第32卷 第14期 计 算 机 工 程 2006年7月
Vol.32 № 14 Computer Engineering July 2006
·网络与通信· 文章编号：1000—3428(2006)14—0098—03 文献标识码：A 中图分类号：TP 393

网络处理器简介及其微引擎设计

周彩宝
(华东计算技术研究所，上海 200233)

摘 要：网络规模的膨胀型增长、用户对宽带需求的急速增加、各种新业务的层出不穷和智能化管理、应用可升级的技术需求催生了网络处理器，形成了以网络处理器为核心的新一代网络设备体系结构。该文简要介绍了网络处理器的定义、结构及其特点，分析了网络处理器设计与通用处理器的主要不同点，着重阐述了网络处理器的核心部分——微引擎的结构和设计重点。

关键词：网络处理器；微引擎；微码；多线程；线速

Network Processor and Design of Micro-engine

ZHOU Caibao
(East-China Institute of Computer Technology, Shanghai 200233)

【Abstract】The dilatability increase of network scale, the rapidity enhancement need of network bandwidth by users, the emergence in endlessly of vary kinds of new services, the technical need of intellectualized management and upgrading applications speed up the naissance of network processor. A new generation of network equipment architecture based on the kernel of network processor has been formed. This paper introduces simply the definition and functions of NP, the architecture and characteristics of NP, the main differences between NP and general CPU are analyzed, and the architecture of micro-engine which is the main logic block of NP is described. Then some keystones how to design micro-engine are given.

【Key words】Network processor(NP); Micro-engine; Microcode; Multithread; Wire-speed

1 网络处理器简介

1.1 定义及其功能

网络高速发展，对下一代网络设备提出了以下要求[1]：具有优异性能，支持高速分组处理；具有高度灵活性，支持不断变换高层网络服务。传统的基于通用处理器 (General Purpose Processor，GPP)的网络设备只满足灵活性要求；基于ASIC(Application Specific Integrated Circuit)的网络设备只满足高性能要求；网络处理器能够通过灵活的软件体系提供硬件级的处理性能，基于NP的网络设备具有高性能和灵活性。因此，对交换机智能化与全部7层的线速处理的要求导致了网络处理器的产生，可编程的NP给系统提供了极大的灵活性，与此同时还具有数据发送的高速性。

不同的协议(如TCP/IP、ATM)包头信息和格式是完全不同的，为了面向不同的网络协议和网络服务，需要网络处理器具有可编程和可重载功能，以便随技术的发展而进行现场升级。根据国际网络处理器会议的定义：“网络处理器(NP)是一种可编程器件，它特定地应用于通信领域的各种任务，比如包处理、协议分析、路由查找、声音/数据的汇聚、防火墙、QoS等”。其主要的功能包括以下几部分[2]：

(1)协议识别和分类：根据数据包的协议类型、端口号、目的地址以及其它特定于协议的信息对数据包进行识别。

(2)拆装和重组：数据包的拆分、处理以及为转发而重组。

(3)排队和接入控制：识别出数据包之后，将这些数据包送往相应的队列中以进行下一步处理，如优先处理，流量整形等。同时，可根据某些安全接入策略进行数据包过滤，确定是继续转发，还是丢弃。

(4)流量整形和流量工程：某些协议或应用要求对流量进行整形以使之在进入输出线或输出光纤时满足时延和时延抖动的要求。

(5)QoS(Quality of Service)和CoS(Class of Service)：除了对数据包进行流量整形外，还可以将其打上标签送往下一网络节点进行更加有效的处理。

(6)修正数据包：编辑数据包并添加额外的信息。

(7)差错检测：正确检测来自数据链路层的有差错的数据包并能采取有效的处理措施。

虽然传统的 CPU 包含的通用指令集可以处理上面所列的各项任务，但网络处理器可以通过增加指令、优化体系结构、设置硬件加速器等方法，使任务的执行速度更快。

1.2 结构及其特性

网络处理器存在于物理接口器件与交换结构之间。网络处理器硬件结构一般包括多个片内处理器、丰富的 I/O 接口单元、高速多总线单元以及专用协处理单元等(图1)。

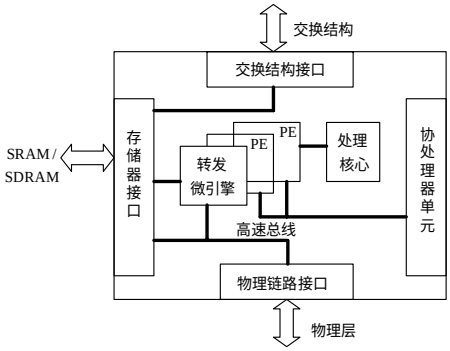

图1 网络处理器结构

(1)片内处理器。网络处理器一般包含多个片内处理器，构成多处理器系统。这些片内处理器可按任务分工大致分为

作者简介：周彩宝(1966—)，女，高工，主研方向：体系结构，系统芯片设计

收稿日期：2005-10-20 **E-mail：**zcb@ecict.com.cn

控制平面的处理核心和数据平面的转发微引擎两种类型。前者用于控制平面的系统维护、管理以及复杂数据处理，一般用通用的 RISC 核实现；后者用于数据平面的快速包处理，提供包分类、调度等服务功能，一般具有专用的精简指令集，这些指令专门针对网络数据包处理而优化设计的，例如数据块传输、状态判断、堆栈操作、哈希查找等，设备制造商可以通过微码编程实现各种协议和业务。转发微引擎一般有几个甚至几十个，每个微引擎又包含多个硬件线程，每个线程都有一套专门的硬件来存放程序运行的上下文，可获得线程切换的零开销。

(2)高速 I/O 接口单元。网络处理器有丰富的 I/O 接口单元，包括物理链路接口、交换结构接口、存储器接口以及与其他外部处理部件的接口等。

(3)内部高速总线。多组处理器和 I/O 接口单元通过内部高速总线(一般为多总线结构)连接在一起，组成优化的数据通路结构，从而提供很强的硬件并行处理能力。

(4)专用协处理器或称专用硬件加速处理单元。采用专用硬件实现高速处理(线速)的通用功能模块，以提高系统性能。如 CRC 效验、哈希查找、字符匹配；针对安全产品，提供加/解密、大数运算等硬件单元。

网络处理器主要有如下几个技术特性：

(1)可编程性。网络处理器的这种特质可以使设备制造商通过修改微码和升级软件以快速满足各种网络通信业务的需求，而不用更改替换任何硬件，从而改变 ASIC 灵活性差的缺点。如通过网络处理器可编程性就可以实现对下一代网络协议 IPv6 技术的支持。

(2)并行处理。网络处理器可实现不同级别的并行处理：通过多级流水线实现指令级的并行，通过硬件多线程实现线程级的并行，通过片内多处理器结构实现处理器级的并行。

(3)高速数据处理。网络处理器自身的硬件结构为达到线速包处理能力提供保障，避免节点设备成为瓶颈。

(4)深层数据处理。也叫智能处理(Intelligent Processing)。根据不同的服务要求，可对分组(帧)进行不同深度的处理。例如路由查找只需处理第 3 层(IP 头部)，分类需要处理到第 4 层(TCP/UDP 协议)，而安全则需要处理到应用层(分组携带的有效载荷)。

(5)模块化设计。网络处理器体系结构的模块化也包含不同的层次：硬件层面和软件层面的模块化。通过模块化设计，力图在保持高性能的基础上获得很好的可扩展性和灵活性，并能使设备厂商容易研发不同性能和不同特性的设备。

(6)可扩展性。网络处理器的可扩展性同样包含硬件可扩展性和服务可扩展性。前者指网络处理器除了可以用来研制小型设备，还可以通过交换机构的连接研制大型设备。后者是指可以在对原有软件结构做很小改动的基础上加入新的服务和功能。

2 微引擎设计

2.1 微引擎结构设计考虑要点

如何满足上述可编程性、高速数据处理等技术特性，微引擎的体系结构制定是设计网络处理器的关键。

我们在制定网络处理器微引擎体系结构时必须考虑很多问题，如：如何设置高效的指令集，微码如何实现，设置多少个微引擎，多个微引擎如何互连，如何调度，如何构建多总线结构，每条总线宽度多少、总线如何定义，如何支持多引擎与存储器、PCI 等的并发操作，如何支持多 NP 扩展，片内、片外存储器大小以及访问方式如何设置、如何管理，如何制定 I/O 接口、存储器接口以满足高速数据块的传输等。这些问题的解决往往是网络处理器的应用定位、性价比等权衡的结果。

虽然控制平面的处理核心和数据平面的转发微引擎都有 RISC 核，但是，控制平面用 RISC 核与数据平面用 RISC 核两者架构和指令的设置却有很大的区别：首先，前者主要负责整个网络处理器的管理和控制，同时负责协议帧的处理和上层应用程序的处理；后者主要实现数据包的线速处理转发，微引擎编程使用一套专为网络数据流处理应用定制的指令集，去掉了通用 RISC 芯片中对协议及包处理用处不大的部分，同时保留了 RISC 指令长度一致、单周期执行时间、易于并行和流水线处理等优点。其次，前者处于慢速通道中，一般采用通用 RISC 核可以满足要求(如 ARM9、PowerPC、MIPS)，后者处于快速通道中，要求能线速转发，所以，网络处理器的微引擎大多采用并行处理与通信机制、多线程切换控制机制、块传输机制、多总线机制、灵活访存机制等，而这些机制是通用 RISC 核天生所不具备的。上述区别是在设计网络处理器微引擎时的重点和难点。

下面是作者在着重研究Intel公司网络处理器IXP1200[3~5]的基础上，给出一个微引擎结构的参考设计，并重点阐述微引擎中多线程管理以及上下文事件仲裁与切换、I/O访问处理的设计要点。

2.2 一个微引擎结构的参考设计

图 2 所示微引擎结构支持硬件多线程，有 4 个程序计数器(μPC)以支持 4 个线程(又称 context 上下文)同时工作，4 个线程共享微控存。每个线程可以执行相同或不同的微码程序，采用内部线程通信机制实现线程同步，提高系统效率。

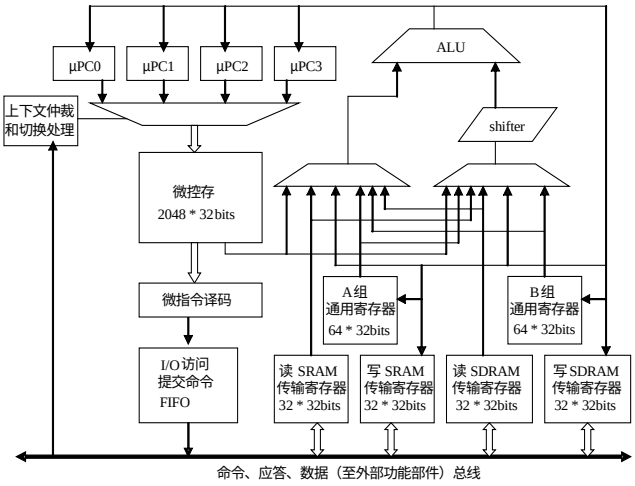

图 2 微引擎内部结构

微指令的设置：算术逻辑运算、移位类；分支及跳转类；I/O 访问类；本地寄存器操作类；杂类。

微码指令采用 5 级流水线执行，执行周期为 1 个时钟周期，即 P0 为指令预取，P1 为指令译码并形成源寄存器地址，P2 为从源寄存器地址读操作数， P3 为执行算术逻辑运算或移位类操作并产生条件码，P4 为写结果到目标寄存器。

设置 2 组通用寄存器 A 和 B 以方便同时取用 2 个源操作数；每组通用寄存器可以被 4 个线程绝对共享，也可以分成 4 小组分别由 4 个线程相对独立使用。相应地，在微指令的编址方式中也设置了绝对地址和线程相对地址两种模式。

设置很大的传输寄存器集，单个指令就可以实现功能单元之间 64B 的数据块移动，和功能单元与总线单元之间 128B 的数据块移动。块数据移动在充分利用微引擎计算资源的同时，还可以减小微码程序规模。

设置多个独立数据总线和控制总线，可以实现数据并发移动：SDRAM 单元和微引擎或总线单元之间的双向同时读

写；SRAM 单元和微引擎或总线单元之间的双向同时读写；SDRAM 单元和 PCI 单元之间读写；总线单元和微引擎之间读写。

2.3 多线程管理以及上下文仲裁与切换

微引擎在硬件上设置了对多线程的支持：有 4 个程序计数器(μPC)以支持 4 个线程同时工作，每个线程有独立的寄存器组以支持快速的线程切换。每个线程总是处于以下 4 个状态之一：

(1)不活动：其控制状态寄存器 CSR 的 CTX_Enable 为 0 禁止其活动；

(2)执行：控制状态寄存器 CSR 的 Active_CTX_Status 为当前执行的线程号，执行状态切换到睡眠状态由软件控制；

(3)准备：准备执行。当执行的线程进入睡眠状态的时候，微引擎线程仲裁器采用循环机制，选择处于准备状态的某个线程进入执行状态；

(4)睡眠：一般情况下，当有 I/O 访问时，控制状态寄存器 CSR 的 CTX_#_Wakeup_Events 将线程切换到睡眠状态。

微引擎中的各个 uPC 可以由控制平面的 CPU 设置，以指明相应线程执行的开始；也可以由程序员用 br=ctx 或 br!=ctx 微指令显式分配。微指令 br=ctx[ctx,label#](ctx 是上下文标识，用 0，1，2 或 3 表示，label# 为转去执行微程序的地址)表示当前 context 号与指令中指定的一致时，转去 label#指定微控存地址处执行。例如：在微控存起始处设置以下 3 条微指令将引起 3 个线程起始地址装入 3 个μPC：

br=ctx [0, thread_0_start#]；

br=ctx [1, thread_1_start#]；

br!=ctx[3, thread_2_start#]。

虽然有 4 个程序计数器(μPC)可以支持 4 个线程同时工作，但是任何时候最多只有一个线程处于执行状态。线程的切换不是可以随意发生的，而是由微程序显式控制的。一种情况是：使用 ctx_arb 上下文仲裁指令；另一种情况是：I/O 访问指令中带 ctx_arb 上下文仲裁选项，并且还要满足一定的条件：(1)该线程允许运行(其控制状态寄存器 CSR 的 CTX_Enable 为 1 允许其活动)；(2)该线程处于准备状态。满足上述条件才允许线程仲裁器发生切换(唤醒某个线程)。

仲裁切换策略一般采用轮询算法以保证每一个线程执行的平等性。

2.4 I/O 访问的处理

数据平面的处理涉及大量的 I/O 访问，所以微引擎设置了 I/O 访问类指令，I/O 访问包括对 DRAM、SRAM、SCRATCH、HASH、PCI、MSF、CAP 等部件的读、写、清除、原子等操作。

通用微处理器通过输出存储器地址直接访问存储器，然后等待数据返回，如果数据不能立即返回，则生成等待状态，如此会浪费计算周期。而微引擎对 I/O 的访问则运用一种被称为“提交”的命令形式，设置“提交”命令 FIFO。即：当微引擎访问 I/O 部件时，输出一个“提交”命令放入“提交”命令 FIFO 中，不必等待请求的任务完成就可以切换并执行其它线程，然而必须知道请求的任务何时完成以便及时得到响应，所以，I/O 部件需要识别出是哪个微引擎的线程(每个微引擎有微引擎号 Engine ID，每个线程有线程号 Thread ID)发出的请求并返回相应线程一个完成的信号，以它来实现“提交”命令输出与任务完成的同步机制。被切换出去的线程则被暂时禁止参与仲裁，直到收到 I/O 部件的完成应答信号。信号还可用于处理器之间和线程之间的同步通信。

每个微引擎自带 2 条目的命令 FIFO，多个微引擎又通过各自的命令 FIFO 共享命令总线访问 I/O 部件，所以命令总线仲裁器需要对各个命令 FIFO 作出仲裁，来决定谁被允许占用命令总线。命令总线仲裁策略：(1)根据命令的类型设置优先级(如链接 SDRAM、SRAM、非链接 SDRAM、FBI/PCI 依次由高到低)；(2)微引擎之间的轮转优先机制；(3)来自 I/O 功能部件的完成应答信号。

3 结束语

网络处理器可以被用于从边缘网络至核心网络的一系列设备，可用于开发从第 2 层到第 7 层的各种网络服务和应用，例如交换、路由、虚拟专用网、多协议标记交换、服务质量、计费、负载均衡、安全和监测以及 3G 通信等，是新一代网络通信设备的关键部件，网络处理器在网络通信设备中的角色相当于 CPU 在 PC 中的角色。本文着重阐述了网络处理器的核心部分——微引擎的结构和设计重点，希望对自主开发网络处理器有所帮助。

参考文献

1 谭章熹, 林　闯, 任丰源等. 网络处理器的分析与研究[J]. 软件学报, 2003, 14(2): 253-265.

2 Comer D E. 网络处理器与网络系统设计[M]. 北京: 机械工业出版社, 2004.

3 IXP1200 Network Processor Datasheet[Z]. http://www.intel.com/, 2001.

4 IXP1200 Network Processor Hardware Reference Manual[Z]. http://www.intel.com/, 2001.

5 Microcode Programmer's Reference Manual[Z]. http://www.intel.com/, 2002.

（上接第 88 页）

参考文献

1 任丰原, 黄海宁, 林　闯. 无线传感器网络[J]. 软件学报, 2003, 14(7): 1282-1290.

2 李建中, 李金宝, 石胜飞. 传感器网络及其数据管理的概念、问题与进展[J]. 软件学报, 2003, 14(10): 1717-1727.

3 Ten Emerging Technologies That Will Change the World[Z]. http://www. techreview.com/arti cles/emerging0203.asp, 2003.

4 Krishnan K R, Neidhardt A L, Erramilli A. Scaling Analysis in Traffic Management of Self-similar Processes[C]. Proceedings of ITC-15, 1997, 2: 1087-1096.

6 Yao Y, Gehrke J. The Cougar Approach to In-network Query Processing in Sensor Networks[J]. SIGMOD Record, 2002, 31(3): 918.

6 Elson J. Time Synchronization Services for Wireless Sensor Networks [C]. Proceedings of the 15th International Parallel & Distributed Processing Symposium, Los Alamitos, 2001.

7 Madden S R, Szewczyk R, Franklin M J, et al. Supporting Aggregate Queries over Ad-hoc Wireless Sensor Networks[C]. Proceedings of the Workshop on Mobile Computing and Systems Applications, Los Alamitos, 2002: 49-58.

8 Sinha P, Sivakumar R, Bharghavan V. MCEDAR: Multicast Core Extraction Distributed Ad-Hoc Routing[C]. Proc. of the Wireless Communications and Networking Conference, New Orlean, LA, 1999-09.