面向 IXP 网络处理器的内联优化*)

汤 伟 吴承勇 张兆庆
(中国科学院计算技术研究所 北京 100080)

摘 要 内联优化是一种有效的编译优化技术,它通过将函数体直接嵌入到调用点来消除函数调用开销。然而,网络处理器特殊的体系结构对内联优化提出了新的要求,需要新的技术辅助传统内联优化来更好地适应这种特殊的体系结构。本文描述了如何利用关键路径提取技术和迭代编译技术对传统内联优化技术进行扩充和改造,来更好地适应IXP体系结构。实验数据表明,改进后的内联优化能够有效地提高网络系统的性能。

关键词 内联优化,IXP网络处理器,编译器优化

Function Inlining for IXP Network Processor

TANG Wei WU Cheng-Yong ZHANG Zhao-Qing
(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

Abstract Function inlining has been proposed as an efficient compiler optimization technique. It expands a function call site into the actual implementation of the function. This reduces overhead associated with the function call. However, current inlining techniques do not meet the demands of network processors, e.g. Intel's IXP series. The novel architecture features of IXP call for new techniques for adapting the traditional function inlining. This paper presents how to extend function inlining technique for IXP architecture using critical path extraction and iterative compilation. The experimental result shows that the extended function inlining can effectively improve network system performance.

Keywords Function inlining, IXP network processor, Compiler optimization

1 引言

网络处理器是专门针对网络包处理而设计的处理器,它的特点是速度快和可编程,兼顾了ASIC(Application Specific Integrated Circuit,专用集成电路)硬连电路的高效性和通用处理器编程的灵活性。Intel公司的IXP2400是网络处理器的一个代表,它包含Xscale和微引擎两种可编程处理器。其中Xscale是一个通用处理器,用于处理较少出现的复杂操作,微引擎是专门用于快速包处理的处理器。为了满足网络程序高速处理能力的要求,网络处理器具有特殊的体系结构特征,这些特殊的体系结构特征对传统的编译优化技术提出了新的要求。

为了编写高效的网络处理程序,程序员需要了解底层硬件结构,这使得编写网络应用程序变得复杂且易出错。此外由于程序对底层硬件的依赖,导致编写的程序不利于移植,增加程序重用维护的成本。为此,中科院计算所和Intel公司合作开发了一个针对IXP网络处理器的编程环境,旨在提高网络应用程序的开发效率和可移植性。在该编程环境中,底层的体系结构特性对程序员来说是透明的,程序员只需要描述网络应用程序的功能,最终的资源分配以及程序的优化由编译器和运行时系统来完成。

内联优化是一种通过在调用点嵌入被调用函数来消除函数调用开销的编译优化技术。通过内联优化能够增加函数体的指令数,这为其他优化提供更多机会。IXP微引擎指令存储区所能存储的指令数目非常有限(IXP2400只能存储4k条指令),这需要保证内联优化后的代码不超过指令存储区的大小限制;此外,IXP对函数调用没有提供硬件支持,需要用软件实现,对于高速的网络处理设备来说代价较大。为此,需要编译器在不超过指令存储区限制的情况下,尽可能通过内联消除函数调用的开销。本文描述了如何利用关键路径提取技术和迭代编译技术来支持和改造传统内联优化,更好地适应IXP网络处理器特殊体系结构的要求。

本文后面的内容组织如下:第2节介绍IXP编程环境;第3节介绍针对IXP网络处理器的内联优化技术;第4节给出相应的实验数据;最后总结整个工作。

2 IXP编程环境

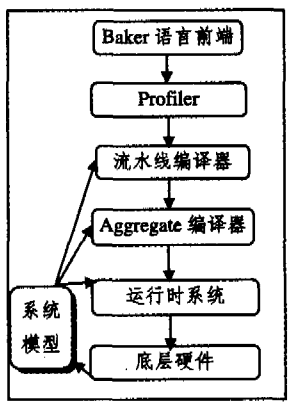

图1 IXP编程环境

图1给出了IXP编程环境结构图,它由6部分组成:网络程序设计语言Baker、Profiler、流水线编译器、Aggregate编译

*)基金项目:863软件重大专项(2004AA1Z2200)课题"高性能编译系统"、Intel公司资助合作项目。汤 伟 硕士生,研究方向为编译技术;吴承勇 副研究员、硕士生导师,研究方向为编译技术;张兆庆 研究员、博士生导师,研究方向为编译技术。

器、运行时系统和底层硬件组成。

Baker语言是IXP编程环境中所定义的高级语言。一方面，它是一种模块化编程语言，与C语言的语法比较类似，使得我们可以把网络程序组织成用若干包处理函数的形式。另一方面，它又是一种面向网络数据流和包处理的专用语言，直接支持网络程序中常用的数据结构。Baker采用用户所熟悉的面向过程的编程语言形式，同时隐藏网络处理器的硬件细节(比如多核、多线程、多存储器等)。Baker简化了网络程序的开发，同时使得用Baker开发的程序可以移植到不同的网络处理器平台。

包处理函数(PPF，Packet Processing Function)是Baker中执行网络数据包处理的基本功能单元，包含多个输入输出通道以及相应的包处理代码。包处理函数具有原子性，即一个包处理函数要么运行在微引擎上要么运行在Xscale上，不会同时被拆分到多个处理器。

包处理函数之间的通道可以被看作一个先进先出的队列。数据通过通道流入(输入通道)或者流出(输出通道)包处理函数。输入输出通道相互绑定，形成数据流处理路径。

profiler阶段的主要功能是通过仿真来收集代码和数据结构的profiling信息，比如数据结构的存储性质、包处理函数的执行频率、包处理函数之间的通信量、循环次数、分支跳转的频率等等。收集到的profiling信息被标示到中间表示上，利用收集到的profiling信息可以更好地指导后续流水线编译器阶段进行资源分配。

流水线编译器所要解决的问题是如何组织应用程序来有效地使用硬件系统的各种资源。首先，它使用数据结构的profiling信息显式地管理存储层次；其次，流水线编译器将多个包处理函数合并划分到相应的流水级，每个流水级被称为一个aggregate，并为每个aggregate分配硬件资源。

aggregate编译器为每个aggregate生成可执行目标代码。aggregate编译器根据流水线编译器分配的不同处理器类型，生成对应于该处理器指令集的目标代码。很多与体系结构相关的优化都是在aggregate编译阶段完成的。

运行时系统包含一套系统管理工具，通过这些工具运行时系统可以完成加载和运行aggregate，监控流量变化、系统性能、能量消耗，并进行相应资源调整等任务。此外，运行时系统还包含一个资源抽象层，它为各种硬件资源向外提供高层接口，从而隐藏了它们的实现细节。

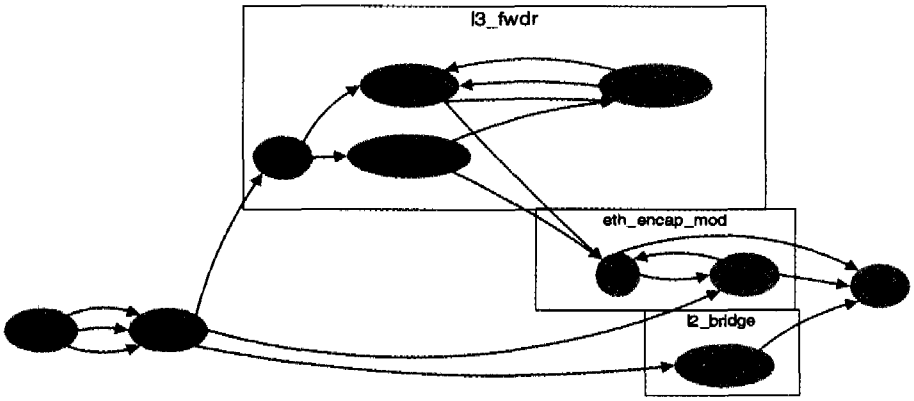

图2　13-switch的数据流图

3　针对IXP网络处理器的内联优化

3.1　介绍

函数内联优化通过消除函数调用开销来提高整个程序性能的编译优化技术，它通常是以代码膨胀为代价。前面提到过IXP不提供函数调用硬件支持，这导致函数调用开销对高速网络处理来说代价较大。此外，IXP指令存储区所能容纳的指令数有限，这限制了能够内联的函数数目。IXP体系结构特性导致传统的内联优化已经无法适应其要求，需要设计、扩充针对IXP特定体系结构的内联优化技术，来满足网络处理程序对性能的要求。为此，我们对传统的内联优化结构进行了扩充，加入了关键路径提取技术和迭代编译技术。通过这两个技术能够减轻传统内联优化导致的代码膨胀问题，充分利用内联优化带来的好处。

3.2　关键路径提取技术

加入关键路径提取技术是为了将多个包处理通路共享的任务进行分离。图2是根据Baker语言写的l3-switch程序(一个第三层交换程序)的数据流图，图中lpm_lookup处理结点有三个输入箭头，表明lpm_lookup需要处理三个不同通路上的处理任务。其中icmp_processors发送过来的包较少，属于负载较轻的处理通路。因此icmp_processors包处理函数在流水线编译器划分流水级的时候，将被分配到Xscale上执行。但是从l3_cls到lpm_lookup的处理通路上的负载较重，导致l3_cls将被分配到微引擎上执行。从lpm_lookup包处理任务来看，它同时处在负载不同的多条处理通路上。为了保证重负载通路的执行速率，lpm_lookup将被分配到微引擎上执行，但lpm_lookup能够接收icmp_processors处理任务发送过来的包。这样虽然实现了任务共享，但是却为内联优化带来了问题。因为这个包处理任务由多个处理通路共享，当编译器在一个通路上进行函数内联时，必须为其他通路保留一份副本，这样导致了代码的膨胀。由于IXP微引擎指令存储区的限制，代码膨胀意味着在同一个微引擎上所能包含的处理任务变少了，完成相同处理功能需要多个微引擎，在微引擎数目一定的情况下(IXP2400有8个微引擎)，整个网络应用的并行度降低了。为此，我们提出采用关键路径提取技术将负载重的一条关键路径从原来的网络处理程序通路中分离出来。由于分离之后只存在一个调用点，在进行内联的时候不需要为其他通路保留副本，也就不会导致代码膨胀。

3.3　迭代编译技术

引入迭代编译技术的主要目的是为内联优化提供较准确的函数体大小的数据。传统的内联优化是根据函数体的中间表示估算函数体大小，用来指导内联分析。由于内联分析在后端优化之前实现，而编译器后端实现了很多优化，这些优化导致最终生成的代码实际大小和估算值可能相差较大。对于

通用处理器来说，由于其指令存储区较大，对于这种估算产生的误差并不敏感。然而网络处理器由于其指令存储区大小的限制，如果误差较大，可能导致最终生成的指令数超过存储区限制，导致编译失败。

在 IXP 编译优化中，我们通过迭代编译保证在函数内联的时候能够得到较准确的函数体大小。在进行内联优化之前，编译器将先进行一遍不含内联的编译，在代码发射前收集各个函数体的指令数，并将收集的信息反馈到下一次编译过程，用于指导内联优化。在第二遍编译时，利用第一遍的反馈信息指导实际的内联优化。

表 1 有无迭代编译时函数体大小比较

函数名	函数体大小（估算）	函数体大小（迭代）	误差(%)
ops_process_incoming	152	143	6.3
app_l2_cls_process	157	143	9.8
ilm_process	57	50	14
ops_process_mpls	218	206	5.8
ops_update_ttl	57	80	28.8
eth_encap_process	117	124	5.6

表 1 给出了一个实际的网络应用程序在缺省编译选项情况下，开关迭代编译时函数体大小的比较。表中第三列给出了开关的实际误差值，从中可以看出估算的数据和反馈数据之间还是存在较大的误差。并且，随着后端优化选项选择的不同，这种误差会随之改变。由于迭代编译是通过多遍编译来收集反馈信息，因此后端编译选项的变化对函数体大小的影响能够通过迭代编译反馈到第二遍编译阶段，所以相应的内联优化能够更加激进。

3.4 内联优化的实现

关键路径提取和迭代编译都是辅助内联的准备工作，本节所要介绍的是实际的内联过程，其中包括如下几个主要部分：

a)建造函数调用图。函数调用图是一个反映函数调用关系的有向图，图中的结点表示每个函数，边表示函数间的调用关系，箭头从调用者指向被调用者。

b)执行内联分析。如果使用迭代编译，在内联分析的时候需要首先读入上次迭代产生的反馈信息，当前的反馈信息主要包括函数体的大小。如果不使用迭代编译，则直接进行函数内联分析。通过分析可以判断出哪些函数需要内联，并将分析结果标示到调用图中。

c)执行实际的内联扩展。这个阶段的作用是根据内联分析的结果，对那些允许内联的函数，在函数调用点执行实际的函数内联。相应的函数内联包括：复制被调用函数的抽象语法树；用实参代替形参；为被调用函数生成相应的符号表信息；维护 profiling 信息，并更新函数调用图。

4 实验数据分析

为了评估关键路径提取对内联优化的辅助效果，我们比较了加入关键路径提取前后内联优化对程序性能的影响。图 3 中给出了开关内联优化 l3-switch 程序的性能图。图中横坐标表示使用的微引擎数目，纵坐标表示数据包的转发率。BASE 包括基本优化选项，CPE 表示关键路径提取。从图中数据的比较可以看出，在使用 1 个微引擎的时候，对于 BASE +CPE 选项，包转发率可以提高 10%；随着使用微引擎数量的增加，网络系统的转发率不断提高，到 5 个 ME 的时候，由于相应的访存瓶颈导致转发速率不再提高，达到最大值。

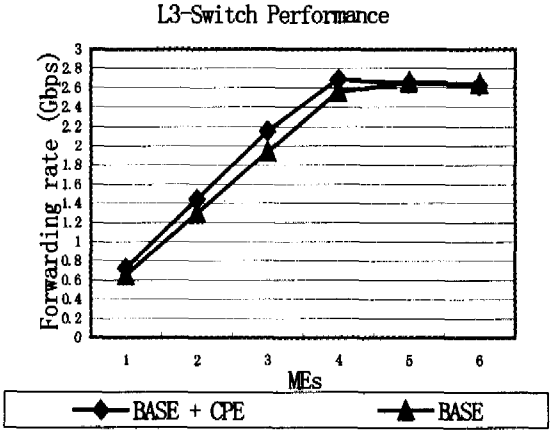

图 3 关键路径提取性能对比图

结论 本文提出了如何改造传统的内联优化技术来更好地适应 IXP 网络处理器特殊体系结构特征。针对 IXP 的硬件特征，设计并实现了关键路径提取技术和迭代编译技术，辅助内联优化达到更好的效果。通过关键路径提取技术能够让编译器自动实现处理任务的分离，有效地避免了由于处理任务共享导致的内联代码膨胀问题。通过迭代编译为内联分析提供了较准确的函数体大小信息，为内联提供了更好的数据支持。实验结果显示，利用内联优化技术编译器可以显著提高网络处理程序包的转发率。

参 考 文 献

1 谭章熹，林闯，任丰原，等. 网络处理器的分析与研究. 软件学报，2003

2 Zhao Peng, Amaral J N. To Inline or Not to Inline? Enhanced Inlining Decisions. LCPC 2003

3 Cooper K, Hall M, Torczon L. Unexpected Side Effects of Inline substitution: A Case Study. ACM Letters on Programming Languages and Systems 1992,1(1):22～32

4 Vin H, Jason J, Johnson E, et al. A Programming Environment for Packet Processing Systems: Design Considerations. 2004

5 Leupers R, Marwedel P. Function inlining under code size constraints for embedded processors. In: Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design, San Jose, California, 1999

6 Ayers A, Gottlieb R, Schooler R. Aggressive inlining. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), May 1997

7 Chang P P, Mahlke S A, Chen W Y, et al. Profile-guided automatic inline expansion for c programs. Software - Practice and Experience, 1992, 22(5):349～369

8 Intel Corp. Intel Microengine C Compiler Support: Reference Manual. 2002