第33卷 第23期 计 算 机 工 程 2007年12月
Vol.33 No.23 Computer Engineering December 2007
·软件技术与数据库· 文章编号：1000—3428(2007)23—0078—03 文献标识码：A 中图分类号：TP391

动态二进制翻译中热路径优化的软件实现

史辉辉，管海兵，梁阿磊
（上海交通大学计算机科学与工程系，上海 200240）

摘 要：在动态二进制翻译中，热路径的识别和生成是提高二进制翻译器效率的重要环节。为了提高热路径预测的命中率，必须在程序的运行中搜集较为详细的信息，这必然增加系统的开销。因此，在准确率和开销之间做出权衡十分必要。该文在研究现有热路径算法的基础上，提出了一种改进的基于路径的热路径识别和优化算法，并对结果进行了分析。

关键词：动态二进制翻译；热路径；动态优化

Hot Path Optimization in Software Dynamic Binary Translation

SHI Hui-hui, GUAN Hai-bing, LIANG A-lei
(Dep. of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200240)

【Abstract】 The recognition and creation of hot paths is one of the key factors in promoting the performance of a dynamic binary translator. To promote the accuracy of the hot path prediction, it is necessary to do a more thorough run-time profiling, which will lead to a rise in overhead. So a tradeoff between accuracy and overhead is needed. This paper studies the current profiling strategies and proposes an improved hot path recognition scheme.

【Key words】 dynamic binary translation; hot path; dynamic optimization

热路径的识别和生成是动态二进制翻译中优化的重要方面。一般的程序都具有局部性原理，即90%的时间都在执行10%的代码。如果能识别出这10%的代码并进行优化，系统的性能就可得到显著提高。

目前的二进制翻译器大都把基本块作为翻译和执行的基本单位，运行时信息的收集也是针对基本块的。热路径基于的思想是：当前较热的一条路径在以后的一段时间内很有可能也是热的。因此，可以根据前面已经运行的信息来预测将要运行的信息。但是只有热路径带来的收益大于开销时才能从整体上提高系统的效率。另外，错误的预测只会增加开销，因此，要尽量提高预测的准确率。越是准确的预测，需要越为详尽的统计信息，系统的开销就越大。所以，既要做出尽可能准确的预测，又必须尽量减小其开销。为此，本文提出了一种改进的热路径识别算法。

1 动态二进制翻译中热路径的识别

在动态二进制翻译中，翻译是即时进行的，对效率有较高的要求。一般的动态二进制翻译器都采用热路径预测的方法。一旦识别出热路径，就对其进行优化，包括热代码连接、建立间接跳转cache等[1]。

常见的动态二进制翻译中的热路径识别方法有3种：基于基本块的识别，基于边的识别及基于路径的识别。这三者预测的准确率呈递增之势，实现的难度和算法的复杂度亦相应增加。

1.1 基于基本块(basic block)的热路径识别

在程序运行过程中，对每个基本块的执行次数进行统计，一旦达到预先设定的阈值便开始生成热路径。其过程是：(1)找到循环的入口基本块，把它作为热路径的开始块；(2)在每次遇到条件跳转时就比较两个目标块执行的次数，并把执行次数较多的那个块作为热路径上的下一个基本块；(3)当遇到循环出口时便认为热路径结束。

1.2 基于边(edge)的热路径识别

上述算法可以以较低的代价生成一条热路径，但由于收集的信息不够全面，预测错误的概率较大。

基于边的热路径识别算法中记录了基本块之间的跳转次数。用一个二元组(A, B)表示从基本块A到基本块B的一次跳转，用$N(A, B)$表示该跳转发生的次数。本文对第1.1节中的算法稍作修改：在遇到条件跳转时，比较以A为源块的两条边(A, B)和(A, C)的跳转次。若$N(A, B)>N(A, C)$，则把B作为热路径上的下一个基本块；否则，选择C。

该算法需要的信息比第1.1节中的算法多，但预测更准确；同时，其实现简单，因此，该算法为许多二进制翻译器采用。文献[2]为基于边的算法提供了理论基础。

1.3 基于路径(path)的热路径识别

基于边的热路径算法在路径之间有较多重叠的基本块时，预测难免发生错误。原因是每条边只是一条路径的一部分，用部分来预测整体明显不够准确。改进的方法是为每条从循环入口到出口的边建立一个档案，记录每条路径被执行的次数。但是由于每条边有多个基本块，每条边所包含的基本块的信息需被保存。

2 改进的热路径算法

在上述基于基本块和基于跳转边的算法中，每次跳转都

基金项目：国家“973”计划基金资助项目(2004CCA02600)
作者简介：史辉辉(1982－)，男，硕士研究生，主研方向：二进制翻译；管海兵、梁阿磊，副教授
收稿日期：2006-12-25 **E-mail：**hhshi@sjtu.edu.cn

要更新计数器。基于路径的算法只对整条路径进行计数，而忽略了中间细节。因此，必须找到一种有效的方法来记录一条路径所包含的基本块。

2.1 热路径的编码表示方法

可以用编码的方法表示一条路径。在每个基本块结束时，忽略直接跳转。由于条件跳转有两个目标地址，因此可以用二进制编码的方法记录跳转信息，如图 1 所示。如果跳转的目标地址是地址 1，用 0 表示；如果跳转目标地址是地址 2，用 1 来表示。用上述编码方法，每条路径唯一地对应于一个码字。表 1 给出了图 1 中每条路径的码字。

表 1　图 1 中路径对应的码字

路径	码字
ABDEF	001
ABDE	00
ABDF	01
ACDEF	101
ACDE	10
ACDF	11

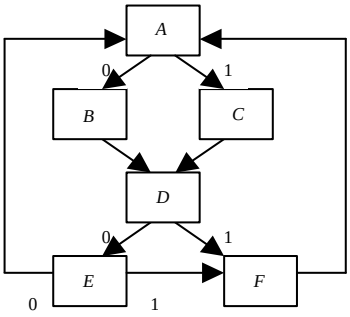


图 1　编码方法

由于(*B, D*)和(*C, D*)之间的跳转是直接跳转，只有一个跳转地址，因此不必对其编码，以有效减短码字的长度。

2.2 基于路径的热路径算法的分析

上述编码方法只针对条件跳转，码字的每一位都对应一个分支。如果将这些分支用二叉树表示(图 2)，则从根节点到叶子节点的每条路径都对应从循环入口到出口的一条路径。

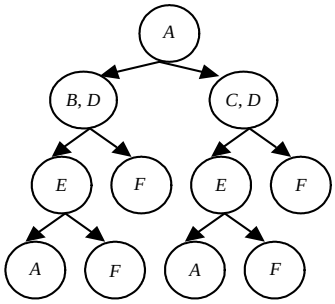


图 2　路径的二叉树表示

假设路径的平均长度位k，即二叉树的高度为k，那么不同路径的数目约为 2^k，它将随路径的长度呈指数型增长。当k比较大时，为了找出一条热路径，不可能为每条路径都设置一个计数器。文献[3]用NET预测算法，通过路径头的信息来预测热路径。该算法可以减小长度k，从而降低预测延迟，但仍有一定的盲目性，而且在生成热路径时还需要做一些额外工作。

基于路径的热路径算法的另一个缺陷是，随着路径长度的增加，路径的数目急剧膨胀，预测的命中率也就急剧下降。最坏的情况下热路径命中的概率为 2^{-k}。由于程序的运行有局部性原理，每条路径的概率都不一样。但是在复杂程序中，频繁执行的路径不止一条，假设为 m，则命中概率为 $1/m$，随 m 的增加而迅速减小。这种情况同样存在于另外两种热路径算法。

因此，要在尽可能减少信息收集的工作量同时，针对各条路径均匀执行的情况进行特别处理，以提高优化后代码的利用率。

2.3 基于编码的路径的形式化定义

由于每条路径都唯一对应一个码字，可以定义起始地址为 A，长度为 k 的路径如下：

$$p(A,k)=\{A,x_1x_2,\cdots,x_k \mid x_i=0,1\}$$

即一条路径可唯一地表示为一个基本块的地址和一串 0, 1 序列，其中，0 或 1 代表了从循环入口到出口之间所经历的基本块。

同理定义一条路径上的一个长为 l 的段(fragment)如下：

$$f(B,l)=\{B,x_1x_2\cdots x_l \mid x_i=0,1\}$$

它表示从循环中任一个基本块 B 开始，长度为 l 的一段路径。

有了路径和段的概念之后，定义两条路径的交运算如下：

$$F = p(A,k)\cap p(B,j)$$

其中，F 是一个段的集合：

$$F=\{f(B_i,l_i) \mid f(B_i,l_i)\subseteq p(A,k),\ f(B_i,l_i)\subseteq p(B,j),1\le l_i\le \min(k,j)\}$$

F 中包含了两条路径 $p(A,k)$ 和 $p(B,j)$ 相重叠的基本块序列。

假设在热路径被识别出之后路径 p 被执行的次数，即执行频率为 $q(p)$，阈值为 N，那么当某个循环内部有 k 条路径时，总的循环次数大约为 $\sum_{i=1}^{k}q(p_i)+N*k$。定义路径 $p_j(1\le j\le k)$ 的命中率为

$$h(p_j)=q(p_j)\Big/(\sum_{i=1}^{k}q(p_i)+N*k)$$

那么预测错误的概率即为 $1-h(p_j)$。本文目标就是用尽可能低的代价来提高命中率 $h(p_j)$。

2.4 算法实现

为了准确识别出热路径，必须进行较为详尽的信息收集。传统的做法是对所有的基本块执行情况进行跟踪，并通过计数器来保存执行信息。由于热代码只占所有代码的一小部分，因此应该尽可能只收集那些有可能成为热代码的基本块的信息。本文的做法是寻找那些往回跳的边，而这往往是一个循环的开始。因此，只收集可能在循环内部的基本块的信息，则可大大减少开销。

用二进制编码的方法可以使码字与路径一一对应，为每条路径维护一个计数器即可。路径的出口发生在回跳处或路径达到规定的最大长度处。设所有路径的最大长度为 L，这样防止了程序在顺序执行时路径过长。

当某条路径的计数器达到阈值 N 时，便认为它是热路径，其码字决定了它所包含的基本块的信息。当一条路径的计数器值远远大于同一循环内其他路径时，可称它为绝对热路径，否则称其为相对热路径。对于绝对热路径，由于它的所有基本块都是热块，可以直接进行优化。但是在有些情况下，特别是循环体内条件分支比较多的情况下，会出现两条或多条路径平均执行的情况。这时识别出的是相对热路径，它的基本块即有热的，也有冷的。由于要将优化作用在热代码中，以减少不必要的开销，因此需识别出相对热路径中的热段。

热段在路径中有 3 种可能出现的位置，如图 3 所示。当热段位于路径两端，实现比较简单；当热段位于路径中部时，实现较为复杂，但由于位于路径中部的热段相对另外两种情况出现的频率较低，因此该情况不予考虑。

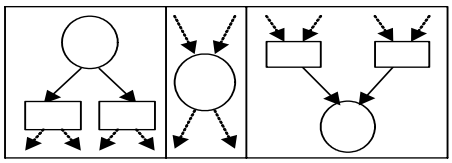


图 3　热段在热路径中的位置

为了找出热段，将同一循环内两条执行次数最多的路径对应的码字分别按高位对齐和低位对齐进行按位异或，结果码字中连续为 0 的片断对应的段即为候选热段。当候选热段在原路径中的首基本块相同时便可被确认为热段。

假设有两条相对热路径：$p(A,7)=\{A,0110010\}$，$p(A,6)=\{A,011110\}$，按位异或后结果为

```
0110010          0110010
^011110          ^011110
-------          -------
000111           111100
```

这样前 3 个基本块序列便可被确认为一个热段；如果两条路径的倒数第 2 个基本块相同，则最后 2 个基本块序列便也可被确认为热段。由于热段长度一般都比较短，可以将其中的基本块合并为一个超级块，为多条路径所共用。

3　实验结果与分析

本节给出了在 Valgrind 上运行 nbench benchmark suite 的实验结果和分析。Valgrind 是 Linux 下的一个内存跟踪工具。它有一整套与平台无关的中间代码，可以动态地完成从不同体系结构源二进制代码到中间代码和从中间代码到目标二进制代码的翻译。

信息收集时采用不同的策略在效率上有较大的差异。表 2 中给出了 3 种策略的具体数据。由表 2 可知，跳转边的数目最多，为每条边维护一个计数器将会为系统增加较大的负担。相比之下，路径的数目要少得多。由于每条路径仅需一个码字即可表示，因此不但可以节省内存空间，也减少了更新计数器的频率。

表 2　计数器数目对比

Benchmark	基本块	跳转边	路径
Num Sort	1 666	2 186	700
String Sort	1 793	2 358	750
Bitfield	1 703	2 235	713
Emulation	1 889	2 526	827
Fourier	1 701	2 215	697
Assignment	1 761	2 339	762
Idea	1 754	2 333	763
Huffman	1 743	2 296	739

上文定义的绝对热路径和相对热路径的处理方法略有不同。图 4 给出了 benchmark 中各个程序所包含的绝对热路径和相对热路径的情况。由结果中可知，一般情况下绝对热路径数目略多，但是在 Assignment 和 huffman 中，相对热路径和绝对热路径相当。这是因为这些程序中条件语句和 switch 语句数目众多，路径执行情况比较均匀。

图 5 给出了 3 个有代表性程序的平均命中率。其中，横坐标表示预测延迟[4]，即热路径在达到阈值之前执行次数占总次数的百分比。本文在定义命中率时已经考虑了预测延迟。这是因为，如果仅考虑命中的概率，预测延迟越大预测越准确。但由于希望热路径带来的收益尽可能的大，因此预测延迟应尽可能小。最终本文选择从总体收益来衡量。由图 5 可知，大部分程序的最高命中率在约 10%的预测延迟处获得。此时，热路径已经被正确地识别出，如果继续增大预测延迟，只会减少热路径的收益，因此，从 20%往后，命中率将平滑地下降，按照本文定义，当预测延迟达到 100%时，命中率为 0，即此时热路径将不会带来任何收益。

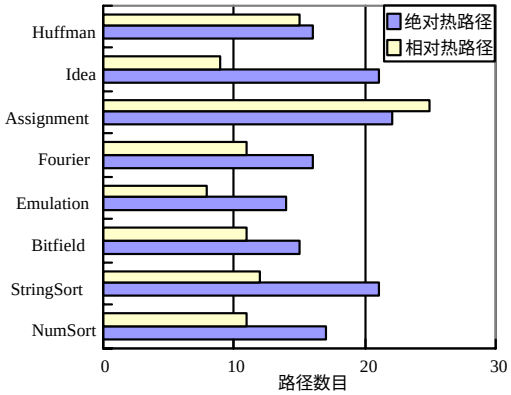


图 4　绝对热路径和相对热路径

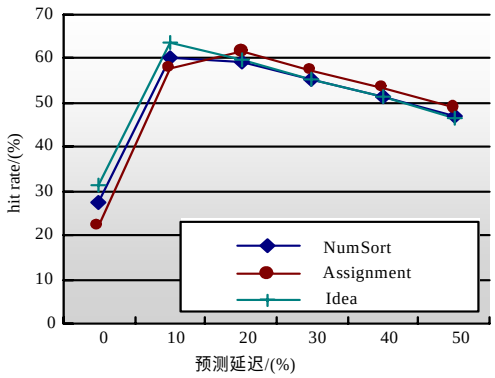


图 5　3 个测试程序的命中率

由图 5 可知，在预测延迟为 0，即进行盲预测时，Assignment 的命中率最低。这是因为其中的相对热路径最多，直接预测时选择较冷路径的概率较大。这与图 4 中相对热路径的统计结果一致。

图 6 所示是以本地直接执行为基准的运行时间。纵坐标表示边预测和路径预测相对于本地执行花费时间的倍数。由上述分析可知，Assignment 和 Huffman 中条件分支众多，相对热路径数目相对较大，因此，边预测效果应该和路径预测相差较大。这和实验结果基本吻合。

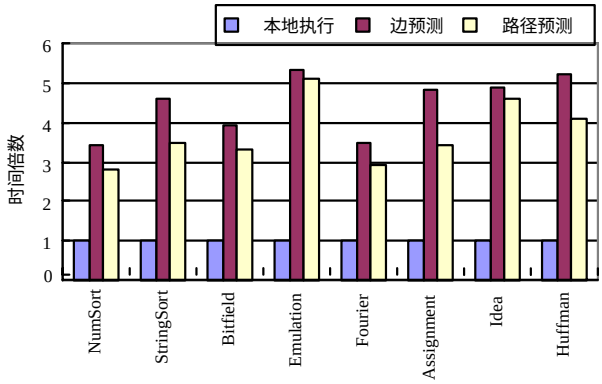


图 6　运行时间比较

总之，改进的路径预测算法无论在提高预测准确度方面，还是在降低系统开销方面都有明显优势。

（下转第 83 页）