ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.7, July 2007, pp.1806−1817
DOI: 10.1360/jos181806

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

龙芯 2 号同时多线程处理器的软硬件接口设计*

李祖松[1,2+], 许先超[1,2], 胡伟武[1], 唐志敏[1]

[1](中国科学院 计算技术研究所,北京 100080)

[2](中国科学院 研究生院,北京 100049)

Hardware/Software Interface Design of Godson-2 Simultaneous Multithreading Processor

LI Zu-Song[1,2+], XU Xian-Chao[1,2], HU Wei-Wu[1], TANG Zhi-Min[1]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: Phn: +86-10-62565533 ext 9318, E-mail: {lisoon,xuxianchao,hww,tang}@ict.ac.cn

Li ZS, Xu XC, Hu WW, Tang ZM. Hardware/Software interface design of godson-2 simultaneous multithreading processor. *Journal of Software*, 2007,18(7):1806−1817. http://www.jos.org.cn/1000-9825/18/1806.htm

Abstract: With the development of VLSI (very large scale integrated circuit) technology, a single chip can contain over one billion transistor. Multithreading technique is the developing trend of high performance processor in the future. How to design hardware/software interface has become a major problem for multithreading processor. According to simultaneous multithreading processor requirement and the original Godson-2 architecture, the hardware/software interface for Godson-2 simultaneous multithreading processor is advanced and defined. Based on this interface, the corresponding Linux for Godson-2 SMT (simultaneous multithreading) processor is designed and implemented on Linux 2.4.20. The SPEC CPU2000 and some other benchmark programs are used to compare the performance of Godson-2 SMT processor with the superscalar processor. The results show that the Godson-2 SMT processor designed with the interface can get a much better performance. The hardware/software interface and operating system design in this paper are also very useful for the multi-core processor design.

Key words: Godson-2; simultaneous multithreading; microarchitecture; Linux operating system

摘 要: 随着生产工艺的提高,芯片上能集成越来越多的晶体管,多线程技术也逐步成为一种主流的处理器体系结构技术,而多线程处理器的软硬件接口也就成为急需解决的问题.在分析同时多线程的软件需求的基础上,提出龙芯 2 号同时多线程处理器的软硬件接口协同设计解决方案,给出相应的操作系统实现方案.同时,在 Linux 2.4.20 的

* Supported by the National Natural Science Foundation of China for Distinguished Young Scholars under Grant No.60325205 (国家杰出青年基金); the National High-Tech Research and Development Plan of China under Grant Nos.2002AA110010, 2005AA110010, 2005AA119020 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2005CB321601 (国家重点基础研究发展计划(973)); the Knowledge Innovation Program of the Chinese Academy of Sciences under Grant No.KGCX2-109 (中国科学院知识创新工程); the Basic Research Foundation of the Institute of Computing Technology, the Chinese Academy of Sciences under Grant No.20056020 (中国科学院计算技术研究所基础研究基金)

Received 2006-03-22; Accepted 2006-06-09

基础上实现了龙芯 2 号同时多线程处理器相应的操作系统.通过运行 SPEC CPU2000 等测试程序进行性能评测,充分说明实现软硬件接口的龙芯 2 号同时多线程处理器极大地提高了多进程负载的性能.分析和设计方案不仅适用于同时多线程处理器,而且对于片内多核处理器的设计也有借鉴作用.

关键词: 龙芯 2 号;同时多线程;微体系结构;Linux 操作系统
中图法分类号: TP302 文献标识码: A

龙芯处理器的设计目标是基于办公用途的 Linux PC 和网络服务器以及许多嵌入式环境,包括网络终端、网络交换机、工业控制和游戏机等等.龙芯 2 号[1]是实现 64 位指令集的 RISC(reduced instruction set computing)处理器,采用四发射超标量超流水结构.该芯片采用寄存器重命名、动态调度、动态分支预测、非阻塞 CACHE 访问、取数推测(load speculation)、分离读取(split read)等先进体系结构技术.龙芯 2 号高性能通用 CPU 芯片的总体性能已达到 2000 年左右的国际先进水平,居国内通用 CPU 研制领先水平.

龙芯 2 号已实现的超标量处理器的关键技术只能解决指令间的假相关,单线程程序中指令间的相关性依然存在,使得进一步的性能提高遇到了瓶颈.目前,国际上的先进处理器体系结构已经从挖掘指令级并行发展到挖掘线程级并行,多线程技术也逐步成为一种主流的处理器体系结构技术.多线程处理器多种多样,主要可分为两大类——隐式多线程(implicit multithreading)和显式多线程(explicit multithreading)[2].隐式多线程技术是通过将单个进程分解成多个线程,对线程进行猜测执行,以此实现单进程的加速.由于隐式多线程处理器的结构很复杂,实现困难,目前主要还处于研究阶段.显式多线程技术是通过运行多个不相关的线程,以此提高处理器的处理能力.由于显式多线程处理器的设计较为简单,目前许多商用处理器都采用此技术.

在显式多线程技术中,同时多线程(simultaneous multithreading,简称 SMT)[3−5]技术巧妙地将线程级并行转化为指令级并行,以高效的资源利用率来提高处理器的性能.龙芯 2 号处理器也将引入同时多线程技术.然而,任何系统都由硬件和软件两部分组成:硬件设计需要考虑软件的需求,为软件提供接口(例如增加一些指令和控制寄存器);而软件也要尽量利用好硬件提供的功能.现在要在龙芯处理器上实现多线程,同样也要考虑操作系统怎样才能够很好地利用硬件提供的功能,而硬件也需要提供好的接口让软件能够正确和高效地利用硬件增加的功能.本文的目的就是介绍硬件需要给软件提供什么样的接口,以及操作系统如何利用这些接口进行处理器的管理.接口设计的好坏直接影响到最后性能的高低,我们的目标就是确定一个功能完备而且实现简单的接口:功能完备指的是操作系统需要的一些指令硬件都必须提供;实现简单就是指硬件上不能太复杂.

本文主要介绍在龙芯 2 号同时多线程处理器上软硬件接口及操作系统的设计方案.第 1 节介绍龙芯 2 号同时多线程处理器的微体系结构.第 2 节分析操作系统和应用程序的需求.第 3 节讨论相关问题的解决方案及接口设计.第 4 节介绍龙芯 2 号同时多线程操作系统的设计方案.第 5 节介绍龙芯 2 号同时多线程的性能评测.最后总结并介绍未来进一步的研究工作.

1 龙芯 2 号同时多线程处理器

龙芯 2 号同时多线程处理器支持两个线程,有如下两种运行模式:

- 超标量模式
- 同时多线程模式

超标量模式与原来龙芯 2 号处理器相同,处理器只运行一个线程,此线程占用所有的硬件资源,包括队列、流水线通路、物理寄存器堆、功能部件、CACHE 等.

同时多线程模式是在处理器上同时运行两个线程,这两个线程来自不同的程序.每个线程拥有独立的程序计数器(program counter,简称 PC)、逻辑寄存器、控制寄存器等,其他资源包括队列、流水线通路、功能部件、CACHE 等由两个线程共享.

龙芯 2 号同时多线程处理器的微体系结构如图 1 所示.由于要支持两个线程的执行,需要同时保存两个线程的上下文,所以,同时多线程处理器比原来的超标量处理器增加了一些硬件资源.增加的硬件资源主要包括

PC、定点浮点寄存器重命名、定点浮点寄存器堆、控制寄存器、转移历史寄存器、地址返回栈和转移指令队列等,每个线程独自占有一套.其他资源则根据资源的特点不同选择不同的共享策略进行共享.例如与线程上下文密切相关的资源,包括指令 TLB(translation lookaside buffer)、指令顺序提交队列和 CP0 队列等,则每个线程占用一半的项数,互不干扰.对于指令 CACHE、数据 CACHE、转移模式表、BTB(branch target buffer)、DTLB(data translation lookaside buffer)和功能部件等,两个线程完全共享,不作限制.而对于定点发射队列和浮点发射队列,由于指令进入队列时可以放在任意的空项里,指令从队列中发射后,所占用的项就立即释放,指令也就在发射前的这段时间处在发射队列中,因此,发射队列的共享方式也是允许每个线程占用其中任意一项.但为了避免阻塞的线程过多地占用发射队列项,使得另一线程无法发射执行,所以为每个线程至少保留 4 项.

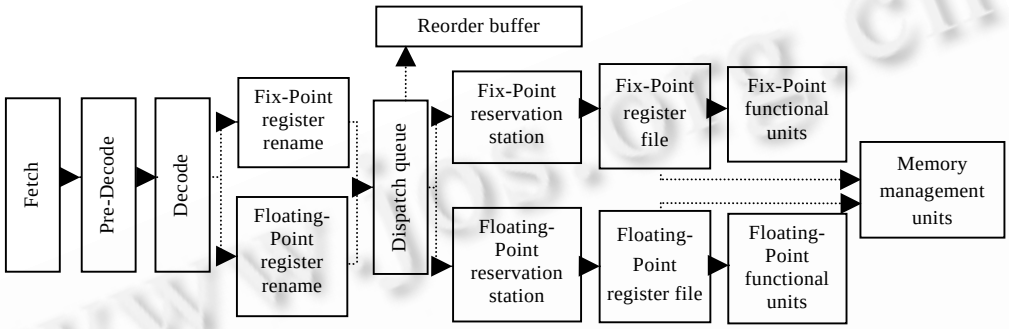

Fig.1 Microarchitecture of Godson-2 simultaneous multithreading processor

图 1 龙芯 2 号同时多线程处理器的微体系结构

2 操作系统和应用程序的需求分析

当龙芯 2 号同时多线程处理器处于同时多线程模式时,可以同时运行两个独立的线程,每个线程有自己的逻辑寄存器和控制寄存器,相互之间的运行互不影响.此时,龙芯 2 号同时多线程处理器可以被当作两个逻辑 CPU 来对待.因此,龙芯 2 号同时多线程处理器同时多线程模式的应用情况是这样的:在龙芯 2 号同时多线程处理器上运行一个类似于对称多处理器(symmetrical multiprocessor,简称 SMP)内核的同时多线程(SMT)操作系统,它像对待含有两个 CPU 的 SMP 处理器那样来利用同时多线程模式的龙芯 2 号同时多线程处理器,调度两个进程同时运行在龙芯 2 号同时多线程处理器上,并负责两个逻辑 CPU 间的通信和同步等.实际上,现阶段 Linux 操作系统就是这样来利用 Intel Hyper-thread[6]处理器的.这个 SMT 操作系统和 SMP 操作系统基本上类似,只是在 TLB 和 CACHE 的处理上有所不同:SMP 处理器每个 CPU 有自己的 TLB 和 CACHE;而 SMT 处理器上两个逻辑 CPU 是共用一个 TLB 和 CACHE 的.因此,SMT 操作系统只要在 SMP 操作系统上修改处理 TLB 和 CACHE 的相关代码就可以了.

在 SMT 操作系统中,内核将会调度合适进程来运行,为每个进程保存和恢复正确的上下文,每个进程都有一个独立的执行环境.应用程序在 SMT 操作系统上运行时与原来相同,不需要做什么改动,所以也就不需要硬件再提供额外的接口.而 SMT 操作系统在实现时,需要硬件提供相关接口来协同解决与 SMP 操作系统类似的问题:

1) 启动即模式切换机制.龙芯 2 号同时多线程处理器在刚启动的时候是运行在超标量模式下的,此时只可以运行一个进程.因此,在 SMT 操作系统启动过程中,内核需要通过硬件提供的接口进行模式切换,使龙芯 2 号同时多线程处理器由超标量模式转变为同时多线程模式,并且调度第 2 个逻辑 CPU 从正确的指令开始运行.

2) 中断机制.在 SMT 操作系统中,内核将会让两个进程同时运行在龙芯 2 号同时多线程处理器上.现在的操作系统基本上都是多任务的操作系统,SMT 操作系统也将是一个多任务操作系统.在多任务操作系统中,每个进程执行一段时间后将交出处理器的使用权,内核将会调度另一个进程去执行,而用户进程需要中断来触发,使其进入内核进行调度或信号处理等.因此,每个逻辑 CPU 都需要有中断,至少必须有一个时钟中断,否则,这个逻

辑 CPU 将会一直运行一个进程,直到这个进程运行结束.另外,由于龙芯 2 号同时多线程处理器是在同一个物理 CPU 上实现两个逻辑 CPU,而此时外部中断只有一个,当外部中断到来时,是两个逻辑 CPU 轮流处理还是一个 CPU 单独处理也需要考虑.

3) CPU 间消息传递机制.在 SMT 操作系统中,内核将会统一管理两个逻辑 CPU,为每个逻辑 CPU 调度进程运行.两个逻辑 CPU 将使用同一个内核的数据,共享内核资源.由于两个逻辑 CPU 的执行程序都可能进入内核改变内核数据,而当一个逻辑 CPU 改变了共用的内核数据结构时,就需要及时通知另一个逻辑 CPU.因此,SMT 操作系统中需要硬件提供一个 CPU 间的消息传递机制来实现两个逻辑 CPU 间的通信.

4) 同步共享机制.在前面提到,SMT 操作系统中,两个逻辑 CPU 将使用同一内核中的数据,共用一些内核资源,因此会出现两个逻辑 CPU 对某些内核资源的竞争使用,需要硬件提供支持来实现一个同步共享机制,对会出现竞争的资源使用同步.龙芯体系结构的 CPU 一般是通过 ll/sc 指令来解决同步共享问题的,虽然现在的龙芯 2 号超标量处理器中也提供了 ll/sc 指令,但是它们的实现没有达到要求,所以需要重新设计同步共享接口.

3 相关问题的解决方案及接口设计

为了实现 SMT 操作系统的控制和管理,首先必须解决上一节提出的 4 个问题.龙芯 2 号同时多线程处理器针对 SMT 操作系统的需求,提供新的接口以满足操作系统解决以上几个问题的需要,下面将针对这些问题给出具体的解决方案和接口设计.

3.1 模式切换接口设计方案

在 SMT 操作系统中,内核在启动过程中需要硬件提供模式切换接口来将龙芯 2 号同时多线程处理器从超标量模式切换到同时多线程模式,并调度另一个逻辑 CPU 从一个指定的地址开始执行.

龙芯 2 号同时多线程处理器增加了 modech imm,crt *a*1 和 kill 三条指令来实现模式切换:modech imm 指令将会使处理器切换到 imm 所表示的模式,0,3 分别表示超标量模式和同时多线程模式;crt *a*1 指令用于启动另一个逻辑 CPU,让它从 *a*1 寄存器的值所指向的地址开始取指执行;而 kill 指令用于停止本逻辑 CPU 的运行.

下面介绍具体的模式切换机制.龙芯 2 号同时多线程处理器的每个逻辑 CPU 增加了一个状态位,用来表示这个逻辑 CPU 的状态——激活或僵死,当一个逻辑 CPU 处于僵死状态时,它将不进行取指、执行.当龙芯 2 号同时多线程处理器需要从运行单个进程切换模式去运行多个线程时,实际上需要进行两步操作:首先进行模式切换分出两个逻辑 CPU,此时,硬件将划分资源分配给两个线程使用;然后再激活另一个逻辑 CPU 开始执行.例如,在 SMT 操作系统启动过程中,内核可以通过图 2 中的代码来实现从超标量模式到同时多线程模式的切换.

```
…
la      a1, second_start_pc
modech 3
crt     a1
…
```

Fig.2 Mode switch example codes

图 2 模式切换实例代码

其中,modech 3 指令只是将硬件资源分开,分出两个逻辑 CPU,为运行两个线程准备硬件基础.此时,它并没有让另一个逻辑 CPU 处于激活状态,所以,另一个逻辑 CPU 并没有开始执行,它的激活工作由 crt 指令完成;crt 指令将会激活另一个逻辑 CPU,并让它从指定的地址开始执行.如果需要从同时多线程模式切换回超标量模式,只要在第 2 个逻辑 CPU 上执行 kill 指令,让这个逻辑 CPU 进入僵死状态,停止取指和执行,然后再让第 1 个逻辑 CPU 执行 modech 0,完成模式切换.龙芯 2 号同时多线程处理器通过上述模式切换机制满足不同应用模式切换的需要.

3.2 中断解决方案

首先,我们来分析外部中断如何进行处理.外部中断的处理有下面两种解决方案:

1) 采用轮流的方式,也就是外部来的中断由两个逻辑 CPU 轮流来处理.

2) 由第 1 个逻辑 CPU 专门处理,另一个逻辑 CPU 不接受外部中断.

第 1 种方案需要确定什么时候进行轮流.最简单的办法是:一个逻辑 CPU 在清除完一个中断以后通知硬件,硬件再将所有的外部中断送到另一个逻辑 CPU 上.这种方案需要增加一些硬件资源,至少需要增加一个控制寄存器或专门的指令,用于触发外部中断处理的轮换.另外,操作系统也要做相应的改动,在中断处理过程中清除中断后要通过指令或控制寄存器通知硬件进行轮换.当然,还有其他的轮流办法,例如,Intel 采用动态优先级来轮流处理外部中断,但这些方法需要的硬件支持更多.

第 2 种方案硬件实现比较简单,只要将外部中断全部连到第 1 个逻辑 CPU 上就可以了.这种方案的缺点是:第 1 个逻辑 CPU 可能会频繁进入中断,上面运行的程序可能会不停地被迫进入内核态,程序的性能可能会有些降低.然而,外部中断的个数并没有增加,所以整个系统的性能并不会降低.

综合考虑这两种方案,龙芯 2 号同时多线程处理器采用了硬件实现比较简单的第 2 种方案来处理外部中断,也就是外部中断由第 1 个逻辑 CPU 专门处理.

前面提到,每个逻辑 CPU 都需要有中断来促使其进入内核态,以便运行在它上面的进程能够被调度.另外,现阶段的操作系统则有更进一步的要求,那就是处理器上至少有一个时钟中断;而在第 2 种方案下,第 2 个逻辑 CPU 没有外部中断,因此也就没有外部的硬件时钟中断.由于在龙芯 2 号同时多线程处理器中每个逻辑 CPU 都有一套完整的控制寄存器,所以可以由操作系统设置 count 和 compare 控制寄存器的初值.而处理器每经过两个时钟周期会自动将 count 寄存器的值加 1,当 count 和 compare 控制寄存器的值相等时会产生一个时钟中断,这样就解决了第 2 个逻辑 CPU 的时钟中断问题.

3.3 CPU间消息传递解决方案及接口设计

在 SMP 处理器中,CPU 间消息传递一般是通过 CPU 间中断来实现的.龙芯 2 号同时多线程处理器也将通过 CPU 间中断来实现 CPU 间的消息传递.下面我们首先分析 CPU 间需要传递的消息类型,然后介绍龙芯 2 号同时多线程处理器通过 CPU 间中断来实现的 CPU 间消息传递机制的相关接口设计.

1) CPU 间消息传递的类型

在 SMP 操作系统中,内核需要在 CPU 间传递的消息类型通常有如下 4 种:RESCHEDULE,SMP_FUNCTION,STOP,INVALIDATE_TLB.它们都有各自的应用情况,具体说明如下:

RESCHEDULE 主要是在一个逻辑 CPU 要让另一个逻辑 CPU 进行重新调度时使用.典型的应用情形是一个逻辑 CPU 收到了一个外部中断,而等待这个中断的进程此时正运行在另一个逻辑 CPU 上,它并不知道有中断需要处理,因此就需要给这个逻辑 CPU 发送 RESCHEDULE 消息,让这个进程进入内核,及时对收到的中断进行处理;

SMP_FUNCTION 主要是在一个逻辑 CPU 让另一个逻辑 CPU 执行一个函数时使用.这个消息的使用情况较多,当一个逻辑 CPU 需要另一个逻辑 CPU 去进行某个处理时,就会发出这个消息,让另一个逻辑 CPU 去执行相关的函数;

STOP 是在一个逻辑 CPU 让另一个逻辑 CPU 结束当前任务时发出的.当内核收到停机或重启的命令时,它需要将所有当前运行的任务结束,并将相关数据保存回硬盘.由于此时另一个逻辑 CPU 上可能还在运行某个进程,所以需要通过这个消息来通知它停止执行所有任务,以便正确保存数据;

INVALIDATE_TLB 是一个在龙芯体系结构的处理器上经常被使用的消息.它被用来通知另一个逻辑 CPU 清空某个相关 TLB 表项.这个消息主要用于下述情况:内核某些时候会改变某个进程页表内容,由于其他逻辑 CPU 上的 TLB 表项可能包含有这个进程的旧的页表内容,此时就需要向这些逻辑 CPU 发送这个消息,让它们将相关 TLB 项清除.

2) 龙芯 2 号同时多线程处理器的 CPU 间消息传递机制和接口设计

龙芯 2 号同时多线程处理器将通过 CPU 间中断来向另一个逻辑 CPU 发送消息,每个 CPU 间中断类型对应一种消息类型.当需要向另一个逻辑 CPU 发送消息时,只要向它发送相应的 CPU 间中断就可以了.龙芯 2 号同时多线程处理器的 CPU 间中断的实现方法如下:如图 3 所示,我们为每个逻辑 CPU 增加一个控制寄存器来记录另一个逻辑 CPU 发送过来的 CPU 间中断和本逻辑 CPU 的 CPU 间中断屏蔽信息,并增加指令来实现对这个控制寄存器的读取和设置;当一个逻辑 CPU 需要向另一个发送中断时,就将另一个逻辑 CPU 的 CPU 间中断控制寄存器的相关中断位设置为 1,如果这个逻辑 CPU 没有屏蔽这个中断类型的 CPU 间中断,那么它将会产生一个中断;另外,每个逻辑 CPU 都可以读取自己的 CPU 间中断控制寄存器来查看另一个逻辑 CPU 发送过来的中断类型,并可以设置自己的 CPU 间中断控制寄存器来屏蔽和清除另一个逻辑 CPU 发送过来的中断.

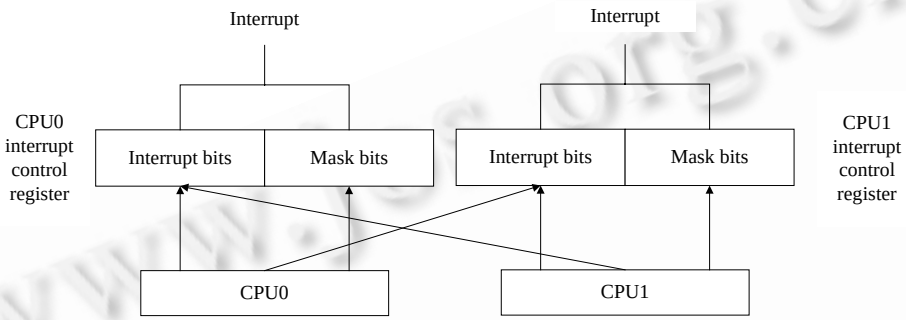

Fig.3 Message passing mechanism in Godson-2 simultaneous multithreading processor

图 3 龙芯 2 号同时多线程处理器的 CPU 间消息传递机制示意图

在上面的 CPU 间中断机制中,每个逻辑 CPU 需要读取和设置另一个逻辑 CPU 的控制寄存器.因此,龙芯 2 号同时多线程处理器增加 *mtc*0_*t* *a*1,*a*2 和 *mfc*0_*t* *a*1,*a*2 两条指令来实现对另一个逻辑 CPU 控制寄存器的读取和设置.这两条指令的格式与 *mtc*0 *a*1,*a*2,*mfc*0 *a*1,*a*2 相同,只是 *mtc*0_*t* *a*1,*a*2 和 *mfc*0_t *a*1,*a*2 操作的是另外一个逻辑 CPU 的 CP0 控制寄存器.如果需要向另外一个逻辑 CPU 发送中断,就先通过 *mfc*0_*t* 指令读出另一个逻辑 CPU 的 CPU 间中断控制寄存器,然后修改相应的位,再将修改后的值通过 *mtc*0_*t* 指令写入另一个逻辑 CPU 的 CPU 间中断控制寄存器.

需要说明的一点是,在对 CPU 间中断控制寄存器进行设置时,需要考虑同步问题:一个逻辑 CPU 向另外一个逻辑 CPU 发送中断时,如果另一个逻辑 CPU 正在修改自己的 CPU 间中断控制寄存器,就可能造成一些信息的不一致.因此,操作系统要在所有需要修改 CPU 间中断控制寄存器的地方加上同步锁,防止两个逻辑 CPU 同时操作,以保证 CPU 间中断控制寄存器的正确使用.

3.4 同步共享解决方案

关于同步共享问题,我们先分析在 Linux 操作系统中同步共享机制的使用情况,然后再介绍龙芯 2 号同时多线程处理器的同步共享机制.

Linux 内核中同步主要用于两种情况:防止内核资源的竞争使用和防止关键代码的重入.

第 1 种情况是防止内核资源的竞争使用.内核中的一些资源可能被多个进程竞争使用,Linux 用 up/down 等函数来同步内核竞争资源的使用.当一个进程要使用某个竞争资源时,它会先调用 down 函数来查看这个资源的同步锁是否已经被锁住:如果是,那么说明另一个进程正在使用这个资源,它将被内核放进这个锁的等待队列中,同时被调度其他进程执行;否则,它将使用这个资源,同时设置这个资源的同步锁,并在停止使用时释放掉相关的同步锁.这样就可以防止多个进程同时使用内核竞争资源.

第 2 种情况是防止关键代码的重入.在 Linux 内核中,一些代码不允许同时进入,Linux 用了 spin_lock 函数来保证对关键代码的单一使用.由于在 SMP 或 SMT 内核中可能产生多个 CPU 上的进程进入同一段代码的情

况,因此在 Linux 中,进入不可重入的关键代码前都会调用 spin_lock 函数来检查一下当前是否可以进入这段代码.如果已经有其他进程进入,那么它将一直等待,直到另一个 CPU 上的进程离开这段代码为止.

基于上述分析,同步在操作系统中是必不可少的,特别是在 SMT 内核中.而软件上同步一般都是通过读取、修改和设置信号量来实现的,例如上面的 down 和 spin_lock 函数都是通过查看相关信号量是否为 0 来判断相关资源或代码是否已经被锁住,同时设置这个信号量来锁住相关资源.然而,当多个进程同时操作一个信号量时,由于需要进行读取、修改和设置信号量 3 个操作,本身就存在着同步问题,因此就需要硬件提供相关的接口来支持同步.

龙芯体系结构的处理器一般是使用 ll/sc 指令对来实现同步共享的:当一个进程要操作一个信号量时,它先通过 ll 指令读取这个信号量的当前值,ll 指令将会设置 llbit 位,同时记录这个信号量的物理地址,然后修改读取回来的信号量,并将修改后的值通过 sc 指令写回.而 sc 是否成功写回由 llbit 的值决定,只有 llbit 为 0,sc 才会将值写回.如果在 ll 和 sc 之间发生了异常或其他处理器对这个信号量进行了设置,那么,llbit 将被清 0.目前的龙芯 2 号超标量处理器也对这两条指令做了实现,然而,它并没有考虑到多个处理器的情况,只是在出现异常的时候才修改 llbit 位,所以不能满足两个逻辑 CPU 间同步共享的要求.

另外,针对 SMT 处理器,国外的研究者提出 acquire/release/try_acquire 的共享同步机制[7],这种机制有较好的同步共享效率,能够达到细粒度同步共享.然而,这种机制硬件实现上比较麻烦,acquire 指令需要实现读取数据、判断并进行写数据或等待.这种指令实现很复杂,不符合 RICS 处理器使用简单指令的思想.所以,龙芯 2 号同时多线程处理器不实现这种同步共享机制,而是完善 ll/sc 指令,即另一个线程发生对这个信号量进行设置,则将 llbit 位清 0,以此来满足同步共享的要求.另外,为了避免等待的线程不停地尝试,浪费处理器资源,于是增加一条 wait *a*1 指令来暂时停止逻辑处理器的执行,以达到 acquire/release/try_acquire 相同的效果.Spin_lock 函数可以通过图 4 中的代码来实现,当发现有其他进程已经设置了相关信号量时,就通过 wait 指令停止一段时间,而不是不停地尝试,这样性能将会有所提高.

```
1:    ll a1,offset (base)
      blez a1,2f
      sub a1,1
      sc a1,offset (base)
      beqz a1,1b
      nop
      bneqz a1,3f
      nop
      lui a1,0xffff
2:    wait a1
      bal 1b
3:
```

Fig.4 Spin_lock example codes

图 4 Spin_lock 代码示意图

4 龙芯 2 同时多线程操作系统设计

龙芯 2 同时多线程操作系统的设计目的,就是利用龙芯 2 号同时多线程处理器可以同时运行两个独立线程的功能来提高整个软件系统的性能.从操作系统来看,龙芯 2 号同时多线程处理器就像是含有两个完全独立的 CPU 的 SMP 系统,从而内核可以像对待 SMP 一样来对它进行支持和利用.因此,我们要设计的龙芯 2 同时多线程操作系统实际上就是一个龙芯 2 号的 SMP 操作系统,只是这个操作系统在硬件相关方面的代码将采用龙芯 2 号同时多线程处理器提供的接口.

由于 Linux 操作系统对 SMP 处理器有很好的支持,在处理器间同步、通信、内核资源同步和共享上已经比较成熟,所以,我们将对 Linux 进行相关修改,编写一些硬件相关的接口,并利用它的 SMP 相关处理来实现对龙芯 2 号同时多线程处理器的支持.下面将首先分析 Linux 操作系统在对 SMP 处理器进行处理时与对普通单 CPU 处理器进行处理时有何不同,然后介绍龙芯 2 同时多线程 Linux 操作系统的实现.

Linux 在支持 SMP 处理器时对以下几个方面作了特别处理:(1) 启动过程.Linux 先从主 CPU 上开始执行,初始化完主 CPU 后,还将依次启动其他次 CPU,让每个 CPU 都处于一个正确的状态,并准备好执行分配给它的进程;(2) 进程调度.Linux 将会为每个 CPU 调度一个需要运行的进程,充分利用每个 CPU;(3) 资源同步共享.Linux 中对不可重入代码采取了同步措施,并为每个可能出现竞争使用的资源增加了同步锁;(4) CPU 间消息传递.当一个 CPU 修改了内核的数据结构,如页表,或处理了一个中断而这个中断的处理需要向另一个 CPU 运行的进程发送信号时,Linux 通过 CPU 间消息传递来通知另一个 CPU.Linux 提供了一套 CPU 间消息传递接口,并在所有需要在 CPU 间传递信息时调用这些接口.

在这些处理中,进程调度与硬件没有任何关系,而且 Linux 针对 SMP 处理器的进程调度也比较完善,所以这部分保留原来的代码,不做修改.其他 3 个方面都与硬件相关,是实现 Linux 对龙芯 2 号同时多线程处理器支持的关键所在.下面将分别介绍这 3 个部分.

1) 启动过程

首先分析 Linux 启动龙芯体系结构的 SMP 处理器的过程,然后介绍如何实现龙芯 2 同时多线程操作系统的启动.

Linux 启动龙芯体系结构的 SMP 处理器的过程如图 5 所示.

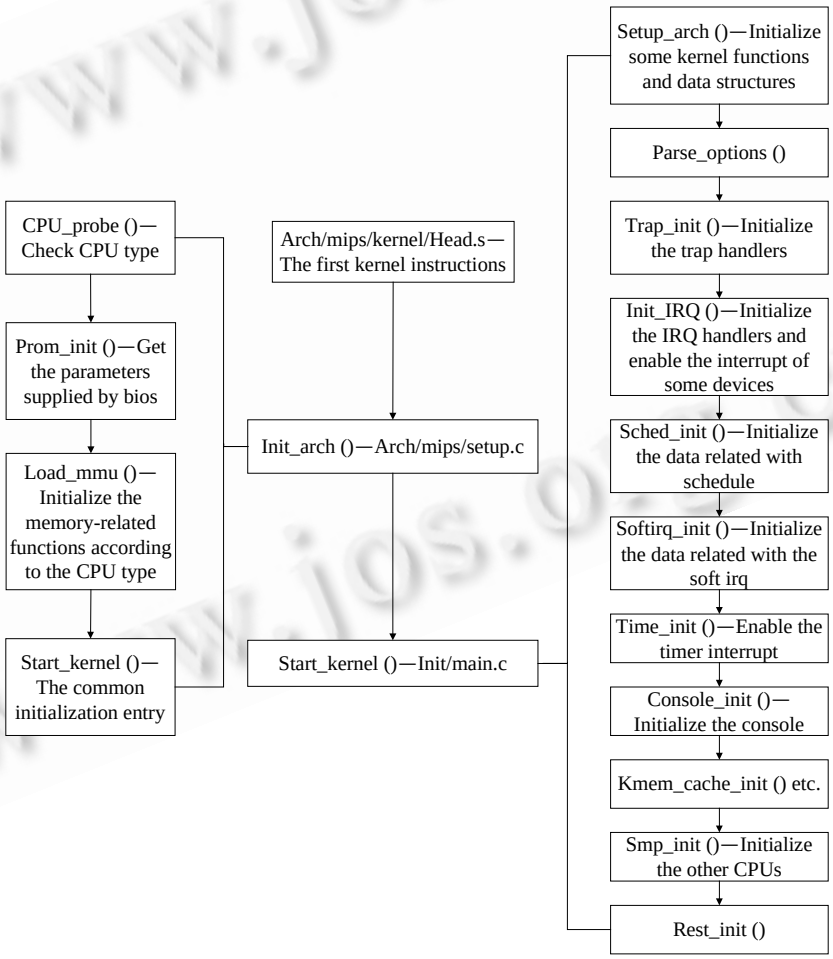

Fig.5 SMP initialization procedure in Linux

图 5 Linux 启动 SMP 处理器的过程

内核在经过 init_arch()函数进行一些基本的硬件初始化后,将跳转到 Linux 的通用启动入口 start_kernel(). start_kernel()函数首先通过调用 setup_arch()等函数设置了一些与硬件相关的内核函数指针,确定例外、中断处理函数,并初始化所有外设和当前运行的主 CPU.这些函数中有一部分是初始化主板及外设的代码,由于龙芯 2 号同时多线程处理器在外部接口上没有改变,将使用龙芯 2 号现有的主板,所以这部分代码可以用 Linux-godson (支持龙芯 2 的 Linux 操作系统)原有的代码.另外,由于龙芯 2 号同时多线程处理器中每个逻辑 CPU 将与原来的龙芯 2 号有一样的例外入口和控制寄存器等,只是增加了一个新的例外类型——CPU 间中断,所以,用于初始化主 CPU 以及设置例外和中断处理函数的代码基本上都可以使用 Linux-godson 原有的代码,只要增加一个 CPU 间中断的异常处理函数.

在初始化外设和主 CPU 完成后,start_kernel()函数将通过 sched_init(),mem_init()等函数对内核的数据结构进行初始化.这部分代码与硬件无关,因此不需要进行修改.然后,start_kernel()函数将调用 smp_init()函数,这个函数将启动次 CPU,完成对次 CPU 的初始化.下面具体介绍 Linux 中 smp_init()函数的实现,分析其中需要进行改动之处.

smp_init()函数的代码如图 6 所示.这个函数先通过调用 smp_boot_cpus()函数为启动次 CPU 准备一些内核数据,并向次 CPU 发出启动命令.所有次 CPU 在启动完毕后,将等待主 CPU 发出命令同时开始执行任务,这个任务将通过 smp_commence()发出.可见,Linux 为启动 SMP 处理器的次 CPU 设定了一个统一的框架,同时预留了一些实现统一功能的接口——smp_boot_cpus()和 smp_commence(),每种处理器根据自己的情况实现这些接口.由于 Linux 对龙芯体系结构的处理器统一实现了 smp_commence()函数,所以,我们只需要实现 smp_boot_cpus()函数的相关代码.

```
//init/main.c
static void__init smp_init (void)
{
  /* Get other processors into their bootup holding patterns */
  smp_boot_cpus();
  wait_init_idle=cpu_online_map;
  clear_bit (current→processor, &wait_init_idle); /* Don't wait me*/

  smp_threads_ready=1;
  smp_commence();

  /* Wait for the other cpus to set up their idle processes */
  printk ("Waiting on wait_init_idle (map=0x%lx)\n",wait_init_idle);
  while (wait_init_idle) {
        cpu_relax();
        barrier();
  }
  printk("All processors have done init_idle\n");
}
```

Fig.6 The codes of smp_init function

图 6 Smp_init 函数代码

参考其他处理器对这个函数的实现,龙芯 2 号同时多线程处理器结合硬件提供的接口作了如图 7 所示的实现.由于每个 CPU 都需要有一个 IDLE 进程以及一个初始的内核堆栈,所以,smp_boot_cpus()函数第 1 步先将当前进程复制了一份,并修改了进程数据结构中有关逻辑 CPU 的信息,为次 CPU 准备好了一个执行环境;然后,通过龙芯 2 号同时多线程处理器提供的 modech 将处理器切换到同时多线程模式,并通过 crt 指令让次 CPU 从 smp_bootstrap 处开始执行.在 smp_bootstrap 地址处,次 CPU 首先会将自己的内核堆栈指针 sp 指向前面为它准备的内核堆栈上,然后跳转到重新编写的 start_secondary 函数来进行初始化.初始化的工作主要是设置次 CPU 的控制寄存器,将次 CPU 调整到可以运行独立进程的状态.在次 CPU 初始化完成后,它将设置启动完成信号,同

时自己进入等待状态,等待主 CPU 发送出起步信号再开始执行任务.smp_bootstrap 函数在发现次 CPU 的启动完成信号为 1 后,将先重置一些内核变量,如 CPU 的数目以及物理 CPU 号和逻辑 CPU 号的映射表等,然后返回到前面的 smp_init 函数中.

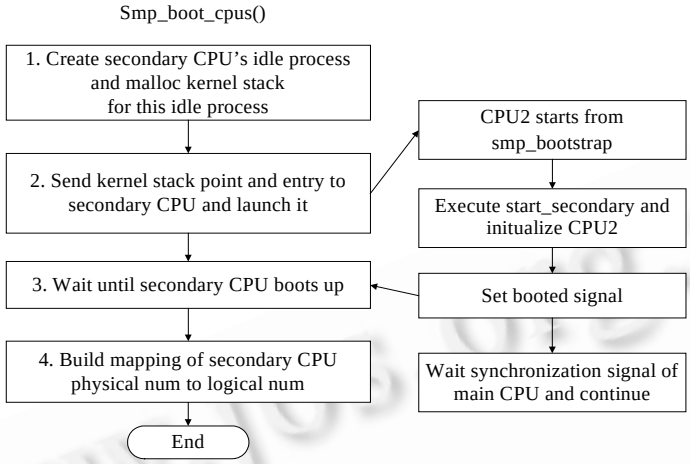

Fig.7 Initialization procedure of the secondary logic CPU of Godson-2 simultaneous multithreading processor

图 7 龙芯 2 号同时多线程处理器次逻辑 CPU 的启动过程

2) 关于同步共享相关代码的修改分析

前面已经介绍了 Linux 中将通过 spin_lock 等函数来实现内核资源的同步,这些函数针对不同的处理器有不同的实现.Linux 中龙芯体系结构的处理器基本上都是通过 ll/sc 来实现这些函数.龙芯 2 号同时多线程处理器完善了这两条指令的功能,使它们满足同步共享的要求.所以,我们将直接使用 Linux 中龙芯体系结构处理器的统一代码,不需要对这些函数作任何修改.

3) CPU 间消息传递的相关实现

上面已经介绍,Linux 有一套完善的 SMP 处理器的 CPU 间消息传递机制,并为不同的处理器留了几个接口函数,相应的接口函数见表 1.龙芯 2 号同时多线程处理器是通过 CPU 间中断来实现消息传递的,我们将增加一个中断响应函数和一个中断发送函数,用中断发送函数来实现表 1 中的接口,而中断响应函数则根据 CPU 间中断的类型跳转到 Linux 中写好的相应处理函数.

Table 1 CPU message passing functions in Linux

表 1 Linux 中 CPU 间消息传递接口函数表

Function	Description
smp_send_reschedule	Send RESCHEDULE to other CPUs
smp_call_function	Send SMP_FUNCTION to other CPUs
smp_send_stop	Send STOP to other CPUs
Flush_tlb_all	Send INVALIDATE_TLB to other CPUs
Flush_tlb_mm	Send INVALIDATE_TLB to other CPUs
Flush_tlb_range	Send INVALIDATE_TLB to other CPUs
Flush_tlb_page	Send INVALIDATE_TLB to other CPUs

龙芯 2 号同时多线程处理器在发生 CPU 间中断的时候,会产生一个 19 号类型的异常(即 CPU 间中断的异常).同时,操作系统在异常入口处增加相关代码,判断异常类型是否为 19 号:如果是,就跳到我们实现的 CPU 间中断响应函数中.这个函数将会读取 23 号 CP0 控制寄存器(即 CPU 间中断控制寄存器),获取 CPU 间中断的类型,并根据不同的类型跳到相应的处理函数中,同时将通过设置 23 号 CP0 控制寄存器清除 CPU 间中断.

龙芯 2 号同时多线程处理器的 CPU 间中断发送函数 core_ipi_send()的代码如图 8 所示.这个函数通过 *mfc*0_*t* 和 *mtc*0_*t* 指令来实现对另一个逻辑 CPU 的 23 号 CP0 控制寄存器进行读取,并设置需要产生的中断类型位,代码中的 action 代表要发送的中断类型.

```
//arch/mips/kernel/smp.c
 void core_send_ipi(int CPU, unsigned int action)
 {
        spin_lock(&smp_interrupt_lock);
            __asm(
            "set mips3\n\t"
            "mfc0_t k0,$23\n\t"
            "or     k0,k0,%0\n\t"
            "mtc0_t k0,$23\n\t"
            "set mips0\n\t"
            :: "r" (action)
        );
            spin_unlock(&smp_interrupt_lock);
 }
```

Fig.8 Interrupt sending function for Godson-2 simultaneous multithreading processor

图 8 龙芯 2 号同时多线程处理器的 CPU 间中断发送函数

5 性能评测

我们在龙芯 2 号处理器的 RTL(register transfer level)代码上实现了龙芯 2 号同时多线程处理器,同时在 Linux 2.4.20 的基础上实现了龙芯 2 号同时多线程处理器相应的操作系统.通过仿真加速器,在龙芯 2 号同时多线程处理器上启动 Linux 操作系统,运行测试程序.测试结果如图 9 所示,平均 IPC(instructions per cycle)提高 31.1%,充分说明了龙芯 2 号多线程处理器软硬件接口和操作系统设计的有效性.

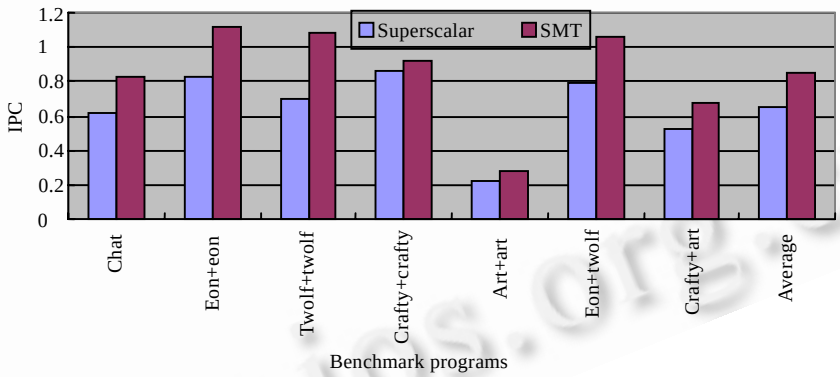

Fig.9 Multi-Process benchmark performance of SMT, compared with superscalar processor

图 9 同时多线程与超标量的多进程负载性能比较

6 总 结

本文在分析同时多线程的软件需求的基础上,提出龙芯 2 号同时多线程处理器的软硬件接口协同设计解决方案,给出相应的操作系统实现方案.我们在龙芯 2 号处理器的基础上实现了龙芯 2 号同时多线程处理器的 RTL 代码,并且支持本文所介绍的软硬件接口.同时,在 Linux 2.4.20 的基础上实现了龙芯 2 号同时多线程处理器相应的操作系统,能够正确运行像 SPEC CPU2000 这样的大型应用软件.今后将进一步对 Linux 操作系统进行改进,充分发挥其在两个逻辑 CPU 上调度不同的进程的能力,更大程度地提高龙芯 2 号同时多线程处理器的性能.

由于片内多核处理器中的多个逻辑处理器也被当作 SMP 来使用,因此,本文的分析和设计方案对于片内多核处理器的设计也有一定的借鉴作用.

致谢 在此,我们向对本文的工作给予支持和建议的同行,尤其是中国科学院计算技术研究所龙芯课题组的老师和同学表示感谢.

References:

[1] Hu WW, Zhang FX, Li ZS. Microarchitecture of the Godson-2 processor. Journal of Computer Science and Technology, 2005,20(2): 243−249.

[2] Ungerer T, Robic B, Silc J. Multithreaded processors. The Computer Journal, 2002,45(3):320−348.

[3] Tullsen DM, Eggers SJ, Levy HM. Simultaneous multithreading: Maximizing on-chip parallelism. In: Proc. of the 22nd Annual Int'l Symp. on Computer Architecture. New York: ACM Press, 1995. 392−403.

[4] Tullsen DM, Eggers SJ, Emer JS, Levy HM, Lo JL, Stamm RL. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In: Proc. of the 23rd Annual Int'l Symp. on Computer Architecture. New York: ACM Press, 1996. 191−202.

[5] Eggers SJ, Emer JS, Levy HM, Lo JL, Stamm RL, Tullsen DM. Simultaneous multithreading: A platform for next-generation processors. IEEE Micro, 1997,17(5):12−19.

[6] Marr DT, Binns F, Hill DL, Hinton G, Koufaty DA, Alan Miller J, Upton M. Hyper-Threading technology architecture and microarchitecture. Int'l Technology Journal Q1, 2002,6(1):4−15.

[7] Tullsen DM, Lo JL, Eggers SJ, Levy HM. Supporting fine-grained synchronization on a simultaneous multithreading processor. In: Proc. of the 5th Annual Int'l Symp. on High-Performance Computer Architecture. Washington: IEEE Computer Society, 1999. 54−58.

李祖松(1977－),男,福建永安人,博士生,主要研究领域为高性能计算机体系结构,功能验证,VLSI 设计.

胡伟武(1968－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为高性能计算机体系结构,并行处理,VLSI 设计.

许先超(1979－),男,硕士,主要研究领域为高性能计算机体系结构,操作系统.

唐志敏(1966－),男,博士,研究员,博士生导师,主要研究领域为高性能计算机体系结构,并行处理.