Journal of Computer Applications
计算机应用,2012,32(6):1730 - 1733
ISSN 1001-9081
CODEN JYIIDU
2012-06-01
http://www.joca.cn

文章编号:1001 - 9081(2012)06 - 1730 - 04
doi:10.3724/SP.J.1087.2012.01730

CPS 处理器中时间指令扩展研究及实现

高振华[1*],杨　帆[1],陈闻杰[2],柴志雷[1]
(1.江南大学 物联网工程学院,江苏 无锡 214122;　2.华东师范大学 软件学院,上海 200062)
(*通信作者电子邮箱 zzgzhh101@163.com)

摘　要:物理进程具有内在的并发及实时特性,因此发展信息—物理融合系统(CPS)需要计算进程能表达这类特性。而传统的计算模式为了方便用户逻辑描述,随着抽象程度的提高逐步丢弃了对时间特性的精确描述。在嵌入式 Java 处理器 JPOR-32 基础上面向 CPS 应用增加了时钟寄存器和时钟计数器,并根据程序员对时间特性的需求,结合异常机制扩展了四条时间指令,使得用户可根据不同需求对时间进行精确控制。最后通过采用时间指令后图像处理程序在该 CPS 处理器上的运行结果验证了该时间控制机制的可行性、正确性及精确性。

关键词:信息-物理融合系统;嵌入式系统;Java 处理器;异常;时间指令
中图分类号:TP391.41　　**文献标志码**:A

Implementation of precision timed instructions for systems processors in cyber-physical

GAO Zhen-hua[1*], YANG Fan[1], CHEN Wen-jie[2], CHAI Zhi-lei[1]
(1. *School of Internet of Things Engineering, Jiangnan University, Wuxi Jiangsu* 214122, *China*;
2. *Software Engineering Institute, East China Normal University, Shanghai* 200062, *China*)

Abstract: The physical processes is inherently concurrent and real-time, so to express these characteristics is required when developing Cyber-Physical Systems (CPS). While the traditional computing model for the convenience of the user logic description, along with the abstract degree rise, description on the time characteristic was gradually discarded. On the basis of the embedded Java processor JPOR-32, a clock register and a clock counter were added for the CPS applications in this paper, and combined with the exception mechanisms, four time instructions were extended to face different requirements of the temporal characteristics, which enabled the user to control the time accurately according to different demands. Finally, the feasibility, validity and accuracy of this time control mechanism have been verified by analyzing the running results of the image processing program that uses time instructions in the CPS.

Key words: Cyber-Physical Systems (CPS); embedded systems; Java processor; exception; time instructions

0　引言

信息—物理融合系统(Cyber-Physical Systems, CPS)[1]指的是网络化的计算系统与现实世界的物理过程紧密融合、相互作用、相互影响的系统。由于物理世界中各个不同物理过程从本质上具有并发性及实时性两个特性,而传统的计算模式也具有顺序执行和追求平均性能最佳两个明显特征,因此需要重新考虑适应 CPS 发展的软硬件结构。我们知道虽然用来实现处理器的硬件单元(如门逻辑或更大的计算单元)本身具有明确的时间特性,但为了追求整体性能最佳,现代处理器多数采用了流水线、分支预测、推测及乱序执行、高速缓存等技术。在提高处理器整体性能的同时,使得指令执行时间变得难以预测[2]。

因此,现在许多研究人员意识到单纯从软件层次上进行改进很难取得理想效果,必须从计算机系统结构的层次进行时间特性的改进。如 Anantaraman 等[3]的 VISA(VIrtual Simple Architecture)结构,以快速处理器提高处理性能,而以慢速处理器确保时间限制。Lee 等[4]则提出了一条允许程序员设置和访问时钟寄存器和计数器的"deadline"时间指令,并运用到他们的 PRET 机器上,后来又借用到基于 SPARC 指令集结构的多线程处理器上;最近他们又针对不同的程序需求提出了 4 条时间指令[2]来对代码段进行准确的时间控制。与此同时 JOP(Java Optimized Processor)[5]在不改变指令集的基础上,利用时钟计数器实现了对 I/O 接口的时间控制;Andalam 等[6]在 PRET-C 机器上扩展了 5 条 C 语言指令用于 C 语言程序段的时间控制。据我们所掌握的资料,国内在 CPS 处理器方面的研究还不多见,直到 2011 年才首次出现针对 CPS 环境下的时间特性的研究的综述性文献[7]。

本文在前面基于 WCET(Worst-Case Execution Time)可预测的 Java 处理器 JPOR-32[8-9]基础上,在体系结构上增加了时钟寄存器与时钟计数器,结合 Java 语言的异常处理机制提出了 4 条时间扩展指令,实现了程序员针对不同的时间控制需求进行编程,以此实现对时间的精确控制。

1　基于 JPOR-32 的实时 Java 平台

实时 Java 平台 JPOR-32 由 Java 虚拟机(Java Virtual Machine, JVM)、类库和应用程序接口(Application Programming Interface,API)组成。为了确保 Java 平台[8]的实时性并降低指令实现的复杂度,采用的是转换器(Converter)和 Java 处理器相结合的方式,把 Java 应用程序分两个阶段来

收稿日期:2011-12-09;**修回日期**:2012-01-31。　**基金项目**:国家自然科学基金资助项目(60703106)。
作者简介:高振华(1986 -),女,山东枣庄人,硕士研究生,主要研究方向:嵌入式 Java 图像处理;　杨帆(1986 -),男,湖北荆门人,硕士研究生,主要研究方向:嵌入式实时处理器;　陈闻杰(1977 -),男,浙江金华人,副教授,博士,主要研究方向:信息—物理融合系统与实时系统;柴志雷(1975 -),男,山西太原人,副教授,主要研究方向:信息—物理融合系统处理器系统。

执行。如图 1 所示转换器读取 Java 文件编译后的 class 文件和类库,完成装载、解析和连接、字节码的优化、空间分配等一些非实时操作,生成可被 Java 处理器直接执行的内存映像文件。由于 JPOR-32 处理器是在 FPGA 上实现,读取的文件格式为.coe,内存映像文件经过相应的格式转换后被处 JPOR-32 直接读取和处理。JPOR-32 采用取指、缓存、译码、执行及访存 5 级流水段来执行 JPOR-32 的可执行文件。由于 JPOR-32 执行 Java 字节码指令的时钟周期可以被准确检测,因此所执行程序的 WCET 可预测,具体的指令周期计算公式会在第 4 章中列出。

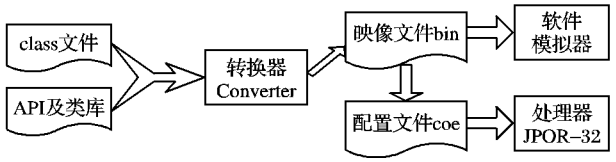


图 1　实时 Java 平台

2　异常处理机制

异常处理是在异常引发后,应用程序能够自动转移到相应的异常处理模块中去进行修复处理。Java 中的异常是以对象来表示的,异常对象主要有两个来源[9]:一是在方法中抛出了此异常,但是在方法定义中并未使用 throws 声明的异常称为 Unchecked Exceptions。这些异常与某些种类的程序代码有特定的对应关系,且无论是否会被捕获和处理总会被抛出。二是在方法定义时用 throws 声明的异常,称为 Checked Exceptions。用户可以通过编写带有 throw 关键字的程序来明确指定要在何时强制 Java 虚拟机抛出此异常。

一般的 Java 异常处理机制是利用 catch 语句捕获由 try 语句块抛出的异常,并由 catch 语句块来处理此异常。如果在 try 语句块中抛出异常,athrow 字节码指令执行的时候会抛出这个异常,与异常表比对捕获(catch)并且用某种合理的方法处理该异常。

根据以上知识可以把程序中的异常分为系统抛出系统捕获、系统抛出用户捕获、用户抛出用户捕获三类形式。系统抛出系统捕获是一个内部处理过程,不能在程序中很好地显式表达时间控制机制,本文选用系统抛出用户捕获和用户抛出用户捕获两种形式来分析时间控制机制。

3　扩展时间指令及时间控制

3.1　处理器体系结构上的改进

处理器在原有 JPOR-32 的体系结构[8-9]上针对 CPS 应用新增了时钟寄存器和时钟计数器:时钟寄存器存放的是程序员在编程过程中所赋的 CPU 时钟周期的时限值,时钟计数器则是负责对程序段的 CPU 时钟周期进行计数。Lee 等[2,4]在研究的过程中使用的计数器均为递减,由于计数器是对 CPU 时钟周期进行计数,且一般的计数器都是自动递增计数,因此 CPS 处理器使用的计数器是严格按照 CPU 时钟周期递增的。时钟寄存器每赋一次时限值,计数器就会重新从零开始递增计数,因此时间控制机制的触发与否是由时钟寄存器值是否为正整数决定的。

3.2　扩展时间指令

实时 Java 平台处理的是经过编译的 class 文件,在程序中直接使用扩展指令会导致编译不通过,因为在编程时扩展指令是作为本地方法来声明和使用。转换器对方法进行装载并解析时,会先判断此方法是否为本地方法,如果为本地方法,则会提取方法名作为指令并转换成相应的指令字节码,因此在处理器执行转换后的内存映像文件时,读取此字节码并进行译码执行;如果不是本地方法,则对此方法进行加载和解析。此种使用方式在无需改变编译器的情况下,对指令集进行了扩展。下面列出了在 JPOR-32 指令集基础上 4 条扩展时间指令,在程序中以本地方法来声明和使用。

1) Deadset。时限值作为参数给出,对时钟寄存器赋值,时钟计数器开始计时。

2) Deadline。如果在限制时间内代码段执行完毕结束,对时间准确性要求高的程序,则等待至时限到达。

3) TimeInt。对时钟寄存器和计数器初始化,或者在限制时间内代码段执行完毕,对实时性要求高的程序,无需等待则直接将时钟寄存器和计数器重新初始化为 0。

4) TimeOver。通过判断时钟寄存器值和计数器值的大小来判定是否超时,超时则返回真,否则为假。

指令周期是取出并执行一条指令的时间,由若干个 CPU 时钟周期组成。处理器从复位开始读字节码到缓冲器到译码段共需要花费 20 个 CPU 时钟,假设处理器执行的字节码指令长度为 N, 每条字节码指令从译码开始到执行完成所用 CPU 时钟数为 T(在文献[8]中以表格的形式被列出)。在处理器中单条字节码指令的指令周期[8]为:$20 + T + (N-2) + [(T+1)-3]$。由于 JPOR-32 采用的 5 级流水线结构,且各指令长度及操作不同,使得指令周期也就不尽相同。程序员可以对代码段进行指令分析,根据上述算式及流水线结构分析得出代码段指令执行所用的 CPU 时钟周期数,程序员依据需求选定一个时限值 T, 通过传递方法系数的方式设置此时限值。

3.3　时间控制

在代码段前利用 Deadset(int T) 设置时限值 T,此时时钟寄存器被赋值 T, 系统对程序进入时间控制,时钟计数器则开始计数指令消耗的 CPU 时钟数。由上文可看到对于不用的异常类型,因为处理方式和格式的不同使用的代码也有所不同。针对在时限内代码段执行完毕的情况而言,即使对于异常类型相同,对实时和准确性要求不同,使用的程序代码也会有所差异。因此紧随代码段后的主要操作是针对时限内代码段是否执行完毕的情况进行处理。如果未超时,针对准确性要求高的程序 Deadline 会使程序处于等待状态,直至计数器值与时钟寄存器值相等,而 TimeInt 针对实时性要求高的程序直接对计数器和时钟寄存器初始化,免于等待。如果超时,Checked Exceptions 需要显示抛出异常,因此相对于 Unchecked Exceptions 多出一条 TimeOver 指令来判断是否超时。但是这种异常的抛出方法是在代码段所有的代码都被执行以后才会被判断发现超时,不具有准确性和实时性。而 Unchecked Exceptions 则是当时钟计数器的值大于时钟寄存器值时立即停止运行,系统抛出异常,这恰好弥补了上述缺点,确保了异常发现和处理的准确性和实时性。

在此针对异常和时间操作类型,提出了两种不同的对程序段进行时间控制的程序样式。针对 Unchecked Exceptions 的程序样式如下(此处以系统异常 RuntimeException 为例):

```
try{
  Deadset( int T);
  //code segment
  Deadline( ) / TimeInt( )
```

```
} catch( RuntimeException e) {
//code
}
```

而针对 checked Exceptions 的程序样式如下(此处为自定义异常类 DeadlineMissed):

```
try{
  Deadset( int T)
  //code segment
  If( TimeOver( ) )
  throw new DeadlineMissed( ) ;
  else
  Deadline( ) / TimeInt( ) ;
} catch( DeadlineMissed e) {
//code }
```

JPOR-32 处理器根据超时抛出异常类型的不同设计了两种反应机制,在计数器递增时 Unchecked Exceptions 与 Checked Exceptions 相比需要增添实时判断是否超时的机制,但在程序编辑上相对比较简洁。而对于 Checked Exceptions 的判断机制不具有实时性,编辑上也复杂一些。所以在编程过程中推荐使用 Unchecked Exception 的样式。

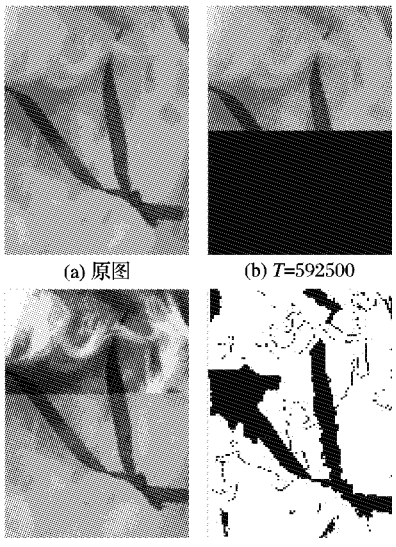
4 应用示例及实验结果

程序员在编写有关时间控制的应用程序时,首先要根据在时间特性上需求选用必需的扩展时间指令,并逐一以本地静态方法来声明,便于在程序中引用。然后是根据指令分析及指令周期算式得出代码段所用的总 CPU 时钟数,进而根据需求确定时限值 T(T 以 CPU 时钟周期为单位)。在程序中可以通过设置不同的时限值,经过不同的处理分支来得到不同的结果,以此来得出时间控制机制的正确性、准确性及实时性。

针对四种样式,利用两种处理机制和三个时限值分别对相同的代码段分别进行了测试,在此本文依据操作类型一中抛出系统异常的样式,以一个棉花异纤维检测中的图像处理程序输出的图像结果来具体说明指令及机制的实现。

```
public class TimeTest{
  public static native byte get_read( ) ;
  public static native void Deadset( int T) ;
  public static native void Deadline( ) ;
  public static void main( String args[ ] ) {
    byte a[ ] = new byte[ 15000] ;
    try{
      Deadset( 592500) ;
      for( int i = 0; i < a. length; i ++ ) {
        a[ i] = get_read( ) ;                    //读取图像数据
      }
      int historgrame[ ] = getHistogram( a) ;
      a = equalizeHistogram( a, historgrame) ;
      Deadline( ) ;
      a = convolve( a) ;
      threshold( a, 35) ;
    } catch( RuntimeException e) {
    System. out. println( "there is an Exception! ") ;
    }
  }
  //直方图数据统计
  public static int[ ] getHistogram( byte[ ] pixels) {
  …
  }
  //直方图均衡化
  public static byte[ ] equalizeHistogram( byte[ ] pixels, int[ ] histogram) {
  …
  }
  //卷积滤波
  public static byte[ ] convolve( byte[ ] pic) {
  …
  }
  //边缘特征提取
  public static void threshold( byte[ ] pixels, int level) {
  …
  }
}
```

本次实验所用图像如图 2(a)所示规格为 100 × 150,处理顺序为读取原图像—直方图均衡化—卷积滤波—边缘特征提取,根据处理结果确定杂质的位置及形态,以便进行清理。上述程序对前两个阶段进行了时间控制,设置的时限值 T 为 592 500,并赋值给时钟寄存器,且时钟计数器开始递增计数。分析循环读取图像数据代码段指令可知,读一个图像数据所用 CPU 时钟周期总数为 79,读取整幅图像所需周期为 79 × 15 000 = 1 185 000,程序设定的时限值 592500 = 118 500/2,由此分析得在时钟计数器递增至 592 500 时读取了半幅图像信息并存储在数组 a 里。再次循环读取下个图像数据时,时钟计数器递增为 592 501 > 时钟寄存器值,超出时限系统会抛出 RuntimeException 异常,转到 catch 语句块执行输出语句。此时实验数组 a 成像如图 2(b)所示。



(a) 原图 (b) T=592500

(c) T=3500000 (d) $T \geqslant$4505493

图 2 图像处理结果

如果时限值 T = 3 500 000 时,图像读取完毕且进行了一部分直方图均衡化处理,当时钟计数器递增至 3 500 001 时超时停止图像处理并进入异常处理机制。此时数组 a 里一部分是直方图均衡化处理之后的数组,另一部分为未处理的原图像信息,所以程序运行后数组 a 的成像如图 2(c)所示。当 $T \geqslant$4 505 493 时,直方图均衡化处理代码段执行结束,此时运行 Deadline 指令,将当前时钟计数器值与时钟寄存器值进行比较,如果相等,则对时钟计数器和寄存器初始化后继续执行卷积滤波和特征值提取处理;如果不等,则等待 N 个(N = 时钟寄存器值 - 时钟计数器值) CPU 时钟后继续执行图像处理。在这种时限值范围内,数组 a 成像如图 2(d)所示。根据时限值范围的不同可以得出不同的图像运行结果,表 1 列出了所有时限值范围的运行结果。

表 1　不同时限值的运行结果

时限值 T 范围	运行结果
$T < 1\,185\,000$	超时,读取的图像信息不完整, 输出提示信息“there is an Exception”
$1\,185\,000 \leqslant T < 4\,505\,493$	超时,图像信息读取完毕, 直方图均衡化处理不完整, 输出提示信息“there is an Exception”
$T \geqslant 4\,505\,493$	未超时,图像均衡化处理后等待 $N(N = T - 4\,505\,493)$ 个 CPU 时钟后, 进入卷积滤波和特征值提取处理

为了验证这种实现方式是否导致处理器的执行效率降低或实时性发生改变,特地在主频为 2.8 GHz 的 PC 机和 JPOR-32 处理器上分别运行图像处理程序,执行所用的 CPU 时钟分别为表 2 所示。

表 2　不同平台的执行时钟周期

运行平台	CPU 时钟周期数
PC	543 477 994
JPOR-32	4 505 493

由 CPU 时钟周期数的比较可得:JPOR-32 处理器的处理效率远高于普通 PC,时间控制机制是在处理器执行效率高、准确性强、实时性得以保证的前提下实现的。这种时间控制机制实现了除机器控制、设置断点以外的程序执行时间控制,对棉花异纤维检测过程中遇到的异物能够准确及时的清理提供了程序支持。

5　结语

本文结合异常处理的研究,在原有 JPOR-32 指令集基础上提出了 4 条时间指令,并在硬件上增添了时钟寄存器和时钟计数器,时钟寄存器值的改变影响了时间控制机制的开和闭合,实现了程序员根据自身需求来对程序段进行时间控制。并以图像处理程序验证了时间指令和时间控制的可行性、精确性。对于 Checked Exception 虽然可以用户自定义异常,但使用指令较多,程序写起来较复杂,超时异常的处理不具有实时性。Unchecked Exceptions 实现了超时异常抛出的实时性,同时相对地简化了指令集和程序。并在棉花异纤维检测处理程序中能够准确实时地得到处理结果,便于异物的剔除。

参考文献:

[1] Wikipedia. Cyber-physicalsystem[EB/OL]. [2011-10-20]. http://en.wikipedia.org/wiki/Cyber-physical_system.

[2] LIU I, LICKLY B, LEE E A, *et al*. Poster abstract: Timing instructions - ISA extensions for timing guarantees[EB/OL]. [2011-10-22]. http://chess.eecs.berkeley.edu/pubs/542/liulickly_rtas2009_poster.pdf.

[3] ANANTARAMAN A, SETH K, PATIL K, *et al*. VIrtual Simple Architecture (VISA): exceeding the complexity limit in safe real-time systems[C]// Proceedings of the 30th Annual International Symposium on Computer Architecture. New York: ACM Press, 2003:350-361.

[4] LICKLY B, LIU I, LEE E A, *et al*. Predictable programming on a precision timed architecture[C]// Proceedings of the International Conference on Compilers. New York: ACM Press, 2008: 137-146.

[5] SCHOEBERL M, PATEL H D, LEE E A. Fun with a deadline instruction [EB/OL]. [2011-10-24]. http://www.jopdesign.com/doc/deadline.pdf.

[6] ANDALAM S, ROOP P, GIRAULT A, *et al*. PRET-C: A new language for programming precision timed architectures [EB/OL]. [2011-10-26]. http://www.ece.auckland.ac.nz/~roop/pub/2009/andalam09.pdf.

[7] 陈淑敏, 黄云峰, 黄振滔. CPS 环境下时间特性的研究[J]. 信息与电脑, 2011(3):170-172.

[8] 杨帆, 高振华, 柴志雷. WCET 可预测的 Java 指令集硬件实现[J]. 计算机工程, 2012, 38(1):14-18.

[9] GAO ZHEN-HUA, YANG FAN, CHAI ZHI-LEI. WCET predictable exception mechanism for real-time Java platform [EB/OL]. [2011-10-30]. http://www.asaas.org/ICNECS2011/N0290.pdf.

(上接第 1729 页)

混合并发模式也有一定的缺陷:1)通常嵌入式系统的硬件资源有限,而热方法表会占用额外的内存,编译和执行的并发也会增加处理器的负担。因此该模式不适用于配置较低的系统。2)当 Dalvik 中运行需要并发大量线程的 Java 应用,该模式会因为难以预测 Java 代码的执行顺序而无法提前编译,进而退化成 JIT 模式。

因此,针对混合并发模式现有的不足,改进 Java 程序执行过程的预测算法以及对热方法表的动态维护,将是后续研究工作的重点。

参考文献:

[1] 姚昱昱, 刘卫国. Android 的架构与应用开发研究[J]. 计算机系统应用, 2008, 17(11):110-112.

[2] Google. Android project official website[EB/OL]. [2011-11-12]. http://www.android.com.

[3] 李锡祚, 霍华, 胡冠. 嵌入式 Java 虚拟机优化研究[J]. 计算机应用与软件, 2009, 26(7):257-259.

[4] CRAMER T. Compiling Java just in time[J]. IEEE Micro, 1997, 17(3):36-43.

[5] 杨丽洁. 面向 Linux NC 的 Java 虚拟机性能优化[J]. 计算机应用, 2006, 26(5):1152-1154.

[6] CHAN B, ABDELRAHMAN T S. Run-time support for the automatic parallelization of Java programs[J]. Journal of Supercomputing, 2004, 28(1):91-117.

[7] 于勐, 陈贵海, 阳雪林, 等. JAVA 并行化编译器 JAPS-Ⅱ[J]. 软件学报, 2002, 13(4): 739-747.

[8] McFADDEN A. Dalvik Porting Guide[EB/OL]. [2011-11-08]. http://android.git.kernel.org.

[9] 魏达, 金英, 张晶, 等. 基于开源 JVM 的安全策略强制实施[J]. 电子学报, 2008, 37(4A):35-41.

[10] LIM H, PARK H. A method for detecting the theft of Java programs through analysis of the control flow information[J]. Information Software Technology, 2009, 51(9): 1338-1350.

[11] 蒋曹清, 应时, 倪友聪, 等. 一种 Java 程序 Chopping 方法[J]. 计算机科学, 2011, 38(1):150-155.

[12] 苏超云, 柴志雷, 涂时亮. 实时 Java 平台的类预处理器研究[J]. 计算机工程, 2010, 36(7):246-248.

[13] 肖增良, 何锫, 康立山. 并行程序验证的调度策略[J]. 计算机工程与应用, 2009, 45(11):39-41.

[14] 陶荣, 何锫, 黄道昌. 基于依赖分析的并行化验证策略[J]. 计算机工程, 2010, 36(12):64-65.