第 33 卷 第 5 期 计 算 机 工 程 2007 年 3 月
Vol.33 No.5 Computer Engineering March 2007
·网络与通信· 文章编号：1000—3428(2007)05—0106—03 文献标识码：A 中图分类号：TP393.05

网络处理器中最长匹配算法的优化

陈 静，吴 非，黄 祚
(华中科技大学计算机科学与技术学院信息存储国家专业实验室暨教育部重点实验室，武汉 430074)

摘 要：传统的最长匹配路由算法都是基于双线程并行查找方法来实现的，没有充分利用新一代网络处理器无延时线程切换和可编程的特点。该文基于 IXP2350 平台对传统的最长匹配算法进行了优化改进，充分利用硬件特性和微引擎中异步内存读写的特点，用单线程来完成整个路由的查找。实验测试结果表明，这种优化改进使路由器的包转发效率提高了 20%。

关键词：网络处理器；最长匹配算法；嵌入式系统；微引擎；微码

Improvement of Longest Prefix Match Algorithm in Network Processor

CHEN Jing, WU Fei, HUANG Zuo
(Peripheral Memory System Laboratory of the Ministry of Education, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

【Abstract】The realization of longest prefix match algorithm traditionally uses double threads to lookup the route information, which not fully utilize the network processor's no-delay thread switching and programmable characters. This article discusses how to fully utilize the hardware characteristics and asynchronous read/write of memory method in the micro engine on the IXP2350 platform to realize the whole route lookup with only one thread. The test result also shows that a performance improves 20% than double threads realization on the micro engine.

【Key words】Network processor; Longest prefix match algorithm; Embedded system; Micro engine; Microcode

最长匹配算法(Longest Prefix Match, LPM)作为一种比较通用的路由查找算法在网络路由中得到广泛的应用，但是其思想基本上都是对给定的 IP 地址进行分段后，采用多线程路由查找方法和并行处理技术，提高路由的查找速度。

随着硬件技术的提高，针对网络中日益增多的应用提出的高带宽的需求，专门应用于网络处理的专用芯片也随之推出。这种专用的处理器一方面具有很强的网络处理性能，能够高速处理网络中的转发包；另一方面，为了满足不断变化的网络应用的需求，网络处理器具有很强的灵活性。

如果按照传统的设计思想在网络处理器中实现 LPM 算法，不仅查找的效率低下，而且其实现也比较复杂，这导致在整个包转发的处理过程中，该算法的执行成为处理的瓶颈，制约整个系统性能的发挥。因此，有必要对最长匹配算法进行优化和改进，使其更符合网络处理器的系统体系架构，用最小的资源发挥最大的性能。

1 网络处理器 IXP2350 中微引擎的体系结构

IXP2350 网络处理器采用的是 Intel 的 IXA(Intel Exchange Architecture)体系结构，主要应用在接入和边缘网络。IXP2350 之所以能有强大的网络处理能力，一方面通过微引擎来满足系统性能的提升，另一方面通过芯片内部结构的优化，包括扩展 SRAM、SDARM 通道，从而增加内存的吞吐率和并行性。正是由于整个体系结构有针对性的优化设计，才有针对网络应用的整体性能的提高。

1.1 微引擎的内部结构

IXP2350 具有 4 个最高可运行于 900MHz 的微引擎，每个微引擎具有并行运行 4 个或者 8 个线程的能力，每个线程都有独立的一套寄存器组、PC 计数器和本地内存(Local Memory)。由于进行线程切换时不需要保存各个寄存器的内容，因此进行切换操作只需要 2~3 个微引擎时钟周期，相对于现在通用 CPU 动辄几百个时钟周期的切换时间而言，微引擎中进行线程切换的时间几乎是可以忽略不计的。

微引擎的内部结构如图 1。控制存储区(Control Store)是微引擎的程序存放的区域，属于微引擎内部资源，可以单时钟周期对其中的指令和数据进行存取，这样，通过把程序代码内置在微引擎中可以减小指令访问的时间，从而提高微引擎的执行效率；执行数据通道(Execution Data Path)是微引擎的指令执行单元，该部件采用 5 级流水线：取指令，指令译码，取操作数，指令运算和写运算结果。每条指令的执行时间都只需要一个时钟周期，通过流水线技术，使 5 条指令能被并行处理，这也提高了微引擎执行的效率。

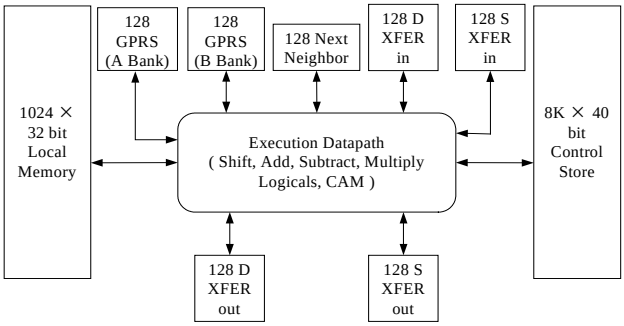


图 1 微引擎内部结构

作者简介：陈 静(1979－)，男，硕士生，主研方向：信息存储与处理和嵌入式系统；吴 非，博士、讲师；黄 祚，硕士生

收稿日期：2006-03-06 **E-mail：**cj_603@163.com

1.2 微引擎中的异步内存读写

在通用处理器中，一般对内存进行读写都是同步完成的，即处理器等待内存返回结果后继续执行。虽然由于 Cache 的利用，可以大大减小这种访问的延时，但是，当出现没有命中的情况时，系统就不得不停下来等待内存返回结果。另外，当发出一个读写请求之后，无法预测结果会在多长时间之后返回。

在 IXP2350 中，微引擎不是通过 Cache 策略来减小访问内存的时延，而是通过异步实现的方式，即发出一个对内存的读写请求放入传输寄存器，然后进行线程切换，让微引擎去完成其它线程的任务。虽然现在内存的速度提高很快，但是相对于处理器的速度，一次存取的时间还是需要处理器等待几十个甚至上百个的指令周期，表 1 列出了 IXP2350 中不同存储部件的平均访问时间，可以看出，即使对 SRAM 的访问，微引擎等待结果返回也还是需要等待大约 100 个指令周期。异步策略有效利用了这个延时，不仅让处理器的利用效率得以提高，而且也让其它处于就绪状态的线程有机会得以运行。

表 1 不同部件的平均访问时间

访问部件类型	访问需要的指令周期数
Local Memory	3
Scratchpad	60
QDR SRAM	100
DDR SDRAM	400

微引擎的异步读写方式是通过其内部的传输寄存器实现的，微引擎中的传输寄存器(Transfer Register)是用于微引擎与其它功能单元进行数据交换的窗口，可以分为 Xfer in reg 和 Xfer out reg 两大类。

进行读操作时，微引擎发出一个读请求并将该内存地址放入某个 Xfer in reg，然后设置信号，进行线程切换，当读取结果返回后，该线程转为就绪状态，当其获得微引擎的控制权后就可以继续执行；进行写操作时，数据从通用寄存器或者本地寄存器写到 Xfer out reg，当该微引擎等到获得总线控制权之后，Xfer out reg 的内容会自动写入相应的功能单元，这样微引擎把数据放到 Xfer out reg 后就可以继续执行下去，而不用等待数据的传输完成。

2 最长匹配算法

LPM 算法通过一个给定的 IP 地址作为关键字，在路由表中进行查找，获得最匹配该 IP 地址的转发端口信息，包含该 IP 地址的网络包就通过这个查找的端口转发出去。例如，如果获得 IP 地址 211.69.5.3 的两个转发端口的信息：211.69.5.*:5(即从 5 端口转发出去)，211.69.*.*:4(即从 4 端口转发出去)，则采用 LPM 算法就应该采取前者，从 5 端口转发出去。

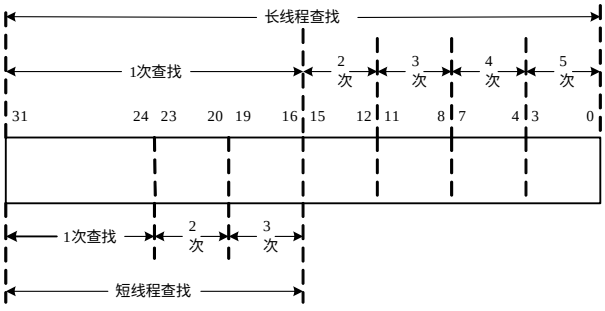


图 2 LPM 对 IP 地址进行分段

为了减小路由表的规模，算法对给定的 IP 地址(在此只针对 IPv4 进行讨论)进行分段，如图 2 所示。通过分段，相应路由表也分成 3 个表：64k 表，256 表和 trie 表。搜索路由表时，将其中某个 IP 地址位段作为偏移地址在路由表中进行查找，从而最终获得下一跳(NextHop)的信息。

3 LPM 算法中路由查找的过程

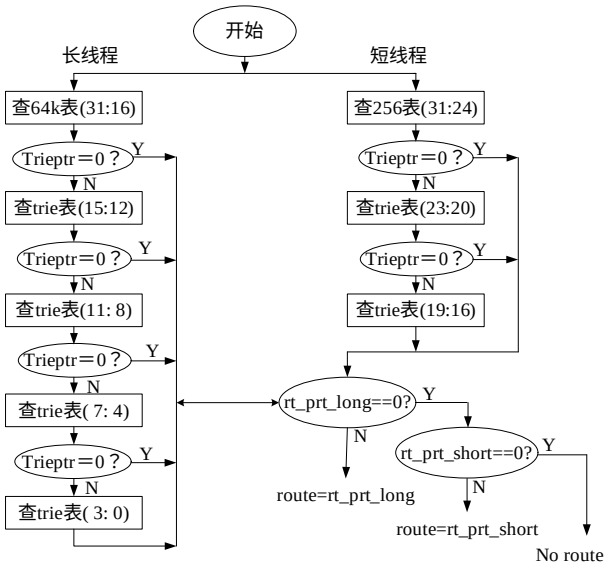
从效率出发，LPM 算法通过两个线程来同时进行下一跳偏移地址的查找，分别称之为长线程和短线程。

长线程首先提取 IPv4 的[31::16]这高 16 位作为偏移地址，在 64k 表中查找下一跳信息。如果找到的相应表项无效则停止查找，线程结束 如果有效，则它会根据 IP 地址的[15::0]这低 16 位，每次提取 4 位(如图 2)在 trie 表中步进查找，最后保存最长匹配的结果。因此，长线程最多进行 5 次路由表的查找。

另外一个线程可以称之为短线程，其查找过程和长线程类似，所不同的是，它只将[31::16]位作为关键字进行查找。第 1 次以[31::24]这个位段作为偏移地址在 256 表中查找，如果有效，则它会根据 IP 地址的[23::16]这 8 位，每次提取 4 位在 trie 表中步进查找，最后保存结果。因此，短线程最多进行 3 次路由表的查找。

之所以采用双线程进行路由表的查找，是因为有可能长线程并不能够一定找到结果，在长线程失去匹配的情况下，就需要回溯，重新从[31::16]进行查找。如果采用单线程，最坏情况下，长线程访问 5 次内存，短线程访问 3 次，共需要 8 次访问内存的时间，而采用双线程查找只需要 5 次，后者的效率是前者的 1.6 倍。

整个最长匹配算法的查找过程如图 3 所示。



Trieptr是指表项中trie表偏移地址一项的指针
rt_ptr_long是指在长线程查找的结果 （即指向NextHop的偏移地址）
rt_ptr_long是指在短线程查找的结果 （即指向NextHop的偏移地址）

图 3 LPM 算法查找过程

4 LPM 在微引擎中的优化实现

在微引擎中实现 LPM 算法时，如果采用两个线程同时进行查找的方法，不仅浪费资源(在微引擎中多使用一个线程)，并且为了最终能获得“最匹配”的结果，还要对两个线程查找的结果进行同步，增加了算法的复杂性。

我们可以利用网络处理器无延时线程切换和异步内存读写的特点，以及微引擎中微码特殊的编程方式，只用一个线

程完成 LPM 算法，即将长查找和短查找两个线程在微引擎中用一个线程实现。

另外，为了提高查找的效率，还可以用多个线程同时进行 LPM 算法的查找，通过这种并行性让网络处理器可以在同一时间进行多个 IP 地址路由的查找。

4.1 用单线程实现 LPM 算法

单线程实现 LPM 算法的流程如下：

(1)从 IP 地址中提取长线程需要的[31::16]位作为 64k 表的偏移地址，加上该表的基地址形成一个内存请求，将该请求放入 Xfer in reg0；提取短线程需要的[31:24]位作为 256 表的偏移地址，加上 256 表的基地址形成一个内存请求，然后将该请求放入 Xfer in reg1。

(2)设置信号(等待两次 SRAM 读的结果都返回)，进行硬件级的线程切换，把控制权交给该微引擎上的另一个线程。

(3)两个读内存的结果返回，该线程重新进入运行状态。

(4)判断 Xfer in reg1 的查找结果：

1)Trie 表地址为空，短线程查找过程结束，进行长线程单线程路由的查找。

2)Trie 表地址不为空，需要短线程继续查找，转入(5)。

(5)判断 Xfer in reg0 的查找结果：

1)Trie 表地址为空，长线程查找过程结束，进行短线程单线程路由的查找。

2)Trie 表地址不为空，需要长线程继续查找。

如果进入(5)中的 2)情况，表明长短线程都需要进一步检索路由表，因此，这种情况下又转入(1)，有所不同的是，提取的 IP 地址的位段不一样。

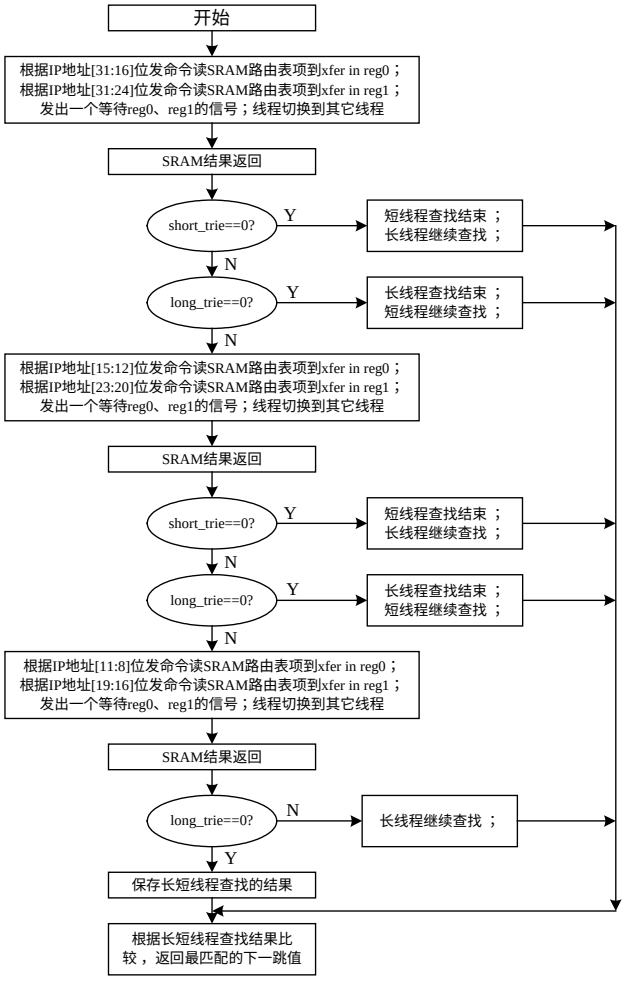


图 4 改进算法查找流程

由于短线程最多需要 3 次访问路由表以获得最终的结果，因此，在长短两个线程的查找结果都不为空的情况下，最多需要 3 次进行(1)～(5)这样的流程。如果在这个过程中，短线程或者长线程查找结束，该线程就转入单线程查找执行路径。

当长短两个线程的查找结果都查找完毕后，就根据查找的结果进行比对，将最匹配的下一跳结果作为值返回。整个查找过程可参考图 4。

可以看出，改进的 LPM 算法并没有增加代码复杂度或者浪费微引擎的执行时间，整个过程算法都是利用了无延时线程切换和内存异步读写的特性，不仅如此，在微引擎中用一个线程来完成比用两个线程的优势还在于：两个线程同时进行查找要涉及同步的问题，而用一个线程则根本不需要考虑，减小了实现的复杂度。

4.2 多线程并发查找路由表

对于担负路由功能的设备来讲，一段时间内，可能有多个网络包的到来或者转出。因此，需要考虑系统的并行处理能力。

在对转发包进行流水线处理的过程中，LPM 算法查找路由操作是一个非常耗时的工位，因为该操作需要多次访问内存以获得转发端口的信息。单线程处理这个工位会使其成为系统的瓶颈，制约系统的性能。

基于对系统并行性的需求和路由查找耗时的特性，我们需要用多个线程同时处理该事务。不过，用多个线程进行路由查找时，需要考虑的是在不同的微引擎上用多个线程来完成，还是在一个微引擎上面用多个线程来完成的问题。

在同一个微引擎中采用多个线程(4 个或者 8 个)来进行路由查找，它的优点是不仅可以共享保存在控制存储区的代码，而且有些资源也可以共享，比如，LPM 的基地址(一般来讲，一旦微码运行起来，这个值就不会再修改)。但是，这样也存在着一些问题，如果其中一个查找线程运行时，其它查找线程会受到阻塞，不能实现该工位的并行性；另外，如果共享的资源被修改，其它线程也会受到影响。

因此，我们在不同的微引擎上面采用多个线程的方法。这样，即使一个线程等待或崩溃了，其它线程也能正常地工作。对于其它采用多线程进行处理的工位也需要采用这样的方法，这样才能提高系统的健壮性和并行性。

5 测试

为评估改进算法的性能，我们在 IXP2350 开发版上采用传统算法和改进算法两种方法来实现路由查找，并对其进行多个指标的测试，结果如图 5 所示。其中，吞吐量和丢包率的单位是百分比，后 3 个指标的单位是 μs。

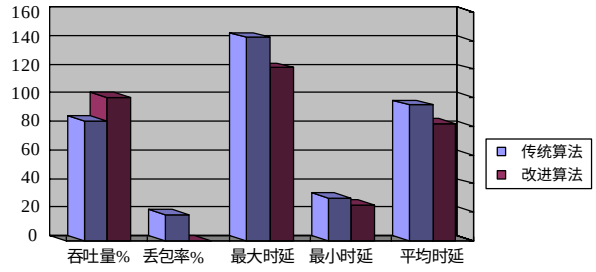


图 5 传统算法和改进算法的性能对比

可以看出，改进算法吞吐量和丢包率的指标已经达到线速转发，而传统算法分别只能达到 83%和 18%；后 3 项时延

（下转第 111 页）