(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3894698号
(P3894698)
(45)発行日　平成19年3月22日(2007.3.22)　(24)登録日　平成18年12月22日(2006.12.22)

(51)Int.Cl.　F I
G06F 9/45 (2006.01)　G06F 9/44 3 2 0 C
G06F 9/44 3 2 2 A

請求項の数 7　(全 12 頁)

(21)出願番号　特願2000-64480(P2000-64480)
(22)出願日　平成12年3月9日(2000.3.9)
(65)公開番号　特開2000-267862(P2000-267862A)
(43)公開日　平成12年9月29日(2000.9.29)
審査請求日　平成14年8月22日(2002.8.22)
(31)優先権主張番号　264755
(32)優先日　平成11年3月9日(1999.3.9)
(33)優先権主張国　米国(US)

(73)特許権者　398038580
ヒューレット・パッカード・カンパニー
HEWLETT-PACKARD COMPANY
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000
(74)代理人　100063897
弁理士　古谷　馨
(74)代理人　100076680
弁理士　溝部　孝彦
(74)代理人　100087642
弁理士　古谷　聡

最終頁に続く

(54)【発明の名称】資源の消費を最小限にするハイブリッド式ジャストインタイム・コンパイラ

(57)【特許請求の範囲】
【請求項１】
ハードウェアプラットフォーム上でクロスプラットフォームコードのセットの実行を可能とする仮想マシンであって、
前記ハードウェアプラットフォーム用のネイティブコードブロックにコンパイルされる前記クロスプラットフォームコードに含まれる基本ブロックを識別するブロック検出器と、
前記ブロック検出器からの識別に応答して、前記ネイティブコードブロックに前記基本ブロックをコンパイルするコードジェネレータと、
前記基本ブロックの代わりに前記ネイティブコードブロックを実行し、及び、前記ブロック検出器によって識別されなかったクロスプラットフォームコードの残りの部分を解読することによって、前記クロスプラットフォームコードを実行するインタープリタ
とを備え、
前記コードジェネレータと前記インタープリタとの両方が、同じテーブルにアクセスし、
前記テーブルは、前記クロスプラットフォームコードにおいて使用可能な演算コードのセットのそれぞれを、前記ハードウェアプラットフォームのネイティブコードに対応する前記アクションコードに関連付けることからなる、仮想マシン。
【請求項２】
前記ブロック検出器は、前記クロスプラットフォームコードに含まれるメソッドにおけ

るプログラムループを識別することによって前記基本ブロックを識別することからなる、請求項１の仮想マシン。

【請求項３】

前記ブロック検出器は、前記クロスプラットフォームコードにおいて複数回呼び出されるメソッドを識別することによって前記基本ブロックを識別することからなる、請求項１の仮想マシン。

【請求項４】

前記ブロック検出器は、前記クロスプラットフォームコードにおいて複数回呼び出される関数を識別することによって前記基本ブロックを識別することからなる、請求項１の仮想マシン。

【請求項５】

前記コードジェネレータが、前記ネイティブコードブロックを最適化する、請求項１乃至４のいずれかの仮想マシン。

【請求項６】

前記ブロック検出器は、前記クロスプラットフォームコードにおける前記基本ブロックの境界のセットを識別することによって前記基本ブロックを識別することからなる、請求項１の仮想マシン。

【請求項７】

前記コードジェネレータは、前記境界に応答して、前記基本ブロックをコンパイルする、請求項６の仮想マシン。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、処理システムの分野に関する。より詳細には、本発明は、資源の消費を最小限にするハイブリッド式ジャストインタイム・コンパイラに関する。

【０００２】

【従来の技術】

処理資源を埋め込んだコンピュータシステムおよび装置は、典型的には、種々の異なるアーキテクチャの１つに準拠する。通常、各アーキテクチャを、ある特定の命令セット、ハードウェアレジスタセットおよびメモリ構成等によって定義することもできる。アーキテクチャをソフトウェア実行のためのハードウェアプラットフォームという場合もある。アプリケーションプログラム等のソフトウェアは、ある特定のハードウェアプラットフォーム上で実行するために書き込まれ、またはコンパイルされ、ネイティブコードと呼ばれることもある。ある特定のハードウェアプラットフォームのネイティブコードにおけるアプリケーションプログラムは、通常、互換性のない他のハードウェアプラットフォーム上では動作しない。

【０００３】

ソフトウェア環境の中には、種々の異なるハードウェアプラットフォーム上でアプリケーションプログラムを実行できるようにしたものがある。このようなソフトウェア環境下で実行するアプリケーションプログラムは、通常、ソフトウェア環境により支援される予め定義された命令セットにそれぞれが準拠する命令列の形式を取る。かかるソフトウェア環境は、典型的には、命令列の各命令を解読し、特定のハードウェアプラットフォームのネイティブコードにおける命令のエミュレーションを行う。ソフトウェア環境自体は、ネイティブコードで実行する仮想マシン（バーチャルマシン）の形式を取ることもできる。

【０００４】

このようなソフトウェア環境の一例は、Java仮想マシンである。典型的なJava仮想マシンは、Javaアプリケーションプログラム用インタープリタとして機能する。Javaアプリケーションプログラムは、典型的には、Javaバイトコード命令列の形式を取り、Java仮想マシンは、特定のハードウェアプラットフォーム（このプラットフォーム下でJava仮想マシンが実行する）のネイティブコードを使用することで、各Javaバイトコード命令をエミュレ

ートする。残念ながら、この種のエミュレーションは、通常、ネイティブコードにおけるアプリケーションプログラムと比較して、命令実行性能が大幅に低下する。

【０００５】

このようなソフトウェア環境において命令実行性能を高める１つの従来の方法は、ジャストインタイム・コンパイラと呼ばれるものをソフトウェア環境に備えるものである。典型的なジャストインタイム・コンパイラは、アプリケーションプログラムをネイティブコードにコンパイルするプロセスである。ジャストインタイム・コンパイラは、アプリケーションプログラムを解読する仮想マシンプロセスと同時に、またはそれに逐次的に実行することができる。典型的には、ジャストインタイム・コンパイラは、アプリケーションプログラムのネイティブコードバージョンを生成するが、これは、その後、アプリケーションプログラムを連続して動作させる場合に、それを実行するために利用することができる。

【０００６】

【発明が解決しようとする課題】

しかし、従来のジャストインタイム・コンパイラは、一般的に、メモリおよびプロセッササイクル等のリソース（資源）を大量に消費し、このため、リソースが比較的制限される装置にはあまり適さない。

【０００７】

【課題を解決するための手段】

ハードウェアプラットフォーム上でクロスプラットフォームコードのセットの実行を可能とするハイブリッド式ジャストインタイム・コンパイラを備えた仮想マシンを開示する。従来のジャストインタイム・コンパイラに対して、本明細書中で開示されるジャストインタイム・コンパイラは、クロスプラットフォームコードの選択された基本ブロックのみをコンパイルし、かつ他の場合にはクロスプラットフォームを解読するために使用される仮想マシンの既存の構成要素を活用することによって、ハードウェアプラットフォームの資源の消費を比較的少なくする。このジャストインタイム・コンパイラは、クロスプラットフォームの選択された基本ブロックのみをコンパイルする一方で、残りの部分を解読するという点でハイブリッド式である。

【０００８】

一実施形態において、ハイブリッド式ジャストインタイム・コンパイラを備えた仮想マシンは、クロスプラットフォームコードの選択された基本ブロックをハードウェアプラットフォーム用のネイティブコードブロックにコンパイルするコードジェネレータを備える。また、本実施形態における仮想マシンは、選択された基本ブロックの代わりにネイティブコードブロックを実行し、かつクロスプラットフォームコードの残りの部分を解読することによって、クロスプラットフォームコードを実行するインタープリタをさらに備える。該インタープリタによって使用される既存のアクションテーブルは、ネイティブコードブロックをコンパイルするために、コードジェネレータによっても使用される。

【０００９】

本発明を、その特定の典型的な実施形態に関して図面を参照して説明するが、本発明の上記以外の特徴および利点は、以下の詳細な説明から明らかとなろう。

【００１０】

【発明の実施の形態】

図１に、本発明に従うハイブリッド式ジャストインタイム・コンパイラを備えた仮想マシン１２を含む装置１０を示す。仮想マシン１２により、装置１０が実装されている特定のハードウェアプラットフォーム上でクロスプラットフォームコード１４のセットを実行することが可能となる。仮想マシン１２は、クロスプラットフォームコード１４を解読し、かつクロスプラットフォームコード１４の選択した部分をコンパイルすることで、コンパイルにより消費される装置１０の資源量を最小限にしつつ命令実行性能を高める。

【００１１】

クロスプラットフォームコード１４は、ハードウェアプラットフォーム上で実行可能な任意のソフトウェアであって、これに含まれる命令の解読に適する仮想マシン環境を提供す

るソフトウェアである。クロスプラットフォームコード１４は、アプリケーションプログラムまたはロード可能なクラスファイル、あるいは個別の関数ないしメソッド（method）とすることができる。一実施形態において、クロスプラットフォームコード１４は、Javaバイトコードにおけるアプリケーションプログラムであり、仮想マシン１２は、Java仮想マシンである。

【００１２】

以下の説明では、クロスプラットフォームコード１４をJavaバイトコードとし、仮想マシン１２をJava仮想マシンとする。しかしながら、本発明の技術は、クロスプラットフォームの実行を可能とする他のプログラミング言語に対しても容易に適用できるということが明らかである。

【００１３】

仮想マシン１２は、クロスプラットフォームコード１４に含めることができるメソッド、ルーチンまたは関数を部分的に解読または部分的にコンパイルする。最初に、仮想マシン１２は、クロスプラットフォームコード１４に含まれる各バイトコード命令を解読することによって、クロスプラットフォームコード１４を実行する。さらに、仮想マシン１２は、コンパイルに適したクロスプラットフォームコード１４における基本ブロックを一つ以上検出する。仮想マシン１２は、これらの基本ブロックをコンパイルして、それぞれが装置１０内に設けられた特定のハードウェアプラットフォーム用のネイティブコード内にあるネイティブコードブロック２０～２４のセットを提供する。ネイティブコードブロック２０～２４を一旦コンパイルしてしまうと、仮想マシン１２は、適宜ネイティブコードブロック２０～２４を実行し、かつクロスプラットフォームコード１４の残り部分を解読することによって、クロスプラットフォームコード１４を実行する。

【００１４】

ほとんどのプログラムが、それらのコードの２０パーセントに満たない部分に実行時間の８０パーセントを越える時間を費やすということは、周知の経験則である。したがって、実行速度を上げるために、従来技術のようにアプリケーションプログラム全体またはメソッド全体をコンパイルする必要はない。実行が重いメソッド、ルーチンまたは関数の部分のみをコンパイルすればよい。実行時間のほとんどが、10000回実行される小ループで費やされるメソッドの場合を考える。仮想マシン１２において実施されるハイブリッド式ジャストインタイム・コンパイラの機能は、このループのみをコンパイルし、該メソッドの残りは、メソッドの実行中に解読される。

【００１５】

以下は、クロスプラットフォームコード１４に対応するJavaソースコードの例である。本例は、２つのベクトルのドット積を計算するメソッドである。

クロスプラットフォームコードのソースコードの例

```
void loopit(){
      double a[ ],b[ ];
      int i;
      double sum=0.0;
            Date d1=new date();
            a = new double[count];
            b = new double[count];
            for  (i=0; i< count; i++)
                  a[i] = b[i] = 10.0;
            for  (i=0; i< count; i++)
                  sum = a[i]*b[i]+sum;
            Date d2= new Date();
            long l1 = d1.getTime();
            long l2 = d2.getTime();
            l2 = l2-l1;
            System.out.println("Time ="+ l2);
}
```

本例のJavaソースコードに対応するクロスプラットフォームコード１４は、約１３６個のJavaバイトコード命令で構成される。本例のJavaソースコードにおいて上記で強調したforループは、下記する約１５個のJavaバイトコード命令で構成される。

基本ブロックの例

```
・・  ・・  ・・
57 aload_1
58 iload_3
59 daload
60 aload_2
61 iload_3
62 daload
63 duml
64 dload 4
66 dadd
67 dstore 4
69 iinc 3 1
72 iload_3
73 aload_0
74 getfield #14 <Field Dot.count I>
77 if_icmplt 57
・・  ・・  ・・
```

強調したforループを、本メソッドにおける１３６個のバイトコード命令のうち１５個の基本ブロックとしてコンパイルすることにより、メソッド全体またはプログラム全体をコ

ンパイルする従来のジャストインタイム・コンパイラに比べ、コンパイル作業の約８９パーセントが省かれる。これにより、コンパイルを実行するために装置１０において必要される資源が大幅に減少する。

【００１６】

さらに、本例のメソッドの実行時間の大部分は、仮想マシン１２のハイブリッド式ジャストインタイム・コンパイラ機能によってネイティブコードにコンパイルされる基本ブロック、すなわち２つのforループに費やされる。本例のメソッドの残りのバイトコード命令は、該メソッドの１実行あたり１回だけ実行される。これにより、コンパイルにかける労力を最小限とした大幅に改良された命令実行性能がもたらされる。

【００１７】

図２は、仮想マシン１２におけるジャストインタイム・コンパイラの一構成を示す。仮想マシン１２は、クロスプラットフォームコード１４を取り出して、それに含まれる各バイトコード命令を順次解読するインタープリタ５０を備える。本構成では、case（ケース）値を仮想マシン１２によってサポートされる命令セットの演算コード（オペコード）とした、きわめて大きなswitch文（スイッチステートメント）として、インタープリタ５０を特徴付けることが可能である。

【００１８】

インタープリタ５０は、バイトコード命令を解読しながらアクションテーブル５６にアクセスする。アクションテーブル５６は、クロスプラットフォームコード１４に含まれ得る各バイトコード命令、すなわち各Java演算コードをアクションコードのセットに対応づける。アクションコードの各セットは、ネイティブコード命令のセットであって、対応するJava演算コードをエミュレートするのに適切なオペレーションを実行する。

【００１９】

クロスプラットフォームコード１４を２回目に呼び出すとき、インタープリタ５０は、ブロック検出器５２を呼出す。他の実施形態では、３回目または４回目等にクロスプラットフォームコード１４を呼び出した後、またはある他の状態の後に、ブロック検出器５２の呼出しを行うことができる。ブロック検出器５２は、コンパイルの対象であるクロスプラットフォームコード１４内の基本ブロックを識別する。ブロック検出器５２により識別される基本ブロックの一例は、前述したforループである。コンパイルに適する基本ブロックの他の例として、何回も実行される関数呼出しまたはメソッド呼出しがある。

【００２０】

ブロック検出器５２は、ブロック境界構造５８を生成するが、この構造５８は、コンパイルに好適であるクロスプラットフォームコード１４内の基本ブロックの境界をマークするデータ構造である。上記で示した本例のメソッドでは、コンパイルに適する基本ブロックは、ソースコードで示したforループに対応するバイトコードブロックである。たとえば、ブロック境界構造５８は、強調したforループの境界５７および７７をリストする。

【００２１】

基本ブロックの各境界を識別した後、インタープリタ５０は、ブロック境界構造５８を使用しながらクロスプラットフォームコード１４内のバイトコード命令をそれぞれ取り出し、解読して、コンパイルに好適な基本ブロックに到達したときを判定する。基本ブロックの１つに出くわすと、インタープリタ５０は、コードジェネレータ５４を呼び出す。

【００２２】

コードジェネレータ５４は、インタープリタ５０から基本ブロックを取得して、対応するネイティブコードブロック２０～２４を生成する。たとえば、コードジェネレータ５４は、上記したバイトコード命令５７～７７を取得し、これに応答してネイティブコードブロック２０を生成する。一実施形態において、コードジェネレータ５４は、ネイティブコードブロック２０を参照してクロスプラットフォームコード１４内の基本ブロックを上書きし、次にクロスプラットフォームコード１４が実行され、基本ブロックに直面した時に、ネイティブコードブロック２０が実行されるようにする。

【００２３】

図３は、クロスプラットフォームコード１４の基本ブロックをコンパイルするためにコードジェネレータ５４により使用される手順を示す。コードジェネレータ５４は、上記した本例のバイトコード命令５７～７７のような基本ブロックの各バイトコード命令について、ステップ６０～６２を実行する。

【００２４】

ステップ６０において、コードジェネレータ５４は、コンパイルされる（またはコンパイル中の）現在のバイトコード命令の演算コードに対応するアクションコードのセットを取り出す。一実施形態において、アクションコードは、アクションテーブル５６から取得されるネイティブコード命令のセットである。このように、コードジェネレータ５４は、他の場合ではクロスプラットフォームコード１４を解読するために使用される仮想マシン１２の既存の構成要素を利用する。これにより、従来のジャストインタイム・コンパイラが必要とする追加の構成要素およびこれらが消費する資源は必要ではなくなる。

【００２５】

たとえば、アクションテーブル５６が以下のコンテンツを含むものとする。

アクションテーブルの例

【００２６】

【表１】

演算コード１	ネイティブコード命令Ａ
演算コード２	ネイティブコード命令Ｂ
演算コード３	ネイティブコード命令Ｃ
演算コード４	ネイティブコード命令Ｄ

【００２７】

また、演算コード１がaloadバイトコード命令に対応し、演算コード４がiloadバイトコード命令に対応し、演算コード２がdaloadバイトコード命令に対応するものとする。コードジェネレータ５４がステップ６０においてaload命令に直面すると、コードジェネレータ５４は、アクションコードとしてネイティブコード命令Ａをアクションテーブル５６から取得する。同様に、コードジェネレータ５４がステップ６０においてiload命令に直面すると、コードジェネレータ５４は、アクションテーブル５６からネイティブコード命令Ｄを取得し、以下同様である。４つの項目のみを示したが、アクションテーブル５６は、実際には、適用可能な命令セットにおいて可能な演算コードそれぞれに対するネイティブコード命令のセットを含む。この適用可能な命令セットは、本例ではJavaバイトコードである。

【００２８】

ステップ６２において、コードジェネレータ５４は、取得したアクションコードをネイティブコードブロック２０と連結する。上記した本例の基本ブロックの場合、バイトコード命令５７についてステップ６２を行った後、ネイティブコードブロック２０は、以下のようになる。

ネイティブコードブロック

【００２９】

【表２】

ネイティブコード命令Ａ

【００３０】
バイトコード命令５８についてステップ６２を行った後、ネイティブコードブロック２０は、以下のようになる。
ネイティブコードブロック
【００３１】
【表３】

ネイティブコード命令Ａ
ネイティブコード命令Ｄ

【００３２】
バイトコード命令５９についてステップ６２を行った後、ネイティブコードブロック２０は、以下のようになる。
ネイティブコードブロック
【００３３】
【表４】

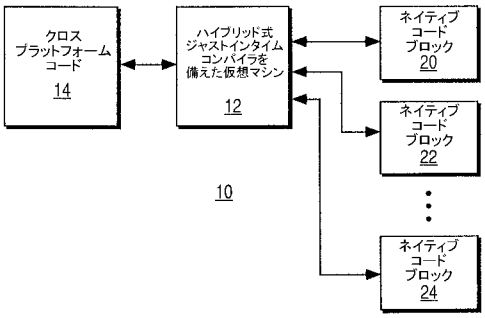
ネイティブコード命令Ａ
ネイティブコード命令Ｄ
ネイティブコード命令Ｂ

【００３４】
ネイティブコードブロック２０は、適当なプロローグ命令のセットと適当なエピローグ命令のセットを含こともできる。
【００３５】
基本ブロックにおける各バイトコード命令についてステップ６０～６２を行った後、コードジェネレータ５４は、ネイティブコードブロック２０の最適化を実行することができる。たとえば、ピープホール最適化を実行することができる。パターン駆動型ピープホール最適化手段を採用して移植性（可搬性）を高めることが好ましい場合がある。より頻繁に呼び出されるメソッドにおいてブロックを最適化するために使用されるより大きなウィンドウサイズを使用して、ピープホール最適化を複数回繰り返すことができる。これにより、インクリメンタル最適化技術がもたらされる。
【００３６】
ハイブリッド式ジャストインタイム・コンパイラを備えた仮想マシン１２は、資源が比較的制限されている装置において利点がある。これには、ファイルシステム資源が比較的少ないかまたは全くない装置、および／またはメモリ資源が最小限の装置、および／またはプロセッサ資源が制限されている装置が含まれよう。このような装置は、埋め込みシステムの場合がある。埋め込みシステムの例として、電話、オーディオ／ビデオ機器、家庭用電気器具、コンピュータ周辺機器がある。
【００３７】
上述した本発明の詳細な説明は、例示のためになされたものであり、これをもって全てとすることを意図したものでもなければ、開示した実施形態そのままのものに本発明を限定することを意図したものでもない。本発明の範囲は、特許請求の範囲により規定されるものである。
【００３８】
以下においては、本発明の種々の構成要件の組み合わせからなる例示的な実施態様を示す

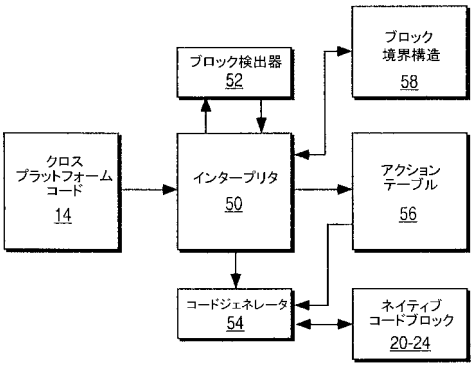
。
１．ハードウェアプラットフォーム上でクロスプラットフォームコードのセットの実行を可能とする仮想マシンであって、
前記クロスプラットフォームコードの選択された基本ブロックを前記ハードウェアプラットフォーム用のネイティブコードブロックにコンパイルするコードジェネレータと、
前記選択された基本ブロックの代わりに前記ネイティブコードブロックを実行し、前記クロスプラットフォームコードの残りの部分を解読することによって、前記クロスプラットフォームコードを実行するインタープリタ、
を備える仮想マシン。
２．前記選択された基本ブロックは、前記クロスプラットフォームコードに含まれるメソッドにおけるプログラムループである、上項１の仮想マシン。
３．前記選択された基本ブロックは、前記クロスプラットフォームコードにおいて複数回呼び出されるメソッドである、上項１の仮想マシン。
４．前記選択された基本ブロックは、前記クロスプラットフォームコードにおいて複数回呼び出される関数である、上項１の仮想マシン。
５．前記コードジェネレータは、前記クロスプラットフォームコードにおいて使用可能な演算コードのセットのそれぞれを、前記ハードウェアプラットフォームのネイティブコードに関して対応するアクションコードに対応づけるために、前記インタープリタにより使用されるテーブルに応答して前記ネイティブコードブロックを生成する、上項１の仮想マシン。
６．前記コードジェネレータが、前記ネイティブコードブロックを最適化する、上項１の仮想マシン。
７．前記クロスプラットフォームコードにおける前記基本ブロックの境界のセットと、前記クロスプラットフォームコードにおける追加基本ブロックのセットの各々についての境界のセットを判定するブロック検出器をさらに備える、上項１の仮想マシン。
８．前記コードジェネレータは、前記境界に応答して、前記選択された基本ブロックと各追加基本ブロックをコンパイルする、上項７の仮想マシン。
９．ハードウェアプラットフォーム上でクロスプラットフォームコードのセットを実行するための方法であって、
コンパイルのために、前記クロスプラットフォームコードの基本ブロックを選択するステップと、
前記基本ブロックを前記ハードウェアプラットフォーム用のネイティブコードブロックにコンパイルするステップと、
前記基本ブロックの代わりに前記ネイティブコードブロックを実行し、かつ前記クロスプラットフォームコードの残りの部分を解読することによって、前記クロスプラットフォームコードを実行するステップ
を含む方法。
１０．基本ブロックを選択する前記ステップが、前記クロスプラットフォームコードに含まれるメソッドにおけるプログラムループを選択するステップを含む、上項９の方法。
１１．基本ブロックを選択する前記ステップが、前記クロスプラットフォームコードにおいて複数回呼び出されるメソッドを選択するステップを含む、上項９の方法。
１２．基本ブロックを選択する前記ステップが、前記クロスプラットフォームコードにおいて複数回呼び出される関数を選択するステップを含む、上項９の方法。
１３．前記基本ブロックをコンパイルする前記ステップが、前記クロスプラットフォームコードにおいて使用可能な演算コードのセットのそれぞれを、前記ハードウェアプラットフォームのネイティブコードに関して対応するアクションコードに対応づけるために、インタープリタにより使用されるテーブルに応答して前記ネイティブコードブロックを生成するステップを含む、上項９の方法。
１４．前記ネイティブコードブロックを最適化するステップをさらに含む、上項９の方法。

１５．基本ブロックを選択する前記ステップが、前記クロスプラットフォームコードにおける前記基本ブロックの境界のセットを判定するステップを含む、上項９の方法。
１６．前記基本ブロックをコンパイルする前記ステップが、前記境界に応答して、前記基本ブロックをコンパイルするステップを含む、上項１５の方法。
１７．基本ブロックを選択する前記ステップが、前記クロスプラットフォームコードにおける追加基本ブロックのセットのそれぞれについて境界のセットを判定するステップを含む、上項９の方法。
１８．前記境界に応答して、前記追加基本ブロックをコンパイルするステップをさらに含む、上項１７の方法。
【００３９】
【発明の効果】
本発明によれば、従来のジャストインタイム・コンパイラに対して、ハードウェアプラットフォームの資源の消費が少ないジャストインタイム・コンパイラを備えた仮想マシンを提供することができる。
【図面の簡単な説明】
【図１】本発明の教示に従うハイブリッド式ジャストインタイム・コンパイラを具備した仮想マシンを備える装置を示す。
【図２】仮想マシンにおけるジャストインタイム・コンパイラの一構成を示す。
【図３】クロスプラットフォームコードの基本ブロックをコンパイルするためにコードジェネレータによって使用される手順を示す。
【符号の説明】
１２　仮想マシン
１４　クロスプラットフォームコード
２０～２４　ネイティブコードブロック
５０　インタープリタ
５２　ブロック検出器
５４　コードジェネレータ
５６　アクションテーブル
５８　ブロック境界構造

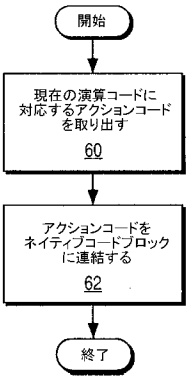
【図１】


クロス
プラットフォーム
コード
14
ハイブリッド式
ジャストインタイム
コンパイラを
備えた仮想マシン
12
ネイティブ
コード
ブロック
20
ネイティブ
コード
ブロック
22
ネイティブ
コード
ブロック
24
10

【図２】


ブロック検出器
52
ブロック
境界構造
58
クロス
プラットフォーム
コード
14
インタープリタ
50
アクション
テーブル
56
コードジェネレータ
54
ネイティブ
コードブロック
20-24

【図３】


開始
現在の演算コードに
対応するアクションコード
を取り出す
60
アクションコードを
ネイティブコードブロック
に連結する
62
終了

フロントページの続き

(72)発明者　ギーザ・マニューナス
インド国バンガロール５６００５０，ビーエスケイ・ファースト・ステージ・セカンド・ブロック，フォーティーンス・メイン・ナンバー２０２

審査官　石川　正二

(56)参考文献　特開平１０－２４０５４６（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/45