

(19) 日本国特許庁 (JP)

(12) 特 許 公 報 (B2)

(11) 特許番号

特許第6193151号

(P6193151)

(45) 発行日 平成29年9月6日 (2017.9.6)

(24) 登録日 平成29年8月18日 (2017.8.18)

(51) Int.Cl.

F 1

G 0 6 F 9/45 (2006.01)

G 0 6 F 9/44 3 2 2 F

請求項の数 4 (全 9 頁)

(21) 出願番号	特願2014-37149 (P2014-37149)	(73) 特許権者	000003078
(22) 出願日	平成26年2月27日 (2014.2.27)		株式会社東芝
(65) 公開番号	特開2014-194759 (P2014-194759A)		東京都港区芝浦一丁目1番1号
(43) 公開日	平成26年10月9日 (2014.10.9)	(74) 代理人	110000235
審査請求日	平成28年9月23日 (2016.9.23)		特許業務法人 天城国際特許事務所
(31) 優先権主張番号	特願2013-40932 (P2013-40932)	(72) 発明者	嶋村 誠
(32) 優先日	平成25年3月1日 (2013.3.1)		東京都港区芝浦一丁目1番1号 株式会社
(33) 優先権主張国	日本国 (JP)		東芝内
		(72) 発明者	山地 圭
			東京都港区芝浦一丁目1番1号 株式会社
			東芝内
		(72) 発明者	金松 基孝
			東京都港区芝浦一丁目1番1号 株式会社
			東芝内

最終頁に続く

(54) 【発明の名称】 多分岐判断構文の最適化処理装置

(57) 【特許請求の範囲】

【請求項 1】

トークンを含む多分岐判断構文を保持するメモリと、

前記多分岐判断構文の最適化処理装置内に組み込まれるCPUに依存するパラメータであって、多分岐判断処理に要する時間に基づいて設定するCPUの能力情報と、

予めソースコードを静的に分析し、字句解析するクエリの中で、出現頻度順に並び替えたトークンの種別と出現率を表した頻出字句表を保持するデータベース部と、

前記CPUの能力情報と頻出字句表を参照して前記多分岐判断構文に対して判定を実行し、出現頻度の高いトークン種別では投機的実行可能な分岐をさせ、出現頻度の低いトークン種別ではジャンプテーブルを使用した分岐をさせるように変換した分岐コードを生成する変換部を備える多分岐判断構文の最適化処理装置。

10

【請求項 2】

前記多分岐判断構文は、switch 文であり、前記投機的実行可能な分岐は、if-else型のswitch 文による分岐である請求項1記載の多分岐判断構文の最適化処理装置。

【請求項 3】

前記分岐コードは、コンパイラが解釈する中間コードである請求項1又は請求項2に記載の多分岐判断構文の最適化処理装置。

【請求項 4】

トークンを含む多分岐判断構文を蓄積した第1ファイルと、

CPUに依存するパラメータであって、多分岐判断処理に要する時間に基づいて設定す

20

るCPUの能力情報を蓄積した第2ファイルと、

予めソースコードを静的に分析し、字句解析するクエリの中で、出現頻度順に並び替えたトークンの種別と出現率を表した頻出字句表を蓄積したデータベースファイルとを含んでおり、

コンピュータに、前記CPUの能力情報と頻出字句表を参照して前記多分岐判断構文に対して判定を実行し、出現頻度の高いトークン種別では投機的実行可能な分岐をさせ、出現頻度の低いトークン種別ではジャンプテーブルを使用した分岐をさせるように変換した分岐コードを生成させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

10

【技術分野】

【0001】

本発明の実施形態は、多分岐判断構文の最適化処理装置に関する。

【背景技術】

【0002】

一般に、字句解析においては、まず処理すべき初めの文字を検査し、どのトークンを判定すればよいかを分けて、その後に詳細な判定を行っている。

【0003】

リレーショナルデータベースマネジメントシステムでは、字句解析に要する処理時間の割合が小さいので、字句解析の処理速度は大きな問題とはならない。

20

【0004】

しかし、いわゆるインメモリデータベースでは、I/O機器の処理速度も高速であるから、字句解析に要する処理時間の高速化が求められる。

【0005】

字句解析で用いる多分岐判断構文ではジャンプテーブル（分岐先テーブル）が利用されることが多い。しかし、ジャンプテーブルでは、分岐先のアドレスをメモリからレジスタへロードするレジスタ間接ジャンプであるため、メモリを読むまで分岐先が決まらない。

このため分岐先の投機的実行ができず、結果として分岐前のパイプラインフラッシュ待ちやリロードのため時間を要し処理時間が長くなるという問題があった。

【先行技術文献】

30

【特許文献】

【0006】

【特許文献1】特開平9-288581号公報

【特許文献2】特開平7-160490号公報

【発明の概要】

【発明が解決しようとする課題】

【0007】

本発明が解決しようとする課題は、投機的実行を上手く活用可能な多分岐判断構文の最適化処理装置を提供することである。

【課題を解決するための手段】

40

【0008】

実施形態の多分岐判断構文の最適化処理装置は、トークンを含む多分岐判断構文を保持するメモリと、前記多分岐判断構文の最適化処理装置内に組み込まれるCPUに依存するパラメータであって、多分岐判断処理に要する時間に基づいて設定するCPUの能力情報と、予めソースコードを静的に分析し、字句解析するクエリの中で、出現頻度順に並び替えたトークンの種別と出現率を表した頻出字句表を保持するデータベース部と、前記CPUの能力情報と頻出字句表を参照して前記多分岐判断構文に対して判定を実行し、出現頻度の高いトークン種別では投機的実行可能な分岐をさせ、出現頻度の低いトークン種別ではジャンプテーブルを使用した分岐をさせるように変換した分岐コードを生成する変換部を備える。

50

【図面の簡単な説明】

【0009】

【図1】投機的実行が可能／不可能な表現形式を説明する図である。

【図2】アセンブラレベルでのif-else型のswitch文とジャンプテーブルとの対比を説明する図である。

【図3】投機的実行による高速処理を説明する図である。

【図4】本発明の実施形態に係る多分岐判断構文の最適化処理装置の概略構成を示すブロック図である。

【図5】実施形態に係る多分岐判断構文の最適化処理装置における最適化処理の流れを示すフローチャートである。

10

【発明を実施するための形態】

【0010】

以下、本発明の一実施の形態について、図面を参照して説明する。尚、各図において同一箇所については同一の符号を付すとともに、重複した説明は省略する。以下は本発明をSQLクエリの字句解析処理について実施した形態である。

【0011】

まず、本実施形態で用いる主要な用語について説明する。

【0012】

「字句解析」とは、プログラミング言語で記述されたプログラムを解析する処理であって、クエリ中の文字、スペース、記号を判別し、文字列をトークンに区切る作業をいう。

20

【0013】

「構文解析」とは、プログラミング言語で記述されたプログラムを解析する処理であって、トークン列がプログラミング言語の文法に沿っているかを調べる処理をいう。

【0014】

「クエリ」は、字句解析装置へ入力する文字列である。なお、以下では、クエリを構成する文字列中のアルファベットは大文字にあらかじめ統一しているものとする。

【0015】

「トークン」とは、文字列の最小単位であって、構文解析装置が受理する予約語あるいは記号ID等をいう。トークンの種類には、他に、演算子、空白文字、テキスト、エスケープ、カッコ、コメント等がある。

30

【0016】

「ジャンプテーブル」とは、分岐で用いる値に対して分岐先のアドレスが書いてある表をいう。

【0017】

「投機的実行」：多分岐判断構文をコンパイルしてオブジェクトコードを生成する際の最適化処理の方法として、投機的実行がある。投機的実行は、プログラムが途中で分岐している場合に、分岐した先の処理を予め実行しておくことである。予測が当たっていれば処理の確定を待つことがないので、その分岐処理速度は高速化される。予測が外れたら投機的に実行された結果は破棄し、正しい分岐先の命令を実行する。

【0018】

一般に、字句解析では、まず処理すべき初めの文字を検査して、どの種別のトークンを判定（例えば、アルファベットなのか数字なのか）すればよいかを分岐し、その後に詳細な判定（例えば、アルファベットであれば、“S”か“F”か等）を行っている。ここで、各トークン種別に対する分岐が発生する。

40

【0019】

C言語などの手続き型プログラミング言語において、多分岐判断構文としてswitch文がある。switch文は、評価する変数の値に応じてその後の処理を異にするもので、コンパイラによって、ジャンプテーブルを用いた形に変換されることがある。後述するif-else型のswitch文であれば、静的な分岐の連続なので、投機的実行が可能な表現形式である。図1は、投機的実行が可能／不可能な表現形式を説明する図である。図2は、アセンブ

50

ラレベルでのif-else 型の switch 文とジャンプテーブルとの対比を説明する図である。図 3 は、投機的実行による高速処理を説明する図である。図 3 に示す例では、1 億回、ほぼランダムに分岐するループを測定した場合を比較するもので、投機的実行の活用により処理時間が約 3 倍の差になっている。

【0020】

本実施形態においては、投機的実行を有効なものとするため、字句解析におけるトークンの判断を行うswitch 文に対して、出現頻度の高いトークン種別では投機的実行可能な分岐をさせ、出現頻度の低いトークン種別ではジャンプテーブルを使用した分岐をするようにソースコードを変換する。

【0021】

図 4 は、本発明の実施形態に係る多分岐判断構文の最適化処理装置の概略構成を示すブロック図である。この装置は汎用のコンピュータ（例えばパーソナルコンピュータ（PC）等）と、同コンピュータ上で動作するソフトウェアとを用いて実現される。コンピュータとしては、CAD（Computer Aided Design）やCAE（Computer Aided Engineering）に好適なエンジニアリングワークステーション（EWS）等も含む。本実施形態はこのようなコンピュータに、switch 文から出現頻度の高いトークン種別、分岐判定に係る一連の手続きを実行させるプログラムとして実施することもできる。

【0022】

図 4 に示すように、本実施形態に係る多分岐判断構文の最適化処理装置は、主として、入力部 11、メモリ 12、データベース 13、変換部 14 から構成されている。

【0023】

入力部 11 は、多分岐判断構文であるswitch 文を入力する。入力したswitch 文は、メモリ 12 に保持される。メモリ 12 に保持したswitch 文は、変換部 14 に送られる。

【0024】

データベース 13 は、“CPU の能力情報”と“頻出字句表”を保持する。

【0025】

ここで、CPU の能力情報とは、CPU に依存するパラメータであって、多分岐判断処理に要する時間に基づいて設定する。CPU の能力情報 = n とおくと、 n 回の比較命令の実行時間 = ジャンプテーブルのアクセス時間となるように決定する比較数である。

【0026】

n の設定では、例えば、else 節が記述されている if 文であって、分岐の条件式が FALSE だった場合は else の直後に指定された文を実行するif-else 文による連続比較と、ジャンプテーブルを利用した場合を対比する。本発明者は、CPU としてIntel社製のCore i7（登録商標）で実験した。それによれば、15 分岐の場合：if 文による処理時間は1874 ミリ秒であり、一方、ジャンプテーブルによる処理時間は1892 ミリ秒で、ほぼ同等であった。16 分岐の場合：if 文による処理時間は531 ミリ秒であり、一方、ジャンプテーブルによる処理時間は804 ミリ秒であった。さらに、32 分岐の場合：if 文による処理時間は1184 ミリ秒であり、一方、ジャンプテーブルによる処理時間は873 ミリ秒であった。実験から、16 分岐までは、ジャンプテーブルによる処理の方が遅いことが分かった。以上のことから、CPU がIntel社製のCore i7（登録商標）の場合、CPU の能力情報 = n は、16 比較まで本実施形態による最適化の有効性判断に適しているといえることができる。

【0027】

頻出字句表とは、字句解析するクエリの中で、どの字句が頻出するかを記述した表（字句種類と確率のペア）である。例えば、SQLであれば、アルファベット、スペース、数字、*、+、-、（ ）が頻出とする。

【0028】

<頻出字句表の決め方>

一例として、次のSQLクエリで説明する。頻出字句表は、予めソースコードを読み込んで静的に分析した結果を受けて作成する。

【0029】

“SELECT FUNC(*) FROM TABLE WHERE COND(P+1)=1 AND X=Y”

予約語の出現回数：4回 (SELECT, FROM, WHERE, AND)

I Dの出現回数：6回 (FUNC, TABLE, COND, P, X, Y)

スペースの出現回数：7回

()の出現回数：合わせて4回

数字の出現回数：2回 (1が2回)

=の出現回数：2回

+の出現回数：1回

*の出現回数：1回

10

したがって、この例では、トークンの総数は27となる。

【0030】

そして、トークン毎の出現率は、種別毎に以下ようになる。

【0031】

アルファベットの出現率：37% (10/27)

スペースの出現率：26% (7/27)

()の出現率：15% (4/27)

数字の出現率：7% (2/27)

= の出現率：7% (2/27)

+の出現率：3% (1/27)

*の出現率：3% (1/27)

20

したがって、出現頻度順は、アルファベット、スペース、()、数字、=、+、*となる。

【0032】

次に、読み込んだソースコードにおいて、各トークンの種別を判定する場合を考える。例えば、ASCIIコードで表された文字が“アルファベット”であることを判定するには、“A”以上で“Z”以下であることを判定するため、計2回の比較が必要である。同様に、“数字”であることを判定するには、“0”以上“9”以下、2回の比較が必要である。また、“スペース”であることを判定するには、SQL言語上における“スペース”がタブと改行コード2種類(キャリッジリターンCR、ラインフィードLF)を含むため、4回の比較が必要である。したがって、出現頻度順に並べた頻出字句表からは、2回の比較判定で1個の種別(アルファベット)が判定可能であり、6回の比較判定で2個の種別(アルファベットとスペース)が判定可能である。上述した例では、頻出字句表における種別数は7となる。

30

【0033】

本実施形態においては、巨大なswitch文のコンパイルにおいて、if-else型のswitch文として分岐させるのか、ジャンプテーブルを使用した分岐をさせるのが高速化できるかを、CPUの能力情報と頻出字句表を参照する。すなわち、n-比較に消費する回数=0になると、if-else型のswitch文として分岐させる方が遅くなる。換言すれば、出現頻度順に並べた頻出字句表からいくつかの種別を取り出し、それらの出現率の和がある閾値(例えば60%)を超えなかったら、高速化の効果は薄いことになる。

40

【0034】

そこで、例えば、出現率の高いトークンからそれら出現率を累積した合計が60%を超えるまでの種別数をmとする。上述した例では、アルファベットの出現率37%とスペースの出現率26%を合計すると63%となるから、mの最小値は2となる。CPUの能力を表すnがm以下であれば、if-else型のswitch文として分岐させることなく、ジャンプテーブルを使用した分岐をさせる方が高速となることを意味している。例えば、SQL文の場合には、文字cを例にとると、 $c \geq 'A'$ 、 $c \leq 'Z'$ 、 $c == ' '$ を判定しないと出現率が稼げないから、mは3以上となる。

【0035】

50

変換部14は、データベース13に保持されているCPUの能力情報と頻出字句表を参照しつつ、switch文を解析して、switch文から頻出するトークンの種別を抽出し、出現頻度の高いトークン種別では投機的実行可能な分岐をさせ、出現頻度の低いトークン種別ではジャンプテーブルを使用した分岐をさせるように変換した分岐コードを生成する。尚、この分岐コードは、いわゆる高級言語に限られず、コンパイラが解釈する中間コードであってもよい。

【0036】

＜字句解析処理の流れ＞

次に、以上のように構成された多分岐判断構文の最適化処理装置100における字句解析処理の流れについて説明する。

10

【0037】

図5は、実施形態に係る多分岐判断構文の最適化処理装置100における字句解析処理の流れを示すフローチャートである。上述したように、頻出するトークンの種別として、アルファベット、スペース、数字がある。そこで、これらをif文で判定していく。尚、予め小文字は大文字に変換しておく方が判定の際に好適である。

【0038】

まず、if文でアルファベットを判定する（ステップS51）。上述したように、アルファベットの特定には、3回の比較が必要である。

【0039】

アルファベットであれば（ステップS51でYes）、続きの文字列と合わせて予約語を検索する（ステップS52）。

20

【0040】

予約語であれば（ステップS52でYes）、トークンの種類は予約語であると決定（ステップS53）して、字句解析処理を終了する。予約語でなければ（ステップS52でNo）、トークンの種類はID名であると決定（ステップS54）して、字句解析処理を終了する。

【0041】

ステップS51でアルファベットでなければ（ステップS51でNo）、if文で数字を判定する（ステップS55）。数字の特定には、 $c \geq '0'$ 、 $c \leq '9'$ の2回の比較が必要である。

30

【0042】

数字であれば（ステップS55でYes）、続きの文字列が数字か否かを判定（ステップS56）して、字句解析処理を終了する。

【0043】

ステップS55で数字でなければ（ステップS55でNo）、if文で空白文字を判定する（ステップS57）。

【0044】

空白文字であれば（ステップS57でYes）、空白が続く間、入力を読み飛ばし（ステップS58）て、字句解析処理を終了する。

【0045】

ステップS57で空白文字でなければ（ステップS57でNo）、if-else型のswitch文で頻出字句を判定する（ステップS59）。

40

【0046】

頻出字句であれば（ステップS59でYes）、各字句に応じた処理を行った（ステップS60）後、字句解析処理を終了する。上述の例で言えば、()、=などの特定である。

【0047】

ステップS59で頻出字句でなければ（ステップS59でNo）、ジャンプテーブルのswitch文で頻出でない字句を判定する（ステップS61）。

【0048】

50

字句であれば（ステップ S 6 1 で Y e s）、各字句に応じた処理を行った（ステップ S 6 0）後、字句解析処理を終了する。字句でなければ（ステップ S 6 1 で N o）、エラーであり、字句解析処理を終了する。

【0049】

＜最適化の効果＞

本実施形態による最適化の効果を、SQLite の字句解析ルーチンと比較した。SQLite の字句解析では、演算子、特殊記号の判定に switch 文を使用しており、分岐のcase が43個である。default 部で予約語とID名を判定し、文脈判定は構文解析器に任せている。

【0050】

次の S Q L 文を100万回字句解析し、時間を測定した。

10

【0051】

‘ SELECT FUNC(*) FROM TABLE WHERE COND(P+1)=1 AND X=Y ’

字句解析は、アルファベット、スペース、数字、()、=、*、+ を先に判定し、その他をジャンプテーブルで判定した。C P Uは、Intel Core i7（登録商標）で、n = 16とした。

【0052】

その結果、SQLiteの字句解析ルーチンでは597ミリ秒であった。一方、本実施形態によれば381ミリ秒で、約36%速度が向上している。

【0053】

以上説明したように、本実施形態によれば、投機的実行を上手く活用可能な多分岐判断構文を最適化することにより、字句解析の高速化が図られる。

20

【0054】

本発明のいくつかの実施形態を説明したが、これらの実施形態は、例として提示したものであり、発明の範囲を限定することは意図していない。これら新規な実施形態は、その他の様々な形態で実施されることが可能であり、発明の要旨を逸脱しない範囲で、種々の省略、置き換え、変更を行うことができる。これら実施形態やその変形は、発明の範囲や要旨に含まれるとともに、特許請求の範囲に記載された発明とその均等の範囲に含まれる。

【符号の説明】

【0055】

30

100・・・多分岐判断構文の最適化処理装置

11・・・入力部

12・・・メモリ

13・・・データベース

14・・・変換部

【図 1】

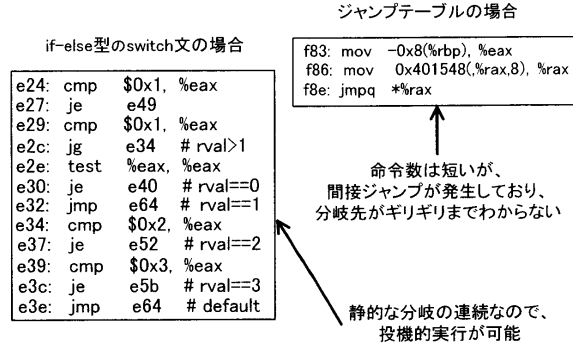
投機的実行可能な形
(if-else型のswitch文)

```
switch(rval) {
  case 0: ~ ; break ;
  case 1: ~ ; break ;
  case 2: ~ ; break ;
  case 3: ~ ; break ;
  default: ~ ; break ;
}
```

投機的実行不可能な形
(ジャンプテーブルを使用)

```
switch(rval) {
  case 0: ~ ; break ;
  case 1: ~ ; break ;
  case 2: ~ ; break ;
  case 3: ~ ; break ;
  case 4: ~ ; break ;
  case 5: ~ ; break ;
  default: ~ ; break ;
}
```

【図 2】



【図 3】

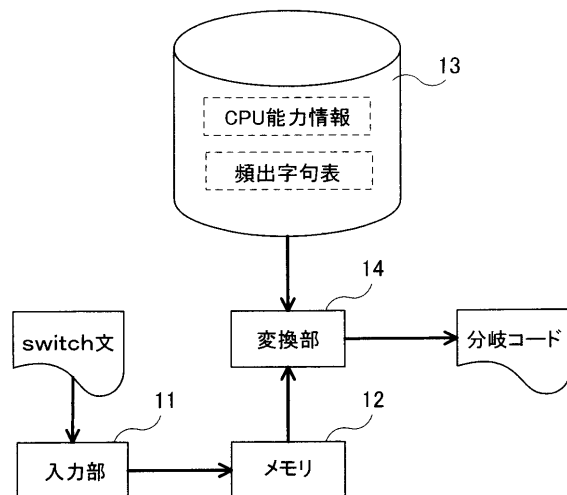
```
for(i=0;i<100*1000*1000;i++){
  r *= 103;
  int rval = (r >> 8) % 8;
  volatile int result;
  switch(rval){
    case 0: result = 0; break;
    case 1: result = 2; break;
    case 2: result = 7; break;
    case 3: result = 4; break;
    default:
      switch(rval){
        case 4: result = 100; break;
        case 5: result = 10; break;
        case 6: result = 100; break;
        case 7: result = 10; break;
      }
  }
}
```

304ミリ秒

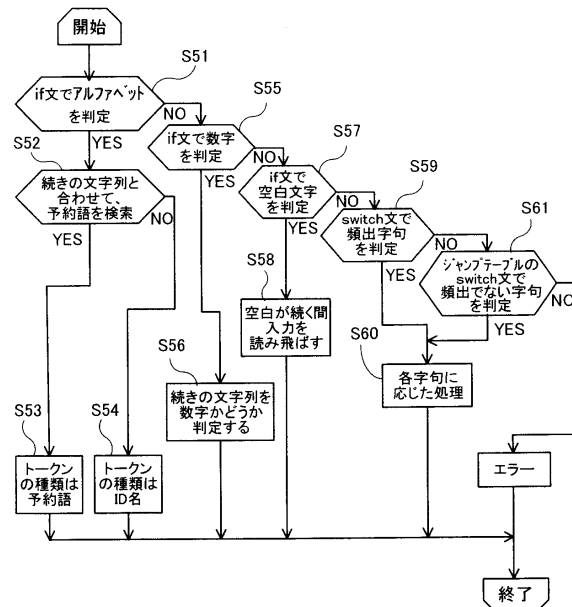
```
for(i=0;i<100*1000*1000;i++){
  r *= 103;
  int rval = (r >> 8) % 8;
  volatile int result;
  switch(rval){
    case 0: result = 0; break;
    case 1: result = 2; break;
    case 2: result = 7; break;
    case 3: result = 4; break;
    case 4: result = 100; break;
    case 5: result = 10; break;
    case 6: result = 100; break;
    case 7: result = 10; break;
  }
}
```

888ミリ秒

【図 4】



【図 5】



フロントページの続き

審査官 坂庭 剛史

(56)参考文献 特開平04-330527 (JP, A)

特開平07-160490 (JP, A)

特開平07-160499 (JP, A)

特開平09-288581 (JP, A)

特開2008-090439 (JP, A)

大貫 徹, 最新プロセッサを活かす設計手法と性能比較: 1章 最新プロセッサの性能を活かすシステム設計ノウハウ, Interface, 日本, CQ出版株式会社, 1998年 2月 1日, 第24巻, 第2号 (通巻248号), p. 118, 123

(58)調査した分野(Int.Cl., DB名)

G06F 9/45