(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第5466601号
(P5466601)

(45)発行日　平成26年4月9日(2014.4.9)　(24)登録日　平成26年1月31日(2014.1.31)

(51)Int.Cl.　　F I
G06F 9/44 (2006.01)　　G06F 9/44 530P
G06F 9/45 (2006.01)　　G06F 9/44 530J
　　G06F 9/44 322F

請求項の数 14　(全 15 頁)

(21)出願番号　特願2010-194224 (P2010-194224)
(22)出願日　平成22年8月31日 (2010.8.31)
(65)公開番号　特開2012-53566 (P2012-53566A)
(43)公開日　平成24年3月15日 (2012.3.15)
審査請求日　平成25年5月1日 (2013.5.1)

(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】コード生成方法、システム及びプログラム

(57)【特許請求の範囲】
【請求項１】
コンピュータの処理に基づき、メソッドをインラインする機能をもつコード生成方法であって、
前記コンピュータの処理により、メソッドを含むコードを走査するステップと、
前記コンピュータの処理により、メソッドが、そのままインライン化するには大きすぎると判断した場合に、引数の実行時型に基づく分岐を含むかどうか判断するステップと、
前記メソッドが、引数の実行時型に基づく分岐を含むなら、前記コンピュータの処理により、前記メソッドのコールサイトにおける実際の引数の実行時型をプロファイルするステップと、
前記コンピュータの処理により、前記プロファイルに基づき、複数引数に基づく多相インライン・キャッシュ（ＰＩＣ）コードを生成しつつ、前記ＰＩＣコードから呼び出されうる、引数の実行時型の頻出する組み合わせに特化したメソッドを、生成するステップと、
前記コンピュータの処理により、前記特化したメソッドの本体のサイズの大きさが許容範囲であるなら、前記特化したメソッドの本体を、前記コールサイトにインラインするステップを含む、
方法。
【請求項２】
前記プロファイルするステップが、メソッド・プロローグ計装、またはコールサイト計

装のプロファイル結果からの型推論によって行われる、請求項１に記載の方法。
【請求項３】
前記方法が、Java(R) VMのバイトコードを変換するＪＩＴコンパイラ上で実装された方法である、請求項１に記載の方法。
【請求項４】
前記バイトコードが、Java(R) VMのバイトコードを生成する機能をもつ動的スクリプト言語処理系によって生成されたものである、請求項３に記載の方法。
【請求項５】
コンピュータの処理に基づき、メソッドをインラインする機能をもつコード生成プログラムであって、
前記コンピュータに、
メソッドを含むコードを走査するステップと、
メソッドが、そのままインライン化するには大きすぎると判断した場合に、引数の実行時型に基づく分岐を含むかどうか判断するステップと、
前記メソッドが、引数の実行時型に基づく分岐を含むなら、前記メソッドのコールサイトにおける実際の引数の実行時型をプロファイルするステップと、
前記プロファイルに基づき、複数引数に基づく多相インライン・キャッシュ（ＰＩＣ）コードを生成しつつ、前記ＰＩＣコードから呼び出されうる、引数の実行時型の頻出する組み合わせに特化したメソッドを、生成するステップと、
前記特化したメソッドの本体のサイズの大きさが許容範囲であるなら、前記特化したメソッドの本体を、前記コールサイトにインラインするステップを実行させる、
プログラム。
【請求項６】
前記プロファイルするステップが、メソッド・プロローグ計装、またはコールサイト計装のプロファイル結果からの型推論によって行われる、請求項５に記載のプログラム。
【請求項７】
前記プログラムが、Java(R) VMのバイトコードを変換するＪＩＴコンパイラの機能をもつ、請求項５に記載のプログラム。
【請求項８】
前記バイトコードが、Java(R) VMのバイトコードを生成する機能をもつ動的スクリプト言語処理系によって生成されたものである、請求項７に記載のプログラム。
【請求項９】
前記動的スクリプト言語処理系が、P8、Quercus、JRuby、Jython、及びGroovyから選ばれたものである、請求項８に記載のプログラム。
【請求項１０】
コンピュータの処理に基づき、メソッドをインラインする機能をもつコード生成システムであって、
記憶手段と、
前記記憶手段に記憶された、メソッドを含むコードを走査する手段と、
メソッドが、そのままインライン化するには大きすぎると判断した場合に、引数の実行時型に基づく分岐を含むかどうか判断する手段と、
前記メソッドが、引数の実行時型に基づく分岐を含むなら、前記メソッドのコールサイトにおける実際の引数の実行時型をプロファイルする手段と、
前記プロファイルに基づき、複数引数に基づく多相インライン・キャッシュ（ＰＩＣ）コードを生成しつつ、前記ＰＩＣコードから呼び出されうる、引数の実行時型の頻出する組み合わせに特化したメソッドを、生成する手段と、
前記特化したメソッドの本体のサイズの大きさが許容範囲であるなら、前記特化したメソッドの本体を、前記コールサイトにインラインする手段を有する、
コード生成システム。
【請求項１１】

前記プロファイルする手段が、メソッド・プロローグ計装、またはコールサイト計装のプロファイル結果からの型推論を用いる、請求項１０に記載のシステム。
【請求項１２】
前記コードが、Java(R) VMのバイトコードである、請求項１０に記載のシステム。
【請求項１３】
前記インラインする手段によるインラインの後、前記バイトコードをＪＩＴコンパイルする手段をさらに有する、請求項１２に記載のシステム。
【請求項１４】
前記コードが、Java(R) VMのバイトコードを生成する機能をもつ動的スクリプト言語処理系上のコードである、請求項１２に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
この発明は、プログラミング言語処理系における実行可能コードの最適化に関し、より詳しくは、インライニング技法に関するものである。
【背景技術】
【０００２】
従来より、サーバ環境で使用されるプログラミング言語処理系、あるいは実行系として、PHPのような動的スクリプト言語と、Java(R)のようなより静的なプログラミング言語が使用されているが、近年になって、Java(R)のクラス資産をPHPなどから簡易に呼び出せるように、Java(R)仮想マシンあるいは、コンピュータ共通言語基盤（ＣＬＩ）のような静的な言語プラットフォーム上でPHPのような動的スクリプト言語が、静的な言語プラットフォームのクラスを宣言し、タイプ付けされていないアクセスを可能ならしめる仕組みが提供されるようになってきている。
【０００３】
特に、Java(R)仮想マシン上で動作するPHPとしてのP8及びQuercus、RubyとしてのJRuby、PythonとしてのJython、Groovyなどが知られている。
【０００４】
それらのスクリプト言語が生成するほとんど全てのバイトコードは、言語構造の複雑なセマンティクスを扱うために、サブルーチン・スレッディング(subroutine threading)スタイルを形成する。サブルーチンとして呼び出される関数またはメソッドの機能は、ランタイム・ヘルパ機能とも呼ばれる。そのようなサブルーチン・コールは、呼び出される関数またはメソッドをインライン化し、他のインラインされた関数またはメソッドと組み合わせることによって最適化することができる。
【０００５】
そのようなインライニングの例を示すと次のとおりである。先ず、下記のようなバイトコードが生成されたとする。

```
Obj f(Obj a, Obj b, Obj c) {
load a
load b
invoke add(Obj,Obj)
load c
invoke sub(Obj,Obj) ..
invoke print(Obj) …
```

これは、わかりやすさのために擬似ソースコードで示す（以降同様）ことにすると以下のようになる。

```
Obj f(Obj a, Obj b, Obj c) {
print(sub(add(a,b), c))  …
```

【０００６】
そこで呼び出されるadd(Obj,Obj)のコードは次のようであるとする。

```
Obj add(Obj x, Obj y) {
  if (x instanceof Int
      && y instanceof Int) {
    return new Int(
      x.getInt() + y.getInt());
 } else if (x instanceof Dbl ..) {
    ..
  } else if (..) {
...
```

【０００７】

また、呼び出されるsub(Obj,Obj)のコードは次のようであるとする。

```
Obj sub(Obj x, Obj y) {
  if (x instanceof Int
      && y instanceof Int) {
    new Int(
      x.getInt() - y.getInt());
  } else if (x instanceof Dbl ..) {
    ..
  } else if (..) {
...
```

【０００８】

このとき、もとのバイトコードに、add(Obj x, Obj y)とsub(Obj x, Obj y)をそれぞれインライン化できれば、共通部分式の評価を一度で済ませたり、中間データの生成を取り除いたりしてコードの実行を高速化できる。

【０００９】

ところが、そのようなサブルーチンのコードは大きすぎるので、既存のスクリプト言語のコンパイラでは、インライン化することが困難である。すなわち、インライン先のコードサイズが大きくなりすぎて、現実的な時間で最適化の処理ができなくなるためである。

【００１０】

特開２００７－１０９０４４号公報は、プログラム群から呼び出し先プログラムと呼び出し先プログラムの処理を異ならせる引数と該引数の内容とを検出し呼び出し先プログラムの処理のうち引数の内容に応じた処理のみを実行するプログラムを専用呼び出し先プログラムとして、引数と引数の内容毎に生成する専用呼び出し先プログラム生成部と、プログラム群から呼び出し先プログラムを呼び出す呼び出し箇所を検出する呼び出し箇所検出部と、専用呼び出し先プログラム生成部により生成された専用呼び出し先プログラムを呼び出すように呼び出し箇所を書き換える呼び出し元プログラム書き換え部とを備えたコード変換プログラムを開示する。しかし、この技法は、コードのインライニングへの応用を示唆するものではない。また、複数の引数の内容が潜在的に多岐にわたる際に、専用呼び出し先プログラムの数が組み合わせの場合の数に応じて膨大になってしまい、少なくとも実行時にこれを行うことは現実的でない。

【００１１】

Toshio Suganuma Toshiaki Yasue Toshio Nakatani, "An Empirical Study of Method Inlining for a Java Just-In-Time Compiler", Proceedings of the 2nd JavaTM Virtual Machine Research and Technology Symposium (JVM '02) San Francisco, California, USA August 1-2, 2002 は、大きいメソッド・ボディにおけるホット・パスのみをインライン化しようと試みる技法を開示する。しかし、この技法は、異なるコンテキストで多くのコール・サイトから呼ばれるような、ランタイム・ヘルパ機能の場合には、ホット・パスが定まらないため適用できない。

【００１２】

John Whaley, "Dynamic Optimization through the use of Automatic Runtime Special

ization", Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of Bachelor of Science in Computer Science and Master of Engineering in Electrical Engineering and Computer Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY, May 1999 (http://suif.stanford.edu/~jwhaley/papers/masterstheｓis.pdf)は、特にその3.4.16章には、パラメータ・タイプに基づき特殊化されたコードから得られる恩恵を予測することに関する議論が述べられる。

【００１３】

Matthew Arnold, Stephen Fink, Vivek Sarkar, Peter F. Sweeney, "A Comparative Study of Static and Profile-Based Heuristics for Inlining", ACM SIGPLAN Notices archive Volume 35, Issue 7 (July 2000) Pages: 52 - 64には、プロファイルに基づくインライニングのヒューリスティックスが議論されている。

【００１４】

しかし、上記２つの非特許文献の内容は、本発明にとって、背景技術にとどまるものである。

【先行技術文献】

【特許文献】

【００１５】

【特許文献１】特開２００７－１０９０４４号公報

【非特許文献】

【００１６】

【非特許文献１】Toshio Suganuma Toshiaki Yasue Toshio Nakatani, "An Empirical Study of Method Inlining for a Java Just-In-Time Compiler", Proceedings of the 2nd JavaTM Virtual Machine Research and Technology Symposium (JVM '02) San Francisco, California, USA August 1-2, 2002

【非特許文献２】John Whaley, "Dynamic Optimization through the use of Automatic Runtime Specialization", Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of Bachelor of Science in Computer Science and Master of Engineering in Electrical Engineering and Computer Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY, May 1999 (http://suif.stanford.edu/~jwhaley/papers/masterstheｓis.pdf)

【非特許文献３】Matthew Arnold, Stephen Fink, Vivek Sarkar, Peter F. Sweeney, "A Comparative Study of Static and Profile-Based Heuristics for Inlining", ACM SIGPLAN Notices archive Volume 35, Issue 7 (July 2000) Pages: 52 - 64

【発明の概要】

【発明が解決しようとする課題】

【００１７】

この発明の目的は、従来の技法では困難であったような大きいサブルーチンまたはメソッドのインライニングを可能ならしめる技法を提供することにある。

【課題を解決するための手段】

【００１８】

本発明は、Java(R)VMのＪＩＴコンパイラのような動的コンパイラにおけるインライン化の方法に関するものであり、あるメソッド呼び出しをインライン化する際、以下のようなステップをたどる。すなわち、メソッド本体が十分小さい場合はそのままインライン化し、そうでなくとも、プロファイルされたメソッド本体中高頻度で実行されるパス（ホット・パス）が十分小さい場合には、そのホット・パスを切り出した部分のみをインライン化する。ここまでは従来技術の範囲である。

【００１９】

本発明のコード変換器は、インラインできない場合に、メソッドが、みなしマルチメソッドかどうか検査する。ここで、みなしマルチメソッドとは、この発明の文脈で定義され

る用語であり、その定義は次のように与えられる。
すなわち、みなしマルチメソッド(deemed multimethods)とは、メソッド本体が以下のいずれかの条件によくあてはまるコードを含むメソッドのことである。
(a) 仮引数のオブジェクトに対してinstanceof や checkcast による実行時型検査の結果に基づいて分岐する。
(b) 仮引数のオブジェクトに対して invokevirtual や invokeinterface により、型ごとに結果が変わらない（ことの多い）メソッド呼び出しを行い、その結果に基づいて分岐する
(c) 仮引数のオブジェクトを他のみなしマルチメソッドの実引数として渡す。
【００２０】
本発明によれば、みなしマルチメソッドであれば、実引数の組の実行時の型の分布が（型推論やプロファイリングにより）検査される。
実引数の型の組の出現に偏りがあらわれていれば、高頻度の型の組について、メソッドを特化し、メソッドと型の組をキーとしてキャッシュしておく。
高頻度の型の組のいくつかについて、実行時検査をして特化されたメソッドを呼び出すように、呼び出し元のコードを最適化する。
特化されたメソッドが十分小さいときにはインライン化する。
【発明の効果】
【００２１】
本発明により、従来、最終的にはコードを呼び出し元の文脈に基づく特化を行うことにより小さくインライン化が可能にもかかわらずインライン化されなかったメソッド呼び出し、例えば、スクリプト言語実装における実行時ヘルパーの呼び出しなどが、効率よくインライン化できるようになる。
【図面の簡単な説明】
【００２２】
【図１】本発明を実施するためのハードウェアの一例のブロック図である。
【図２】機能ブロックのレイヤを示す図である。
【図３】インライン処理の全体の概要のフローチャートを示す図である。
【図４】みなしマルチメソッド検出処理のフローチャートを示す図である。
【図５】メソッド・プロローグ計装処理のフローチャートを示す図である。
【図６】メソッド・プロローグ計装処理のフローチャートを示す図である。
【図７】プロファイリング・テーブルの例を示す図である。
【図８】コード・レベルでの、インライン処理の全体の概要を示す図である。
【発明を実施するための形態】
【００２３】
以下、図面に従って、本発明の実施例を説明する。これらの実施例は、本発明の好適な態様を説明するためのものであり、発明の範囲をここで示すものに限定する意図はないことを理解されたい。また、以下の図を通して、特に断わらない限り、同一符号は、同一の対象を指すものとする。
【００２４】
図１を参照すると、本発明の一実施例に係るシステム構成及び処理を実現するためのコンピュータ・ハードウェアのブロック図が示されている。図１において、システム・バス１０２には、ＣＰＵ１０４と、主記憶（ＲＡＭ）１０６と、ハードディスク・ドライブ（ＨＤＤ）１０８と、キーボード１１０と、マウス１１２と、ディスプレイ１１４が接続されている。ＣＰＵ１０４は、好適には、３２ビットまたは６４ビットのアーキテクチャに基づくものであり、例えば、インテル社のPentium(商標)4、インテル社のCore(商標) ２ DUO、AMD社のAthlon(商標)などを使用することができる。主記憶１０６は、好適には、１ＧＢ以上の容量、より好ましくは、２ＧＢ以上の容量をもつものである。
【００２５】
ハードディスク・ドライブ１０８には、オペレーティング・システムが、格納されてい

る。オペレーティング・システムは、Linux（商標）、マイクロソフト社のWindows(商標) 7、Windows XP(商標)、Windows(商標)2003サーバ、アップルコンピュータのMac OS(商標)などの、ＣＰＵ１０４に適合する任意のものでよい。

【００２６】

ハードディスク・ドライブ１０８にはまた、Apacheなどの、Ｗｅｂサーバとしてシステムを動作させるためのプログラムが保存され、システムの起動時に、主記憶１０６にロードされる。

【００２７】

ハードディスク・ドライブ１０８にはさらに、Java(R)仮想マシン（VM）を実現するためのJava(R) Runtime Environmentプログラムが保存され、システムの起動時に、主記憶１０６にロードされる。

【００２８】

ハードディスク・ドライブ１０８にはさらに、動的スクリプト言語のJava(R)バイトコード生成器、及び、その動的スクリプト言語の書かれたソースコードが保存されている。この実施例における動的スクリプト言語は、Java(R)仮想マシン上で動作する動的スクリプト言語としてのP8及びQuercus、RubyとしてのJRuby、PythonとしてのJython、Groovyなどのうちの任意のものでよいが、特にこの実施例では、典型的にはP8である、PHPのJava(R)バイトコード生成器であるとする。

【００２９】

キーボード１１０及びマウス１１２は、オペレーティング・システムが提供するグラフィック・ユーザ・インターフェースに従い、ディスプレイ１１４に表示されたアイコン、タスクバー、ウインドウなどのグラフィック・オブジェクトを操作するために使用される。

【００３０】

ディスプレイ１１４は、これには限定されないが、好適には、１０２４×７６８以上の解像度をもち、３２ビットtrue colorのＬＣＤモニタである。

【００３１】

通信インターフェース１１６は、好適には、イーサネット(R)プロトコルにより、ネットワークと接続されている。通信インターフェース１１６は、クライアント・コンピュータ（図示しない）からApacheが提供する機能により、ＴＣＰ／ＩＰなどの通信プロトコルに従い、処理リクエストを受け取り、あるいは処理結果を、クライアント・コンピュータ（図示しない）に返す。

【００３２】

図２において、最下層は、オペレーティング・システム４０２であり、これには限定されないが、この実施例では、Windows（商標）２００３サーバまたはLinux（商標）である。

【００３３】

オペレーティング・システム２０２の上では、オペレーティング・システム２０２に適合するバージョンのJava(R)　VM２０４が動作する。

【００３４】

Java(R)　VM２０４のレイヤの上では、PHP用Java(R)バイトコード生成器２０６と、本発明に係るコード変換器２０８が動作する。

【００３５】

PHPソースコード２１０は、ハードディスク・ドライブ１０８に保存され、周知のように、<?php ～ ?>で記述されるステートメントを含み、拡張子phpをもつファイルである。ネットワークを介してクライアント・コンピュータ（図示さない）から受け取ったリクエストに応答して、PHP用Java(R)バイトコード生成器４０６が、指定されたPHPソースコード４１２を解釈実行して、バイトコードを生成する。

【００３６】

コード変換器２０８は、PHP用Java(R)バイトコード生成器４０６が生成したバイトコー

ドに対して、後述する処理により適宜メソッド・インライン化することによって、より最適化されたバイトコードに変換する。

【００３７】

コード変換器２０８の機能は、好適にはＪＩＴコンパイラ（図示しない）の機能の一部として実装される。すなわち、ＪＩＴコンパイラは、PHP用Java(R)バイトコード生成器４０６から入力したバイトコードを、コード変換器２０８の機能を使用して一旦メソッド・インライン的に最適化した後、ＣＰＵ１０４とオペレーティング・システム２０２にネイティブな実行コードに変換する。

【００３８】

次に、図３のフローチャートを参照して、コード変換器２０８のインライン処理を説明する。コード変換器２０８は、バイトコードをスキャンして、メソッドの呼び出しのコードを検出したとき、図３のフローチャートの処理を実行する。

【００３９】

ステップ３０２で、コード変換器２０８が、メソッドのサイズが十分小さいかどうか判断し、もしそうなら、ステップ３０４で、そのメソッドを、呼び出し元のコードにインラインして、処理を終る。

【００４０】

ステップ３０２で、メソッドのサイズが十分小さくないと判断されると、ステップ３０６で、コード変換器２０８は、メソッドにおけるホット・パスが十分小さいかどうか判断し、もしそうなら、ステップ３０４で、そのホット・パスを、呼び出し元のコードにインラインして、処理を終る。この場合、メソッドにおけるホット・パスを見出すのに、例えば、非特許文献１の技術が使用される。

【００４１】

ステップ３０６で、ホットパスサイズが十分小さくないと判断されると、コード変換器２０８は、ステップ３０８で、メソッドが、みなしマルチメソッドかどうか判断する。ここで、みなしマルチメソッド(deemed multimethods)とは、特にこの実施例で定義された用語であり、メソッド本体が以下のいずれかの条件によくあてはまるコードを含むメソッドのことである。

(a) 仮引数のオブジェクトに対してinstanceof や checkcast による実行時型検査の結果に基づいて分岐する。

(b) 仮引数のオブジェクトに対して invokevirtual や invokeinterface により、型ごとに結果が変わらない（ことの多い）メソッド呼び出しを行い、その結果に基づいて分岐する

(c) 仮引数のオブジェクトを他のみなしマルチメソッドの実引数として渡す。

みなしマルチメソッドの判定処理については、図４のフローチャートを参照して、後で説明する。

【００４２】

ステップ３０８で、メソッドが、みなしマルチメソッドでないと判断すると、コード変換器２０８は、インラインすることなく、処理を終了する。

【００４３】

ステップ３０８で、メソッドが、みなしマルチメソッドであると判断すると、コード変換器２０８は、ステップ３１０に処理を進める。

【００４４】

ステップ３１０では、コード変換器２０８は、プロファイリング、すなわち、実引数の組の実行時の形の分布を検査する。これは例えば、コード変換器２０８によるインライン処理を経ないで、バイトコードをＪＩＴコンパイラにより実行コードに変換し、その実行コードの実行結果を計測することによって行われる。プロファイリングの詳細については、図５及び図６のフローチャートを参照して、後で説明する。

【００４５】

次のステップ３１２では、コード変換器２０８は、プロファイリングの結果得られた高

頻度の型の組について、メソッドを特化し、メソッドと型の組をキーとしてキャッシュする。

【００４６】

次のステップ３１４では、コード変換器２０８は、特化されたメソッドを呼び出すように、呼び出し元のコードを最適化する。

【００４７】

次のステップ３１６では、コード変換器２０８は、特化されたメソッドのサイズが十分小さいかどうか判断し、もしそうなら、ステップ３１８で、特化されたメソッドをインラインし、そうでなければ、インラインしないで処理を終了する。

【００４８】

次に、図４のフローチャートを参照して、コード変換器２０８による、みなしマルチメソッドの判定処理について説明する。コード変換器２０８は、ステップ４０２で、パラメータから、データのフローをたどる。ここで、パラメータとは、intやdoubleなどの型のことである。

【００４９】

コード変換器２０８は、ステップ４０４で、パラメータまたはその派生からデータに基づき分岐を調べる。これは例えば、instanceof や checkcast による実行時型検査の結果に基づいて分岐するかどうか、あるいは、invokevirtual や invokeinterface により、型ごとに結果が変わらない（ことの多い）メソッド呼び出しを行い、その結果に基づいて分岐するかどうかである。

【００５０】

コード変換器２０８がステップ４０６で、そのような分岐があると判断すると、ステップ４０８で、当該のメソッドを、みなしマルチメソッドとしてマークする。そうでないなら、ステップ４１０で、当該のメソッドを、非みなしマルチメソッドとしてマークする。

【００５１】

ここでマークした結果が、図３のステップ３０８で使用される。

【００５２】

次に、図５及び図６のフローチャートを参照して、図３のステップ３１０のプロファイリング処理について説明する。この処理は、メソッド・プロローグ計装と呼ばれる。

【００５３】

図５のフローチャートは、呼ばれる、みなしマルチメソッド側の処理を示す。ステップ５０２では、コード変換器２０８は、みなしマルチメソッドのボディの最初に、パラメータ・タイプの組み合わせを記録するコードを挿入する。これは例えば、所定の配列名で主記憶１０６に、みなしマルチメソッドのパラメータを記録するコードである。

【００５４】

こうしておいて、ステップ５０４では、コード変換器２０８は、みなしマルチメソッドにコードが挿入された状態で、もとのコードを実行する。好適には、ＪＩＴコンパイラでネティブなコードに変換して実行することになる。

【００５５】

図６のフローチャートは、みなしマルチメソッドを呼ぶ側の処理を示す。コード変換器２０８は、ステップ６０２で、このみなしマルチメソッドをコールするコールサイトがあるかどうか、コール・スタックを見る。

【００５６】

そして、ステップ６０４では、コード変換器２０８は、みなしマルチメソッドをコールするメソッド・パラメータの実際のタイプを見る。

【００５７】

ステップ６０６では、コード変換器２０８は、このみなしマルチメソッドについて、プロファイリング・テーブルから、コールサイトとパラメータの組み合わせのエントリを検索し、もし存在しないなら、カウンダ = 0のエントリを作成する。ここで、コール・サイトとは、実際は、生成された所定の数値のコールサイトＩＤである。コールサイトＩＤは

、あるコールサイトが、あるメソッドを呼び出したときに、そのコールサイトに一意的に、好適にはコード変換器２０８によって、決定される。プロファイリング・テーブルの例は、図７を参照して後で説明する。

【００５８】

コード変換器２０８は、ステップ６０８で、プロファイリング・テーブにおける、その組み合わせのエントリのカウンタをインクリメントして、処理を終わる。

【００５９】

図７は、プロファイリング処理によって、好適には主記憶１０６に生成されるプロファイリング・テーブル７０２を示す。プロファイリング・テーブル７０２において、パラメータ・タイプとは、(Int,Int)あるいは、(Int,Str)などの、メソッドを呼び出すときに使われる情報であり、図５のステップ５０２で挿入されたコードによって記録されたものである。

【００６０】

プロファイリング・テーブル７０２は、呼ばれるメソッド毎に設けられるかまたは、メソッドを識別するフィールドを付与して作成される。図７に示すプロファイリング・テーブル７０２は、便宜上、単一のメソッドに関するものとして示されている。

【００６１】

一方、コールサイトＩＤは、元のバイトコードの、呼び出し側の位置に固有に、コード変換器２０８によって与えられた値であり、図７の例では、バイトコード７０４の所定位置に対して、コールサイトＩＤ = 1234が付与されている。

【００６２】

プロファイリング・テーブル７０２において、カウントのフィールドは、図６のステップ６０８でインクリメントされる。すなわち、同一のコールサイトで、同一のパラメータ・タイプの組み合わせで当該のメソッドが呼ばれる度に、そのコールサイトＩＤ及びそのパラメータ・タイプのエントリのカウントがインクリメントされる。

【００６３】

ところで、本実施例によれば、図３のステップ３１４で示すように、プロファイリングの結果に従う、特化されたメソッドの呼び出しに対して、コード変換器２０８が、呼び出し元のコードを最適化したコードを生成する。それは、マルチパラメータＰＩＣ(polymorphic inline cache、すなわち、多形インライン・キャッシュ)コードと呼ばれる。マルチパラメータＰＩＣコードについては後述する。

【００６４】

さらに、本実施例によれば、図３のステップ３１２で示すように、コード変換器２０８は、マルチパラメータＰＩＣからの呼び出しに応答して、その呼び出すパラメータに特化した、みなしマルチメソッドにおけるコードの一部であるコードを、呼び出し先コード７０６から、オンデマンドで生成する。そのような特化したコードは、図７では、コード７０８ａ、７０８ｂ及び７０８ｃとして示されている。

【００６５】

次に、図３のステップ３１４に対応する、マルチパラメータＰＩＣの生成の例を、より具体的に説明する。すなわち、invoke add(Obj,Obj)という呼び出し側のコードに対して、コード変換器２０８は、(Int,Int)に特化した場合、次のようなＰＩＣコードを生成する。

```
dup2
instanceof Int
ifeq NOT_X_Int
instanceof Int
ifeq GENERAL_CALL
invoke add_Int_Int(Int,Int)
jmp END_PIC
NOT_X_Int:
```

```
pop
GENERAL_CALL:
invoke add(Obj,Obj)
END_PIC:
```

【００６６】

あるいは、(Int,Int)及び(Int,Str)に特化した場合、次のようなＰＩＣコードを生成する。

```
dup2
instanceof Int
ifeq NOT_X_Int
instanceof Int
ifeq GENERAL_CALL
invoke add_Int_Int(Int,Int)
jmp END_PIC
NOT_X_Int:
dup_x1
instanceof Str
ifeq NOT_X_Str
instanceof Int
ifeq GENERAL_CALL
invoke add_Int_Str(Int,Str)
jmp END_PIC
NOT_X_Str:
pop
GENERAL_CALL:
invoke add(Obj,Obj)
END_PIC:
```

【００６７】

どのようなパラメータに特化するかは、そのコールサイトにおけるプロファイリングの結果に基づく。

【００６８】

一方、図３のステップ３１２に対応する、パラメータに基づく特殊なコード生成の例を、より具体的に説明する。すなわち、呼ばれる側のコードが次のようだったとする。

```
Obj add(Obj x, Obj y) {
  if (x instanceof Int
      && y instanceof Int) {
    return new Int(
      x.getInt() + y.getInt());
 } else if (x instanceof Dbl ..) {
    ..
 } else if (..) {
  …
```

【００６９】

すると、これの、(Int,Int)に特化(specizlized)されたコードは、次のとおりとなる。

```
Int add_Int_Int(Int x, Int y) {
  return new Int(x.val + y.val);  }
```

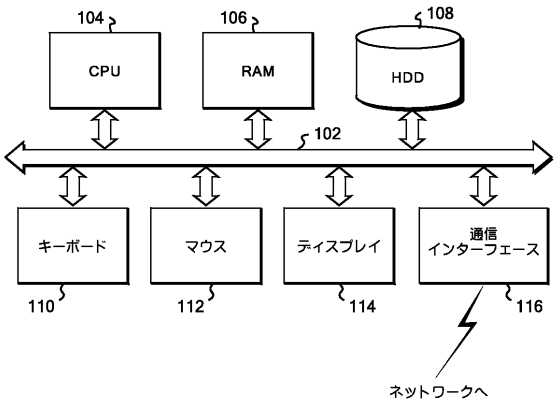
これは、上記みなしマルチメソッドのコードにおいて、(x instanceof Int && y instanceof Int)という条件をみたす箇所だけを抜き出したコードであることが見て取れよう。

【００７０】

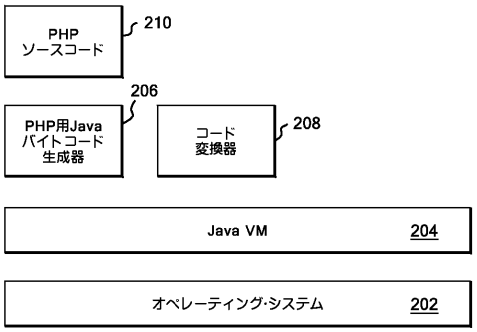
次に、図８を参照して、まとめとして、今まで説明した処理の全体の流れを説明する。

【００７１】
図８において、最初のステップは、みなしマルチメソッドのコードを検出することである。これは、図３では、ステップ３０８に対応する。こうして、みなしマルチメソッドのコード８０２が見出されると、次のステップは、コールサイトのコンテキストでパラメータ・タイプをプロファイルすることである。これは、図３では、ステップ３１０に対応する。
【００７２】
次のステップは、プロファイリングの結果に基づき、呼び出し元のコード８０４から、マルチパラメータＰＩＣ８０６を生成することである。これは、図３では、ステップ３１４に対応する。
【００７３】
一方、呼び出し先では、プロファイリングの結果に基づき、高頻度の特化されたコード８０８ａ、８０８ｂをキャッシュしておく。これは、図３では、ステップ３１２に対応する。
【００７４】
こうして、マルチパラメータＰＩＣ８０６から呼ばれたときに、特化されたコード８０８ａ、８０８ｂのうちのパラメータに対応するものが、もしコードのスペースが許されるなら、コード８１０で示すように、インラインされることになる。
【００７５】
なお、上記実施例では、プロファイリングとしてメソッド・プロローグ計装が使用されたが、これには限定されず、コールサイト計装などの任意の利用可能な計装方法を使用することができる。ここで、コールサイト計装とは、コールサイトに、パラメータの型を調べてカウントするコードを入れておく方法である。
【００７６】
また、上記実施例では、Java(R) VM用のバイトコードを、ＪＩＴコンパイラでコンパイルする例で説明したが、マルチパラメータＰＩＣと、メソッドにおけるパラメータに特化したコードをインラインに使用できるようになっているなら、任意のプラットフォーム、ＯＳ及び言語処理系の上で本発明を実施可能であることを、この分野の当業者なら理解するであろう。
【符号の説明】
【００７７】
１０４　ＣＰＵ
１０６　主記憶
１０８　ハードディスク・ドライブ
２０２　オペレーティング・システム
２０６　バイトコード生成器
２０８　コード変換器
４０２　オペレーティング・システム
４０６　バイトコード生成器
４１２　ソースコード
７０２　プロファイリング・テーブル
７０４　バイトコード
８０６　マルチパラメータＰＩＣ

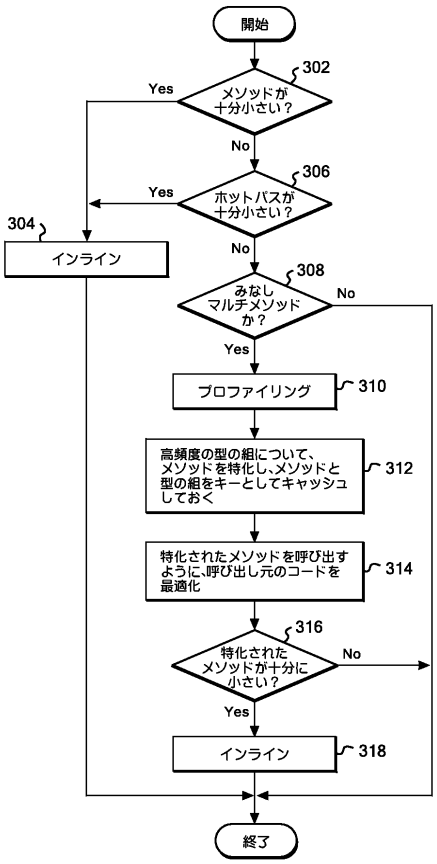
【図１】


104
CPU
106
RAM
108
HDD
102
キーボード
マウス
ディスプレイ
通信
インターフェース
110
112
114
116
ネットワークへ

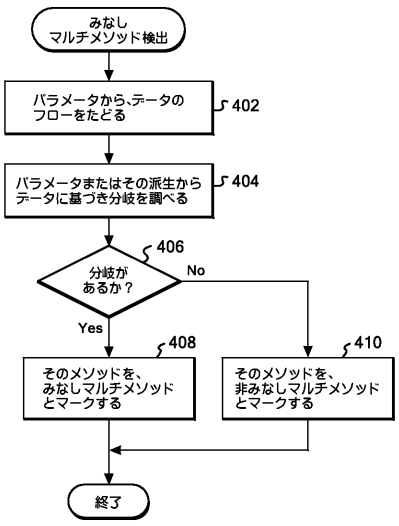
【図２】


PHP
ソースコード
210
PHP用Java
バイトコード
生成器
206
コード
変換器
208
Java VM
204
オペレーティング・システム
202

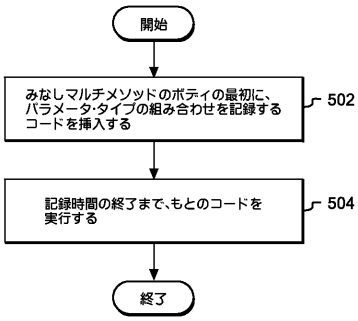
【図３】


開始
302
メソッドが
十分小さい？
Yes
No
306
ホットパスが
十分小さい？
Yes
304
インライン
No
308
みなし
マルチメソッド
か？
No
Yes
プロファイリング
310
高頻度の型の組について、
メソッドを特化し、メソッドと
型の組をキーとしてキャッシュ
しておく
312
特化されたメソッドを呼び出す
ように、呼び出し元のコードを
最適化
314
316
特化された
メソッドが十分に
小さい？
No
Yes
インライン
318
終了

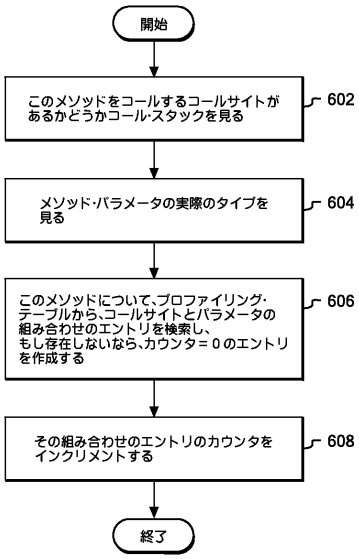
【図４】


みなし
マルチメソッド検出
パラメータから、データの
フローをたどる
402
パラメータまたはその派生から
データに基づき分岐を調べる
404
406
分岐が
あるか？
No
Yes
408
そのメソッドを、
みなしマルチメソッド
とマークする
410
そのメソッドを、
非みなしマルチメソッド
とマークする
終了

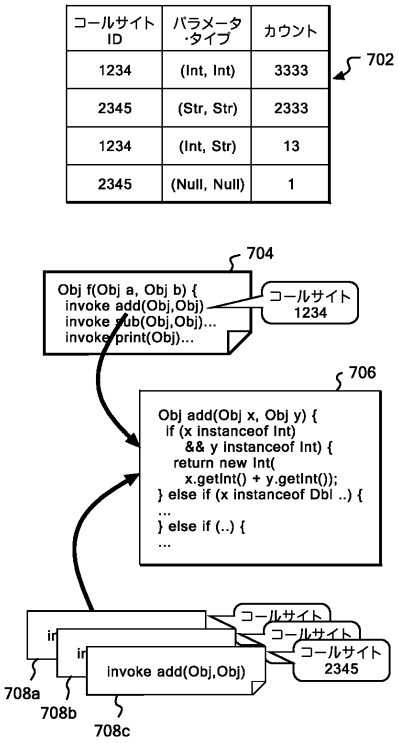
【図５】


開始
みなしマルチメソッドのボディの最初に、
パラメータ・タイプの組み合わせを記録する
コードを挿入する
502
記録時間の終了まで、もとのコードを
実行する
504
終了

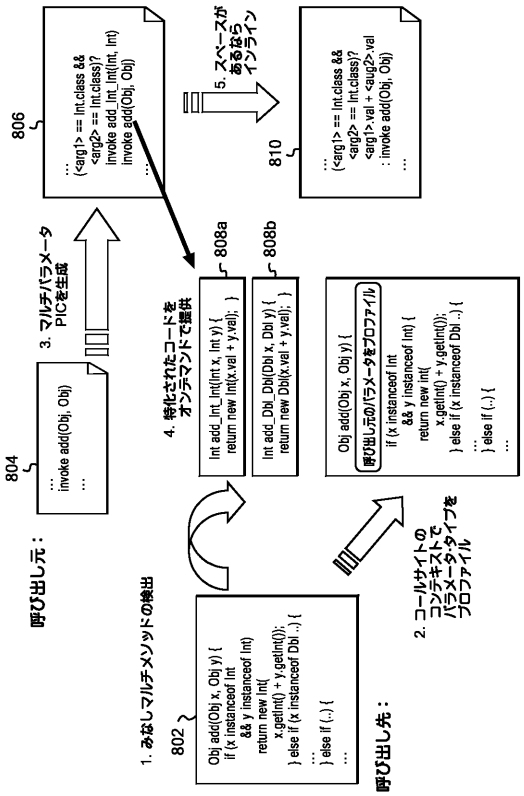
【図６】


開始
このメソッドをコールするコールサイトがあるかどうかコール･スタックを見る
602
メソッド･パラメータの実際のタイプを見る
604
このメソッドについて、プロファイリング･テーブルから、コールサイトとパラメータの組み合わせのエントリを検索し、もし存在しないなら、カウンタ＝０のエントリを作成する
606
その組み合わせのエントリのカウンタをインクリメントする
608
終了

【図７】


コールサイトID
パラメータ･タイプ
カウント
1234
(Int, Int)
3333
2345
(Str, Str)
2333
1234
(Int, Str)
13
2345
(Null, Null)
1
702
704
Obj f(Obj a, Obj b) {
invoke add(Obj,Obj)
invoke sub(Obj,Obj)...
invoke print(Obj)...
コールサイト 1234
706
Obj add(Obj x, Obj y) {
if (x instanceof Int)
&& y instanceof Int) {
return new Int(
x.getInt() + y.getInt());
} else if (x instanceof Dbl ..) {
...
} else if (..) {
...
コールサイト
コールサイト
コールサイト 2345
invoke add(Obj,Obj)
708a
708b
708c

【図８】


806
804
呼び出し元：
3. マルチパラメータPICを生成
4. 特化されたコードをオンデマンドで提供
5. スペースがあるならインライン
808a
808b
810
1. みなしマルチメソッドの検出
802
呼び出し先：
2. コールサイトのコンテキストでパラメータタイプをプロファイル

フロントページの続き

(72)発明者　立堀　道昭
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内
(72)発明者　河内谷　清久仁
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内
(72)発明者　デレック・イングリス
カナダ国　ＬＧ６　１Ｃ７　オンタリオ州　マーカム　トロント・ラボ　ワーデン・アベニュ　８２００
(72)発明者　小野寺　民也
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内

審査官　多賀　実

(56)参考文献　特開２０００－２０７２１２（ＪＰ，Ａ）
特開２００５－３４６４０７（ＪＰ，Ａ）
小野寺民也，オブジェクト指向言語におけるメッセージ送信の高速化技法，情報処理，日本，社団法人情報処理学会，１９９７年　4月１５日，第３８巻，第４号，ｐ．３０１－３１０

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４４
Ｇ０６Ｆ　　９／４５