2010年 7月 第 7期
Vol. 31 No. 7 2010

分布式 DSP实时操作系统的设计与实现

舒红霞，王继红

（武汉数字工程研究所，湖北 武汉 430074）
E-mail: shirleyshu@ustc.edu

摘 要：为了适应 DSP应用的复杂性和多样性，缩短开发周期，针对多片 ADSP-TS201 S DSP构成的并行数字信号处理系统，从任务管理、调度管理、中断管理、时钟管理、系统路由管理、存储管理、信号量、资源、事件、消息传递等方面，自主研发了分布式 DSP实时操作系统，简称 TFOS. 相关性能指标表明，TFOS完全可以满足用户需求及实时性要求，并具备可移植性，能适用由不同 DSP芯片组成的 DSP并行系统.

关 键 词：分布式；实时操作系统；数字信号处理；并行；ADSP-TS201 S
中图分类号：TP316 文献标识码：A 文 章 编 号：1000-1220(2010)07-1407-06

Design and Implementation of a Distributed DSP Real Time Operating System

SHU Hong-xia, WANG Ji-hong
(Wuhan Digital Engineering Institute, Wuhan 430074, China)

Abstract: In order to adapt the complexity and variety for the application of DSP and to short the period of development, a distributed DSP real-time operation system named TFOS was developed for a parallel DSP system, which consists of multi-DSPs (ADSP-TS201 S DSP). TFOS was designed and implemented from ten aspects including task management, task scheduling management, interrupt handling, timer management, system route management, memory management, semaphore, resource, event and message passing. The performance figures show that TFOS is satisfied with user need and real time constraint. TFOS is also general, transplantable and fit for various DSP parallel system.

Key words: distributed, RTOS, DSP, parallel, ADSP-TS201 S

1 引 言

随着数字信号处理（DSP）技术的迅猛发展，各种高性能的数字信号处理芯片应运而生，在雷达、声纳、图像处理、通信、生物医学、自动控制等领域发挥越来越重要的作用[1,2]. 然而，在实际应用中，由于需要处理的数据量非常大、实时性要求高，即使采用目前最先进的 DSP芯片，单个器件也无法满足实际的应用需求，因此，通常采用多 DSP构成的并行数字信号处理系统. 并行数字信号处理系统日益庞大，应用趋于复杂化，系统实时性、可靠性需求的提高，需要考虑的各种情况也成指数增加，使得软件开发的难度和工作量相应增加，但开发周期却要求越来越短. 软件开发人员不仅要面对复杂的软件算法，同时还必须对系统的硬件结构有深入的了解，把大部分精力放在各种外围设备和相关硬件控制上，整个过程复杂而艰辛，这些都极大地影响着开发进度，阻碍了 DSP系统的推广应用. 因此，为并行数字信号处理系统开发小巧精致、便于移植的分布式实时操作系统[3-6]，具有实际的意义.

针对并行 DSP应用的复杂性和多样性，结合自行研制的多 DSP并行数字信号处理系统，设计并实现了运行在美国 ADI公司的 ADSP-TS201 S数字信号处理器上的满足用户需要的高性能分布式 DSP实时操作系统，简称 TFOS. TFOS具有一个基于优先级的抢占式实时内核，并具有通信透明、编程方便、应用程序容易维护和升级等优点，其主要功能包括任务管理、调度管理、中断管理、时钟管理、系统路由管理、存储管理、信号量、资源、事件、消息传递等. 并且，TFOS具有通用性和可移植性，它能很容易地移植到不同的 DSP并行系统上，并且保证系统的实时性.

2 并行数字信号处理系统

图 1（见下页）是并行数字信号处理系统的硬件总体框图，称为 TS201并行系统. 系统中的 DSP芯片采用的是美国 ADI公司的高性能数字信号处理器 ADSP-TS201 S，它具有很强的数据处理能力和很高的运算速度[7,8].

系统由主控板、AD板、DA板以及多块 DSP板组成. 主控板用做 CPCI总线仲裁器，用于应用程序的编码、编译、链接、下载等功能. AD板、DA板上各有一片 ADSP-TS201 S（简称 TS201），并通过该处理器及共享的 CPCI总线与宿主机进行通信. 每块 DSP板上有 6片 TS201，每片 TS201称为一个节点，采用松耦合结构互连，不共总线，这些节点之间通过 LINK口进行通信，CPCI总线只能和一个节点交换数据，称这个节点为根节点，称其它节点为叶子节点. AD板、DA板和 DSP

收稿日期：2009-04-20 作者简介：舒红霞，女，1980年生，硕士，主要研究方向为分布式实时操作系统、并行计算；王继红，男，1960年生，研究员，主要研究方向为分布式实时操作系统、并行处理系统软件开发环境.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

板之间通过 TS201的 LINK口互连进行通信.

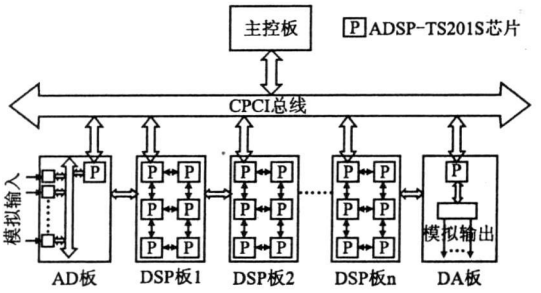

图 1 系统硬件总体框图

Fig. 1 Overall block diagram of system hardware

系统中 TS201的内核时钟为 600MHz,每个 LINK口单方向速率可高达 600MB/s,每个 LINK口组合起来最高吞吐量可达 1.2GB/s,因此, DSP之间可以实现低开销的高速传输. 根据系统的应用需求和硬件系统特点, TFOS具有一个单处理器操作系统内核, 支持系统的分布式处理环境, 支持传统实时操作系统的多任务管理机制, 支持多处理器上任务的同步与互斥, 并提供各处理器之间可靠、高效的通信机制.

3 TFOS的设计与实现

3.1 体系结构

操作系统的结构是影响操作系统灵活性和可扩展性的关键因素, 最好的操作系统的内部结构模型是一个层次性的结构, 相当于一个倒置的金字塔, 每一层都建立在较低层的功能之上. 针对并行数字信号处理系统, TFOS系统结构如图 2 它由 3层结构组成:

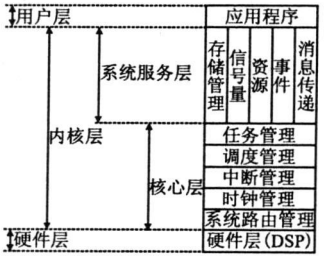

图 2 体系结构

Fig. 2 Architecture

最底层是硬件层, 即数字信号处理器 DSP层.

在硬件层上面是内核层, 内核层又分为核心层和系统服务层. 其中, 核心层包括系统必不可少的与硬件相关的管理功能, 如任务管理、调度管理、中断管理、时钟管理、系统路由管理和系统初始化. 在核心层上面是系统服务层, 这一层提供了操作系统的基本功能, 主要包括: 存储管理、信号量、资源、事件和消息传递. 系统服务层独立于硬件系统, 与硬件相关的操作全部通过对核心层的调用实现.

最上层是用户层, 即应用层, 包括各种应用程序.

3.2 核心层

核心层与硬件系统密切相关, 在设计并实现 TFOS时, 将该层中与 DSP芯片直接相关的部分提取成单独的函数, 便于今后其它 DSP芯片的移植.

3.2.1 任务管理

TFOS是为多任务操作系统设计的, 为了突显任务的相对重要性, 实现任务按优先级进行调度, 为任务设置了优先级. 定义空闲任务的优先级为 63, 任务优先级最高为 0.

TFOS的任务具有就绪态、挂起态和运行态 3种状态, 关系如图 3 并设置了就绪队列, 所有处于就绪态的任务按照优先级从高到低被存储于就绪队列中.

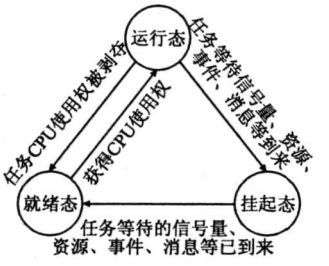

图 3 任务状态转换图

Fig. 3 Task state transition diagram

每个任务都有一个称为任务控制块的结构体, 它所含内容是对任务特性的精确描述, 在控制处理任务状态中至关重要. TFOS的任务控制块结构如下:

```
typedef struct
    unsigned ID;  //任务 ID
    void * link;  //链表指针
    int prio;  //优先级
    int suspend;  //挂起标识
    MsgQueue * msgQue;  //消息队列
    unsigned * jsp;  //现场保护区位置指针
    unsigned * ksp;  //现场保护区位置指针
} PCS;
```

由于 TFOS是一个分布式操作系统, 为了使信号量、资源等具有分布式特征, TFOS均采用面向对象的管理机制, 将任务、信号量、资源、事件等均看成是一种对象. 每种对象都拥有一个 32位的对象 ID, 由三部分组成: ID的高 16位确定对象所在节点, 低 16位中的高 8位确定对象类型, 低 16位中的低 8位确定本节点中的同类对象. 在任务控制块结构体 PCS中, 通过任务 ID可以将任务与其他对象如信号量、事件等区分; 通过链表指针可以将任务等对象链接; 通过优先级使系统具有按优先级高低进行任务调度的功能; 通过挂起标识可以记录任务的状态; 通过消息队列可以实现任务到任务之间的直接消息传递; 通过现场保护区位置指针可以找到任务栈的位置, 保存或恢复任务的现场, 实现任务切换. 为了充分利用 TS201的内部存储器资源和总线带宽, 提高操作系统内核的响应时间, 采用了双堆栈技术, 即给每个任务分配两个堆栈, 因此, 这里保存两个现场保护区位置指针.

3.2.2 调度管理

为了保证系统实时性, TFOS采用了基于优先级的可抢

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

占调度机制, 并且, 为了防止某个任务独占 CPU, 相同优先级的任务之间实行时间片轮转调度方式. 因此, TFOS允许高优先级任务抢占低优先级任务的执行. 通过这种调度机制, 高优先级的任务一旦就绪, 总能很快得到 CPU的控制权, 从而增强了系统响应的实时性.

但是, 在实际应用中, 并不是每次抢占都是合理的, 非期待的抢占会导致系统出现无法预料的情况, 因此, TFOS将任务设置成可抢占和不可抢占两种方式. 对于一个正在运行的任务, 这时有一个更高优先级的任务处于就绪态, 若当前运行任务被设置成可抢占方式, 那么当前运行任务的 CPU使用权被剥夺, 即更高优先级的任务立刻得到 CPU的控制权; 若当前运行任务被设置成不可抢占方式, 那么该任务的 CPU使用权不能被剥夺, 直到其自动放弃 CPU使用权. 其流程如图 4所示.

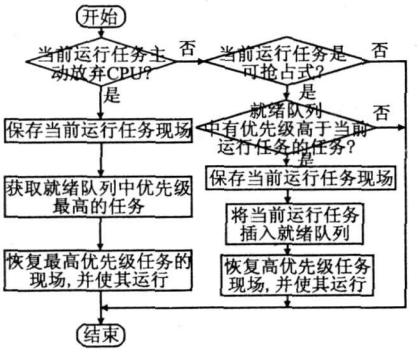

图 4　可抢占调度流程图

Fig. 4　Preemptive scheduling flow chart

保存和恢复现场的过程, 实际是 CPU当前寄存器内容入任务栈和从任务栈加载到 CPU的过程. TS201是一款性能极高的静态超标量处理器, 专对大的信号处理任务和通信结构进行了优化. TS201的静态超标量结构使 DSP每周期能执行多达 4条指令, 并且, 4条相互独立的 128位宽的内部数据总线分别连接 6个 4Mbit内部存储器块中的一个, 提供四字的数据、指令及 I/O访问和 33.6GB/s的内部存储器带宽. 为了保证系统实时性及响应时间, 针对这些特点在设计保存和恢复现场部分时, 采用四字存取汇编指令来提高数据存取带宽, 同时通过在同一周期访问不同的存储块并遵从指令并行执行规则来提高指令级并行程度.

调度管理中仅保存和恢复任务运行现场与 DSP芯片的特性相关, 其调度方式与硬件无关, 因此, 将保存和恢复现场提取成现场保存函数 TFTaskSave和现场恢复函数 TFTaskRestore.

3.2.3　中断管理

TFOS中, 中断不属于任何一个任务. 每当中断发生, 内核能够暂停正在执行的任务, 保护现场后自动地转去执行相应的中断处理程序, 处理完该事件后, 根据被 "打断 "任务的性质来判断是否需要进行调度: 若被 "打断 "的任务是不可抢占任务, 那么恢复现场后直接返回到被 "打断 "任务的断点处继续运行; 若被 "打断 "的任务是可抢占任务, 那么进行一次调度, 进入就绪态的优先级最高的那个任务运行.

TFOS允许中断嵌套, 并设定了最大允许嵌套层数. 当中断发生时, 如果, 内核正在处理中断, 并且刚刚发生的中断优先级高于正在被处理的中断优先级, 那么, 内核将打断正在处理的中断, 保护现场后自动地转去执行刚刚发生的中断的中断处理程序, 处理完成后, 直接恢复到刚才被打断的中断的现场继续运行. 中断使得内核可以在事件发生时才予以处理, 而不必连续不断地查询是否有事件发生. 并且, 可以通过两条特殊指令关闭全局中断 (不响应中断)和打开全局中断 (响应中断), 让内核在被允许的情况下适时地处理实时事件. 这种需要关闭全局中断的区域, 在 TFOS中, 称为 "临界区 ", 关闭全局中断称为进入临界区, 打开全局中断称为退出临界区, 这两个操作与硬件有关, 将其抽象成进入临界区函数 TFCriticalEntry和退出临界区函数 TFCriticalExit在实时环境中, 关中断和开中断的时间应该尽量地短. 因此, 在 TFOS中, 要尽可能减小开关中断时间, 但是有些操作又必须是在不可被抢占的情况下进行, 为此, 设计了一种称为 "不可调度区 "的区域. 在这种区域中, 即使中断发生了, 也不会影响被 "打断 "任务的执行, 同时又及时响应了实时事件.

中断服务和任务切换机制的结合使得 TFOS具有处理系统内和应用层事件的能力, 可以使中断服务程序足够短, 提高了中断响应时间, 同时又保证了事件的实时处理.

3.2.4　时钟管理

TFOS还提供了时钟管理, 它可以用来控制任务延时, 防止任务在等待事件、消息、信号量时的无限等待. TFOS提供了时钟滴答中断和时钟管理任务, 一旦设定的时间片到达, 就会引起时钟滴答中断, 从而进入时钟管理任务将睡眠任务唤醒.

DSP芯片一般都提供了定时器, 根据 DSP芯片提供的定时器设置定时器时间片大小、定时器中断并启动定时器, 这一部分是与 DSP芯片相关, 为此也提取成一个定时器设置函数 TFTimerSet. TS201提供了 2个定时器, 选择 TIMER0作为系统时钟.

TFOS设计了时钟滴答中断处理函数和时钟管理任务. 其中, 时钟滴答中断处理函数主要用来记录系统时间, 并唤醒时钟管理任务工作. 时钟管理任务管理一个差分链表, 该差分链表用来记录各种等待时间的任务, 其数据结构如下:

```
typedef struct {  //差分链表
    TimerChan * link;  //链表指针
    unsigned ticks;  //等待时间
    PCS * pcsLink;  //等待任务链表
} TimerChan;
```

其中, 等待任务链表是由所有等待时间相同的任务构成的链表.

当响应一个时钟滴答中断, 差分链表中的等待时间减 1. 当时钟管理任务被激活, 它将差分链表上等待时间已经完成的所有任务同时唤醒, 并设置成就绪态, 插入就绪队列中.

3.2.5　系统路由管理

TFOS是针对 DSP并行系统设计的, 支持分布式处理, 系

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

统路由管理是 TFOS的核心. TS201并行系统中, AD、DA、DSP板之间以及板上的 DSP之间均通过 LINK口互连, 因此, 可以将这 3种板构成的整个结构 (即图 1的 CPCI总线下半部分)看作一个 DSP网络, 并选择其中一块板中能与宿主机通信的那个节点为该 DSP网络的根节点. 因此, 将 TS201并行系统的通信分为 2种:

(1) DSP节点间的通信.

对于 DSP节点之间的通信, 它是通过 TS201之间的 LINK口实现的 DMA通信方式, 速度非常快, 因此, TFOS的系统路由管理中设置了两种类型的任务, 发送路由任务和接收路由任务. 其中, 发送路由任务首先从其消息队列中接收数据, 然后, 将接收到的数据通过 LINK口发送到目的 DSP上. 接收路由任务主要用来接收或者转发来自其它 DSP节点上的数据. 为了保证系统的实时性, TFOS的发送和接收方式是非死等方式, 即为发送路由任务设置了发送完成事件, 为接收路由任务设置了接收完成事件. 只有当发送路由任务将所有数据发送完毕, 发送完成事件被触发, 发送路由任务才能继续发送下一个数据, 否则挂起到该事件上. 同样地, 只有当接收路由任务接收完所有数据, 接收完成事件被触发, 接收路由任务才能继续接收下一个数据, 否则挂起. 以发送路由任务为例, 其流程图如图 5.

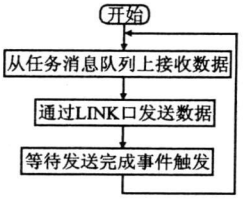

图 5 发送路由任务流程图

Fig. 5 Send routing task flow chart

由于 TS201具有 4个 LINK口, 两片 TS201之间通过 LINK口互连, 因此, 在通过 LINK口传输数据时, 需要通过一套路由机制来获得发送 LINK口.

定义 1. 节点最短路径表定义为 $SPATH_i = \{P_{i1}, \cdots, P_{ij}, \cdots, P_{in}\}$, 其中, i表示节点号, n表示 DSP网络中的总节点个数, DSP网络中节点号命名为 1, 2, …, n; i表示其中的一个节点号. P_{ij}对应于从节点 i到达节点 j最短路径中经过的第一个节点号, 当它为 0时, 表示节点 i不能到达节点 j.

定义 2. 节点路由表定义为 $ROUTE_i = \{L_{i1}, \cdots, L_{ij}, \cdots, L_{in}\}$, 其中, i表示节点号, n表示 DSP网络中的总节点个数, DSP网络中节点号命名为 1, 2, …, n; i表示其中的一个节点号. 从定义 1可以获得节点 i和节点 j之间的最短路径值 P_{ij}, L_{ij}对应于节点 i和节点 j之间的直接互连 LINK口号, 即节点 i与节点 P_{ij}之间的 LINK口号. L_{ij}可以为 0, 1, 2, 3, 4. 其中, 0~3分别对应 LINK0~LINK3, 4表示无 LINK口.

DSP之间进行通信时, 根据本地 DSP节点号、目的 DSP节点号, 查找节点路由表, 找到对应 LINK号, 并从该 LINK口将数据发送到中间节点, 直到到达目的节点.

在发送路由任务和接收路由任务中, 除了发送数据和接收数据时与硬件有关外, 其它过程与硬件无关. 因此只需要将发送数据和接收数据分别提取成发送数据函数 TFRouteSend和接收数据函数 TFRouteReceive.

(2) 宿主机与 DSP网络根节点之间的数据通信.

宿主机与 DSP根节点之间通过 TS201S总线与 CPCI总线连接, 并通过读写双口 RAM实现与宿主机的通信. TS201并行系统为宿主机与 DSP根节点之间的通信提供了一对中断, 当 DSP根节点向双口 RAM读写完数据会发送一个 HOST中断通知宿主机, 当宿主机向双口 RAM读写完数据会发送一个 IRQ中断通知根节点. 通信方式与 DSP节点间的通信类似, TFOS也设置了向宿主机端发送数据任务和从宿主机端接收数据任务两种任务, 其流程图与第 1种通信方式中的发送路由任务和接收路由任务类似, 只是根节点与宿主机之间进行通信是通过读写双口 RAM, 并在读写完成后发送一个 IRQ中断通知宿主机读写已完成.

3.3 系统服务层

3.3.1 存储管理

TFOS对存储器进行了管理, 用户可以根据需要申请和释放内存. TFOS提供了两种存储管理机制: 定长存储管理和变长存储管理. 定长存储管理是一段物理上连续的内存空间, 被分成了一组大小相同的缓冲区, 每次申请和释放都以缓冲区为单位. 变长存储管理是在一段物理上连续的内存空间中用户根据需要从中分配一个可变长度的内存空间. 两种管理机制各有优势: 定长存储管理的灵活性不如变长存储管理, 但其分配效率较高, 也不会产生内存碎片; 变长存储管理分配内存时比较灵活, 但是容易产生内存碎片.

为了方便用户使用, 根据实际需要选择不同的内存分配方式, TFOS提供了下列原语:

1) 申请定长内存 TFFixMalloc;

2) 申请变长内存 TFVarMalloc;

3) 释放内存 TFFree.

3.3.2 信号量

TFOS实现了信号量用于控制共享资源的互斥存取, 通知某一事件的发生或两个任务之间的同步. 设计的信号量具有分布式特征, 既可以用于处于同一个处理节点上的任务之间的同步与互斥, 也可以用于处于不同处理节点上任务之间的同步与互斥.

TFOS实现了计数信号量, 其数据结构如下:

```
typedef struct{
    unsigned semaCount;t  //信号量值
    void * link;  //等待任务链表
} SEMA;
```

其中, 等待任务链表是由所有等待该信号量的任务构成的链表.

提供的原语有:

1) 发送信号量 TFSemaSignal, 对信号量值加 1.

2) 等待信号量 TFSemaWait, 对信号量值减 1.

调用 TFSemaSignal原语时, 如果信号量是本地节点的, 那么进行如下操作: 若没有任何任务等待这个信号量, 则将其

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

值加 1; 若有任务等待该信号量, 按照先来先服务原则, 将最早等待该信号量的任务唤醒 (设置成就绪态), 信号量值不改变. 如果信号量是异地节点的, 那么打包一个发送信号量信息包, 并经系统路由管理将该信息包发送到信号量所在节点. 调用 TFSemaWait原语时, 如果信号量是本地节点的, 那么进行如下操作: 若信号量值是 0, 则将任务挂起到该信号量的等待队列上; 若信号量值不是 0, 则任务不需要等待, 可以继续执行, 并将信号量值减 1. 如果信号量是异地节点的, 那么打包一个等待信号量信息包, 并经系统路由管理将该信息包发送到信号量所在节点, 同时阻塞本任务, 直到收到反馈信息为止.

3.3.3 资源

由于 TFOS中采用了抢先式调度算法, 任务在使用信号量对共享资源进行互斥访问时, 可能会出现优先级反转[4], 这时优先级高的任务必须等待优先级低的任务完成后才能执行. 为了有效地避免此类现象的发生, 根据优先级天花板协议[9], TFOS设计了能有效防止优先级反转的资源用于控制共享资源的互斥访问. 它与信号量的设计方式类似, 同样具有分布式特征, 设计的原语有:

1)资源上锁 TFResourceLock: 当访问共享资源时, 将该共享资源上锁.

2)资源解锁 TFResourceUnlock: 当访问共享资源完毕时, 将该共享资源解锁.

这两种原语语意与信号量的类似. 唯一不同的是, 任务在资源上锁到解锁区间内, 其优先级被暂时性地提高, 退出该区间后, 恢复任务的初始优先级.

3.3.4 事件

TFOS实现了事件机制, 用于驱动任务执行. 事件是 TFOS对硬件中断的软件处理方式, 与信号量不同的是事件不具备分布式特征. 事件的数据结构如下:

```
typedef struct{
    unsigned count;  //事件值
    PCS * link;  //等待任务链表
} EVENT;
```

其中, 等待事件链表是由所有等待该事件发生的任务构成的链表.

设计的原语有:

1)发送事件 TFEventSignal: 事件发生, 则对事件值置 1.

2)等待事件 TFEventWait: 一旦响应事件, 则对事件值置 0.

调用 TFEventSignal原语时, 若没有任何任务等待这个事件, 则将其值置 1; 若有任务等待该事件, 将等待该事件的任务唤醒, 事件值不改变.

调用 TFEventWait原语时, 若事件值是 0, 则将任务挂起到该事件的等待队列上; 若事件值不是 0, 则任务不需要等待, 可以继续执行, 并将事件值置 0.

3.3.5 消息传递

基于直接消息传递机制[10], TFOS实现了一种简单可行的消息传递机制, 简称 TF-IPC. TF-IPC允许一个任务直接向另一个任务发送消息, 提供了异步通信和同步通信两种通信方式, 并且, 这两个通信的任务可以位于同一个处理节点也可以位于不同的处理节点, 使用的通信原语完全一致, 用户只需在发送原语中指定接收任务和发送方式, 就可以将消息直接传递到接收任务.

在分布式系统中使用 TF-IPC, 应用程序只需知道消息是从哪个任务传递到哪个任务即可, 而不必了解分布式系统的结构, 因此, TF-IPC使任务间的通信变得非常简单. 此外, TF-IPC不需要消息队列或邮箱的介入, 与传统的借用消息队列或邮箱的消息传递机制相比, 避免了多次拷贝的问题, 具有很高的通信性能.

TFOS实现了 4种消息原语:

1)发送消息原语 TFMsgSend: 向用户指定的任务发送消息.

2)无等待接收消息原语 TFMsgReceive: 若没有接收到需要的消息, 则直接返回.

3)无限等待接收消息原语 TFMsgReceiveW: 若没有接收到需要的消息, 那么将任务挂起, 直到等到该消息到来.

4)超时等待接收消息原语 TFMsgReceiveWT: 增加一个超时控制时间, 若在该时间内没有接收到需要的消息, 则直接返回.

如果发送任务和接收任务在同一个节点上, 那么不进行消息的拷贝, 直接将消息指针从发送任务传送到接收任务. 如果发送任务和接收任务在不同的节点上, 那么就需要由系统路由管理将该消息发送给接收任务.

4 TFOS性能及测试

针对图 1所示的 TS201并行系统, TFOS的代码占用存储空间为 19KB, 系统中断响应时间为 0.27微秒, 任务上下文切换时间为 0.58微秒.

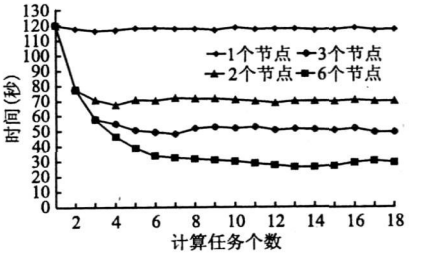

图 6　性能测试

Fig. 6 Performance testing

应用 TFOS开发了需要大量数据通信的 Mandelbrot Set图形实例[11,12]. 对于 Mandelbrot Set计算, 由于复平面上每个点的计算过程是独立的, 因此可以把复平面均匀分割成若干个区域, 每个区域上的点的计算对应一个计算任务, 并将这些计算任务分配到若干个处理器上并行计算.

在 TS201并行系统上, 插入一块具有 6个节点的 DSP板. 宿主机环境为: CPU Mobile Genuine Intel(R) 1.6GHz, 512M RAM, Windows2000操作系统. Mandelbrot Set计算区

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

域大小为 1024 * 768点, 单点最大迭带次数 10000次. 测试结果如图 6, 图中节点个数表示实际使用的 DSP节点个数.

从上页图 6可以知道, 当计算任务个数一定时, 节点个数增加, 时间减小地非常快, 说明在并行系统中增加节点个数能大大提高系统性能. 当节点个数一定时, 增加计算任务, 运行时间相应减小, 并且, 当计算任务增加到一定程度时, 运行时间达到最小值(如图 6中放大的点), 之后在小范围内上下浮动, 整个系统的性能并没有得到充分发挥, 这说明使用 TFOS开发应用程序时, 设置适当的任务个数将有利于整个系统性能的充分发挥.

影响系统性能的主要因素是:

(1)宿主机端程序主要负责画图, 但由于显卡的原因, 导致在屏幕上画图非常缓慢, 这在一定程度上影响了系统性能.

(2)DSP端程序是运行在 TFOS下的多任务程序. 当某个节点上的计算任务设置过少, 就会出现由于这些计算任务等待新的计算任务到来导致的 CPU空闲时间过多, 因此, 适当增加计算任务个数能有效提高 CPU的使用率. 但是, 如果每个节点上的计算任务过多, 同样存在多个任务分享 CPU, 并且, 各叶子节点与根节点之间交互数据的次数增多, 导致切换十分频繁, 从而影响整个系统性能. 因此, 必须在每个 DSP节点上合理地分配计算任务, 为了方便用户合理分配应用程序任务, 正在开发 TFOS调试工具.

5 结束语

为了适应 DSP应用的复杂性和多样性, 缩短其开发周期, 结合自行研制的多 DSP并行数字信号处理系统, 设计并实现了满足用户需要的高性能分布式 DSP实时操作系统(TFOS). 本文从基本结构和功能实现上介绍了 TFOS, 将其中与硬件相关的部分划分到核心层, 与硬件无关的部分划分到系统服务层, 这样可以方便地进行 DSP并行系统的移植. 目前, 已经将 TFOS成功地移植到基于多片 ADSP-TS101S DSP构成的 DSP并行系统中, 今后将尝试移植到更多其它 DSP芯片组成的 DSP并行系统中.

TFOS实现了可抢占式调度方式, 能有效地保证系统实时性; TFOS实现了具有分布式特征的信号量和资源机制, 能有效地实现同节点和不同节点任务间的互斥与同步; TFOS实现了针对多 DSP设计的直接消息传递机制, 能有效地简化任务间的通信, 提高系统性能, 满足实时性要求.

Mandelbrot Set图形显示需要大量数据通信, 基于 TS201并行系统, 将 TFOS应用到该实例, 运行效果良好, 并且, 取得了一定的加速比, 系统性能良好. 通过该实例, 说明了 TFOS能满足用户的需求, 并且用户可以在不必了解 DSP硬件知识的情况下, 直接使用 TFOS进行应用开发, 能大大缩短开发周期.

References:

[1] EYRE J, BIER J. The evolution of DSP processors from early architectures to the latest developments[J]. IEEE Signal Processing Magazine, 2000, 17(2): 43-51.

[2] Su Tao, Wu Shun-jun, Li Zhen-fang, et al. High performance DSP and high speed Real-time signal processing[M]. Xi'an: Xidian University Press, Jan. 2002.

[3] Jean J. Labrosse. MicroC/OS II: the real time kernel(second edition)[M]. Burlington, MA, USA: CMP Books, 2002.

[4] Baskiyar S, Meghanathan N. A survey of contemporary Real-Time operating systems[J]. Informatica, 2005, 29(2): 233-240.

[5] Andrew S. Tanenbaum, Albert S. Woodhull. Operating systems: design and implementation (second edition)[M]. USA: Prentice Hall Inc., 1997.

[6] Shi Yan, Mao Hai-cen, Zhang Tian-xu, et al. Design and Implementation of a Real-Time distributed operating system for parallel Image processing[J]. Mini-Micro Systems, 2005, 26(10): 1821-1827.

[7] ADSP-TS201 TigerSHARC processor hardware reference[M]. Norwood, Mass: Analog Devices Inc, 2004.

[8] ADSP-TS201 TigerSHARC processor programming reference[M]. Norwood, Mass: Analog Devices Inc, 2005.

[9] Luo Lei. Embedded Real-time operating system and application development[M]. Beijing: Beijing University of Aeronautics and Astronautics Press, 2005.

[10] Enea Embedded Technology. OSEck kernel User's guide[M]. Kista, Sweden: Enea Embedded Technology, 2004.

[11] Sun Bo-wen. Fractal algorithm and programming design: implemented with visual C++[M]. Beijing: Science Press, 2004.

[12] Benoit B. Mandelbrot. Fractals and chaos: the mandelbrot set and beyond[M]. German: Springer-Verlag, 2004.

附中文参考文献:

[2] 苏 涛, 吴顺君, 李真芳, 等. 高性能 DSP与高速实时信号处理[M]. 西安: 西安电子科技大学出版社, 2002.

[6] 石 岩, 毛海岑, 张天序, 等. 面向并行图像处理的实时分布式操作系统的设计与实现[J]. 小型微型计算机系统, 2005, 26(10): 1821-1827.

[9] 罗 蕾. 嵌入式实时操作系统与应用开发[M]. 北京: 北京航空航天大学出版社, 2005.

[11] 孙博文. 分形算法与程序设计: Visual C++实现[M]. 北京: 科学出版社, 2004.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net