文章编号:1001 - 9081(2010)11 - 2873 - 03

基于嵌入式 Java 处理器的高速图像处理

朱明凯,高振华,柴志雷
(江南大学 物联网工程学院,江苏 无锡 214122)
(lyuewusheng@163.com)

摘　要:Java 技术正越来越受到图像处理研究人员的关注,希望以此提升开发效率,增强可移植性。但软件方式的 Java 虚拟机运行速度慢、实时性差,无法满足图像处理复杂计算对性能的需求。为此,提出一种以硬件方式直接执行字节码的 Java 处理器结构,并实现了其模拟器及预处理器构成完整测试平台。从实验结果可看出:该平台的执行效率是虚拟机方式的 860 倍,表明将 Java 处理器用于嵌入式图像处理将是一种可行选择。

关键词:Java 虚拟机;Java 处理器;图像处理;嵌入式系统
中图分类号: TP391.41　　**文献标志码**:A

High speed image processing based on Java processor for embedded systems

ZHU Ming-kai, GAO Zhen-hua, CHAI Zhi-lei
(*School of Internet of Things Engineering, Jiangnan University, Wuxi Jiangsu* 214122, *China*)

Abstract: Currently, in order to improve development efficiency and enhance portability, Java technology is paid more attention by image processing researchers. However, the Java virtual machine implemented by software, meaning low speed and poor real-time performance, can not meet the image processing demand of complex computations on performance. So a Java processor architecture implemented by hardware was presented, which can execute byte code directly, and the monitor and pre-processor were implemented at the same time, making up a complete test platform. The experimental results show that the efficiency of the platform is 860 times as the same as the Java virtual machine and applying Java processor to image processing for embedded systems would be a viable option.

Key words: Java virtual machine; Java processor; image processing; embedded system

0　引言

由于 Java 具有面向对象、强大的类库支持、多线程机制、垃圾回收机制,以及异常处理机制等其他许多语言不具有的一系列特性,使得 Java 在图像处理领域的应用及研究正受到越来越多的关注。本文在应用广泛的嵌入式系统上进行 Java 图像处理的研究,而图像处理是机器视觉和视频处理等研究的基础,因此,嵌入式系统上的图像处理研究具有很大的实用价值和研究意义。针对 Java 技术在图像处理中的应用,人们从不同的角度进行了研究:比如 NIH 开发的开源 Java 图像处理分析软件 ImageJ[1];Vision Systems Group 的 Netvision[2],Netvision 提供了基于 Java 的图像分析和软件开发环境,开发人员可以使用预定义的图像处理算法库来创建机器视觉解决方案;而 Sun 的 JMF(Java Media Framework API)[3] 则是通过扩展标准的 Java 平台(J2SE),提供功能强大的可扩展的工具包和跨平台技术以支持多媒体应用的开发,可以使音频,视频和其他实时媒体添加到以 Java 技术构建的应用程序和小程序中。

传统的 Java 应用程序一般由 Java 虚拟机采用解释的方式来执行,然而这种方式却极大地降低了 Java 程序的执行速率。为了提高 Java 程序的执行效率,许多研究采用硬件加速的方式来提高性能,如 ARM 的 Jezelle[4] 通过执行一些频繁出现的指令来提高整体性能;Sun 的 MAJC、PicoJava/PicoJava-II 等[5] 直接以硬件方式执行字节码。由于嵌入式系统多数都是对执行时间有苛刻要求的实时系统,因此不仅要考虑以硬件方式加速字节码执行,更需考虑系统的实时性能,即运行时间的可预测,这方面的工作有 JOP[6]、aJile system[7] 的 aJile-80、aJile-100、JPOR[8] 等。

为了探索 Java 处理器对图像处理的有效支持,本文在前面工作[8-9] 基础上,针对计算性能要求较高的图像处理应用,将处理器数据通路扩展到 32 位,并采用五级流水线结构提升运行效率。该处理器能够直接以硬件的方式执行 Java 字节码,同时还具备以下特点:1)不支持类的动态装载,而采用类转换器[10] 对图像处理应用程序进行预处理,生成适合 Java 处理器直接执行的内存映像;2)已装载的类以处理器可直接执行的二进制形式存在存储器中;3)不支持垃圾收集;4)不支持接口方法调用;5)提供精简的 Java 库。使得在 Java 处理器的运行环境中,可以对给定应用程序各个代码块的最坏情况执行时间进行准确统计。为了方便收集各种统计数据,本文采用模拟器的方式实现了这种 Java 处理器并对图像应用程序进行了处理。

1　JPOR 处理器体系结构及其 Java 平台

为了确保 Java 平台的实时性,采用转换器 CConverter[10] 和 Java 处理器相结合的方式,将 Java 应用程序分两阶段执行。在初始化阶段,转换器 CConverter 读取标准 Class 文件,

收稿日期:2010 - 04 - 18;**修回日期**:2010 - 08 - 04。
基金项目:国家自然科学基金资助项目(60703106);中央高校基本科研业务费专项资金资助项目(JUSRP30907)。
作者简介:朱明凯(1989 -),男,山东枣庄人,软件工程师,主要研究方向:Java 虚拟机、Java 图像处理、计算机仿真;　高振华(1986 -),女,山东枣庄人,硕士研究生,主要研究方向:Java 虚拟机、Java 图像处理;　柴志雷(1975 -),男,山西运城人,副教授,博士,主要研究方向:嵌入式系统、嵌入式实时 Java、机器视觉实时处理。

完成装载、连接等可以在运行前完成的工作以及其他一些非实时操作,生成可被 Java 处理器直接执行的内存映像文件,内存映像文件中所有的符号引用都已被替换为直接引用,所有影响运行实时性的操作也都得到处理。在运行阶段,Java 处理器只需要直接执行内存映像文件。这种方法保证了 Java 处理器的实时性,同时能够降低指令实现的复杂度、减少 CPI。

JPOR-32 是研究小组提出的一款 RISC 型 Java 处理器,采用 5 个流水段:取字节码、字节码排序、译码、执行、访存。在取字节码阶段,如指令 Cache 中有字节码可用,每一个时钟同时获取 4 字节的指令并存入指令寄存器。该阶段具有 Cache 的预取功能,由于 Java 字节码的长度大部分都只占 1 或 2 个字节,因此实际从取字节码阶段所获取的 4 字节指令往往不能全部进入执行阶段,因此缓存部件的设立很好地缓解了 Java 字节码不定长所造成的流水线停顿现象。流水线的第三级主要实现两大功能:其一,对复杂指令(如 invoke 指令)进行微指令转换;其二,对简单指令或微指令进行译码操作。第四级是指令的具体执行阶段,它将进行实际的运算操作并将结果放入操作数栈。第五级是访存阶段,将具体执行访存操作。

图 1 显示了 JPOR-32 基本数据通路,JPOR-32 的存储区域主要分为 4 大块:指令存储区、微指令存储区、操作数栈区以及数据存储区。其中操作数栈采用 16 个寄存器来实现。

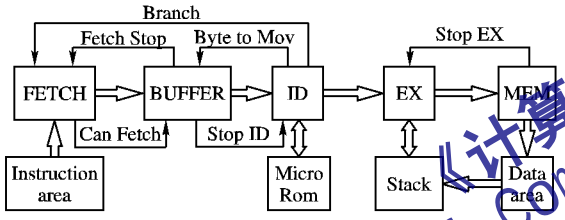


图 1 JPOR-32 的数据通路

2 JPOR-32 模拟器

为了尽早发现错误并方便的对处理器进行测试和优化,我们基于 JPOR-32 处理器的体系结构,采用软件模拟的方式对图像处理程序进行处理,模拟的部分主要包括字节码的转换、数据的寻址方式、操作数栈的设计、指令的译码和执行等。

标准的 Java Bytecodes 是 CISC 结构,存在一些复杂指令(如 new、invokevirtual、ireturn、athrow 等),这些指令需要不固定的、多个功能段的处理,因此很多情况下会处于空闲状态,影响处理器的效率,因此我们把这些复杂指令转换成微代码来代替执行,以简化数据通路,统一操作。

针对数据的寻址方式,把存储空间从逻辑上划分为如下几个部分:Constant Pool、Class、Static fields、Heap、Others,如图 2 所示。上述几部分起始地址和区域大小都要做到可调整,由 CConverter 把初始值写到内存某个位置,CPU 执行一段初始化程序取这些初始值送到相应寄存器。其中类(方法表)地址为 Cls_base + ClassAddr,对象地址为 heap_base + objectref(32-bit)。

在目前的设计中我们用 16 个寄存器(SR0 ~ SR15)实现操作数栈,即图 1 中的 Stack。由于操作数栈频繁在栈顶操作,操作数栈中的数据不用备份到 cache 中,寄存器要确保够用,这样的设计在拥有栈的特性的同时,还具备了寄存器的特性(多个寄存器能被同时访问和操作,可以提高执行效率)。

取指令和译码 从 CConverter 生成的内存映像文件中读取 JPOR 可以识别的机器指令进行译码,在对指令译码的同时,可以完成相关寄存器的读操作。将译码信号及相关数据通过寄存器保存起来,以方便执行阶段使用。

执行 对于 ALU 类型指令,执行后直接将结果写回寄存器;对于访存指令,计算存储器地址送到流水线寄存器 ALU_out。

对于整个模拟器的功能模拟,初步设计了 Branch. java(用于实现分支控制结构)、Constant. java(用于分配地址和空间常量)、Logic. java(用于实现逻辑运算和算术运算)、Mem. java(针对内存的操作和管理)、Native. java(针对本地方法的操作处理)、Monitor. java(协调和控制整个模拟器的运行)等用于硬件执行功能的模拟,其主要的执行方式是从内存映像文件中读取字节码和相关信息,然后进行译码和执行。

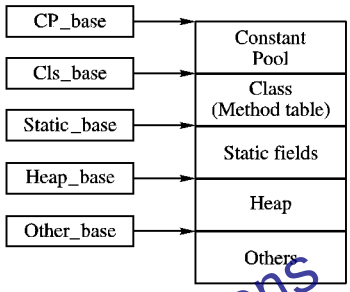


图 2 存储空间的逻辑分配

在对图像采集模拟时,发现基于 PC 机的输入输出都是调用系统的本地方法进行实现,即与支持的操作系统有关,因此需要设计自己的输入输出驱动程序,为了能够方便地进行模拟,同时又能在真正的图像采集处理中得到应用,我们把读取输入流方法设计成一个本地方法 data_input()(native method),而此方法对应着一个字节码 get_data,get_data 字节码的具体执行由模拟器进行解释执行,具体应用例子如下:

```
public class Processor{
  public static native byte data_input();
  public static void main(String args[]){
    byte pixel[] = new byte[15000];
    for(int i = 0; i < pixel.length; i++){
      pixel[i] = data_input();
    }
    int histogram[] = getHistogram(pixel);
    pixel = equalizeHistogram(pixel, histogram);
    histogram = getHistogram(pixel);
  }
  //直方图数据统计
  public static int[] getHistogram(byte[] pixel){
    …
  }
  //直方图均衡化
  public static byte[] equalizeHistogram(byte[] pixel, int[]
    histogram){
    …
  }
}
```

3 实验结果

随着嵌入式系统的发展和纺织装备的更新换代,使得图像处理技术在纺织行业的应用具有了实际意义,本文选择棉花杂质的图像识别和处理作为具体的应用实例。棉花在采摘、加工和运输等过程中经常混入一些杂质,为保证纱线质量,这些杂质在纺纱前必须彻底清除。一般的杂质,如沙石泥土等颗粒性物质,其比重或形状与棉纤维有较大差异,因此,

可以通过机械的方式进行除杂。但是像头发、细绳头、尼龙草等纤维状的杂质很难用机械的方法将它们清除,需要用其他方法进行清除,因此通常采用图像处理的技术来解决此问题。

针对棉花杂质的图像识别和处理,首先,使用 CCD 摄像机实时拍摄图像,然后图像处理模块存储数据并处理,根据处理结果判别是否含有异纤,接着发送信号给下位机。图像采集处理系统结构如图 3 所示。

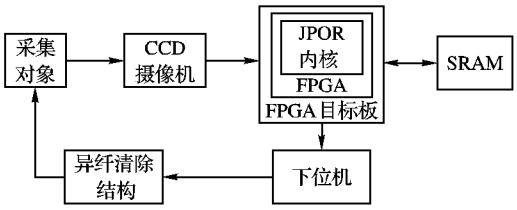
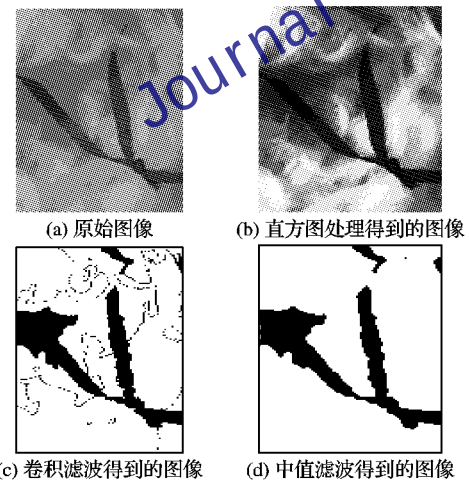


图 3　图像采集处理示意图

FPGA 控制模块是整个系统的核心部分,FPGA 主要负责图像采集及处理,由采集模块和处理模块构成。处理模块主要由 JPOR 处理器负责,图像处理主要包括直方图均化、卷积滤波、中值滤波、判断等。

图像采集模块主要任务是实时存储摄像机拍摄的图像数据;图像的预处理模块主要任务消除图像中无关的信息,恢复有用的真实信息,增强有关信息的可检测性和最大限度地简化数据,从而改善视觉效果便于人和机器对图像的理解和分析;图像处理模块的主要任务是把待检测对象的计算机感官指标与标准指标比较,从而确定待检产品是否合格。

实验拍摄的原始图像为含有棉花杂质的 8 位灰度图像,如图 4(a)所示,首先进行的是直方图处理,包括直方图统计和直方图均衡化两项操作,处理后的图像如图 4(b)所示,然后,进行的是卷积滤波处理,目的是为了对杂质进行特征提取,勾勒出杂质特征,如图 4(c)所示;最后,为了消除噪声,得到更好的图像处理效果,进行的是中值滤波处理,如图 4(d)所示。



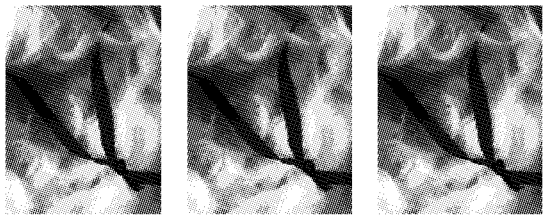
(a) 原始图像　(b) 直方图处理得到的图像

(c) 卷积滤波得到的图像　(d) 中值滤波得到的图像

图 4　棉花杂质图像处理结果

对比图 4(a)、(b)可发现,通过直方图处理后,杂质和棉花两者之间的对比度增强,棉花的灰度变弱成白色,杂质的灰度增强成黑色,杂质和棉花的特征差距增大。在此基础上,对图像进行卷积滤波处理,提取杂质的特征,得到处理后的图 4(c),比较发现,卷积滤波基本可以得到杂质的整个轮廓,可以清晰的分辨杂质,但是效果不是很好,有许多的噪声,因此还需要对图像进一步处理,进行过滤噪声操作,对此我们采用中值滤波,得到如图 4(d)所示的图像,细小的边缘部分和噪声被过滤,得到了杂质的完整轮廓。

为了进一步验证在处理器上进行图像处理是否可行,以及是否有所偏差,以直方图处理为例,在 Matlab、模拟器和标准 PC 机 3 种平台上进行了效果分析,如图 5 所示。直方图均衡化的目的是增强图像对比度,便于机器对图像的理解与分析。仔细比较 3 幅图像,并没有发现模拟器上的图像处理有所失真或者偏差,处理的效果和其他两种平台基本一致。



(a) Matlab处理　(b) 模拟器处理　(c) 标准PC机处理

图 5　棉花杂质图像处理对比

在嵌入式系统上实现图像处理研究,不仅需要能够得到很好的图像处理效果,而且更为重要的是,在嵌入式系统资源稀缺的情况下,性能是否能够满足实际应用的要求,如时间、资源的限制等。因此需要对在处理器平台上的图像处理研究,进行性能方面的分析。如表 1 所示。

表 1　不同平台的执行时钟周期对比

运行平台	执行的时钟周期
PC 机上的软 JVM	963 200 000
JPOR-32 模拟器	1 119 285

通过计算 PC 机上图像处理程序(示例程序所示)的运行时间乘以 PC 机的频率,粗略估计出 PC 机上的执行时钟周期;在模拟器上,通过计算执行的指令条数,粗略估计出执行的时钟周期,从而获得结果如表 1 所示。从表 1 可看出,基于 Java 处理器的程序执行所需时钟周期要远小于 PC 机上的软件 JVM 的执行周期,即在同样时钟频率下,Java 处理器的性能约为标准 PC 机的 860 倍,表明在此平台上的执行效率要远高于软件方式的 JVM。也说明利用硬件 JVM 方式进行嵌入式的图像处理具有高性能、低功耗的优势。

4　结语

本文在一个适用于低端嵌入式设备的实时 32 位 Java 处理器上研究针对棉花杂质图像的识别和处理。为了提高实时性,用类转换器 CConverter 对 Java Class 文件中影响实时性的行为进行预处理,生成适合 Java 处理器直接执行的内存映像文件,同时也简化了处理器的操作。通过软件方式进行 JPOR-32 处理器功能模拟,得到了正确的图像处理结果。从而说明在此处理器上进行图像处理应用的研究是可行的和有效的,具有很强的实用价值。下一步工作中,将继续完成卷积、特征提取、识别等模块,以最终构成完整的棉花异纤维识别处理系统。

参考文献:

[1] FERREIRA T A, RASBAND W. The ImageJ user guide, Version 1.43 [EB/OL]. [2010 - 04 - 30]. http://rsbweb.nih.gov/ij/docs/user-guide.pdf.

[2] WHELAN P F, SADLEIR R J T. A visual programming environment for machine vision engineers [J]. Sensor Review, 2004, 24(3): 265 - 270.

[3] Sun Microsystems. Java se desktop technologies [EB/OL]. [2010 - 02 - 18]. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html.

(下转第 2879 页)

仿真程序在内存中开辟一个堆,包含 FromSpace 和 ToSpace 半区,每个半区由 1 024 个块组成,每个块里有 32 个对象。这里假设对象大小都相同,这样一个半区最多可容纳 32 768 个对象。一个对象的地址由块序号和块内偏移唯一确定。

FromSpace 半区里对象存活率 50%,即有 16 384 个存活对象。为了取得一般性的测试结果,假设存活对象均匀地分布在半区之中。指向半区的根集合元素个数有 64 个。仿真程序运行时首先生成 FromSpace 半区能够容纳的全部 32 768 个对象,然后开始并行垃圾收集。仿真程序运行时测试参数包括在使用 1 ~ 4 个核心下垃圾收集每个阶段的耗费时间和总的收集时间。为了防止出现的偶然因素影响测试结果,分别在用于垃圾收集的线程数目为 1,2,3,4 的情况下各测试 10 次,然后取平均值作为最终测试结果,如表 1 所示。

表 1　测试结果

核心数	阶段时间/ms				总的时间/ms	性能提升/%	加速比
	标记	迁移地址	引用修正	移动对象			
1	2 016	3 876	3 995	3 987	13 874	—	—
2	1 209	2 338	2 439	2 402	8 388	39.54	1.65
3	898	1 784	1 978	1 786	6 446	23.15	2.15
4	814	1 548	1 769	1 657	5 788	10.21	2.40

从表 1 可以看出,收集 FromSpace 半区垃圾对象,当使用 2 个核心时垃圾收集时间从 1 个核心的 13 874 ms 减少到 8 388 ms,性能提升了 39.54%;当使用 3 个核心时垃圾收集时间从 2 个核心的 8 388 ms 减少到 6 446 ms,性能提升了 23.15%;当使用 4 个核心时垃圾收集时间从 3 个核心的 6 446 ms减少到 5 788 ms,性能提升了 10.21%。从图 4 可以看出,随着使用核心数目的增多,每增加一个核心程序性能提升幅度呈降低趋势。我们推测可能是随着收集垃圾的线程数目增加,线程间同步开销与共享变量同步开销增加所致,理论上应该还有提升的空间。

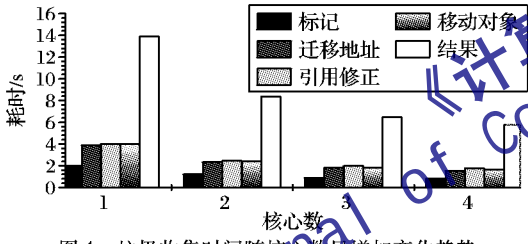


图 4　垃圾收集时间随核心数目增加变化趋势

3　结语

本文给出了一种把 LISP2 垃圾收集算法并行化用于节点复制垃圾收集的算法,克服了文献[1]中经过垃圾收集存活对象游离于不连续区域中的缺点,支持碰撞指针方式高效分配新对象。虽然本文算法在多核环境下运行测试,但在多处理器系统或单处理器系统下均可正常运行。下一步将尝试把这一算法集成进实际的虚拟机里面,如开放平台 Apache Harmony[3]。

参考文献:

[1] FLOOD C H, DETLEFS D, SHAVIT N, *et al*. Parallel garbage collection for shared memory multiprocessors [C]// Proceedings of the 2001 Symposium on Java Virtual Machine Research and Technology Symposium. Berkeley, CA, USA: USENIX Association, 2001, 1: 21 - 21.

[2] ABUAIADH D, OSSIA Y, PETRANK E, *et al*. An efficient parallel heap compaction algorithm [C]// The ACM Conference on Object-Oriented Systems, Languages and Applications. New York: ACM Press, 2004: 224 - 236.

[3] Apache harmony is the Java SE project of the Apache software foundation [EB/OL]. [2009 - 12 - 12]. http://harmony.apache.org.

[4] LI XIAO-FENG, WANG LI-GANG, YANG CHEN. A fully parallel LISP2 compactor with preservation of the sliding properties [C]// Languages and Compilers for Parallel Computing. Berlin: Springer, 2008: 264 - 278.

[5] JONES R, LINS R. Garbage collection: Algorithms for automatic dynamic memory management [M]. New York: John Wiley & Sons, 1996.

[6] WU MING, LI XIAO-FENG. Task-pushing: a scalable parallel GC marking algorithm without synchronization operations [C]// IPDPS 2007: IEEE International Parallel and Distributed Processing Symposium. Washington, DC: IEEE, 2007: 1 - 10.

[7] SIEBERT F. Limits of parallel marking garbage collection [C]// Proceedings of the 7th International Symposium on Memory Management. New York: ACM Press, 2008: 21 - 29.

[8] 周伟明. 多核计算与程序设计[M]. 武汉: 华中科技大学出版社, 2009.

[9] OpenMP3.0 规范[EB/OL]. [2009 - 12 - 12]. http://www.openmp.org/mp-documents/OpenMP3.0-SummarySpec.pdf.

(上接第 2875 页)

[4] STEELE S. Accelerating to meet the challenge of embedded Java [EB/OL]. [2010 - 03 - 21]. http://www.jp.arm.com/document/whitepater/pdf/Jazelle_White_Paper.pdf.

[5] Sun Microsystems. PicoJava-II: Java processor core[EB/OL]. [2010 - 03 - 21]. http://java.epicentertech.com/Archive_Embedded/Sun_Microsystems/Micro%20&%20Pico%20Java/picoJava-II.pdf.

[6] SCHOEBERL M. JOP: A Java optimized processor [EB/OL]. [2010 - 02 - 10]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.164.4497&rep=rep1&type=pdf.

[7] HARDIN D S. aJile Systems: Low-power direct-execution Java microprocessors for real-time and networked embedded applications [EB/OL]. [2010 - 03 - 10]. http://www.ajile.com/index.php?option=com_content&task=view&id=50&Itemid=71.

[8] CHAI ZHILEI, XU WENBO, TU SHILIANG, *et al*. Java processor optimized for RTSJ [EB/OL]. [2010 - 02 - 12]. http://downloads.hindawi.com/journals/es/2007/057575.pdf.

[9] CHAI ZHILEI, ZHAO WENKE, XU WENBO. Real-time Java processor optimized for RTSJ [C]// Proceedings of the 2007 ACM Symposium on Applied Computing. New York: ACM Press, 2007: 1540 - 1544.

[10] 柴志雷,高丽强,陈章龙,等. 一种用于硬实时 Java 处理器的类转换器设计及实现[J]. 小型微型计算机系统, 2006, 27(12): 2336 - 2340.