基于网络处理器 IXP2400 系统的软件设计

葛敬国
(中国科学院计算机网络信息中心　北京 100080)

摘　要　网络处理器高性能的包处理能力及可编程的灵活性适应了当前网络发展需求，广泛应用于高端路由器、边缘多业务宽带接入、媒体网关和安全等领域。基于网络处理器成功构建一个网络系统的关键在于网络处理器软件系统的设计与开发，其核心问题就是要软件系统充分发挥网络处理器灵活性和高性能的特点，面向网络处理器的硬件体系结构编程，合理利用网络处理器，为优化数据包处理的各种硬件资源设计高效的多处理器、多线程并行机制。本文以网络处理器 IXP2400 实现高速网络应用为例，介绍基于网络处理器系统的软件开发过程和设计方法，探讨开发高性能的微码软件的策略和技术。首先介绍了基于网络处理器系统的硬件体系结构配置和软件开发框架、应用软件的系统分析和总体设计，着重分析了基于网络处理器系统的多微引擎、多线程的并行处理机制，以及互斥问题和包排序问题的解决方法，最后讨论了系统的性能评估方法。

关键词　网络处理器，并行，软件设计

Design on Network Processor IXP2400 Based on Systems

GE Jing-Guo
(Computer Network Information Center, the Chinese Academy of Sciences, Beijing 100080)

Abstract　As the Internet gets more and more complicated with the rise of new protocols and standards, the network product vendors are in need of intelligent processing at wire speed. Network processors satisfy the demand for intelligent processing at wire speed and system flexibility. The Intel IXP2400 network processor is designed for a wide range of applications including multi-service switches, routers, broadband access devices and wireless infrastructure systems. It is important for the product designers to leverage parallel processing in the data plane to make an efficient, cost effective end product. This paper will give the outline of design issues and challenges which a designer may face while designing a wire speed performance network system using network processors. It introduces the hardware architecture and software framework of network processor based system and describes the system development processes. Then it discusses the parallel programming models and challenges for multi-threaded multiple micro-engines. In the end, the performance evaluation methods for the system are provided.

Keywords　Network processor, Parallel, Software development

1　引言

随着 Internet 主干网络流量的指数性增长，主干路由器的处理线速已从 622Mbps 增加到 2.5Gbps、10Gbps，并且很快将达到 40Gbps。与此同时，一些复杂的控制与服务，如服务质量控制、网络安全以及其它附加功能如 AAA(Authentication, Authorization and Accounting)等均要求网络设备具有更多的灵活性和可扩展能力。而基于 ASIC(application specific integrated circuit)和 GPP(general purpose processor)的传统网络处理方案已经不能同时满足处理速度和灵活性这两方面的要求。为此，基于 ASIP(application specific instruction processor)技术的网络处理器(network processor)得到了广泛的发展，它将 RISC 处理器的低成本、灵活性与 ASIC 专用网络处理芯片的高性能、可扩展性很好地结合在一起，集成了多个为网络处理优化过的处理器以及一些专用硬件处理单元，如 CRC 校验、Hash 单元、硬件队列系统等，这些设计保证了网络处理器强大的数据包处理能力，在处理线速上已经达到或超过 10 Gbps 的处理能力[1]。网络处理器高性能的包处理能力及可编程的灵活性适应了当前网络发展需求，不仅在高端路由器市场已经被许多网络设备制造商选作新一代高端路由器设备的核心处理器，作为核心交换和路由设备中转发引擎 ASIC 芯片的取代品，而且正在向边缘多业务宽带接入、媒体网关和安全领域等更广泛的应用领域扩展。

基于网络处理器成功构建一个网络系统的关键在于网络处理器软件系统的设计与开发，其核心问题就是要软件系统充分发挥网络处理器灵活性和高性能的特点。实现网络处理器软件系统的一个挑战在于软件设计与网络处理器的硬件结构关系非常紧密，必须面向网络处理器的硬件体系结构编程，通过合理分配和使用网络处理器为优化数据包处理的各种硬件资源，如多处理引擎、专用硬件处理单元、各类寄存器、片上内存和其它硬件单元，才能得到一个高性能的系统[2,3]。由于不同厂商的网络处理器硬件体系结构不同，相应的软件实现差别很大，在不同系统中移植处理模块也非常困难。当前大多数网络处理器微引擎处理核心采用私有指令系统，只能使用其厂商提供的微码编程接口和编程工具。虽然一些厂商提供 C 编译器或其它高层语言接口，但为获得高性能的代

葛敬国　博士，主要研究领域为网络体系结构。

码，通常建议设计者采用基于微码的程序设计。网络处理器软件设计的另外一个挑战在于多处理器、多线程的并行程序设计。在一个并行多线程的系统中，主要涉及的问题有：如何保持数据包的顺序？如何在多个线程间传递数据包的状态？如何实现数据互斥？如何隐藏存储器访问时延？如何对数据包进行排队？实现线速处理需要硬件和软件共同解决这些问题，基于处理任务的不同特点，采用不同并行处理机制，消除处理瓶颈，以线速处理到达的数据包。

IXP2＊＊＊系列(2325/2350/2400/2800/2850)是 Intel 的网络处理器家族的第二代产品，首次实现了 Intel 的超级任务流水线技术(Hyper Task Chaining technology)，具有执行复杂算法和多层协议分析能力，可以扩展到 0C-192/10GbE 端口的线速处理性能，广泛应用于核心路由/交换设备、无线设施、宽带接入、多业务边缘设备等领域[4]。网络处理器的功能是由它所需要实施的操作以及它在目标网络系统中的任务决定的。本文以基于网络处理器 IXP2400 的网络系统，实现 2.5Gbps 速率的 POS 端口的 IPv4 转发引擎应用为例，介绍基于网络处理器系统的软件开发过程和设计方法，探讨开发高性能的微码软件的策略和技术，特别是讨论线速处理高速网络数据时遇到的很多问题。

本文的结构如下：第 2 部分给出了基于网络处理器系统的硬件体系结构配置和软件开发框架。第 3 部分给出 IPv4 转发引擎应用的系统分析和总体设计。第 4 部分分析讨论了基于网络处理器系统的微码编程模型、并行处理机制。最后是性能评估方法和本文的总结。

2 基于 IXP2400 网络系统的体系结构

本节首先给出软件设计人员看到的网络处理器 IXP2400 的硬件体系结构，介绍 IXP2400 的功能模块和特征，然后给出了一个典型的基于 IXP2400 网络系统的硬件平台。最后给出了系统的软件开发平台和软件框架。

2.1 IXP2400 的硬件体系结构

图 1 是 IXP2400 网络处理器的硬件体系结构，主要包括 6 个主要的功能模块，分别为一个 Intel XScal 嵌入式 RISC 处理器内核及其与外部设备接口的 XPI 单元、8 个 Intel 第二代可编程 RISC 微引擎(MicroEngine Version 2，简称 MEv2)、物理接口与交换接口 MSF 单元(Media and Switch Fabric Interface Unit)、QDR SRAM&DDR SDRAM 控制器单元、SHaC 多功能控制单元和 PCI 控制单元[5]。

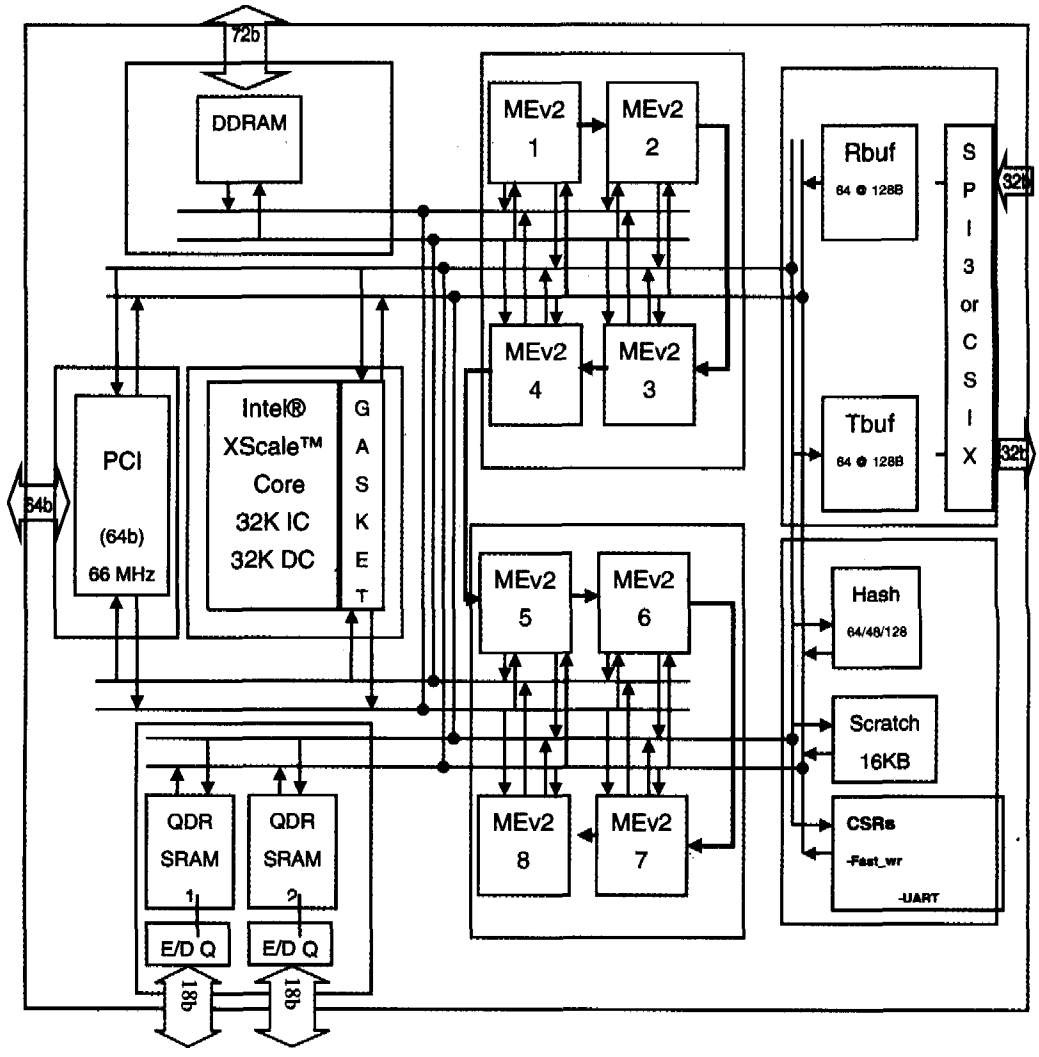

图 1 网络处理器 IXP2400 硬件体系结构

IXP2400 硬件体系结构具有以下一些特征：

·采用多内核并行处理器结构，片内处理器按任务分为控制平面处理器和数据层面处理器。控制平面处理器通常负责非实时的管理任务；数据平面处理器进行实时、线速数据分组处理。处理器 XScal 工作在控制平面，提供总的控制，处理高层协议，8 个并行微引擎工作在数据平面。微引擎是精简

的可编程处理器，在入口和出口处线速处理数据分组。

• 支持硬件多线程。为了提高网络处理器的资源利用率，每个微引擎还支持 4 个或 8 个硬件线程。每个线程都有一套专门的硬件来存放上下文(Context)，以获得线程切换的零开销。

• 优化指令集，设计专用硬件加速处理单元。采用 RISC 技术，结合多级流水线技术，大部分指令在一个时钟周期完成。针对网络协议处理特点，设置专用硬件加速处理单元，提供专用，指令如乘法指令、CRC 校验指令、哈希计算指令、字节对齐指令、硬件队列与环操作指令、CAM(Content Addressing Memory)查找指令、MSF 与 DRAM 间快速通道指令、状态判断与数据读写指令等。

• 优化的分级存储组织和分布式存取。网络数据处理需要进行大量的数据分组的接收、存储、复制、转发，对于高速网络处理来说，存储操作成为系统开销的一大瓶颈。为了解决这个问题，通常采用块数据移动技术和特殊优化的分布式数据存储结构。在微引擎内部有大量的不同类别的寄存器、本地存储器、CAM，在微引擎外部有片内存储器 Scratchpad。在网络处理器外部可以扩展大容量的片外存储器 SRAM 和 SDRAM，SRAM 用于存放需要快速查找的各种表结构，SDRAM 用于存放数据分组信息。模块可并行访问多种数据存储单元，不同数据存储单元的存取时间周期差异很大。

• 硬件支持的环与队列操作。网络数据处理涉及很多队列或环的数据结构操作，而入队或出队操作需要多次访问内存，极大地影响数据处理分组处理的周期。IXP2400 中的 SRAM 控制器提供了基于 SRAM 的先入先出队列，通过硬件实现了环与队列操作。在 SRAM 控制器内有一个称为队列矩阵的高速缓冲区，保存了环与队列操作需要的详细信息，减少环与队列操作中访问外部存储器的次数。

2.2 基于 IXP2400 的网络系统硬件平台

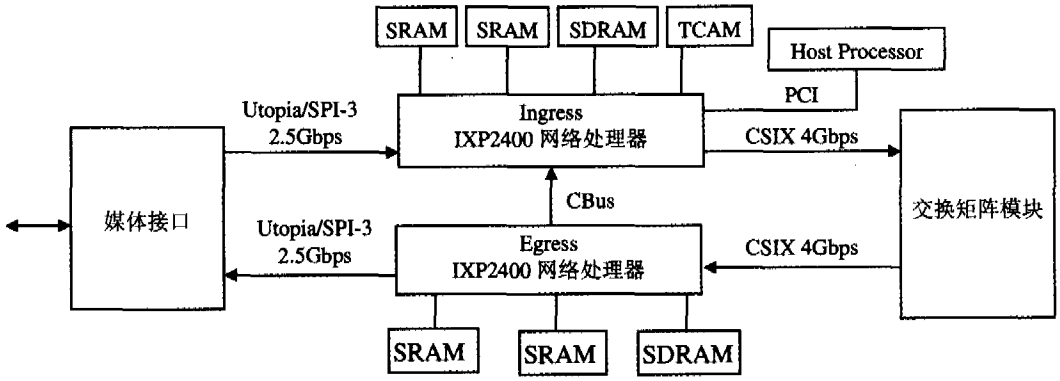

图 2 基于 IXP2400 的网络系统硬件结构

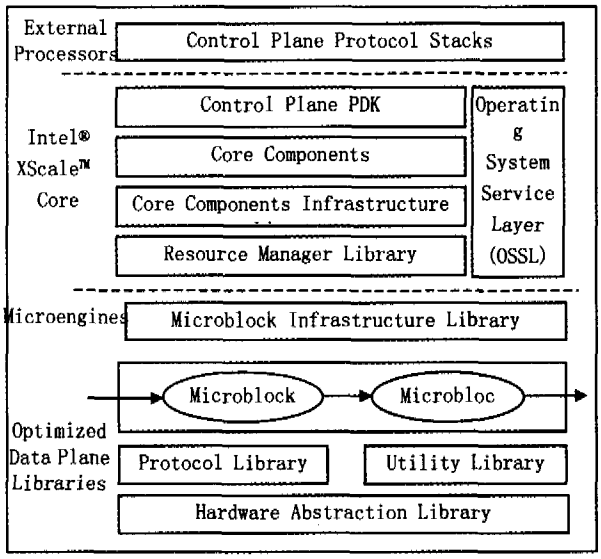

图 3 IXA 软件框架

图 2 是典型的基于 IXP2400 的网络系统硬件结构，系统包括物理媒体模块、网络处理器模块、交换矩阵模块、外部存储模块。系统按功能分为两大部分：入口处理部分和出口处理部分，使用两片 IXP2400 处理器芯片[6]。入口网络处理器从网络物理接口上接收数据分组，经处理后向交换矩阵模块转发。出口网络处理器从交换矩阵模块接收数据并向网络物理接口发送数据。两个处理器间有 CBus 总线，完成流量控制功能。出口网络处理器一般作为主设备，提供各种仲裁逻辑。媒体模块负责从网络链路上接受数据或者向网络链路发送数据，交换模块负责网络系统内部不同端口间数据的交换。数据网络处理器与媒体和交换模块之间通过 32 位宽的接口连接，接口可以通过编程设定为不同的标准，如 Utopia、SPI-3 或 CSIX-L1。接口的数据通道宽度和个数可以设置为不同的组合模式，如 4 * 8 位、2 * 16 位、1 * 32 位或者 8 位与 16 位的混合模式。内部存储模块负责待处理数据包以及转发表或与转发相关的状态信息的存储。协处理器是具有一些特殊功能的模块，如 TCAM 查找模块、IPSec 加密模块等。网络处理器 IXP2400 通过工业标准接口与外部存储系统或者协处理器连接。网络处理器还通过 PCI 接口与主控处理器连接。

2.3 Intel 网络处理器的软件开发环境

Intel 为基于其网络处理器的网络系统产品开发提供了一个较为完善的软件开发环境 IXA SDK(Internet Exchange Architecture Software Development Kit)[7]，可以让网络系统的硬件系统和软件系统并行开发。IXA SDK 包括工具集和软件框架两个部分。工具集包括微引擎的开发环境与开发工具，用于开发、调试运行于微引擎之上的微码应用程序。软件框架包含一些常用软件构件、粘合这些构件的框架以及基于构件与框架开发的示范应用。图 3 给出了 IXA 软件框架，在 XScale 核和微引擎处理层面都采用分层的体系结构，各层提供大量的可复用模块和函数接口。

3 网络系统应用需求

一个典型网络应用系统包括 3 个层面，即控制层面、管理平面和数据层面。控制层面处理协议消息，建立、配置和更新

一些转发表和一些状态数据。数据层面进行包处理时需要查找这些信息。管理层面通过用户界面负责配置系统、启动或终止某项功能、收集和报告系统统计信息等。数据层面负责线速处理和转发数据包，由于每个包都需要经过数据层面处理，因此性能是设计数据层面时考虑的最重要的因素。本文主要讨论数据层面的设计问题。

3.1 数据层面功能需求

基于 IXP2400 的网络系统，其数据层面包括两条数据通道，即慢速数据通道和快速数据通道[8]。快速数据通道是由微引擎负责处理的数据转发通道，大部分数据包通过此通道完成处理与转发。慢速数据通道是由 XScale 核负责处理的数据通道。由于一些例外数据包需要复杂的处理，处理过程也不一致，如数据包分段、带扩展头部的数据包处理等，它们不适合由快速数据通道处理，一般分发到慢速数据通道处理。在 IXP2400 多处理器、多线程的结构下，数据包流到达后需要经过一系列的处理过程。这些处理过程作为许多任务分发给 XScal 核或 8 个微引擎，每个微引擎被编程处理特殊的任务。当一个微引擎处理完自己的任务后，将处理上下文传递到下一个微引擎继续处理。

在入口方面，一个典型的数据包转发过程完成的任务有数据包接收、解封装/分类、查表转发、输入队列管理和调度、发送至交换模块。在出口方面，完成的任务有从交换模块接收组装包、帧头封装、输出队列管理和调度、发送至网络接口。具体的系统流程为：媒体设备接收到一个数据包后交入口网络处理器的 MSF 单元处理，MSF 接收单元将数据包拆分为多个固定大小的 mpacket，将 mpacket 写入 RBUF 单元，MSF 通知空闲接收线程表中等待接收数据的线程。接收线程根据接收状态字将 mpacket 重组为数据包，并从 RBUF 单元读到 DRAM 的包缓冲区中。同时，抽取包在缓冲区的描述信息、数据包头一些待处理的重要字段以及包接收上下文中各种信息，填充到此数据包的元数据结构相应位置，将该结构放到 Scratchpad Ring 中排队，等待处理模块处理。处理模块从 Scratchpad Ring 中按照严格的顺序取出待处理包的元数据信息，从 DRAM 中读出数据包中需要处理的内容，通过修改元数据信息去掉链路层的帧头结构，识别数据包并分类，根据分类结果进入不同的处理流程。若是普通数据包则进入转发模块，用最长前缀匹配算法查找路由表，得到下一跳 ID、线卡 ID 和端口 ID 等转发信息，这些信息填入包元数据中。队列管理和包调度模块负责发送队列的入队和出队操作。发送模块从队列管理模块接收到发送消息，将包分割为一个或多个 CFrame，写入 TBUF 单元并通过 MSF 发送单元发送到交换模块。出口网络处理器从交换矩阵获得 CFrame 片断，组装成完整的数据包，进行链路层封装。队列管理模块将包插入相应的队列排队，调度模块将包从队列中取出来，从网络接口发送出去。

3.2 性能需求与资源分配

以线速处理到达的数据包是网络系统最基本、最重要的性能需求，贯穿于数据层面快速数据通道设计的整个过程。为满足线速要求，必须合理划分模块并分配系统的资源，采用多处理器、多线程以及流水线技术等并行机制，充分发挥网络处理器的硬件性能特征。

IXP2400 工作的时钟频率是 600MHz，假定一个 OC-48 (2.5Gbps) 的 POS 端口，其 SONET 帧头开销占 3%，最小 POS 帧大小为 49 字节，则最坏情况下包到达时间间隔为 97 个微引擎时钟周期。这意味着如果网络系统以线速运行，则在流水线的任何阶段每 97 个微引擎处理周期需要流出一个数据包。若流水线某个阶段由单个微引擎完成，则为每个包分配 97 个微引擎处理周期；若同时启动 8 个线程工作，则每个包在微引擎里的延迟至多为 97 * 8 个微引擎处理周期；若流水线某阶段启用 n 个微引擎并行工作，则为每个包分配 97 * n 个微引擎处理周期；若同时启动 8 个线程工作，则每个包在微引擎里的延迟至多为 97 * 8 * n 个微引擎处理周期。从上面的分析可以看出，并行处理和多线程是网络处理器提高网络流的处理能力，解决处理器的处理速度和 I/O 操作访问速度不匹配问题的主要手段。

系统资源的分配主要涉及微引擎分配和分布式存储系统的分配。根据处理任务的自然内聚和耦合程度划分模块，估算各模块需要的处理能力、存取周期时间延迟和指令存储空间大小等。IXP2400 可使用的资源是固定的，包括 8 个微引擎，每个微引擎可以支持 4 个或 8 个硬件线程，有 4k 条指令存储空间，还包括各级数据存储空间和 I/O 资源。基于以上的约束条件为模块分配微引擎和存储空间，确定并行机制和存储策略，确保满足线速处理的性能需求。

3.3 模块间接口

模块间接口有多种通信机制，如状态字的自动推送、基于 Scratchpad 环结构或者基于 SRAM 队列结构、相邻寄存器等，通过预定义的数据结构高效交换模块间的处理信息。典型的通信机制是基于 Scratchpad 环结构或者基于 SRAM 队列结构，二者的基本原理是一样的，区别在于实现的部件不同。基于 Scratchpad 环结构是在 IXP2400 的片内存储器 Scratchpad 内实现的，环大小受 Scratchpad 容量限制；基于 SRAM 队列由 SRAM 控制器配合 SRAM 存储器实现，队列大小只受 SRAM 容量的限制。SRAM 队列通常用于数据包处理阶段，需要大量的队列结构，而 Scratchpad 环通常用于数据包的接收和发送过程，需要少量的环结构。

环或队列数据结构中通常包含数据包在不同任务处理阶段中需要的各种信息，不同模块间接口的数据结构也不完全相同。这些数据结构一般包括数据包相关的元数据信息，如存储数据包的缓冲区信息、包长、在缓冲区中的起始位置以及包开始和结束标志、输入端口、接收状态标记。

4 微引擎编程方法

4.1 微引擎编程模型

通常情况下，在微引擎上运行的应用软件包含多个线程，多个线程运行在多个微引擎上。此类软件需要考虑并行程序设计问题。有两类微引擎编程模型：有序线程模型和无序线程模型。

有序线程模型要求线程在临界区按照应用定义的顺序执行，线程执行顺序由编程人员仔细控制。在此模型中，网络应用被划分成多个不同的阶段，每个线程依次执行一个阶段后把微引擎控制权交给下一个线程，最后一个线程执行完后通知第一个线程，依次循环进行。通过控制线程的处理顺序，有序线程模型带来的优点是在包处理过程中维护数据流中包的次序，同时隐含地完成了互斥处理。但有序线程模型也有潜在的缺点，即当包的处理过程随包类型变化而变化时，所有的线程都将以最慢线程的速度运行。此种模型适合于数据包的类型相近、处理相近、处理时间差别不大而且数据包数量很大的情况。

对于无序线程模型，线程以任意的顺序独立执行，各线程完成当前任务后，获取新任务，继续执行。当各线程处理任务的时间变化较大或者某些操作（如访存、TCAM查找）时间变化较大时，无序线程模型具有较高的效率。另外，当长短包的到达时间变化时，此模型也能很好地工作。无序线程模型的缺点是要执行互斥操作。当各线程在同一个微引擎上运行相同任务时，可以用本地资源实现锁机制，互斥的代价不大。但执行相同任务的各线程在多个微引擎上运行时，必须通过共享资源实现锁机制，增加了同步的代价，性能提高困难。当需要实现有顺序要求的一些任务时，无序线程模型必须通过明确的机制满足有序约束，增加了系统的开销。

4.2 多微引擎并行编程方法

不仅单个微引擎中多个线程涉及并行处理程序设计，网络处理器中多个微引擎协同工作也涉及并行处理程序设计问题。与有序线程模型和无序线程模型相对应，多个微引擎的并行处理机制有两大类：一类是超级任务流水线机制，一类是线程池机制。在超级任务流水线机制中，微引擎采用有序线程模型，而线程池机制中，微引擎采用无序线程模型。

超级任务流水线机制采用流水线方式进行数据包处理，适用于大量数据包的处理时间基本相同的情况。因为流水线的每个阶段都有预定的执行时间，所以超级任务链具有确定的性能。根据任务划分方式的不同，超级任务流水线包括两种配置模式：功能流水线和上下文流水线。

一个典型的功能流水线由一系列任务组成，这些任务运行在多个微引擎。微引擎的每个线程完成对一个数据包的多个处理任务。功能流水线方式要求任务数与微引擎数相同。如任务数大于微引擎数，会出现没有足够的线程接收到达的数据包；微引擎数大于任务数，会出现在同一时间内有两个微引擎处理同一个任务，因此会访问同一段数据区，存在互斥问题。

上下文流水线也是由多个任务组成，但一个微引擎只运行一个任务，通过多个微引擎串联起来完成所需要的处理。上下文流水线通常对状态信息或数据描述符等进行操作，而较少对数据包本身进行操作。利用SRAM环或Scratch环以及Next Neighbor寄存器在微引擎间传递包的状态信息。一个微引擎的处理操作必须在一个数据包到达间隔内完成。

线程池机制是一种直观的并行机制，数据包接收模块完成接收处理后，将数据包分配给线程池中的空闲线程。线程的分配方式有两种，即功能线程分配方式和通用线程分配方式。采用功能线程分配方式，线程池中的每个线程只处理一种类型的数据包，线程按功能分组。数据包接收模块需要先进行分类，决定数据包分配给哪个线程组。这种分配方式可以使得指令空间和处理时间最小。采用通用线程分配方式，线程池中的任何线程都可以处理任何类型的数据包。数据包接收模块不需要进行分类，只是简单地分发给线程池中的空闲线程。因为需要进行数据包类型判断，并适应多种数据包类型的处理，所以线程占用的代码空间大，处理时间长。对于复杂处理，如果微引擎指令存储空间不够时，可采用双线程池方式，每个线程池中的线程只实现一半功能。

一般来说，线程池机制比超级任务流水线更容易实现，并且更具有可扩展性。对于线程池机制，增加微引擎只是增加了更多可用于对数据包处理的线程，软件功能可以不变。而对于超级任务流水线，当增加微引擎以处理更高线速的数据包时，流水线必须重新设计，因为流水线的阶段与数据包的到达速率相关。选择哪种并行机制，与具体的应用相关。如果应用中大量的网络流量的处理流程相同、处理时间变化不大时，超级任务流水线是比较好的并行机制。在这种情景下，业务流的相关性可能较大，因此需要高性能的互斥处理机制，而超级任务流水线采用的基于CAM的互斥机制要比线程池采用的基于SRAM互斥机制速度更快。同时，由于处理时间可以预先确定，相对固定，适合于设计高效率的流水线，合理安排各阶段的处理任务。当应用中包含大量不同类型的业务流，其相应的处理时间有很大的差别时，采用线程池机制比较合适。若采用超级任务流水线机制，流水线的设计要适应最长的时间处理要求，因此大量处理时间短的任务时，在处理结束后，需要等待，影响并行效率。而线程池机制中线程独立运行，线程处理结束后继续处理下一个数据包。

在一个应用中，超级任务流水线机制和线程池机制一般是混合使用的。典型情况下，接口驱动程序接收、发送和队列管理模块采用超级任务流水线机制，微引擎采用有序线程模型。而包处理模块既可以使用超级任务流水线机制，也可以使用线程池机制。当采用有序线程模型时，在临界区必须严格控制线程的顺序，保证在任一时刻只能有一个线程执行。

4.3 并行处理中的几个问题

网络系统并行处理体系结构中一般需要处理两个重要问题：一是互斥问题，一是数据包排序问题。

互斥问题要求在多处理器、多线程的并行执行过程中如何确保共享的数据结构完整性和一致性不被破坏。IXP2400微引擎内部采用分布式缓存机制高效解决互斥问题，将资源的状态和资源本身存放在微引擎内部，利用CAM操作检查资源状态，互斥机制以微引擎的运行速率完成。在超级任务流水线方式中，微引擎间的互斥通过线程间信号实现，同一时间内只有一个微引擎在执行同一任务，可以确保不会出现多个微引擎访问同一个临界区的情况。在线程池方式中，微引擎间的互斥利用SRAM或Scratchpad存储器提供的test_and_set原子测试实现。由于需要轮询存储器中变量，互斥效率不高。

数据包排序问题要求同一个流的数据包按照到达的顺序处理或者处理后按到达的先后顺序发送到下一个处理模块。一方面，乱序会造成网络某些协议性能的下降；另一方面，某些处理功能如AAL5重组和IP头压缩等功能要求数据包不能错序。

超级任务流水线采用阻塞方式处理乱序问题，线程的有序执行潜在地保证了包的顺序，没有复杂的算法，但会因等待而浪费处理时间。下面仅讨论无阻塞方式的包排序问题。在线程池并行机制中采用无阻塞方式处理乱序问题，缓存错序的包，采用较复杂算法排序，但是线程不需要阻塞等待，效率高。包排序方式有两类：端到端排序和局部排序。端到端排序确保一个流的数据包离开系统的顺序与其进入系统的顺序相同，在包处理过程中次序可以被打乱。一般采用AISR（Asynchronous Insert，Synchronous Remove）算法保证数据流的端到端顺序。包进入处理模块前被指派一个流序列号，完成处理后送到排序模块。排序模块采用异步方式将乱序包插入AISR数组中临时存放，同时按照流序列号的顺序从AISR数组中取出数据包，实现排序功能。局部排序要求在中间处理过程中包也需要保持顺序。由于某些需要局部排序的操作并不处理全部包，仅对它处理的部分包排序，实现困难。一般，

（下转封四）

(上接第 273 页)

在需要局部排序的操作阶段设置一个变量记录处理过的最后一个数据包序号;设置一个向量,描述该操作处理的部分包序号。二者配合实现局部包排序。

5 性能评估

进行性能评估需要考虑数据包到达率、IXP2400 可以提供的数据包处理资源和数据包处理任务三个方面。性能评估是考虑在当前资源配置和数据包到达速率情况下,分析数据包处理任务是否能够实现。数据包到达率可以用数据包到达的时间间隔表示。当系统需要支持的接口速率确定时,由于数据包的类型和包长变化很大,要确保不丢包,进行性能评估时需要按最坏情况考虑。确定一种应用对资源的使用需求,需要对应用中的每个任务进行检查,确定设计的系统是否满足性能需求。估算过程要考虑:分析应用中需要存储的状态信息和其它数据结构,确定各级存储器容量和带宽能否满足要求;分析数据包的处理算法,确定需要的指令数量;分析是否有需要互斥的数据,确定对数据包的总体处理时间是否满足数据包到达率的要求;分析存储器带宽、内部总线带宽是否满足应用访存需求;考虑指令周期数是否能够尽量掩藏存储器访问延时。

在网络处理器应用开发中,确保设计与实现的正确性并不困难,而提高系统的性能是开发的主要任务。如何消除系统中的瓶颈,减少数据包处理时间,增大系统的吞吐量,是提高性能的关键。从软件的总体结构上根据应用的特点选择合适的并行处理机制,划分任务并为微引擎指派任务。多线程设计中,区分时延限制的多线程和计算限制的多线程,分别采取措施加以改进。理想状态下,在一个线程等待存储器访问或进行其它延时较大的操作时允许其它线程运行,从而掩藏了存储器延时。对于时延限制的多线程设计,此时存储器时延超过指令执行时间,可以考虑增加线程数来掩藏访存时延。由于存储器延时是当前系统的主要瓶颈,尤其在处理器速度不断增加的情况下,时延限制的多线程比较常见。对于计算限制的多线程,指令的计算时间超过了存储器时延,可以使用更少的线程来平衡系统的性能。在程序编码中采用一些技巧提高系统性能,如连续的硬件访问中插入尽量多的其它无关指令、减少分支指令判断次数、减少冗余指令、利用微引擎特有的指令直接操作数据结构等。

总结 本文以基于网络处理器 IXP2400 实现高速网络应用为例,介绍基于网络处理器系统的软件开发过程和设计方法,探讨开发高性能的微码软件的策略和技术。网络处理器系统软件开发的关键在于面向网络处理器的硬件体系结构编程,充分利用网络处理器为优化数据包处理的各种硬件资源,实现多微引擎、多线程的并行处理机制,解决并行程序设计中遇到的互斥问题和包排序问题。

参 考 文 献

1 谭章熹,林闯,任丰原,等. 网络处理器的分析与研究,软件学报,2003, 14(2): 253~267

2 Comer D. Network Systems Design Using Network Processors. 1st edition. Prentice Hall, 2003

3 Shimonishi H, Murase T. A network processor architecture for flexible QoS control in very high-speed line interfaces. In: Proceedings of the 2001 IEEE Workshop on High Performance Switching and Routing(HPSR 2001). Dallas: IEEE Computer Society Press, 2001. 402~406

4 Intel IXP2400 Network Processor, Product Information, available at: http://www.intel.com/design/network/products/npfamily/ixp2400.htm

5 Intel Corporation. Intel(r)IXP2400 Network Processor Hardware Reference Manual, November 2003

6 Intel Corporation. Intel® Internet Exchange Architecture Software Building Blocks Applications Design Guide, November 2003

7 Intel Corporation. Intel® Internet Architecture Portability Developer's Manual, November 2003

8 Johnson E J, Kunze A R. IXP2400/2800 Programming: The Complete Microengine Coding Guide. Bk&CD Rom edition. Intel Press, 2003

计算机科学

(1974 年 1 月创刊)
第 33 卷第 2 期 (月刊)
2006 年 2 月 25 日出版

国际标准连续出版物号 ISSN 1002-137X
国内统一连续出版物号 CN50-1075/TP

定价:30.00 元 国外定价:5 美元
邮发代号:78-68
发行范围:国内外公开

主管单位:国家科学技术部
主办单位:国家科技部西南信息中心
编辑出版:《计算机科学》杂志社
重庆市渝中区胜利路 132 号 邮政编码:400013
电话:(023) 63500828 E-mail:jsjkx@swic.ac.cn
网址:www.jsjkx.com
社 长:牟炳林
总 编:彭 丹
主 编:朱宗元
主编助理:徐书令
印刷者:重庆科情印务有限公司
总发行处:重庆市邮政局
订购处:全国各地邮政局
国外总发行:中国国际图书贸易总公司(北京 399 信箱)
国外代号:6210-MO