(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2011-22993
(P2011-22993A)
(43)公開日　平成23年2月3日(2011.2.3)

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

審査請求 未請求　請求項の数 24　ＯＬ　（全 23 頁）

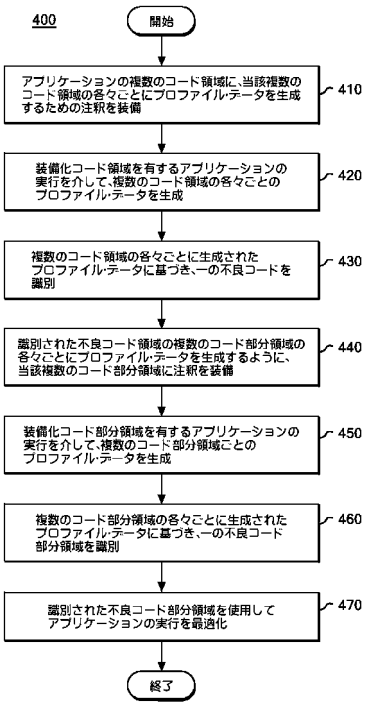
(21)出願番号　特願2010-109001 (P2010-109001)
(22)出願日　平成22年5月11日 (2010.5.11)
(31)優先権主張番号　2672337
(32)優先日　平成21年7月15日 (2009.7.15)
(33)優先権主張国　カナダ(CA)

(71)出願人　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　ＭＡＳＣＨＩＮＥＳ　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】マルチパス動的プロファイリングのためのコンピュータに実行させる方法、システム及びコンピュータ・プログラム

(57)【要約】　（修正有）
【課題】コンパイラにおいて、アプリケーションの実行を最適化する。
【解決手段】アプリケーションの複数のコード領域に、プロファイル・データを生成するための注釈が装備され（４１０）、装備化コード領域を有する前記アプリケーションの実行を介して、前記複数のコード領域の各々のためのプロファイル・データが生成され（４２０）、前記プロファイル・データに基づき、不良コード領域が識別され（４３０）、当該不良コード領域の複数のコード部分領域に、当該プロファイル・データを生成するための注釈が装備され（４４０）、装備化コード部分領域を有する前記アプリケーションの実行を介して、前記プロファイル・データが生成され（４５０）、前記前記生成されたプロファイル・データに基づき、不良コード部分領域が識別され（４６０）、前記不良コード部分領域を使用して、前記アプリケーションの実行が最適化される（４７０）。
【選択図】図４



【特許請求の範囲】
【請求項１】
コンピュータ内の１つ以上のプロセッサによって実行される、マルチパス動的プロファイリングのためのコンピュータに実行させる方法であって、
アプリケーションの複数のコード領域が前記１つ以上のプロセッサ上でリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記複数のコード領域に装備するステップと、
前記アプリケーションを実行することにより、前記複数のコード領域の各々のためのプロファイル・データを生成するステップと、
前記生成されたプロファイル・データから、キャッシュ・ミスのカウントが予定のしきい値を超える一のコード領域を、一の不良コード領域として識別するステップと、
前記複数のコード領域がリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記不良コード領域に装備するステップと、
前記アプリケーションを実行することにより、前記不良コード領域を含む前記複数のコード領域のための追加のプロファイル・データを生成するステップと、
前記追加のプロファイル・データから、前記不良コード領域の一の部分領域を識別するステップとを含む、コンピュータに実行させる方法。
【請求項２】
前記複数のコード領域が、前記アプリケーションの少なくとも１つのコンパイル単位を評価することによって識別される、請求項１記載のコンピュータに実行させる方法。
【請求項３】
前記プロファイル・データが、少なくとも１つのハードウェア性能カウンタに基づいて生成される、請求項１記載のコンピュータに実行させる方法。
【請求項４】
前記プロファイル・データが、各コード領域を実行する間に生じるキャッシュ・ミスのカウントを含む、請求項１記載のコンピュータに実行させる方法。
【請求項５】
前記複数のコード領域が、前記アプリケーションのソース・コード、前記アプリケーションのオブジェクト・コード及び前記アプリケーションの中間コンパイラ表現から選択される、請求項１記載のコンピュータに実行させる方法。
【請求項６】
前記識別された部分領域が、メモリ参照を含む前記アプリケーションのソース・コード内の命令に対応する、請求項１記載のコンピュータに実行させる方法。
【請求項７】
前記識別された部分領域を最適化するステップをさらに含み、
当該最適化するステップが、インライニング、クローニング、アウトライニング、間接呼び出しの特化、不良ロード主導型データ・プリフェッチング、データ再編成及び命令スケジューリングのうち少なくとも１つを行うことを含む、請求項１記載のコンピュータに実行させる方法。
【請求項８】
各注釈が、少なくとも実行時ライブラリへの関数呼び出し又は命令のインライン・シーケンスから選択された注釈コードに拡張される、請求項１記載のコンピュータに実行させる方法。
【請求項９】
マルチパス動的プロファイリングのためのコンピュータ・プログラムであって、前記プログラムが、プロセッサに、
アプリケーションの複数のコード領域が前記プロセッサ上でリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記複数のコード領域に装備するステップと、
前記アプリケーションを実行することにより、前記複数のコード領域の各々のためのプ

ロファイル・データを生成するステップと、
前記生成されたプロファイル・データから、キャッシュ・ミスのカウントが予定のしきい値を超える一のコード領域を、一の不良コード領域として識別するステップと、
前記複数のコード領域がリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記不良コード領域に装備するステップと、
前記アプリケーションを実行することにより、前記不良コード領域を含む前記複数のコード領域のための追加のプロファイル・データを生成するステップと、
前記追加のプロファイル・データから、前記不良コード領域の一の部分領域を識別するステップとを実行させる、コンピュータ・プログラム。
【請求項１０】
前記複数のコード領域が、前記アプリケーションの少なくとも１つのコンパイル単位を評価することによって識別される、請求項９記載のコンピュータ・プログラム。
【請求項１１】
前記プロファイル・データが、少なくとも１つのハードウェア性能カウンタに基づいて生成される、請求項９記載のコンピュータ・プログラム。
【請求項１２】
前記プロファイル・データが、各コード領域を実行する間に生じるキャッシュ・ミスのカウントを含む、請求項９記載のコンピュータ・プログラム。
【請求項１３】
前記複数のコード領域が、前記アプリケーションのソース・コード、前記アプリケーションのオブジェクト・コード及び前記アプリケーションの中間コンパイラ表現から選択される、請求項９記載のコンピュータ・プログラム。
【請求項１４】
前記識別された部分領域が、少なくとも前記アプリケーションのソース・コード命令、前記アプリケーションのオブジェクト・コード命令及び前記アプリケーションの中間コンパイラ表現の命令から選択された命令に対応する、請求項９記載のコンピュータ・プログラム。
【請求項１５】
前記プログラムが、前記プロセッサに、
前記複数のコード領域の各々のための前記生成されたプロファイル・データに基づき、前記識別された部分領域を最適化するステップをさらに実行させ、
当該最適化するステップが、インライニング、クローニング、アウトライニング、間接呼び出しの特化、不良ロード主導型データ・プリフェッチング、データ再編成及び命令スケジューリングのうち少なくとも１つを行うことを含む、請求項９記載のコンピュータ・プログラム。
【請求項１６】
各注釈が、少なくとも実行時ライブラリへの関数呼び出し又は命令のインライン・シーケンスから選択された注釈コードに拡張される、請求項９記載のコンピュータ・プログラム。
【請求項１７】
プロセッサと、
マルチパス動的プロファイリングのためのプログラムを保持するメモリとを備えるシステムにして、
前記プログラムが、前記プロセッサに、
アプリケーションの複数のコード領域が前記プロセッサ上でリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記複数のコード領域に装備するステップと、
前記アプリケーションを実行することにより、前記複数のコード領域の各々のためのプロファイル・データを生成するステップと、
前記生成されたプロファイル・データから、キャッシュ・ミスのカウントが予定のしき

い値を超える一のコード領域を、一の不良コード領域として識別するステップと、
前記複数のコード領域がリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記不良コード領域に装備するステップと、
前記アプリケーションを実行することにより、前記不良コード領域を含む前記複数のコード領域のための追加のプロファイル・データを生成するステップと、
前記追加のプロファイル・データから、前記不良コード領域の一の部分領域を識別するステップとを実行させる、システム。
【請求項１８】
前記複数のコード領域が、前記アプリケーションの少なくとも１つのコンパイル単位を評価することによって識別される、請求項１７記載のシステム。
【請求項１９】
前記プロファイル・データが、少なくとも１つのハードウェア性能カウンタに基づいて生成される、請求項１７記載のシステム。
【請求項２０】
前記プロファイル・データが、各コード領域を実行する間に生じるキャッシュ・ミスのカウントを含む、請求項１７記載のシステム。
【請求項２１】
前記複数のコード領域が、前記アプリケーションのソース・コード、前記アプリケーションのオブジェクト・コード及び前記アプリケーションの中間コンパイラ表現から選択される、請求項１７記載のシステム。
【請求項２２】
前記識別された部分領域が、メモリ参照を含む前記アプリケーションのソース・コード内の命令に対応する、請求項１７記載のシステム。
【請求項２３】
前記プログラムが、前記プロセッサに、
前記識別された部分領域を最適化するステップをさらに実行させ、
当該最適化するステップが、インライニング、クローニング、アウトライニング、間接呼び出しの特化、不良ロード主導型データ・プリフェッチング、データ再編成及び命令スケジューリングのうち少なくとも１つを行うことを含む、請求項１７記載のシステム。
【請求項２４】
各注釈が、少なくとも実行時ライブラリへの関数呼び出し又は命令のインライン・シーケンスから選択された注釈コードに拡張される、請求項１７記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、コンパイラに係り、さらに詳細に説明すれば、マルチパス及び多目的の動的分析を促進するためのコンパイラ装備化（instrumentation）インフラストラクチャに係る。
【背景技術】
【０００２】
コンパイラは、高水準プログラミング言語で書かれたコンピュータ・プログラムを、コンピュータ・システム内の１つ以上の中央処理装置（ＣＰＵ）によって実行されるマシン・コードに変換するために使用されるツールである。コンパイラがこの変換をどのように行うかに依存して、結果的なプログラムは、コンピュータ上で異なる速度で実行されるか、或いはより多い又はより少ないシステム・メモリ及びストレージ空間を必要とすることがある。
【０００３】
従来、ソース・コードからマシン・コードへの直接変換以上のことを行うコンパイラを作成するために、多くの研究開発がなされてきた。一般に、かかるコンパイラは、最適化コンパイラと呼ばれる。最適化コンパイラは、ソース・コードを分析し、ターゲット・コ

ンピュータ・システム上でより効率的に実行可能な戦略を実装する。この文脈において、「最適化」とは、一般に、ソース・コードのコンパイル方法を選択的に修正することにより、コンピュータ・プログラムの速度又は効率を改良することを意味する。

【発明の概要】

【発明が解決しようとする課題】

【０００４】

最適化コンパイラは、効率的なマシン・コードを生成するために、ループ変換又はデータ・リマッピングのような多くの技術を使用することがあるが、コンピュータ・ハードウェアの進歩は、コンパイラの設計者に対し新しい挑戦を突きつけている。具体的には、最近ではＣＰＵ装置のクロック速度が増加しているのに対し、システム・メモリの速度は遅れを取っている。この速度の差（一般にメモリ待ち時間と呼ばれる）を管理しないと、データがシステム・メモリと授受される間に、ＣＰＵがアイドル状態に留まることになる。コンパイラ内で使用される１つの最適化戦略は、キャッシュの使用率（従って実行時間）を改良し且つプログラム実行中に生じるキャッシュ・ミスの数を減少させるというものである。

【課題を解決するための手段】

【０００５】

本発明の１つの側面は、コンピュータ内の１つ以上のプロセッサによって実行される、マルチパス動的プロファイリングのためのコンピュータに実行させる方法に向けられている。前記コンピュータに実行させる方法は、アプリケーションの複数のコード領域（code regions）が前記１つ以上のプロセッサ上でリンクされ且つ実行されるときにプロファイル・データを生成するための注釈（annotations）を、前記複数のコード領域に装備（instrument）するステップと、前記アプリケーションを実行することにより、前記複数のコード領域の各々のためのプロファイル・データを生成するステップと、前記生成されたプロファイル・データから、キャッシュ・ミスのカウントが予定のしきい値を超える一のコード領域を、一の不良コード領域（delinquent code region）として識別するステップと、前記複数のコード領域がリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記不良コード領域に装備するステップと、前記アプリケーションを実行することにより、前記不良コード領域を含む前記複数のコード領域のための追加のプロファイル・データを生成するステップと、前記追加のプロファイル・データから、前記不良コード領域の一の部分領域（sub-region）を識別するステップとを含む。前記コンピュータに実行させる方法は、前記識別された部分領域を最適化するステップをさらに含むことがある。

【０００６】

本発明の他の側面は、マルチパス動的プロファイリングのためのコンピュータ・プログラム（または、そのコンピュータ・プログラムを保持するコンピュータ可読ストレージ媒体）に向けられている。前記プログラムは、プロセッサに、アプリケーションの複数のコード領域が前記プロセッサ上でリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記複数のコード領域に装備するステップと、前記アプリケーションを実行することにより、前記複数のコード領域の各々のためのプロファイル・データを生成するステップと、前記生成されたプロファイル・データから、キャッシュ・ミスのカウントが予定のしきい値を超える一のコード領域を、一の不良コード領域として識別するステップと、前記複数のコード領域がリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記不良コード領域に装備するステップと、前記アプリケーションを実行することにより、前記不良コード領域を含む前記複数のコード領域のための追加のプロファイル・データを生成するステップと、前記追加のプロファイル・データから、前記不良コード領域の一の部分領域を識別するステップとを実行させる。また、前記プログラムは、前記プロセッサに、前記複数のコード領域の各々のための前記生成されたプロファイル・データに基づき、前記識別された部分領域を最適化するステップをさらに実行させることがある。

【０００７】
本発明の他の側面は、プロセッサと、マルチパス動的プロファイリングのためのプログラムを保持するメモリとを備えるシステムに向けられている。前記プログラムは、前記プロセッサに、アプリケーションの複数のコード領域が前記プロセッサ上でリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記複数のコード領域に装備するステップと、前記アプリケーションを実行することにより、前記複数のコード領域の各々のためのプロファイル・データを生成するステップと、前記生成されたプロファイル・データから、キャッシュ・ミスのカウントが予定のしきい値を超える一のコード領域を、一の不良コード領域として識別するステップと、前記複数のコード領域がリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記不良コード領域に装備するステップと、前記アプリケーションを実行することにより、前記不良コード領域を含む前記複数のコード領域のための追加のプロファイル・データを生成するステップと、前記追加のプロファイル・データから、前記不良コード領域の一の部分領域を識別するステップとを実行させる。前記プログラムは、前記プロセッサに、前記識別された部分領域を最適化するステップをさらに実行させることがある。
【発明の効果】
【０００８】
本発明は、アプリケーションの実行を最適化するという効果を奏する。具体的には、本発明は、アプリケーションの複数のコード領域がリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記複数のコード領域に装備し、前記アプリケーションを実行することにより、前記複数のコード領域の各々のためのプロファイル・データを生成し、前記生成されたプロファイル・データから、キャッシュ・ミスのカウントが予定のしきい値を超える一のコード領域を、一の不良コード領域として識別し、前記複数のコード領域がリンクされ且つ実行されるときにプロファイル・データを生成するための注釈を、前記不良コード領域に装備し、前記アプリケーションを実行することにより、前記不良コード領域を含む前記複数のコード領域のための追加のプロファイル・データを生成し、前記追加のプロファイル・データから、前記不良コード領域の一の不良コード部分領域を識別し、当該識別された不良コード部分領域を使用して、アプリケーションの実行を最適化するという効果を奏する。
【図面の簡単な説明】
【０００９】
【図１】本発明の実施形態に従った、アプリケーションの実行を最適化するためのシステムを示すブロック図である。
【図２】本発明の実施形態に従った、図１のマルチパス・プロファイラのコンポーネントを示すブロック図である。
【図３】本発明の実施形態に従った、ユーザの観点からアプリケーションの実行を最適化するための方法を示すフローチャートである。
【図４】本発明の実施形態に従った、アプリケーションの実行を最適化するための方法を示すフローチャートである。
【図５】本発明の実施形態に従った、マルチパス動的プロファイリングを行うための方法を示すフローチャートである。
【図６】本発明の実施形態に従った、マルチパス動的プロファイリングを行うための方法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】
【００１０】
本発明の実施形態は、マルチパス（すなわち、複数の装備化サイクル）及び多目的（すなわち、キャッシュミス・プロファイリング及び呼び出しセンシティブなブロック・カウントのプロファイリングのような複数のタイプのプロファイリングをサポートすること）の動的プロファイリングのための、一般化されたコンパイラ装備化インフラストラクチャを提供する。このインフラストラクチャは、マルチパス・プロファイリングをサポートし

、その場合、後続するパスが先行するパスからのプロファイル・データを使用して、プロファイリングを洗練化する。マルチパス・プロファイラは、粗粒度（coarse-grained）のキャッシュミス・プロファイリングを行うことにより、不良コード領域を識別する。不良コード領域とは、キャッシュ・ミスの許容可能なしきい値カウントよりも大きな数のキャッシュ・ミスを生ずるコード領域（これは準最適なアプリケーションの性能をもたらすことがある）を意味する。同様に、不良メモリ参照（delinquent memory reference）とは、キャッシュの許容可能なしきい値カウントよりも大きな数のキャッシュ・ミスを生ずるメモリ参照を意味する。その後、マルチパス・プロファイラは、前記不良コード領域内のメモリ参照だけを対象として、細粒度（fine-grained）のキャッシュミス・プロファイリングを行う。例えば、キャッシュミス・プロファイリングについて、コンパイラは、最初に粗粒度のキャッシュミス・プロファイリングを行うことにより、不良ロード（すなわち、キャッシュ・ミスに帰着する可能性が高いロード）を保持するコード領域を識別し、次に、当該コード領域内で細粒度のキャッシュミス・プロファイリングを行うことにより、個々の不良ロードを正確に識別する。すなわち、キャッシュ・ミスに帰着する可能性が高いメモリ参照を含む、ソース・コードの特定の行を識別する。

【００１１】

本明細書に開示したコンパイラ・インフラストラクチャは、マルチパス・プロファイリングを静的分析とともに使用することにより、アプリケーションのプロファイリング・オーバーヘッドを減少させる。さらに、このコンパイラは、一様な内部表現を使用することにより、不良コード領域を注釈し且つコードの低水準表現から当該コードの高水準表現に正確にマッピングする。さらに、複数の装備化サイクルは、コンパイラとプロファイリング・ツール（例えば、性能ライブラリ・パッケージ）との間の相互作用を改良することにより、プロファイリング・オーバーヘッドを減少させ且つ最適化の結果を改良する。例えば、複数の装備化サイクルは、異なる性能カウンタ・グループにわたってアプリケーションの特性をプロファイルすることを可能にする。さらに、複数の装備化サイクルは、異なるプロファイリング機構（例えば、ブロックカウンタ・プロファイリング、値のプロファイリング及び性能カウンタのプロファイリング）を組み合わせることを可能にする。

【００１２】

コンパイラ内で使用される最適化戦略は、プログラムのプロファイル・データに依存することがある。プロファイル・データは、コンパイル時の分析（静的プロファイリングと呼ばれる）及び／又は実行時の分析（動的プロファイリングと呼ばれる）を通して、これを収集することができる。プロファイル・ディレクテッド・フィードバック（profile-directed feedback：ＰＤＦ）とは、プログラム用のプロファイル・データを生成するために代表的なデータ・サンプルを使用して、当該プログラムの実行をプロファイルする方法を意味する。次に、コンパイラは、生成されたプロファイル・データを使用することにより、最適化をガイドする。

【００１３】

プロファイリング技術は、制御フローのプロファイリング、値のプロファイリング及び性能カウンタのプロファイリングを含む。さらに、制御フローのプロファイリングは、ノード・プロファイリング（vertex profiling）、エッジ・プロファイリング及びパス・プロファイリングに分類される。ノード・プロファイリングは、実行時の間に、コードの各基本ブロックが実行される頻度を測定する。基本ブロックとは、連続的な動作のシーケンスであって、制御のフローが当該シーケンスの開始点に入り且つ当該シーケンスの終了点を除き停止又は分岐する可能性なしに当該シーケンスの終了点から出るというようなシーケンスを意味する。エッジ・プロファイリングは、実行時の間に、各分岐遷移が実行される頻度を測定する。パス・プロファイリングは、実行時の間に、各パス（すなわち、相関のある分岐）が実行される頻度を測定する。値のプロファイリングは、不変性、生じそうな値、変数の範囲（例えば、分岐予測目的のため）を測定する。性能カウンタのプロファイリングは、特定のハードウェア・イベント（例えば、キャッシュ・ミス）を追跡する、ハードウェア性能カウンタを測定する。一般に、コンパイラは、性能ライブラリ・パッケ

ージによって提供される、アプリケーション・プログラム・インタフェース（ＡＰＩ）を起動するための呼び出しを挿入する。性能カウンタのプロファイリングは、アプリケーションに特有のメトリクス（例えば、ＣＰＩ(cycles per instruction)、ＦＬＯＰＳ(floating point operations per second)、ＭＩＰＳ（million instructions per second）及びキャッシュミス・レート）を決定するために使用される。

【００１４】

以下、本発明の実施形態を参照する。しかし、本発明が本明細書に開示した特定の実施形態に制限されないことを理解されたい。以下で説明する特徴及び要素の任意の組み合わせは、異なる実施形態に関係するか否かに拘わらず、本発明を実装し且つ実施することが意図されるからである。さらに、本発明の実施形態は、他の可能な解決法及び／又は従来技術に比較して優れた効果を奏することがあるが、所与の実施形態が特定の効果を奏するか否かは、本発明を制限するものではない。従って、以下で説明する側面、特徴、実施形態及び効果は、単に例示的なものであって、請求項に明示的に記載されている場合を除き、請求項に係る発明の要素又は制限事項であると解すべきではない。同様に、「本発明」という用語が使用されている場合であっても、このことは、本明細書に開示した発明性のある任意の主題の一般化と解すべきではなく、請求項に明示的に記載されている場合を除き、請求項に係る発明の要素又は制限事項であると解すべきではない。

【００１５】

本発明の１つの側面は、コンピュータ・システムに関連して使用するためのコンピュータ・プログラムに向けられている。このプログラムは、実施形態の機能（本明細書に開示した方法を含む）を定義し、種々のコンピュータ可読ストレージ媒体上に保持されることができる。コンピュータ可読ストレージ媒体は、情報を永久に格納するための書き込み不能ストレージ媒体（例えば、ＣＤ－ＲＯＭドライブによって読み取り可能なＣＤ－ＲＯＭディスクのようなコンピュータ内の読み取り専用メモリ装置）、変更可能な情報を格納するための書き込み可能ストレージ媒体（例えば、ディスケット・ドライブ内のフレキシブル・ディスク又はハードディスク・ドライブ）等を含む。かかるコンピュータ可読ストレージ媒体は、本発明の機能を指示するコンピュータ可読命令を担持する場合、本発明の実施形態である。他の媒体は、例えば、コンピュータ又は電話網を通してコンピュータに情報を伝達する通信媒体であって、無線通信ネットワークを含む。特に、後者の実施形態は、インターネット及び他のネットワークと情報を授受することを含む。かかる通信媒体は、本発明の機能を指示するコンピュータ可読命令を担持する場合、本発明の実施形態である。広義には、コンピュータ可読ストレージ媒体及び通信媒体は、本明細書において、これをコンピュータ可読媒体と称することがある。

【００１６】

一般に、本発明の実施形態を実装するために実行されるルーチンは、オペレーティング・システム又は特定アプリケーションの一部、コンポーネント、プログラム、モジュール、オブジェクト又は命令シーケンスとすることができる。一般に、本発明のコンピュータ・プログラムは、ネイティブ・コンピュータによってマシン可読フォーマット（従って、実行可能命令）に変換される、複数の命令から構成される。さらに、プログラムは、当該プログラムに対しローカルに存在するか、或いはメモリ内に又はストレージ装置上に置かれる、変数及びデータ構造から構成される。さらに、後述する種々のプログラムは、本発明の特定の実施形態において当該プログラムがそのために実装されるアプリケーションに基づき、これを識別することができる。しかし、以下で使用する特定のプログラム命名法は、便宜上のものであるに過ぎないから、本発明は、かかる命名法によって識別されるか又は暗示される特定のアプリケーション内でのみ使用するように制限されないことを理解されたい。

【００１７】

図１は、本発明の実施形態に従った、アプリケーションの実行を最適化するためのシステム１００を示す。ネットワーク化されたシステム１００は、コンピュータ１０２を含む。コンピュータ１０２は、ネットワーク１３０を介して、他のコンピュータに接続するこ

とができる。一般に、ネットワーク１３０は、通信ネットワーク及び／又は広域ネットワーク（ＷＡＮ）である。特定の実施形態では、ネットワーク１３０は、インターネットである。

【００１８】

一般に、コンピュータ１０２は、バス１１２を介してメモリ１０６に接続されたプロセッサ１０４、ネットワーク・インタフェース装置１１０、ストレージ１０８、入力装置１１４及び出力装置１１６を含む。一般に、コンピュータ１０２は、オペレーティング・システム（図示せず）の制御下にある。かかるオペレーティング・システムの例は、ＵＮＩＸ（Ｔｈｅ　Ｏｐｅｎ　Ｇｒｏｕｐの商標または登録商標）、マイクロソフト社のＷｉｎｄｏｗｓ（Ｍｉｃｒｏｓｏｆｔ　Ｃｏｒｐｏｒａｔｉｏｎの商標または登録商標）オペレーティング・システムのバージョン及びＬｉｎｕｘ（Ｌｉｎｕｓ　Ｔｏｒｖａｌｄｓの商標または登録商標）オペレーティング・システムを含む。より一般的に云えば、本明細書に開示した機能をサポートするものであれば、任意のオペレーティング・システムを使用することができる。

【００１９】

メモリ１０６は、単一の実体であるランダム・アクセス・メモリとして示されている。しかし、メモリ１０６は、実際には複数のモジュールから構成され、そして高速レジスタ及びキャッシュから低速大容量のＤＲＡＭチップに至るまでの、複数のレベルにおいて存在可能であることを理解されたい。

【００２０】

ネットワーク・インタフェース装置１１０は、ネットワーク１３０を介してコンピュータ１０２と他のコンピュータとの間のネットワーク通信を可能にする。例えば、ネットワーク・インタフェース装置１１０は、ネットワーク・アダプタ又は他のネットワーク・インタフェース・カード（ＮＩＣ）である。

【００２１】

ストレージ１０８は、単一のユニットであるハードディスク・ドライブとして示されている。しかし、ストレージ１０８は、固定及び／又は取り外し可能なストレージ装置（例えば、ハードディスク・ドライブ、フレキシブルディスク・ドライブ、テープ・ドライブ、取り外し可能なメモリ・カード又は光ストレージ）の組み合わせとすることができる。メモリ１０６及びストレージ１０８は、複数の１次及び２次ストレージ装置をカバーする、１つの仮想アドレス空間の一部とすることができる。

【００２２】

入力装置１１４は、コンピュータ１０２に対し入力を提供するための任意の装置である。例えば、入力装置１１４として、キーボード、キーパッド、ライト・ペン、タッチ・スクリーン、トラック・ボール、音声認識ユニット、オーディオ／ビデオ・プレーヤ等を使用することができる。

【００２３】

出力装置１１６は、コンピュータ１０２のユーザに対し出力を提供するための任意の装置である。例えば、出力装置１１６は、ビデオ・カード及びサウンド・カード（図示せず）のようなインタフェース・カードを有する、通常のディスプレイ・スクリーン又は１セットのスピーカとすることができる。出力装置１１６は、入力装置１１４とは別々に示されているが、出力装置１１６及び入力装置１１４を組み合わせてもよい。例えば、統合タッチ・スクリーンを有するディスプレイ・スクリーン、統合キーボードを有するディスプレイ又はテキスト・スピーチ変換器と組み合わされた音声認識ユニットを使用することができる。

【００２４】

図示のように、メモリ１０６は、コンパイラ１４８及び装備化アプリケーション（instrumented application）１５２を含む。コンパイラ１４８は、マルチパス・プロファイラ１５０を含む。さらに、ストレージ１０８は、アプリケーション・コード（単に「コード」とも称する）１５４、プロファイル・データ１５６、粒度レベル（granularity levels

）１５８、ユーザ・オプション１６０、しきい値１６２及び注釈１６４を含む。コード１５４は、アプリケーションのソース・コード、当該アプリケーションのオブジェクト・コード及び当該ソース・コードの任意の中間コンパイラ表現を含むことができる。図２～図５及びこれらの図面に関連する説明は、コンピュータ１０２上で走るマルチパス・プロファイラ１５０の構造及び動作を詳述する。

【００２５】

本明細書では、コンパイラ１４８の一部であるマルチパス・プロファイラ１５０を参照して、実施形態を説明する。しかし、かかる説明は、他の実施形態（例えば、スタンド・アロンのマルチパス・プロファイラ１５０、リンカの一部であるマルチパス・プロファイラ１５０、及びコンパイラ１４８の一部であり且つリンカの一部でもあるマルチパス・プロファイラ１５０）も広くカバーすることを意図している。

【００２６】

図２は、本発明の実施形態に従った、図１のマルチパス・プロファイラ１５０のコンポーネントを示す。図示のように、マルチパス・プロファイラ１５０は、粒度マネージャ２１０、オプション・マネージャ２２０、しきい値マネージャ２３０、注釈マネージャ２４０及び領域エバリュエータ２５０を含む。

【００２７】

マルチパス・プロファイラ１５０は、コード１５４を受け取り、（第１の装備化サイクルの間に）注釈１６４をコード１５４に装備する。コンパイラ１４８は、注釈が付けられたコードに基づき、実行可能コードを生成する。この実行可能コードは、注釈が付けられていないコードから生じる実行可能コードから区別するために、これを「装備化」実行可能コードと称することがある。ユーザは、（例えば、サンプル・データセットを使用して）この装備化アプリケーションをランさせることにより、プロファイル・データ１５６を生成する。プロファイル・データ１５６は、アプリケーションの実行特性を記述する。例えば、当該アプリケーションの実行中に生成されるプロファイル・データ１５６は、ハードウェア性能カウンタに基づき、複数のコード領域の各々に関連するキャッシュ・ミスのカウントを含む。すなわち、この装備化は、プロファイリング目的のために、コード１５４を複数のコード領域に分割する。各コード領域は、コンパイラ１４８によって生成されるコードの１ブロックに対応する。また、各コード領域は、当該コードの高水準表現（例えば、ソース・コード）の対応する１ブロックにマッピングされる。さらに、前記複数のコード領域は、当該アプリケーションの少なくとも１つのコンパイル単位を（例えば、静的分析を介して）評価することにより、これを識別することができる。また、前記複数のコード領域は、これをコード領域の「候補セット」と称することもできる。

【００２８】

マルチパス・プロファイラ１５０は、プロファイル・データ１５６に基づき、（第２の装備化サイクルの間に）コード１５４を再装備化する。例えば、マルチパス・プロファイラ１５０は、「ホット」ブロック（例えば、予定のしきい値を超えるキャッシュ・ミスのカウントをもたらした、コードの１基本ブロック）を識別することができる。このような場合、マルチパス・プロファイラ１５０は、注釈１６４を使用して、識別されたホット・ブロック内の複数のメモリ参照を装備化する。コンパイラ１４８は、この再装備化コードに基づき、装備化実行可能コードを生成する。ユーザは、この装備化実行可能コードをランさせることにより、アプリケーションの実行特性を一層詳細なレベル（例えば、基本ブロックのレベルではなく、識別されたホット・ブロックのメモリ参照レベル）で記述する、追加のプロファイル・データ１５６を生成する。

【００２９】

さらに、マルチパス・プロファイラ１５０は、この新しく生成されたプロファイル・データ１５６に基づき、コード１５４を再装備化する。マルチパス・プロファイラ１５０は、任意の数の装備化サイクルを行うことにより、追加のプロファイル・データ１５６に基づき、注釈されたコードをさらに洗練化する。複数の装備化サイクルを提供する（その結果、漸進的及び反復的な動的プロファイリングをサポートする）ことにより、マルチパス

・プロファイラ１５０は、柔軟且つ効率的な態様で、不良コード領域を識別することができる。例えば、マルチパス・プロファイラ１５０は、（コード１５４における全てのメモリ参照とは対照的に）識別された不良コード領域内のメモリ参照だけをプロファイルすることにより、プロファイリング・オーバーヘッドを減少させることができる。

【００３０】

異なるサンプル・ランの間に、異なるタイプのハードウェア実行イベント情報が収集される。コード１５４のマルチパス動的プロファイリングは、異なるタイプ（すなわち、サンプル・ランごとに１つのタイプ）のハードウェア実行イベント情報に基づき、コード１５４を最適化することを可能にする。さらに、コード１５４のマルチパス動的プロファイリングは、累積的なプロファイリングを可能にする。すなわち、累積的なプロファイリングでは、１つ以上の先行するランからのプロファイル・データ１５６を使用して、後続するプロファイリングを洗練化する。さらに、コード１５４のマルチパス動的プロファイリングは、過剰な装備化（例えば、コード１５４の全ての装備化）を行うことなしに、最適化のためのコード領域を効率的に識別する。過剰な装備化は、観察されるアプリケーション行動の（プロファイリングのための）有用性を害することがある。例えば、過度に装備化されるアプリケーションの追加の時間及び空間要件は、非装備化アプリケーションを表すことがより少ない（すなわち、非装備化アプリケーションの行動をより不正確に反映する）、アプリケーション行動に結びつくことがある。

【００３１】

粒度マネージャ２１０は、アプリケーション・コード１５４のプロファイリングが行われるべき、複数の粒度レベル１５８を定義する。コード１５４は、アプリケーションのソース・コード、当該アプリケーションのオブジェクト・コード及び当該ソース・コードの任意の中間コンパイラ表現を含むことができる。粒度マネージャ２１０は、（例えば、入力装置１１４を介する）ユーザ入力に基づき、複数の粒度レベル１５８を定義することができる。表１は、粒度レベル１５８の１例を示す。

【表１】

粒度レベルの例

粒度レベル	プロファイリングの単位	説明
1	基本ブロック	粗粒度
2	メモリ参照	細粒度

【００３２】

この特定の例では、粒度マネージャ２１０は、２つの粒度レベル１５８を定義する。これらの粒度レベルは、コード１５４の基本ブロックをプロファイルするための第１の粒度レベル（粗粒度）と、コード１５４の個々のメモリ参照をプロファイルするための第２の粒度レベル（細粒度）とを含む。各粒度レベル１５８は、コード１５４の任意の単位（例えば、ソース・コード、オブジェクト・コード、中間コンパイラ表現等の単位）を意味することがある。例えば、粒度レベルは、プロファイリング単位として「手順」を指定する（その結果、コード１５４の各手順が個々にプロファイルされることを指定する）ことがある。

【００３３】

マルチパス・プロファイラ１５０は、選択された粒度レベル１５８に基づき、注釈１６４を異なった方法で使用して、コード１５４を装備化する。例えば、ユーザが粗粒度プロファイリングを選択すれば、マルチパス・プロファイラ１５０は、コード１５４の各基本ブロックの開始点及び終了点に注釈１６４を挿入することにより、コード１５４を装備化する。各注釈は、関数呼び出しを提供することにより、コード１５４がコンパイルされ、リンクされ、実行されるとき、プロファイル・データを提供する。例えば、コード１５４の各基本ブロックのキャッシュ・ミスをプロファイルするために、マルチパス・プロファ

イラ１５０は、
（１）各基本ブロックの開始点を通知するための関数呼び出し（例えば、__pdf_profile_PM_start()という名前の関数）を提供する注釈を、各基本ブロックの開始点に装備し、
（２）各基本ブロックの終了点を通知するための関数呼び出し（例えば、__pdf_profile_PM_end()という名前の関数）を提供する注釈を、各基本ブロックの終了点に装備する。
さらに、ユーザが細粒度プロファイリングを選択すれば、マルチパス・プロファイラ１５０は、コード１５４のメモリ参照ごとに注釈１６４を挿入することにより、コード１５４を装備化することができる。このような場合、マルチパス・プロファイラ１５０は、プロファイルすべき各メモリ参照の位置を通知するためのマーカ（例えば、__pdf_profile_cache_miss()という名前の関数）を提供する注釈を、各メモリ参照に装備する。注釈１６４については、注釈マネージャ２４０に関連して以下で説明する。

【００３４】

オプション・マネージャ２２０は、ユーザがアプリケーションのマルチパス動的プロファイリングを構成することを可能にする。表２は、ユーザ・オプション１６０を例示する。

【表２】

ユーザ・オプションの例

オプション	説明	条件
1	プロファイリング用の注釈を除去	-qnopdf（当該オプション用のデフォルト値）
2	粗粒度のプロファイリング	-qpdf1 及び以前のプロファイル・データの不在
3	細粒度のプロファイリング	-qpdf1 及び以前のプロファイル・データの存在
4	実行可能コードを最適化	-qpdf2

【００３５】

この特定の例では、オプション・マネージャ２２０は、マルチパス動的プロファイリングを構成するために４つのユーザ・オプション１６０を定義する。さらに、オプション・マネージャ２２０は、各ユーザ・オプション１６０を一の条件と関連付けることにより、一のユーザ・オプション１６０を適用することができる。表２に示すように、ユーザ・オプション１６０は、プロファイリング用の注釈を除去するためのオプションを含む（すなわち、ユーザがコンパイラ１４８に対し-qnopdfフラグを提供する場合）。また、ユーザ・オプション１６０は、粗粒度プロファイリングを行うためのオプションを含む（すなわち、ユーザがコンパイラ１４８に対し-qpdf1フラグを提供し、そして以前のプロファイル・データ１５６が使用不能である場合）。また、ユーザ・オプション１６０は、細粒度プロファイリングを行うためのオプションを含む（すなわち、ユーザがコンパイラ１４８に対し-qpdf1フラグを提供し、そして以前のプロファイル・データ１５６が使用可能である場合）。最後に、ユーザ・オプション１６０は、プロファイル・データ１５６を使用して、実行可能コードを最適化するためのオプションを含む（すなわち、ユーザがコンパイラ１４８に対し-qpdf2フラグを提供する場合）。当業者には明らかなように、本発明の実施形態は、本発明の範囲から逸脱することなく、他のユーザ・オプション及び条件をサポートするように適応させることができる。

【００３６】

しきい値マネージャ２３０は、アプリケーションの実行を最適化するための少なくとも１つのしきい値１６２を定義する。例えば、しきい値マネージャ２３０は、アプリケーションのプロファイリングを行う際に１つのパスから得られた情報をどのように使用して、後続するパスにおいて当該アプリケーションが装備化される（従って、プロファイルされ

る）方法をどのように修正すべきかを、ユーザが指定することを可能にする。表３は、しきい値１６２の例を示す。

【表３】

しきい値の例

ＩＤ	説明	条件
1	不良コード領域	＞一のコード領域からの２００個のキャッシュ・ミス
2	不良メモリ参照	＞一のメモリ参照からの７５個のキャッシュ・ミス

【００３７】

この特定の例では、アプリケーションを最適化するために、２つのしきい値１６２が指定されている。その１つは、コード領域用のものであり、他の１つは、個々のメモリ参照用のものである。第１に、２００個のキャッシュ・ミスのしきい値が、一のコード領域を不良として識別するために使用される。例えば、かかる領域は、コンパイラによって生成されたコードの１ブロック又はソース・コード・レベルの手順、機能、メソッド、モジュール等を含む。第２に、７５個のキャッシュ・ミスのしきい値が、一のメモリ参照を不良として識別するために使用される。実施形態では、コードの異なる領域を（コード領域用のしきい値に従って）プロファイルすることにより、コンパイル済み実行可能コード内でキャッシュ・ミスが生じるような広い領域が識別される。一旦識別されると、（メモリ参照用のしきい値に従った）連続的な細粒度プロファイリングを使用することにより、キャッシュ・ミスの原因を突き止めることができる。

【００３８】

或る場合には、少数のメモリ参照が、キャッシュ・ミス全体の大きな割合の原因となることがある。かかる多数のキャッシュ・ミスは、実行中アプリケーションの性能を著しく低下させることがある。不良コード領域を識別すると、コンパイラ１４８は、アプリケーションをより良く最適化することができる。例えば、コンパイラ１４８は、不良コード領域（具体的には、不良メモリ参照）に対しデータ・プリフェッチング及びデータ再編成技術を使用することにより、実行可能コードを生成することができる。従って、マルチパス・プロファイラ１５０は、装備化アプリケーション１５２を実行してプロファイル・データ１５６を評価することにより、キャッシュ・ミスを減少させるように最適化すべきコード１５４のサブセットを識別することができる。

【００３９】

注釈マネージャ２４０は、複数の注釈１６４を定義し、コード１５４に当該定義された複数の注釈１６４を装備する。表４は、注釈１６４の例を示す。

【表４】

注釈の例

ＩＤ	注釈のタイプ	注釈
1	高水準	unsigned int __pdf_profile_PM_start()
2	高水準	unsigned int __pdf_profile_PM_end()
3	高水準	void * __pdf_profile_cache_miss()
4	低水準	void __profile_cache_miss()
5	低水準	void __pdf_get_data()
6	最適化	void __mem_delay()
7	最適化	void __expect_value()

【００４０】

この特定の例では、注釈マネージャ２４０は、７つの注釈１６４を定義する。各注釈は

、コンパイル済みコードの実行中に、データをプロファイルするための関数呼び出しを提供する。この例では、注釈１６４は、高水準の注釈、低水準の注釈及び最適化注釈を含む。低水準のコンパイラは、これらの関数呼び出しを実行可能命令のインライン・シーケンスに変換する。高水準の注釈は、関数呼び出しを高水準のオプティマイザ・インタフェースに提供することにより、高水準プログラミング言語（例えば、当該アプリケーションのソース・コード又はその中間コンパイラ表現）におけるメモリ参照をプロファイルする。低水準の注釈は、関数呼び出しを低水準のオプティマイザ・インタフェースに提供することにより、（例えば、当該アプリケーションのソース・コード又はその中間コンパイラ表現における）特定のメモリ参照をプロファイルする。例えば、注釈マネージャ２４０は、低水準コードの生成中（例えば、高水準の中間表現に基づき、低水準の中間表現を生成中）に、高水準の注釈を低水準の注釈に変換する。すなわち、注釈マネージャ２４０は、低水準の注釈を、高水準の注釈を有する高水準コード（例えば、不良ステートメント）に対応する、低水準コード（例えば、ロード又はストア動作における不良メモリ参照）に関連付ける。言いかえれば、高水準の注釈及び低水準の注釈は、種々のコンパイル及び最適化段階（すなわち、異なるレベルのコード表現を含む段階）にわたって、これを維持することができる。さらに、コンパイラ１４８は、最適化用の注釈に基づき、コード１５４を最適化することができる。言いかえれば、最適化用の注釈は、コンパイラに対する「ヒント」として作用する。

【００４１】

低水準の注釈は、関数呼び出しを、仮パラメータ・リストを有するvoid __profile_cache_miss()に提供する。この仮パラメータ・リストは、メモリ・アドレス（例えば、void *addr）、カウンタ・アドレス（例えば、long long *counter）、ロード又はストア動作（例えば、unsigned int LoadStoreType）、メモリ参照タイプ（例えば、unsigned int dataType）及びメモリ参照長さ（例えば、unsigned int length）を含む。このメソッドは、次の動作を行う。

（１）性能カウンタを読み取る。

（２）指定されたdataType及びlengthを有する、LoadStoreTypeの動作を行う。

（３）再び性能カウンタを読み取り、性能カウンタが１だけ増加されているか否かをチェックする。

（４）そうであれば、１を加えることによりcounterを更新する（そうでなければ、リターンする）。

簡述すると、低水準の注釈は、アプリケーションの実行中に、プロファイル・データ（例えば、一のコード領域又はコード部分領域についてのキャッシュ・ミスのカウント）を収集する。実施形態では、コンパイラ１４８は、性能カウンタを直接的に読み取るためのコード・セグメントを生成することにより、プロファイリング・オーバーヘッドを減少させることができる。例えば、コンパイラ１４８は、関数呼び出しを、実行可能コードのインライン・シーケンスに変換することができる。

【００４２】

高水準の注釈は、粗粒度プロファイリングを行うための２つの関数呼び出し（すなわち、unsigned int __pdf_profile_PM_start()及びunsigned int __pdf_profile_PM_end()）を提供する。これらのメソッドは、一のイベント・タイプ（例えば、unsigned int eventType）を含む、仮パラメータ・リストを有する。例えば、マルチパス・プロファイラ１５０の第１パス中に、マルチパス・プロファイラ１５０は、粗粒度のキャッシュミス・プロファイリングを行うべき、コード１５４の複数の領域を識別する。マルチパス・プロファイラ１５０は、識別された複数の領域の各々を、（例えば、各領域の開始点及び終了点のそれぞれにおける）__pdf_profile_PM_start()及び__pdf_profile_PM_end()注釈内に囲むことができる。マルチパス・プロファイラ１５０の第２パス中に、ユーザが-qnopdfフラグを提供すれば、マルチパス・プロファイラ１５０は、コード１５４から全ての__pdf_profile_PM_start()及び__pdf_profile_PM_end()注釈を除去する。しかし、ユーザが-qpdf1フラグを提供すれば、マルチパス・プロファイラ１５０は、__pdf_profile_PM_start()及

び__pdf_profile_PM_end()注釈を、性能メトリクスＡＰＩのメソッドを起動するための注釈に変換する。さらに、ユーザが-qpdf2フラグを提供すれば、マルチパス・プロファイラ１５０は、__pdf_profile_PM_start()及び__pdf_profile_PM_end()の高水準の注釈を、低水準の__mem_delay()呼び出しに変換する。

【００４３】

また、高水準の注釈は、細粒度のキャッシュミス・プロファイリングを行うための関数呼び出し（void * __pdf_profile_cache_miss()）を提供する。このメソッドが有するパラメータ・リストは、内部マッピング用のシーケンス番号（例えば、unsigned int sequenceNumber）、メモリ・アドレス（例えば、void *addr）、ロード又はストア動作（例えば、unsigned int LoadStoreType）、メモリ参照タイプ（例えば、unsigned int dataType）及びメモリ参照長さ（例えば、unsigned int length）を含む。例えば、マルチパス・プロファイラ１５０の第１パス中に、マルチパス・プロファイラ１５０は、細粒度のキャッシュミス・プロファイリングを行うべき、コード１５４内の複数の命令を識別する。各命令は、メモリ・アドレスを参照することがある。マルチパス・プロファイラ１５０は、識別された命令ごとに（例えば、LoadStoreTypeがゼロである場合はメモリ・ロード参照ごとに、LoadStoreTypeが１である場合はメモリ・ストア動作ごとに）、__pdf_profile_cache_miss()注釈を挿入する。マルチパス・プロファイラ１５０の第２パス中に、ユーザが-qnopdfフラグを提供すれば、マルチパス・プロファイラ１５０は、コード１５４から全ての__pdf_profile_cache_miss()注釈を除去する。しかし、ユーザが-qpdf1フラグを提供すれば、マルチパス・プロファイラ１５０は、__profile_cache_miss()注釈を、低水準の__profile_cache_miss()注釈に変換する。さらに、ユーザが-qpdf2フラグを提供すれば、マルチパス・プロファイラ１５０は、高水準の__pdf_profile_cache_miss()注釈を、低水準の__mem_delay()注釈に変換する。

【００４４】

例示的に説明すると、これらの最適化用の注釈は、メモリ遅延注釈を含む。このメモリ遅延注釈は、メモリ参照アドレス（例えば、void *addr）及び予測される遅延サイクル数（例えば、unsigned int delayCycles）の仮パラメータ・リストを有する、void __mem_delay()に対し関数呼び出しを提供する。このメモリ遅延注釈は、（例えば、プロファイル・データ１５６に基づき）キャッシュ・ミスが生じそうな場所を指定する。例えば、このメモリ遅延注釈は、アプリケーションの実行可能コード（すなわち、マシン・コード）の不良メモリ参照に対応するコード１５４（又はコード１５４の中間コンパイラ表現）における一の命令のアドレスを指定する。しかし、これらのメモリ遅延注釈に基づきコード１５４に対しどのような最適化を導入すべきかという点については、コンパイラ１４８の裁量に任せることができる。言いかえれば、コード１５４のメモリ遅延注釈は、アプリケーションを最適化するに際し、コンパイラに対する「ヒント」として作用する。

【００４５】

また、これらの最適化用の注釈は、予測値注釈を含むことがある。この予測値注釈は、予測値（例えば、int expectedValue）の仮パラメータ・リストを有する、void __expect_value()に対し関数呼び出しを提供する。この予測値注釈は、一の表現の生じ得る値を指定するから、コンパイラ１５６は、当該予測値を有する当該表現を支持するように（例えば、分岐予測において）最適化のトレードオフを行うことができる。

【００４６】

領域エバリュエータ２５０は、注釈１６４及びプロファイル・データ１５６（第１パスの後にプロファイル・データが使用可能である場合）に基づき、コード１５４の複数の領域（又は部分領域）を装備化する。例えば、領域エバリュエータ２５０は、表３のしきい値１６２と対照してプロファイル・データ１５６を評価することにより、装備化すべきコード１５４の複数の領域を決定するとともに、高水準（粗粒度の分析用）又は低水準の注釈（細粒度の分析用）のどちらを使用すべきであるかを決定する。

【００４７】

図３は、本発明の実施形態に従った、アプリケーションの実行を最適化するための方法

３００を示すフローチャートである。図示のように、方法３００はステップ３１０で開始し、そこでユーザは、プロファイリングを行うべきであることを指示するコンパイラ・フラグ（例えば、-qpdf1）を使用して、ソース・コード１５４をコンパイル及びリンクする。これに応答して、マルチパス・プロファイラ１５０は、コンパイル・プロセスの一部として、粗粒度プロファイリング用の注釈を、当該コードに装備する。次に、ユーザは、サンプル入力３２０のような代表的なデータセットを使用して、装備化アプリケーション１５２の１つ以上のサンプル・ランを実行する。装備化アプリケーション１５２は、その実行に応じて、プロファイル・データ１５６を生成する、例えば、装備化アプリケーション１５２は、キャッシュ・ミスの数を記録する。代表的なデータセットを使用して、装備化アプリケーション１５２の１つ以上のサンプル・ランを実行した後、ユーザは、ステップ３１０に戻ることにより、-qpdf1コンパイラ・フラグを使用してソース・コード１５４を再コンパイル及び再リンクする。これに応答して、ステップ３１０で、マルチパス・プロファイラ１５０は、粗粒度プロファイリングの結果に基づき、コードに細粒度プロファイリング用の注釈を装備する。ユーザは、（複数のサンプル・データセットを使用して）装備化アプリケーション１５２のプロファイリング及び実行を継続することができる。

【００４８】

代表的なデータセットを使用して、装備化アプリケーション１５２の１つ以上のサンプル・ランを実行した後、ユーザは、ステップ３３０に進むことにより、-qpdf2コンパイラ・フラグを使用してソース・コード１５４を再コンパイル及び再リンクする。ステップ３３０で、コンパイラ１４８は、使用可能なプロファイル・データ１５６及び注釈１６４に基づき（例えば、__mem_delay() 呼び出しを生成することにより）、最適化されたアプリケーション３４０を生成する。ステップ３４０の後、本方法３００は終了する。

【００４９】

図４は、本発明の実施形態に従った、アプリケーションの実行を最適化するための方法４００を示すフローチャートである。図示のように、方法４００はステップ４１０で開始し、そこでマルチパス・プロファイラ１５０は、アプリケーションの複数のコード領域に、当該複数のコード領域の各々ごとのプロファイル・データを生成するための注釈を装備する。例えば、マルチパス・プロファイラ１５０は、コンパイル済みコードが実行されるときにプロファイル・データを収集するように、コード１５４の複数の基本ブロックを装備化する。ステップ４２０で、装備化アプリケーションは、これらのコード領域用のプロファイル・データ１５６を（装備化コンパイル済みコードの実行を介して）生成する。例えば、このプロファイル・データ１５６は、装備化コードによってモニタされるハードウェア性能カウンタに基づき、コード領域ごとのキャッシュ・ミスのカウントを含む。

【００５０】

ステップ４３０で、マルチパス・プロファイラ１５０は、プログラムの実行中に生成されたプロファイル・データ１５６に基づき、一の不良コード領域を識別する。例えば、マルチパス・プロファイラ１５０は、指定されたしきい値と対照してプロファイル・データ１５６を評価する。ステップ４４０で、マルチパス・プロファイラ１５０は、コンパイル済みコードが実行されるときにプロファイル・データを収集するように、前記不良コード領域の複数のコード部分領域に注釈を装備する。例えば、マルチパス・プロファイラ１５０は、コンパイル済みコードが実行されるときに前記不良コード領域内のメモリ参照ごとのプロファイル・データ１５６が収集されるように、当該メモリ参照を装備化する。

【００５１】

ステップ４５０で、ステップ４４０でコンパイル及びリンクされたアプリケーションが実行される。前述のように、アプリケーションは、その実行中に、前記各メモリ参照用のプロファイル・データ１５６を生成する。例えば、プロファイル・データ１５６は、ハードウェア性能カウンタに基づき、メモリ参照ごとのキャッシュ・ミスのカウントを含む。ステップ４６０で、マルチパス・プロファイラ１５０は、これらのコード部分領域のために収集されたプロファイル・データから、一の不良コード部分領域を識別する。

【００５２】

ステップ４７０で、ユーザは、コンパイラの最適化をガイドするために、しきい値を超えるキャッシュ・ミスに帰着するコードの部分領域をアドレスするために、コードがどのようにコンパイルされるかを指示することができる。また、ユーザは、注釈（例えば、__mem_delay()注釈）を手動的に追加することにより、コンパイラの最適化をガイドすることができる。例えば、マルチパス・プロファイラ１５０は、識別された不良コード部分領域及び生成されたプロファイル・データ１５６に基づき、アプリケーションをコンパイルすることにより、最適化されたアプリケーションを生成する。

【００５３】

図５は、本発明の実施形態に従った、マルチパス動的プロファイリングを行うための方法５００を示すフローチャートである。図示のように、方法５００はステップ５１０で開始し、そこでユーザは、マルチパス・プロファイラ１５０の第１パスを開始する。例えば、コンパイラ１４８は、アプリケーションのソース・コードをコンパイル及びリンクすべしという命令を受け取ることができる。この命令に含まれるコンパイラ・フラグは、コンパイラに対し、コンパイルされ且つリンクされたコード内に（例えば、-qpdf1フラグを使用して）プロファイリング用の装備化を含めるように指示する。ステップ５２０で、マルチパス・プロファイラ１５０は、粗粒度プロファイリング又は細粒度プロファイリングのどちらを行うべきかを決定する。例えば、マルチパス・プロファイラ１５０は、表２の条件として示される以前のプロファイル・データが使用可能であるか否かを評価する。もし、以前のプロファイル・データが使用可能でなければ、マルチパス・プロファイラ１５０は、粗粒度のアプローチを使用して、動的プロファイリングの第１パスを行う。もし、以前のプロファイル・データが使用可能であれば、マルチパス・プロファイラ１５０は、動的プロファイリングの先行するパスから得られた以前のプロファイル・データに基づき、追加の細粒度プロファイリングを行う。

【００５４】

マルチパス・プロファイラが粗粒度プロファイリングを行うことを決定すれば、ステップ５３０で、マルチパス・プロファイラ１５０は、プロファイリング関数（例えば、__pdf_profile_PM_start() 及び __pdf_profile_PM_end()）に対する呼び出しを、コードに装備する。例えば、マルチパス・プロファイラ１５０は、コードの複数のブロックを識別するとともに、対応する複数のコンパイル単位を一体化ブロックとしてプロファイルするための呼び出しを（例えば、表３の注釈を使用して）当該複数のコンパイル単位に装備する。サンプル・データを使用してコードを実行した後、ステップ５４０で、ユーザは、マルチパス・プロファイラ１５０の第２パスを開始し、一のフラグを再び指定することにより、アプリケーション内でどのプロファイリングを行うべきかを指示する。これに応答して、マルチパス・プロファイラ１５０は、ユーザによって提供されたフラグを識別する。もし、このフラグが（例えば、-qnopdfコンパイラ・フラグを使用して）プロファイリングを行うべきでないことを指示すれば、ステップ５６０で、マルチパス・プロファイラ１５０は、当該プロファイリング関数への参照（例えば、__pdf_profile_PM_start() 及び __pdf_profile_PM_end()呼び出し）を除去する。もし、このフラグが（例えば、-qpdf1フラグを使用して）動的プロファイリングを継続すべきことを指示すれば、ステップ５６２で、マルチパス・プロファイラ１５０は、ステップ５３０からの注釈を、（例えば、全ての以前のプロファイル・データ１５６をマージするための）__pdf_get_data()呼び出しに変換する。もし、このフラグが-qpdf2であれば、本方法５００はステップ５６４に進み、そこでコンパイラ１４８は、プロファイル・データ１５６及び注釈に基づき、__mem_delay()呼び出しを生成する。すなわち、-qpdf2フラグは、コンパイラ１４８に対し、使用可能なプロファイル・データ１５６に基づき、コード１５４をコンパイルし且つ最適化するように命令するために使用される。

【００５５】

しかし、ステップ５２０で、マルチパス・プロファイラ１５０が細粒度プロファイリングを行うことを決定すれば、ステップ５３５で、マルチパス・プロファイラ１５０は、__pdf_profile_cache_miss()呼び出しを生成する。例えば、マルチパス・プロファイラ１５

０は、コード１５４に、表３の対応する注釈を付ける。ステップ５４５で、ユーザは、マルチパス・プロファイラ１５０の第２パスを開始する。例えば、コンパイラ１４８は、ユーザからコード１５４及び一のフラグを受け取る。ステップ５５５で、マルチパス・プロファイラ１５０は、ユーザによって提供されたフラグを識別する。もし、このフラグが-qnopdf であれば（すなわち、表２のシナリオ１に対応）、本方法５００はステップ５６６に進み、そこでマルチパス・プロファイラ１５０は、コード１５４から全ての__pdf_profile_cache_miss()呼び出しを除去する。もし、このフラグが-qpdf1であれば、本方法５００はステップ５７０に進み、そこでマルチパス・プロファイラ１５０は、ステップ５３５からの注釈を__profile_cache_miss()呼び出しに変換する。もし、このフラグが-qpdf2であれば、本方法５００はステップ５６８に進み、そこでマルチパス・プロファイラ１５０は、ステップ５３５からの注釈を__mem_delay()呼び出しに変換する。

【００５６】

ステップ５６０又はステップ５６６の後、コンパイラ１４８は、コード１５４のみに基づき、アプリケーションを生成する。ステップ５６２又はステップ５７０の後、コンパイラ１４８は、コード１５４及びステップ５６２又はステップ５７０からの注釈に基づき、装備化アプリケーション１５２を生成する。ステップ５６４又はステップ５６８で、コンパイラ１４８は、コード１５４及びステップ５６４又はステップ５６８からの注釈に基づき、最適化されたアプリケーションを生成する。

【００５７】

マルチパス・プロファイラ１５０は、コンパイル中に装備化を行うことができる（コンパイル・ステップの装備とも称する）。例えば、ユーザが-qpdf1フラグを提供すれば、マルチパス・プロファイラ１５０は、アプリケーション用の以前のプロファイル・データ１５６をチェックする。もし、以前のプロファイル・データ１５６が存在すれば、マルチパス・プロファイラ１５０は、全ての以前のプロファイル・データ１５６をマージするか又は当該以前のプロファイル・データ１５６を使用する（例えば、細粒度プロファイリングを行うために粗粒度のプロファイル・データを使用する）ことにより、不良コード領域の識別を改良／洗練化する。もし、ユーザが-qpdf2フラグを提供すれば、マルチパス・プロファイラ１５０は、プロファイル・データ１５６及び先行する装備化サイクルからの注釈１６４に基づき、注釈（例えば、一の不良メモリ参照用の__mem_delay()及び一の表現用の__expect_value()）を生成する。コンパイラ１４８は、これらの生成された注釈及びプロファイル・データ１５６に基づき、アプリケーションを最適化する。最適化の例は、インライニング、クローニング、アウトライニング、間接呼び出しの特化（indirect call specialization）、不良ロード主導型データ・プリフェッチング（delinquent-load-driven data prefetch）、データ再編成、命令スケジューリング等を含む。

【００５８】

また、マルチパス・プロファイラ１５０は、アプリケーションのコンパイル後に装備化を行うことができる（リンク・ステップの装備化とも称する）。例えば、マルチパス・プロファイラ１５０は、全てのコンパイル単位に基づき、呼び出しグラフを生成する。もし、ユーザが-qnopdfフラグを提供すれば、マルチパス・プロファイラ１５０は、コンパイル中に生成された全ての注釈１６４を除去する。もし、ユーザが-qpdf1フラグを提供すれば、マルチパス・プロファイラ１５０は、各コンパイル単位用のメモリを予約するとともに、識別されたコード領域（又は識別されたコード部分領域）ごとにメモリ・マッピングを行う。また、マルチパス・プロファイラ１５０は、動的プロファイリング用の呼び出しを生成する。さらに、マルチパス・プロファイラ１５０は、コンパイルからの任意の最適化を調査することにより、冗長なプロファイリングを除去する。次に、マルチパス・プロファイラ１５０は、低水準オプティマイザとインタフェースすることにより、注釈１６４を命令シーケンスに拡張するか又は実行時ライブラリへの呼び出しを生成する。すなわち、これらの注釈は、プログラム実行中にプロファイル・データを収集する実際の命令（又は関数への呼び出し）で置き換られる。

【００５９】

図６は、本発明の実施形態に従った、マルチパス動的プロファイリングを行うための方法６００を示すフローチャートである。図示のように、本方法６００はステップ６１０で開始し、そこでユーザは、コンパイラ１４８を起動することにより、アプリケーション用のソース・コードをコンパイルする。ステップ６２０～６５０は、実施形態に従って、コンパイラ１４８の高水準オプティマイザ６１２によって行うことができる。ステップ６２０で、高水準オプティマイザ６１２は、ユーザが装備化を望むか否かを決定する。そうでなければ、本方法６００はステップ６２５に進み、そこで高水準オプティマイザ６１２は、ソース・コード及び任意のプロファイル・データ１５６に基づき、アプリケーション用の中間コードを最適化し且つ生成する。

【００６０】

しかし、ユーザが装備化を望むのであれば、本方法６００はステップ６３０に進み、そこで高水準オプティマイザ６１２は、以前のプロファイル・データ１５６が存在するか否かを決定する。そうでなければ、本方法６００はステップ６４０に進み、そこで高水準オプティマイザ６１２は、キャッシュ・ミスについてプロファイルすべきコード領域を識別する（すなわち、粗粒度プロファイリングを行う）。しかし、以前のプロファイル・データ１５６が存在すれば、本方法６００はステップ６３５に進み、そこで高水準オプティマイザ６１２は、プロファイル・データ１５６に基づき、キャッシュ・ミスについてプロファイルすべき個々のメモリ参照を識別する（すなわち、細粒度プロファイリングを行う）。ステップ６４０又はステップ６３５の後、本方法６００はステップ６５０に進み、そこで高水準オプティマイザ６１２は、アプリケーション用の装備化中間コードを生成する。

【００６１】

ステップ６５０又はステップ６２５の後、本方法６００はステップ６６０に進み、そこでコンパイラ１４８の低水準オプティマイザは、装備化中間コードに基づき、アプリケーション用の１つ以上のオブジェクト・ファイルを生成する。ステップ６６０の後、リンカは、当該１つ以上のオブジェクト・ファイル及び任意のライブラリをリンクすることにより、プログラム（実行可能コード）を生成する。もし、このプログラムが（ステップ６２５及び６６０から生成された）最適化済みのプログラムであれば、ステップ６７０で、ユーザは、オプションとして、当該最適化済みのプログラムを実行する。しかし、当該最適化済みのプログラムは、プロファイル・データ１５６を生成しないことがある。すなわち、もし、ユーザが最終的にコンパイルすることを決定すれば（すなわち、ステップ６２５及び６６０）、「実行」ステップ６７０を行う必要はない。一方、このプログラムが（ステップ６５０及び６６０から生成された）装備化プログラムであれば、ステップ６７０で、ユーザは、サンプル・データを使用して当該装備化プログラムを実行することにより、プロファイル・データ１５６を生成する。ステップ６７０の後、ユーザがプロファイリングを継続することを望むのであれば、本方法６００はステップ６１０に戻る。そうでなければ、本方法６００は終了する。

【００６２】

もちろん、本明細書に開示した実施形態は、例示を目的とするものであって、本発明を制限するものではないことが意図される。他の実施形態も、広く予測されるからである。当業者には明らかなように、本発明の実施形態は、例えば、他の粒度レベル、ユーザ・オプション、しきい値及び注釈をサポートするように、これを適合させることができる。さらに、本発明の実施形態は、コードに注釈を付ける他の方法をサポートするように、これを適合させることができる。例えば、マルチパス・プロファイラは、コードの中間表現を生成することができる。次に、マルチパス・プロファイラは、コードの中間表現に注釈を付けて、コードそれ自体をそのままにしておくことができる。さらに、本発明の実施形態は、キャッシュミス・プロファイリングとは異なるプロファイリングのタイプ（例えば、ブロック・カウントのプロファイリング及び値のプロファイリング）をサポートするように、これを適合させることができる。

【００６３】

本発明の実施形態は、アプリケーションの実行を最適化するという点で有利である。１

つの実施形態では、マルチパス・プロファイラは、アプリケーションの複数のコード領域に、当該複数のコード領域の各々ごとのプロファイル・データを生成するための注釈を装備する。ユーザが装備化コード領域を有するアプリケーションを実行するとき、これらの注釈は、前記複数のコード領域の各々のためのプロファイル・データを生成する。マルチパス・プロファイラは、前記複数のコード領域の各々ごとに生成されたプロファイル・データに基づき、一の不良コード領域を識別する。さらに、マルチパス・プロファイラは、前記識別された不良コード領域の複数のコード部分領域に、当該複数のコード部分領域の各々ごとのプロファイル・データを生成するための注釈を装備する。ユーザが装備化コード部分領域を有するアプリケーションを実行するとき、これらの注釈は、前記複数のコード部分領域の各々ごとのプロファイル・データを生成する。マルチパス・プロファイラは、前記複数のコード部分領域の各々ごとに生成されたプロファイル・データに基づき、一の不良コード部分領域を識別する。コンパイラは、当該識別された不良コード部分領域を使用して、アプリケーションの実行を最適化する。

【００６４】

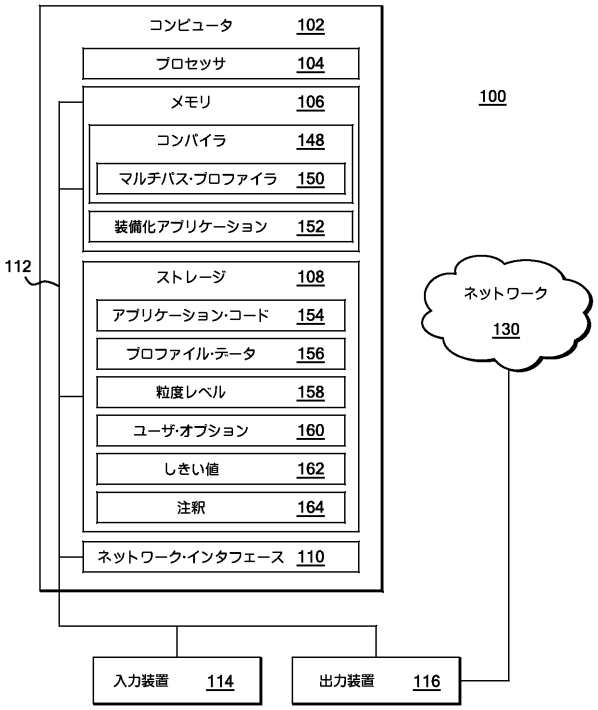
前述の説明は、本発明の実施形態に向けられているが、本発明の基本的な範囲から逸脱することなく、他の実施形態を考案することができる。本発明の範囲は、以下の請求項の記載によって決定される。

【符号の説明】

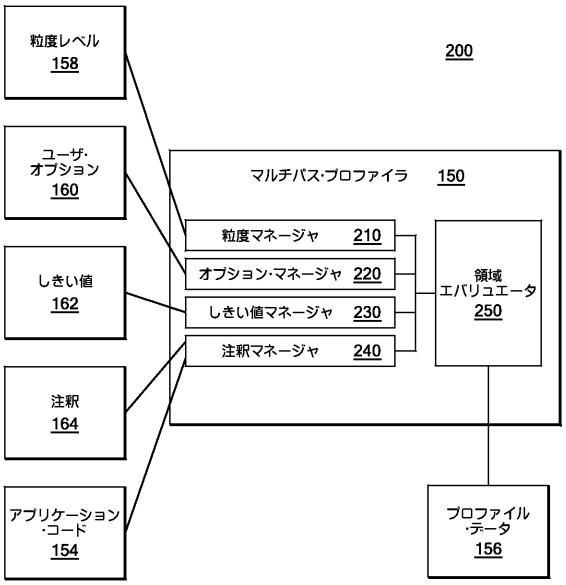
【００６５】

１００・・・アプリケーションの実行を最適化するためのシステム
１０２・・・コンピュータ
１０４・・・プロセッサ
１０６・・・メモリ
１０８・・・ストレージ
１４８・・・コンパイラ
１５０・・・マルチパス・プロファイラ
１５２・・・装備化アプリケーション
１５４・・・アプリケーション・コード
１５６・・・プロファイル・データ
１５８・・・粒度レベル
１６０・・・ユーザ・オプション
１６２・・・しきい値
１６４・・・注釈
２１０・・・粒度マネージャ
２２０・・・オプション・マネージャ
２３０・・・しきい値マネージャ
２４０・・・注釈マネージャ
２５０・・・領域エバリュエータ

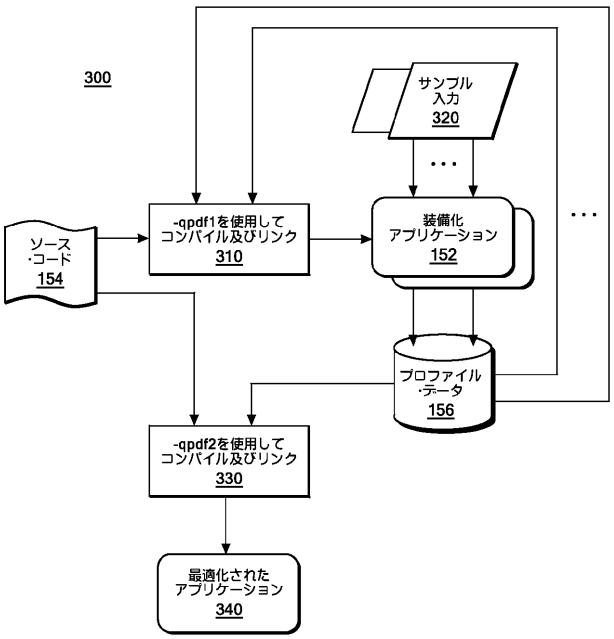
【図１】


コンピュータ 102
プロセッサ 104
メモリ 106
コンパイラ 148
マルチパス・プロファイラ 150
装備化アプリケーション 152
ストレージ 108
アプリケーション・コード 154
プロファイル・データ 156
粒度レベル 158
ユーザ・オプション 160
しきい値 162
注釈 164
ネットワーク・インタフェース 110
100
112
ネットワーク 130
入力装置 114
出力装置 116

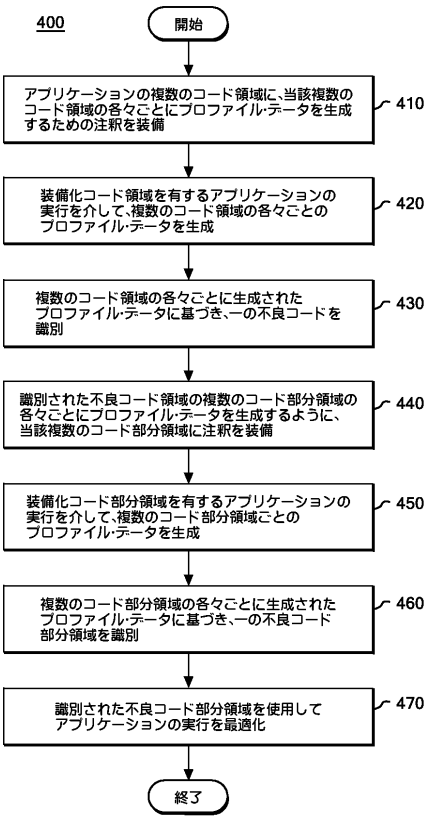
【図２】


粒度レベル 158
ユーザ・オプション 160
しきい値 162
注釈 164
アプリケーション・コード 154
200
マルチパス・プロファイラ 150
粒度マネージャ 210
オプション・マネージャ 220
しきい値マネージャ 230
注釈マネージャ 240
領域エバリュエータ 250
プロファイル・データ 156

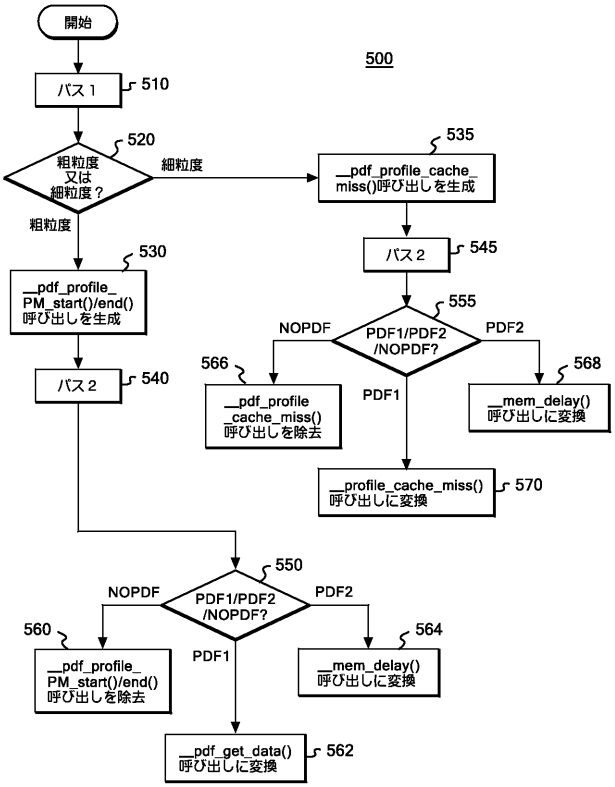
【図３】


300
サンプル入力 320
ソース・コード 154
-qpdf1を使用してコンパイル及びリンク 310
装備化アプリケーション 152
プロファイル・データ 156
-qpdf2を使用してコンパイル及びリンク 330
最適化されたアプリケーション 340

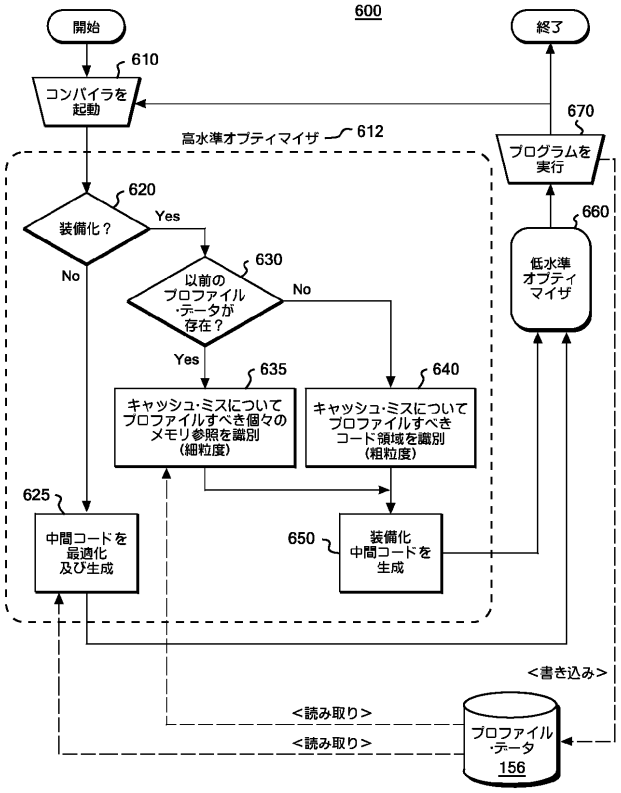
【図４】


400
開始
アプリケーションの複数のコード領域に、当該複数のコード領域の各々ごとにプロファイル・データを生成するための注釈を装備
410
装備化コード領域を有するアプリケーションの実行を介して、複数のコード領域の各々ごとのプロファイル・データを生成
420
複数のコード領域の各々ごとに生成されたプロファイル・データに基づき、一の不良コードを識別
430
識別された不良コード領域の複数のコード部分領域の各々ごとにプロファイル・データを生成するように、当該複数のコード部分領域に注釈を装備
440
装備化コード部分領域を有するアプリケーションの実行を介して、複数のコード部分領域ごとのプロファイル・データを生成
450
複数のコード部分領域の各々ごとに生成されたプロファイル・データに基づき、一の不良コード部分領域を識別
460
識別された不良コード部分領域を使用してアプリケーションの実行を最適化
470
終了

【図５】


500
開始
パス１
510
粗粒度又は細粒度？
520
細粒度
粗粒度
__pdf_profile_cache_miss()呼び出しを生成
535
パス２
545
__pdf_profile_PM_start()/end()呼び出しを生成
530
パス２
540
PDF1/PDF2/NOPDF?
555
NOPDF
PDF2
PDF1
__pdf_profile_cache_miss()呼び出しを除去
566
__mem_delay()呼び出しに変換
568
__profile_cache_miss()呼び出しに変換
570
PDF1/PDF2/NOPDF?
550
NOPDF
PDF2
PDF1
__pdf_profile_PM_start()/end()呼び出しを除去
560
__mem_delay()呼び出しに変換
564
__pdf_get_data()呼び出しに変換
562

【図６】


600
開始
終了
コンパイラを起動
610
高水準オプティマイザ
612
プログラムを実行
670
装備化？
620
Yes
No
以前のプロファイル・データが存在？
630
No
Yes
低水準オプティマイザ
660
キャッシュ・ミスについてプロファイルすべき個々のメモリ参照を識別（細粒度）
635
キャッシュ・ミスについてプロファイルすべきコード領域を識別（粗粒度）
640
中間コードを最適化及び生成
625
装備化中間コードを生成
650
<書き込み>
<読み取り>
<読み取り>
プロファイル・データ
156

フロントページの続き

(72)発明者　ヤオチン・ガオ
カナダＬ６Ｇ　１Ｃ７　オンタリオ州　マーカム　ウォーデン・アヴェニュー　８２００
(72)発明者　ラウル・エステバン・シルベラ
カナダＬ６Ｇ　１Ｃ７　オンタリオ州　マーカム　ウォーデン・アヴェニュー　８２００
(72)発明者　ロシュ・ジョージズ・アーシャンボー
カナダＬ６Ｇ　１Ｃ７　オンタリオ州　マーカム　ウォーデン・アヴェニュー　８２００
(72)発明者　グラハム・ユー
カナダＬ６Ｇ　１Ｃ７　オンタリオ州　マーカム　ウォーデン・アヴェニュー　８２００
(72)発明者　マーク・ピーター・メンデル
カナダＬ６Ｇ　１Ｃ７　オンタリオ州　マーカム　ウォーデン・アヴェニュー　８２００
(72)発明者　アラン・ラッセル・マーティン
カナダＭ４Ｊ　１Ｅ９　オンタリオ州　トロント　クイーン・ビクトリア・ストリート　１－８

Ｆターム(参考) 5B081 CC21