嵌入式多核平台调试技术①

余攀峰

(湖北轻工职业技术学院 机电工程系, 武汉 430079)

摘 要: 多核处理器因其处理能力和功耗的潜在优势, 逐渐应用于嵌入式系统中. 而多核体系的并行程序调试难度将直接影响到产品的释放周期, 其调试过程的复杂度将随着片上核数的增加而呈指数上升. 本文针对多核平台中常见的异常问题, 通过分析 MIPS 架构的软硬件平台工作原理, 介绍了一种嵌入式多核系统的调试方法, 实现对简单程序的跟踪分析, 方便了多核系统的调试.

关键词: 多核; 调试; 异常; MIPS

Debug Technology for Embedded Multi-Core Platform

YU Pan-Feng

(Hubei Light Industry Technology Institute, Mechanical & Electrical Department, Wuhan 430079, China)

Abstract: The multi-core processor is widely used in embedded systems because of its potential advantage in power consumption and data processing capability. The debugging difficulty of parallel programming will directly affect the release time of products. The complexity of the debugging will rise in index rate with the increase of cores. To resolve the common abnormal issues in multi-core platform, this thesis introduces a type of debugging method of the multi-core embedded system to trace simple executive programs by means of analyzing of the working mechanism of MIPS architecture, which makes the debugging of multi-core platform more convenient.

Key words: multi-core; debug; abnormal; MIPS

多核处理器是指在一个芯片中集成多个可独立运行的 CPU 内核, 随着多核处理器技术的发展, 基于多核系统的处理能力和功耗、以及面积上的优势, 开始将多核处理器应用于个人数码设备, 而不仅仅局限于军事或者工业应用. 采用并行编程方式以提高产品性能成为多核处理器的主要方式, 但并行编程方式的执行结果不仅依赖于单个线程的执行, 还往往取决于对共享资源冲突访问的竞争结果, 而多核之间存在较多的共享资源, 程序设计不当就很容易造成死锁[1]. 并且随着核数的增加, 其可能产生的错误将呈指数上升, 且定位错误的工作更加难以进行[2]. 在外场调试过程中, 受限于应用环境而无法使用 JTAG 等硬件调试方法, 只能进行远程软调试, 必然要占用一定的 RAM 实现确定性重放, 以录制必要的调试信息[3]. 多核开发商所提供的 SDK 中提供了若干的调试方式, 但这些调试系统并不合适外场调试和性能调试, 例如在一个 Linux 核上模拟多核工作的方式, 或者基于 GDB 变量调试, 均需要占用串口或者以太网口等硬件资源, 由于占用大量的内存, 在性能测试时也会造成较大的误差, 并且模拟的方式会影响多核间的时序, 而多核调试中的难点就在于多核间的时序保持. 因此本文根据产品的自身使用方式和要求, 设计了一种调试方法.

1 多核平台

1.1 硬件平台

本文采用了一款 Cavium 公司提供的 OCTEON 64 位多核处理器, 该芯片集成了如下模块: 用于填充数据包消息头的 PIP(packet input processing) 模块, 确保数据包顺序的调度保序 POW(Packet/Order/Work)模块, 以及管理数据包输出的 PKO (packet output processing)

① 收稿时间:2013-04-19;收到修改稿时间:2013-05-27

模块. 该芯片还提供了 2KB 的主要用于实现同步原子操作的 FAU 寄存器, 但在大型程序中该容量大小显然是不足的. FPA 模块(Free Pool AllocatorFree Pool Allocator)实现硬件内存分配操作, 从而提高系统的处理性能[4].

1.2 软件平台

通常多核系统都是由控制面和数据面两部分组成.

控制面上可采用实时操作系统, 例如 UCOS-II、VxWork 等, 也可采用非实时操作系统, 例如 Linux. 而现在多数厂商都是采用后者的软件架构以降低硬件成本, 也因此而造成了加载集的不一样, 适用于 Linux 的调试方法也就不再适用于 SE(Simple Executive)调试. 而数据面为多核系统的主要工作模块, 其强调高性能, 对于会存在缓存缺失、TLB 缺失以及总线竞争等问题的 Linux 不适用于数据核上, 而采用 SE-E(独立模式)程序. 所以其调试所占用的比例也较多, 花费的时间也更长, 对于这一部分进行调试研究就很有必要.

数据面通常采用的是 run-to-compliance 模式, 即数据面的每个核所执行的程序相同, 采用该方式实现的 SMP(对称多处理)模式, 具有最高的系统性能, 也因此不存在数据包的核间调度问题, 方便了数据包的跟踪, 但也同时造成了数据包对共享资源的频繁访问, 即发生资源竞争.

2 异常的产生

并行处理会访问共享资源, 虽然可以使用 hash 散列表降低资源竞争, 但无法避免冲突. 本多核系统中的共享资源是指 cvmx_shared 所定义的共享内存段, 该共享内存仅可在同一个加载集中由不同的核共享, 实际上编译器优化以及处理器本身采用的优化可能使指令实际执行顺序与程序指定的顺序不一致. 在某些需要严格保证处理器按程序顺序执行的场合, 会造成意想不到的程序缺陷, 甚至导致系统死锁, 在专用内存总线中进行访存优化, 一般会提升性能而不会产生逻辑错误, 但如果在共享总线中进行类似优化很可能造成致命错误, 例如在数据加载时由于 MIPS 架构并不能保证执行写指令之后就能准备好数据, 而数据地址转换并不在页表中时, 就会导致 TLB 未命中异常, 这些隐蔽错误在多核系统中很难被发现. 内存屏障的作用是严格保证 CPU 内存事件次序, 确保程序在执行上遵循顺序一致性模型, 从而避免因系统优化而产生的死锁. 其实现方式是引入 sync 指令, 保证该指令后的指令停止(延期)进入流水线直到该指令前的所有指令被提交(完成), 所有 Load/Store 指令对内存的访问请求在该指令发射前全部完成.

除此之外对共享内存的保护是通过自旋锁、读写锁和原子操作实现, 在排除了以上可能之后, 产生异常则可能是因不合理释放锁资源而导致, 其根本原因是资源个数比要求该资源的核数少. 其他的诸如在系统执行流执行了无法被处理器辨识的非法指令, 或者访问了没有映射的虚拟地址, 和试图访问只读页面异常等等原因就不再阐述. 如何定位异常产生的原因成为难点, 并且异常的产生可能是程序在实际使用中几个月才会出现, 难以调试, 因此记录异常产生的状态也成为重点之一.

3 调试方法

3.1 死锁定位

所有的 CPU 都会提供一个详细的异常和中断处理机制, 而在 MIPS 架构中是通过属于 CP0 寄存器的异常程序计数器(exception program counter, EPC)的寄存器实现, 该寄存器会保存造成异常的那条指令的地址, 从而为死锁调试提供了基础. 查看控制寄存器的方法是把它复制到通用寄存器里, 通过 mfc0 指令将 EPC 中的地址复制到指定的通用寄存器中, 再通过跳转语句(jr), 程序就可以返回到造成异常的那条指令处继续执行.

在本案中是通过在控制面运行的 watchdog 程序定时向其他核发送 mailbox 中断, 在 mailbox 中断服务程序中获取当前核的 EPC 寄存器, 并判断当前核工作状态, 若出现异常则将 EPC 寄存器存放的中断返回地址写入文档, 通过这个地址就可定位死锁产生的位置, 因此重点就在于如何进行现象复现. EPC 中存储的是地址, 需将地址转换为函数名以方便观察问题点, 通过加载执行文件镜像实现该功能. 将执行程序加载在内存中, 分析其包含的定位符号表, 其定义如下结构:

```
typedef struct elf32_sym {
    Elf32_Word      st_name;  /* 字符串表 */
    Elf32_Addr      st_value; /* 符号值   */
    Elf32_Word      st_size;  /* 符号大小 */
    unsigned char   st_info;  /* 符号属性 */
    unsigned char   st_other;
    Elf32_Half      st_shndx;
} Elf32_Sym;
```

通过查找 EPC 地址为 st_value 与 st_value + st_size 之间所对应的 Elf32_Sym 数据结构获取对应的函数名. 使用该方式只可获取发生死锁的函数, 而不能具体到函数语句, 若存在 inline 函数则其定位会变得困难, 使用较小的函数体可避免该问题, 同时在调试时可编译不同的版本实现对 inline 定义的屏蔽. 但若要精确到具体的函数语句, 则必须使用反汇编工具查看具体的汇编代码.

3.2 调用链分析

根据并行式编程的特点, 死锁的产生必然是两个及其以上的数据核运行的程序导致的, 因此需在确定死锁地址后了解各数据核当前的函数调用链和当前数据包的内容, 以方便确定具体问题. 而多核系统往往应用在大流量情况下, 打印函数调用链将会严重影响时序, 从而造成无法复现死锁产生的原因, 因此调用链只可保存在系统内部, 只有当发生死锁等其他异常时情况时才进行串口输出. 并且只要数据包未通过 PKO 命令发送则一直保存在内存中, 可根据需要输出打印由 FPA 管理的指定数据包中的内容. 数据核的增多和处理数据的复杂程度, 会产生成百上千条的调试信息记录, 根据文献[1][5]的思想采用可视化方式展现给用户是必然趋势, 而本案中也是将这些信息输入分析软件中, 自动寻找出最耗资源的函数.

调用链的数据信息以提供必要的信息为主, 并同时支持 run-to-compliance 和 pipeline 两种设计模式, 对于前一种模式因不存在 WQE 的核间调度, 只需在每个核上设置全局变量保存形如 struct dbg_fpa_s 的数据结构即可.

```
struct dbg_fpa_s
{
    UINT64 action:2;
    UINT64 pool:3;
    UINT64 num:24
    UINT64 tag_type:3;
    UINT64 tag_value:32;
    UINT64 stamp;
    void * addr;
    void * func;
};
```

struct dbg_fpa_s 中的 action 用于记录该数据包的执行命令, pool 记录缓冲池序号, num 记录缓冲池的序号, tag_type 和 tag_value 用于记录数据包的相关信息, 而 stamp 记录当前数据包的执行函数的机器时间戳, 不仅有利于分析各个数据核的函数调用次序, 还可用于分析系统性能, 通过查找最大时间差来确定最耗时的函数, addr 记录函数的地址信息, func 记录当前执行函数名.

而对于 pipeline 方式运行的程序因存在核间调度, 还需重点分析数据包在核间的流动. 为保证不发生乱序问题而导致网络应用质量的降低, 通常会采用基于流的硬件保序功能, 该方式的实现是依赖于 Tag_type 和 Tag_value 实现. 对于普通 IP 包而言, 可采用 tuple 算法; 对非 IP 包采用 mask 算法, 其区别在于前者是利用五元组信息进行 hash 算法, 而后者可针对数据包的前 128 个字节实现 hash 计算, 但无论是哪一种算法, 因同一个流或者数据类型其 hash 值是一样的, 而无法确定唯一的数据包, 因此采用缓冲池加缓冲区序号来确定唯一的数据包.

其部分信息显示如下:

FPA-Track On No.0 Core

Index	Stamp	Address	Pool	Action	Func
5	00000002bf81685d	00000000349df080	0[299967]	Free	free_wqe
4	00000002bf0f1a8b	00000000349d0080	0[299967]	----	handle_packet
3	00000002be21a4b	00000000449e2085	0[299967]	----	--------
2	00000002ba4545df	00000000447f1880	0[299965]	----	--------
1	00000002fe650914	000000003fce883a	0[299965]	Alloc	get_wqe

图 1 调用链输出

对每一个核分别建立一个环形缓冲区以记录数据包调用链的前后函数, 只要缓冲区足够大即可获取足够的信息. Action 栏显示当前数据包是获取(Alloc)还是释放(Free), 若是数据信息处理则不显示; Func 栏显示当前处理的函数, 若显示为空则代表当前数据包进行核间调度. 该调用链可在超级终端中查看, 或者直接输入到可视界面进行分析, 以方便分析问题点. 因多核系统的异常往往是由多个核相互影响的, 在产生调用链时也需要多种方式进行控制, 例如在某个核上发生异常时, 其他核立即自旋, 只将已发生问题核的调用链输出; 或者其他核在处理完当前 work 任务后自旋, 再输出调用链, 需根据不同的情况选择不同的调用链输出方式.

在确定死锁产生的位置和相关调用链之后, 如何对其进行分析成为关键点. 在调试初期可将函数调用链与设计的流程图相比较, 并辅以读取 dmfc0 的值了

(下转第 203 页)

INSTANCE_NAME

Rac2

可以看到, 用户连接自动切换到了节点 2, 这样解决了在某个节点故障后, 保持服务连续的需求.

除此之外, 还可以利用Oracle RAC环境下的Load Balance 特性, 解决负载问题. oracle10g 提供了两种手段来实现分散负载:其一是通过 Connection Balancing, 在客户端按照某种算法把用户分配到不同的节点;其二是通过 service, 在应用层进行分散.

6 小结

数据库的高可用性、负载能力与许多方面是相关的, 如存储、网络和主机等等, Oracle RAC 是在本地环境进行的数据库冗余措施, 对于地震、火灾等不可抗力导致的机房的损毁, 将无法得到可用性保障. 因此需要其他的容灾方案进行保护, 如 Oracle DataGuard、存储级别的 Mirror 镜像等等, 可以实现快速切换与灾难恢复. 这是每个企业数据中心所必须考虑和面对的.

参考文献

1 文平.Oracle 大型数据库在 AIX/UNIX 上的实战详解.北京:电子工业出版社.2012:446–449.

2 陈吉平.Oracle 高可用环境.北京:电子工业出版社.2009:35–36.

3 张晓明.大话 Oracle RAC 集群高可用性备份与恢复.北京:人民邮电出版社.2009:71–73.

4 何明.Oracle DBA 培训教程—从实践中学习 Oracle 数据库管理与维护.北京:清华大学出版社.2009:98–99.

5 红帽软件(北京)有限公司.RedHat Enterprise Linux 系统管理.北京:电子工业出版社.2009:112–117.

(上接第 189 页)

解异常产生的原因, 以确定是否存在逻辑错误, 而对于隐蔽性错误则需根据现象进行分析. 将异常时的函数执行时间与该函数正常执行时间相比较, 当出现执行时间明显增加的情况时, 则死锁与该函数有关, 并进一步将与该函数有关的函数、锁信息进行上报到分析软件中进行处理, 通过分析软件进行可视化调试有助于快速定位问题. 对于资源竞争造成的死锁, 还可以通过自旋其他核实现核数量上的变化辅助定位.

4 结束语

本文结合 OCTEON 的多核硬件平台, 分析该多核处理器的软硬件工作方式, 阐述异常产生的原因, 提出了一种调试方法, 在不使用 JTAG 调试器以降低系统硬件复杂度的基础上, 具有占用内存资源小等优点, 缩短系统开发时间的同时, 方便了在工程现场的调试, 具有很高的实用价值.

参考文献

1 曾令将,王继红,舒红霞.并行嵌入式系统可视化性能分析工具的设计与实现.计算机与数字工程,2012,40(3):130–132.

2 刘磊,黄河,唐志敏.支持多核并行程序确定性重放的高效访存冲突记录方法.计算机研究与发展,2012,49(1):64–75.

3 杨旭,刘江,钱诚,苏孟豪,吴瑞阳,陈云霁,胡伟武.一种面向多核处理器的通用可调试性架构.计算机辅助设计与图形学学报,2011,23(10):1656–1644.

4 杨启军,鲁士文.基于多核的入侵防御系统的设计与实现.计算机工程与设计,2010,31(21):4595–4598.

5 林贻珀.可视化并行性能调试环境的设计与实现[硕士学位论文].北京:清华大学,2009.