Conservative Parallel Detection Algorithm for OpenACC

吴昊泽 · 张煜皓 · 陈牧歌

December 24, 2016

1 选题概述

OpenACC 指令在加速科学代码方面是一种简单而又可移植的方式。利用 OpenACC，只要在自己的 Fortran 或 C 语言代码中插入编译器提示，编译器即可将代码中计算量繁重的部分自动交由 GPU 处理，以实现更高的性能。

OpenACC 指令最大的长处就是简单，只需要插入一句编译器的提示，就可以将代码中的某个语句块进行并行优化。正因如此，我们小组选择的课题就是：针对 C 代码中的大量循环，通过对程序的静态分析，以检测哪些循环能够并行。

2 问题抽象

2.1 工具功能

由于 OpenACC 的指令较多，我们小组无法对其进行系统的分析，故我们选取了最有代表，也最关键的

```
#pragma acc parallel loop
```

这条指令进行分析。其指令的作用就是将下一条语句进行并行处理。

我们工具达到的效果就是分析给定的一段代码后，自动在代码中一定可以并行的地方加入指令；如果无法判断一处是否可以并行，也会输出相应的注释，去帮助 OpenACC 的使用者更佳方便地优化自己的程序。

2.2 工具情况

我们的工具能够检测大部分情况下的并行情况，但也有部分情况受制于小组水平与静态分析本身的缺陷，无法检测是否并行。在接下来的分析中，可以看到我们的工具是有安全性的保证的。而准确性的不足，也会在后面的部分详细给出。

2.3 简化 C 语言

我们的代码分析都是在简化之后的 C 语言上进行的。之所以需要简化 C 语言，是因为由于程序静态分析的特殊需要，我们小组写了个针对 C 语言的解释器，还因为这可以简化

我们的工具的制作。但值得注意的是，针对其中的每一条简化，我们都有相应的原因作为支撑，以表明这是一个合理的简化。

Table 1: 简化 C 语言

简化	原因
不支持任何形式的&运算	为了提高对于数组分析的精度，不得不对指针做出一点限制
所有的变量定义要放在一个代码块的开头	早期的 C 语言标准就是如此，并且这可以简化解释器的编写
忽略#define语句	这可以简化解释器的编写
不允许调用函数	暂时做的是过程内的分析，但可特殊处理 min,max 等简单函数
不允许一个语句中修改多个变量	算法上支持，目的仅为简化解释器的编写

3 算法安全性证明

3.1 算法简述

首先，将整个程序 parser 成一个 CFG 图，之后在这个 CFG 上做程序流分析：逆向做一遍程序流分析，分析在该语句之后，有哪些变量是活跃变量；正向做一遍程序流分析，得到每个 for 循环与表达式的摘要，具体摘要的定义将在接下来给出。对于每个循环，我们通过对这两次分析的结果分析其能否被并行。

- 活跃变量分析：

  1) 逆向分析
  2) 半格元素：一个集合，这个操作之前，哪些变量是活跃的。
  3) 交汇操作：集合的并集
  4) 变换函数

  var 被修改 $KILL = \{var\}, GEN = \varnothing$
  var 被使用 $KILL = \varnothing, GEN = \{var\}$

- 语句摘要分析：

  0) 摘要：5 个集合

  Array_Modify 表示在这个语句中哪些数组被修改，我们除了记录数组之外，还需要记录其被修改的位置 $(i_1, i_2 \ldots, i_d)$，其中 d 表示数组的维数，i_x 为一个表达式。

Array_Use 表示在这个语句中哪些数组被使用，同Array_Modify的记录一样，需要记录数组与其被使用的位置。

Var_Modify 表示在这个语句中有哪些普通变量被修改，如果语句是 if,while,for，那么这里的变量指的是在语句之外的变量，而不考虑内部的局部变量。

Var_Use 表示在这个语句中有哪些普通变量被使用，同Var_Modify的记录一样，对 if,while,for 语句特殊处理。

Var_Reduction 表示在这个语句中有哪些普通变量被修改了，并且记录对它们的操作满足结合律，即可以 reduction，集合元素就是var, op。

1) 正向分析
2) 半格元素：摘要的 5 个集合
3) 交汇操作：摘要集合的并集
4) 变换函数

if,while,for 递归处理，将内部的语句的摘要进行合并，并剔除那些在语句块中的局部变量，作为对这些语句的摘要。

其他普通语句 将相应的变量（数组变量和普通变量）放入对应的集合之中。并且分析每个被改变的变量的操作是否满足结合律，如果是，也放入Var_Reduction集合之中。

- for 循环并行分析：
  对于 for 内部而言，只关注其变量能否并行，变量分为数组变量和普通变量，for 内部的局部变量不纳入考虑。

  数组 对于每个数组 a 而言，我们去该 for 语句的摘要中查找其可能被访问或者修改的位置。

  a) 如果 a 只被修改，则改数组不影响并行性。
  b) 如果 a 只被访问，则改数组不影响并行性。
  c) 如果 a 既被修改，也被访问，那么需要知道修改的位置和访问的位置是否有重合的部分。如果有，则不能并行，反之则可以。在这里可以调用 SMT Solver 求解（详情见 3.3 准确性的不足）。但是在工具中我们只考虑最简单的可以并行的情况：即改数组的所有下标均是单个循环变量构成的表达式，并且访问和修改处的每个下标构成的表达式完全相等的情况（具体而言：如 a[i][j+1]=a[i][j+1]+1，访问的位置为 (i,j+1)，修改的位置也为 (i,j+1)，其中该访问和修改处的每个下标构成的表达式完全相等）。（安全性证明见 3.4 安全性证明）

  普通变量 对于每个变量 var 而言，仅需考虑其在 for 之外的影响。

  a) 如果 var 只被修改，要看在下文中，这个变量是否是活跃的。因为并行的副作用就是导致这个变量在 for 之行结束后的值不确定，如果活跃，则认为这个 for 不能并行。
  b) 如果 var 只被访问，则改变量不影响并行性。

c) 如果 var 既被修改，也被访问，需要从摘要中得到这个变量是否是 reduction 的，如果是，则这个变量不影响并行性；如果不是，则不能并行

3.2 安全性证明

3.3 准确性的不足

4 安全性检查

5 小组分工

吴昊泽同学负责 OpenACC 平台的搭建，编写项目工具代码，算法讨论。
张煜皓同学负责 llvm 的借口尝试（失败），编写项目报告及少量代码，算法讨论。
陈牧歌同学负责对最后工具的测试，算法讨论。