(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2016-526220
(P2016-526220A)
(43)公表日　平成28年9月1日(2016.9.1)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 9/30	(2006.01)	G06F	9/30	350F	5B033
G06F 15/173	(2006.01)	G06F	15/173	683C	5B045

審査請求 未請求　予備審査請求 未請求　（全 85 頁）

(21)出願番号　特願2016-515113 (P2016-515113)
(86)(22)出願日　平成26年5月23日 (2014.5.23)
(85)翻訳文提出日　平成28年1月14日 (2016.1.14)
(86)国際出願番号　PCT/US2014/039345
(87)国際公開番号　WO2014/190263
(87)国際公開日　平成26年11月27日 (2014.11.27)
(31)優先権主張番号　61/827,117
(32)優先日　平成25年5月24日 (2013.5.24)
(33)優先権主張国　米国 (US)

(71)出願人　505000815
コーヒレント・ロジックス・インコーポレーテッド
アメリカ合衆国・７８７４６・テキサス州・オースティン・サウス　キャピタル　オブ　テキサス　ハイウェイ・１１２０・ビルディング　３，スイート　３１０
(74)代理人　100064621
弁理士　山川　政樹
(74)代理人　100098394
弁理士　山川　茂樹
(72)発明者　ドーア，マイケル・ビー
アメリカ合衆国・７８６２０・テキサス州・ドリッピング　スプリングス・オーク　メドウ　ドライブ・１２１０

最終頁に続く

(54)【発明の名称】プログラム可能な最適化を有するメモリネットワークプロセッサ

(57)【要約】
高パフォーマンス及び低電力散逸のために最適化された処理要素を有するマルチプロセッサシステムと、関連する上記処理要素をプログラミングする方法との、様々な実施形態が開示される。各処理要素は、フェッチユニットと、複数のアドレス生成器ユニットと、複数のパイプライン化されたデータ経路とを備えてよい。フェッチユニットは、マルチパート命令を受信するよう構成されてよく、このマルチパート命令は複数のフィールドを含む。第１のアドレス生成器ユニットは、複数のフィールドのうちの第１のフィールドに応じた算術演算を実施するよう構成されてよい。第２のアドレス生成器ユニットは、複数のアドレスのうちの少なくとも１つのアドレスを生成するよう構成されてよく、各アドレスは、複数のフィールドそれぞれに依存する。並列アセンブリ言語を用いて、複数のアドレス生成器ユニット及び複数のパイプライン化されたデータ経路を制御してよい。
【選択図】図２０

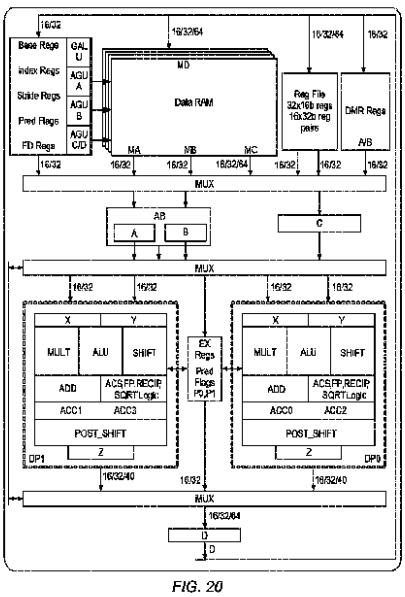


FIG. 20

【特許請求の範囲】
【請求項１】
マルチパート命令を受信するよう構成されたフェッチユニットであって、前記マルチパート命令は複数のフィールドを含む、フェッチユニット；及び
複数のアドレス生成器ユニット
を備える、装置であって、
前記複数のアドレス生成器ユニットのうちの第１のアドレス生成器ユニットは、前記複数のフィールドのうちの第１のフィールドに応じた算術演算を実施するよう構成され、
第２のアドレス生成器ユニットは、複数のアドレスのうちの少なくとも１つのアドレスを生成するよう構成され、
前記複数のアドレスはそれぞれ、前記複数のフィールドそれぞれに依存する、装置。
【請求項２】
前記複数のアドレスのうちの第１のアドレスに応じた第１のデータを保存するよう構成された、保存ユニットを更に備える、請求項１に記載の装置。
【請求項３】
前記フェッチユニットは更に、前記複数のアドレスのうちの第２のアドレスに応じた第２のデータをフェッチするよう構成される、請求項２に記載の装置。
【請求項４】
第３のアドレス生成器ユニットは、前記複数のアドレスのうちの少なくとも１つの別のアドレスを生成するよう構成される、請求項１に記載の装置。
【請求項５】
前記複数のフィールドのサブセットの各フィールドは、前記複数のアドレス生成器ユニットそれぞれが実施する演算を符号化する、請求項１に記載の装置。
【請求項６】
プロセッサを動作させるための方法であって、前記方法は：
マルチパート命令を受信するステップであって、前記マルチパート命令は複数のフィールドを含む、ステップ；
前記複数のフィールドのうちの第１のフィールドに応じた算術演算を実施するステップ；及び
前記複数のフィールドそれぞれに応じた、複数のアドレスのうちの所定のアドレスを生成するステップ
を含む、方法。
【請求項７】
前記複数のアドレスのうちの第１のアドレスに応じた第１のデータを保存するステップを更に含む、請求項６に記載の方法。
【請求項８】
前記複数のアドレスのうちの第２のアドレスに応じた第２のデータをフェッチするステップを更に含む、請求項７に記載の方法。
【請求項９】
前記プロセッサは複数のアドレス生成器ユニットを含み、
前記算術演算を実施するステップは、前記複数のアドレス生成器ユニットのうちの少なくとも１つのアドレス生成器ユニットを用いて前記算術演算を実施するステップを含む、請求項６に記載の方法。
【請求項１０】
前記複数のフィールドのサブセットの各フィールドは、前記複数のアドレス生成器ユニットそれぞれが実施する演算を符号化する、請求項９に記載の方法。
【請求項１１】
複数のプロセッサ；及び
複数の動的に構成可能な通信要素
を備える、システムであって、

前記複数のプロセッサ及び前記複数の動的に構成可能な通信要素は、散在型の配置において連結され、
前記複数のプロセッサのうちの所定のプロセッサは：
マルチパート命令を受信し、ここで前記マルチパート命令は複数のフィールドを含み；
前記複数のフィールドのうちの所定のフィールドに応じた算術演算を実施し；
前記複数のフィールドのサブセットに応じた複数のアドレスを生成する
よう構成される、システム。
【請求項１２】
前記複数のプロセッサはそれぞれ、前記複数のアドレスのうちの第１のアドレスに応じた第１のデータを保存するよう、更に構成される、請求項１１に記載のシステム。
【請求項１３】
前記複数のプロセッサのうちの前記所定のプロセッサは、前記複数のアドレスのうちの第２のアドレスに応じた第２のデータをフェッチするよう、更に構成される、請求項１２に記載のシステム。
【請求項１４】
前記複数のプロセッサのうちの前記所定のプロセッサは、複数のアドレス生成器ユニットを含む、請求項１１に記載のシステム。
【請求項１５】
前記複数のフィールドのサブセットの各フィールドは、前記複数のアドレス生成器ユニットそれぞれが実施する演算を符号化する、請求項１４に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明はマルチプロセッサシステムに関し、より詳細には、プロセッサ動作及び実行の改善、並びにこのようなシステムを標的にしたソフトウェアの開発に関する。
【背景技術】
【０００２】
一般的なハードウェアシステムの主たる目的は、完全なプログラム可能性を保持したまま、特定用途向け（非プログラマブル）ハードウェアパフォーマンスを達成することである。従来、これら２つの概念は対極にある。特定用途向けハードウェアは、可能な限り最も効率のよい方法で特定の機能を実施する、固定型ハードウェアソリューションである。これは通常、機能あたりのエネルギ又は１つ若しくは複数の演算あたりのエネルギに関して、及び製品の部分コストに関連し得る（回路）面積あたりの機能に関して、測定される。コスト及び市場ダイナミクスにより、この目的を満たそうという試みのための革新が推進されてきた。チップ製品のコストは、ダイ面積及び最終パッケージを含む多くの因子で構成される。コストはまた、製品を開発するための全体的なエコシステムも考慮しなければならない。このエコシステムのコストは、特定の用途を特定のハードウェアソリューションに還元するための時間、システム全体を構成するために必要な特定のハードウェアソリューションの個数、並びにカスタマイズされた通信及びメモリ構造によって全ての特定のハードウェアソリューションを集積するためにかかる時間で構成される。従って、多数の特定ハードウェアソリューションの全てをそのカスタム相互接続によってサポートするために、完全に集積されたソリューションが必要となり、これにより単一チップダイに関して極めて大きな面積が必要となる。従来このプロセスは、面積、エネルギ及び市場に出るまでの時間に関して効率的でないソリューションの原因となっていた。
【０００３】
プログラム可能性の世界及び標的のハードウェアの概念を考慮すると、ハードウェアアーキテクチャ及びソフトウェア開発スタイルの視点からの市場又は展望は：Ｉｎｔｅｌ、ＡＭＤ（Ｉｎｔｅｌ又はＡｒｍ命令セットベース）、ＡＲＭ等の企業が提供する汎用プロセッサ（ＧＰＰ）；ｎＶｉｄｉａ及びＡＭＤ（以前はＡＴＩであり、２００６年にＡＭＤにより買収）等からのグラフィカルプロセッシングユニット（ＧＰＵ）；ＴＩ及びＡｎａ

ｌｏｇＤｅｖｉｃｅｓからのデジタル信号プロセッサ（ＤＳＰ）；Ｘｉｌｉｎｘ、Ａｌｔｅｒａ等からのフィールドプログラマブルゲートアレイ（ＦＰＧＡ）；Ｃａｖｉｕｍ及びＴｉｌｅｒａからのマルチコア／メニーコアプロセッサ；特定用途向け集積回路（ＡＳＩＣＳ）又はシステムオンチップ（ＳｏＣ）によって提示される。

【０００４】

汎用プロセッサ（ＧＰＰ）

ＧＰＰは、４０年以上に亘って考えられてきた、極めて古いものの検証はされているハードウェアアーキテクチャに基づく、汎用処理のためのものであり、即ち「何でも屋（ａ　ｊａｃｋ　ｏｆ　ａｌｌ　ｔｒａｄｅｓ）」になろうとしている。ＧＰＰの主要な目的は、ユーザインタフェース（ＵＩ）と、ＭＳＷｏｒｄ、Ｅｘｃｅｌ、ｅメール等の、高度にインタラクティブなＵＩ集約型アプリケーションとを、これらをサポートするオペレーティングシステム（例えばＷｉｎｄｏｗｓ（登録商標）、Ｌｉｎｕｘ（登録商標））によって駆動することである。電力散逸に大きな影響を及ぼすハードウェア特性は、マルチレベルキャッシュ、複雑なハードウェアメモリ管理ユニット、大型のバス、大型のクロック生成構造である。要するにこれらは、そのタスクを実施するために多くの電力を散逸する。ソフトウェア開発の視点から、これは、標的とするのが最も簡単なソフトウェアプログラミングモデルであると考えられる。これは、ユーザが、連続して又は順次実行される単一のスレッドを開発しているという視点からのものである。並列処理又は複数ハードウェアスレッド（約４スレッド超）を導入する場合、これらを効率的にプログラムする能力は大幅に困難となる。これは、このアーキテクチャが根本的に並列スレッド演算をサポートするように開発されておらず、従ってハードウェアアーキテクチャは途方も無い量のオーバヘッド及び複雑さを管理する必要があるという事実による。ソフトウェアプログラミングモデルは、複数ソフトウェアスレッドの定義をサポートするためにＡＰＩ又は言語拡張の導入を必要とする。これは必ずしも複雑でないものの、残念ながら現行のＧＰＰハードウェアアーキテクチャはこのような複雑さを必要とする。

【０００５】

高次において、世界中の全てのスーパコンピュータでＣ、Ｃ＋＋、Ｆｏｒｔｒａｎ等を用いて長年に亘って広く使用されてきたＡＰＩは、ＭＰＩ（メッセージ受け渡しインタフェース）ＡＰＩであり、これは１９９０年代初頭から業界標準である。これは極めて単純な、よく理解されている、ハードウェア実装経路を制限しないＡＰＩである。ＭＰＩ　ＡＰＩは、ハードウェア非依存性様式での、ソフトウェアスレッド及び通信の定義を可能とする。これはＯｐｅｎＭＰ、ＣｏａｒｒａｙＦｏｒｔｒａｎ、ＯｐｅｎＣＬ等、及び仮定された下層のハードウェアモデルを本来的に記述する他の言語／ＡＰＩとは異なっており、従って解釈の柔軟性を制限し、前方互換性の問題を引き起こす。換言すると、これらの言語／ＡＰＩを用いる場合、プログラマは、標的とする全ての新規のハードウェアプラットフォームのためにプログラムを書き直す必要がある。

【０００６】

グラフィカルプロセッシングユニット（ＧＰＵ）

ＧＰＵは従来、データを処理して表示することを標的として開発されてきた。これらは、そのコア外（外部）メモリモデル要件及び内部コアメモリモデル要件により、ハードウェアアーキテクチャ的に制約されている。コア外メモリは、ＧＰＰに、データをＧＰＵメモリ空間に配置するよう要求する。続いてＧＰＵはそのデータを引き込み、そのデータに対してパイプライン様式で動作して、上記データを外部メモリ空間に戻す。ここからデータをディスプレイに送信でき、又はＧＰＰは、汎用処理下での動作において更に使用／記憶するために、データをＧＰＵメモリ空間外に移動させる必要がある。ハードウェアの非効率性は、（１）コア外ハードウェア制約をサポートするためにデータを移動させるために必要なサポート、及び（２）深いパイプライン構造のＳＩＭＤ機械と同様の、能率化されたパイプラインにおいて処理されるよう、データが制約を受ける、内部コアメモリ構造によるものである。その結果、データを処理するためのハードウェアの非効率性により、電力が高くなる。使用されるソフトウェアプログラミングモデルは、極めてハードウェア

中心的なものであり、ＯｐｅｎＣＬ、ＣＵＤＡ等であり、従って、効率性を達成するには複雑であり、移植可能性があまり高くなく、新規のハードウェア標的プラットフォームに移動させようとする際には、コードを書き直して再構成しなければならない。

【０００７】

デジタル信号プロセッサ（ＤＳＰ）

ＤＳＰは、一般的な信号処理のために削減及び標的化された命令セットを有するＧＰＰとして考えることができる。これらは、その兄弟／姉妹ＧＰＰと同一のキャッシュ、ＭＭＵ及びバスからの悪影響を受ける。Ｖｉｔｅｒｂｉ／Ｔｕｒｂｏ復号化又は動作推定等の、いずれの実際に高スループットの処理機能は、通常は市場における特定の規格の限定的なセットをサポートするだけの、限定された能力を有するＡＳＩＣ加速器へと削減されている。プログラミングモデルは、単一のハードウェアスレッドを標的とする場合のＧＰＰと同様であるが、実行ユニットハードウェアにおける信号処理命令アプローチにより、いずれの高効率を達成するには、複数の機能のハンドアセンブリ、又はＤＳＰ製造企業のライブラリの使用を必要とする。上述の並列ＧＰＰと同様に、多数の並列ＤＳＰアーキテクチャを生成する際、問題は更に悪化する。

【０００８】

フィールドプログラマブルゲートアレイ（ＦＰＧＡ）

ＦＰＧＡは完全に異なるハードウェアアプローチであり、この場合機能性の定義はビットレベルで実施でき、論理機能間の通信はプログラム可能な配線構造を通して実施される。このハードウェアアプローチは、途方も無い量のオーバヘッド及び複雑さを導入する。これにより、効率的なプログラミングが、Ｖｅｒｉｌｏｇ又はＶＨＤＬ等のハードウェアプログラミング言語で実施される。コンパイルプロセスは、ＡＳＩＣ／ＳＯＣにおいて必要とされるものと同様であるものの、構造化された配線ファブリックを有する、プログラム可能な配線及びプログラム可能な論理導入タイミング収束障害により、遥かに複雑となる。ＦＰＧＡはそのＦＰＧＡがプログラムされた目的の機能そのもののみを実施し、それ以外を一切実施しないため、１度に１つの機能のみを比較する場合、特定の機能に関する電力散逸及びパフォーマンススループットは、ＧＰＰ又はＧＰＵより明らかに遥かに良好である。しかしながら、ＧＰＰの能力の全てをＦＰＧＡに実装しようとした場合、ＧＰＰよりも明らかに悪くなる。ハードウェアレベルにおけるプログラミングの困難さは明らかである（例えばタイミング収束）。ＦＰＧＡのプログラミングは実際には「プログラミング」ではなく、論理／ハードウェア設計であり、またＶＨＤＬ／Ｖｅｒｉｌｏｇはプログラミング言語ではなく、論理／ハードウェア設計言語である。

【０００９】

マルチコア／メニーコア

殆ど全てのマルチコア／メニーコアアーキテクチャは、ハードウェアの視点から、コアプロセッサ、キャッシュ、ＭＭＵ、バス、全ての関連する論理を採用し、これらを、その周囲の通信バス／ファブリックによってダイ上で複製している。マルチコアアーキテクチャの例はＩＢＭのＣｅｌｌ、Ｉｎｔｅｌ及びＡＭＤのクアッドコア及びＮマルチコア、Ｃａｖｉｕｍ及びＴｉｌｅｒａの製品、多数のカスタムＳｏＣ等である。更に、マルチコアアーキテクチャにおいて達成される電力の低減は主に取るに足らないものである。この明らかな結果は、マルチコアアプローチがＧＰＵアプローチを単に複製するという事実に由来する。マルチコアアーキテクチャにおける唯一の実際の電力節約は、以前は別個のダイ上にあったコアが追加の通信バスに接続されることにより、もはや不要となったＩＯドライバの削減である。従って、マルチコアアプローチは電力のいずれの低下につながらない。第２に、ソフトウェアプログラミングモデルは上述のＧＰＰから改善されない。

【００１０】

特定用途向け集積回路（ＡＳＩＣ）又はシステムオンチップ（ＳｏＣ）

他のアプローチに関して識別される事項のリストは、特定の市場に関して、パフォーマンス効率及びコスト目標を達成するための唯一の方法が、特定のＧＰＰ、ＤＳＰ及びＡＳＩＣ加速器を有するカスタムチップを開発してＳｏＣを形成することであると考えられる

ことが多い理由である。ＳｏＣは、必要な場合はプログラム可能性と、電力散逸とコストとの間のバランスを取るための特定の機能に関するＡＳＩＣ性能とを提供する。しかしながら現在、ソフトウェアプログラミングモデルは、上述のプログラム可能なハードウェアソリューションのもとで議論されたものよりも更に複雑である。更にＳｏＣは、完全にプログラム可能なソリューションと関連する柔軟性の損失につながり得る。

【００１１】

これらのプログラム可能なハードウェアソリューション全てに共通なのは、現在市場において提示されているソフトウェアプログラミングモデルが、その標的化をより効率的にサポートするために、実行モデル及び下層のハードウェアアーキテクチャを外挿することに焦点を合わせている点である。実行モデルの特徴をソフトウェアプログラミングモデルに外挿することは、より一般的な並列プログラミング言語のうちのいくつかの重要な特徴を見ると観察できる。現在使用されているアプローチを示すいくつかの例は、ＯｐｅｎＭＰ、ＯｐｅｎＣＬ、ＭＰＩである。

【００１２】

ＯｐｅｎＭＰ

ＯｐｅｎＭＰ（オープンマルチプロセッシング）は、共有メモリマルチプロセッシングプログラミングをサポートする業界標準のＡＰＩである。ＯｐｅｎＭＰは、コンパイラ指示文、ライブラリルーチン、及びランタイム挙動に影響を与える環境変数のセットを含む。これは、並列化方法によるマルチスレッディングをサポートし、これによって親スレッド（連続して実行される一連の命令）は特定の数のスレーブスレッドに分岐し、タスクがこれらの間で分割される。そしてこれらのスレッドは同時に実行され、ランタイム環境はスレッドを、利用法、機械負荷及び他の因子に応じて異なるリソース又はプロセッサに割り当てる。スレッドの数は、ランタイム環境によって、環境変数に基づいて、又は関数を使用するコード内に割り当てることができる。従って、並列に実行されることが意図されたコードのセクションは、このセクションの実行前にスレッドを形成させるプリプロセッサ指示文によってマークされる。Ｃ／Ｃ＋＋では、これは＃ｐｒｏｇｒａｍｓの使用による。デフォルトでは、各スレッドはコードの並列化されたセクションを独立して実行する。タスクの並列処理及びデータの並列処理の両方を達成できる。並列化されたコードの実行後、複数のスレッドは親スレッドへ連結され、これはプログラムの最後まで進み続ける。図１は、ＯｐｅｎＭＰを利用したマルチスレッディングを示しており、ここで親スレッドは、コードの複数のブロックを並列に実行する多数のスレッドに分岐する。スレッド間通信をサポートするために、ＯｐｅｎＭＰの拡張、又は別の異なる業界標準ＡＰＩ（例えばＭＰＩ（メッセージ受け渡しインタフェース））を使用してよい。

【００１３】

ＯｐｅｎＣＬ

オープン計算言語（ＯｐｅｎＣＬ）は、中央演算処理装置（ＣＰＵ）グラフィカルプロセッシングユニット（ＧＰＵ）、デジタル信号プロセッサ（ＤＳＰ）、フィールドプログラマブルゲートアレイ（ＦＰＧＡ）及び他のプロセッサを備える異質な複数のプラットフォームに亘る実行を可能とすることを目的として、プログラムを記述するためのフレームワークである。これは抽象化が限定されたクロース・トゥ・ハードウェアインタフェースをサポートするよう設計されている。このため、ＯｐｅｎＣＬベースのプログラムは一般に、満足できるパフォーマンスを達成するために、下層のハードウェアに関する高度な知識を必要とする。ＯｐｅｎＣＬプログラムはまた、異なるハードウェアアーキテクチャを再標的化する際にリファクタリングを必要とする。図２に示すように、ＯｐｅｎＣＬは、厳密な処理モデル、制御モデル、メモリモデル、通信モデル階層をサポートする。

【００１４】

ＯｐｅｎＣＬは、何らかの制限及び追加を伴って、ＡＮＳＩ　Ｃプログラミング言語を用いてカーネルを記述することをサポートする。これは、関数へのポインタ、再帰、ビットフィールド、可変長配列、標準ヘッダファイルの使用を可能としない。複数のベクタタイプ及び演算を伴う並列処理、同期化、並びに複数のワークアイテム／グループを伴う作

業を行うための関数をサポートするよう、上記言語を拡張する。アプリケーションプログラミングインタフェース（ＡＰＩ）を用いて、プラットフォームを定義して制御する。ＯｐｅｎＣＬは、コースレベルにおいて、タスクベース及びデータベースの並列処理を用いた並列計算をサポートする。

【００１５】

ＭＰＩ

メッセージ受け渡しインタフェース（ＭＰＩ）は、標準化された、言語非依存の、スケーラブルな、移植可能な、メッセージ受け渡し通信プロトコルＡＰＩである。ＭＰＩ　ＡＰＩは、言語特異性構文を用いて、言語非依存性の方法で、（ノード／サーバ／コンピュータインスタンスに対してマッピングされた）プロセスのセット間の、必須の仮想トポロジ、同期化、及び通信機能を提供する（バインディング）。ＭＰＩ　ＡＰＩ規格は、様々な挙動を定義できるポイントツーポイント及び集合／同報通信送受信動作並びにプロセスの同期のためのサポートを含む、ライブラリルーチンのコアの構文及び意味規則を定義する。ＭＰＩは今日のハイパフォーマンス計算において使用される主要なモデルであり続けている。

【００１６】

ＭＰＩ　ＡＰＩは、最もハードウェア非依存性のアプローチであり、従って提示された例の基礎として使用される。

【００１７】

マルチプロセッサシステム上での並列実行のためのソフトウェアアプリケーションの開発の、従来技術によるアプローチは、一般に、開発の容易さと並列実行の効率性との間のトレードオフを必要とする。換言すると、一般に、プログラマにとって開発プロセスが容易であればあるほど、結果として得られる実行可能なプログラムをハードウェア上で同時に実行する効率が低くなり、また反対に、より効率的な並列実行は一般に、プログラマによる相当に多大な努力を必要とし、即ち非効率的な処理を回避するために、及び標的ハードウェアの、効率増進性特徴を使用するために、プログラムを更に詳細に設計する必要がある。

【発明の概要】

【発明が解決しようとする課題】

【００１８】

従って、ソフトウェアプログラミングモデルを駆動するためのアプリケーションのソフトウェア記述又はシステムレベルビューと、実行モデル及び下層ハードウェアアーキテクチャを標的化するためのこれらの後の使用とを促進するための、改善されたシステム及び方法が望まれている。また、このプロセスを通してアプリケーションの効率的かつプログラム可能な実装を可能とする機構を提供する改善も望まれている。

【課題を解決するための手段】

【００１９】

高パフォーマンス及び低電力散逸のために最適化された処理要素を有するマルチプロセッサシステムと、関連する上記処理要素をプログラミングする方法との、様々な実施形態が開示される。

【００２０】

第１の実施形態は、フェッチユニット及び複数のアドレス生成器ユニットを備えるプロセッサ装置に関する。フェッチユニットはマルチパート命令を受信するよう構成され、マルチパート命令は複数のフィールドを含む。第１のアドレス生成器ユニットは、複数のフィールドのうちの第１のフィールドに応じた算術演算を実施するよう構成される。第２のアドレス生成器ユニットは、複数のアドレスのうち少なくとも１つのアドレスを生成するよう構成され、各アドレスは、複数のフィールドそれぞれに依存する。

【００２１】

多数のデータ経路を制御することに関する第２の実施形態は、フェッチユニット及び実行ユニットを備えるプロセッサ装置を伴う。フェッチユニットはマルチパート命令を受信

するよう構成され、マルチパート命令は複数のフィールドを含む。実行ユニットは複数のパイプラインユニットを含み、実行ユニットは：１）複数のフィールドのうちの第１のフィールドに応じて、複数のパイプラインユニットのうちの所定のパイプラインユニットを用いて第１の演算を実施するよう；及び２）複数のフィールドのうちの第２のフィールドに応じて、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて第２の演算を実施するよう、構成される。

【００２２】

累算の転送に関する第３の実施形態は、フェッチユニット及び実行ユニットを備えるプロセッサ装置を伴う。フェッチユニットは命令を受信するよう構成される。実行ユニットは複数のパイプラインユニットを含み、上記複数のパイプラインユニットの各パイプラインユニットはアキュムレータユニットを含み、実行ユニットは：１）受信した命令に応じて、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第１の演算を実施して、ある結果を生成するよう；２）複数のパイプラインユニットのうちの第１のパイプラインユニットのアキュムレータユニットに上記結果を保存するよう；及び３）複数のパイプラインユニットのうちの第１のパイプラインユニットのアキュムレータユニットに保存された結果を、プロセッサのアキュムレータユニットに転送するよう、構成される。

【００２３】

アドレス生成器ユニットの連結に関する第４の実施形態は、フェッチユニット及び複数のアドレス生成器ユニットを備えるプロセッサ装置を伴う。フェッチユニットはマルチパート命令を受信するよう構成され、マルチパート命令は複数のフィールドを含む。第１のアドレス生成器ユニットは、複数のフィールドのうちの第１のフィールドに応じて第１の演算を実施して、第１の結果を生成するよう構成される。第２のアドレス生成器ユニットは、複数のフィールドのうちの第２のフィールド及び上記第１の結果に応じて第２の演算を実施するよう構成される。

【００２４】

第５の実施形態は、単一パート／マルチパート決定を伴う命令を受信できるプロセッサ装置に関する。プロセッサ装置は、命令を受信するよう構成されたフェッチユニットと、実行ユニットとを備えてよい。実行ユニットは、複数のパイプラインユニットを備えてよい。実行ユニットは：１）命令がマルチパート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第１の演算を実施するよう（ここでマルチパート命令は複数のフィールドを含み、第１の演算は複数のフィールドのうちの第１のフィールドに依存する）；及び２）複数のフィールドのうちの第２のフィールドに応じて、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて第２の演算を実施するよう、構成してよい。

【００２５】

第６の実施形態は、反復ループ中にプロセッサの未使用部分をパワーダウンさせることができる、プロセッサ装置に関する。このプロセッサ装置は、フェッチユニット及び実行ユニットを備えてよい。フェッチユニットは、複数の命令を受信して、受信した複数の命令に基づいて少なくとも１つの反復命令シーケンスを識別するよう構成される。上記少なくとも１つの反復命令シーケンスは、複数の命令のうちの少なくとも１つを含む。実行ユニットは複数のパイプラインユニットを含み、実行ユニットは：１）識別された反復命令シーケンスに基づく少なくとも第１のサイクルに関する、複数のパイプラインユニットの第１のサブセットを選択的に無効化するよう；及び２）識別された反復命令シーケンスに応じた第２のサイクルに関する、複数のパイプラインユニットの第２のサブセットを選択的に無効化するよう、構成される。

【００２６】

マルチプロセッサシステム上での並列実行のために標的化されたソフトウェアを開発するための方法の実施形態を提示する。

【００２７】

例えばマルチプロセッサシステム上での展開及び実行のために標的化されたアプリケーションの、所望のシステムの複数のビューを指定する入力を受信してよい。各ビューは、システムの各態様を提示又は指定してよく、またこれらビューは総体として、効率的な並列実行のためにマルチプロセッサシステムが展開できる実行可能なプログラムを生成するためにコンパイラ（又は他のソフトウェアツール）が使用できる情報を提供してよい。

【００２８】

この入力は、様々な形態のいずれで、また様々なツールのいずれを介して受信されてよい。例えばいくつかの実施形態では、入力はユーザによって提供されてよく、即ちユーザ入力であってよい。他の実施形態では、入力は所望に応じて別のシステム又はプロセス、ストレージ媒体等から受信されてよい。更に入力はローカルに提供されてよく、又はローカルエリアネットワーク（ＬＡＮ）若しくはインターネット等の広域ネットワーク（ＷＡＮ）を介して受信されてよい。例示的な一実施形態では、ユーザはスプレッドシート中でビューを指定してよい。別の例示的実施形態では、ユーザはウィザード、即ちグラフィカルユーザインタフェース（ＧＵＩ）への入力を行ってよく、これは指定プロセスを通して、例えばプロンプト、有用な提案等によってユーザをリードする。更なる実施形態では、ユーザは入力／指定プロセスを管理するためにチェックリストを使用してよく、指定される各アイテム、例えばビュー、サブビュー等がチェックリストに挙げられており、チェックリストは各チェックリストアイテムが指定されているか（されていないか）を示す。別の例示的実施形態では、１つ又は複数のテンプレート又はフォームを提供してよく、ユーザはビューを指定する情報で上記テンプレート又はフォームを埋めてよい。

【００２９】

各ビューは、アプリケーションの動作又は実行の各態様を含むか又はその表現となってよい。多数のビューはメモリビュー、通信ビュー、制御ビュー、処理ビューを含んでよい。他のビューも所望に応じて定義又は指定してよい。一実施形態では、各ビューは、ビューの更に詳細な属性を指定する複数のサブビュー（又は「ファセット（ｆａｃｅｔ）」）を含むか又は指定してよい。例えば各ビューは、サイズ、挙動及びアクセシビリティサブビュー（又はファセット）を含んでよく、各サブビューは、各サブビューがその一部となっているビューに関連する特定の意味を有してよい。

【００３０】

従って例えば、メモリビューは：アプリケーションによる使用に必要な又は利用可能なメモリのサイズ（又は量）、即ちデータを処理するためにアプリケーションが使用するメモリ構造サイズ；メモリの挙動、即ちメモリ構造が所定のタイミングで挙動する様式；並びにメモリのアクセシビリティ、即ち例えばアプリケーション及び／又はシステムによるメモリ構造のアクセシビリティを指定してよい。

【００３１】

同様に、入力は：通信ビュー（通信ビューの各サブビュー、例えば通信サイズ、挙動、アクセシビリティを定義することを含む）；制御ビュー（制御サイズ、挙動、アクセシビリティを含む）；並びに処理ビュー（処理サイズ、挙動、アクセシビリティを含む）を定義又は指定してよい。

【００３２】

いくつかの実施形態では、ビュー又はサブビューのうちのいくつかは、他のサブビュー又はビューの指定によって、例えば同一の又は異なるビューのサブビューによって、自動的に定義又は指定され得ることに留意されたい。従って例えば、メモリサイズ、挙動及びアクセシビリティが指定されると、通信挙動が自動的に指定され得る。別の様式で考えると、いくつかの実施形態では、ビューは、方程式の数が変数の数を超える線形方程式の優決定系と同様に、「過剰に指定される」か又は「過剰に決定される」場合がある。

【００３３】

本出願で使用される特定の用語又は標識は単なる例示であること、並びに所望に応じて本出願で開示される新規の構成要素、情報及びプロセスに対していずれの名称を使用してよいことに留意されたい。例えば所望に応じて、ビュー又は態様はモデル等と呼ばれる場

合もあり、本出願で開示されるサブビューはサブモデル、ファセット、プロパティ等と呼ばれる場合もある。

【００３４】

システムのビューが指定又は定義されると、これらのビューを表す情報は、アプリケーションのソースコード内に含まれ得る。この包含は、広範な方法のうちのいずれにおいて実施され得る。例えばいくつかの実施形態では、情報はアプリケーションの１つ又は複数のヘッダファイルに含まれ得る。他の実施形態では、情報は、アプリケーションプログラム要素又は構成要素の中でとりわけ、１つ又は複数の動的リンクライブラリ（ＤＬＬ）又はマクロ定義に含まれ得る。より一般には、ビューを表す情報は、所望に応じていずれの様式及びいずれの形態でアプリケーションソースコードに組み込まれてよい。

【００３５】

ソースコードは例えば、コンパイラ又は他のツールによって処理でき、これはシステムに関して指定又は定義された複数のビューを表す情報を分析することを含む。例えば一実施形態では、コンパイラは、アプリケーションソースコード内の複数のビューを表す情報を認識するよう構成してよく、またこの情報を抽出及び分析してよい。他の実施形態では、コンパイラは情報を原位置で分析してよい。

【００３６】

実行可能なプログラムは上記処理に基づいて生成してよく、ここで上記実行可能なプログラムは、効率的な並列実行のためにマルチプロセッサシステムに展開可能である。換言すると、コンパイラは、指定されたビューの分析を含む上記処理に基づいて、実行可能なプログラムを生成してよい。

【００３７】

従って、本技術の実施形態は、上述のようなソフトウェア開発への従来技術のアプローチの様々な欠点に対処でき、これによってユーザは、システムの動作、例えばマルチプロセッサシステム上でのアプリケーションの動作に関する様々な要件又は制約を指定でき、これら指定された要件又は制約をコンパイラ（又は他のツール）が使用して、システム上で効率的に実行できる実行可能なコードを生成してよい。

【図面の簡単な説明】

【００３８】

【図１】図１は、ＯｐｅｎＭＰを利用したマルチスレッディングを示し、親スレッドはコードのブロックを並列に実行する多数のスレッドに分岐する。

【図２】図２は、ＯｐｅｎＣＬの厳密処理モデル、制御モデル、メモリモデル、通信モデル階層を示す。

【図３】図３は、マルチプロセッサシステム（ＭＰＳ）の一実施形態を示すブロック図である。

【図４】図４は、ＭＰＳ接続スキームの一実施形態を示すブロック図である。

【図５】図５は、ＭＰＳファブリックの一実施形態を示す更に詳細な図である。

【図６】図６は、図５のアーキテクチャ例に従ってＤＭＲ（円）と共に均一に分散させたＰＥ（正方形）からなる、例示的なＭＰＳを示す。

【図７】図７は、動的に構成可能なプロセッサ（ＰＥ）の一実施形態を示すブロック図である。

【図８】図８は、一実施形態による、マルチプロセッサシステム上での並列実行のために標的化されたアプリケーションソフトウェアを開発するための方法のフローチャートである。

【図９Ａ】図９Ａは、一般的な２Ｄフィルタリング組織スキームを示す。

【図９Ｂ】図９Ｂは、別の一般的な２Ｄフィルタリング組織スキームを示す。

【図１０】図１０は、単一のプロセッサによる画像フィルタリングのブロック図を示す。

【図１１】図１１は、多数のプロセッサによる画像フィルタリングのブロック図を示す。

【図１２】図１２は、例示的なビデオ２Ｄフィルタシステムのための、ＭＰＳ上での２つの異なるリソースレイアウトを示す。

【図１３】図１３は、単一のプロセッサを用いたＦＩＲフィルタを示すブロック図である。
【図１４】図１４は、多数のプロセッサを用いたＦＩＲフィルタを示すブロック図である。
【図１５】図１５は、サンプルバッファに関するメモリ構造を示す。
【図１６】図１６は、メモリ通信挙動に関する経時的構造を示す。
【図１７】図１７は、ＦＩＲフィルタの一部に関する、メモリ通信挙動に関する経時的構造を示す。
【図１８】図１８は、９ステージＰＥパイプラインの一実施形態のパイプラインステージを示す。
【図１９】図１９は、単一ＰＥデータ経路アーキテクチャ及び改善されたＰＥデータ経路アーキテクチャの実施形態のブロック図を示す。
【図２０】図２０は、データ経路の一実施形態を示すブロック図である。
【図２１】図２１は、アキュムレータ転送を実施する２つのＰＥの一実施形態を示すブロック図である。
【図２２】図２２は、ＰＥのアドレス生成論理の一実施形態を示すブロック図である。
【図２３】図２３は、アドレス生成ユニット（ＡＧＵ）の一実施形態を示すブロック図である。
【図２４】図２４は、循環バッファの一実施形態の概念図である。
【図２５】図２５は、図２４の循環バッファの制御の一実施形態を示すブロック図である。
【図２６】図２６は、従来のプログラミングモデル及びＨｙｐｅｒＯｐプログラミングモデルを示す２つのブロック図を示す。
【図２７】図２７は、ＨｙｐｅｒＯｐ命令の一実施形態を示す。
【図２８】図２８は、命令フェッチ及び復号化ユニットの一実施形態を示すブロック図である。
【図２９】図２９は、マルチパート命令を受信及び実行する第１の実施形態を示すフローチャートである。
【図３０】図３０は、マルチパート命令を受信及び実行する第１の実施形態を示すフローチャートである。
【図３１】図３１は、プロセッサによる演算の実施の一実施形態を示すフローチャートである。
【図３２】図３２は、プロセッサによる演算の実施の一実施形態を示すフローチャートである。
【図３３】図３３は、プロセッサを動作させる一実施形態を示すフローチャートである。
【図３４】図３４は、複数のパイプラインユニットを有するプロセッサを動作させる第１の実施形態を示すフローチャートである。
【発明を実施するための形態】
【００３９】
本開示は様々な修正及び代替形態を許容するものであるが、その具体的な実施形態を例として図面に示し、また本明細書で詳細に説明する。しかしながら、上記具体的実施形態の図及び詳細な説明は、図示されている特定の形態に開示を限定することを意図したものではなく、反対に、添付の請求項によって定義されるような本開示の精神及び範囲内にある全ての修正例、均等物及び代替例を包含することを意図したものであることを理解されたい。本明細書において使用されている見出しは、単に組織化を目的としたものであり、これらの使用は本説明の範囲の限定を意味しない。本出願全体を通して使用される単語「してよい／し得る／できる（ｍａｙ）」は、許容の意味で（即ち「可能性がある」ことを意味して）使用されており、強制の意味で（即ち「しなければならない」ことを意味して）使用されるものではない。同様に、単語「含む（ｉｎｃｌｕｄｅ／ｉｎｃｌｕｄｉｎｇ／ｉｎｃｌｕｄｅｓ）」は、ある対象を含むもののそれに限定されないことを意味する。

【００４０】
様々なユニット、回路又はその他の構成部品は、１つ又は複数のタスクを実施する「よう構成される（ｃｏｎｆｉｇｕｒｅｄ　ｔｏ）」として記載され得る。このような文脈において「よう構成される」は、動作中に上記１つ又は複数のタスクを実施する「回路構成を有する」ことを一般に意味する、構造の広範な説明である。従ってユニット／回路／構成部品は、ユニット／回路／構成部品が現在オンでなくても上記タスクを実施するよう構成できる。一般に「よう構成される」に対応する構造を形成する回路構成は、ハードウェア回路を含んでよい。同様に、記載を簡略化するために、様々なユニット／回路／構成部品は、１つ又は複数のタスクを実施するとして記載され得る。このような記載は「よう構成される」という語句を含むものとして解釈されるものとする。１つ又は複数のタスクを実施するよう構成されるユニット／回路／構成部品の列挙は、これらユニット／回路／構成部品に関して米国特許法第１１２条第６段落の解釈を援用しないことを明示的に意図したものである。より一般には、いずれの要素の列挙は、「…のための手段（ｍｅａｎｓ　ｆｏｒ）」又は「…のためのステップ（ｓｔｅｐ　ｆｏｒ）」という語句が具体的に使用されていない限り、上記要素に関して米国特許法第１１２条第６段落の解釈を援用しないことを明示的に意図したものである。

【００４１】
参照による援用
Ｍｉｃｈａｅｌ　Ｂ．Ｄｏｅｒｒ、Ｗｉｌｌｉａｍ　Ｈ．Ｈａｌｌｉｄｙ、Ｄａｖｉｄ　Ａ．Ｇｉｂｓｏｎ、Ｃｒａｉｇ　Ｍ．Ｃｈａｓｅを発明者とする、発明の名称「Ｐｒｏｃｅｓｓｉｎｇ　Ｓｙｓｔｅｍ　Ｗｉｔｈ　Ｉｎｔｅｒｓｐｅｒｓｅｄ　Ｓｔａｌｌ　Ｐｒｏｐａｇａｔｉｎｇ　Ｐｒｏｃｅｓｓｏｒｓ　Ａｎｄ　Ｃｏｍｍｕｎｉｃａｔｉｏｎ　Ｅｌｅｍｅｎｔｓ」の米国特許第７４１５５９４号は、その全体を参照することにより、本明細書においてその全体が完全に論述されているかのように、本明細書に援用されるものとする。

【００４２】
用語
コンピュータシステム‐用語「コンピュータシステム」は、パーソナルコンピュータシステム（ＰＣ）、メインフレームコンピュータシステム、ワークステーション、ネットワーク家電、インターネット家電、パーソナルデジタルアシスタント（ＰＤＡ）、テレビジョンシステム、グリッドコンピューティングシステム若しくはその他のデバイス又はデバイスの組み合わせを含む、様々なタイプの計算又は処理システムのいずれかを指す。一般に、用語「コンピュータシステム」は、メモリ媒体からの命令を実行する少なくとも１つのプロセッサを有するいずれのデバイス（又は複数のデバイスの組み合わせ）を包含するものとして広く定義できる。

【００４３】
ソフトウェアアプリケーション‐用語「ソフトウェアアプリケーション」（本出願では単に「アプリケーション」とも呼ぶ）は、その一般的な意味の全範囲を有することを意図したものであり、１つ又は複数のメモリに記憶でき、かつ１つ又は複数のプロセッサが実行できる、あらゆるタイプのプログラム命令、コード、スクリプト及び／又はデータ又はこれらの組み合わせを含む。例示的なソフトウェアアプリケーションは、Ｃ、Ｃ＋＋、ＦＯＲＴＲＡＮ、Ｊａｖａ（登録商標）、アセンブリ言語等のテキストベースプログラム言語で書かれたプログラム；グラフィックプログラム（グラフィックプログラム言語で書かれたプログラム）；アセンブリ言語プログラム；機械言語にコンパイルされたプログラム；スクリプト；並びに他のタイプの実行可能なソフトウェアを含む。一般にプログラムは、１つ又は複数のデータ構造を指定し、またこれらの構造内のデータに対して取るべき手順のステップを指定して１つ又は複数の機能を実施する、命令のセットである。プログラムは、特定の機械アーキテクチャに関して標的化される場合が多い。更に抽象的には、プログラムの手順のステップは、上記プログラムのアルゴリズムと呼ばれる場合がある。

【００４４】

アプリケーションは、マルチプロセッサシステム（ＭＰＳ）の１つ又は複数のプロセッサ上で実行でき、ＭＰＳのローカルメモリのうちの１つ若しくは複数からデータを読み出すことができ、及び／又はＭＰＳのローカルメモリのうちの１つ若しくは複数にデータを書き込むことができる。アプリケーションは、１つ又は複数の計算タスクを含んでよく、ここで各タスクは典型的にはＭＰＳの単一のプロセッサ上で実行され、１つ又は複数のアプリケーションからの１つ又は複数のタスクと上記プロセッサを共有してよい。アプリケーションは、特定の機能又は演算を実施してよい。アプリケーションが２つ以上のタスクを含む場合、これらタスクは互いに通信して上記機能又は演算を実施してよい。

【００４５】

ＭＰＳは複数のアプリケーションを同時に実行してよく、例えばこれらアプリケーションは互いに並列に実行される。アプリケーションは互いに通信してよく、これらアプリケーションが実施する各機能又は演算は、より大きい又はより高いレベルの機能又は演算を実施するために互いを利用してよい。

【００４６】

ソフトウェアプログラミングモデル‐ソフトウェアプログラミングモデルは、簡単に言うと、ユーザの、機械及びその動作環境のビューである。ソフトウェアプログラミングモデルは、アプリケーションを記述できる１つ又は複数の言語、並びに上記１つ又は複数の言語で直接表現できるものを超えた抽象化及びプセル化された機能性を提供するライブラリを含む。ソフトウェアプログラミングはまた、それを通してアプリケーションがその外部のエンティティ（Ｉ／Ｏ、拡張メモリ等）と相互作用する、及びそれを通して上記アプリケーションに関するメタ情報（例えばパフォーマンス制約又は要件）が表現される、機構も含む。プログラミングモデルの２つの主要な部分は、並列処理がアプリケーション内でどのように表現されるか又はアプリケーションからどのように引き出されるかを示す制御モデル、及びアプリケーションの並列のエンティティがどのように情報を共有するかを示す通信モデルである。

【００４７】

ソフトウェアプログラミングモデルは、実際の制御及びデータフロー及びアプリケーションが最終的に実行された場合に発生することになる通信の、「理想化された」ビューを提示する。演算の意味規則は、下層の実装がソフトウェアプログラミングモデルにおいて記述された通りに正確に実施されているかのようになるものであり、同一の効果（答え）が得られる限り、実際に行われるステップは重要ではない。実際の実装ステップは、コードの効率及び／又はデータサイズ、速度、電力消費等の理由により異なり得る。

【００４８】

ソフトウェアプログラミングモデルの考慮すべき重要な事柄は、ツールセット（コンパイラ等）による、及びそれに続く実行モデル下でのアプリケーションの正確かつ効率的な処理をサポートするために十分な情報もキャプチャしながら、便利かつ自然でユーザにとって直感的な用語でのアプリケーションの表現をサポートする機構を、ソフトウェアプログラミングモデルがユーザに同時に提供することである。

【００４９】

ハードウェアプログラミング／実行モデル‐ハードウェアプログラミングモデル又は実行モデルは、アプリケーションがどのように実行されるかを表す。これは、アプリケーションの論理及びデータオブジェクトに対応する情報のセットがどのように表現されるか、並びにその情報が経時的にどのように処理されて、アプリケーションによって指定された機能が達成されるかを定義する。システムツール（コンパイラ、並列処理抽出器、配置配線（ｐｌａｃｅ‐ａｎｄ‐ｒｏｕｔｅ）等）の目的は、アプリケーションをそのソフトウェアプログラミングモデル表現から対応する実行モデル表現に変換することである。実行モデルは、（例えばライブラリを通して）ソフトウェアプログラミングモデルによって記述された機能性をサポートし、（例えばＯ／Ｓを通して）ハードウェアの使用を監視、調停、管理するために必要な機構を含む。

【００５０】

実行モデルは、ソフトウェアプログラミングモデルと極めて密接に対応してよく、又は完全に異なっていてよい。ソフトウェアプログラミングモデルの異なる複数の態様は、実行モデルへの異なる直接的対応度を有してよい。対応のレベルは、下層のハードウェアアーキテクチャが、元々の（ソフトウェア）プログラミングモデルとどの程度類似しているかに関連する。類似性が高いほど対応性が高い。

【００５１】

下層のハードウェアアーキテクチャ‐下層のハードウェアアーキテクチャは、計算が実行される物理デバイスのアーキテクチャである。このレベルでは、全ての動作はデバイスが実行する物理的動作に直接対応する。下層のハードウェアアーキテクチャを記述できる抽象のレベルは、（設計空間開発中の評価、シミュレーション、特性決定、トレードオフ分析に有用な）高次概念アーキテクチャから、（製作されるデバイスの物理的設計の実施に有用な）低次実装アーキテクチャに変化し得る。実装レベルにおいてさえ、下層のハードウェアアーキテクチャの異なる複数の例は、能力又は性能において異なり得る。例えばある例は１０×１０グリッドの処理ユニットを実装してよく、その一方で別の例はたった６×６グリッドしか実装しなくてよい。これらはそれぞれ能力が異なるものの、下層のハードウェアアーキテクチャとの整合性を保持している。

【００５２】

自動的に（ａｕｔｏｍａｔｉｃａｌｌｙ）：その作用又は動作を直接指定又は実施するユーザ入力を必要とせずに、コンピュータシステム（例えばコンピュータシステムが実行するソフトウェア）又はデバイス（例えば回路構成、プログラム可能なハードウェア要素、ＡＳＩＣ等）が実施する動作又は操作について用いる。従って用語「自動的に」は、ユーザが手動で実施又は指定する操作（ここでユーザが操作を直接実施するために入力を提供する）と対照的なものである。自動処理は、ユーザが提供する入力によって開始される場合があるが、これに続く「自動的に」実施される動作は、ユーザが指定するものではなく、即ち「手動で」実施される（ユーザが各動作の実施を指定する）ものではない。例えばユーザが、各フィールドを選択し、（例えば情報をタイピングすることによって、チェックボックスを選択することによって、無線選択によって等で）情報を指定する入力を提供することによって、電子フォームを埋める場合、仮にコンピュータシステムがユーザの動作に応答して上記フォームを更新しなければならないとしても、これは上記フォームを手動で埋めたことになる。このようなフォームはコンピュータシステムによって自動で埋めることができ、この場合コンピュータシステム（例えばコンピュータシステム上で実行されるソフトウェア）は、フォームのフィールドを分析して、フィールドへの回答を指定するいずれのユーザ入力を必要とせずにフォームを埋める。上述のように、ユーザはフォームを自動で埋める動作を発動する場合はあるが、実際にフォームを埋める動作には関わらない（例えばユーザはフィールドへの回答を手動で指定せず、回答は自動的に完了する）。本明細書は、ユーザが行う動作に応答して自動的に実施される操作の様々な例を提供する。

【００５３】

ＭＰＳシステムの概要

マルチプロセッサシステム（ＭＰＳ）及び関連する方法の様々な実施形態を説明する。マルチプロセッサシステム（ＭＰＳ）は、複数の処理要素（ＰＥ）を含むシステムとして定義できる。ＭＰＳは、これらＰＥ間に散在する複数のメモリを有してよく、あるいは単一の共有メモリを有してよい。本明細書で使用される場合、用語「処理要素（ｐｒｏｃｅｓｓｉｎｇ　ｅｌｅｍｅｎｔ）」は、プロセッサ若しくはＣＰＵ（中央演算処理装置）、マイクロプロセッサ又はプロセッサコアを指す。ＭＰＳは２つ以上のいずれの個数のＰＥを含んでよいが、いくつかのＭＰＳは、典型的には汎用プロセッサ（ＧＰＰ）を１つのみ、又は数個のＧＰＰのみを含む従来のコンピュータシステムよりも有意に多い個数のＰＥを含んでよいことに留意されたい。例えばいくつかのＭＰＳは、４、８、１６、３２又は６４個のＰＥを含んでよい（他の例は例えば数ダース、数百個又は数千個ものＰＥを含む）。いくつかの実施形態では、大型ＭＰＳに好適なＰＥは、低電力消費を目的とした特別

な構成により、従来のコンピュータシステムによって使用される汎用プロセッサよりもエネルギ効率が高いものであってよい。

【００５４】

ＭＰＳはまた、ＰＥ及び／又はメモリを相互接続する相互接続ネットワーク（ＩＮ）も含んでよい。ＰＥ及びメモリは、円形次元（例えばループ又はリング）を含む１、２、３又は４以上の次元で相互接続してよい。より高い次元のＭＰＳは、より低い次元のファブリケーション媒体上にマッピングできる。例えば４次元（４Ｄ）超立方体の形状を有するＭＰＳは、シリコン集積回路（ＩＣ）チップの３Ｄスタック上に、又は単一の２Ｄチップ上に、又は計算ユニットの１Ｄの線上にさえ、マッピングできる。低次元のＭＰＳをより高次元の媒体にマッピングすることもできる。例えば計算ユニットの１Ｄの線を、ＩＣチップの２Ｄ平面上に曲がりくねった形状で展開でき、又はチップの３Ｄ積層体へと巻くことができる。ＭＰＳは複数のタイプの計算ユニットと、プロセッサ及びメモリが散在する構成とを含んでよい。広い意味でのＭＰＳには、ＭＰＳの階層又は入れ子構成、特に相互接続されたＩＣチップからなるＭＰＳも含まれ、この場合ＩＣチップは１つ又は複数のＭＰＳを含み、これらＭＰＳもまた更に深い階層構造を有してよい。

【００５５】

本出願において使用される用語ＭＰＳは、複数のプロセッサの比較的均一なセットと、いわゆる「プラットフォームＩＣ」チップ上に集積された汎用プロセッサ及び特殊化されたプロセッサの異種集団との両方を包含する。プラットフォームＩＣチップは数個～多数のプロセッサを含んでよく、これらは典型的には共有メモリによって相互接続され、場合によってはオンチップネットワークによって相互接続される。ＭＰＳと「プラットフォームＩＣ」チップとの間には違いがあってもなくてもよい。しかしながら「プラットフォームＩＣ」チップは、特定の垂直的市場における特定の技術要件に対処するために市販されているものであってよい。

【００５６】

一般に、ＭＰＳのためのメモリは階層として組織してよく、この階層は頂部に高速なメモリを有し、階層を１段ずつ下がるにつれてより低速であるがより大容量のメモリを有する。ＭＰＳ中において、階層の頂部の補助メモリは、各ＰＥの近傍に位置してよい。各補助メモリは、最適な命令又は最適なデータを保持するよう特殊化できる。特定のＰＥのための補助メモリは、そのＰＥ専用のものであっても、又は他のＰＥと共用であってもよい。

【００５７】

メモリ階層を更に下がると、各ＰＥに隣接する補助メモリの何倍も大きいビット容量を有する半導体同期ダイナミックランダムアクセスメモリ（ＳＤＲＡＭ）等の、比較的大型の共有メモリが存在してよい。ＳＤＲＡＭは、その製作を特定化するために、ＰＥ及び補助メモリとは分離された１つ又は複数のＩＣチップ上に位置してよい。メモリ階層を更に下がると、フラッシュメモリ、磁気ディスク及び光学ディスク等の他のタイプのメモリが存在してよい。

【００５８】

ＭＰＳは、特定の機能を達成できるよう、ソフトウェアプログラムを用いてプログラムされてよい。各機能は、ＭＰＳ中のＰＥのうちの１つ又は複数によって実行されてよい。しばしば、ＭＰＳ上で複数のプログラムを同時に実行してよい。プログラムは、より複雑な機能を実施するために、及び並列処理技術を採用することによってより単純な機能をより迅速に実施するために、共に実行され、互いに通信してよい。ＰＥ間のこのような協調を、本出願では連携処理と呼ぶ。

【００５９】

ＭＰＳは、関連するデータ及びコマンドのソースが入力データ及びコマンドを提供するよりも早くこれらを受信でき、無視できる程度に十分に低いレイテンシで結果を提供できるよう、十分な速さでアプリケーション又はプログラムを実行できる。このようなアプリケーションは、遅延なくリアルタイムで動作する、即ち「リアルタイムアプリケーション

」と呼ばれる。関連する入力データ（又はコマンド）は「リアルタイムデータ」（又は「リアルタイムコマンド」）と呼ばれる。例えばＭＰＳは入力信号を介してリアルタイムデータを受信してよい。アプリケーション、プログラム又は機能のうちの１つ又は複数は入力信号を処理してよく、場合によっては、１つ又は複数のプログラムに基づいて修正された又は追加のリアルタイムデータを伴う出力信号を生成してよい。

【００６０】

図３‐ＭＰＳブロック図及び概要

図３は、マルチプロセッサシステム（ＭＰＳ）の一実施形態を示すブロック図である。図示されている実施形態では、ＭＰＳ１０は複数の処理要素（ＰＥ）及び複数のデータメモリルータ（ＤＭＲ）を含み、ＤＭＲは、データ及び命令を互いに通信するよう連結された、動的に構成可能な通信器、又は動的に構成可能な通信要素と呼ぶこともできる。本出願で使用される場合、ＰＥはＰＥノードとも呼ばれる場合があり、またＤＭＲはＤＭＲノードとも呼ばれる場合がある。

【００６１】

処理システム（ＭＰＳ）１０は、ＧＰＭＣ、ＤＳＰ、ＦＰＧＡ又はＡＳＩＣが現在使用されている様々なシステム及びアプリケーションのいずれにおいて使用してよい。従って例えば、処理システム１０は、様々なタイプのコンピュータシステム又は計算を必要とする他のデバイスのいずれにおいて使用してよい。１つの考えられる実施形態では、処理システム１０を、デジタルビデオディスプレイシステムの信号処理デバイスとして使用する。

【００６２】

一実施形態では、ＰＥは、データを操作するために構成された１つ又は複数の算術論理ユニット（ＡＬＵ）、ＡＬＵを制御するために構成された１つ又は複数の命令処理ユニット（ＩＰＵ）、命令又はデータを保持するよう構成された１つ又は複数のメモリ、並びに様々な種類の多重化装置及び復号器を含んでよい。このような実施形態は多数のポート（「プロセッサポート」）を含んでよく、これらのうちのいくつかはＤＭＲに接続されるよう構成されてよく、残りは他のＰＥに接続されるよう構成されてよい。図７はＰＥの一実施形態のブロック図であり、これについて以下に更に説明する。

【００６３】

一実施形態では、ＤＭＲは、データ及び命令を保持するよう構成された１つ又は複数のランダムアクセスメモリ（ＲＡＭ）、構成可能なコントローラ、クロスバースイッチ等のネットワークスイッチ、レジスタ並びに多重化装置を含んでよい。このような実施形態は多数のポートを含んでよく、これらのうちのいくつかはＰＥに接続されるよう構成されてよく（本出願ではＰＥタイプポートと呼ばれる）、残りはＤＭＲに接続されるよう構成されてよい（本出願ではＤＭＲタイプポートと呼ばれる）。ＤＭＲ又はＰＥのいずれに又はいずれから接続されるように構成されているかにかかわらず、いずれの所定のポートに関して、このような所定のポートを通して特定のクロックサイクル内に伝送可能なデータの量は、様々な実施形態において変化し得ることに留意されたい。例えば一実施形態では、所定のポートは、１クロックサイクルにつき１データ語を伝送するよう構成されてよく、その一方で別の実施形態では、所定のポートは１クロックサイクルにつき複数のデータ語を伝送するよう構成されてよい。更に別の実施形態では、所定のポートは、時分割多重化等の技術を採用して、複数のクロックサイクルに亘って１つのデータ語を伝送してよく、これによってポートを含む物理的接続の個数を削減できる。

【００６４】

ＭＰＳ１０の一実施形態では、各ＰＥは、命令のために予約された小型ローカルメモリを含んでよく、ローカルデータストレージを極めて僅かしか有しなくてよい。このような実施形態では、各ＰＥに近接するＤＭＲは、所定のＰＥにオペランドを提供するよう構成されてよい。特定の実施形態では、多数のＰＥ命令に関して、１クロックサイクル中に、所定のＰＥが近隣のＤＭＲからオペランドを読み出し、ＡＬＵ演算を実行し、ＡＬＵ結果を所定の近隣のＤＭＲに保存してよい。これにより、１つのＰＥからのＡＬＵ結果を、そ

のクロックサイクル中に、実行の直後に複数の他のＰＥにとって利用可能とすることができる。このようにして結果を生成することにより、近隣のＰＥの実行を密接に連携したもの、即ち「緊密に連結されたもの」とすることができる。

【００６５】

本出願で使用される場合、所定のＤＭＲ又はＰＥの視点から、近隣のＤＭＲ又はＰＥは、特定のレイテンシ範囲内で上記所定のＤＭＲ又はＰＥからアクセスできるＤＭＲ又はＰＥを指す。いくつかの実施形態では、近接関係の程度を定義するレイテンシは、例えばクロック速度等の因子に応じて変化し得る。更にいくつかの実施形態では、近接の複数の度合いを定義してよく、これらの度合いは、異なる複数のアクセスレイテンシに対応してよい。例えば一実施形態では、「最も近い近接」は、データが要求されたクロックサイクルと同一のクロックサイクル中にこのデータを供給できるデバイスとして定義してよく、「次に近い近接」は、データが要求された後、１クロックサイクル以内にこのデータを供給できるデバイスとして定義してよく、これ以降も同様である。他の実施形態では、近接関係を定量化するために他のメトリクスを使用してよいことも考えられる。

【００６６】

所定のＭＰＳ実施形態では、いくつかのＤＭＲ及びＰＥは、他のＤＭＲ及びＰＥと論理的に隣接してよい。本出願で使用される場合、「論理的に隣接する」は、あるデバイスの１つ又は複数のポートが、他のデバイスの各ポートに、介在するＤＭＲ又はＰＥを通過せずに直接接続されるような、あるＤＭＲ及び別のＤＭＲ又はあるＤＭＲ及びあるＰＥといった２つのデバイス間の関係を指す。更に所定のＭＰＳの実施形態では、いくつかのＤＭＲ及びＰＥは、他のＤＭＲ及びＰＥと物理的に隣接してよい。本出願で使用される場合、「物理的に隣接する」は、あるＤＭＲ及び別のＤＭＲ又はあるＤＭＲ及びあるＰＥといった２つのデバイス間に他のいずれのＤＭＲ又はＰＥも物理的に位置していないような、これら２つのデバイス間の関係を指す。

【００６７】

いくつかのＭＰＳの実施形態では、論理的及び／又は物理的に隣接するＤＭＲ及びＰＥ等のデバイスは、近接している又は近隣のデバイスでもある。しかしながらいくつかの実施形態では、所定のデバイス間の論理的及び／又は物理的な隣接は、これら所定のデバイス間に、近接関係又は特定の度合いの近接関係を伴わないことに留意されたい。例えば一実施形態では、あるＤＭＲは、相当な距離だけ離れて位置する別のＤＭＲに直接接続されてよい。このようなペアは論理的には隣接しているものの物理的には隣接していない場合があり、一方のＤＭＲから他方のＤＭＲへの信号伝播時間は、近接のレイテンシ要件を満たすには長過ぎるものとなり得る。同様に一実施形態では、あるＤＭＲは別のＤＭＲと物理的には隣接しているものの直接接続されておらず、従って論理的には隣接していない場合がある。一方のＤＭＲから他方のＤＭＲへのアクセスは、１つ又は複数の中間ノードを横断し得、結果として発生する通過遅延は、近接のレイテンシ要件を満たすには大き過ぎるものとなり得る。

【００６８】

ＭＰＳ１０の所定の実施形態の技術及び実装形態に応じて、ＤＭＲの複数のポートの具体的な数及びＤＭＲのメモリのサイズは、全体としてのＤＭＲの所望の実行速度及びサイズとバランスを取ってよい。例えばあるＤＭＲの実施形態は、４つのＰＥタイプポート、４つのＤＭＲタイプポート、４Ｋ語のメモリを含んでよい。このようなＤＭＲの実施形態は、ダイレクトメモリアクセス（ＤＭＡ）機構を提供するよう構成されてよい。ＤＭＡ機構により、ＰＥが結果を計算している間に、所定のＤＭＲが、他のＤＭＲへ若しくは他のＤＭＲから、又はＭＰＳ１０の外部の位置へ若しくはＭＰＳ１０の外部の位置から、効率的にデータをコピーできるようになる。

【００６９】

ＭＰＳ１０の一実施形態では、データ及び命令は、複数の異なる方法のうちの１つによってＤＭＲ間で伝送されてよい。シリアルバスはＭＰＳ１０内の全てのメモリに供給されてよく、このようなバスは、外部メモリからＭＰＳ１０を初期化するために、又はＭＰＳ

データ構造の試験をサポートするために使用できる。短距離伝送に関して、所定のＰＥは、その近隣のＤＭＲへ又は近隣のＤＭＲからデータを直接移動させるようプログラムされてよい。データ又は命令を更に長距離に亘って伝送するために、ＤＭＲのネットワーク内において通信経路を動的に生成及び破壊してよい。

【００７０】

このような比較的長距離のデータ伝送のために、ＭＰＳ１０内の相互接続されたＤＭＲのネットワークは、通信経路用のスイッチドルーティングファブリック（ＳＲＦ）を構成してよい。このような実施形態では、ＳＲＦ内の通信経路を管理するために少なくとも２つの方法が存在し得る。第１の方法はグローバルプログラミングによるものであり、経路はソフトウェア制御によって（例えば人であるプログラマによって又はルーティング能力を有するコンパイラによって）選択でき、命令をＤＭＲ構成コントローラにコードしてクロスバーを適切にプログラムできる。経路を生成するために、その経路に沿った全てのＤＭＲを、特定のルーティング機能によって明白にプログラムしてよい。経路が頻繁に生成及び破壊される動的な環境において、多数のクロスバー構成コードが必要となり得、これらのストレージは潜在的に限定されたＤＭＲ　ＲＡＭリソースを消費し得る。

【００７１】

通信経路を管理するための第２の方法は、「ワームホールルーティング」と呼ばれる。ワームホールルーティングを実装するために、各ＤＭＲは、ステアリング機能のセットと、メッセージと呼ばれる語のシーケンスの、ＳＲＦを通る進行を停止及び再開させる機構とを含んでよい。全ての通信経路がステアリング機能を共通して使用及び再使用してよいため、ＤＭＲ　ＲＡＭを占有し得る構成コードの量は、上述のグローバルプログラミング法よりも大幅に小さくなり得る。ワームホールルーティング法に関して、ここでもまたソフトウェア制御を使用して、経路が使用する特定のリンクを選択してよいが、経路生成（本出願ではセットアップとも呼ぶ）及び破壊／リンク解放（本出願ではティアダウンとも呼ぶ）のプロセスは、最小のソフトウェア介入を伴ってハードウェア内に実装できる。

【００７２】

経路上でのデータ語の潜在的な損失を防止するために、ＭＰＳ１０の実施形態は、経路に沿った受信器と伝送器との間のフロー制御を実装してよい。フロー制御は、対応する受信器がそれ以上データを受信できない場合に伝送器を停止させることができ、また対応する受信器がデータを受信する準備ができた状態となった場合に伝送器を再開させることができる機構を指す。経路上でのデータのフローの停止及び再開は、ワームホールルーティングにおけるメッセージの進行の停止及び再開と多くの類似点を有し、これら２つは統合されたスキーム内で組み合わせることができる。

【００７３】

一実施形態では、ＭＰＳ１０は複数のＰＥ及びＤＭＲを含んでよく、これらＰＥは同一であってよく、これらＤＭＲは同一であってよく、均一なアレイ内で接続されてよい。均一なアレイにおいて、ＰＥの大半は同一であってよく、ＰＥの大半はそれぞれ、ＤＭＲへの同一数の接続を有してよい。また均一なアレイにおいて、ＤＭＲの大半は同一であってよく、ＤＭＲの大半はそれぞれ、他のＤＭＲ及びＰＥへの同数の接続を有してよい。ＭＰＳの一実施形態のＰＥ及びＤＭＲは、実質的に均一な様式で散在していてよい。本出願で使用される場合、「実質的に均一な散在」は、ＤＭＲに対するＰＥの比が、アレイの複数の部分領域の大半に亘って一貫している構成を指す。

【００７４】

実質的に均一な様式で配設された均一なアレイは、予測可能な相互接続パターンを提供する、アレイ全体に亘ってソフトウェアモジュールを再使用できるようにする等、特定の有利な特徴を有し得る。一実施形態では、均一なアレイにより、少数のＰＥ及びＤＭＲの例を設計及び試験できる。システムは、１つのＤＭＲ及び１つのＰＥを備えるユニットを製作し、このユニットを複数回反復する、即ち「タイルリングする（ｔｉｌｉｎｇ）」ことによって、組み立てることができる。このようなアプローチは、共通のシステム要素の再使用により、設計及び試験コストを低下させることができる。

【００７５】
また、ＰＥ及びＤＭＲの構成可能な性質により、物理的に均一なアレイ上で極めて広範な不均一挙動が発生するようにプログラムできるようにすることができることに留意されたい。しかしながら代替実施形態では、ＭＰＳ１０は不均一なＤＭＲ及びＰＥユニットで形成してもよく、これらユニットは、規則的な若しくは不規則なアレイに、又はランダムにさえ接続してよい。一実施形態では、ＰＥ及びＤＭＲの相互接続は、例えば集積回路（ＩＣ）、セラミック基材又はプリント回路基板（ＰＣＢ）上の回路トレースとして実装してよい。しかしながら代替実施形態では、このような相互接続は、例えば電磁エネルギ（即ち電波若しくは光学エネルギ）、無線（即ち無誘導）エネルギ、粒子（電子ビーム等）、又は分子上の電位のための導波路といった様々な細密通信リンクのいずれであってよい。

【００７６】
ＭＰＳ１０は単一の集積回路上に実装してよい。一実施形態では、複数のＭＰＳ集積回路を組み合わせて、より大型のシステムを生成してよい。ＭＰＳ１０の所定の実施形態は、シリコン集積回路（Ｓｉ‐ＩＣ）技術を用いて実装してよく、またこのような技術の固有の特徴の原因となる様々な特徴部分を採用してよい。例えばＳｉ‐ＩＣチップ上の回路は薄い平面内に制限され得る。これに対応して、ＭＰＳ１０の所定の実施形態は、図３に示すようなＰＥ及びＤＭＲの２次元アレイを採用してよい。しかしながら、ＰＥ及びＤＭＲの異なる構成を含む、ＭＰＳの代替実施形態が考えられる。

【００７７】
更に、Ｓｉ‐ＩＣチップ上で使用可能な配線密度は、このようなチップ間でのものより遥かに高くてよく、各チップは、オンチップ信号とオフチップ信号とをインタフェース接続するために特別な入力／出力（Ｉ／Ｏ）回路の周縁部を有してよい。これに対応して、ＭＰＳ１０の所定の実施形態は、チップのコアにあるＰＥ及びＤＭＲの均一なアレイと、チップの周縁部に沿った修正されたＰＥ／ＤＭＲユニットとで構成された、若干不均一なアレイを採用してよい。しかしながら、均一なＰＥ／ＤＭＲユニットと修正されたＰＥ／ＤＭＲユニットとの異なる構成及び組み合わせを含む、ＭＰＳの代替実施形態が考えられる。

【００７８】
また、Ｓｉ‐ＩＣ回路によって実施される計算動作により熱が生成される場合があり、これはＩＣパッケージングによって除去できる。ＩＣパッケージングの増大には追加の空間が必要となり得、ＩＣパッケージングを通る、及びＩＣパッケージングの周囲の相互接続は、経路の長さに比例する遅延を招き得る。従って上述のように、極めて大型のＭＰＳは、多数のチップを相互接続することによって構成してよい。このような多チップＭＰＳ実施形態のプログラミングは、チップ間信号遅延がチップ内遅延より遥かに長いことを考慮したものであってよい。

【００７９】
所定のＳｉ‐ＩＣ　ＭＰＳ１０の実施形態では、単一のチップ上に実装できるＰＥ及びＤＭＲの最大数は、所定のＳｉ‐ＩＣ技術によって可能な小型化、並びに各ＰＥ及びＤＭＲの複雑さによって決定できる。このようなＭＰＳの実施形態では、ＰＥ及びＤＭＲの回路複雑性は、標的レベルの計算スループットを達成することを条件として最小化できる。このように最小化されたＰＥ及びＤＭＲを、本出願では「合理化された（ｂｅｉｎｇ　ｓｔｒｅａｍｌｉｎｅｄ）」と呼ぶ場合がある。ＭＰＳ１０の一実施形態では、ＰＥに関するスループットの標的レベルは、同一のＳｉ‐ＩＣ技術において作製される最良のデジタル信号プロセッサ（ＤＳＰ）の算術実行ユニットのスループットの標的レベルに匹敵するものであってよい。しかしながら、標的ＰＥスループットに関する代替的な基準を使用してよい他のＭＰＳの実施形態が考えられる。

【００８０】
いくつかの実施形態では、ＭＰＳ１０は、ＤＳＰ及びＦＰＧＡアーキテクチャの最良の特徴を採用してよい。ＤＳＰと同様、ＭＰＳ１０は、多数の処理ユニット及びオンチップ

メモリを有するプログラム可能なチップであってよい。しかしながらＤＳＰに対して、ＭＰＳ処理ユニットは合理化されてよく、個数が多くてよく、またＭＰＳ処理ユニット間のデータ移動並びにオンチップ及びオフチップのデータ移動の帯域幅を最大化するために、新規の方法で相互接続されてよい。ＤＳＰより多くの処理ユニットを有することにより、ＭＰＳ１０は、単位時間あたりに更に多くの多重化を実施でき、合理化された処理ユニットはエネルギ使用を最小化できる。内部並列処理を行う多くのＤＳＰは、バス指向性アーキテクチャであってよい。いくつかの実施形態では、ＭＰＳ１０はバスを含まなくてよく、ＳＲＦ内に埋め込まれた、例えばＤＭＲ内で近接した共有ローカルメモリを含んでよく、これにより、バス指向性アーキテクチャより有意に高い総帯域幅を提供できる。

【００８１】

ＦＰＧＡアプローチと比べて、いくつかのＭＰＳの実施形態はより粗粒化されていてよい。例えばＭＰＳの一実施形態では、演算は自然語長（例えば１６ビット）を有してよく、自然語長の倍数のデータを用いて実施した場合、計算はより効率的となり得る。いくつかのＭＰＳの実施形態では、ＰＥ及びＤＭＲは、ＦＰＧＡにおいて実現される同等の構造よりも密であってよく、これにより、比較的短い平均配線長、低い配線抵抗、少ないエネルギ使用を得ることができる。ＦＰＧＡ実装形態とは対照的に、いくつかのＭＰＳの実施形態では、ＭＰＳ内の全てのＡＬＵはプロセッサの一部（即ちＰＥ）であってよく、これは、オペランドのフェッチ、及びＤＭＲ内にある周囲の迅速なメモリへの結果のライトバック（ｗｒｉｔｅ　ｂａｃｋ）を促進できる。ＡＬＵ、フェッチ及びライトバック動作に関するタイミング及びクロックスキューの問題は、ＩＣチップの設計中に一度に解決でき、またＦＰＧＡ実装形態において典型的であるように、新規のアプリケーションそれぞれにおいて再度解決する必要はない。

【００８２】

ＭＰＳトポロジ及び通信

図３に示すＭＰＳ１０は、図示したようにＰＥ間にＤＭＲを散在させることによって、迅速なメモリへの十分な接続をＰＥに提供できる。このような構成は、分離型（即ち非散在型）構成に対して、所定のＰＥがＤＭＲ内のメモリにアクセスするために必要な時間を削減でき、本出願ではこれを散在型グリッド構成と呼ぶ場合がある。図３の実施形態では、ＰＥとＤＭＲとの比はおおよそ１：１である。しかしながら、異なるＰＥとＤＭＲとの比を含んでよい他のＭＰＳの実施形態が考えられる。

【００８３】

ＤＭＲとＰＥとの間の接続は、図３には明示されていない。というのは、異なるタイプ及び数の接続を用いる、多数の可能な接続スキームが存在し得るためである。

【００８４】

図４‐ＭＰＳ接続スキーム

図４は、ＭＰＳ接続スキームの一実施形態を示すブロック図である。ＭＰＳ接続スキーム２０は、複数のＤＭＲ及びＰＥを含み、図３のＭＰＳの一部分を説明するものであり得る。ＭＰＳ接続スキーム２０では、各ＰＥは４つの近隣のＤＭＲに接続され、各ＤＭＲは４つの近隣のＰＥ及び４つの近隣のＤＭＲに接続される。従ってＭＰＳ接続スキーム２０は、上述のＰｌａｎａｒＡ接続スキームを説明するものであり得る。

【００８５】

ＭＰＳ接続スキーム２０において高帯域幅ポートをサポートするために、ポート間（ＰＥ‐ＤＭＲ間又はＤＭＲ‐ＤＭＲ間）の接続は短くてよく（即ち近隣に限定されてよく）、また語の幅であってよい（これは、接続のデータ部分の導電体（ライン）の数が、ＡＬＵオペランド中で使用されるビット数と同一であり得ることを意味する）。ＰＥ‐ＤＭＲ間接続はアドレスラインを含んでよい。ＤＭＲ‐ＤＭＲ間接続は必ずしもアドレスラインを有しなくてよいが、フロー制御用ラインを有してよい。

【００８６】

ＰＥノードを簡素なまま維持することによって、大型アレイ（例えばＭＰＳの一実施形態では、１６行×１６列＝２５６個のＰＥ）を、単一のＶＬＳＩ　ＩＣ上に、それほど高

くないコストで配置できる。好適なＶＬＳＩ技術は、シリコン又は他の半導体内に双極トランジスタを有する又は有しない、相補型金属酸化膜半導体（ＣＭＯＳ）電界効果トランジスタを含んでよいが、これに限定されない。

【００８７】

いくつかのＭＰＳの実施形態では、ノード間の通信はプログラマの制御下であってよい。ＭＰＳにおいて、各ＰＥは近隣のＤＭＲとデータ／命令を通信してよく、また任意にこれらのＤＭＲを通して他のＤＭＲ及びＰＥへと通信してよい。これは、短距離に亘る少量のデータの伝送に関して非常に効率的である。しかしながら、より大きなデータのブロック又はより長い距離に関しては、ＤＭＡエンジンを用いてデータを移動させることにより、ＰＥを、ＡＬＵ演算を実施できるよう自由にすると、より効率的である。

【００８８】

比較的長距離のブロックの移動に関して、いくつかのＭＰＳの実施形態は、ＰＥを関与させずにＤＭＲ間でメモリ‐メモリ間伝送を行うための手段を提供できる。ＰＥは、近隣のＤＭＲのＤＭＲタイプポートに、このようなポートと関連する特別なＳＭアドレスによって、間接的にアクセスできる。これにより、ＰＥは、メッセージを送信するための新規の経路を生成して、後にこの経路をティアダウンし、あるいはメッセージを受信してよい。ＰＥはまた、伝送されるデータのブロックを近隣のＤＭＲのＳＭバッファ内に保存してよく、続いてこの近隣のＤＭＲに、ＤＭＡの動作を、上記動作と関連する特別なＳＭアドレスによって開始させるよう指示してよい。これによりＰＥは、近隣のＤＭＲがデータのＤＭＡ伝送を調整している間に他のタスクを進めることができるようになる。

【００８９】

ＭＰＳの様々な実施形態は、有用なアルゴリズムを実行するための有利な環境を提供できる。（例えば画像データを分析するための）関心対象となるアルゴリズムは、ＡＬＵのフロー図へと分解できる。各フロー図はＭＰＳアレイ上にツリー、格子又は多数のフィードバック／フィードフォワード経路を含むいずれの任意のネットワークとしてマッピングできる。１つのＡＬＵの有限精度は、複数のＰＥ及びＤＭＲを組み合わせることによって多数の語の正確な結果を得られるように拡張されてよい。フロー図をＭＰＳにマッピングする際、ノード間の距離に比例するＰＥ／ＤＭＲノード間の通信遅延は上昇し得る。また、通信キューが大きい場合又は再構成が頻繁に行われる場合、マッピングは各ノードにおいてより多くのメモリを必要とし得る。これらの因子は、通信遅延、キュー及び再構成を考慮に入れることができる綿密なプログラミングによって補償できる。

【００９０】

シストリックアルゴリズムは、ＭＰＳの様々な実施形態に対して特に効率的にマッピングを行うことができるアルゴリズムのクラスを表す。シストリックアルゴリズムは、行列演算、画像処理、信号処理における幅広い応用のために開発された。シストリックアルゴリズムでは、多数のプロセッサが同期して協働し、困難な計算を実施できる。理想的なアルゴリズム実装形態では、各プロセッサは、アルゴリズムが必要とされる限り、同一の演算（又は複数の演算の短いループ）を何度も何度も実施でき、データは、データ語のバランスの取れた生成及び消費を伴って、近隣の接続によって、プロセッサのネットワークを通って流れることができる。生成される各中間結果データ語が後続の計算によって即座に消費される場合、必要なメモリの量を最小化できる。シストリックアルゴリズムの利点は、合理化されたプロセッサを使用でき、メモリ要件を最小化でき、標準的な低コストのＶＬＳＩ技術を使用して高い算術演算速度を達成できる能力を含んでよい。

【００９１】

ＭＰＳのある実施形態は、１つのチップ及びＭＩＭＤアーキテクチャ全体につき多数のプロセッサを有してよく、上記ＭＩＭＤアーキテクチャは、ＳＩＭＤシステム及び分散型ＭＩＭＤシステムといった他のクラスのシステムの演算をエミュレートするよう構成されてよい。いくつかの実施形態では、ＭＰＳはチップの異なる領域で異なるアルゴリズムを同時に実行できる。またいくつかの実施形態では、電力を節約するために、プログラマは、少なくともいくつかのＰＥ及びＤＭＲへのクロックを選択的に有効化及び無効化できる

。従って、使用されていないＰＥ及びＤＭＲを無効化できる。
【００９２】
ＰＥ／ＤＭＲファブリック
図５は、ＭＰＳファブリックの一実施形態を示す更に詳細な図である。図５では、各ＰＥは４つのＤＭＲに取り囲まれ、これを用いて各ＰＥはメモリ要求及びメッセージを通信できる。各ＤＭＲは、ファブリックの縁部に近い場所を除いて、４つの他のＤＭＲに取り囲まれ、上記縁部では各ＤＭＲはチップＩ／Ｏポートに隣接してよい。各ＤＭＲは近隣のＤＭＲ又はチップＩ／Ｏポートと通信して、通信経路をセットアップし、上記通信経路上でメッセージを送受信できる。
【００９３】
ＭＰＳの演算
図６は、図５に示すアーキテクチャ例に従って、９×９のＤＭＲ（円）のアレイと共に均一に散在する８×８のＰＥ（正方形）のアレイからなる、例示的なＭＰＳを示す。プログラムは、ＰＥに割り当てられるタスクへとコンパイルできる。第１の例示的なプログラムは、タスクＩＤの＝６２でコンパイルされており、アレイの左上隅の特定のＰＥに割り当てられている。変数ｕ、ｖ、ｗは、プログラムソースコード中の宣言された通信変数であり、隣接するＤＭＲの特定のメモリアドレスに割り当てられる。ｕ、ｖはＩ／Ｏポートのためのバッファであり、ｗは、関連するＤＭＲとのオンチップネットワーク通信のためのバッファである。第２の例示的なプログラムは、タスクＩＤ＝７１でコンパイルされており、アレイの内部の特定のＰＥに割り当てられている。変数ｘは宣言された通信変数であり、図示されているＤＭＲに割り当てられている。変数ｘに関連する通信経路は、ｘが割り当てられたＤＭＲから、他のＤＭＲを通って、最上行のＩ／Ｏポートへと続く。図示したように、これら２つの例示的なプログラムは互いに通信しないが、これらは、タスク７１に別の通信変数を追加し、タスク７１のＤＭＲと、タスク６２に隣接するＤＭＲの変数ｗとの間に経路を追加することによって、容易に通信させることができる。
【００９４】
動的に構成可能なプロセッサ
図７は、動的に構成できる処理要素（ＰＥ）の例を示すブロック図である。ＰＥ３００は、図３～６に示す動的に構成可能なプロセッサ（ＤＣＰ）を説明するものであってよい。ＰＥ３００は、その命令メモリを再プログラムする方法を有しているため、動的に構成可能である。図７ではこれは、シリアルバスインタフェースからのロード経路を通して行われる。シリアルバスは、全てのＤＭＲ及びＰＥメモリをバスコントローラと接続する、２次相互接続ネットワークであってよく、バスコントローラは１つ又は複数のＰＥによって及びチップＩ／Ｏポートによってアクセスできる。ＰＥを再プログラムするためには、ＰＥを待機状態にしてよく、続いて命令メモリを新規のプログラムで上書きし、プログラムカウンタを設定し、その後ＰＥを待機状態から解放して上記新規のプログラムの実行を開始させてよい。ＰＥ３００は、少なくとも１つの算術論理ユニット（ＡＬＵ）３２０を制御するために連結された命令処理ユニット（ＩＰＵ）３１０を含む。ＰＥ３００はまた、複数の多重化装置（本出願ではｍｕｘとも呼ばれる）に連結された複数のデータ入力ポート３０１も含み、上記多重化装置は、ＡＬＵ３２０のための少なくとも第１及び第２のオペランド入力を選択するために連結される。ＰＥ３００は更に、ＡＬＵ３２０から結果データを受信するためにｍｕｘを介して連結された複数のデータ出力ポート３０２と、命令処理ユニット３１０からアドレスデータを受信するために連結された複数のアドレスポート３０３とを含む。
【００９５】
アドレスポート３０３は、近隣のＤＭＲのメモリに対する読み書きを行うためにアドレスを搬送するよう構成されてよい。データ入力ポート３０１及びデータ出力ポート３０２は、近隣のＤＭＲから及び近隣のＤＭＲへデータを搬送するよう構成されてよい。図示されているＰＥ３００の実施形態では、データ入力ポート３０１、データ出力ポート３０２、アドレスポート３０３はそれぞれ４つのポートを含み、これは図５の例示的なアーキテ

クチャと一致する。

【００９６】

図７の例では、米国特許第７４１５５９４号に記載されているような従来技術において典型的である、単一のＡＬＵ３２０が示されている。しかしながら、１つのＰＥにつき更に多くのＡＬＵが存在し、これによってＰＥが遥かに高い処理スループットのための潜在能力を有する、代替実施形態が考えられる。その例を本明細書中のこれ以降において示し、議論する。

【００９７】

ＰＥ３００は、データ語に対する算術／論理ユニット演算を実施するよう構成され、選択される演算は、ＩＰＵ３１０が処理している現在の命令に依存する。柔軟なプログラミングをサポートするために、ＩＰＵ３１０は、複数のアドレス指定可能な位置を含む少なくとも１つの命令メモリ３１２、命令復号器３１４、アドレス生成器３１６を含んでよく、これらはそれぞれ様々な相互接続機構を介して相互接続される。他の実施形態では、ＩＰＵ３１０は２つ以上の命令メモリを含んでよいこと、又は追加の機能性を含んでよいことが考えられる。他の実施形態では更に、ＩＰＵ３１０に示した機能性を異なるタイプの複数のユニットに分割してよいこと、又は単一の機能ユニットで実装してよいことも考えられる。

【００９８】

ＩＰＵ３１０は、データ入力ポート３０１に連結されたプログラムロード経路を介して、命令メモリ３１２内での保存のためにプログラムデータを受信するよう構成されてよい。命令メモリ３１２はまた、グローバルシリアルバス（図示せず）を介して読み書きしてよい。命令復号器３１４による特定の命令の復号化に応じて、ＩＰＵ３１０は、データ入力ポート３０１及びデータ出力ポート３０２に連結された様々なｍｕｘを制御して、近隣のＤＭＲへ及び近隣のＤＭＲからデータを案内するよう構成されてよい。ＩＰＵ３１０は更に、アドレス生成器３１６が生成したアドレスを、アドレスポート３０３を介して近隣のＤＭＲへと搬送することによって、例えばそこに位置するＲＡＭに対して読み出し又は書き込みを行うよう構成されてよい。アドレス生成器３１６はまた、命令メモリ３１２からフェッチされ、命令復号器３１４によって復号化される、次の命令アドレスを生成するよう構成された、プログラムカウンタレジスタ（図示せず）も含んでよい。

【００９９】

一実施形態では、ＰＥ３００はデータレジスタファイル、データキャッシュ、又はデータオペランド若しくは結果データ用のいずれのローカルストレージを含まなくてよい。このような実施形態では、ＰＥ３００は、データオペランドの読み出し及び結果データの書き込みを行うことができる迅速なストレージ媒体としてＰＥ３００が直接接続されているＤＭＲに含まれるメモリを利用するよう構成されてよい。いくつかの実施形態では、所定のＰＥは異なる近隣のＤＭＲから同時に又は異なる時点に異なるデータを得ることができる。以下に更に詳細に説明するように、いくつかの実施形態では、所定のＰＥは、上記所定のＰＥが直接接続されていないＤＭＲ内のデータの読み出し及び書き込みを、このような遠隔ＤＭＲから上記所定のＰＥの近隣のＤＭＲへの経路を確立することによって行うよう構成されてもよい。

【０１００】

ＰＥ３００が実装する命令は、算術演算及び論理演算並びにメタ命令をサポートしてよい。ＰＥ命令は、２つのオペランド及び１つの結果に関するメモリをアドレス指定するために十分なビット長を有してよく、これにより１クロックサイクルでこれらの値の読み出し及び書き込みを行うことができる。

【０１０１】

他の実施形態は、追加の命令又は異なる命令のセットを実装してよいことに留意されたい。いくつかの実施形態では、１つ又は複数のデータオペランドを必要とする所定の命令の実行中、所定のＰＥは、近隣のＤＭＲメモリに直接アクセスして上記必要なオペランドにアクセスするよう構成されてよい。

【０１０２】
ＰＥ３００はメタ命令を実行するよう構成されてよい。本出願で使用される場合、メタ命令は、命令メモリ３１２等のＰＥ命令メモリに保存された命令に対する演算を実施できる命令を指す。基本的なメタ命令は、近隣のＤＭＲのＲＡＭから命令メモリ３１２をロードすること（即ちオーバレイをロードすること）であってよい。ＤＭＲメモリから命令メモリをロードすることにより、データと命令との間のメモリの分割をソフトウェアプログラミングによって決定できる。従ってアプリケーションのプログラマは、自身のソフトウェアを、利用可能なメモリの最良の利用に関して最適化できる。いくつかの実施形態では、ＰＥ３００は、ＩＰＵ命令メモリを修正できる他のメタ命令を含んでよく、又は例えば試験、エラー分析及び／若しくはエラー修復のためにＤＭＲメモリに命令メモリをセーブしてよい。

【０１０３】
ＡＬＵ３２０は、特定のＰＥ３００の実施形態でサポートされる命令によって定義される演算を含む、少なくとも固定小数点数系に関する演算を実施するよう構成されてよい。例えば一実施形態では、ＡＬＵ３２０は固定小数点型加算、減算、乗算、積和、論理及びシフト演算を実施するよう構成されてよい。いくつかの実施形態では、ＡＬＵ３２０は、拡張精度演算をサポートするために、過去の計算から得られたキャリービットを保持するよう構成されてよい。他の実施形態では、ＡＬＵ３２０は、浮動小数点型演算、又は特定のアルゴリズムを実装するために選択された特定目的のための演算を実施するよう構成されてよい。

【０１０４】
図８‐並列実行のためのソフトウェアを開発するための方法のフローチャート
図８は、一実施形態による、マルチプロセッサシステム上での並列実行を標的としたソフトウェアを開発するための方法を示す。図５に示す方法は、とりわけここで説明するコンピュータシステム又はデバイスのいずれと併用してよい。様々な実施形態では、図示されている方法の要素のうちのいくつかは同時に、若しくは図示されているものとは異なる順序で実施されてよく、又は省略されてよい。所望に応じて追加の方法の要素を実施してもよい。図示したように、この方法は以下のように動作し得る。

【０１０５】
ｘ８０２に示すように、例示的な一実施形態では、例えばマルチプロセッサシステム上での展開及び実行のために標的化されたアプリケーションの、所望のシステムの複数のビューを指定する入力を受信してよい。各ビューは、システムの各態様を提示又は指定してよく、またこれらビューは総体として、効率的な並列実行のためにマルチプロセッサシステムが展開できる実行可能なプログラムを生成するためにコンパイラ（又は他のソフトウェアツール）が使用できる情報を提供してよい。

【０１０６】
この入力は、様々な形態のいずれで、また様々なツールのいずれを介して受信されてよい。例えばいくつかの実施形態では、入力はユーザによって提供されてよく、即ちユーザ入力であってよい。他の実施形態では、入力は所望に応じて別のシステム又はプロセス、ストレージ媒体等から受信されてよい。更に入力はローカルに提供されてよく、又はローカルエリアネットワーク（ＬＡＮ）若しくはインターネット等の広域ネットワーク（ＷＡＮ）を介して受信されてよい。例示的な一実施形態では、ユーザはスプレッドシート中でビューを指定してよい。別の例示的実施形態では、ユーザはウィザード、即ちグラフィカルユーザインタフェース（ＧＵＩ）への入力を行ってよく、これは指定プロセスを通して、例えばプロンプト、有用な提案等によってユーザをリードする。更なる実施形態では、ユーザは入力／指定プロセスを管理するためにチェックリストを使用してよく、指定される各アイテム、例えばビュー、サブビュー等がチェックリストに挙げられており、チェックリストは各チェックリストアイテムが指定されているか（されていないか）を示す。別の例示的実施形態では、１つ又は複数のテンプレート又はフォームを提供してよく、ユーザはビューを指定する情報で上記テンプレート又はフォームを埋めてよい。

【０１０７】
各ビューは、アプリケーションの動作又は実行の各態様を含むか又はその表現となってよい。多数のビューはメモリビュー、通信ビュー、制御ビュー、処理ビューを含んでよい。他のビューも所望に応じて定義又は指定してよい。一実施形態では、各ビューは、ビューの更に詳細な属性を指定する複数のサブビュー（又は「ファセット」）を含むか又は指定してよい。例えば各ビューは、サイズ、挙動及びアクセシビリティサブビュー（又はファセット）を含んでよく、各サブビューは、各サブビューがその一部となっているビューに関連する特定の意味を有してよい。
【０１０８】
従って例えば、メモリビューは：アプリケーションによる使用に必要な又は利用可能なメモリのサイズ（又は量）、即ちデータを処理するためにアプリケーションが使用するメモリ構造サイズ；メモリの挙動、即ちメモリ構造が所定のタイミングで挙動する様式；並びにメモリのアクセシビリティ、即ち例えばアプリケーション及び／又はシステムによるメモリ構造のアクセシビリティを指定してよい。
【０１０９】
同様に、入力は：通信ビュー（通信ビューの各サブビュー、例えば通信サイズ、挙動、アクセシビリティを定義することを含む）；制御ビュー（制御サイズ、挙動、アクセシビリティを含む）；並びに処理ビュー（処理サイズ、挙動、アクセシビリティを含む）を定義又は指定してよい。
【０１１０】
いくつかの実施形態では、ビュー又はサブビューのうちのいくつかは、他のサブビュー又はビューの指定によって、例えば同一の又は異なるビューのサブビューによって、自動的に定義又は指定され得ることに留意されたい。従って例えば、メモリサイズ、挙動及びアクセシビリティが指定されると、通信挙動が自動的に指定され得る。別の様式で考えると、いくつかの実施形態では、ビューは、方程式の数が変数の数を超える線形方程式の優決定系と同様に、「過剰に指定される」か又は「過剰に決定される」場合がある。
【０１１１】
本出願で使用される特定の用語又は標識は単なる例示であること、並びに所望に応じて本出願で開示される新規の構成要素、情報及びプロセスに対していずれの名称を使用してよいことに留意されたい。例えば所望に応じて、ビュー又は態様はモデル等と呼ばれる場合もあり、本出願で開示されるサブビューはサブモデル、ファセット、プロパティ等と呼ばれる場合もある。
【０１１２】
システムのビューが指定又は定義されると、これらのビューを表す情報は、ｘ８０４に示すように、アプリケーションのソースコード内に含まれ得る。この包含は、広範な方法のうちのいずれにおいて実施され得る。例えばいくつかの実施形態では、情報はアプリケーションの１つ又は複数のヘッダファイルに含まれ得る。他の実施形態では、情報は、アプリケーションプログラム要素又は構成要素の中でとりわけ、１つ又は複数の動的リンクライブラリ（ＤＬＬ）又はマクロ定義に含まれ得る。より一般には、ビューを表す情報は、所望に応じていずれの様式及びいずれの形態でアプリケーションソースコードに組み込まれてよい。
【０１１３】
ｘ８０６では、ソースコードは例えば、コンパイラ又は他のツールによって処理でき、これはシステムに関して指定又は定義された複数のビューを表す情報を分析することを含む。例えば一実施形態では、コンパイラは、アプリケーションソースコード内の複数のビューを表す情報を認識するよう構成してよく、またこの情報を抽出及び分析してよい。他の実施形態では、コンパイラは情報を原位置で分析してよい。
【０１１４】
ｘ８０８に示すように、実行可能なプログラムは上記処理に基づいて生成してよく、ここで上記実行可能なプログラムは、効率的な並列実行のためにマルチプロセッサシステム

に展開可能である。換言すると、コンパイラは、指定されたビューの分析を含むｘ８０６の処理に基づいて、実行可能なプログラムを生成してよい。

【０１１５】

従って、本技術の実施形態は、上述のようなソフトウェア開発への従来技術のアプローチの様々な欠点に対処でき、これによってユーザは、システムの動作、例えばマルチプロセッサシステム上でのアプリケーションの動作に関する様々な要件又は制約を指定でき、これら指定された要件又は制約をコンパイラ（又は他のツール）が使用して、システム上で効率的に実行できる実行可能なコードを生成してよい。

【０１１６】

以下に、上述の技術の様々な例示的実施形態を提示する。

【０１１７】

１．マルチプロセッサシステム内での並列実行のためのアプリケーションソフトウェアを開発するための方法であって、上記方法は、コンピュータが：第１の入力に応答して所望のシステムの複数のビューを指定するステップ（ここで上記複数のビューは：メモリビュー；通信ビュー；制御ビュー；及び処理ビューを含む）；アプリケーションプログラムのソースコードに上記複数のビューを表す情報を含めるステップ；アプリケーションプログラムのソースコードを処理するステップ（上記複数のビューを表す情報を分析するステップを含む）；並びに上記処理ステップに基づいて、実行可能なプログラムを生成するステップ（上記実行可能なプログラムは並列実行のためにマルチプロセッサシステムに展開可能である）を実施することを含む、方法。

【０１１８】

２．複数のビューを指定する上記ステップは、各ビューに関して：サイズ；挙動；及びアクセシビリティを指定するステップを含む、実施形態１の方法。

【０１１９】

３．上記入力は：ウィザード；グラフィカルユーザインタフェース；スプレッドシート；又はデータファイルのうちの少なくとも１つが受信するユーザ入力を含む、実施形態１の方法。

【０１２０】

４．ソースコードに上記複数のビューを表す情報を含める上記ステップは：アプリケーションプログラムの１つ若しくは複数のヘッダファイル；アプリケーションプログラムに関するマクロ定義；又は１つ若しくは複数の動的リンクライブラリ（ＤＬＬ）のうちの１つ又は複数に上記情報を含めるステップを含む、実施形態１の方法。

【０１２１】

５．上記処理ステップ及び上記生成ステップはコンパイラによって実施される、実施形態１の方法。

【０１２２】

６．上記マルチプロセッサシステムはＨｙｐｅｒＸアーキテクチャを備える、実施形態１の方法。

【０１２３】

更なる実施形態

以下に、上述の技術の更なる例示的実施形態の説明を提示するが、開示される実施形態は単なる例示であり、本技術の実装をいずれの特定の形態、機能又は外観に限定することを意図したものではないことに留意されたい。特定の（非限定的な）使用のケースとして２つの例示的実施形態：撮像フィルタ及び有限インパルス応答（ＦＩＲ）フィルタを提示する。

【０１２４】

定義されたシステムのビュー及びプロセス

あるシステムにおいてデータは、上記システムがリアルタイム型であるかそうでないかに関わらず、無線受信機、画像センサ又は他の入力収集デバイスのいずれかからの（実際の又は概念上の）ストリームとして受信されることが多い。受信されるデータは、アルゴ

リズム及びその代表データ構造に対して自然である方法で処理されることが望ましい。これは、処理のために、サンプル形態、ブロック形態又はハイブリッド形態のデータ構造を取り扱う能力を必要とし得る。これはまた、処理システムの通信及びメモリアーキテクチャが、様々なアルゴリズム及びデータ構造システムをサポートできるよう、リアルタイムに動的であり、かつ適合可能であることを必要とし得る。

【０１２５】

例示的な使用のケース：撮像フィルタ

一例として、等式１によって特徴付けられる例示的な撮像フィルタを考える。

【０１２６】

【数１】

$$y[r,c,t] = \sum_{r=-M}^{r=M} \sum_{c=-N}^{c=N} h[r,c,t]x[r,c,t]$$

等式１

【０１２７】

ここでｔは時間であり、ｒは行インデックスであり、ｃは列インデックスであり、ｘ［ｒ，ｃ，ｔ］は入力ピクセルであり、ｈ［ｒ，ｃ，ｔ］はフィルタ係数であり、ｙ［ｒ，ｃ，ｔ］は出力ピクセルである。

【０１２８】

合計は２Ｍ＋１行及び２Ｎ＋１列に亘っており、従ってフィルタの寸法的なメモリサイズは（２Ｍ＋１）×（２Ｎ＋１）であり、最小のレイテンシが望ましい。

【０１２９】

撮像フィルタシステムを用いた、図８の方法の記述

例示的な撮像フィルタリングプロセスを、図９Ａに図式的に示す。このアプローチでは、ピクセル（ピクセルデータ）のデータストリームは通常、行毎に受信される。フィルタ及び／又は境界条件のデータ要件を満たすために十分な行数のデータが受信されると、図示されているように、データを水平に横断するように２次元フィルタを走らせて、フィルタリングされた画像を生成してよい。図９Ａのアプローチは図１０においてブロック図の形態で表されており、この図１０は、単一のプロセスを有する画像フィルタリングのブロック図を示す。

【０１３０】

画像データをより迅速に処理するために、一般的なスキームは、画像を垂直方向に分割するものであり、各垂直セグメントのデータは、ピクセル毎、行毎に（処理リソースに）受信されて、図９Ｂに示すように並列にフィルタリングが実施される。図９Ｂの並列アプローチは図１１においてブロック図の形態で表されており、この図１１は、複数のプロセスを有する画像フィルタリングのブロック図を示す。

【０１３１】

ＭＰＩをサポートするＡＮＳＩ　Ｃ言語ベースのソフトウェアプログラミングモデルにおいてスレッドを処理するための本技術の実施形態を実装する例示的なプログラム（フィルタカーネルのコード例）は、以下のように書くことができる：

```
//* filter_kernel.c
//* example hx3xxx pseudo code for 2D filter

#include <mpx.h>
#include <math.h>
#include "filter_system.h"
```

```
// initialize variables
int 2d_lin_buffer [(ROW+3)*COL];

// initialize 2D circular line buffer on 2d_buffer
void init_2D_circ_lin_buf ( .... )
{ .... }

// update 2D circular line address on 2d_buffer
void update_2D_circ_lin_address ( .... )
{ .... }

// 2D filter function with calculations written through
// line based pointers for efficiency. This is unnatural.
// Performs function while receiving next line of data and
// sending out previously calculated results.
void 2d_filter ( .... )
{ .... }

mpx_cell filter_kernel ( .... )
{ // begin mpi_cell..

  // initialize
  init 2D circ lin buf ( .... );

  while (1)
    { // begin while
      MPX_Recv( .... ); // non-blocking line receive
      2d filter( ....); // perform filter across line of data

      MPX_Send( .... ); // non-blocking line send of results

      update_2D_circ_lin_address ( .... );

    } // end while

} // end mpx_cell
```

コード部分Ａ：ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃ‐フィルタカーネルのコード例

【０１３２】

スレッドを処理する「ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃ」を構成するシステムは、以下のように書くことができる（フィルタシステムのコード例）：

```
//* filter_system.c
//* example hx3xxx pseudo code for 2D filter system

#include <mpx.h>
#include <math.h>
#include "filter_system.h"
```

```
mpx_cell filter_system ( .... )
{ // begin mpx_cell..
  // system defined through MPX_PARALLEL
  if (MPX_RANK == MPX_PARALLEL)
    { // begin MPX_PARALLEL
    distribution_and_collection: di_co_sys( .... );
    filter_thread_00: filter_kernel ( .... );
    filter_thread_01: filter_kernel ( .... );
    filter_thread_02: filter_kernel ( .... );
      ....

    } // end MPX_PARALLEL

} // end mpx_cell
```

コード部分Ｂ：ｆｉｌｔｅｒ＿ｓｙｓｔｅｍ．ｃ‐フィルタシステムのコード例

【０１３３】

「ｖｏｉｄ　２ｄ＿ｆｉｌｔｅｒ（　．．．．　）」関数は以下の例のように書くことができる：

```
// apply filter across line of data

for( col = begin; ... )
  { // begin col = begin..
    result [ col ] = 0;
    for ( i = 0; ... )
      { // begin i = 0..
        for ( j = 0; ... )
          { // begin j = 0..
            // operations
            result[ col ] += (long)( filter[ i ] [ j ] ...
            * data[ i ] [ address_line[ i ] - offset + j ] );
          } // end for j = 0..
      } // end for i = 0..
  } // end for col = begin..
```

コード部分Ｃ：Ｍ×Ｎフィルタとして実装される２ｄ＿ｆｉｌｔｅｒ関数

```
// apply Laplacian filter across line of data

for ( col = begin; ... )
  { // begin for begin = 0..

    // operations..
    temp = in_data[ address_line[ center ] + col - Lap ] ...
      + in_data[ address_line[ center ] + col + Lap ];
    temp += (long)( in_data[ address_line[ center - Lap ] + col ] ...
      + in_data[ address_line[ center + Lap ] + col ] );
    temp -= (long)( 4 * in_data[ address_line[ center ] + col ] );
    result_data[ col ] = abs(temp);
```

```
  } // end for col = begin..
```

コード部分Ｄ：ラプラシアンフィルタとして実装される２ｄ＿ｆｉｌｔｅｒ関数

```
// apply Laplacian filter across line of data

address_center = address_line[ center ];
address_n_Lap = address_line[ center - Lap ];
address_p_Lap = address_line[ center + Lap ];

for ( col = begin; ... )
  { // begin for begin = 0..
    // operations ..
    temp = in_data[ address_center + col - Lap ] ...
                + in_data[ address_center + col + Lap ];
    temp += (long)( in_data[ address_n_Lap + col ] ...
                + in_data[ address_p_Lap + col ] );
    temp -= (long)( 4 * in_data[ address_center + col ] );
    result_data[ col ] = abs(temp);
  } // end for col = begin..
```

コード部分Ｅ：ラプラシアンフィルタとして実装される、簡略化されたｖｏｉｄ　２ｄ＿ｆｉｌｔｅｒ関数

【０１３４】

ここで提示したコード部分／プログラムは単なる例示であり、いずれの特定のプログラム言語に実施形態を限定することを意図したものではないことに留意されたい。

【０１３５】

論理的には、ＡＮＳＩ　Ｃソフトウェアプログラミングモデルを有するＭＰＩは、雑多なメモリ及び適合可能な通信スキームの全要件をサポートできる。実行モデルは、ソフトウェアプログラミングモデルの要件を満たす能力を提供できる。更にソフトウェアプログラミングモデルは、可変リアルタイムパフォーマンスを標的とすることができるスケーラブルなコードをサポートできる。

【０１３６】

結果の例は、図１２に示すような、例えば（ＣｏｈｅｒｅｎｔＬｏｇｉｘ，Ｉｎｃｏｒｐｏｒａｔｅｄから提供される）例示的なｈｘ３ＸＸＸプロセッサ等のメモリ‐ネットワークプロセッサ上で実現される。より具体的には、図１２は、ｈｘ３１００プロセッサ上で、Ｃコードの行を変更せず、３０ｆｐｓ（左の画像）、続いて６０ｆｐｓ（右の画像）で示されている、例示的な４Ｋビデオ２Ｄフィルタシステムのリソースのレイアウトを示す。ここでソフトウェアスレッドの数は増大しており、追加のハードウェアスレッドを使用してフレームレートスループットを上昇させることができる。

【０１３７】

上述のように、現行のアプローチの１つの問題は、ソフトウェアプログラミングモデルが標的ハードウェアの特徴の外挿をサポートする場合、システムの効率的な記述ができないという点である。システムを適切に記述するためには、システムの挙動及びインタラクティブ性の全ての重要な態様を、プログラミングモデルの制御、通信、メモリ及び処理モデルそれぞれの中で何らかの方法でキャプチャしなければならない。最後の２つ、即ちメモリ及び処理モデルは、通常はアドレス指定されず、実行モデルから暗黙的に割り当てられると想定される。

【０１３８】

従って、いずれのシステムを効率的に定義することは、システム全体を効率的に定義す

ることを必要とし得る。この例では、議論の焦点は、通信構造のプロパティ及びメモリプロパティとのインタラクティブ性からシステムを記述し、またシステム要件から処理を記述することに当てられている。図８の方法を参照して上述したように、これは、メモリ構造のプロパティ、即ち：通信をサポートするために使用される構造のサイズ；処理をサポートするために使用される構造のサイズ；処理をサポートするための構造の挙動；通信をサポートするための構造の挙動；処理をサポートするための構造のアクセシビリティ；及び通信をサポートするための構造のアクセシビリティを定義することを伴い得る。これに続いて、通信、制御及び処理プロパティを引き出すか又は明確に定義できる。

【０１３９】

上述の撮像フィルタの例に関して、これは以下の様式で実施できる。

【０１４０】

メモリ構造のサイズの定義：

図１１に示すように、この撮像フィルタの例では、システムに関して最も自然なメモリ構造をサポートするために、循環ラインバッファアドレス指定スキームを生成する必要があり得る。メモリ構造は、フィルタのサイズ、即ち現在のデータのフィルタ処理に必要な行数（このケースでは２Ｍ＋１）と、処理の現在の結果をキャプチャするための更なる行と、次のデータのラインを同時に受信するための更なる行と、フィルタ処理の過去に計算された結果を送信するための更なる行とを加算した数によって定義できる。従ってこの例示的なアプリケーションに関して、メモリ構造のサイズは、積（２Ｍ＋４）＊（２Ｎ；Ｈ／$Ｎ_{v}$）個のピクセルとして定義又は指定できる。

【０１４１】

メモリ構造の挙動の定義：

構造のサイズに加えて、所定の時点における構造の挙動を明確に定義する必要がある。この特定のケースでは、メモリの挙動は、過去に受信された処理済みのデータと、利用可能な、即ち到着済みの新規のデータとに対して定義される。具体的には、この例示的実施形態では、現在の処理のために使用されることになる、「現在のデータ」として表示できるメモリのセクションが存在し、そのサイズは「２Ｍ＋１」行×「２Ｎ＋１」超の列であり、また、次のデータのラインを受信するメモリのセクションが存在し、過去に計算された結果を保持するメモリのセクションが存在し、最後に、処理から現在の結果を収集するメモリのセクションが存在する。現在の処理が完了し、過去に計算された結果が送信された後、受信されたばかりの次のデータのラインが、今度は現在のデータの第１のラインとなるように、メモリの挙動を更新できる。そして現在の結果は、送信されるべき過去に計算された結果となり、現在の結果はその位置を再割り当てされる。従って、これらメモリセクションをこのように回転式で使用することにより、メモリ構造の挙動を定義でき、又は特徴付けることができる。

【０１４２】

メモリ構造のアクセシビリティの定義

各メモリセクションは、定義されたアクセシビリティを有する必要がある。この特定の例示的なケースでは、現在のデータに関しては、このデータは標準的な２次元アレイの形態でアクセス可能である必要がある。上述のメモリ構造の挙動に従って、新規のデータのラインが到着して新規のデータの第１の行となり、データの最も古い又は最後の行が脱落する度に、ユーザ／プログラマの視点からのデータの自然な書き込み及びアクセスをサポートするために、物理アドレスへの２次元アクセスを更新してよい。他の３つのメモリのセクション（上述）は、データ及びフィルタリングの結果の受信並びに結果の送信をサポートする、データの１次元アレイを効果的に実装できる。

【０１４３】

通信プロパティ（サイズ、挙動、アクセシビリティ）の定義

メモリ構造のサイズ、挙動、アクセシビリティが定義されると、通信間又は通信内、制御、及び処理プロパティを、そのシステム又は別のシステム内のインタラクティブ性に基づいて引き出すことができ、又は明確に定義できる。この特定のケースでは、システムの

入力プロパティを、メモリ構造のサイズ、挙動、アクセシビリティの定義から引き出すことができる。例えば通信無線機内のインターリーバ及び／又はデインターリーバの場合に関してといった他のシステムは、より明確な定義が必要となり得る。

【０１４４】

一実施形態では、次のステップは、ソフトウェアプログラミングモデル内でシステムのプロパティを効果的に表現することであってよい。これを行うには、ＭＰＩ等のＡＰＩを生成又は拡張してＡＮＳＩ　Ｃをサポートすること、Ｃ＋＋で特定のクラス構造を生成すること等を含むがこれらに限定されない、多数の方法が存在する。しかしながら、具体的な語彙的表現は重要ではない。重要なのは、プログラミングモデルがこれらの（動的な）システムの定義を認識すること、即ちツールフローがシステムの定義を解釈でき、続いてシステムを標的の実行モデル及び下層のハードウェアアーキテクチャに効果的にマッピングできることである。

【０１４５】

例示的なコードであるコード部分Ａ、ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃは、制御モデル、通信モデル並びに補助メモリ構造及び処理に明らかにアプローチし、またそのように解釈される必要があり得る。これは、システムを直感的に表現でき、システムを定義でき、又は効率的にシステムを解釈できるような方法で、制御、通信及びメモリ構造の間で動的インタラクティブ性を定義できない。

【０１４６】

撮像フィルタの例を用いて続けると、ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃは以下のように書き換えできる：

```
//* filter_kernel.c
//* example pseudo code for 2D filter incorporating system
// property defintion
#include <mpx.h>
#include <math.h>
#include "filter_system.h"

// initialize variables
mpx_2d_lin int buffer[(ROW+3)*COL];

// 2D filter function with calculations written naturally as for (i, j)
// and performs function while receiving next line of data and
// sending previously calculated results.
void 2d_filter( .... )
{.... }

mpx_cell filter_kernel ( .... )
{ // begin mpi_cell..
  while(1)
    { // begin while
      MPX_Recv( .... ); // non-blocking line receive
      2d_filter( .... ); //perform filter across line of data
      MPX_Send( .... ); // non-blocking line send of results
    } // end while
} // end mpx_cell
```

コード部分Ｆ：ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃ‐一実施形態によるシステムのプロパティの定義を組み込んだ、フィルタカーネルの更新されたコード例

【０１４７】
理解できるように、上記更新されたコード例であるコード部分Ｆにおいて、プログラム文「ｍｐｘ＿２ｄ＿ｌｉｎ　ｉｎｔ　ｂｕｆｆｅｒ［（ＲＯＷ＋３）＊ＣＯＬ］；」は変数を宣言しており、具体的には、この時点ではプログラミングモデルによってサポートされている、上述のサイズ、挙動、アクセシビリティプロパティを備えるメモリ構造を宣言している。
【０１４８】
従ってここで、コード部分Ａの「ｖｏｉｄ　２ｄ＿ｆｉｌｔｅｒ（　．．．．　）」関数を自然な形態で書くことができ、以下の２つの例において提示されるように、その処理において高い演算効率を達成できる：

```
// perform filter across line of data

for( col = begin; ... )
  { // begin col = begin..
    result data [ col ] = 0;
    for ( i = 0 ; ... )
      { // begin i = 0..
        for ( j = 0 ; ... )
          { // begin j = 0..
            // operations
            result_data[ col ] += (long)( filter [ i ] [ j ]...
            * in_data[ i ][ col - offset + j ] );
          } // end for j = 0..
       } // end for i = 0..
  } // end for col = begin..
```

コード部分Ｇ：Ｍ×Ｎフィルタとして実装される２ｄ＿ｆｉｌｔｅｒ関数の、更新されたコード例

```
// apply Laplacian filter across line of data

for ( col = begin; ... )
  { // begin for begin = 0..
    // operations ..
    result_data[ col ] = abs ( (4 * in_data [ center ] [ col ] )...
                         - in_data[ center ] [ col - Lap ]...
                         - in_data[ center ] [ col + Lap ]...
                         - in_data[ center - Lap ] [ col ] ...
                         - in_data[ center + Lap ] [ col ] );
  } // end for col = begin..
```

コード部分Ｈ：ラプラシアンフィルタとして実装されるｖｏｉｄ　２ｄ＿ｆｉｌｔｅｒ関数の、更新されたコード例
【０１４９】
続いて、プログラム文「ＭＰＸ＿Ｒｅｃｖ（　．．．．　）；　／／　ｎｏｎ-ｂｌｏｃｋｉｎｇ　ｌｉｎｅ　ｒｅｃｅｉｖｅ」は、挙動及びアクセシビリティのプロパティを自動的に更新する単回更新を提供できる。これは、受信されるべき次のデータのライン、処理するための現在のデータ、収集されるべき現在の結果をセットアップする。
【０１５０】

ツールフロー内において、コンパイラは、システムの挙動を解釈するよう、及び定義されたシステムの機能性をサポートするために、ハードウェアリソースをより効果的にマッピングするよう、設計されてよい。

【０１５１】

上述の撮像フィルタの例は、主としてメモリ定義（メモリビュー）の視点から記述されている。これを限定として解釈するべきではない。サイズ、挙動、アクセシビリティの観点から記述したシステムのメモリモデルに加えて、又はこれの代わりに、制御、通信、処理ビュー（及びサブビュー）を同様に使用して、システムを記述又は定義してよい。

【０１５２】

従って、ソフトウェアプログラミングモデル内において、効果的なシステムの記述をサポートするために、全てのモデルは、処理、メモリ、通信及び制御並びにこれらの間の及び／又はこれらの内部でのアクティビティのサイズ、挙動、アクセシビリティを記述するか又は暗黙的に解釈するプロセスをサポートする必要があり得る。

【０１５３】

例示的な使用のケース：有限インパルス応答フィルタ

別の例として、等式２によって特徴付けられる例示的な有限インパルス応答（ＦＩＲ）フィルタを考える。

【０１５４】

【数２】

$$y[t] = \sum_{i=0}^{i=N-1} c_i x[t-i]$$

等式２

【０１５５】

ここでｔは時間であり、c_iは係数であり、ｘ［ｔ］は入力サンプルであり、Ｎはフィルタ長であり、ｙ［ｔ］は出力サンプルである。

【０１５６】

ＦＩＲフィルタを使用した、図８の方法の記述

このタイプのシステム（ＦＩＲフィルタ）では、データストリーム入力は典型的にはサンプル毎に、即ち１度に１サンプルずつ受信される。ＦＩＲフィルタはその長さよって特徴付けられ、これは、ＦＩＲフィルタが出力サンプルを生成するために使用する入力サンプルの数に等しい。ＦＩＲフィルタのデータ要件（例えば長さ）を満たすためにデータの十分なサンプルを受信すると、上記データに対してＦＩＲフィルタの計算が実施され、フィルタリングされたデータサンプル出力が生成される。データをより迅速に処理するために、一般的なスキームは、フィルタ処理を複数の段階及び／又はパイプライン化されたセグメントに分割することである。ブロック図形態の単一のＦＩＲフィルタプロセスを図１３に示す。ブロック図形態の、段階及びパイプライン並列処理を示す多数のプロセスを用いたＦＩＲフィルタを、図１４に示す。

【０１５７】

ＭＰＩをサポートするＡＮＳＩ　Ｃ言語ベースのソフトウェアプログラミングモデルを用いて、図１２、１３に示すスレッドプロセスを記述するためのプログラムは、以下のように記述できる：

```
//* fir_filter_kernel.c
//* example hx3xxx pseudo code for FIR filter
#include <mpx.h>
```

```
#include <math.h>
#include "fir_filter_system.h"

// initialize variables
int ld_sample_buffer[ length_fir_filter + 3 ];

// initialize 1D circular sample buffer on 1d_sample_buffer void init_1D_circ_sa
m_buf( .... )
{ .... }

// update 1D circular sample address on 1d_sample_buffer void update_1D_circ_sam
_address( .... )
{ .... }

// FIR filter function with calculations written with sample based pointers
// for efficiency. This is unnatural. Performs function while receiving
// next sample of data and sending out previously calculated sample result.
void fir filter( .... )
{.... }

mpx_cell fir_filter_kernel ( .... )
{ // begin mpi_cell..
  // initialize
  init_1D_circ_sam_buf( .... );
    while (1)
      { // begin while
        MPX_Recv( .... ); // non-blocking sample receive
        fir_filter( .... ); //perform filter on current data
        MPX_Send ( .... ); // non-blocking sample send of results
        update_1D_circ_sam_address( .... );
      } // end while
} // end mpx_cell
```

コード部分Ｉ：ｆｉｒ＿ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃ‐ｆｉｒフィルタカーネルのコード例

【０１５８】

従って、コード部分Ｊのシステムは、「ｆｉｒ＿ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃ」処理スレッドで構成されるＦＩＲフィルタのパイプライン相及びアルゴリズム相両方の並列処理を示す図１４の構文的表現を表し、以下のように書くことができる：

```
//* fir_filter_system.c
//* example hx3xxx pseudo code for FIR filter system

#include <mpx.h>
#include <math.h>
#include "fir_filter_system. h"

mpx_cell fir_filter_system ( .... )
{ // begin mpx_cell..
  // system defined through MPX_PARALLEL
```

```
  If (MPX_RANK == MPX_PARALLEL)
    { // begin MPX_PARALLEL
    distribution_and_collection: di_co_sys( .... );
    fir_filter_phasesegment_00: fir_filter_kernel( .... );
    fir_filter_phasesegment_01: fir_filter_kernel( .... );
    fir_filter_phasesegment_10: fir_filter_kernel( ....);
      ....
    } // end MPX_PARALLEL
} // end mpx_cell
```

コード部分Ｊ：ｆｉｒ＿ｆｉｌｔｅｒ＿ｓｙｓｔｅｍ．ｃ‐ｆｉｒフィルタシステムのコード例

【０１５９】

以下のコード部分Ｋは、図１５と一致した方法で循環バッファを用いてＦＩＲ計算を実装する。

【０１６０】

「ｖｏｉｄ　ｆｉｒ＿ｆｉｌｔｅｒ（　．．．．　）」関数は、以下のように書くことができる：

```
// apply FIR filter to current sample data
// calculate filter using circular data buffer
// "1d_sample_buffer"
// assumptions:
// a. "begin" and "result" are indices into 1d_sample
// buffer [N+2] and are known to be in the range 0
// to N+2. "begin" is the index of the newest
// sample and (result-1) is the index of the oldest
// sample.
// b. "filter [N] " array variable contains filter
// coefficients//

// intialize variables
int i;
int end;
int ind;
long temp_result;

// process part 1
// calculates the filter from newest sample to oldest
// sample or the end of the data buffer
// "1d_sample_buffer"

// determine the "end" condition for the
// processing of part 1
 if (begin > 3) {
  end = N+3;
 }
 else {
   end = (N + begin);
 }
```

```
// processing of filter for "process part 1"
ind = 0;
for( i = begin; i < end; i=i+l )
  { // begin for i = 0..
    temp_result += ( long) 1d_sample_buffer [ i ] * filter [ ind ];
    ind += 1;
  } // end for i = 0..

// process part 2, if necessary
// calculates the remaining data for the filter
//continuing from newest to oldest if process 1 did
//not perform all the processing
if (begin > 3)
  { // begin if begin..
    for ( i = 0; i < (result) ; i=i+l )
      { // begin for i = 0..
        temp_result += ( long) 1d_sample_buffer [ i ] * filter [ ind ];
        ind += 1;
      } // end for i = 0..
  } // end i f begin..

1d_sample_buffer[result] = temp_result;
```

コード部分Ｋ：ｆｉｒ＿ｆｉｌｔｅｒ関数

【０１６１】

この実装形態では、「１ｄ＿ｓａｍｐｌｅ＿ｂｕｆｆｅｒ」は、図１５に示した経時的メモリにおける構造と同様に演算できる。

【０１６２】

撮像フィルタの例に関して上述したように、論理的には、ＡＮＳＩ　Ｃソフトウェアプログラミングモデルを有するＭＰＩは、雑多なメモリ及び適合可能な通信スキームの全要件をサポートできる。実行モデルは、ソフトウェアプログラミングモデルの要件を満たす能力を提供でき、またソフトウェアプログラミングモデルは、可変リアルタイムパフォーマンスを標的とすることができるスケーラブルなコードをサポートできる。

【０１６３】

これもまた上述のように、現行のアプローチでは、標的ハードウェアの特徴の外挿をサポートするソフトウェアプログラミングモデルは、システムの効率的な記述を可能とすることができないか、又は促進しない。

【０１６４】

この例示的なＦＩＲフィルタの例では、議論の焦点は、通信構造のプロパティ及びメモリプロパティとのインタラクティブ性からシステムを記述し、またシステム要件から処理を記述することに当てられている。これは、通信構造のプロパティ、即ち：通信をサポートするために使用される構造のサイズ；処理をサポートするために使用される構造のサイズ；処理をサポートするための構造の挙動；メモリをサポートするための構造の挙動；処理をサポートするための構造のアクセシビリティ；及びメモリをサポートするための構造のアクセシビリティを定義することを伴い得る。この通信ビューが定義されると、メモリ、制御及び処理プロパティを引き出すか又は明確に定義できる。例示的な一実施形態では、これは以下の方法で実施できる。

【０１６５】

通信構造のサイズの定義

図１４のＦＩＲフィルタ例において示すように、システムに関して最も自然なメモリ構造をサポートするために、循環サンプルバッファアドレス指定スキームが生成され得る。通信構造のサイズは、フィルタのサイズ、即ち現在のデータのフィルタ処理に必要な行数（このケースでは１Ｎ）と、処理の現在の結果をキャプチャするための更なるサンプルと、次のデータのサンプルを同時に受信するための更なるサンプルと、ＦＩＲフィルタ処理の過去に計算された結果を送信するための更なるサンプルとを加算した数によって定義できる。従ってこの例示的なアプリケーションに関して、通信構造のサイズは、「Ｎ＋３」として定義又は指定できる。

【０１６６】

通信構造の挙動の定義：

通信構造のサイズに加えて、所定の時点における構造の挙動を明確に定義する必要がある。この特定のケースでは、通信の挙動は、過去に受信された処理済みのデータと、利用可能な、即ち到着済みの新規のデータとに対して定義される。具体的には、この例示的実施形態では、現在の処理のために使用されることになる、「現在のデータ」として表示される（通信メモリの）セクションが存在し、そのサイズは「Ｎ」サンプル超であり、また、次のデータのサンプルを受信するセクション（又はサンプル空間）が存在し、過去に計算された結果を保持するセクション（又はサンプル空間）が存在し、最後に、処理から現在の結果を収集するセクションが存在する。現在の処理が完了し、過去に計算された結果が送信された後、受信されたばかりの次のデータのサンプルが、今度は現在のデータの第１のサンプルとなるように、通信の挙動を更新できる。そして現在の結果は、送信されるべき過去に計算された結果となり、現在の結果はその位置を再割り当てされる。従って、これら通信（メモリ）セクションをこのように回転式で使用することにより、通信構造の挙動を定義でき、又は特徴付けることができる。

【０１６７】

通信構造のアクセシビリティの定義

各通信セクションは、定義されたアクセシビリティを有する必要がある。この特定の例示的なケースでは、現在のデータに関しては、このデータは標準的な１次元アレイの形態でアクセス可能である必要がある。上述の通信構造の挙動に従って、新規のデータのサンプルが到着して又は受信されて新規のデータの第１のサンプルとなり、データの最も古い又は最後のサンプルが脱落する度に、ユーザ／プログラマの視点からのデータの自然な書き込み及びアクセスをサポートするために、物理アドレスへの１次元アクセスを更新してよい。他の３つの通信のセクション（上述）は、データ及びフィルタリングの結果の受信並びに結果の送信をサポートする、データのサンプル空間を効果的に実装できる。

【０１６８】

メモリプロパティ（サイズ、挙動、アクセシビリティ）の定義

通信構造のサイズ、挙動、アクセシビリティが定義されると、メモリ間又はメモリ内、制御、及び処理プロパティを、そのシステム又は別のシステム内のインタラクティブ性に基づいて引き出すことができ、又は明確に定義できる。この特定のケースでは、システムの入力プロパティを、通信構造のサイズ、挙動、アクセシビリティの定義から引き出すことができる。

【０１６９】

ここでもまた、次のステップは、ソフトウェアプログラミングモデル内でシステムのプロパティを効果的に表現することであってよく、これを行うには、ＭＰＩ等のＡＰＩを生成又は拡張してＡＮＳＩ　Ｃをサポートすること、Ｃ＋＋で特定のクラス構造を生成すること等を含むがこれらに限定されない、多数の方法が存在する。しかしながら上述のように、具体的な語彙的表現は重要ではない。寧ろ重要なのは、プログラミングモデルがこれらの（動的な）システムの定義を認識すること、即ちツールフローがシステムの定義を解釈でき、続いてシステムを標的の実行モデル及び下層のハードウェアアーキテクチャに効果的にマッピングできることである。

【０１７０】

例示的なコードであるコード部分Ｉ、ｆｉｒ＿ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃは、制御モデル、メモリモデル、補助通信構造及び処理に明らかにアプローチし、またそのように解釈される必要があり得、これは上述のように、システムを直感的に表現でき、システムを定義でき、又は効率的にシステムを解釈できるような方法で、制御、通信、処理及びメモリ構造の間で動的インタラクティブ性を定義できない。

【０１７１】

ＦＩＲフィルタの例を用いて続けると、ｆｉｒ＿ｆｉｌｔｅｒ＿ｋｅｒｎｅｌ．ｃは以下のように上書きできる：

```
//* fir_filter_kernel.c
//* example pseudo code for FIR filter incorporating system
// property defintion
#include <mpx.h>
#include <math.h>
#include "filter_system.h"

// initialize communication memory
struct com_fir {
  int com [N];
  int receive;
  int send;
  int result;
}
com_fir com;

// FIR filter function with calculations written naturally as for(i)
// and performs function while receiving next sample of data and
// sending previously calculated result.
void fir_filter( .... )
{.... }

mpx_cell fir_filter_kernel( .... )
{ // begin mpi_cell..
  // receive initilization of FIR filter properties of generic FIR filter
  // kernel including type (size and amount) of input and output for
  // MPX_FIFO_IN and MPX_FIF0_0UT
  MPX_Recv( .... ) ;

  while(1)
    { // begin while
      // non-blocking sample receive or data sample receive and partial
      // accumulation
      MPX_Recv( com.recieve,....,MPX_FIFO_FIR |
                MPX_NONBLOCKING             |
                MPX_FIFO_IN );
      fir filter( .... ); //perform FIR filter across current data in
      // "com"
      // non-blocking send of resulting sample or sample data
      // and partial accumulation
      MPX_Send( com.send,....,MPX_FIFO_FIR |
```

```
                MPX_NONBLOCKING ...        |
                MPX_FIFO_OUT );
    } // end while
} // end mpx_cell
```

コード部分Ｌ：　ｆｉｒ　ｆｉｌｔｅｒ　ｋｅｒｎｅｌ．ｃ－一実施形態によるシステムのプロパティの定義を組み込んだ、ｆｉｒフィルタカーネルの更新されたコード例

【０１７２】

コード部分Ｌの更新されたコード例に示すように、プログラム文「ｉｎｔ　ｃｏｍ［Ｎ］；」は変数を宣言しており、これは、この時点ではプログラミングモデルによってサポートされている、上述のサイズ、挙動、アクセシビリティプロパティを備える通信構造として使用される。これはコード部分Ｌ中の、「ＭＰＸ＿Ｒｅｃｖ（ｃｏｍ，．．．．，ＭＰＸ＿ＦＩＦＯ＿ＦＩＲ　｜　ＭＰＸ＿ＮＯＮＢＬＯＣＫＩＮＧ　｜　ＭＰＸ＿ＦＩＦＯ＿ＩＮ）；」及び「ＭＰＸ＿Ｓｅｎｄ（ｃｏｍ，．．．．，ＭＰＸ＿ＦＩＦＯ＿ＦＩＲ　｜　ＭＰＸ＿ＮＯＮＢＬＯＣＫＩＮＧ　｜　ＭＰＸ＿ＦＩＦＯ＿ＯＵＴ）；」という例示的なＭＰＩ構造体によって示される。

【０１７３】

ここで、コード部分Ｋの「ｖｏｉｄ　ｆｉｒ＿ｆｉｌｔｅｒ（　．．．．　）」関数を自然な形態で書くことができ、以下のように、その処理において高い演算効率を達成できる：

```
// apply FIR filter to current sample data
int40 temp = 0;

for( i = 0; i < FILTER_LENGTH; i++ )
  { // begin i = begin..
    temp += (long) com.com[ i ] * filter[ i ];
  } // end for i = 0..

com.result = temp >> SCALE;
```

コード部分Ｍ：更新されたｆｉｒ＿ｆｉｌｔｅｒ関数

【０１７４】

ここで「ｃｏｍ」通信メモリは、図１６に示した経時的メモリにおける構造と同様に演算できる。

【０１７５】

最後に、プログラム文「ＭＰＸ＿Ｒｅｃｖ（ ｃｏｍ．ｒｅｃｉｅｖｅ，．．．．，ＭＰＸ＿ＦＩＦＯ＿ＦＩＲ　｜　ＭＰＸ＿ＮＯＮＢＬＯＣＫＩ Ｇ　｜　ＭＰＸ＿ＦＩＦＯ＿Ｉ ）；　／／　ｎｏｎ-ｂｌｏｃｋｉｎｇ　ｓａｍｐｌｅ　ｒｅｃｅｉｖｅ　ｏｒ　ｄａｔａ　ｓａｍｐｌｅ　ｒｅｃｅｉｖｅ　ａｎｄ　ｐａｒｔｉａｌ　ａｃｃｕｍｕｌａｔｅ」は、挙動及びアクセシビリティのプロパティを自動的に更新する単回更新を提供できる。これは、受信されるべき次のデータのサンプル、処理するための現在のデータ、収集されるべき現在の結果をセットアップする。

【０１７６】

上述のＦＩＲフィルタの例の例示的な１つの変形例では、ＦＩＲフィルタの一部はスレッドで実施され、計算のためにＦＩＲの部分累算を受信する必要があり、計算が完了すると、適切なデータサンプルを有する別のスレッドへ転送する。例示的なコード部分Ｎは、サンプルデータ及び部分累算を受信するスレッドを示し、これにコード部分Ｏが続き、これは部分ＦＩＲフィルタカーネル、及び図１７に示すような代表的な通信メモリ構造の使用を示し、図１４のＦＩＲ相／セグメントの中段のブロックに対応する。

```
//* fir_filter_kernel.c
//* example pseudo code for FIR filter incorporating system
// property definition
// it receives partial_result, sampled data, and sends sampled data

#include <mpx.h>
#include <math.h>
#include "filter_system.h"

// initialize communication memory
struct com_fir {
  int com[FILTER_LENGTH];
  int receive;
  int40 result_partial;
}
com_fir com;

// FIR filter function with calculations written naturally as
//for (i)
// and performs function while receiving next sample of data and
// sending previously calculated result.
void fir_filter( .... )
{.... }

mpx_cell fir_filter_kernel ( .... )
{ // begin mpi_cell..
  // receive initilization of FIR filter properties of generic FIR filter
  // kernel including type (size and amount) of input and
  // output for MPX_FIFO_IN and MPX_FIFO_OUT
  MPX_Recv( .... );

  while (1)
    { // begin while
      // non-blocking sample receive or data sample receive and
      // partial accumulation
      MPX_Recv( com. receive |
                com. result_partial,....,MPX_FIFO_FIR | ...
                MPX_NONBLOCKING ... |
                MPX_FIFO_IN );

      fir_filter( .... ); //apply FIR filter across
      //current data in "com.com"
      // non-blocking send of resulting sample or sample data
      // and partial accumulation
      MPX_Send ( com.com [N-1] | com.result_partial , ...., ...
                MPX_FIFO_FIR  | ... MPX_NONBLOCKING |
                MPX_FIFO_OUT );

    } // end while
```

```
} // end mpx_cell
```

コード部分Ｎ：サンプリングされたデータ及び部分累算転送をサポートする通信モデル構造のコンテキスト内の、部分フィルタカーネル組み込みシステムのプロパティの定義の例

```
// perform part of a FIR filter on current sample data
// supporting receiving partial accumulation and
// forwarding partial accumulation

for( i = 0; i < FILTER_LENGTH; i++ )
  { // begin i = begin..
    com.result_partial += (long)com.com[ i ] * filter[ i ];
  } // end for i = 0..
//not needed: com. result = temp >> SCALE;
```

コード部分Ｏ：部分累算をサポートする、自然な形態で書かれたＦＩＲフィルタカーネルの一部

【０１７７】

サンプルのみを受信し、部分累算結果と共にサンプルを送信するためのコード、並びにサンプル及び部分累算を受信して最終的なＦＩＲフィルタ結果を生成するためのコードは、提供された２つのＦＩＲコードの例の変形例として書くことができ、ここでは簡潔にするためにこれを提示しない。本出願に開示される技術のこのような具体的実装形態は、プログラミング技術を有する者の能力の範囲内である。多数の他のフィルタタイプも同様の様式で再構成でき、他のアルゴリズム及びプロセスについても同様である。換言すると、上述の例は単なる例示及び例証であり、本発明の範囲を制限することを意図したものではない。

【０１７８】

従って、上で開示されている技術の実施形態は、マルチプロセッサシステム上でのソフトウェアの並列実行に関する従来技術のアプローチを上回る有意な効率を提供できる。

【０１７９】

コード部分Ｍに比べて、コード部分Ｏは、整数に適合するためのアキュムレータのリスケールを必要としない。

【０１８０】

撮像フィルタの例に関して上述したように、ツールフロー内において、コンパイラは、システムの挙動を解釈するよう、及び定義されたシステムの機能性をサポートするために、ハードウェアリソースをより効果的にマッピングするよう、設計されてよい。

【０１８１】

上述のＦＩＲフィルタの例は、主として通信定義（通信ビュー）の視点から記述されている。ここでもまた、これを限定として解釈するべきではない。サイズ、挙動、アクセシビリティの観点から記述したシステムの通信モデルに加えて、又はこれの代わりに、制御、メモリ、処理ビュー（及びサブビュー）を同様に使用して、システムを記述又は定義してよい。

【０１８２】

システム設計の効率的な捕捉に続いて、ハードウェアからのより効率的な操作を達成するために、定義されたシステムプロパティをサポートするための新規のプログラム可能なハードウェア特徴部分が生成されている。これらの特徴部分は、処理要素、データ経路、アドレス生成、制御フロー等を含んでよいがこれらに限定されない。これらの特徴部分は、所定の演算サイクルにおいて（実質的に）最大の演算及びエネルギ効率を達成すること、並びに多くの計算負荷が高いアプリケーションにおいてオーバヘッド（セットアップ、インデックスレジスタ更新等）をゼロ近くまで低減することを目的としている。

【０１８３】
ハードウェア要素
以下は、改善されたシステムを提供する、様々な新規のハードウェア要素を説明する。
【０１８４】
ハードウェアの設計は、標的とされたシステムアプリケーションに関して、エネルギ効率の良い様式で実施できるよう、密接に連結され得る。循環データバッファリングを、モジュロアドレッシング及び（以下に説明する）ＤＭＲ　ＦＩＦＯ技術をハードウェア反復ループと共に用いて行ってよく、これによりループにおけるアドレス指定が自動化される。（以下に説明する）ＨｙｐｅｒＯｐは、従来の技術が必要とする高い最適化コストを要することなく、高度な並列処理を制御する方法を提供する。（以下に説明する）マルチデータ経路ハードウェアは、実際のアルゴリズム数学セクションのより良好な並列処理を、純粋なＡＳＩＣ、非プログラマブルソリューションと略同じ程度に効率的な様式で可能とする。自動反復バッファリングは、フェッチ（及び復号化）電力を削減する。限定されたアドレス空間は、データ及び命令の両方に関して、効率的なアドレス指定及び読み出し／書き込み動作を提供する。データの再整列及びアキュムレータ転送は、データ移動オーバヘッドを低減するため、及びアルゴリズムをより迅速に実行できるよう、多数のＰＥを１つに結合して、追加のリソースを動作させるための機構を提供する。
【０１８５】
目標
ハードウェアシステムのいずれの工学的実装の目標は、システムに必要な機能性を最小のコスト関数で提供することである。このコスト関数は多数の態様を含み、ここで最も重要なもののうちのいくつかは、実際のハードウェアコスト、システム全体を実装するための時間／コスト、使用コスト（電力及び面積）等である。これは様々な方法でトレードオフを克服する多数のオプションを何度も利用できる。前の節において、提示された様々なソリューションは、ＧＰＰ、ＧＰＵ、ＤＳＰ、ＦＰＧＡ、メニーコア／マルチコア、ＡＳＩＣを含んでいた。これらのタイプのハードウェアを用いて利用可能な既存のシステムの実装は、複雑性、使いやすさ、プログラム可能性、柔軟性、コスト、電力の点で大きく異なる。与えられた例のうち、所定のシステムに関して最も電力効率が良いのはＡＳＩＣである。しかしながら、ＡＳＩＣは最も柔軟性のないものでもあり、また開発の時間及びコストに関して最も犠牲の大きいものである。その他のものはより柔軟性を提供するが、大抵比較的低いコスト／時間及び増大された電力においてトレードオフを有している。本出願において記載するハードウェアの目標は、純粋なＡＳＩＣハードウェアソリューションの電力効率に依然としてアプローチしながら、特定のタイプの計算負荷のための従来の手段によって高度にプログラム可能なハードウェアプラットフォームを提供することである。
【０１８６】
概念
この目標に挑戦し、到達するために、提示されている他のソリューションの実質的に全てから概念を借用する。これらの概念の革新的な修正及び組織化は、より迅速な実装（市場に出るまでの時間）を可能とするために望まれる高度なプログラム可能性を依然として提供しながら、電力効率に関して純粋なハードウェアソリューションにアプローチする電力効率の良い実装を可能とする。使用されるソフトウェア技術は、ハードウェアを補助するパフォーマンス及び電力最適化を提供するアーキテクチャに密接に連結される。
【０１８７】
ここでの主要な焦点は、低電力で高スループットを提供するために実行ユニット又は処理要素（ＰＥ）をどのように設計するかを説明することである。これは、処理のためにデータがＰＥへと及びＰＥから流れるファブリックを提供するデータメモリルータ（ＤＭＲ）における、メモリ及び通信システムに連結される。
【０１８８】
処理要素

従来技術（ＧＰＰ、ＧＰＵ、マルチコア、ＤＳＰ等）における処理要素（ＰＥ）は複数の命令アーキテクチャを有し、これら命令アーキテクチャは、従来の完全に符号化された命令、又は多数の演算ユニットの制御のための多数のスロットを有する超長命令語（ＶＬＩＷ）であった。これらのアーキテクチャにおけるデータ経路に関して、従来技術のうちのいくつかは、様々な程度の独立性によって並列に演算を行うことができる多数の演算ユニットを含む。スーパースケーラ型の実装において混合される多数の命令ストリームをサポートできるものもあれば、シングルスレッド化されるものもある。

【０１８９】

ここで提示する革新的なＰＥのアーキテクチャは、従来の完全に符号化された命令及びサブ命令のためのＶＬＩＷ型スロットのハイブリッドである。これは極めて柔軟かつ効率的な命令アーキテクチャであり、乗算器、ＡＬＵユニット、アドレス生成器等といった多数の演算ユニット及びこれらを相互接続するための機構を有する多種多様なデータ経路をサポートできる。命令ストリームの実行をスケジューリングするために多くのハードウェアを必要とするスーパースケーラアーキテクチャの動的スケジューリングとは異なり、このアーキテクチャは、命令ストリームを生成するソフトウェアツール（コンパイラ、アセンブラ等）によって提供される静的スケジューリングを提供する。この静的スケジューリングは、静的スケジューリングを例えばＧＰＰのランダムな命令ストリームに関してあまり最適でないものとするいくつかの制限を有しているものの、適切な最適化コンパイラ及びソフトウェア開発ツールと連結される場合、ＤＳＰ及び画像処理のための高度に構造化されたリアルタイム型アルゴリズムにとって完全に十分であり、大いに改善された実行時の電力効率を提供する。

【０１９０】

ＰＥは、演算ユニットが実行される多数のパイプラインステージで構成される。主要な演算ユニットは、命令フェッチ、復号化／発行、オペランドフェッチ、実行ユニット及びライトバック（フェッチに連結される）である。使用される公称９ステージパイプラインを図１８に示し、演算ユニットがパイプラインのどこで動作するかが示されている。

【０１９１】

このパイプライン構造は寧ろ、今日の基準に照らしても浅く、これは目的を有している。個々のユニット及び特徴の設計を、低電力設計を可能とする固有の設計点に重点を置いて、以下の節で議論する。読者は情報が多数のセクションに亘って情報が広がる傾向にあることに気付くだろう。というのは、特徴及びアーキテクチャが共に動作する多数の機能性ユニットを横断して、固有の全体的なハードウェア設計を生成するからである。大半のものよりも少ないステージを有するパイプラインは、パフォーマンスを向上させ電力を削減するために行われるハードウェア最適化の一部である。より小さく、より単純なハードウェアを用いて、パイプラインを、浅いパイプラインによってでさえ高いパフォーマンレベルで実行されるよう構築できる。

【０１９２】

実行パイプライン

基本的なパイプライン

図１９は、単一データ経路アーキテクチャに対して改善されたＰＥデータ経路の図である。より詳細には、図１９は、（前世代と同様の）幾分標準的な単一データ経路、及び改善されたＰＥデータ経路アーキテクチャ（命令フェッチ、復号化、及び制御論理は図示せず）の概念ブロック図である。この改善されたデータ経路は、単一データ経路よりも約６０％多いロジックで実現できる。換言すると、改善されたデータ経路におけるパイプ０又はパイプ１は、本質的に１サイクル毎に同等又はそれ以上の演算が可能である単一データ経路よりも、２０％論理的でない。単一のデータ経路は、論理演算及び算術演算のための別個の経路を有する。乗算器は、浮動小数点仮数乗算をサポートするために２４ｂ×２４ｎであってよい。これには４０ｂアキュムレータ及び加算－比較－選択（ＡＣＳ）ユニットが続いてよい。続いて全ての結果を再びシフト及び累算してよい。限定された量のパイプライン並列処理は、アキュムレータを半分に分割すること及び単一命令マルチデータ処

理（ＳＩＭＤ）モードで動作させることにより、８ビットデータ上で可能である。

【０１９３】

改善されたデータ経路アーキテクチャは、図においてパイプ０、パイプ１として示した二重の独立したパイプラインを含む。これらのパイプはそれぞれ、デュアル１６ｂ×１６ｂ乗算器を有する。パイプは、乗算の結果の範囲：クアッド１６ｂ、デュアル１６ｂ＋１つの３２ｂ、デュアル３２ｂ、又は１つの６４ｂ（これは浮動小数点演算のための単一の２４ｂ乗算もサポートする）を達成するために圧縮器回路によって結合される。各パイプに関して、圧縮器の後には、シフト、論理、加算、ＡＣＳ、続いて最終的なアキュムレータのステージが続いてよい。単純な単一データ経路パイプラインでは、１６ｂ演算、リアルタイムＤＳＰエンジンのための一次データサイズのために、単一の演算操作（乗算、又は加算／減算）のみを１クロック毎に実施してよい。新規のアーキテクチャにより、ハードウェアは、特定の演算に関して１クロック毎に最高４つの同一のタイプの演算を提供するために、より多くの最適な構造に分割される。このアーキテクチャは、１つのパイプライン毎に削減された量のハードウェアと共に、ＤＳＰ及び画像演算を行うために十分な遥かにエネルギ効率の高い方法を提供する。達成可能なエネルギ効率は純粋なハードウェアソリューションに匹敵するが、達成可能なエネルギ効率はここでは完全にソフトウェアプログラム可能なＰＥにおいて達成され、これは開発努力を低減する。デュアル／クアッドパイプラインアーキテクチャは１ＰＥサイクル毎に多数の異なる組み合わせの演算を可能とするが、従来のアセンブリ言語命令セットは固有の柔軟性の全てをサポートできない。この増大した柔軟性を十分に利用するためのモデルを符号化及びプログラミングする命令全体における変化についても議論する。

【０１９４】

図１９、２０において、データ経路が乗算器の後で２つの圧縮器を（パイプ１とパイプ０との間で）連結していることに留意されたい。ＦＩＲフィルタ等に必要とされるような多数のオペランドの積和演算のような、演算の加速のための単一の乗算の累算を提供する。これは、これらの演算及び差分絶対値の和（ＳＡＤ）タイプの演算を行う、極めて効率的な方法である。この最適化は、その他のものよりも、このアーキテクチャの電力効率を大幅に増大させる。これは、ＭＡＣのための乗算器アレイ又はＳＡＤタイプの演算のための加算器アレイからの部分積に対する別のビットの圧縮演算を単に行う一方で、別の全加算器の有効性を提供する、少数の単純ゲートである。再び、これらの追加のゲートを効率的に使用する能力は、後に議論の対象となる改善された命令符号化及び制御技術を必要とし得る。

【０１９５】

改善されたＰＥデータ経路の特徴を要約する：

【０１９６】

それぞれ（１サイクル毎に）以下が可能である２つのデータ経路：

・１つ／２つの１６×１６乗算又は１つの３２×３２乗算
・１つ／２つの１６ｂ加算／減算又は１つの３２ｂ加算／減算
・４０ｂバレルシフト
・３２ｂ論理演算
・２つの４０ｂアキュムレータを用いた４０ｂ累算

【０１９７】

データ経路は共に（１サイクル毎に）以下を実施できる：

・１つの３２×３２乗算又は積和演算
・１つの３２ｂ浮動小数点加算／減算／乗算
・二重又は単一の累算を伴う４つの１６ｘ１６乗算
・累算による４つの差分絶対値の和

【０１９８】

２つのデータ経路における乗算器及び加算器の４つの演算に基づいて、ＤＳＰアルゴリズムに必要である、効率的に実施できる更に多くの機能が存在する。純粋に命令符号化レ

ベル（後のＨｙｐｅｒＯｐを参照）においてではなく、プログラミングレベルにおいて実装されることになるこれらの演算を可能とするために、改善された命令セットを、かなり低いレベルのプログラミングモデルにおいてハードウェアを露出させるよう設計してよい。これは、このアーキテクチャによる別の革新をもたらす。

【０１９９】

オペランド整列

メモリからのオペランド及びメモリに対する結果の書き込みのフェッチは、全てのアーキテクチャのより大きな電力を消費するタスクである。大きなローカルキャッシュ及び複雑なキャッシングシステムにより、多くの電力がこれらの演算に消費されるため、全体的な効率を損なう。従来技術のシステムの大半において、メモリサブシステムは、多数の異なるオペランドサイズ、及びマルチレベルキャッシュシステムを通してアクセスされる大きなアドレス空間を可能とする。メニープロセッサＩＣチップの各ＰＥのためのオンチップサポーティングメモリは、サイズが更に制限され、同数のオペランドサイズ及び整列演算をサポートする必要はない。

【０２００】

ＨｙｐｅｒＯｐをサポートするためのデータ整列多重化装置及びＰＥアーキテクチャにおけるデータ整列多重化装置の使用を説明するために、オペランドサイズ及び整列についてここで簡潔に説明する。データオペランド及び結果については後の節でより詳細に議論するが、本議論では、アーキテクチャが様々な条件において、かつ必ずしも同時にではなく、以下のオプションを提供すればよい：

・２つの１６ビットオペランドのフェッチ
・２つの３２ビットオペランドのフェッチ
・１６ビット及び３２ビットオペランドの組み合わせのフェッチ
・追加の６４ビットの整列オペランドのフェッチ
・単一の１６ビットの結果の書き込み
・単一の３２ビットの結果又は２つの１６ビットの結果の書き込み
・２つの３２ビットの結果又は６４ビットの結果の書き込み

【０２０１】

メモリインタフェースを単純なままとするために、全ての上述の演算を物理メモリに関するデータサイズ境界上に整列させるべきである。これは。特定のアルゴリズムにおけるそれらの使用の実現可能性を制限するように思われる。この制限に対処するために、オペランド（及び結果）整列多重化装置を上述のデュアル／クアッドパイプラインアーキテクチャに追加する。整列多重化装置は、アセンブリ言語プログラミングモデルを有する単一パイプラインにおいて使用が非常に制限される場合があるものの、整列多重化装置は、ＨｙｐｅｒＯｐプログラミングモデルを有するデュアル／クアッドパイプラインアーキテクチャの柔軟性と良好に一致する。

【０２０２】

図２０は、本議論のためにいくつかの追加のフローレジスタの拡張を有する、デュアル高スループット演算ユニットの指定データパイプライン０（ＤＰ０）及びデータパイプライン１（ＤＰ１）を用いたデータ経路の例を示す。これらは追加のパイプラインレジスタＸ、Ｙ（入力用）、Ｚ（出力用）である。また、オペランドステージＡ、Ｂ、Ｃ及び宛先ステージＤも示す。

【０２０３】

Ａ、Ｂ、Ｃレジスタを、以前に議論したのと同程度に利用可能である最大６４ビット幅のオペランドを保存するために使用される。ＨｙｐｅｒＯｐは、２つのデータ経路のＡ、Ｂ、ＣレジスタとＸ、Ｙレジスタとの間のシステム多重化を用いて、データ経路が実施する各演算に必要なオペランド及び単語整列を制御する。ＨｙｐｅｒＯｐ処理中にＡ、Ｂ、Ｃレジスタへのオペランドフェッチは、メモリへの整列アクセスを可能とするプログラム制御下にあり、これにより、データ経路算術演算ユニットに再整列様式で（及び従って必要に応じて整列されていないように見えるもの）にオペランドを提供する。この革新によ

り、より単純な低電力メモリ構造及びアドレス指定モードが可能となり、これにより、どのようにしてオペランドがメモリに保存されるかにかかわらずピークスループットを提供するために、複雑なデータ経路／ＨｙｐｅｒＯｐの組み合わせに十分なオペランドを供給する方法を提供する。

【０２０４】

結果は、オペランドの議論におけるものと同様である。データ経路結果は、ＨｙｐｅｒＯｐ実行中にアキュムレータ又はＺレジスタのいずれかに入れられる。続いてこれらを、他の場所にライトバックするためにＤに移動させるか、又は後続の命令において追加のオペランドとして使用されることになる例示した経路上にフィードバックされる。オペランドと同様に、ここで結果の再整列ができ、これにより、非整列データのためにメモリ／レジスタに整列されたライトバック提供する。この場合も、これらの演算はＨｙｐｅｒＯｐによって独立して制御される。

【０２０５】

１つのパイプラインにつき二重のアキュムレータ

同様の算術演算の長い文字列を加算して単一の合計とするためのアキュムレータをサポートする多くのアーキテクチャでは、単一のアキュムレータが存在する。更に、場合によっては全ての演算がこのアキュムレータを結果で修飾する（この場合でも我々の前世代）。この構造は、追加のデータ経路の追加及びＨｙｐｅｒＯｐによって１サイクル毎に多数のオペランドに対して演算する能力の追加により、１サイクル毎の主に単一のスカラー演算であるアーキテクチャのために良好に作用するのと同時に、これはこの概念を拡張するために必須のものとなる。現行の設計点は、データ経路毎に２つの独立したアキュムレータを含む。各演算は、必要であれば、どのアキュムレータを更新するかを選択できる。従ってこれらを用いて、前で議論したシステム多重化構造によって後に処理するための中間値を保存できるか、又は多数のデータストリームを追加のサイクル若しくはアキュムレータ値をセーブ及び復元する電力オーバヘッドを要することなく、インタリーブ形式で処理できる。二重のデータ経路及びオペランド／結果アライメントといった他の特徴と連結される場合、二重のアキュムレータ構造のこれらの特徴は、パイプラインがより十分に利用される状態を保つための機構を提供し、これは同様に設計のための演算毎の全体的な電力を削減する。

【０２０６】

アキュムレータ転送

アキュムレータに関連する別の特徴は、いくつかのアルゴリズムの内側ループを高速化するための、及びチップ内の多数のＰＥに亘る並列実行を増大させるための別の方法を提供する。例えばこれは、多数のＰＥに亘って広がる必要がある計算ループに対するオーバヘッドを最小化して、マルチタップ高帯域幅ＦＩＲフィルタ等の場合に帯域幅を処理する十分なデータを提供するために必要である。これは、図２１に示したＰＥ間の「アキュムレータ転送（ａｃｃｕｍｕｌａｔｏｒ　ｆｏｒｗａｒｄｉｎｇ）」によって達成できる。

【０２０７】

この図は２つのＰＥを示すが、これは所望の帯域幅に到達するために必要とされるのと同じ数まで、容易に拡張可能である。経路は、１つのアキュムレータが更新されると更新された値が別のＰＥに対するＤＭＲのファブリック拡張部分に転送されるように、設けられる。ここで、到達したこのアキュムレータ値はローカル計算に加算して新たなローカルアキュムレータに入れるために使用される。この新たなアキュムレータ値は、更なる計算のために別のＰＥに再び転送される。

【０２０８】

また、ローカルＭＡＣ（又はその他の）計算のための４つの係数値を保持するためのＣレジスタの使用が図示されている。これは、例えばＦＩＲフィルタの４つのタップを提供できる。鎖線の中の各ＰＥは更なる４つのタップを提供できる。よって例えば、しかしそれによって限定されないが、８個のＰＥは、１クロックあたり１つのサンプルのフルクロックレートで、３２タップのＦＩＲフィルタを実装してよい。単一のＰＥにこの同一のフ

ィルタを実装することは、フィルタの帯域幅を８クロック毎に１つのサンプルまで制限することになる。

【０２０９】

データは、フィルタを通してデータストリームを前進させるためのＦＩＦＯ構造を有するＤＭＲ間を通過する。前の節で言及したリアライメントバッファを用いて、各ローカルＰＥを処理する現在のデータセットのための適切な係数を選択する。データ移動は実行によるロックステップにおけるものであり、これにより、アルゴリズムと一致した方法で次の計算を行うためのデータの到着時にトリガが起こる。このようにして、フィルタを、同一のデータ移動、及び同様のフィルタを実装する純粋なハードウェア構造が必要とする同一の計算のみを用いて実装してよい。これにより、このプログラマブルソリューションの電力散逸が、同一の機能を実施するＡＳＩＣに匹敵するものとなる。

【０２１０】

このケースに基づいて、単一のクアッド乗算－累算命令をフェッチして、このアルゴリズムの内側ループを実行してよい。後の節では更なる詳細が提供されるが、この命令を（必要な場合他のものと共に）命令フェッチ及び分配ユニットによってローカル命令バッファに保存してよい。ループでは更なる命令フェッチは不要であり、単一の命令に関しては更なる復号化は不要であり、潜在的に多くの電力散逸をセーブする。

【０２１１】

例えばデュアル／クアッドアーキテクチャを有する多数のＰＥを用いたＦＩＲフィルタの場合、各ＰＥは各サイクルにおいて、単一命令を有する最大４つの入力サンプルまで処理できる。この命令は、制御ラインのトグリングなしに、何度も実行できる。システムがこの１つ又は複数の反復命令を検出すると、メモリからのフェッチは起こらないため、命令メモリは節電モードに入る。ＦＩＲフィルタのようなタイトループＤＳＰアルゴリズムのうちのいくつかに関して、このプログラマブルシステムにおけるデータ移動及び計算及びマイナー制御トグリングは、ＡＳＩＣハードウェア実装と同様であり、従って略同一の動的電力散逸を達成する。

【０２１２】

命令フェッチ／復号化におけるアドレス生成

図２０では、アドレス生成器ユニット（ＡＧＵ）を含むアドレス生成器セクションを示したが、これについて詳述していなかった。ＰＥアーキテクチャのアドレス生成器セクションは、ハードウェアによってサポートされる様々なアドレス指定モードのためのアドレスを生成する。その固有の特徴をこの節で詳述する。

【０２１３】

アドレス生成器セクションは、アドレス生成に使用するための多数のプログラマブル算術演算ユニットを有してよい。更に、パイプラインのアドレス計算部分において更なる計算を行うために使用できる１つ又は複数の拡張された算術演算及び論理ユニット（ＧＡＬＵ）が存在してよい。これらの計算は、パイプの機能性及びパフォーマンスを拡張するのに、並びにテーブルルックアップ型演算及びその他のものにおけるパイプライン遅延を排除するのに有用である。図２０において、例示的なアドレス生成器セクションは、３つのＡＧＵ及び１つのＧＡＬＵ及び一連のサポートレジスタを含む。

【０２１４】

標準的な符号化方法における典型的な演算のために、ＡＧＵを用いて２つのソースオペランド及びデスティネーションのための、又はこれらのサブセット及びいくつかのアドレス若しくは拡張数学演算のためのアドレスを生成する。ユニットは符号化を用いて密接に連結される。ＨｙｐｅｒＯｐ符号化を介した拡張演算のために、これらのユニットは更に連結解除され、命令ストリームによって個別に制御できる。これは、演算の更なる柔軟性及び更なる並列化を可能とする。最適化はコンパイル時に実施してよく、これにより、リアルタイムの順序変更が不要となり、従って上記最適化の動作パワーペナルティがなくなる。

【０２１５】

このハードウェアのより詳細なブロック図を図２２に示す。この図により、多数のＡＧＵの連結が、このセクションにおける使用のために利用可能なレジスタと共にはっきりと分かる。例示的なハードウェアを以下のように詳述できる：

・アドレス型計算のための３つの標準的なＡＧＵ
・追加の算術／論理サポート（ＧＡＬＵ）のための１つの拡張ＡＧＵ
・８つのベースレジスタＢ０．．Ｂ７
　○アドレス指定モードにあるＢ０は値ゼロを得る
　○Ｂ０はスタックポインタとして使用される（ＳＰ相対アドレス指定モード）
・８つのインデックスレジスタ１０．．１７
　○アドレス指定モードにある１０は値ゼロを得る
　○１０を他のＡＧＵ算術のための一時レジスタとして使用できる
・８つのストライドレジスタＳ０．．Ｓ７
　○ＳｎはＩｎ又はＢｎと共に使用される
・２次インデックス又はベースレジスタ用の４つの追加のインクリメンタ
　○インデックスレジスタ１４．．１７
　○ベースレジスタＢ４．．Ｂ７
　○ストライドレジスタＳ４．．によるインクリメント

【０２１６】

最後の３つの項目はパイプの早期の命令フェッチ部分におけるアドレス計算及び単純計算において使用され得ないだけでなく、後に説明するハードウェア反復動作に連結されてクリティカルループ計算のためにゼロオーバヘッドルーピングを提供する。

【０２１７】

図２３は、単一のＡＧＵのより詳細な図を示すブロック図である。各ＡＧＵの特徴部分は以下を含んでよい：

１）更なるアドレス計算がより多くのレジスタ上で利用可能である；

２）拡張された迅速なＡＧＵの動作のための新たなブロック。（これらのうちの１つのみが存在し得る。入力は既存の多重化装置及びＡ、Ｂソースアドレス復号化からのフォール度を使用する）；

３）加算器の条件コード出力は迅速な分岐性能を可能とする。（これは、追加のパイプライン遅延を挿入するメインデータ経路の代わりにＡＧＵが決定算術演算を実施した場合に、ゼロオーバヘッド条件の分岐を提供する）；

４）（循環バッファリングのための）剰余インデックス算術演算；

５）（平坦なメモリへの多次元アレイのマッピングのための）マルチストライドインデックス算術演算；

６）加速されたテーブルルックアップのための複雑な算術演算。

【０２１８】

アドレス計算に単一のＡＧＵを使用するのに加えて、特定の命令によって複数のＡＧＵを組み合わせて、より複雑なアドレス計算を実施できる。このようにして、ＨｙｐｅｒＯｐは、アドレスに対する複雑な算術演算を実施するためにＡＧＵを配分でき、これによってメモリアドレスの使用方法に更なる柔軟性を提供できる。これらのより複雑なアドレス指定モデルは、循環アドレス指定（後のセクションで更に詳細に取り扱う）、剰余アドレス指定、２Ｄ及び３Ｄアドレス指定と共に使用するためのマルチストライドインデクシング、複雑なテーブルルックアップアドレス指定、ｓｉｎ／ｃｏｓによる特別なアドレス指定等を含むことができる。ハードウェアパイプラインのこのセクションに加算器及び論理性能を有するという柔軟性により、特定のアルゴリズムに関して複雑なアドレス算術演算を、このアドレス算術演算を通常の実行パイプラインで実施した場合に可能なものよりも効率的に、実施できる。ＡＧＵはメインデータ経路ＡＬＵとは別個のハードウェアであるため、アドレス計算はデータ計算と同時に実施できる。

【０２１９】

ＡＧＵ分岐

ＡＧＵパイプライン領域の別のＡＬＵ機能性は、ゼロオーバヘッド分岐を可能とする、早期の分岐計算である。分岐の決定を計算するための算術演算を、パイプラインの通常の実行セクションで実施すると、これはパイプラインのタイミング構造の範囲内において非常に後で行われる。これは、分岐を誤って予測した場合、複数の命令をフェッチして、分岐の決定を終了させている間にこれら命令を推測で実行することになる（命令は、結果を書き込むことによってその不変状態を変更できない）ことを意味する。決定が誤っている場合、これらの結果は無効であり、使用されず、これらのための電力及び時間が無駄になる。ＡＧＵ算術演算は、実施される分岐決定よりもパイプ内で極めて早期に行われるため、誤った命令をフェッチすることはなく、推測による実行も発生しない。従って、分岐によって、使用できない命令に対するいずれの実行サイクル又は電力が浪費されることがなくなる。

【０２２０】

循環アドレス指定

ＤＳＰアルゴリズムの加速のための複雑なストリーミングデータ構造をサポートするために、拡張されたアドレス指定モードが望まれる。１つのこのようなモードは、循環アドレス指定である。このタイプのアドレス指定により、算術演算をより簡潔な形態で書くことができ、またかなりの複雑性がデータ経路からＡＧＵセクションへと移され、このＡＧＵセクションにおいて、上記算術演算をより効率的に、かつパフォーマンスのためにより良いパイプラインのタイムスロットにおいて実行できる。これを達成するためにハードウェアをどのように利用するかは、以下の例から理解できる。

【０２２１】

２つのＡＧＵからの２つのインデックスインクリメンタのペアを組み合わせて、１つのインデックスに対する循環アドレス指定、例えば｛ｉ６，ｓ６｝及び｛ｉ７，ｓ７｝を生成すると想定する。

【０２２２】

図２４に示す循環バッファを想定する、ここで：

・ｉ６は現在のアレイインデックスを保持し、

・ｓ６は通常のストライドを保持し、

・ｉ７は最後の有効な循環アレイインデックスを保持し、

・ｓ７はラップ状況のためのストライドを保持する。

【０２２３】

これは、２つのインクリメンタを図２５に示すように連結することによって達成できる。これは、アドレスに対するＮを法とする完全な剰余演算を実装し、また算術演算は、１回のアドレス計算につき最高１回のラップしか行わないようなものである。この制約はソフトウェアによって維持できる。４つのインクリメンタが存在するため、このようなアプローチにおいて２つの循環バッファ、｛ｉ４，ｓ４，ｉ５，ｓ５｝及び｛ｉ６，ｓ６，ｉ７，ｓ７｝を実装できる。

【０２２４】

典型的なＧＰＰアーキテクチャでは、データフローのアドレス指定及び管理のためのアルゴリズム及び算術演算は、総計算負荷の極めて大きな部分を占める。上述のＰＥアーキテクチャの特徴部分は、データフローのコンパイル時間管理と共に、あるアプリケーションに関する総計算負荷を大幅に低減し、従って電力散逸を大幅に低減する。

【０２２５】

命令フォーマット及び符号化

このアーキテクチャの命令セットは、以下の２つの別個のサブセットに分解される：

・６４ｂの従来のアセンブリ命令

・１２８ｂのＨｙｐｅｒＯｐ命令

【０２２６】

これら２つのサブセットは、図２６に示す、プログラミングのための関連するモデルを有する。従来のアセンブリモード（ＡＳＭ）は、従来技術において典型的な、固定長６４

ビット命令に符号化される単一命令ストリームモードであってよい。ＡＳＭ符号化は、異なる命令タイプのために複数のフォーマットをサポートしてよく、主要な形態は、単一のデータ経路演算による３オペランド（２ソース、１宛先）フォーマットである。このモデルでは、１２８ビットＨｙｐｅｒＯｐモードよりも並列処理のサポートが弱い。しかしながら、単一のパイプラインの２つの半体、又は２つの（若しくはそれより多い）パイプライン上での演算を制御するＳＩＭＤ命令のための特定の命令符号化が存在してよい。また、パフォーマンス及び電力の最適化のためのクアッドデータ経路演算を実行することになる、極めて限定的な命令のセットのための符号化も存在し得る。

【０２２７】

もう１つの命令サブセットは、ＨｙｐｅｒＯｐモードである。これについては以下のセクションで議論する。これら２つの符号化モデルからの命令は、所定のアルゴリズムに関する最良の演算を提供するいずれの形態及びシーケンスの命令のストリームへと混合でき、上記アルゴリズムは最適化コンパイラによって決定できる。

【０２２８】

ＨｙｐｅｒＯｐ

ＨｙｐｅｒＯｐと呼ばれる命令のセットは、高い柔軟性のために、デュアル／クアッドパイプラインを粒度が高い様式で制御する。命令符号化スタイルは、「従来の」アセンブリ（ＡＳＭ）スタイルと、超長命令語（ＶＬＩＷ）スタイルとの間の何らかのスタイルである。これは以下を提供する：

・実行ユニット内の両方のデータ経路パイプの、分離され、かつ独立した制御
・ソースオペランドの位置の、分離され、かつ独立した制御
・結果ライトバックの位置の、分離され、かつ独立した制御
・１クロックあたり最大３つのＡＧＵの、別個の制御
・ハードウェア反復モデルの制御
・（多くの場合、分岐よりも効率的である）術語型実行のための制御

【０２２９】

例示した改善されたデュアル／クアッドデータ経路を用いて、幅広いパイプライン化及び同時並列処理をプログラムできる。データ経路要素の独立した制御、並びにデータ経路実行からのデータ（オペランド）フェッチ及び１つ又は複数の結果ライトバックの連結解除は、従来のＳＩＭＤ機械の厳格性とは異なり、演算において極めて高い柔軟性を提供する。ＨｙｐｅｒＯｐ命令は、データ経路のパイプライン化及び同時並列処理の、粒度が高い制御を提供する。各パイプは、使用されるＨｙｐｅｒＯｐに応じて、１クロックあたり複数の命令を実行できる。

【０２３０】

ＨｙｐｅｒＯｐに関する基本的な動機は、ＲＦ波形処理からＭＰＥＧ画像圧縮及び暗号化までの例を用いた、幅広いアプリケーションソフトウェアのためのアセンブリコードの分析に由来している。アセンブリ（ＡＳＭ）コードは「Ｃ」コードよりも効率的に実行されることが多いが、最適ではなく、実行が非効率的となる場合も多数発生していた。本発明者らは、アルゴリズムに関わる算術演算を実行するために使用されるＰＥサイクルの個数に関して効率を定義している。データストリームを何らかの階層フォーマットに従って解釈し、（例えば２Ｄフィルタを用いて）処理し、同一の又は別の階層フォーマットに従って再度パッケージングしなければならない場合に、非効率が発生していた。他の場合には、様々なタイプ｛短い、整数、浮動、長い｝の間での変換を行うためにかかる時間が過剰であった。これらの場合において、様々な寸法及び場合によっては可変サイズのアレイに保存されたデータを走査するために、ループを使用することが多い。ループアドレス計算を、データの処理に使用されるものと同一のハードウェア上で実施する場合、アドレス計算はループ遅延（オーバヘッド）を含む。

【０２３１】

更に、新規のデュアル／クアッドアーキテクチャは、より独立性の高い演算ユニットを提供し、従って、１つ又は複数のパイプラインが含む並列演算又は迅速な順次演算に関し

て、より高い柔軟性及びより多くの機会を提供する。ＨｙｐｅｒＯｐ命令は、ＡＳＭフォーマットに限定することなくこのハードウェアを制御する方法を提供する。独立したスロットという観念は、この業界で使用されている超長命令語（ＶＬＩＷ）符号化スタイルから借用したものである。ＶＬＩＷスタイルでは、命令内の特定のフィールドを、特定の機能ユニットを制御するための「スロット」と呼ぶ。スロットのコーディングの独立性は、機能ユニットの独立性に対応する。ＶＬＩＷスタイルの命令は、全ての機能ユニットに何かを実施させない場合があるが、プログラマは一般に殆どのスロットに、プログラム内の命令の殆どにとって有用な何かを実施させようとする。このようなＡＳＭ及びＶＬＩＷ符号化スタイルのハイブリッドにより：

・プログラム可能なデータ経路の並列処理

・データ経路要素の独立した制御

が可能となる。

【０２３２】

ＨｙｐｅｒＯｐ命令は、標準的な符号化スキームを用いた場合には実行できない方法で命令を並列処理及び加速できるように、実行パイプ演算、レジスタ、多重化の、より粒度の高い制御を提供する。これにより、３つのＡＧＵ及び２つのデータ経路の独立した制御が現行の実装形態において提供される。従来の符号化方法を用いて従来のパイプラインを通過するに従ってオペランドに結び付けられたパイプラインレジスタが、このモードでプログラマに提示される。これにより、メモリに保存する必要がない中間結果を有するタイトなループを遥かに効率的に実行でき、オペランドの移動及び整列を実行パイプラインから切り離して、コンパイラによるパイプラインの最適化においてより高い柔軟性を提供できる。Ｅｒｒｏｒ！Ｒｅｆｅｒｅｎｃｅ ｓｏｕｒｃｅ ｎｏｔ ｆｏｕｎｄ．これらのレジスタは、図２０のＰＩＰＥ部分においてＸ、Ｙ、Ｚで標識されている。これらは常に存在し、従来の命令によってハードウェアをパイプライン化するためにさえ使用される。しかしながらＨｙｐｅｒＯｐモードでは、これらは、図示されている多重化ハードウェアと共に、符号化される命令によって直接制御され、これにより、パフォーマンス及び電力の最適化のための、粒度が極めて高いデータフローの制御が提供される。

【０２３３】

これは、更なるビットの符号化によって、追加のハードウェアに容易に拡張できる。図２０はＰＥアーキテクチャの単なる一例である。

【０２３４】

５つの実行ユニットを制御するためのＨｙｐｅｒＯｐ命令に関するビットフィールドのフォーマットの例を、図２７に示し、これは同様の命令セットの一般的な符号化を例示するものである。いくつかの高いビット数は、図２７に示すものとは異なる特別な命令フォーマットを信号伝達できる。第１のフィールド（番号１２７）は、命令がマルチパート命令に関しては１２８ビットで符号化され、又は単一パート命令に関しては６４ビットで符号化されることをＰＥに信号伝達するために使用される。「ｃｏｎｄＥｘｅｃ」フィールドは、計算され、ＰＥにおいて状態として保存されている条件に基づく、命令の特定のフィールドの条件付き実行を可能とするために使用される。３つの「ｔｙｐ」フィールドは、制御情報の一部としてＡＧＵフィールドと組み合わせて使用される。アドレス生成ユニット（Ｇ０、Ｇ１、Ｇ２）に関する２セットのフィールドは、これらのユニットそれぞれにおいて実施される演算を制御する。最後の（ビット数が最も小さい）２つのフィールドはそれぞれ、実行ユニット内の２つのパイプライン化されたデータ経路それぞれにおいて実施されている演算を制御するための２０ビットフィールドである。

【０２３５】

最大５つの実行ユニットに関して、独立した制御が可能であることに留意されたい。各ユニットは、スループットとして、１クロックあたり１つ以上の演算を実施できる。典型的なＨｙｐｅｒＯｐは、１クロックあたり約８～１２個の基本演算を制御できる。反復ハードウェアの全てが使用中であり、かつロードを計数して、各１６ビットデータ語を「ｏｐ」として保存する場合、このＨｙｐｅｒＯｐ命令フォーマットを用いて、１クロックあ

たり約３６個のｏｐを得ることができる。１．５ＧＨｚのクロック及び２５６個のＰＥ設計点を用いると、これは略１４ＴＯＰＳ（テラ演算／秒）となる。

【０２３６】

これは、命令レベル並列処理を制御し、このような並列処理のスケジューリング及び最適化をコンパイラ及びフォフトウェア開発ツールの領域に残すための、極めて効率的な方法である。これにより、スーパースケーラタイプのスキームに比べてハードウェアを大幅に小型化でき、上記スーパースケーラタイプのスキームは、ピークパフォーマンスに到達するためのコンパイラ最適化が必要である一方で、オペランド、演算及び結果を完全に順序変更及び最適化するために、実行時に遥かに多量のハードウェア及び電力も必要とする。

【０２３７】

「ＦＩＲ」フィルタのための並列アセンブリ言語の例は、このフィルタがどのように作用するかを理解する助けとなり得る。この並列アセンブリ言語の例は、コード部分Ｍにおいて提供されるＡＮＳＩ　Ｃコードの例に対応する。「｜｜」の記号は、ＨｙｐｅｒＯｐ符号化命令によって並列に実行される命令を隔てるものである。

```
// perform FIR filter on current sample data and produce sample out

  // %bl points to structure com.com
  // %b2 points to the coefficient array filter
  // %dl points to structure com. result
  // %il is index i

  { repeat1 $0, FILTER_LENGTH - 1, $4, %il, ACCO = 0;
  |A| ldi com.com, %bl; |B| ldi filter, %b2; | C | void;
  |D| void; |G| void; }

  { | A | ld64 0[%bl + %il], %AB; | B| void; | C | ld64 0[%b2 + %il] , %C;
  |DP0| qmadd16 %AB , %C, %ACC0; |D| st32 %ACC0, %dl;
|G| void; }
```

コード部分Ｐ：ＡＮＳＩ　Ｃのコード部分Ｍに対応する、ＦＩＲフィルタ関数に関する並列アセンブリ言語コード

【０２３８】

フェッチ／フロー制御

ハードウェアは、プリフェッチ法によってパイプラインに命令を供給し続ける、非常に標準的な命令フェッチユニットを備えてよい。このフェッチユニットにはフロー制御が付随し、これは命令に連結されて、特定の制御条件に対処するための効率的な方法を提供する。上記制御条件は以下を含む：

・オーバヘッドが低い様式でのループの加速のためのハードウェアの反復
・バッファリングの反復
・条件付き実行（マルチレベル）

【０２３９】

命令フェッチ／バッファリング

命令フェッチ及び復号化ユニットの例を図２８に示す。命令はＦＩＦＯ形式で命令バッファ内に維持でき、出力は、保存された命令のシーケンスをもたらす現在の命令を含む。

【０２４０】

命令が発行されると、ＩＦＵは新規の命令アドレスを計算して命令メモリ（ＩＭ）を読み出し、命令は命令バッファにロードされる。ＩＦＵは、例えば６４ｂと１２８ｂといった、複数の長さが混ざった命令を処理してよい。これらはパディングを伴わずに命令メモ

リにパックされる。ＩＭからの第１の命令語（ＩＷ）は、命令の長さに関するコードを含む。スタートアップ時、１２８ビットの命令が読み出される。そして命令の発行に続いて適切なビット数が読み出され、これによって発行された命令の語を補充する。

【０２４１】

多数の命令を命令バッファに保存でき、典型的な例は８～１６個である。この例では、命令はシーケンスのまま維持され、順序変更されない。いくつかのマイクロプロセッサアーキテクチャでは動的順序変更が行われるが、これは複雑な論理を必要とし、相当な電力を散逸させ得る。

【０２４２】

現在の命令が完了すると、この命令は「リタイア」し、次の命令が現在の命令として指定され、これ以降も同様である。命令バッファに適合できる程度に十分に小さい命令のグループは、ループ内で効率的に実行でき、またループが完了するまでいずれの追加のフェッチも必要としなくてよい。反復ハードウェアプログラミングを用いてこれらのシーケンスを検出し、バッファ内に適切な命令を保持する。これは、極めて電力効率の高い内側ループを提供する。バッファ及び反復ハードウェアは、最大３深度（又は３次元ループ実行）までのネスティングをサポートする。単一の命令の最も効率的な内側ループに関して、この単一の命令は現在の命令として休止状態であり、ループカウントが満たされるまで、単に復号化に対する制御を駆動する。これは、プログラム可能なアーキテクチャに関して極めて電力効率が高い。

【０２４３】

反復バッファからの実行中、フェッチが必要ないため、電力を節約するために命令メモリを低電力状態とすることができる。関数の最も内側のループの命令の数は、ループの各繰り返しのための実行時間において大きな因子であり、従って関数の速度、スループット、帯域幅に反比例する。反復バッファの目的は、最も重要な内側ループがバッファに適合するように、十分な深度（殆どの場合８又は１６個の命令）を提供することである。

【０２４４】

このハイブリッドアーキテクチャプロセスは、１つのＰＥについて一度に１つの命令を処理し、従ってスケーラと呼ぶことができる（スーパースケーラではない）。しかしながらＨｙｐｅｒＯｐのサブ命令は独立して復号化され、従ってサブ命令の２つ以上のスレッドを維持できる。最適化コンパイラツールは、ユーザプログラムを命令ストリームにコンパイルした後、これをサブ命令の複数のスレッドに分割してより高い並列性を達成することにより最適化できる。データハザードを回避するためのルールを適用すると、命令ストリームを最適化するために、個々のサブ命令を命令ストリーム内で前後に移動させることができる。サブ命令をロックしてロックステップで発行するこのような制限により、復号器、制御及びフェッチユニットの複雑性は大幅に低減され、従って電力散逸が節減される。

【０２４５】

反復

反復ハードウェアは、ＧＰＰの典型的な試験及び分岐方法よりも効率的なリアルタイムＤＳＰアルゴリズムのタイトな内側ループに対する、ハードウェア内でのルーピングを提供するための機構である。

【０２４６】

反復ループハードウェアは：

・ゼロオーバヘッドループ形成のための内側ループのためのハードウェアの事前ロード；

・１次インデックスレジスタを用いたネスティングの３つのレベルのサポート；

・４つの２次ベース／インデックスレジスタの自動インクリメント

を提供する。

【０２４７】

反復ループの関数は、以下のＣのようなコードによって示される：

```
do {
  PC = TOP
  while (PC <= BOTTOM_NEW)
    execute instructions;
} while (CURRENT != END;
    CURRENT += STRIDE)
```

コード部分Ｑ：反復ループの擬似コード

【０２４８】

このような演算を実装するためのハードウェアレジスタを表１に示す。これらのうちのいくつかについては、アドレス計算の議論において既に議論した。反復ループのために使用されるインデックス値は、処理されるデータのアレイに対するアドレスの計算に使用される場合が多いため、これらのレジスタはＡＧＵによってアクセス可能である。

【０２４９】

【表１】

HW ループレジスタ	注記
ids_start_<>	開始時のインデックス
ids_current_<>	現在のインデックス
ids_end_<>	終了時のインデックス
ids_stride_<>	インデックスをインクリメントするためのストライド
ids_top_<>	ループの頂部のアドレス
ids_bottom_<>	ループの底部のアドレス

表１：反復レジスタ

【０２５０】

６４ビット及び１２８ビット両方の符号化において、これらの必要なレジスタ全てを単一のクロックでロードし、任意に反復ハードウェアを始動させるための、特別な命令が存在してよい。ここで過去の世代を上回る１つの革新は、反復ハードウェアをロードして、ループのＴＯＰアドレスに到達した時に上記反復ハードウェアが始動するよう準備するための方法が含まれることである。これにより、３深度ループが得られ、内側の２つのループは、外側のループの開始前にロードされる。これにより、内側ループに関する反復命令のフェッチは行われず、従って内側ループはサイクルカウントに関してゼロオーバヘッドループとなる。

【０２５１】

このハードウェアを用いると、命令のセットに対して３深度ネスティングを実装でき、ループは命令のために必要なクロックだけを用いて実行される。ループカウンタ、インクリメント、比較、分岐、アドレス計算のためのサイクルオーバヘッドは不要である。これによりデータ経路ハードウェアを自由にし、アルゴリズムの算術部分のみに集中させることができる。

【０２５２】

図２９

図２９は、マルチパート命令の受信及び実行の第１の実施形態を示すフローチャートである。

【０２５３】

本方法は、フェッチユニット及び複数のアドレス生成ユニットを備えるマルチプロセッ

サ装置において動作し得る。フェッチユニットは、マルチパート命令を受信するよう構成されてよく、マルチパート命令は複数のフィールドを含む。複数のアドレス生成器ユニットは、複数のフィールドのうちの第１のフィールドに応じた算術演算を実施するよう構成された第１のアドレス生成器ユニットを備えてよい。複数のアドレス生成器ユニットはまた、複数のアドレスのうちの少なくとも１つのアドレスを生成するよう構成された第２のアドレス生成器ユニットを備えてよく、複数のアドレスそれぞれは、複数のフィールドそれぞれに依存する。マルチプロセッサ装置は更に、複数のアドレスのうちの第１のアドレスに応じた第１のデータを保存するよう構成された保存ユニットを備えてよい。更にフェッチユニットは、複数のアドレスのうちの第２のアドレスに応じた第２のデータをフェッチするよう構成されてよい。マルチプロセッサ装置は更に、複数のアドレスのうちの少なくとも１つの別のアドレスを生成するよう構成された第３のアドレス生成器ユニットを備えてよい。更に、複数のフィールドのサブセットの各フィールドは、複数のアドレス生成器ユニットそれぞれが実施する演算を符号化してよい。

【０２５４】

マルチパート命令を受信及び実行するための方法は、以下のように動作できる。

【０２５５】

まず２９０２において、プロセッサはマルチパート命令を受信してよく、このマルチパート命令は複数のフィールドを含む。次に２９０４においてプロセッサは、複数のフィールドのうちの第１のフィールドに応じた算術演算を実施してよい。２９０６においてプロセッサは更に、複数のフィールドそれぞれに応じて、複数のアドレスのうちの所定のアドレスを生成してよい。

【０２５６】

追加の実施形態では、本方法は更に、複数のアドレスのうちの第１のアドレスに応じた第１のデータを保存するステップを含んでよい。また本発明は更に、複数のアドレスのうちの第２のアドレスに応じた第２のデータをフェッチするステップも含んでよい。一実施形態では、プロセッサは複数のアドレス生成器ユニットを含み、算術演算を実施するステップは、複数のアドレス生成器ユニットのうちの少なくとも１つのアドレス生成器ユニットを用いて上記算術演算を実施するステップを含む。更に一実施形態では、複数のフィールドのサブセットの各フィールドは、複数のアドレス生成器ユニットそれぞれが実施する演算を符号化してよい。

【０２５７】

図２９の方法は、複数のプロセッサと、例えばデータメモリルータ（ＤＭＲ）であってよい、複数の動的に構成可能な通信要素とを備えるシステムにおいて動作でき、複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結されてよい。複数のプロセッサのうちの所定のプロセッサは：１）マルチパート命令を受信し（ここでマルチパート命令は複数のフィールドを含み）；２）複数のフィールドのうちの所定のフィールドに応じた算術演算を実施し；３）複数のフィールドのサブセットに応じた複数のアドレスを生成するよう構成されてよい。複数のプロセッサはそれぞれ、複数のアドレスのうちの第１のアドレスに応じた第１のデータを保存するよう、更に構成されてよい。更に、複数のプロセッサのうちの所定のプロセッサは、複数のアドレスのうちの第２のアドレスに応じた第２のデータをフェッチするよう構成されてよい。

【０２５８】

更に、複数のプロセッサのうちの所定のプロセッサは、複数のアドレス生成器ユニットを含んでよく、また複数のフィールドのサブセットの各フィールドは、複数のアドレス生成器ユニットそれぞれが実施する演算を符号化してよい。

【０２５９】

図３０

図３０は、マルチパート命令の受信及び実行の第１の実施形態を示すフローチャートである。

【０２６０】

本方法は、フェッチユニット及び実行ユニットを備えるマルチプロセッサ装置において動作し得る。フェッチユニットは、マルチパート命令を受信するよう構成されてよく、マルチパート命令は複数のデータフィールドを含む。実行ユニットは複数のパイプラインユニットを含んでよく：１）複数のフィールドのうちの第１のフィールドに応じた、複数のパイプラインユニットのうちの所定のパイプラインユニットを用いて、第１の演算を実施し；２）複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう、構成されてよい。複数のパイプラインユニットはそれぞれ、複数の乗算器ユニット及び複数の加算器ユニットを含んでよい。複数のパイプラインユニットはそれぞれ、部分積の第１の数を部分積の第２の数へと圧縮するよう構成された圧縮器ユニットも含んでよい。複数のパイプラインユニットのうちの第１のパイプラインユニットの圧縮器ユニットは、複数のパイプラインユニットの第２のパイプラインユニットの圧縮器ユニットから少なくとも１つの部分積を受信するよう構成されてよい。また、複数のフィールドのサブセットの各フィールドは、複数のパイプラインユニットそれぞれが実施する演算を符号化してよい。

【０２６１】

マルチパート命令を受信及び実行するための方法は、以下のように動作できる。まず３００２において、プロセッサはマルチパート命令を受信してよく、このマルチパート命令は複数のフィールドを含む。次に３００４においてプロセッサは、複数のフィールドのうちの第１のフィールドに応じた、複数のパイプラインユニットのうちの所定のパイプラインユニットを用いて、第１の演算を実施するステップを実施してよい。３００６においてプロセッサは、複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施してよい。

【０２６２】

一実施形態では、複数のフィールドのサブセットの各フィールドは、複数のパイプラインユニットそれぞれが実施する演算を符号化する。

【０２６３】

第２の演算を実施するステップは、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットのうちの第１のパイプラインユニットによって第１の部分積のセットを生成するステップと、複数のパイプラインユニットのうちの上記少なくとも２つのパイプラインユニットのうちの第２のパイプラインユニットによって第２の部分積のセットを生成するステップとを含んでよい。

【０２６４】

複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットのうちの第１のパイプラインユニットによる、第１の部分積のセットの生成は、部分積の第１の数を部分積の第２の数へと圧縮するステップを含んでよく、第２の数は第１の数未満である。複数のパイプラインユニットはそれぞれ、複数の乗算器ユニット及び複数の加算器ユニットを含んでよい。

【０２６５】

図３０の方法は、複数のプロセッサと、複数の動的に構成可能な通信要素、例えばＤＭＲとを備えるシステムにおいて動作できる。複数のプロセッサはそれぞれ複数のパイプラインユニットを含んでよい。複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結されてよい。複数のプロセッサはそれぞれ：１）マルチパート命令を受信し（ここでマルチパート命令は複数のフィールドを含み）；２）複数のフィールドのうちの第１のフィールドに応じた、複数のパイプラインユニットのうちの所定のパイプラインユニットを用いて、第１の演算を実施し；３）複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう構成されてよい。

【０２６６】

各プロセッサの複数のパイプラインユニットはそれぞれ、複数の乗算器ユニット及び複数の加算器ユニットを含んでよい。複数のプロセッサそれぞれの複数のパイプラインユニットそれぞれは更に、部分積の第１の数を部分積の第２の数へと圧縮するよう構成された圧縮器ユニットを含んでよい。各プロセッサの第１のパイプラインユニットの圧縮器ユニットのうちの少なくとも１つは更に、各プロセッサの第２のパイプラインの別の圧縮器ユニットから、少なくとも１つの部分積を受信するよう構成されてよい。

【０２６７】

一実施形態では、複数のフィールドのサブセットの各フィールドは、複数のパイプラインユニットそれぞれが実施する演算を符号化する。

【０２６８】

図３１

図３１は、プロセッサによる演算の実施の一実施形態を示すフローチャートである。

【０２６９】

本方法は、命令を受信するよう構成されたフェッチユニットと、複数のパイプラインユニットを含む実行ユニットとを備えるマルチプロセッサ装置において動作し得る。各パイプラインユニットはアキュムレータユニットを含んでよい。実行ユニットは：１）複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて、受信した命令に応じた第１の演算を実施して結果を生成し；２）複数のパイプラインユニットのうちの第１のパイプラインユニットのアキュムレータユニットに上記結果を保存し；３）複数のパイプラインユニットのうちの第１のパイプラインユニットのアキュムレータユニットに保存された結果を、プロセッサのアキュムレータユニットに伝送するよう、構成されてよい。複数のパイプラインユニットはそれぞれ、複数の乗算器ユニット及び複数の加算器ユニットを含んでよい。

【０２７０】

一例として、第１の演算は乗算演算を含んでよく、第１のパイプラインユニットのアキュムレータユニットに結果を保存するために、実行ユニットは更に、第１のパイプラインユニットに、乗算演算の少なくとも１つの部分積を保存するよう構成されてよい。

【０２７１】

一実施形態では、命令は複数のフィールドを含んでよく、第１のパイプラインユニットを用いて、受信した命令に応じた第１の演算を実施するために、実行ユニットは更に、第１のパイプラインユニットを用いて、複数のフィールドのうちの第１のフィールドに応じた第１の演算を実施するよう構成されてよい。

【０２７２】

実行ユニットは更に、複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう構成されてよい。

【０２７３】

演算を実施するための方法は、以下のように動作できる。

【０２７４】

３１０２において、本方法は、複数のプロセッサのうちの第１のプロセッサを用いて第１の演算を実施するステップを含んでよい。第１のプロセッサは、第１の複数のアキュムレータを含んでよく、第１の演算を実施するステップは、第１の複数のアキュムレータのうちの所定のアキュムレータに値を保存するステップを含んでよい。

【０２７５】

３１０４において、本方法は、第１の複数のアキュムレータのうちの上記所定のアキュムレータに保存された値を、複数のプロセッサのうちの第２のプロセッサが含む第２の複数のアキュムレータのうちの所定のアキュムレータに伝送するステップを含んでよい。

【０２７６】

３１０６において、本方法は、第２の複数のアキュムレータのうちの上記所定のアキュムレータに保存された値に応じた、複数のプロセッサのうちの第２のプロセッサを用いて

、第２の演算を実施するステップを含んでよい。
【０２７７】
一実施形態では、本方法は更に、第１のプロセッサによって第１のマルチパート命令を受信するステップを含んでよく、第１のマルチパート命令は、第１の複数のフィールドを含む。第１のプロセッサを用いて第１の演算を実施するステップは、第１の複数のフィールドのうちの第１のフィールドに応じた、複数のプロセッサのうちの第１のプロセッサを用いて、第１の演算を実施するステップを含んでよい。
【０２７８】
本方法は更に、複数のプロセッサのうちの第２のプロセッサによって第２のマルチパート命令を受信するステップを含んでよく、第２のマルチパート命令は、第２の複数のフィールドを含む。
【０２７９】
所定のアキュムレータに保存された値に応じた第２のプロセッサを用いて第２の演算を実施するステップは、第２の複数のフィールドのうちの第１のフィールドに応じた第２のプロセッサを用いて第２の演算を実施するステップを含んでよい。
【０２８０】
図３１の方法は、複数のプロセッサと、複数の動的に構成可能な通信要素、例えばＤＭＲとを備えるシステムにおいて動作できる。複数のプロセッサはそれぞれ複数のアキュムレータユニットを備えてよい。複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結されてよい。第１のプロセッサは：１）第１の命令を受信し；２）第１の命令に応じた第１の演算を実施して結果を生成し；３）第１のプロセッサの複数のアキュムレータユニットのうちの所定のアキュムレータユニットに上記結果を保存し；４）第１のプロセッサの複数のアキュムレータユニットのうちの上記所定のアキュムレータユニットに保存された結果を、複数のプロセッサのうちの第２のプロセッサの複数のアキュムレータユニットのうちの所定のアキュムレータユニットに伝送するよう、構成されてよい。
【０２８１】
複数のプロセッサのうちの第２のプロセッサは：１）第２の命令を受信し；２）第２の命令と、複数のプロセッサのうちの第２のプロセッサの複数のアキュムレータユニットのうちの上記所定のアキュムレータユニットに保存された値とに応じた第２の演算を実施するよう、構成されてよい。
【０２８２】
第１の命令は第１のマルチパート命令を含んでよく、第１のマルチパート命令は第１の複数のフィールドを含み、第２の命令は第２のマルチパート命令を含み、第２のマルチパート命令は第２の複数のフィールドを含む。第１の命令に応じた第１の演算を実施するために、第１のプロセッサは更に、第１の複数のフィールドのうちの第１のフィールドに応じた第１の演算を実施するよう構成されてよい。第２の命令及び所定のアキュムレータユニットに保存された値に応じた第２の演算を実施するために、第２のプロセッサは更に、第２の複数のフィールドのうちの第１のフィールドに応じた第２の演算を実施するよう構成されてよい。
【０２８３】
図３２
図３２は、プロセッサによる演算の実施の一実施形態を示すフローチャートである。
【０２８４】
本方法は、フェッチユニットと、複数のアドレス生成器ユニットと、複数のバッファとを備えるマルチプロセッサ装置において動作し得る。フェッチユニットはマルチパート命令を受信するよう構成されてよく、マルチパート命令は複数のフィールドを含む。複数のアドレス生成器ユニットのうちの第１のアドレス生成器ユニットは、第１のフィールドに応じた第１の演算を実施して、結果を生成してよい。更に、複数のアドレス生成器ユニットのうちの第２のアドレス生成器ユニットは、複数のフィールドのうちの第２のフィール

ド及び上記第１の結果に応じた第２の演算を実施するよう構成されてよい。
【０２８５】
一実施形態では、第１のバッファは、複数のフィールドのうちの少なくとも１つのフィールドに応じた第１のストライド値を保存するよう構成され、第２のバッファは、複数のフィールドのうちの少なくとも１つの別のフィールドに応じた第２のストライド値を保存するよう構成される。
【０２８６】
複数のフィールドのうちの第１のフィールドに応じた第１の演算を実施するために、第１のアドレス生成器ユニットは更に、第１のストライド値に応じた第１の演算を実施するよう構成されてよい。
【０２８７】
演算を実施するための方法は、以下のように動作できる。本方法は、複数のアドレス生成器ユニット及び複数のバッファを備えるプロセッサにおいて動作できる。
【０２８８】
３２０２において、本方法はマルチパート命令を受信してよく、このマルチパート命令は複数のフィールドを含む。３２０４において、本方法は、複数のフィールドのうちの第１のフィールドに応じた第１のアドレス生成器ユニットを用いて、第１の演算を実施してよい。３２０６において、本方法は、複数のフィールドのうちの第２のフィールドと、第１の結果とに応じた第２のアドレス生成器ユニットを用いて、第２の演算を実施してよい。
【０２８９】
本方法は更に、複数のバッファのうちの第１のバッファに第１のストライド値を保存するステップ、及び複数のバッファのうちの第２のバッファに第２のストライド値を保存するステップを含んでよい。
【０２９０】
第１のフィールドに応じた第１のアドレス生成ユニットを用いて第１の演算を実施するステップは、第１のストライド値に応じた第１の演算を実施するステップを含んでよい。
【０２９１】
本方法は、複数のプロセッサと、複数の動的に構成可能な通信要素、例えばＤＭＲとを備えるシステムにおいて動作できる。複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結されてよい。複数のプロセッサのうちの所定のプロセッサは：１）マルチパート命令を受信し（ここでマルチパート命令は複数のフィールドを含み）；２）複数のフィールドのうちの第１のフィールドに応じた、複数の生成器のうちの第１のアドレス生成器を用いて、第１の演算を実施して、第１の結果を生成し；３）複数のフィールドのうちの第２のフィールドと上記第１の結果とに応じた、複数の生成器のうちの第２のアドレス生成器を用いて、第２の演算を実施するよう構成されてよい。
【０２９２】
一実施形態では、複数のプロセッサはそれぞれ複数のバッファを含む。所定のプロセッサは更に、上記所定のプロセッサの複数のバッファのうちの第１のバッファに各第１のストライド値を保存するよう構成されてよい。所定のプロセッサは更に、上記所定のプロセッサの複数のバッファのうちの第２のバッファに各第２のストライド値を保存するよう構成されてよい。
【０２９３】
第１のフィールドに応じた第１のアドレス生成器を用いて第１の演算を実施するために、所定のプロセッサは更に、各第１のストライド値に応じた第１の演算を実施するよう構成されてよい。
【０２９４】
図３３
図３３は、プロセッサを動作させる一実施形態を示すフローチャートである。
【０２９５】

本方法は、フェッチユニットと、実行ユニットと、複数のアドレス生成器ユニットとを備えるマルチプロセッサ装置において動作し得る。フェッチユニットは命令を受信するよう構成されてよい。実行ユニットは複数のパイプラインユニットを備えてよい。実行ユニットは：１）命令がマルチパート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第１の演算を実施し、マルチパート命令は複数のフィールドを含み、第１の演算は、複数のフィールドのうちの第１のフィールドに応じたものであり；２）複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう、構成されてよい。

【０２９６】

実行ユニットは更に、命令が単一パート命令であることの決定に応答して、第１のパイプラインユニットを用いて第３の演算を実施するよう構成されてよい。命令が単一パート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第３の演算を実施するために、実行ユニットは更に、過去に受信したマルチパート命令に応じた、複数のパイプラインユニットのうちの第２のパイプラインユニットを用いて、第４の演算を実施するよう構成されてよい。第１のアドレス生成器は、複数のフィールドのうちの第２のフィールドに応じた算術演算を実施するよう構成されてよい。

【０２９７】

プロセッサを動作させるための方法は、以下のように動作できる。本方法は、複数のパイプラインユニットを備えるプロセッサにおいて動作できる。

【０２９８】

３３０２において、本方法は命令を受信してよい。

【０２９９】

３３０４において、本方法は、命令がマルチパート命令であることの決定に応答して、第１のパイプラインユニットを用いて第１の演算を実施してよい。マルチパート命令は複数のフィールドを含んでよく、第１の演算は複数のフィールドのうちの第１のフィールドに応じたものであってよい。

【０３００】

３３０６において、本方法は、複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施してよい。

【０３０１】

本方法は更に、命令が単一パート命令であることの決定に応答して、第１のパイプラインユニットを用いて第３の演算を実施してよい。第１のパイプラインユニットを用いて第３の演算を実施するステップは、過去に受信したマルチパート命令に応じた、複数のパイプラインユニットのうちの第２のパイプラインユニットを用いて、第４の演算を実施するステップを含んでよい。

【０３０２】

本方法は更に、命令が単一パート命令であることの決定に応答して、第１のアドレスをフェッチするステップを含んでよい。本方法はまた、複数のフィールドのサブセットに応じた複数のアドレスを生成するステップも含んでよい。

【０３０３】

本方法は、複数のプロセッサと、複数の動的に構成可能な通信要素、例えばＤＭＲとを備えるシステムにおいて動作できる。複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結されてよい。複数のプロセッサはそれぞれ複数のパイプラインユニットを備えてよい。各プロセッサは：１）命令を受信し；２）命令がマルチパート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第１の演算を実施し、ここでマルチパート命令は複数のフィールドを含み、第１の演算は、複数のフィールドのうちの第１のフィールドに応じたもの

であり；３）複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう構成されてよい。

【０３０４】

各プロセッサは更に、命令が単一パート命令であることの決定に応答して、第１のパイプラインユニットを用いて第３の演算を実施するよう構成されてよい。命令が単一パート命令であることの決定に応答して、第１のパイプラインユニットを用いて第３の演算を実施するために、各プロセッサは更に、過去に受信したマルチパート命令に応じた第２のパイプラインユニットを用いて、第４の演算を実施するよう構成されてよい。

【０３０５】

一実施形態では、各プロセッサは更に、複数のフィールドのサブセットに応じた複数のアドレスを生成するよう構成される。各プロセッサは更に、複数のアドレスのうちの第１のアドレスに応じた第１のデータを保存するよう構成されてよい。

【０３０６】

図３４

図３４は、複数のパイプラインユニットを有するプロセッサを動作させる第１の実施形態を示すフローチャートである。

【０３０７】

本方法は、フェッチユニット及び実行ユニットを備える装置において動作し得る。フェッチユニットは複数の命令を受信するよう、及び受信した複数の命令に応じた少なくとも１つの反復命令シーケンスを識別するよう、構成されてよい。少なくとも１つの反復命令シーケンスは、複数の命令のうちの少なくとも１つの命令を含んでよい。実行ユニットは複数のパイプラインユニットを含んでよく、識別された反復命令シーケンスに応じた少なくとも第１のサイクルに関して、複数のパイプラインユニットの第１のサブセットを選択的に無効化するよう構成されてよい。実行ユニットは更に、識別された反復命令シーケンスに応じた第２のサイクルに関して、複数のパイプラインユニットの第２のサブセットを選択的に無効化するよう構成されてよい。例えば上記無効化は、複数のパイプラインユニットのサブセットを低電力モードとするステップを含んでよい。装置は更に、複数のアドレス生成器ユニット、フェッチユニット、保存ユニットを備えてよい。保存ユニットは、上記少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成されてよい。フェッチユニットは、上記少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成されてよい。複数のアドレス生成器ユニットのうちの少なくとも１つは、上記少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成されてよい。

【０３０８】

プロセッサを動作させるための方法は、以下のように動作できる。

【０３０９】

まず３４０２において、プロセッサは複数の命令を受信してよい。次に３４０４においてプロセッサは、受信した複数の命令に応じた少なくとも１つの反復命令シーケンスを識別してよく、上記少なくとも１つの反復命令シーケンスは少なくとも１つの命令を含む。３４０６においてプロセッサは、識別された反復命令シーケンスに応じた少なくとも第１のサイクルに関して、複数のパイプラインユニットの第１のサブセットを選択的に無効化してよい。３４０８においてプロセッサは、識別された反復命令シーケンスに応じた第２のサイクルに関して、複数のパイプラインユニットの第２のサブセットを選択的に無効化してよい。

【０３１０】

３４１０において、プロセッサは、上記少なくとも１つの反復命令シーケンスに応じた保存ユニットを無効化してよい。あるいは、又はこれに加えて、３４１２においてプロセッサは、上記少なくとも１つの反復命令シーケンスに応じたフェッチユニットを無効化してよい。あるいは、又はこれに加えて、３４１４においてプロセッサは、上記少なくとも

１つの反復命令シーケンスに応じた、複数のアドレス生成器ユニットのサブセットを、選択的に無効化してよい。

【０３１１】

図３４の方法は、複数のプロセッサと、複数の動的に構成可能な通信要素とを備えるシステムにおいて動作でき、複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結されてよく、複数のプロセッサはそれぞれ複数のパイプラインユニットを含んでよく、また：１）複数の命令を受信し；２）受信した複数の命令に応じた少なくとも１つの反復命令シーケンスを識別し、ここで上記少なくとも１つの反復命令シーケンスは、上記複数の命令のうちの少なくとも１つの命令を含み；３）識別された反復命令シーケンスに応じた少なくとも第１のサイクルに関して、複数のパイプラインユニットの第１のサブセットを選択的に無効化し；４）識別された反復命令シーケンスに応じた第２のサイクルに関して、複数のパイプラインユニットの第２のサブセットを選択的に無効化するよう構成されてよい。追加の実施形態では、複数のプロセッサはそれぞれ、複数のアドレス生成器ユニット、フェッチユニット、保存ユニットを含んでよい。保存ユニットと、フェッチユニットと、複数のアドレス生成器ユニットのうちの少なくとも１つとのうちの１つ又は複数は、少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成されてよい。

【０３１２】

以下の番号付き段落は、追加の実施形態を記載する。

【０３１３】

多数のデータ経路の制御

１．マルチパート命令を受信するよう構成されたフェッチユニットであって、上記マルチパート命令は複数のデータフィールドを含む、フェッチユニットと；複数のパイプラインユニットを含む実行ユニットであって、上記実行ユニットは：複数のフィールドのうちの第１のフィールドに応じた、複数のパイプラインユニットのうちの所定のパイプラインユニットを用いて、第１の演算を実施し；複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう構成される、実行ユニットとを備える、装置。

【０３１４】

２．複数のパイプラインユニットはそれぞれ、複数の乗算器ユニット及び複数の加算器ユニットを含む、実施形態１の装置。

【０３１５】

３．複数のパイプラインユニットはそれぞれ、部分積の第１の数を部分積の第２の数へと圧縮するよう構成された圧縮器ユニットを含む、実施形態１の装置。

【０３１６】

４．複数のパイプラインユニットのうちの第１のパイプラインユニットの圧縮器ユニットは、複数のパイプラインユニットの第２のパイプラインユニットの圧縮器ユニットから少なくとも１つの部分積を受信するよう構成される、実施形態４の装置。

【０３１７】

５．複数のフィールドのサブセットの各フィールドは、複数のパイプラインユニットそれぞれが実施する演算を符号化する、実施形態１の装置。

【０３１８】

６．プロセッサを動作させるための方法であって、このプロセッサは複数のパイプラインユニットを含み、本方法は：マルチパート命令を受信するステップであって、このマルチパート命令は複数のフィールドを含む、ステップ；複数のフィールドのうちの第１のフィールドに応じた、複数のパイプラインユニットのうちの所定のパイプラインユニットを用いて、第１の演算を実施するステップ；複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するステップを含む、方法。

【０３１９】

７．複数のフィールドのサブセットの各フィールドは、複数のパイプラインユニットそれぞれが実施する演算を符号化する、実施形態６の方法。

【０３２０】

８．第２の演算を実施するステップは、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットのうちの第１のパイプラインユニットによって第１の部分積のセットを生成するステップと、複数のパイプラインユニットのうちの上記少なくとも２つのパイプラインユニットのうちの第２のパイプラインユニットによって第２の部分積のセットを生成するステップとを含む、実施形態６の方法。

【０３２１】

９．複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットのうちの第１のパイプラインユニットによって、第１の部分積のセットを生成するステップは、部分積の第１の数を部分積の第２の数へと圧縮するステップを含み、第２の数は第１の数未満である、実施形態８の方法。

【０３２２】

１０．複数のパイプラインユニットはそれぞれ、複数の乗算器ユニット及び複数の加算器ユニットを含む、実施形態６の方法。

【０３２３】

１１．複数のプロセッサであって、この複数のプロセッサはそれぞれ複数のパイプラインユニットを含む、複数のプロセッサと；複数の動的に構成可能な通信要素とを備える、システムであって、複数のプロセッサ及び複数の動的に構成可能な通信要素は散在型の配置において連結され、複数のプロセッサはそれぞれ：マルチパート命令を受信し（ここでマルチパート命令は複数のフィールドを含み）；複数のフィールドのうちの第１のフィールドに応じた、複数のパイプラインユニットのうちの所定のパイプラインユニットを用いて、第１の演算を実施し；複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを用いて、第２の演算を実施するよう構成される、システム。

【０３２４】

１２．各プロセッサの複数のパイプラインユニットはそれぞれ、複数の乗算器ユニット及び複数の加算器ユニットを含む、実施形態１１のシステム。

【０３２５】

１３．複数のプロセッサそれぞれの複数のパイプラインユニットはそれぞれ、部分積の第１の数を部分積の第２の数へと圧縮するよう構成された圧縮器ユニットを含む、実施形態１１のシステム。

【０３２６】

１４．複数のプロセッサのうちの所定のプロセッサの複数のパイプラインユニットのうちの第１のパイプラインユニットの圧縮器ユニットは更に、複数のプロセッサのうちの所定のプロセッサの複数のパイプラインユニットのうちの第２のパイプラインユニットの別の圧縮器ユニットから、少なくとも１つの部分積を受信するよう構成される、実施形態１３のシステム。

【０３２７】

１５．複数のフィールドのサブセットの各フィールドは、複数のパイプラインユニットそれぞれが実施する演算を符号化する、実施形態１１のシステム。

【０３２８】

累算の転送

１．命令を受信するよう構成されたフェッチユニットと；複数のパイプラインユニットを含む実行ユニットとを備える、装置であって、上記複数のパイプラインユニットはそれぞれアキュムレータユニットを含み、実行ユニットは：複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて、受信した命令に応じた第１の演算を実施して結果を生成し；複数のパイプラインユニットのうちの第１のパイプラインユニットのアキュムレータユニットに上記結果を保存し；複数のパイプラインユニットのうちの第１のパ

イプラインユニットのアキュムレータユニットに保存された結果を、プロセッサのアキュムレータユニットに伝送するよう、構成される、装置。

【０３２９】

２．複数のパイプラインユニットはそれぞれ、複数の乗算器ユニット及び複数の加算器ユニットを含む、実施形態１の装置。

【０３３０】

３．第１の演算は乗算演算を含み、複数のパイプラインユニットの第１のパイプラインユニットのアキュムレータユニットに結果を保存するために、実行ユニットは更に、複数のパイプラインユニットのうちの第１のパイプラインユニットに、乗算演算の少なくとも１つの部分積を保存するよう構成される、実施形態１の装置。

【０３３１】

４．命令は複数のフィールドを含み、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて、受信した命令に応じた第１の演算を実施するために、実行ユニットは更に、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて、複数のフィールドのうちの第１のフィールドに応じた第１の演算を実施するよう構成される、実施形態１の装置。

【０３３２】

５．実行ユニットは更に、複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう構成される、実施形態５の装置。

【０３３３】

６．複数のプロセッサを動作させるための方法であって、本方法は：複数のプロセッサのうちの第１のプロセッサを用いて第１の演算を実施するステップであって、第１のプロセッサは第１の複数のアキュムレータを含み、第１の演算を実施するステップは、第１の複数のアキュムレータのうちの所定のアキュムレータに値を保存するステップを含む、ステップ；第１の複数のアキュムレータのうちの上記所定のアキュムレータに保存された値を、複数のプロセッサのうちの第２のプロセッサが含む第２の複数のアキュムレータのうちの所定のアキュムレータに伝送するステップ；第２の複数のアキュムレータのうちの上記所定のアキュムレータに保存された値に応じた、複数のプロセッサのうちの第２のプロセッサを用いて、第２の演算を実施するステップを含む、方法。

【０３３４】

７．第１のプロセッサによって第１のマルチパート命令を受信するステップを更に含み、第１のマルチパート命令は、第１の複数のフィールドを含む、実施形態６の方法。

【０３３５】

８．第１のプロセッサを用いて第１の演算を実施するステップは、第１の複数のフィールドのうちの第１のフィールドに応じた、複数のプロセッサのうちの第１のプロセッサを用いて、第１の演算を実施するステップを含む、実施形態７の方法。

【０３３６】

９．複数のプロセッサのうちの第２のプロセッサによって第２のマルチパート命令を受信するステップを含み、第２のマルチパート命令は、第２の複数のフィールドを含む、実施形態８の方法。

【０３３７】

１０．第２の複数のアキュムレータのうちの所定のアキュムレータに保存された値に応じた第２のプロセッサを用いて第２の演算を実施するステップは、第２の複数のフィールドのうちの第１のフィールドに応じた、複数のプロセッサのうちの第２のプロセッサを用いて第２の演算を実施するステップを含む、実施形態９の方法。

【０３３８】

１１．複数のプロセッサであって、これら複数のプロセッサはそれぞれ複数のアキュムレータユニットを含む、複数のプロセッサと；複数の動的に構成可能な通信要素とを備える、システムであって、複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在

型の配置において連結され、複数のプロセッサのうちの第１のプロセッサは：第１の命令を受信し；第１の命令に応じた第１の演算を実施して結果を生成し；第１のプロセッサの複数のアキュムレータユニットのうちの所定のアキュムレータユニットに上記結果を保存し；第１のプロセッサの複数のアキュムレータユニットのうちの上記所定のアキュムレータユニットに保存された結果を、複数のプロセッサのうちの第２のプロセッサの複数のアキュムレータユニットのうちの所定のアキュムレータユニットに伝送するよう構成される、システム。

【０３３９】

１２．複数のプロセッサのうちの第２のプロセッサは：第２の命令を受信し；第２の命令と、複数のプロセッサのうちの第２のプロセッサの複数のアキュムレータユニットのうちの上記所定のアキュムレータユニットに保存された値とに応じた第２の演算を実施するよう、構成される、実施形態１１のシステム。

【０３４０】

１３．第１の命令は第１のマルチパート命令を含み、第１のマルチパート命令は第１の複数のフィールドを含み、第２の命令は第２のマルチパート命令を含み、第２のマルチパート命令は第２の複数のフィールドを含む、実施形態１２のシステム。

【０３４１】

１４．第１の命令に応じた第１の演算を実施するために、複数のプロセッサのうちの第１のプロセッサは更に、第１の複数のフィールドのうちの第１のフィールドに応じた第１の演算を実施するよう構成される、実施形態１３のシステム。

【０３４２】

１５．第２の命令及び所定のアキュムレータユニットに保存された値に応じた第２の演算を実施するために、複数のプロセッサのうちの第２のプロセッサは更に、第２の複数のフィールドのうちの第１のフィールドに応じた第２の演算を実施するよう構成される、実施形態１３のシステム。

【０３４３】

ＡＧＵカップリング（循環アドレス指定）

１．マルチパート命令を受信するよう構成されたフェッチユニット（ここでマルチパート命令は複数のフィールドを含む）と；複数のアドレス生成器ユニットとを備える、装置であって、複数のアドレス生成器ユニットのうちの第１のアドレス生成器ユニットは、複数のフィールドのうちの第１のフィールドに応じた第１の演算を実施して、第１の結果を生成し、更に、複数のアドレス生成器ユニットのうちの第２のアドレス生成器ユニットは、複数のフィールドのうちの第２のフィールド及び上記第１の結果に応じた第２の演算を実施する、装置。

【０３４４】

２．複数のバッファを更に備える、実施形態１の装置。

【０３４５】

３．複数のバッファのうちの第１のバッファは、複数のフィールドのうちの少なくとも１つのフィールドに応じた第１のストライド値を保存するよう構成される、実施形態２の装置。

【０３４６】

４．複数のバッファのうちの第２のバッファは、複数のフィールドのうちの少なくとも１つの別のフィールドに応じた第２のストライド値を保存するよう構成される、実施形態３の装置。

【０３４７】

５．複数のフィールドのうちの第１のフィールドに応じた第１の演算を実施するために、複数のアドレス生成器ユニットのうちの第１のアドレス生成器ユニットは更に、第１のストライド値に応じた第１の演算を実施するよう構成される、実施形態３の装置。

【０３４８】

６．プロセッサを動作させるための方法であって、上記プロセッサは、複数のアドレス

生成器ユニットを含み、本方法は：マルチパート命令を受信するステップであって、このマルチパート命令は複数のフィールドを含む、ステップ；複数のフィールドのうちの第１のフィールドに応じた、複数のアドレス生成器ユニットのうちの第１のアドレス生成器ユニットを用いて、第１の演算を実施するステップ；複数のフィールドのうちの第２のフィールドと、第１の結果とに応じた、複数のアドレス生成器ユニットのうちの第２のアドレス生成器ユニットを用いて、第２の演算を実施するステップを含む、方法。

【０３４９】

７．プロセッサは更に、複数のバッファを含む、実施形態６の方法。

【０３５０】

８．複数のバッファのうちの第１のバッファに第１のストライド値を保存するステップを更に含む、実施形態７の方法。

【０３５１】

９．複数のバッファのうちの第２のバッファに第２のストライド値を保存するステップを更に含む、実施形態８の方法。

【０３５２】

１０．複数のフィールドのうちの第１のフィールドに応じた、複数のアドレス生成器ユニットのうちの第１のアドレス生成器ユニットを用いて第１の演算を実施するステップは、第１のストライド値に応じた第１の演算を実施するステップを含む、実施形態７の方法。

【０３５３】

１１．複数のプロセッサと；複数の動的に構成可能な通信要素とを備える、システムであって、複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結され、複数のプロセッサのうちの所定のプロセッサは：マルチパート命令を受信し、ここでマルチパート命令は複数のフィールドを含み；複数のフィールドのうちの第１のフィールドに応じた、複数の生成器のうちの第１のアドレス生成器を用いて、第１の演算を実施して、第１の結果を生成し；複数のフィールドのうちの第２のフィールドと上記第１の結果とに応じた、複数の生成器のうちの第２のアドレス生成器を用いて、第２の演算を実施するよう構成される、システム。

【０３５４】

１２．複数のプロセッサはそれぞれ複数のバッファを含む、実施形態１１のシステム。

【０３５５】

１３．複数のプロセッサのうちの所定のプロセッサは更に、上記所定のプロセッサの複数のバッファのうちの第１のバッファに各第１のストライド値を保存するよう構成される、実施形態１２のシステム。

【０３５６】

１４．複数のプロセッサのうちの所定のプロセッサは更に、上記所定のプロセッサの複数のバッファのうちの第２のバッファに各第２のストライド値を保存するよう構成される、実施形態１３のシステム。

【０３５７】

１５．複数のフィールドのうちの第１のフィールドに応じた、複数の生成器のうちの第１のアドレス生成器を用いて第１の演算を実施するために、複数のプロセッサのうちの所定のプロセッサは更に、各第１のストライド値に応じた第１の演算を実施するよう構成される、実施形態１３のシステム。

【０３５８】

単一パート／マルチパートの決定を有する命令の受信

１．命令を受信するよう構成されたフェッチユニットと；複数のパイプラインユニットを含む実行ユニットとを備える、装置であって、実行ユニットは：命令がマルチパート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第１の演算を実施し、マルチパート命令は複数のフィールドを含み、第１の演算は、複数のフィールドのうちの第１のフィールドに応じたものであり；複数の

フィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう、構成される、装置。

【０３５９】

２．実行ユニットは更に、命令が単一パート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第３の演算を実施するよう構成される、実施形態１の装置。

【０３６０】

３．命令が単一パート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第３の演算を実施するために、実行ユニットは更に、過去に受信したマルチパート命令に応じた、複数のパイプラインユニットのうちの第２のパイプラインユニットを用いて、第４の演算を実施するよう構成される、実施形態２の装置。

【０３６１】

４．複数のアドレス生成器ユニットを更に備える、実施形態１の装置。

【０３６２】

５．複数のアドレス生成器のうちの第１のアドレス生成器は、複数のフィールドのうちの第２のフィールドに応じた算術演算を実施するよう構成される、実施形態４の装置。

【０３６３】

６．プロセッサを動作させるための方法であって、プロセッサは複数のパイプラインユニットを含み、本方法は；命令を受信するステップ；命令がマルチパート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第１の演算を実施するステップであって、ここでマルチパート命令は複数のフィールドを含み、第１の演算は複数のフィールドのうちの第１のフィールドに応じたものである、ステップ；複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するステップを含む、方法。

【０３６４】

７．命令が単一パート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第３の演算を実施するステップを更に含む、実施形態６の方法。

【０３６５】

８．複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第３の演算を実施するステップは、過去に受信したマルチパート命令に応じた、複数のパイプラインユニットのうちの第２のパイプラインユニットを用いて、第４の演算を実施するステップを含む、実施形態６の方法。

【０３６６】

９．命令が単一パート命令であることの決定に応答して、第１のアドレスをフェッチするステップを更に含む、実施形態６の方法。

【０３６７】

１０．複数のフィールドのサブセットに応じた複数のアドレスを生成するステップを更に含む、実施形態６の方法。

【０３６８】

１１．複数のプロセッサであって、複数のプロセッサはそれぞれ複数のパイプラインユニットを含む、複数のプロセッサと；複数の動的に構成可能な通信要素とを備える、システムであって、複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結され、複数のプロセッサはそれぞれ：命令を受信し；命令がマルチパート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第１の演算を実施し、ここでマルチパート命令は複数のフィールドを含み、第１の演算は、複数のフィールドのうちの第１のフィールドに応じたものであり；

複数のフィールドのうちの第２のフィールドに応じた、複数のパイプラインユニットのうちの少なくとも２つのパイプラインユニットを並列に用いて、第２の演算を実施するよう、構成される、システム。

【０３６９】

１２．複数のプロセッサはそれぞれ、命令が単一パート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第３の演算を実施するよう、更に構成される、実施形態１１のシステム。

【０３７０】

１３．命令が単一パート命令であることの決定に応答して、複数のパイプラインユニットのうちの第１のパイプラインユニットを用いて第３の演算を実施するために、複数のプロセッサはそれぞれ、過去に受信したマルチパート命令に応じた、複数のパイプラインユニットのうちの第２のパイプラインユニットを用いて、第４の演算を実施するよう、更に構成される、実施形態１２のシステム。

【０３７１】

１４．複数のプロセッサはそれぞれ、複数のフィールドのサブセットに応じた複数のアドレスを生成するよう、更に構成される、実施形態１１のシステム。

【０３７２】

１５．複数のプロセッサはそれぞれ、複数のアドレスのうちの第１のアドレスに応じた第１のデータを保存するよう、更に構成される、実施形態１１のシステム。

【０３７３】

反復ループ中のプロセッサの非使用部分の電力ダウン

１．複数の命令を受信するよう、及び受信した複数の命令に応じた少なくとも１つの反復命令シーケンスを識別するよう、構成された、フェッチユニットであって、少なくとも１つの反復命令シーケンスは、複数の命令のうちの少なくとも１つの命令を含む、フェッチユニットと；複数のパイプラインユニットを含む実行ユニットとを備える、装置であって、上記実行ユニットは：識別された反復命令シーケンスに応じた少なくとも第１のサイクルに関して、複数のパイプラインユニットの第１のサブセットを選択的に無効化し；識別された反復命令シーケンスに応じた第２のサイクルに関して、複数のパイプラインユニットの第２のサブセットを選択的に無効化するよう、構成される、装置。

【０３７４】

２．複数のアドレス生成器ユニット、フェッチユニット、保存ユニットを更に備える、実施形態１の装置。

【０３７５】

３．保存ユニットは、上記少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成される、実施形態２の装置。

【０３７６】

４．フェッチユニットは、上記少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成される、実施形態２の装置。

【０３７７】

５．複数のアドレス生成器ユニットのうちの少なくとも１つは、上記少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成される、実施形態２の装置。

【０３７８】

６．プロセッサを動作させるための方法であって、プロセッサは複数のパイプラインユニットを含み、本方法は：複数の命令を受信するステップ；受信した複数の命令に応じた少なくとも１つの反復命令シーケンスを識別するステップであって、上記少なくとも１つの反復命令シーケンスは少なくとも１つの命令を含む、ステップ；識別された反復命令シーケンスに応じた少なくとも第１のサイクルに関して、複数のパイプラインユニットの第１のサブセットを選択的に無効化するステップ；識別された反復命令シーケンスに応じた第２のサイクルに関して、複数のパイプラインユニットの第２のサブセットを選択的に無効化するステップを含む、方法。

【０３７９】
７．プロセッサは、フェッチユニット、保存ユニット、複数のアドレス生成器ユニットを更に含む、実施形態６の方法。
【０３８０】
８．上記少なくとも１つの反復命令シーケンスに応じた保存ユニットを無効化するステップを更に含む、実施形態７の方法。
【０３８１】
９．上記少なくとも１つの反復命令シーケンスに応じたフェッチユニットを無効化ステップを更に含む、実施形態７の方法。
【０３８２】
１０．上記少なくとも１つの反復命令シーケンスに応じた、複数のアドレス生成器ユニットのサブセットを、選択的に無効化するステップを更に含む、実施形態７の方法。
【０３８３】
１１．複数のプロセッサであって、複数のプロセッサはそれぞれ複数のパイプラインユニットを含む、複数のプロセッサと；複数の動的に構成可能な通信要素とを備える、システムであって、複数のプロセッサ及び複数の動的に構成可能な通信要素は、散在型の配置において連結され、複数のプロセッサはそれぞれ：複数の命令を受信し；受信した複数の命令に応じた少なくとも１つの反復命令シーケンスを識別し、ここで上記少なくとも１つの反復命令シーケンスは、上記複数の命令のうちの少なくとも１つの命令を含み；識別された反復命令シーケンスに応じた少なくとも第１のサイクルに関して、複数のパイプラインユニットの第１のサブセットを選択的に無効化し；識別された反復命令シーケンスに応じた第２のサイクルに関して、複数のパイプラインユニットの第２のサブセットを選択的に無効化するよう構成される、システム。
【０３８４】
１２．複数のプロセッサはそれぞれ、複数のアドレス生成器ユニット、フェッチユニット、保存ユニットを含む、実施形態１１のシステム。
【０３８５】
１３．保存ユニットは、少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成される、実施形態１２のシステム。
【０３８６】
１４．フェッチユニットは、少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成される、実施形態１２のシステム。
【０３８７】
１５．複数のアドレス生成器ユニットのうちの少なくとも１つは、少なくとも１つの反復命令シーケンスに応じた低電力モードに入るよう構成される、実施形態１２のシステム。
【０３８８】
様々な実施形態において、コンピュータ可読メモリ媒体は、ソフトウェアアプリケーションのスワップに関わる機能等の上述の様々な機能を実装するためにＭＰＳのプロセッサ及び／又は１つ若しくは複数の外部プロセッサによって実行可能なプログラム命令を記憶してよい。一般に、コンピュータ可読メモリ媒体は、実行した場合に本出願に記載の機能の一部分又は全てを実装する、命令のいずれのセットを含んでよい。一般にコンピュータ可読メモリ媒体は、コンピュータシステムに命令及び／又はデータを提供するために使用中にコンピュータがアクセスできる、いずれのストレージ媒体を含んでよい。例えばコンピュータ可読メモリ媒体は、磁気又は光媒体、例えばディスク（固定若しくは消去可能）、テープ、ＣＤ‐ＲＯＭ、ＤＶＤ‐ＲＯＭ、ＣＤ‐Ｒ、ＣＤ‐ＲＷ、ＤＶＤ‐Ｒ、ＤＶＤ‐ＲＷ又はＢｌｕ‐ｒａｙ（登録商標）といったストレージ媒体を含んでよい。ストレージ媒体は更に、ユニバーサルシリアルバス（ＵＳＢ）インタフェース、フラッシュメモリインターフェイス（ＦＭＩ）、シリアルペリフェラルインタフェース（ＳＰＩ）等の周辺インタフェースを介してアクセス可能な、ＲＡＭ（例えば同期ダイナミックＲＡＭ（ＳＤ

ＲＡＭ）、ラムバスＤＲＡＭ（ＲＤＲＡＭ）、スタティックＲＡＭ（ＳＲＡＭ）等）、ＲＯＭ、フラッシュメモリ、不揮発性メモリ（例えばフラッシュメモリ）等の揮発性又は不揮発性メモリ媒体を含んでよい。ストレージ媒体は、微小電気機械システム（ＭＥＭＳ）、並びにネットワーク及び／又は無線リンク等の通信媒体を介してアクセス可能なストレージ媒体を含んでよい。キャリア媒体は、コンピュータがアクセス可能なストレージ媒体、及び有線又は無線伝送等の伝送媒体を含んでよい。

【０３８９】

好ましい実施形態との関連で以上の実施形態について説明したが、本明細書に記載した具体的形態に上記好ましい実施形態を限定することは意図されておらず、反対に、添付の請求項によって定義されるような本発明の精神及び範囲内に合理的に含まれ得るような代替例、修正例、均等物を上記好ましい実施形態が包含することが意図されている。

【図１】

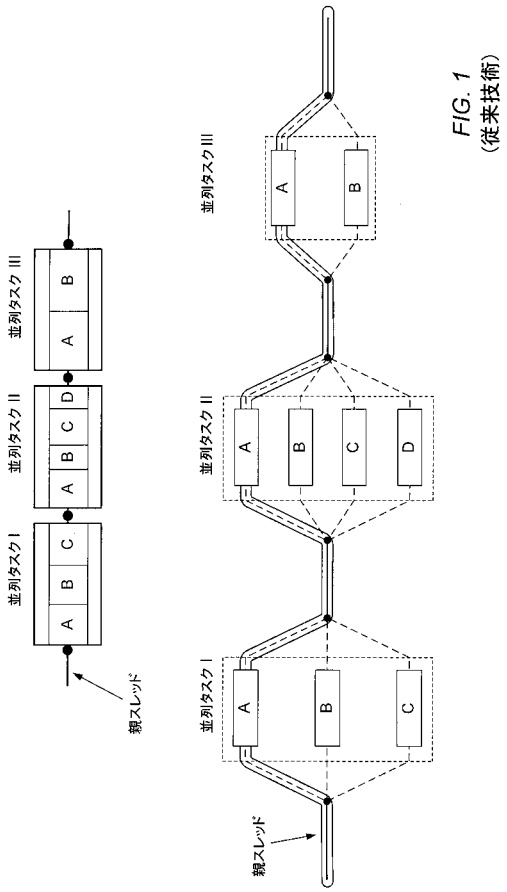


FIG. 1
(従来技術)

【図２】

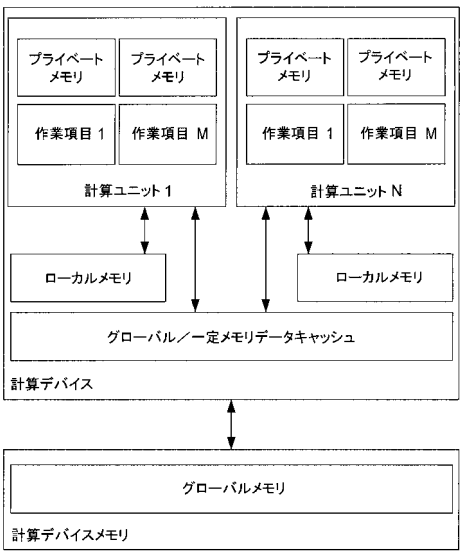


FIG. 2
(従来技術)

【図３】

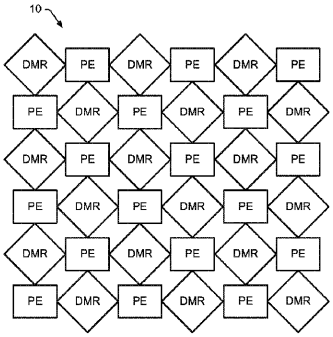


FIG. 3

【図４】

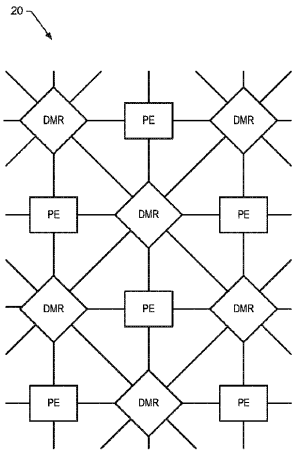


FIG. 4

【図５】

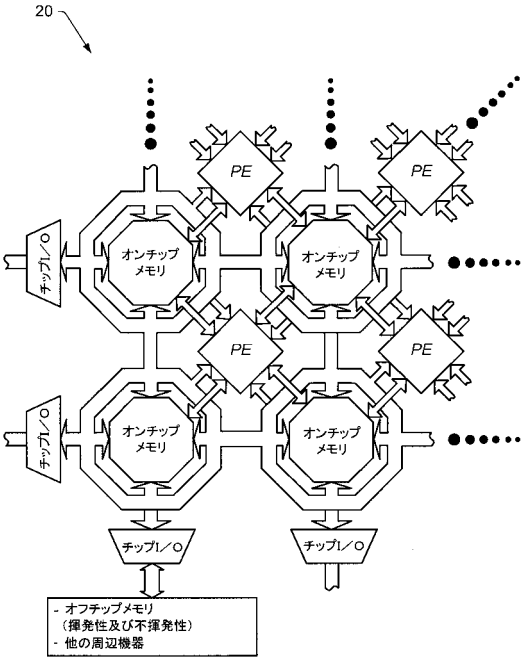


FIG. 5

【図６】

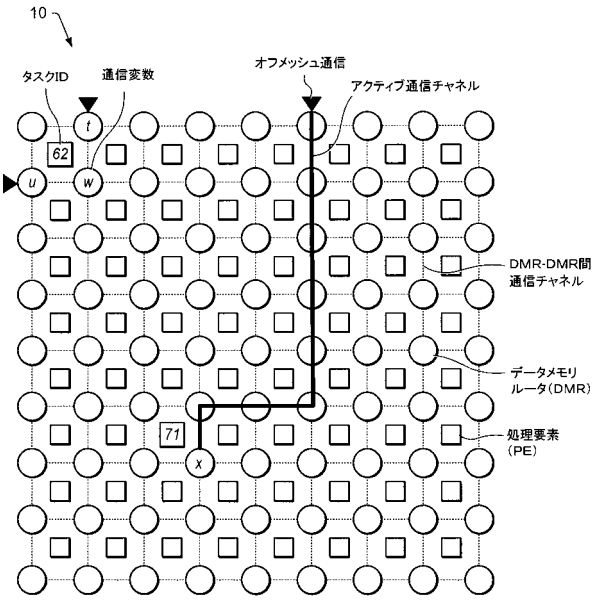


FIG. 6

【図７】

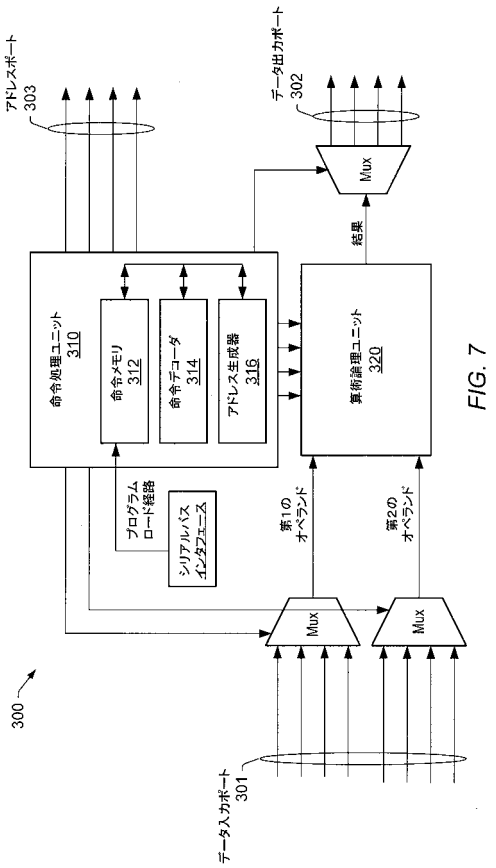


FIG. 7

【図８】

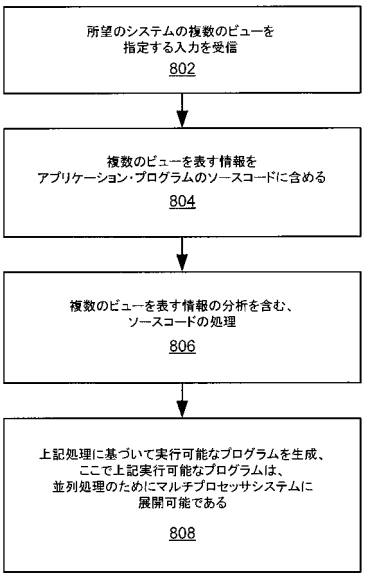


FIG. 8

【図９Ａ】

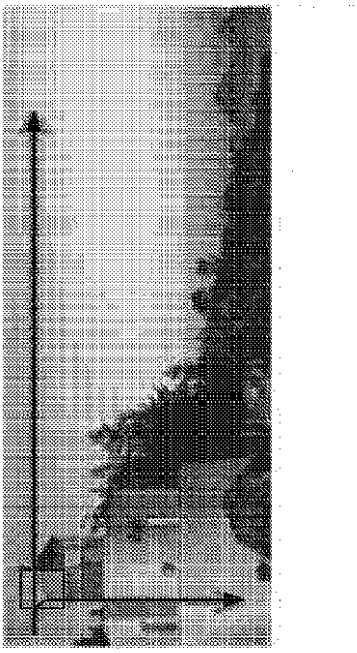


FIG. 9A

【図９Ｂ】

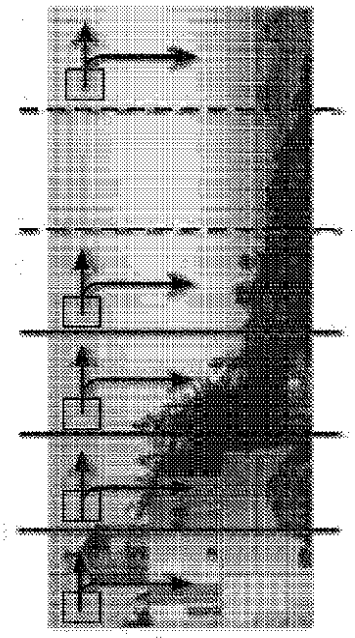


FIG. 9B

【図１０】

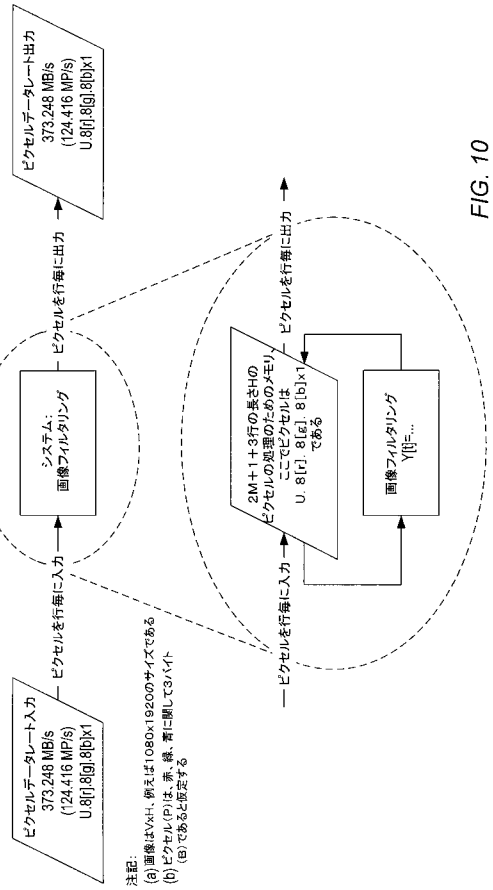


FIG. 10

【図１１】

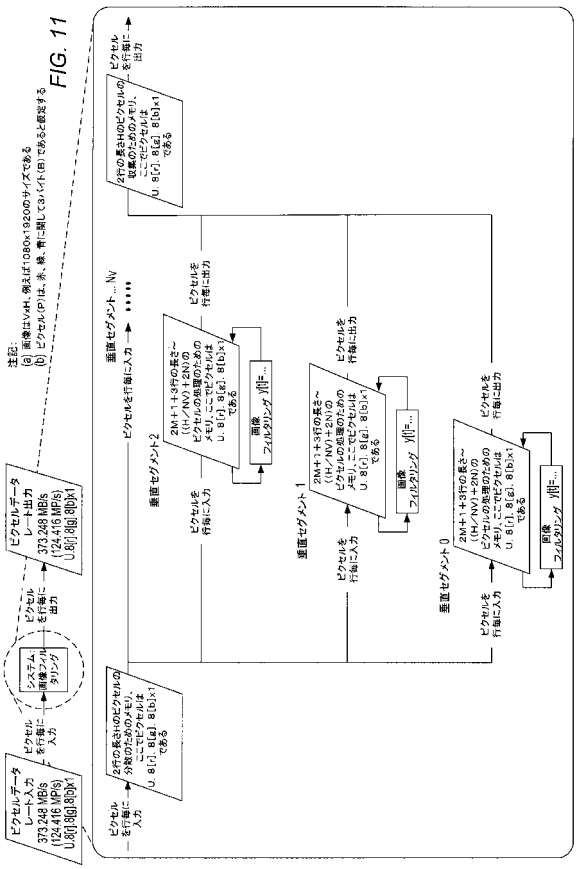


FIG. 11

【図１２】

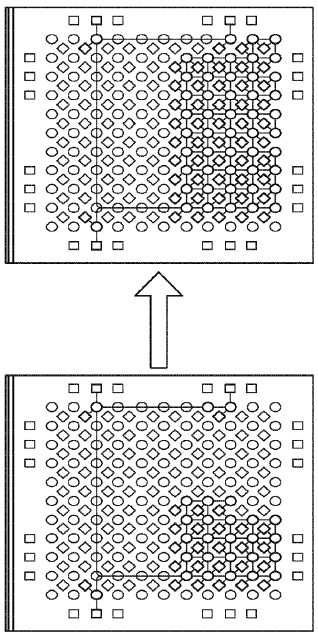


FIG. 12

【図１３】

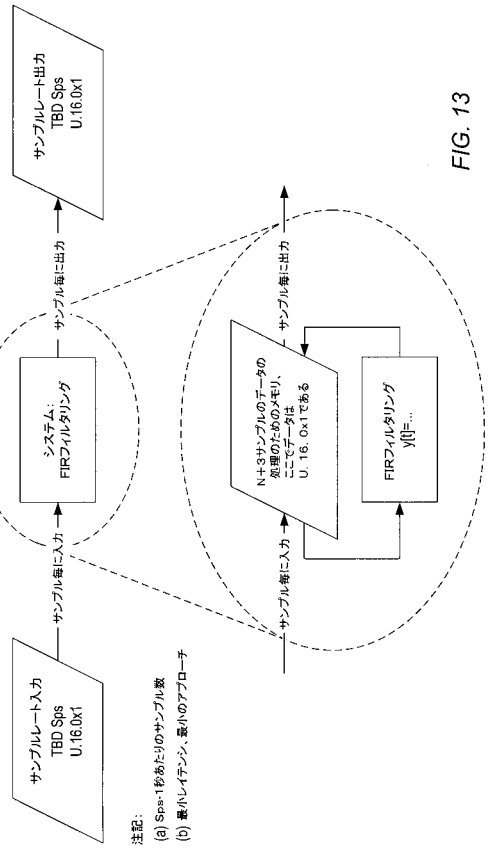
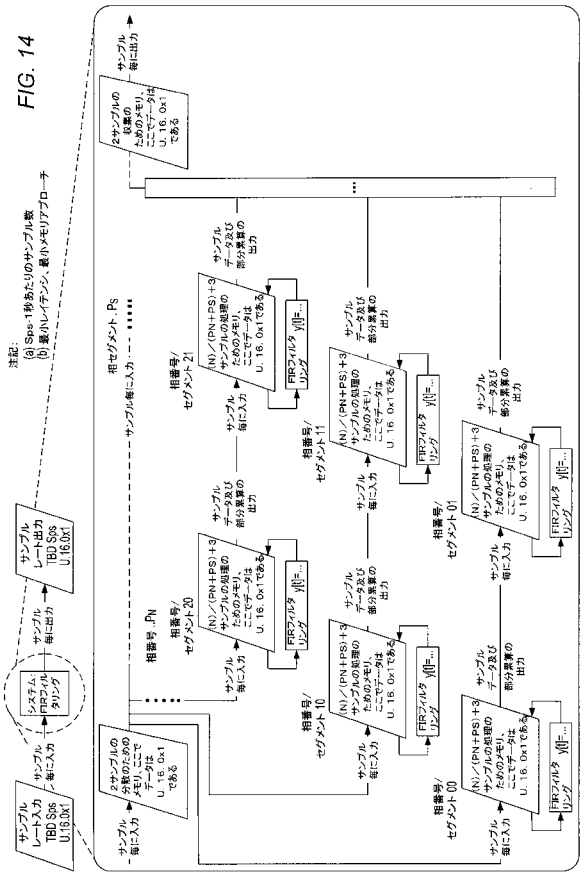


FIG. 13

【図１４】



【図１５】

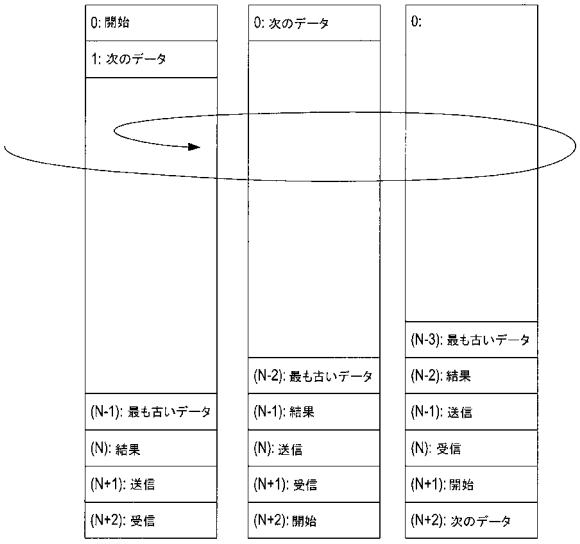


FIG. 15

【図１６】

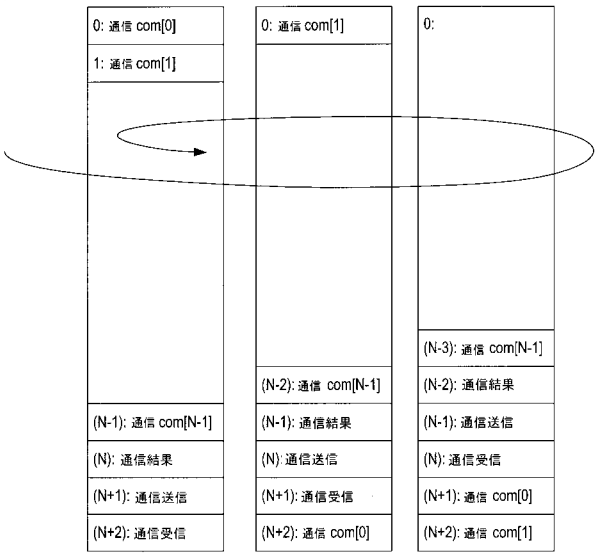


FIG. 16

【図１７】

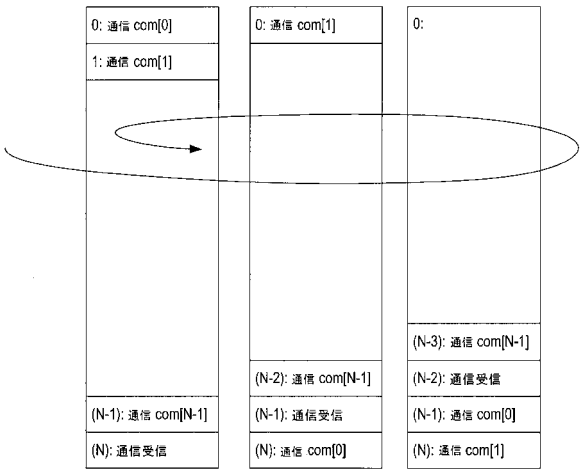


FIG. 17

【図１８】

第4世代(公称9)		
ステージ	説明	
IFA	命令のフェッチ (mem rd)	フェッチ
IFB	命令の復号	復号／発行
IFC	命令の発行&オペランドアドレス	
OFA	調停	オペランドのフェッチ
OFB	オペランドの読み出し (mem rd)	
OFC	オペランドの返信	
EXE	実行(変数)&結果アドレス	実行
WBA	結果データ&調停	ライトバック
WBB	結果書き込み (mem wrt)	

FIG. 18

【図１９】

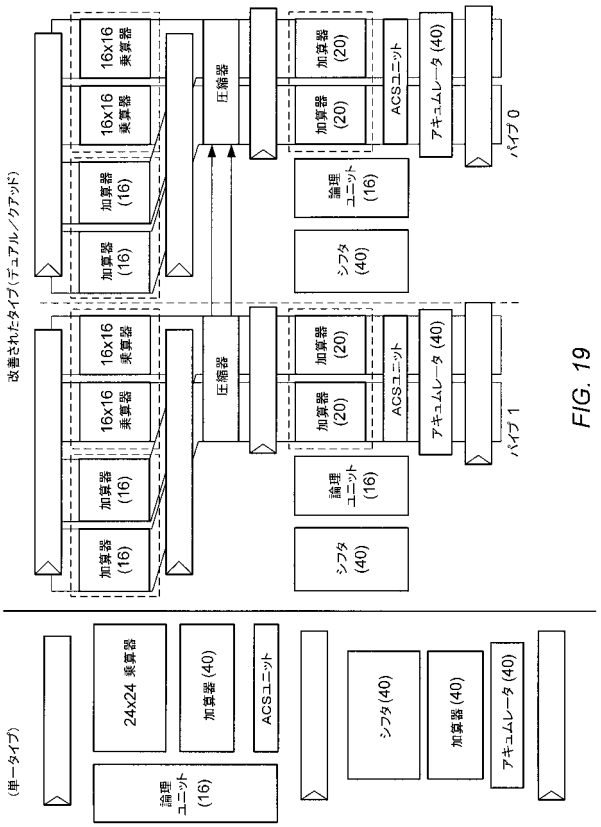


FIG. 19

【図２０】

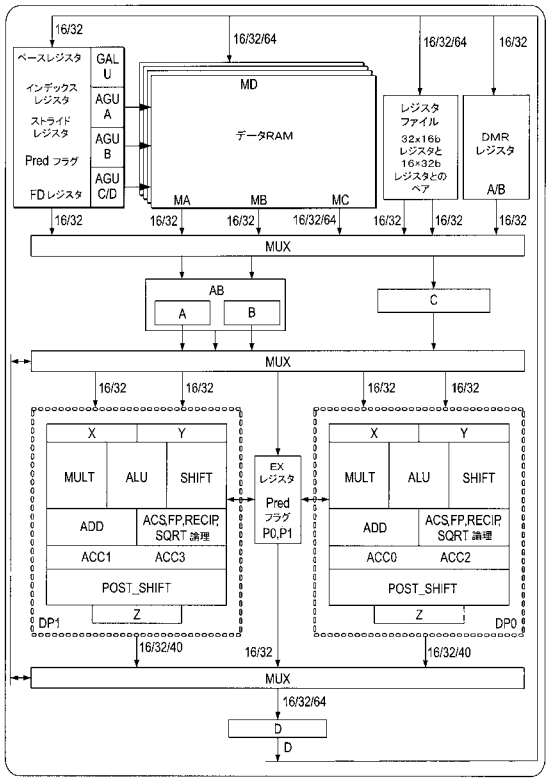


FIG. 20

【図２１】

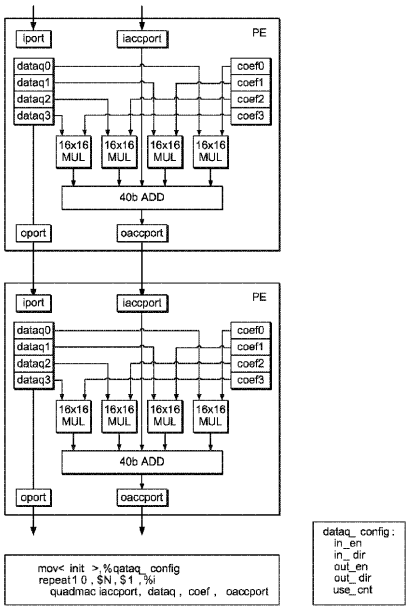


FIG. 21

【図２２】

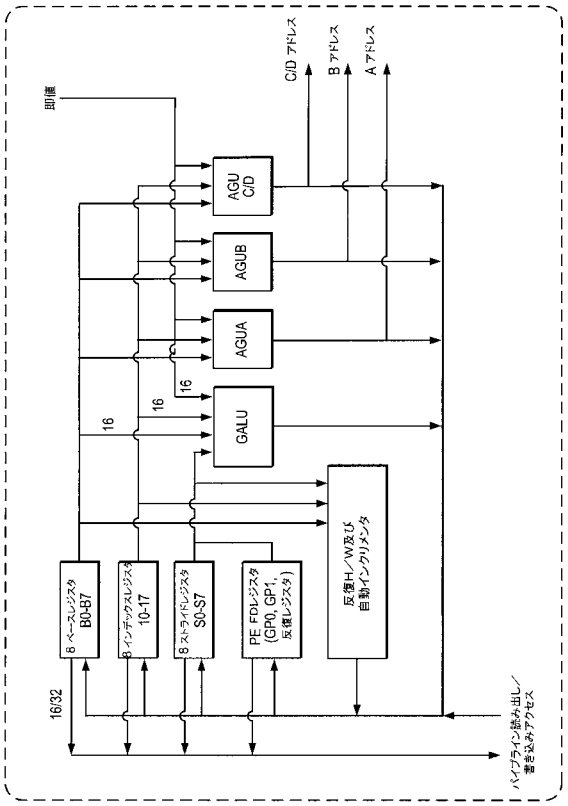


FIG. 22

【図２３】

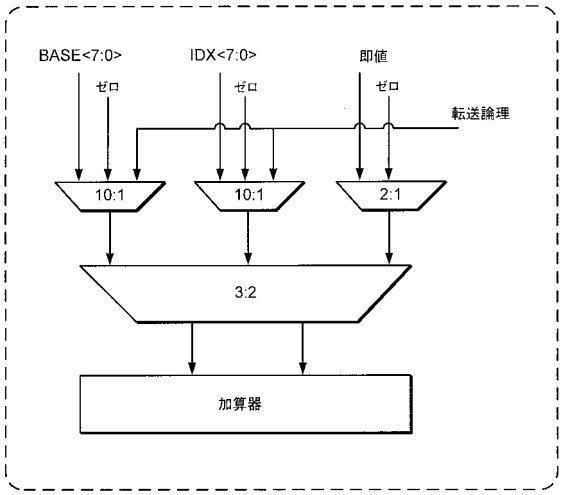
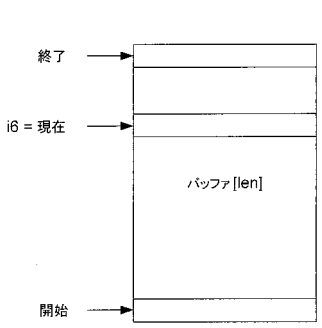


FIG. 23

【図２４】



通常モード:

```
i6 = start;
s6 = incr;
for() {
    i6 += s6;
}
```

1D循環モード:

```
i6 = start;
i7 = end;
s6 = incr;
s7 = wrap_incr = (incr-len) ;
wrap = 0;
for() {
    if (wrap) i6 += s7;
    else i6 += s6;
    wrap = (i6 >= i7) ;
}
```

FIG. 24

【図２５】

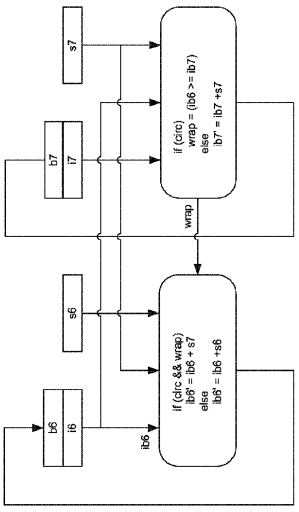


FIG. 25

【図２６】

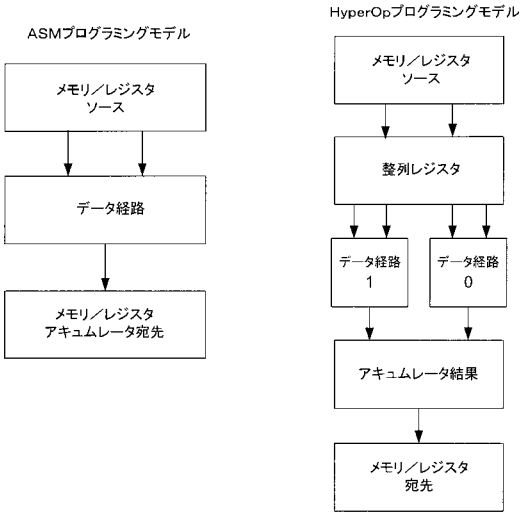


FIG. 26

【図２７】

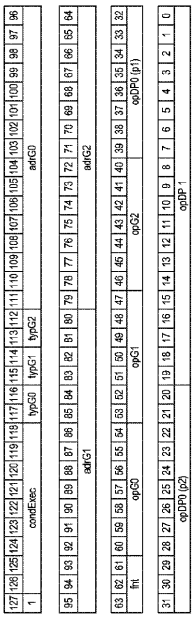


FIG. 27

【図２８】

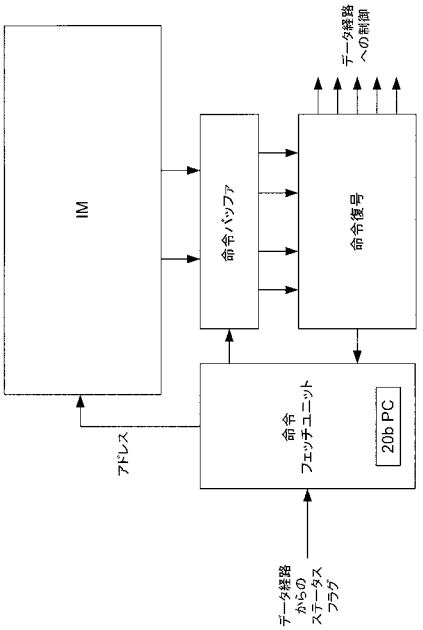


FIG. 28

【図２９】

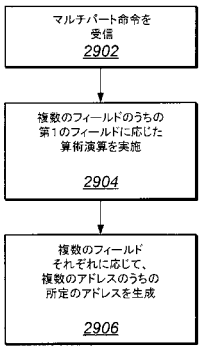


FIG. 29

【図３０】

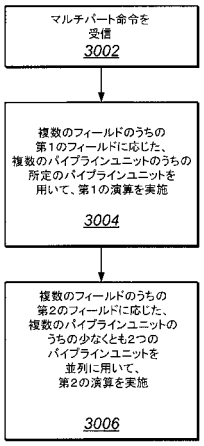


FIG. 30

【図３１】

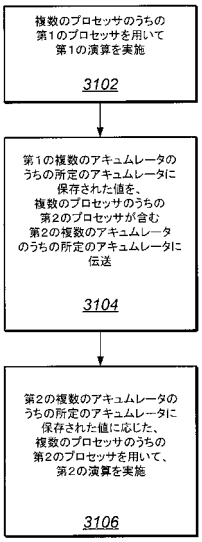


FIG. 31

【図３２】

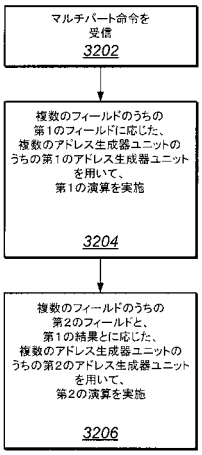


FIG. 32

【図３３】

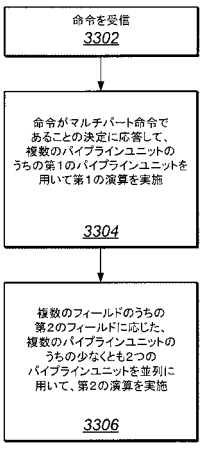


FIG. 33

【図３４】

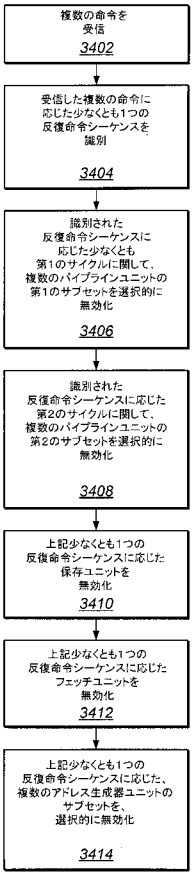

複数の命令を
受信
3402
受信した複数の命令に
応じた少なくとも1つの
反復命令シーケンスを
識別
3404
識別された
反復命令シーケンスに
応じた少なくとも
第1のサイクルに関して、
複数のパイプラインユニットの
第1のサブセットを選択的に
無効化
3406
識別された
反復命令シーケンスに応じた
第2のサイクルに関して、
複数のパイプラインユニットの
第2のサブセットを選択的に
無効化
3408
上記少なくとも1つの
反復命令シーケンスに応じた
保存ユニットを
無効化
3410
上記少なくとも1つの
反復命令シーケンスに応じた
フェッチユニットを
無効化
3412
上記少なくとも1つの
反復命令シーケンスに応じた、
複数のアドレス生成器ユニットの
サブセットを、
選択的に無効化
3414

FIG. 34

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2014/039345

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/30 G06F9/38
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 2010/093828 A1 (QUARTICS INC [US]; AHMAD MOHAMMAD [US]; USMAN MOHAMMAD [US]; AHMED SHE) 19 August 2010 (2010-08-19) page 47, line 76 - page 49, line 18	1-10
X	US 6 453 405 B1 (HOYLE DAVID [US] ET AL) 17 September 2002 (2002-09-17) paragraphs [0038], [0039] paragraphs [0061], [0063]	1-10
A	HINTON G ET AL: "PERFORMANCE ENHANCEMENTS OF THE SUPERSCALAR I960 PROCESSOR FAMILY", WESCON TECHNICAL PAPERS, WESTERN PERIODICALS CO. NORTH HOLLYWOOD, US, vol. 34, 1 November 1990 (1990-11-01), pages 463-466, XP000227912, *Page 464: "Superscalar Dispatch opportunities"*	1,6

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
6 November 2014	02/02/2015

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Daskalakis, T

1

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/039345

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

1-10

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

International Application No. PCT/US2014/039345

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. claims: 1-10

VLIW processor comprising multiple address generation units, and method of processing.

---

2. claims: 11-15

Apparatus comprising a plurality of VLIW processors and configurable communication elements.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2014/039345

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2010093828 A1	19-08-2010	CN 102804165 A	28-11-2012
		EP 2396735 A1	21-12-2011
		US 2010321579 A1	23-12-2010
		WO 2010093828 A1	19-08-2010
US 6453405 B1	17-09-2002	NONE	

Form PCT/ISA/210 (patent family annex) (April 2005)

フロントページの続き

(81)指定国 AP(BW,GH,GM,KE,LR,LS,MW,MZ,NA,RW,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,RU,TJ,TM),EP(AL,AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,HR,HU,IE,IS,IT,LT,LU,LV,MC,MK,MT,NL,NO,PL,PT,RO,RS,SE,SI,SK,SM,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,KM,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AO,AT,AU,AZ,BA,BB,BG,BH,BN,BR,BW,BY,BZ,CA,CH,CL,CN,CO,CR,CU,CZ,DE,DK,DM,DO,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,GT,HN,HR,HU,ID,IL,IN,IR,IS,JP,KE,KG,KN,KP,KR,KZ,LA,LC,LK,LR,LS,LT,LU,LY,MA,MD,ME,MG,MK,MN,MW,MX,MY,MZ,NA,NG,NI,NO,NZ,OM,PA,PE,PG,PH,PL,PT,QA,RO,RS,RU,RW,SA,SC,SD,SE,SG,SK,SL,SM,ST,SV,SY,TH,TJ,TM,TN,TR,TT,TZ,UA,UG,US

(72)発明者 ドブス，カール・エス
アメリカ合衆国・７８７３７・テキサス州・オースティン・ジム ブリッジャー ドライブ・１２０１０
(72)発明者 ソルカ，マイケル・ビー
アメリカ合衆国・７８７３１・テキサス州・オースティン・キャット マウンテン コーヴ・６２０９
(72)発明者 トロシーノ，マイケル・アール
アメリカ合衆国・７８７３５・テキサス州・オースティン・マグデレナ ドライブ・５３４４
(72)発明者 フォークナー，ケネス・アール
アメリカ合衆国・７８７３９・テキサス州・オースティン・ロスト カバーン コーヴ・３８０９
(72)発明者 バインドロス，キース・エム
アメリカ合衆国・９２６２０・カリフォルニア州・アーバイン・ドーブクレスト・７６
(72)発明者 アーヤ，サミール
アメリカ合衆国・７８７３９・テキサス州・オースティン・ウェールブリッジ レーン・６７２２
(72)発明者 ベアーズリー，ジョン・マーク
アメリカ合衆国・９４０２５・カリフォルニア州・メンロ パーク・シャーウッド ウェイ・４３１
(72)発明者 ギブソン，デビッド・エー
アメリカ合衆国・７８７２９・テキサス州・オースティン・マーブル フォールズ コーヴ・１３１１７

Ｆターム(参考) 5B033 BE05
5B045 GG12 GG15