

基于调用链分析的访存优化技术

时 磊¹, 王红梅²

(1 国家计算机网络应急技术处理协调中心, 北京 100029; 2 中科院声学所, 北京 100190)

摘 要: 本文提出一个基于调用链分析的访存优化编译技术. 该技术通过建立一个扩展的调用关系图, 分析程序从而估算出最小的寄存器保存集合, 达到提高寄存器的利用率, 并减少冗余访存操作的目的. 本文将该算法在 IM-PACT[1]编译框架中实现, 并对 DSPstone 测试集进行实验. 实验数据表明, 该算法能够平均减少 40.36% 的访存操作, 并获得平均 2.99% 的性能提升.

关键词: 函数调用; 访存操作; 上下文保存; 调用关系图; 编译优化

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2012)07-0032-03

Memory Operation Optimization Based on Call-graph Analysis

SHI Lei¹, WANG Hong-mei²

(1 CNCERT/CC, Beijing 100029, China; 2 Institution of Acoustics, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: In this paper, an optimal compiler algorithm is proposed for better utilization of registers and elimination of redundant memory operation. By building an extended call graph, the algorithm is able to saving least context during function call. The experimental results show a decrease in memory accesses by an average of 40.36% and an improvement in performance by an average of 2.99% applications.

Key words: function call; memory access; context saving; extended call graph; compiler strategy

1 引言

尽管微处理器的发展已经进入多核时代, 内存问题依然是制约性能的一个重要瓶颈. 由于内核执行速率远远超过了内存访问速率, 大大影响了处理器的性能. 同时, 访存指令具有更大的功耗, 会显著增加处理器的能量消耗. 为了减少访存指令, 人们需要从软硬件方面寻求解决方法. 和硬件方法相比, 编译优化手段更有效, 并且代价更少^[1].

在过去的十年中, 编译器采用寄存器分配的方法^[2-3]来减少访存操作. 然而随着应用程序设计的模块化, 函数调用也更加频繁. 传统的编译器在进行上下文保存时, 由于调用函数和被调用函数都不知道对方的寄存器用法, 因而生成了大量的寄存器保存/恢复指令. 通过分析若干应用程序发现, 由于编译器采用了传统数据流分析方法, 使得这些访存操作的大部分都是多余的.

本文提出了一个改进的编译优化算法, 可以在函数调用点减少所保存的上下文, 消除冗余访存操作的同时提升性能. 通过分析扩展的函数调用图中每个函数的寄存器使用集合, 算法可以估算出每个函数调用点进行上下文保存所需要的最小访存集合, 从而消除那些冗余的访存操作.

2 相关领域的研究进展

访存优化编译技术在文献^[2-7]中获得了深入的研究. 文献[2, 7]提出了改进的局部和全局寄存器分配方法, 是一种最有效的减少访存操作的编译技术. 该方法通过将最频繁使用的局部变量分配到寄存器中, 可以减少先前访问变量时所需要的访存指令. Davidson^[4]等人采用和 MIMD (Multiple Instruction Multiple Data, 多指令多数据流)类似的方法, 将多次小的内存连续访问合并成一次宽的内存访问. 而 Barik^[5]等人通过对并行应用中的方法调用、

并行结构进行流敏感和区域敏感分析,可以消除冗余的内存加载操作.文献[6]首次研究了函数调用点的上下文保存问题.为了获得最小的寄存器保存、恢复代价,文章通过采用缩小包裹的方式得到最小的包裹函数,并在该函数的开头和结尾处提供处理代码.

和文献[5]的方法相比,本文的优势主要体现在两个方面:首先,文献[5]假设所有的寄存器都是调用者保存的,而本文则假设寄存器的一部分是调用者保存,另一部分是被调用者保存,这是当今更普遍的一种函数调用规则.第二,文献[5]方法的作用区域是单个程序,而本文算法的作用域是整个应用程序.本文通过分析扩展的调用关系图,搜集每个函数的寄存器使用信息,最终可以估算出每个函数应当保存的最小寄存器集合,从而消除冗余的访存操作.

3 算法的研究动机和实现方案

3.1 算法动机

为了简要说明本算法的思想,本文以图 1 中的程序为例(由于空间有限,本文没有给出 C 等价的汇编代码).该程序对给定的输入向量计算有限冲击相应序列,由三个子程序组成:main(),DSP_fir()和fir().通常来说,当函数调用发生时,调用函数需要保存调用者保存寄存器(caller save register),防止被调函数可能需要使用它们;额外的,当它本身也是一个被调函数时,它还需要保存被调用者保存寄存器(callee save register),因为它的调用者可能会用到被调用者保存寄存器.

```
main()
{
    DSP_fir()
}
DSP_fir()
{
    while(i<32)
    {
        fir(x[],r[])
    }
}
fir(x[], r[])
{
    ...
    return 1
}
```

图 1 一个例子

通过观察汇编代码,发现作为fir()的调用者,DSP_fir()需要保存调用者保存寄存器d37,然而这个寄存器并没有被fir()使用到;同时作为main()函数的被调用者,DSP_fir()保存了7个额外的被调用

者保存寄存器,这几个寄存器同样在程序返回到main()之后没有被访问到.因此上述的寄存器保存操作属于冗余的内存访问操作.

为了减少这些冗余的内存操作,本文提出的算法首先通过剖面分析收集数据,建立一个扩展的函数调用关系图.扩展函数调用图的定义是:普通的函数调用关系图上扩展了相应函数的寄存器使用信息和执行频率信息.图2给出了示例程序的扩展函数调用关系图.

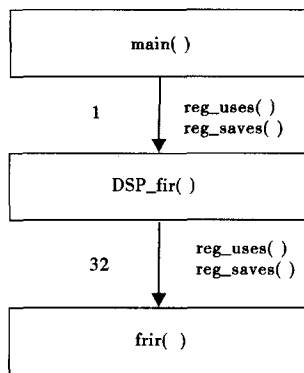


图 2 函数调用关系图

算法流程如下:(1)算法在得到程序间的控制流信息和数据流信息后,进行图着色寄存器分配.结合剖面分析得到的数据,可以建立一个扩展的调用关系图;(2)结合目标体系结构和应用约定,算法采用广度优先搜索方法遍历扩展调用关系图,计算每个函数节点的最小寄存器保存集合;(3)算法检查每个函数的堆栈结构,并更新堆栈状态.以上步骤完成后,算法便为应用程序生成了具有更少的访存操作的汇编代码.

3.2 算法流程阐述

本算法作用于程序的寄存器传输级语言(Register Transfer Language, RTL)代码. RTL 代码是编译器用来表示中间代码的一种常用格式.本算法的输入是已经进行了传统优化和前遍指令调度、寄存器分配阶段的 RTL 中间代码,输出是访存优化了的汇编代码.图3是算法的伪代码,对算法的主要过程进行描述.

在对算法流程进行详细介绍前,本文给出一些关键变量的定义:(1)结构体变量存储cge了扩展的调用关系图,包含三个主要成员变量,分别是函数名、函数的执行频率和寄存器使用集合;(2)根据调用惯例,每个函数的寄存器使用集合都会被分为两类:uses_caller和uses_called,分别表示当前函数所

Algorithm: saving least context during function calls

Input: extended call graph with register usage set which is divided into uses_caller and ruses_called and function list func_list

Output: functions with minimized context saving

```

1. Optimization and Profiling();
2. Pre_instruction_scheduling();
3. Register_Allocation();
4. cge=Build_extended_call_graph();
5. foreach f in func_list do{
6.   If! leaf_func(f) then { //steps for non-leaf function
7.     used_regs(f)=ecg. f. use_caller  $\cup$  ecg. f. uses_called;
8.     foreach g_func in ecg. f. child do {
9.       called_used_regs=called_used_regs  $\cup$  ecg. g. uses_caller;
10.    }
11.    foreach h_func in ecg. f. father do {
12.      caller_used_regs=caller_used_regs  $\cup$  ecg. h. uses_called;
13.    }
14.    min_saved_regs(f)=used_regs(f)  $\cap$ 
15.      (called_used_regs  $\cup$  caller_used_regs);
16.  }
17.  Else { //do optimizations for leaf function
18.    leaf_func_opti(f);
19.  }
20. update_stack_status(f)

```

图3 核心算法的伪代码

使用的调用者保存寄存器和被调用者保存寄存器。(3) $\text{used_regs}(f)$ 表示函数 f 自身的寄存器使用集合;(4) 为了计算函数 f 的最小寄存器保存集合 $\text{min_saved_regs}(f)$, 需要首先计算变量 called_used_regs 和 caller_used_regs , 它们分别表示 f 的调用函数所使用的寄存器集合和 f 的被调用函数所使用的寄存器集合。

代码的第一至三行表示在 for 循环之前, 算法进行一些必要的编译优化阶段来收集后面所需要的各种数据; 第四行构造 cge 变量, 记录函数调用图中每个函数的执行频率和寄存器使用信息, 需要在寄存器分配阶段之后才能进行; 算法的第五行至第十五行遍历整个应用的调用图, 找到非叶子函数. 对于非叶子函数, 它的最小寄存器保存集合通过以下四步进行估算: (1) 算法找到当前函数 f 的寄存器使用集合 $\text{used_regs}(f)$, 它是最终计算结果的一个超集. (2) 算法计算函数 f 所调用函数的寄存器使用集合 called_used_regs . 它的计算范围必须是当前函数 f 在调用关系图中的直接后代. (3) 算法计算当前函数 f 的调用者使用的寄存器集合 caller_used_regs . 它的计算范围为当前函数 f 在调用关系图中的所有父节点. (4) 根据 caller_used_regs 和 called_used_regs , 估算出函数 f 的最小保存集合

$\text{min_saved_regs}(f)$.

由于叶子函数在应用程序中占了很大的比例, 所以本算法针对叶子函数设计了优化函数. 在 $\text{leaf_func_opti}(f)$ 函数中, 它检查叶子函数是否还有额外的内存空间需求: (1) 如果没有, 那么叶子函数的堆栈空间可以进行优化. 这将加速叶子函数的执行并提升性能; (2) 如果有额外需求, 那么对叶子函数的堆栈空间进行必要更新.

4 实验数据与分析

将算法应用到 IMPACT^[7] 编译框架中, 然后对不同的测试程序进行编译, 最后对测试结果进行比较和分析, 从而评估算法的性能效率. IMPACT 是一个由 Illinois 大学开发的开源交叉编译器, 而测试集选自 DSPstone^[7], 它主要包含 DSP 的核心算法例如 fir, fft 和若干向量矩阵运算.

测试集的访存指令优化比例在图 4 中给出, 测试集的性能加速在图 5 中给出. 可见本文中的算法减少了平均 40.36% 的内存访问指令, 这可以有效缩小代码体积, 并改善性能. 同时测试集平均的性能提升在 2.99%. 需要注意的是, autocor 程序被消除的内存操作占 43.6%, 而性能提升只有 0.07%. 这是因为大部分执行时间都花在了大量的循环上, 这种情况可以使用文献[5]中的方法进行改善.

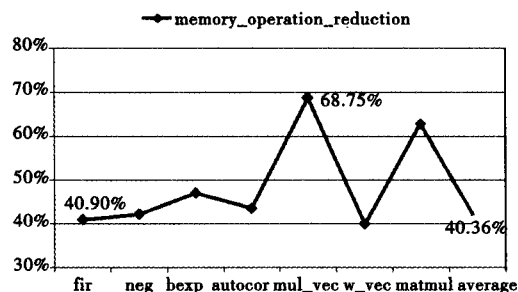


图4 测试集的访存操作比较

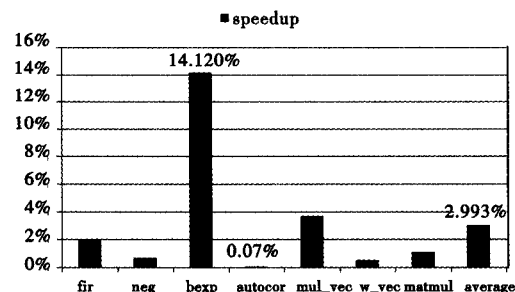


图5 测试集的性能加速比较

- mization using SOP balancing[C]// Proceedings of International Workshop on Logic and Synthesis. San Diego, USA, 2011:75-82.
- [5] Hassoun S, Sasao T. Logic synthesis and verification [M]. New York: Kluwer Academic Publishers, 2002.
- [6] Das D, Wang J, Zhou H. iRetILP: an efficient incremental algorithm for min-period retiming under general delay model[C]// Proceedings of the 2010 Asia and South Pacific Design Automation Conference. Taipei, Taiwan, 2010:61-67.
- [7] Malik S, Sentovich E M, Brayton R K, et al. Retiming and resynthesis: optimization of sequential networks with combinational techniques[J]. IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems, 1991, 1(10):74-84.
- [8] Jie-Hong R Jiang, Brayton R K. Retiming and resynthesis: a complexity perspective[J]. IEEE Transactions on Computer-Aided Design Of Integrated Circuits and Systems, 2006, 12(25): 2674-2686.
- [9] Mishchenko A, Case M L, Brayton R K, et al. Scalable and scalably-verifiable sequential synthesis[C]// Proceedings of IEEE/ACM International conference on Computer-aided design. San Jose, USA, 2008: 234-241.
- [10] Case M L. On invariants to characterize the state space for sequential logic synthesis and formal Verification [EB/OL](2011-02-14) <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-46.html>.
- [11] Mishchenko A, Chatterjee S, Robert Brayton. FRAIGs: a unifying representation for logic synthesis and verification [R]. ERL Technical Report, EECS Dept, U.C. 2005.
- [12] Bjesse P, Kukula J. Automatic generalized phase abstraction for formal verification[C]// IEEE/ACM International conference on Computer-aided design Proceedings, San Jose, USA, 2006:1076-1082.
- [13] IWLS 2005 Benchmarks[EB/OL][2011-05-16]. <http://iwls.org/iwls2005/benchmarks.html>.
- [15] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification[EB/OL][2011-05-06]. <http://www.eecs.berkeley.edu/~alanmi/abc>.

作者简介:

郝亚男 女, (1983-), 博士研究生, 研究方向为规模集成电路计算机辅助设计。

杨海钢 男, (1960-), 研究员, 博士生导师. 研究方向为高性能可编程逻辑芯片设计技术、数模混合信号 SOC 设计技术。

(上接第 34 页)

5 结束语

本文提出了一个在函数调用点减少冗余内存访问的编译优化算法。该算法基于扩展的调用关系图, 统计每个函数的寄存器使用信息, 可以计算得出每个函数的最小寄存器保存集合, 从而达到减少冗余内存访问的目的。而且由于访存指令功耗较大, 该算法同时也可以减少内存占用空间和功耗。通过对 DSPstone 测试集进行评估, 该算法能够平均减少 40.36% 的访存操作, 并获得平均 2.99% 的性能提升。下一步计划将文献[4]中的算法移植到本算法中, 从而对数组密集访问应用的性能进行优化。另外, 本文还将使用更复杂的测试集对本方案进行评估。

参考文献:

- [1] Aho A V, Sethi R, Ullman J D. Compilers, principles, techniques and tools [M]. Boston: Addison Wesley, 1988.
- [2] Zhang Y, Lee H J, Register allocation by systematic

merge of register-reuse chains [J]. ELECTRONICS Letters, 2000, 36(1):16-17.

- [3] Benitez M, Davidson J. A retargetable integrated code improver[R]. Univ. of. Virginia, Dept of Computer Sc., Technical Report CS-93-64. 1993.
- [4] Davidson J, Jinturkar S, Memory access coalescing: a technique for eliminating redundant memory accesses [C]// proceedings of PLDI. USA: Flor: da, June 1994.
- [5] Barik R, Sarkar. V. Inter-procedural load elimination for dynamic optimization of parallel programs [C]// proceedings of PACT. Canda: Toronto, 2009: 41-52.
- [6] Frederick C. Minimizing register usage penalty at procedural calls [C]// Proceedings of PLDI USA: Georgia, 1988: 85-94.
- [7] DSPstone benchmark suits. [EB/OL][2011-10-12] <http://www.ert.rwth-aachen.de/Projekte/Tools/DSP-STONE/dspstone.html>.

作者简介:

时磊 男, (1983-), 博士. 研究方向为计算机系统于网络应用。