(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2020-109605
(P2020-109605A)
(43)公開日　令和2年7月16日(2020.7.16)

(51)Int.Cl.	F I	テーマコード（参考）
G06F 9/46 (2006.01)	G06F 9/46 410	

審査請求 有　請求項の数 17　ＯＬ　外国語出願　（全 37 頁）

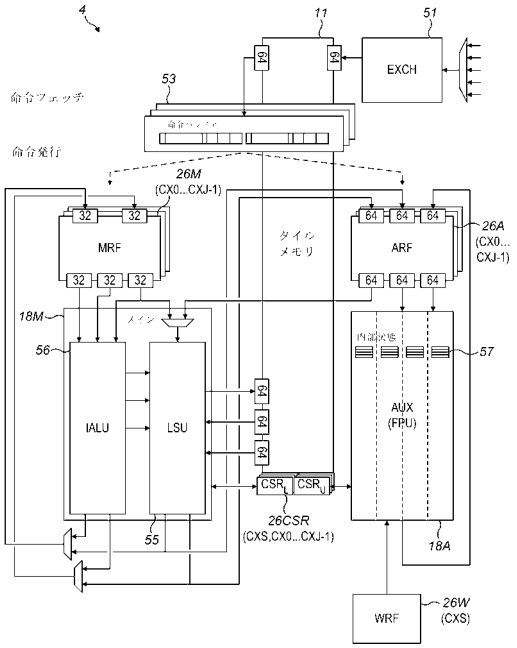
(21)出願番号　特願2019-113329 (P2019-113329)
(22)出願日　令和1年6月19日 (2019.6.19)
(31)優先権主張番号　1821301.7
(32)優先日　平成30年12月31日 (2018.12.31)
(33)優先権主張国・地域又は機関
英国 (GB)

(71)出願人　518371892
グラフコアー　リミテッド
Ｇｒａｐｈｃｏｒｅ　Ｌｉｍｉｔｅｄ
イギリス国，ビーエス１　２ピーエイチ，ブリストル，ワイン　ストリート　１１－１９
１１－１９ Ｗｉｎｅ　Ｓｔｒｅｅｔ，Ｂｒｉｓｔｏｌ，ＢＳ１　２ＰＨ，Ｕｎｉｔｅｄ　Ｋｉｎｇｄｏｍ
(74)代理人　100169904
弁理士　村井　康司
(74)代理人　100121120
弁理士　渡辺　尚

最終頁に続く

(54)【発明の名称】マルチスレッドプロセッサのレジスタファイル

(57)【要約】
【課題】マルチスレッドプロセッサのレジスタファイルを提供する。
【解決手段】並列スレッドを実行するためのバレルスレッド実行ユニットと、各並列スレッドに対するコンテキストレジスタのそれぞれのセットを含む１つまたは複数のレジスタファイルとを含むプロセッサ。レジスタファイルのうちの１つは、並列スレッドのいくつかまたはすべてに共通の共有の重みレジスタのセットをさらに含む。プロセッサの命令セットにおいて定義される命令のタイプは、オペランドを有する算術命令を含み、オペランドは、算術命令が実行されるスレッドの算術レジスタのそれぞれのセットの中から送信元および送信先を指定する。実行ユニットは、算術命令のオペコードに応答して、送信元からの入力に共有重みレジスタのうちの少なくとも１つからの重みのうちの少なくとも１つを乗じることを含む演算を実行し、結果を送信先に入れるように構成される。
【選択図】図４



【特許請求の範囲】
【請求項１】
１つまたは複数のレジスタファイルと、
命令セットにおいて定義される命令タイプのインスタンスを実行するように構成された実行ユニットであって、前記命令セットの各命令が、オペコードおよび０またはそれ以上のオペランドからなる、実行ユニットと
を含むプロセッサであって、
前記実行ユニットは、多数の並列スレッドを実行するように構成されたバレルスレッド実行ユニットであり、前記並列スレッドの各々は、インターリーブされた時間スロットの繰り返しシーケンスに対応し、前記１つまたは複数のレジスタファイルは、前記並列スレッドの各々に対し、前記対応するスレッドのプログラム状態を保持するように構成されたコンテキストレジスタのそれぞれのセットを含み、コンテキストレジスタの各セットは、前記対応するスレッドによる使用のための算術オペランドレジスタの対応するセットを含み、
前記１つまたは複数のレジスタファイルのうちの１つは、前記並列スレッドのいくつかまたはすべてに共通の共有の重みレジスタのセットをさらに含み、
前記命令セットにおいて定義される前記命令のタイプは、オペランドを有する算術命令を含み、前記オペランドは、前記算術命令が実行される前記スレッドの算術レジスタの前記対応するセットの中から送信元および送信先を指定し、
前記実行ユニットは、前記算術命令の前記オペコードに応答して、前記送信元からの入力に前記共有の重みレジスタのうちの少なくとも１つを乗じることを含む演算を実行し、結果を前記送信先に入れるように構成される、プロセッサ。
【請求項２】
前記共有の重みレジスタのうちの前記少なくとも１つは、前記算術命令の前記オペコードから暗黙的であり、前記算術命令のいかなるオペランドによって指定されない、請求項１に記載のプロセッサ。
【請求項３】
前記算術命令は、前記共有の重みレジスタのセットの中から前記共有の重みレジスタのうち少なくとも１つを指定するさらなるオペランドを取る、請求項１に記載のプロセッサ。
【請求項４】
前記入力は、ベクトルを含み、前記乗算は、前記入力と前記共有の重みレジスタからの重みのベクトルとのドット積を含む、請求項１～３のいずれか一項に記載のプロセッサ。
【請求項５】
前記共有の重みレジスタのうちの前記少なくとも１つは、複数のサブセットのうち前記共有の重みレジスタのサブセットを含み、各サブセットは、それぞれの重みベクトルを保持し、前記さらなるオペランドは、前記乗算において使用するためにどのサブセットから前記重みベクトルを取り入れるかを選択する、請求項３または４に記載のプロセッサ。
【請求項６】
前記算術命令は、ベクトルドット積命令、累積ベクトルドット積命令、行列積命令、累積行列積命令または畳み込み命令のうちの１つを含む、請求項１～５のいずれか一項に記載のプロセッサ。
【請求項７】
前記並列スレッドは、複数のワーカースレッドを含み、前記実行ユニットは、少なくともいくつかの時間に、前記ワーカースレッドを管理するように構成された少なくとも１つのスーパーバイザスレッドを含むスーパーバイザサブプログラムを実行するようにさらに構成される、請求項１～６のいずれか一項に記載のプロセッサ。
【請求項８】
前記スーパーバイザサブプログラムは、前記共有の重みレジスタファイルに前記重みを書き込むように構成される、請求項７に記載のプロセッサ。

【請求項９】
前記共有の重みレジスタの前記重みは、前記スーパーバイザサブプログラムのみが書き込むことができ、前記ワーカースレッドは、前記共有の重みレジスタを読み取ることしかできないように構成される、請求項８に記載のプロセッサ。
【請求項１０】
前記コンテキストレジスタは、並列に実行することができる前記ワーカースレッドの各々に対する前記コンテキストレジスタのセットのうち対応する１つを含み、コンテキストレジスタの追加のセットは、前記スーパーバイザサブプログラムのプログラム状態を保持するように構成される、請求項７～９のいずれか一項に記載のプロセッサ。
【請求項１１】
前記スーパーバイザサブプログラムは、最初に前記スロットのすべてにおいて動作することによって開始し、前記ワーカースレッドを起動する前に前記重みを書き込むように構成され、
前記スーパーバイザサブプログラムは、前記スーパーバイザサブプログラムが動作していた前記スロットの各々を対応する前記ワーカースレッドに委ねることによって、前記ワーカースレッドを起動する、請求項１０に記載のプロセッサ。
【請求項１２】
前記命令セットは、実行命令を含み、前記実行命令は、前記スーパーバイザサブプログラムの一部として実行されると、前記実行命令が実行された前記スロットを前記ワーカースレッドのうちの１つに委ね、前記スーパーバイザサブプログラムの代わりに前記ワーカースレッドがそのスロットにおいて起動される、請求項１１に記載のプロセッサ。
【請求項１３】
前記命令セットは、終了命令を含み、前記終了命令は、前記ワーカースレッドのうちの１つの一部として実行されると、前記終了命令が実行された前記スロットを前記スーパーバイザサブプログラムに返し、前記ワーカースレッドの代わりに前記スーパーバイザサブプログラムがそのスロットにおいて再び実行を続ける、請求項１２に記載のプロセッサ。
【請求項１４】
前記レジスタファイルは、各並列ワーカースレッドに対する別個の算術レジスタファイルを含み、前記それぞれの算術レジスタファイルは、前記それぞれの算術オペランドレジスタを含む、請求項７～１３のいずれか一項に記載のプロセッサ。
【請求項１５】
前記レジスタファイルは、前記重みレジスタを含む別個の重みレジスタファイルを含む、請求項１４に記載のプロセッサ。
【請求項１６】
前記重みレジスタファイルは、前記スーパーバイザサブプログラムのみが書き込むことができ、前記ワーカースレッドは前記重みレジスタファイルを読み取ることしかできないように構成される、請求項１５に記載のプロセッサ。
【請求項１７】
請求項１～１６のいずれか一項に従って構成されたプロセッサを操作する方法であって、前記実行ユニットを通じて前記プロセッサ上で前記算術命令の１つまたは複数のインスタンスを含むプログラムを実行するステップを含む、方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本開示は、マルチスレッドプロセッサにおいて実行される機械語命令による使用のためのレジスタファイルの構成に関する。
【背景技術】
【０００２】
グラフィックス処理装置（ＧＰＵ）およびデジタル信号プロセッサ（ＤＳＰ）などの特定のアプリケーション用に設計されたプロセッサの開発への関心が高まってきている。最

近関心を集めた別のタイプのアプリケーション特有のプロセッサは、出願人によって「ＩＰＵ（ｉｎｔｅｌｌｉｇｅｎｃｅ ｐｒｏｃｅｓｓｉｎｇ ｕｎｉｔ）」（知能処理装置）と呼ばれる機械知能アプリケーション専用のものである。これらは、例えば、ニューラルネットワークなどの知識モデルを訓練するかまたは知識モデルの訓練を補助するため、あるいは、そのようなモデルに基づいて予測もしくは推論を実行するかまたは予測もしくは推論の実行を補助するために、ホストによって割り当てられた仕事を実行するように構成されたアクセラレータプロセッサとして採用することができる。

【０００３】

機械知能アルゴリズムは、複数の相互接続ノードのグラフによって表すことができる「知識モデル」に反復更新を実行することに基づく。各ノードは、その入力の関数を表す。ノードは、グラフへの入力を受信するものもあれば、１つまたは複数の他のノードからの入力を受信するものもある一方で、ノードの出力は、他のノードの入力を形成するものもあれば、グラフの出力を提供するものもある（いくつかの事例では、所定のノードは、これらのグラフへの入力、グラフからの出力および他のノードとの接続のすべてを有するものさえもあり得る）。さらに、各ノードにおける関数は、１つまたは複数のそれぞれのパラメータ（例えば、重み）によってパラメータ化される。学習段階では、経験に基づく入力データセットに基づいて、グラフが全体として、可能な入力の範囲に対して所望の出力を生成するように、様々なパラメータに対する値を見出すことを目的とする。この学習を行うための様々なアルゴリズムは、確率的勾配降下法に基づく誤差逆伝播アルゴリズムなど、当技術分野において知られている。入力データに基づく複数の反復にわたり、パラメータは、それらの誤差を減少するように徐々に調節され、グラフは、解に向けて収束する。次いで、後続の段階では、学習済みのモデルを使用して、指定された入力セットに与えられる出力の予測を行うこと、または、指定された出力セットに与えられる入力（原因）に関する推論を行うことができる。

【０００４】

機械知能アプリケーション用に設計されたプロセッサは、機械知能アプリケーションにおいて一般的に採用されている算術演算を実行するための専用命令をプロセッサの命令セットに含み得る（命令セットは、機械語命令タイプの基本セットであり、プロセッサの実行ユニットは、それぞれのオペコードおよび０またはそれ以上のオペランドによって定義されるタイプの各々を認識するように構成される）。例えば、ニューラルネットワークなどの機械知能アプリケーションにおいて必要な共通の演算は、入力データセットにわたるカーネルの畳み込みであり、カーネルは、ニューラルネットワークのノードの重みを表す。データにわたって大きなサイズのカーネルの畳み込みを実行するため、畳み込みは、複数のベクトルまたは行列積に分解することができ、その各々は、後続の積の出力と累計される部分和を出力する。畳み込みの実行において使用するためのベクトルおよび行列乗算タイプの演算を実行するための専用の算術命令をプロセッサの命令セットに含むプロセッサは、既に存在している。

【発明の概要】

【課題を解決するための手段】

【０００５】

プロセッサは、複数のプログラムスレッドの並列実行に対するサポートを提供することもできる。このサポートは、典型的には、各スレッドに対するコンテキストレジスタのそれぞれのセットを含み、各セットは、並列に実行される多数のスレッドのうちのそれぞれのスレッドのプログラム状態（「コンテキスト」）を保持するために使用される。所定のスレッドのコンテキストレジスタに保持されるプログラム状態は、典型的には、そのスレッドに対するプログラムカウンタと、スレッドの状態（例えば、一時停止している、実行されているなど）を記録する１つまたは複数の制御状態レジスタと、それぞれのスレッドの算術命令に基づいて操作される値およびそれぞれのスレッドの算術命令によって出力される値を一時的に保持するための算術オペランドレジスタを含む多数のオペランドレジスタとを含む。異なる並列スレッドは、異なるそれぞれの実行時間スロットにおいて共通の

実行パイプラインを通じて時間的にインターリーブ（interleave）され、各スロットは、コンテキストレジスタのセットのうちの異なるそれぞれのコンテキストレジスタのレジスタのみを利用する。

【０００６】

従来の方式では、各スレッドは、各スレッドに対する別個の算術レジスタファイルの算術オペランドレジスタのそれ自体の対応するセットを含む。所定のスレッドの一部として命令が実行される際、その命令は、暗黙的に、その特定のスレッドの算術レジスタファイルの算術オペランドレジスタを常に使用する。

【０００７】

スレッドが、実際に、同じオペランド値のいくつかに基づいて動作することがある一方で、各スレッドに対して特定の値を有する他のいくつかのオペランドに基づいて動作することもあるアプリケーションが発生しうることが知られている。その例として、各スレッドの命令が個々のスレッドに特有の入力データに共通の重みのセット（スレッド間で共有される）を乗じる畳み込みの一部として実行されるベクトルまたは行列乗算がある。この特定の例は、畳み込みニューラルネットワーク発生し、多くのノードは、実際に、同じ重みを含むが、異なる接続を有する。例えば、ある特徴を検出するためにそれぞれの入力データに共通の重みのカーネルを畳み込むなど、ニューラルネットワークの異なるそれぞれのノードの処理を実行するように各スレッドが構成されるシナリオを考える。そのようなシナリオでは、所定の算術命令が、スレッド間で共有される１つまたは複数の共通の重みオペランドと、個々の対応するスレッドに特有の１つまたは複数のオペランドとの組合せに基づいて動作可能となるメカニズムを提供することが有利である。その論点は、決してニューラルネットワークに特有ではなく、最後にはいくつかの共有されたオペランド値およびいくつかのスレッド特有のオペランドを使用することになるいかなるアプリケーションにおいても生じ得る。

【０００８】

本明細書で開示される一態様によれば、１つまたは複数のレジスタファイルと、命令セットにおいて定義される命令タイプのインスタンスを実行するように構成された実行ユニットとを含むプロセッサが提供され、命令セットの各命令は、オペコードおよび０またはそれ以上のオペランドからなる。実行ユニットは、多数の並列スレッドを実行するように構成されたバレルスレッド実行ユニットであり、並列スレッドの各々は、インターリーブされた時間スロットの繰り返しシーケンスのうちの異なるそれぞれのものであり、並列スレッドの各々に対し、１つまたは複数のレジスタファイルは、それぞれのスレッドのプログラム状態を保持するように構成されたコンテキストレジスタのそれぞれのセットを含み、コンテキストレジスタの各セットは、それぞれのスレッドによる使用のための算術オペランドレジスタのそれぞれのセットを含む。１つまたは複数のレジスタファイルのうちの１つは、並列スレッドのいくつかまたはすべてに共通の共有重みレジスタのセットをさらに含む。命令セットにおいて定義される命令のタイプは、オペランドを有する算術命令を含み、オペランドは、算術命令が実行されるスレッドの算術レジスタのそれぞれのセットの中から送信元および送信先を指定する。実行ユニットは、算術命令のオペコードに応答して、上記送信元からの入力に共有重みレジスタのうちの少なくとも１つからの重みのうちの少なくとも１つを乗じることを含む演算を実行し、結果を上記送信先に入れるように構成される。

【０００９】

実施形態では、共有重みレジスタのうちの上記少なくとも１つは、算術命令のオペコードから暗黙的であってもよく、算術命令のいかなるオペランドによっても指定されなくてもよい。

【００１０】

実施形態では、算術命令は、共有重みレジスタのセットの中から共有重みレジスタのうちの上記少なくとも１つを指定するさらなるオペランドを取ってもよい。

【００１１】

実施形態では、入力は、ベクトルを含んでもよく、乗算は、入力と共有重みレジスタからの重みのベクトルとのドット積を含んでもよい。

【００１２】

実施形態では、共有重みレジスタのうちの上記少なくとも１つは、多数のサブセットの中から共有重みレジスタのサブセットを含んでもよく、各サブセットは、それぞれの重みベクトルを保持し、上記さらなるオペランドは、上記乗算において使用するためにどのサブセットから重みベクトルを取り入れるかを選択可能であってもよい。

【００１３】

実施形態では、上記算術命令は、ベクトルドット積命令、累積ベクトルドット積命令、行列積命令、累積行列積命令または畳み込み命令のうちの１つを含んでもよい。

【００１４】

実施形態では、並列スレッドは、多数のワーカースレッドを含んでもよく、実行ユニットは、少なくともいくつかの時間に、ワーカースレッドを管理するように構成された少なくとも１つのスーパーバイザスレッドを含むスーパーバイザサブプログラムを実行するようにさらに構成されてもよい。

【００１５】

実施形態では、スーパーバイザサブプログラムは、共有重みレジスタファイルに重みを書き込むように構成されてもよい。

【００１６】

実施形態では、共有重みレジスタの重みは、スーパーバイザサブプログラムのみが書き込むことができ、ワーカースレッドは、共有重みレジスタを読み取りのみ可能であってもよい。

【００１７】

実施形態では、コンテキストレジスタは、並列に実行することができるワーカースレッドの各々に対するコンテキストレジスタのセットのそれぞれを含んでもよく、コンテキストレジスタの追加のセットは、スーパーバイザサブプログラムのプログラム状態を保持するように構成されてもよい。

【００１８】

実施形態では、スーパーバイザサブプログラムは、最初にスロットのすべてにおいて動作することによって開始し、ワーカースレッドを起動する前に重みを書き込むように構成されてもよい。すなわち、スーパーバイザサブプログラムは、スーパーバイザサブプログラムが最初に動作していたいくつかまたはすべてのスロットの各々を対応するワーカースレッドに委ねることによって各ワーカースレッドを起動してもよい。

【００１９】

実施形態では、命令セットは、実行（run）命令を含んでもよい。実行命令は、スーパーバイザサブプログラムの一部として実行されると、実行命令が実行されたスロットをワーカースレッドのうちの１つに委ね、スーパーバイザサブプログラムの代わりにワーカースレッドをそのスロットにおいて起動させる。

【００２０】

実施形態では、命令セットは、終了（exit）命令を含んでもよい。終了命令は、ワーカースレッドのうちの１つの一部として実行されると、終了命令が実行されたスロットをスーパーバイザサブプログラムに返し、ワーカースレッドの代わりに、スーパーバイザサブプログラムがそのスロットにおいて再び実行を続ける。

【００２１】

実施形態では、レジスタファイルは、各並列ワーカースレッドに対する別個の算術レジスタファイルを含み、それぞれの算術レジスタファイルは、それぞれの算術オペランドレジスタを含む。

【００２２】

実施形態では、レジスタファイルは、重みレジスタを含む別個の重みレジスタファイルを含んでもよい。

【００２３】
実施形態では、重みレジスタファイルは、スーパーバイザサブプログラムのみが書き込むことができ、ワーカースレッドは重みレジスタファイルを読み取ることしかできないように構成されてもよい。
【００２４】
本発明の別の態様によれば、上記または本明細書の他の場所の任意の実施形態に従って構成されたプロセッサを操作する方法であって、実行ユニットを通じてプロセッサ上で算術命令の１つまたは複数のインスタンスを含むプログラムを実行するステップを含む、方法が提供される。
【００２５】
本開示の実施形態を理解することを支援するため、および、そのような実施形態をどのように実施するかを示すため、単なる例示として、添付の図面を参照する。
【図面の簡単な説明】
【００２６】
【図１】例示的なマルチスレッドプロセッサの概略ブロック図である。
【図２】インターリーブされた時間スロットのスキームを概略的に示す。
【図３】多数のインターリーブされた時間スロットにおいて実行するスーパーバイザスレッドおよび多数のワーカースレッドを概略的に示す。
【図４】例示的なプロセッサの論理ブロック構造を概略的に示す。
【図５】構成プロセッサのアレイを含むプロセッサの概略ブロック図である。
【図６】機械知能アルゴリズムにおいて使用されるグラフの概略図である。
【図７】あるタイプのロード／ストア命令を実装する際に使用するためのアドレスパッキングスキームを概略的に示す。
【図８】ストライドレジスタセットにおける既定のストライド値の構成を概略的に示す。
【図９】あるボリュームの入力データと３Ｄカーネルとの畳み込みを概略的に示す。
【図１０】累積行列積命令の位相のシーケンスによって実行された行列乗算を概略的に示す。
【図１１】累積行列積命令の動作をさらに示す。
【図１２】畳み込みを実行するように構成された累積行列積命令のシーケンスの一連のループの例を与える。
【図１３】畳み込み命令の動作を概略的に示す。
【図１４】一連の畳み込み命令の例を与える。
【発明を実施するための形態】
【００２７】
実施形態の詳細な説明
図１は、本開示の実施形態によるプロセッサ４の例を示す。プロセッサ４は、バレルスレッド処理装置の形態のマルチスレッド処理装置１０と、ローカルメモリ１１（すなわち、マルチタイルアレイの場合は同じタイル上、または、単一プロセッサチップの場合は同じチップ上）とを含む。バレルスレッド処理装置は、パイプラインの実行時間が、インターリーブされた時間スロットの繰り返しシーケンス（その各々は、所定のスレッドによって占有される）に分割されるタイプのマルチスレッド処理装置である。また、これは、間もなくさらに詳細に論じられるように、並列実行と呼ぶこともできる。メモリ１１は、命令メモリ１２およびデータメモリ２２（異なるアドレス可能メモリユニットまたは同じアドレス可能メモリユニットの異なる領域において実装することができる）を含む。命令メモリ１２は、処理装置１０によって実行される機械語を格納し、データメモリ２２は、実行コードによって処理されるデータと、実行コードによって出力されたデータ（例えば、そのような処理の結果として）の両方を格納する。
【００２８】
メモリ１２は、プログラムの複数の異なるスレッドを格納し、各スレッドは、所定の１つまたは複数のタスクを実行するための命令の対応するシーケンスを含む。本明細書で言

及される命令は、単一のオペコードおよび０またはそれ以上のオペランドからなる機械語命令（すなわち、プロセッサの命令セットの基本命令のうちの１つのインスタンス）を意味することに留意されたい。実施形態では、プログラムは、多数のワーカースレッドと、１つまたは複数のスーパーバイザスレッドとして構造化することができるスーパーバイザサブプログラムとを含む。これらについては、間もなくさらに詳細に論じる。

【００２９】

マルチスレッドプロセッサは、典型的には並列的に、互いに並んで配置された複数のプログラムスレッドの実行が可能なプロセッサである。並列実行は、スレッドが、共通の実行パイプライン（またはパイプラインの少なくとも共通部分）を共有し、異なるスレッドが、繰り返しサイクルの異なるインターリーブされた時間スロットにおいて、この同じ共有された実行パイプラインを通じてインターリーブされることを意味する。これにより、パイプライン遅延を隠蔽する機会が増大するので、性能が向上する。プロセッサは、複数の異なるスレッドに共通の何らかのハードウェア（例えば、共通の命令メモリ、データメモリおよび／または実行ユニット）を含むが、マルチスレッドをサポートするため、プロセッサは、各スレッドに特有の何らかの専用ハードウェアも含む。

【００３０】

専用ハードウェアは、並列に実行することができるスレッドの少なくとも各々に対するコンテキストレジスタ２６の個別のセット（すなわち、サイクルの１スロット当たり１セット）を含む。マルチスレッドプロセッサについて述べる際、「コンテキスト」は、互いに並んで実行されている複数のスレッドのそれぞれのスレッドのプログラム状態（例えば、プログラムカウンタ値、ステータスおよび現在のオペランド値）を指す。コンテキストレジスタは、それぞれのスレッドのこのプログラム状態を表すためのそれぞれのレジスタを指す。レジスタファイルのレジスタは、レジスタアドレスが命令語においてビットとして固定されるという点で、一般的なメモリとは別個のものであり、メモリアドレスは、命令を実行することによって演算することができる。

【００３１】

処理装置１０内では、命令メモリ１２からのスレッドの複数の異なるスレッドを、単一の実行パイプライン１３を通じてインターリーブできる（ただし、典型的には、命令メモリに格納された全スレッドのサブセットのみが全プログラムの任意の所定のポイントにおいてインターリーブされうる）。マルチスレッド処理装置１０は、コンテキストレジスタ２６の多数のセットを含み、各セット２６は、並列に実行することができる複数のスレッドのうちの異なるそれぞれのスレッドの状態（コンテキスト）を表すように構成される。また、マルチスレッド処理装置１０は、並列に実行されるスレッドに共通の共有された実行パイプライン１３や、インターリーブ方法で（例えば、ラウンドロビン方法で）共有されたパイプラインを通じて実行するための並列スレッドをスケジュールするためのスケジューラ２４も含む。処理装置１０は、多数のスレッドに共通の共有された命令メモリ１２と、これもまた多数のスレッドに共通の共有されたデータメモリ２２とに接続される。

【００３２】

実行パイプライン１３は、フェッチステージ１４と、復号ステージ１６と、命令セットアーキテクチャによって定義されるような、算術および論理演算、アドレス計算、ロードおよびストア操作ならびに他の動作を実行することができる実行ユニットを含む実行ステージ１８とを含む。

【００３３】

コンテキストレジスタ２６の各セットは、対応する１つまたは複数の制御レジスタを含み、制御レジスタは、それぞれのスレッドに対する少なくともプログラムカウンタ（ＰＣ）を含み（スレッドが現在実行し命令アドレスを追跡するため）、また、実施形態では、対応するスレッドの現在の状態（スレッドが現在実行されているかまたは一時停止しているかなど）を記録する１つまたは複数の制御状態レジスタ（ＣＳＲ）のセットも含む。また、コンテキストレジスタファイル２６の各セットは、それぞれのスレッドによって実行される命令のオペランド（すなわち、実行される際にそれぞれのスレッドの命令のオペコ

ードによって定義された動作に基づいて操作される値または定義された動作から生じる値）を一時的に保持するためのオペランドレジスタのそれぞれのセットも含む。レジスタ２６の各セットは、１つまたは複数のレジスタファイルにおいて実装することができる。

【００３４】

フェッチステージ１４は、複数のコンテキストの各々のコンテキストのプログラムカウンタ（ＰＣ）へアクセスできる。対応するスレッドのそれぞれに対し、フェッチステージ１４は、プログラムカウンタによって示されたプログラムメモリ１２の次のアドレスからそのスレッドの次の命令をフェッチする。プログラムカウンタは、分岐命令によって分岐されない限り、各実行サイクルを自動的にインクリメントする。次いで、フェッチステージ１４は、フェッチした命令を復号のために復号ステージ１６に渡し、次いで、復号ステージ１６は、命令を実行するために、命令において指定されたいかなるオペランドレジスタの復号化されたアドレスと共に、復号化された命令を実行ユニット１８に渡す。実行ユニット１８は、オペランドレジスタおよび制御状態レジスタへアクセスできる。実行ユニットは、算術命令の場合のように、復号化されたレジスタアドレスに基づいて命令を実行するときに、そのオペランドレジスタおよび制御状態レジスタを用いてもよい（例えば、２つのオペランドレジスタの値の加算、乗算、減算または除算を行い、対応するスレッドの別のオペランドレジスタに結果を出力する）。あるいは、命令がメモリアクセス（ロードまたはストア）を定義する場合は、実行ユニット１８のロード／ストアロジックは、命令に従って、データメモリからそれぞれのスレッドのオペランドレジスタに値をロードするか、または、それぞれのスレッドのオペランドレジスタからデータメモリ２２に値を格納する。

【００３５】

フェッチステージ１４は、スケジューラ２４の制御の下で、実行される命令を命令メモリ１２からフェッチ可能に接続される。スケジューラ２４は、時間スロットの繰り返しシーケンスにおいて次々と並列に実行しているスレッドのセットの各々から命令をフェッチするようにフェッチステージ１４を制御するように構成され、従って、間もなくさらに詳細に論じられるように、パイプライン１３のリソースを多数の時間的にインターリーブされた時間スロットに分割する。例えば、スケジューリングスキームは、ラウンドロビンまたは重み付けラウンドロビンであり得る。そのような方法で動作するプロセッサに対する別の用語は、バレルスレッドプロセッサである。

【００３６】

スケジューラ２４によって実装されるインタリーブスキームの例は、図２に示されている。ここでは、並列スレッドは、ラウンドロビンスキームに従ってインターリーブされ、それにより、スキームの各ラウンド内では、ラウンドは、時間スロットのシーケンスＳ０、Ｓ１、Ｓ２…ＳＪ－１（例えば、Ｊ＝４またはＪ＝６）に分割され、各スロットは、それぞれのスレッドを実行するためのものである。典型的には、各スロットは、１つの実行サイクルの長さであり、異なるスロットが均等にサイズ指定されるが、すべての可能な実施形態において必ずしもそうであるとは限らず、例えば、１回の実行ラウンド当たりいくつかのスレッドが他のスレッドより多くのサイクルを獲得する重み付けラウンドロビンスキームも可能である。一般に、バレルスレッドは、均一なラウンドロビンまたは重み付けラウンドロビンスケジュールを採用することができ、後者の場合、重み付けは、固定することも、適応化させることもできる。

【００３７】

１回の実行ラウンド当たりのシーケンスが何であれ、このパターンは繰り返され、各ラウンドは、複数の時間スロットの各々の対応するインスタンスを含む。従って、本明細書で言及される場合は、時間スロットは、シーケンスにおける繰り返し割り当てられる場所を意味し、シーケンスの特定の繰り返しにおける時間スロットの特定のインスタンスを意味しないことに留意されたい。別の言い方をすれば、スケジューラ２４は、パイプライン１３の実行サイクルを多数の時間的にインターリーブされた（時分割多重化）実行チャネルに分配し、各実行チャネルは、時間スロットの繰り返しシーケンスにおけるそれぞれの

時間スロットの再現を含む。示される実施形態では、４つの時間スロットが存在しているが、これは単なる例示を目的としており、他の数も可能である。例えば、好ましい一実施形態では、実際に、６つの時間スロットが存在する。

【００３８】

実施形態では、コンテキストレジスタ２６は、並列に実行することができるＪ個の数のスレッド（示される例では、Ｊ＝３であるが、これに限定されない）の各々に対するワーカーコンテキストレジスタＣＸ０…ＣＸ（Ｊ－１）のそれぞれのセットと、１つの追加のスーパーバイザコンテキストレジスタファイルＣＸＳとを含む。ワーカーコンテキストレジスタファイルは、ワーカースレッドのコンテキストを保持し、スーパーバイザコンテキストレジスタファイルは、スーパーバイザスレッドのコンテキストを保持する。実施形態では、スーパーバイザコンテキストは、ワーカーの各々とは異なる数のレジスタを有することに留意されたい。従って、処理装置１０は、時間スロットよりも１つ多いコンテキストレジスタファイル２６を含む（すなわち、処理装置１０は、バレルスレッドが可能なインターリーブされた時間スロットの数よりも１つ多いコンテキストをサポートする）。

【００３９】

ワーカーコンテキストＣＸ０…ＣＸＪの各々は、プログラマが望むアプリケーション特有の演算タスクは何でも実行するために、４つの実行時間スロットＳ０…ＳＪのうちの１つに現在割り当てられている多数のワーカースレッドのそれぞれのワーカースレッドの状態を表すために使用される（この場合もやはり、このワーカーコンテキストは、命令メモリ１２に格納されるようなプログラムのワーカースレッドの総数の単なるサブセットであり得ることに留意されたい）。追加のコンテキストＣＸＳは、「スーパーバイザスレッド」（ＳＶ）の状態を表すために使用され、その役割は、少なくとも実行のためにどのワーカースレッドＷを全プログラムのどのポイントにおいてどの時間スロットＳ０、Ｓ１、Ｓ２…に割り当てるかという意味で、ワーカースレッドの実行を調整することである。任意に、スーパーバイザスレッドは、外部の交換またはバリア同期の実行などの他の「オーバーシーア」または調整責任を有し得る。当然ながら、Ｊ＝４の示される事例は、例示を目的とする実装形態の単なる一例であることに留意されたい。例えば、別の実装形態では、Ｊ＝６（６つの時間スロット、６つのワーカーコンテキストおよび１つのスーパーバイザコンテキスト）である。

【００４０】

図３を参照すると、実施形態では、スーパーバイザスレッドＳＶは、インターリーブされた実行時間スロットのスキームにそれ自体の時間スロットを有さない。また、ワーカースレッドへのスロットの割り当ては柔軟に定義されるため、ワーカーも同様である。むしろ、各時間スロットは、ワーカーコンテキストを格納するためのコンテキストレジスタのそれ自体のセットを有し、そのセットは、ワーカーによって、スロットがワーカーに割り当てられる際に使用されるが、スロットがスーパーバイザに割り当てられる際には使用されない。所定のスロットがスーパーバイザに割り当てられる際は、そのスロットは代わりに、スーパーバイザのコンテキストレジスタファイルＣＸＳを使用する。スーパーバイザは、常に、それ自体のコンテキストへのアクセスを有し、ワーカーは、スーパーバイザコンテキストレジスタファイルＣＸＳを占有することはできない。

【００４１】

スーパーバイザスレッドＳＶは、ありとあらゆる時間スロットＳ０…Ｓ３（またはより一般的にはＳ０…ＳＪ－１）において動作する能力を有する。スケジューラ２４は、プログラムが全体として開始されると、スーパーバイザスレッドを時間スロットのすべてに割り当てることで開始するよう、すなわち、スーパーバイザＳＶがＳ０…ＳＪ－１のすべてにおいて動作を開始するように構成される。しかし、スーパーバイザスレッドには、後続のある時点で（すぐにまたは１つもしくは複数のスーパーバイザタスクを実行した後に）、実行しているスロットの各々をワーカースレッドのそれぞれに（例えば、図３に示される例では、最初にワーカーＷ０…Ｗ３（またはより一般的にはＷ０…ＷＪ－１）に）一時的に委ねるためのメカニズムが提供される。これは、命令メモリ１２のワーカースレッド

のアドレスをオペランドとして少なくとも取る実行（run）命令を実行するスーパーバイザスレッドによって達成される。ワーカースレッドは、互いに並列に実行することができるコードの部分であり、その各々は、実行される１つまたは複数のそれぞれの演算タスクを表す。

【００４２】

実行命令は、この命令自体が実行されている現在の時間スロットを、オペランドによって指定されたワーカースレッドに委ねるように、スケジューラ２４に作用する。実行命令では、委ねられるものが、この命令が実行される時間スロットであることは暗黙的であることに留意されたい（機械語命令の文脈では、暗黙的であるということは、オペランドがこれを指定する必要がないということ、すなわち、オペコード自体から暗黙のうちに理解されていることを意味する）。従って、渡される時間スロットは、スーパーバイザが実行命令を実行する時間スロットである。

【００４３】

スーパーバイザスレッドＳＶは、ワーカースレッドＷ０…ＷＪ－１の異なるそれぞれのワーカースレッド（命令メモリ１２の可能なワーカースレッドのより大きなセットから選択される）にその時間スロットのいくつかまたはすべてを渡すために、複数の時間スロットのうちの１つまたは複数の他の時間スロットの各々において同様の動作を実行する。最後のスロットに対してそのように行われた時点で、スーパーバイザは、中断される（次いで、後に、スロットのうちの１つがワーカーＷから返された際に、中断したところから再開される）。従って、スーパーバイザスレッドＳＶは、各々が１つまたは複数のタスクを実行する異なるワーカースレッドを、インターリーブされた実行時間スロットＳ０…ＳＪ－１（例えば、示されるようにＪ＝４、または、Ｊ＝６）の異なるインターリーブされた実行時間スロットに割り当てることができる。ワーカースレッドを実行する時間であるとスーパーバイザスレッドが決定すると、スーパーバイザスレッドは、実行命令を使用して、実行命令が実行された時間スロットにこのワーカーを割り当てる。

【００４４】

いくつかの実施形態では、命令セットは、「すべてを実行する（run-all）」という実行命令の変形形態も含み得る。この命令は、複数のワーカーのセットを共に起動するために使用され、すべてが同じコードを実行する。実施形態では、この命令は、処理装置のスロットＳ０…Ｓ３（またはより一般的にはＳ０…Ｓ（Ｊ－１））内のワーカーを１つ残らず起動する。

【００４５】

いったん起動されると、現在割り当てられているワーカースレッドＷ０…ＷＪ－１の各々は、それぞれの実行命令によって指定されたコードにおいて定義された１つまたは複数の演算タスクの実行に進む。次いで、この実行の終了時には、それぞれのワーカースレッドは、実行している時間スロットをスーパーバイザスレッドに返す。これは、それぞれのワーカースレッドにおいて終了（exit）命令を実行することによって達成される。終了命令は、この命令自体が実行されている現在の時間スロットをスーパーバイザスレッドに戻すように、スケジューラ２４に作用する。それに応答して、スケジューラ２４は、そのスロットにおいてスーパーバイザの実行を続ける。

【００４６】

図４は、実行ユニット１８およびコンテキストレジスタ２６の詳細を含むプロセッサ４のさらなる例示的な詳細を示す。プロセッサは、並列実行が可能なＭ個のスレッドの各々に対するそれぞれの命令バッファ５３を含む。コンテキストレジスタ２６は、ワーカーＭコンテキストおよびスーパーバイザコンテキストの各々に対する対応するメインレジスタファイル（ＭＲＦ）２６Ｍを含む。コンテキストレジスタは、ワーカーコンテキストの少なくとも各々に対するそれぞれの補助レジスタファイル（ＡＲＦ）２６Ａをさらに含む。コンテキストレジスタ２６は、共通の重みレジスタファイル（ＷＲＦ）２６Ｗをさらに含む。現在実行しているワーカースレッドのすべてが、重みレジスタファイル（ＷＲＦ）から読み出しアクセス可能である。スーパーバイザスレッドをＷＲＦに書き込むことができ

る唯一のスレッドとするよう、ＷＲＦはスーパーバイザコンテキストと関連付けられてもよい。また、コンテキストレジスタ２６は、スーパーバイザおよびワーカーコンテキストの各々に対する制御状態レジスタ２６ＣＳＲの対応するグループも含んでもよい。実行ユニット１８は、メイン実行ユニット１８Ｍおよび補助実行ユニット１８Ａを含む。メイン実行ユニット１８Ｍは、ロード／ストアユニット（ＬＳＵ）５５および整数演算論理ユニット（ＩＡＬＵ）５６を含む。補助実行ユニット１８Ａは、少なくとも浮動小数点演算ユニット（ＦＰＵ）を含む。

【００４７】

Ｊ個のインターリーブされた時間スロットＳ０…ＳＪ－１の各々では、スケジューラ２４は、命令メモリ１２から、現在の時間スロットに対応するＪ個の命令バッファ５３のそれぞれに、対応するスレッドの少なくとも１つの命令をフェッチするようにフェッチステージ１４を制御する。実施形態では、各時間スロットは、プロセッサの１つの実行サイクルであるが、他のスキーム（例えば、重み付けラウンドロビン）も除外されない。プロセッサ４の各実行サイクル（すなわち、プログラムカウンタ時間を記録するプロセッサクロックの各サイクル）では、フェッチステージ１４は、実装形態に応じて、単一の命令または小さな「命令バンドル」（例えば、２命令バンドルまたは４命令バンドル）をフェッチする。次いで、各命令は、復号ステージ１６を介して、命令がメモリアクセス命令であるか、整数演算命令であるか、浮動小数点演算命令であるかに応じてそれぞれ（そのオペコードに従って）、メイン実行ユニット１８ＭのＬＳＵ　５５もしくはＩＡＬＵ　５６または補助実行ユニット１８ＡのＦＰＵのうちの１つに発行（issue）される。メイン実行ユニット１８ＭのＬＳＵ　５５およびＩＡＬＵ　５６は、ＭＲＦ　２６Ｍからのレジスタを使用してそれらの命令を実行し、ＭＲＦ　２６Ｍ内の特定のレジスタは、命令のオペランドによって指定される。補助実行ユニット１８ＡのＦＰＵは、ＡＲＦ　２６ＡおよびＷＲＦ　２６Ｗのレジスタを使用して演算を実行し、ＡＲＦ内の特定のレジスタは、命令のオペランドによって指定される。実施形態では、ＷＲＦのレジスタは、命令タイプにおいて暗黙的であり得る（すなわち、その命令タイプに対して事前に決定される）。また、補助実行ユニット１８Ａは、浮動小数点演算命令のタイプのうちの１つまたは複数の演算を実行する際に使用するためのいくつかの内部状態５７を保持するための補助実行ユニット１８Ａの内部にある論理ラッチの形態の回路も含んでもよい。

【００４８】

バンドルの命令をフェッチして実行する実施形態では、所定の命令バンドルの個々の命令は、下流の独立したパイプライン１８Ｍ、１８Ａ（図４に示される）において並列して、同時に実行される。２つの命令のバンドルを実行する実施形態では、２つの命令は、下流のそれぞれの補助およびメインパイプラインにおいて同時に実行することができる。この場合、メインパイプラインは、ＭＲＦを使用するタイプの命令を実行するように構成され、補助パイプラインは、ＡＲＦを使用するタイプの命令を実行するために使用される。命令の適切な相補バンドルへのペアリング（pairing）は、コンパイラによって実現（handled）してもよい。

【００４９】

各ワーカースレッドコンテキストは、メインレジスタファイル（ＭＲＦ）２６Ｍおよび補助レジスタファイル（ＡＲＦ）２６Ａのそれ自体のインスタンス（すなわち、バレルスレッドスロットの各々に対して１つのＭＲＦおよび１つのＡＲＦ）を有する。ＭＲＦまたはＡＲＦに関連して本明細書で説明される機能は、コンテキスト別に動作するものと理解されたい。しかし、スレッド間で共有される単一の共有された重みレジスタファイル（ＷＲＦ）が存在する。各スレッドは、それ自体のコンテキスト２６のＭＲＦおよびＡＲＦのみにアクセスすることができる。しかし、現在実行しているワーカースレッドはすべて、共通のＷＲＦにアクセスすることができる。従って、ＷＲＦは、すべてのワーカースレッドによる使用のための重みの共通のセットを提供する。実施形態では、スーパーバイザのみがＷＲＦに書き込むことができ、ワーカーは、ＷＲＦから読み取ることしかできない。

【００５０】

プロセッサ４の命令セットは、少なくとも１つのタイプのロード（load）命令を含み、そのロード命令は、実行されると、そのオペコードが、データメモリ２２からロード命令が実行されたスレッドのそれぞれのＡＲＦ　２６ＡへのデータのロードをＬＳＵ　５５に行わせる。ＡＲＦ内の送信先の場所は、ロード命令のオペランドによって指定される。ロード命令の別のオペランドは、それぞれのＭＲＦ　２６Ｍのアドレスレジスタを指定し、それぞれのＭＲＦ　２６Ｍのアドレスレジスタは、データをロードするためのデータメモリ２２のアドレスへのポインタを保持する。また、プロセッサ４の命令セットは、少なくとも１つのタイプのストア（store）命令も含み、そのストア命令は、実行されると、そのオペコードが、ストア命令が実行されたスレッドのそれぞれのＡＲＦからデータメモリ２２へのデータの格納をＬＳＵ　５５に行わせる。ＡＲＦ内のストアの送信元の場所は、ストア命令のオペランドによって指定される。ストア命令の別のオペランドは、ＭＲＦのアドレスレジスタを指定し、ＭＲＦのアドレスレジスタは、データを格納するためのデータメモリ２２のアドレスへのポインタを保持する。一般に、命令セットは、別個のロードおよびストア命令タイプ、ならびに／あるいは、ロードおよびストア操作を組み合わせて単一の命令にした少なくとも１つのロード／ストア命令タイプを含んでもよい。間もなくさらに詳細に論じられるように、命令セットは、２つのロード操作および１つのストア操作をすべて単一の命令で実行する特定のタイプのロード／ストア命令を含んでもよい。所定のプロセッサ４の命令セットは、複数の異なる種類のロード、ストアおよび／またはロード／ストア命令タイプを含み得ることに留意されたい。

【００５１】

また、プロセッサの命令セットは、算術演算を実行するための１つまたは複数のタイプの算術命令も含む。本明細書で開示される実施形態によれば、これらの命令セットは、共通の重みレジスタファイルＷＲＦ　２６Ｗを利用する少なくとも１つのタイプの算術命令を含んでもよい。このタイプの命令は、算術命令が実行されたスレッドのそれぞれのＡＲＦ　２６Ａの対応する算術演算の少なくとも１つの送信元を指定する少なくとも１つのオペランドを取る。しかし、算術命令の少なくとも１つの他の送信元は、共通のＷＲＦにあり、すべてのワーカースレッドに共通である。実施形態では、この送信元は、問題の算術命令において暗黙的である（すなわち、このタイプの算術命令に対して暗黙的である）。機械語命令の意味において、暗黙的であるということは、オペランドを指定する必要がないことを意味する。すなわち、この場合、ＷＲＦの送信元の場所は、オペコードに固有のものである（その特定のオペコードに対して事前に決定される）。あるいは、他の実施形態では、算術命令は、ＷＲＦの少数の異なるセットの中のどの重みレジスタのセットから重みを取り入れるかを指定するオペランドを取ることができる。しかし、重みの送信元はＷＲＦで見られるという事実は（例えば、汎用ＭＲＦまたはＡＲＦとは対照的に）、依然として暗黙的である。

【００５２】

算術命令の関連タイプのオペコードに応答して、補助実行ユニット１８Ａの演算ユニット（例えば、ＦＰＵ）は、オペコードによって指定されるように算術演算を実行し、算術演算は、スレッドのそれぞれのＡＲＦの指定された送信元レジスタおよびＷＲＦの送信元レジスタの値に基づいて演算するステップを含む。また、演算ユニットは、算術命令の送信先オペランドによって明示的に指定されるように、スレッドのそれぞれのＡＲＦの送信先レジスタへの算術演算の結果の出力も行う。

【００５３】

共通のＷＲＦ　２６Ｗの送信元を採用することができる算術命令の例示的なタイプは、１つもしくは複数のベクトル乗算命令タイプ、１つもしくは複数の行列乗算命令タイプ、１つもしくは複数の累積ベクトル乗算命令タイプおよび／または累積行列乗算命令タイプ（命令のあるインスタンスから次のインスタンスまで乗算の結果を累計する）ならびに／あるいは１つもしくは複数の畳み込み命令タイプを含んでもよい。例えば、ベクトル乗算命令タイプは、ＡＲＦ　２６Ａからの明示的な入力ベクトルにＷＲＦからの既定の重みベクトルを乗じることができる。あるいは、行列乗算命令タイプは、ＡＲＦからの明示的な

入力ベクトルにＷＲＦからの既定の重み行列を乗じることができる。別の例として、畳み込み命令タイプは、ＡＲＦからの入力行列にＷＲＦからの既定の行列を畳み込むことができる。多数のスレッドに共通の共有重みレジスタファイルＷＲＦを有することにより、各スレッドは、それ自体のそれぞれのデータに共通のカーネルを乗じるまたは畳み込むことができる。これは機械学習アプリケーションにおいて、例えば、各スレッドがニューラルネットワークの異なるノードを表し、共通のカーネルが検索または訓練されている特徴（例えば、グラフィカルデータのエリアまたはボリュームのエッジまたは特定の形状）を表す場合に、多く現れるシナリオであるため、これは有益である。

【００５４】

実施形態では、ＷＲＦ　２６Ｗの値は、スーパーバイザスレッドによって書き込むことができる。スーパーバイザ（実施形態では、すべてのスロットＳ０・・・ＳＭにおいて実行することによって開始される）は、最初に、いくつかの共通の重み値をＷＲＦの既定の場所に書き込むために、一連のプット（put）命令を実行する。次いで、スーパーバイザは、スロットＳ０…ＳＪ－１のうちのいくつかまたはすべてのそれぞれのワーカーを起動するための命令を実行する（またはすべての命令を実行する）。次いで、各ワーカーは、上記で論じられるタイプの１つまたは複数の算術命令の１つまたは複数のインスタンスを含み、それにより、それらの対応するＡＲＦ　２６Ａにロードされると、それら自体の対応する入力データに対して対応する算術演算を実行する（ただし、スーパーバイザによってＷＲＦ　２６Ｗに書き込まれた共通重みを使用して）。各スレッドがそのそれぞれのタスクを終了すると、各ワーカーは、そのスロットをスーパーバイザに返すために、終了命令を実行する。起動されたすべてのスレッドがそれらの対応するタスクを終了すると、スーパーバイザは、新しい値をＷＲＦに書き込み、新しいスレッドセットを起動することができる（または、新しいセットを起動して、ＷＲＦの既存の値の使用を続けることができる）。

【００５５】

「メイン」、「補助」および「重み」というラベルは必ずしも制限されないことが理解されよう。実施形態では、それらのラベルは、第１のレジスタファイル（１つのワーカーコンテキスト当たり）、第２のレジスタファイル（１つのワーカーコンテキスト当たり）および共有された第３のレジスタファイル（例えば、スーパーバイザコンテキストの一部であるが、すべてのワーカーがアクセス可能である）のいずれかであってもよい。ＡＲＦ　２６Ａおよび補助実行ユニット１８Ａは、算術命令（または少なくとも浮動小数点演算）のために使用されるため、算術レジスタファイルおよび算術実行ユニットと呼ぶこともできる。ＭＲＦ　２６Ｍおよびメイン実行ユニット１８Ｍは、それらの使用のうちの１つがメモリにアクセスするためのものであるため、メモリアドレスレジスタファイルおよびロード／ストアユニットと呼ぶこともできる。重みレジスタファイル（ＷＲＦ）２６Ｗは、間もなくさらに詳細に論じられるように、一定の１つまたは複数のタイプの算術命令で使用される乗法重みを保持するために使用されるため、そのように呼ばれる。例えば、これらの重みレジスタファイル（ＷＲＦ）２６Ｗは、ニューラルネットワークのノードの重みを表すために使用することができる。別の方法で見ると、ＭＲＦは、整数オペランドを保持するために使用されるため、整数レジスタファイルと呼ぶことができ、ＡＲＦは、浮動小数点オペランドを保持するために使用されるため、浮動小数点レジスタファイルと呼ぶことができる。２つのバンドルの命令を実行する実施形態では、ＭＲＦは、メインパイプラインによって使用されるレジスタファイルであり、ＡＲＦは、補助パイプラインによって使用されるレジスタである。

【００５６】

しかし、代替の実施形態では、レジスタ空間２６は、必ずしもこれらの異なる目的のためにこれらの別個のレジスタファイルに分割されるとは限らないことに留意されたい。代わりに、メインおよび補助実行ユニットを通じて実行される命令は、同じ共有されたレジスタファイルの中から、レジスタを指定することができる場合がある（マルチスレッドプロセッサの場合、１つのコンテキスト当たり１つのレジスタファイル）。また、パイプラ

イン１３は、命令のバンドルを同時に実行するために、必ずしも並列構成パイプライン（例えば、補助およびメインパイプライン）を含まなければならないわけではない。

【００５７】

また、プロセッサ４は、メモリ１１と１つまたは複数の他のリソース（例えば、プロセッサの他の例および／またはネットワークインタフェースもしくはネットワーク接続記憶装置（ＮＡＳ）デバイスなどの外部のデバイス）との間でデータを交換するための交換インタフェース５１も含んでもよい。図５に示されるように、実施形態では、プロセッサ４は、相互接続プロセッサタイルのアレイ６のうちの１つを形成することができ、各タイルは、より広いプログラムの一部を実行する。従って、個々のプロセッサ４（タイル）は、より広いプロセッサまたは処理システム６の一部を形成する。タイル４は、それらのそれぞれの交換インタフェース５１を介して接続される相互接続サブシステム３４を介して、共に接続することができる。タイル４は、同じチップ（すなわち、ダイ）上、異なるチップ上またはそれらの組合せ（すなわち、アレイは、各々が複数のタイル４を含む複数のチップから形成することができる）で実装することができる。従って、相互接続システム３４および交換インタフェース５１は、内部（オンチップ）相互接続メカニズムおよび／または外部（インターチップ）交換メカニズムを相応に含み得る。

【００５８】

マルチスレッドおよび／またはマルチタイルプロセッサまたはシステムの例示的なアプリケーションの１つでは、複数のスレッドおよび／またはタイル４にわたって実行されるプログラムは、ニューラルネットワークの訓練および／またはニューラルネットワークに基づく推論の実行を行うように構成されたアルゴリズムなどの機械知能アルゴリズムを含む。そのような実施形態では、各ワーカースレッド、各タイル上で実行されるプログラムの一部または各タイル上の各ワーカースレッドは、ニューラルネットワーク（グラフのタイプ）の異なるノード１０２を表すために使用され、それに従って、スレッド間および／またはタイル間の通信は、グラフのノード１０２間のエッジ１０４を表す。これは、図６に示されている。

【００５９】

機械知能は、機械知能アルゴリズムが知識モデルを学習する学習段階から始まる。モデルは、相互接続ノード（すなわち、頂点）１０２およびエッジ（すなわち、リンク）１０４のグラフを含む。グラフの各ノード１０２は、１つまたは複数の入力エッジと、１つまたは複数の出力エッジとを有する。ノード１０２のいくつかの入力エッジのうちのいくつかは、ノード１０２の他のいくつかの出力エッジであり、それにより、ノードが互いに接続され、グラフが形成される。さらに、ノード１０２の１つまたは複数の入力エッジのうちの１つまたは複数は、全体としてのグラフへの入力を形成し、ノード１０２の１つまたは複数の出力エッジのうちの１つまたは複数は、全体としてのグラフの出力を形成する。さらに、所定のノードは、グラフへの入力、グラフからの出力および他のノードとの接続のすべてを有するものであってもよい。各エッジ１０４は、値又はテンソル（ｎ次元行列）を伝達し、これらは、ノード１０２の入力エッジに提供される入力を形成するか、又は、出力エッジから提供される出力を形成する。

【００６０】

各ノード１０２は、その１つまたは複数の入力エッジにおいて受信される１つまたは複数の入力についての関数を表し、この関数の結果は、１つまたは複数の出力エッジにおいて提供される出力である。各関数は、１つまたは複数の対応するパラメータ（重みと呼ばれる場合もあるが、必ずしも乗法重みである必要はない）によってパラメータ化される。一般に、異なるノード１０２によって表される関数は、関数の異なる形態であることおよび／または異なるパラメータによってパラメータ化することが可能である。

【００６１】

さらに、各ノードの関数の１つまたは複数のパラメータの各々は、対応する誤差値によって特徴付けられる。さらに、対応する条件は、各ノード１０２のパラメータの誤差と関連付けることができる。単一のパラメータによってパラメータ化された関数を表すノード

１０２の場合、条件は、簡単な閾値であってもよく、すなわち、条件は、誤差が指定閾値内にある場合に満たされるが、誤差が閾値を超える場合には満たされない。複数の対応するパラメータによってパラメータ化されたノード１０２の場合、誤差の許容レベルに達したそのノード１０２に対する条件は、より複雑なものであってもよい。例えば、条件は、そのノード１０２のパラメータの各々がそれぞれの閾値内に収まる場合にのみ満たされ得る。別の例として、同じノード１０２に対する異なるパラメータの誤差を組み合わせた組合せ計量を定義することができ、条件は、組合せ計量の値が指定閾値内に収まるという条件によって満たされてもよいが、そうでなければ、条件は、組合せ計量の値が閾値を超える場合は満たされない（あるいは、計量の定義に応じて、その逆も同様である）。条件が何であれ、これにより、ノードのパラメータの誤差が許容度の一定のレベルまたは程度を下回るかどうかの尺度が得られる。一般に、任意の適切な計量を使用することができる。条件または計量は、すべてのノードに対して同じものでも、異なるそれぞれのノードに対して異なるものでもよい。

【００６２】

学習段階では、アルゴリズムは、経験データ（すなわち、グラフへの入力の異なる可能な組合せを表す複数のデータポイント）を受信する。多くの経験データが受信されるにつれて、アルゴリズムは、パラメータの誤差をできうる限り最小化するよう、経験データに基づいて、グラフの様々なノード１０２のパラメータを段階的に調節する。目標は、所定の入力に対してグラフの出力が所望の出力にできる限り近くなるようなパラメータの値を見出すことである。グラフが全体としてそのような状態に向かう傾向となったときに、グラフは収束したと言える。適切な収束度が得られた後、グラフは、予測または推論を実行するため（すなわち、何らかの所定の入力に対する結果を予測するかまたは何らかの所定の出力に対する原因を推論するため）に使用することができる。

【００６３】

学習段階は、多くの異なる可能な形態を取ることができる。例えば、教師ありアプローチ（supervised approach）では、入力経験データは、訓練データ（すなわち、既知の出力に対応する入力）の形態を取る。各データポイントを用いて、アルゴリズムは、出力が所定の入力に対して既知の出力により近くなるように、パラメータを調節することができる。後続の予測段階では、グラフは、入力クエリを近似予測出力にマッピングするために使用することができる（または、推論する場合は、その逆も同様である）。また、他の手法も可能である。例えば、教師なしアプローチ（unsupervised approach）では、１つの入力データ当たり１つの参照結果という概念は存在せず、代わりに、機械知能アルゴリズムは、出力データにおけるそれ自体の構造の識別を任される。あるいは、強化学習アプローチ（reinforcement approach）では、アルゴリズムは、入力経験データの各データポイントに対して少なくとも１つの可能な出力を試し、この出力が、正か否か（および、潜在的には、正または否の度合い）（例えば、勝つか負けるか、利益か損失かまたは同様のもの）が知らされる。多くの試行にわたり、アルゴリズムは、正の結果をもたらす入力を予測することができるように、グラフのパラメータを徐々に調節することができる。グラフを学習するための様々な手法およびアルゴリズムは、機械学習の当業者に知られている。

【００６４】

本明細書で開示される技法の例示的なアプリケーションによれば、各ワーカースレッドは、ニューラルネットワークなどの機械知能グラフのノード１０２のそれぞれの個々のノードと関連付けられた演算を実行するようにプログラムされる。この場合、ノード１０２間のエッジ１０４の少なくともいくつかは、スレッド間のデータの交換に相当するものであってもよいし、タイル間の交換に関与するものであってもよい。１つのタイル４当たり複数のスレッドを有するマルチタイル構成６の場合、各タイル４は、グラフのサブグラフを実行する。各サブグラフは、１つまたは複数のスーパーバイザスレッドを含むスーパーバイザサブプログラムと、それぞれのサブグラフのノード１０２を表すワーカースレッドのセットとを含む。

【００６５】

機械知能などのアプリケーションでは、プロセッサ４（例えば、タイル）のデータメモリ２２におよびデータメモリ２２からデータを効率的に流すことができることが望ましい。例えば、このことは、ベクトルドット積命令、行列積、累積ベクトルドット積、累積行列積または専用畳み込み命令（高度の算術複雑性を単一の算術命令に詰め込んだもの）などの複雑な算術命令のシーケンスを伴わせる上で特に（ただし、排他的にではない）有益であろう。

【００６６】

そのような問題に対処するため、実施形態は、プロセッサ４の命令セットにおいて、２つのロード操作および１つのストア操作を実行し、次いで、独立ストライドをロードおよびストアアドレスの各々に自動的に適用するタイプのロード／ストア命令を提供し、それらの動作はすべて、命令のオペコードの単一のインスタンスに応答して行われる。その上、命令は、適切なレジスタファイル（例えば、ＭＲＦ　２６Ｍ）における効率的なアドレスパッキングを使用し、それにより、３つのアドレス（２つのロードアドレスおよび１つのストアアドレス）が２つのレジスタの空間にパッキングされる。これにより、単一の命令において、ＭＲＦ　２６ＭからＬＳＵへの単に３つのポートのみを用いて、３つのメモリアドレスおよびストライドレジスタへのアクセス（３つのレジスタの各々へのアクセスに対してそれぞれのポート１つずつ）が可能になる。

【００６７】

命令は、本明細書では、「ｌｄ２ｘｓｔ６４ｐａｃｅ」と呼ぶことができ、これは、２つの６４ビット値をロードし、１つの６４ビット値を格納するという事実を指し、「ｐａｃｅ」は、ストライドと同義語である。しかし、このラベルまたは特定のビット幅は、必ずしも限定的なものと受け止められない。命令は、より一般的には、単にロード／ストア命令（ただし、これは、必ずプロセッサの命令セットのロード／ストア命令の唯一の種類であるということを示唆するものでない）または「ｌｄｘ２ｓｔ」（ロード×２およびストア）と呼ぶことができる。その構文は以下の通りである。

ｌｄ２ｘｓｔ　＄ａＤｓｔ，＄ａＳｒｃ，＄ｍＡｄｄｒ，＄ｍＳｔｒｉｄｅ，Ｓｔｒｉｍｍ

【００６８】

＄ａＤｓｔは、補助（または算術）レジスタファイル（ＡＲＦ）２６Ａの送信先を識別する１つまたは複数のオペランドを指し、送信先は、ＡＲＦのレジスタのうちの１つまたは複数を含む。実施形態では、ＡＲＦのレジスタの各々は、３２ビット幅であり、送信先＄ａＤｓｔは、４つのそのような３２ビットレジスタ（＄ａＤｓｔ０：Ｄｓｔ０＋３）からなり得る。ｌｄ２ｘｓｔ６４ｐａｃｅの送信先オペランドは、ＡＲＦのこれらの送信先レジスタのうちの単に１つの送信先レジスタの場所（例えば、最も低い＄ａＤｓｔ０）を指定できるのみであり、他の送信先レジスタの場所は、これに関して暗黙的であってもよい（例えば、ＡＲＦの次の３つの隣接するレジスタ）。あるいは、他の実装形態では、複数の送信先オペランドの各々が、別個のオペランドによって明示的に独立して識別できることは除外されない（ただし、これには、命令幅の増加が必要になる）。

【００６９】

＄ａＳｒｃは、ＡＲＦ　２６Ａの送信元を識別する１つまたは複数のオペランドを指し、送信元は、ＡＲＦのレジスタのうちの１つまたは複数を含む。実施形態では、ＡＲＦのレジスタの各々は、３２ビット幅であり、送信元＄ａＳｒｃは、２つのそのような３２ビットレジスタ（＄ａＳｒｃ０：Ｓｒｃ＋１）からなり得る。ｌｄ２ｘｓｔ６４ｐａｃｅの送信元オペランドは、ＡＲＦのこれらの送信元レジスタのうちの単に１つの送信元レジスタの場所（例えば、最も低い＄ａＳｒｃ０）を指定できるのみであり、他の送信元レジスタの場所は、これに関して暗黙的であり得る（例えば、ＡＲＦの次の隣接するレジスタ）。あるいは、他の実装形態では、複数の送信元オペランドの各々が、別個のオペランドによって明示的に独立して識別できることは除外されない（ただし、これには、命令幅の増加が必要になる）。

【００７０】

＄ｍＡｄｄｒは、メインレジスタファイル（ＭＲＦ）２６Ｍの２つのレジスタ（＄ｍＡｄｄｒ０：Ａｄｄｒ０＋１）の場所を指定する１つまたは複数のオペランドを指し、２つのレジスタ間では、３つのアドレス（２つのロードアドレスおよび１つのストアアドレス）が保持される。実施形態では、ＭＲＦのレジスタの各々は、３２ビット幅である。メモリアドレスオペランドは、ＭＲＦのこれらのレジスタのうちの単に１つのレジスタの場所（例えば、最も低い＄ｍＡｄｄｒ０）を指定できるのみであり、他のレジスタの場所は、これに関して暗黙的であってもよい（例えば、ＭＲＦの次の隣接するレジスタ）。あるいは、他の実装形態では、メモリアドレスレジスタの各々が、別個のオペランドによって明示的に独立して識別できることは除外されない（ただし、これには、命令幅の増加が必要になる）。

【００７１】

Ｓｔｒｉｍｍは、即値オペランドのセットであり、それぞれの即値オペランドは、２つのロードアドレスおよび１つのストアアドレスの各々に対するものである。ＭＲＦ　２６Ｍでは、多数のフィールドを含むストライドレジスタが提供される。各フィールドは、異なる可能なストライド値の既定のセットの異なる対応するストライド値を保持する。ストライド値は、メモリアドレスをどの程度インクリメントするかを示す値（すなわち、メモリアドレスステップ）であり、典型的には、一連のステップにおいて使用するためのものである。２つのロードおよび１つのストアの各々に対し、対応する即値ストライドオペランドは、命令の現在のインスタンスの対応するロードまたはストア操作を実行した後に、対応するロードまたはストアアドレスに適用するためにストライドレジスタのどのフィールドからストライド値を取り入れるかを指定する。この動作は、ロード／ストア命令の後続のインスタンスの利益のために、ＭＲＦ　２６Ｍのアドレスに沿って移動する。

【００７２】

実施形態では、ロード／ストアユニット５５は、ある特徴をサポートし、その特徴により、即値ストライドオペランドの１つの特別な値は、レジスタフィールドを指し示すというよりむしろ、１のストライド値（すなわち、使用されるアドレス空間における１アドレス単位の増加）を直接指定することができる。すなわち、即値ストライドオペランドが取りうる値のうち、１つは、１のストライドを指定し、他の値のいくつかまたはすべては、プログラム可能なストライド値を保持しているストライドレジスタの異なる可能なフィールドを指定する。ここでの１単位は、データアクセスのアトミックサイズを意味する。例えば、オペランドが１６ビット浮動小数点値の４要素ベクトルである場合は、インクリメントは、１単位当たり１アトムであり、それは、８バイト（６４ビット）に等しい。

【００７３】

ストライドレジスタは、ＭＲＦ　２６Ｍのレジスタである。実施形態では、ワーカー自体は、それ自体のストライドレジスタ（それ自体のＭＲＦ）のフィールドのストライド値を書き込むことに対する責任を有する。あるいは、他の実装形態では、ストライドレジスタのフィールドのストライド値がスーパーバイザスレッドＳＶによって書き込めること、または、手法の組合せを使用できることは除外されない。

【００７４】

オペランドによる＄ｍＳｔｒｉｄｅの指定に対する要件は、実装形態に応じたプロセッサ４の任意の特徴である。実施形態では、ロード／ストア命令は、ＭＲＦ　２６Ｍのストライドレジスタ＄ｍＳｔｒｉｄｅの場所を指定するオペランドを取る。従って、プログラムは、多数の可能なストライドレジスタの中からストライドレジスタを選択し、ストライドの選択におけるさらなる一層の柔軟性を提供することができる。しかし、代替の実装形態では、＄ｍＳｔｒｉｄｅの場所を固定するかまたは暗黙的にし、ロード／ストア命令のオペランドを必要としないようにすることができることは除外されない。

【００７５】

図７は、３つのアドレスをＭＲＦ　２６Ｍの２つの３２ビットレジスタにパッキングする例を示し、各アドレスは、２１ビットである。また、図７は、３つのストライド値をＭＲＦの１つの３２ビットレジスタにパッキングする例も示し、各ストライド値は、１０ビ

ットである（この事例では、レジスタの２ビットは未使用である）。従って、ロード／ストア命令は、単一の命令において、ＭＲＦの単に２つの３２ビット幅ポートを介して、３つのメモリアドレスにアクセスすることができる。第３の３２ビット幅ポートは、同じ命令によって、ストライドレジスタにアクセスするために使用することができ、ストライドレジスタは、例えば、ＭＲＦの３２ビット幅ストライドレジスタにパッキングされた３つの１０ビット幅フィールドの３つの１０ビットストライド値（アドレスデルタ）を含む（そのレジスタの２ビットは未使用のままである）。図７に示される特定のパッキングは単なる例であることに留意されたい。例えば、別の実装形態では、ａｄｄｒ１というよりむしろ、ａｄｄｒ２が２つのレジスタをまたぐことができる。

【００７６】

図８は、ＭＲＦ　２６Ｍにおける複数のストライドレジスタ＄ｍＳｔｒｉｄｅＡ、＄ｍＳｔｒｉｄｅＢ、＄ｍＳｔｒｉｄｅＣ…の構成を示す。これらはいずれも、ロード／ストア命令の対応するオペランドによって、ストライドレジスタ＄ｍＳｔｒｉｄｅとして指定することができる。可能なストライドレジスタの各々は、多数のフィールドを含み、例えば、実施形態では、３つのフィールドを含む（ただし、実装形態に応じて、必ずしもロードおよびストアアドレスのものと同じ数とは限らないフィールドやストライドレジスタが存在する）。２つのロードアドレスおよび１つのストアアドレスの各々に対し、即値Ｓｔｒｉｍｍのセットのそれぞれの即値オペランドは、それぞれのストライド値（それぞれのロードまたはストアに続いて適用されるアドレスデルタ）を取り入れるためのストライドレジスタ＄ｍＳｔｒｉｄｅの可能なフィールドのうちの１つを指定することができる。例えば、実施形態では、ストライドオペランドが２つのロードおよび１つのストアの各々に対して２ビットである場合は、可能な値のうちの３つは、３つのストライドフィールドのうちの異なるストライドフィールドを指定し、他の可能な値は、レジスタに保持されている値を参照することなく、単に、１のストライドを指定するのみである。例えば、００は、１（アトム）のストライドを指定し、０１は、ストライドレジスタの第１のフィールドを指定し、１０は、ストライドレジスタの第２のフィールドを指定し、１１は、ストライドレジスタの第３のフィールドを指定することになる。

【００７７】

実施形態では、即値ストライドの可能な値のうちの１つは、ストライドレジスタ＄ｍＳｔｒｉｄｅのフィールドというよりむしろ、１のデフォルトインクリメントを指定することに留意されたい。

【００７８】

動作の際、実行ユニット１８によって実行されると、ロード／ストア命令のオペコード（復号ステージ１６によって復号された後）は、以下の動作を実行するようにＬＳＵ　５５をトリガする。ＬＳＵ　５５は、ＭＲＦ　２６Ｍの＄ｍＡｄｄｒに保持されるアドレスによって指定されるメモリ２２の２つのロードアドレスから、＄ａＤｓｃによって指定されるＡＲＦ　２６Ａの送信先に、値をロードする。さらに、ＬＳＵ　５５は、ＡＲＦの＄ａＳｒｃから、ＭＲＦの＄ｍＡｄｄｒに保持されたアドレスによって指定されるメモリ２２のストアアドレスに、値をロードする。次いで、ＬＳＵ　５５は、３つの即値オペランドＳｔｒｉｍｍのそれぞれの即値オペランドによって指定されるようなＭＲＦのストライドレジスタ＄ｍＳｔｒｉｄｅのそれぞれのフィールドからのそれぞれのストライド値の分だけ、２つのロードアドレスおよび１つのストアアドレスの各々を独立してポストインクリメントする。

【００７９】

注：ロード／ストア命令の現在のインスタンスの２つのロードアドレスおよび１つのストアアドレスの各々に対し、現在の命令のストライド操作は、現在のロード／ストア操作のロードおよびストア操作の各々に続いてそれぞれ、それぞれのストライドの分だけそれぞれのアドレスをインクリメントする。実装形態に応じて、このことは、すべてのインクリメントがロードとストアの両方の後に共に適用されること（例えば、ロード、ロード、ストア、インクリメント、インクリメント、インクリメント）を意味し得る。あるいは、

ストライド操作は、各ロードおよびストアの直後に、それぞれのアドレスをインクリメントすることができる（例えば、ロード、インクリメント、ロード、インクリメント、ストア、インクリメント）。実際に、ストアは、ロードのうちの一方または両方の前に来ることもできる。重要なことは、単に、各ロードアドレスのインクリメントがそのそれぞれのロードの後に行われること、そして、ストアアドレスのインクリメントがストアの後に行われることである。肝心なことは、ロード／ストア命令の後続のインスタンスに備えて、ＭＲＦのロードおよびストアアドレスに沿って移動することである。

【００８０】

実施形態では、メモリ１１（または少なくともデータメモリ２２）は、メモリからデータをロードするための単に２つの６４ビットポートと、メモリにデータを格納するための単に１つの６４ビット幅ポートとを有するのみである。実施形態では、ＭＲＦ　２６Ｍ（所定のコンテキストの）は、ロード／ストアユニット５５への単に３つの３２ビット幅ポートを有するのみであり、ＡＲＦ　２６Ａ（所定のコンテキストの）は、ロード／ストアユニット５５への単に１つの６４ビット幅ポートを有するのみである（注：示される例では、ＩＡＬＵ　５６は、ＭＲＦ　２６Ｍからポインタを回収し、ＬＳＵ　５５に渡すためにこれらのポインタからアドレスを演算するために使用され、従って、事実上、ＩＡＬＵ　５６は、ＬＳＵ　５５の一部として動作する。従って、この例では、ＭＲＦ　２６ＭからＬＳＵ　５５への３つのポートは、ＭＲＦ　２６ＭからＩＡＬＵ　５６への３つのポートを含む。また、他の実装形態では、ＬＳＵ　５５は、ＭＲＦ　２６Ｍから直接、ポインタを回収し、そのポインタに基づいてそれ自体のアドレスを演算できることも除外されない）。

【００８１】

実施形態では、ＡＲＦの４つの３２ビット送信先レジスタ＄ａＤｓｔ０：Ｄｓｔ０＋３（合計で１２８ビット）は、メモリ２２から、例えば、１６ビット浮動小数点（ｆ１６）値の４要素ベクトルおよび３２ビット浮動小数点（ｆ３２）値の２要素ベクトルをロードするために使用することができる。ＡＲＦの２つの３２ビット送信元レジスタ＄ａＳｒｃ０：Ｓｒｃ＋１（合計で６４ビット）は、メモリ２２に、例えば、ｆ３２値の３２ビットの２要素ベクトルを格納するために使用することができる。

【００８２】

プログラムにおいて使用するため、ロード／ストア命令は、算術命令などの他のタイプの命令の中に散在し、算術命令は、ロード命令の送信先から入力を取り入れ、その入力に基づいて演算を実行し、ロード／ストア命令の送信元に結果を出力する。すなわち、プログラムは、ロード／ストア命令のインスタンスと、少なくとも１つの算術命令（以前に言及されたベクトルドット積命令、行列積命令、累積ベクトルドット積命令、累積行列積命令または畳み込み命令など）のインスタンスとを含み、ロード／ストア命令のうちの少なくともいくつかのロード／ストア命令の送信先は、算術命令のうちの少なくともいくつかの算術命令の送信元であり、算術命令のうちの少なくともいくつかの算術命令の送信先は、ロード／ストア命令のうちの少なくともいくつかのロード／ストア命令の送信元である。ロード／ストア命令の高い意味的密度に起因して、その２つのロードおよびストライド機能により、プログラムは、ロードおよびストア操作に費やされるコードオーバーヘッドが小さな状態で、効率的にデータを処理することができる。

【００８３】

例えば、プログラムは、一連の複数対の命令を含んでもよく、命令の各対は、ロード／ストア命令のインスタンスと、それに続く算術命令の対応するインスタンスとからなる。命令の各対では、ロード／ストア命令の送信元は、先行する対からの算術命令の送信先として設定され、ロード／ストア命令の送信先は、現在の対または後続の対の対応する算術命令の送信元として設定される。例えば、以下の構文を有する「ａｒｉｔｈ」という算術命令を考える。

ａｒｉｔｈ　＄ａＤｓｔ，＄ａＳｒｃＡ，＄ａＳｒｃＢ

【００８４】

構文中、＄ａＤｓｔは、ＡＲＦ　２６Ａの送信先を指定するオペランドを指し、＄ａＳｒｃＡ、＄ａＳｒｃＢは、ＡＲＦの２つの送信元を指定するオペランドを指す（「ａｒｉｔｈ」は、少なくともこの構文を有する算術命令のためにここで使用される一般名であることが理解されよう）。次いで、プログラムは、例えば、以下のような一連の複数対の命令によってプログラムすることができる。

・・・　・・・
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｒｉｔｈ　Ｐｏｕｔ，Ｘｉｎ，Ｐｉｎ；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｒｉｔｈ　Ｐｏｕｔ，Ｘｉｎ，Ｐｉｎ；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｒｉｔｈ　Ｐｏｕｔ，Ｘｉｎ，Ｐｉｎ；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｒｉｔｈ　Ｐｏｕｔ，Ｘｉｎ，Ｐｉｎ；
・・・　・・・
または
・・・　・・・
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ＿Ａ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｒｉｔｈ　Ｐｏｕｔ＿Ａ，Ｘｉｎ，Ｐｉｎ；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ＿Ｂ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｒｉｔｈ　Ｐｏｕｔ＿Ｂ，Ｘｉｎ，Ｐｉｎ；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ＿Ａ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｒｉｔｈ　Ｐｏｕｔ＿Ａ，Ｘｉｎ，Ｐｉｎ；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ＿Ｂ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｒｉｔｈ　Ｐｏｕｔ＿Ｂ，Ｘｉｎ，Ｐｉｎ；
・・・　・・・
など。

【００８５】
実施形態では、各対は、命令バンドルであり、すなわち、下流のそれぞれのパイプラインにおいて同時に実行される。例えば、実施形態では、ロード／ストア命令は、ＭＲＦ　２６Ｍを使用してメインパイプラインによって実行され、算術命令は、ＡＲＦ　２６Ａを使用して補助パイプラインによって並列して実行される。しかし、代替の実現形態では、単一のパイプライン処理装置を通じてロード／ストアおよび算術命令を次々と実行できることは除外されない。

【００８６】
ロード／ストア命令（ｌｄｘ２ｓｔ）のインスタンスでは、Ｘｉｎ－Ｐｉｎは、ＡＲＦ　２６Ａの２つのロード操作の送信先＄ａＤｓｃを指し（例えば、４つの３２ビットレジスタ＄ａＤｓｃ０：Ｄｓｃ０＋３にロードされる４×ｆ１６入力ベクトルＸｉｎおよび２×ｆ３２入力ベクトルＰｉｎ）、Ｐｏｕｔ　ＡまたはＰｏｕｔ　Ｂは、ＡＲＦのストアの送信元を指す（例えば、２つの３２ビットレジスタ＄ａＳｒｃ０：Ｓｒｃ０＋１から格納される２×ｆ３２値）。「Ｔｒｉｐａｃｋｅｄ」は、ＭＲＦ　２６Ｍのトリプルパッキングアドレスレジスタ対＄ｍＡｄｄｒ０：Ａｄｄｒ０＋１を指す（例えば、２つの３２ビットレジスタに３つの２１ビットアドレスポインタを保持する）。「Ｓｔｒｉｄｅｓ」は、３つの即値Ｓｔｒｉｍｍによって指定されたＭＲＦのストライドレジスタ＄ｍＳｔｒｉｄｅ内のフィールドを参照することによって２つのロード操作および１つのストア操作の各々に対するストライドを指定するストライドオペランド＄ｍＳｔｒｉｄｅおよびＳｔｒｉｍｍを指す（再び図８を参照）。算術命令のインスタンスでは、Ｘｉｎ、Ｐｉｎは、算術命令の入力を取り入れる対象となる送信元レジスタを指し、その送信元レジスタは、現在のまたは以前の命令対のロード／ストア命令によって実行されたロードの送信先と同じになるように設定される。Ｐｏｕｔは、ＡＲＦの算術命令の出力の送信先を指し、その送信

先は、後続の命令対のロード／ストア命令のストア操作の送信元と同じになるように設定される。２つの同時命令のバンドルとして各対を実行する実施形態では、Ｘｉｎ、Ｐｉｎは、以前のバンドルのロード／ストア命令によって実行されたロードの送信先と同じになるように設定され、Ｐｏｕｔは、次のバンドルのｌｄｘ２ｓｔのストア操作の送信元と同じになるように設定される。この実施形態は、同じレジスタを再利用するが、２つの命令が並列して実行されるため、現在の算術命令に対する入力データは、以前のロード命令によって読み取られたものである。次いで、現在のｌｄｘ２ｓｔは、次の算術命令に対する入力データを準備するために、それらの同じレジスタを再利用する。

【００８７】

従って、各対では、ロード／ストア（ｌｄｘ２ｓｔ）は、以前の命令の以前の対（例えば、バンドル）の以前の算術命令から現在見出されている出力をレジスタＰｏｕｔに格納し、また、メモリからレジスタＸｉｎおよびＰｉｎに値をロードする。同じ対（例えば、バンドル）の次の算術命令は、以前にロードされた値に基づいて算術演算を実行し、後続の対の後続のロード／ストア命令のストア操作によって格納するためにレジスタＰｏｕｔに出力する。

【００８８】

実施形態では、算術命令は、共有されたＷＲＦ　２６Ｗの少なくとも第３の暗黙的な送信元を有することに留意されたい。これらの送信元は、ＷＲＦに書き込むスーパーバイザスレッドによって設定することができる。従って、スーパーバイザは、すべての実行ワーカースレッドの算術演算によって暗黙的に使用される共通の重みを設定する。

【００８９】

また、ＭＲＦ　２６Ｍの＄ｍＡｄｄｒ０：Ａｄｄｒ０＋１を指定するメモリアドレス「Ｔｒｉｐａｃｋｅｄ」の値は、それぞれのワーカーによっていくつかの初期値に設定される。また、ＭＲＦの可能なストライドレジスタ＄ｍＳｔｒｉｄｅの各々のフィールドのストライド値は、それぞれのワーカーによって設定される。ロード／ストア命令が実行される度、メモリアドレスレジスタ＄ｍＡｄｄｒ０：Ａｄｄｒ０＋１内の３つのアドレス（２つのロードアドレスおよび１つのストアアドレス）の各々は、それぞれのストライド値の分だけポストインクリメントされる。再び図７および８を参照すること。例えば、仮に、ロード／ストア命令が＄ｍＳｔｒｉｄｅ＝＄ｍＳｔｒｉｄｅＡおよびＳｔｒｉｍｍ＝０１１０００（第１の即値ストライドオペランド＝０１、第２の即値ストライドオペランド＝１０および第３の即値ストライドオペランド＝００）と指定するとする。これは、第１のロードを実行した後、＄ｍＳｔｒｉｄｅＡのフィールド０のストライド値の分だけ第１のロードアドレスがインクリメントされ、第２のロードを実行した後、＄ｍＳｔｒｉｄｅＡのフィールド１のストライド値の分だけ第２のロードアドレスがインクリメントされ、格納後、アトミックインクリメントの分だけストアアドレスがインクリメントされる（使用されているアドレス空間の１メモリアドレスステップずつインクリメントされる）ことを意味する。ロード／ストア命令の後続のインスタンスは、＄ｍＳｔｒｉｄｅの同じまたは異なる値および３つの即値オペランドの各々の同じまたは異なる値を指定するように設定することができる。従って、一連の命令の各ロード／ストア命令では、ロードおよびストアの場所は制御可能な方法で移動することができる。従って、コードは、新しいアドレスを演算するための追加の整数演算命令を必要とすることなく、入力データ空間を通じて柔軟な方法で効率的に走査することができる。

【００９０】

図９は、上記の実施形態が有益であり得る例示的なアプリケーション、すなわち、入力データＸｉｎの多次元部分とカーネルＫとの畳み込みの実行を示す。例えば、入力データは、３Ｄボリュームの入力データであってもよく、カーネルＫは、３Ｄカーネルであってもよい。また、入力データは、入力チャネルと呼ぶこともでき、出力データは、出力特徴チャネルと呼ぶことができる。データが多次元であると言われる場合は、これは、データが二次元以上の座標系の座標の多数の組合せの各々における値を取ることを意味し、各次元（すなわち、各軸）は、異なる独立変数を表す（そして、座標の所定の組合せにおける

データの値は、従属変数である）。

【００９１】

これの例は、各データ値がピクセルを表す画像処理におけるものである。例えば、典型的には、赤、緑および青の各々に対して８ビットが使用され、次いで、それらの８ビット整数値は、畳み込み演算の前にｆ１６値に変換される。各Ｒ、Ｇ、Ｂ値は、別個の入力平面（すなわち、別個の入力チャネル）として扱われる。従って、各２Ｄ入力画像は、ｘ、ｙ、ｚ次元を有し、Ｒ、Ｇ、Ｂ平面は、ｚ軸に存在する。データのボリュームは、例えば、フレームのシーケンスを含んでもよく、ボリュームの２つの次元は、フレームエリアのｘ、ｙ空間軸を表し、第３の次元は、シーケンスの異なるフレームに対応する時間軸を表す。別の例では、ボリュームの３つの次元は、空間または物体の３Ｄモデルまたは画像の３つのｘ、ｙ、ｚ空間次元を表してもよい。別の例では、データは、静止画像のｘ、ｙ空間軸を表す単なる二次元であってもよく、ｚ軸は、同時に処理される入力画像の数を表す。開示される技法の適用性は、画像またはグラフィックス処理に限定されず、より高い次元のデータも除外されないことに留意されたい。例えば、二次元、三次元またはそれ以上の次元は、ユーザ挙動などを学習するアプリケーションにおいてユーザまたはそれらの状況の異なる態様を表してもよい。

【００９２】

カーネルＫは、既定の重みの行列である。カーネルＫは、ニューラルネットワークが認識のために訓練されているかまたは後続の推論において検索されているある特徴を表してもよい。例えば、画像データの場合、カーネルは、エッジまたは他の何らかの特定の種類の形状を表してもよい。機械学習アプリケーションでは、ニューラルネットワークの各ノードは、それぞれのカーネルに相当してもよい（ただし、後にさらに論じられるように、多くのニューラルネットワークでは、同じカーネル値を使用するが、単に異なる接続１０４を有する複数のノード１０２が存在する）。実施形態では、各ノード１０２は、それぞれのワーカースレッドによって表され、従って、各スレッドは、そのスレッドのそれぞれの入力データとカーネルＫとのそれぞれの畳み込みを実行する。ワーカースレッドの少なくともいくつかは、共有の重みレジスタファイル（ＷＲＦ）２６Ｗにその値が格納されている同じカーネルＫを使用することができるが、そのようなスレッドの各々は、それ自体のそれぞれの補助（または算術）レジスタファイル（ＡＲＦ）２６Ａを介してメモリ２２からおよびメモリ２２に流れるスレッド自体のそれぞれの入力データにそのカーネルを畳み込む。

【００９３】

三次元の離散畳み込み（＊）は、以下のように表すことができる。

$$Xin * K[x, y, z] = \Sigma_{x', y', z'} \left(K[x', y', z'] \, Xin[x - x', y - y', z - z'] \right)$$

従って、畳み込みは、入力データにカーネルを重ね合わせることができる可能な位置のいくつかまたはすべてにわたってカーネルＫをシステマチックに走査することを含む。そのような位置の各々では、カーネルと重なり合う入力データの部分の各ピクセル（またはデータポイント）と、対応するオーバーレイポイントにおけるカーネルの値とが乗算され、これらの個々の乗算の各々の結果は加算され、その特定のオーバーレイ位置における入力データとカーネルとの畳み込みを表すスカラ出力が得られる。次いで、この動作は、出力データ（１つのカーネル位置当たり１つのスカラ値）の対応するボリュームを得るために、入力データＸｉｎに対するカーネルの異なる走査位置で繰り返される。例えば、図９に示される例では、カーネルＫを入力データＸｉｎのボリューム内の一隅に置いて開始することができ、各カーネル値とそのカーネルがその位置にあるときにそのカーネルが重なり合うデータポイントとが乗じられる。次いで、カーネルは、ボリュームの長さに沿って１ステップ移動し、この新しい位置におけるカーネルを用いてポイント乗算および加算が再び実行されるなど、以下同様である。ボリュームの全長に沿ってカーネルが走査されると、カーネルは新しい座標位置にシフトされ、全ボリュームが走査されるまで、長さ方向走査が再び実行されるなど、以下同様である。所定の各位置におけるカーネルは、出力デ

ータの１つのピクセル（１つのスカラ値）を生成する。従って、出力データＸｏｕｔは、畳み込み走査のステップサイズに応じて、入力データと同じまたは同様のサイズを有することになる。

【００９４】

実際には、この動作は、より小さな動作に分割する必要がある。このことは、所定の位置におけるデータとカーネルとの畳み込みを多くの個々のベクトルドット積および部分和に分けることによって行うことができる。このことは、図９の中央および右側に示されている。仮に、例えば、カーネルのサイズが３×３×１６ピクセルであるとする（「ピクセル」は、本論考のために、必ずしも画像を指すとは限らない）。すなわち、カーネルは、３×３×１６の既定の重み値からなる。このカーネルは、９つの１６要素ベクトルＣＷ（本明細書では、カーネルＫの構成カーネルとも呼ばれる）に分割することができる。最初に、これらの９つのベクトルのうちの１つと対応する位置における入力データとの間でドット積（すなわち、スカラ積）が取られ、部分和Ｐ（スカラ）が得られる。次いで、９つのベクトルのうちの別のものとの間でドット積が取られ、第１の部分和と累計して、第２の部分和（この場合も、スカラ）が得られる。次いで、９つのベクトルのうちの別のものとの間でドット積が取られ、この結果を第２の部分和と累計して、第３の部分和が得られるなど、以下同様である。この動作が終了し、９つのベクトルのすべてを累計した時点で、総累計結果は、出力データＸｏｕｔにおける単一のポイントまたはピクセルの値を与える。次いで、この動作は、入力データＸｉｎのボリュームに対する異なる位置にわたってカーネルＫが走査されるにつれて、各カーネル位置で繰り返され、従って、出力データＸｏｕｔ（１つのカーネル位置当たり１つの出力データポイント）の対応するボリュームが得られる。

【００９５】

実施形態では、畳み込みなどの演算を効率的な方法で実行することを支援するため、プロセッサ４の命令セットは、累積積（「ａｍｐ」）命令の形態の算術命令を含む。その命令は、以下の構文を有する。

ａｍｐ　＄ａＤｓｔ，＄ａＳｒｃＡ，＄ａＳｒｃＢ，Ｐｈａｓｅ

構文中、＄ａＤｓｔは、この場合もやはり、ＡＲＦ　２６Ａの送信先を指定するオペランドを指し、＄ａＳｒｃＡ、＄ａＳｒｃＢは、ＡＲＦの２つの送信元を指定するオペランドを指す。「Ｐｈａｓｅ」は、累積の位相を指定する即値オペランドである。例えば、実施形態では、位相オペランドは、４つの可能な位相０・・・３のうちの１つを指定する２ビットで形成される。実施形態では、ａｍｐ命令は、ｆ１６ｖ４ｓｉｓｏａｍｐと呼ぶことができ、これは、第１の入力として４つの半精度（１６ビット）浮動小数点値の１つのベクトルを取り入れ、第２の入力として１つの単精度（３２ビット）浮動小数点値を取り入れ、１つの単精度（３２ビット）浮動小数点値を出力するという事実を指す（「ｓｉｓｏ」は、「単精度入力、単精度出力」を指す）。実際に、実施形態では、第２の入力は、２つの単精度浮動小数点値（２×３２ビット）の対であり、出力も同様である。しかし、これらの特定の精度の値および要素の数は、すべての可能な実施形態に制限されないことが理解されよう。

【００９６】

ａｍｐなどの１つまたは複数のタイプの命令に対し、実施形態では、ＷＲＦの重みの場所は、完全に暗黙的である。その代替としてまたはそれに加えて、１つまたは複数の他のタイプの命令は、ＷＲＦの少数の異なるセットの中からどの重みのセットを使用するかを指定する追加のオペランド（図示せず）を取ることができる。例えば、その例は、後に言及されるスリムな畳み込み（ｓｌｉｍ　ｃｏｎｖｏｌｕｔｉｏｎ）（「ｓｌｉｃ」）命令であり得る。

【００９７】

以前に説明されたロード／ストア命令（ｌｄ２ｘｓｔ）と組み合わせると、ａｍｐ命令などの命令は、メモリからデータを効率的に流し、ドット積および部分和を実行し、部分和をメモリに戻すために使用することができる。例えば、以下のような４つの命令対のル

ープを経るシーケンスを含むプログラムを考慮する。
Ｌｏｏｐ　{
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｍｐ　Ｐｏｕｔ，Ｘｉｎ，Ｐｉｎ，Ｐｈａｓｅ＝０；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｍｐ　Ｐｏｕｔ，Ｘｉｎ，Ｐｉｎ，Ｐｈａｓｅ＝１；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｍｐ　Ｐｏｕｔ，Ｘｉｎ，Ｐｉｎ，Ｐｈａｓｅ＝２；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｍｐ　Ｐｏｕｔ，Ｘｉｎ，Ｐｉｎ，Ｐｈａｓｅ＝３；
}

【００９８】

別の例示的なバージョンは、以下の通りである。

Ｌｏｏｐ　{
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ＿Ａ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｍｐ　Ｐｏｕｔ＿Ａ，Ｘｉｎ，Ｐｉｎ，Ｐｈａｓｅ＝０；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ＿Ｂ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｍｐ　Ｐｏｕｔ＿Ｂ，Ｘｉｎ，Ｐｉｎ，Ｐｈａｓｅ＝１；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ＿Ａ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｍｐ　Ｐｏｕｔ＿Ａ，Ｘｉｎ，Ｐｉｎ，Ｐｈａｓｅ＝２；
ｌｄｘ２ｓｔ　Ｘｉｎ－Ｐｉｎ，Ｐｏｕｔ＿Ｂ，Ｔｒｉｐａｃｋｅｄ，Ｓｔｒｉｄｅｓ；ａｍｐ　Ｐｏｕｔ＿Ｂ，Ｘｉｎ，Ｐｉｎ，Ｐｈａｓｅ＝３；
}
など。

【００９９】

この場合もやはり、実施形態では、各対は、命令バンドルであり、すなわち、下流のそれぞれのパイプライン（例えば、メインおよび補助パイプライン）において同時に実行される。

【０１００】

命令の各対は、ロード／ストア命令のインスタンスおよびそれに続く累積積（ａｍｐ）命令のインスタンスを含む。ａｍｐ命令の送信元は、同じまたは先行する命令対のロード／ストア命令によって実行された２つのロードの送信先である。２つの同時命令のバンドルとして各対を実行する実施形態では、ａｍｐの送信元（Ｘｉｎ、Ｐｉｎ）は、以前のバンドルにおいて実行されたロードの送信先と同じになるように設定される。ａｍｐ命令の送信先は、後続の命令対（例えば、次のバンドル）においてロード／ストア命令によって実行される１つのストアの送信元である。ａｍｐ命令の送信元＄ａＳｒｃＡ、＄ａＳｒｃＢのうちの一方は、入力データＸｉｎから入力ベクトルｘを取り入れるために使用される。他方は、部分和を取り入れるために使用される。ａｍｐ命令は、実行されると、その入力ベクトルｘと重みレジスタファイルＷＲＦ　２６Ｗからの重みＣＷの対応するベクトル（このベクトルＣＷは、全体または３Ｄカーネルｋの構成カーネルである）とのドット積を実行する。位相オペランドは、ドット積の結果を累計するための位相を指定する。ループの所定のインスタンスにおけるシーケンスの命令の各対では、位相オペランドは、シーケンスの異なる連続位相を指定する異なるそれぞれの値に設定される。位相オペランドによって指定されるようなシーケンスのこれらの連続位相にわたり、ａｍｐ命令の効果は、連続ドット積の各々の結果を累計することである。入力部分和との累計は、シーケンスの第１の位相において始まる。

【０１０１】

ロード／ストア命令のストライド機能により、プログラムは、ＭＲＦ　２６Ｍに含めるための新しいメモリアドレスを演算するための別個の整数演算命令を必要とすることなく、メモリアドレスに沿って、命令の各対を有する次のデータ片に自動的にシフトすること

ができる。ストライドレジスタ＄ｍＳｔｒｉｄｅから多くの事前にプログラムされたストライドのうちのいずれかを選択できるとの上記ロード／ストア命令の能力は、図９において例示として示されるものなどの多次元データを取り扱う上で特に有益である。メモリ空間は、一次元であるが、データは、二次元、三次元またはそれ以上の次元を有し得る。多次元データ空間への１Ｄメモリ空間のマッピングに応じて、カーネルＫが入力データＸｉｎを通じて走査されるにつれて、メモリ空間の対応するメモリアドレスを通じて異なるサイズのステップを取る必要がある場合がある。例えば、カーネルＫが入力データＸｉｎの一次元に沿って長さ方向に走査されるにつれて、各走査位置におけるデータのメモリアドレスは、アトミックステップでインクリメントするが、垂直平面において走査をシフトする必要がある際は、入力データＸｉｎの次のデータ片またはサブセットに到達するために、メモリアドレスの異なるサイズのステップが必要とされる（そして、独立して、第２の入力および出力アドレスに対して同じプロセスが行われる）場合がある。１つのストライドレジスタ＄ｍＳｔｒｉｄｅまたは複数のストライドレジスタ（例えば、＄ｍＳｔｒｉｄｅＡ、＄ｍＳｔｒｉｄｅＢ、・・・）の異なるフィールドは、有利には、メモリ空間の異なるサイズのジャンプで事前にプログラムすることができる。次いで、プログラマまたはコンパイラは、ロード／ストア命令別にどのジャンプサイズを使用するかを選択するために、即値オペランドＳｔｒｉｍｍを設定することができる（次のアドレスまたは次のステップサイズを再計算するための別個の整数演算命令を必要とすることなく）。

【０１０２】

実施形態では、ａｍｐ命令は、累積行列積命令である。この命令の動作については、図１０～１２を参照して説明する。図１０に示されるように、ａｍｐ命令は、ＦＰＵによって実行されると、入力データＸｉｎからのＮ要素入力ベクトルと重みのＭ×Ｎ行列（ＷＲＦ　２６Ｗから）との乗算を実行する。実施形態では、Ｍ＝８およびＮ＝１６である。行列のＭ個の行の各々は、異なる対応するカーネルＫ０、Ｋ１…ＫＭ－１からの構成ベクトルに相当することに留意されたい。従って、ａｍｐ命令は、入力データとＭ個の異なるカーネルＫとの並列畳み込みのコンポーネントを実行するために使用される。例えば、これらのカーネルの各々は、入力データと異なるそれぞれの特徴（例えば、異なるエッジまたは形状）との畳み込みに相当してもよい。畳み込みニューラルネットワークの場合、それらのカーネルは、特徴フィルタであり、Ｍ個のカーネルの各々は、入力層における異なる特徴の存在の検出を試みる。所定のカーネルＫに対し、ａｍｐ命令は、１つのドット積および累積のみを実行する。しかし、行列は、データと複数のカーネルとの効率的な並列畳み込みを行えるようにする。

【０１０３】

Ｎ要素入力ベクトルは、Ｎ１個のセグメントに分けられ、その各々は、Ｎ２個の要素のサブベクトルを含む（従って、Ｎ＝Ｎ１×Ｎ２）。実施形態では、Ｎ１＝４およびＮ２＝４である。ａｍｐ命令の位相オペランドは、Ｎ１個の異なるセグメントに対応するＮ１個の異なる可能な値０…Ｎ１－１のうちの１つを取り入れる。ループを経るシーケンスの各インスタンスでは、各連続対のａｍｐ命令は、位相０…Ｎ１－１のうちの異なる連続位相を指定する位相オペランドの異なる値を取る。各位相は、Ｎ要素入力ベクトルのＮ１個のサブベクトルのうちの対応するものと行列のＭ個の行の各々とのドット積（従って、Ｍ個の異なるカーネルＫの各々に対する演算の一部）を実行する。実行ユニット１８（実施形態では、補助または算術実行ユニット１８Ａ）のＦＰＵは、Ｍ個のアキュムレータ状態＄ＡＡＣＣ（１スレッド当たり）を有する。実施形態では、これらのアキュムレータ状態＄ＡＡＣＣは、ＦＰＵの内部状態５７として実装される。しかし、代替の実装形態では、これらのアキュムレータ状態＄ＡＡＣＣを複数のレジスタファイル２６（例えば、それぞれのスレッドのＡＲＦ　２６Ａ）のうちの１つのレジスタファイル２６のレジスタとして実装できることは除外されない。示される例では、これらのアキュムレータ状態＄ＡＡＣＣは、偶数番号が付けられた状態＄ＡＡＣＣ［０］、＄ＡＡＣＣ［２］…＄ＡＡＣＣ［１４］であり、奇数番号が付けられた状態＄ＡＡＣＣ［１］、＄ＡＡＣＣ［３］…＄ＡＡＣＣ［１５］は、シーケンスの異なるループ間で値を伝播するために使用することができるが

、これは、実装形態の詳細であり、これに限定されない。

【０１０４】

図１１は、ａｍｐ命令の動作を図に示す。図１２は、一連のａｍｐ命令の例を与える。各々は、上記で示されるように、命令対の対応するロード／ストア命令が先行すると理解されている。図１２では、P_nは、単精度入力部分和ｎであり、x_nは、ｆ１６ｖ４入力ベクトルであり、$CW_{m,n}$は、共通の重み状態＄ＣＷＥＩ＿ｍ＿ｎであり、R_nは、P_nで始まる連続ドット積累積の最終的な単精度結果である。

【０１０５】

動作の際、ａｍｐ命令のオペコード（復号済みの）に応答して、実行ユニット１８（実施形態では、補助実行ユニット１８Ａ）のＦＰＵは、以下の動作を実行する。

・　ａｍｐ命令の送信元オペランドのうちの１つによって指定されたＡＲＦ　２６Ａのレジスタから、Ｍ／Ｎ１個の数の部分和を取り入れ、これらの部分和を次のループに対するプロパゲータ状態に一時的に置く。このプロパゲータ状態は、ＦＰＵの内部状態５７で実装することができるか、または、代替の実装形態では、レジスタファイル２６（例えば、ＡＲＦ）のうちの１つのレジスタファイル２６の別のレジスタであり得る。各位相は、Ｍ／Ｎ１個の部分和の異なるサブセットを取り入れる。Ｎ１個の位相にわたり、Ｍ個のそのような部分和は、プロパゲータ状態にされる。従って、Ｍ＝８およびＮ１＝４の場合は、各ａｍｐ命令は、入力として２つの部分和（例えば、２つのｆ３２値）を取り入れ、これらをプロパゲータ状態に保持し、すべての４つの位相にわたり、８つの部分和が受信される。

【０１０６】

・　ａｍｐ命令の送信元オペランドのうちの１つによって指定されたＡＲＦ　２６Ａのレジスタから、入力ベクトルＸｉｎのＮ２要素セグメント（サブベクトル）を取り入れる。行列のＭ個の行の各々（従って、Ｍ個のカーネルＫ０…ＫＭ－１の各々）に対し、この入力ベクトルＸｉｎのＮ２要素セグメントとＷＲＦからの重みＣＷの対応するサブベクトルとのドット積を実行する。どの列から重みのサブベクトルが取り入れられるかは、位相オペランドの値に依存する。さらに、位相がシーケンスの第１の（すなわち、初期の）位相である場合は、部分和のうちの対応するものがＭ個のドット積の各々に加えられる。以前のループの各ａｍｐ命令はそれぞれのＭ／Ｎ１個（Ｎ１は、位相の数）の部分和を取り入れたため、以前のループは、現在のループに備えてすべてのＭ個の部分和をプロパゲータ状態に置いたことになることに留意されたい。

【０１０７】

・　Ｍ個の行の各々に対し、上記の計算の結果は、Ｍ個のアキュムレータ状態＄ＡＡＣＣのうちの対応する状態に既にある任意の値と累計され（総和が求められ）、それによって、Ｍ個の結果Ｒ０・・・ＲＭ－１のうちの対応するものを生成する。従って、Ｎ１個の位相にわたり、ａｍｐ命令は、完全なＮ要素入力ベクトルとＷＲＦからの重みの対応するＮ要素ベクトルとのドット積を実行する。

【０１０８】

・　現在のループでは、以前のループからのＭ／Ｎ１個の数の結果Ｒは、ａｍｐ命令の送信先オペランドによって指定されたＡＲＦの送信先レジスタに出力される。各位相は、Ｍ／Ｎ１個の部分和の異なるサブセットを出力し、Ｎ１個の位相にわたり、Ｍ個のそのような部分和がプロパゲータ状態にされる。従って、Ｍ＝８およびＮ１＝４の実施形態では、１つのａｍｐ命令当たり２つの結果（例えば、２つのｆ３２）が出力され、すべての４つの位相にわたってすべての８つの結果Ｒ０・・・Ｒ７が出力される。

【０１０９】

・　現在のループの最後の位相の後（または結果が上書きされる前の次のループの第１の位相の開始時）、現在のループの結果Ｒは、次のループの出力に備えて、プロパゲータ状態に一時的に置かれる。このプロパゲータ状態は、ＦＰＵの内部状態５７で実装することができるか、または、代替の実装形態では、レジスタファイル２６（例えば、ＡＲＦ）のうちの１つのレジスタファイル２６の別のレジスタであってもよい。

【０１１０】
要約すると、単一の部分和は、（１）第１のドット積結果に加えられ、アキュムレータ状態（以前に保持された値を上書きする）で格納され、（２）さらなる３つのドット積結果と累計され、（３）一時的なプロパゲータ状態にコピーされる（その場合、その値は、今後のａｍｐ命令の送信先オペランドに書き込まれる）。ステップ（３）の代替手段は、アキュムレータからそのまま得られた結果として生じた部分和をａｍｐ命令の送信先オペランドに書き込むことである。いずれの方法にせよ、ステップ（１）および（３）は、ハードウェアにおいて重複してもよく、４サイクルループを提供する。
【０１１１】
実施形態では、アキュムレータ状態は、明示的にリセットする必要はない。各行列ベクトル乗算の出発点はメモリから読み取られた入力部分和であるため、出力ボリュームは、畳み込みの開始時に、アキュムレータ状態をリセットするというよりむしろ、すべての要素がゼロに設定される。そうは言っても、実施形態では、タイル４は、実際に、１から０にアキュムレータ状態のすべてを初期化することができる。
【０１１２】
現在のａｍｐ命令の入力プロパゲータ状態は、次のループでの使用のために保持されることに留意されたい。また、ａｍｐ命令によってＡＲＦの送信先レジスタに出力される結果は、現在のａｍｐの累積演算が適用される前の既存のアキュムレータ状態である（すなわち、累積の以前の短いシーケンスの結果であり、現在の入力を含まない）。
【０１１３】
プログラムは、結果Ｒが後続のループに対する部分和入力Ｐになるように構成される（プログラマまたはコンパイラによって）。その正確な関係は、プログラマまたはコンパイラにかかっており、データおよびカーネルの構造に依存する。典型的には、部分和は、０の初期値（メモリ内の）を有する。
【０１１４】
ループ間の時間差により、シーケンスは、ウォームアップ期間を必要とすることに留意されたい。提示されている特定の入力ベクトルとａｍｐ命令によって返されている４位相計算の結果との間には待ち時間がある。その待ち時間は固定されており、これは、最初のＭ個の結果が無効であるか、関心のないものであるかまたは意味のないものであり、ウォームアップコードにおいて処理されることを意味する。有益な結果が生み出されないこのウォームアップ期間の後、コードは繰り返しループに入り、それにより、先に進み、有益な結果が必要なメモリ場所に書き込まれる（示される例では、一度に６４ビット）。また、内側のループの後にはクールダウン期間もあり、クールダウン期間では、新しい入力は提供されないが、最終的な出力値が格納される。図１２は、ｆ１６ｖ４ｓｉｓｏａｍｐに対するウォームアップ期間および「結果待ち時間」を示す。
【０１１５】
実施形態では、カーネルの数はＭ＝８であり、入力ベクトルの要素の数はＮ＝１６であり、位相の数はＮ１＝４である。実施形態では、入力ベクトルの各要素は、ｆ１６値であり、各入力部分和および各結果は、ｆ３２値である。そのような実施形態では、上記のプロセスは、１つの命令対当たり２つのｆ３２部分和および入力ベクトルの４つのｆ１６要素のレートで、データをロードおよび処理することができ、１つの命令対当たり２つのｆ３２値のレートで、データを出力および格納することができる。それは、１つの命令対当たり１２８ビットの入力および６４ビットの出力である。これは、ＬＳＵ　５５とデータメモリ２２との間の２つの６４ビット幅ロードポートおよび１つの６４ビット幅ストアポートに相当する（再び図４を参照）。また、アドレスのトリプルパッキングのため、プロセスは、ＭＲＦ　２６ＭからＬＳＵへの２つの３２ビット幅ポートおよびストライドに対する別のポートを介して処理することができる。２つの６４ビット幅ポートは、ＡＲＦ２６Ａから補助実行ユニット１８ＡのＦＰＵへと使用される。図１２は、各１６要素入力ベクトルに対し、ループが８個の出力値（１つのカーネル当たり１つの出力値）を生成することを示す。この特定のシナリオでは、それらの出力値は、各々が３２ビットである。

上記のループは、４ティック（ｔｉｃｋ）ごとに１６個の入力値（命令ごとに４×ｆ１６の入力値）を処理する。従って、その入力レートを維持するため、プロセッサ４は、４ティックループごとに８×３２ビット値を生成しなければならず、それは、１つのループ当たり４×６４ビットストアまたは１ティック当たり１×６４ビットストアを意味する。

【０１１６】

重みＣＷは、共有されたＷＲＦ　２６Ｗから取り入れられ、各スレッドに対して同じである。ニューラルネットワークなどのアプリケーションでは、異なるデータと重みの同じセットとを乗じる（畳み込みの一部など）必要がある多くのシナリオがあるということが分かる。例えば、ニューラルネットワークのいくつかのノードは、全く同じ重みを含むが、異なる接続を含む場合がある。例えば、各カーネルＫ（ｍ＝０・・・Ｍ）は、入力データ（例えば、グラフィック画素のエリアまたはボリューム）と特徴（例えば、エッジまたは形状）との畳み込みを表すことができる。従って、これらのカーネルに対応するワーカースレッドには、別個の重みは不要である。代わりに、スーパーバイザが所有する重み状態の単に１つのコピーが提供されるのみである。従って、共有の重みレジスタファイルからの共有されたスレッドオペランドの使用により、有利には、重みに対するレジスタ空間が少なくて済む。図１２は、各１６要素入力ベクトルに対しループが８個の出力値（１つのカーネル当たり１つの出力値）を生成することを示す。この特定のシナリオでは、それらの出力値は、各々が３２ビットである。上記のループは、４つのサイクルごとに１６個の入力値（命令ごとに４×ｆ１６の入力値）を処理する。従って、その入力レートを維持するため、プロセッサ４は、４サイクルループごとに８×３２ビット値を生成し、それは、１つのループ当たり４×６４ビットストアまたは１つのサイクル当たり１×６４ビットストアを意味する。

【０１１７】

実施形態では、奇数アキュムレータ＄ＡＡＣＣ［１］、＄ＡＡＣＣ［３］・・・＄ＡＡＣＣ［１５］は、結果（Ｒｘ）をずらすために使用され、部分和入力（Ｐｘ）にも同じことをするために使用される。部分入力は、一度に２つずつ後方のエンジン、すなわち、アキュムレータレジスタ＄ＡＡＣＣ［１３］および＄ＡＡＣＣ［１５］に供給される。各サイクルでは、それらの値は、前方に移動し、この場合には、＄ＡＡＣＣ［１１］および＄ＡＡＣＣ［９］へと移動する。それらの値が送信先ＡＭＰユニットに到達すると、それらの値は、累積に備えて、奇数番号が付けられたアキュムレータというよりむしろ、偶数番号が付けられたアキュムレータに移動する。従って、｛＄ＡＡＣＣ［１５］，＄ＡＡＣＣ［１３］｝－＞｛＄ＡＡＣＣ［１１］，＄ＡＡＣＣ［９］｝－＞｛＄ＡＡＣＣ［７］，＄ＡＡＣＣ［５］｝－＞｛＄ＡＡＣＣ［２］，＄ＡＡＣＣ［０］｝となる。

【０１１８】

図１３および１４は、本明細書で説明されるタイプのロード／ストア命令（すなわち、それぞれのスレッドのＡＲＦ　２６Ａを介して流された入力データに共通の重みレジスタファイル（ＷＲＦ）２６Ｗからの重みの２Ｄセットを畳み込む畳み込み命令）から恩恵を得ることができる別のタイプの算術命令を示す。実施形態では、この命令は、半精度浮動小数点ベクトルのスリムな畳み込み（ｓｌｉｃ命令）の形態を取ることができ、例えば、「ｆ１６ｖ４ｓｌｉｃ」は、ｆ１６値の４要素サブベクトルに基づいて演算するという事実を指す。この命令は、１つの半精度および１つの単精度入力を有し、単精度出力を有する。ｓｌｉｃはａｍｐと同様であり、ａｍｐと同じハードウェアを使用する。しかし、この命令は、８つのカーネルの各々に対する１×１×１６累積ドット積というよりむしろ、２つのカーネルの各々に対する１×４×４畳み込み（真の畳み込みである）を実行する。実際に、実施形態では、この命令は、１×Ｎ×４畳み込み（Ｎは、２、３または４である）を実行するように構成することができる。

【０１１９】

累積行列積（「ａｍｐ」）およびスリムな畳み込み（「ｓｌｉｃ」）命令は、ワーカーコンテキストにわたる重み共有が適切なシナリオに対して、高性能の乗累算シーケンスを容易にする。実施形態では、ａｍｐおよびｓｌｉｃ命令は、単精度数フォーマットと半

精度数フォーマットの両方をサポートし、同じ基本的な方法で動作することができる。共通の演算構成状態（ＷＲＦ　２６Ｗの共有の重みを含む）は、最初に、スーパーバイザコンテキストによって初期化される。次いで、入力データの２つのストリーム、すなわち、入力起動（ピクセル）データおよび後続の乗累算シーケンスに対する初期値を指定する部分和値（畳み込みの場合、これらは、部分的に演算された出力ピクセル／起動である）が処理される。出力データ（結果として生じた累計済み部分和値）の単一のストリームが生成される。各部分和入力値は、最終的な部分和結果が出力として示される前に、固定長シーケンスの乗累算の対象となる。多くのドット積および累算は、演算エンジンによって実行され、並列して発生する。第１の被乗数は、入力データストリームによって提供され、第２の被乗数は、共通の演算構成状態によって提供される。

【０１２０】

以下の表は、いくつかの例示的なａｍｐおよびｓｌｉｃ命令の変形形態をリストし、実施形態では、それらのいずれかまたはすべては、プロセッサの命令セットに含めることができる。

【０１２１】

【表１】

命令	入力データフォーマット	入力部分和 フォーマット	出力部分和 フォーマット
f16v4sisoamp	半精度 (v4)	単精度 (v2)	単精度 (v2)
f16v4hihoamp	半精度 (v4)	半精度 (v2)	半精度 (v2)
f16v4sihoamp	半精度 (v4)	単精度 (v2)	半精度 (v2)
f32sisoamp	単精度 (スカラ)	単精度 (v2)	単精度 (v2)
f16v4sisoslic	半精度 (v4)	単精度 (v2)	単精度 (v2)
f16v4hihoslic	半精度 (v4)	半精度 (v2)	半精度 (v2)
f16v4sihoslic	半精度 (v4)	単精度 (v2)	半精度 (v2)
f32sisoslic	単精度 (スカラ)	単精度 (v2)	単精度 (v2)

注:いくつかのタイプの命令では、重みの場所は完全に暗黙的であるが、他のタイプでは、多くの異なるセットのうちのどれをＷＲＦにおいて使用するかを選択するオペランドを取る。f16v4sisoslic命令は、例えば、f16v4sisoampの重み状態の4分の1のみを必要とする。その結果、ソフトウェアは、最大で４セットの重みをプリロードすることができる。重みセット選択は、即値オペランドの２ビットによって指定される。他方では、f16v4sisoampの場合は、完全な重み状態を取り入れ、従って、その中での選択はない。

【０１２２】

上記の実施形態は単なる例示として説明されていることが理解されよう。

【０１２３】

例えば、本開示の範囲は、上記で説明されるアーキテクチャであり、そのアーキテクチャでは、スーパーバイザスレッドに対して別個のコンテキストが提供されるか、または、スーパーバイザスレッドは、スロットで実行し、次いで、そのスロットをワーカーに委ねる。スーパーバイザは、代わりに、汎用コンテキストを使用することができる。あるいは、別の構成では、例えば、スーパーバイザは、それ自体の専用スロットで動作できる。さらに、実装形態は、スーパーバイザの役割さえも有するスレッドのうちの特定のものに限定されない。さらに、本開示の範囲は、タイルのアレイにおけるタイルであるプロセッサ

４に限定されない。代替の実施形態では、プロセッサ４は、例えば、スタンドアロンプロセッサでも、単一チッププロセッサでもあってもよい。

【０１２４】

実行ユニット１８の実装形態は、別個のメイン実行ユニット１８Ｍおよび補助実行ユニット１８Ａにも、別個のＭＲＦおよびＡＲＦにも限定されない。一般に、レジスタは、１つまたは複数のレジスタファイルのいずれからのものでもあってもよく、異なるメモリアクセスと算術演算との間で共有することも、分離することもできる。

【０１２５】

共有の重みレジスタファイルを使用する算術命令は、行列積または畳み込み命令に限定されない。より一般的には、共有の重みレジスタファイルは、いかなるタイプの算術命令または命令の組合せに対しても使用することができ、例えば、別個の命令もしくは汎用コードにおいて累積が行われる非累積行列積命令、または、以前の積の結果が必ずしも後続の積の入力であるとは限らない行列乗算（畳み込み以外）の他のシーケンスに対して使用することができる。他の例は、ベクトルドット積または累積ベクトルドット積命令（すなわち、Ｍ個の異なるカーネルを並列して適用する必要がない算術命令）を含んでもよい。

【０１２６】

さらに、本開示の範囲は、例示として上記で開示される、特定のレジスタサイズ、ポートビット幅、ポートの数、値精度、ベクトルサイズまたは行列サイズに限定されない。他のレジスタおよびポートビット幅が可能であり、他の値の精度および他のベクトルまたは行列サイズ（例えば、要素の数の観点から）も同様に可能である。

【０１２７】

さらに、他の実装形態では、２つのレジスタへの２つのロードアドレスおよび１つのストアアドレスの他のパッキング（例えば、２つの１６ビットレジスタへの３つの１０ビットアドレス、または、２つの６４ビットレジスタへの３つの４２ビットアドレス）が可能である。また、ロードおよびストアアドレスは、必ずしも同じ長さである必要はない。さらに、代替の実装形態では、ストライドレジスタへのストライド値の他のパッキング（例えば、３２ビットレジスタへの４つの８ビットストライドフィールドなど）も可能である。

【０１２８】

さらに、さらなる変形形態では、本開示の適用性は、画像処理に限定されない。「カーネル」という用語は、本明細書で使用される場合は、ベクトル乗算、行列乗算、畳み込みまたは他のもの（例えば、相関）などの演算を含む任意のプロセスの一部として適用されるいかなる重みの行列も意味してもよい。また、アプリケーション能力は、３Ｄボリュームのデータに限定されない。入力データ（およびカーネル）は、線形でも、２Ｄでも、より高い多次元性（３より大きい独立変数または自由度）を有するものでもあってもよい。さらに、本開示の範囲は、機械学習アプリケーションに限定されない。並列スレッド間で共通の１つまたは複数のオペランドと各スレッドに特有の１つまたは複数のオペランドとの組合せに基づいて動作を実行することが望ましい場合がある他の多くのアプリケーションが存在する。

【０１２９】

開示される技法の他の変形形態または使用事例は、本明細書の本開示が与えられた時点で当業者に明らかになるであろう。本開示の範囲は、説明される実施形態による制限は受けず、添付の請求項による制限のみを受ける。

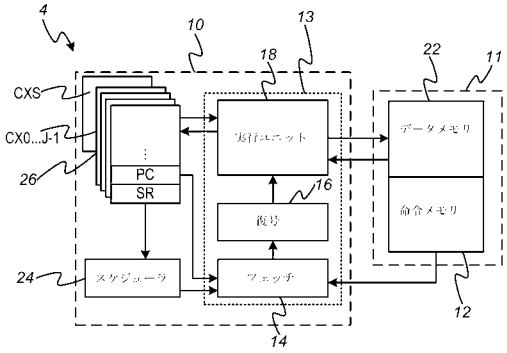
【符号の説明】

【０１３０】

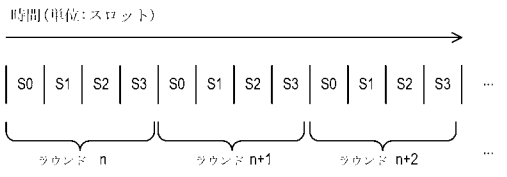
４　プロセッサ
６　タイルのアレイ
１０　マルチスレッド処理装置
１１　ローカルメモリ
１２　命令メモリ

１３　実行パイプライン
１４　フェッチステージ
１６　復号ステージ
１８　実行ユニット
１８Ａ　補助実行ユニット
１８Ｍ　メイン実行ユニット
２２　データメモリ
２４　スケジューラ
２６　コンテキストレジスタファイル
２６Ａ　補助レジスタファイル
２６ＣＳＲ　制御状態レジスタ
２６Ｍ　メインレジスタタファイル
２６Ｗ　共通の重みレジスタファイル
３４　相互接続システム
５１　交換インタフェース
５３　命令バッファ
５５　ロード／ストアユニット
５６　整数演算論理ユニット
５７　内部状態
１０２　ノード
１０４　エッジ

【図１】

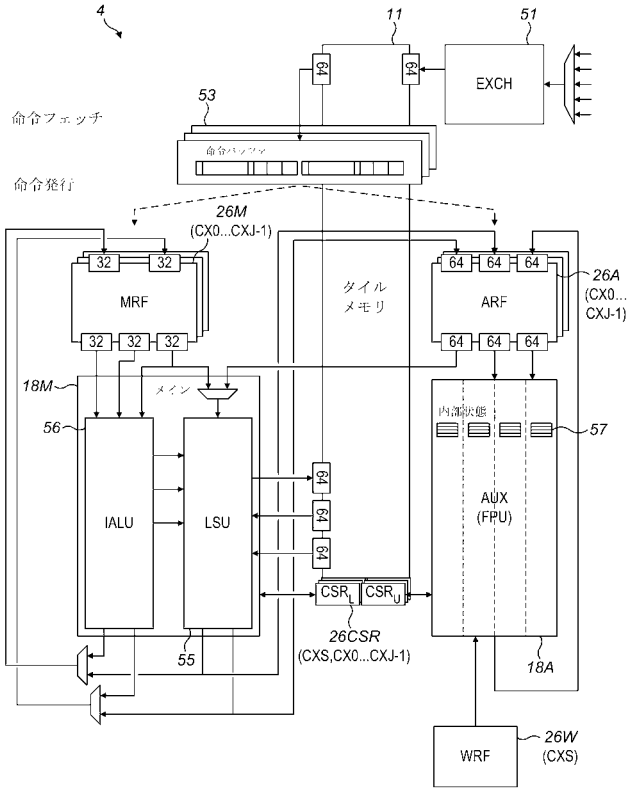


【図２】

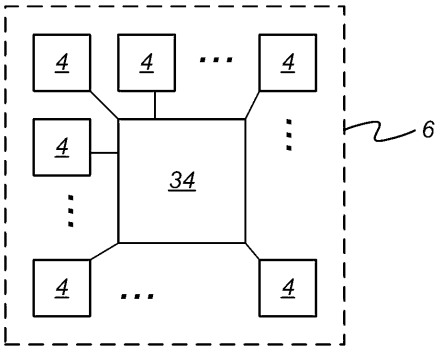


【図３】

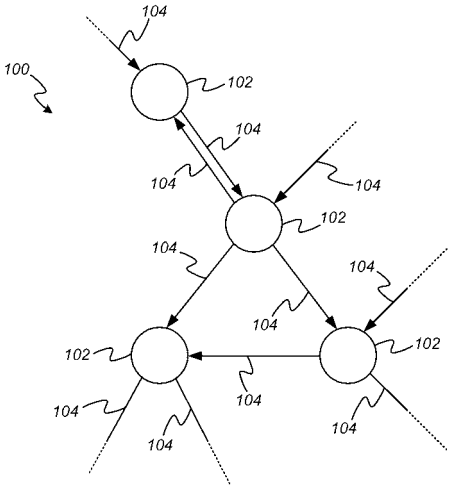
【図４】



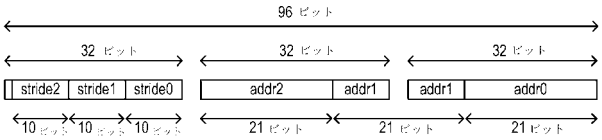
【図５】



【図６】



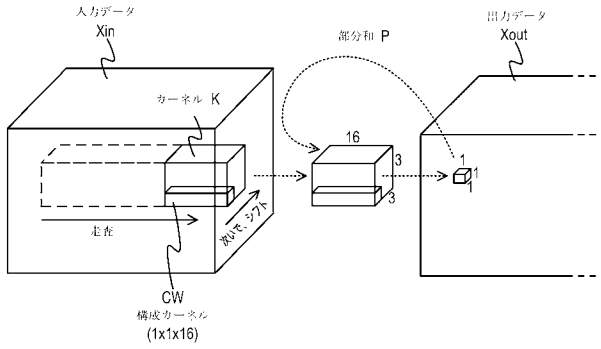
【図７】



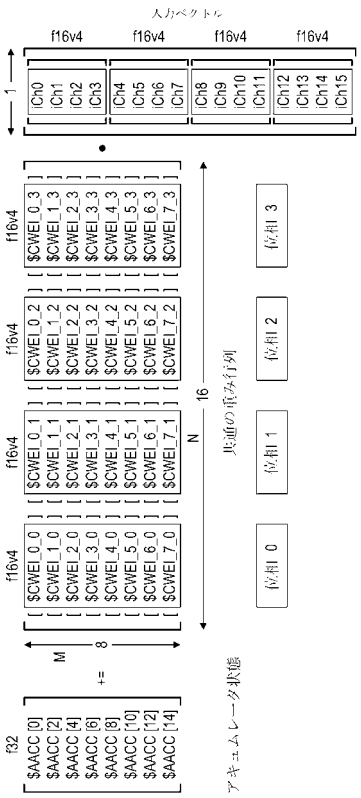
【図８】

$mStrideA	strideA2	strideA1	strideA0
$mStrideB	strideB2	strideB1	strideB0
$mStrideC	strideC2	strideC1	strideC0
⋮		⋮	

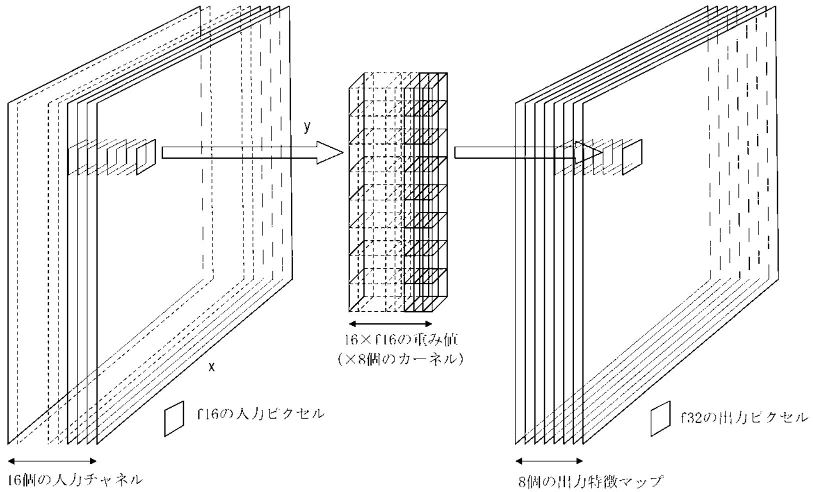
【図９】



【図１０】



【図１１】



【図１２－１】

$aSrc0|1	$AACC[14]	$AACC[12]	$AACC[10]	$AACC[8]
$0^3\|P_0,P_1$	-	-	-	-
$0\|P_2,P_3$	-	-	-	[WARM UP
$0\|P_4,P_5$	-	-	-	-
$0\|P_6,P_7$	-	-	-	-
$x_0\|P_8,P_9$	$R_7=x_0.CW_{7,0}+P_7$	$R_6=x_0.CW_{6,0}+P_6$	$R_5=x_0.CW_{5,0}+P_5$	$R_4=x_0.CW_{4,0}+P_4$
$x_1\|P_{10},P_{11}$	$R_7=x_1.CW_{7,1}$	$R_6=x_1.CW_{6,1}$	$R_5=x_1.CW_{5,1}$	$R_4=x_1.CW_{4,1}$
$x_2\|P_{12},P_{13}$	$R_7=x_2.CW_{7,2}$	$R_6=x_2.CW_{6,2}$	$R_5=x_2.CW_{5,2}$	$R_4=x_2.CW_{4,2}$
$x_3\|P_{14},P_{15}$	$R_7=x_3.CW_{7,3}$	$R_6=x_3.CW_{6,3}$	$R_5=x_3.CW_{5,3}$	$R_4=x_3.CW_{4,3}$
$x_4\|P_{16},P_{17}$	$R_{15}=x_4.CW_{7,0}+P_{15}$	$R_{14}=x_4.CW_{6,0}+P_{14}$	$R_{13}=x_4.CW_{5,0}+P_{13}$	$R_{12}=x_4.CW_{4,0}+P_{12}$
$x_5\|P_{18},P_{19}$	$R_{15}=x_5.CW_{7,1}$	$R_{14}=x_5.CW_{6,1}$	$R_{13}=x_5.CW_{5,1}$	$R_{12}=x_5.CW_{4,1}$
$x_6\|P_{20},P_{21}$	$R_{15}=x_6.CW_{7,2}$	$R_{14}=x_6.CW_{6,2}$	$R_{13}=x_6.CW_{5,2}$	$R_{12}=x_6.CW_{4,2}$
$x_7\|P_{22},P_{23}$	$R_{15}=x_7.CW_{7,3}$	$R_{14}=x_7.CW_{6,3}$	$R_{13}=x_7.CW_{5,3}$	$R_{12}=x_7.CW_{4,3}$
$x_8\|P_{24},P_{25}$	$R_{31}=x_8.CW_{7,0}+P_{23}$	$R_{22}=x_8.CW_{6,0}+P_{22}$	$R_{21}=x_8.CW_{5,0}+P_{21}$	$R_{20}=x_8.CW_{4,0}+P_{20}$
$x_9\|P_{26},P_{27}$	$R_{23}=x_9.CW_{7,1}$	$R_{22}=x_9.CW_{6,1}$	$R_{21}=x_9.CW_{5,1}$	$R_{20}=x_9.CW_{4,1}$
$x_{10}\|P_{28},P_{29}$	$R_{23}=x_{10}.CW_{7,2}$	$R_{22}=x_{10}.CW_{6,2}$	$R_{21}=x_{10}.CW_{5,2}$	$R_{20}=x_{10}.CW_{4,2}$
$x_{11}\|P_{30},P_{31}$	$R_{23}=x_{11}.CW_{7,3}$	$R_{22}=x_{11}.CW_{6,3}$	$R_{21}=x_{11}.CW_{5,3}$	$R_{20}=x_{11}.CW_{4,3}$
$x_{12}\|P_{32},P_{33}$	$R_{31}=x_{12}.CW_{7,0}+P_{31}$	$R_{30}=x_{12}.CW_{6,0}+P_{30}$	$R_{29}=x_{12}.CW_{5,0}+P_{29}$	$R_{28}=x_{12}.CW_{4,0}+P_{28}$

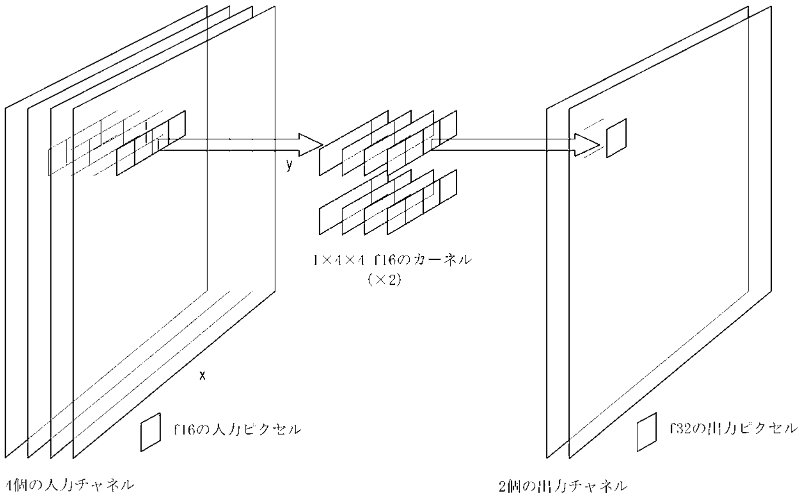
P_nは、単精度入力部分和nである。
x_nは、f16v4入力ベクトルである。
$CW_{m,n}$は、共通の重み状態$CWEI_m_nである。
R_nは、P_nで始まる連続ドット積累積の最終的な単精度結果である。

【図１２－２】

$AACC[6]	$AACC[4]	$AACC[2]	$AACC[0]	$aDstO
-	-	-	-	-
PERIOD]	-	-	-	-
-	-	-	-	-
-	-	-	-	-
$R_3=x_0.CW_{3,0}+P_3$	$R_2=x_0.CW_{2,0}+P_2$	$R_1=x_0.CW_{1,0}+P_1$	$R_0=x_0.CW_{0,0}+P_0$	-
$R_3=x_1.CW_{3,1}$	$R_2=x_1.CW_{2,1}$	$R_1=x_1.CW_{1,1}$	$R_0=x_1.CW_{0,1}$	-
$R_3=x_2.CW_{3,2}$	$R_2=x_2.CW_{2,2}$	$R_1=x_2.CW_{1,2}$	$R_0=x_2.CW_{0,2}$	-
$R_3=x_3.CW_{3,3}$	$R_2=x_3.CW_{2,3}$	$R_1=x_3.CW_{1,3}$	$R_0=x_3.CW_{0,3}$	-
$R_{11}=x_4.CW_{3,0}+P_{11}$	$R_{10}=x_4.CW_{2,0}+P_{10}$	$R_9=x_4.CW_{1,0}+P_9$	$R_8=x_4.CW_{0,0}+P_8$	R_0,R_1
$R_{11}=x_5.CW_{3,1}$	$R_{10}=x_5.CW_{2,1}$	$R_9=x_5.CW_{1,1}$	$R_8=x_5.CW_{0,1}$	R_2,R_3
$R_{11}=x_6.CW_{3,2}$	$R_{10}=x_6.CW_{2,2}$	$R_9=x_6.CW_{1,2}$	$R_8=x_6.CW_{0,2}$	R_4,R_5
$R_{11}=x_7.CW_{3,3}$	$R_{10}=x_7.CW_{2,3}$	$R_9=x_7.CW_{1,3}$	$R_8=x_7.CW_{0,3}$	R_6,R_7
$R_{19}=x_8.CW_{3,0}+P_{19}$	$R_{18}=x_8.CW_{2,0}+P_{18}$	$R_{17}=x_8.CW_{1,0}+P_{17}$	$R_{16}=x_8.CW_{0,0}+P_{16}$	R_8,R_9
$R_{19}=x_9.CW_{3,1}$	$R_{18}=x_9.CW_{2,1}$	$R_{17}=x_9.CW_{1,1}$	$R_{16}=x_9.CW_{0,1}$	R_{10},R_{11}
$R_{19}=x_{10}.CW_{3,2}$	$R_{18}=x_{10}.CW_{2,2}$	$R_{17}=x_{10}.CW_{1,2}$	$R_{16}=x_{10}.CW_{0,2}$	R_{12},R_{13}
$R_{19}=x_{11}.CW_{3,3}$	$R_{18}=x_{11}.CW_{2,3}$	$R_{17}=x_{11}.CW_{1,3}$	$R_{16}=x_{11}.CW_{0,3}$	R_{14},R_{15}
$R_{27}=x_{12}.CW_{3,0}+P_{27}$	$R_{26}=x_{12}.CW_{2,0}+P_{26}$	$R_{25}=x_{12}.CW_{1,0}+P_{25}$	$R_{24}=x_{12}.CW_{0,0}+P_{24}$	R_{16},R_{17}

図１２　続き

【図１３】



【図１４－１】

$aSrc0|1	$AACC[14]	$AACC[10]	$AACC[6]	$AACC[2]
$x_0\|P_0,P_1$	-	$R_1=x_0.CW_{5,0}+P_1$	-	-
$x_1\|P_2,P_3$	-	$R_3=x_1.CW_{5,0}+P_3$	$R_1=x_1.CW_{3,0}$	-
$x_2\|P_4,P_5$	-	$R_5=x_2.CW_{5,0}+P_5$	$R_3=x_2.CW_{3,0}$	$R_1=x_2.CW_{1,0}$
$x_3\|P_6,P_7$	-	$R_7=x_3.CW_{5,0}+P_7$	$R_5=x_3.CW_{3,0}$	$R_3=x_3.CW_{1,0}$
$x_4\|P_8,P_9$	-	$R_9=x_4.CW_{5,0}+P_9$	$R_7=x_4.CW_{3,0}$	$R_5=x_4.CW_{1,0}$
$x_5\|P_{10},P_{11}$	-	$R_{11}=x_5.CW_{5,0}+P_{11}$	$R_9=x_5.CW_{3,0}$	$R_7=x_5.CW_{1,0}$
$x_6\|P_{12},P_{13}$	-	$R_{13}=x_6.CW_{5,0}+P_{13}$	$R_{11}=x_6.CW_{3,0}$	$R_9=x_6.CW_{1,0}$

表3.156:f16v4sisoslic、1x4のシーケンス例。

$aSrc0|1	$AACC[14]	$AACC[10]	$AACC[6]	$AACC[2]
$x_0\|P_0,P_1$	$R_1=x_0.CW_{7,0}+P_1$	-	-	-
$x_1\|P_2,P_3$	$R_3=x_1.CW_{7,0}+P_3$	$R_1=x_1.CW_{5,0}$	-	-
$x_2\|P_4,P_5$	$R_5=x_2.CW_{7,0}+P_5$	$R_3=x_2.CW_{5,0}$	$R_1=x_2.CW_{3,0}$	-
$x_3\|P_6,P_7$	$R_7=x_3.CW_{7,0}+P_7$	$R_5=x_3.CW_{5,0}$	$R_3=x_3.CW_{3,0}$	$R_1=x_3.CW_{1,0}$
$x_4\|P_8,P_9$	$R_9=x_4.CW_{7,0}+P_9$	$R_7=x_4.CW_{5,0}$	$R_5=x_4.CW_{3,0}$	$R_3=x_4.CW_{1,0}$
$x_5\|P_{10},P_{11}$	$R_{11}=x_5.CW_{7,0}+P_{11}$	$R_9=x_5.CW_{5,0}$	$R_7=x_5.CW_{3,0}$	$R_5=x_5.CW_{1,0}$
$x_6\|P_{12},P_{13}$	$R_{13}=x_6.CW_{7,0}+P_{13}$	$R_{11}=x_6.CW_{5,0}$	$R_9=x_6.CW_{3,0}$	$R_7=x_6.CW_{1,0}$

P_nは、単精度入力部分和nである。
x_nは、f16v4入力ベクトルである。
$CW_{m,n}$は、共通の重み状態$CWEI_m_nである。
R_nは、P_nで始まる連続ドット積累積の最終的な単精度結果である。

【図１４－２】

$AACC[12]	$AACC[8]	$AACC[4]	$AACC[0]	$aDstO
-	$R_0=x_0.CW_{4,0}+P_0$	-	-	-
-	$R_2=x_1.CW_{4,0}+P_2$	$R_0=x_1.CW_{2,0}$	-	-
-	$R_4=x_2.CW_{4,0}+P_4$	$R_2=x_2.CW_{2,0}$	$R_0=x_2.CW_{0,0}$	-
-	$R_6=x_3.CW_{4,0}+P_6$	$R_4=x_3.CW_{2,0}$	$R_2=x_3.CW_{0,0}$	R_0,R_1
-	$R_8=x_4.CW_{4,0}+P_8$	$R_6=x_4.CW_{2,0}$	$R_4=x_4.CW_{0,0}$	R_2,R_3
-	$R_{10}=x_5.CW_{4,0}+P_{10}$	$R_8=x_5.CW_{2,0}$	$R_6=x_5.CW_{0,0}$	R_4,R_5
-	$R_{12}=x_6.CW_{4,0}+P_{12}$	$R_{10}=x_6.CW_{2,0}$	$R_8=x_6.CW_{0,0}$	R_6,R_7

$AACC[12]	$AACC[8]	$AACC[4]	$AACC[0]	$aDstO
$R_0=x_0.CW_{6,0}+P_0$	-	-	-	-
$R_2=x_1.CW_{6,0}+P_2$	$R_0=x_1.CW_{4,0}$	-	-	-
$R_4=x_2.CW_{6,0}+P_4$	$R_2=x_2.CW_{4,0}$	$R_0=x_2.CW_{2,0}$	-	-
$R_6=x_3.CW_{6,0}+P_6$	$R_4=x_3.CW_{4,0}$	$R_2=x_3.CW_{2,0}$	$R_0=x_3.CW_{0,0}$	-
$R_8=x_4.CW_{6,0}+P_8$	$R_6=x_4.CW_{4,0}$	$R_4=x_4.CW_{2,0}$	$R_2=x_4.CW_{0,0}$	R_0,R_1
$R_{10}=x_5.CW_{6,0}+P_{10}$	$R_8=x_5.CW_{4,0}$	$R_6=x_5.CW_{2,0}$	$R_4=x_5.CW_{0,0}$	R_2,R_3
$R_{12}=x_6.CW_{6,0}+P_{12}$	$R_{10}=x_6.CW_{4,0}$	$R_8=x_6.CW_{2,0}$	$R_6=x_6.CW_{0,0}$	R_4,R_5

図１４ 続き

フロントページの続き

(72)発明者　アラン　グラハム　アレクサンダー
イギリス国，ジーエル１２　８アールワイ，ウォットン－アンダー－エッジ，キングスウッド，ティンデル　ビュー　２４
(72)発明者　サイモン　クリスチャン　ノウルズ
イギリス国，ビーエー２　９エーエヌ，コーストン，ザ　バートン，ヒル　ハウス
(72)発明者　マルドゥラ　ゴア
イギリス国，ビーエー１　３ビーダブリュ，バス，セントジョーンズ　ロード　１４

【外国語明細書】
2020109605000001.pdf