基于 PRET 的编程模型①

李晓飞, 陈香兰

(中国科学技术大学 计算机学院, 合肥 230039)

摘 要: 时间可预测性在信息物理系统设计领域正变得越来越重要, 目前时间可预测性系统的设计分为编程模型和体系结构两个层次, 编程模型的研究往往是基于传统 RTOS 而提出新的时间模型, 体系结构层则是现有体系结构, 设计的具有时间属性的指令集、流水线等等. 基于时间可预测体系结构 PRET 和可预测时间模型 LET 的研究, 提出将 PRET 和 LET 模型相结合的编程模型, 并通过分析和实验证明了这种设计的可行性和优势, 进一步证明了时间在系统设计中的重要性.

关键词: 时间可预测性; 精确时间机; 逻辑时间模型; 系统设计

PRET-Based Programming Model

LI Xiao-Fei, CHEN Xiang-Lan

(School of Electronic Science and Technology, University of Science and Technology of China, Hefei 230039, China)

Abstract: In CPS related embedded system design area, time predictability is becoming increasingly important. The system of time predictability can be divided into two level: the programming level based on the traditional RTOS or platform and the architecture level extended with time behavior. In this paper, based on the research between the predictable architecture and the predictable programming model, and a new programming model based on both PRET(PRecised Timed machine) was proposed. Finally an implementation based on this new model was exhibited, through this exhibition we can find that this new model had advantages to the Giotto model.

Key words: time predictability; precised timed machine; logical execution time; system design

1 引言

时间可预测性目前在学术界还没有统一的定义. Grund 给出了一种比较宽泛的定义[1], 但无法通过穷举的方法找到定义中的最大值和最小值, 无法给出其确切的时间可预测性值. 不过通过静态分析的方法给出其执行时间的上界和下界, 通过减少两者之间的差值, 肯定会提高定义中的时间时间可预测性. 因此该定义并不能给出测量时间可预测性的方法, 但是给出了提高时间可预测性的思路.

为了提高系统时间可预测性, 从体系结构层次出发, Lee 提出了精确时间机的概念[2], 也就是 PRET (Precison Timed Machines), 他和 S.Edward 设计出一种基于 XilinxSpartan 3 XC3S200 平台的 PRET, 不过由于其完全了抛弃了处理器的流水线功能, 效率并不高. 之后 Ip 和 Liu 分别提出了基于 SPARC 指令集[3]和 ARM 指令集[4]的 PRET, Liu 的 PTARM 采用了五级细粒度多线程流水线, 大大提高了效率, 然而由于只是将线程进行简单隔离, 并不能进行线程的调度, 当线程数量小于 4 的时候, 就会有大量空闲的时钟周期浪费. Zimmer 在 PTARM 之上提出了 FlexPRET 平台[5], 该平台解决 PTARM 的线程调度问题.

编程模型方面, 时间可预测性模型的研究也在快速发展, 最早由 G.Berry 提出的同步语言[6], 通过假设程序的执行时间为瞬时, 更加容易确定程序执行的顺序, 进而提高其确定性, 不过由于使用同步语言设计程序比较复杂, 需要验证满足定点原理(fixed-point)[7], 使用起来并不方便. Kirsch 在同步语言基础上, 提出了逻辑执行时间(LET)的概念[8], 通过使用 LET 时间模型

① 基金项目:国家自然科学基金(61379040,61272131);江苏省自然科学基金(SBK2012194)
收稿时间:2015-04-19;收到修改稿时间:2015-05-07

© 中国科学院软件研究所 http://www.c-s-a.org.cn

的编程语言 Giotto[9]编写控制程序, 可以方便的控制程序的逻辑执行时间, 从而使得任务时间的依赖关系不会影响到其执行顺序而仅仅和时间相关, 在执行顺序确定的情况下, 时间可预测性可以得到保证, Giotto 语言相比同步语言简洁很多, 但其描述能力有限, 它只能通过基于状态的方式描述系统的执行过程, 当系统状态很多的时候, 描述起来就会非常的繁冗. Henziger 提出的另外一个编程语言 HTL 很好的解决了描述能力平坦化的问题[10], 通过使用 Refinement 原理, 支持任务的层次化描述, 使得任务描述从线性变成了树型, 此时其可调度性分析的难度加大.

本文通过对体系结构层和编程模型层的研究, 提出在 LET 和 PRET 相结合的编程模型,并通过实例证明了该模型的可行性和相对于 LET 时间模型的优势.

2 相关工作

PRET 的思想是系统的时序特性需要像其功能特性一样清晰. 就像能够保证处理器上的算术是移植、可预测并且可以文档化的. 我们也希望它的执行速度也是如此. 当然, 将时钟频率降低是一种方法. 但是这样会明显大幅降低处理器性能. PRET 的目标是重新思考体系结构的特点并对其进行重新设计使其具有时间可预测性.

在 SPARC-PRET 中, 使用 DEAD 指令进行时间的同步, 在 PTARM 中则使用 DELAY_UNTIL 完成时间同步. 它们的实际作用是相同的, 在此并不详细介绍两种 PRET 的实现, 而是通过实例介绍基于 PRET 的编程模型.

2.1 时间指令

DEAD 指令允许程序员直接在代码中设定精确到时钟周期的定时器. 该指令设定某段代码的执行下界.

图 1 中代码两次使用了 DEAD 指令, 第 2 行中的 DEAD 指令将定时器设定为 28, 此时可能之前也使用了 DEAD 指令, 因此首先等待并不会立即执行下面的指令而是首先等待定时器变为 0, 之后再将定时器设定为 28, 并开始执行下面的指令. 在遇到第二个 DEAD 指令的时候, 在此会等待定时器变为 0, 并再设定为 26.这样就可以保证在指定时刻开始执行 DEAD 后面的指令完成同步操作.

Producer

```
1:int main(){
2:  DEAD(28);
3:  volatile unsigned in * buf =
4:      (unsigned int *)(0x3F800200);
5:  unsigned int i = 0;
6:  for (i = 0; ; i++){
7:      DEAD(26);
8:      *buf = I;
9:  }
10:  return 0;
11:}
```

图 1 DEAD 代码片段

2.2 线程交错流水线

PRET 大多采用流水线线程分离技术来防止出现数据访问冲突并且提高系统吞吐率. 所谓流水线线程分离指的是在流水线中将不同的线程依次分配在各级流水线中, 当一个线程取值之后下一个时钟周期的取值的为不相关的线程, 只有在某个线程写回周期执行之后才会继续执行下条指令, 从而不会出现数据冲突、控制冲突等会导致流水线停留的的情况发生. 另外, 同时执行多条线程增加了处理器的吞吐量, 在线程少于流水线级别的情况下, 在某个机器周期可能会出现空闲状况, 因此合理的划分线程数量也是充分利用 PRET 流水线的思路之一.

2.3 逻辑运行时间

Henziger 将系统时间模型分为三种, 零执行时间模型(ZET), 边界执行时间模型(BET)及逻辑执行时间模型(LET). 其中 BET 是目前系统设计的最主要的模型, 通过限定任务执行的截止时间, 只要尽量安排任务在截止时间之前执行完毕就能够保证模型的正确执行, 基于截止时间的 EDF 算法是 BET 的典型应用. ZET 模型指的是将程序的执行时间假设为瞬时, 从而只需要关注任务的执行序列, 而不需要考虑任务在执行过程中发生的竞态条件, 典型的应用为同步语言, 如 Esterel、Lustre 等. LET 模型和 ZET 模型的区别是 ZET 将执行时间假设为瞬时, 而 LET 将通信时间假设为瞬时. 三者在执行时间的区别可以通过图 2 准确的看出. 图中横向坐标代表时间轴, 横线箭头代表任务的执行时间, 可以看出, ZET 中没有横线箭头, 执行时间为 0, LET 横线箭头横跨整个时间段, 而 BET 则执行时间处在时间段的任意位置. 也就是说, BET 并不能确定任务的真实完成时刻, 从而增加其不确定性.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

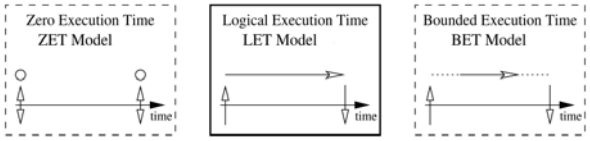

图 2 ZET、LET 和 BET 的区别

和逻辑运行时间相对应的为物理运行时间, 从图 2 中可以看出两者之间的关系和区别. 在考虑抢占、调度的情况下, 任务的物理执行时间可能是不连续的, 并且只占用其对应逻辑运行时间的部分时间. 逻辑运行时间是模型假设的运行时间, 在实际情况下, 为了保证可调度性往往还会比其物理时间大很多. 图 3 显示了 LET 模型中逻辑时间和物理时间的区别.

LET 时间模型和其它两种时间模型相比具有一定的优势, LET 比 BET 具有更为确定的输入输出时间, 相对于 ZET 则具有更为宽松的限制条件.

编程模型的时间模型实际上是粗粒度的时间同步机制, 通过任务级别的时间同步保证任务能够在指定时间完成同步, 与代码级别的同步相比, 性能损耗较高.

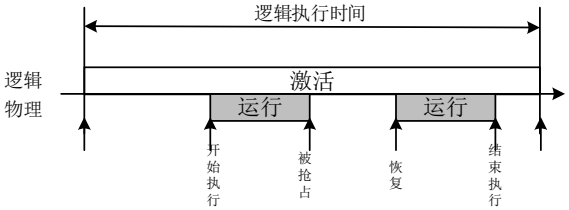

图 3 逻辑运行时间和物理运行时间的比较

典型 LET 编程模型语言为 Giotto. Giotto 拥有配套编译器和解析器, 其中编译器负责将 Giotto 代码编译为 E code, 解析器 E machine 则负责 E code 的解析执行. E machine 将任务周期划分为若干个 Unit 时间段, 该时间段也就是 E machine 的逻辑时钟, 它记录了逻辑时间的变化, 每经过 Unit 的时间, 逻辑时钟加一, 同时 E machine 需要解析 E code. 为了保证 LET 中的任务 IO 时间瞬时的, 需要保证每个 Unit 时间足够执行完所有可能的输入输出操作.

Giotto 的系统设计流程较为繁琐. 用户首先需要编写 Giotto 代码和任务代码, 然后使用 Giotto 编译器将 Giotto 代码编译转换为 E code, 再将任务代码和 E code 放在具有 E machine 的 RTOS 平台上运行, 可以看出, Giotto 的设计模式中, 时序代码和功能代码是分离的, 用户需要编写时间和功能两份代码, 并且需要对任务执行的整个过程进行建模然后才能转化为 Giotto 代码.

3 LET在PRET上的设计

本文目标平台选用基于 ARM 指令集的 PTARM 作为硬件平台, PTARM 是基于 armv4t 指令集的平台并配有 PTARM 仿真器. 本节首先提出代码级逻辑执行时间概念并给出并发模型同时提出基于 Ptolemy 的面向角色建模方法, 最后通过实例设计说明该设计方法.

3.1 执行时间优化

将 Giotto 模型应用到 PRET 上的最简单实现已经存在, 对于周期为 1000ms 的 task1, 其实现代码如图 4.

Giotto on PRET

```
Task1:
While(true){
  DEAD(1000);
  task1();
  DEAD(1000);
  output1();
}
```

图 4 PRET 平台上的 Giotto 实现

DEAD 指令在指定的定时器的时间为下界的时间内, 指定时间到达之前, 在执行完两个 DEAD 之间的代码之后不会执行任何指令. 以 task1 为例, 在执行完 task1()的代码之后, 处理器会处于空转状态, 直到定时器的值降为 0, 这实际上是处理器资源的浪费. 这实际上是由于粗粒度的时间同步导致的, 如图 5 中为对其改进.

Giotto on PRET

```
Task1:
While(true){
  DEAD(1000);
  task1();
  sleep(8000);
  DEAD(1000);
  output1();
}

Task2:
While(true){
  DEAD(1000);
  task2();
  sleep(18000);
  DEAD(1000);
  output2();
}
```

图 5 PRET 平台上的实现改进

task1 和 task2 给出执行的上界为 1000us, 并给出两个 output 驱动的执行时间上界也为 1000us. 给出的时间需要保证大于 WCET 时间同时又不会造成过多延迟, 如此在使用 DEAD 指令时就不会出现过多的处理

© 中国科学院软件研究所 http://www.c-s-a.org.cn

器浪费. 之后使用睡眠原语将处理器安全的交出, 从而将多余的时间让给其它的任务执行.

3.2 代码级逻辑运行时间

传统 LET 时间模型是任务级的, 它要求任务在执行的过程中不能发生交互, 这一点在很多应用场景并不符合, 例如生产者消费者模型中, 生产者任务和消费者任务在执行过程中一直在进行数据的交互. 为此本文提出了代码级逻辑运行时间概念, 本文称这种通过 DEAD 指令设定的时间为代码级逻辑运行时间, 它需要大于指定代码的 WCET 时间, 同时在此期间内 CPU 不能被抢占. 在消除代码执行时间抖动的同时确定了代码的输出在指定时间有效.

通过代码级 LET 模型将任务分为不可抢占的若干代码片段, 为提高 CPU 利用率, 使用 sleep 来设置可抢占区域执行时间, 从而实现了代码级的执行时间控制.

LET 时间模型限制了任务的交互时间不能发生在任务执行的过程中, 决定了其时间同步粒度为任务级的, 同时也限制了任务的设计. 与此不同的是, PRET 平台的指令集中的 DEAD 指令能够从代码级别控制某段代码的执行时间.

定义 1(代码级逻辑运行时间)代码级逻辑运行时间为一段代码的执行时间抽象, 这段时间记录了该段程序从读取数据到写入数据的时间而并不关心实际执行时间.

代码级逻辑运行时间和传统逻辑运行时间本质上的区别是同步粒度. 在传统逻辑执行时间模型中, 需要进行同步的数据由单个任务周期性更新, 因此只需要将任务的周期作为 mode[10]. 此时任务划分应当是以数据的更新周期为原则并不灵活. 而在代码级别上, 使用逻辑执行时间来抽象某段代码的逻辑执行时间, 因此在任务设计商更为灵活.

3.3 并发模型

DEAD 指令不能够被抢占, CPU 利用率不高. 为此本文为 PRET 加入并发模型. 添加 PAR 语言扩展来指定其多个任务的并发操作.

定义 2 PAR(time, thread1,thread2,thread3,..) 指定多个 thread 可以并发执行. 并且在 time 时间之后进行同步.

为保证执行的过程不发生等待, 禁止各个 thread 存在资源竞争. time 设定通过各个 thread 的 WCET 时间相加得到.

3.4 细粒度时间精确

LET 模型为了保证硬实时一般采用时间触发的方式, Giotto 中使用 Unit 来更新逻辑时钟, 并且需要使用时钟中断来更新物理时钟, 由于时钟中断还需要完成触发调度的工作, 其频率是毫秒级的, 这使得 Giotto 的实时性能也是毫秒级的[10]. 在 PRET 中, 采用 cycle 级别的定时器, 从而使其精确到 cycle 级别. 大幅提高其精度.

3.5 面向角色设计方法

面向角色设计[11]是对于带参行为的封装, 实现的是输入数据到输出数据的过程.

本文采用面向角色的设计方法将任务的执行过程根据其功能划分为各个独立的角色单元, 各个角色具有自身的代码级逻辑执行时间.

3.5.1 避障循迹小车设计流程

以避障循迹小车系统为例来说明整个设计过程. 首先给出设计流程图如图 6, 然后对各个步骤进行解释.

图 6 基于 PRET 平台设计流程

3.5.2 面向角色建模

按照面向角色的设计思路首先对任务进行划分, 避障循迹小车包括有避障和循迹两个功能. 其中避障主要基于双目超声波传感器实现, 循迹主要基于两侧灰度传感器实现. 按照输入数据输出数据的设计原则.

超声波数据包括信号数据到距离数据的转换, 距离数据到停车动作的转换. 灰度数据包括由信号数据到偏离方向的转换, 由偏离方向到小车转弯动作的转换. 因此可以得到四个角色. 分别为距离计算、停止计算、偏离计算和拐弯计算. 由于停止和拐弯都属于小车动作而且两者互斥, 将两者合二为一可以得到如下的面向角色执行流图如图 7.

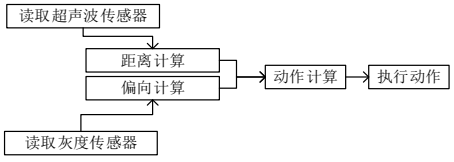

图 7 避障循迹小车面向角色建模

© 中国科学院软件研究所 http://www.c-s-a.org.cn

很多建模工具如(Simulink、LabVIEW、SPW)中都提供面向角色建模. 本文使用开源的 UC Berkeley 实现的 Ptolemy 工具, Ptolemy 本身就是基于面向角色设计的, 而且它的模型生成文件为 XML, 方便进行代码生成.

3.5.3 功能代码编写

功能代码需要按照数据读入, 数据处理, 数据写出三个部分进行设计. 由于逻辑执行时间内是不允许抢占的, 因此不需要对于上述三个部分进行顺序的限定, 只需要角色代码能够完成相应功能即可.

3.5.4 测量角色代码 WCET

硬实时系统设计中, WCET 的测量是至关重要的. 角色代码中不允许存在相互等待的情况, WCET 的测量可以使用如 aiT、Bound-T 等测量工具.

角色的 WCET 时间决定了其在相应平台上执行的物理时间, 因此 WCET 测量的是否直接决定系统能否正确运行.

3.5.5 代码生成

使用 Ptolemy 进行设计之后可以直接保存为 XML 文件. XML 的解析可以使用 Python 调用 XML 解析库完成.

Giotto on PRET

```
int main()
{
        par(20,thread1,thread2);
        dead(10);
        computing();
        dead(15);
        output();
}

void thread1()
{
        dead(10);
        get_data1();
        compute_distance();
}

void thread2()
{
        dead(10);
        get_data2();
        compute_direction();
}
```

图 8 避障小车生成代码

图 8 为图 7 中设计的生成代码, 代码中已经将之前 Giotto 代码+任务代码的设计方法缩减为只需要任务代码, 同时不需要生成 E code 调用 E machine 解析, 减少了 E machine 层带来的额外负载.

4 结语

硬实时系统设计一直对时间要求苛刻, 传统设计方式已经不能满足其对于时间的需求. 本文重新给出逻辑执行时间的细粒度定义, 将 LET 模型应用到 PRET 平台上, 通过给出并发拓展原语和代码设计框架, 给出了 PRET 平台的编程模型, 解决了目前 PRET 平台设计困难的问题. 当然不论体系结构平台还是给出的编程模型, 都还具有不足的地方, 下一步的工作应当包括提高 PTARM 可用性和完善编程模型实现上. 精确时间机作为硬实时系统的设计的新方法将成为未来研究热点之一.

参考文献

1 Grund D, Jan R, Wilhelm R. A Template for Predictability Definitions with Supporting Evidence. PPES. 2011.

2 Edwards SA, Lee EA. The case for the precision timed (PRET) machine. Proc. of the 44th annual Design Automation Conference. ACM. 2007.

3 Ip NJH, Edwards SA. A processor extension for cycle-accurate real-time software. Embedded and Ubiquitous Computing. Springer Berlin Heidelberg, 2006: 449–458.

4 Liu I, et al. A PRET microarchitecture implementation with repeatable timing and competitive performance. 2012 IEEE 30th International Conference on Computer Design (ICCD). IEEE. 2012.

5 Zimmer M, et al. FlexPRET: A processor platform for mixed-criticality systems[Techinque Report]. ucb/eecs-2013-172. California Univ Berkeley Dept of Electrical Engineering and Computer Sciences. 2013.

6 Benveniste A, Berry G. The synchronous approach to reactive and real-time systems. Proc. of the IEEE 79.9 (1991): 1270–1282.

7 Edwards SA. The specification and execution of heterogeneous synchronous reactive systems[Thesis]. University of California, Berkeley, 1997.

8 Kirsch CM, Sokolova A. The Logical Execution Time Paradigm. Advances in Real-Time Systems. Springer Berlin Heidelberg, 2012. 103–120.

9 Henzinger TA, Horowitz B, Kirsch CM. Giotto: A time-triggered language for embedded programming. Embedded Software. Springer Berlin Heidelberg, 2001.

10 Ghosal A, et al. A hierarchical coordination language for interacting real-time tasks. Proc. of the 6th ACM & IEEE International Conference on Embedded Software. ACM, 2006.

11 Lee EA, Neuendorffer S, Wirthlin MJ. Actor-oriented design of embedded hardware and software systems. Journal of Circuits, Systems, and Computers, 2003, 12(3): 231–260.

© 中国科学院软件研究所 http://www.c-s-a.org.cn