Linux 下优化编译单元的程序调用共享连接技术*)

芦运照　张兆庆

（中国科学院计算技术研究所体系结构室　北京100080）（中国科学院研究生院　北京100080）

摘　要　本文分析了 LINUX 下的动态连接和静态连接，默认的动态连接方式在一定程度上限制了编译器的优化，而静态连接方式又有很多不足。本文在 ORC 编译器中提出了一种程序调用共享连接技术，对程序编译单元应用冗余的代码优化，并且展望了这种优化技术的未来研究，具有很强的实际意义。

关键词　ORC(开放源码研究编译器)，调用共享连接，编译单元，主程序

CALL_SHARED Linkage for Compile Unit Optimization under Linux

LU Yun-Zhao　ZHANG Zhao-Qing

(Institute of Computing Technology, Chinese Academy of Science, Beijing 100080)

(Graduate School, Chinese Academy of Science, Beijing 100080)

Abstract　Dynamic(shared) and static linking modes under LINUX are introduced in this paper. While default dynamic linking limits some optimizations of compiler, static linking has its weakness; so the paper developes a CALL_SHARED linking mode in ORC compiler, which can help optimize redundant code for compile unit, and it's beneficial for future research, such as gp-relative data promotion.

Keywords　ORC(Open Research Compiler), CALL_SHARED link, Compile unit, Main program

1　引言

一个应用程序一般要经过编写、编译、连接、加载和运行这样五个阶段，这五个阶段对程序的操作处理是严格按照时间序的，它们之间一方面是相互独立的，另一方面又有紧密的联系。Linux 系统使用 ELF 文件格式，支持两种连接方式：静态连接和动态连接；在程序的生成过程中，由于一些公用代码需要反复使用，就把它们预先编译成目标文件并保存在"库"中，形成库文件。当库文件与用户程序的目标文件连接时，连接器得从库中选取用户程序需要的代码，然后复制到生成的可执行文件中。这种库称为静态库，相应的连接方式称为静态连接，其特点是可执行文件中包含了库代码的一份完整拷贝。显然，当静态库被多个程序使用时，磁盘上、内存中都是多份冗余拷贝。而使用动态连接库和动态连接就克服了这个缺陷，当它与用户程序的目标文件连接时，连接器只是作上标记，说明程序需要该动态连接库，而不需要把库代码复制到可执行文件中；仅当可执行文件运行时，加载器根据这个标记，检查该库是否已经被其它可执行文件加载进内存。如果已存在于内存中，不用再从磁盘上加载，只要共享内存中已有的代码即可。这样磁盘、内存中始终只有一份代码，较静态库为优。

使用静态连接生成的程序占用较多的内存空间，运行时直接加载可执行文件，从一定程度上减少了系统调用，并且能够提高单个程序的运行性能，然而程序的运行会不断地增加冗余代码，加重了垃圾回收，影响了系统的整体性能；相比之下动态连接生成的程序就有更多的优点，减少了内存中的冗余代码，程序执行时不被激活的系统调用在运行时不必加载，减少了系统开销。

一个 C 语言源文件就是一个编译单元，经过编译后将生成一个目标文件，LINUX 系统的目标文件使用地址无关代码 PIC(position independent code)，即通常所称的浮动代码，由于动态连接都假定每一份代码来自不同的地址空间，它们对应于 GOT(Global Offset Table)表中不同的表项，在 IA-64 的 ABI 中 GOT 表项的当前值是用 gp 寄存器来表示的。在每一个函数调用处，编译器会生成一小段代码保存当前 gp 寄存器，而在函数返回处，编译器也会生成一小段代码恢复 gp 寄存器，以便当前函数能够正确执行。如果函数调用发生在一个编译单元内部，也就是在同一个地址空间，它们处在 GOT 表的一个表项所指的首地址，这样编译器为函数之间的调用生成的一小段代码初始化和恢复 gp 寄存器就是冗余的了，因为它们在同一段地址空间，共享同一个 gp 首地址。

在本文里提出程序调用共享连接技术，并在此基础上实施新的优化技术，解决了上述的代码冗余问题，在面向 IA-64 体系结构的编译器 ORC 中的实验证明这种技术能够提高程序性能。本文第1节介绍 LINUX 系统下的连接技术，并由此提出程序共享连接技术；第2节介绍程序共享连接技术及其在 ORC 编译器中的实验；第3节介绍在使用程序共享连接下使用的优化方法；最后通过在 IA-64 体系结构上的实验来证明这种技术及其优化效果。

2　程序调用共享连接技术

LINUX 系统下的动态连接在编译器中是被设为缺省连接方式的，这样相应的编译优化也就基于动态连接方式，对程序的编译单元的分开编译会假定每一处函数调用将跨越一个地址空间，使用不同的 gp 首地址，而 ABI 规定在函数调用处必须保存和恢复 gp 寄存器。为了在编译阶段获得更多的优化机会，我们需要改进动态连接方式，提高程序分析以及对编译

*)本课题得到国家863计划软件重大专项(2002AA1Z2104和2001AA111061)的支持。芦运照　博士生，研究方向是先进编译技术。张兆庆　研究员，主要研究方向是先进编译技术及相关工具环境。

单元的编译优化，识别函数调用处是否跨越不同的地址空间、使用不同的 gp 首地址；LINUX 系统下的动态连接方式为 SHARED，我们在这里提出一种适合动态连接的程序调用共享连接技术 CALL_SHARED。

2.1 相关述语

二进制文件(binary)是一个可执行的目标文件，包括动态共享目标文件(*.so 的后缀)和传统的可执行文件(如 a.out)；符号表(symbol table)是记录程序的数据和函数名称、类型、范围、地址信息等的数据结构；抢占(preemptible)是指一个符号在运行时可能被修改成不同的地址或含义；保护(protected)是指一个符号在本编译单元外可见但不可修改；输出(export)是指在本编译单元或模块定义的符号能够在另外一个编译单元或模块引用；输入(import)是指在本编译单元或模块引用另外编译单元或模块定义的符号。

2.2 问题描述

ABI 规定在函数调用处必须保存和恢复 gp 寄存器，是因为如果被调用者跟调用者在不同的二进制文件中，gp 寄存器就可能指向不同的首地址。如果在函数调用处调用者和被调用者基于同一个 gp 首地址，那么此时对 gp 寄存器的保存和恢复就冗余了；我们需要删除冗余的指令以提高程序性能，这样的优化机会重点是我们如何来更准确地识别哪些函数调用是发生在同一 gp 首地址的地址空间。

2.3 优化技术

在动态连接方式下，对于一个全局的符号编译器会把其属性设置为可抢占，如果函数符号是可抢占的，那么对它的调用是一定要保存和恢复 gp 寄存器的；而对于一个调用发生在同一个编译单元，而被调用者是一个本编译单元的全局强定义，这时的调用是一定发生在同一个 gp 首地址空间，因此我们提出了程序调用共享连接技术，这样可以更准确地分析出全局的符号定义是否可抢占。表1和表2对比了动态连接与程序调用共享连接。

表1

	SHARED	CALL_SHARED
编译单元	编译单元可能是动态库(*.so)	编译单元是主程序(main)的一部分

表2

函数符号定义	SHARED	CALL_SHARED
Xdef(全局、已定义)	可抢占	保护
XUdef(全局、未定义)	可抢占	可抢占
Static(静态定义)	局部	局部
WXdef(全局、弱定义)	可抢占	保护
WXUdef(全局、弱未定义)	可抢占	可抢占
Wstatic(静态弱定义)	无	无

从表1中可以知道程序调用共享连接技术的提出是为编译一个主程序及其相关编译单元，即要生成一个带 main 的可执行二进制文件，而不能生成一个动态库文件；表2列出了函数符号在使用动态连接和程序调用共享连接两种方式下编译时的属性，对于一个未定义的输入函数符号，总是可以被抢占的，而局部的函数符号是不可抢占的，对局部函数的调用一定发生在同一个 gp 首地址空间；对于本编译单元已定义的全局函数符号，程序调用共享连接方式下的编译设置其为受保护的，即不被抢占；这是因为对一个在本编译单元已定义的函数的调用在连接或者运行时都不会被动态库文件中相同的函数符号定义抢占，而动态库文件中的函数符号定义的优先级是最低的。

对函数调用处的优化技术是根据函数符号的输出(EXPORT)规则来应用的，如果函数符号是不可抢占的，那么对该函数的调用就是发生在同一 gp 首地址所指的空间，可以优化掉冗余的 gp 寄存器的保存和恢复；ORC 编译器在应用了程序调用共享连接技术后，编译阶段可以对编译单元的优化取得明显的效果。

3 编译单元进行优化实例

对一个主程序来说，调用动态库文件中的函数或者不在同地址空间的函数时，gp 寄存器记录着当前地址空间的首地址，当发生函数调用程序跳转到一个新的地址空间时，gp 寄存器就指向当前地址空间的首地址，为了保证函数返回时程序的控制能回到调用前的地址空间，程序在函数调用前必须保存 gp 寄存器，而在调用返回后必须恢复 gp 寄存器的值，保证程序运行的正确性。程序调用共享连接技术的提出是帮助分析发生函数调用时调用者和被调用者是否在相同的 gp 寄存器所指的地址空间，从而优化冗余的 gp 寄存器的保存与恢复。

这里来看一个例子，如图1所示，这是从 SPEC2000 基准程序中选取的例子 BZIP2，编译单元 bzip2.c 中定义了函数“sortIt”和“panic”，因而“sortIt”调用“panic”发生在同一个以 gp 首地址的地址空间，所以对 gp 寄存器的恢复也就是冗余的了。由于在动态连接方式下，编译器会把全局函数符号“panic”设置为可抢占属性，所以对 gp 寄存器的恢复是被认为必须的。

```
.BB75_sortIt:///0xbba0
 {.
        ld8r78=[2];;
        adds r78=736,r78
        br.call.sptk.many b0=panic#;;
 }
//Block:76 Pred:75 Succ:77
.BB76_sortIt://0xbbc0
 {.mib
        ld8.s r24=[r50]
        mov gp=r57
        br.few.Lt_32_306;;
 }
```

图1 发生在一个编译单元内的调用

在 ORC 编译器中提出并应用程序调用共享连接技术之后，对编译单元 bzip2.c 的编译优化之后减少了相应的 gp 寄存器的保存与恢复指令，整个编译单元 bzip2.c 中，gp 寄存器的保存和恢复指令数从344 减少到131。

4 实验结论

程序调用共享连接技术的提出给编译阶段的优化提供了帮助，在 ORC 编译器中的实现给程序的性能带来了明显的改进。ORC 编译器是中科院计算所与英特尔公司合作研发的开放源代码研究编译器，面向新一代 IA-64 体系结构设计，目前被国内外众多知名大学、研究机构、公司采用作为编译技术研究平台。本文所收集的实验数据选取了标准测试基准程序 SPEC2000 的12 个定点程序，使用了 LINUX 交叉环境，编译端是 IA-32 的计算机，在 LINUX 下安装了 IA-64 的 NUE 模拟环境，在 NUE 模拟环境下编译生成可执行代码，然后在 ITANIUM2 计算机上运行测试，ITANIUM2 计算机有3 个

833MHz 的 CPU、1G 内存，16K L1 D cache 和 16K L1 I cache。

优化的 gp 指令数——对 SPEC2000 基准程序的12定点程序的测试从图2中可以看出，ORC 编译器使用 O3 优化级别，经过程序调用共享连接技术优化后，对 gp 寄存器的保存和恢复指令数比原始数据都有不同程度的减少。

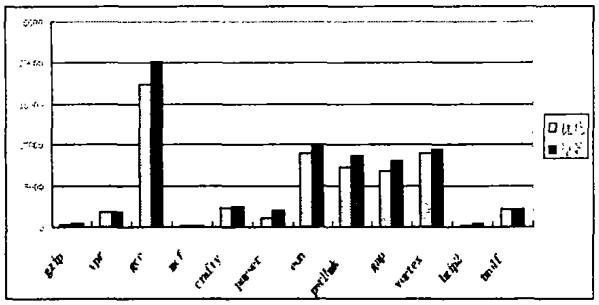

图2 程序共享连接技术优化的指令数

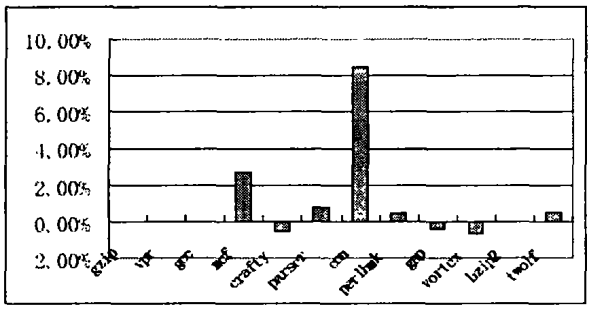

图3 程序共享连接技术性能加速比

O3 优化的性能改进——相对于图2来说，图3显示了这些定点程序经过 ORC 编译器在 O3 优化级别编译生成的二进制代码在 ITANIUM2计算机上测试运行的性能，这些程序的性能多数变好，最好的改善达到8.5%，一些程序的性能没有变化，而有少数程序的性能会有微弱的下降，这是因为一方面跟计算机运行的不稳定性有关，另一方面汇编代码的指令数的减少如果在比较小的基本块里，会导致跳转指令(br)的距离变近，影响了 CPU 的指令跳转预测，同时这里的指令减少又没有减少机器周期数，我们考虑到程序的总体性能有明显的改进，平均性能加速比达到1 个百分点，这对作为编译器的一个优化来说是非常可观的，特别对一个先进的编译器来讲，各种优化方法已经有了相当充分的研究。

在程序调用共享连接技术下，我们对函数符号属性的设置从而达到优化编译单元的冗余代码的目的，我们知道一个程序的符号表包括函数符号和数据符号，我们的研究计划一方面是研究函数调用的冗余代码删除，另一方面就是对数据变量的研究，因为考虑到如果一个全局数据变量符号是不可抢占受保护的，那么我们可以改善变量的布局，优化变量的存取，改进存储优化，对存储的优化是我们很感兴趣的方向，数据变量的布局优化对程序的性能的提升将有更大的帮助。

参 考 文 献

1 Intel Corporation. Intel® IA-64 Architecture Software Developer's Manual, Jan. 2000

2 Levine J R. Linkers and Loaders. Morgan Kaufmann Publishers, 2000

3 Intel Corporation. IA-64 Software Conventions and Runtime Architecture Guide, Jan. 2000

4 Intel Corporation. UNIX System V Application Binary Interface, Jan. 2000

5 Intel Corporation. Intel® Itanium®2 Processor Reference Manual, June 2002

6 Linux 下的动态连接库及其实现机制. http://www.antpower.org/

7 Aho A V, Sethi R, Ullman J D. Compilers: Principles, Techinques, and Tools. Addison Wesley, Pearson Education, 1986

(上接第134页)

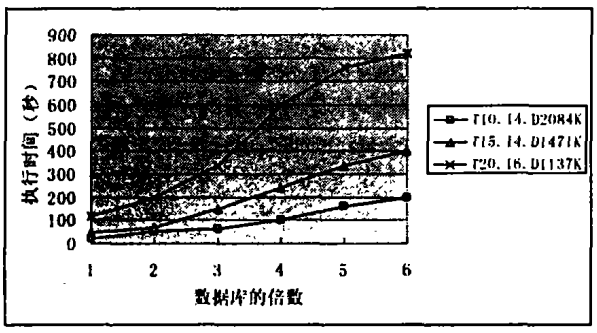

图4 处理机配置固定时，在不同数据库规模下算法的执行时间

结束语 本文提出了一个并行挖掘最大频繁项集的新算法 P-MinMax，它采用数据库的垂直表示和基于前缀关系的等价类划分，以等价类长度的指数函数为权值，并利用因子项集的完全包含关系在处理机之间贪心分配等价类，根据等价类的需要相应地划分和复制数据库记录，使各处理机得以异步计算，达到了较好的负载平衡、较高的剪枝效率和较少的数据库记录复制，减少了算法的执行时间。分析和实验表明，P-MinMax 有较好的可扩展性，其性能优于已有同类算法。

参 考 文 献

1 Burdick D, Calimlim M, Gehrke J. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In: Proc. of 17th Intl. Conf. on Data Engineering, Heidelberg, Germany, April 2001. 443～452

2 Gouda K, Zaki M J. Efficiently Mining Maximal Frequent Itemsets. In: Proc. of 2001 IEEE Intl. Conf. on Data Mining (ICDM'01), San Jose, California, November 2001. 163～170

3 Wang Hui, Li Qinghua, Ma Chuanxiang, Li Kenli. A Maximal Frequent Itemset Algorithm. Lecture Notes in Computer Science, Springer, 2003, 2639: 484～490

4 Agrawal R, Shafer J C. Parallel Mining of Association Rules. IEEE Transaction On Knowledge And Data Engineering, Dec. 1996, 8(6): 962～969

5 Zaki M J, Parthasarathy S, Ogihara M, Li Wei. New Parallel Algorithms for Fast Discovery of Association Rules. Data Mining and Knowledge Discovery: An International Journal. special issue on Scalable High-Performance Computing for KDD, Dec. 1997, 1(4): 343～373

6 Zaki M J. Parallel and Distributed Association Mining: A Survey. IEEE Concurrency, 1999, 7(4): 14～25