(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 107250983 B
(45)授权公告日 2020.12.15

(21)申请号 201680010877.6
(22)申请日 2016.03.31
(65)同一申请的已公布的文献号
申请公布号 CN 107250983 A
(43)申请公布日 2017.10.13
(30)优先权数据
14/687,473 2015.04.15 US
(85)PCT国际申请进入国家阶段日
2017.08.17
(86)PCT国际申请的申请数据
PCT/CN2016/078078 2016.03.31
(87)PCT国际申请的公布数据
WO2016/165562 EN 2016.10.20
(73)专利权人 华为技术有限公司
地址 518129 广东省深圳市龙岗区坂田华为总部办公楼

(72)发明人 丁永华 张国根 朱澄

(74)专利代理机构 深圳市深佳知识产权代理事务所(普通合伙) 44285
代理人 王仲凯
(51)Int.Cl.
G06F 16/24(2019.01)
G06F 16/242(2019.01)
G06F 9/448(2018.01)

(56)对比文件
US 2007294679 A1,2007.12.20
US 2003070161 A1,2003.04.10
CN 1848088 A,2006.10.18

审查员 洪艳萍

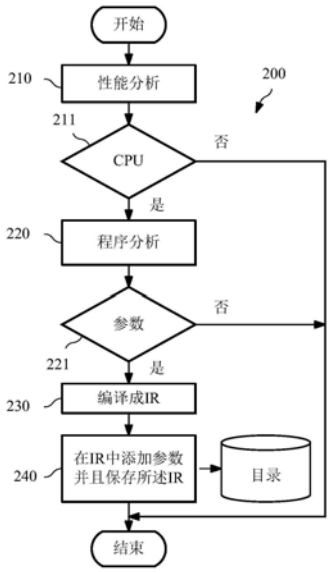
权利要求书2页 说明书9页 附图4页

(54)发明名称

查询即时JIT编译和执行的装置和方法

(57)摘要

本文所提供的实施例用于将参数化中间表示IR用于数据库查询执行引擎中的即时JIT编译。在一实施例中,一种支持在数据库管理系统中查询JIT编译和执行的方法包括:识别查询中的中央处理器密集函数,并在所述CPU密集函数中识别一个或者多个参数,其中,所述一个或者多个参数代表在不同查询实例中可以变化的值的变量;将所述CPU密集函数编译成包括所述一个或者多个参数的参数化IR;将所述CPU密集函数的参数化IR保存在参数化IR的目录中。



CN 107250983 B

1.一种查询即时JIT编译和执行的方法,其特征在于,所述方法包括:

识别查询中的中央处理器CPU密集函数,所述CPU密集函数用于执行数据库查询;

在所述CPU密集函数中识别一个或者多个参数,其中所述一个或者多个参数代表在进行数据库查询时不同查询实例中可以变化的值的变量;

将所述CPU密集函数编译成包括所述一个或者多个参数的参数化中间表示IR;其中,所述一个或者多个参数用于在执行传入查询时被所述变量的常数值替代;

将所述CPU密集函数的参数化IR保存在参数化IR的目录中。

2.根据权利要求1所述的方法,其特征在于,还包括:

在准备执行传入查询时,从所述目录加载所述参数化IR;

在所述参数化IR中利用所述变量的常数值替代所述一个或者多个参数;

通过JIT编译利用替代所述一个或者多个参数的常数值编译所述参数化IR,以生成用于执行所述传入查询的机器代码。

3.根据权利要求2所述的方法,其特征在于,还包括:将所述参数化IR的唯一标识符ID保存在所述目录中,其中所述参数化IR是利用所述参数化IR的唯一标识符ID从所述目录加载。

4.根据权利要求2所述的方法,其特征在于,还包括:将所述编译的参数化IR的指针保存在用于通过所述机器代码执行所述传入查询的运行时间数据结构中。

5.根据权利要求1至4任一项所述的方法,其特征在于,在运行时间利用所述变量的值执行传入查询之前,将所述CPU密集函数编译成所述参数化IR。

6.根据权利要求1至4任一项所述的方法,其特征在于,所述识别所述CPU密集函数包括:通过CPU性能分析识别受益于数据库查询执行引擎中JIT编译的CPU密集函数;所述CPU性能分析包括:利用函数追踪之前运行的查询中的CPU性能,以确定随着时间的推移CPU利用率的百分比和/或CPU资源或者性能指标,并且将追踪到的函数的所述CPU性能与未追踪到的函数或者预定阈值的CPU性能进行比较,其中,高于所述CPU利用率的平均百分比或者资源的函数被归类为所述CPU密集函数。

7.根据权利要求1至4任一项所述的方法,其特征在于,所述在所述CPU密集函数中识别一个或者多个参数包括:利用数据库模式信息、表达式或者与查询中所述CPU密集函数相关的数据类型进行程序分析,以确定所述变量是否不变。

8.根据权利要求1至4任一项所述的方法,其特征在于,所述在所述CPU密集函数中识别一个或者多个参数包括:选择期望提供包括不可达代码去除、循环展开、常量折叠和传播、内联的虚函数调用或者通过函数指针进行调用等更多编译器优化的变量。

9.根据权利要求1至4任一项所述的方法,其特征在于,所述将所述CPU密集函数编译成所述参数化IR包括:在所述CPU密集函数中同时编译多个指令。

10.根据权利要求1至4任一项所述的方法,其特征在于,所述将所述CPU密集函数编译成参数化IR包括:

利用查询的CPU性能分析信息识别查询中的所述CPU密集函数;

利用程序分析、数据库模式信息、表达式或者与所述CPU密集函数相关的数据类型识别所述参数化IR的代表在不同查询实例中可以变化的值的变量的一个或者多个参数。

11.一种查询即时JIT编译和执行的装置,其特征在于,包括:

至少一个处理器；

一种非瞬时性计算机可读取存储介质，用于存储供所述至少一个处理器执行的程序，其中，所述程序包括指令，用于：

识别查询中的中央处理器CPU密集函数；

在所述CPU密集函数中识别一个或者多个参数，其中所述一个或者多个参数代表在不同查询实例中可以变化的值的变量；

将所述CPU密集函数编译成包括所述一个或者多个参数的参数化中间表示IR；其中，所述一个或者多个参数用于在执行传入查询时被所述变量的常数值替代；

将所述CPU密集函数的参数化IR保存在参数化IR的目录中。

12.根据权利要求11所述的装置，其特征在于，所述程序还包括指令，用于：

在准备执行传入查询时，从所述目录加载所述参数化IR；

在所述参数化IR中利用所述变量的常数值替代所述一个或者多个参数；

通过即时JIT编译利用替代所述一个或者多个参数的常数值编译所述参数化IR，以生成用于执行所述传入查询的机器代码。

13.根据权利要求12所述的装置，其特征在于，所述程序还包括指令用于：将所述参数化IR的唯一标识符ID保存在所述目录中，其中所述程序还包括指令，用于利用所述参数化IR的唯一标识符ID从所述目录加载所述参数化IR。

14.根据权利要求12所述的装置，所述程序还包括指令用于：将所述编译的参数化IR的指针保存在用于通过所述机器代码执行所述传入查询的运行时间数据结构中。

15.根据权利要求11至14任一项所述的装置，其特征在于，所述程序还包括指令用于：在运行时间利用所述变量的值执行传入查询之前，将所述CPU密集函数编译成所述参数化IR。

16.根据权利要求11至14任一项所述的装置，其特征在于，所述用于识别所述CPU密集函数的指令包括用于通过CPU性能分析识别受益于数据库查询执行引擎中JIT编译的CPU密集函数的指令；所述CPU性能分析包括：利用函数追踪之前运行的查询中的CPU性能，以确定随着时间的推移CPU利用率的百分比和/或CPU资源或者性能指标，并且将追踪到的函数的所述CPU性能与未追踪到的函数或者预定阈值的CPU性能进行比较，其中，高于所述CPU利用率的平均百分比或者资源的函数被归类为所述CPU密集函数。

17.根据权利要求11至14任一项所述的装置，其特征在于，所述用于在所述CPU密集函数中识别一个或者多个参数的指令包括用于利用数据库模式信息、表达式或者与查询中所述CPU密集函数相关的数据类型进行程序分析，以确定所述变量是否不变的指令。

18.根据权利要求11至14任一项所述的装置，其特征在于，所述用于在所述CPU密集函数中识别一个或者多个参数的指令包括用于利用查询的CPU性能分析信息识别查询中的所述CPU密集函数；利用程序分析、数据库模式信息、表达式或者与所述CPU密集函数相关的数据类型识别所述参数化IR的代表在不同查询实例中可以变化的值的变量的一个或者多个参数。

查询即时JIT编译和执行的装置和方法

[0001] 相关申请案交叉申请

[0002] 本申请要求于2015年4月15日递交的发明名称为“在数据库查询执行引擎中利用参数化中间表示进行即时编译的装置和方法”申请号为14/687,473的美国非临时专利申请案的权益和在先申请优先权,该在先申请的内容以引入的方式并入本文。

技术领域

[0003] 本发明涉及数据库处理,在特定实施例中,涉及在数据库查询执行引擎中利用参数化中间表示进行即时编译的装置和方法。

背景技术

[0004] 在现代数据系统中,例如通过采用固态磁盘(solid state drive,简称SSD)设备很大程度上避免了慢磁盘访问时,查询执行的中央处理器(central processing unit,简称CPU)的成本变得越来越关键。即时(Just-in-time,简称JIT)编译是用于提高数据库系统中CPU性能的方法。JIT编译指的是在程序执行的运行时间而不是执行之前进行的查询执行中的编译方案。因为在代码优化、本地代码生成以及编译后的代码的质量方面的有效性,低级虚拟机(low level virtual machine,简称LLVM)编译器的结构对于JIT编译是很好的候选。所述LLVM包括用于在运行时间生成特定查询函数的中间表示(intermediate representation,简称IR)的组合函数(称作IRBuilder)。相比编译后的特定查询函数,所述LLVM可以更加高效地从所述IR生成优化的机器代码。但是LLVM IR在运行时间通过LLVM IRBuilder生成代码在时间和/或计算资源例如内存等方面代价较高且容易出错。或者可以利用LLVM提供的clang/clang++等工具将C/C++代码编译成LLVM IR。如果所述C/C++代码不是专门用于传入查询,该方法可能不会受益于JIT编译。因此需要一种更加高效地为查询的JIT编译生成IR的方案。

发明内容

[0005] 根据一实施例,一种支持在数据库管理系统中查询即时(just-in-time,简称JIT)编译和执行的方法包括:识别查询中的中央处理器(central processing unit,简称CPU)密集函数,并且在所述CPU密集函数中识别一个或者多个参数。所述一个或者多个参数代表在不同查询实例中可以变化的值的变量。将所述CPU密集函数编译成包括所述一个或者多个参数的参数化中间表示(intermediate representation,简称IR)。将所述CPU密集函数的参数化IR保存在参数化IR的目录中。

[0006] 根据另一实施例,一种支持在数据库管理系统中查询JIT编译和执行的方法包括:将CPU密集函数编译成包括一个或者多个参数的参数化IR。所述一个或者多个参数代表在不同的查询实例中可以变化的的值的变量。所述方法还包括:将所述CPU密集函数的参数化IR保存在参数化IR的目录中;并且在准备执行传入查询时,从目录加载参数化IR。在所述参数化IR中利用所述传入查询的变量的常数值替代所述一个或者多个参数;通过所述JIT编

译利用替代所述一个或者多个参数的常数值编译所述参数化IR,以生成执行所述传入查询的机器代码。

[0007] 根据又一实施例,一种用于数据库查询执行引擎的装置包括:至少一个处理器以及用于存储供所述至少一个处理器执行的程序的非瞬时性计算机可读取存储介质。所述程序包括指令,用于识别查询中CPU密集函数,并且在所述CPU密集函数中识别代表在不同查询实例中可以变化的值的变量的一个或者多个参数。所述程序还包括指令,用于将所述CPU密集函数编译成包括所述一个或者多个参数的参数化IR,并且将所述CPU密集函数的参数化IR保存在参数化IR的目录中。

[0008] 上述宽泛地概括了本发明实施例的特征,以便能够更好理解以下本发明详细描述。以下将描述本发明实施例其他的特征和优势,其构成本发明权利要求书的主体。本领域的技术人员应当理解,所公开的概念和特定实施例易被用作修改或设计其他实现与本发明相同目的的结构或过程的基础。本领域的技术人员还应当意识到,这种等同构造不脱离所附权利要求书所阐述的本发明的精神和范围。

附图说明

[0009] 为了更完整地理解本发明及其优点,现在参考下文结合附图进行的描述,其中:

[0010] 图1示出一种在数据库管理系统中进行查询处理的框架;

[0011] 图2示出针对查询函数生成参数化IR的方法实施例;

[0012] 图3示出对查询函数的参数化IR进行JIT编译的方法实施例;

[0013] 图4是一种能够用于实现不同实施例的处理系统的示意图。

[0014] 除非另有指示,否则不同图中的对应标号和符号通常指代对应部分。绘制各图是为了清楚地说明实施例的相关方面,因此未必是按比例绘制的。

具体实施方式

[0015] 下文将详细论述当前优选实施例的制作和使用。然而,应了解,本发明提供可在各种具体上下文中体现的许多适用的发明性概念。所论述的具体实施例仅仅说明用以实施和使用本发明的具体方式,而不限制本发明的范围。

[0016] 本文所提供的系统和方法实施例用于将参数化中间表示(intermediate representation,简称IR)用于数据库查询执行引擎中的即时(just-in-time,简称JIT)编译。具体地,对于规定的查询函数,例如由所述数据库执行引擎处理的处理器(或者CPU)密集函数,识别所述函数中针对特殊查询可能不变(固定)的变量。所述变量在不同的查询实例中有可以变化的值,因此可以用作查询的参数。所述CPU密集函数可以是要求更多处理资源例如时间、内存或者其他处理资源的函数。所述CPU密集函数可以通过CPU性能分析识别。所述识别的变量可以包括模式信息、表达式或者与所述变量相关的数据类型等。识别为不变量的变量可以提供更多的编译器优化,例如不可达代码去除、循环展开、常量折叠和传播、内联的虚函数调用或者通过函数指针进行调用。所述识别的变量均设置在CPU密集函数的IR模板中用作所述IR的参数。在利用IR编译器例如LLVM JIT编译器等进行JIT编译时,利用所述查询特殊信息的常数值替代所述IR中的参数,以生成用于执行的优化的机器代码。

[0017] 这里称作参数化IR的带有参数的IR模板是从初始函数(翻译码)静态编译,并且所

述IR可以与唯一ID和参数名一起保存在目录表中。所述参数化IR在准备执行特殊查询的运行时间加载，因此利用常数值替代所述IR中的参数以获得修改后的IR。然后，对所述修改后的IR进行JIT编译，以生成用于所述函数的本地机器代码。利用查询函数中的一组指令同时生成和编译参数化IR可以避免在运行时间为查询特殊IR函数代价较高且容易出错的逐条生成指令。通过保存参数化IR函数的通用版本，在运行时间加载所述IR，在向所述参数化IR添加查询特殊信息之后，对所述IR函数进程JIT编译，不需要在运行时间针对每个JIT编译生成C/C++代码。相比针对查询生成C/C++代码并且利用C/C++编译器对查询进行JIT编译，生成和编译参数化IR的方案更快。

[0018] 图1示出一种在数据库管理系统中进行查询处理的框架100的示例。首先由解析函数110对SQL查询进行解析，然后由改写函数120以适合执行的形式相应改写。可以由计划函数130对改写查询进行分析，以确定其执行参数或设置，然后生成查询计划。执行函数140在本文也称作查询执行引擎，其用于执行改写查询计划。所述执行函数140包括以下三个步骤：准备执行器步骤141：分配运行时间数据结构，并且准备查询执行；运行执行器步骤142：以迭代的方式执行所述查询；结束执行器步骤143：释放运行时间数据结构，并且结束所述查询执行。上述函数可以通过在CPU或者合适的处理器上运行的软件实现。

[0019] 在一实施例中，所述执行器140(所述数据库查询执行引擎)利用参数化IR实现JIT编译。图2示出针对查询函数生成参数化IR的方法200的流程图。所述方法包括：在查询执行引擎启动之前，利用参数(参数化IR)识别并准备IR模板。通过CPU性能分析(步骤210)识别可能受益于数据库查询执行引擎中JIT编译的CPU密集函数(步骤211)。CPU性能分析包括：利用函数追踪之前运行的查询中的CPU性能，以确定随着时间的推移CPU利用率的百分比和/或其他CPU资源或者性能指标等；并且将追踪到的所述函数的CPU性能与其他函数或者预定阈值的CPU性能进行比较。相应地，高于CPU利用率的平均百分比或者资源的函数被归类为CPU密集函数。在利用数据库模式信息对每个CPU密集函数进行程序分析之后(步骤220)，就可以识别所述函数中变量的参数(步骤221)。每个参数的值对于特定的传入查询是不变的，在IR中利用常数值替代所述参数期望能有益于JIT编译。然后，将所述函数静态编译成IR(步骤230)，并且将所述参数信息添加至所述编译后的IR，以获得参数化IR，这些参数化IR均保存在目录表中(步骤240)。所述表格也包括每个参数化IR的唯一标识符(identifier，简称ID)，如下面示例性的表1所示。

[0020] 表1：用于保存参数化IR的IR目录表

ID(整数)	姓名(文本类型)	IR(长度可变的二进制数据类型)	Parm_num (int)	P1 (Varchar)	P2 (Varchar)	P3 (Varchar)
1	Sdt_loop	……	1	NATTS		
2	Comparetup_index_btree_IR	……	3	NKEYS	ISNULL	KEYTYPE
……	……	……	……	……	……	……

[0021] 所述唯一ID可以有匹配其在表格中的IR的枚举数据类型，并且用于在后期查询执行引擎中(在执行运行时间)检索所述IR。所述枚举数据类型可以定义如下：

[0022]
```
enum IRFunctionsID {
```

[0023]
```
    SDT_LOOP_IR = 1,
```

[0024]
```
    COMPARETUP_INDEX_BTREE_IR = 2,
```

[0025] …

[0026] }.

[0027] 图3示出通过利用参数化IR的JIT编译方法300进行的查询进程。由于所述参数化IR是在查询执行之前准备，数据库查询执行引擎(执行器140)可以在准备执行器步骤141基于唯一ID从参数化IR的目录表加载需要的参数化IR(步骤351)，并将特定查询信息添加至所述参数化IR中(步骤352)。然后所述执行引擎可以对修改后的IR进行JIT编译，以生成优化的机器代码(步骤353)，并将JIT编译后的函数的指针保存在运行时间数据结构中(步骤354)。在运行执行器步骤142的执行查询时，所述执行引擎检查所述JIT编译后的函数是否存在(步骤361)，以确定调用这个JIT编译后的函数(步骤362)还是调用初始的解释函数(步骤363)。

[0028] 下面(C语言中的)代码示出从PostgreSQL提取出来的sdt_loop函数的示例：

[0029] sdt_loop (int natts, datum *values)

[0030] {

[0031] int i;

[0032] for (i=0; i < natts; i++) {

[0033] values[i] = …

[0034] }

[0035] }.

[0036] 在本示例中，所述sdt_loop函数被简化并且从PostgreSQL的查询执行引擎提取。所述sdt_loop函数用于从内存元组中提取列值。变量natts的值针对查询中的特定关系是不变的，该值是需要从元组中提取的列值的数量。基于CPU性能分析和程序分析，所述sdt_loop函数识别为所述数据库查询执行引擎中JIT编译的候选。所述函数中的变量natts也识别为针对查询中特定关系的编译后的IR中的参数，在查询执行时natts的值保持不变。将所述函数静态编译成增加了参数natts的IR，如下所示：

[0037] define void @sdt_loop(i32 %natts, i64* %values) {

[0038] entry:

[0039] %cmp4 = icmp sgt i32 %natts, 0

[0040] br i1 %cmp4, label %for.body, label %for.end

[0041] for.body:

[0042] %indvars.iv = phi i64 [%indvars.iv.next, %for.body], [0, %entry]

[0043] %arrayidx = getelementptr inbounds i64* %values, i64 %indvars.iv

[0044] %rhs_val = ... ; compute the value

[0045] store i64 %rhs_val, i64* %arrayidx, align 8

[0046] %indvars.iv.next = add i64 %indvars.iv, 1

[0047] %lftr.wideiv = trunc i64 %indvars.iv.next to i32

[0048] %exitcond = icmp eq i32 %lftr.wideiv, %natts

[0049] br i1 %exitcond, label %for.end, label %for.body

[0050] for.end:

[0051] ret void

[0052] }.

[0053] 上述参数化IR保存在为PostgreSQL的查询执行引擎中的JIT编译加载的目录表中。在一种场景下,接收查询,其中natts等于3,如下所示:

```
[0054] sdt_loop (int natts, datum *values)
[0055] {
[0056]     int i;
[0057]     for (i=0; i < 3; i++) {
[0058]         values[i] = …
[0059]     }
[0060] };
```

[0061] 从T中选择计数(*),其中C3>0;

[0062] 在上述示例中,C3是表格T中的第三列。

[0063] 因此所述参数化IR从目录表加载,在IR中利用常数3替代所述参数natts的引用,如下所示:

```
[0064] define void @sdt_loop(i32 %natts, i64* %values) {
[0065] entry:
[0066] %cmp4 = icmp sgt i32 3, 0
[0067] br i1 %cmp4, label %for.body, label %for.end
[0068] for.body:
[0069] %indvars.iv = phi i64 [ %indvars.iv.next, %for.body ], [ 0, %entry ]
[0070] %arrayidx = getelementptr inbounds i64* %values, i64 %indvars.iv
[0071] %rhs_val = ... ; compute the value
[0072] store i64 %rhs_val, i64* %arrayidx, align 8
[0073] %indvars.iv.next = add i64 %indvars.iv, 1
[0074] %lftr.wideiv = trunc i64 %indvars.iv.next to i32
[0075] %exitcond = icmp eq i32 %lftr.wideiv, 3
[0076] br i1 %exitcond, label %for.end, label %for.body
[0077] for.end:
[0078] ret void
[0079] }.
```

[0080] 上述LLVM IR等效于下面的C代码:

```
[0081] sdt_loop (int natts, datum *values)
[0082] {
[0083] int i;
[0084] for (i=0; i < 3; i++) {
[0085] values[i] = …
[0086] }
[0087] }.
```

[0088] 然后对所述函数进行JIT编译,并且生成用于执行该查询的本地机器代码。利用生

成的优化代码对所述sdt_loop函数进行JIT编译变成：

[0089] define void @sdt_loop(i32 %natts, i64* %values) {

[0090] entry:

[0091] %rhs_val.0 = ... ; compute the first value

[0092] store i64 %rhs_val.0, i64* %values, align 8

[0093] %arrayidx.1 = getelementptr inbounds i64* %values, i64 1

[0094] %rhs_val.1 = ... ; compute the second value

[0095] store i64 %rhs_val.1, i64* %arrayidx.1, align 8

[0096] %arrayidx.2 = getelementptr inbounds i64* %values, i64 2

[0097] %rhs_val.2 = ... ; compute the third value

[0098] store i64 %rhs_val.2, i64* %arrayidx.2, align 8

[0099] ret void

[0100] }.

[0101] 等效的C代码是：

[0102] sdt_loop (int natts, datum *values)

[0103] {

[0104] values[0] = …

[0105] values[1] = …

[0106] values[2] = …

[0107] }.

[0108] 不失一般性，上述方法可以应用于数据库查询执行引擎中的其他CPU密集函数，例如哈希连接、按组排序或者合并、建立索引或其它。在该方法中，程序分析可以用于识别JIT编译的候选函数中的参数。如果变量的值对于特定查询(查询特定信息)是不变的，并且用常数替代所述变量期望引进针对所述函数更多的编译器优化，例如不可达代码去除、循环展开、常量折叠和传播、内联的虚函数调用或者通过函数指针进行调用，则该变量是用于JIT编译的函数的IR中的参数。查询特定信息可以从查询中关系的模式信息(例如表格、视图或者索引)中识别，或者从查询中表达式和数据类型中识别。例如，在上述sdt_loop函数中，在基于查询中的模式信息和访问列执行之前，可以针对特定查询确定变量natts的值。例如，NOT NULL等部分模式信息可以有利于去除不可达代码(查询中NOT NULL列的NULL值检查是多余的)。(模式或者查询本身的)数据类型信息可以有利于为函数指针或者虚拟函数解决函数。

[0109] 在一实施例中，可以实现两种方法中的任意一种以将查询特定信息(常数值)添加至参数化IR。如上述sdt_loop函数的示例所示，第一种方法包括利用与特定查询相关的常数值替代每个参数的引用。从目录表中加载IR时，也可以获得参数的数量以及这些参数在IR中的姓名。解析所述IR，以利用相关常数值替代任意指令中参数的引用。第二种方法包括：在函数的开始部分，针对每个参数插入赋值语句，以下以sdt_loop函数作为示例：

[0110] sdt_loop (int natts, datum *values)

[0111] {

[0112] int i;

[0113] natts = 3;

[0114] for (i=0; i < natts; i++) {

[0115] values[i] = …

[0116] }

[0117] }.

[0118] 然后JIT编译器利用常数值替代所述参数的引用。所述LLVM IR是静态单一指派形式(static single assignment form,简称SSA形式),并且在LLVM IR中没有分配指令。同样地,所述加法指令可以用于将所述常数加0作为分配给参数变量的常数。生成的代码变成:

[0119] define void @sdt_loop(i32 %natts1, i64* %values) {

[0120] entry:

[0121] %natts = add i32 0, 3 ;assign the value 3 to natts

[0122] %cmp4 = icmp sgt i32 %natts, 0

[0123] br i1 %cmp4, label %for.body, label %for.end

[0124] for.body:

[0125] %indvars.iv = phi i64 [%indvars.iv.next, %for.body], [0, %entry]

[0126] %arrayidx = getelementptr inbounds i64* %values, i64 %indvars.iv

[0127] %rhs_val = ... ; compute the value

[0128] store i64 %rhs_val, i64* %arrayidx, align 8

[0129] %indvars.iv.next = add i64 %indvars.iv, 1

[0130] %lftr.wideiv = trunc i64 %indvars.iv.next to i32

[0131] %exitcond = icmp eq i32 %lftr.wideiv, %natts

[0132] br i1 %exitcond, label %for.end, label %for.body

[0133] for.end:

[0134] ret void

[0135] }.

[0136] 所述JIT编译可以应用于函数,以生成优化的机器代码并返回JIT编译后的函数的函数指针。在一种场景下,CPU密集函数可以只包括CPU密集型的且有特定查询信息的相对较少的的代码(例如循环次数相对较少的多次迭代)。但是,相同函数中的其他代码可以不是CPU密集型的或者可以没有特定查询信息。在这种情况下,为了减少JIT编译的成本,CPU密集型的和特定查询部分的代码可以脱离初始函数,并针对此部分构造新函数。然后,对新构造的相对较小的函数而不是初始大函数进行JIT编译。

[0137] 图4为一种可用于实现包括上述方法在内的各种实施例的处理系统400的方框图。所述处理系统400可以是数据库管理系统或者数据库查询执行引擎的一部分。特定设备可利用所有所示的部件或所述部件的仅一子集,且设备之间的集成程度可能不同。此外,设备可以包括部件的多个实例,例如多个处理单元、处理器、存储器、发射器、接收器等。所述处理系统400可以包括配备一个或者多个输入/输出设备,例如扬声器、麦克风、鼠标、触摸屏、按键、键盘、打印机、显示器等的处理单元401。所述处理单元401可以包括连接到总线的中央处理器(central processing unit,简称CPU)410、存储器420、大容量存储设备430、视频

适配器440以及I/O接口460。所述总线可以为任何类型的几个总线架构中的一个或多个，包括存储总线或者存储控制器、外设总线或者视频总线等等。

[0138] 所述CPU 410可以包括任意类型的电子数据处理器。所述存储器420可以包括任意类型的系统存储器，例如静态随机存取存储器(static random access memory，简称SRAM)、动态随机存取存储器(dynamic random access memory，简称DRAM)、同步DRAM(synchronous dynamic random access memory，简称SDRAM)、只读存储器(read-only memory，简称ROM)或其组合等等。在一实施例中，所述存储器420可以包括开机时使用的ROM以及在执行程序时使用的存储程序和数据的DRAM。在一实施例中，所述存储器420是非瞬时的。所述大容量存储设备430可以包括任意类型的存储设备，其用于存储数据、程序和其它信息，并使这些数据、程序和其它信息可以通过总线访问。所述大容量存储设备430可以包括如下项中的一种或多种：固态磁盘、硬盘驱动器、磁盘驱动器以及光盘驱动器等等。

[0139] 所述视频适配器440和所述I/O接口460提供接口，以将外部输入和输出设备耦合到所述处理单元。如图所示，输入输出设备的示例包括耦合至所述视频适配器440的显示器490和耦合至所述I/O接口460的鼠标/键盘/打印机470的任意组合。其它设备可以耦合至所述处理单元401，并且可以利用附加的或更少的接口卡。例如，串行接口卡(未图示)可以用于为打印机提供串行接口。

[0140] 所述处理单元401还包括一个或多个网络接口450，所述网络接口450可以包括以太网电缆等有线链路，和/或到接入节点或者一个或多个网络480的无线链路。所述网络接口450允许所述处理单元401通过所述网络480与远端单元通信。例如，所述网络接口450可以通过一个或多个发射器/发射天线以及一个或多个接收器/接收天线提供无线通信。在一实施例中，所述处理单元401耦合到局域网或广域网上用于数据处理以及与远端设备通信，其中所述远端设备是其它处理单元、因特网、远端存储设施等等。

[0141] 在一示例性的实施例中，数据库管理系统包括：CPU识别模块，用于识别查询中的中央处理器(central processing unit，简称CPU)密集函数；参数识别模块，用于在所述CPU密集函数中识别一个或者多个参数，其中所述一个或者多个参数代表在不同查询实例中可以变化的值的变量；编译模块，用于将所述CPU密集函数编译成包括所述一个或者多个参数的参数化中间表示(intermediate representation，简称IR)；以及保存模块，用于将所述CPU密集函数的参数化IR保存在参数化IR的目录中。在部分实施例中，所述数据库管理系统可以包括其他或者附加模块用于执行所述实施例中描述的任一或组合的步骤。

[0142] 在一示例性的实施例中，数据库管理系统包括：CPU编译模块，用于将中央处理器(central processing unit，简称CPU)密集函数编译成包括一个或者多个参数的参数化中间表示(intermediate representation，简称IR)，其中所述一个或者多个参数代表在不同的查询实例中可以变化的的值的变量；保存模块，用于将所述CPU密集函数的参数化IR保存在参数化IR的目录中；加载模块，用于在准备执行传入查询时，从所述目录加载参数化IR；替代模块，用于在所述参数化IR中利用所述传入查询的变量的常数值替代所述一个或者多个参数；参数化IR编译模块，用于通过所述JIT编译利用替代所述一个或者多个参数的常数值编译所述参数化IR，以生成用于执行所述传入查询的机器代码。在部分实施例中，所述数据库管理系统可以包括其他或者附加模块用于执行所述实施例中描述的任一或组合的步骤。

[0143] 虽然本发明中已提供若干实施例，但应理解，在不脱离本发明的精神或范围的情况下，本发明所公开的系统和方法可以以许多其他特定形式来体现。本发明的实例应被视为说明性而非限制性的，且本发明并不限于本文本所给出的细节。例如，各种元件或部件可以在另一系统中组合或合并，或者某些特征可以省略或不实施。

[0144] 此外，在不脱离本发明的范围的情况下，各种实施例中描述和说明为离散或单独的技术、系统、子系统和方法可以与其它系统、模块、技术或方法进行组合或合并。展示或论述为彼此耦合或直接耦合或通信的其它项也可以采用电方式、机械方式或其它方式通过某一接口、设备或中间部件间接地耦合或通信。其他变化、替代和改变的示例可以由本领域的技术人员在不脱离本文精神和所公开的范围的情况下确定。

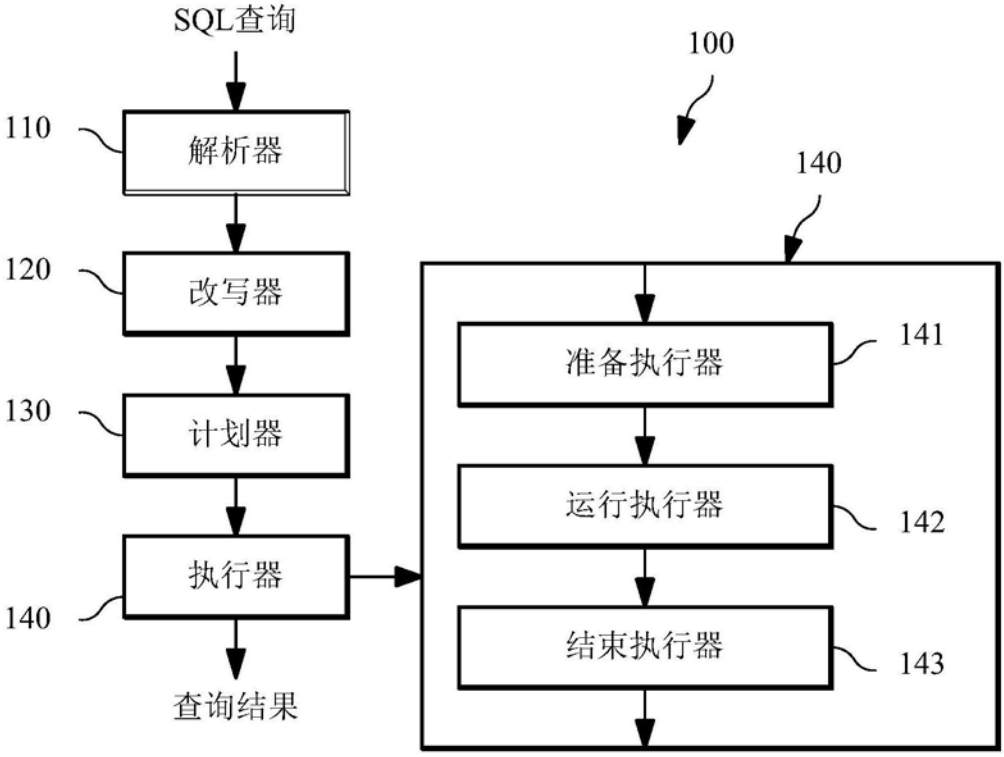

SQL查询
110
解析器
120
改写器
130
计划器
140
执行器
查询结果
100
140
准备执行器
141
运行执行器
142
结束执行器
143

图1

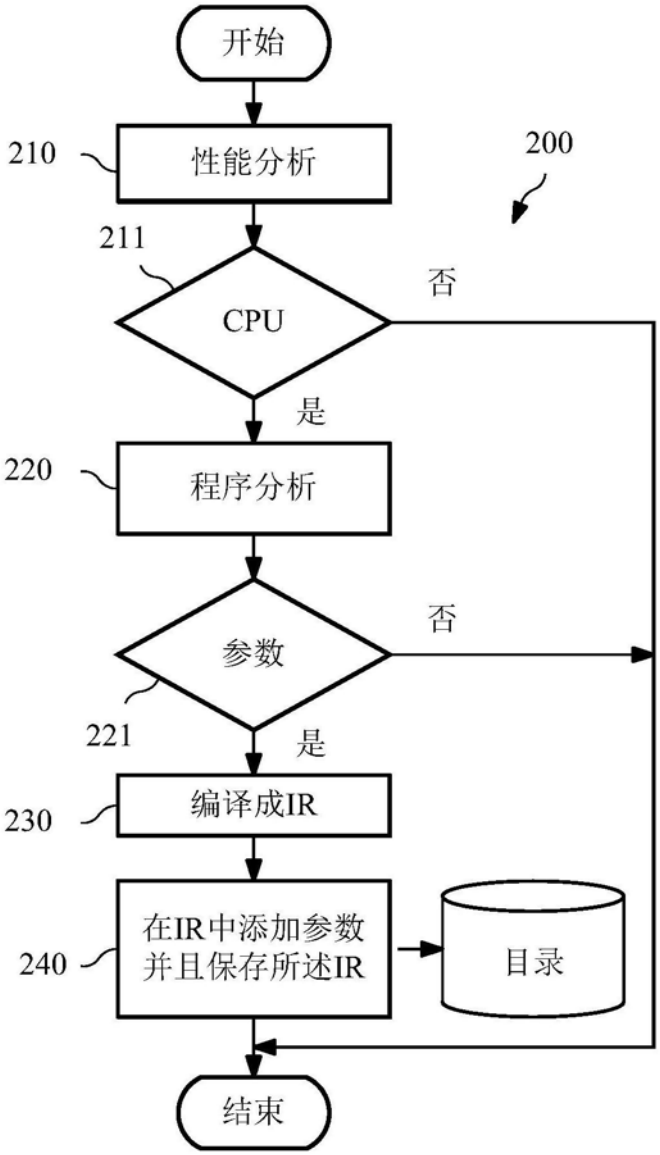

开始
210
性能分析
200
211
CPU
否
是
220
程序分析
参数
否
221
是
230
编译成IR
在IR中添加参数
并且保存所述IR
240
目录
结束

图2

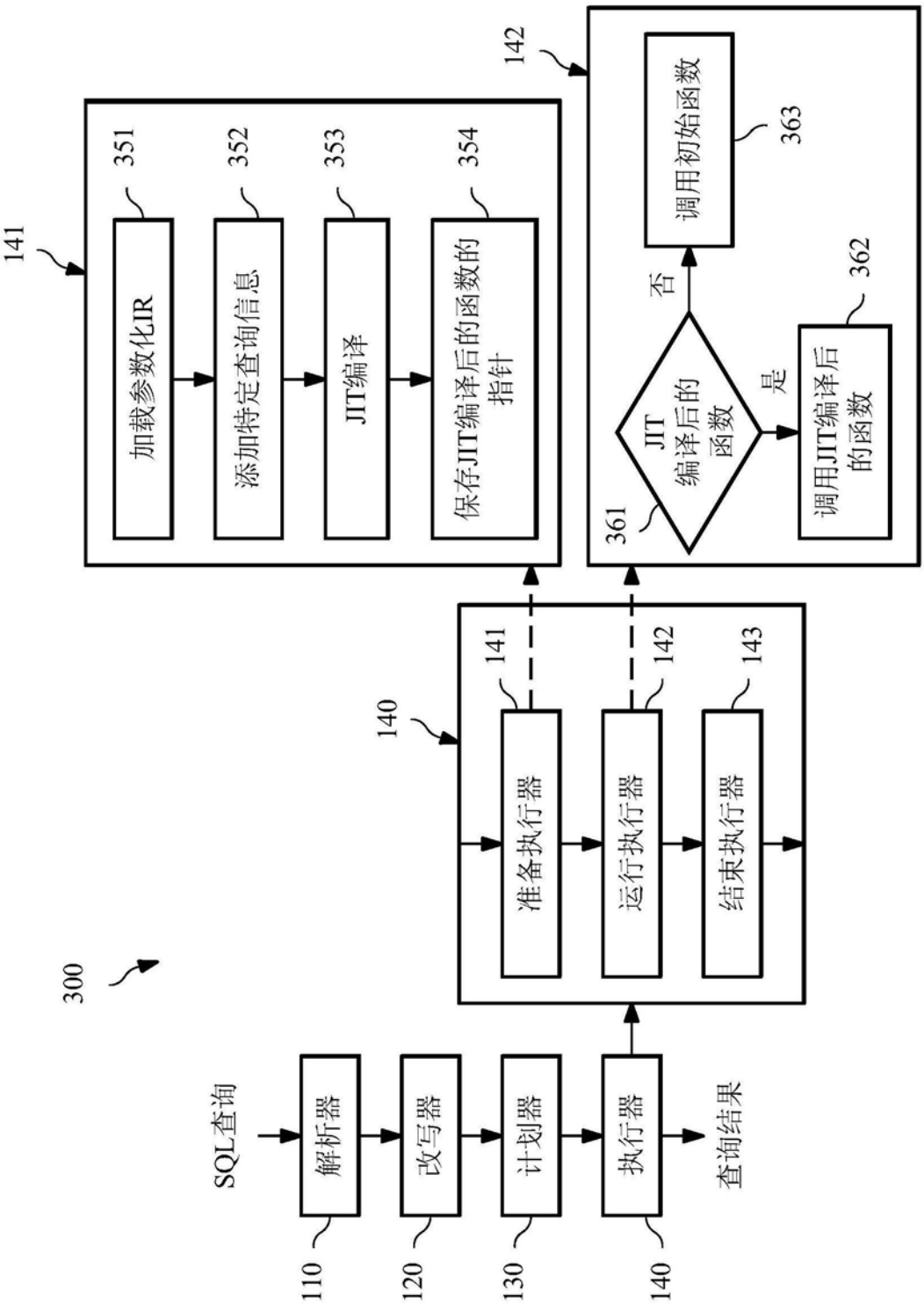

300
SQL查询
解析器
改写器
计划器
执行器
查询结果
110
120
130
140
140
准备执行器
运行执行器
结束执行器
141
142
143
141
加载参数化IR
添加特定查询信息
JIT编译
保存JIT编译后的函数的指针
351
352
353
354
142
JIT编译后的函数
361
否
是
调用初始函数
363
调用JIT编译后的函数
362

图3

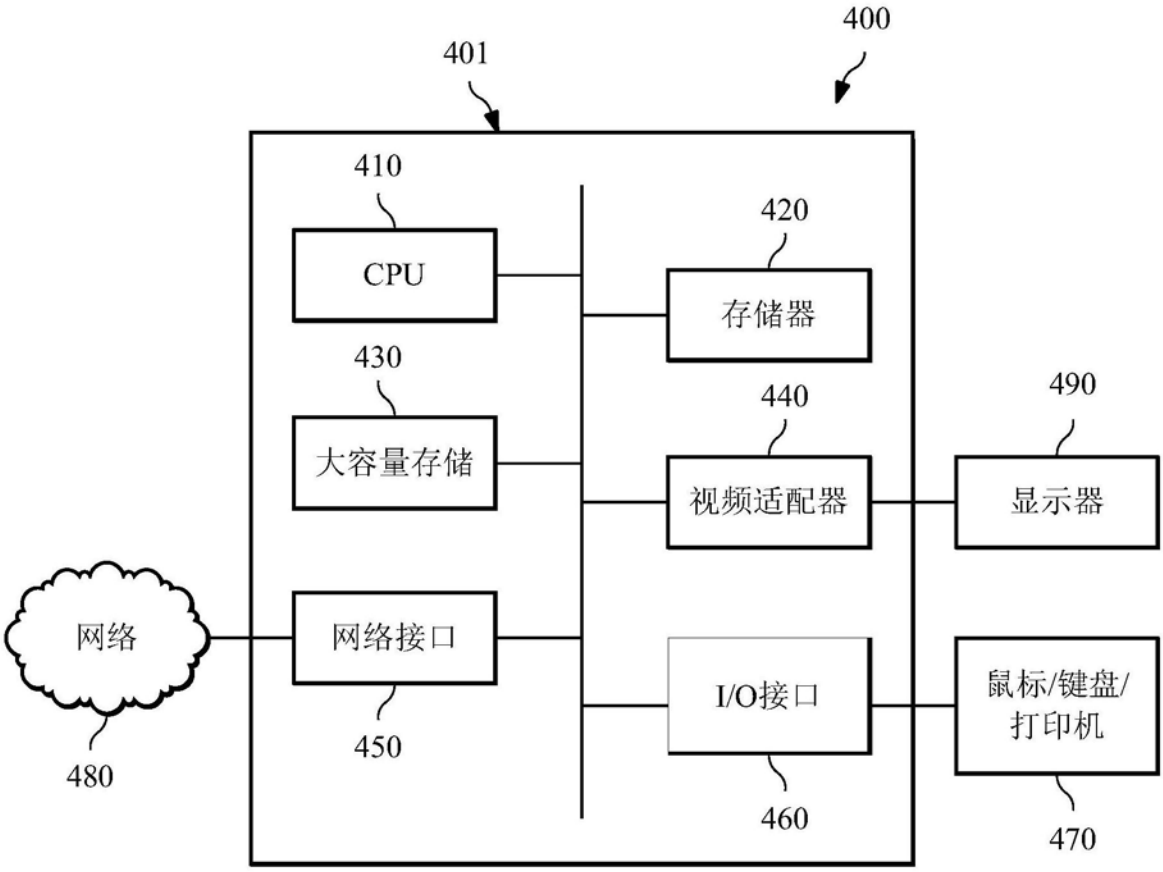

400
401
410
CPU
420
存储器
430
大容量存储
440
视频适配器
490
显示器
网络
网络接口
I/O接口
鼠标/键盘/
打印机
480
450
460
470

图4