ以上のデータ構造メタデータレジスタの前記セットを更新させるように構成されている、請求項１から１３のいずれか一項に記載の装置。
【請求項１５】
前記ロード準備命令に応答して、前記処理回路は、前記目標入力データ構造識別子に関連付けられたデータ構造が、前記目標アドレス識別情報に基づいて決定されたメモリアドレスからデータ構造保持回路に転送されることを要求するためのロード要求を生成するように構成されている、請求項１４に記載の装置。
【請求項１６】
目標出力データ構造識別子を指定する記憶準備命令に応答して、前記命令デコーダは、前記処理回路を制御して、
前記記憶準備命令によって暗黙的又は明示的に指定された目標アドレス識別情報と、
前記記憶準備命令のプロパティと
のうちの１つに基づいて、前記目標出力データ構造識別子に対応する１つ以上のデータ構造メタデータレジスタの前記セットを更新させるように構成されている、請求項１から１５のいずれか一項に記載の装置。
【請求項１７】
前記複数のメモリアドレスは、メモリアドレスの複数の不連続ブロックを含む、請求項１から１６のいずれか一項に記載の装置。
【請求項１８】
前記アドレス表示メタデータは、
メモリアドレスの前記複数の不連続ブロックのうちの少なくとも１つの開始アドレスを示す開始アドレス情報と、
メモリアドレスの前記複数の不連続ブロックの開始アドレス間の分離を示すオフセット情報と、
メモリアドレスの各不連続ブロックのサイズを示す第１のサイズ情報と、
前記データ構造を形成するメモリアドレスの前記不連続ブロックの数を示す第２のサ

イズ情報と、
前記データ構造のデータ要素サイズを示す要素サイズ情報と、
前記開始アドレス情報、前記オフセット情報、前記第１のサイズ情報、及び前記第２のサイズ情報のうちの少なくとも１つを保持する１つ以上の汎用レジスタを識別する１つ以上のレジスタ識別子と、
前記開始アドレス情報、前記オフセット情報、前記第１のサイズ情報、及び前記第２のサイズ情報のうちの少なくとも１つを保持するレジスタを指定する命令のアドレスを識別する命令アドレス表示と
のうちの少なくとも１つを含む、請求項１７に記載の装置。
【請求項１９】
前記データ構造処理命令は、２つの入力データ構造識別子を指定する行列乗算命令を含み、前記処理動作は、前記出力データ構造を生成するために、前記２つの入力データ構造識別子によって識別された２つの入力データ構造に対して実行される行列乗算演算を含む、請求項１から１８のいずれか一項に記載の装置。
【請求項２０】
前記出力データ構造は、少なくとも６４バイトのサイズを有する、請求項１から１９のいずれか一項に記載の装置。
【請求項２１】
前記データ構造処理命令に応答して、前記処理回路は、前記少なくとも１つの入力データ構造識別子又は前記出力データ構造識別子に対応する１つ以上のデータ構造メタデータレジスタの少なくとも１つのセットに保持された前記アドレス表示メタデータに応じて、前記少なくとも１つの入力データ構造上で実行される前記処理動作を適合させるように構成されている、請求項１から２０のいずれか一項に記載の装置。
【請求項２２】
データ処理方法であって、
少なくとも１つの入力データ構造識別子及び出力データ構造識別子を指定するデータ構造処理命令の復号に応答して、処理回路を制御して、前記出力データ構造識別子によって識別された出力データ構造を生成するために、前記少なくとも１つの入力データ構造識別子によって識別された少なくとも１つの入力データ構造上で処理動作を実行させることであって、前記少なくとも１つの入力データ構造及び前記出力データ構造は各々、複数のメモリアドレスに対応するデータの配列を含む、ことと、
１つ以上のデータ構造メタデータレジスタの複数のセットにアドレス表示メタデータを保持することであって、１つ以上のデータ構造メタデータレジスタの各セットは、対応するデータ構造識別子に関連付けられ、前記対応するデータ構造識別子によって識別された前記データ構造のための前記複数のメモリアドレスを識別するためのアドレス表示メタデータを保持するように指定される、ことと
を含む、方法。
【請求項２３】
目標データ処理装置による目標プログラムの実行をシミュレートするためにホストデータ処理装置を制御するためのコンピュータプログラムを記憶する非一時的記憶媒体であって、
前記コンピュータプログラムは、
前記目標プログラムの命令を復号するための命令復号プログラムロジックであって、少なくとも１つの入力データ構造識別子及び出力データ構造識別子を指定するデータ構造処理命令に応答して、前記命令復号プログラムロジックは、前記ホストデータ処理装置を制御して、前記出力データ構造識別子によって識別された出力データ構造を生成するために、前記少なくとも１つの入力データ構造識別子によって識別された少なくとも１つの入力データ構造上で処理動作を実行させるように構成されており、前記少なくとも１つの入力データ構造及び前記出力データ構造は各々、複数のメモリアドレスに対応するデータの配列を含む命令復号プログラムロジックと、

前記目標データ処理装置のレジスタをエミュレートするための記憶構造へのアクセスを制御するレジスタエミュレートプログラムロジックであって、前記レジスタは、１つ以上のデータ構造メタデータレジスタの複数のセットを含み、１つ以上のデータ構造メタデータレジスタの各セットは、対応するデータ構造識別子に関連付けられており、前記対応するデータ構造識別子によって識別された前記データ構造のための前記複数のメモリアドレスを識別するためのアドレス表示メタデータを保持するように指定される、レジスタエミュレートプログラムロジックと

を備える、非一時的記憶媒体。

【発明の詳細な説明】

【技術分野】

【０００１】

本技術は、データ処理分野に関するものである。

【発明の概要】

【発明が解決しようとする課題】

【０００２】

いくつかのデータ処理アプリケーションは、複数のメモリアドレスに記憶されたメモリ内のデータの配列を含むデータ構造に適用される処理動作を必要とする場合がある。このようなデータ構造の処理は、例えば、機械学習、信号処理、又は圧縮アルゴリズムなどの様々な用途に有用であり得る。

【０００３】

少なくともいくつかの実施例は、命令を復号するための命令デコーダと、命令デコーダによって復号された命令に応答してデータ処理を実行する処理回路とを備える装置を提供し、少なくとも１つの入力データ構造識別子及び出力データ構造識別子を指定するデータ構造処理命令に応答して、命令デコーダは、処理回路を制御して、出力データ構造識別子によって識別された出力データ構造を生成するために、少なくとも１つの入力データ構造識別子によって識別された少なくとも１つの入力データ構造上で処理動作を実行させるように構成されており、少なくとも１つの入力データ構造及び出力データ構造は各々、複数のメモリアドレスに対応するデータの配列を含み、本装置は、１つ以上のデータ構造メタデータレジスタの複数のセットを含み、１つ以上のデータ構造メタデータレジスタの各セットは、対応するデータ構造識別子に関連付けられ、対応するデータ構造識別子によって識別されたデータ構造のための複数のメモリアドレスを識別するためのアドレス表示メタデータを保持するように指定される。

【０００４】

少なくともいくつかの実施例は、データ処理方法を提供し、少なくとも１つの入力データ構造識別子及び出力データ構造識別子を指定するデータ構造処理命令の復号に応答して、処理回路を制御して、出力データ構造識別子によって識別された出力データ構造を生成するために、少なくとも１つの入力データ構造識別子によって識別された少なくとも１つの入力データ構造上で処理動作を実行させることであって、少なくとも１つの入力データ構造及び出力データ構造は各々、複数のメモリアドレスに対応するデータの配列を含む、ことと、１つ以上のデータ構造メタデータレジスタの複数のセットにアドレス表示メタデータを保持することであって、１つ以上のデータ構造メタデータレジスタの各セットは、対応するデータ構造識別子に関連付けられ、対応するデータ構造識別子によって識別されたデータ構造のための複数のメモリアドレスを識別するためのアドレス表示メタデータを保持するように指定される、こととを含む。

【０００５】

少なくともいくつかの例は、目標データ処理装置による目標プログラムの実行をシミュレートするためにホストデータ処理装置を制御するためのコンピュータプログラムを記憶する非一時的記憶媒体を提供し、コンピュータプログラムは、目標プログラムの命令を復号するための命令復号プログラムロジックであって、少なくとも１つの入力データ構造識別子及び出力データ構造識別子を指定するデータ構造処理命令に応答して、命令復号プ

ログラムロジックは、ホストデータ処理装置を制御して、出力データ構造識別子によって識別された出力データ構造を生成するために、少なくとも１つの入力データ構造識別子によって識別された少なくとも１つの入力データ構造上で処理動作を実行させるように構成されており、少なくとも１つの入力データ構造及び出力データ構造は各々、複数のメモリアドレスに対応するデータの配列を含む、命令復号プログラムロジックと、目標データ処理装置のレジスタをエミュレートするための記憶構造へのアクセスを制御するレジスタエミュレートプログラムロジックであって、レジスタは、１つ以上のデータ構造メタデータレジスタの複数のセットを含み、１つ以上のデータ構造メタデータレジスタの各セットは、対応するデータ構造識別子に関連付けられ、対応するデータ構造識別子によって識別されたデータ構造のための複数のメモリアドレスを識別するためのアドレス表示メタデータを保持するように指定される、レジスタエミュレートプログラムロジックとを備える。

【図面の簡単な説明】

【０００６】

本技術の更なる態様、特徴、及び利点は、添付の図面と併せて読まれるべき以下の実施例の説明から明らかとなるであろう。

【図１】図１は、データ構造処理命令のための命令デコーダサポートを有する処理回路を含むデータ処理装置の一例を概略的に示す。

【図２】図２は、データ構造が行列である例を示す。

【図３】図３は、複数の入力及び出力データ構造を記憶するためのデータ構造記憶領域の例を示す。

【図４】図４は、特定のデータ構造がメモリ内に記憶されているアドレスを識別するためにアドレス表示メタデータを使用する例を示す。

【図５】図５は、処理回路のレジスタ及びデータ構造記憶装置の例をより詳細に示す。

【図６】図６は、入出力データ構造が別個の識別子空間を有する実施例を示す。

【図７】図７は、所与のデータ構造に関するデータ構造メタデータレジスタが汎用レジスタの固定サブセットである例を示す。

【図８】図８は、汎用レジスタとは別個の専用データ構造メタデータレジスタが提供される代替アプローチを示す。

【図９】図９は、コンテキスト保存トリガイベントに応答して状態保存を制御するための方法を示すフロー図である。

【図１０】図１０は、データ構造処理命令が、処理回路に所与の入力データ構造を、これがまだ利用可能ではないときにロードさせる別の実施例を示す。

【図１１】図１１は、コンテキスト状態回復を制御する方法を示すフロー図である。

【図１２】図１２は、使用される可能性のあるシミュレータの例を示す図である。

【発明を実施するための形態】

【０００７】

データ処理装置は、プログラム命令を復号するための命令デコーダと、命令デコーダによって復号された命令に応答してデータ処理を実行する処理回路とを有し得る。例えば、命令デコーダは、復号されたプログラム命令を、処理回路に送信されたマイクロ動作又は他の制御信号にマッピングして、どの動作が処理回路によって実行されるかを制御することができる。いくつかのソフトウェアワークロードは、複数のメモリアドレスに対応するメモリ内の場所に記憶されたデータの配列を含む特定のデータ構造の処理を必要とすることがある。例えば、いくつかの信号処理又は機械学習アルゴリズムは、データの２次元配列である行列に適用される動作を含み得る。また、いくつかの圧縮アルゴリズムは、所与のサイズのブロック上で動作してもよく、そのため、データ構造は、例えばそれらのブロックのうちの１つに対応してもよい。

【０００８】

プログラマが、このような処理アプリケーションを実施するために高レベルコードを書き込むとき、プログラマは典型的には、行列乗算演算又は所与のブロックに適用される圧縮アルゴリズムの反復を実施するための命令などの、そのようなデータ構造の粒度で適

用される特定の基本演算を定義する関数を有するプログラミング言語を使用してコードを書き込むことができる。しかしながら、実際には、そのような高レベルコードが処理回路によってサポートされるネイティブ命令セットにコンパイルされるとき、そのような高レベル関数は、多くの場合、データ構造内の個々のデータ値に適用される複数の個々のロード／記憶命令及び算術命令に分解してもよく、各データ値は単一のメモリアドレスに対応する。例えば、行列乗算演算は、行列の個々の要素をロードするための複数の個々のロードと、各要素のペアを乗算するための乗算命令とにコンパイルされ、その後にいくつかの記憶命令によって、結果をメモリ内の関連するアドレスに再び記憶する。メモリ内の関連するアドレスからの、又は関連するアドレスへのロード及び記憶をどのように整列させ、スカラ命令を使用して乗算を実行するかを決定することは、コンパイラ次第であり得る。

【０００９】

データ構造（複数のアドレスにまたがるデータの配列を含む）の処理を加速するための１つのアプローチは、このようなデータ構造の処理を高速化するように設計された専用ハードウェアを有するハードウェアアクセラレータを提供することであり得る。典型的には、このようなハードウェアアクセラレータは、メモリシステムに対して実行されるロード／記憶動作を通じてアクセスすることができるデバイスとしてシステムオンチップ内に実装してもよい。ＣＰＵ（中央処理装置）の観点から、ＣＰＵは、データ構造処理を実行するようにハードウェアアクセラレータを制御するメモリに単にデータを書き込んでもよく、次いで、ハードウェアアクセラレータがその動作を終了した後、ＣＰＵは、メモリから結果を読み戻し、それらの処理を継続してもよい。しかしながら、ＣＰＵの命令デコーダによってサポートされる命令が専用のデータ構造処理命令をサポートしない、そのようなハードウェアアクセラレータアプローチは、何らかのワークロードについていくつかの問題を有し得る。第１に、ハードウェアアクセラレータの使用は、データ構造処理が、実行される動作の大部分を表すワークロードに適している可能性があるが、そのようなデータ構造の処理を他のタイプの処理動作と散在させる混合ワークロードの場合、ＣＰＵとハードウェアアクセラレータとの間でデータを前後に転送する必要がある場合が頻繁に発生し、例えば、ＣＰＵにおいて命令が、データ構造処理の結果がハードウェアアクセラレータから読み戻されるのを待つ必要がある場合、アプリケーションの全体的な性能を大幅に低下させる可能性があるため、ハードウェアアクセラレータを使用することは非効率的であり得る。別の問題は、単一のハードウェアアクセラレータが複数のＣＰＵ間で、又は同じＣＰＵ上で実行されている複数のソフトウェアタスク間で共有されるシステムでは、所与のタスクがハードウェアアクセラレータ上で十分な処理時間を得ることが困難であり、これがまた性能を低下させ得ることである。したがって、競合用途の数が増加するにつれて、ハードウェアアクセラレータのアプローチは、ハードウェアアクセラレータを並列に使用することが予想されるタスクが比較的少ない場合には、このアプローチは持続不可能であり得る。

【００１０】

したがって、データ構造内の個々の要素とは対照的に、全体のデータ構造の粒度で処理動作を実行することをサポートする、ＣＰＵアーキテクチャ自体内の専用データ構造処理命令のための命令デコーダサポートを提供することが有用であり得る。当然ながら、命令デコーダはまた、個々の要素上での動作を可能にする他の命令もサポートすることができる。しかしながら、処理回路を制御して、命令によって指定された少なくとも１つの入力データ構造識別子によって識別された少なくとも１つの入力データ構造上で、処理動作を実行させる命令を定義することによって、命令によって指定された出力データ構造識別子によって識別された出力データ構造を生成し、入力データ構造及び出力データ構造の各々は、複数のメモリアドレスにまたがるデータの配列であり、これにより、そのようなデータ構造の著しい量の処理に依存するソフトウェアアプリケーションの性能を大幅に促進するが、上述したようなハードウェアアクセラレータに関連する性能及びスケーラビリティの問題を伴わない。

【００１１】

しかしながら、データ構造処理命令のための命令デコーダサポートがＣＰＵ又は他の処理要素にもたらされると、これは、コンテキスト切り替え待ち時間の点で問題を引き起こし得る。ＣＰＵ又はプログラム命令を復号するための命令デコーダを有する他の処理要素について、第１のソフトウェアワークロードと第２のソフトウェアワークロードとの間のコンテキスト切り替えが実行されることが一般的であり、コンテキスト切り替えの前に実行されている第１のワークロードに関連付けられた任意のアーキテクチャ状態は、メモリに保存される必要があることがあり、その結果、第２のソフトウェアワークロードが、処理要素のレジスタ又は他の記憶要素内のその状態を上書きすることができる。第１のワークロード内の状態をメモリに保存することによって、これは、第１のソフトウェアワークロードが後で再開する必要がある場合には、その後、その状態をロードし直すことができ、その後、到来ワークロードの処理が中断された時点から再開することができることを意味する。

【００１２】

しかしながら、データ構造処理命令のサポートが提供される場合、これは、いくつかの実装が、プロセッサにローカルなデータ構造を記憶するための記憶要素を提供し得ることを意味し得、コンテキスト切り替えが発生すると、この状態をメモリに保存すべきであると予想され得る。しかしながら、データ構造が有意なサイズ（例えば、６４バイト、２５６バイト、又は更には１キロバイト以上）のものであり得、処理回路の記憶回路によって保持される複数のそのようなデータ構造が存在することがあり、そのため、その状態の全てをメモリに保存することは、コンテキスト切り替え待ち時間又は割り込み処理待ち時間に多大な影響を及ぼし得る。

【００１３】

以下で論じられる技術では、処理回路及びデータ構造処理命令をサポートする命令デコーダを有する装置には、２つ以上のデータ構造メタデータレジスタのセットが提供され、各セットは１つ以上のレジスタを有する。データ構造メタデータレジスタの各セットは、対応するデータ構造識別子（入力データ構造識別子又は出力データ構造識別子のいずれかであり得る）に関連付けられる。１つ以上のデータ構造メタデータレジスタの各セットは、対応するデータ構造識別子によって識別されたデータ構造に対応するメモリアドレスを識別するためのアドレス表示メタデータを保持するために指定される。したがって、データ構造メタデータレジスタは、対応するデータ構造がメモリ内で発見され得る場所に関する情報を提供する。データ構造メタデータレジスタは、このアドレス表示メタデータをアーキテクチャレベルで保持するものとして指定されるため、データ構造を処理している間に、対応するアドレス表示メタデータは、１つ以上のデータ構造メタデータレジスタの対応するセット内に発見され得ることが理解される。例えば、特定のデータ構造識別子に対してどのレジスタがデータ構造メタデータレジスタとして使用されるかの固定の割り当てがあってもよい。

【００１４】

典型的には、プロセッサがメモリから値をロードするとき、プログラマ又はコンパイラは、データがロードされるアドレスを定義するレジスタとして、プロセッサの複数の汎用レジスタのいずれかを自由に選択することができるため、このアプローチはまれである。また、典型的なロード／記憶アーキテクチャでは、所与のメモリアドレスからデータがロードされると、そのアドレス情報は典型的には保持されず、ロード目標アドレスを計算するために使用されるレジスタは、多くの場合、他の情報で上書きされ得る。したがって、最も多くのプロセッサアーキテクチャは、対応するデータ構造に対するアドレス表示メタデータを継続的に保持するために割り当てられたレジスタとして、特定のレジスタを指定しない。

【００１５】

しかしながら、本発明者らは、データ構造の関連するメモリアドレスがメモリアドレス空間内にある場所を識別するメタデータを保持するソフトウェアによって理解され得る指定されたデータ構造メタデータレジスタを提供することによって、これは、状態保存を

制御する役割を担うオペレーティングシステム若しくは他のソフトウェア、又は、状態保存を行うためのいくつかのマイクロアーキテクチャの実装において提供されたコンテキスト保存ハードウェアは、データ構造自体全体を記憶する必要なくメモリからロードされたため、変更されていない特定のデータ構造のためのメタデータレジスタのみを保存することを選択することができることを意味するため、コンテキスト切り替え待ち時間を大幅に改善することができる。対照的に、データ構造がロードされている対応するアドレスに関する情報が失われた場合、コンテキスト保存ルーチンは、データ構造全体を保存して、現在のコンテキストの処理が回復されたときにデータを復元できるようにする必要があり、これは、どのアドレスが対応するデータ構造をメモリに記憶するか知られていないため、更新されていないクリーンなデータ構造でさえ復元する必要がある。したがって、対応するデータ構造のアドレス表示メタデータを保持するために指定されたアーキテクチャレベルで指定されたレジスタを提供することは、コンテキスト切り替え待ち時間を改善するために利用することができるアーキテクチャ機能を提供する。

【００１６】

処理回路は、データ構造識別子の少なくともサブセットについて、データ構造識別子のサブセットのうちの１つに対応するデータ構造がクリーン又はダーティであるかを、ソフトウェアが識別することを可能にする命令セットアーキテクチャに従って動作し得る。必須ではないが、マイクロアーキテクチャの実装では、メモリから直接データ構造にアクセスするよりも高速にアクセスできるデータ構造を保持するためのローカル記憶装置を提供することを選択することが一般的であり得る。例えば、ローカルデータ構造記憶装置は、プロセッサ内に提供されるレジスタであってもよく、又はスクラッチパッドメモリ又はキャッシュであってもよい。このようなローカルデータ構造記憶装置が提供される場合、処理回路が、記憶装置内に保持された所与のデータ構造を更新する場合、そのデータ構造は、メモリシステム内の関連するメモリアドレスに記憶された対応するデータ構造とは異なり得るため、ダーティであってもよい。処理回路上で実行されるソフトウェアプログラムが、データ構造識別子の少なくともあるサブセットがクリーン又はダーティなデータ構造に対応するかどうかを判定することを可能にするアーキテクチャサポート機能を提供することによって、これはコンテキスト切り替え上の状態保存をスピードアップするのを助けることができ、これは、状態保存に関与するソフトウェア又は回路が、任意のクリーンなデータ構造を、その状態保存の一部としてメモリに保存する必要がないことを識別できることを意味し、代わりに、クリーンなデータ構造のためのデータ構造メタデータレジスタのセットに保持されたアドレス表示メタデータによって示される関連するメモリアドレスから、後でそれらを処理回路にロードし直すことができる。対照的に、ダーティなデータ構造は、関連するメモリアドレスにおける基礎データとは異なる場合があるため、これらのダーティなデータ構造は、コンテキスト切り替えで実行される状態保存の一部として保存してもよい。なお、ダーティなデータ構造については、状態保存中に、これらのダーティなデータ構造は、アドレス表示メタデータによって識別される実際のメモリアドレスに記憶しなくてもよい。代わりに、コンテキスト切り替えで実行される状態保存は、任意のプロセッサ状態を、状態保存の制御を担うオペレーティングシステム又は他のソフトウェアに関連付けられたメモリアドレスの異なるセットにおけるデータ構造に保存することができる。

【００１７】

したがって、一般に、処理回路は、データ構造識別子の少なくともサブセットがクリーンであることをソフトウェアが識別することを可能にする特徴を有する命令セットアーキテクチャに従って、動作することが有用である。全てのデータ構造識別子は、クリーン又はダーティとして識別可能である必要はなく、その場合、データ構造がクリーンであるかダーティであるか知られていないいずれかのデータ構造識別子は、コンテキスト切り替え上の状態保存を受けることができる。

【００１８】

命令セットアーキテクチャが、ソフトウェアが特定のデータ構造識別子がクリーンで

あるか又はダーティであるかを識別することを可能にし得る、複数の方法が存在し得る。一例では、（データ構造処理命令に応答して生成される出力データ構造を識別するために使用される）出力データ構造識別子は、（データ構造処理命令のオペランドとして使用される少なくとも１つの入力データ構造を識別する）少なくとも１つの入力データ構造識別子に対して完全に別個の識別子空間内で定義してもよい。したがって、１つ以上のデータ構造メタデータレジスタの少なくとも１つのセットは、入力データ構造を表すためにのみ使用することができ、出力データ構造を表すために使用することができないデータ構造識別子と、関連付けられてもよい。入力及び出力データ構造の識別子を別個の識別子空間に定義することにより、これらの識別子は、データ構造処理命令に応答して更新される出力データ構造には決して使用できないため、入力データ構造識別子に関連付けられたデータ構造は、常にクリーンであることが保証され得ることを意味する。このアプローチでは、入力データ構造識別子はデータ構造処理命令への読み出し専用入力を効果的に識別し、データ構造処理命令はメモリからロードされると、メモリシステムに記憶された対応するデータ構造のコピーとして作用するが、メモリに対して更新することはできない。したがって、このアプローチでは、状態保存を制御するオペレーティングシステム又は他のソフトウェアは、入力データ構造識別子空間内の入力データ構造識別子に対応するデータ構造のいずれもクリーンであり、そのため、状態保存中にメモリに保存する必要がないことを暗黙的に判定することができる。代わりに、データ構造自体を保存するのではなく、１つ以上のデータ構造メタデータレジスタの関連するセットから対応するメタデータを保存するだけで十分であり得る。これにより、コンテキスト切り替え上のメモリに保存されるデータの量を大幅に低減し、コンテキスト切り替え待ち時間を改善することができる。

【００１９】

あるいは、データ構造がクリーンであるかダーティであるかを識別するための別のアーキテクチャアプローチは、少なくとも１つのダーティインジケータを保持するために少なくとも１つのアーキテクチャレジスタが提供され得ることであってもよく、各ダーティインジケータは、所与のデータ構造識別子に対応するデータ構造がクリーンであるかダーティであるかを示す。いくつかの例では、全てのデータ構造識別子に対してダーティインジケータが提供され得る。あるいは、ダーティインジケータは、データ構造識別子のサブセットに対してのみ提供され得る。このアプローチは、データ構造識別子が入力データ構造用と出力データ構造用の別個の識別子空間にそれぞれ定義される必要がないことを意味する。これにより、データ構造識別子の共通のプールを入力データ構造又は出力データ構造のいずれかに使用することが可能になり、これは、１つのデータ構造処理命令から生じる出力データ構造を後の演算に対する入力オペランドとして使用する必要がある場合に利点を有してもよい。あるいは、入力及び出力データ構造識別子を別個の識別子空間に分離するアーキテクチャであっても、出力データ構造識別子に関連付けられた何らかのダーティインジケータを提供することにより、オペレーティングシステムが、所与の出力データ構造識別子に関連付けられた出力データ構造がロードされてから実際に更新されたかどうかを判定することを可能にすることができ、これにより、クリーンなデータ構造をメモリに保存する必要を回避する更なる機会を提供することができる。ダーティインジケータを保持するために使用されるアーキテクチャレジスタは、上述のデータ構造メタデータレジスタから分離することができるか、又は、レジスタの共通セットは、アドレス表示メタデータ及び少なくとも１つのダーティインジケータの両方を保持することができる。例えば、所与のデータ構造識別子のダーティインジケータは、所与のデータ構造識別子のアドレス表示メタデータとデータ構造メタデータレジスタの同じセットに保持することができる。

【００２０】

上述したように、データ構造メタデータレジスタのアーキテクチャ上の提供は、データ構造がクリーンであることが知られている場合、特定のデータ構造識別子が、基礎データ構造ではなく、状態保存プロセスの一部として保存された対応するメタデータのみを有することを可能にし得るので、コンテキスト保存の効率を改善するのに役立つ。多くの場合、状態保存プロセスは、プロセッサ上で実行するソフトウェアによって制御してもよく

、そのため、状態保存動作を実際に実行するためのハードウェアは存在しなくてもよい。例えば、状態保存は、アプリケーション間のコンテキスト切り替え時にオペレーティングシステムによって実行してもよく、又は仮想マシン若しくはオペレーティングシステム間の切り替え時にハイパーバイザによって実行してもよい。それにもかかわらず、上述したアーキテクチャ特徴は、コンテキスト保存動作を高速化するためにソフトウェアによって実行されるそのようなコンテキスト保存をサポートする。

【００２１】

あるいは、命令セットアーキテクチャによって必ずしも必要とされないが、いくつかのマイクロアーキテクチャ実装は、コンテキスト保存トリガイベントに応答してコンテキスト保存回路を提供して、プロセッサ状態の所定のサブセットをメモリに保存することを選択してもよい。プロセッサ状態の所定のサブセットは、クリーンデータ構造に対応する少なくとも１つのデータ構造識別子に対応する１つ以上のデータ構造メタデータレジスタのセットを含んでもよいが、少なくとも１つのデータ構造識別子に対応するデータ構造を除外してもよい。いくつかのプロセッサ状態をメモリに記憶するコンテキスト保存回路をハードウェアに提供することによって、純粋にソフトウェア制御された実施例と比較してコンテキスト切り替えを更に早めることができ、コンテキストの保存が完了するまで進行を遅延させるのではなく、他の命令の実行と並行して、ハードウェア制御コンテキスト保存を進めることを可能にする。例えば、ハードウェア内に提供されたいくつかのコンテキスト保存回路を提供して、入力データ構造識別子又はクリーンな出力データ構造識別子に関連付けられたアドレス表示メタデータからメモリ内の場所に自動的に保存することができる。

【００２２】

データ構造メタデータレジスタは、異なる方法で実装することができる。一例では、プロセッサアーキテクチャの任意の汎用レジスタとは別に、メタデータレジスタの専用セットを提供してもよい。汎用レジスタは、整数演算又は論理命令を含む演算命令の一般的なオペランド及び結果に使用することができる。汎用レジスタはまた、命令をロード又は記憶するためのアドレスを計算するためのオペランドとして使用してもよい。データ構造メタデータレジスタの専用セットを提供することにより、これは、ソフトウェアが他の目的のために汎用レジスタを使用する必要がある場合でも、データ構造のアドレス表示メタデータをアーキテクチャ内に保存できることを意味する。あるいは、別のアプローチは、専用データ構造メタデータレジスタを提供する代わりに、データ構造メタデータレジスタは、処理回路上の命令デコーダによってサポートされる非データ構造処理命令に応答してもアクセス可能である汎用レジスタの固定サブセットであることである。例えば、所与のデータ構造識別子に関して、対応するデータ構造のアドレス表示メタデータが、汎用レジスタの特定の直結されたサブセット内に保持されることを暗示してもよい。このアプローチは、ハードウェアに提供される必要があるレジスタの総数を低減することができる。しかしながら、このアプローチでは、ソフトウェアは、所与のデータ構造のアドレス表示メタデータが確立された時点から、対応するデータ構造がもはや必要とされなくなるまで、他の命令が、対応するデータ構造識別子の１つ以上のデータ構造メタデータレジスタのセットとして使用される汎用レジスタの固定サブセットに書き込まれないことを保証して、コンテキスト保存及び復元を制御する必要がある場合にアドレス情報を保持できることを保証する必要があり得る。

【００２３】

いくつかの実施例では、データ構造処理命令に応答して、処理回路は、所与の入力データ構造が処理回路に利用不可能であるときに、所与の入力データ構造をメモリからロードするためのロード要求を生成してもよい。マイクロプロセッサがロード／記憶アーキテクチャに従って動作することは一般的であり、メモリからデータをロードし、メモリにデータを戻して記憶するためのロード／記憶命令は、メモリから以前にロードされたオペランドに対して算術演算、論理演算、又は他の処理動作を実行するデータ処理命令とは別個である。したがって、データ構造をメモリからローカル記憶装置にロードするために、デ

ータ構造処理命令とは別の命令が定義されるべきであり、その後、データ構造処理命令はロード自体を実行する必要がないと予想され得る。このロード／記憶アプローチは、上述のデータ構造処理命令のためのサポートを含むアーキテクチャを実装する１つの方法であってもよい。

【００２４】

しかしながら、別の例では、データ構造処理命令自体はまた、処理回路が所与の入力データ構造を利用できないときに、メモリからそれをロードするためのロード要求を処理回路に生成させてもよい。場合によっては、実行されるデータ構造処理動作が出力データ構造の以前の値及び入力データ構造の値に依存する場合、データ構造処理命令はまた、メモリから出力データ構造の以前の値をロードするためのロード要求の生成をトリガすることもできる。例えば、処理動作は、入力データ構造を処理した結果の値が出力データ構造の以前の値に加算されて出力データ構造の新しい値を決定する累積演算であり得る。したがって、場合によっては、データ構造処理命令は、処理動作に含まれる入力及び出力データ構造のうちの１つ以上に対するロード要求をトリガすることができる。

【００２５】

前の命令がメモリからデータ構造をロードした場合と比較して、データ構造処理命令に応答した時点でデータ構造をロードすると、データ構造に対して実行される処理動作の開始を遅延させる可能性があるため、このアプローチは直感に反すると考えるかもしれない。しかしながら、本発明者らは、所与の入力データ構造が利用不可能であるときに、所与の入力データ構造をメモリにロードする必要があるように、アーキテクチャ内のデータ構造処理命令を定義することによって、ソフトウェアワークロード間でコンテキスト切り替えを実行する際にいくつかの利点を提供することを認識した。コンテキスト切り替えの上記の考察は、ほとんどが状態保存態様に焦点を当てているが、コンテキスト切り替えの別の部分は、コンテキスト切り替えの後に実行される到来コンテキストの状態の復元である。上述のようなデータ構造メタデータレジスタの提供は、状態保存動作の一部としてデータ構造の一部を保存する必要性を回避するのに役立つことができるが、それらのデータ構造がクリーンである場合、それにもかかわらず、コンテキスト切り替えから所与のソフトウェアワークロードの処理に戻ると、そのワークロードによって必要とされる任意のデータ構造を再びロードする必要があり、この状態復元は、データ構造が比較的大きなサイズを有することがあるため、時間がかかる可能性がある。

【００２６】

所与のデータ構造処理命令に必要なデータ構造が、データ構造処理命令に先行する別個のロード命令によってロードされ、データ構造処理命令自体が必要に応じてデータ構造をロードする能力を有さないアーキテクチャでは、その後ロード命令と後続のデータ構造処理命令との間でコンテキスト切り替えをトリガするための割り込みが発生する可能性があるため、状態復元を実行するオペレーティングシステム又はハードウェア回路は、システムがそのワークロードに戻った後にそのワークロードから離れるコンテキスト切り替え前に、所与のソフトウェアワークロードのためのローカル記憶装置に保持されていた所与のデータ構造が依然として存在することを保証できないと仮定しなければならない場合がある。したがって、データ構造処理命令自体がデータ構造をロードする能力を有さないアーキテクチャでは、これは、コンテキスト切り替え後に実行される復元されたソフトウェアワークロードのデータ構造状態を復元することがオペレーティングシステムにとって不可欠であることを意味し得る。

【００２７】

このことは、状態回復を実行する完全に実現可能な方法であるが、監視ソフトウェア（例えば、オペレーティングシステム又はハイパーバイザ）がデータ構造の復元に関与しなければならない場合、これはいくつかの欠点を有する。第１に、状態復元を担当する監視ソフトウェアは、到来ワークロードにおける処理の再開後にどのデータ構造が実際に使用されることになるかについての可視性を有さない可能性があり、そのため、オペレーティングシステム又は監視ソフトウェアは、到来ワークロードのその後の命令によって実際

には使用されないデータ構造を復元する時間を浪費する可能性がある。対照的に、データ構造処理命令がまだ利用できないときに、入力（又は出力）データ構造にロードすることを要求することによって、これは、特定のデータ構造を必要とする任意の後続のデータ構造処理命令が実行されるかのように、オペレーティングシステムがデータ構造を復元することを心配する必要がないことを意味し、その場合、その構造はロードされ、処理は継続することができるが、ソフトウェアによって再び使用されることがない任意のデータ構造は復元されないため、状態復元中に余分な待ち時間が不必要に生じることが回避される。

【００２８】

データ構造処理命令が利用不可能なデータ構造にロードされるアプローチの別の利点は、これがセキュリティを改善することができることである。典型的には、監視ソフトウェアが状態回復を実行するとき、監視ソフトウェアのメモリの割り当ての一部ではないが、状態が復元されているスレッドに関連付けられたメモリの割り当ての一部である、メモリの一部の領域から状態を復元することであり得る。したがって、悪意のあるアプリケーションが、一組の管理者アドレスをアドレス表示メタデータにロードすることができ、これはその後、状態回復によってアクセスされることがあり、監視ソフトウェアのメモリ割り当てをサイドチャネル攻撃にさらすリスクがある。対照的に、所与の入力データ構造がまだ利用可能でない場合にそれをロード（又は再ロード）するデータ構造処理命令を提供することによって、これは、データ構造アドレス表示メタデータを監視ソフトウェアに使用する必要がないことを意味し、悪意のある者が監視ソフトウェアを攻撃するこの潜在的な機会を閉じることができる。

【００２９】

いくつかの実装形態では、マイクロアーキテクチャは、関連するデータ構造処理命令が受信されるまで所与の入力（又は出力）データ構造のロードを常に延期することができるため、そのデータ構造処理命令を復号／処理する前に、所与のデータ構造処理命令によって要求されるデータ構造をロードする機能を提供しないことがある。また、いくつかの実装形態では、データ構造処理命令のアーキテクチャ機能は、必要に応じて要求に応じて処理回路にロードされ、次いでメモリに書き戻されるメモリ内のデータ構造に直接作用する回路を介して実装してもよく、そのため、追加のローカル記憶装置はまったく必要ないことがある。

【００３０】

しかしながら、メモリへのアクセスは比較的遅い場合があるため、他の実装形態は、メモリとは別個のデータ構造記憶回路として機能する何らかのローカル記憶装置を提供することを選択することができ、これはメモリ自体よりも速くアクセスされ得、最近使用されたデータ構造又はこれから実行されるデータ構造処理命令に対して要求されるデータ構造のための何らかの一時的記憶装置を提供することができる。したがって、そのような実装形態は、所与のデータ構造処理命令に必要な所与の入力データ構造（又は出力データ構造）がデータ構造記憶回路で既に利用可能であるかどうかをチェックするためのチェック回路を処理回路に提供することができる。例えば、それぞれのデータ構造識別子に関連付けられたデータ構造が利用可能であるか利用不可能であるかを示すデータ構造可用性情報を保持するために少なくとも１つの可用性レジスタが設けられてもよく、処理回路は、そのデータ構造可用性情報に基づいて所与の入力データ構造の可用性を判定してもよい。可用性レジスタは、アドレスメタデータレジスタ自体の一部であってもよく、又はレジスタの別個のセットであってもよい。このアプローチでは、所定のデータ構造処理命令を実行する前にデータ構造をロードするためにプリロード動作が実行される場合、可用性情報は、対応するデータ構造が既に利用可能であることを示すように設定することができ、これは、データ構造処理命令に応答してメモリから対応するデータ構造を再ロードする必要がないことをチェック回路に示すことができる。したがって、いくつかの例では、処理回路は、データ構造記憶回路にまだ保持されていない場合、又はデータ構造処理命令に遭遇する前にメモリからのそのデータ構造のロードがまだ要求されていない場合に、所与の入力データ構造又は出力データ構造が利用できないと判定してもよい。事前のデータ構造のプ

リロードは、命令デコーダによって復号された明示的なロード若しくはプリロード命令に応答して、又は（例えば、同じコードを実行した過去の事例における過去の挙動の追跡に基づいて）将来の命令によって必要とされ得るデータ構造の予測を行ってもよいプリロード回路によって、行うことができる。

【００３１】

メモリとは別個のデータ構造記憶回路を提供するシステムでは、命令デコーダは、データ構造処理命令に応答して、処理回路を制御して、データ構造記憶回路から少なくとも１つの入力データ構造を読み取らせ、出力データ構造をデータ構造記憶回路に書き込ませるか又は更新させるようにしてもよい。データ構造記憶回路は、異なる方法で実装することができる。場合によっては、データ構造記憶回路は、対応するデータ構造を読み出すソフトウェア命令によって明示的に参照され得るアーキテクチャ的にアドレス指定可能なレジスタであってもよい。あるいは、データ構造記憶回路は、命令セットアーキテクチャに準拠するために必須ではないかもしれないが、処理システムの特定のマイクロアーキテクチャ実装の設計者によって選択されたように任意選択的に提供することのできる、非アーキテクチャ的にアドレス指定可能なレジスタを備えてもよい。あるいは、データ構造記憶回路は、スクラッチパッドメモリ又はキャッシュを含み得る。

【００３２】

場合によっては、所与のデータ構造がデータ構造記憶回路内に保持されると、処理回路は、データがデータ構造記憶回路に書き込まれた順序とは異なる順序で、所与のデータ構造からデータを読み取ることができてもよい。これは、データ構造がメモリに記憶され得る異なるデータ構造形式に対処するための柔軟性を提供することができる。例えば、データ構造がデータの行列又は他の２次元配列である場合、データはメモリ内の行方向又は列方向のパターンに記憶されることができ、行方向又は列方向パターンのいずれかにデータの読み出し又は書き戻しをサポートする読み出し／書き込みポートをデータ構造記憶回路に提供することが有用であり得る。

【００３３】

データ構造処理命令自体と同様に、データ構造と相互作用するために複数の他の命令を定義することができる。一例は、目標入力データ構造識別子又は目標出力データ構造識別子を指定するデータ構造ロード命令であってもよく、これに応答して、命令デコーダは、処理回路を制御して、メモリからロードされたデータ構造をデータ構造記憶回路に書き込ませる。そのようなデータ構造ロード命令は、データ構造処理命令自体にロード能力が与えられないアーキテクチャにおいて特に有用であり得る。この場合、データ構造処理命令を実行する前に、ソフトウェアは、関連する入力又は出力データ構造をロードするための１つ以上のデータ構造ロード命令を最初に含む必要があることが予想され得る。

【００３４】

データ構造メタデータレジスタが実装される方法に応じて、データ構造ロード命令は、異なる方法で実装することができる。一例では、データ構造ロード命令は、ロードされるデータ構造の目標入力又は出力データ構造識別子に対応する１つ以上のデータ構造メタデータレジスタのセットに保持されたアドレス表示メタデータから導出されたメモリアドレスからデータ構造をロードしてもよい。例えば、データ構造ロード命令は、データ構造がロードされるアドレスを識別するための目標アドレス識別情報を提供する１つ以上の汎用レジスタを指定することができ、ロードされたデータ構造をデータ構造記憶回路に書き込むように処理回路をトリガすることに加えて、データ構造ロード命令はまた、処理回路を制御して、データ構造ロード命令の命令又は別のプロパティによって参照される汎用レジスタにおいて指定された目標アドレス識別情報に基づいて、ロードされているデータ構造についての１つ以上のデータ構造メタデータレジスタの関連するセットを更新させることができる。このアプローチは、データ構造ロード命令によって指定された明示的なアドレス識別情報以外の形式を使用してメタデータレジスタ内のアドレス識別メタデータを表すための柔軟性を提供してもよい。例えば、実際のアドレス識別情報自体を保持する代わりに、データ構造メタデータレジスタに書き込まれるデータ構造ロード命令のプロパティ

は、データ構造ロード命令によってどの汎用レジスタが指定されたかの表示を含むことができるか、又はデータ構造ロード命令のアドレスを示すプログラムカウンタを含むことができ、これにより、データ構造メタデータレジスタのサイズを低減するために、よりコンパクトな形式のメタデータを記録することができてもよい。目標アドレス識別情報を指定するレジスタ又はデータ構造ロード命令のプログラムカウンタを記録することは、状態復元を実行するオペレーティングシステム又は他のソフトウェアが、所与のデータ構造に関連付けられたアドレスを復元することができる十分な情報を復元することを可能にするのに十分であり得る。例えば、汎用レジスタもまた、コンテキスト切り替え上で復元されるため、単に、レジスタがアドレス情報を含む参照を記録するだけで十分であり得る。また、データ構造ロード命令のプログラムカウンタが記録されている場合、これにより、ソフトウェア又は命令復号回路要素は、参照された命令を取得し、それによって参照される汎用レジスタを識別するためにそれを復号し、次いで、それらの汎用レジスタ内の復元された状態を使用して、目標アドレス識別情報を識別することができる。したがって、アドレスを直接識別するために必要なアドレス情報を明示的に示すことは、他のパラメータを介して間接的に識別することができるため、データ構造メタデータレジスタにとって必須ではない。

【００３５】

あるいは、データ構造のロードをトリガする特定の命令を提供するのではなく、他のアーキテクチャ実装では、アーキテクチャは、目標入力データ構造識別子を指定するロード準備命令を定義してもよい。ロード準備命令に応答して、命令デコーダは、処理回路を制御して、ロード準備命令によって暗黙的若しくは明示的に指定された目標アドレス識別情報、又はロード準備命令のプロパティのうちの１つに基づいて、目標入力データ構造識別子に対応する１つ以上のデータ構造メタデータレジスタのセットを更新させる。ここでも、このプロパティは、プログラムカウンタ、又は、ロード準備命令によって指定された汎用レジスタのリストであってもよい。データ構造処理命令が必要に応じてメモリからデータ構造をロードすることができ、必須のアーキテクチャ機能としてロードを実行するためのより早い命令の必要がないため、そのようなロード準備命令は、ロード機能を有するデータ構造処理命令をサポートするシステムにおいて特に有用であり得る。代わりに、ロード準備命令は、目標アドレス識別情報のあるセットを特定の入力又は出力データ構造のデータ構造識別子と関連付けるための命令として単純に解釈してもよい。これは、後続のデータ構造処理命令が関連するデータ構造をロードすることができるアドレスを設定することができる。

【００３６】

ロード準備命令が、目標入力データ構造識別子に関連するデータ構造を実際にロードすることはアーキテクチャレベルでは必須ではないが、任意選択的にマイクロアーキテクチャレベルで、システム設計者は、いずれの場合にも、ロード準備命令への応答として、目標入力データ構造識別子に関連付けられたデータ構造が、目標アドレス識別情報に基づいて決定されたメモリアドレスからデータ構造記憶回路に転送されることを要求するためのロード要求を生成する処理回路を提供することを選択してもよい。すなわち、ロード準備命令は、後続のデータ構造処理命令によってロードが行われるため、ロードを実行する義務がない場合であっても、ロードをより早く開始することができるため、ロード準備命令の時点でロード要求を生成することは、性能にとってより効率的であり得る。したがって、ロード準備命令は、プリロード命令として効果的に機能することができるが、それにもかかわらず、ロード準備命令の実行後であって後続のデータ構造処理命令の実行前に処理が中断された場合、データ構造処理命令は、依然として必要な入力データ構造を再ロードすることができる。これは、状態復元を行うオペレーティングシステム又は他のソフトウェアがデータ構造の復元に責任を負わなければならないことを回避するのに有用である。

【００３７】

目標出力データ構造識別子を指定する記憶準備命令をサポートすることができ、これに応答して、命令デコーダは、処理回路を制御して、記憶準備命令によって暗黙的又は明

示的に指定された目標アドレス識別情報、又は記憶準備命令の別のプロパティのいずれかに基づいて、目標出力データ構造識別子に対応する１つ以上のデータ構造メタデータレジスタのセットを更新してもよい。記憶準備命令は、アドレス識別メタデータを特定の出力データ構造識別子に付加する機能を有する。

【００３８】

上述の技術は、様々なタイプのデータ構造の範囲に適用することができる。いくつかの例では、データ構造は、データの２次元配列を含むことができ、２次元構造内に何らかの順序付き配列を有する複数のデータ要素が提供される。例えば、データ構造は、行列又は他の形態の２次元構造であってもよい。データ構造が行列である場合、データ構造処理命令は、２つの入力データ構造識別子を指定する行列乗算命令を含み、処理動作は、出力データ構造を生成するために、２つの入力データ構造識別子によって識別された２つの入力データ構造に対して実行される行列乗算演算を含んでもよい。行列乗算命令は、２つの入力行列を乗算した結果が出力行列の前の値に加算されて出力行列の新しい値を生成する行列乗累積命令とすることができる。また、入力行列に演算を適用する他の形態の行列操作命令も定義することができる。ＣＰＵ自体の中で行列演算を実行するためのネイティブプロセッサのサポートを提供することは、例えばニューラルネットワークを訓練するため、又はモデルを使用して予測を行うために、ニューラルネットワークモデルを定義する重みに入力変数を乗算するために、ニューラルネットワークが多数の行列乗算を必要とし得る機械学習などの、いくつかの分野に非常に有用であり得る。機械学習アルゴリズムでは、行列乗算は非常に一般的であり、そのような演算を高速化することは、機械学習ベースのソフトウェアの性能を大幅に改善することができる。

【００３９】

いくつかの実施例では、データ構造は、メモリシステム内のメモリアドレスの複数の不連続ブロックに対応する構造であってもよい。したがって、データ構造は、メモリアドレス空間内のメモリアドレスの単一の連続したブロックに記憶されるのではなく、メモリアドレス空間の別個の隣接していない領域にある複数のブロックにまたがってもよい。これは、例えばニューラルネットワークを表す行列の全体サイズが、実際には所与のデータ構造の命令セットアーキテクチャで定義された単位サイズよりも大きくなり得る行列の例に特に有用であり得る。例えば、データ構造は、４×４、８×８、１６×１６などの特定のサイズの行列に対応するものとして定義することができる（又は、異なる数の行及び列を有する矩形行列をサポートすることができる）が、ソフトウェアアルゴリズムがより大きな行列サイズで行列計算を適用する必要がある場合、これらは、命令セットアーキテクチャの命令においてサポートされる粒度で個々の演算に分解してもよい。これは、各行がメモリアドレス空間内の隣接するアドレスにあり、１つの行がアドレス空間内の前の行の要素から順次に続くように、又は代替として、各列がメモリ内の連続したアドレスのブロックを占有し、１つの列が前の列のアドレスから順次に続くように、より大きな行列が全体としてメモリに記憶される場合、これは、データ構造処理命令が、行列内の個々のタイルと考えることができるそのより大きな行列の個々の部分に対して演算を実行するとき、タイルの行又は列のいずれかがメモリアドレスの複数の不連続ブロックに対応することを意味する。アドレス空間の不連続部分にわたってストライプにされたそのようなタイルを処理するための命令サポートを提供することによって、これは、命令セットアーキテクチャでサポートされるタイルサイズよりも大きくても小さくてもよい行列のより任意のサイズに行列演算を適用できるようにすることをはるかに簡単にする。

【００４０】

メモリアドレスの複数の不連続ブロックに対応するデータ構造の場合、アドレス表示メタデータは、複数の情報を含むことができ、複数の形態をとってもよい。例えば、アドレス表示メタデータは、メモリアドレスの複数の不連続ブロックのうちの少なくとも１つの開始アドレスを示す開始アドレス情報と、メモリアドレスの複数の不連続ブロックの開始アドレス間の分離を示すオフセット情報と、メモリアドレスの各不連続ブロックのサイズを示す第１のサイズ情報と、データ構造を形成するメモリアドレスの不連続ブロックの

数を示す第２のサイズ情報とのうちの少なくとも１つを含んでもよい。これらのタイプの情報は、より大きい行列内の任意のタイルを識別するのに十分であり得、この場合、より大きい行列は、メモリアドレス空間内で行方向又は列方向のいずれかで記憶されると仮定され、タイルは、隣接していないアドレス空間内の個々のストライプに対応する。開始アドレス情報は、例えば、タイルの第１の行又は第１の列の開始アドレスを示すことができ、オフセット情報は、例えば、それぞれの行又は列の開始アドレス間のストライドを示すことができる。第１及び第２のサイズ情報は、より大きな行列全体が個々のタイル内の行又は列の数の正確な倍数ではない複数の行又は列を有する場合に対処するのに有用であり得、その結果、タイルがより大きな行列の端部に到達したとき、第１及び第２のサイズ情報は、行列の端部を超えて延びるアドレスをマスクするために使用され得る。第１のサイズ情報は、メモリアドレスの各不連続ブロックのサイズを示してもよく、第２のサイズ表示は、データ構造を形成するメモリアドレスの不連続ブロックの数を示してもよい。データ構造が行方向に記憶されているか列方向に記憶されているかに応じて、第１のサイズ情報の一方は、処理されるデータ構造内の列の数を効果的に示し、他方は、行の数を示すが、これはどちらでも可能である。第１及び第２のサイズ情報は、異なる方法で符号化することができる。一例では、第１又は第２のサイズ情報は、行又は列の数のカウントを示す数値として符号化することができ、例えば、第１のサイズ情報は、メモリアドレスの不連続ブロック内のデータ要素の数を示すことができ、第２のサイズ情報は、データ構造全体の不連続ブロックの数を示すことができる。あるいは、第１及び／又は第２のサイズ情報は、マスク、例えば、データ構造の一部を形成するメモリアドレスの不連続ブロック内のアクティブ要素を表す１（又は０）のビットと、データ構造の一部を形成しない不連続ブロック内の要素を示すマスク内の０（又は１）のビットとを有するビットマップ、として符号化することができる。これを使用して、任意のデータ構造処理命令が、個々のデータ構造処理命令によって処理されるべき全体的なより大きなデータ構造の一部を表すメモリアドレスの不連続ブロックの部分にのみ適用されることを保証することができる。

【００４１】

開始アドレス、オフセット、ならびに第１及び第２のサイズ情報に加えて、データ構造内の個々のデータ要素のデータ要素サイズを示す要素サイズ情報など、データ構造に関する他の情報を記録することも可能である。いくつかの形態のデータ構造処理命令は、例えば、異なるデータ要素サイズをサポートすることができてもよい。

【００４２】

あるいは、開始アドレス情報、オフセット情報、又は第１及び第２のサイズ情報を直接示すのではなく、他の例は、例えば、これらのパラメータのうちの少なくとも１つを保持する汎用レジスタを識別する１つ以上のレジスタ識別子、及び／又はこれらのパラメータのうちの１つ以上を保持するレジスタを指定する命令のアドレスを識別する命令アドレス表示を介して、これを間接的に識別するアドレス表示メタデータを提供してもよい。上述したように、レジスタ識別子又は命令アドレス表示（例えば、プログラムカウンタ）は、データ構造を処理回路の記憶回路に復元してデータ構造を見つけることができる場所を識別する必要があるソフトウェア又は回路にとって十分であり得る。

【００４３】

場合によっては、データ構造は、３を超えるオーダーの３次元又は多次元データ構造であり得るため、データ構造が２次元データ構造であることは必須ではない。一般に、データ構造は、データ要素の何らかの順序付き配列とすることができる。

【００４４】

あるいは、データ構造の別の例は、テーブル又は行列などの２次元以上の配列なしで、単にデータ要素の１次元ストリームであり得る。上述した技術は、かなりのサイズのデータストリームを提供するデータ構造にも有用であり得る。データ構造は、比較的大きなサイズを有する可能性があるため、これはコンテキスト切り替え待ち時間に課題を課す可能性があり、したがって、上述の技術は、これに対処するのに有用であり得る。６４バイトのデータ構造は、例えば、３２ビット要素の４×４行列に対応してもよい。他の例では

、出力データ構造は、少なくとも２５６バイト（３２ビット要素の８×８行列）又は１キロバイト（３２ビット要素の１６×１６行列）のサイズを有してもよい。他の例は、１６ビット又は６４ビットなどの異なる数のビットのデータ要素サイズを使用することができ、この場合もデータ構造サイズは変化し得ることが理解されよう。しかしながら、一般に、出力データ構造が少なくとも６４バイトのサイズを有する場合、これはコンテキスト切り替え待ち時間に大きな課題をもたらし始め、したがって、コンテキスト切り替え上で保存及び復元される必要があるデータの量を減らす方法としてアドレス表示メタデータを使用することが有用である。

【００４５】

なお、入力データ構造は、必ずしも出力データ構造と同じサイズである必要はないことに留意されたい。例えば、行列乗算演算を実行する場合、２つのＮビット値の乗算は２Ｎビットの結果を生成することができるため、入力データ構造よりも大きなサイズの出力データ構造が生成されることになり得る。あるいは、いくつかの実装形態は、例えば、入力と同じビット数に収まるように乗算の結果を切り詰めるか丸めることによって、入力と同じサイズの出力データ構造を生成することができる。

【００４６】

いくつかの例では、上述の任意選択の拡張で説明したように、メモリからの入力又は出力データ構造のロードを制御するためにデータ構造メタデータレジスタからのアドレス表示メタデータを任意選択的に使用すること以外に、データ構造処理命令に応答して処理回路によって実行されるデータ構造処理動作は、アドレス表示メタデータとは無関係であってもよい。すなわち、少なくとも１つの入力データ構造及び出力データ構造の以前の値を出力データ構造の新しい値にマッピングするために使用される関数は、アドレス表示メタデータとは無関係であり得る。

【００４７】

しかしながら、他の実施例では、処理回路は、データ構造処理命令に応答して、少なくとも１つの入力データ構造識別子又は出力構造識別子に対応する１つ以上のデータ構造メタデータレジスタの関連するセット内に保持されたアドレス表示メタデータに応じて、少なくとも１つの入力データ構造上で実行される処理動作を適合させてもよい。例えば、アドレス表示メタデータが、特定の次元におけるデータ構造のサイズを制限する第１のサイズ情報又は第２のサイズ情報を含む場合、これは、例えば、電力を節約するために、又は性能を向上させるために、入力データ構造のデータ要素に対して実行される乗算の数を低減するために使用され得る。例えば、サポートされる最大サイズのデータ構造が処理された場合にアクティブになる回路ロジックは、第１及び第２のサイズ情報に基づいてマスクされたいくつかの非アクティブ要素を有するデータ構造が処理されるときに、パワーゲーティングによって非アクティブにすることができる。また、処理動作が複数のステップで実行される場合、これらのステップのうちのいくつかは、第１及び第２のサイズ情報に基づいて必要とされないことがある。

【００４８】

図１は、メモリシステム８へのアクセスを共有するいくつかのマスタ装置４、６を備えるデータ処理システム２の一例を概略的に示す。マスタ装置４、６によって生じたメモリトランザクションは、トランザクションをメモリシステムにルーティングする相互接続１０に渡され、また、それぞれのマスタにおけるキャッシュデータ間のコヒーレンシを管理することもできる。マスタは、例えば、１つ以上の中央処理装置（ＣＰＵ）４、グラフィック処理ユニットなどの命令を実行することができる他のマスタ装置、又は命令実行能力を有さなくてもよいが、ネットワークへのアクセスの制御すること、又は信号処理若しくは行列乗算などの特定の処理機能を加速するためのハードウェアアクセラレータとして機能することなどの特定の専用機能を実行することができる他のデバイス６を備えることができる。

【００４９】

各ＣＰＵ４は、キャッシュ又はメモリからフェッチされたプログラム命令を復号する

ための命令デコーダ１２と、復号された命令に応答して命令デコーダ１２によって生成された制御信号に応答してデータ処理動作を実行するための処理回路１４とを有することができる。レジスタ１６は、実行される命令のオペランドを記憶し、処理回路１４によって生成された結果を記憶するために提供することができる。ＣＰＵは、メモリシステムからデータ、命令、又は制御情報（ページテーブルデータなど）をキャッシュするための１つ以上のキャッシュ１８を有してもよい。任意選択的に、ＣＰＵは、コンテキスト切り替え上の状態保存を処理するためのコンテキスト保存回路２０を有することができるが、この機能は、ＣＰＵ４上で実行されるソフトウェアによって実行することもできる。図１の２つのＣＰＵ４は、同等の構成要素で示されているが、実際には、全てのＣＰＵが同一であることは必須ではなく、一部のシステムは、処理性能を消費電力と引き換えにするために、より高性能のＣＰＵとより低性能のＣＰＵとを組み合わせるなど、異なる特性を有する非対称ＣＰＵを有してもよい。また、いくつかのシステムは、３つ以上のＣＰＵを有してもよい。

【００５０】

いくつかのデータ処理アプリケーションは、個々のデータ値の処理を含むことができ、データ値は、特定のアドレスからロードされ、レジスタ１６に書き込まれる結果を生成するために処理回路１４によって実行されるいくつかの命令によって処理され、結果がもはやレジスタ１６内に保持されなくなると、キャッシュ１８又はメモリ８に書き戻される。

【００５１】

しかしながら、他の処理アルゴリズムは、データ構造の要素を形成するいくつかの別個のデータ値を含むより大きなデータ構造に対して動作することができる。例えば、データ構造は、例えば圧縮されるデータのブロックなど、かなりのサイズの１次元範囲のデータのストリームとすることができる。あるいは、データ構造は、処理動作が行列／テーブル全体に適用される２つ以上の次元の行列又はテーブルとすることができる。

【００５２】

図２に示すように、そのようなデータ構造の一例は、行列（この例では２次元行列）であり得る。行列処理は、いくつかの信号処理アプリケーション又は機械学習アプリケーションなどの特定の処理アルゴリズムに有用であり得る。例えば、ニューラルネットワークなどの機械学習モデルは、モデルの特性を定義する重みを示す行列を使用して定義することができ、これは重みを入力で乗算することによって入力のセットに適用してもよい。したがって、ニューラルネットワークベースの処理及び他の機械学習アプリケーションは、有意な数の行列乗算演算を含み得る。そのような行列乗算を実行するためにハードウェアアクセラレータ６を提供することができるが、ハードウェアアクセラレータとの相互作用は、ハードウェアアクセラレータを構成するためにＣＰＵ４が特定のメモリ位置にデータを書き込み、その後、メモリからの読み出しを使用してハードウェアアクセラレータから結果を読み戻すことを必要とすることがあり、行列乗算演算をＣＰＵ自体で実行される他の演算の間に散在させる必要があるアプリケーションがある場合、これは性能が悪い可能性がある。また、行列演算のためにハードウェアアクセラレータを使用することは、複数のＣＰＵ４又はソフトウェアスレッドがハードウェアアクセラレータ６へのアクセスを共有する必要があるシステムにおいて困難を引き起こすことがある。したがって、ハードウェアアクセラレータ６を使用する必要はなく、命令デコーダ１２によってサポートされるネイティブ命令セットの一部として行列又は他のデータ構造の処理を実行することができる命令についてＣＰＵ４内でサポートを提供することが望ましくてもよい。

【００５３】

図２は、そのような行列を保持するために設けることができるローカル記憶構造３０の一例を示す。例えば、記憶構造３０は、ＣＰＵ内のいくつかのレジスタ１６であってもよく、又は非アーキテクチャ的に必須のレジスタ又はＣＰＵ４内のキャッシュなどのマイクロアーキテクチャ記憶要素として提供してもよい。また、ローカル記憶構造３０は、相互接続１０内にあってもよい。記憶構造３０は、何らかの所与のサイズの行列を記憶するのに十分な容量を有してもよい。この例では、行列はＮ×Ｎ正方行列であり、Ｎは１６で

あるが、他のサイズの行列もサポートできることが理解されよう。記憶構造３０は、データ要素が記憶構造３０に書き込まれた順序とは異なるデータ要素の順序で記憶構造から行列を読み出すことが可能である、十分な読み出しポート及び書き込みポートを有してもよい。例えば、行列は、列又は行のいずれかとして読み取ることができる。これは、行列を、連続したメモリアドレスのブロックを占有する列と、前の列のアドレスから順次に続く１つの列とのいずれかの異なる形式でメモリに記憶してもよく、同じ行内の要素は、アドレス空間内の不連続なアドレスにあり（このアプローチは列優先アドレス指定と呼ばれてもよい）、又は同じ行内の要素は、連続したメモリアドレスのブロックに記憶してもよく、１つの行は前の行のアドレスから順次に続いてもよく、この場合、列内の要素は、各行に割り当てられた部分内の不連続なアドレスにあり得る（このアプローチは、行優先アドレス指定と呼ばれる）という事実を説明する。異なるソフトウェアアルゴリズムは、行優先アプローチ及び列優先アプローチのうちの異なるものを使用することができるため、これらのアプローチのいずれかのハードウェアにおけるサポートを提供することが有用であり得る。行列の要素が一度にある行又は一度にある列のいずれかを読み出すことができるように追加の読み出しポートを提供することによって、これは、メモリ列優先に記憶された行列が行優先又はその逆に読み出されることができるように行列のオンザフライ転置を可能にし、これは、異なるタイプのソフトウェアとの互換性を大幅に改善することができ、列優先行列を行優先形式又はその逆に転置するためにベクトルの要素をシャッフルするための追加の命令を実行する必要性を回避する。

【００５４】

データ構造記憶回路３１には、図３の例に示すように、いくつかの行列又は行列の一部（タイル）を保持するのに十分な記憶空間が設けられてもよい。以下の説明では、メモリに記憶された行列のＮ×Ｎ部分を指す用語「タイル」が使用される。当然ながら、処理されるべきメモリ内の行列自体が次元Ｎ×Ｎである場合、タイルは、実際には行列全体に対応し得る。

【００５５】

データ構造記憶回路３１は、図２に示す記憶構造３０のいくつかのインスタンスを含んでもよい。例えば、記憶回路３１は、図３に示すように、一度に８つのＮ×Ｎタイルを保持するのに十分な容量を有してもよい。提供される８セットの記憶領域は、行列乗算又は他の演算用の入力タイルを保持するための４つの領域３０　Ａ０、Ａ１、Ｂ０、Ｂ１と、行列乗算演算の出力を保持するための４つの領域Ｃ０～Ｃ３とに分割され得る。

【００５６】

別のアプローチは、単一の行列乗算を一度に実行できるように（例えば、入力タイルＡ０、Ｂ０及び出力タイルＣ０を記憶するために）３つのタイル記憶領域３０のみを提供することであるが、このアプローチは、各行列乗算への入力が同じ２つの入力タイルレジスタを共有しなければならないため、各行列乗算演算に対してメモリから少なくとも２つのタイルロードを実行する必要がある。対照的に、ローカル記憶装置３１内に少なくとも８つのタイル記憶領域３０を提供することによって、これは、入力タイルが複数の乗算演算（例えば、Ｃ０＝Ａ０＊Ｂ０、Ｃ１＝Ａ０＊Ｂ１、Ｃ２＝Ａ１＊Ｂ０、Ｃ３＝Ａ１＊Ｂ１）にわたって共有されることが一般的であるため、４つの行列乗算演算にわたって４つのロードをメモリからロードして入力タイルＡ０、Ａ１、Ｂ０、Ｂ１をロードすることができることを意味し、これは、必要な乗算に対するロードの比を低減することによって性能を高めることができる。タイルは、比較的大量のデータをロードすることがあるため、タイルのロードは、比較的時間がかかることがあるので、これは性能にとって重要であり得る。例えば、タイルの各データ要素が３２ビットの値であり、Ｎ＝１６である場合、各タイルは１キロバイトのデータに対応し得る。

【００５７】

２つ以上の乗算演算を処理するためのローカル記憶領域３０を提供する同じアプローチは、入力タイル及び出力タイル用の記憶回路３１に追加の記憶容量を提供することによって、より多くの乗算に拡張することができるが、性能の向上と、ＣＰＵ４内の行列の数

を増やすための記憶回路を提供する更なるオーバヘッドとの間にバランスがあってもよく、したがって、図３に示すような８つの入力及び出力行列を有するアプローチは、性能と面積コストとの間のより好ましいバランスである可能性がある。それにもかかわらず、必要に応じて異なる数のタイルを記憶構造３１に保持することができる。

【００５８】

図３に示すタイル記憶領域３０の各々は、図４に示すように、より大きな行列５０内のタイル４０を表すデータを保持することができる。処理される行列５０全体は、ＣＰＵ４内の命令によってサポートされるＮ×Ｎのタイル４０のサイズよりも大きくてもよい。したがって、より大きな行列５０に対して実行される行列乗算は、行列の個々のタイル４０に対して実行される個々の命令に分解する必要があり得る。したがって、行列５０の所与のタイル４０がメモリ内に存在する位置を識別するために、いくつかのアドレス識別パラメータを定義することができる。より大きな行列５０のデータ要素は、メモリアドレス空間の連続したブロックに記憶してもよく、この例では、行列は行優先で記憶され、それにより行列の第１の行が連続したブロック内のメモリアドレスに記憶され、行列の左上要素が最下部メモリアドレスにあり、次にその先頭行の後続の要素が、メモリ内の連続して増加するメモリアドレスにある。第１の行の終わりに達すると、その後の次のアドレスは、行列の第２の行の左端の要素であるデータ要素に対応し、以下同様であり、その結果、行列のメモリアドレスのシーケンスが増加するにつれて、これは連続する行で読み出される要素に対応する。当然、行列は列優先で記憶することもでき、この場合、順序は行ごとではなく列ごとに進む。

【００５９】

したがって、行列の個々のタイル４０を識別するためにいくつかのパラメータを定義してもよく、タイルの開始アドレスを指すベースアドレス４２、より大きな行列５０の１つの行又は列のサイズを効果的に示し、タイル４０の１つの行（又はタイルが列優先で記憶されている場合は列）の開始アドレスと次の行（又は列）との間の差を表すストライドパラメータ４４を含む。また、水平サイズ及び垂直サイズ情報４６、４８は、タイル４０の水平及び垂直範囲を示すように定義してもよい。一般に、タイルは、通常、可能な最大サイズを占有することができ、例えば、Ｎ×Ｎで、完全なプロセッサ能力を使用することができる。しかしながら、図４の右下に示す例では、タイルがより大きな行列構造５０の境界に近づくと、タイルは、より大きな行列５０の縁部に部分的に重なってもよく、より大きな行列構造５０の外側にある部分４９は、処理されることが望ましくない場合がある。したがって、水平及び垂直サイズパラメータ４６、４８は、ＣＰＵ４のタイル記憶領域３０にロードされるデータの部分を制限する（又は行列処理命令で処理される部分を制限する）ために使用することができる。例えば、水平サイズ及び垂直サイズは、行若しくは列カウントとして、又はビットマップであるマスクとして符号化することができ、１のビットは行列の有効な要素を示し、０のビットは無効な要素を示し、又はその逆も可能であり得る。

【００６０】

図５は、ＣＰＵ４内のレジスタ１６のサブセットの例を示す。これは、提供され得るレジスタの全てを示さないことが理解されるであろう。レジスタ１６は、処理回路１４によって処理される命令のオペランド及び結果の一般的な記憶に使用される汎用レジスタ（ＧＰＲ）６０のセットを含む。特定の数、例えば３２又は６４の汎用レジスタを設けることができる。例えば、整数演算命令は、それらのオペランドについて汎用レジスタを参照し、それらの結果を汎用レジスタに書き戻すことができる。

【００６１】

ＣＰＵはまた、図３のようにいくつかの入出力データ構造Ａ０～Ｃ３を保持するための記憶領域３０を提供するデータ構造記憶装置３１を有することができる。このデータ構造記憶装置３１は、例えば、アーキテクチャ的に識別されたレジスタのセット、アーキテクチャ的に識別されていないレジスタのセット、又はキャッシュ１８の一部とすることができる。

【００６２】
ＣＰＵのレジスタ１６はまた、データ構造記憶装置３１内のデータ構造に関する情報を提供するいくつかのレジスタを含む。各データ構造は、データ構造識別子に関連付けられてもよく、例えば、表示Ａ０、Ａ１、Ｂ０、Ｂ１及びＣ０～Ｃ３は、それぞれのデータ構造識別子（第１／第２の入力及び出力識別子をそれぞれ区別するために、ここでは理解のためにＡ、Ｂ、Ｃ表記が提供されており、識別子では暗黙的であるが明示的に符号化されていないことがあることが理解されよう）に対応してもよい。したがって、レジスタは、対応するデータ構造がメモリに記憶されているアドレスを識別するために使用することができるアドレス表示メタデータを保持するために指定されるデータ構造メタデータレジスタ６２のセットを含むことができる。データ構造メタデータレジスタ６２は、ＧＰＲ６０とは別個のレジスタの専用セットであってもよいし、図５の点線に示すようにＧＰＲの指定されたサブセットであってもよい。これらの代替オプションについては、以下の図８及び図９に関してより詳細に説明する。
【００６３】
データ構造メタデータレジスタ６２は、各々が１つのデータ構造（タイル）に対応するレジスタのセットに分割することができる。所与のレジスタセットについて、これは単一のレジスタ又は２つ以上のレジスタのいずれかを含むことができる。一例では、所与のデータ構造について記録されたメタデータのセットは、図４について上述したパラメータ４２、４４、４６、４８を含むことができる。あるいは、データ構造メタデータは、（データ構造メタデータレジスタ６２がＧＰＲ６０から分離されている実施態様では）これらのパラメータを保持するＧＰＲ６０のセットを識別することによって、又はこれらのパラメータを保持するレジスタを指定する命令のプログラムカウンタアドレスを識別することなどによって、別の方法で記録することができる。行列以外のタイプのデータ構造については、異なるセットのアドレスパラメータをメタデータとして提供してもよいことが理解されよう。
【００６４】
任意選択で、レジスタ１６はまた、所与のデータ構造識別子に対して対応するデータ構造がデータ構造記憶装置３１において利用可能であるか、又はメモリからロードされる必要があるかを示すデータ構造可用性レジスタ６４などのデータ構造に関する情報を提供する追加のレジスタを含むこともできる。また、レジスタは、所与のデータ構造識別子について、データ構造記憶装置３１に記憶された対応するデータ構造がクリーン（メモリに記憶された対応するデータ構造と一致する）であるか、又はダーティ（メモリに記憶された基礎となるデータ構造とは異なる）であるかを示すデータ構造ダーティインジケータレジスタ６６を含むことができる。これらの追加のレジスタ６４、６６の両方は任意選択であり、場合によっては、それらをデータ構造メタデータレジスタ６２とは別個に提供するのではなく、代わりにレジスタ６４、６６内の情報をメタデータレジスタ自体の関連するセット内で符号化することができる。
【００６５】
コンテキスト保存動作を高速化するのを助けるために、アーキテクチャレベルで、所与のデータ構造識別子がクリーン又はダーティデータ構造に対応するかどうかをソフトウェアが判定できるようにする機能を提供することが有用であり得る。図６に示すように、場合によってはデータ構造ダーティインジケータレジスタ６６を不要にする可能性がある１つのアプローチは、出力データ構造Ｃ０～Ｃ３に使用される出力データ構造識別子空間７２と比較して、別個の入力データ構造識別子空間７０に識別子を有する入力データ構造Ａ０、Ａ１、Ｂ０、Ｂ１を定義することであり得る。例えば、一組の識別子７０を入力データ構造に使用し、別の組の識別子７２を出力データ構造に使用することができる。図６の例では、行列乗算演算への「Ａ」及び「Ｂ」入力は、別々の入力データ構造識別子空間７０－Ａ、７０－Ｂにおいて定義され、ハードウェアに必要なタイル読み出し／書き込みポートの数を減らすのに役立つ。しかしながら、別のアプローチは、共通の入力データ構造識別子空間内に識別子を有する全ての入力データ構造を割り当てることであり得る。

【００６６】
したがって、識別子が別個の識別子空間から定義される場合、これは、それぞれの識別子空間内の対応するデータ構造が同じ識別子を共有できないこと（又は、それぞれの識別子空間７０－Ａ、７０－Ｂ、７２が参照されている識別子を含むために使用されるフィールドなどの、命令符号化の他の態様から暗黙的であること）を意味する。入力及び出力データ構造に使用される識別子空間を分離することによって、これは、更新された値を有する出力データ構造を、入力データ構造識別子空間７０から入力データ構造識別子に関連する領域に書き込むことができないため、任意の入力データ構造識別子がクリーンなデータ構造に関連すると仮定できることを意味する。これは、入力データ構造識別子空間７０内の入力データ構造識別子に関連付けられたデータ構造の全てが常にクリーンであることを保証することができるので、コンテキスト保存を実行するソフトウェア又は回路要素に有用であり得、したがって、これは、データ構造記憶装置３１内の対応するデータ構造が単にメモリ内の同一のデータのコピーになることを意味し、したがってこれはコンテキスト切り替え上のメモリに保存し直す必要はない。代わりに、関連データがメモリ内に位置する場所に関する情報を提供するデータ構造メタデータレジスタ６２のみを保存すれば十分であってもよく、これは、コンテキスト復元のソフトウェアが、処理がその後再開する必要があるときに関連データ構造をメモリから再びロードするのに十分であり得るからである。

【００６７】
任意選択で、図６の点線に示すように、それぞれの出力データ構造識別子に関連するデータ構造がクリーンであるかダーティであるかを示すダーティインジケータを記憶するために、１つ以上のデータ構造ダーティレジスタ６６を設けることができる。これにより、コンテキスト保存を更に加速するために、任意のクリーンな出力データ構造のメモリへの保存を省略することを決定するコンテキスト保存手順が可能になる。

【００６８】
図７は、データ構造全体の粒度で処理動作を実行するためのデータ構造処理命令８０を含む一連の命令を処理する第一の例を示す。この例では、データ構造は上述のように行列のタイルであり、データ処理命令８０は、出力データ構造識別子Ｃｉ及び２つの入力データ構造Ａｊ、Ｂｋを指定し、結果$Ci' = Ci + A_j^* B_k$を生成するように処理回路１４を制御する行列乗累積命令である。この例では、ｉは、０～３の任意の値であってもよく、ｊ及びｋは、上述のようにデータ構造記憶装置３１に提供される記憶領域の数を考えると、０又は１のいずれかであってもよいが、他の例では、追加の記憶装置がある場合、識別子は必要に応じてより大きな値を有してもよい。この例では、Ａ、Ｂ、Ｃ表示は、命令符号化内のオペランド識別子の位置から想定され得るので暗黙的である。

【００６９】
図７の例では、データ構造メタデータレジスタ６２は、汎用レジスタ６０の固定サブセットである。各入力データ構造識別子について、ＧＰＲ６００の対応するサブセットは、そのデータ構造識別子のメタデータレジスタ６２を表すものとして割り当てられる。例えば、レジスタＸ０～Ｘ３は、入力データ構造Ａ０のメタデータレジスタのセットを表すことができ、同様にレジスタＸ４～Ｘ７、Ｘ８～Ｘ１１、及びＸ１２～Ｘ１５は、入力データ構造Ａ１、Ｂ０、Ｂ１のメタデータを提供することができる（メタデータを提供するために４つのレジスタを使用することは一例であり、他の例はメタデータをより少ないレジスタに圧縮することができることが理解されよう）。メタデータレジスタ６２の各セットは、１つ以上のレジスタを含むことができ、ソフトウェアが対応するデータ構造に対応するアドレスのセット、例えば、上述のベースアドレス４２、ストライド４４、ならびに水平及び垂直サイズパラメータ４６、４８を決定することを可能にするための情報を提供することができる。出力データ構造に対応するメタデータレジスタを提供することは必須ではないが、これも提供することができる。

【００７０】
図７に示すように、命令デコーダ１２は、入力データ構造識別子を指定し、指定され

た入力データ構造識別子に関連付けられたデータ構造記憶装置３１の領域に、指定された入力データ構造レジスタに関連付けられたＧＰＲの対応する固定サブセットから導出されたアドレスのセットからデータ構造をロードするように、処理回路１４を制御するデータ構造ロード命令８２をサポートすることができる。例えば、図７に示される第１の負荷命令に関しては、入力データ構造識別子はＡ０であり、したがって、メモリ内の対応するデータ構造のアドレスを識別するためのアドレス情報が、入力データ構造Ａ０のアドレス情報を保持すると指定されるＧＰＲ　Ｘ０～Ｘ３の固定サブセットから取得されるべきであることを暗示してもよい。命令シーケンスは、ＧＰＲのそれぞれのサブセット内の情報に基づいてそれぞれの入力データ構造Ａ０、Ａ１、Ｂ０、Ｂ１をロードするためのロード命令のいくつかを含み得る。任意選択的に、累積演算に使用される出力データ構造の以前の値がデータ構造記憶装置３１内にまだ保持されていない場合、出力データ構造にロードするための更なるロードが存在する可能性もある。出力データ構造に永続的に割り当てられたメタデータレジスタがない場合、これらの出力データ構造ロード命令は、対応するデータ構造のアドレス情報を定義する汎用レジスタを明示的に参照する必要があり得る。あるいは、データ構造記憶装置３１内の出力データ構造Ｃ０～Ｃ３のうちの１つの内容が常に関連する値を含み、次いで出力データ構造を０に初期化することによって、後続の命令が単に出力データ構造に蓄積され得ると単純に仮定することができるので、出力データ構造のためのそのようなロード命令は必要とされないことがある。したがって、Ｃ０～Ｃ３のロードは任意である。

【００７１】

必要な入力のいずれかをロードした後、一連の行列乗累積（ＭＭＬＡ）演算は、それぞれの出力Ｃ０～Ｃ３を計算する。４つの入力Ａ０～Ｂ１が４つの乗累積演算の間で共有されるので、この例では乗累積演算ごとに１つのロードがあることに留意されたい。それぞれのロード及び乗累積命令８２、８０は、アドレスパラメータを変更して行列構造５０全体の他のタイルを指すようにするために、メタデータレジスタｘ０～ｘ１５の関連するセットを更新する汎用レジスタ移動命令又は算術若しくはロード命令を散在させて、複数回繰り返してもよい。あるいは、行列ロード命令８２は、メタデータレジスタ６２によって指し示される現在のアドレスにおいてメモリからデータ構造をロードすることに加えて、処理回路がまた、例えばベースアドレスをインクリメントすることによって次のタイルを指し示すようにメタデータレジスタを暗黙的に更新するように定義してもよい。いくつかの例では、例えば、水平／垂直サイズ情報４６、４８によって示される無効な要素をマスキングすることによって、対応するアドレスメタデータのセットに基づいて、行列乗算を修正することができる。最終的に、実行されるべき行列乗累積演算がなくなると、それらのデータ構造が、メタデータレジスタに示されるメタデータと同様の形式で記憶されるべきアドレスを識別するために、汎用レジスタを使用することができるデータ構造記憶命令８４を使用して、行列乗算命令８０の最新のセットの結果をメモリに記憶し直すことができる。

【００７２】

所与のデータ構造の処理が進行中である間、そのデータ構造のアドレスメタデータを定義するメタデータレジスタ６２として汎用レジスタ６０のサブセットを使用することは、データ構造が依然として必要とされている間に、汎用レジスタ６０のそのサブセットに他の値を書き込むことを回避するためにソフトウェアが必要とされることを意味する。これは、割り込みが発生し、処理が異なる実行コンテキストに切り替わる場合に、データ構造記憶装置３１に保持された所与のデータ構造のアドレス表示メタデータが依然として存在し、汎用レジスタ６０のコンテキスト保存の一部としてメモリに保存することができることを保証し、これは、コンテキスト切り替え待ち時間を低減する対応するデータ構造自体を保存する必要がないことを意味する。

【００７３】

図８は、図７のシーケンスと同様のコードシーケンスの例を示しているが、ここでは、データ構造のためのアドレス表示メタデータのそれぞれのセットを提供するデータ構造

メタデータレジスタ６２は、汎用レジスタ６０とは別個である。データ構造処理命令８０及び記憶命令８４は、図７と同じ役割を有することができる。

【００７４】

しかしながら、ロード命令８２については、アドレス情報を提供する暗黙的に識別された汎用レジスタが存在しないため、ロード命令は、アドレス情報のそれぞれの部分４２、４４、４６、４８を提供する汎用レジスタへの明示的な参照のセット８６を含むことができ、次いで、これらのロード命令８２は、命令デコーダ１２をトリガして、処理回路１４を制御して、指定された汎用レジスタ内の情報によって識別されたアドレスのセットからデータ構造をメモリからロードさせ、ロードされたデータ構造を、ロード命令８２によって指定された入力データ構造識別子Ａ０～Ｂ１に関連するデータ構造記憶装置３１の関連領域に記憶させるようにすることができる。データ構造をロードすることに加えて、ロード命令はまた、指定された入力データ構造識別子についての１つ以上のデータ構造メタデータレジスタ６２の関連するセットにアドレス表示メタデータを書き込むように処理回路をトリガしてもよい。アドレス表示メタデータは、多数の形態をとることができる。これは、上述したように、単にベースアドレス４２、ストライド４４ならびに水平及び垂直サイズ４６、４８を明示的に符号化する汎用レジスタのアクセスされたセットからの値であり得る。あるいは、メタデータは、アドレス情報を含むレジスタを識別するレジスタ指定子、例えば図８の例における第１のロードのＸａ～Ｘｄであってもよい。このアプローチが使用される場合、ソフトウェアは、それらのレジスタがアドレス情報を保持し続け、ロード命令の実行とデータ構造記憶装置内のデータ構造に依存する最終命令との間に更新されないことを保証する必要がある。しかしながら、このアプローチは、どの特定の汎用レジスタがアドレス情報を指定するかの変更を可能にすることによって、図７の例と比較して追加の柔軟性を提供することができる。あるいは、アドレス表示メタデータは、データ構造が依然として必要である間に、やはり参照レジスタＸａ～Ｘｄなどの内容が変化しないことが保証されるならば、関連するロード命令自体のアドレスを示すプログラムカウンタアドレス８８の表示とすることができる。アドレス情報４２、４４、４６、４８自体がデータ構造メタデータレジスタ６２に書き込まれる例では、その情報を指定するために使用されるソースレジスタ８６での後続の更新を制限する必要はなく、この場合、ソフトウェアは、対応する情報がデータ構造メタデータレジスタ６２に保持されるので、アドレス情報４２～４８を定義するために使用されるＧＰＲ６０の内容を変更することができる。

【００７５】

そうでなければ、図８の例は図７と同様であり、ここでも、アドレス表示メタデータが発見される場所、又はアドレス表示メタデータを直接示す情報を記録することによって、コンテキスト復元のソフトウェアは、対応するデータ構造がメモリ内にある場所を判定することができ、これは、コンテキスト保存アルゴリズムの一部としてクリーンなデータ構造を保存する必要がないことを意味する。

【００７６】

したがって、図９は、コンテキスト保存を制御する方法を示すフロー図である。これは、ＣＰＵ４内のハードウェアに設けられたコンテキスト保存回路２０によって実行することができ、又はオペレーティングシステム、ハイパーバイザ若しくは他の監視プロセスによってソフトウェアで行うことができる。状態保存方法は、状態保存トリガイベント、例えばハードウェア又はソフトウェアによってトリガされる例外又は割り込みに応答して実行される。ステップ１００では、コンテキスト状態の保存は、メモリ内のデータ構造に対する送出スレッドの一般的なコンテキストの保存を含む。例えば、一般的なコンテキストは、現在の実行時点を表す命令アドレスを示すプログラムカウンタ、汎用レジスタ６０の内容、及び上記のデータ構造の処理に直接関連しないことがある保持される必要がある任意の他のプロセッサ状態を含むことができる。この状態の保存は、その監視プロセスのアドレス空間内の、例えば、オペレーティングシステム又はハイパーバイザのアドレス空間内の、監視プロセスによって維持されるデータ構造に対して実行され得る。例えば、スタックデータ構造を使用して、所与のスレッドに保存された状態ブロックを記憶すること

ができる。

【００７７】

ステップ１０２において、データ構造記憶装置３１内の任意のクリーンなデータ構造について、状態保存は、メタデータレジスタ６２の対応するセットを保存するが、データ構造記憶装置３１に記憶されているデータ構造自体は保存しない。これらのクリーンなデータ構造は、任意の入力データ構造、ならびに任意選択的に、ダーティインジケータ６６が出力データ構造がクリーンであることを示す任意の出力データ構造を含むことができる。データ構造メタデータレジスタ６０がＧＰＲ６０のサブセットである場合、ステップ１０２は、ＧＰＲ６０がメモリに保存されたときにステップ１００の一部として既に暗黙的に実行されている可能性があることに留意されたい。しかしながら、データ構造メタデータレジスタ６２がＧＰＲ６０から分離されている例では、ステップ１０２を実行してもよい。

【００７８】

ステップ１０４において、任意のダーティデータ構造又はそれがクリーンであるかダーティであるかが未知であるデータ構造について、データ構造自体及びメタデータレジスタ６２の対応するセットの両方を保存することができる。

【００７９】

したがって、データ構造が取得されたメモリ内の場所に関するアドレス情報を保持し続けるいくつかのメタデータレジスタ６２が存在することを保証することによって、これは、ステップ１０４におけるデータ構造全体の保存をそれらのダーティ構造のみに制限することを可能にすることができ、それらはメモリ内のデータの単なるコピーであるため、クリーンなデータ構造を保存する必要はない。

【００８０】

図１０は、使用することができる別の例を示す。この例では、データ構造処理動作自体を実行することに加えて、データ構造処理命令８０はまた、入力又は出力値がデータ構造記憶装置３１内で既に利用可能でない場合、その命令に必要な任意の入力又は出力データ構造のロードをトリガしてもよい。これは、上述のロード命令８２を、ロード準備命令１２０及び記憶準備命令１２２と置き換えることができることを意味し、これらはメモリから対応するデータ構造をロードする必要はないが、指定されたデータ構造識別子Ａ０～Ｃ３にアドレス情報のセットを割り当てるために使用され得る。これらの命令１２０、１２２は、上述したタイプのアドレス情報４２～４８を定義する汎用レジスタ１２４のセットを指定することができ、命令デコーダ１２に、処理回路１４を制御して、指定されたデータ構造識別子Ａ０～Ｃ３に関連付けられたメタデータレジスタ６２のセットにアドレス情報を書き込ませるようにすることができる。ロード準備命令１２０を、アドレス情報を入力データ構造識別子Ａ０、Ａ１、Ｂ０、Ｂ１に割り当てるために使用してもよく、記憶準備命令１２２は、アドレス情報を出力データ構造識別子Ｃ０～Ｃ３に割り当ててもよい。場合によっては、ロード準備命令はまた、データ構造がまだ利用可能でない場合にデータ構造をデータ構造記憶装置３１にプリロードすることもできるが、これはアーキテクチャ的に必要な機能ではないことがある。

【００８１】

したがって、データ構造処理命令８０に達すると、処理回路は、例えばデータ構造可用性レジスタ６４の可用性情報に基づいて、任意の入力の可用性をチェックし、データ構造記憶装置３１でまだ利用できない命令に必要な任意の入力をロードすることができる。また、処理回路は、入力データ構造に対して関連するデータ処理動作を実行して出力データ構造を生成する。

【００８２】

命令セットアーキテクチャ内のＭＭＬＡ命令８０を定義して入力のロードも必要とすることにより、これは、必要に応じて、データ処理動作８０自体が利用できないデータ構造にロードするため、コンテキスト復元機能が特定のデータ構造を復元する必要がないことを意味する。したがって、例えば、図１０の例でＣ０出力を生成する最初のデータ構造

処理命令８０の後に割り込みが発生し、現在のスレッドから離れたコンテキスト切り替えが実行された場合、この時点で、任意のダーティデータ構造Ｃ０がメモリに保存され得、以前にメタデータレジスタ６２にコピーされたアドレス表示メタデータも状態保存の一部として保存されるが、任意のクリーンなデータ構造Ａ０～Ａ１、Ｂ０～Ｂ１又はＣ１～Ｃ３を保存することは必須ではない。続いてスレッドに戻るとき、コンテキスト復元が実行され、メタデータレジスタ６２が復元されると、処理が再開するときに各後続のデータ構造処理命令８０がまだ行われていなければその入力にロードするため、レジスタに他のデータ構造を復元する必要はない。これは、データ構造が、監視プロセスのアドレス空間内で使用されなければならないのではなく、それらを使用するスレッドのアドレス空間内に残ることを意味するので、セキュリティを向上させることができる。また、これは、コンテキスト復元に続くスレッドの再開後に実行される任意の命令に、その後必要とされない任意のデータ構造に対してコンテキスト復元を実行する必要がないことを意味する。

【００８３】

アーキテクチャ的に必要ではないが、ロード準備命令１２０は、まだ行われていない場合、それらの対応するデータ構造をデータ構造記憶装置３１に任意選択的にプリロードすることができ、これは、ハードウェアがロードを開始する前に対応するデータ構造処理命令８０まで待機する場合よりも早く、これらの構造のロードを開始することによって性能を改善するのに役立つことができる。したがって、可用性レジスタ６４は、データ構造処理命令８０に達したときに、ハードウェアが任意のデータ構造を再ロードする必要があるかどうかを判定できるように、ロード準備命令１２０と対応するデータ処理命令との間に割り込みがあったかどうかを追跡するのに有用であり得る。

【００８４】

図１１は、コンテキスト状態復元プロセスを示すフロー図であり、コンテキスト状態復元プロセスは、例えば、基礎となる処理スレッドのコンテキスト復元の管理を担当するソフトウェア内のオペレーティングシステム又はハイパーバイザによって実行され得る。ステップ１５０において、ソフトウェアは、プログラムカウンタ及び汎用レジスタなどの到来スレッドの一般的なコンテキストを復元する。汎用レジスタ復元の一部としてまだ行われていない場合、ステップ１５２において、コンテキスト復元プロセスは、データ構造メタデータレジスタ６２を復元し、その結果、後続の命令は、メモリ内のどこからデータ構造をロードする必要があるかを識別することができる。ステップ１５４において、ソフトウェアは、後続のデータ構造処理命令８０によってオンデマンドでロードすることができるため、データ構造自体を復元する必要はない。

【００８５】

上述の例は行列に関し、説明されたデータ構造処理動作は行列乗算であるが、コンテキスト保存及び復元オーバヘッドを低減するための同様の技術を他のデータ構造タイプに適用できることが理解されよう。

【００８６】

図１２は、使用可能なシミュレータの実装形態を示している。先に説明した実施形態は、当該技術をサポートする特定の処理ハードウェアを動作するための装置及び方法の観点から本発明を実装するものであるが、コンピュータプログラムを使用して実装される本明細書に記載の実施形態に従った命令実行環境を提供することも可能である。このようなコンピュータプログラムは、ハードウェアアーキテクチャのソフトウェアベースの実装形態を提供する限りにおいて、シミュレータと呼ばれることが多い。シミュレータコンピュータプログラムの種類には、エミュレータ、仮想マシン、モデル、及び動的バイナリトランスレータを含むバイナリトランスレータが含まれる。典型的には、シミュレータの実装形態は、シミュレータプログラム７１０をサポートする、任意にホストオペレーティングシステム７２０を実行するホストプロセッサ７３０上で実行してもよい。いくつかの構成では、ハードウェアと提供される命令実行環境との間に複数のシミュレーション層が存在してもよく、及び／又は、同じホストプロセッサ上で提供される複数の異なる命令実行環境が存在してもよい。歴史的に、合理的な速度で実行するシミュレータの実装形態を提供

するためには、強力なプロセッサが必要とされてきたが、互換性又は再利用の理由から別のプロセッサにネイティブなコードを実行したい場合など、特定の状況では、そのようなアプローチが正当化される場合がある。例えば、シミュレータの実装形態では、ホストプロセッサのハードウェアではサポートされていない追加機能を備えた命令実行環境を提供すること、又は異なるハードウェアアーキテクチャに典型的に関連する命令実行環境を提供することができる。シミュレーションの概要は、「Ｓｏｍｅ　Ｅｆｆｉｃｉｅｎｔ　Ａｒｃｈｉｔｅｃｔｕｒｅ　Ｓｉｍｕｌａｔｉｏｎ　Ｔｅｃｈｎｉｑｕｅｓ」、Ｒｏｂｅｒｔ　Ｂｅｄｉｃｈｅｋ、１９９０年冬ＵＳＥＮＩＸ　Ｃｏｎｆｅｒｅｎｃｅ、５３～６３頁に記載されている。

【００８７】

これまで、特定のハードウェア構成又は機能を参照して実施形態を説明してきたが、シミュレーションされた実施形態では、同等の機能を適切なソフトウェア構成又は機能によって提供することができる。例えば、特定の回路は、シミュレーションされた実施形態において、コンピュータプログラムロジックとして実装してもよい。同様に、レジスタ又はキャッシュなどのメモリハードウェアは、ソフトウェアのデータ構造としてシミュレーションされた実施形態で実装することができる。先に説明した実施形態で参照されるハードウェア要素の１つ又は複数がホストハードウェア（例えば、ホストプロセッサ７３０）上に存在する構成では、いくつかのシミュレートされた実施形態は、適切な場合にはホストハードウェアを利用してもよい。

【００８８】

シミュレータプログラム７１０は、コンピュータ読み取り可能な記憶媒体（非一時的媒体であってもよい）に格納してもよく、シミュレータプログラム７１０によってモデル化されているハードウェアアーキテクチャのアプリケーションプログラムインタフェースと同じであるプログラムインタフェース（命令実行環境）を目標コード７００（アプリケーション、オペレーティングシステム、ハイパーバイザを含んでもよい）に提供する。したがって、上述したデータ構造処理命令を含む目標コード７００のプログラム命令は、シミュレータプログラム７１０を使用する命令実行環境内から実行してもよく、上述した装置２のハードウェア機能を実際には有していないホストコンピュータ７３０がこれらの機能をエミュレートすることができる。

【００８９】

シミュレータプログラム７１０は、目標コード７００の命令を復号して、ホストデータ処理装置７３０がサポートするネイティブ命令を使用して定義された同等の機能にマッピングするための命令復号プログラムロジック７１２を含んでもよい。例えば、命令復号プログラムロジック７１２は、目標コード７００内の目標命令の符号化ビットを検査し、特定の命令で見つかったビットに応じて、目標命令のアーキテクチャ機能をエミュレートするためにネイティブ命令セットで定義された命令シーケンスを選択するｉｆ／ｔｈｅｎ／ｅｌｓｅ文を含むことができる。

【００９０】

また、シミュレータプログラム７１０は、ホストデータ処理装置７３０のメモリ内に定義された記憶構造を管理し得るレジスタエミュレートプログラムロジック７１４を含んでもよく、この記憶構造は、目標コード７００に関連付けられた命令セットアーキテクチャによって要求されるアーキテクチャレジスタをエミュレートする。例えば、目標コード７００の目標命令内のレジスタ参照を使用して識別された読み出し／書き込み動作を、ホストデータ処理装置７３０のメモリ内のデータを読み出し／書き込みするためのメモリアクセス命令にマッピングすることができる。レジスタエミュレートプログラムロジック７１４を使用してエミュレートされたレジスタは、上述のデータ構造メタデータレジスタ６２を含んでもよい。

【００９１】

本出願において、「～ように構成される（configured to）」という用語は、装置の要素が、定義された動作を実行することができる構成を有することを意味するために使用

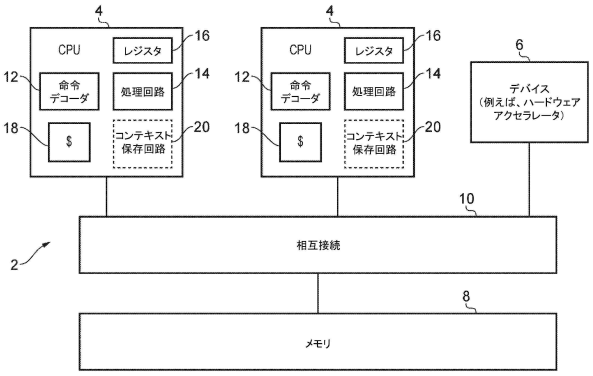
される。このコンテキストにおいて、「構成」は、ハードウェア又はソフトウェアの相互接続の構成又は方法を意味する。例えば、装置は、定義された動作を提供する専用ハードウェアを有してもよく、又はプロセッサ若しくは他の処理デバイスは、機能を実行するようにプログラムしてもよい。「～ように構成される」は、定義された動作を提供するために、装置要素を任意の方法で変更する必要を意味しない。

【００９２】

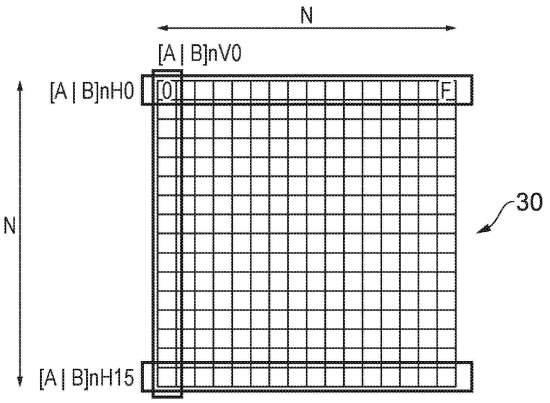
本発明の例示的な実施形態が添付の図面を参照して本明細書で詳細に説明されてきたが、本発明はそれらの正確な実施形態に限定されず、添付の特許請求の範囲によって定義される本発明の範囲から逸脱することなく、当業者によって様々な変更及び修正を行うことができることを理解されたい。

【図面】

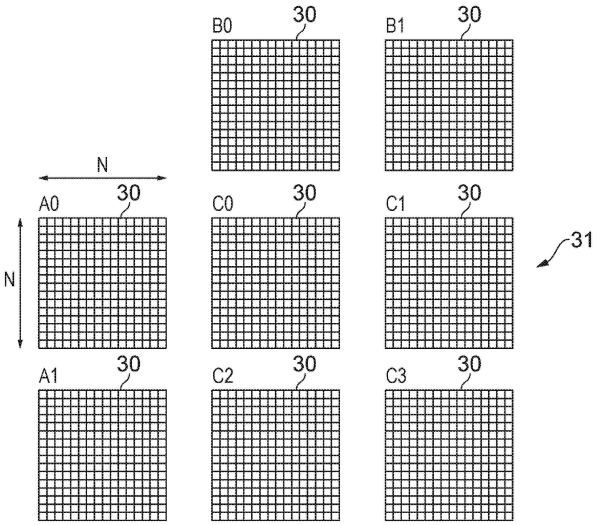
【図１】


4
CPU
レジスタ
16
12
命令デコーダ
処理回路
14
18
$
コンテキスト保存回路
20
4
CPU
レジスタ
16
12
命令デコーダ
処理回路
14
18
$
コンテキスト保存回路
20
6
デバイス（例えば、ハードウェアアクセラレータ）
10
2
相互接続
8
メモリ

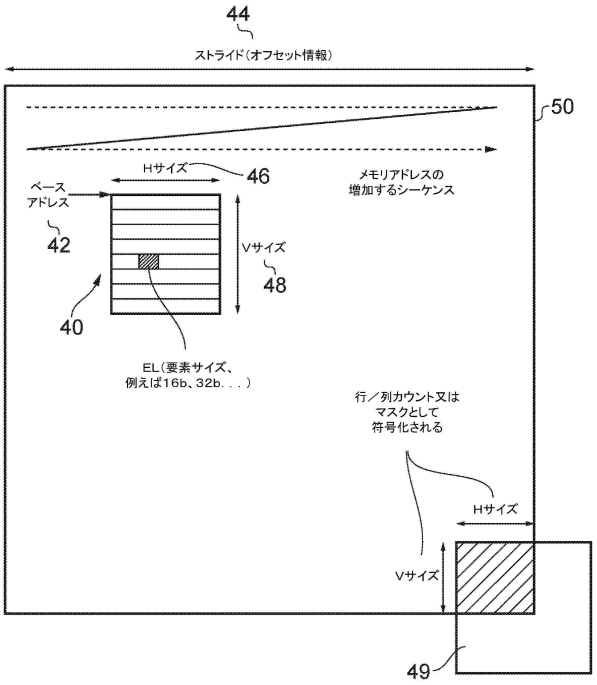
【図２】


N
[A | B]nV0
[A | B]nH0
0
F
N
30
[A | B]nH15

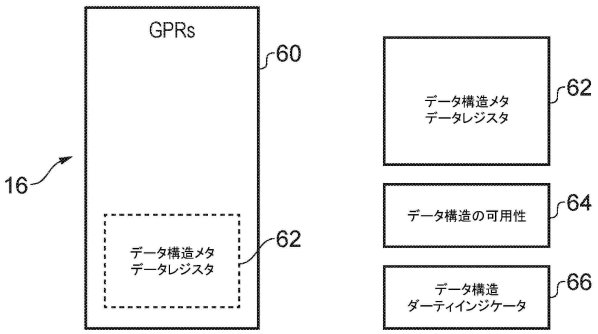
【図３】


B0
30
B1
30
N
A0
30
C0
30
C1
30
31
N
A1
30
C2
30
C3
30

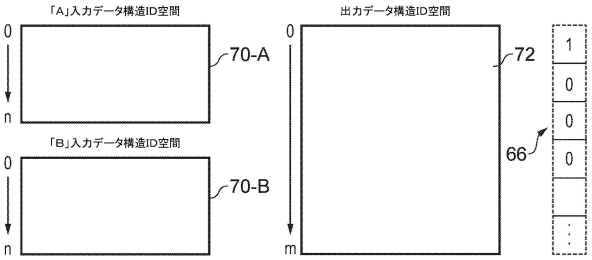
【図４】

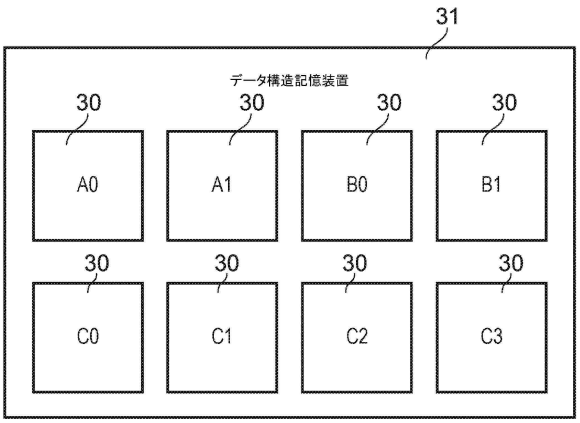

44
ストライド（オフセット情報）
50
Hサイズ
46
メモリアドレスの増加するシーケンス
ベースアドレス
42
Vサイズ
48
40
EL（要素サイズ、例えば16b、32b. . . ）
行／列カウント又はマスクとして符号化される
Hサイズ
Vサイズ
49

【図５】

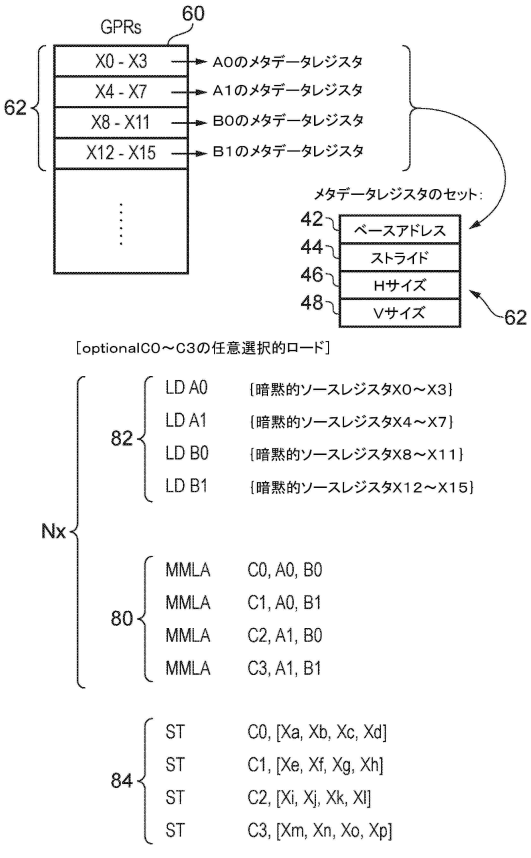

GPRs
60
16
62
データ構造メタ
データレジスタ
62
データ構造メタ
データレジスタ
64
データ構造の可用性
66
データ構造
ダーティインジケータ

【図６】

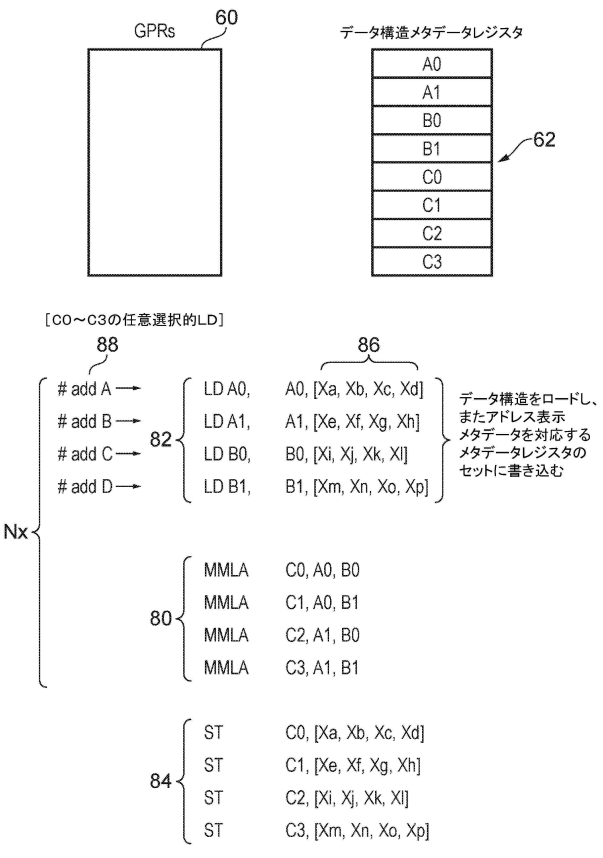

「A」入力データ構造ID空間
0
n
70-A
「B」入力データ構造ID空間
0
n
70-B
出力データ構造ID空間
0
m
72
1
0
0
0
66


31
データ構造記憶装置
30
A0
A1
B0
B1
C0
C1
C2
C3

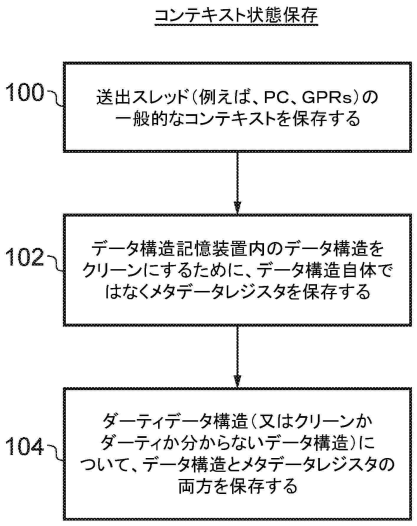
【図７】


GPRs
60
62
X0 - X3
X4 - X7
X8 - X11
X12 - X15
A0のメタデータレジスタ
A1のメタデータレジスタ
B0のメタデータレジスタ
B1のメタデータレジスタ
メタデータレジスタのセット:
42
ベースアドレス
44
ストライド
46
Hサイズ
48
Vサイズ
62
[optionalC0～C3の任意選択的ロード]
Nx
82
LD A0 {暗黙的ソースレジスタX0～X3}
LD A1 {暗黙的ソースレジスタX4～X7}
LD B0 {暗黙的ソースレジスタX8～X11}
LD B1 {暗黙的ソースレジスタX12～X15}
80
MMLA C0, A0, B0
MMLA C1, A0, B1
MMLA C2, A1, B0
MMLA C3, A1, B1
84
ST C0, [Xa, Xb, Xc, Xd]
ST C1, [Xe, Xf, Xg, Xh]
ST C2, [Xi, Xj, Xk, Xl]
ST C3, [Xm, Xn, Xo, Xp]

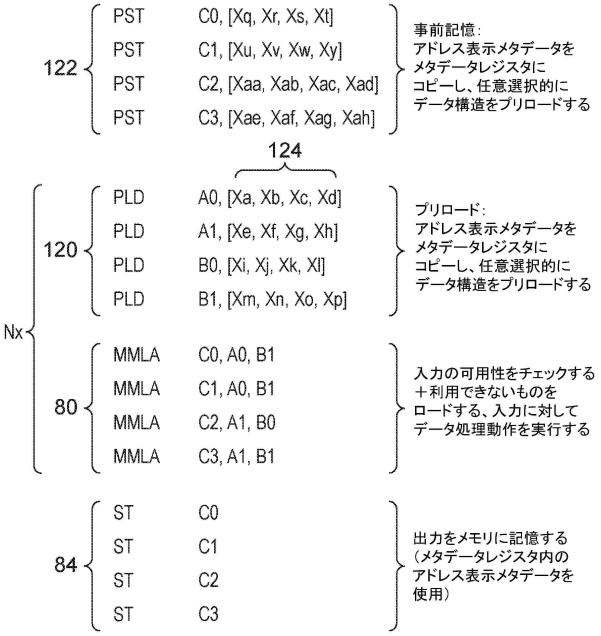
【図８】


GPRs
60
データ構造メタデータレジスタ
A0
A1
B0
B1
C0
C1
C2
C3
62
[C0～C3の任意選択的LD]
88
86
Nx
add A
add B
add C
add D
82
LD A0, A0, [Xa, Xb, Xc, Xd]
LD A1, A1, [Xe, Xf, Xg, Xh]
LD B0, B0, [Xi, Xj, Xk, Xl]
LD B1, B1, [Xm, Xn, Xo, Xp]
データ構造をロードし、
またアドレス表示
メタデータを対応する
メタデータレジスタの
セットに書き込む
80
MMLA C0, A0, B0
MMLA C1, A0, B1
MMLA C2, A1, B0
MMLA C3, A1, B1
84
ST C0, [Xa, Xb, Xc, Xd]
ST C1, [Xe, Xf, Xg, Xh]
ST C2, [Xi, Xj, Xk, Xl]
ST C3, [Xm, Xn, Xo, Xp]

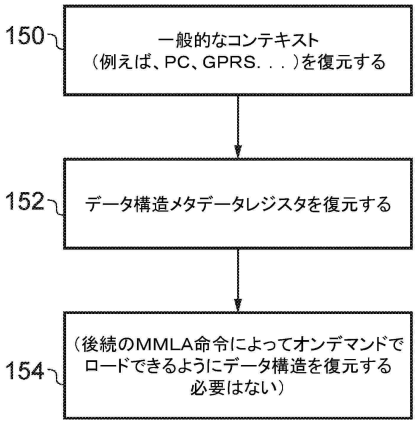
【図９】



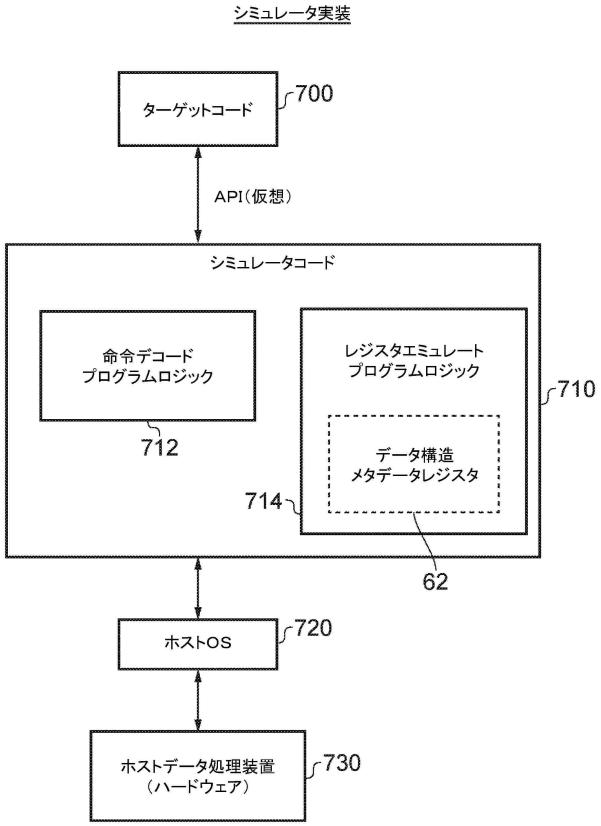
【図１０】



【図１１】



【図１２】



フロントページの続き

ド １１０， アームリミテッド宛
(72)発明者　グリゼンスウェイト、リチャード ロイ
イギリス国 ケンブリッジシャー ＣＢ１ ９ＮＪ ケンブリッジ，フルボーン ロード １１０， アームリミテッド宛
(72)発明者　エヴァンス、マシュー ルシアン
イギリス国 ケンブリッジシャー ＣＢ１ ９ＮＪ ケンブリッジ，フルボーン ロード １１０， アームリミテッド宛

審査官　坂庭 剛史
(56)参考文献　米国特許出願公開第２０１９／００４２４４８（ＵＳ，Ａ１）
米国特許出願公開第２０１８／０００４５１０（ＵＳ，Ａ１）
米国特許出願公開第２０１７／０３３７１５６（ＵＳ，Ａ１）
米国特許出願公開第２００５／００５５５４３（ＵＳ，Ａ１）
(58)調査した分野　(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　９／３１２
Ｇ０６Ｆ　９／３４
Ｇ０６Ｆ　９／３４５