(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5128602号
(P5128602)
(45)発行日　平成25年1月23日(2013.1.23)　(24)登録日　平成24年11月9日(2012.11.9)

(51)Int.Cl.　F I
G06F 9/455 (2006.01)　G06F 9/44 310A

請求項の数 20 (全 20 頁)

(21)出願番号	特願2009-529783 (P2009-529783)
(86)(22)出願日	平成19年10月1日 (2007.10.1)
(65)公表番号	特表2010-506252 (P2010-506252A)
(43)公表日	平成22年2月25日 (2010.2.25)
(86)国際出願番号	PCT/GB2007/050600
(87)国際公開番号	WO2008/041028
(87)国際公開日	平成20年4月10日 (2008.4.10)
審査請求日	平成21年11月17日 (2009.11.17)
(31)優先権主張番号	0619389.0
(32)優先日	平成18年10月2日 (2006.10.2)
(33)優先権主張国	英国 (GB)
(31)優先権主張番号	60/854,054
(32)優先日	平成18年10月24日 (2006.10.24)
(33)優先権主張国	米国 (US)

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国１０５０４　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100108501
弁理士　上野　剛史
(74)代理人　100112690
弁理士　太佐　種一
(74)代理人　100091568
弁理士　市位　嘉宏

最終頁に続く

(54)【発明の名称】プログラムコード変換に関して動的にリンクされた関数呼び出しを行うための方法及び装置

(57)【特許請求の範囲】
【請求項１】
コンピュータ装置であって、
ターゲット・プロセッサと、
メモリと
を備えており、
前記メモリは、
前記ターゲット・プロセッサ上で実行されるターゲット・オペレーティング・システムと、
前記ターゲット・オペレーティング・システム上で実行するトランスレータであって、関数リンクテーブルを保持している前記トランスレータと
を記憶しており、
前記トランスレータは、前記コンピュータ装置に、サブジェクト・コードを前記ターゲット・プロセッサで実行されるターゲット・コードへ動的に変換することを実現させ、前記サブジェクト・コードは当該サブジェクト・コードにおいて中間制御構造である手続きリンクテーブルを介してサブジェクト・リンカー・コードへサブジェクト制御フローを渡すように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを含み、
前記トランスレータは、前記コンピュータ装置に、
前記サブジェクト・コードの動的にリンクされたサブジェクト関数呼び出しに関連付け

られた手続きリンクテーブルに対する修正を検出すること、
　前記サブジェクト・コードにおいて、前記動的にリンクされた関数呼び出しを識別し、当該識別された動的にリンクされた関数呼び出しを、前記中間制御構造に対応するターゲット・コードを生成することなしに当該中間制御構造に対応する前記ターゲット・コードでない対応する関数を実行するコードに関連付けること、
　前記修正の検出に応じて、前記関数リンクテーブルに、前記動的にリンクされた関数呼び出しに対応する識別子及び前記動的にリンクされた関数呼び出しを実行するコードの位置を格納すること
　を実現させる、前記コンピュータ装置。
【請求項２】
　前記トランスレータは、前記コンピュータ装置に、前記サブジェクト・コードにおいて、動的にリンクされたサブジェクト関数呼び出しを識別し、かつ前記サブジェクト・コードの最初の変換における前記動的にリンクされた関数呼び出しに関するリンク情報を収集し、かつ前記サブジェクト・コードの後続の変換において前記収集された情報を使用することを実現させる、請求項１に記載のコンピュータ装置。
【請求項３】
　前記トランスレータは、前記コンピュータ装置に、後続の変換において前記収集された情報を用いて、前記識別された関数呼び出しから、前記対応する関数を実行する前記コードへ制御フローを受け渡すことを実現させる、請求項２に記載のコンピュータ装置。
【請求項４】
　前記トランスレータは、前記コンピュータ装置に、識別された関数呼び出しに対する識別子の形式で、前記トランスレータにアクセス可能な関数リンクテーブルにおいて、前記識別された関数呼び出しに関する情報を格納することを実現させ、かつ前記関数リンクテーブルにおいて、前記識別された関数を実行するコードの位置を格納することを実現させる、請求項１に記載のコンピュータ装置。
【請求項５】
　前記トランスレータは、前記コンピュータ装置に、各識別された関数に対応する入力に対する前記関数リンクテーブルを照合することを実現させる、前記関数リンクテーブルが前記識別された関数に対応する入力を含む場合、前記関数リンクテーブルに格納される前記情報を用いて、前記対応する関数を実行するコードに、前記関数呼び出しを関連付けることを実現させる、請求項４に記載のコンピュータ装置。
【請求項６】
　前記トランスレータは、前記コンピュータ装置に、サブジェクト・リンカー・コードの動作を監視するか、又は特有のサブジェクト・コード命令シーケンスのキャッシュフラッシュ命令を検出することにより、前記関数呼び出しに関するリンク情報を収集することを実現させる、請求項２に記載のコンピュータ装置。
【請求項７】
　前記トランスレータは、前記コンピュータ装置に、前記対応する関数を実行するコードに関連付けることにおいて、前記動的にリンクされた関数呼び出しと、前記関数を実行する前記コードとの間の直接リンクを設定することを実現させる、請求項１に記載のコンピュータ装置。
【請求項８】
　前記トランスレータは、前記コンピュータ装置に、前記関数リンクテーブルにおいて、前記識別された関数を実行するネイティブ・コードの位置、前記識別された関数を実行する事前に変換されたターゲット・コードの位置、及び前記トランスレータに既知の最適化された中間表現の部分の位置であって、当該部分から、ターゲット・コードが生成されて前記識別された関数を実行することが可能である、前記中間表現の部分の位置のうちの一つ又は複数を格納することを実現させる、請求項４に記載のコンピュータ装置。
【請求項９】
　前記トランスレータは、前記コンピュータ装置に、関数に関連した動的にリンクされた

関数呼び出しを識別し、かつ前記動的にリンクされた関数呼び出しを、前記対応する関数を実行するコードに置換することを実現させる、請求項１に記載のコンピュータ装置。
【請求項１０】
前記トランスレータは、前記コンピュータ装置に、前記サブジェクト・コードの手続きリンクテーブル（ＰＬＴ）を介してサブジェクト・リンカー・コードへサブジェクト制御フローを渡すように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを含むサブジェクト・コードを受信し、前記受信されたサブジェクト・コードにおいて、前記動的にリンクされた関数呼び出しを識別し、かつ前記動的にリンクされた関数呼び出しを、前記ＰＬＴに対応するターゲット・コードを生成することなしに当該ＰＬＴに対応する前記ターゲット・コードでない前記対応する関数を実行するコードに関連付けることを実現させる、請求項１に記載のコンピュータ装置。
【請求項１１】
サブジェクト・コードをターゲット・プロセッサ上で実行するターゲット・コードに動的に変換する方法であって、前記サブジェクト・コードは当該サブジェクト・コードにおいて中間制御構造を介してサブジェクト・リンカー・コードへサブジェクト制御フローを渡すように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを含み、コンピュータが、
前記サブジェクト・コードの動的にリンクされたサブジェクト関数呼び出しに関連付けられた手続きリンクテーブルに対する修正を検出するステップと、
前記ターゲット・プロセッサにおいて実行するターゲット・コードを生成するステップであって、前記ターゲット・プロセッサにおいて、前記動的にリンクされた関数呼び出しは、前記中間制御構造に対応するターゲット・コードを生成することなしに当該中間制御構造に対応する前記ターゲット・コードでない前記対応する関数を実行するコードに関連付けられる、前記生成するステップと、
前記修正の検出に応じて、前記ターゲット・オペレーティング・システム上で実行するトランスレータに保持された関数リンクテーブルに、前記動的にリンクされた関数呼び出しに対応する識別子及び前記動的にリンクされた関数呼び出しを実行するコードの位置を格納するステップと、
を実行することを含む、前記方法。
【請求項１２】
前記サブジェクト・コードの最初の変換において、前記識別された動的にリンクされた関数呼び出しに関するリンク情報が、前記サブジェクト・コードの後続の変換における使用のために収集される、請求項１１に記載の方法。
【請求項１３】
後続の変換において、最初の変換において収集された前記情報は、前記識別された関数呼び出しから、前記対応する関数を実行する前記コードへ、制御フローを受け渡すために用いられる、請求項１２に記載の方法。
【請求項１４】
前記コンピュータが、
関数リンクテーブル中に、前記識別された関数呼び出しに関する情報を格納するステップと、
識別された関数の各々に対応する入力に対する前記関数リンクテーブルを照合すること、及び前記関数リンクテーブルが前記識別された関数と対応する入力を含む場合、前記関数リンクテーブルに格納された前記情報を用いて前記対応する関数を実行するコードに前記関数呼び出しを関連付けるステップと
を実行することを含む、請求項１１に記載の方法。
【請求項１５】
前記ターゲット・コードを生成するステップは、前記ターゲット・コードにおいて、前記動的にリンクされた関数呼び出しと、前記関数を実行する前記コードとの間に直接リンクを設定するステップを含む、請求項１１に記載の方法。

【請求項１６】
前記コンピュータが、
前記関数リンクテーブル中に、前記識別された関数を実行するネイティブ・コードの位置、前記識別された関数を実行する事前に変換されたターゲット・コードの位置、最適化された中間表現の部分の位置であって、当該部分から、ターゲット・コードが生成されて前記識別された関数を実行することが可能である、前記最適化された中間表現の部分の位置、及びデリファレンスされる変数を含むグループから選択された一つ又は複数を格納するステップ
を実行することを含み、
前記デリファレンスされる変数は、前記識別された関数を実行するサブジェクト・コード、前記識別された関数を実行するネイティブ・コード、前記識別された関数を実行する事前に変換されたターゲット・コード、前記トランスレータに既知の最適化された中間表現の部分であって、当該部分からターゲット・コードを生成して前記識別された関数を実行することが可能である、前記トランスレータに既知の最適化された中間表現の部分のうちの一つの位置に関連する、
請求項１４に記載の方法。
【請求項１７】
前記コンピュータが、
関数に関連した動的にリンクされる関数呼び出しを識別するステップと、
前記動的にリンクされた関数呼び出しを、前記対応する関数を実行するコードに置換するステップと
を実行することを含む、請求項１２に記載の方法。
【請求項１８】
前記コンピュータが、
前記サブジェクト・コードの手続きリンクテーブル（ＰＬＴ）を介してサブジェクト・リンカー・コードへサブジェクト制御フローを渡すように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを受信するステップと、
前記サブジェクト・コードにおいて、前記動的にリンクされた関数呼び出しを識別するステップと、
前記動的にリンクされた関数呼び出しを、前記ＰＬＴに対応するターゲット・コードを生成することなしに当該ＰＬＴに対応する前記ターゲット・コードでない前記対応する関数を実行するコードと関連付けるステップと
を実行することを含む、請求項１２に記載の方法。
【請求項１９】
ターゲット・プロセッサにおいて実行されるプログラムコード変換のためのコンピュータ・プログラムであって、コンピュータに、請求項１１～１８のいずれか一項に記載の方法の各ステップを実行させる、前記コンピュータ・プログラム。
【請求項２０】
サブジェクト・コードをターゲット・プロセッサ上で実行するターゲット・コードへ動的に変換するように構成されたコード変換装置であって、前記サブジェクト・コードは、当該サブジェクト・コードにおいて中間制御構造を介してサブジェクト・リンカー・コードへサブジェクト制御フローを渡すように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを含み、前記コード変換装置は、
前記サブジェクト・コードの動的にリンクされたサブジェクト関数呼び出しに関連付けられた手続きリンクテーブルに対する修正を検出すること、
前記ターゲット・プロセッサ上で実行するターゲット・コードを生成することであって、前記識別された動的にリンクされた関数呼び出しは、前記中間制御構造に対応するターゲット・コードを生成することなしに当該中間制御構造に対応する前記ターゲット・コードでない対応する関数を実行するコードに関連付けられる、前記生成すること、
前記修正の検出に応じて、前記関数リンクテーブルに、前記動的にリンクされた関数呼

び出しに対応する識別子及び前記動的にリンクされた関数呼び出しを実行するコードの位置を格納すること
を行うように構成されている、前記コード変換装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、一般に計算機の分野に関し、より詳細には、例えば動的にリンクされた関数呼び出しを含むプログラムコード変換を行うコード変換装置、エミュレータ、及びアクセラレータにおいて有用な、プログラムコード変換の方法及び装置に関する。
【背景技術】
【０００２】
組み込みＣＰＵ及び非組み込みＣＰＵ市場にわたって、有力な命令セットアーキテクチャ（ＩＳＡ）が知られており、それらのアーキテクチャに対して、多数のソフトウェアが存在し、そのソフトウェアは、性能に関して「加速される」ことが可能であり、またはプロセッサが、その関連するソフトウェアに透過的にアクセスすることが可能な場合に、より一層良好なコスト／性能という利点をもたらすことができる無数の高機能プロセッサへ「変換される」（トランスレートされる）ことが可能である。また、ＩＳＡに合わせて固定され、かつ性能又は市場範囲の点において進化が不可能であるが、「合成ＣＰＵ」共通アーキテクチャの恩恵を享受するであろう、有力なＣＰＵアーキテクチャが存在することも知られている。
【０００３】
第１タイプのコンピュータプロセッサ（“サブジェクト”プロセッサ、対象プロセッサともいう）に関して記述されるプログラムコードを、第２タイプのプロセッサ（“ターゲット”プロセッサ、目的プロセッサともいう）上で実行することが大抵の場合望ましい。この場合、エミュレータまたはトランスレータを使用してプログラムコード変換を実行することにより、サブジェクト・プログラムをターゲット・プロセッサ上で実行することが可能となる。国際公開第ＷＯ００／２２５２１号パンフレットには、本発明の実施形態に用い得るようなアクセラレーション、変換、及び共通アーキテクチャ機能を容易にする、プログラムコード変換方法及び装置が開示されている。
【０００４】
変換されるべきサブジェクト・プログラムは大抵、サブジェクト・コードの多数のユニットを備え、サブジェクト・コードはサブジェクト・アプリケーションと多数のサブジェクト・ライブラリを含み、そのいくつかは所有権のあるものである可能性があり、そのいくつかは、サブジェクトＯＳ（“システムライブラリ”）の一部として提供される。サブジェクト・プログラムが実行される際、関数呼び出しとしてのサブジェクト・コードのこれらの異なるユニット間の制御フローのパスがライブラリに対して形成される。
【０００５】
例えばサン　マイクロシステムズ　Ｉｎｃ．のソラリス（Solaris）のような、特定のオペレーティング・システムにおいては、関数呼び出しを、その関数を実行するライブラリ・コードへリンクするプロセスが実行時に実施されることが可能であり、この手続きは動的リンクとして知られている。動的リンクは、動的リンカー・コードにより実行され、手続きリンクテーブル（ＰＬＴ）として知られる中間制御構造の使用を含む。
【０００６】
ＰＬＴはコンパイルされたプログラムの不可欠な要素であり、そのプログラムに必要とされるライブラリ関数の各々に対するリンク情報を含む入力を備える。プログラムの標準的な実行において、ライブラリ関数への最初の呼び出しが起きた場合、制御フローはその関数に関連したＰＬＴ入力へジャンプする。その関数に対するＰＬＴ入力は、動的リンカー・コードを含むことによりそのステージのリンクプロセスを制御する。動的リンカー・コードは、当該関数に対するリンク情報を更新させる。リンク情報を更新することにより、動的リンカー・コードは、その関数に対するＰＬＴ入力から、その関数を実行するライ

ブラリ・コードへのリンクを形成する。このようにして構築されたリンクは、プログラムの実行の終了まで持続する。

【０００７】

その結果、動的リンカー・コードは、ライブラリにおいてコードの制御フローを渡して関数が実行されるようにすることができる。

プログラムによる関数の後続の呼び出しは、事前にＰＬＴ入力に対して行われたように、制御フローを渡す。ＰＬＴ入力は、関数を実行するライブラリ・コードへのリンクと共に更新されたので、これら後続の呼び出しにおいて、ＰＬＴは制御フローをＰＬＴからライブラリへ直接的に渡す。これら後続の呼び出しは、ＰＬＴ入力の更なる更新も、動的リンカーの更なる呼び出しも必要としない。

【０００８】

特定のオペレーティング・システムにおいて、関数に対するＰＬＴ入力と、その関数を実行するためのコードを含むライブラリとの間のリンク形成の実行は、グローバル・オフセット・テーブルのような、ＰＬＴに関連するデータを書き換える効果を有する。その結果、書き換えられたグローバル・オフセット・デーブルは、ＰＬＴ入力においてコードによってランタイムに読み込まれることが可能であり、リンクの形成を可能にする。例えばソラリスのような、特定の別のオペレーティング・システムにおいては、特定の関数に対するＰＬＴ入力と、その関数を含むライブラリとの間のリンク形成の実行は、ＰＬＴ入力自体を形成する実行可能なコードを書き換える別の効果を有する。

【０００９】

上述のような中間制御構造としてのＰＬＴの使用、特に、関数が呼び出される第一の時間においてＰＬＴを備えるコードを書き換えることによる、ＰＬＴにおけるリンク情報の書き換えは、動的リンク方法に不可欠な要素としてＰＬＴを用いるサブジェクト・コードに対するプログラムコード変換を複雑にする。

【００１０】

書き換えられたサブジェクト・コード（対象コードともいう）は、既に変換されたターゲット・コード（目的コードともいう）に対応する可能性があるので、ランタイムにおけるコード書き換えは、動的トランスレータに対して問題を提起する。サブジェクト・コードのそのような書き換えが生じた場合、書き換えられたサブジェクト・コードの全てのターゲット・コード変換は、識別され、かつ陳腐化したものとして廃棄されなければならない。従って、トランスレータは、書き換えられている特定のサブジェクト・コード・アドレスに対応する全てのターゲット・コード・シーケンス（すなわち、変換）を識別できなくてはならない。

【００１１】

動的トランスレータにおいて、所与のサブジェクト・アドレスに対応するターゲット・コードを発見して消去することは困難であり、時に不可能ですらある。ある状況においては、変換の間に最適化が適用され、その変換は、それが示すサブジェクト・アドレスの範囲にもはや正確に関連付けることが不可能な変換を生じさせる。これらの状況において、サブジェクト・プログラムが、特定のサブジェクト・アドレスにおいて自身のコードを書き換える場合、トランスレータは、どの個別の変換されたターゲット・コードが無効にされるべきかを識別するすべがない。加えて、マルチスレッド環境における変換されたターゲット・コードの安全な消去は、更なる問題を提起する可能性がある。

【００１２】

コード書き換えに関連する技術は、国際公開第ＷＯ０５／００８４８７号パンフレットにおいて記載されている。これらの技術は有用であるが、本発明は、ランタイムのＰＬＴにおいて生じるコード書き換えの多数の集中により、国際公開第ＷＯ０５／００８４８７号パンフレットに記載されているような技術が、ＰＬＴの更新を処理する非効率的な方法となり得ることを確認した。このような技術は、無効な変換されたターゲット・コードが実行されないことを保証し得るが、これらの技術を用いてＰＬＴの更新を処理する場合、制御フローの管理はプロセッサ及びメモリ資源の観点で高価であることが分かった。

【００１３】
更に、発明者は、ソラリスオペレーティング・システムにおいて、動的リンカーにより使用されるＰＬＴの更新方法は、ある別のオペレーティング・システムにおける動的リンカーと比較して、標準的な動的リンクされたプログラムを実行するのに必要とされるパーティションの数を顕著に増加させることを確認した。
【００１４】
サブジェクト・プログラム・コードの変換に関して動的にリンクされた関数呼び出しを行う方法が、本方法を活用するコンピュータ装置と共に提供される。好適な実施形態においては、中間制御構造に対応するターゲット・コードを生成することなく、関数リンクテーブルを用いて、サブジェクト・コードにおいて関数呼び出しからサブジェクト・コード関数へリンクを設定することを可能とする。サブジェクト・コードをターゲット・コードへ変換する働きをするトランスレータを含むコンピュータ装置は、サブジェクト・プログラムにおいて動的にリンクされた関数呼び出しが識別されるように構成されても良く、かつ関数を実行するコードとの直接的な関連付けであって、関数を実行するトランスレータによって関数リンクテーブルに集約される情報に基づいた関連付けを特徴付けるターゲット・コードが生成されるように構成されても良い。
【００１５】
動的にリンクされた関数呼び出しを行う技術の好適な実施形態は、サブジェクト・コードにおいて動的にリンクされた関数呼び出しをその関数を実行するコードと都合よく関連付けることを可能とする。好適な実施形態は、コード書き換えに関する国際公開第ＷＯ０５／００８４８７号パンフレットから知られるような、技術と関連したプロセッシング及びメモリのオーバーヘッドを減少させ得る。
【発明の概要】
【発明が解決しようとする課題】
【００１６】
従って、本発明は、例えばプログラムコード変換を行いつつ、コンピュータシステムの性能を改善する。
【課題を解決するための手段】
【００１７】
本発明によれば、添付の特許請求の範囲に記載されているような装置及び方法が提供される。本発明の好適な特徴は、従属する特許請求の範囲及び以下の記載から明らかになるであろう。
【００１８】
一つの側面においては、コンピュータ装置であって、ターゲット・プロセッサ、及びサブジェクト・プロセッサ上での実行のためのサブジェクト・コードを受信し、かつターゲット・プロセッサ上での実行のためのターゲット・コードを生成するように構成されたトランスレータを備え、トランスレータは、（ａ）サブジェクト・コードにおいて中間制御構造を介してサブジェクト・リンカー・コードへサブジェクト制御フローを渡すように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを備えるサブジェクト・コードを受信すること、ここで前記中間構造を介して動的にリンクされた関数呼び出しを、関数を実行するサブジェクト・コードに関連付けることが可能である、（ｂ）受信したサブジェクト・コードにおいてそのような動的リンクされた関数呼び出しを識別して、そのような動的にリンクされた関数呼び出しを、中間制御構造に対応するターゲット・コードを生成することなく、対応する関数を実行するコードに関連付けることを行うように構成されるコンピュータ装置が提供される。
【００１９】
トランスレータは、受信したサブジェクト・コードにおいて、動的にリンクされたサブジェクト関数呼び出しを識別し、受信したサブジェクト・コードの最初の変換において動的にリンクされた関数呼び出しに関するリンク情報を収集し、及び受信したサブジェクト・コードの後続の変換において収集された情報を使用するように構成されても良い。コン

ピュータ装置は、後続の変換において収集された情報を用いて、識別された関数呼び出しから、対応する関数を実行するコードへ制御フローを受け渡すように構成されても良い。

【００２０】

コンピュータ装置は、トランスレータが、サブジェクト・コードの手続きリンクテーブル（ＰＬＴ）を介してサブジェクト・リンカー・コードへサブジェクト制御フローを渡すように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを含むサブジェクト・コードを受信する場合、これらの関数呼び出しは、ＰＬＴに対応するターゲット・コードを生成することなく、受信されたサブジェクト・コードにおいて、トランスレータにより識別され、対応する関数を実行するコードに関連付けられるように構成されても良い。

【００２１】

別の側面においては、ターゲット・プロセッサにおいて実行されるプログラムコード変換の方法が提供され、本方法は、サブジェクト・コードを受信することであって、受信されたサブジェクト・コードは、サブジェクト・コードにおいて中間制御構造を介してサブジェクト制御フローをサブジェクト・リンカー・コードへ渡して、それにより関数を実行するサブジェクト・コードに動的にリンクされたサブジェクト関数呼び出しを関連付けるように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを含む、前記受信すること、受信されたサブジェクト・コードにおいてこのような動的にリンクされた関数呼び出しを識別すること、及びターゲット・プロセッサにおいて実行するターゲット・コードを生成することであって、ターゲット・プロセッサにおいて、そのような動的にリンクされた関数呼び出しは、中間制御構造に対応するターゲット・コードを生成することなく、対応する関数を実行するコードに関連付けられる、生成することを備える。

【００２２】

更に別の側面においては、コンピュータで読み取り可能なメディアが提供され、メディアは、その上に記録されたコンピュータにより実行可能な、ターゲット・プロセッサ上で実行されるプログラムコード変換の方法を実行するための命令を有し、その方法は、サブジェクト・コードを受信することであって、受信されたサブジェクト・コードはサブジェクト・コードにおいて中間制御構造を介してサブジェクト制御フローをサブジェクト・リンカー・コードに渡して、それにより関数を実行するサブジェクト・コードに動的にリンクされたサブジェクト関数呼び出しを関連付けるように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを含む、前記受信すること、受信されたサブジェクト・コードにおいてこのような動的にリンクされた関数呼び出しを識別すること、及びターゲット・プロセッサにおいて実行するターゲット・コードを生成することであって、ターゲット・プロセッサにおいて、そのような動的にリンクされた関数呼び出しは、中間制御構造に対応するターゲット・コードを生成することなく、対応する関数を実行するコードに関連付けられる、生成することを備える。

【００２３】

更に別の側面においては、トランスレータが提供され、トランスレータは、サブジェクト・コードを受信し、かつターゲット・プロセッサ上で実行するターゲット・コードを生成するように構成され、トランスレータは、サブジェクト・コードを受信するように構成され、受信されたサブジェクト・コードは、サブジェクト・コードにおいて中間制御構造を介してサブジェクト・リンカー・コードをサブジェクト制御フローへ渡して、それにより関数を実行するサブジェクト・コードに動的にリンクされたサブジェクト関数呼び出しを関連付けるように構成された、一つ又は複数の動的にリンクされたサブジェクト関数呼び出しを含み、受信されたサブジェクト・コードにおいて、このような動的にリンクされた関数呼び出しを識別するように構成され、及びターゲット・プロセッサ上で実行するターゲット・コードを生成するように構成され、ターゲット・プロセッサにおいて、そのような動的にリンクされた関数呼び出しは、中間制御構造に対応するターゲット・コードを生成することなく、対応する関数を実行するコードに関連付けられる。

【００２４】
上記は、本発明の実施形態の種々の側面の概要である。これは、当業者がこの後の本発明の詳細な議論をより迅速に理解するように導入として提供されたものであって、本明細書に添付した特許請求の範囲の技術思想を決して限定するものではなく、そのように意図されたものでもない。
【図面の簡単な説明】
【００２５】
【図１】本発明の実施形態が適用される装置を示すブロック図である。
【図２】本発明の実施形態に用いられる変換ユニットの概略図である。
【図３】本発明の実施形態に用いられる装置を示すブロック図である。
【図４】関数呼び出しを行う方法の例を説明するフロー概略図である。
【発明を実施するための形態】
【００２６】
本明細書の一部に組み込まれ、そして本明細書の一部を構成する添付の図面は、現時点において好適である実施形態を示す。
以下の記載は、当業者が本発明を実施し、かつ使用することを可能とするよう提供され、本願発明者により意図される、本発明を実施する最良のモードを説明する。しかしながら、本発明の一般的原理が本明細書において詳細に定義され、改良されたプログラムコード変換方法及び装置が提供されているので、当業者には種々の変更が容易に理解されるであろう。
【００２７】
図１を参照すると、サブジェクト・プログラム１７は、サブジェクト・プロセッサ３を有するサブジェクト計算プラットフォーム１上で実行されることを意図されている。しかしながら、ターゲット計算プラットフォーム１０を代わりに使用して、サブジェクト・プログラム１７を、プログラムコード変換を実行するトランスレータ・コード１９を介して実行する。トランスレータ・コード１９は、サブジェクト・コード１７からターゲット・コード２１へのコード変換を実行して、ターゲット・コード２１をターゲット計算プラットフォーム１０上で実行可能にする。
【００２８】
当業者には良く知られているように、サブジェクト・プロセッサ３は一組のサブジェクト・レジスタ５を有する。サブジェクト・メモリ８は、とりわけ、サブジェクト・コード１７及びサブジェクト・オペレーティング・システム２を格納する。同様に、図１の例示的なターゲット計算プラットフォーム１０は、複数のターゲット・レジスタ１５を有するターゲット・プロセッサ１３と、ターゲット・オペレーティング・システム２０、サブジェクト・コード１７、トランスレータ・コード１９、及び変換されたターゲット・コード２１を含む複数の動作コンポーネントを格納するメモリ１８とを含む。ターゲット計算プラットフォーム１０は通常、マイクロプロセッサ・ベースのコンピュータ、又は他の適切なコンピュータである。
【００２９】
一実施形態においては、トランスレータ・コード１９は、サブジェクト命令セットアーキテクチャ（ＩＳＡ）のサブジェクト・コードを、別のＩＳＡの変換されたターゲット・コードに、最適化を行なって、または最適化を行なうことなく、変換するエミュレータである。別の実施形態においては、トランスレータ・コード１９は、プログラムコードの最適化を実行することにより、サブジェクト・コードを各コードが同じＩＳＡであるターゲット・コードに変換するアクセラレータとして機能する。
【００３０】
トランスレータ・コード１９は、適切には、トランスレータを実装するソースコードのコンパイルされたバージョンであり、オペレーティング・システム２０と連動してターゲット・プロセッサ１３上で実行される。図１に示す構造は例示に過ぎず、例えば、本発明の実施形態によるソフトウェア、方法、及びプロセスは、オペレーティング・システム２

０内の、又はオペレーティング・システム２０の下位にあるコードとして実装し得ることを理解されたい。サブジェクト・コード１７、トランスレータ・コード１９、オペレーティング・システム２０、及びメモリ１８の格納方法は、当業者に既知のような広範囲の種々のタイプの内の任意のものであって良い。

【００３１】

図１による装置においては、プログラムコード変換はランタイムに動的に行なわれ、ターゲット・コード２１が実行されている状態で、ターゲット・アーキテクチャ１０上で実行される。すなわち、トランスレータ・コード１９は、変換されたターゲット・コード２１とインラインで実行される。トランスレータ・コード１９を介してサブジェクト・プログラム１７を実行することは、二つの異なるタイプのコードを含み、トランスレータ・コード１９及びターゲット・コード２１を交互に実行する。従って、ターゲット・コード２１は、変換されているプログラム中の格納されたサブジェクト・コード１７に基づいて、トランスレータ・コード１９によってランタイムを通して生成される。

【００３２】

一実施形態では、トランスレータ・コード１９は、サブジェクト・プログラム１７をターゲット・コード２１としてターゲット・プロセッサ１３上で実際に実行しつつ、サブジェクト・プロセッサ３、及び特にサブジェクト・レジスタ５のような、サブジェクト・アーキテクチャ１の関連部分をエミュレートする。好適な実施形態では、少なくとも一つのグローバル・レジスタ格納部２７が設けられる（サブジェクト・レジスタ・バンク２７又は抽象レジスタ・バンク２７とも表記される）。マルチプロセッサ環境では、任意的に、一つよりも多くの抽象レジスタ・バンク２７が、サブジェクト・プロセッサのアーキテクチャに従って設けられる。サブジェクト状態の表現は、トランスレータ・コード１９及びターゲット・コード２１の構成要素によって提供される。すなわち、トランスレータ・コード１９はサブジェクト状態を、変数及び／又はオブジェクトのような種々の明示的なプログラミング言語装置において格納する。これと比較すると、変換されたターゲット・コード２１は、ターゲット・レジスタ１５及び記憶領域１８に暗黙のうちに格納されるサブジェクト・プロセッサ状態を供給し、これらのターゲット・レジスタ１５及び記憶領域１８は、ターゲット・コード２１のターゲット命令によって操作される。例えば、グローバル・レジスタ格納部２７の低レベル表現は、割当てられたメモリの単なる領域である。しかしながら、トランスレータ・コード１９のソースコードにおいては、グローバル・レジスタ格納部２７は、高いレベルでアクセスされ、かつ操作されることが可能なデータアレイまたはオブジェクトである。

【００３３】

「基本ブロック」という用語は、当業者には良く理解されるであろう。基本ブロックは、厳密に一つの入力ポイント、及び厳密に一つの終了ポイントを有するコードセクションであり、このコードセクションは、ブロックコードを単一の制御パスに制限する。この理由で、基本ブロックは有用な基本制御フロー単位である。適切には、トランスレータ・コード１９はサブジェクト・コード１７を複数の基本ブロックに分割し、各基本ブロックは、単一の入力ポイントにおける先頭命令と、単一の終了ポイントにおける末尾命令との間の連続的な命令セットである（ジャンプ、呼び出し、又は分岐命令のような）。トランスレータ・コード１９は、これらの基本ブロックの内の一つの基本ブロックだけを選択しても良く（ブロックモード）、又は１グループの基本ブロックを選択しても良い（グループブロックモード）。一つのグループブロックは適切には、二つ以上の基本ブロックを含み、これらの基本ブロックは一括して単一ユニットとして処理されることになる。更に、トランスレータは、サブジェクト・コードの同じ基本ブロックを異なる入力条件で表現するアイソ・ブロックを形成しても良い。

【００３４】

好適な実施形態では、中間表現ツリーは、ターゲット・コード２１を元のサブジェクト・プログラム１７から生成するプロセスの一部として、サブジェクト命令シーケンスに基づいて生成される。ＩＲツリーは、サブジェクト・プログラムによって計算される表現及

び実行される動作の抽象表現である。後の時点で、ターゲット・コード２１はＩＲツリーに基づいて生成される。ＩＲノード群の集合は、実際は非巡回有向グラフ（ＤＡＧ）であるが、通称として「ツリー」と表記される。

【００３５】

当業者に理解されるように、一実施形態では、トランスレータ・コード１９は、Ｃ＋＋のようなオブジェクト指向プログラミング言語を用いて実装される。例えば、ＩＲノードは、Ｃ＋＋オブジェクトとして実装され、他のノードへの参照は、これらの他のノードに対応するＣ＋＋オブジェクトへのＣ＋＋参照として実装される。従って、ＩＲツリーは、互いに対する種々の参照を含むＩＲノードオブジェクトの集合として実装される。

【００３６】

更に、議論されている実施形態においては、ＩＲ生成は、一組の抽象レジスタ定義を使用し、これらの抽象レジスタ定義は、サブジェクト・アーキテクチャの特定の機能に対応し、サブジェクト・プログラム１７は、サブジェクト・アーキテクチャ上で実行されることが意図されている。例えば、固有の抽象レジスタ定義がサブジェクト・アーキテクチャ上の各物理レジスタ（すなわち、図１のサブジェクト・レジスタ５）に対して存在する。そのように、トランスレータにおける抽象レジスタ定義は、ＩＲノードオブジェクト（すなわちＩＲツリー）への参照を含むＣ＋＋オブジェクトとして用いることができる。一組の抽象レジスタ定義によって参照される全てのＩＲツリーの集合は、ワーキングＩＲフォレストと表記される（「フォレスト」と表記されるのは、フォレストが複数の抽象レジスタルートを含み、これらの抽象レジスタルートの各々は一つのＩＲツリーを参照するからである）。これらのＩＲツリー及び他のプロセスは、トランスレータ・コード１９の一部を適切に形成する。

【００３７】

図２は、ターゲット計算プラットフォーム１０上で実行される場合のトランスレータ・コード１９を更に詳細に示している。上に議論したように、トランスレータ・コード１９のフロントエンドはデコーダユニット１９１を含み、デコーダユニット１９１はサブジェクト・プログラム１７の現時点で必要とされるセクションをデコードして複数のサブジェクト・コードブロック１７１ａ，１７１ｂ，１７１ｃ（これらのサブジェクト・コードブロックは通常それぞれサブジェクト・コードの一つの基本ブロックを含む）を提供することもでき、各サブジェクト・ブロックに関するデコーダ情報１７２、及びサブジェクト・ブロックに含まれ、かつトランスレータ・コード１９の後の時点での動作を支援するサブジェクト命令を提供することもできる。或る実施形態では、トランスレータ・コード１９のコアにおけるＩＲユニット１９２は中間表現（ＩＲ）を、デコードされたサブジェクト命令に基づいて生成し、最適化が中間表現に対して適宜行なわれる。トランスレータ・コード１９のバックエンドの一部としてのエンコーダ１９３は、ターゲット・プロセッサ１３により実行されることが可能なターゲット・コード２１を生成する（設置する）。この単純化された例では、三つのターゲット・コードブロック２１１ａ～２１１ｃが生成され、サブジェクト・コードブロック１７１ａ～１７１ｃをサブジェクト・プラットフォーム１上で実行することと等価なことが、ターゲット・プラットフォーム１０上で実行される。また、エンコーダ１９３は、ヘッダコード及び／又はフッタコード２１２を、ターゲット・コードブロック２１１ａ～２１１ｃの幾つかの、または全てのターゲット・コードブロックに対して生成することができ、ヘッダコード及び／又はフッタコード２１２は、ターゲット・ブロックが動作する環境を設定するような機能、及び必要に応じて制御をトランスレータ・コード１９に戻すような機能を実行する。

【００３８】

図３は、本発明の実施形態により用いられるような装置を示す更に詳細な概略図である。図３の説明のための具体例において、トランスレータ・コード１９はＳＰＡＲＣからｘ８６への変換を実行するように構成されている。

【００３９】

サブジェクト・コード１７は、ターゲット・コード２１ａに変換されるべきサブジェク

ト実行ファイル１７ａを備える。サブジェクト実行ファイル１７ａは、所有権のあるライブラリ及び／又はシステムライブラリを含む多数のサブジェクト・ライブラリを順に参照してもよく、その中の関数を利用しても良い。二つの例示的なライブラリ１７ｂ、１７ｃが図示されている。

【００４０】

サブジェクト実行ファイル１７ａは、サブジェクト・プロセッサのオペレーティング・システムと互換性のある実行ファイルフォーマットにより構成されている。典型的には、サブジェクト実行ファイル１７ａは、コード１７ａ＿１の本文と、ヘッダ１７ａ＿２とを備える。ヘッダ１７ａ＿２は、コード１７ａ＿１の本文に関する情報を提供し、例えばその情報は、サブジェクト実行ファイルを解析する際に有用な情報、およびサブジェクト実行ファイル１７ａが実行される場合、動的リンクを実行するために用いられる情報である。「ヘッダ」として参照されているが、ヘッダ１７ａ＿２は、完全に、又は部分的に、実行ファイル１７ａの最初の部分から離れて存在しても良い。

【００４１】

図３の説明のための具体例において、サブジェクト実行ファイル１７ａは、実行可能リンクフォーマット（ＥＬＦ）に従って構成されている。ＥＬＦ標準は広く用いられており、この構造のファイルのヘッダ１７ａ＿１は、サブジェクト実行ファイル１７ａに関連した手続きリンクテーブル（ＰＬＴ）１７ｄに関する情報を含む。

【００４２】

サブジェクト・コード１７がサブジェクト・プロセッサ３上でネイティブに実行されている場合、サブジェクト実行ファイル１７ａにおける関数ライブラリ１７ｂ及び１７ｃへの呼び出しは、ＰＬＴ　１７ｄ（ＰＬＴ）の利用及びサブジェクト・リンカー・コード１７ｅにより実施される。この意味で、サブジェクト・リンカー・コード１７ｅは、ＰＬＴ　１７ｄを中間制御構造として用い、サブジェクト実行ファイル１７ａにおいて動的にリンクされた関数を処理する。

【００４３】

サブジェクト関数呼び出しは、サブジェクト・リンカー・コード１７ｅへ制御フローを渡すように構成され、サブジェクト・リンカー・コード１７ｅは、サブジェクト関数呼び出しと関連したＰＬＴ　１７ｄにおいてリンク情報を書き換えて、それにより関数を実行するサブジェクト・コードへサブジェクト関数呼び出しをリンクするように構成されている。

【００４４】

サブジェクト・コードがサブジェクト・プロセッサ上でネイティブに実行されている場合、サブジェクト・ライブラリ１７ｂの関数へサブジェクト実行ファイル１７ａの関数呼び出しをリンクするためにＰＬＴ　１７ｄが用いられる手続きは、本明細書の導入部分において記載されており、この手続きは、当業者にとって既知であろう。ソラリスオペレーティング・システムにおけるランタイム・リンク及びＰＬＴの使用に関する更なる情報は、以下において参照することが可能である。

【００４５】

ｈｔｔｐ：／／ｄｏｃｓ．ｓｕｎ．ｃｏｍ／ａｐｐ／ｄｏｃｓ／ｄｏｃ／８１７－１９８４／６ｍｈｍ７ｐｌ１ｂ

また、図３には、リンク検査関数を実行するサブジェクト・コードが示されており、これは、以後リンク・オーディター１７ｆとして参照される。ソラリスオペレーティング・システムにおいて、リンク・オーディター１７ｆは、オペレーティング・システムによって提供されるリンク検査インターフェースと互換性のあるコードを用いることによって、リンカーの動作を監視することが可能である。ソラリスリンク検査インターフェースは、ｒｔｌｄ－ａｕｄｉｔとして知られている。ｒｔｌｄ－ａｕｄｉｔに関する更なる情報は、以下において参照することが可能である。

【００４６】

ｈｔｔｐ：／／ｄｏｃｓ．ｓｕｎ．ｃｏｍ／ａｐｐ／ｄｏｃｓ／ｄｏｃ／８１７－１９

８４／６ｍｈｍ７ｐｌ２４
ｒｔｌｄ－ａｕｄｉｔインターフェースに適合するリンク・オーディターは、実行ファイルに不可欠な要素としてロードされ、そこに含まれる検査ルーチンは、リンカーの実行の種々のステージにおいて、サブジェクト・リンカー・コード１７ｅによって自動的に呼び出される。ｒｔｌｄ－ａｕｄｉｔインターフェースを用いることで、リンク・オーディターが、サブジェクト・コード実行ファイル１７ａ、及びサブジェクト・ライブラリ１７ｄ及び１７ｃのようなロードされたオブジェクトに関する情報にアクセスすることが可能となる。更に、リンク・オーディターは、アプリケーションとライブラリとの間の情報の伝達に関連した他の情報と同様に、そのようなロードされた複数のオブジェクト間において形成される関係に関する情報へアクセスすることが可能である。更なる情報は、以下において参照することが可能である。

【００４７】
ｈｔｔｐ：／／ｄｏｃｓ．ｓｕｎ．ｃｏｍ／ａｐｐ／ｄｏｃｓ／ｄｏｃ／８０６－０６４１／６ｊ９ｖｕｑｕｊｍ
リンク・オーディターは、リンカーにより実行されるＰＬＴ更新の期間中、関数呼び出しの個別の起動、及びのその戻り値を監視することも可能である。

【００４８】
サブジェクト・プロセッサ３上でネイティブに実行されている場合、サブジェクト・ライブラリ１７ｄにおける関数呼び出しは、ＰＬＴ　１７ｄへサブジェクト制御フローを渡し、その後、サブジェクト・リンカー・コード１７ｅへ渡すことになる。サブジェクト・リンカー・コード１７ｅは、サブジェクト関数呼び出しに関連するＰＬＴ　１７ｅに固有のリンク情報を書き換えて、それにより、サブジェクト関数に対するＰＬＴ入力への後続の呼び出しを、サブジェクト・ライブラリ関数１７ｄに固有の関数を実行するための関連するサブジェクト・コードへリンクすることになる。

【００４９】
リンカーの動作を監視するためのリンク・オーディターを用いた関数リンクテーブルの生成
サブジェクト実行ファイル１７ａに対応するターゲット・コード２１を生成する前に、トランスレータ・コード１９は、サブジェクト実行ファイル１７ａのヘッダ１７ａ＿１を参照することによって、及び／又はサブジェクト実行ファイル１７ａの一回又は複数回のスキャンを実行することによって、サブジェクト実行ファイル１７ａに関する情報を収集する。トランスレータ・コード１９は、収集された情報を用いて、サブジェクト実行ファイル１７ａに関連するＰＬＴ　１７ｄを識別する。

【００５０】
トランスレータ・コード１９がサブジェクト実行ファイル１７ａに関する情報を収集した後、トランスレータ・コード１９は初めてサブジェクト実行ファイル１７ａを介して動作し、そのように動作する際、サブジェクト実行ファイル１７ａにおける動的にリンクされた関数呼び出しと、対応する関数を実行するコードとの間のリンクを設定する。

【００５１】
トランスレータ・コード１９がサブジェクト実行ファイル１７ａを介して動作する際、サブジェクト・コード１７ａにおいて発見される、動的にリンクされた関数呼び出しの各々は、トランスレータ・コード１９に保持された関数リンクテーブル（ＦＬＴ）１９ａと照合される。サブジェクト・コード１７ａの最初の変換において予測されるように、ＦＬＴ　１９ａにおいて入力が存在しない関数呼び出しが発生する場合、ＰＬＴ　１７ｄを介してリンカー・コード１７ｅへ、サブジェクト・コード１７の制御フローが進められる。トランスレータ・コード１９は、その後、関数を実行するコードを介して動作を続けることが可能である。

【００５２】
リンカー１７ｅはリンク・オーディター１７ｆへリンク情報を渡し、トランスレータ・コード１９はこの情報を用いてＦＬＴ　１９ａへ入力を加える。特に、トランスレータ・

コード１９は、サブジェクト実行ファイル１７ａにより呼び出される関数の各々の識別子、及びその関数の位置を受信し、かつ格納する。ＦＬＴ　１９ａにおける各入力は、関数識別子及び対応する関数の位置を備える。しかしながら、トランスレータ・コード１９は、リンカー・コード１７ｅによるＰＬＴ領域１７ｄの更新の変換を含むターゲット・コードを生成しない。

【００５３】

上述した例示的な実施形態において、トランスレータ・コード１９は、ＦＬＴ　１９ａを構築する際にｒｔｌｄ－ａｕｄｉｔインターフェースと互換性のあるリンク・オーディター１７ｆを用いる。しかしながら、他の実施例においては、リンカーの動作を識別する別の技術を用い、それによりＦＬＴ　１９ａを入力及び維持しても良い。

【００５４】

関数リンクテーブルの生成　－　リンカー動作の直接監視

さらなる例示的な実施形態においては、ＰＬＴ　１７ｄへの書き換えを検出し、かつサブジェクト実行ファイル１７ａにより呼び出される動的にリンクされたサブジェクト関数を実行するコードの位置を確認するために別の方法を用いても良い。そのような実施形態において、トランスレータ・コード１９は、実行ファイルのファイル形式の情報からＰＬＴを認証しても良い。ＰＬＴ領域の位置についての情報を用いて、トランスレータは、特有のサブジェクト・コード命令シーケンスを検出することによってＰＬＴを書き換えるような、リンカーの動作を識別することができる。ＳＰＡＲＣから変換した例においては、認証されたＰＬＴ領域に影響を与える、特有のサブジェクト命令シーケンスのキャッシュフラッシュ命令が検出されても良い。

【００５５】

サブジェクト・コード命令のキャッシュフラッシュ命令は、ＰＬＴからリンカーへ、サブジェクト・コード中の制御フローが渡される場合、及びリンカーが応答してＰＬＴ入力を書き換える場合に実行される。一度、フラッシュの適正なシーケンスが発生すると、トランスレータは、書き換えられたＰＬＴ領域を読み取り、サブジェクト実行ファイルにより呼び出される関数を実行するコードのアドレスを決定する。このアドレスは、サブジェクト実行ファイルにおいて動的にリンクされた関数呼び出しと、その関数を実行するコードとの間の関連付けを生成することに用いることができる。しかしながら、トランスレータは、ＰＬＴ領域により提供される中間制御構造の書き換えに対応するターゲット・コードの生成は行わない。上述したように、トランスレータは、関数の識別子、及びＦＬＴにおいて関数を実行するコードのアドレスを格納することができる。

【００５６】

関数リンクテーブルにおける情報の利用

標準的なアプリケーションにおいて、トランスレータ・コード１９が、ＦＬＴ入力が形成された、一つ又は複数の動的にリンクされた関数呼び出しを含むサブジェクト・コード１７と遭遇することは非常に起こり得ることである。例えば、トランスレータは、サブジェクト・コードの事前に変換された部分を再び変換することを必要とされ得る。この状況において、対応する入力がＦＬＴ　１９ａに存在する関数呼び出しが遭遇される。トランスレータ・コード１９は、ＦＬＴ　１９ａに事前に記録された情報を用いて、サブジェクト・コードにおける関数呼び出しと、その関数を実行する、対応するコードとの間の関連付けを設定する。この関連付けは、便宜的に直接リンク形式としても良い。

【００５７】

特定の関数呼び出しの再変換は、ＰＬＴ　１７ｄ及びリンカー・コード１７ｅの両方をバイパスし、トランスレータ・コード１９が、動的にリンクされた関数呼び出しを、その関数を実行する関連するコードに効果的に関連付けることを可能とする。すなわち、サブジェクト・コードにおいて関数呼び出しを、ＦＬＴを用いてその関数を実行する対応するコードに関連付けることは、サブジェクト・コードにおいて中間制御構造に対応するターゲット・コードを生成することを含まない。当該関数呼び出しに関連するＰＬＴ１７ｄに固有のリンク情報を書き換えるサブジェクト・リンカー・コード１７ｅは、同様に変換さ

れない。再変換の間、サブジェクト・リンカー・コード１７ｅは、サブジェクト・コードにおいてバイパスされ、従ってトランスレータ・コード１９によってアクセスされない。

【００５８】

上述のように、トランスレータ・コード１９は第一の時間にサブジェクト・コードの一部分を介して動作しつつ、ＦＬＴ　１９ａに入力する。後続の再変換は、トランスレータ・コード１９がリンカー・コード１７ｅをバイパスし、ＦＬＴ　１９ａの情報を利用するので、最初の変換よりも効率的となり得る。再変換は、トランスレータがＦＬＴ　１９ａに入力する動作を行うことを必要としない。

【００５９】

間接関数呼び出し

トランスレータが遭遇する動的にリンクされた関数呼び出しの一部は、間接的な関数呼び出しであることがある。間接関数呼び出しは、変数値に依存する呼び出しである。変数値は、呼び出される位置を決定し、変動し得る。従って、トランスレータは、間接関数呼び出しに対応するターゲット・コードが実行されるポイントにおいて、必要とされる関数を実行するコードの位置を決定することのみが可能である。

【００６０】

図１に関連して上述したように、トランスレータが関数呼び出しに遭遇する場合、トランスレータは新しい基本ブロックを設定する。トランスレータは、ＦＬＴに格納された情報と、間接的関数呼び出しによって参照されるサブジェクト・アドレスとを照合することが可能である。ＦＬＴに既にアドレスが存在する場合、トランスレータは、間接的関数呼び出しが現在参照している関数への呼び出しと事前に遭遇していた必要がある。トランスレータは、その結果、同じ機能を実行するようにその関数呼び出しと関連して事前に設定された基本ブロックを、新規に設定された基本ブロックとして扱うことが可能である。トランスレータが、新規に設定された基本ブロックの代わりに、事前に設定された基本ブロックを使用することを可能とすることにより、実行するために必要とされるトランスレータの仕事量を更に減少させることが可能である。

【００６１】

サブジェクト・コード以外のコードと、サブジェクト・コードにおいて動的にリンクされた関数呼び出しとの関連付け

上述した実施形態において、ＦＬＴに保持される位置は、便宜的に、対応する関数を実行するサブジェクト・コード命令のアドレスであった。しかしながら、本発明の別の実施形態においては、ＦＬＴ入力は、対応する関数を実行する他のコードを指し示すＦＬＴにおけるアドレスとして、トランスレータ・コード１９により選択されても良い。この方法でＦＬＴを用いることにより、トランスレータはその作業負荷を減少させることが可能である。ＦＬＴ入力は、

事前に変換されたターゲット・コード、ターゲット・オペレーティング・システム２０のネイティブ・ライブラリ２８における関数、

トランスレータに知られ、かつターゲット・コードが便宜的に生成される最適化されたＩＲの一部分、または

サブジェクト・コードの位置または上記の中の任意の一つを示す、またはそれを生成するのに用いられる、デリファレンスされる変数

の中の任意の一つを指し示すように、トランスレータによって選択されても良い。

【００６２】

上述したような、デリファレンスされる変数の使用は、トランスレータが、関数呼び出しと、対応する関数を実行するコードとの間の関連付けの制御をより一層働かせることを許可する。このことは、特定の関数を実行するコードの位置が、サブジェクト・プログラムの現在の呼び出しの存続中、固定されない場合に望ましい。

【００６３】

説明のための例において、トランスレータ・コード１９は、ＳＰＡＲＣからｘ８６への変換を実行するように構成されている。ＳＰＡＲＣターゲットシステムライブラリは、そ

の中に、特定の引数における関数の実行の結果が厳密に定義されている一つ又は複数の関数を実行するルーチンを含み得る。そのように厳密に定義された関数は、ＡＢＩ「ドット」関数として知られており、例えば、．ｕｍｕｌ、．ｓｍｕｌ等の計算関数の領域を含む。これらのＡＢＩドット関数は、以下に示されている。

【００６４】

ｈｔｔｐ：／／ｗｗｗ．ｓｐａｒｃ．ｃｏｍ／ｓｔａｎｄａｒｄｓ／ｐｓＡＢＩ３ｒｄ．ｐｄｆ

ＡＢＩドット関数の動作は厳密に定義されているので、トランスレータは、ＡＢＩドット関数への呼び出しとして特定される関数呼び出しを、ＡＢＩドット関数と同じ効果を有する単純な命令として扱うことが可能である。このことは、例えば、ＦＬＴ入力を設定し、かつ対応する関数を実行する非サブジェクト・コードの一部分と関数呼び出しを直接的に関連付けることによって、トランスレータがＰＬＴを完全にバイパスすることを可能とする。この方法でＰＬＴをバイパスすることは、トランスレータの作業負荷を減少させる。更に、この例において、トランスレータは、関数呼び出し自体の代わりに、関連するＡＢＩドット関数の効果を有するコードを付加することによって、関数呼び出しを扱うことに関連する仕事を完全に回避することが可能である。

【００６５】

ネイティブの関数ライブラリとの既知の対応を有するサブジェクト関数への呼び出しは、トランスレータ・コード１９により識別されることが可能であり、ネイティブの関数との対応は、トランスレータの作業負荷を減少させるように利用される。例えば、サブジェクト・ライブラリ１７ｃにおけるｍｅｍｃｐｙへの呼び出しは、ＦＬＴ　１９ａにおいて、ネイティブ・ライブラリ２８におけるネイティブのｘ８６に相当するものの位置に関連付けられることが可能である。このことは、ｍｅｍｃｐｙ関数のサブジェクト（ＳＰＡＲＣ）バージョンの変換コストを削減する。加えて、ｍｅｍｃｐｙ関数のネイティブの（ｘ８６）バージョンは、ネイティブのハードウェアの複雑さに適応し、そのハードウェアに対する最も効率的な方法でその関数の望ましい効果を得ることが可能である。

【００６６】

図４は、動的にリンクされた関数呼び出しを行う方法の例示的な実施形態を説明するフロー概略図である。サブジェクト・コードは、受信され、読み取られてＰＬＴ領域を識別する（ステップ１０１）。動的にリンクされた関数呼び出しは、サブジェクト・コードにおいて識別され（ステップ１０２）、動的にリンクされた関数呼び出しは、ＰＬＴにおいてサブジェクト関数呼び出しに関連付けられたリンク情報を書き換えるサブジェクト・リンカー・コードへ、サブジェクト制御フローを渡し、それにより、その関数を実行するサブジェクト・コードへサブジェクト関数呼び出しをリンクするように構成される。ＦＬＴは識別された関数呼び出しに対応する入力について照合を行い（ステップ１０３）、ＦＬＴが関連する入力を含む場合は、本方法はステップ１０７へ進み、ステップ１０７においては、ターゲット・コードが生成され、そのターゲット・コードにおいては、関数呼び出しが、その関数を実行するコードに関連付けられている。ステップ１０７において設定された関連付けは、ＦＬＴに格納されているような関数識別子及び位置に基づくものである。生成されたターゲット・コードは、ＰＬＴにより提供される中間制御構造に対応するターゲット・コードを含まない。

【００６７】

ステップ１０３において、ＦＬＴが、識別された関数呼び出しに対する関連する入力を含んでいない場合、リンク・オーディターは関数識別子及び位置を得るように動作することができ（ステップ１０４）、及び／又は関数識別子及び位置はサブジェクト・コード命令のキャッシュフラッシュを監視することにより得ることができる（ステップ１０５）。ステップ１０４、及び／又はステップ１０５において得られる情報は、その後ＦＬＴへ入力される（ステップ１０６）。その後、ステップ１０４、及び／又はステップ１０５において得られた情報に基づいて、ステップ１０７が実行される。

【００６８】

上記に詳細に記載された例示的な実施形態において、サブジェクト・リンカー・コードはトランスレータにアクセス可能なサブジェクト・コードの本文に存在する。しかしながら、他の実施形態においては、サブジェクト・コードにおいてリンクを実行するリンカーの機能は、特に、図３に示されたリンカー・コード１９ｅのような、ターゲット・プロセッサ上での実行のために記述されたターゲット・コードを用いるトランスレータによって提供されても良い。更に、例示的実施形態はサブジェクト実行ファイルに関連したＰＬＴに焦点を合わせたものであったが、サブジェクト・ライブラリ１７ｂ、１７ｃは、それら自身のＰＬＴ領域の形式において、中間制御構造も含むことがある。本明細書に記載された方法及び装置は、これらのＰＬＴ領域や、中間制御構造を含み、かつ対応する問題を生じさせるコードの、別の識別可能な一部分に対して等しく適用されることが可能である。

【００６９】

本明細書に記載された技術を用いることによって、トランスレータ・コード１９は、ＰＬＴやそれに類するような、中間制御構造における間接リンク情報の更新に対応する変換を生成するために、サブジェクト・コードの制御フローに従う必要がなく、従って、ランタイムにおいて書き換えられるコードの多数の集中を含むサブジェクト・コードの一部の変換のプロセッシング及びメモリコストを避けることができる。

【００７０】

加えて、トランスレータが、間接リンク情報の更新を含むサブジェクト・コードの実行を行う状況に対処する効率的な方法が説明された。

特に、本発明は、プログラムコード変換を実行するコンピュータシステムにおいて有用な方法及びユニットを開発した。そのような方法及びユニットは、サブジェクト・プログラム・コードのターゲット・コードへの動的バイナリ変換を提供するランタイムトランスレータのように構成されたコンピュータシステムに関連して特に有用である。

【００７１】

本発明は、本明細書に明示された任意の方法を実行するように構成されたトランスレータにまで及ぶ。同様に、本発明は、本明細書に明示された任意の方法を実行するコンピュータによって実装可能な命令を記録した、コンピュータが読み取り可能な記録メディアにまで及ぶ。

【００７２】

少なくとも、本発明のいくつかの実施形態は、専用のハードウェアを用いて単独に構成されても良く、本明細書で用いられる「モジュール」または「ユニット」のような単語は、それに限られるわけではないが、特定のタスクを実行するフィールド・プログラマブル・ゲート・アレイ（ＦＰＧＡ）、又は特定用途向け集積回路（ＡＳＩＣ）のような、ハードウェアデバイスを含んでも良い。あるいは、本発明の要素は、アドレス可能な記録メディアに存在するように構成されても良く、一つ又は複数のプロセッサ上で実行されるように構成されても良い。従って、本発明の機能的要素は、ある実施形態においては、例として、ソフトウェア要素、オブジェクト指向ソフトウェア要素、クラス要素及びタスク要素、プロセス、機能、属性、手続き、サブルーチン、プログラムコードのセグメント、ドライバ、ファームウェア、マイクロコード、回路構成、データ、データベース、データ構造、テーブル、アレイ、及び変数のような要素を含んでも良い。更に、好適な実施形態が、以下に説明される要素、モジュール及びユニットを参照して説明されるが、このような機能的要素はより少ない要素に合成されても良く、又は追加的要素に分割されても良い。

【００７３】

本発明の装置及び方法の種々の特徴は、上記した実施形態の各々において別々に説明されている。しかしながら、本明細書に記載された各実施形態の別々の側面は、本明細書において記載された他の実施形態に結合され得るということが、本発明の発明者の完全なる意図である。

【００７４】

記載された通りの好適な実施形態の種々の適応と変更が、本発明の範囲と技術思想から逸脱することなく、構成されることが可能であることを当業者は理解するであろう。従っ

て、本発明は、添付の特許請求の範囲の範囲内においてならば、本明細書に明確に記載されているようにではなく実施されても良いことが理解されたい。

【００７５】

いくつかの好適な実施形態が示され、説明されたが、当業者であれば、本発明の範囲を逸脱することなく、種々の変形や修正が、添付の特許請求の範囲に明示されるように、行われ得ることを理解するであろう。

【００７６】

同時に提出された、又は本願と関連したこの明細書に先立つ全ての文書、及びこの明細書を公衆の閲覧のために公開した文書に対して注意が向けられ、全てのそのような文書の内容が、参照として本明細書に組み入れられる。

【００７７】

本明細書（任意の添付の特許請求の範囲、要約、及び図面を含む）で開示される全ての特徴、及び／又はそのように開示された任意の方法又はプロセスの全てのステップは、そのような特徴及び／又はステップの少なくともいくつかが相互に排他的である組み合わせを除いて、任意の組み合わせで合成され得る。

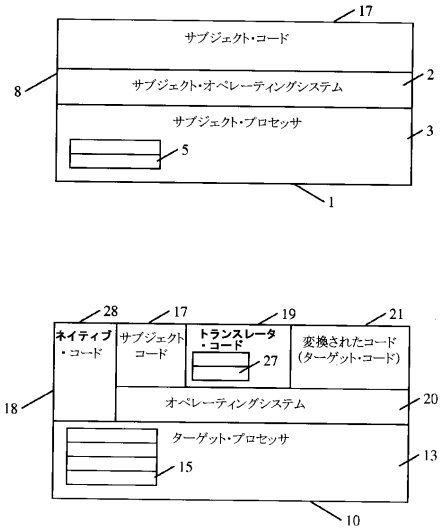
【００７８】

本明細書（任意の添付の特許請求の範囲、要約、及び図面を含む）において開示された各特徴は、明示的に別の方法で述べられていない限り、同じ、均等な、又は同様の目的で扱う別の特徴により置き換えられ得る。従って、明示的に別の方法で述べられていない限り、開示された各特長は、等価な、又は類似の特徴の一般的なシリーズの一つの例に過ぎない。

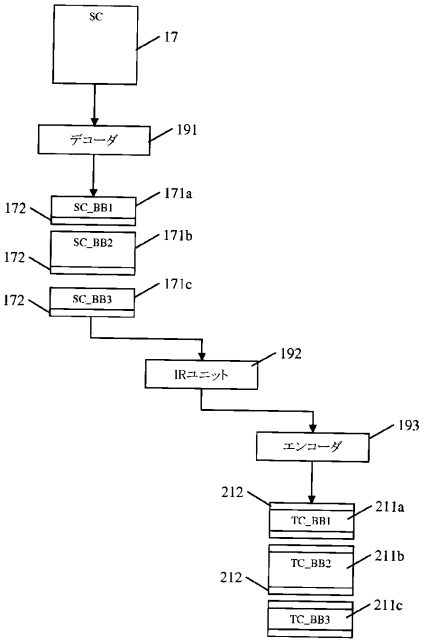
【００７９】

本発明は、前述の実施形態の詳細に制限されない。本発明は、本明細書（任意の添付の特許請求の範囲、要約、及び図面を含む）において開示された特徴の、任意の新規なもの、又は任意の新規な組み合わせ、又は、そのように開示された任意の方法又はプロセスのステップの任意の新規なもの、又は任意の新規な組み合わせにまで及ぶ。

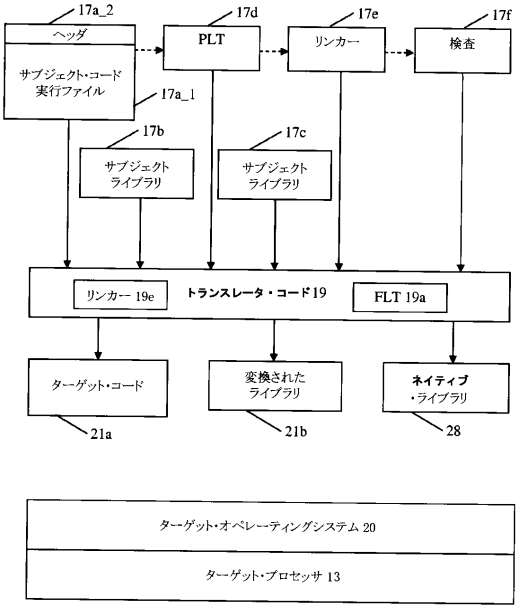
【図１】


17
サブジェクト・コード
8
サブジェクト・オペレーティングシステム
2
サブジェクト・プロセッサ
3
5
1
28
17
19
21
ネイティブ・コード
サブジェクトコード
トランスレータ・コード
27
変換されたコード(ターゲット・コード)
18
オペレーティングシステム
20
ターゲット・プロセッサ
15
13
10

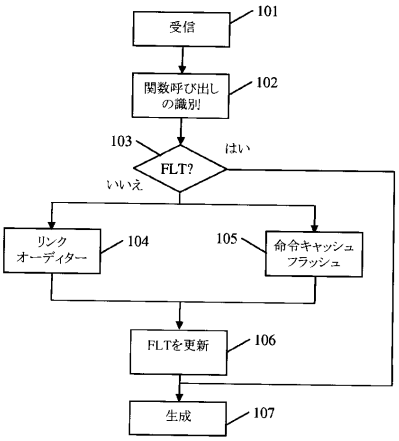
【図２】


SC
17
デコーダ
191
172
SC_BB1
171a
172
SC_BB2
171b
172
SC_BB3
171c
IRユニット
192
エンコーダ
193
212
TC_BB1
211a
TC_BB2
211b
212
TC_BB3
211c

【図３】


17a_2
ヘッダ
サブジェクト・コード実行ファイル
17a_1
17d
PLT
17e
リンカー
17f
検査
17b
サブジェクトライブラリ
17c
サブジェクトライブラリ
リンカー 19e
トランスレータ・コード19
FLT 19a
ターゲット・コード
21a
変換されたライブラリ
21b
ネイティブ・ライブラリ
28
ターゲット・オペレーティングシステム 20
ターゲット・プロセッサ 13

【図４】


受信
101
関数呼び出しの識別
102
103
FLT?
はい
いいえ
リンクオーディター
104
105
命令キャッシュフラッシュ
FLTを更新
106
生成
107

フロントページの続き

(72)発明者　ブラウン、アレクサンダー　バラクラフ
イギリス国　Ｓ１０　２ＰＬ　ヨークシャー　シェフィールド　ブルームヒル　ニューボールド　レーン　１２５

審査官　坂庭　剛史

(56)参考文献　国際公開第２００５／００８４７８（WO, Ａ１）
特開平１０－３１２２９０（ＪＰ, Ａ）
特開昭５９－１１９４４７（ＪＰ, Ａ）
特開平１０－２２２３７８（ＪＰ, Ａ）
国際公開第２００６／０９５１５５（WO, Ａ１）
特集2 フリー・エミュレータの活用に挑む！ Ｐａｒｔ２ 仮想Ｗｉｎｄｏｗｓ！ Ｗｉｎｅ, Ｌｉｎｕｘ WORLD, 日本, 株式会社アイ・ディ・ジー・ジャパン, ２００２年１０月　1日, 第1巻, 第１０号（通巻１１号）, ｐｐ. １１４～１１６

(58)調査した分野(Int.Cl., ＤＢ名)
G06F　9/455