(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2005-293080
(P2005-293080A)
(43)公開日　平成17年10月20日(2005.10.20)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 9/45	G06F	9/44　322G	5B081

審査請求 未請求　請求項の数 9　ＯＬ　（全 7 頁）

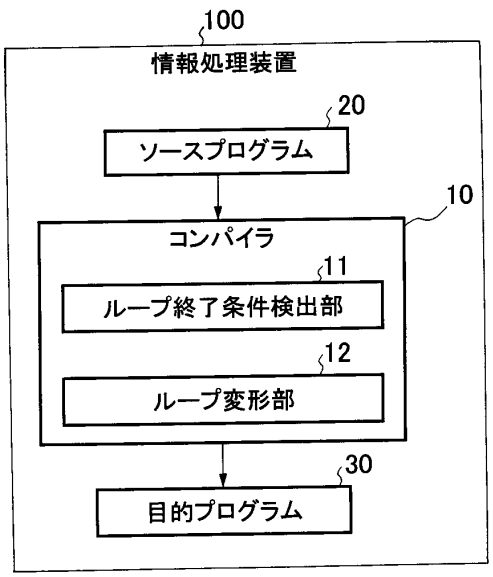
(21)出願番号　特願2004-105437 (P2004-105437)
(22)出願日　平成16年3月31日 (2004.3.31)

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号
(74)代理人　100065385
弁理士　山下　穣平
(74)代理人　100122921
弁理士　志村　博
(74)代理人　100130029
弁理士　永井　道雄
(74)代理人　100065385
弁理士　山下　穣平
(72)発明者　稲葉　勝
東京都港区芝五丁目７番１号　日本電気株式会社内

Ｆターム(参考)　5B081 AA06　AA09　CC33　CC41

(54)【発明の名称】情報処理装置、プログラム処理方法及びコンパイル処理プログラム

(57)【要約】
【課題】　ループの継続条件式に非等値演算子を使用したループ制御文が示すループのベクトル演算を迅速に実行する方法を提供する。
【解決手段】　情報処理装置（100）は、ソースプログラム（20）に対するコンパイル処理によりベクトル命令を含む目的プログラム（30）を生成するためのコンパイル処理手段（10）を備える。コンパイル処理手段は、入力されたソースプログラムから、制御変数とループ不変の終値とが非等値演算子により関連付けられた継続条件式を有するループ制御文（6-1）を検出する検出処理部（11）と、検出されたループ制御文を、該ループ制御文に等価であり且つそれぞれが前記終値を基点とするループを実行するための複数のループ制御文に変形する変形処理部（12）とを有する。
【選択図】　図１



【特許請求の範囲】
【請求項１】
ソースプログラムに対するコンパイル処理によりベクトル命令を含む目的プログラムを生成するためのコンパイル処理手段を備え、
前記コンパイル処理手段は、
入力されたソースプログラムから、制御変数とループ不変の終値とが非等値演算子により関連付けられた継続条件式を有するループ制御文を検出する検出処理部と、
前記検出処理部により検出されたループ制御文を、該ループ制御文に等価であり且つそれぞれが前記終値を基点とするループを実行するための複数のループ制御文に変形する変形処理部とを有することを特徴とする情報処理装置。
【請求項２】
前記変形処理部は、前記制御変数と前記終値とが大小比較演算子により関連付けられた継続条件式を有する第１のループ制御文と、前記制御変数の初期値が空値であり且つ前記制御変数と前記終値とが非等値演算子により関連付けられた継続条件式を有する第２のループ制御文とを生成することを特徴とする請求項１記載の情報処理装置。
【請求項３】
前記変形処理部は、前記制御変数の増減条件に対応した前記第１のループ制御文及び第２のループ制御文を生成することを特徴とする請求項２記載の情報処理装置。
【請求項４】
ソースプログラムに対するコンパイル処理によりベクトル命令を含む目的プログラムを生成する情報処理装置が、
ソースプログラムを入力されたとき、該ソースプログラムから制御変数とループ不変の終値とが非等値演算子により関連付けられた継続条件式を有するループ制御文を検出し、該検出したループ制御文を、該ループ制御文に等価であり且つそれぞれが前記終値を基点とするループを実行するための複数のループ制御文に変形することを特徴とするプログラム処理方法。
【請求項５】
前記情報処理装置が、前記検出したループ制御文を変換するとき、前記制御変数と前記終値とが大小比較演算子により関連付けられた継続条件式を有する第１のループ制御文と、前記制御変数の初期値が空値であり且つ前記制御変数と前記終値とが非等値演算子により関連付けられた継続条件式を有する第２のループ制御文とを生成することを特徴とする請求項４記載のプログラム処理方法。
【請求項６】
前記情報処理装置が、前記制御変数の増減条件に対応した前記第１のループ制御文及び第２のループ制御文を生成することを特徴とする請求項５記載のプログラム処理方法。
【請求項７】
ソースプログラムに対するコンパイル処理によりベクトル命令を含む目的プログラムを生成するコンパイル処理ステップをコンピュータに実行させるコンパイル処理プログラムであって、
ソースプログラムを入力されたとき、前記コンパイル処理ステップに先立ち、
前記入力されたソースプログラムから、制御変数とループ不変の終値とが非等値演算子により関連付けられた継続条件式を有するループ制御文を検出する検出処理ステップと、
前記検出処理ステップにおいて検出されたループ制御文を、該ループ制御文に等価であり且つそれぞれが前記終値を基点とするループを実行するための複数のループ制御文に変形する変形処理ステップとを前記コンピュータに実行させることを特徴とするコンパイル処理プログラム。
【請求項８】
前記変換処理ステップにおいて、前記制御変数と前記終値とが大小比較演算子により関連付けられた継続条件式を有する第１のループ制御文と、前記制御変数の初期値が空値であり且つ前記制御変数と前記終値とが非等値演算子により関連付けられた継続条件式を有

する第２のループ制御文とを生成することを前記コンピュータに実行させることを特徴とする請求項７記載のコンパイル処理プログラム。

【請求項９】

前記変換処理ステップにおいて、前記制御変数の増減条件に対応した前記第１のループ制御文及び第２のループ制御文を生成することを前記コンピュータに実行させることを特徴とする請求項８記載のコンパイル処理プログラム。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、Ｃ言語あるいはＣ＋＋言語のようなプログラミング言語にて記述されたソースプログラムからベクトル命令を含む目的プログラムを生成するコンパイル処理に関し、特に、繰り返し回数が指定されていないループ制御文が含まれるソースプログラムに前記コンパイル処理を施すための手法に関する。

【背景技術】

【０００２】

従来、ソースプログラムにコンパイル処理を施すにあたり、例えば図５の５－１式ように、繰り返し回数が指定されているループ制御文に対しては、ベクトル命令を適用することにより演算処理の効率化を図ることが知られている。図示の５－１式は、ループ制御文の一例として示したＣ言語のfor文であり、文中の継続条件式５ａにより、ループの繰り返し回数が規定される。

【０００３】

ところで、図６の６－１式のようなループ制御文によれば、上記５－１式による演算結果と同様な結果が得られると考えられるが、図示の６－１式は、その継続条件式６ａに「!=」なる非等値演算子が使用されていることから、ループ外への飛び出しを伴うアルゴリズムとなる。

【０００４】

図６のようなループ制御文は、繰り返し回数が不明であることから、これをベクトル化すると、演算処理が長期化するか、あるいは無限ループに陥る可能性が高い。そのため、一般的には、この種のループ制御文はベクトル化しないよう予めコンパイラ側に設定を行うが、例えば、後述の特許文献１及び２の手法を用いれば、コンパイル処理にマスク演算やループ後の処理を付加することにより、上記のような繰り返し回数が不明のループ制御文であっても適正なベクトル演算が可能となる。

【特許文献１】特開平０８－２７２７７７号公報

【特許文献２】特開平１１－２５００３５号公報

【発明の開示】

【発明が解決しようとする課題】

【０００５】

しかしながら、上記の特許文献１及び２の手法を用いた場合、マスク演算やループ後の処理といった付加的な処理を必要とされることから、その分、プログラムの実行時間が長引くという不都合がある。

【０００６】

本発明は、上記課題に鑑みてなされたものであり、継続条件式に非等値演算子が用いられるループ制御文が示す演算内容をベクトル命令に沿って迅速に実行し得る情報処理装置、及び、該装置のためのプログラム処理方法、並びに、コンパイル処理プログラムを提供することを目的とする。

【課題を解決するための手段】

【０００７】

本発明に係る情報処理装置は、ソースプログラムに対するコンパイル処理によりベクトル命令を含む目的プログラムを生成するためのコンパイル処理手段を備え、前記コンパイル処理手段は、入力されたソースプログラムから、制御変数とループ不変の終値とが非等値演算子により関連付けられた継続条件式を有するループ制御文を検出する検出処理部と、前記検出処理部により検出されたループ制御文を、該ループ制御文に等価であり且つそれぞれが前記終値を基点とするループを実行するための複数のループ制御文に変形する変形処理部とを有する。

【０００８】

本発明に係るプログラム処理方法は、ソースプログラムに対するコンパイル処理によりベクトル命令を含む目的プログラムを生成する情報処理装置が、ソースプログラムを入力されたとき、該ソースプログラムから制御変数とループ不変の終値とが非等値演算子により関連付けられた継続条件式を有するループ制御文を検出し、該検出したループ制御文を、該ループ制御文に等価であり且つそれぞれが前記終値を基点とするループを実行するための複数のループ制御文に変形するという方法である。

【０００９】

本発明に係るコンパイル処理プログラムは、ソースプログラムに対するコンパイル処理によりベクトル命令を含む目的プログラムを生成するコンパイル処理ステップをコンピュータに実行させるコンパイル処理プログラムであって、ソースプログラムを入力されたとき、前記コンパイル処理ステップに先立ち、前記入力されたソースプログラムから、制御変数とループ不変の終値とが非等値演算子により関連付けられた継続条件式を有するループ制御文を検出する検出処理ステップと、前記検出処理ステップにおいて検出されたループ制御文を、該ループ制御文に等価であり且つそれぞれが前記終値を基点とするループを実行するための複数のループ制御文に変形する変形処理ステップとを前記コンピュータに実行させる。

【発明の効果】

【００１０】

本発明によれば、継続条件式に非等値演算子を使用したループ制御文を、終値を基点とした等価なループ制御文に変形することから、コンパイル処理により生成される当該ベクトル命令を単純化することができる。これにより、ループのベクトル演算を高速に実行することが可能となる。

【発明を実施するための最良の形態】

【００１１】

［実施例］

以下、本発明の実施例について図面を用いて詳細に説明する。図１は、本発明による実施例の情報処理装置の構成を示すブロック図である。実施例の情報処理装置１００は、本発明に係るコンパイル処理手段に対応するコンパイラ１０を備える。コンパイラ１０は、Ｃ言語あるいはＣ＋＋言語などにより記述されたソースプログラム２０に対するコンパイル処理により、ベクトル命令を含む目的プログラム３０を生成することを情報処理装置１００に実行させるコンパイル処理プログラムである。

【００１２】

コンパイラ１０は、本発明の情報処理装置における検出処理部の機能を果たすループ終了条件検出部１１と、変形処理部の機能を果たすループ変形部１２とを有する。ループ終了条件検出部１１は、コンパイラ１０に入力されたソースプログラム２０から、制御変数とループ不変の終値とが「!=」なる非等値演算子により関連付けられた継続条件式を有するループ制御文を検出する。ループ変形部１２は、ループ終了条件検出部１１が検出したループ制御文を、このループ制御文に等価であり且つそれぞれが前記終値を基点とするループを実行するための複数のループ制御文に変形する。

【００１３】

図２に示すフローチャートに沿って、情報処理装置１００のコンパイル処理手順を説明する。ここでは、説明のため、処理対象となるソースプログラム２０に図６に示す６－１

式のfor文が含まれており、また、「（初期値）＜（終値）」及び「（増分値）＞０」であることと、「（終値）」はループ演算中に値が変化しないループ不変のオペランドであることとが成り立っているとする。

【００１４】

まず、コンパイラ１０にソースプログラム２０が入力されると（ステップＳ１）、ループ終了条件検出部１１は、上記説明した検出条件に対応するループ制御文として図６の６－１式を検出する（ステップＳ２）。６－１式のfor文は、その継続条件式６ａとして、制御変数である「ｉ」と、ループ不変のオペランドの「（終値）」との関係が非等値演算子の「!=」により定義されたものである。ここでは、制御変数「ｉ」の値が、「（終値）」を除き、「（初期値）」から「（増分値）」の間隔で順次増加するよう制御される。

【００１５】

ループ変形部１２は、ループ終了条件検出部１１が検出した６－１式を、これに等価なループ制御文に変換する（ステップＳ３）。図３に、ループ変形部１２による変換結果を示す。図示の３－１式は、大小比較演算子を使用した「ｉ＜（終値）」が継続条件式３ａとして設定されたfor文であり、また、他方の３－２式は、「（初期値）」の設定が空値であり、継続条件式３ｂとして「ｉ!=（終値）」が設定されたfor文である。

【００１６】

図３に示す変換後のループ制御文に、上記の「（初期値）＜（終値）」及び「（増分値）＞０」なる前提を加味すると、３－１式は、制御変数「ｉ」に与える値の遷移に関し、「（初期値）」から「（終値）」の直前までの漸増を制御し、また、他方の３－２式は、「（終値）」を含まない、それ以降の漸増を制御する。すなわち、３－１式及び３－２式の組み合わせは、「（終値）」を基点として、変換前の６－１式の制御変数「ｉ」が取り得る値の範囲を２分割したものに相当する。よって、変換前の６－１式と、変換後の３－１式及び３－２式の組み合わせとは等価の関係にある。

【００１７】

ループ変形部１２により変換された図３のループ制御文は、ベクトル処理が可能な従来のコンパイラと同様な処理によりベクトル命令に変換され（ステップＳ４）、コンパイル処理された他の命令列と共に、目的プログラム３０として出力される（ステップＳ５）。

【００１８】

なお、図３に示す変換後のループ制御文のうち、「（終値）」より以降の漸増を担う３－２式は、継続条件式３ｂに「!=」を使用していることから、無限ループの発生や演算処理の長期化といった問題が生じ易い。よって、この３－２式は、ベクトル化する前に削除する、あるいは従来と同様に、ベクトル化されても演算を実行しないような設定を、予めコンパイラ１０に行うことが望ましい。

【００１９】

以上説明した実施例によれば、ソースプログラム２０のコンパイル処理に先立ち、継続条件式に「!=」を使用したループ制御文を、「＜」のような大小比較演算子を用いた等価なループ制御文に変換することから、コンパイル処理により生成されるベクトル命令列を単純化することができる。これにより、ベクトル演算を高速に実行することが可能となる。

【００２０】

上記実施例で説明した図３に示すループ制御文は、制御変数「ｉ」の値を「（初期値）」から「（終値）」に向けて漸増させることを前提として、図６の６－１式を変形したものであるが、本発明を実施するにあたっては、このような前提に依存しない汎用的なループ制御文を設定することもできる。

【００２１】

図４は、６－１式をループ変形部１２により汎用的なものに変換した結果であり、そのアルゴリズムは、上記説明した等価なループ制御文を、「ｉ」の増減条件に応じて生成するための判断処理を行うものとなっている。図４において、４－１式は、図３の３－１式及び３－２式と同様である。すなわち、４－１式は、６－１式のオペランドが、上記実施

例にて説明した「（初期値）＜（終値）」及び「（増分値）＞０」の場合に適用されるループ制御文である。

【００２２】

また、４－２式は、６－１式のオペランドが「（初期値）＞（終値）」及び「（増分値）＜０」の場合、すなわち、制御変数「ｉ」の値を「（初期値）」から「（終値）」に向けて漸減させる場合に適用するループ制御文である。

【００２３】

４－３式は、例えば、「（初期値）＝（終値）」、又は「（増分値）＝０」、又は「（初期値）＜（終値）」及び「（増分値）＜０」、又は「（初期値）＞（終値）」及び「（増分値）＞０」のように、実質的にループが形成されない場合には、制御変数「ｉ」の値が変化せず、ベクトル化の必要はないことから、ループ制御文は６－１式のままの形状とする。また、明示的なループからの飛び出しがない限り無限ループになる場合にも、４－３式が適用される。

【００２４】

このように、図４に示すような汎用的なループ制御文を用いることにより、コンパイラ１０は、ループ終了条件検出部１１が検出するループ制御文のオペランド内容を事前に把握することなく、上記説明した実施例により生成されるベクトル命令と同様なものを生成することができる。

【００２５】

上記実施例では、本発明に係るループ制御文としてfor文を用いたが、for文に限らず、例えば、while文や、goto文を使用したループに適用することも可能である。また、Ｃ＋＋言語のＳＴＬ（Standard Template Library）を利用した際は、従来知られているように、ループの継続条件式に「!=」が使用されるが、本発明は、Ｃ＋＋言語のソースプログラムのうち、ＳＴＬのvectorやvalarrayのようなメモリに対し連続的なアクセスを実行するアルゴリズムを含むものに好適である。

【図面の簡単な説明】

【００２６】

【図１】本発明による実施例の情報処理装置の構成を示すブロック図である。

【図２】実施例の動作手順を示すフローチャートである。

【図３】実施例の変換処理ステップを説明するための説明図である。

【図４】他の実施例の変換処理ステップを説明するための説明図である。

【図５】一般的なループ制御文（その１）を説明するための説明図である。

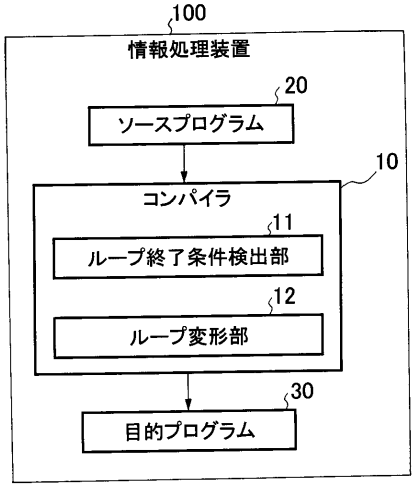
【図６】一般的なループ制御文（その２）を説明するための説明図である。

【符号の説明】

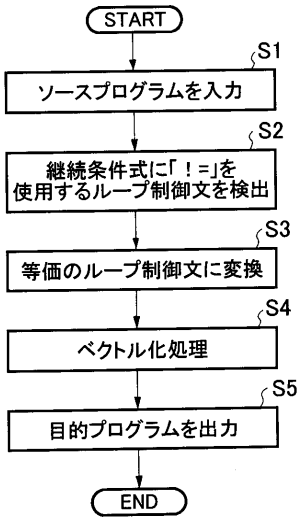
【００２７】

１００　情報処理装置
１０　コンパイラ
１１　ループ終了条件検出部
１２　ループ変形部
２０　ソースプログラム
３０　目的プログラム

【図１】



【図２】



【図３】

```
for(i=(初期値);i<(終値);i+=(増分値)) {
…
}                                          3-1
for(;i!=(終値);i+=(増分値)) {
…
}                                          3-2
```

【図４】

```
int i;
if((初期値)<(終値)) {
  if(増分値>0) {
    for(i=(初期値);i<(終値);i+=(増分値)) {
    …
    }
    for(;i!=(終値);i+=(増分値)) {
    …
    }                                      4-1
  }
}else if(増分値<0) {
  for(i=(初期値);i>(終値);i+=(増分値)) {
  …
  }
  for(;i!=(終値);i+=(増分値)) {
  …
  }                                        4-2
} else {
  for(i=(初期値); i!=(終値);i+=(増分値)) {
  …
  }                                        4-3
}
```

【図５】

```
                5a:継続条件式
int i;
for(i=(初期値); i<(終値) ;i+=(増分値)) {
…
}                                          5-1
```

【図６】

```
                6a:継続条件式
int i;
for(i=(初期値); i!=(終値) ;i+=(増分値)) {
…
}                                          6-1
```