

抽象机在 Java 微处理器中的应用研究

王海晨, 赵祥模

(长安大学 信息工程学院, 陕西 西安 710064)

摘要: 抽象机通常用在软件程序编译器中。提出了一个基于硬件抽象机的处理器设计方法, 使用该方法设计了一个 Java 微处理器, 并且利用硬件抽象机增强了处理器的指令级并行能力, 提高了微处理器性能。描述了用于 Java 处理器的硬件抽象机设计方法, 阐述了它的实现基本原理, 给出了 Java 处理器的逻辑设计。通过软件仿真, 证明了采用硬件抽象机的 Java 处理器可以获得从 78% 到 173% 的指令级并行增强, 处理器性能提高平均 31%。说明了提出的方法可以用于嵌入式微处理器的设计, 提高系统性能。

关键词: 指令级并行; Java; 微处理器; 抽象机

中图分类号: TP368

文献标识码: A

文章编号: 1673-629X(2011)06-0242-04

Research on Application of Abstract Machine in Java Processors

WANG Hai-chen, ZHAO Xiang-mo

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

Abstract: The abstract machine is generally used to construct a software compiler. Proposed a method of building a hardware-based abstract machine, applied the method to design a Java processor, enhance the processor's instruction level parallelism (ILP), promote the processor's performance. Described the method of how to design the abstract machine, and gave the Java processor's logic architecture. Through simulation, the work we did demonstrated that the Java processor based on hardware abstract machine is able to enhance ILP in Java programs from 78% to 173%, and increase the processor's performance 31% in average. Also demonstrated that our method can be used to design embedded processor to improve the performance.

Key words: ILP; Java processor; micro-processor; abstract machine

0 引言

经典的微处理器普遍采用指令流水线技术来提高性能。随着国内自主研发的发展已有一些 RISC 处理器诞生, 如文献[1~5]这些微处理器也都采用了流水线技术。流水线技术实现了指令的重叠执行, 提高了性能。但是, 由于控制相关、数据相关、资源相关等一系列问题, 导致流水线断流, 影响了性能。为了更进一步提高性能, 现代微处理器发展了指令级并行技术, 超标量执行, 乱序执行和预测执行^[6], 允许多条流水线同时处理多条指令流。

超标量执行允许程序块中, 临近的多条指令位于相同的流水段中同时处理, 这样它可以并行执行两条

以上指令仅当它们以程序的顺序连续出现时。乱序执行允许指令越过其它指令, 进入指令处理的其他阶段, 按照与原程序顺序不同的顺序进行。预测执行通过减少程序的顺序限制, 增强了可用的并发性。这些方法的关键是以任意顺序处理指令, 但是从程序执行的外部表象看它们是顺序执行的。超标量执行技术需要增加额外的硬件支持, 比如它要求并行译码器的支持。另外, 超标量执行也需要有足够的功能执行单元、寄存器读写口、存储器口以及支持指令间数据传递的路由网络。乱序执行减弱了执行顺序的约束, 可以增强程序执行的并发能力, 但是, 需要额外的工作来保证程序执行的正确性。预测执行可以克服条件转移指令的影响, 实现更大的并发性。根据以上分析, 看到指令相关问题的解决是提高指令级并行能力的关键。

Java 微处理器由于采用堆栈结构产生的指令相关严重影响处理器性能, 针对这一问题, 国外也有一些研究^[7~11]。文中首先提出了一个基于抽象机技术设计微处理器的理论框架。该框架在现有基本流水线的基础上, 引入了一个硬件抽象机。由于硬件抽象机实现基于标签的模拟执行, 所以比较简单, 容易实现。该框

收稿日期: 2010-10-29; 修回日期: 2011-01-10

基金项目: 国家自然科学基金(50978030); 中央高校基本科研业务费专项资金(CHD2009JC125); 长安大学基础研究支持计划专项资金; 陕西省工程研究中心重点实验室开放基金(CHD2009JC125)

作者简介: 王海晨(1969-), 陕西西安人, 副教授, 博士, 研究方向为高性能处理器结构、并行与分布式计算; 赵祥模, 教授, 博士, 博士生导师, 研究方向为分布式网络测控。

架可以适用于 RISC, CISC 或堆栈基于的处理器。利用文中的设计框架,我们设计了一个 Java 微处理器逻辑结构,以便进一步说明该框架的适用性。设计的 Java 处理器能够有效解决流水线处理器中的指令相关,挖掘出 Java 程序内部的指令级并行能力,提高处理器性能。

利用软件仿真方法对处理器验证,证明了所提出的基于硬件抽象机的 Java 处理器可获得从 78% 到 173% 的指令级并行增强,提高处理器性能平均 31%。

1 基于硬件抽象机的处理器设计框架

抽象机^[12]通常用在软件程序编译器中,它们在高级编程语言和真实硬件处理器之间起到桥梁作用,并且省略了许多与实际硬件处理器实现有关的细节^[12]。把抽象机技术引入到微处理器设计中,提出了一个基于硬件抽象机的处理器设计框架(见图1)。该框架融合了流水线技术和许多现有的计算机体系结构技术,通过使用带标签的指令执行技术,挖掘出存在于程序中的指令级并行。

硬件实现的抽象机一般位于流水线的指令译码和分析阶段(阶段2,见图1),实现带标签指令的“模拟执行”。在传统的处理器中,数据被输入到流水线,通过流水线执行,产生输出结果(具体数值),然后结果被写回寄存器(或内存)。硬件抽象机使用标签“执行”,而不使用具体数据。在标签执行过程中,只有标签从抽象机(或堆栈)中被抹去,以及将模拟执行后生成的新标签放回到抽象机(或堆栈)中。这种“模拟执行”过程类似于流水线的执行,没有真实的数据参与,也可以称作“虚拟”执行。

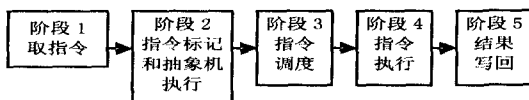


图1 基于硬件抽象机的流水线设计框架

硬件抽象机实现“模拟执行”就是要在流水线的译码阶段,通过对标签的状态、标签之间的相关性进行分析,比如某个标签会在哪几条指令中被使用,从而识别出程序块中的数据流关系;通过数据流分析进而得到程序中的指令相关关系,揭示出指令级并行的可能性,为在后期流水段的指令调度做好准备工作。指令标记单元仅完成基于标签的执行,可以使用简单的硬件实现。它顺序地处理输入指令流,但其处理速度可以远超过真实指令的顺序执行,可以满足并行执行的需要。

2 堆栈硬件抽象机的工作过程

这里将描述一个基于堆栈的硬件抽象机的工作过

程。硬件抽象机由指令标记单元(TU)和操作数标签堆栈(OTS)。指令标记单元把生产者指令(比如,数据加载指令)产生的结果标签放入操作数标签堆栈,而消费者指令(比如,数据存储指令等)在指令标记过程中从OTS去除使用过的标签。这样,指令标记技术就在生产者指令与消费者指令之间建立了一个数据流关系。使用下面的Java代码片段说明指令标记与抽象机执行过程: `iload_1, iload_2, imul, iload_4, iload_5, iadd, iadd, istore_3`。

表1显示了一个硬件抽象机如何工作的过程,以及标签在操作数标签堆栈(OTS)上的改变过程。每条堆栈指令,当它进入指令标记单元时,被分配一个独有的标签。当一条算术逻辑指令在标记后,当它要执行时,作为指令的输入标签要从OTS弹出;执行后,生成的结果标签将被压入OTS,以便传递给后来需要这一结果的指令作为输入标签。在例子中,指令T2加载后,标签T2被压入OTS;指令T3需要T1、T2作为输入,将T1、T2从OTS弹出,或者说消费掉T1、T2;生成的结果在T3中保存,T3被压入OTS,等待后续指令把它作为输入变量。

在堆栈处理器中,操作数被操作者指令使用后,就会被从执行堆栈中抹去。因此,一个数据加载指令的值只能提供给一个操作者指令使用,而操作者指令生成的结果,可以用一个标签唯一识别。我们看到,通过上面的指令标记过程,“Java字节码指令对”之间的消费者-生产者关系可以通过标签识别出来,而且类似于RISC处理器的执行,使用标签实现了一个程序的“模拟执行”。

表1 一个JAVA抽象机的工作实例

字节码指令流	指令标记单元(TU)	操作数标签堆栈(OTS)	指令折叠后类RISC指令
1 iload_1	T1 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		

文中提出的硬件抽象机技术很容易与Java指令折叠技术融合(篇幅有限,具体的指令折叠技术实现见文献[10])。上述代码段通过指令折叠逻辑后生成的类-RISC指令见第四列。标记后的带标签指令就可以被调度执行。标签用过后必须被释放,以便可以重用。从执行过程看,指令标记技术实现了一个程序的数据流图表达,其中,记录、分析操作数标签的使用就等价于分析指令间的数据相关问题,从而揭示出更多指令级并行。类-RISC指令流可以使用通用的RISC引擎执行。

3 基于抽象机的 Java 处理器(JAM) 结构

文中的 Java 处理器(简称 JAM 处理器)的流水线中包含了硬件抽象机。硬件抽象机首先对指令进行标记,把 Java 字节码指令转化为带标签的指令,使用标签进行抽象执行,发现与解决堆栈依赖问题,把堆栈指令转化为带标签的基于寄存器的指令,把可以并行执行的指令转化后成组发射出去。标签匹配单元根据在堆栈高速缓存(寄存器文件)中标签映像的寄存器的值,来跟踪基于标签的指令的操作数状态。当基于标签指令的操作数为准备好状态时,将它添加到准备好队列,由硬件逻辑发射到 RISC 执行引擎执行。在处理器中,指令是定序发射的,但是允许乱序执行,结果及其处理器状态更新都是按照程序逻辑顺序执行的。程序执行的逻辑行为对外是以程序执行顺序展现的,这样可以保证程序执行的正确性。JAM 处理器逻辑结构如图 2 所示。

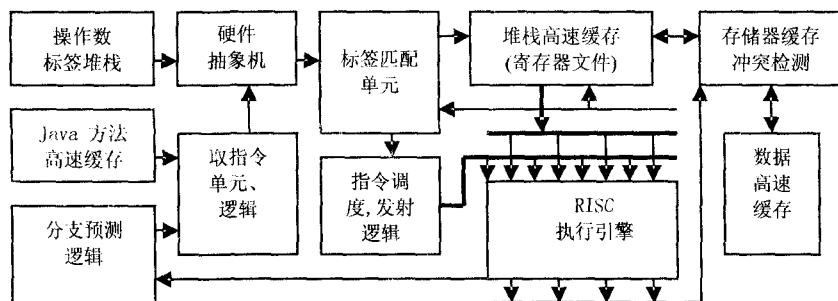


图 2 JAM 处理器逻辑结构图

4 软件模拟和性能评价

根据 JAM 处理器结构,在 Linux 运行环境下,开发了一个踪迹驱动(trace-driven)处理器性能仿真器,模拟处理器的流水线工作过程,对基于堆栈硬件抽象机的 Java 指令并行处理器进行了仿真。在仿真器中,标签匹配单元假定它含有 64 个标签项。利用修改过的开源软件 Kaffe(一个开源 JVM 解释器),获得软件测试程序的字节码执行流。仿真器接收 Java 字节码流作为输入,对这些代码在流水线上进行调度和执行。

实验中,运行了 SPECjvm98 和 Linpack 基准测试程序。SPECjvm98 测试程序包括 7 个程序:Compress, Db, Jack, Javac, Jess, Mpegaudio 和 Mtrt,它们分别测试 Java 程序的不同应用情形。使用 sl 数据集,Mtrt 程序使用单线程版本。仿真器使用了静态预测器^[13],如果预取错误,将产生 3 个周期的惩罚时间。另外,实验中指令调度限制在基本程序块中,支持指令预取,指令缓冲器和数据缓冲器假定为全命中状态。

4.1 ILP 执行增强

为了探索 ILP 执行增强,假定处理器为 4 发射处理器。在同一指令发射窗口内,如果指令间没有依赖

关系,同一周期,处理器可以同时发射 4 条指令并行执行。如果指令间发生了依赖,后续指令必须等到前面的执行结束,才可以发射。运行基准测试程序获得的 JAM 处理器指令并行执行的百分率(见表 2)。为了说明提出的处理器结构在 ILP 执行方面的优势,把它与已有的国外文献进行比较。文献[4]使用了 Cache 来存储一些已执行的 Java 程序代码以便在后续使用时可以直接从 Cache 读出,从而提高程序的执行效率,并且采用定序多发射结构,通过使用堆栈排歧(stack disambiguation)技术提高执行效率。在其给出的结果中,只有少部分三指令组被并行执行,没有四指令并行执行组存在。与之相比,我们的仿真结果中,三指令并行执行组占总执行指令的比率从 1.55% 到 10%;四指令并行执行组占总执行指令的比率从 0.86% 到 33.43%。这里指令并行执行比率的不同与程序的特性有关。这一结果表明,提出的 Java 处理器结构可以挖

掘出更多的指令级并行。

4.2 性能加速

实验中,假定所有 Java 字节码指令以单周期执行,来获得 ILP 性能加速比。图 3 给出了 JAM 处理器与基础的单发射 Java 堆栈处理器比较的 ILP 加速比。结果表明 JAM 处理器可获得从 78% 到

173% 的指令级并行增强,这也表明了 JAM 处理器能够获得显著的平均 ILP 性能加速。

另外,采用与 PicoJava-II^[12] 相同的指令执行周期进行仿真,图 4 显示了 JAM 处理器与单发射 Java 堆栈处理器相比较的 CPI 加速比。对于所有的测试程序, JAM 处理器获得的 CPI 加速比在 13% 到 78% 之间。

表 2 JAM 处理器执行中指令并行执行的百分率

基准测试程序	指令并行执行百分率			
	单发射 指令组	两发射 指令组	三发射 指令组	四发射 指令组
Compress	67.37	15.43	10.78	6.42
Db	79.97	14.98	3.78	1.27
Jack	79.54	14.22	3.89	2.35
Javac	72.85	21.87	4.24	1.04
Jess	81.51	13.47	3.26	1.76
Mpegaudio	43.26	16.53	6.78	33.43
Mtrt	87.92	9.67	1.55	0.86

从实验结果中发现,Mpegaudio 能够获得更高的性能加速比,这是因为与其他程序相比,它有更多的指令被并行执行。还注意到有三个程序:Compress, Mpegaudio 和 Linpack,得到了比其他程序更好的 CPI 加速比结果。经过分析,发现它们都是计算密集型的。

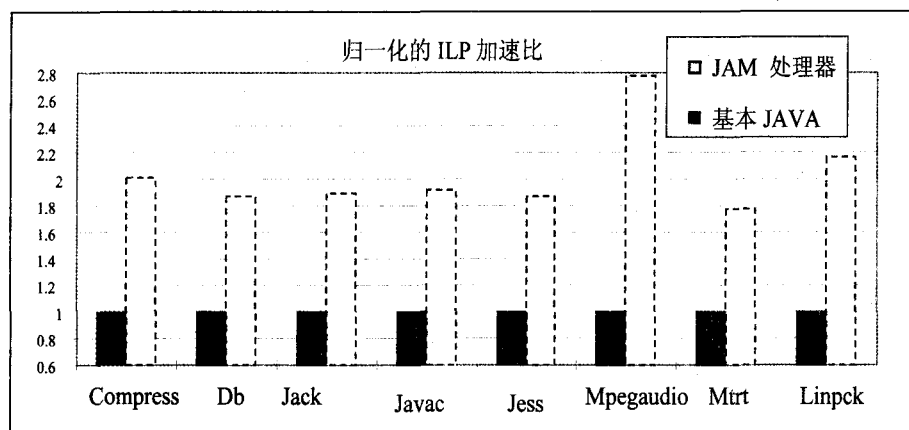


图3 JAM处理器的ILP加速比

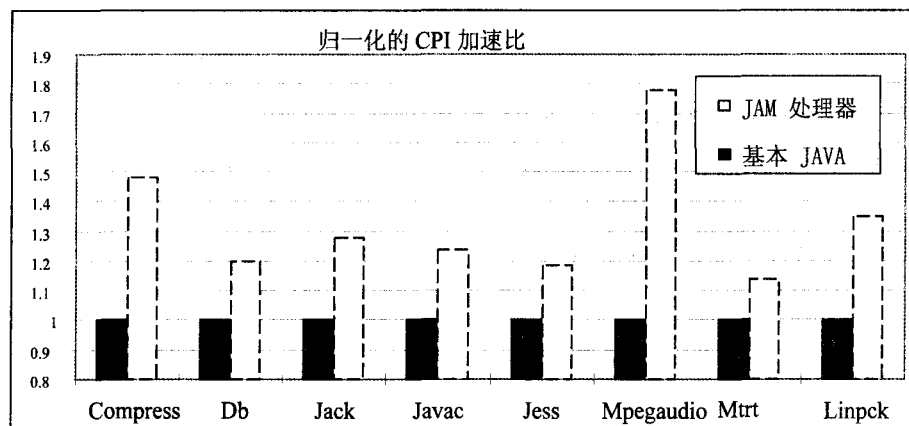


图4 JAM处理器的CPI加速比

这也表明JAM处理器更适合用于计算密集型程序。除此以外,对8个程序计算平均CPI增益,得到31%。

5 结束语

给出了一个使用硬件抽象机的JAVA高性能处理器——JAM处理器逻辑设计,描述了堆栈硬件抽象机的工作过程。该处理器使用硬件抽象机识别指令级并行,并且使用了一个通用的RISC执行引擎。通过软件仿真验证,确认JAM处理器可获得从78%到173%的指令级并行增强,提高处理器性能平均31%。

提出的硬件抽象机技术可以进一步扩展,具体结合动态指令翻译,利用现有的处理器执行引擎,动态执行不同的机器代码。同时,基于硬件抽象机的处理器设计方法,也可以用于执行CISC指令集计算机,诸如X86^[14]处理器。使用现有的高性能处理器内核(如MIPS执行引擎),可以实现与X86兼容的微处理器结构。这是今后的研究方向之一,同时它也有助于实现自有知识产权的X86微处理器。

参考文献:

[1] 王得利,高德远,张骏,等. 32位嵌入式CISC微处理器设计[J]. 计算机科学, 2009, 36(5):291-295.

[2] 刘振宇,齐家月. 高性能RISC微处理器硬件仿真器设计[J]. 计算机研究与发展, 2004, 41(8): 212-218.

[3] 张伟功,段青亚,刘曙蓉,等. 一种适于16位RISC处理器的伪四级流水结构研究[J]. 微电子学与计算机, 2008, 25(1):73-75.

[4] 张英武,袁国顺. 32位嵌入式RISC处理器的设计与实现[J]. 微电子学与计算机, 2008, 25(6):14-17.

[5] 张英武,郭天雷,袁国顺. 高可靠微处理器的设计[J]. 微电子学与计算机, 2009, 26(1):281-286.

[6] Stallings W. 计算机组织与体系结构性能设计[M]. 第6版. 张昆藏,等译. 北京:清华大学出版社, 2005.

[7] McGhan H, Connor M O'. PicoJava: A Direct Execution Engine for Java Bytecode[J]. IEEE Computer, 1998, 31:22-30.

[8] Radhakrishnan R, Talla D, John L K. Allowing for ILP in an Embedded Java Processor[C]//Proceedings of the 27th International Symposium on Computer Architecture. [s. l.]: [s. n.], 2000:294-305.

[9] Wang H C, Yuen C K. Exploiting Dataflow to Extract Java Instruction Level Parallelism on a Tag-based Multi-Issue Semi In-Order (TMSI) Processor[C]//IEEE International Parallel & Distributed Processing Symposium 2006, Rhodes, Greece:[s. n.], 2006.

[10] 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.

[11] Connor M O', Michael J. picoJava-I: The Java Virtual Machine in Hardware[J]. IEEE Micro, 1997, 17:45-53.

[12] Diehl S, Hartel P, Sestoft P. Abstract machines for programming language implementation[J]. Future Generation Computer Systems, 2000, 16:739-751.

[13] Lee J, Smith A J. Branch prediction strategies and branch target buffer design[J]. IEEE Computer, 1984, 17:6-22.

[14] Hu S L, Kim I, Lipasti M H. et al. An approach for implementing efficient superscalar cisc processors[C]//In Proc. of the 12th International Symposium on High-Performance Computer Architecture (HPCA-12). Austin, TX:[s. n.], 2006.