第 38 卷 第 3 期
Vol.38 No.3

计 算 机 工 程
Computer Engineering

2012 年 2 月
February 2012

·博士论文· 文章编号：1000—3428(2012)03—0004—03 文献标识码：A 中图分类号：TP314

一种基于增量式实例学习的迭代编译方法

马晓东，李中升，漆锋滨，尉红梅

(江南计算技术研究所，江苏 无锡 214083)

摘 要：为提高编译器的自适应性，以应对复杂的体系结构，提出一个结合迭代编译和机器学习的编译框架。编译器可将在优化空间中搜索到的最佳编译选项信息保存到知识库中，并能从知识库中学习获得适合当前程序的最佳编译选项。实例学习算法具有增量式的特点，可有效利用编译过程中积累的数据。通过避免冗余实例入库以及从库中剔除噪声实例，保证学习的精度与效率。

关键词：迭代编译；机器学习；增量式算法；冗余实例

Iterative Compilation Method Based on Incremental Instance Learning

MA Xiao-dong, LI Zhong-sheng, QI Feng-bin, WEI Hong-mei

(Jiangnan Institute of Computing Technology, Wuxi 214083, China)

【Abstract】For the purpose of making the compiler more adaptive and dealing with complex architecture, a compiler framework is proposed which combines iterative compilation and instance-based learning. On one hand, the compiler can search the optimization space and save the best compiler options into the knowledge library; on the other hand, the compiler can learn from the library to get the best compiler options for the current program. The incremental algorithm can make full use of the accumulated data of the compilation. The algorithms are proposed which can keep the redundant instance out of the knowledge library and filter the noise from the library.

【Key words】iterative compilation; machine learning; incremental algorithm; redundant instance

DOI: 10.3969/j.issn.1000-3428.2012.03.002

1 概述

很多编译优化问题都具有指数级的时间复杂度，属于 NP 问题，在编译器中实现对其精确求解是不现实的。传统的编译器开发人员需要借助一些启发式方法，通过对体系结构和程序特征的静态建模，并结合经验公式的方法获取优化问题的近似解。随着体系结构的日趋复杂，包括多核、众核结构的出现，使得获取有效的模型或者经验公式的任务也更加困难，通过静态分析得到的性能提升也越来越有限。

迭代编译[1-3]是一种有效的应对方法，与反馈式编译[4]相同，在优化时都利用了程序执行时的反馈信息。不同的是，迭代编译采用一定的搜索策略，在优化空间中寻找最优解或者满足条件的较优解。和基于静态模型分析的优化方法相比，迭代编译能够减少对体系结构模型的依赖，能更好地适应于不同的体系结构。

在初始阶段，迭代编译的研究主要是搜索编译优化空间，通过多次运行，测量并评估程序的性能指标，选择空间中的最优或较优点。其性能评估标准可以是运行时间、目标程序的大小或者嵌入式程序的功耗等。

本文给出迭代编译的总体框架、知识库的增量式学习算法，介绍了在向知识库中加入新实例时对于冗余实例的判断，以及对库中噪声实例的剔除。

2 迭代编译的研究工作

当前迭代编译的研究工作主要集中于 2 大类。第 1 类着力于优化空间的搜索。在编译实际的应用程序时，编译器面临庞大甚至是无穷的优化空间。文献[1]研究了数组填充、循环分块和循环展开 3 种优化，把这 3 种优化的参数视为三维空间中的 3 个维，并采用网格搜索算法在空间中搜索。PathScale 编译器中的工具 PathOpt2，可以通过多次的编译执行，在一定的优化空间中选择最好的优化。并且，在迭代的过程中，前期编译执行的结果可被用来指导后继的编译执行，指示搜索方向。在此类方法中，每次搜索往往都是独立的，未能充分利用以前的搜索结果。在同一体系结构下，具有相似特性的程序通常具有相似的最优变换。迭代编译结果的适当利用，可以有效提高编译器的效率。第 2 类的研究主要着力于如何充分利用迭代编译过程中所积累的历史数据，揭示程序的一些特征值和优化变换之间的关系——通常采用机器学习[5]的方法。文献[6]中使用了基于实例学习的迭代编译方法。该方法建立了一个库，库中保存着一些已经优化好的程序，并抽取了程序的一些静态特征，包括程序的代码量、访存操作比例等。在编译新的程序的时候，从库中找到与之最相似的几个程序，并仿照这些程序进行转换。该方法中的库是静态的，若能够在迭代编译的数据积累过程中动态地对库进行更新，则可以进一步提高编译优化的质量。文献[7]中提出了一种元优化方法，利用遗传算法来获取编译器在优化过程中所使用到的经验公式。该方法的学习过程所耗时间较久，经验公式中所支持的表达式的种类也是有限的。

基金项目：“核高基”重大专项“支持国产 CPU 的编译系统及工具链”(2009ZX01036-001-001)

作者简介：马晓东(1979－)，男，博士，主研方向：高性能编译优化技术；李中升，高级工程师；漆锋滨，高级工程师、博士；尉红梅，高级工程师

收稿日期：2011-07-28 **E-mail：**xdma@163.com

3　迭代编译框架

在迭代编译框架中，编译器与若干个知识库相结合——不同的优化变换对应于不同的知识库。知识库中保存了历次迭代编译过程中所形成的有助于编译器做优化选择的信息。编译器具有 3 种工作模式，分别是普通模式、搜索模式和学习模式。用户可以选择任一种模式执行，当知识库中的内容达到一定数量的时候，学习模式能够生成性能较优的代码。

在普通模式下，当编译器判断是否需要进行某种优化变换以及如何选择优化变换的参数时，采用传统的方法，即通过对系统的静态建模以及经验公式的方法决定。

在搜索模式下，当需要选择优化变换及对应参数时，编译器不再使用静态方法计算，而是在优化空间搜索。通过调用适当的搜索策略(穷尽搜索、爬山搜索和网格搜索等)，使用优化空间中的全部或者部分点所对应的信息编译并执行程序，比较它们的性能。以循环展开为例，编译器所需要知道的是在展开多少次的时候程序的性能最优。影响展开次数的因素包括循环体的大小、循环迭代次数的估值等，在本文中，这些特征值被称为实例。迭代编译的搜索结果就是这些特征值所对应的性能最优的展开次数，称为实例的分类。实例及其对应的分类可作为一条新的记录被保存在知识库中。用新记录更新知识库的时候，提出一种冗余记录的鉴别算法，仅当新记录非冗余的时候才被加入到知识库中。通过这样对知识库的动态更新，可以不断增加知识库中所含的信息量，从而提高学习结果的准确度。

在学习模式下，当需要选择优化变换及对应参数时，编译器可以直接从库中学习获得。首先分析程序，从中抽取出影响优化选择的实例。然后从知识库中学习，由知识库中相似实例的已知分类决定当前程序所应该做的优化。在学习过程中采用了 k 近邻法，也即由库中和当前实例最相似的 k 个实例的已知分类，通过投票机制判断当前实例的分类。

另外，知识库中可能有噪声的存在。噪声是数据中有害的异常，不能够正确反映实例及其分类之间的关系，其存在会影响到学习的效率及精度。有 2 个可能产生噪声的原因：(1)受环境的影响，程序执行时，性能的统计出现了偏差；(2)由于体系结构的复杂性，可能存在没有考虑到的特征值或者是难以观测到的特征值，而它们实际上会影响到实例的分类。这些被忽略的特征值所造成的影响是随机的，是噪声的一部分。

为知识库中的每条记录都附加了一些数据信息，根据这些数据信息可以计算出一个噪声值。编译器学习模式和搜索模式的功能流程如图 1 所示。

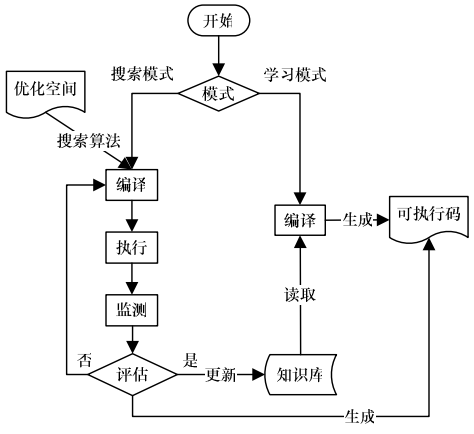

图 1　编译器学习模式和搜索模式的功能流程

在学习模式下，对知识库的更新过程结束之后，有一些记录的噪声信息会被更新。当某一条记录的噪声值超过预先设定的阈值之后，则认为该记录为噪声，把它从知识库中剔除。

4　知识库的增量式学习

在监督式机器学习中，需要一组训练实例。特定的学习算法被应用到这些训练实例上，从中学习出一定的抽象形式，例如决策树或者一系列的分类规则等，以尽可能准确地描述训练实例。当遇到新的实例时，使用学习结果所获得的规则预测新实例的分类。基于实例的学习是监督学习的一种，具有惰性学习的特点。

所谓惰性学习，就是在获得训练实例之后，并不立刻从中学习出分类规则，而是当需要对新的实例做分类的时候，才从实例库中学习，计算出新实例的分类。在一般的机器学习中，训练实例一旦形成，就不再改变。因此，学习的精度对训练实例的依赖较强。

本文使用了一种增量式学习算法，可以用搜索模式的结果更新训练实例所组成的知识库，通过动态改变训练实例库提高学习的精确度。

在搜索模式下更新知识库时，引入了鉴别冗余实例的算法，以保持知识库的精简。无论是鉴别实例，还是从知识库中学习，都用到了 k 近邻法，也即从知识库中查找 k 个与当前实例最相似的实例。假设实例 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别为一个由 n 个特征值组成的向量：$\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$, $\boldsymbol{y}=(y_1,y_2,\cdots,y_n)$，则 $\boldsymbol{x}$、$\boldsymbol{y}$ 可以分别被视为 n 维空间中的一个点，其相似度通过函数 *similarity* 计算，定义为：

$$similarity(\boldsymbol{x},\boldsymbol{y})=-\sqrt{\sum_i^n k_i(x_i-y_i)^2}$$

更新知识库的算法步骤如下：

输入 k: k 最近邻参数

(x, c_x): 用来更新库的记录，实例 x 的分类是 c_x

L: 旧库

T: 精度阈值

输出 updated library of L

```
1. if |L| < k
2.    insert(x, c_x) into L
3. else
4.    for each (y_i, c_yi) in L
5.       d_i= similarity(x, y_i)
6.       try to insert d_i into a stack s consisting of k element
7.    endfor
8. endif
9. using the k-nearest neighbor in s to predicate the classification of x to be c_x'
10. if c_x != c_x'
11.    insert (x, c_x) into L
12. endif
13. for each record (z_i, c_zi) saved in s
14.    increase predicate((z_i, c_zi)) by 1
15.    if c_zi == c_x
16.       increase accuracy ((z_i, c_zi)) by 1
17.    endif
18.    if accuracy((z_i, c_zi))/predicate((z_i, c_zi)) < T
19.       delete (z_i, c_zi) from L
20.    endif
21. endfor
```

权重系数 k_i 表明了向量空间中第 i 维的重要性，也就是第 i 个特征值对优化变换选择的影响。若满足条件：$\forall i$：k_i=1，则每一维上的重要性相同，*similarity*($\boldsymbol{x}$,$\boldsymbol{y}$)类似于向量 $\boldsymbol{x}$、$\boldsymbol{y}$ 在多维空间中的距离。函数取负值，是为了与相似度的说法保持一致：在取负值的情况下，*similarity*($\boldsymbol{x}$,$\boldsymbol{y}$)的值越大，则 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 在多维空间中的距离就越小，$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 之间的相似度就越大。

对知识库的更新操作主要包括 2 个部分：判断新记录是否为冗余的；从知识库中剔出可能的噪声。

一般而言，知识库中的训练实例越多，对其进行机器学习时所得到的结果也就越准确。许多增量式算法不加区别地把新记录加入到知识库中，但是由于机器学习的过程需要对知识库做遍历、比较等操作，庞大的知识库往往会造成较长的学习时间。因此，采用了一种鉴别机制，以避免把冗余的记录添加到知识库中。噪声的存在会损害到知识库的精确度，在本文的算法中，当做完冗余鉴定之后，会产生一些有助于判断噪声的信息，用这些信息更新每一条记录的噪声值，并把可能的噪声剔出。

算法有 4 个输入：

(1) k 近邻法中的 k 值；

(2)搜索模块所产生的新记录(x, c_x)，c_x 是实例 x 的分类；

(3)待更新的知识库 L；

(4)噪声阀值 T。

k 值的选择对算法有较大影响：若 k 值过大，则计算所需的时间代价较高；若 k 值过小，则有可能影响学习的精确度。k = 1 时，算法就退化为最近邻法。

算法的输出是经过更新操作(可能插入新记录或者剔除噪声)的知识库 L。

算法中的第 1 行~第 12 行为鉴别冗余实例的部分。鉴别过程可以分为 2 个步骤：

(1)比较 L 中每条记录中的实例与当前实例 x 的相似度，找出最相似的 k 个近邻。计算过程中采用了一个有 k 个元素的栈。从栈顶到栈底，栈中元素与实例 x 的相似度依次增减小。在遍历知识库时，比较当前被遍历的实例和栈顶元素哪个与 x 更为相似。若栈顶元素更相似，则不做任何操作。否则，继续与栈中的下一个元素作比较。最后把当前被遍历的实例插入到栈中适当的位置，栈底到栈顶的元素仍然保证递减的顺序。同时，原来的栈顶元素自动出栈。

(2)查找到 k 个最近邻之后，利用它们的分类对 x 的分类进行预测。因为 x 的分类已知，可通过比较来判断预测是否准确。若预测准确，即说明实例 x 的分类可以从知识库中现有的实例中推断出来，可以认为新记录是一条冗余记录，不把它加入知识库。若预测不正确，则把新记录插入知识库中。在预测实例 x 的分类时，使用投票机制，也即把在最邻近的 k 个记录中出现次数最多的分类当作预测结果。若 k 个记录中的实例各不相同，则引入权重机制，把距离 x 最近的实例的分类当作预测结果。

算法中的第 13 行~第 21 行为剔除噪声的相关计算。定义并使用了 2 个函数：

predicate((z_i, c_{zi}))：定义了知识库中的记录(z_i, c_{zi})参与预测新实例的分类次数。也即每次更新操作时，所选中的 k 近邻中的记录所对应的 *predicate* 值都会增 1。

accuracy((z_i, c_{zi}))：定义了知识库中记录(z_i, c_{zi})参与预测新实例的分类时，预测准确的次数。因为采用的是一种投票机制，所以若 c_{zi} 与 x 的实际分类相同，则预测正确，反之则预测错误。

使用 *accuracy*((z_i, c_{zi}))和 *predicate*((z_i, c_{zi}))的比值作为评估记录(z_i, c_{zi})是否为噪声的标准。因为在机器学习中，尤其是在基于实例的学习中，默认的一个前提是：相邻的数据总是具有相似性。若库中的某一记录总是错误地预测其邻居实例的分类，则有较大把握认为该记录是噪声，并把它剔出知识库，以保持学习的精确性。

在学习模式下，通过分析被编译的程序获得当前实例，然后使用 k 近邻法，从知识库中学习出当前实例的分类，编译并生成可执行文件。

5 结束语

本文提出了一种结合迭代编译和机器学习的编译器框架。在该框架下，为编译器附加了一个知识库。在搜索模式下，编译器通过不断地编译执行，在优化空间中搜索性能最优的点。并使用搜索的结果更新知识库。在更新的过程中，采用了一种鉴别机制，以防止冗余记录入库。同时，采用了识别噪声的算法，把可能的噪声从知识库中剔除。编译器还可以从知识库中学习，预测未知的程序或者程序在不同输入下的最优变换。

与相关的工作相比，本文提出的迭代编译框架具有以下特点：利用知识库保存迭代编译过程中所积累的数据，用以指导未来对新程序的编译；知识库可以动态更新，并有噪声的识别和剔除功能。

参考文献

[1] Kisuki T, Knijnenburg P, O'Boyle M, et al. A Feasibility Study in Iterative Compilation[C]//Proc. of International Symposium of High Performance Computing. [S. l.]: Springer, 1999: 121-132.

[2] Kisuki T, Knijnenburg P M W. Iterative Compilation in Program Optimization[C]//Proc. of the 8th International Workshops on Compilers for Parallel Computers. Edinburgh, UK: [s. n.], 2000: 35-44.

[3] Knijnenburg P M. The Effect of Cache Models on Iterative Compilation for Combined Tiling and Unrolling[J]. Concurrency and Computation: Practice and Experience, 2004, 16(2/3): 247-270.

[4] 郝云龙, 赵荣彩, 侯永生, 等. 反馈式编译在循环级性能分析中的应用[J]. 计算机工程, 2011, 37(9): 32-34.

[5] Tom M. Mitchell Machine Learning[M]. [S. l.]: McGraw-Hill Press, 1997.

[6] Long Shun, O'Boyle M. Adaptive Java Optimization Using Instance-based Learning[C]//Proc. of the 18th ACM International Conference on Supercomputing. [S. l.]: ACM Press, 2004: 237-246.

[7] Stephenson M, Amarasinghe S, Martin M, et al. Meta Optimization: Improving Compiler Heuristics with Machine Learning[C]//Proc. of ACM Conference on Programming Language Design and Implementation. New York, USA: ACM Press, 2003.

编辑 顾逸斐