乱序执行机器上的 load 指令调度

周 谦 冯晓兵 张兆庆

(中国科学院计算技术研究所系统结构重点实验室 北京 100080)

摘 要 随着处理器和存储器速度差距的不断拉大,访存指令尤其是频繁 cache miss 的指令成为影响性能的重要瓶颈。编译器由于无法得知访存指令动态执行的拍数,一般假定这些指令的延迟为 cache 命中或者 cache miss 的延迟,所以并不准确。我们引入 cache profiling 技术来收集访存指令运行时的 cache miss 或者命中的信息,利用这些信息来计算访存的延迟。乱序机器上硬件的指令调度对于发射窗口内的指令能进行很好的动态调度,编译器则对更长的范围内的指令调度更有优势。在 reorder buffer 中 cache miss 一旦发生,容易引起 reorder buffer 满,导致流水线阻塞。调度容易 cache miss 的指令,使其并行执行,从而隐藏 cache miss 的长延迟,就可以提高程序性能。因此,我们针对 load 指令,一方面修改频繁 miss 的指令的延迟,一方面修改调度策略,提高存储级并行度。实验证明,我们的调度对于 bzip2 有高达 4.8%的提升,art 有 4%的提升,整体平均提高 1.5%。

关键词 指令调度,cache profiling,存储级并行

Scheduling Load Instruction on Out-of-Order Machine

ZHOU Qian FENG Xiao-Bing ZHANG Zhao-Qing

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

Abstract With the gap between the speed of processor and memory become wider and wider, memory access instructions especially frequently cause cache miss are the bottleneck of the performance. As the compiler does not know the exact cycles of memory access instructions, it assumes the memory access instructions always hit or miss. So this is not accurate. We introduces cache profiling, which collects run time cache miss or hit information of memory access instructions. Then we use this information to calculate the latency of these instructions. On out-of-order machine the hardware instruction scheduler can schedule the instruction inside issue window well, and the compiler have advantage on scheduling the instruction of long distance. Once cache miss occurs, reorder buffer may easily be full, causing the stall of pipeline. Then scheduling the instruction frequently causes cache misses, trying to paralleling them, can hide the long latency of cache misses and improve the performance. So we adjust the latency of the instruction frequently cause cache misses, and modify the scheduling policy, trying to improve memory level parallelism. Experiment shows that our scheduling policy can improve the performance 1.5% on average, with bzip2 4.8% and art 4%.

Keywords Scheduling, Cache profiling, MLP

1 引言

随着处理器和存储器速度差距的不断拉大,访存指令尤其是频繁 cache miss 的指令成为影响性能的重要瓶颈。编译器由于无法得知访存指令动态执行的拍数,一般假定这些指令的延迟为 cache 命中或者 cache miss 的延迟,所以并不准确。以 cache 命中的延迟来调度,而运行时发生 cache miss,用来隐藏 cache miss 的其他计算指令就会比用 cache miss 的延迟少。以 cache miss 的延迟来调度,会把过多的指令与 load 指令同时执行,而 load 指令的延迟实际没有那么多,load 及其后继指令的优先级就被降低了。乱序机器上硬件的指令调度对于发射窗口内的指令能进行很好的动态调度,编译器则对更长范围内的指令调度更有优势。在 reorder buffer 中 cache miss 一旦发生,容易引起 reorder buffer 满,导致流水线阻塞。如何调度容易 cache miss 的指令,使其并行执行,从而隐藏 cache miss 的长延迟,就可以提高程序性能。因此,我们就针对 load 指令,一方面修改延迟,一方面修改调度策略,提高存储级并行度。

2 相关技术简介

2.1 cache profiling 技术

在编译器对程序进行编译的时候,编译器需要一些程序运行时的状态统计信息来指导程序进行编译优化工作,这样才能有针对性地对程序进行优化,以使编译生成的程序有较好的性能。程序进行 profiling 的时候会纪录下程序相应运行状态信息,这些信息对基于 profiling 的编译各个方面有一定的指导作用。Profiling 实现的方法是通过在编译过程中生成的代码中插入相关的指令或函数来收集所需要的信息。

cache profiling 技术主要通过插桩技术,调用库函数收集运行时信息,在库函数中实现 cache 行为的模拟。这里的 cache 通过软件的模拟来实现。具体的实现过程参见文[1]。

cache profiling 收集到的信息主要是每条访存指令 miss 和命中的次数,记录在反馈文件中。这些信息可以用来计算每条访存指令的命中或者 miss 的概率,从而算出这条指令的平均延迟。

$$average\_latency=\frac{hit\_counter}{exec-counter}\times T_{HIT}+\frac{miss-counter}{exec-counter}$$

$\times T_{MISS}$

公式中的 hit_counter、miss_counter 和 exec_counter 都记录在反馈文件中。

2.2 表调度(List scheduling)

表调度是编译器中常用的一种调度方式。它先建立数据依赖图,称之为 DAG(有向无环图),每个基本块的指令之间的依赖关系就反映在 DAG 图中。然后按照时间顺序,在每一拍排放适当的指令。在每一拍进行调度的时候,维持一个 Candidate List。如果指令依赖的操作数都已经准备好并满足结构相关约束,就可以作为一个 Candidate 加入 Candidate List,在 Candidate List 中按照一定的优先级选取适当的指令进行排放。当所有可能的指令都被处理过之后,进入下一个节拍进行调度。

2.3 存储级并行

随着存储器和处理器速度差距不断拉大,访存的延迟已经达到了几十甚至上百拍。一旦发发生 cache miss,尤其是最后一级的 cache miss,代价就非常高,处理器要花费大量的时间等待访存的结果。于是,人们想办法减少访存延迟或者隐藏访存延迟。Glew 提出存储级并行是一个有希望的方向[5]。然后很多人提出了很多方法来提高存储级进行或者评估存储级并行度。Zhou 用值预测来提高存储级并行度[7]。Pai 用代码移动来增加 read miss clustering[6],但是仅限于一些矩阵运算。

Chou 给出了存储级并行度的形式化定义:当至少有一个片外长延迟的访存时这种片外访存的平均数目[8]。但是在乱序执行的机器上计算存储级并行度有些困难,因为在编译时间很难确定在当前时刻有哪些执行运行时并行执行,所以 Chou 提出了存储级并行的分析模型。他把整个指令序列划分成一些时间片,称之为 EPOCH。存储级并行度就是片外访存的次数对 EPOCH 集合的比值。每个 EPOCH 有四个主要的窗口关闭条件,如发射窗口的大小、串行的指令等。

3 ORC 原有的调度策略

ORC 原有的调度策略是基于表调度技术。在 Candidate List 中选取候选指令时,需要根据一定的优先级。在 ORC 原有的调度策略中,优先级是:延迟长的优先调度。延迟相同或者相近情况下再基于以下策略确定优先级:

1)非投机的指令优先于投机指令

2)延迟长的指令优先(源基本块不是目标基本块)

3)发射槽(Issue_port)多的指令优先

4)Store 指令优先

5)fan_out 高者优先

这几个策略中排在前面的优先。

fan_out(i) = the number of outgoing edges of i, where $(i,j)\in E \wedge j\in succs(i)$

这样安排调度策略的原因是优先调度延迟长的指令,这样依赖于长延迟指令的指令就可以提前被调度,也利于延迟较长的指令并行执行,坏处是可能会引起重排队列(reorder buffer)满。

投机的指令由于已经作了投机,不再需要太提前调度,而且投机的指令实效可能会带来一定的代价。如果源基本块和目标基本块不是一个基本块,延迟长的指令仍然优先,原因同前面所述。发射槽多的指令优先调度是因为这样的指令有更为空闲的功能部件。Store 指令优先是由于访存执行对性能影响非常大,由于有 store buffer store 指令可以不必等到结果写回存储器,因此不用阻塞就可以继续执行其他指令。Fan_out 高者优先是因为这样的指令有更多的后继,优先调度 fan_out 高者就可以使更多的指令处于准备好的状态,利于调度,从而有更好的性能。

4 乱序执行机器上的指令调度

定义(delinquent load) 频繁 cache miss 的 load 指令,即 miss 的概率超过 90%的指令。

Load 指令的 miss 概率可以通过 cache profiling 收集到的信息来得到。

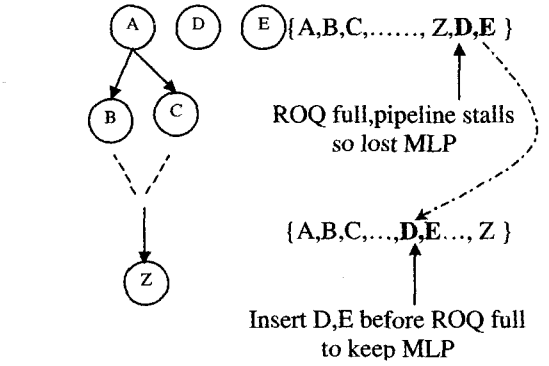

图 1 乱序执行机器上提高 MLP 的指令调度示意图

因为 cache miss 的指令的拍数最长,将数据依赖图中访存指令中 delinquent load 的指令的延迟调长,调度策略仍然按照 ORC 原有的长延迟优先的策略,就会优先调度 delinquent load 指令,从而提高存储级并行度。如图 1,如果 D,E 是 delinquent load 的指令,由于它们的延迟最长,优先级最高,就可以与其他的 miss 的访存指令并行执行,存储级并行度就会提升。

对于依赖于 delinquent load 的指令,我们使其优先级最低,使得不依赖于 delinquent load 的其他指令可以与 delinquent load 指令并行执行,就可以隐藏部分长访存延迟。为了使依赖于 delinquent load 的指令优先级最低,我们将依赖于 delinquent load 的指令的延迟,做比较时延迟临时减去 delinquent load 的指令的延迟,这样延迟为负数,优先级最低,依赖图中的依赖于 delinquent load 的指令的延迟并没有改变。

在延迟相同或者相近的情况下,增加关键路径长度(critical_path_length)作为调度策略的一个衡量标准。

critical_path_length 是 DAG 图中每个顶点达到底部的最长权值路径的长度。

优先级为:

1)非投机的指令优先于投机指令

2)关键路径长度长者优先

3)延迟长的指令优先(源基本块不是目标基本块)

4)发射槽(Issue_port)多的指令优先

5)Store 指令优先

6)fan_out 高者优先

5 实验结果

我们的实验是在 900MHz 的 MIPS 机器测得。机器的配置是 godson2E 芯片,一级指令和数据 cache 都是 64k 4 路组

相连，块大小为32B，伪随机替换算法，2级cache 512k，4路组相连，块大小为32B，伪随机替换算法，内存256M。一级cache命中延迟为4拍，二级cache命中延迟为17拍，二级cache miss访存的延迟为30拍。编译器为ORC[12] for MIPS版本。实验基准数据的选项为O3＋SWP。测试程序为SPEC CPU 2000[13]中的整数和浮点的例子。

表1 各种调度策略的实验结果

	Base	Version1	Version2	Version3	CSS
mcf	431	436	432	434	429
bzip2	273	290	286	292	286
parser	279	278	279	278	280
swim	383	383	385	385	388
mgrid	211	210	213	211	212
applu	243	254	248	249	249
art	448	466	468	467	437
equake	339	331	341	328	333
ammp	405	406	407	407	369
Geomean	324.28	328.85	329.30	328.40	322.54

实验结果如表1所示，这些调度策略都修改依赖图的load指令的延迟，并且都修改依赖于delinquent load指令的指令的延迟。

Base是按照原有的调度策略，但不修改依赖于delinquent load指令的指令延迟。

Versonl是按照关键路径长度最优先的策略来调度的。

Version2是按照原有的调度策略来调度的。

Version3是按照长延迟优先，关键路径长度次之的调度策略来调度的。

CSS是文[10]中的调度策略。

Base和Version2相比，修改依赖于delinquent load指令的指令延迟，性能明显提高，bzip2提高4.8%，applu提高2%，art提高4.5%，平均提高1.5%。Version2性能最佳，按照关键路径长度的调度Version1和Version3次之，这是因为长延迟优先的策略更容易提升存储级并行度。CSS策略是性能最差的一个策略，主要是因为我们的平台是乱序执行的机器，原来的参数不适应乱序执行的机器。

6 相关工作

文[11]根据profile工具DCPI来收集访存指令的miss与命中信息，然后根据这些信息来进行load指令的调度。在没有性能监视工具的机器上，显然这种方法就没法用。文[10]用编译器插桩函数方式收集profile信息，但是CSS调度策略只适用于VLIW型的机器，实验表明在乱序机器上我们的方法优于CSS算法。

Bernstein[1]提出了超标量机器上的全局指令调度技术。Kerns[2]在访存延迟不确定的情况下，提出balanced scheduling来调整访存延迟，提高指令级并行度。

结论 为了提高乱序执行机器上的存储级并行度，我们引入cache敏感的调度技术，将依赖图中频繁miss的访存指令的延迟调长，并修改调度的策略，使得依赖于频繁miss指令的指令优先级最低。实验结果显示，这样调度有很好的效果，平均性能提高1.5%，个别例子的性能提高4.8%。

参考文献

1 Bernstein D. Global Instruction Scheduling for Superscalar Machines. SIGPLAN, 1991

（下转第311页）

（上接第285页）

表2 测试用例

测试编号 Plan4.1.1
测试项目 计划管理的计划退回处理窗口
测试目的 保证能够通过输入条件查询出正确的数据
测试配置 物资信息管理系统能够正常登录，计划退回窗口能够正常打开，并且各个输入和输出控件都显示了正确的数据。
测试步骤 1 检查退回计划记录数据窗口是否正确显示了所有的退回计划 2 计划类型、年、月、采购部门的下拉框分别选择全部计划、2005.1.供应处 3 点击查询按钮
预期测试结果 在退回计划记录数据窗口显示2005年1月份供应处所有的退回计划

(2)如果没有一种规则为依据进行测试用例编写，面对着窗口中众多的控件，要生成覆盖率高的测试用例集很困难。CICE图法给出了根据控件之间的因果关系生成CICE图，将众多的控件联系起来，生成了覆盖率高的CICE图。所以CICE图减少了窗口测试的随意性，并提高了测试用例覆盖率。

(3)对于测试用例的回归，如果是手工编写的测试用例，在程序修改后，需要找到所有与修改程序有关测试用例进行修改，修改的数目可能很多。如果是根据CICE图法生成的测试用例，可以修改CICE图再生成测试用例覆盖原来的测试用例即可。大大增加测试用例的可维护性，减少了修改测试用例的时间。

(4)在实际的测试中，CICE图法有很强的可用性。比FSM测试模型实际操作性强。

有时候几个关联的窗口也可以进行CICE图的构建，只要能够准确地作出CICE图，就可以自动生成测试用例。

结束语 在图形用户界面的系统中，使用CICE图法可以对窗口进行有效的功能性测试，本文给出了CICE图的形式化定义以及测试用例生成算法。依据本文的内容和算法可以设计窗口自动化测试软件，软件的CICE图的绘制方法可以按照Rational Rose的动态图的设计方法进行，且需要的变量可以保存于图形的变量结构中，这样可以使用本文的算法进行测试用例的自动生成。

参考文献

1 杜栓柱，谭建荣，陆国栋．基于界面构件关联图的软件功能测试技术．计算机研究与发展，2002，39(2)：148～152

2 金义富．基于可视化编程环境的软件测试．华南理工大学学报，2002，30(7)

3 White L. Almezen H. Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences. In: Int Symp on Software Reliability Engineering, San JoseCA. 2000-10. 110～121

4 Rosaria S, Robinson H. Applying models in your testing process. Information and Software Technology, 2000, 42(12): 815～824

向图 $TG=(V,E)$。$V=\{T_1,T_2,\cdots,T_n\}$表示要执行的一组计算任务，有向连线的集合 $E=\{e_{ij}\}$则表示任务间的先后关系，即任务间的优先级关系，e_{ij} 是从任务 T_i 到 T_j 的有向连线，表示任务 T_i 优先级高于 T_j。一个具有 10 个任务的简单任务关系如图 4 所示。

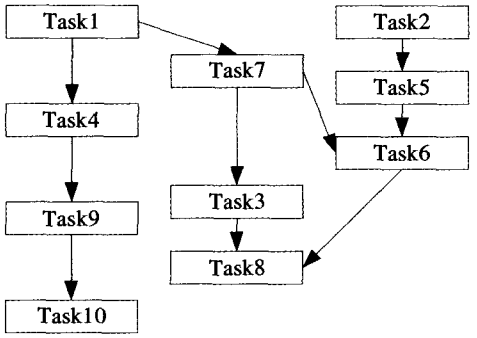

图 4　任务关系逻辑结构图

2.3　多嵌入式系统协作理论的应用研究

通过对普适计算和多嵌入式系统理论的研究，从中提取对嵌入式技术发展具有提升作用的关键共性理论和技术，指导和应用于嵌入式系统开发，并进而推广应用到诸如：汽车、仪器仪表、环境监控、智能家电等行业中，使我国的嵌入式系统开发水平得到迅速发展，适应未来普适计算发展的需求。

3　应用前景

随着 Internet 的迅速普及应用，网络计算方式的兴起，数据通信技术和多媒体技术的发展，嵌入式系统日益受到人们关注，其使用也越来越普及，促使了普适计算和多嵌入式系统的发展。而正在孕育发展的普适计算和嵌入式系统也使得协作技术的应用具有广泛的应用空间，它不仅把计算和信息融入到人们的生活空间，改变了人们对信息技术的思考，改变了整个生活和工作的方式，还帮助提高人们的办公效率和人们的生活水平。普适计算环境下的多嵌入式系统可以应用于商业、工业控制、医学领域、交通控制等。目前，数字家庭就是其简单应用之一。数字家庭可以通过家庭网关将宽带网络接入家庭，形成一个无缝、交互的环境。在家庭内部，PC、手持设备、智能电器等嵌入式设备通过无线或有线网络动态协作，共同完成人们的服务，满足人们的需求。

结束语　普适计算和嵌入式技术的发展给人们带来了极大的便利，但是就目前情况而言，还存在着许多挑战和尚需解决的问题。比如，现在无线网络通道数量少、通信距离短，所以要实现多嵌入式系统的有效协作，就要求开发新的通讯协议套件，要求有更快的数据传输速率，同时对宽带（尤其是无线网络）的需求也更多。

正在孕育发展的普适计算是嵌入的、互联的、动态的，它是多个计算领域的集成和综合，包含了嵌入式系统、计算机网络、个人计算等。普适计算把计算和信息融入到人们的生活空间，从根本上改变了人们的思考和工作方式。我们希望多嵌入式系统概念的提出和对多嵌入式系统基础理论的研究，能够有助于普适计算的发展，应用于每个领域，服务于人们。

参 考 文 献

1　Weiser M. The Computer for the 21st Century. Scientific American, 1991,265(3):94～104

2　罗从难，孙玉芳. 嵌入式系统及其特点. 信息系统工程，2000，10：32

3　Lu Jing-jian, Xiao Hai-qiao. Embeded system in the 21 century. Semiconductor Technology, 2001, 26(1)

4　Dey A K, Abowd G D, Salber D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human Computer Interaction ,2001,16:97～166

5　Hong J I, Landay J A. An Infrastructure Approach to Context-Aware Computing. Human Computer Interaction, 2001, 16

6　Oviatt S, DeAngeli A, Kuhn K. Integration and Synchronization of Input Modes during Multimodal Human-Computer Interaction. In: Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems, Vol. 1, 1997. 415～422

7　Ciarletta L, Dima A. A Conceptual Model for Pervasive Computing. In: Proceedings of International Workshops on Parallel Processing, 2000. 9～15

8　Muhlhauser M. Ubiquitous Computing and Its Influence on MSE. In: Proceedings of International Symposium on Multimedia Software Engineering, 2000. 48～55

9　Ponnekanti S R, Lee B, Fox A, Hanrahan P, Winograd T. ICrafter: A Service Framework for Ubiquitous Computing Environments. Proceedings of the Ubiquitous Computing Conference, Lecture Notes in Computer Science, Vol. 2201, 2001. 56～75

10　Tafat-Bouzid A, Courant M, Hirsbrunner B. A Generic Coordination Model for Pervasive Computing Based on Semantic Web Languages. Lectures Notes in Computer Science, Springer. In: F. Meziane, E. Métais, eds. 2004, 3136: 265～275

（上接第 300 页）

2　Kerns D R, Eggers S J. Balanced Scheduling Instruction Scheduling When Memory Latency is Uncertain. SIGPLAN, 1993

3　aMantripragada S, Jain S. A New Framework for Integrated Global Local Scheduling. In: Proceeding of International Conference on Parallel Architectures and Compilation Techniques, 1998

4　Karkhanis T, Smith J. A Day in the Life of a Data Cache Miss. In: Workshop on Memory Performance Issues, May 2002

5　Glew. MLP yes! ILP no! In: ASPLOS Wild and Crazy Idea Session'98, October 1998

6　Pai V, Adve S. Code Transformation to Improve Memory Parallelism. In: 32nd International Symposium on Microarchitecture, November 1999

7　Zhou H, Conte T. Enhancing Memory Level Parallelism via Recovery-Free Value Prediction. In: International Conference on Supercomputer, June 2003

8　Chou Yuan, Fahs B. Microarchiteture Optimizations for Exploiting Memory-Level Parallelism. In: Proceedings of the 31st Annual International Symposium on Computer Architeture'04, 2004

9　Mutlu O, Stark J. Runahead Execution: An Alternative to Very Large Instruction Windows for Out-Of-Order Processors. In: 9th International Symposium on High Performance Computer Architecture, February 2003

10　Hardnet C R, Palem K V, Rabbah R M, et al. Scheduling Load Operations on VLIW Machines: [Technical Report]

11　Lindenmaier G, McKinley K S, Temam O. Load Scheduling with Profile Information. In: Proceedings of 6th International Euro-Par Conference, 2000

12　www. spec. org

13　http://ipf-orc. sourforge. net