(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6236093号
(P6236093)
(45)発行日　平成29年11月22日(2017.11.22)　(24)登録日　平成29年11月2日(2017.11.2)

| (51)Int.Cl. | | | F I | | |
|---|---|---|---|---|---|
| G06F | 9/38 | (2006.01) | G06F | 9/38 | 370A |
| G06F | 9/32 | (2006.01) | G06F | 9/32 | 320A |
| G06F | 9/30 | (2006.01) | G06F | 9/30 | 350F |
| G06F | 9/45 | (2006.01) | G06F | 9/44 | 322G |

請求項の数 20　(全 20 頁)

(21)出願番号　特願2015-555420 (P2015-555420)
(86)(22)出願日　平成26年1月28日(2014.1.28)
(65)公表番号　特表2016-504699 (P2016-504699A)
(43)公表日　平成28年2月12日(2016.2.12)
(86)国際出願番号　PCT/US2014/013455
(87)国際公開番号　WO2014/120690
(87)国際公開日　平成26年8月7日(2014.8.7)
審査請求日　平成28年2月3日(2016.2.3)
(31)優先権主張番号　13/753,098
(32)優先日　平成25年1月29日(2013.1.29)
(33)優先権主張国　米国(US)

早期審査対象出願

(73)特許権者　591016172
アドバンスト・マイクロ・ディバイシズ・インコーポレイテッド
ＡＤＶＡＮＣＥＤ　ＭＩＣＲＯ　ＤＥＶＩＣＥＳ　ＩＮＣＯＲＰＯＲＡＴＥＤ
アメリカ合衆国、９４０８８－３４５３　カリフォルニア州、サニィベイル、ピィ・オゥ・ボックス・３４５３、ワン・エイ・エム・ディ・プレイス、メイル・ストップ・６８（番地なし）
(74)代理人　100108833
弁理士　早川　裕司
(74)代理人　100111615
弁理士　佐野　良太

最終頁に続く

(54)【発明の名称】並列パイプラインにおいてブランチを分岐するためのハードウェアおよびソフトウェアソリューション

(57)【特許請求の範囲】
【請求項１】
コンピュータシステムの少なくとも１つのプロセッサによる実行のために構成された少なくとも１つのプログラムを格納するコンピュータ可読記憶媒体であって、
前記少なくとも１つのプログラムは、前記プロセッサによって実行されると、
複数のプログラム命令を解析することと、
前記複数のプログラム命令内でループおよび対応する基本ブロックを識別することと、
前記複数のプログラム命令内の識別されたループ内の所与の分岐点を識別したことに応じて、前記識別されたループ内の複数の命令を複数の超大命令語（ＶＬＩＷ）に配置することであって、少なくとも１つのＶＬＩＷは、前記所与の分岐点と対応する収束点との間の異なる基本ブロックから混ぜ合わされた命令を含む、ことと、
を前記プロセッサに実行させる命令を含む、
コンピュータ可読記憶媒体。
【請求項２】
前記少なくとも１つのプログラムは、前記識別されたループ内の前記所与の分岐点を検出したことに応じて、第１の命令を前記複数のプログラム命令に追加すること、を前記プロセッサに実行させるための命令をさらに含み、
前記第１の命令は、所与のＶＬＩＷ内の命令を、単一命令複数データ（ＳＩＭＤ）マイクロアーキテクチャを含むターゲットプロセッサ内の複数の並列実行レーンに実行時に割り当てることを前記ターゲットプロセッサに実行させるためのものである、請求項１に記

載のコンピュータ可読記憶媒体。
【請求項３】
前記第１の命令は、実行時に、前記所与の分岐点において所与のレーンに対して検出された分岐方向に少なくとも部分的に基づいて、前記複数の命令のうち何れかの命令を前記所与のレーンに割り当てることを前記ターゲットプロセッサに実行させる、請求項２に記載のコンピュータ可読記憶媒体。
【請求項４】
前記少なくとも１つのプログラムの前記命令は、次のプログラムカウンタ（ＰＣ）に対応するＶＬＩＷの格納されたサイズを前記ターゲットプロセッサに更新させるように構成された第２の命令を追加することを前記プロセッサに実行させる、請求項２に記載のコンピュータ可読記憶媒体。
【請求項５】
前記少なくとも１つのプログラムの前記命令は、前記ターゲットプロセッサに対して、前記検出された所与の分岐点と前記対応する収束点との間で第１の命令シーケンスが第２の命令シーケンスよりも小さいことに応じて、ｎｏｐを、ＶＬＩＷ内の前記第２の命令シーケンスに対応する命令とグループ化させるように構成された第３の命令を追加することを前記プロセッサに実行させる、請求項４に記載のコンピュータ可読記憶媒体。
【請求項６】
実行時に、前記所与のＶＬＩＷ内の命令を前記複数の並列実行レーンに割り当てるために、前記第１の命令は、前記複数の実行レーンのうち対応するレーンと関連付けられているベクトルレジスタ内の特定のビット範囲にオフセットを書き込むように前記ターゲットプロセッサに実行させ、前記オフセットは、実行のために関連付けられたレーンに対してフェッチされた所与のＶＬＩＷ内の所与の命令を識別する、請求項４に記載のコンピュータ可読記憶媒体。
【請求項７】
前記少なくとも１つのプログラムの前記命令は、
所与の命令シーケンスが、前記識別されたループの終わりに達していることに応じて、
実行時に、前記所与の命令シーケンスが、前記識別されたループの開始に分岐して戻るようにスケジューリングされているという判断に応じて、前記ベクトルレジスタ内の対応するビット範囲内にスリープ状態を書き込むことと、
実行時に、前記所与の命令シーケンスが、前記識別されたループの外部に分岐するようにスケジューリングされているという判断に応じて、前記ベクトルレジスタ内の前記対応するビット範囲内に終了状態を書き込むことと、
を前記ターゲットプロセッサに実行させるように構成された第４の命令を追加することを前記プロセッサに実行させる、
請求項６に記載のコンピュータ可読記憶媒体。
【請求項８】
前記少なくとも１つのプログラムの前記命令は、
前記複数のレーンのうち、前記スリープ状態または前記終了状態にある前記所与の命令シーケンスに対応するレーンを検出したことに応じて、
前記所与の命令シーケンスの実行を停止することと、
少なくとも次のプログラムカウンタ（ＰＣ）および前記所与の命令シーケンスに対応する識別子（ＩＤ）を格納することと、
を前記ターゲットプロセッサに実行させるように構成された第５の命令を追加することを前記プロセッサに実行させる、
請求項７に記載のコンピュータ可読記憶媒体。
【請求項９】
前記少なくとも１つのプログラムの前記命令は、
レーンが前記スリープ状態にあるか前記終了状態にあることに応じて、前記レーン内の命令の実行を、個別に格納された次のＰＣで再開することを前記ターゲットプロセッサに

実行させるように構成された第６の命令を追加することを前記プロセッサに実行させる、請求項８に記載のコンピュータ可読記憶媒体。

【請求項１０】

前記少なくとも１つのプログラムの前記命令は、

スリープ状態にある命令のみの実行を、個別に格納された次のＰＣで再開することを前記ターゲットプロセッサに実行させるように構成された第７の命令を追加することを前記プロセッサに実行させる、請求項８に記載のコンピュータ可読記憶媒体。

【請求項１１】

単一命令複数データ（ＳＩＭＤ）マイクロアーキテクチャ内の複数の並列実行レーンと、

超大命令語（ＶＬＩＷ）のサイズを格納するように構成されたサイズレジスタと、

プログラムコードを実行するように構成された制御ロジックと、

を備え、

前記プログラムコードは、

複数のプログラム命令内でループおよび対応する基本ブロックを識別することと、

前記複数のプログラム命令内の識別されたループ内の所与の分岐点を識別したことに応じて、前記識別されたループ内の複数の命令を複数の超大命令語（ＶＬＩＷ）に配置することであって、少なくとも１つのＶＬＩＷは、前記所与の分岐点と対応する収束点との間の異なる基本ブロックから混ぜ合わされた命令を含む、ことと、により生成される、複数のＶＬＩＷを含むものである、

プロセッサ。

【請求項１２】

前記複数の並列実行レーンのうち対応するレーンと関連付けられている特定のビット範囲内にオフセットを格納するように構成されたベクトルレジスタをさらに備え、

前記オフセットは、実行のために関連付けられたレーンに対してフェッチされた所与のＶＬＩＷ内の所与の命令を識別する、請求項１１に記載のプロセッサ。

【請求項１３】

前記ベクトルレジスタの前記ビット範囲に格納される前記オフセットの有効な値としてありうる、互いに異なる値の数は、前記サイズレジスタに格納されたサイズと等しい、請求項１２に記載のプロセッサ。

【請求項１４】

オフセットが、前記ＶＬＩＷ内の複数の資源に依存しない命令に対応しているとの検出に応じて、前記複数の並列実行レーンのうち前記オフセットに関連付けられたレーンは、関連付けられたレーン内の前記複数の命令を同時に実行するようにさらに構成されている、請求項１２に記載のプロセッサ。

【請求項１５】

所与の命令シーケンスが前記識別されたループの終わりに達しており、且つ、前記所与の命令シーケンス、及び、対応するレーンが、前記識別されたループの開始に分岐して戻るようにスケジューリングされていることを示すスリープ状態、または、前記識別されたループの外側に分岐するようにスケジューリングされていることを示す終了状態のいずれかであることに応じて、前記制御ロジックは、

前記所与の命令シーケンスの実行を停止することと、

少なくとも次のプログラムカウンタ（ＰＣ）および前記所与の命令シーケンスに対応する前記レーンの識別子（ＩＤ）を格納することと、

を行うようにさらに構成されている、請求項１２に記載のプロセッサ。

【請求項１６】

前記複数の並列実行レーンの各レーンが、前記スリープ状態または前記終了状態にあることに応じて、前記制御ロジックは、個別に格納された次のＰＣに分岐することにより、レーンごとの実行を再開するようにさらに構成されている、請求項１５に記載のプロセッサ。

【請求項１７】
前記複数の並列実行レーンの各レーンが、前記スリープ状態または前記終了状態にあることに応じて、前記制御ロジックは、個別に格納された次のＰＣに分岐することにより、スリープ状態にあるレーンのみの実行を再開するようにさらに構成されている、請求項１５に記載のプロセッサ。
【請求項１８】
複数のプログラム命令内でループおよび対応する基本ブロックを識別することと、
識別されたループ内の所与の分岐点に応じて、前記識別されたループ内の複数の命令を複数の超大命令語（ＶＬＩＷ）に配置することとであって、少なくとも１つのＶＬＩＷは、前記所与の分岐点と対応する収束点との間の異なる基本ブロックから混ぜ合わされた命令を含む、ことと、
を含む、方法。
【請求項１９】
前記識別されたループ内の前記所与の分岐点に応じて、前記所与の分岐点において所与のレーンの実行時に検出した分岐方向に基づいて、実行時に、所与のＶＬＩＷ内の命令を、単一命令複数データ（ＳＩＭＤ）マイクロアーキテクチャを含むターゲットプロセッサ内の複数の並列実行レーンに割り当てることをさらに含む、請求項１８に記載の方法。
【請求項２０】
実行時に、前記所与のＶＬＩＷ内の命令を前記複数の並列実行レーンに割り当てるために、前記複数の並列実行レーンのうち対応するレーンと関連付けられている指示を格納することをさらに含み、
前記指示は、実行するために関連付けられたレーンに対して前記所与のＶＬＩＷ内の所与の命令を識別する、請求項１９に記載の方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本開示は、コンピューティングシステムに関し、より詳細には、プロセッサ内のハードウェア並列実行レーンで命令を効率的に処理することに関する。
【背景技術】
【０００２】
コンピュータシステムのスループットを向上させるために、タスクの並列化が用いられている。この目的のために、コンパイラは、並列化されたタスクをプログラムコードから抽出して、システムハードウェア上で並行して実行し得る。ハードウェア上での並列実行を向上するために、プロセッサは、複数の並列実行レーン（例えば、単一命令複数語（ＳＩＭＤ）マイクロアーキテクチャ内など）を含み得る。このタイプのマイクロアーキテクチャは、特定のソフトウェアアプリケーションに対して、単一レーンのマイクロアーキテクチャまたは汎用マイクロアーキテクチャよりも高い命令スループットを提供し得る。ＳＩＭＤマイクロアーキテクチャから恩恵を受けるタスクのいくつかの例は、ビデオグラフィックスレンダリング、暗号化、およびガーベジコレクションを含む。
【０００３】
多くの場合、特定のソフトウェアアプリケーションは、各作業項目の実行や並列関数呼出しがその内部のデータ依存となるようなデータ並列処理を有する。例えば、第１の作業項目が第２の作業項目から独立したデータである場合には、第１および第２の作業項目の各々は、ＳＩＭＤマイクロアーキテクチャ内の別個の並列実行レーン上に同時にスケジューリングされている。しかし、第１および第２の作業項目の各々で実行されるある量の命令は、データ依存の場合がある。分岐命令として実装された条件テストは、第１の作業項目に対してパスし得るが、各作業項目に対するデータに依存する第２の作業項目に対しては不合格になり得る。
【０００４】
第２の作業項目が、実行を停止して、第１の作業項目が進行中の実行を継続するのを待

機するので、並列実行の効率が低下し得る。パスしたテストに起因して２～３の作業項目だけが実行を継続し、他方、不合格になったテストのために、ほとんどの作業項目がアイドルである場合には、効率の悪さが増大する。

【発明の概要】

【課題を解決するための手段】

【０００５】

プロセッサ内のハードウェア並列実行レーンで命令を効率的に処理するためのシステムおよび方法を検討する。様々な実施形態では、バックエンドコンパイラは、ソフトウェアアプリケーションのプログラム命令を検査して、ターゲットプロセッサ上で命令を効率的に処理するように、命令を配置してコードを生成する。ターゲットプロセッサは、単一命令複数データ（ＳＩＭＤ）マイクロアーキテクチャ内に複数の並列実行レーンを含んでもよい。コンパイラは、ループと、対応する基本ブロックとを識別してもよい。ループ内の分岐点は、分岐命令を含んでもよい。例えば、ｉｆ－ｅｌｓｅｉｆ－ｅｌｓｅ構成、ｉｆ－ｅｌｓｅ構成、ｃａｓｅ構成などが、識別されたループ内のプログラム命令で使用されてもよい。分岐点と対応する収束点との間での、翻訳されコンパイルされたプログラム命令の実行中に、複数のトレースパスがトラバースされてもよい。

【０００６】

コンパイル中、識別されたループ内の所与の分岐点に応じて、コンパイラは、識別されたループ内の命令を、１つ以上の超大命令語（ＶＬＩＷ：ｖｅｒｙ　ｌａｒｇｅ　ｉｎｓｔｒｕｃｔｉｏｎ　ｗｏｒｄ）に配置してもよい。少なくとも１つのＶＬＩＷは、所与の分岐点と対応する収束点との間の異なる基本ブロックから混ぜ合わされた命令を含んでもよい。例えば、４つの命令を有する基本ブロックＡと、６つの命令を有する基本ブロックＢとが、所与の分岐点と対応する収束点との間に存在する場合には、コンパイラは、命令を６つのＶＬＩＷ内に配置してもよい。最初の４つのＶＬＩＷは、基本ブロックＡおよび基本ブロックＢの各々から１つの命令を含んでもよい。第１のＶＬＩＷは、基本ブロックＡおよび基本ブロックＢの各々からの第１の命令を含んでもよい。第２のＶＬＩＷは、基本ブロックＡおよび基本ブロックＢの各々からの第２の命令を含んでもよく、以下同様である。最後の２つのＶＬＩＷは、ｎｏｐ（ノーオペレーション）とともにグループ化された基本ブロックＢからの命令を含んでもよい。コンパイラは、各ＶＬＩＷをポイントするプログラムカウンタ（ＰＣ）値を追跡してもよい。

【０００７】

コンパイラは、翻訳されコンパイルされた命令で、挿入するためのコードを生成してもよい。挿入されたコードは、実行される際に、所与のＶＬＩＷ内の命令を、ターゲットプロセッサ内の複数の並列実行レーンに実行時に割り当ててもよい。所与のレーンに対する割り当ては、実行時に、所与の分岐点で、所与のレーンに対して見つかった分岐方向に基づいてもよい。前述の例を続けると、ＶＬＩＷが、基本ブロックＡおよび基本ブロックＢから生成された第２のＶＬＩＷであって、所与のレーンに対する分岐命令が選択された場合には、所与のレーンは、第２のＶＬＩＷ内の基本ブロックＡ内の第２の命令が割り当てられてもよい。分岐命令が選択されない場合には、所与のレーンは、第２のＶＬＩＷ内の基本ブロックＢ内の第２の命令が割り当てられてもよい。様々な実施形態では、ＶＬＩＷは可変長である。挿入されたコードは、実行される際に、次のＰＣに対応するＶＬＩＷのサイズを更新してもよい。

【０００８】

いくつかの実施形態では、プロセッサは、単一命令複数データ（ＳＩＭＤ）マイクロアーキテクチャ内に複数の並列実行レーンを含む。プロセッサは、可変長ＶＬＩＷのサイズを格納するためのサイズレジスタを含んでもよい。プロセッサ内の制御ロジックは、格納したサイズに等しい所与のＶＬＩＷ内のいくつかの命令を、それぞれのサイクル内でフェッチして復号してもよい。複数の実行レーンは、所与のＶＬＩＷ内でいくつかの命令を同時に実行してもよい。さらに、プロセッサは、複数の実行レーンのうち対応するレーンと関連付けられたビット範囲を有するベクトルレジスタを含んでもよい。ビット範囲はオフ

セットを格納してもよい。所与のオフセットは、実行するために関連付けられたレーンに対してフェッチされたＶＬＩＷ内の所与の命令を識別してもよい。

【０００９】

これらの実施形態および他の実施形態は、以下の記載および図面を参照することによって、さらに理解されるであろう。

【図面の簡単な説明】

【００１０】

【図１】単一命令複数データ（ＳＩＭＤ）パイプライン実行フローの一実施形態の一般化されたブロック図である。

【図２】制御フローグラフの一実施形態の一般化されたブロック図である。

【図３】制御フローグラフに対する実行順序の一実施形態の一般化されたブロック図である。

【図４】プロセッサに対するＳＩＭＤマイクロアーキテクチャの論理レイアウトの一実施形態の一般化されたブロック図である。

【図５】オブジェクトコード配置の一実施形態の一般化されたブロック図である。

【図６】コンパイラ技術を用いて、プロセッサ内で複数の作業項目の並列実行を最適化するための方法の一実施形態の一般化されたフロー図である。

【図７】ハードウェア技術を用いて、プロセッサ内で複数の作業項目の並列実行を最適化するための方法の一実施形態の一般化されたフロー図である。

【図８】オブジェクトコード配置の別の実施形態の一般化されたブロック図である。

【発明を実施するための形態】

【００１１】

実施形態は、様々な修正および代替形式を受け入れる余地があるが、特定の実施形態が例として図面に示され、本明細書で詳細に説明されている。しかし、図面およびそれに関する詳細な説明は、実施形態が、開示された特定の形式に限定されることを意図しておらず、それとは逆に、添付の請求項によって定義されるように、実施形態の趣旨および範囲に含まれる全ての修正、均等物および代替手段を包含することが理解されるべきである。

【００１２】

以下の記載では、実施形態の完全な理解を提供するために、多数の具体的な詳細が記載されている。しかし、当業者は、これらの具体的な詳細なしに、実施形態が実施され得ることを理解するはずである。いくつかの場合には、実施形態を曖昧にするのを避けるために、周知の回路、構造および技術が詳細に示されていない。

【００１３】

図１を参照すると、単一命令複数データ（ＳＩＭＤ）パイプライン実行フロー１００の一実施形態を示す一般化されたブロック図が示されている。命令１０２～１０８はフェッチされ、関連するデータと共にＳＩＭＤパイプラインに送信されてもよい。並列で垂直な実行レーン内の複数の計算ユニットが示されている。いくつかの計算ユニットは、アクティブな計算ユニット１１０である。他の計算ユニットは、所与のパイプステージ中に無効にされていることに起因する、非アクティブな計算ユニット１１２である。制御ロジックおよび記憶素子（例えば、パイプラインレジスタなど）は、説明を容易にするために示されていない。

【００１４】

ハードウェア計算ユニットは、関連付けられたデータを使用して、所与の作業項目の所与の命令の実行を行うハードウェアを含む。このハードウェアは、加算、乗算、ゼロ検出、ビット単位シフト、除算、ビデオグラフィックスおよびマルチメディア命令、または、プロセッサ設計の当業者に周知の他の操作を実行するように構成された演算論理装置を含んでもよい。ＳＩＭＤパイプライン内に並列実行レーンを有するプロセッサの例は、グラフィック処理装置（ＧＰＵ）、デジタル信号処理（ＤＳＰ）などを含む。一実施形態では、ＳＩＭＤパイプラインは、ビデオカード上に配置されてもよい。別の実施形態では、ＳＩＭＤパイプラインは、マザーボード上に統合されてもよい。

【００１５】
ＳＩＭＤパイプラインは、ゲーム、エンタテインメント、科学および医療分野で使用される多種多様なデータ並列アプリケーションに対する計算性能を向上させ得る。かかるアプリケーションは、一般に、多数のオブジェクトについて同じプログラムを実行することを伴う。各オブジェクトは、他のオブジェクトと関係なく処理されるが、同じ順序の操作が使用されるので、ＳＩＭＤマイクロアーキテクチャは、相当な性能強化を提供する。ＧＰＵは、非グラフィック計算用にも想定されている。
【００１６】
ソフトウェアアプリケーションは、関数呼出しまたは計算カーネルの集合、および、内部関数の集合を含んでもよい。ソフトウェアプログラマは、関数呼出しを定義してもよく、他方、内部関数は所与のライブラリ内で定義されてもよい。例えば、ソフトウェアアプリケーションは、例えば画像ファイルなどの２次元（２Ｄ）配列のデータのデータ処理を実行し得る。ソフトウェアアプリケーションは、ソフトウェアプログラマによって開発されたアルゴリズムを、２Ｄ画像の画素ごとまたは２次元行列の要素ごとに、実行し得る。所与の関数呼出しは、インデックス空間を介して呼び出されてもよい。インデックス空間は、次元空間とも呼ばれ得る。データ並列ソフトウェアアプリケーションに対して、Ｎ次元計算領域は、１、２もしくは３次元空間、または、インデックス空間を定義してもよい。一例は、２Ｄ画像内の画素である。
【００１７】
関数呼出しは、データの１つ以上のレコードと照合されて、１つ以上の計算の作業項目を生成してもよい。従って、２つ以上の作業項目は、単一の関数呼出しの同じ命令を利用し得るが、データの異なるレコードについて動作し得る。関数呼出しは、フォーク（ｆｏｒｋ）を生成する制御フロー転送命令を含んでもよく、他方、コンピュータプログラム内のフォークは、通常、共通の定義によってソフトウェアスレッドを生成する。インデックス空間内の所与の時点における関数呼出しの所与のインスタンスが、「作業項目」と呼ばれてもよい。作業項目は、作業ユニットとも呼ばれてよい。前述の例を続けると、作業項目は、２Ｄ画像の所与の画素（所与のインデックス）に対応するデータのレコードについて、関数呼出し内の１つ以上の命令で動作してもよい。通常、作業項目は、関連付けられた一意の識別子（ＩＤ）を有する。
【００１８】
インデックス空間は、十分なハードウェアサポートがある場合に並行して実行する作業項目の総数を定義してもよい。例えば、インデックス空間は、２８０の数の作業項目を定義してもよいが、ＧＰＵは、いつでも６４の作業項目の同時実行をサポートし得る。作業項目の総数は、グローバルな作業サイズを定義し得る。作業項目は、さらに作業グループにグループ化され得る。各作業グループは、一意の識別子（ＩＤ）を有してもよい。所与の作業グループ内の作業項目は、相互に通信して、実行を同期させ、メモリアクセスを調整することが可能であってよい。いくつかの作業項目は、ＳＩＭＤ方式でＧＰＵ上での同時実行のためにウェーブフロント（ｗａｖｅ　ｆｒｏｎｔ）にクラスタ化されてもよい。２８０の総作業項目に対する前述の例に関して、ウェーブフロントは６４の作業項目を含んでもよい。
【００１９】
命令１０２～１０８は、フェッチされて、関連付けられたデータと共にＳＩＭＤパイプラインに入ってもよい。命令１０４は、例えば条件分岐などの制御フロー転送命令であってよい。命令１０６は、条件が真の場合に実行されるパス内の第１の命令であってよい。命令１０８は、条件が偽の場合に実行されるパス内の第１の命令であってよい。例えば、分岐命令１０４は、高水準言語プログラムにおけるＩＦ文と関連付けられてもよい。命令１０６は、高水準言語プログラムにおけるＴＨＥＮ文と関連付けられてもよい。命令１０８は、高水準言語プログラムにおけるＥＬＳＥ文と関連付けられてもよい。
【００２０】
所与の行内の各計算ユニットは、同じ計算ユニットであってよい。これらの計算ユニッ

トの各々は、同じ命令であるが、異なる作業項目と関連付けられた異なるデータについて動作してもよい。図に示すように、いくつかの作業項目は、条件分岐命令１０４によって提供されたテストをパスし、他の作業項目はテストに不合格になる。ＳＩＭＤパイプライン内の制御ロジックは、利用可能なパスの各々を実行して、現在のパスを選択しなかった作業項目に対応する実行ユニット（例えば、計算ユニットなど）を選択的に無効にしてもよい。例えば、Ｉｆ－Ｔｈｅｎ－Ｅｌｓｅ構成文の実行中、ＳＩＭＤアーキテクチャの各列内には、「Ｔｈｅｎ」（パスＡ）および「Ｅｌｓｅ」（パスＢ）のパスを実行するように構成された実行ユニットがある。

【００２１】

第１および第２の作業項目が実行を停止して、第３の作業項目が進行中の実行を継続するのを待機すると、並列実行の効率が低下し得る。従って、分岐命令１０４の実行後、所与の行における全ての計算ユニットがアクティブな計算ユニット１１０というわけではない。図に示すように、１つ以上の計算ユニットは、実行に関して無効にされた非アクティブな計算ユニット１１２である。多数の計算ユニットが所与のパイプステージ中に非アクティブである場合には、ＳＩＭＤコアの効率およびスループットが低下する。一実施形態では、「Ｅｌｓｅ」パスは、関数呼出しに対するリターン（ｒｅｔｕｒｎ）である。関数呼出しの実行が終了して、対応する作業項目がアイドルになる。しかし、ＳＩＭＤコア内の隣接する作業項目は、実行を継続してもよい。

【００２２】

ここで図２を参照すると、制御フローグラフ２００の一実施形態を示す一般化されたブロック図が示されている。一般的に言えば、制御フローグラフは、コンパイラ最適化器および静的解析ツールによって使用され得る。制御フローグラフ２００は、プログラムまたはプログラムの一部が、その実行中にトラバースされ得る全てのパスを表し得る。制御フローグラフでは、グラフ内の各ノードは、基本ブロックを表している。ほとんどの表現は、制御が制御フローグラフに入るための入口ブロックと、制御が制御フローグラフを出るための出口ブロックと、を含む。

【００２３】

コンパイル中、ソフトウェアアプリケーションは、基本ブロック０（ＢＢ　０）から基本ブロック７（ＢＢ　７）まで番号付けされた８つの基本ブロック（ＢＢ）を有する制御フローグラフ２００を提供し得る。８つの基本ブロックが示されているが、他の例では、別の数の基本ブロックが使用され得る。制御フローグラフ２００において、基本ブロック１が入口ブロックであり、基本ブロック６が出口ブロックである。基本ブロック０～７の各々は、１つの入口点と１つの出口点とを有する命令のストレートラインシーケンスである。制御フローグラフ２００は、ループを表してもよい。ループの内部では、制御フローグラフ２００は、基本ブロック１～４を有するＩＦ－ＴＨＥＮ－ＥＬＳＥ構成と、基本ブロック４～６を有するＩＦ構成と、を表してもよい。

【００２４】

ここで図３を参照すると、制御フローグラフに対する実行順序３００の一実施形態を示す一般化されたブロック図が示されている。実行順序３１０は、既に示した制御フローグラフ２００がＳＩＭＤパイプラインに割り当てられた場合の典型的な実行順序を表している。単一のループの繰返しに対する実行時間は、ループ内の各基本ブロック（例えば、ＢＢ　１～ＢＢ　６など）の実行時間の合計である。しかし、所与の作業項目および対応するハードウェア実行レーンに対して、ＢＢ　２およびＢＢ　３のうち１つのみが実行される。同様に、ＢＢ　５が所与の作業項目に対してスキップされ得る。特定の基本ブロックが所与の作業項目に対して実行されない可能性があるが、関連付けられた実行時間は、ループの繰返しに対する実行時間の総合計に寄与する。

【００２５】

実行順序３２０は、既に示した制御フローグラフ２００が、修正されたＳＩＭＤパイプラインに割り当てられた場合の代替の実行順序を表している。単一のループの繰返しに対する実行時間は、ループ内の各基本ブロックの実行時間の合計ではなく、単一のループの

繰返し内で実際に実行された基本ブロックの実行時間の合計である。実行順序３２０は、制御フローグラフ２００の実行を変換する。コンパイラは、ソースコードのオブジェクトコードへのコンパイル中に、この変換を実行し得る。

【００２６】

いくつかの実施形態では、コンパイラは、各基本ブロックの終わりにコードを生成し、そのコードは、実行される際に、次に実行する基本ブロックを識別する。生成コードは、基本ブロックの終わりに挿入され得る。あるいは、制御を次の基本ブロックに転送する前に、制御フローを追加のコードに転送するために、分岐命令が基本ブロックの終わりに挿入され得る。追加のコードは、中央基本ブロックを示すＢＢ　Ｃによって表される。実行時、分岐する基本ブロックの各々（例えば、ＢＢ　１、ＢＢ　４、ＢＢ　６など）は、制御の転送先となる次の基本ブロックを識別する。識別は、分岐解決に基づくものであり、分岐解決は、データの特定レコードおよび分岐命令にさらに基づいている。実行時、ＢＢ　Ｃは、ターゲットの基本ブロックのアドレスを各作業項目から受信して、それぞれのターゲットの基本ブロックを実行するためのスレッドレジスタをセットアップする。ＳＩＭＤパイプライン内で実行されている異なる作業項目にわたって、所与の作業項目は、分岐、ジャンプおよびケース文などのように、制御フロー転送命令に対する単一のターゲットを有する。

【００２７】

実行順序３２０では、ＢＢ　４の完了時に、第１の作業項目は、ＢＢ　５に分岐して、対応するアドレスをＢＢ　Ｃに渡し得る。ＢＢ　４の完了時に、第２の作業項目は、ＢＢ　６に分岐して、対応するアドレスをＢＢ　Ｃに渡し得る。コンパイラは、同時に実行される各ターゲット基本ブロックからの命令を含む、動的超大命令語（ＤＶＬＩＷ）を生成し得る。実行される際に、コンパイラによって生成されたコードは、次のプログラムカウンタ（ＰＣ）値に応じて実行する次のＤＶＬＩＷのサイズを更新し得る。加えて、実行される際に、生成されたコードは、所与の並列実行レーン内で実行されている所与の作業項目と、実行する次のＤＶＬＩＷ内の命令へのポインタとの間のマッピングを更新し得る。ポインタは、次のＤＶＬＩＷ内の命令のうち関連付けられた命令であって、フェッチされる命令を識別しているオフセットであってよい。

【００２８】

図４を参照すると、プロセッサに対するＳＩＭＤマイクロアーキテクチャの論理レイアウト４００の一実施形態を示す一般化されたブロック図が示されている。プロセッサは、データおよび命令を格納するためのダイナミックランダムアクセスメモリ（ＤＲＡＭ）４５０を有する。いくつかの実施形態では、所与のレベルのキャッシュメモリサブシステムが、ＤＲＡＭに加えて使用される。図に示すように、プロセッサは、計算ユニットの行ごとに、制御ロジック４２０と一緒にグループ化された比較的小規模のキャッシュメモリサブシステム４３０を有してもよい。説明を簡略にするために、パイプラインレジスタなどの記憶素子は示されていないが、プロセッサ内のデータフローは、パイプライン化されてもよい。所与のパイプラインのステージでは、このステージ内の関連付けられた命令が、既に不合格になったテスト（例えば、選ばれなかった分岐など）に基づいて実行されない場合には、計算ユニットは使用されない可能性がある。

【００２９】

ＳＩＭＤパイプラインは、レーンＡ～Ｆを有する作業項目４６０を含む。レーンＡ～Ｆの各々は、計算ユニットを含む垂直で並列なハードウェアレーンの各々に対応し得る。さらに、パイプラインは、ベクトルレジスタ４６２を含んでもよい。ベクトルレジスタ４６２は、並列実行レーンの各々に対して、エントリ、フィールドまたはビット範囲を含んでもよい。各エントリは、それぞれの作業項目上で実行している所与のトレースを識別するための第１のビット数と、特別コードをサポートするための第２のビット数と、を含むビット総数を含んでもよい。特別コードは、待ち状態またはスリープ状態、ループ終了状態、ループを終了するため以外の実行を停止するためのバリア識別子、イベント識別子などを識別してもよい。特別コードが所与のエントリ内に格納されない場合には、格納された

値は、実行するために関連付けられたレーンについてＤＶＬＩＷ内の命令の各々を識別してもよい。

【００３０】

プログラムカウンタ（ＰＣ）レジスタ４６６は、ｉ－キャッシュなどのメモリからフェッチするために、次のＤＶＬＩＷをポイントしているポインタ値またはアドレスを格納してもよい。プロセッサは、ＤＶＬＩＷのサイズ、すなわち長さを格納するサイズレジスタ４６８をさらに含んでもよい。いくつかの実施形態では、サイズは、可変長ＤＶＬＩＷ内の命令数を表す整数であってよい。

【００３１】

ＤＶＬＩＷ　４６４内の命令Ｉｎｓｔｒ　Ａ～Ｉｎｓｔｒ　Ｇの各々は、制御フローグラフ内の実行トレースを表している。コンパイラは、ＤＶＬＩＷを、ｉ－キャッシュなどのメモリ内に配置してもよい。一例では、作業項目４６０内のレーンＢは、ＳＩＭＤパイプライン内の左から２番目の垂直な実行レーンに対応し得る。ベクトルレジスタ４６２内に格納されたオフセットＢは、レーンＢと関連付けられて、ＤＶＬＩＷ　４６４内の最初の命令をポイントし得るが、その命令は、Ｉｎｓｔｒ　Ａである。従って、レーンＢは、Ｉｎｓｔｒ　Ａを受信して処理し得る。同様に、作業項目４６０内のレーンＡは、ＳＩＭＤパイプライン内の最も左の垂直な実行レーンに対応し得る。ベクトルレジスタ４６２内に格納されたオフセットＡは、レーンＡと関連付けられて、ＤＶＬＩＷ　４６４内の最後の命令（Ｉｎｓｔｒ　Ｇ）をポイントし得る。従って、レーンＡは、Ｉｎｓｔｒ　Ｇを受信して処理し得る。

【００３２】

図示していないが、命令キャッシュ（ｉ－キャッシュ）は、ＤＶＬＩＷをサポートするための複数の実施態様のうち１つを含んでもよい。ｉ－キャッシュは、ＤＶＬＩＷに対応する所与の単一のＰＣに対する１つ以上の命令をフェッチするための複数の小型のキャッシュを含んでもよい。同じＰＣは、ＤＶＬＩＷのサイズに応じて、小型のキャッシュのうち１つ以上のキャッシュ内の有効な命令をインデックスしてもよい。ｉ－キャッシュは、ＰＣレジスタ４６６内に格納されたポインタまたはアドレス値に加えて、サイズレジスタ４６８内に格納されたサイズを受信し得る。代替として、ｉ－キャッシュは、同じ有効なキャッシュラインまたはキャッシュセット内の１つ以上の命令にアクセスするための複数のデータポートを有してもよい。この場合もやはり、フェッチする有効な命令の数は、サイズレジスタ４６８からの受信したサイズに等しい可能性がある。

【００３３】

ここで図５を参照すると、オブジェクトコード配置５００を示す一実施形態の一般化されたブロック図が示されている。コード配置５００は、図２および図３にそれぞれ示す、制御フローグラフ２００および付随の実行順序３２０に対してコンパイラにより生成され得るオブジェクトコードレイアウトの一実施形態を示している。基本ブロックコード５０４は、各基本ブロックに対するコードを表す。例えば、基本ブロック０～３，７に対するコードの１つのコピーが、レイアウト５００内に配置されて示されている。基本ブロック４～５に対するコードの２つのコピーが、レイアウト５００内に配置され示されている。基本ブロック６に対するコードの４つのコピーが、レイアウト５００内に示されている。

【００３４】

コード５０２は、ループなどの領域のエントリに対して、コンパイラによって生成および挿入され得る。後に実行される場合、コード５０２は、次のＤＶＬＩＷのサイズを更新し、フェッチされたＤＶＬＩＷ内の命令と、ターゲットプロセッサ内の並列実行レーンとの間のマッピングを更新し得る。例えば、ターゲットプロセッサ内のサイズレジスタおよびベクトルレジスタは、それらに格納された内容を、実行されたコード５０２によって更新させてもよい。図に示すように、コード５０２は、基本ブロック０，１，７の開始時に挿入され得る。

【００３５】

コード５０６は、基本ブロック１，４，６などの分岐点に移行するために、コンパイラ

によって生成および挿入され得る。後に実行される場合、コード５０６は、ＤＶＬＩＷのサイズ変更、および、フェッチされたＤＶＬＩＷ内の命令とターゲットプロセッサ内の並列実行レーンとの間の対応するマッピング変更を判断し得る。従って、サイズおよびマッピングが、制御フローグラフ内の分岐点および収束点において更新される。コンパイラは、ＤＶＬＩＷのサイズが変わる点、および、マッピングが変わる点を識別する。第１のインデックスがトレース識別子（ＩＤ）を示し、第２のインデックスが基本ブロック（ＢＢ）ＩＤを示す表記法ＢＢＣ（０，１）を使用すると、コード５０６を挿入するための識別された点は、ＢＢＣ（０，１）、ＢＢＣ（０，４）、ＢＢＣ（０，６）、ＢＢＣ（１，４）、ＢＢＣ（１，６）、ＢＢＣ（２，６）およびＢＢＣ（３，６）に存在し得る。この例におけるトレースＩＤは、対応するオフセットと同じであってよい。

【００３６】

オブジェクトコード配置５００の開始時に、コード５０２は、初期化ステップを実行して、ＤＶＬＩＷサイズを１に設定してもよい。ベクトルレジスタの各エントリ内のオフセットは、例えば０のオフセットなどのように、ＢＢ　０内の同じ命令をポイントするように設定され得る。従って、ＰＣは、０または別の適切な開始アドレスに設定され得る。図に示すように、４つの可能な並列トレースが存在しているが、作業項目の数は独立であってよい。例えば、ＳＩＭＤパイプラインは、割り当てられた作業項目を処理するための、８、１６、６４または別の数の並列実行レーンを有してもよい。ＳＩＭＤパイプライン内の各作業項目は、ベクトルレジスタ内に格納された０のオフセットを有しており、同じ命令を実行し得る。各作業項目に対するこの同じ命令は、ＢＢ　０からの命令である。ＢＢ　０内の命令は、各作業項目によって１つずつ実行され、各命令フェッチの後にＰＣが増加する。

【００３７】

ＢＢ　０の実行が完了した後、ループ入口ブロックであるＢＢ　１が次に処理される。コード５０２は、ＢＢ　１の開始時に、ＤＶＬＩＷサイズを１として保持し、各作業項目に対するオフセットを０として保持する。ベクトルレジスタのエントリの各々のオフセットは、例えば０のオフセットなどのように、ＢＢ　１内の同じ命令をポイントするように設定され得る。ＰＣは、ＢＢ　０の完了時に増加された値のままであってよい。ＳＩＭＤパイプライン内の各作業項目は、ベクトルレジスタ内に格納された０のオフセットを有しており、同じ命令を実行することになる。作業項目ごとのこの同じ命令は、ＢＢ　１からの命令である。ＢＢ　１内の命令は各作業項目によって１つずつ実行され、各命令のフェッチ後にＰＣが増加する。

【００３８】

実行される際に、コード５０６は、ＢＢ　１の終わりにあるＢＢＣ（０，１）において、格納されたＤＶＬＩＷサイズを１から２に変更する。ここで、ＢＢ　３内の命令は、フェッチされたＤＶＬＩＷに追加される。さらに、実行される際に、コード５０６は、ＢＢＣ（０，１）にて、ＢＢ　３に分岐する作業項目に対するベクトルレジスタ内のエントリを、値１を格納するように設定する。ＢＢ　２に分岐する作業項目に対するベクトルレジスタ内のエントリは、０を継続して格納することにより、変更されないままである。０および１の値がこのように使用されるが、対応する指示およびマッピングを設定するために他の数値が使用されてもよい。この時点で、ＤＶＬＩＷは２つの命令を有し、これらは、２つの別個の基本ブロックＢＢ　２およびＢＢ　３から混ぜ合わされている。ＰＣが継続して増加されるので、フェッチされたＤＶＬＩＷは、ＢＢ　２の処理が完了するまで、これらの２つの基本ブロックからの混ぜ合わされた命令を含み続ける。コンパイラは、ＤＶＬＩＷ内の命令の並列実行をサポートするために、メモリ内でこのように混ぜ合わされるように命令を配置してもよい。

【００３９】

ＢＢ　２の完了時に、ＤＶＬＩＷサイズは２のままである。ベクトルレジスタ内に格納されたオフセットもそれらの値のままである。しかし、この時、オフセット０は、ＢＢ　２ではなく、ＢＢ　４内の命令に対応する。ＢＢＣ（０，４）の完了時に、コード５０６

は、トレース０内のＢＢ　４の終わりにおいて、３を格納するようにサイズレジスタを更新し、ＢＢ　６に分岐する作業項目に対するエントリを、２を格納するように更新する。単一のＰＣおよび格納されたサイズをｉ－キャッシュに送信した後に、長さ３のＤＶＬＩＷがｉ－キャッシュからフェッチされる。ＤＶＬＩＷは、ＢＢ　３またはＢＢ　４と、ＢＢ　５と、ＢＢ　６とから混ぜ合わされた命令を含む。ベクトルレジスタ内に関連する格納された０のオフセットを有する作業項目は、ＢＢ（０，５）からフェッチされた命令を得る。ベクトルレジスタ内に関連する格納された１のオフセットを有する作業項目は、どの程度までＰＣが増加されているかに応じて、ＢＢ（１，３）またはＢＢ（１，４）の何れかから結果を得る。ベクトルレジスタ内に関連する格納された２のオフセットを有する作業項目は、ＢＢ（２，６）からフェッチされた命令を得る。単一のＰＣおよび格納されたサイズが、フェッチするＤＶＬＩＷのタイプをｉ－キャッシュに対して示すように、コンパイラは、命令をこの方式でメモリ内に既に配置している。

【００４０】

ＢＢＣ（１，４）の完了時に、コード５０６は、トレース１内のＢＢ　４の終わりにおいて、４を格納するようにサイズレジスタを更新し、ＢＢ（３，６）に分岐する作業項目に対するエントリを、３を格納するように更新する。単一のＰＣおよび格納されたサイズをｉ－キャッシュに送信した後に、長さ４のＤＶＬＩＷがｉ－キャッシュからフェッチされる。ＤＶＬＩＷは、ＢＢ　６の第１のコピーと、ＢＢ　５の単一のコピーと、ＢＢ　６の第２のコピーと、ＢＢ　６の第３のコピーと、から混ぜ合わされた命令を含む。ベクトルレジスタ内に関連する格納された０のオフセットを有する作業項目は、ＢＢ（０，６）からフェッチされた命令を得る。ベクトルレジスタ内に関連する格納された１のオフセットを有する作業項目は、ＢＢ（１，５）から結果を得る。ベクトルレジスタ内に関連する格納された２のオフセットを有する作業項目は、ＢＢ（２，６）からフェッチされた命令を得る。ベクトルレジスタ内に関連する格納された３のオフセットを有する作業項目は、ＢＢ（３，６）からフェッチされた命令を得る。単一のＰＣおよび格納されたサイズが、フェッチするＤＶＬＩＷのタイプをｉ－キャッシュに対して示すように、コンパイラは、命令をこの方式でメモリ内に既に配置している。

【００４１】

ＢＢ（０，６）、ＢＢ（１，６）、ＢＢ（２，６）およびＢＢ（３，６）の各々に対して、制御フローは、ＢＢ　６の終わりにおいて、ループの別の繰り返しのためにＢＢ　１に戻ってもよいし、ループを終了してもよい。関連付けられた分岐命令および対応するレコード内のデータは、実行時に制御フローの方向を判断するであろう。いくつかの作業項目は、別の繰り返しを継続してもよく、他の作業項目は、ループを終了してもよい。特別コード状態は、ベクトルレジスタ内の対応するエントリ内に格納され、どのパスが選ばれるかを示してもよい。所与の作業項目が別のループの繰返しを継続し、複数の作業項目のうち少なくとも１つの他の作業項目が、関連付けられた基本ブロックに対するコードを処理しているという判断に応じて、スリープ状態符号化は、所与の作業項目に対するベクトルレジスタ内の関連付けられたエントリに格納され得る。

【００４２】

所与の作業項目がループを終了するという判断に応じて、終了状態符号化は、所与の作業項目に対するベクトルレジスタ内の関連付けられたエントリに格納され得る。スリープ状態符号化および終了状態符号化の各々は、ループ繰返し中に使用されるオフセットから一意であって、互いに一意である。いくつかの実施形態では、スリープ状態または終了状態にある所与の作業項目に対して、コード５０６は、所与の作業項目の実行を停止して、少なくとも次のプログラムカウンタ（ＰＣ）および作業項目識別子（ＩＤ）を、例えば高速読み出しのためのスタックメモリなどのメモリに格納する。

【００４３】

コード５０６は、ＢＢ　６の終わりにおいて、各作業項目の状態をチェックし得る。各作業項目がスリープ状態であるか、または各作業項目が終了状態であるという判断に応じて、プロセッサは、コード５０６を実行している間に、それぞれの格納された次のＰＣに

分岐することにより、各作業項目に対する実行を再開し得る。各作業項目が停止されており、且つ、少なくとも１つの作業項目が別の作業項目と異なる状態にあるという判断に応じて、プロセッサは、コード５０６を実行している間に、それぞれの格納された次のＰＣに分岐することにより、スリープ状態にある作業項目のみについて実行を再開し得る。少なくとも１つの作業項目が、依然として、ループ内の基本ブロックにある命令を処理している場合には、その少なくとも１つの作業項目について実行を継続する一方で、特別な状態にある他の作業項目を待機する。スリープ状態を終えている作業項目は、ＢＢ　１に分岐して戻る。また、コード５０２は、ＢＢ　１の開始時に、ベクトルレジスタおよびサイズレジスタを初期化する。終了状態を終えている作業項目は、ＢＢ　７に分岐する。また、コード５０２は、ＢＢ　７の開始時に、それに応じて、ベクトルレジスタおよびサイズレジスタを再初期化する。

【００４４】

前述の例では、ループは単一の出口を有する。複数の出口を有する他の場合には、例えば、少なくとも次のＰＣおよび作業項目ＩＤなどのような対応する状態情報が、例えばスタックなどのメモリに格納され得る。後に、状態情報は、再開のために、例えばスタックからポップするなどのように取得され得る。スリープ状態または終了状態にある両方の作業項目は、例えばスタックなどのメモリに格納された状態情報を有し得る。異なる作業項目が異なるループ繰返しでループを終了し得るので、状態情報を有する複数のエントリが、例えばスタックなどのメモリに置かれ得る。再開時において、実行される際に、コンパイラ生成コードが状態情報をポップして、同じ次のＰＣから再開する作業項目に対する情報を組み合わせ得る。

【００４５】

ここで図６を参照すると、コンパイラ技術を用いて、プロセッサ内で複数の作業項目の並列実行を最適化するための方法６００の一実施形態が示されている。議論を進めるために、本実施形態、および後述する方法の後続の実施形態におけるステップは、連続した順序で示されている。しかし、他の実施形態では、いくつかのステップは、示されたものとは異なる順番で起こってもよく、いくつかのステップは同時に実行されてもよく、いくつかのステップは他のステップと組み合わされてもよく、また、いくつかのステップは存在しなくてもよい。

【００４６】

ブロック６０２では、ソフトウェアプログラムまたはサブルーチンが検出され解析され得る。プログラムコードは、設計者により、例えばＣまたは別の言語などの高水準言語で書かれてもよい。このソフトウェアプログラムは、ゲーム、ビジネス、医療および他の分野などにおいて、並列データアプリケーションのコンパイルおよび実行のために書かれてもよい。プログラムコードは、ソフトウェアアプリケーション、サブルーチン、ダイナミックリンクライブラリ、または他の任意の部分を指してもよい。パス名は、ユーザーによりコマンドプロンプトに対して入力され得る。あるいは、パス名は、ソースコードのコンパイルを開始するために、所与のディレクトリ位置、またはその他から読み込まれてもよい。プログラムコード内の命令は、検査され、翻訳され、最適化されて、コンパイル中にさらに処理されてもよい。

【００４７】

いくつかの実施形態では、ソースコードが静的にコンパイルされる。かかる実施形態では、フロントエンドのコンパイル中に、ソースコードが中間表現（ＩＲ）に翻訳されてもよい。バックエンドのコンパイルステップは、ＩＲを機械コードに翻訳してもよい。静的なバックエンドコンパイルは、さらなる変換および最適化を実行してもよい。他の実施形態では、ソースコードは、ジャストインタイム（ＪＩＴ）方式でコンパイルされる。ＪＩＴ方式は、システム構成を取得した後に、適切なバイナリコードを生成し得る。何れの方法でも、コンパイラは、関数呼出し、ループ、ループ内のトレースおよびプログラムコード内の基本ブロックを識別し得る。１つ以上の制御フローグラフが、プログラム解析中に構築され得る。

【００４８】
様々な実施形態では、プログラムコードは、例えば汎用プロセッサなどのプロセッサ上でコンパイルされる。プログラムコードは、ターゲットプロセッサに対してコンパイルされてもよく、ターゲットプロセッサは、例えばＳＩＭＤマイクロアーキテクチャなどの並列マイクロアーキテクチャを含む。データの１つ以上の関連付けられたレコードは、１つ以上の作業項目を生成するために、関数呼出しに割り当てられてもよい。
【００４９】
プログラムコード内の任意の分岐点を検出する前に、コンパイラは、解析され翻訳された命令を、それらがプログラムコード内に出現する通りにメモリ内に配置し得る。基本的に、コンパイラは、１のサイズをもつＶＬＩＷを生成している場合がある。コンパイラは、識別されたループ内で分岐点を検出すると（条件ブロック６０４：はい）、ブロック６０６において、超大命令語（ＶＬＩＷ）を生成し得る。コンパイラは、分岐点と対応する収束点との間の複数の基本ブロックから混ぜ合わされた命令をメモリに配置することにより、ＶＬＩＷを生成し得る。データの１つ以上の関連付けられたレコードは、生成されたＶＬＩＷ内の混ぜ合わされた命令を一緒にするために配置され割り当てられて、１つ以上の関連付けられた作業項目を生成し得る。
【００５０】
ブロック６０８では、生成されたコードが挿入されてもよく、生成されたコードは、実行される際に、ＶＬＩＷ内の命令をポイントしているオフセットを、ターゲットプロセッサ内の複数の並列実行レーンの所与のレーンにマッピングする。あるいは、マッピングは、オフセットと作業項目ＩＤとの間であってよい。ブロック６１０では、実行される際に、フェッチするための次のＶＬＩＷのサイズを更新する、生成されたコードが挿入され得る。ブロック６１２では、実行される際に、スリープ状態または終了状態になろうとしている実行のレーンに対する状態情報を格納する、生成されたコードが、ループの終わりに挿入され得る。生成されたコードは、上述した例で説明したように、プログラムコード内の特定のポイントに挿入され得る。分岐点および収束点と関連付けられた基本ブロックは、マッピングおよびＤＶＬＩＷサイズに対する更新を維持するための追加の挿入されたコードを有し得る。
【００５１】
ここで図７を参照すると、ハードウェア技術を使用して、プロセッサ内で複数の作業項目の並列実行を最適化するための方法７００の一実施形態が示されている。議論を進めるために、本実施形態、および後述する方法の後続の実施形態におけるステップは、連続した順序で示されている。しかし、他の実施形態では、いくつかのステップは、示されたものとは異なる順番で起こってもよく、いくつかのステップは同時に実行されてもよく、いくつかのステップは他のステップと組み合わされてもよく、また、いくつかのステップは、存在しなくてもよい。
【００５２】
ブロック７０２では、データの関連付けられたレコードが、複数の作業項目を生成するために、コンパイル済みコードに割り当てられる。ブロック７０４では、作業項目が、単一命令複数データ（ＳＩＭＤ）マイクロアーキテクチャを有するターゲットプロセッサに対してスケジューリングされる。ブロック７０６では、更新されたＶＬＩＷサイズおよび単一のプログラムカウンタ（ＰＣ）を使用して、更新されたＶＬＩＷサイズに等しい長さを有するＶＬＩＷが、例えばｉ－キャッシュなどのメモリからフェッチされる。ＶＬＩＷ内の命令は、ループ内の分岐点と収束点との間の別個の基本ブロックからのものであってよい。
【００５３】
ブロック７０８では、フェッチされたＶＬＩＷ内の命令と、作業項目を実行しているプロセッサ内の並列実行レーンとの間のマッピング情報のために、ベクトルレジスタが読み取られる。マッピング情報は、所与の作業項目および対応する実行レーンに対して、ＶＬＩＷ内のどの命令を処理するかを判断し得る。ブロック７１０では、ＶＬＩＷ内の命令が

、並列実行レーンを使用して同時に実行される。所与の作業項目についてループの終わりに達し（条件ブロック７１２：はい）、アクティブであると検出された作業項目がない（条件ブロック７１４：いいえ）場合には、ブロック７１６において、それぞれの状態情報が、並列実行レーンに割り当てられた各作業項目に対して読み取られる。状態情報は、少なくとも次のＰＣおよび作業項目ＩＤを含んでもよい。状態情報は、並列実行レーン内で実行を継続するために使用されてもよい。所与の作業項目についてループの終わりに達し（条件ブロック７１２：はい）、いずれかの作業項目がアクティブであると検出された（条件ブロック７１４：はい）場合には、ブロック７１８において、所与の作業項目に対する状態情報が後で使用されるために格納される。所与の作業項目は、実行を停止させてもよく、スリープ状態または終了状態に置かれ得る。

【００５４】

ここで図８を参照すると、オブジェクトコード配置８００を示す別の実施形態の一般化されたブロック図が示されている。コード配置８００は、図２および図３にそれぞれ示す制御フローグラフ２００および付随する実行順序３２０に対して、コンパイラによって生成され得るオブジェクトコードレイアウトの一実施形態を示している。コード５０２～５０６は、既に説明したものと同じ機能を実行し得る。

【００５５】

基本ブロックのサイズはコンパイル時に分かっている。コンパイラは、初期化コードを最小限にしてＤＶＬＩＷサイズを削減するようにプログラムコードの命令を配置してスケジューリングし得る。制御フローグラフ２００およびオブジェクトコード配置５００を使用する前述の例では、ＢＢ　６の４つのコピーが使用される。コンパイラは、ギャップを取り入れることにより、トレース数および付随のオフセットを削減し得る。ギャップは、ｎｏｐ操作を利用し得る。

【００５６】

コード配置５００に関して前述したステップがここで使用され得る。ＢＢ　２の完了時に、ＤＶＬＩＷサイズは２のままである。しかし、フェッチされたＤＶＬＩＷ内の２つの命令のうち１つの命令は、配置８００内のギャップを提供するｎｏｐ操作である。オフセット０はｎｏｐ操作に対応しており、オフセット１はＢＢ　３内の命令に対応している。ＢＢ　３の完了時に、ＤＶＬＩＷは、２から１に減らされる。この時、オフセット０はＢＢ　４内の命令に対応しており、オフセット１はｎｏｐ操作に対応している。

【００５７】

ＢＢＣ（０，４）の完了時に、コード５０６は、トレース０内のＢＢ　４の終わりにおいて、サイズレジスタを１から２に更新する。また、コード５０６は、ＢＢ　６に分岐している作業項目に対するエントリを、１を格納するように更新する。単一のＰＣおよび格納されたサイズをｉ－キャッシュに送信した後に、長さ２のＤＶＬＩＷがｉ－キャッシュからフェッチされる。ＤＶＬＩＷは、ＢＢ　５およびＢＢ　６からの混ぜ合わされた命令を含む。ベクトルレジスタ内に関連する格納された０のオフセットを有する作業項目は、ＢＢ（０，５）からフェッチされた命令を得る。ベクトルレジスタ内に関連する格納された１のオフセットを有する作業項目は、ＢＢ（１，６）からフェッチされた命令を得る。

【００５８】

ＢＢ（１，６）の完了時に、コード５０６は、ＤＶＬＩＷサイズを２から１に更新する。ＢＢ（１，６）に対応する作業項目は、実行を停止して、状態情報を格納し、次のＰＣで実行を再開するまで待機する。次のＰＣは、ＢＢ　１またはＢＢ　７をポイントしてもよい。前述したように、他のステップが実行されてもよい。コード配置８００は、検出された所与の分岐点と対応する収束点との間の第１のトレースパスが、所与の分岐点と対応する収束点との間の第２のトレースパスよりも小さいという判断に応じて、コンパイラが、第１のトレースパスの完了と対応する収束点との間で生成されたＶＬＩＷ内の第２のトレースパスに対応する命令とともにｎｏｐをグループ化し得ることを示す。

【００５９】

前述した実施形態の各々に対して、コードを、ループの外部でループ内のコードと並列

化することによって、さらなる並列化が生じてもよい。例えば、ＢＢ　７に対するプログラムコードは、ループを完了する作業項目に対するプログラムを終了するために、ＢＢ１と並列化されてもよい。また、オフセットが、単一の命令ではなく、ＶＬＩＷ内の複数の資源に依存しない命令に対応していることの検出に応じて、関連付けられた作業項目および実行レーンが、実行レーン内の複数の資源に依存しない命令を同時に実行してもよい。

【００６０】

さらに、コンパイラは、レジスタ割当てを使用して、ＤＶＬＩＷサイズを削減してもよい。プログラムコードは、次の文：Ｘ＝（Ａ＋Ｂ）＋（Ｂ＋Ｍ）を含み得る。ここでは、２つの演算が同じ演算コードを使用する。第１のトレースは、Ｔ１＝Ａ＋ＢなどのＡＤＤ演算を含む。第２のトレースは、Ｔ２＝Ｃ＋ＤなどのＡＤＤ演算を含む。０などのオフセットを有する基本ブロックＸ（ＢＢ　Ｘ）からのＴ１を使用する作業項目がある。１などのオフセットを有するＢＢ　ＹからのＴ２を使用する他の作業項目がある。第１のオペランド対「Ｃ」および「Ａ」、第２のオペランド対「Ｂ」および「Ｄ」、ならびに、結果対「Ｔ１」および「Ｔ２」の各々が、ＢＢ　ＸおよびＢＢ　Ｙ内の同じレジスタに割り当てられる場合には、式ｒ３＝ｒ１＋ｒ２が、１のサイズを有するＤＶＬＩＷとして使用され得る。復号時間を節約するか、またはスロットを解放するために、対応するオフセットが０に設定され得る。

【００６１】

前述の実施形態は、ソフトウェアを含み得ることに留意されたい。かかる実施形態では、方法および／または機構を実装するプログラム命令は、コンピュータ可読媒体上で伝達され、または格納され得る。プログラム命令を格納するように構成されている多数のタイプの媒体が利用可能であり、ハードディスク、フロッピィ（登録商標）ディスク、ＣＤ－ＲＯＭ、ＤＶＤ、フラッシュメモリ、プログラマブルＲＯＭ（ＰＲＯＭ）、ランダムアクセスメモリ（ＲＡＭ）、および、様々な他の形式の揮発性または不揮発性記憶装置を含む。一般的に言えば、コンピュータアクセス可能記憶媒体は、命令および／またはデータをコンピュータに提供するために、使用中にコンピュータによってアクセス可能な任意の記憶媒体を含み得る。例えば、コンピュータアクセス可能記憶媒体は、磁気または光媒体などの記憶媒体、例えば、ディスク（固定または取り外し可能）、テープ、ＣＤ－ＲＯＭもしくはＤＶＤ－ＲＯＭ、ＣＤ－Ｒ、ＣＤ－ＲＷ、ＤＶＤ－Ｒ、ＤＶＤ－ＲＷ、または、Ｂｌｕ－Ｒａｙ（登録商標）を含み得る。記憶媒体は、ＲＡＭ（例えば、シンクロナスダイナミックＲＡＭ（ＳＤＲＡＭ）、ダブルデータレート（ＤＤＲ、ＤＤＲ２、ＤＤＲ３など）ＳＤＲＡＭ、低電力ＤＤＲ（ＬＰＤＤＲ２など）ＳＤＲＡＭ、ラムバスＤＲＡＭ（ＲＤＲＡＭ）、スタティックＲＡＭ（ＳＲＡＭ）など）、ＲＯＭ、フラッシュメモリ、ユニバーサルシリアルバス（ＵＳＢ）インタフェースなどの周辺インタフェースを経由してアクセス可能な不揮発性メモリ（例えば、フラッシュメモリ）などの、揮発性または不揮発性メモリ媒体をさらに含み得る。記憶媒体は、微小電気機械システム（ＭＥＭＳ）、ならびに、ネットワークおよび／または無線リンクなどの通信媒体を経由してアクセス可能な記憶媒体を含み得る。

【００６２】

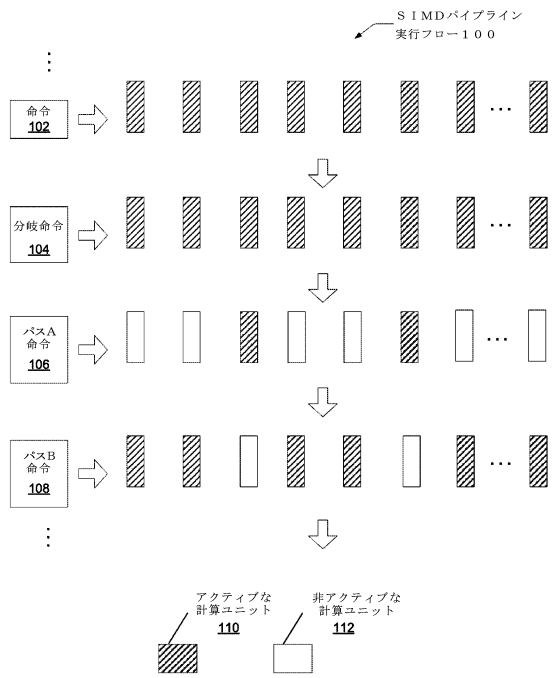
さらに、プログラム命令は、例えばＣなどの高水準プログラミング言語、例えばＶｅｒｉｌｏｇ、ＶＨＤＬなどの設計言語（ＨＤＬ）、または、例えばＧＤＳ　ＩＩストリーム形式（ＧＤＳＩＩ）などのデータベース形式、におけるハードウェア機能の動作レベル記述またはレジスタ転送レベル（ＲＴＬ）記述を含み得る。いくつかの場合には、記述は、合成ライブラリからのゲートのリストを含むネットリストを生成するために記述を合成し得る合成ツールによって読み取られてもよい。ネットリストは、システムを含むハードウェアの機能を表すゲートのセットを含む。ネットリストは、次いで、マスクに適用される幾何学形状を記述するデータセットを生成するために配置され、ルーティングされ得る。マスクは、次いで、システムに対応する半導体回路または複数の回路を製造するために様々な半導体作製ステップで使用され得る。あるいは、コンピュータアクセス可能記憶媒体

上の命令は、必要に応じて、ネットリスト（合成ライブラリの有無にかかわらず）またはデータセットであってよい。また、命令は、Ｃａｄｅｎｃｅ（登録商標）、ＥＶＥ（登録商標）およびＭｅｎｔｏｒ　Ｇｒａｐｈｉｃｓ（登録商標）などのベンダーからのハードウェアベースタイプのエミュレータによるエミュレーションのために利用され得る。

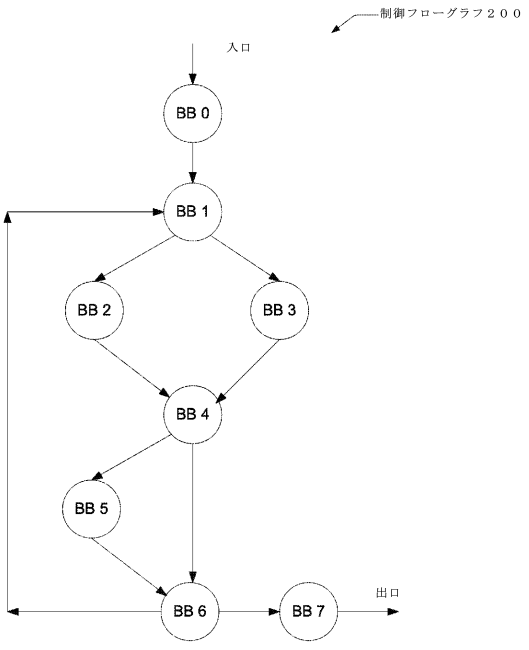
【００６３】

上記実施形態はかなり詳細に説明されているが、上述した開示が完全に理解されると、多数の変形および修正が当業者において明らかになるであろう。以下の請求項は、かかる変形および修正の全てを包含すると解釈されることを意図する。

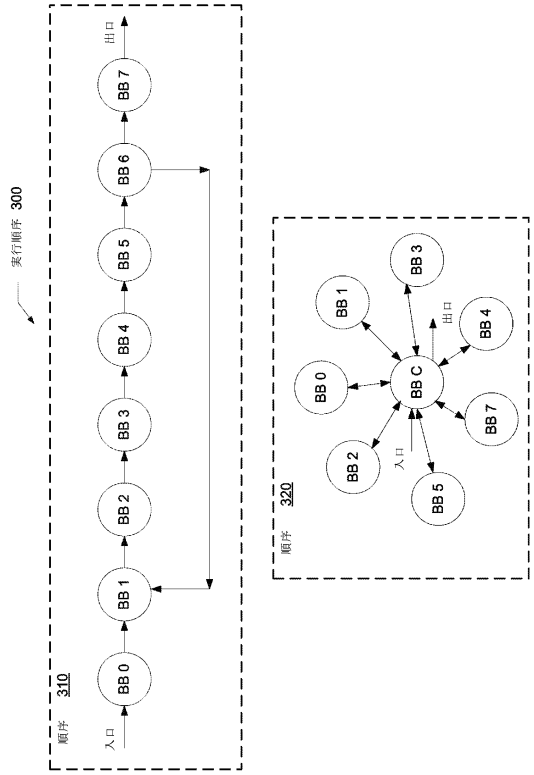
【図１】



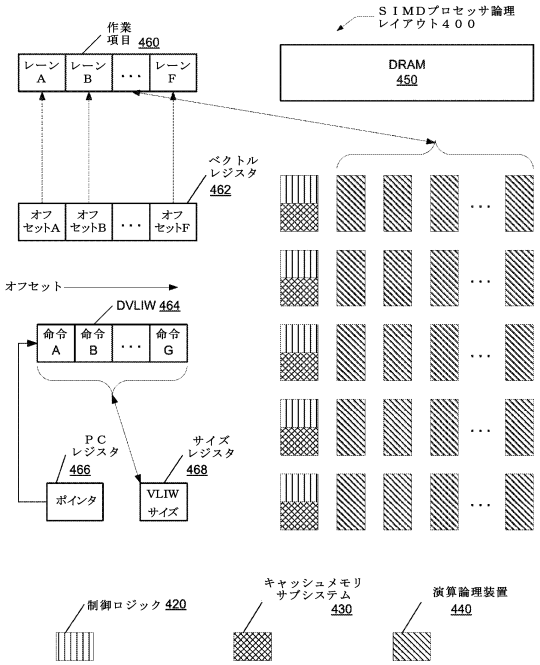

【図２】



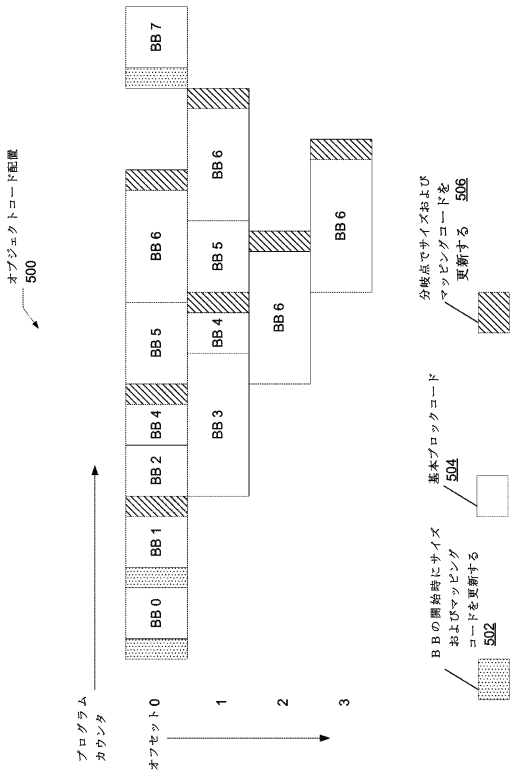
【図３】


実行順序 300
順序 310
入口
出口
BB 0
BB 1
BB 2
BB 3
BB 4
BB 5
BB 6
BB 7
順序 320
BB C

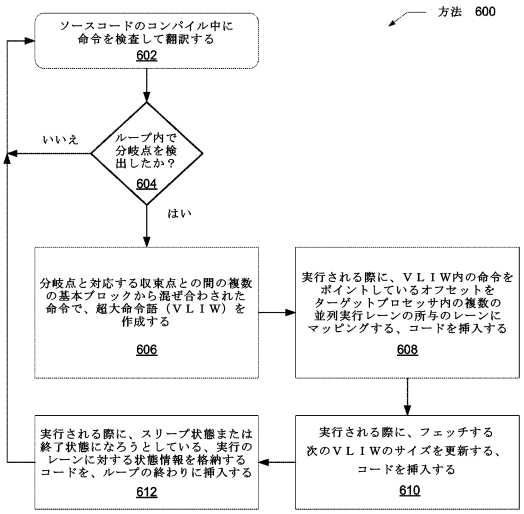
【図４】


作業項目 460
レーンA
レーンB
レーンF
SIMDプロセッサ論理レイアウト４００
DRAM 450
ベクトルレジスタ 462
オフセットA
オフセットB
オフセットF
オフセット
DVLIW 464
命令A
命令B
命令G
PCレジスタ 466
サイズレジスタ 468
ポインタ
VLIWサイズ
制御ロジック 420
キャッシュメモリサブシステム 430
演算論理装置 440

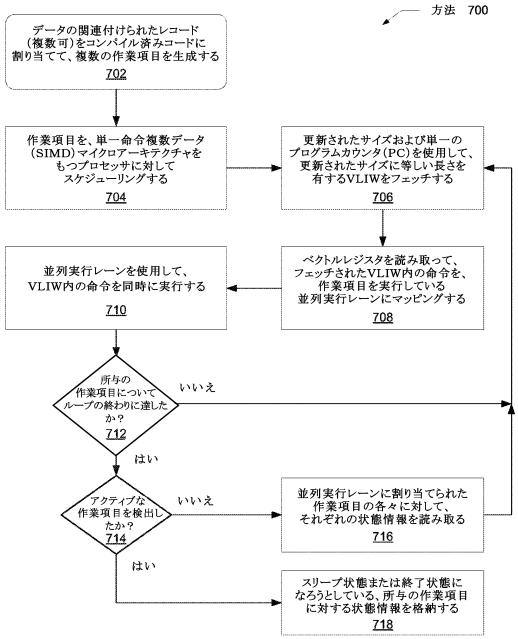
【図５】


オブジェクトコード配置 500
BB 0
BB 1
BB 2
BB 3
BB 4
BB 5
BB 6
BB 7
分岐点でサイズおよびマッピングコードを更新する 506
基本ブロックコード 504
BBの開始時にサイズおよびマッピングコードを更新する 502
プログラムカウンタ
オフセット 0
1
2
3

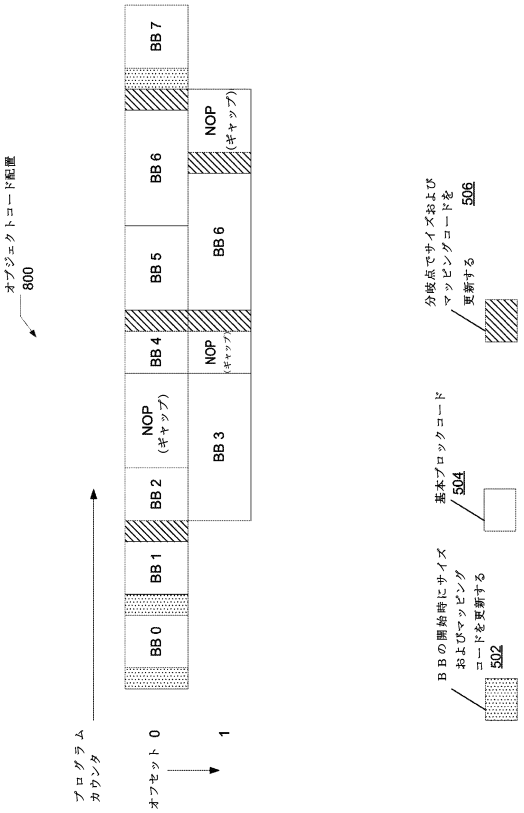
【図６】


方法 600
ソースコードのコンパイル中に命令を検査して翻訳する 602
ループ内で分岐点を検出したか？ 604
いいえ
はい
分岐点と対応する収束点との間の複数の基本ブロックから混ぜ合わされた命令で、超大命令語（ＶＬＩＷ）を作成する 606
実行される際に、ＶＬＩＷ内の命令をポイントしているオフセットをターゲットプロセッサ内の複数の並列実行レーンの所与のレーンにマッピングする、コードを挿入する 608
実行される際に、フェッチする次のＶＬＩＷのサイズを更新する、コードを挿入する 610
実行される際に、スリープ状態または終了状態になろうとしている、実行のレーンに対する状態情報を格納するコードを、ループの終わりに挿入する 612

【図７】


方法 700
データの関連付けられたレコード（複数可）をコンパイル済みコードに割り当てて、複数の作業項目を生成する 702
作業項目を、単一命令複数データ（SIMD）マイクロアーキテクチャをもつプロセッサに対してスケジューリングする 704
更新されたサイズおよび単一のプログラムカウンタ（PC）を使用して、更新されたサイズに等しい長さを有するVLIWをフェッチする 706
ベクトルレジスタを読み取って、フェッチされたVLIW内の命令を、作業項目を実行している並列実行レーンにマッピングする 708
並列実行レーンを使用して、VLIW内の命令を同時に実行する 710
所与の作業項目についてループの終わりに達したか？ 712
いいえ
はい
アクティブな作業項目を検出したか？ 714
いいえ
並列実行レーンに割り当てられた作業項目の各々に対して、それぞれの状態情報を読み取る 716
はい
スリープ状態または終了状態になろうとしている、所与の作業項目に対する状態情報を格納する 718

【図８】


オブジェクトコード配置 800
BB 7
BB 6
NOP（ギャップ）
BB 6
BB 5
BB 4
NOP（ギャップ）
NOP（ギャップ）
BB 3
BB 2
BB 1
BB 0
プログラムカウンタ
オフセット 0
1
分岐点でサイズおよびマッピングコードを更新する 506
基本ブロックコード 504
ＢＢの開始時にサイズおよびマッピングコードを更新する 502

フロントページの続き

(74)代理人 100162156
弁理士 村雨 圭介
(72)発明者 レザ ヤズダニ
アメリカ合衆国 ９４０２４ カリフォルニア州、ロスアルトス、マッツ コート ９１１

審査官 清木 泰

(56)参考文献 米国特許第０７２８７１５２（ＵＳ，Ｂ２）
米国特許出願公開第２００６／０２４２６４５（ＵＳ，Ａ１）
特開２００８－０９０７４４（ＪＰ，Ａ）
特表２００３－５２０３６０（ＪＰ，Ａ）
特表２０００－５０９５２８（ＪＰ，Ａ）
特開２０００－２５９５７９（ＪＰ，Ａ）
Wilson W. L. Fung, Ivan Sham, George Yuan, Tor M. Aamodt, Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow, Proceedings of 40th IEEE/ACM International Symposium on Microarchitecture, IEEE, ２００７年１２月 １日, Pages:407-418
Yaohua Wang, et al., Instruction Shuffle : Achieving MIMD-like Performance on SIMD Architectures, IEEE Computer Architecture Letters, IEEE, ２０１１年１２月 ６日, Vol:11, No:2, Pages:37-40
Nicolas Brunie, Sylvain Collange, Gregory Diamos, Simultaneous Branch and Warp Interweaving for Sustained GPU Performance, Proceedings of 39th Annual International Symposium on Computer Architecture, IEEE, ２０１２年 ６月 ９日, Pages:49-60
Jiayuan Meng, David Tarjan, Kevin Skadron, Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance, Proceedings of the 37th annual International Symposium on Computer Architecture (ISCA'10), ACM, ２０１０年 ６月１９日, Pages:235-246

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｆ ９／３０－ ９／４２
Ｇ０６Ｆ１５／８０
Ｇ０６Ｆ ９／４５
Ｇ０６Ｆ１７／１６
Ｇ０６Ｆ ７／５７－ ７／５７５
Ｇ０６Ｆ１５／１６－１５／１７７