动态二进制翻译的优化

李增祥 管海兵 李晓勇
(上海交通大学信息安全工程学院 上海 200240)

摘 要 提出动态二进制翻译的两种优化方案:基本块和热路径;分析了从代码中抽取值得优化部分的详细过程;同时也给出针对这两种方案的一些优化方法。最后简单介绍了当前一些动态二进制翻译系统所采用的优化技术。

关键词 二进制翻译 动态二进制翻译 动态优化 热路径 基本块

OPTIMIZATION FOR DYNAMIC BINARY TRANSLATION

Li Zengxiang Guan Haibing Li Xiaoyong
(*School of Information Security Engineering, Shanghai Jiaotong University, Shanghai 200240, China*)

Abstract The thesis proposes two optimization approaches for dynamic binary translation techniques: basic block optimization and hot paths optimization. How to find the codes that deserves optimization is discussed in details. At the mean time, the thesis provides some optimization methods for these optimization approaches. At last, it briefly introduces some relative works on the optimization approaches for dynamic binary translation technique at present.

Keywords Binary translation Dynamic binary translation Dynamic optimization Hot paths Basic block

0 引 言

研究和开发新的体系结构必须要有相应的软件支持,才能得到流行和运用;而一些非常宝贵的软件也面临着由硬件升级换代导致的软件更新问题。代码移植成为了软硬件协调发展的一个重要因素。因此二进制翻译方法应运而生,它有助于将老的体系结构上的软件移植到新的体系结构上运行。根据翻译时机的不同,二进制翻译可以分为静态二进制翻译和动态二进制翻译。

静态二进制翻译有其天生的局限性。静态二进制翻译无法发现一个程序中所有的指令代码,往往需要一个解释器,在运行期处理所有未被翻译的代码。而且静态二进制翻译在运行过程中往往需要用户的参与,缺乏透明性。

动态二进制翻译克服了静态二进制翻译的局限性,但损失了性能。因为在运行代码的同时,目标机器还要做翻译的工作。性能较低是动态二进制翻译的瓶颈所在。近年来,动态调度和动态编译器优化技术的发展给动态二进制翻译优化提供了很多可以借鉴的经验。动态二进制翻译的优化成为了非常有潜力和有意义的研究方向。

1 二进制翻译优化的方法

大多数的程序都将大部分的时间花费在很小一部分代码段上。对于一个动态二进制翻译器来说,只有那些频繁执行的程序段才值得花时间优化。动态二进制翻译的优化就是通过一些机制来统计代码的运行情况,然后利用这些统计信息来确定哪些部分的代码值得被优化。

在传统的编译器技术里,优化都是基于静态分析的,优化是针对整个程序,虽然这样能提高系统性能,但花费的代价却非常大。在一个动态环境里,我们可以根据代码的运行情况,根据代码段被执行的次数不同,采用不同等级的优化方法。选择要实施的优化方法在很大程度上取决于程序执行所花费的时间与信息收集过程的优化代码的开销之间的平衡,要求确保使用优化所获得的利益大于它的开销。总之二进制翻译优化的中心思想就是:找出频繁执行的代码段,然后根据代码被执行次数来使用不同等级的优化方法。

1.1 基本块优化技术

1.1.1 确定值得被优化基本块

动态二进制翻译采用一边翻译、一边执行的方法。它处理代码的基本单位是基本块。一个基本块是以一个控制转移(如一个分支、调用或跳转指令)结束的指令序列。根据代码的执行特点,肯定有一些基本块会被频繁的执行,而有些基本块无人问津。基本块的优化技术就是通过计数器计算基本块在程序运行期内被执行的次数。当计数器数值达到某个阈值时调用一个优化器来优化这段代码并将优化后的代码保存在缓存 TCache (Translation Cache 是用于存放翻译后的代码的内存空间)中。如果以后还要运行这个基本块,就可以直接运行保存在 TCache 中优化后的代码,而不再需要重新翻译[1]。其执行过程可以参照图 1:

1.1.2 具体优化技术

通过上面的方法我们可以确定哪些基本块需要被优化。由于优化是要付出一定开销的,而且不同级别的优化方法需要不

收稿日期:2005 - 09 - 16。(973)前期专项(2004CCA02600)。李增祥,硕士生,主研领域:计算机体系结构,二进制翻译。

同数量的开销。为了使优化更加有效率,更加有针对性。我们根据不同的阈值,采用下面不同级别的优化技术:

(1) 不进行优化　只是把翻译后的代码放入内存,这样做的好处是,代码下次运行时就不必重新翻译了。存储电路单元的开销要比逻辑电路的开销要小得多,所以这种做法以空间换取时间,从而提高程序运行速度。

(2) 分析寄存器活动　采用前向替代、常量传播等来提高生成代码的质量并减少被执行指令的数量。

(3) 寄存器分配　通过分析寄存器活动信息,在目标机器上分配硬件寄存器。程序可以直接使用目标机器上的硬件寄存器,而不再是访问存放在内存中的虚拟寄存器。以此来提高指令运行速度。但这也将导致寄存器和内存之间频繁交换数据。为此我们可以通过分析寄存器生命周期来减少不必要的寄存器的读写和与其相关的计算[2]。

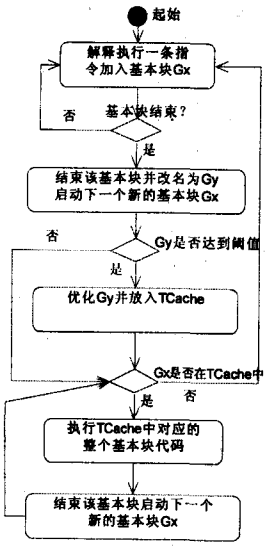

图 1　基本块优化的流程图

(4) 跳转优化　根据上面的流程,跳转发生的时候基本块结束,要求切换回动态二进制翻译系统,并由它来确定下一个基本块是否在 TCache 中。如果跳转的目标地址正好是 TCache 中的某个基本块,可以在原基本块的末尾加上一个跳转,直接跳入到下一个基本块中,这样程序就可以一直在 TCache 中运行,无须切换回动态二进制翻译系统。我们也可以设计一个表把源机器代码的跳转目标地址映射到目标机器上对应的地址。在基本块末尾,查看这个表,如果命中就直接跳入下一个基本块,从而避免切换回二进制翻译系统[3]。

(5) 代码移动　移动代码将一些把经常被连续执行的基本块放在一个连续的地址空间中。这样做可以使得代码更加具有局部性,从而提高 cache 的效率,同时也可以降低代码中的跳转指令的数量。

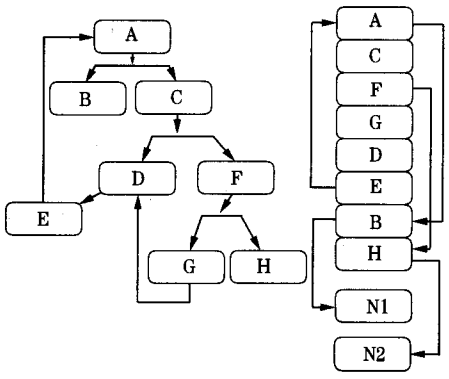

图 2　代码移动的例子

图 2 是一个代码移动的例子。左边就是一个程序的控制流图,这个程序很大一部分时间都是在执行 ACFGDEA 循环。右边显示代码存放的位置关系。其中 ACFGDE 被安排在一个连续的地址空间,B、H 是出口。这样一来我们就提高了代码的局部性,同时也减少了跳转指令的次数。比如说:如果 B 和 C 都不和 A 连续,那么在原来块 A 的结尾处就要两个跳转指令,一种情况跳入 B,一种情况跳入 C。而按照现在的存放基本块的方法,我们只要一个跳转指令,满足某种条件,跳入 B,否则继续执行,这也就是按照最经常执行的路径,转入基本块 C,按照 cache 的原理,基本块 C 紧接着基本块 A,所以很有可能已经预先存入 cache 中,这样就大大提高了 cache 的效率。

1.2　热路径优化技术

描述一个程序的运行控制流可以使用以下两个统计技术:节点权值和边界权值。其中节点权值是指基本块被执行的次数,基本块优化技术就是通过统计基本块的执行次数来确定哪部分代码将被优化;而边界权值则记录两个节点之间的流控制转移的频率。我们采用边界权值是因为它比前者更加能反应在执行过程中基本块之间的连接关系。一个边界权值包括一对基本块和一个计算器,它在代码生成时的初始值为 0。代码执行时每通过一个边界,对应边界的权值就加 1。

一个热路径就是一个指令序列,这个指令序列可以包括好几个基本块。它在程序运行的某个期间被频繁执行[4]。因为大部分程序将它们的大部分执行时间花在小部分代码上,因此能够识别出热路径并优化它,将会大大提高执行速度。

根据边界权值的定义,我们设想一下,如果可以找到一条路径,它经过的边界权值之和最大,那么我们就可以判定,这条路径是最经常走的。但是寻找这条路径的计算方法复杂,需要大量统计数据,而且根据运行的情况,这些统计数据在不断的变化。为此我们采用了下面比较简单的办法来确定热路径。

1.2.1　识别热路径

在运行时,需要设置一个触发器,来启动整个优化过程。当某个边界 t 被经过的次数达到某个阈值,触发器产生,开始寻找 t 触发的热路径并对其进行优化,如图 3 所示。

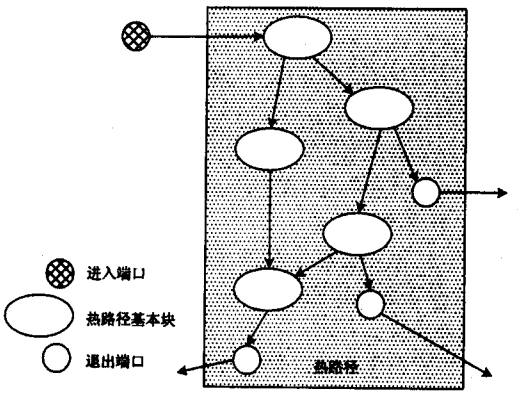

图 3　热路径的进入端口和退出端口

我们首先把那些从 t 触发的候选边界归属到一个热路径中。(候选边界就是权值超过最小基值 BaseCount 的边界,其中 BaseCount 一般设置为 t 的阈值的一半)。从 t 开始,我们只关注热边界 t 的外部边界。t 的外部边界定义如下:

$$\text{outEdge}(t) = \{x \in E \mid \forall a,b,c \in BB \bullet t = (a \to b) \wedge x = (b \to c)\}$$ (BB 为基本块集合,E 为边界集合)

从这些外部边界中,我们只选出那些候选边界(Candidate),并且以此构建我们的下一个热边界集。

$$\text{HotSer}(0) = \{t\}$$

$$\text{HotSet}(1) = \{x \in \text{Candidate} \mid \forall_y \in \text{HosSet}(0) \bullet x \in \text{OutEdge}(y)\}$$

$$\text{HotSet}(2) = \{x \in \text{Candidate} \mid \forall_y \in \text{HosSet}(1) \bullet x \in \text{OutEdge}(y)\}$$

……

$HotSet(n) = \{x \in Candidate \mid \forall_y \in HosSet(n) \bullet x \in OutEdge(y)\}$

最终的从边界 t 生成的热边界集的集合就将用于生成热路径：

$$HotEdgeSet(t) = \bigcup_{i=0}^{n} HotSet(i)$$

上面的 n 是个有限值，它是满足下面条件的最小值：

$$HotSet(n) \subseteq \bigcup_{i=0}^{n} HotSet\{i\}$$

不过如果使用这个算法的话，寻找 HotEdgeSet(t) 的代价也是非常昂贵的。在实际的实现中，HotEdgeSet(t) 通过递归算法递增实现，将那些刚刚达到 BaseCount 的边界不断的加入到 HotEdgeSet(t) 中。

在 HotEdgeSet(t) 确定以后，我们就可以在此基础上进一步寻找热路径

$HotBBSet(t) = \{x,y \in BB\}$，for all $(x \rightarrow y) \in HotEdgeSet(t)$

$HotPathBBSet(t) = \{customize(z)\}$，for all $z \in HotBBSet(t)$

$HotPathEdges(t) = \{e \in E \mid \forall_x, y \in HotPathBBSet(t) \bullet e = (x \rightarrow y)\}$

其中 HotBBSet(t) 是指和 HotEdgeSet(t) 中的边关联的基本块的集合。而 HotPathBBSet(t) 是指优化后的 HotBBSet(t) 中的基本块。HotPathEdges(t) 就是和这些优化后的基本块相关的边组成的集合。

对于那些不属于 HotPathEdges(t) 但是属于 HotPathBB(HotPathBBSet(t) 的元素) 的外部边界的边界，可以看作是热路径的出口。一个退出端口就是一个将控制转出热路径的基本块。一个从 HotPathBBx 出去的退出端口是：

$OutPortal(x) = \exists_y \in BB \bullet (x \rightarrow y) \notin HotPathEdges(t)$

那么从热路径出去的所有可能路径就是从 HotPathBB 到它们的退出端口的边界。离开热路径的关系集就是：

$OutPortalSet = \{e \in E \mid \forall_x \in HotPathBBSet(t) \bullet e = (x \rightarrow OutPortal(x))\}$

进入热路径的边界成为进入端口：

Inportal = (x→customise(x))，条件是：(x→y) 是能够触发优化的边界。

热路径的最终定义如下：

$HotPath = HotPathEdges \cup OutPortalSet \cup \{InPortal\}$

1.2.2 具体优化技术

对热路径优化方法有非常多，很多动态优化的方案都可以采用，这边主要讨论其中的几种：

(1) 拷贝传播

在指令系统中有不少指令用于寄存器之间的拷贝，比如说：add r3,r4,0。如果紧接着的指令要求使用 r3，我们可以直接先使用 r4 来取代，这样这两条指令就可以并行运行了，从而提高代码运行速度[5]。

如果图 4 中右边的分支是热路径，那么我们就可以针对这个热路径进行优化，优化后的情况是：Add r3，r4，0；Add r6，r4，r5 这两条语句没有数据依赖性可以并行运行。如果没有引入热路径的话，优化的单位是一个基本块，而上面两条指令属于不同块，所以上面的优化是没办法实现的。

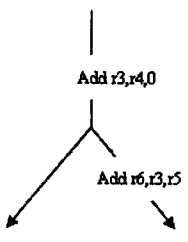

图 4 拷贝传播

(2) 循环展开

循环展开是开发循环并行性的一种重要编译优化技术。它把连续的多个循环体展开组成一个大的循环体，以便更有利于发现指令间的并行性。循环展开不但减少了循环转移开销，还提高了资源利用率[6]。循环展开虽然是一种编译器优化技术，但是我们也完全可以运用到动态二进制翻译当中，比如说程序中，反复进入一个循环体，那么这个循环将会是一条热路径。这样我们就可以对这条路径进行优化，采用相近的办法把循环体展开，从而提高代码执行效率。

(3) 改进代码位置

将频繁连续执行的基本块移到一起能更好的使用缓存同时减少它们之间的控制流开销。举一个例子：图 5(a) 显示了翻译器最初生成的代码。每个生成的基本块 A、B、E 都占用自己的空间，并与其它基本块有一定的距离。把这些基本块移动到一起，得到图 5(b) 显示的结果。

热路径也可以看作是一个特殊版本的执行路径，它包含优化和合并的基本块。热路径本身也可以被看作是一个基本块，只是它允许基本块中的流控制从任何地方退出(退出端口) 而不是只在块末尾。为了使程序进入优化后的热路径，必须对开始的基本块做一定修改，将程序执行重定向到热缓存的开始。所有其它的基本块保持不变，这样才可以确保基本块之间的跳转关系不受影响。图 6 显示了热路径生成以后的翻译代码的状态。

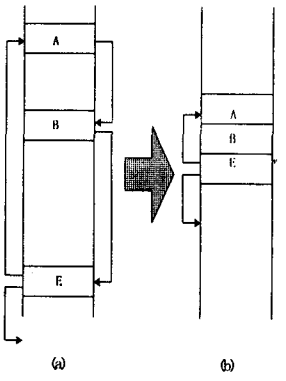

图 5 改进代码位置

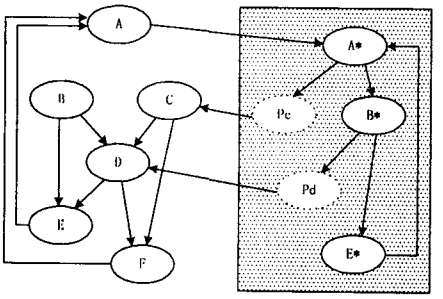

图 6 带热路径的程序状态转换图

初始块 A 开头部分加一条到热路径的开始端的跳转指令。A*、B* 和 E* 是优化后的代码，它们被放在一个连续的存储空间。Pc 和 Pd 是连接回到未优化的 C 和 D 的退出端口。其中 B、C、D、E、F 都没有做任何修改，保持了原来的跳转关系。

2 相关工作

Digital 的 FX!32 的优化方案是：先让程序通过仿真器初始

(下转第 42 页)

4 分 析

Web Service 采用 Soap Over HTTP 对各模块间的交互进行封装,虽然简化了交互过程,降低了集成难度,但大量采用 Web Service 技术会降低系统的效率。以教务管理子系统为例,学生选退课,课件点播等功能模块并发访问量非常巨大,采用 Web Service 技术会导致响应的延迟。对于一些仅仅面向管理的功能,比如毕业生就业分析,学生成绩评估,采用 Web Service 技术进行集成能为其它业务子系统提供服务。

将 Web Service 模式真正应用于国家示范攻关项目,其先决条件是 SOAP 标准、WSDL 和 UDDI 能否得到参与者的广泛认同,是该模式能否得到全面推广应用并获得成功的关键。令人欣慰的是,包括 Sun 和微软以及其他主要的中间件、数据库提供商普遍支持,各类开发平台也日益完善,相关技术人员也正在全面成长,成功范例时有出现,再加上本次携手攻关的各大高校和事业单位都有丰富的 Web 教育系统开发经验,这一切都为选择 Web Service 技术对网络教育系统的集成提供了坚实的基础。

5 结论与展望

本文分析并阐述了在"十五"国家科技攻关专项"网络教育关键技术及示范工程"中系统集成应用中所遇到的问题和相应的 Web Service 解决方案。文中提出了网络教育系统集成的功能架构,并结合 Web Service 技术的体系结构给出了基于 Web Service 的集成流程和功能视图以及用户访问视图。采用 Web Service 技术集成,降低了各模块的耦合度,通过细粒度的 Web 服务实现了丰富的 Web 服务组合功能,这些特性为进一步扩充系统的功能提供了强有力的支持;在实际开发过程中,还提出了在采用 Web Service 技术集成必须遵守 3.4.1 中给出的两个原则,这样才能最大限度地利用 Web Service 技术,降低效率上的损失。针对目前的发展状况,网络教育系统的集成要继续在 Web Service 技术的支撑之上做进一步的研发,提升网络教育系统整体的性能和功能,同时,这不只是一个技术的问题,还需要从国家宏观管理和项目整体实施方面作进一步的保障,制定统一标准,协调整合各个研发和教育实体的力量,从根本上提高各大教育实体网络教育系统之间的互操作性,实现更大程度上的教育资源共享与整合,为我国现代教育事业实现国民大教育的理想做出贡献。

参 考 文 献

[1] http://www-900.ibm.com/developerWorks/cn/webservices/ws-wsca/index.shtml.

[2] Phil Wainewright. Web Service Infrastructure.

[3] 中国教育信息化技术标准 CELTS-20 计算机教学管理系统规范.

[4] 赵明. 国内的网络教育的发展状况. 网站:人民日报 > > IT > > 网络应用,2003.

[5] Charlton J P, Birkett P E. An Integrative Model of Factors Related to Computing Course Performance J. of Education Computing Research, 1999,20(3):237-257.

(上接第 14 页)

运行,在此过程中收集统计资料,并通过这些资料来做本地优化。然后离线翻译并优化程序。它的优化过程是静态的,只有在程序再次被运行时才能够使性能得到提高。

动态优化器方面,如 Dynamo 和 Wiggins/Redstone,它们实际上并没有做翻译动作。它们使用本地二进制代码,代码优化后在同一种机器上运行。他们的动态优化器主要针对特定的机器,所以常常依赖于底层硬件的支持,但是可以获得相当高的效率。Dynamo 是通过指令解释来确定热路径的,而 Wiggins/Redstone 则是通过采样。这两个系统都对热路径进行优化,热路径在该路径初始化时建立。其理论依据是如果一个分支指令被确定是一条路径的开始,那么紧跟其后执行的指令集就很有可能也是热的。

IBM 的 Daisy 是基本块优化的一个很好的例子,Daisy 在程序运行过程中,通过一些规则划分基本块,然后根据这些基本块被执行的次数来决定是否要把代码翻译成 VLIW 指令,从而提高该基本块的执行效率。当代码进入下一个基本块的时候首先判断是否该基本块已经被翻译,如果翻译了,就直接执行翻译后的 VLIW 代码,否则继续解释执行下一个代码[7]。

3 结 论

动态二进制翻译可以采用很多动态优化方法,其中基本块优化和热路径优化都是非常有效的优化方案,两种优化方案都是通过软件的方法统计代码运行的信息,虽然增加了一定的开销,但是它们取消了对硬件的依赖,有更加广泛的适用性。通过这些统计信息,我们可以非常准确的确定哪些部分的代码必须进行优化处理。同时我们针对两种优化方案提出了一些优化代码的方法,他们能够增加代码的并行性,提高代码运行速度,同时通过重新安排基本块的地址,增加了代码的局部性,以此大大提高了 cache 的性能,同时也减少了一些多余的分支指令。如果使用这些技术的话,动态二进制翻译将摆脱对硬件的依赖,适用于多种机器,同时优化后的性能也将得到很大的提高。

参 考 文 献

[1] Altman E R, Ebcioglu K, Gschwind M, Sathaye S. Advances and future challenges in binary translation and optimization. Proceedings of the IEEE,2001,89(11):1710-1722.

[2] Probst M, Krall A, Scholz B. Register liveness analysis for optimizing dynamic binary translation Reverse Engineering, 2002. Proceedings. Ninth Working Conference on 29 Oct.-1 Nov. 2002:35-44.

[3] Scott K, Davidson J, Skadron K. Low-overhead Software Dynamic Translation, Univ. of Virginia Dept. of Computer Science Tech Report CS-2001-18, July, 2001.

[4] Ung D, Cifuentes C. Optimising hot paths in a dynamic binary translator. Second Workshop on Binary Translation, Philadelphia, Pennsylvania, 2000.

[5] Michael Gschwind, Erik R. Altman, Sumedh Sathaye, Paul Ledak, David Appenzeller, Dynamic and Transparent Binary Translation, Computer IEEE, 2002, 33(3):54-59.

[6] Ebcioglu K, Altman E R, Sathaye S, Gschwind M. Optimizations and oracle parallelism with dynamic translation. Microarchitecture 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on 16-18 Nov. 1999:284-295.

[7] Ebcioglu K, Altman E, Gschwind M, Sathaye S. Dynamic binary translation and optimization. Computers, IEEE Transactions, 2001, 50(6):529-548.