(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2011-530768
(P2011-530768A)
(43)公表日　平成23年12月22日(2011.12.22)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 9/45	(2006.01)	G06F	9/44	322F	5B081

審査請求 未請求　予備審査請求 未請求　（全 18 頁）

(21)出願番号　特願2011-522999 (P2011-522999)
(86)(22)出願日　平成21年8月13日 (2009.8.13)
(85)翻訳文提出日　平成23年4月11日 (2011.4.11)
(86)国際出願番号　PCT/US2009/004649
(87)国際公開番号　WO2010/019254
(87)国際公開日　平成22年2月18日 (2010.2.18)
(31)優先権主張番号　61/188,905
(32)優先日　平成20年8月13日 (2008.8.13)
(33)優先権主張国　米国 (US)

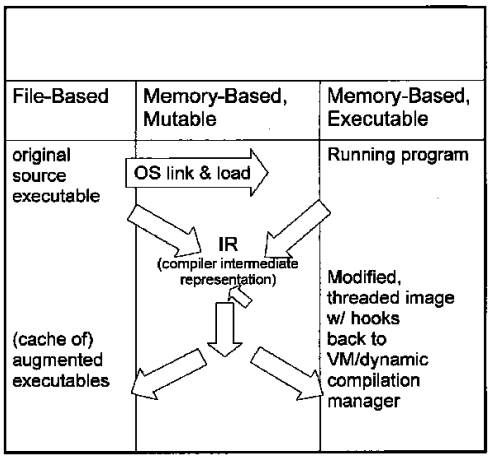
(71)出願人　511036749
トランセラ・インコーポレーテッド
ＴＲＡＮＳＣＥＬＬＡ　ＩＮＣＯＲＰＯＲＡＴＥＤ
アメリカ合衆国　カリフォルニア州９４３０６　パロ・アルト，ケンブリッジ・アベニュー，４４５，スイート　ビー
(74)代理人　110000028
特許業務法人明成国際特許事務所
(72)発明者　ドライアー・ロバート・スコット
アメリカ合衆国　カリフォルニア州９４３０６　パロ・アルト，ケンブリッジ・アベニュー，４４５，スイート　ビー

最終頁に続く

(54)【発明の名称】ソフトウェア・アプリケーションの性能向上

(57)【要約】
【課題】
【解決手段】入力コードから並列に実行可能なコードを生成する処理は、入力コードを静的に解析することにより、入力コードのデータフローおよび制御フローのアスペクトを決定する工程と、入力コードを動的に解析することにより、入力コードのデータフローおよび制御フローの追加のアスペクトを決定する工程と、静的解析によって決定された入力コードのデータフローおよび制御フローのアスペクトと動的解析によって決定された入力コードのデータフローおよび制御フローの追加のアスペクトとに少なくとも部分的に基づいて、入力コードの中間表現を生成する工程と、中間表現を処理して、並列実行が可能な中間表現の部分を決定する工程と、処理された中間表現から並列実行可能なコードを生成する工程と、を備える。
【選択図】図１０

FIG. 10



【特許請求の範囲】
【請求項１】
入力コードから並列実行可能なコードを生成するシステムであって、
１つ以上のプロセッサであって、
前記入力コードを静的に解析することにより、前記入力コードのデータフローおよび制御フローのアスペクトを決定し、
前記入力コードを動的に解析することにより、前記入力コードのデータフローおよび制御フローの追加のアスペクトを決定し、
前記静的解析によって決定された前記入力コードのデータフローおよび制御フローの前記アスペクトと、前記動的解析によって決定された前記入力コードのデータフローおよび制御フローの前記追加のアスペクトと、に少なくとも部分的に基づいて、前記入力コードの中間表現を生成し、
前記中間表現を処理して、並列実行が可能な前記中間表現の部分を決定し、
並列実行可能なコードを生成する、
ように構成される１つ以上のプロセッサと、
前記１つ以上のプロセッサに接続される１つ以上のメモリであって、前記１つ以上のプロセッサに命令を与えるように構成される１つ以上のメモリと、
を備えるシステム。
【請求項２】
請求項１に記載のシステムであって、
前記入力コードがバイナリコードである、システム。
【請求項３】
請求項１に記載のシステムであって、
前記静的解析には、前記入力コードを逆アセンブルして、前記入力コードを静的に生成される中間表現に変換する処理が含まれる、システム。
【請求項４】
請求項１に記載のシステムであって、
前記静的解析には、並列実行が可能な静的中間表現の部分を特定する処理が含まれる、システム。
【請求項５】
請求項１に記載のシステムであって、
前記静的解析には、インスツルメンテーションコードを挿入する処理が含まれる、システム。
【請求項６】
請求項１に記載のシステムであって、
前記静的解析には、インスツルメンテーションコードを挿入する処理が含まれ、
前記動的解析には、前記インスツルメンテーションコードを実行する処理が含まれる、システム。
【請求項７】
請求項１に記載のシステムであって、
前記動的解析には、プロファイル情報を決定する処理が含まれる、システム。
【請求項８】
請求項１に記載のシステムであって、
前記動的解析には、ホットスポット情報を決定する処理が含まれる、システム。
【請求項９】
請求項１に記載のシステムであって、
前記動的解析には、分岐ターゲット情報を決定する処理が含まれる、システム。
【請求項１０】
請求項１に記載のシステムであって、
前記動的解析には、メモリ・エイリアス情報を決定する処理が含まれる、システム。

【請求項１１】
請求項１に記載のシステムであって、
前記動的解析には、動的ループ回数情報を決定する処理が含まれる、システム。
【請求項１２】
請求項１に記載のシステムであって、さらに、
並列部分を含むように前記中間表現を修正し、
修正された中間表現に基づいてアクセラレーション・コードを生成することを含む、システム。
【請求項１３】
請求項１に記載のシステムであって、
並列実行が可能な前記中間表現の部分が投機的に決定される、システム。
【請求項１４】
請求項１に記載のシステムであって、
並列実行が可能な前記中間表現の部分が投機的に決定され、検証コードが挿入される、システム。
【請求項１５】
入力コードから並列実行可能なコードを生成する方法であって、
前記入力コードを静的に解析することにより、前記入力コードのデータフローおよび制御フローのアスペクトを決定する工程と、
前記入力コードを動的に解析することにより、前記入力コードのデータフローおよび制御フローの追加のアスペクトを決定する工程と、
前記静的解析によって決定された前記入力コードのデータフローおよび制御フローの前記アスペクトと、前記動的解析によって決定された前記入力コードのデータフローおよび制御フローの前記追加のアスペクトと、に少なくとも部分的に基づいて、前記入力コードの中間表現を生成する工程と、
前記中間表現を処理して、並列実行が可能な前記中間表現の部分を決定する工程と、を備える方法。
【請求項１６】
入力コードから並列実行可能なコードを生成するためのコンピュータプログラム製品であって、
コンピュータ読み取り可能な記憶媒体に具現化され、
複数のコンピュータ命令であって、
前記入力コードを静的に解析することにより、前記入力コードのデータフローおよび制御フローのアスペクトを決定するコンピュータ命令と、
前記入力コードを動的に解析することにより、前記入力コードのデータフローおよび制御フローの追加のアスペクトを決定するコンピュータ命令と、
前記静的解析によって決定された前記入力コードのデータフローおよび制御フローの前記アスペクトと、前記動的解析によって決定された前記入力コードのデータフローおよび制御フローの前記追加のアスペクトと、に少なくとも部分的に基づいて、前記入力コードの中間表現を生成するコンピュータ命令と、
前記中間表現を処理して、並列実行が可能な前記中間表現の部分を決定するコンピュータ命令と、を備える、コンピュータプログラム製品。
【請求項１７】
入力コードから並列実行可能なコードを生成するシステムであって、
１つ以上のプロセッサであって、
前記入力コードを静的に解析することにより、前記入力コードのデータフローおよび制御フローのアスペクトを決定し、
データフローおよび制御フローの前記アスペクトに少なくとも部分的に基づいて、前記入力コードの中間表現を生成し、
前記中間表現を処理して、並列実行可能なコードを生成し、

前記並列実行可能なコードを格納し、
前記入力コードの実行要求に応じて、前記格納された並列実行可能なコードを実行する、
ように構成される１つ以上のプロセッサと、
前記１つ以上のプロセッサに接続される１つ以上のメモリであって、前記１つ以上のプロセッサに命令を与えるように構成される１つ以上のメモリと、
を備えるシステム。
【請求項１８】
入力コードから並列実行可能なコードを生成する方法であって、
前記入力コードを静的に解析することにより、前記入力コードのデータフローおよび制御フローのアスペクトを決定する工程と、
データフローおよび制御フローの前記アスペクトに少なくとも部分的に基づいて、前記入力コードの中間表現を生成する工程と、
前記中間表現を処理して、並列実行可能なコードを生成する工程と、
前記並列実行可能なコードを格納する工程と、
前記入力コードの実行要求に応じて、前記格納された並列実行可能なコードを実行する工程と、
を備える方法。
【請求項１９】
入力コードから並列実行可能なコードを生成するためのコンピュータプログラム製品であって、
コンピュータ読み取り可能な記憶媒体に具現化され、
複数のコンピュータ命令であって、
前記入力コードを静的に解析することにより、前記入力コードのデータフローおよび制御フローのアスペクトを決定するコンピュータ命令と、
データフローおよび制御フローの前記アスペクトに少なくとも部分的に基づいて、前記入力コードの中間表現を生成するコンピュータ命令と、
前記中間表現を処理して、並列実行可能なコードを生成するコンピュータ命令と、
前記並列実行可能なコードを格納するコンピュータ命令と、
前記入力コードの実行要求に応じて、前記格納された並列実行可能なコードを実行するコンピュータ命令と、を備える、
コンピュータプログラム製品。
【発明の詳細な説明】
【技術分野】
【０００１】
[クロスリファレンス]
本出願は、「ソフトウェア・アプリケーションの性能を向上させるための方法および装置（Ｍｅｔｈｏｄ ａｎｄ Ａｐｐａｒａｔｕｓ ｔｏ Ｅｎｈａｎｃｅ ｔｈｅ Ｐｅｒｆｏｒｍａｎｃｅ ｏｆ Ｓｏｆｔｗａｒｅ Ａｐｐｌｉｃａｔｉｏｎｓ）」の名称で２００８年８月１３日に出願された米国仮出願番号第６１／１８８，９０５号に基づく優先権を主張するものである。前記出願の内容は、参照することによりその全体が本明細書に組み込まれる。
【背景技術】
【０００２】
コンピュータシステムにおいて、ソフトウェアの性能および能力が向上すれば、新しいアプリケーションや機能および改良されたアプリケーションや機能が可能になる。より強力なソフトウェアはより強力なハードウェアを必要とし、より強力なハードウェアがあればより強力なソフトウェアが可能になるという相乗作用が、過去数十年の情報革命を推進してきた。「インストラクションセット（命令セット）対応」コンピュータ処理の歴史において、コンピュータハードウェアの世代が新しくなるにつれ、既存のアプリケーション

が実質的に改善され、一方、新しいアプリケーションや再コンパイルしたアプリケーションにより、ハードウェアの能力以上の改良がなされてきた。

【０００３】

しかしながら、コンピュータのハードウェア・アーキテクチャ技術が成熟するにつれて、マイクロプロセッサの供給元にとって、マイクロアーキテクチャ技術によるこれ以上の性能向上はますます困難になってきた。コンピュータスレッド内で命令レベルの並列性を向上させる費用対効果が高い手法は限界に達しているため、産業界の多くは、スレッドレベルの並列性を向上させることが、終わりのない性能向上を続けるための最適な技術と考えている。したがって、メーカーは、１つの半導体「チップ」内に複数のプロセッサを備える「マルチコア」ＣＰＵの生産を開始した。これに合わせて、主要なソフトウェア開発者たちは、マルチコア・プロセッサの潜在性能を利用するためのアプリケーション書き換えを求めている。

【０００４】

これらの開発の結果、新世代のコンピュータハードウェアを用いても、既存のアプリケーションでは、実質的な性能向上が見込めない場合が多い。最新のマルチコア・プロセッサを特に対象としたアプリケーション書き換え以外では、大きな性能向上は見込めない。また、マルチスレッド・アプリケーションをプログラミングして、マルチプロセッサ・アーキテクチャを利用する方法は、多くの場合、複雑でエラーを招きやすい。

【０００５】

以下、本発明のさまざまな実施例を添付の図面を参照して詳述する。

【図面の簡単な説明】

【０００６】

【図１】マルチプロセッサ・ハードウェアシステムの実施例を示すブロック図。

【０００７】

【図２】アプリケーション性能を向上させたソフトウェアシステムの実施例を示すブロック図。

【０００８】

【図３Ａ】性能を向上させていないマルチプロセッサ・ハードウェアシステムにおけるＣＰＵへのアプリケーションのマッピングを示す図。

【０００９】

【図３Ｂ】性能を向上させたマルチプロセッサ・ハードウェアシステムにおけるＣＰＵへのアプリケーションのマッピングを示す図。

【００１０】

【図４】入力コードから並列実行可能なコードを生成するプロセスの１つの実施例を示すフローチャート。

【００１１】

【図５】入力コードから並列実行可能なコードを生成するプロセスの別の実施例を示すフローチャート。

【００１２】

【図６】静的解析プロセスの実施例を示すフローチャート。

【００１３】

【図７】動的解析プロセスの実施例を示すフローチャート。

【００１４】

【図８】図４および図５に示すプロセスを実行する性能を向上させたシステムの実施例を示すブロック図。

【００１５】

【図９】中間表現の例を示す図。

【００１６】

【図１０】プログラム表現のいくつかの例とそれらの相互関係を示す図。

【発明を実施するための形態】

【００１７】
本発明は、プロセス、装置、システム、物質構成、コンピュータ読み取り可能な記憶媒体上で具現化されるコンピュータプログラム製品、および／または、プロセッサ、たとえば、プロセッサに接続されるメモリ上に記憶される、および／または、メモリにより提供される命令を実行するように構成されるプロセッサ等、種々の態様で実現可能である。本明細書において、これらの実施態様やその他本発明が具現化可能なすべての態様を「手法」と称する。開示されているプロセスにおける各ステップの順序は、本発明の要旨の範囲内で変更可能である。特に断らない限り、あるタスクを実行するように構成されるものして説明されるプロセッサやメモリ等のコンポーネント（部品）は、所定の時間でタスクを実行するように一時的に構成される一般的なコンポーネント、または、タスクを実行するために製造された特定のコンポーネントとして実現可能である。本明細書で用いる「プロセッサ」という用語は、コンピュータプログラム命令等のデータを処理するように構成される１つ以上のデバイス、回路、および／または、処理コアを意味する。

【００１８】
本発明の１つまたは複数の実施形態を、本発明の原理を例示する添付の図面を参照して、以下に詳述する。これらの実施例に基づいて本発明を説明するが、本発明は何らこれらの実施例に限定されるものではない。本発明の範囲は、特許請求の範囲によってのみ限定されるものであり、本発明は、さまざまな変形・変更および等価の形態を網羅するものである。以下、本発明の理解を助ける目的で、種々の特定の詳細に基づいて本発明を説明するが、これらの詳細は例示に過ぎず、これらの詳細の一部または全部がなくても、特許請求の範囲に従って発明を実現可能である。本発明の特徴を明確にするために、本発明に関連する技術分野で周知の技術に関しては詳細を説明しない。

【００１９】
入力コードから並列実行可能なコードを生成する手法を開示する。入力コードを静的に解析して（statically analyzed）、入力コードのデータフローおよび制御フローのアスペクト（aspects）を決定する。一部の実施形態において、入力コードをさらに動的に解析して（dynamically analyzed）、データフローおよび制御フローの追加のアスペクト（additional aspects）を決定するようにしてもよい。決定されたアスペクトに少なくとも部分的に基づいて、中間表現ＩＲ（Intermediate Representation）を生成する。中間表現ＩＲを処理し、並列実行が可能な中間表現ＩＲの部分を特定する。一部の実施形態において、特定された部分を並列に実行するように構成される並列コードを生成して、キャッシュに格納するようにしてもよい。元の入力コードを後で呼び出す場合には、代わりにキャッシュに格納したコードが呼び出される。

【００２０】
図１は、マルチプロセッサ・ハードウェアシステムの実施例を示すブロック図である。実施例のシステム１００は、１つまたは複数のマルチコア中央処理装置（ＣＰＵ）１０２と、１つまたは複数の動的メモリ１０４と、１つまたは複数のネットワークカード、ディスプレイ、キーボード等の入力／出力（Ｉ／Ｏ）装置１０６と、１つまたは複数のディスク記憶装置やフラッシュメモリ等の不揮発性記憶装置１０８と、を備える。各コンポーネントは、相互に接続されている。

【００２１】
図２は、アプリケーション性能を向上させたソフトウェアシステムの実施例を示すブロック図である。この実施例では、アプリケーション性能エンハンサー２０４（「アクセラレータ」とも称する）は、アプリケーション２０２の実行ファイルを処理して、アプリケーションの性能を向上するように構成される。後で詳述するように、アプリケーション性能エンハンサーは、実行ファイルを中間表現に変換して、静的解析および動的解析を用いて、並列に実行可能な中間表現の部分を特定する。

【００２２】
図３Ａは、性能を向上させていないマルチプロセッサ・ハードウェアシステムにおけるＣＰＵへのアプリケーションのマッピングを示す図である。図示した例では、アプリケー

ション３０２は、シングルスレッドであり、１つのＣＰＵ上で実行される。この間、他のＣＰＵコアは用いられていない。アプリケーションがマルチスレッドの場合でも、通常、性能を最大にする方法ではなく、プログラマーにとって最も簡単な方法で、スレッド処理が行なわれる。言い換えると、アプリケーションは、多くの場合、性能を最大にするために複数のスレッドでバランスを取るというよりも、プログラミングを容易にするように分割される。たとえば、ウィンドウズ（登録商標）グラフィカルユーザーインターフェース（ＧＵＩ）を用いるアプリケーションは、多くの場合、主アプリケーション用のスレッドと、ＧＵＩ管理用の別のスレッドに分割される。この形式のマルチスレッドは、主処理スレッドの性能を最適化するものではない。さらに、最新のタイムスライス型マルチタスクーマルチプロセッサ環境下では、複数のアプリケーションを別々のコアで同時に実行可能である。各アプリケーションと対応する実行ＣＰＵとの間の関係は、通常、図３Ａに示す１対１の関係である。この関係では、複数のＣＰＵの処理能力を最大限に利用することにはならない。

【００２３】

図３Ｂは、性能を向上させたマルチプロセッサ・ハードウェアシステムにおけるＣＰＵへのアプリケーションのマッピングを示す図である。アプリケーションの主スレッドを並列コンピューティング（並列計算）のために解析する。別々のＣＰＵ上でマルチスレッドで実行できるように、部分ｉ、部分ｉｉ、・・・部分ｍ等、並列コンピューティングが可能と考えられる部分を再構成することにより、高度な並列化と性能向上が達成できる。

【００２４】

図４は、入力コードから並列実行可能なコードを生成するプロセスの１つの実施例を示すフローチャートである。さまざまな実施形態において、プロセス４００は、エンドユーザのコンピュータで実行されるものでもよく、あるいは、開発者のシステム上でソフトウェア開発時に実行されるものでもよい。また、ソースコード、アセンブリコードおよび／またはバイナリコードを入力コードとして利用できる。ステップ４０２で、入力コードを静的に解析する。本明細書において、「静的解析」は、解析の最中にコードを実行する必要がないコード解析を意味する。システム資源に対する需要が低い時に静的解析をオフラインで実行するようにしてもよい。一部の実施形態において、静的解析プロセスで、入力コードの逆アセンブリを行なって、データフローと制御フローの命令、演算およびアスペクトを含む入力コードのデータ部分とコード部分とを識別するようにしてもよい。識別される情報には、データセグメントに含まれるデータ、コードセグメントに含まれるコード、データ処理に利用されるヘッダ情報が含まれる。

【００２５】

静的解析では、コードの挙動に依存する情報等、データフローおよび制御フローの所定のアスペクトを取得することが難しい場合がある。したがって、ステップ４０４で、入力コードを動的に解析して、データフローと制御フローの追加のアスペクトを決定する。本明細書において、「動的解析」は、コードを実行しながら実施するオンラインまたはランタイム解析を意味する。さまざまな実施形態において、実行周波数、ホットスポットおよびその他のプロファイル情報、分岐ターゲット(branch target：分岐先)、メモリ・エイリアス情報（memory alias information）、および動的ループ回数等が、動的解析によって決定されるアスペクトの例である。ランタイム（実行時）におけるコードの解釈またはエミュレーションを介して、および／または、インスツルメンテーション後のコード（instrumented code）を実行することにより、情報を収集するようにしてもよい。

【００２６】

ステップ４０６で、静的解析により決定されたアスペクトおよび動的解析により決定された追加のアスペクトに基づき、入力コードの中間表現ＩＲを生成する。ステップ４０８で、中間表現ＩＲをさらに処理して、並列実行が可能な部分を特定し、さらに、中間表現ＩＲを操作して、並列化部分を含む修正中間表現ＩＲを形成する。この場合、種々の並列化手法を用いることができる。たとえば、ソフトウェア・パイプライン等の手法を用いて、ループを展開するようにしてもよいし、動的チェックを挿入して、データ依存関係を制

御依存関係に変換するようにしてもよい。ステップ４１０で、中間表現ＩＲをコード・ジェネレータに送信し、コード・ジェネレータは、受信した中間表現ＩＲに基づいて、並列実行可能なコードを生成する。ステップ４１２で、生成された並列実行可能なコードを格納する。

【００２７】

図５は、入力コードから並列実行可能なコードを生成するプロセスの別の実施例を示すフローチャートである。さまざまな実施形態において、プロセス５００は、エンドユーザのコンピュータで実行されるものでもよく、あるいは、開発者のシステム上でソフトウェア開発時に実行されるものでもよい。ステップ５０２で、入力コードを静的に解析して、入力コードのデータフローおよび制御フローのアスペクトを決定する。一部の実施形態において、動的な解析も実行して、入力コードのデータフローおよび制御フローの追加のアスペクトを決定するようにしてもよい。あるいは、一部の実施形態において、動的な解析は省略してもよい。ステップ５０４で、静的解析によって決定されたデータフローおよび制御フローのアスペクトに少なくとも部分的に基づいて、入力コードの中間表現ＩＲを生成する。動的解析が実行される場合には、アスペクトと同様に追加のアスペクトに少なくとも部分的に基づいて、中間表現ＩＲを生成する。ステップ５０６で、中間表現ＩＲを処理する。中間表現ＩＲに含まれる並列処理可能部分を特定し、並列実行可能なコードを生成する。ステップ５０８で、並列実行可能なコードを格納して、後で再利用できるようにする。この場合、並列実行可能なコードと元の入力コードとのマッピングはそのまま保持される。ステップ５１０で、入力コードの実行要求に応じて、格納された並列実行可能なコードをメモリにロードして、元の入力コードの代わりに実行する。以下に詳述するように、さまざまな実施形態において、種々のキャッシング手法を用いて実行ファイルを格納することができる。

【００２８】

図６は、静的解析プロセスの実施例を示すフローチャートである。静的解析プロセスは、たとえば、ユーザー要求に応じて、または、プログラムの起動に応じて、ステップ６０２で開始される。ステップ６０４で、ファイルシステムのスキャンを実行して、実行ファイル、参照されたＤＬＬ（ダイナミック・リンク・ライブラリ）および／または共用オブジェクト（ＳＯ：shared object）ファイルを検索する。さらに、使用頻度の高い／直前に使用された、および／または、更新頻度の高い／一番最近更新されたアプリケーションやライブラリを特定するようにしてもよい。ステップ６０６で、検索された各ファイルを開いて、読み出す。ステップ６０８で、前回の実行で実行時統計（ランタイム統計）のような動的データが得られた場合には、これを解析する。ステップ６１０で、ファイルの逆アセンブリを実行し、静的に生成される中間表現に変換する。ステップ６１２で、中間表現の変換処理を実行する。一部の実施形態において、変換処理は、コードを解析して、並列に実行可能な中間表現の部分を特定するものでもよい。あるいは、一部の実施形態において、変換処理で、ホットスポット解析および他のプロファイル情報等の実行時型情報（ランタイム情報）、分岐ターゲット情報、メモリ・エイリアス情報、および動的ループ回数情報等のインスツルメンテーションを容易にするインスツルメンテーションコード（instrumenting code）を挿入するようにしてもよい。また、一部の実施形態において、変換処理で、コードの修正を行なうようにしてもよい。

【００２９】

ステップ６１４で、チェッカーコードと、エラー回復等の実行時支援（ランタイムアシスタンス）を必要とするアイテムのランタイム（実行時）へのリンクのような他のインスツルメンテーションと、を含む可能なスレッドを作成する。ステップ６１６で、コードとアノテーション（注釈）とを発行する。アプリケーションの並列化処理が完了していない場合には、コードとアノテーションとを格納する。ＤＬＬまたは他のライブラリコードを呼び出しコードに応じた種々の方法で、並列化するようにしてもよい。また、一部の実施形態において、ＤＬＬの多重並列化処理を行ない、各コピーを呼び出しアプリケーション／機能に関連付けてもよい。

【００３０】
図７は、動的解析プロセスの実施例を示すフローチャートである。動的解析プロセスは、たとえば、ユーザー要求に応じて、または、プログラムの起動に応じて、ステップ７０２で開始される。プロセスは、プログラムの起動を監視している。ステップ７０４で、プログラムの起動を検知すると、コードを傍受する。ステップ７０６で、プログラムのアクセラレーションが完了しているか否かが判定される。部分的に、または、完全に並列化されたプログラム・バージョンが格納されている場合には、そのプログラムのアクセラレーションが完了していると考えられる。並列化および／または拡張バージョンは、上述した静的解析プロセスにより得られたものでもよいし、前回の動的解析プロセスにより得られたものでもよい。
【００３１】
プログラムのアクセラレーションが完了していると判定された場合には、ステップ７０８で、以前に格納されたコードを取得し、ステップ７１０で、必要に応じて、最適化処理とリンク処理とを実行する。ステップ７１２で、制御を移し、コードを実行する。コードの実行中に、ステップ７１４で、必要に応じて、実行時統計等の動的データを収集する。コードに挿入されたカウンタ等のインスツルメンテーションコードにより実行時統計が得られる。収集したデータを格納する。
【００３２】
並列化および／または拡張バージョンが見つからず、プログラムの並列化処理がなされていない場合には、ステップ７１６で、プロセスはコードを傍受して、インスツルメンテーション等、侵入を最小限に抑えたモニタリングを開始して、ホットスポットを特定する。ステップ７１７で、以前の実行または今回の実行で得られた実行時統計等の動的データを解析して統合する。ステップ７１８で、プログラムコードの逆アセンブリを実行して、中間表現ＩＲに変換する。ステップ７２０で、中間表現ＩＲを解析して、変換する。必要に応じて、インスツルメンテーションコードを挿入してもよく、コードを挿入した場合には、コードの並列化処理を行なうようにしてもよい。ステップ７２２で、可能なスレッドを作成する。ステップ７２４で、実行可能なコードとアノテーションとを発行し、必要に応じて、不揮発性記憶装置に書き込んで格納する。未修正／最小限のインスツルメンテーション後のコードと修正／並列化コードとの間にはマッピングが確立されている。次に、ステップ７１２で、制御を修正／並列化コードに移し、実行時統計等の動的データを収集する。
【００３３】
必要に応じて、プロセス７００を繰り返し実行してもよい。たとえば、ホットスポットが時間と共に変化した場合、または、以下に詳述するように、投機的（推量的）選択が間違っていた場合には、ランタイムシステムが、プロセスの繰り返しおよびコードの再生成の必要性を示唆するようにしてもよい。
【００３４】
図８は、図４および図５に示すプロセスを実行する性能を向上させたシステムの実施例を示すブロック図である。図示した例では、入力コードには、高級言語で書かれたアプリケーションプログラムのコンパイル処理等により作成される１つまたは複数のソース実行ファイル１が含まれる。一部の実施形態において、ソース実行ファイルは、コンパイラで生成されたコード等のバイナリコード、ＤＬＬ（ダイナミック・リンク・ライブラリ）、共用オブジェクト（ＳＯ）ファイル、またはこれらの組み合わせを含むものでもよい。図示した例では、ソース実行ファイルは、性能エンハンサーと同じシステムにインストールされている。アプリケーションは、直接的な明示スレッド、スレッドライブラリに基づく間接的なスレッド、またはこれらの組み合わせを含むものでもよい。
【００３５】
図示したシステムにおいて、実行ファイルセクション、初期化および非初期化され、静的に割り当てられたデータ、スタック、および動的に割り当てられたデータ等の複数のセグメントにメモリが分割されていると仮定する。（マウスのクリック等）ユーザー操作の

直接の結果として、または、その他の理由で（たとえば、他のプログラムによるトリガーやネットワークイベントに応じて）プログラムが起動すると、メモリ空間が最初に割り当てられる。オペレーティング・システムＯＳが、ローダーを用いてメモリに実行ファイルをロードし、必要に応じて、他の実行ファイルを移転および他の実行ファイルにリンクさせる。次に、新しく起動されたプログラムに制御を移す。

【００３６】

図示した例では、性能エンハンサーは、以下に示す機能コンポーネント（機能部品）、すなわち、逆アセンブラ／コンバータ５と、並列化部７と、コードエミッタ８と、エグゼキューションマネージャ（実行管理部）９と、コンフィグレーションマネージャ（構成管理部）１２と、ファイルキャッシュマネージャ４と、を備える。一部のコンポーネントの機能の一部または全部が、他のコンポーネントに含まれるものでもよい。また、これらのコンポーネントの一部を省略してもよい。

【００３７】

コードの１セクション（すなわち、オペレーションコード（命令コード）の最初のビットの記憶場所）に対するポインタを受信すると、逆アセンブラ／コンバータ５が命令の逆アセンブリを実行する。このプロセスにより、入力コードの命令およびオペランド（演算対象）が特定され、（プロセッサモード等）他の全体的な情報と共に、この情報が中間表現ＩＲ６に変換される。

【００３８】

中間表現ＩＲは、入力コードのデータ構造およびプログラム情報を表わす。中間表現ＩＲに基づく最適化処理および変換処理を実行するコンパイラで、中間表現ＩＲを解析し、操作するようにしてもよい。コンパイラは、複数の中間表現ＩＲを用いて、時間と共にデータフォーマットを変化させて、様々なコンパイル段階を容易にするものでもよい。中間表現ＩＲには、通常、実行するべき演算に関する情報、演算に影響を与える（ソース）データ、および宛先データが含まれる。

【００３９】

図９は、中間表現の例を示す図である。図示した例は、フィボナッチ数列を算出する関数のＬＬＶＭ（Low Level Virtual Machine：ローレベル仮想マシン）中間表現ＩＲを示す。関数のソースコードは以下のとおりである。

```
int fib(int n) {
    if (n==1)       return 0;
    if (n==2)        return 1;
    return (fib(n-1)+fib(n-2));
}
```

【００４０】

図８に戻って、逆アセンブリにより得られた中間表現ＩＲを並列化部７で処理し、新しい中間表現ＩＲを生成する。この例では、並列化部は、最終的に実行可能なコードの生成につながる複数の最適化「パス」を備える最新の最適化コンパイラと同様のものである。ソース入力を中間表現ＩＲに変換後、これらのパスで中間表現を変換する。各パスは、直接的に、または、間接的に、中間表現ＩＲを改善して、次段の最適化段階のための中間表現ＩＲを準備する。最適化処理の例としては、デッドコード削除、定数伝搬、ループ不変解析およびループ展開が挙げられる。一部の実施形態において、一部のパスは、単純に中間表現ＩＲを解析し、次段のパス用に付加的な最適化構造を形成するものでもよい。多重最適化処理および変換処理アルゴリズム自体に加えて、プログラム的にコード化された判定（「ヒューリスティック」とも称する）により、これらの最適化処理を適用するべきタイミングおよび環境を指示するようにしてもよい。たとえば、ループを展開すると、コードが大きくなり、必要なメモリ容量のオーバーフローが生じ、ループの展開による効果が相殺されてしまう場合には、ループの展開は望ましくない。さらに、各パスの順序を決める高レベル制御を行なうようにしてもよい。並列化部の一部の実施形態において、多重パスを用いて、スレッドレベルの並列性を特定し、抽出するようにしてもよい。このような

最適化処理により、共通メモリを共有する別個のＣＰＵで実行可能な独立実行スレッドを形成できる。

【００４１】

並列化部は、静的処理を実行するものでも、動的処理を実行するものでもよい。一部の実施形態において、並列化部は、静的／オフライン解析と動的／ランタイム解析とを混ぜて実行し、性能を向上させるものでもよい。

【００４２】

並列化部が動的処理を行ない、単純な実行時チェックよりも複雑な最適化処理を実行する場合には、追加の処理が性能に与える影響を最小限に抑えるように、処理を実行する場所や方法を変化させるようにしてもよい。一部の実施形態において、実行中のアプリケーションとは別のプロセスやスレッドで処理や解析を実行するようにしてもよい。また、一部の実施形態において、アプリケーションと同じプロセスやスレッドで処理を実行するようにしてもよい。

【００４３】

一部の実施形態において、並列化部は、性能を向上させるスレッドレベルの並列化の推量を伴う最適化処理を行なうようにしてもよい。このような場合、並列化部は、並列化に関する「推量（スペキュレーション）」を行なう、と考えることができる。コードを挿入して、推量が正しかったことを確認する。たとえば、並列化部は、長期手続き呼び出しが「０」という結果を戻すという推量を行なう。推量を行なうことにより、結果が得られるまで実行できなかったコードを、並列に実行することが可能になる。ただし、コードが結果をコミット（確定）する前に、呼び出された手続きが実際に「０」という結果を戻すことを検証する必要がある。別の例として、ループの多重繰り返しを別々のスレッドで実行する場合には、プログラマーが指定した回数繰り返した結果のみがコミットされることを保証する必要があるかもしれない。すなわち、間違った繰り返しの影響を取り消す、または、緩和する必要があるかもしれない。このような形態の投機的（推量的）並列化において、ランタイム環境１１に関連して説明したランタイム支援に依存する、および、ランタイム支援と密接に作用するように、生成されるコードを構築するようにしてもよい。

【００４４】

制御推量、データ推量およびメモリ順序付け推量のいずれか、または、すべてを実行するようにしてもよい。一部の実施形態において、並列化部は、推量（スペキュレーション）やメモリモデルをトランザクションと見なすものでもよい。

【００４５】

一部の実施形態において、リアルタイムまたはオフラインのいずれかで、ターゲットシステム上での再コンパイルが可能なシステムで推量を行なうようにしてもよい。投機（推量）システムおよび／またはトランザクションシステムにおいては、コンフリクトを検出して、データ・バージョニングを支援する必要がある。さまざまな実施形態において、「積極的（ｅａｇｅｒ）」または「消極的（ｌａｚｙ）」に分類される方法が用いられる。たとえば、「積極的」なデータ・バージョニング方法は、「取り消し（ｕｎｄｏ）」ログを用いて、間違った演算値を前の状態に戻す。一方、「消極的」なデータ・バージョニング方法は、書き込みバッファを用いて、データの種々のバージョンを書き込む。いずれの方法にも長所と短所がある。静的コンパイラ、特に、プロファイル誘導フィードバックを行わない静的コンパイラでは、いずれのスキームが最適であるかの推量を行ない、その選択肢を選択することができる。実際には、最適な手法は、多くの場合、コンパイル時には入手できない（実際の遅延等の）ランタイム情報によって決まる。すなわち、最適な選択は、実行時に選択されるものである。さらに、プログラムが非常に異なる特性を有するさまざまな実行段階を経る場合があるため、単一の最適な手法を選択することができない場合もある。動的並列化部は、さまざまな手法の最適な組み合わせを用いて最初の推論を行ない、プログラム挙動の時間変化を観察し、プログラムの変化に応じて、よりより結果を与える別の手法を選択するように新しいコードを生成することができる。

【００４６】

図８に戻って、コードエミッタ８は、並列化された中間表現ＩＲを処理して、バイナリコード化された実行可能な命令を形成する。レジスタは通常この段階で割り当てられ、特定の命令が選択される。一部の実施形態において、元のソースプログラムに直接対応する命令に加えて、追加の命令を用いてインスツルメンテーションを実行するようにしてもよい。インスツルメンテーションにより、コードが繰り返し呼び出されるホットスポット等のプログラム特性、分岐ターゲット等のランタイム情報、またはメモリ・エイリアス情報を特定できる。一部の実施形態において、コードエミッタの機能を実行中のアプリケーションとは別のプロセスやスレッドで実行するようにしてもよい。一部の実施形態において、コードエミッタの機能を、並列化部７またはエグゼキューションマネージャ９が備えるようにしてもよい。

【００４７】

修正された（すなわちインスツルメンテーションおよび／または並列化処理された）アプリケーションは、制御／命令のマルチスレッド、静的に、および／または、動的に割り当てられたデータ、および、追加のランタイム支援を可能にするライブラリへのフックまたはリンクを含むものでもよい。動的に逆アセンブリされて、動的に生成されたコードを備える環境において、メモリ内画像１０では、コードの各ブロック端に制御／管理プロセスへの分岐が含まれている。このような協調マルチスレッド環境下では、各スレッドは「軽量」と考えられ、演算をこのようなスレッドのワークキューとして構築することが望ましい。ワークキューは、ランタイムシステム１１により管理される。

【００４８】

（ソフトウェア開発時における並列化とは対照的に）アプリケーションがインストールされ実行されるシステムにおいてアプリケーションを並列化処理する環境下で、シングルスレッド処理の状況や正確性を維持できるように、アプリケーション用に修正された動作環境を形成する。たとえば、修正アプリケーションが、複数のＣＰＵで同時に実行され、メモリを共有する必要があるマルチスレッドを備える構成でもよい。マルチスレッドは、並列「プログラムカウンター」や並列スタック等、対応するリソースのコピーを備える必要がある。これらのリソースは、仮想マシンおよびランタイム環境１１として、構築され、管理される。ランタイムリソースは、スレッドマネージャ、メモリマネージャ、例外ハンドラ、および共通プログラムライブラリおよびＯＳ機能の新しい／交換コピーが含まれるものでもよい。これらのリソースをすべて用いることにより、投機的（推量）／トランザクション処理が容易になる。一部の実施形態において、ランタイム機能を、修正アプリケーションのメモリ内画像の一部としてもよい。

【００４９】

計算コストが高い最適化処理および変換処理を実行して、コードを発行する場合、一部の実施形態において、今後のソースプログラムの起動に備えて、これらの処理で得られた出力を格納しておくようにしてもよい。たとえば、解析後に、インスツルメンテーションコードを挿入したプログラムの新バージョンを形成し、プロファイル情報を収集するようにしてもよい。次にプログラムが実行される時に、挿入されたインスツルメンテーションコードを用いて、プログラムが実行時間の大部分を費やす「ホットスポット」の位置等の情報を収集することができる。別の例として、実行時メモリ・エイリアス解析を挿入するようにしてもよい。また別の例として、アプリケーションの並列化バージョンや追加の実行時チェックを最小限に抑えた多重並列化バージョンが挙げられる。新しい拡張実行ファイルを１つまたは複数のファイルに書き込む。一部の実施形態において、これらのファイルの記憶に用いられるメモリ空間の容量を制限するために、ファイルキャッシュマネージャ４をオプションで用いるようにしてもよい。

【００５０】

一部の実施形態において、ファイルキャッシュマネージャは、使用頻度の高いファイルを保持するキャッシュを管理するようにしてもよい。別の実施形態において、さまざまな管理ポリシーを用いるようにしてもよい。たとえば、直前に使用されたポリシーまたは使用頻度の高いポリシーの組み合わせを用いるようにしてもよい。ソース実行ファイルとこ

れらのファイルとの間のマッピングを維持する。プログラムが起動され、並列化が望ましいと考えられる場合はいつでも、このキャッシュを調べて、アプリケーションの並列化バージョン（またはインスツルメンテーション後のバージョン）が存在するか否かを判定する。存在する場合には、この情報をエグゼキューションマネージャ９に伝達して、対応するファイルをメモリにロードし、最初に起動される実行ファイルの代わりに実行する。

【００５１】

一部の実施形態において、オプションでエグゼキューションマネージャ９を備え、上述した複数のモジュールおよび複数の「静的」および「動的」ステップ全体にわたる調整を行なうようにしてもよい。ターゲットシステムに対する共通マネージャ下でこれらの機能をリンクさせることにより、動的処理と静的処理とを混ぜて、必要に応じて起動できる。たとえば、エグゼキューションマネージャが、更新のダウンロードおよびインストールによりアプリケーションが変更されたことを検知すると、静的解析を開始する。一部の実施形態において、エグゼキューションマネージャの機能を、キャッシュマネージャ４、並列化部７、または、コードエミッタ８が備えるようにしてもよい。

【００５２】

図１０は、プログラム表現のいくつかの例とそれらの相互関係を示す図である。たとえば、プログラムは、（高級言語またはアセンブリでコード化（符号化）された）ソース形式でも、バイナリ実行ファイルとして（コンパイルされているが実行可能ではない）オブジェクト形式でもよい。システムモジュールが、複数のオブジェクトファイルをリンクさせて、より大きく、より完全な実行ファイルを作成するようにしてもよい。また、システムモジュールが、プログラムが起動すると、起動したプログラムをメモリにロードするようにしてもよい。実行中のメモリ常駐アプリケーションは直接実行されるものでもよいし、あるいは、インタプリタ（解釈プログラム）内で実行されるものでもよい。また、プログラムは、中間表現のコンパイラ内に存在するものでもよく、コンパイラのコードジェネレータは、このプログラムを実行可能な形式に変換するものでもよい。実行可能な形式に変換されたプログラムをファイルに書き込み、メモリにロードして、直接実行することができる。あるいは、実行可能な形式に変換されたプログラムをインタプリタやランタイムに渡して実行させるようにしてもよい。

【００５３】

以上、本発明の理解を助ける目的で実施例を詳細に説明してきたが、本発明は、何ら実施例の詳細に限定されるものではなく、さまざまに変形・変更した態様で実施可能である。上述の実施例は例示に過ぎず、何ら本発明を限定するものではない。

【図１】

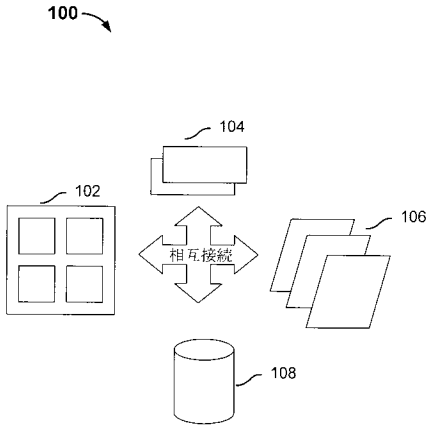


FIG. 1

【図２】

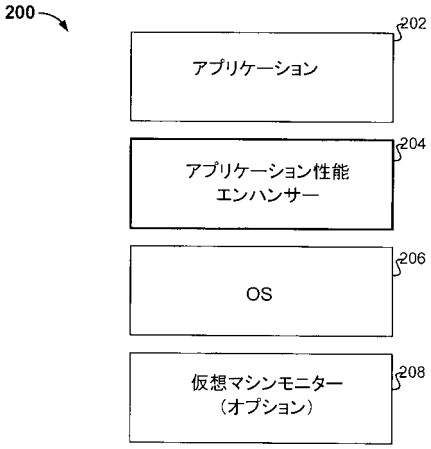


FIG. 2

【図３Ａ】

FIG. 3A

【図３Ｂ】

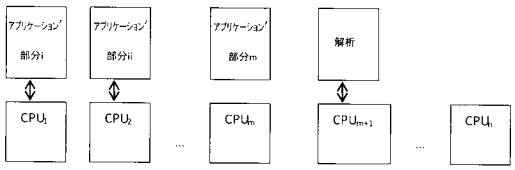


FIG. 3B

【図４】

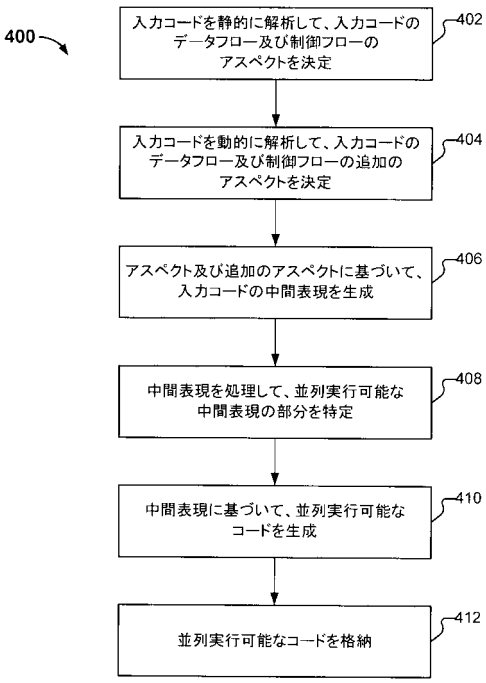


FIG. 4

【図５】

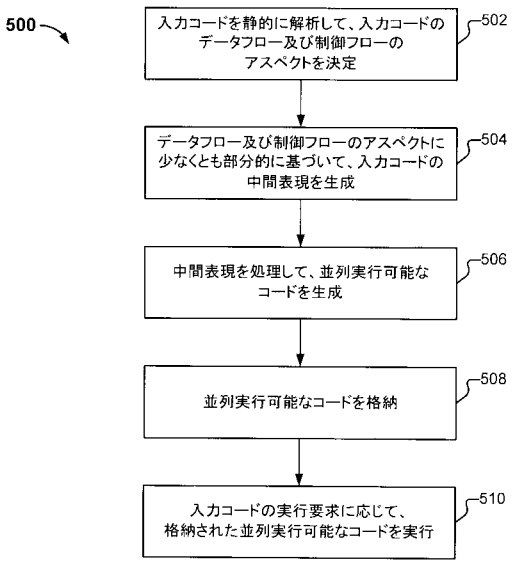


FIG. 5

【図６】

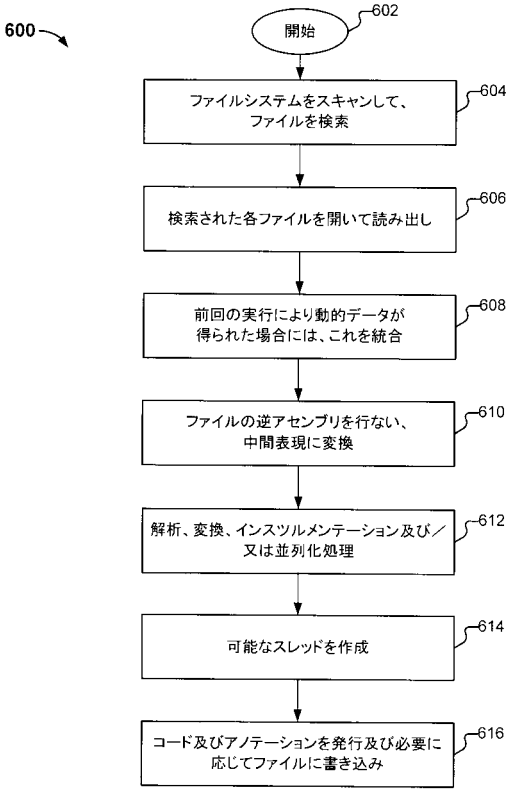


FIG. 6

【図７】

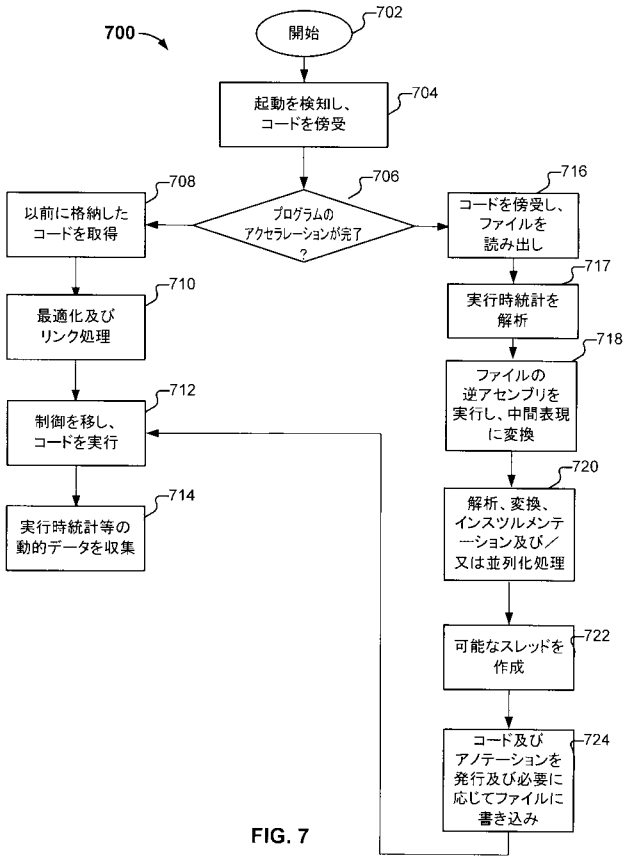


FIG. 7

【図８】

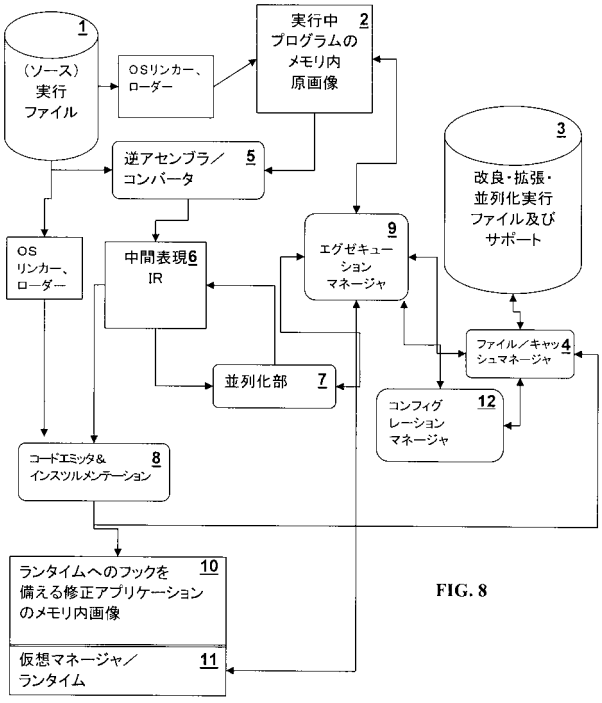


FIG. 8

【図９】

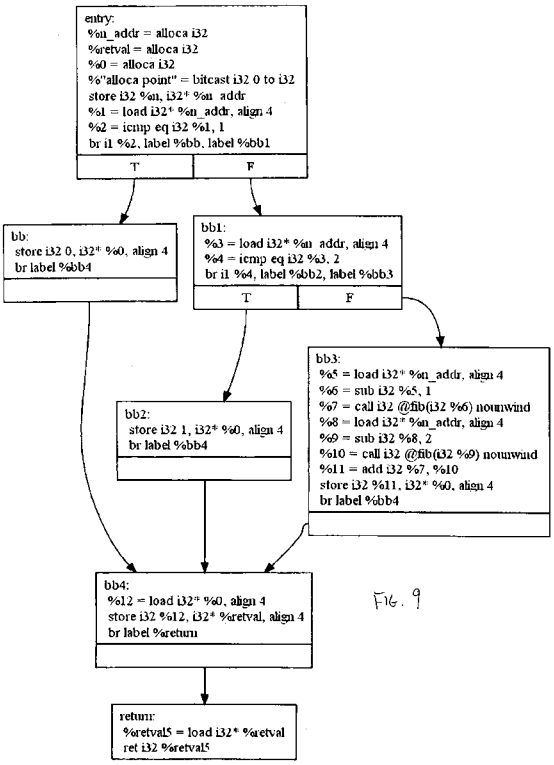
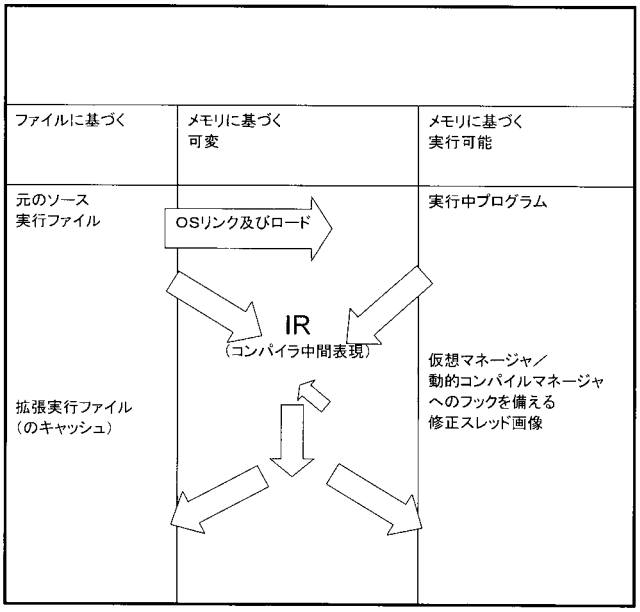


FIG. 9

「fib」関数用の制御フローグラフ

【図１０】

FIG. 10



【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 09/04649

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 15/00 (2009.01)
USPC - 712/235
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)
USPC - 712/235

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC - 712/212-213, 215, 229, 235; 710/8; 716/4, 19-20; 711/120, 130, 141, 150; 707/201; 709/213-214 (keyword limited - see terms below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Electronic Databases Searched: PubWEST(USPT, PGPB, EPAB, JPAB), Google Scholar
Search Terms Used: parallel, parallelization, convert, translate, change, multiple, threads, static, text, line, segment, function, module, content, input, instruction, operation, analyze, examine, study, evaluate, investigate, program, code, dynamically, execute, runt

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X -- Y	US 2008/0005547 A1 (PAPAKIPOS et al.) 03 January 2008 (03.01.2008) entire document; especially Abstract; Figs. 2G, 8A, 11A-11C; para [0028]-[0029], [0032], [0043], [0045]-[0050], [0054], [0062], [0109], [0135], [0153], [0227]-[0230], [0236]-[0237], [0262], [0273], [0276]-[0279], [0297], [0321], [0338], [0357], [0399]-[0408], [0410]-[0415], [0418], [0441]-[0444], [0447]-[0449], [0456], [0460], [0500]-[0501]	1-9, 11-19 ---------- 10
Y	US 2003/0236951 A1 (CHOI et al.) 25 December 2003 (25.12.2003) entire document; especially Abstract; para [0020]-[0023], [0030], [0113], [0121]	10
A	US 2005/0086650 A1 (YATES et al.) 21 April 2005 (21.04.2005) entire document	1-19

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

* 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
13 September 2009 (13.09.2009)	21 SEP 2009
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201	Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (July 2009)

フロントページの続き

(81)指定国 AP(BW,GH,GM,KE,LS,MW,MZ,NA,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(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,SE,SI,SK,SM,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AO,AT,AU,AZ,BA,BB,BG,BH,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,IS,JP,KE,KG,KM,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,PE,PG,PH,PL,PT,RO,RS,RU,SC,SD,SE,SG,SK,SL,SM,ST,SV,SY,TJ,TM,TN,TR,TT,TZ,UA,UG,US,UZ,VC,VN,ZA,ZM,ZW

(72)発明者 ジョンーンズ・ジョル・ケビン
アメリカ合衆国 カリフォルニア州９４３０６ パロ・アルト，ケンブリッジ・アベニュー，４４５，スイート ビー
(72)発明者 シャープ・マイケル・ダグラス
アメリカ合衆国 カリフォルニア州９４３０６ パロ・アルト，ケンブリッジ・アベニュー，４４５，スイート ビー
(72)発明者 バエブ・イワン・ディミトロフ
アメリカ合衆国 カリフォルニア州９４３０６ パロ・アルト，ケンブリッジ・アベニュー，４４５，スイート ビー

Ｆターム(参考) 5B081 CC16 CC30 CC32 CC64