

一种基于 Trie 的流水式 IP 查找结构

张建伟, 滕 飞, 马万里, 陈晓明

(大连理工大学 电子科学与技术学院, 辽宁 大连 116024)

摘 要: 为了减小基于 Trie 的流水式 IP 查找引擎的内存占用和延时, 基于优化线性流水线(OLP), 利用其引入空操作(Nop)能够更灵活布局的特点, 提出了 EOLP(Enhanced Optimized Linear Pipeline)方案, 包括前缀扩展长度确立算法, 流水线长度确立方案, 布局算法和编码算法. 实验结果表明, 相对于 OLP 方案, EOLP 方案的内存占用和流水线延时至少减少了 1/6, 能够完全布局 Trie 节点, 并达到了 72 Gb/s 的吞吐率, 满足高速网络的需求.

关键词: IP 地址查找; 最长前缀匹配; FPGA; Trie; 流水线

中图分类号: TN79+1

文献标识码: A

文章编号: 1000-7180(2017)07-0070-05

A Pipelined IP Lookup Structure Based on Trie

ZHANG Jian-wei, TENG Fei, MA Wan-li, CHEN Xiao-ming

(School of Electronic Science and Technology, Dalian University of Technology, Dalian 116024, China)

Abstract: In order to reduce the memory usage and delay of pipelined IP lookup structure based on Trie, this paper presents a new solution called EOLP(Enhanced Optimized Linear Pipeline), including prefix expansion algorithm, pipeline conforming algorithm, mapping algorithm and encoding algorithm, based on OLP that implements the NOP in the pipeline. Experimental results shows the memory usage and delay can be decreased by 1/6 and all the nodes can be mapped on. It achieves a throughput of 72Gbs, meeting the requirement of high-speed networks.

Key words: IP address lookup; longest prefix matching; field programmable gate array; Trie, pipeline

1 引言

信息技术的发展, 网络速度和网络应用种类不断增加, 给 IP 查找和数据转发引擎带来了新的挑战. 传统的基于软件的查找和数据转发已经远远不能满足要求. 硬件搜索无疑是解决这一问题的最好选择. 目前, 几乎所有的高端查找引擎都是基于 CAM 芯片的设计^[1], 但有功耗高、速度不够快的缺点, 另一方面, 算法和硬件结合的实现方式性能表现突出, 常用的是基于 Trie 和 Hash 的实现方式. 基于基本的 Trie 结构, 人们提出了各种各样的 Trie 的变形结构^[2-3], 但无论是怎么样的一种 Trie 结构, 最终的查找结构都面临一系列共同的问题: 吞吐量是否够大, 内存(本文指电路的硬件存储空间, 如 SRAM)的占用是否够低, 输出延时

是否固定, 输出是否保序, 是否支持路由更新等. 本文试图找到一种合适的方案, 能够很好地处理这些问题.

为了提高吞吐量, 引入了流水线的设计方式. 为了提高流水线的性能, 需要解决流水线级间的负载均衡问题.

针对流水线负载均衡问题, 本文基于 OLP, 对 OLP 进行了讨论和改进, 并在 FPGA 上做了实现和对比, 实现了 72 Gb/s 的吞吐率.

2 基于 Trie 的搜索引擎研究现状

基于 Trie 的研究特别多, 但很多研究都处于软件仿真阶段, 并不支持最终的硬件查找. 有些支持硬件查找的算法或过于复杂.

近年来, 为了推进算法的硬件实现快速走向商

业化,很多相关研究都逐渐由仿真转向 FPGA 实现,.性能比较突出的有文献[4]提出的环形流水线(Ring Pipeline)和文献[5]提出的优化的线性流水线(OLP)等.在 OLP 方案中,为了使节点的布局更加灵活,引入了 Nop 操作.该操作使得子节点不用紧随其后布局,只需保证子节点在父节点之后布局即可,这样使得流水线的任务分配更容易均衡.这种思路也将为本文所采用.但 OLP 方案也存在一些缺点.

OLP 将流水线的级别数作为输入变量,然后再确定初始步长(initial stride),进而进行后续的人列和布局操作.事实上,无法预先确定流水线究竟需要多少级,纵然事先通过实验确定流水线的级数,也可能因为路由表的选取而导致不同的结果.因此,OLP 的这种处理方式可能导致流水线无法容纳完所有的节点或者引入过多的流水线级数的问题.

综上所述,虽然 OLP 方案硬件电路简单,任务分配均匀,具有输出延时固定,输出保序,支持路由更新等特点,但它存在节点可能无法布局完全,流水线级数太长(导致输出延时虽然固定但太长)等缺点.本文基于 OLP 方案提出了一种改进的方案——EOLP(Enhanced Optimized Linear Pipeline).

3 EOLP 方案

通常一个树的节点包含两个信息,自身代表的前缀和指向子节点的指针逐而信息的查找是从头节点开始,通过指针层次地往下查找,但这存在一个问题——最长前缀匹配问题.为了实现最长前缀匹配,往往需要回溯操作.而文献[6]利用 Leaf-Pushing 解决了这一问题,即给只有一个叶节点的节点添加一个新的叶节点,新的叶节点的信息使用距离其最近的前缀,而中间节点只保存指针信息.因此,每个节点只有一个域,或保存前缀信息,或保存索引指针.

3.1 确定前缀扩展长度

如文献[7]所示,路由前缀长度通常都是大于 8 的,且长度处于在 8~18 之间的和 24~32 之间的特别少,而在 20 和 24 之间的特别多.通常情况下,如果用一个较大的数进行前缀扩展,会导致某些前缀不断被复制,比如前缀 001*,本文用 5 bit 进行前缀扩展,将获得 00100*,00101*,00110*,00111* 四个前缀,那么这会导致结点数目的剧增;但用较大步长进行子树构建的时候,可以省去一些

中间结点,又有减少节点的趋势.如图 1 所示,本文对 rrc00,rrc01,rrc08,rrc11 等四个路由表进行了前缀扩展并统计,结果是用小于 18 bit 的步长进行前缀扩展和子树的构建,获得的树的总的节点都比较少,因此,本文将在 8 到 18 之间寻找前缀扩展的长度.为了描述前缀扩展长度的确立算法,这里将相应的符号介绍如下.

I :前缀扩展的初始步长.

T :原始的 Trie.

T :前缀扩展后的 Trie.

$N(T)$:Trie T 的节点数.

算法 1 确定前缀扩展长度

```
input:  $T$ 
output:  $I$ 
1:  $I \leftarrow 8$ ;
2: while( $I \in [8, 18]$ ) do
4:   if( $N(T) < 2^I \times (32 - D)$ ) then
       return  $I$ ;
5:   end
6:    $I \leftarrow I + 1$ ;
7: end while
```

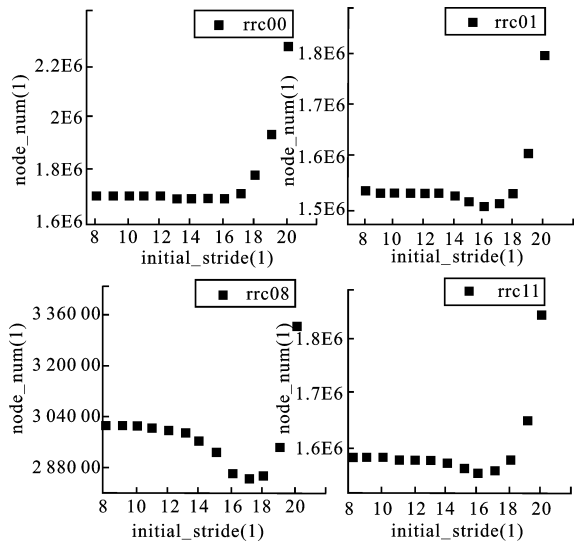


图 1 前缀扩展后的节点数

3.2 布局结点

和 OLP 一样,为了使节点布局得更均匀,本文也引入了 Nop 操作,同时也遵守 OLP 中所提的约束:如果结点 A 是结点 B 的祖先,那么 A 必须布局到结点 B 之前的流水线级别上.为了使节点布局完全,流水线级别数最短,本文的流水线的级别数将在这一阶段确定.为了描述清楚本文的算法,除了前面用到的记号外,其他记号和说明介绍如下.

P :流水线级别数;

M_p :第 P 级流水线布局的节点数;

M_a : 已经布局的节点数;

ST_i : 第 i 个森林, $i = 0, 1, \dots, N$;

节点高度: 树的深度减去节点所在层数.

F_{ather} : 森林第一层结点的列表;

S_{on} : 森林第一层结点的孩子结点(即第二层结点)的列表;

前缀扩展完成后, 形成了 2^l 个子树, 每一个子树在布局过程中将逐渐失去自己的头节点, 也就变成了更多的子树, 即森林. 具体的布局伪代码如算法 2 所示.

算法 2 节点映射

```
input:  $T'$ 
output:  $P$ , filled pipeline
1:  $P \leftarrow 1, M_P \leftarrow 0, M_a \leftarrow 0$ ;
2:  $\text{getN}(T)$ ;
3: fill the roots of the subtrees into  $F_{\text{ather}}$ ;
4: while( $M_a < N(T)$ ) do
5:   sort the nodes in  $F_{\text{ather}}$  in the decrease order of the height in subtree;
6:   while( $M_P \leq 2^l$  and  $F_{\text{ather}} \neq \emptyset$ ) do
7:     get the front node of  $F_{\text{ather}}$  and merge it into stage  $P$ ;
8:     if (the front node has son) then
9:       push the front node's son into  $S_{\text{on}}$ ;
10:    end if
11:     $F_{\text{ather}}$  pop;
12:  end
13:  $M_a \leftarrow M_a + M_P, M_P \leftarrow 0$ ;
14: merge  $S_{\text{on}}$  into  $F_{\text{ather}}$ ;
15:  $P \leftarrow P + 1$ ;
16: end
```

3.3 编码

利用节点布局算法能将 Trie 的结点均匀布局到各个流水线级别上, 但原有的 Trie 的连接信息已经不存在了, 即通过一个节点再也找不到它的子节点. 为了恢复原有的指针信息, 需要对现有的节点进行重新串联, 使得它能像原有 Trie 一样, 通过父节点, 可以索引到相应的子结点, 这样才能使得搜索正常进行. 本文将这个过程称之为编码. 相应的编码伪代码如算法 3 所示. 编码成功后, 树的链接信息就被重新构建了出来.

算法 3 节点编码

```
input:  $P$ , filled pipeline
output: encoded pipeline
1: for  $i = 1$  to  $p$  do
2:   while( this stage has node to be encoded) do
3:     if (next_hop) then
```

```
4:       print{offset_addr1, not_en, offset_addr2}
5:   else
6:     find the node's lchild in the following stage;
7:     print{lchild's stage-i, en, lchild's offset_addr}
8:     find the node's rchild in the following stage;
9:     print{rchild's stage-i, en, rchild's offset_addr}
10:   end if
11: end while
12: end for
```

图 2 是编码后的流水线结构图, 其中第 3 级流水到第 19 级流水的结构和第二级流水的结构一致. 第一级流水线地址由 IP 的前 $i+1$ 位(i 是前文所述的前缀扩展长度)组成, 而地址中的内容或者是 next_hop(下一跳)的地址, 或者是子结点所在流水线级别上的信息, 用于进一步的查找. 图中每一级流水线的 SRAM 大小都为 $32 \text{ k} \times 20 \text{ bit}$ (共 20 级流水, 整个流水线可支持 50 k 路由表项查找), 字宽 20 bit 由三个域(域 1, 域 2, 域 3)组成. 其中第二个域为使能位, 如果使能位为 0, 表示不需要使能下一级查找, 即本次的查找结果为 next_hop 的地址信息, 只需将该信息逐级往后寄存, 直至由最后一级流水输出. 如果使能位为 1, 表示需要继续使能查找, 进而 ram 控制器进行判断, 如果域 1 的值为 1, 则 ram_en 使能, 通过访问 SRAM 获得下一级的操作数据. 如果域 1 的值大于 1, 表示其孩子节点在后面的流水线中, ram_en 不使能, 将数据域 1 的数值减 1 后, 再把整个数据往后传递, 进行后续的查找操作.

3.4 更新

由于本文的布局思路和 OLP 是一致的, 因此本文也可以采用文献[8]中的插入 write bubble 的方式进行更新. Write bubble 通过流水线逐级往下进行更新, 该更新内容需要片外提前进行计算打包. 每一级流水级别的前面都有一个更新表(write bubble table), 而每一个 write bubble 都被分配一个特征识别码, 当 write bubble 通过流水线各个级别的时候, 将检测自身的 ID 和更新表是否匹配, 如果匹配, 将检索更新表中的地址和内容进行更新. 由于一条前缀的搜索路径分散于流水线的各个级别, 因此对一条前缀信息进行更新的时候可能连续插入多个 write bubble.

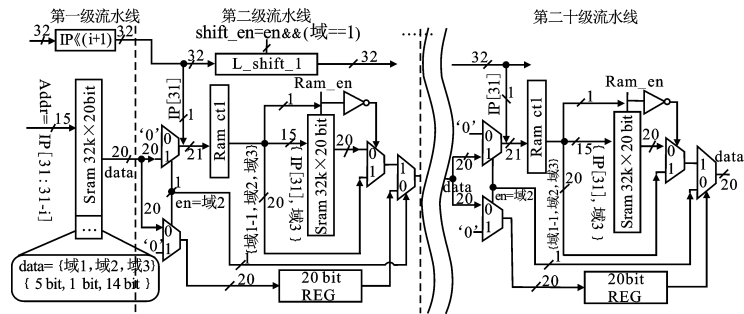


图 2 编码后的流水线部分结构图

4 实验结果与分析

为了验证所提出的 EOLP 方案的正确性和分析其性能,本文将该方案在 FPGA 上进行了实现.并将部分参数介绍如下.

吞吐量:网络最小数据包大小为 40 byte,因此对应的吞吐量为 $f \times 40 \times 8$, f 为芯片工作频率.

内存:占用硬件电路的存储空间,本文指占用 FPGA 的 SRAM 的空间.

LUT&FF:占用 FPGA 的 LUT 硬件资源和 FF 硬件资源.

延时:指输入到输出的延时,跟时钟频率和流水线级别有关, $T_{\text{delay}} = 1/f \times \text{stage}$.

总节点数:整个 Trie 的节点数目.

未布局完的节点数:整个 Trie 的节点数目减去布局到硬件流水线中的节点数.

由于优化的线性流水线(OLP)性能比环形流水线(Ring Pipeline)和单向自适应环形流水线(CAMP)好,本文主要讨论与 OLP 方案的对比. OLP 方案的前缀来自于路由表 rrc00, rrc01, rrc08 和 rrc11^[7],并且验证了 30 k 条前缀.但目前的核心路由器通常有 50 k 条前缀,本文将从以上路由表获取 50 k 的前缀进行试验.由于路由表前缀数目和使用的开发板有所差异,为了对比公平,本文将 OLP 以 50 k 的前缀在本文所用开发板(Xilinx VC707)上重新实现.最终 OLP 方案和 EOLP 方案运行速度均为 225 MHz.由于本文路由前缀为 50 k 条,远多于 OLP 中的 30 k 条,所以用的 SRAM 较大,访问延时较长,时钟频率略低于 OLP 的 250 MHz(30 k 前缀).

实验结果如表 1~4 所示. EOLP 和 OLP 都实现了 72 Gb/s 的吞吐率.但从布局节点完整性来看, OLP 方案针对 rrc00、rrc01 和 rrc11 都有超过 1.7 k 的节点没有布局完,针对 rrc08 有超过 2.4 k 的节点

没有布局完;而 EOLP 方案针将四个路由表所有节点都布局完了,因此 EOLP 布局节点更为完整.从 SRAM 内存占用来看,针对路由表 rrc00、rrc01、rrc8 和 rrc11, OLP 方案占用 SRAM 15.36 Mb;而 EOLP 方案针对路由表 rrc00、rrc01、rrc11 占用 SRAM 12.8 Mb,针对路由表 rrc08 仅占用 SRAM 10.24 Mb,因此 EOLP 大大节约了内存.从延时来看,针对这四个路由表, OLP 的延时(通过 IP 查找引擎的时间,等于流水线级别数乘以时钟周期)都为 106.7 ns;而 EOLP 针对路由表 rrc00、rrc01、rrc11

表 1 基于 rrc00 的 OLP 和 EOLP 的实验结果

方案	OLP	EOLP
吞吐量/(Gb/s)	72	72
内存/Mb	15.36	12.8
LUT&FF	10 949&~945	9 128&~793
流水线级别数	24	20
延时/ns	106.7	88.9
总的节点数	275 427	275 427
未布局完节点数的	1 729	0

表 2 基于 rrc01 的 OLP 和 EOLP 的实验结果

方案	OLP	EOLP
吞吐量/(Gb/s)	72	72
内存/Mb	15.36	12.8
LUT&FF	10 949&~945	9 128&~793
流水线级别数	24	20
延时/ns	106.7	88.9
总的节点数	272 660	272 660
未布局完节点数的	1 758	0

表 3 基于 rrc08 的 OLP 和 EOLP 的实验结果

方案	OLP	EOLP
吞吐量/(Gb/s)	72	72
内存/Mb	15.36	10.24
LUT&FF	10 949&~945	7 241&~596
流水线级别数	24	16
延时/ns	106.7	71.1
总的节点数	196 156	196 156
未布局完节点数的	2 464	0

表 4 基于 rrc11 的 OLP 和 EOLP 的实验结果

方案	OLP	EOLP
吞吐量/(Gb/s)	72	72
内存/Mb	15.36	12.8
LUT&FF	10 949&945	9 128&793
流水线级数	24	20
延时/ns	106.7	88.9
总的节点数	274 709	274 709
未布局完节点数的	1 740	0

的延时为 88.9 ns,针对路由表 rrc08 的延时为 71.1 ns,因此 EOLP 大大减少了延时. 另外, rrc08 在 2004 年以后便没有更新,而 rrc00, rrc01 和 rrc11 一直都在更新. 从表中可以看出, rrc08 的节点大约 19.6 万个,而另外三个路由前缀表的节点已经超过了 27 万个,由于 rrc8 的总结节点数少,相应的流水线级数为 16,而较多节点的 rrc00, rrc01 和 rrc11 对应的流水线级数 20. 相应的, OLP 方案针对这四个路由表的流水线级数都是 24,从这一点看,本文提出的方案更具有灵活性. 同时,从对 FPGA 的 LUT 和 DFF 资源占用来看, EOLP 方案也有较大优势. 因此, EOLP 在内存占用、流水线延时、布局节点完整性和电路规模上较 OLP 都有较大的提升.

值得注意的是,本文是针对 FPGA 进行的讨论,因此讨论了路由表 rrc08 的特殊情况. 如果要进行 ASIC 实现,为了使流片后的芯片适用于所有路由表,应当选取 20 级流水线进行实现,在这种情况下,本文提出的方案比 OLP 方案仍节省了 2.56 MB 内存,延时减少了 17.8 ns.

5 结束语

算法与硬件结合的 IP 查找结构被认为是下一代网络的最佳选择. 本文以 OLP 方案为基础,提出了改进的 EOLP 方案,并在 FPGA 上做了实现,保证了电路的可实现性. 在吞吐量不变的情况下,实现了输出保序,占用内存小,延时短,节点布局均匀等优点. 与对比方案 OLP 相比,本文提出的方案能完全布局节点. 针对 50 的路由前缀,路由表 rrc00、 rrc01 和 rrc11 耗费的内存减少了 2.56 MB,延时减少了 17.8 ns,路由表 rrc08 耗费的内存减少了 5.12

MB,延时减少了 35.6 ns.

参考文献:

[1] JIANG W, WANG Q, PRASANNA V K. Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup [C]// Proceedings - IEEE INFOCOM, Phoenix: IEEE, 2008: 1786-1794.

[2] ERDEM O, CARUS A, LE H. Value-coded trie structure for high-performance IPv6 lookup [J]. Computer Journal, 2014, 58(2): 204-214.

[3] PAO D, LU Z, POON Y H. IP address lookup using bit-shuffled trie [J]. Computer Communications, 2014, 47(2): 51-64.

[4] BABOESCU F, TULLSEN D M, ROSU G, et al. A Tree Based Router Search Engine Architecture with Single Port Memories [J]. Isca '05 Proceedings of Annual International Symposium on Computer Architecture, 2005, 33(2): 123-133.

[5] JIANG W, PRASANNA V K. A Memory-Balanced Linear Pipeline Architecture for Trie-based IP Lookup [C]// High-Performance Interconnects, 2007. Hoti: IEEE, 2007: 83-90.

[6] SRINIVASAN V, VARGHESE G. Fast address lookups using controlled prefix expansion [J]. Acm Transactions on Computer Systems, 1999, 17(1): 1-40.

[7] RIS. RIS Raw Data[EB/OL]. [2016-03-12]. <http://data.ris.ripe.net>.

[8] BASU A, NARLIKAR G. Fast incremental updates for pipelined forwarding engines [J]. IEEE/ACM Transactions on Networking, 2005, 13(3): 690-703.

作者简介:

张建伟 男, (1978-), 博士, 副教授. 研究方向为高速低功耗 TCAM 设计、SOC 设计与验证、低功耗抗辐射集成电路设计、基于 Trie 的搜索引擎设计.

滕飞(通讯作者) 男, (1990-), 硕士, 研究方向为 SOC 设计验证、基于 Trie 的 IP 查找.

E-mail: tengfei@mail.dlut.edu.cn.

马万里 男, (1991-), 硕士. 研究方向为高性能 TCAM 设计、SOC 设计验证.

陈晓明 男, (1971-), 博士, 副教授. 研究方向为超大规模集成电路设计流程和可制造设计方法.