计算机研究与发展
Journal of Computer Research and Development

ISSN 1000-1239/CN 11-1777/TP
44(1): 161~168, 2007

动态二进制翻译与优化技术研究

李剑慧[1,2] 马湘宁[2] 朱传琪[1]
[1](复旦大学计算机科学与工程系 上海 200433)
[2](英特尔中国软件中心 上海 200241)
(Jian.hui.li@intel.com)

Dynamic Binary Translation and Optimization

Li Jianhui[1,2], Ma Xiangning[2], and Zhu Chuanqi[1]
[1](*Department of Computer Science and Engineering, Fudan University, Shanghai* 200433)
[2](*Intel China Software Center, Shanghai* 200241)

Abstract Dynamic binary translator is a just-in-time compiler, which translates the instructions of source architecture to the instructions of target architecture when an application is running. The technology enables the application compiled for source architecture running on top of target architecture without recompilation. This paper begins with the basic framework of dynamic binary translator, and then gives an overview of several leading dynamic binary translators. After that, it has a deep discussion about key challenges of the dynamic binary translator, including supporting precise exception in optimized code, mapping source architectural context to target architectural context, translating self modifying code, reducing translation overhead, and dynamic optimization using profiling data. The paper ends with the hot research topics and possible usage models of the dynamic binary translation technology.

Key words *dynamic binary translation; dynamic binary optimization; precise exception; self modified code translation; translation overhead*

摘 要 动态二进制翻译技术是一种即时编译技术,它将针对源体系结构编译生成的二进制代码(源机器码)动态翻译为可以在目的体系结构上运行的代码(翻译码).动态优化技术是指在运行时获取动态信息并进行代码优化的技术.动态二进制翻译及优化系统使得源软件无需重编译就可以直接在目标体系结构上高效地运行.目前几种比较有影响的动态二进制翻译及优化系统有 Intel 公司的 IA-32 Execution Layer,IBM 公司的 DAISY,Transmeta 的 CMS 及 HP 的 Dynamo 等.这些系统对动态二进制翻译系统关键技术有不同的实现.对动态二进制翻译和优化技术的研究是计算机领域的研究热点,具有深远的现实意义和应用前景.

关键词 动态二进制翻译;动态二进制优化;精确异常;自修改代码翻译;翻译开销

中图法分类号 TP314

动态二进制翻译技术是一种即时编译技术[1-2],它将针对源体系结构编译生成的二进制代码(源机器码)动态翻译为可以在目的体系结构上运行的代码(翻译码).动态优化技术是指在运行时获取动态信息并进行代码优化的技术[3-4].使用这些技术,源软件可以直接在与源机器不兼容的机器上比较高效地运行(见图 1 所示).

动态二进制翻译与优化技术推动了计算机体系结构的发展[5-6].在传统计算机系统中,针对某体系结构编译生成的计算机软件直接运行在实现该体系结构的硬件之上.动态二进制翻译器利用软件的方式实现体系结构,分离了软件对硬件的依赖.利用这种技术,源软件不再直接运行在硬件之上,而是运行在动态二进制翻译优化系统之上.只要动态二进制

收稿日期:2006-01-09;修回日期:2006-05-29

翻译器所实现的指令集体系结构不变,就可以保证源软件的正确执行. 这给硬件的设计带来了空间,推动了计算机体系结构的创新和发展. IBM 的 AS400 利用动态翻译技术提供了一层虚拟的指令体系结构,使得其从原体系结构平滑过渡到 PowerPC 体系结构[7]. Transmeta 的 Crusoe 处理器[8]利用这种技术降低能耗.

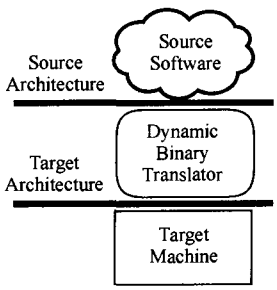

Fig.1 Dynamic binary translator.

图 1 动态二进制翻译器

动态二进制翻译优化器主要被用来保证目的体系结构对源体系结构的向后兼容[9-23]. 当一个主流体系结构被新的体系结构替代时,对所有源软件进行重新移植几乎不可能. 而且有些软件的源代码已经丢失(尤其是一些库),或者由于该源语言不再使用其编译器已经丢失. 即使对这些软件可以进行重编译和调试,这些遗产软件的移植和重优化非常费时. 人们通常利用动态二进制翻译优化器来解决这个问题[24]. 有代表性的系统有 Intel 公司的 IA-32 Execution Layer[19];HP 公司的 Aris[11];DEC 公司的 FX! 32[10];IBM 公司的 DAISY[20];Transmeta 公司的 CMS[9];Transitive 公司的 QuickTransit[15]等等. 此外,在源体系结构和目的体系结构一致的情况下,动态二进制翻译器也可以作为动态优化器在运行时优化二进制代码来加速软件运行[3-4,25]或加强计算机安全保护[26-27]. HP 公司的 Dynamo[28-29]以及 MIT 的 DynamoRIO[12-13]属于这一类系统中的代表.

本文描述了动态二进制翻译及优化系统的基本框架,并概述了几种比较有代表性的动态二进制翻译器;然后分析了动态二进制翻译器中关键技术难点;并比较了各个系统中对这些关键问题的不同解决方法;最后本文指出了该领域的研究热点以及对将来的展望.

1 动态二进制翻译器的基本框架

动态二进制翻译器包含翻译引擎与执行引擎[9,19-20]. 如图 2 所示,翻译引擎读入源软件的源机器码,并将其翻译为目标体系结构上的翻译码;执行引擎运行这些翻译码来模拟源软件的执行. 与此相对应,源软件在动态二进制翻译器的执行分两个基本阶段:翻译阶段与运行阶段. 在翻译阶段,翻译引擎对源机器码进行解码之后将其翻译为目标体系结构上的二进制代码. 动态二进制翻译器的翻译是即时的,它仅对源软件即将要执行的源机器码进行翻译. 它从当前指令开始解码,并且预先读取即将要执行的源机器码块,然后对这些源机器码块进行翻译. 一旦翻译完毕翻译引擎将控制转移至执行引擎,动态二进制翻译器便进入运行阶段. 执行引擎根据要执行的源指令地址来找到翻译码来执行它. 执行引擎可以使用各种手段收集动态轮廓信息[30-34]. 当程序运行至尚未翻译或频繁执行的代码片段,此时执行引擎便将控制交回翻译引擎继续翻译或再次优化翻译. 这样,动态二进制翻译器通过翻译与执行两个阶段的相互交替,可以在目标机上模拟执行源软件.

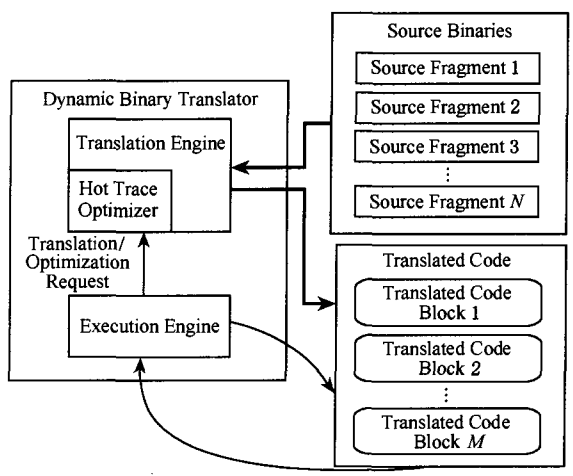

Fig.2 Basic framework of dynamic binary translator.

图 2 动态二进制翻译器基本框架

动态二进制翻译器以代码页为单位存放和管理翻译码块. 代码页有两种,一种是在已用页面池中,另一种是在空余页面池中. 当翻译码块产生时,翻译器首先将其放入已用页面池中的当前页面中. 如果在已用内存池中的内存页超过了一个预先设点的阈值,那么垃圾收集器就会将已用内存池中的一定数目的内存页移出至空余页面池中. 垃圾收集器试图收集那些将来最不可能执行的翻译码页面. 同时,垃圾收集的策略必须非常简单,因为复杂费时的操作可能带来不必要的开销. 如最近最少使用策略就对动态二进制翻译器不合适. 因为这种垃圾收集

算法试图回收最近最少使用的页面,它需要在翻译码被运行时不断更新它所在页面上的一个数据结构.通常,动态二进制翻译器使用一些非常简单的垃圾回收策略,如一次性回收所有页面[28]、回收最早被创建的一部分页面等等[35].

2 有代表性的动态二进制翻译系统

2.1 动态二进制翻译商用系统

Intel 公司的 IA-32® Execution Layer (IA-32 EL)[19,36-37]支持在 Itanium 体系结构直接运行 IA-32® 体系结构的应用程序. 这两种体系结构的指令集完全不兼容. 在 Itanium 体系结构推出之初,它曾用一种特殊的硬件来支持与 IA-32 的兼容性. 该硬件在支持 Itanium 体系结构的 Madison 处理器的芯核(CPU core)上占据了大约 20%的空间. 然而该硬件提供的性能并不令人满意,X86 代码在该硬件上的执行速度仅为它在源体系结构上的 20%～30%. IA-32 EL 大大提高了 X86 代码在 Itanium 上的运行速度. 在 IA-32 EL 上运行 SPEC Int2000,SPEC Fp2000 及 Sysmark2002 时,一台 1.5GHz 的 Itanium2 机器可以达到相当于 1.6GHz Xeon 处理器的 105%,99%及 133%. 由于 IA-32 EL 出色的性能表现,它作为一个软件系统成功地替换了为兼容 IA-32 而增加的特殊硬件.

Transmeta 公司推出的 Crusoe 处理系统[8-9]是一种动态二进制翻译器与硬件协同设计的系统. 它利用动态二进制翻译器 CMS (code morphing software)来简化底层硬件的设计,以达到降低处理器能耗的目的.CMS 的源体系结构为 X86 指令集体系结构,目标机为 Transmeta 开发的 VLIW 处理器. 与 IA-32 EL 不同,Crusoe 上的一切软件都运行在 CMS 之上,包括操作系统、BIOS、设备驱动程序等等. 与 IA-32 EL 相比,CMS 与硬件结合得更紧密,可以获得更多的硬件支持;同时,运行操作系统等程序也给 CMS 带来了新的问题和挑战.

DEC 公司的 FX! 32 系统[10,38-39]在 Alpha 处理器上实现了 IA-32 体系结构.FX! 32 是一个解释器与静态二进制翻译器相结合的系统. 严格地说,它不是一个动态二进智翻译系统. 它在程序第 1 次运行时对程序进行解释执行,同时收集程序的动态运行信息. 在程序第 1 次运行完之后,由系统中另一个后台进程对运行过的源程序片段进行静态二进制翻译. 在程序被多次运行的情况下,后台程序会对曾经翻译过的代码进行进一步的优化翻译.

Transitive 公司的 QuickTransit 系统[15]支持多个源体系结构和目标体系结构. 它的前端读入源体系结构指令,生成内部表示,针对内部表示进行优化,然后它的后端生成目标湾系结构指令. 它的内部表示和具体的源体系结构及目的体系结构都没有关系. 故此,通过修改它的前端和后端就可以实现一个新的二进制翻译器. 此外,通过将一个图像系统的函数调用映射成另一个图像系统的函数调用,QuickTransit 的图像功能映射器避免了二进制翻译,大大提高了系统的性能.

2.2 动态二进制翻译研究系统

DAISY 系统[20]起源于 IBM 对 VLIW 体系结构的研究[40]. 为了解决 VLIW 体系结构上软件缺乏的问题,DAISY 可以与流行的当前各种体系结构相兼容,如 PowerPC,X86,S/390,JVM 等.DAISY 运行在操作系统之下.DAISY 的 VLIW 体系结构是面向服务器市场的,它每节拍可以发射 4～16 条指令,拥有数 10 亿字节的内存.DAISY 保留了 100MB 的空间,以放置 DAISY 系统本身以及翻译码. 与之相比,Crusoe 的硬件仅支持发射 2～4 条指令,仅有 16MB 的内存. 所以在翻译优化上,DAISY 更注重挖掘翻译码的并行性[41],而 Crusoe 更偏向于如何节省翻译开销.

HP 的 Dynamo 系统[28-29]是一个动态二进制优化系统.Dynamo 的输入是 PA-RISC 代码,输出为优化后的 PA-RISC 代码.Dynamo 系统运行在操作系统之上,它在应用程序运行时对二进制代码进行优化以达到更好的运行效率.Dynamo 可以将某些 SPEC 标准程序的性能提高 22%,平均性能提高 10%.Dynamo 的性能提高主要来自利用运行时信息将代码重排序. 譬如,它可以将由间接跳转连接的两个块按顺序排放. 由于运行 Dynamo 的硬件不支持间接跳转预测及跨间接跳转的指令预取,这放大了代码重排序的效果. 之后 Dynamo 系统被 MIT 所扩展,产生了针对 IA-32 系统的 DynamoRIO[12].

Queensland 大学的 UQDBT 系统[23]可以根据用户对源体系结构和目标体系结构进行描述,将源体系结构上的指令动态翻译为目标体系结构指令. UQDBT 提供了 3 种描述语言供用户来描述二进制文件格式、机器指令格式、机器指令语义. 它利用好几级内部表示来完成指令的翻译,这使得它的翻译速度比其他系统较慢. 它平均需要用 18 万个时钟周期来翻译一条 IA32 指令到 sparc 指令. 维也纳大学

的 Bintrans 系统[14]也实现了 UQDBT 类似的功能. 它的体系结构描述文件只有一种,既表示指令格式也表示指令语义. 由于它没有采用内部表示,源体系结构的指令被直接翻译为目标体系结构指令,大大提高了翻译速度.

3 动态二进制翻译带来的挑战

动态二进制翻译器的目标是①提供与源机器兼容的执行环境;②使源软件能在目标机上高速有效地运行. 然而,兼容性与高性能是相互对立的两个方面. 在保证兼容性或者尽可能少地牺牲兼容性的前提下提高动态生成的翻译码的性能,这是动态二进制翻译器面临的主要挑战.

3.1 保证精确异常的优化调度

为了支持源软件的正确运行,动态二进制翻译器表现为一个虚拟的源机器. 动态二进制翻译器必须忠实地模拟源机器上的寄存器与内存状态和对这些状态的更新. 在任一时刻,如应用软件产生异常或在响应外部中断时,动态二进制翻译器必须提交一个与在源机器执行时完全一致的程序状态. 精确异常要求在一条指令发生异常时,该指令之前的任何指令已经执行完毕,而该指令之后的任何指令尚未执行. 在执行翻译码时,如果某条目的指令发生同步异常,翻译器需要保证与该目的指令对应的源指令维持精确异常的特性.

为了生成高效的翻译码,通常翻译器会在生成翻译码时进行优化调度. 这些调度可能改变翻译码中的指令序列,使得翻译码的顺序与源二进制码不同. 在翻译码中的指令发生异常时,如何定位与其相对应的源机器码中的指令,如何恢复出与之对应的源体系结构的状态,都是动态二进制翻译器必须解决的问题[42].

Crusoe 利用硬件的推测执行能力来支持精确异常[11]. 在 Crusoe 处理器中,对每一个映射寄存器都配有一个临时寄存器,同时还有一个受控内存缓存区. 在执行指令时,它更新临时寄存器或者写入受控内存缓存区. 当它执行完一个翻译码块之后,它会执行一个提交操作. 提交指令将这些临时寄存器上的值更新至映射寄存器,将受控内存缓存区的值写入映射内存. 如果在该指令执行之前,有一条指令发生了异常,或者有外来中断需要处理,那么这些临时寄存器和受控内存缓存区中的值将被作废,将状态恢复至上一条提交指令后将控制转移至 CMS 程序. 在发生异常的情况下,Crusoe 会从上一个提交点开始逐条指令解释执行直到发生异常的指令,然后调用 X86 异常处理程序. 有了这种硬件支持,精确异常对该翻译码块的调度基本上不再构成约束.

IA-32 EL 利用优化调度后的翻译码中的"提交点"和"异常点"支持精确异常[19]. 异常点是指翻译码中可能发生异常的指令. 每一个提交点可以对应多个异常点. 与异常点不同,提交点不是某一条具体的 Itanium 指令,而是在翻译码中一个无形的"障碍". 提交点对应于源机器码中的一条指令以及该源指令执行之前的源寄存器状态. 在翻译码中,映射寄存器状态的更新在提交点这一点提交. 所以,在提交点之前的对映射寄存器状态的更新不能调度至提交点之后. 在下一个提交点到来之前,如果存在对这些映射寄存器的更新,需要对这些映射寄存器进行备份. 这样,在与该提交点对应的某异常点发生异常时,IA-32 EL 可以从这些映射寄存器或专用保存寄存器中恢复出从该提交点所对应的源寄存器状态. 故此,本提交点与下一个提交点之间的指令可以比较自由地调度. 为了使得翻译码有更大的调度空间,IA-32 EL 将尽可能多的异常点对应于一个提交点. 第 1 个提交点在块首设置,然后每当遇到一个无法调度的异常点(写内存和跳转指令),或者当保存寄存器被用尽无法保证提交点完整的源寄存器状态之时设置一个新的提交点. IA-32 EL 利用纯软件的方法巧妙地实现了优化调度和精确异常.

3.2 模拟映射不同的体系结构

在理想情况之下,我们希望源体系结构上的所有寄存器都可以被目的寄存器所映射,源体系结构上的内存可以完全被目的机器提供的地址空间所涵盖,并且目的体系结构上的各运算部件的运算模型与源体系结构一致. 通常在动态二进制翻译器和处理器协同设计的系统中,这些条件是作为设计要求被提出的. 然而,对于源体系结构和目的体系结构都已经是客观存在的系统来说,通常这种条件并不成立. 如何设计高效的翻译算法来处理这种情况是产生高效翻译码的基础.

Crusoe 的 VLIW 体系结构虽然与 IA-32® 的超标量体系结构有很大的不同,然而 Crusoe 处理器内部的运算模型都在设计时尽量与 IA-32 的运算模型相近,以消除不必要的翻译开销. 例如,与 IA32 相仿,Crusoe 处理器拥有 FP, MMX, SSE, SSE2 运算部件及类似的寄存器结构. 这样,CMS 可以很高效

地用一条 Crusoe 处理器指令来模拟一个 IA32 指令.

然而,对 IA32EL 来说,由于 IA-32® 体系结构和 Itanium 体系结构是不同时期设计的相对独立的产品,它们存在很大的差异. IA32EL 必须处理这种由于源体系结构与目的体系结构不同引起的优化翻译问题. 首先,IA-32® 体系结构上的 EFLAG 寄存器会在每次整数运算时被更新,而在 Itanium 体系结构上并没有相应功能的寄存器. 此外,IA-32® 体系结构上的浮点运算部件为栈式结构,而在 Itanium 上为寄存器结构.X86 的浮点指令对栈顶进行操作,而这个栈顶所对应的物理寄存器是不固定的[19]. 再次,IA-32 EL 还必须处理 MMX 寄存器与 FP 寄存器共用物理寄存器的问题. 最后,IA-32® 体系结构中的 SIMD 运算部件中的寄存器(从 Pentium Ⅳ开始)可以支持并行整数、并行单精度浮点、并行双精度浮点,及各种标量数据类型. 然而,在 Itanium 上没有相应的寄存器能同时支持这些数据类[37].

3.3 优化翻译自修改代码

二进制代码可以在运行时修改自身. 一些较早开发的应用程序利用自修改代码来节省代码空间,或利用这种功能来加密. 对在操作系统之下的动态二进制翻译器,操作系统从磁盘数据调入内存页可以产生同样的问题. 被修改的代码块必须被重新翻译,以保证与最新的代码同步[19-20].

在一些体系结构上,程序在修改代码之后需要执行一条特殊的指令来保证指令缓存被更新,使得该修改生效,从而处理器能执行到被修改的代码. 对这种机器我们只需对这种指令进行特殊的处理,对与被更新缓存相关的代码块重新翻译即可. 然而另外一些体系结构,如 Intel 公司的 PentiumPro 机器,它利用硬件保证指令缓存的更新. 所以没有一条特殊的指令来提示某一代码块被修改. 在这种情况下,翻译码必须在执行时对源机器码的修改行为进行探测,并且即时对被修改的代码块进行重翻译.

Crusoe 通过多种手段来支持自修改代码[11]. 首先,它通过修改页面属性为只读来探测任何对该页面的写操作,以检测源机器码的自修改行为. 其次,在代码和数据同页的情况下,为了使得检测更为精确,CMS 支持小页面. 再次,对一些确实存在频繁的自修改的代码,它产生探测自修改的翻译码. 最后,Crusoe 还针对一些最常见的自修改模式做出了一些优化:如果修改的仅为本块中某指令的立即数部分,翻译码也可以直接修改翻译码中的相应立即数;保留所有的修改版本以及对应的翻译码,在运行时根据修改来调用相应的翻译码.

3.4 降低翻译开销

对计算密集的标准程序而言,动态二进制翻译器采用自适应翻译策略成功地降低了其翻译开销. 自适应翻译策略根据源二进制代码块的执行频度的不同,采用不同的翻译策略:一个源机器码块可能被多次翻译,在不同的阶段采用不同的翻译及优化手段. 由于对大部分程序而言,10% 的代码的运行占据了 90% 的执行时间. 自适应翻译策略对这部分关键代码块进行优化翻译,而对其他代码块进行比较简单的翻译或解释执行,从而获得了较小的翻译开销.

图 3 给出 CPU2000 在被动态翻译执行的时间分布图[19]. 如图所示,优化翻译码的执行时间为 95%,普通翻译码的执行时间为 1%,而翻译开销仅为 1%. 可见对 CPU2000 这一类计算密集的程序而言,优化翻译码的效率是动态翻译执行的关键.

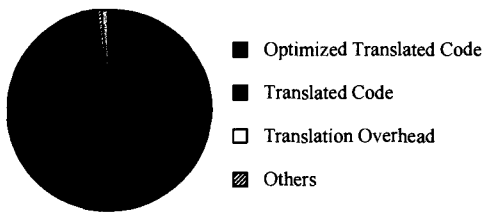

Fig.3 Time distribution of dynamic translation CPU2K benchmark.

图 3 动态翻译 CPU2K 时间分布

在运行其他非计算密集的实用程序时,动态二进制翻译器的翻译开销大幅度上升. 这些程序通常表现为热点代码不集中、执行的代码量较大等,导致动态二进制翻译器忙于不停的翻译之中. 此外,在大多数应用程序的启动过程时,动态二进制翻译器会遇到代码执行前的翻译高峰问题. 图 4 给出了 Sysmark 在被动态翻译执行的时间分布图[19]. 如图

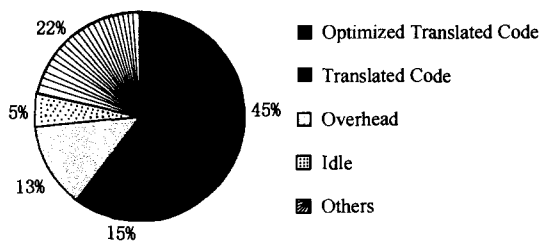

Fig.4 Time distribution of dynamic translating Sysmark benchmark.

图 4 动态翻译 Sysmark 时间分布

所示,优化翻译码的执行时间下降为60%,而翻译开销上升为13%.

动态二进制翻译器多采用复杂度低且有效的翻译算法.即使在高级的优化翻译阶段,类似静态翻译中的彻底分析也会被尽量避免.与这种彻底分析相反,许多翻译算法偏向于采用推测的方法:它假设某种条件成立,然后在这种条件下进行优化翻译;然后在运行时检测该条件是否成立,如果不成立则重新翻译该代码块.对信息的推测和假定可以简化翻译算法中最为费时的信息分析,从而降低翻译开销.

在Crusoe系统的设计中,推测执行作为一个总原则被贯穿始终[11].Crusoe提供硬件来支持推测执行和对执行结果进行检验.在推测执行失败的情况下,动态二进制翻译器首先得到控制,并做出相应的处理.如果一个翻译块反复出现推测执行失败,说明在翻译时的某些假设在运行时已不成立,于是这些块可以被重新翻译以适应最新的运行时情况.

IA-32 EL的翻译引擎只包含2个阶段:快速翻译阶段和优化翻译阶段.它在快速翻译阶段产生的普通翻译代码中插入一些特殊的指令,用这些指令来收集动态执行信息以指导进一步的优化翻译.为了降低实用程序的开销,IA-32 EL的翻译引擎来对实用程序的某些频繁被装载与卸载的"热"模块的翻译码进行重用,以避免反复翻译这些"热"模块[36].

3.5 利用动态信息优化翻译

动态二进制翻译器的优势在于利用动态信息来优化生成翻译码[43-44].对动态信息的利用贯穿在动态翻译系统的自适应翻译策略中.常见的利用动态信息优化的例子包括利用动态控制流信息来提高代码局部性与利用动态地址信息提高数据局部性.由于程序的大部分动态执行发生在一些热点路径之上,如果在执行时收集热点路径信息,将在热点路径上的代码块翻译后放在一段连续的空间,即可提高代码局部性.同样如果能够在动态时收集某一频繁执行的内存操作指令的地址信息,并分析其规律,那么可以对这些内存数据进行预取,来提高数据的存取速度.

动态控制流信息可以不同的方法获取.我们可以在解释执行阶段来收集动态信息,在翻译码中直接插入信息收集信息,或者利用硬件支持对各种处理器内部事件进行样本抽样分析来发现程序的动态执行特征.绝大多数的系统包括HP公司的Dynamo系统及Dec公司的FX32系统,都是在解释执行阶段来收集动态信息,并根据这些信息来进行提高代码与数据局部性的优化.Crusoe的CMS系统大量地运用异常来监视翻译码的运行状况,在必要的时候进行重翻译.在翻译码中插入动态信息收集指令,利用动态信息来处理非对齐数据,插入预取指令及其他优化.

4 动态二进制翻译的研究热点

商用动态二进制翻译器的出现标志着动态二进制翻译技术的成熟[11,19].由动态二进制翻译技术构造的虚拟机将成为计算机系统不可分割的一部分.越来越多的程序在动态二进制翻译器上运行.如何让这些程序更高效地运行为动态二进制翻译技术的进一步研究指引了方向.

如何让编译器与二进制翻译器相配合,使得编译器产生的二进制文件在二进制翻译器上运行地更好,这是目前二进制翻译技术的研究热点[45-47].研究硬件与二进制翻译器相配合使得二进制翻译器的翻译码能在硬件上更有效地运行是另一研究热点[6,11,20].此外,在动态信息的利用及优化方面,热点主要集中在如何利用低开销的硬件支持以及用更精确高效的算法来收集动态信息以生成更优化的翻译码[48-50].最后,二进制翻译器作为一种虚拟机,与其他虚拟机如何互相协调,共同配合分享处理器资源相互作用也是当前研究热点之一[48,51].

5 总结与展望

动态二进制翻译与优化技术推动了计算机系统的发展.首先,它分离了计算机软件与硬件的紧密关系,推动了处理器的发展和更新.其次,它作为一种某些处理器的虚拟机,解决了软件的移植问题.最后,它将编译器、处理器和二进制翻译器连接在一起,让计算机系统作为一个整体发挥其最佳性能.

随着信息技术的发展,计算机硬件和软件将作为一种通过互联网上获取的资源.动态二进制翻译和优化技术所实现的虚拟机功能将使得网络上的计算机资源按照用户需求来运行不同的体系结构上的软件.它还将被广泛应用于实现信息安全和保护,提高计算机性能等等[1].

参 考 文 献

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

[2] Michael Gschwind, *et al*. Dynamic and transparent binary translation [J]. Computer, 2000, 33(3): 54-59

[3] D Bruening, E Duesterwald, S Amarasinghe. Design and implementation of a dynamic optimization framework for windows [C]. The 4th Workshop on Feedback-Directedand Dynamic Optimization, Texas, Austin, 2001

[4] Thomas Kistler, Michael Franz. Continuous program optimization: Design and evaluation [J]. IEEE Trans on Computers, 2001, 50(6): 549-566

[5] Frank G Soltis. Inside the AS/400 [M]. Rahoster: Duke Press, 1996

[6] Michael Gschwind, Erik Altman. Inherently lower complexity architectures using dynamic optimization [C]. 2002 Workshop on Complexity Effective Design(VCED02), Anchorage, AK, 2002

[7] James E Smith, Ravi Nair. Virtual Machine: Verstaile Platforms for Systems and Processes [M]. San Francisco: Morgan Kaufmann, 2005

[8] Alexander Klaiber. The technology behind crusoe processor [OL]. http://www. transmeta. com/corporate/pressroom/whitepapers. html, 2000

[9] J C Dehnert, B K Grant, J P Banning, *et al*. The transmeta code morphing software: Using speculation, recovery, and adaptive retranslation to address real-life challenges [C]. The Int'l Symp on Code Generation and Optimization, San Francisco, California, 2003

[10] Anton Chernoff, Mark Herdeg, Ray Hookway, *et al*. FX! 32: A profile-directed binary translator [J]. IEEE Trans on Micro, 1998, 18: 56-64

[11] Cindy Zheng, Carol Thompson. PA-RISC to IA-64: Transparent execution, no recompilation [J]. IEEE Computer, 2000, 33(3): 47-52

[12] T Garnett. Dynamic optimization of IA-32 applications under dynamo RIO [OL]. http://www. cag. lcs. mit. edu/commit/papers/03/garnett-meng-thesis. pdf, 2003

[13] D Bruening, T Garnett, S Amarasinghe. An infrastructure for adaptive dynamic optimization [C]. The 1st Int'l Symp on Code Generation and Optimization (CGO-2003), San Francisco, California, 2003

[14] M Probst. Dynamic binary translation [OL]. http://www. ukuug. org/events/ linux2002/ papers/ pdf/ DynamicBinary Translation. pdf, 2002

[15] Transitive Corp. QuickTransit® hardware virtualization technology [OL]. http://www. transitive. com/, 2004

[16] Todd M Austin. A hacker's guide to the SimpleScalar architectural research tool set [OL]. http://www. simplescalar. com/docs/hack_guide_v2. pdf, 1996

[17] E Koukourechkov, N Grozdanov, G Gaydadjiev, *et al*. SCISM IA-32 binary translator [OL]. http://ce. et. tudelft. nl/publicationfiles/825_425_koukourechkov. pdf, 2003

[18] E Duesterwald. Design and engineering of a dynamic binary optimizer [J]. Proc of the IEEE, 2005, 93(2): 436-448

[19] L Baraz, T Devor, O Etzion, *et al*. IA-32 execution layer: A two phase dynamic translator designed to support IA-32 applications on Itanium(r)-based systems [C]. The 36th Annual Int'l Symp on Microarchitecture, San Diego, CA, 2003

[20] K Ebcioglu, E Altman. DAISY: Dynamic compilation for 100% architectural compatibility [C]. The 24th Annual Int'l Symp on Computer Architecture, Denver, CO, 1997

[21] Kevin Scott, Naveen Kumar, Siva Velusamy. Retargetable and reconfigurable software dynamic translation [C]. The 1st Int'l Symp on Code Generation and Optimization (CGO-2003), San Francisco, California, 2003

[22] Troger, Jens. Specification-driven dynamic binary translation [Ph D dissertation] [D]. Brisbane: Queensland University of Technology, 2004

[23] Cristina Cifuentes, Mike Van Emmerik, *et al*. Preliminary experiences with the use of the UQBT binary translation framework [C]. Workshop on Binary Translation, Technical Committee on Computer Architecture Newsletter, Newport Beach, USA, 1999

[24] C Cifuentes, V Malhotra. Binary translation: Static, dynamic, retargetable? [C]. Int'l Conf on Software Maintenance, Monterey, CA, 1996

[25] W Chen, S Lerner, R Chaiken, *et al*. Mojo: A dynamic optimization system [C]. The 3rd Workshop on Feedback-Directed and Dynamic Optimization, Monterey, California, 2000

[26] Kevin Scott, Jack Davidson. Safe virtual execution using software dynamic translation [C]. The 18th Annual Computer Security Applications Conference, Las Vegas, Nevada, 2002

[27] M Prasad, T C Chiueh. A binary rewriting defense against stack based buffer overflow attacks [C]. USENIX' 03 Annual Technical Conference, San Antonio, Texas, 2003

[28] V Bala, E Duesterwald, S Banerjia. Transparent dynamic optimization: The design and implementation of dynamo [R] Hewlett Packard Laboratories, Tech Rep: HPL-1999-78, 1999

[29] Vasanth Bala, Evelyn Duesterwald, Sanjeev Banerjia. Dynamo: A transparent dynamic optimization system [C]. The ACM SIGPLAN'2000, Vancouver, Canada, 2000

[30] Bob Cmelik, David Keppel. Shade: A fast instruction-set simulator for execution profiling [C]. The 1994 ACM SIGMETRICS Conf on Measurement and Modeling of Computer Systems, Nashville, Tennessee, 1994

[31] O Taub, S Schechter, M D Smith. Ephemeral instrumentation for lightweight program profiling [OL]. http://www. eecs. harvard. edu/machsuif//publications/publications. html, 2000

[32] T Conte, B Patel, K Menezes, *et al*. Hardware-based profiling: An effective technique forprofile-driver optimization [J]. International Journal Parallel Programming, 1996, 24: 187-206

[33] Thomas Conte, Burzin Patel, J Cox. Using branch handling hardware to support profile-drivenoptimization [C]. The 27th Annual Int'l Symp on Microarchitecture(Micro-27), San Jose, CA, 1994

[34] Thomas Conte, Kishore Menezes, Mary Ann Hirsch. Accurate and practical profile-driven compilation using the profile buffer. The 29th Annual Symp on Microarchitecture (Micro-29), Paris, France, 1996

[35] Kim Hazelwood, Michael D Smith. Generational cache mnagement of code traces in dynamicoptimization systems. The 36th Int'l Symp on Microarchitecture(MICRO-36-2003), San Diego, CA, 2003

[36] Jianhui Li, Peng Zhang, Orna Etzion. Module-aware translation for real-life desktop applications [C]. The 1st ACM/USENIX Int'l Conf on Virtual Execution Environments, Chicago, IL, 2005

[37] *Jianhui Li, Qi Zhang, Shu Xu, et al*. Optimizing dynamic binary translation for SIMD instructions [C]. The Int'l Symp on Code Generation and Optimization, New York, 2006

[38] R Hookway. DIGITAL FX! 32 running 32-Bit x86 applications on alpha NT. IEEECOMPCON 97, San Jose, CA, 1997

[39] R J Hookway, M A Herdeg. Digital FX! 32: Combining emulation and binary translation [J]. Digital Technical Journal, 1997, 9(1): 3-12

[40] K Ebcioglu, J Fritts, S Kosonocky, *et al*. An eight-issue tree-VLIW processor for dynamic binary translation [C]. Int'l Conf on Computer Design. VLSI in Computers and Processors, Austin, TX, 1998

[41] K Ebcioglu, *et al*. Execution-based scheduling for VLIW architectures [G]. In: Proc of Europar99, Lecture Notes in Computer Science 1685. Berlin: Springer Verlag, 1999. 1269-1280

[42] Michael Gschwind, Eric R Altman. Precise exception semantics in dynamic compilation [C]. The Symp on Compiler Constructions, Grenoble, France, 2002

[43] L Anderson, M Berc, J Dean, *et al*. Continuous profiling: Where have all the cycles gone [C]. The 16th ACM Symp of Operating Systems Principles, Saint-Malo, France, 1997

[44] X Zhang, Z Wang, N Gloy, *et al*. System support for automatic profiling andoptimization. The 16th ACM Symp on Operating Systems Principles, Saint-Malo, France, 1997

[45] Ana Azevedo, Alex Nicoloau. An annotation-aware Java virtual machine implementation [C]. ACM SIGPLAN 1999 Java Grande Conference, San Francisco, 1999

[46] Fu-Hwa Wang. Compiler annotation for binary translation tools [G]. US Patent Application. US2003/0088860 A1, 2003

[47] B Grant, M Mock, M Philipose, *et al*. Dyc: An expressive annotation—Directed dynamic compiler for C [R]. University of Washington, Tech Rep: UWCSE-97-03-03, 2000

[48] Suresh Srinivas. Modularity, hardware based profiling, and mixed ISA execution within managed runtimes. Invitational Workshop on the Future of Virtual Execution Environments, New York, 2004

[49] Rahul Joshi, Michael Bond, Craig Zilles. Targeted path profiling: Lower overhead path profiling for staged dynamic optimization systems [C]. The 2nd IEEE/ACM Int'l Symp on Code Generation and Optimization (CGO), Palo Alto, California, 2004

[50] Michael D Bond, Kathryn S McKinley. Practical path profiling for dynamic optimizers [C]. Int'l Symp on Code Generation and Optimization (CGO-2005). San Jose, California, 2005

[51] Yuting Zhang, Azer Bestavros, Mina Guirguis, *et al*. Friendly virtual machines: Leveraging a feedback-control model for application adaptation [C]. The 1st ACM/USENIX Int'l Conf on Virtual Execution Environments, Chicago, IL, 2005

Li Jianhui, born in 1974. is a Ph. D. candidate in Fudan University as well as a senior software engineer in Intel China Software Center. His main research interests include parallelizing compilation, dynamic translation and optimization.

李剑慧,1974 年生,博士研究生,工程师,主要研究方向为并行化编译、动态翻译和优化.

Ma Xiangning, born in 1976. Ph. D. Senior software engineer in Intel China Software Center. Her main research interest is binary translation.

马湘宁,1976 年生,博士,工程师,主要研究方向为二进制翻译.

Zhu Chuanqi, born in 1943. Professor and Ph. D. supervisor of Fudan University, senior member of China Computer Federation. His main research interests are parallel processing and parallelization compilation.

朱传琪,1943 年生,教授,博士生导师,中国计算机学会高级学会,主要研究方向为并行处理和并行化编译.

Research Background

This paper is an overview of dynamic binary translation and optimization. Dynamic binary translation is just-in-time (JIT) compilation from the binary code of one architecture to another. The technology enables the application compiled for source architecture running on top of target architecture without recompilation. Dynamic optimization is run-time improvement of code. This paper begins with the basic framework of dynamic binary translator, and then gives an overview of several leading dynamic binary translators. Following that, the paper focuses on the key challenges of the dynamic binary translator, including supporting precise exception in optimized code, mapping source architectural context to target architectural context, translating self modifying code, reducing translation overhead, and dynamic optimization using profiling data. The paper has a deep discussion about these challenges and describes how they are addressed in different dynamic. The paper ends with the hot research topics and possible usage models of the dynamic binary translation technology.