

应用基于抽象机的动态翻译设计 Java 处理器

王海晨, 赵祥模

(长安大学 信息工程学院, 陕西 西安 710064)

摘 要: 本文提出了一种基于硬件抽象机的动态翻译技术, 它可用于实现 Java 处理器. 该技术采用了硬件抽象机的“模糊执行”(HAM)方法, 通过分析 Java 程序之间的相关性, 动态地将 Java 字节码转换成基于标签的类 RISC 指令. 然后, 将堆栈折叠与动态翻译相结合进一步优化指令. 应用该技术设计了一个 Java 指令级并行处理器, 并且扩展它, 支持 Java 多线程功能.

关键词: 二进制翻译; 抽象机; Java 处理器; 多线程

中图分类号: TP303

文献标识码: A

文章编号: 1000-7180(2012)07-0001-03

Abstract Machine Based Dynamic Translation Technique to Design Java Processors

WANG Hai-chen, ZHAO Xiang-mo

(School of Information Engineering, Chang'an University, Xi'an 710064, China)

Abstract: Binary Translation is a migration technique that allows software to run on other machines achieving near native code performance. The paper proposed an abstract machine based dynamic translation technique in Java processors. The technique employs the “mock execution” of the hardware abstract machine (HAM) to identify and analyze the dependency among Java programs, dynamically translate Java bytecode into tag-based RISC-like instructions. After that, stack folding is combined with the technique to further optimize translated instructions. With the technique, we design a Java ILP processor and extend the processor to support Java multithreading.

Key words: Binary translation; abstract machine; Java processor; multithreading

1 引言

动态二进制翻译 (DBT)^[1]是在程序运行时, 将可执行代码从一个处理器格式翻译、优化转化为目标处理器格式, 然后执行的过程. 自八十年代后期, 微处理器制造商开发了二进制翻译技术, 用以提供从 CISC 机器到 RISC 机器的迁移方案. 现在 DBT 技术作为一个工具, 解决了广泛的系统需求, 同时保证软件的向后兼容. DBT 便于部署新的软件工具而无需重新编译或修改现有的软件. DBT 已成功应用于商业及研究环境, 支持虚拟化, 跨平台代码兼容性以及调试器和软件性能优化.

DBT 可以利用软件通过在运行时修改程序的

二进制代码实现. 这项工作需要软件动态翻译器支持.

大多数软件实现的动态翻译都采用了一个相类似方法; 即在程序与 CPU 之间增加一个软件层虚拟化主机运行环境. 软件层扮演了虚拟机角色, 动态检查和翻译程序指令.

本文应用抽象机^[2]实现了全硬件的 DBT 功能. 这里, 抽象机也就是译码器, 动态识别独立指令, 转化他们为标签基于的类-RISC 指令格式. 与堆栈折叠技术相结合, 处理器可以进一步减少加载和存储指令操作. 基于本文的 DBT 方法, 设计了一个 Java ILP 处理器, 并把它扩展到多线程 Java 处理器结构.

收稿日期: 2011-11-03; **修回日期:** 2011-12-25

基金项目: 国家自然科学基金项目(50978030)

2 基于抽象机的动态翻译技术

2.1 ADBT 概念

抽象机(ABM)大多用于编译系统中. 本文提出了一个实现硬件抽象机的方法^[2], 应用于微处理器设计中. 基于抽象机的 DBT 技术(ADBT)如图 1 所示. ADBT 是一个流水线概念并且能够兼容许多现有计算机体系结构. ADBT 利用硬件抽象机将任意二进制代码转化成基于标签的指令格式, 并且在解决数据冲突后, 实现指令级并行 (ILP) 执行. 因为标签指令与 RISC 指令结构类似, 所以可以通过标签的映射在 RISC 计算机上运行.

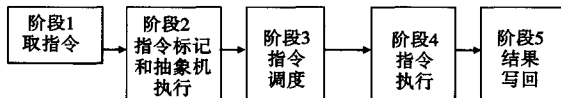


图 1 基于硬件抽象机的流水线设计框架

在流水线处理器中, 译码器逻辑通常要识别指令代码的相关性, 以便在下一个调度窗口中调度已经准备好的指令. 在基于 DBT 的处理器中, DBT 单元要实现动态翻译和程序执行. DBT 实现了一个译码器功能. 在 ADBT 中, 用硬件抽象机(HAM)实现译码器. HAM 通过“模拟执行”实现指令动态翻译. 不像在实际的处理器中那样, HAM 不涉及实际的指令执行. 接下来, 作为一个例子, 用一个 Java ILP 处理器来描述如何应用 ADBT 技术.

2.2 Java 堆栈抽象机

DBT 过程涉及动态翻译现有的二进制代码, 并根据需要替换指令. 在 Java 堆栈处理器中, DBT 由标记单元(TU)实现, 这里 TU 完成了 HAM 的功能. HAM 首先实现了基于标签(tag)的“模拟执行”, 给每个指令都添加标签, 然后将 Java 字节码转换为类 RISC 指令. 其次, 处理器完成 Java 堆栈指令折叠, 减少堆栈加载/存储操作. 第三, 如果折叠后的类 RISC 指令是独立的, 它们会被调度到发射单元, 在 RISC 引擎上执行.

2.3 Java 硬件抽象机

在堆栈计算机中, 一旦操作器指令使用操作数后, 所使用的操作数会被从执行堆栈上清除. 加载或算术运算指令的结果可能只需要提供给一个操作器指令. 生成的结果可以由所对应的标签唯一识别. 结果一旦使用, 会被立即丢弃而不存储到堆栈上.

使用下面的 Java 字节码片段说明一个字节码指令流加标签的过程: iload-1, iload-2, imul, iload-4, iload-5, iadd, iadd, istore-3.

表 1 显示了 HAM 的工作过程, 以及标签在操作数堆栈(OTS)上的改变过程. 每条堆栈指令, 当它进入命名单元时, 被分配一个独有的标签; 当一个指令在 ALU 单元被执行后, 将生成标签结果; 生成的标签将被传递给后面需要的指令. 与在 RISC 处理器的情况下一样, HAM 在程序中实现了一个“模拟执行”.

表 1 一个 JAVA 抽象机的工作实例

字节码 指令	指令标记单元 (TU)	操作数标签 堆栈 (OTS)	指令折叠后的 类 RISC 指令
1 iload-1	T2 iload-1	T1	
2 iload-2	T2 iload-2	T1 T2	
3 imul	T3 imul T1 T2	T3	i1:imul T1, T2, T3
4 iload-4	T4 iload-4	T3 T4	
5 iload-5	T5 iload-5	T3 T4 T5	i2:iadd T4, T5, T6
6 iadd	T6 iadd T4 T5	T3 T6	i3:iadd T3, T6, T8
7 iadd	T7 iadd T3 T6	T7	
8 istore-3	T8 istore-3 T7		

在支持 DBT 的处理器中, 所有代码的执行是由 DBT 控制的, 而且只执行被翻译后的代码块. 高速缓存(Cache)是用于存储已翻译的基本块或跟踪信息, 并用来减少运行时开销. HAM 包括命名单元 (NU)、操作数标签堆栈(OTS)和堆栈缓存. NU 是一个关键部分, 负责添加标签给指令, 并存储带标签的指令. OTS 用来模拟堆栈的执行. 堆栈缓存仅用于存储临时结果和数据. 每个基本块指令存储在 NU 中, 这里 NU 类似于指令缓存的作用, 并等待被翻译成带标签的指令. 基于标签的“模拟执行”过程很简单, 所以 HAM 可以工作的很快^[4].

2.4 Java 堆栈折叠

使用堆栈折叠^[5]技术, 处理器能够有效的减少堆栈加载/存储操作. HAM 可以动态地将 Java 字节码指令转换成基于标签的类 RISC 指令. HAM 只需通过模拟执行就可以识别指令相关性, 并简化了堆栈指令折叠的实现, 使指令标记和堆栈折叠同时运行. 本文使用参考文献^[5]中的方案完成堆栈折叠. 完成堆栈折叠后, HAM 将字节码指令转换成类 RISC 指令. 指令格式见表 2.

表 2 基于标签的指令格式

指令操作码	源标签 1	源标签 2	目标标签
-------	-------	-------	------

3 JAVA ILP 处理器

在这一章介绍如何应用 ADBT 技术设计一个 Java 指令级并行处理器. Java ILP 处理器是一个流水线处理器,它的流水线有六级,包括取指令,译码/标记,堆栈折叠,指令发射和调度,执行阶段. HAM 和 DBT 包括指令标记单元(TU),标签匹配单元(TMU),操作数标签堆栈(OTS)和堆栈折叠逻辑(SFL)组成. 指令标记后, TU 借助 SFL 将堆栈指令转换为基于标签的类 RISC 指令.

TMU 根据堆栈缓存(寄存器文件)中映射寄存器的值,跟踪一个基于标签指令的操作数的状态——就绪或未就绪状态. 如果一个指令的操作数为就绪状态,该指令就加入到就绪指令队列,然后被捆绑为 VLIW 指令. VLIW 指令的产生依赖于数据的可用性,其工作过程与文献[4]类似. 虽然 VLIW 指令束是按照顺序产生的,但是他们可能不会按照程序的顺序被捆绑. Java ILP 处理器的原理框图如图 2 所示.

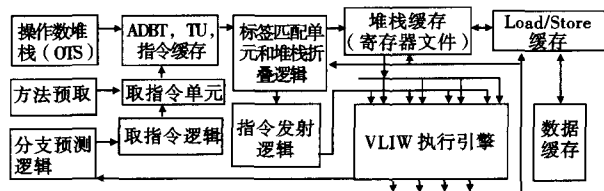


图 2 Java ILP 处理器架构

4 Java 多线程处理器

同步多线程(SMT)是一个多线程的变种,结合了宽发射的超标量处理器和多线程处理器的硬件特征. 它能够以更大的指令吞吐量和加速比使用线程级并行(TLP)和 ILP. 它可同时利用 TLP, ILP.

本文提出的 Java ILP 处理器可以扩展支持 SMT,以达到更高的加速比和吞吐量. 支持多线程的 Java 处理器(MJP),由四个线程单元组成. 每个线程单元位都拥有取指令单元,指令标记单元(TU),私有寄存器文件(PRF),程序计数器(PC)和私有页表. 在 MJP 中,多线程程序驻留在指令缓存中,多个取指令单元可以同时从多个线程读取程序. 来自不同线程的字节码程序,可以被不同 TU 标记,它与 Java ILP 处理器中的标记方式相同. 在基于标签的类 RISC 指令就绪后,他们被捆绑为 VLIW 指令并行执行. 标签指令来自于不同的线程, TLP 是据此实现的. 这里需要一个指令调度器来安排来自

不同单元的指令.

虽然标签就绪指令的发出可以不考虑数据的相关性,但是同一个线程内的指令相关由各单元处理. 以这样的方式设计, MJP 中的多线程可以共享执行引擎,实现高吞吐量. MJP 可以具有分层寄存器文件. 全局寄存器文件(GRF)用于线程之间的通信,存储全局变量. 每个线程的 PRF 用于存储线程内的临时数据.

处理器内存通过虚拟内存机制,由所有线程共享. 在 MJP 中,多个线程可以通过共享虚拟内存来共享公共对象或数据,因此我们考虑采用顺序一致性或释放一致性内存管理模型来管理内存,保证程序执行的正确性.

5 结束语

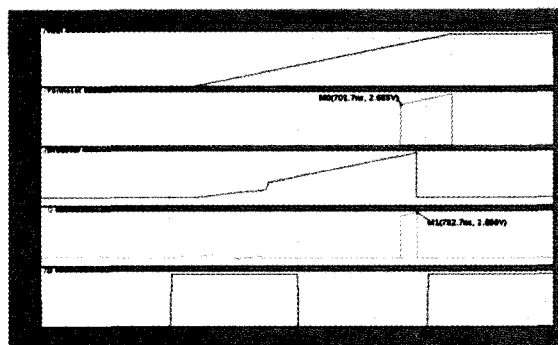
本文提出了一种基于硬件抽象机的 DBT 技术,它可以使用“模拟执行”技术动态将二进制文件转换为基于标签的类 RISC 指令格式. 基于标签的 RISC 指令很容易在 RISC 引擎上执行. ADBT 作为一个译码器单元,能识别所有指令的相关性,发掘程序中的 ILP. 本文描述了如何实现一个 Java ILP 处理器,讨论了多线程 Java 处理器(MJP)的架构和有关问题. 与国内 Java 处理器的研究^[6-7]相比,本文更偏重于理论的研究,对具体的实现具一定指导作用,尤其是方案具可扩展性,可实现多核 Java 处理器.

HAM 中有了指令的“模糊执行”,处理器可以挖掘出所有不相关的独立指令组,并将独立指令组放入发射单元实现 ILP 执行. 该技术也可应用到 X86 CISC 处理器. 在英特尔的 X86 处理器中,一般地 X86 指令被拆分或破解为许多类 RISC 微操作指令^[8],然后再执行它们. HAM 可以实现文献[8]中类似的工作,识别出各微指令直间的相关性,并将独立的微指令放入同一指令发射组. 详细实现 X86 多线程处理器也将是今后的工作.

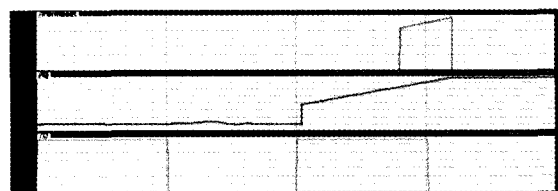
参考文献:

- [1] Sites R L, Chernoff A, Kirk M B, et al. Binary translation[J]. Communications of the ACM, 1993, 36(2): 69-81.
- [2] Stephan Diehl, Hartel P, Sestoft P. Abstract machines for programming language implementation [J]. Future Generation Computer Systems, 2000(16): 739-751.

(下转第 7 页)



(a) 读低阻 read low resistance



(b) 读高阻 read high resistance

图 5 读出保存的数据

表 1 非挥发存储器读写速度比较^[2]

	FLASH	EEPROM	MRAM	FeRAM	PCRAM
Read	60ns	80ns	30ns	80ns	20ns
Write	1 μ s	8ms	30ns	80ns	10ns

4 结束语

基于 0.13 μ m SMIC 标准 CMOS 工艺,本文设计了一种提供快速掉电数据保持功能的触发器,掉电速度 0.15 μ s/V,系统的数据可以在 100ns 内完成数据写入。上电时可以在 30ns 内恢复掉电时的数

据状态。其特点是数据的保持是基于相变存储单元的,可以对该相变存储单元反复进行读写擦操作。由于相变存储单元的制造工艺简单,和 CMOS 工艺兼容,数据保持所需的操作时间在纳秒级,而且可以长时间保持。所以该电路是一种低成本高速掉电数据保护的电路。

参考文献:

- [1] Hamid Mahmoodi-Meimand, Kaushik Roy. Data-retention flip-flops for power-down applications[C]// ISCAS, Canada, 2004; II-677- II-680.
- [2] 宋志棠. 相变存储器 [M]. 北京:科学出版社, 2010:1-19.
- [3] Xiangyu Dong, Norman P Jouppi, Yuan Xie. PCRAMsim: system-level performance, energy, and area modeling for phase-change RAM[C]// 2009 IEEE/ACM International Conference on Computer-Aided Design Digest of Technical Papers. San Jose: ACM, 2009: 269-275.

作者简介:



张怡云 女,(1988—),硕士研究生。研究方向为模拟集成电路设计。

(上接第 3 页)

- [3] Hookway R J, Herdeg M A. Digital FX! 32: combining emulation and binary translation [J]. Digital Technical Journal, 1997, 9(1): 3-12.
- [4] Tremblay M, Chan J, Chaudhry S, et al. The MAJC architecture: a synthesis of parallelism and scalability [J]. IEEE Micro, 2000, 20(6): 12-25.
- [5] Wang H C, Yuen C K. Exploiting an abstract machine-based framework in the design of a Java ILP processor [J]. Journal of Systems Architecture: the EUROMICRO Journal, 2009, 55(1): 53-60.
- [6] 张金钟, 胡平. 基于 FPGA 的 Java 指令折叠器的研究与设计 [J]. 微电子学与计算机, 2011, 28(5): 166-170.

- [7] 张金钟, 胡平. 基于 RISC 结构的 Java 处理器研究与设计 [J]. 微电子学与计算机, 2011, 28(7): 61-64.
- [8] Hu S L. An approach for implementing efficient super-scalar CISC processors [C]// Proc. of the 12th International Symposium on High-Performance Computer Architecture (HPCA-12). IEEE, Austin, TX, 2006.

作者简介:

王海晨 男,(1969—),博士,副教授。研究方向为高性能计算机体系结构。

赵祥模 男,(1966—),博士,教授,博士生导师。研究方向分布式测控系统。