# VLIW

## 编译器

VLIW是将**多条互相独立的指令**，通过**软件（编译器）**的方式打包（Pack）在一起，我们将打包好的多条指令，称为**[instruction bundle](https://www.zhihu.com/search?q=instruction bundle&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3172217408}" \t "_blank)**。取指模块根据打包好的指令，送入各自独立的功能部件，并行执行，如下图所示，取指模块从[指令存储器](https://www.zhihu.com/search?q=%E6%8C%87%E4%BB%A4%E5%AD%98%E5%82%A8%E5%99%A8&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3172217408}" \t "_blank)取出了Instruction Bundle，共包含四条指令，然后同时发送给后级模块，从而实现了指令级并行。

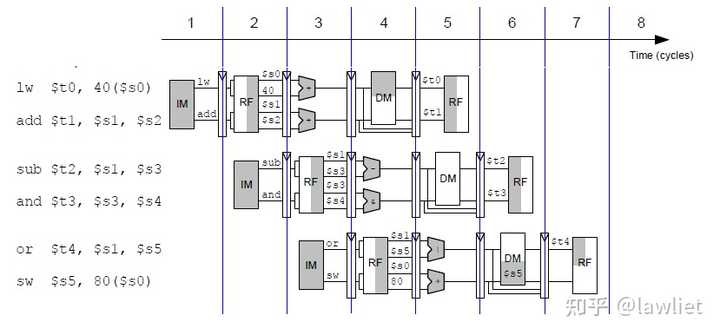
VLIW编译器工作：

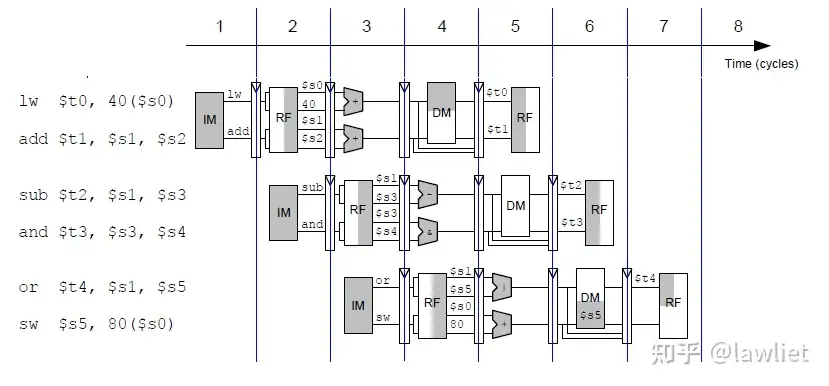
* 通过[静态调度](https://www.zhihu.com/search?q=%E9%9D%99%E6%80%81%E8%B0%83%E5%BA%A6&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3172217408}" \t "_blank)，尽可能的并行化（填满Bundle中的slot）；
* 保证同一个Bundle内的所有指令是互相独立的（必须要保证，因为硬件是不会去做检查的）；
* 通过静态调度，尽可能的避免数据冒险；（因为Lockstep的存在，一个堵住了，整个就堵住了）；

**对于传统的VLIW而言，其一般具有以下的特性：**

* 一次取多个指令（是指令，而不是[指令bundle](https://www.zhihu.com/search?q=%E6%8C%87%E4%BB%A4bundle&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3172217408}" \t "_blank)，记得区分这两个东西），具备多个功能部件（MIMD架构）；
* 对于同一个bundle里面的指令而言，基于lock step的方式执行（后面解释什么是lock step）；
* 同一个bundle里面的指令**静态对齐**，送给功能单元；

有了以上的概念，我们看一个典型五级流水线与VLIW相结合的例子，如下图所示。Bundle包含两条指令，因此理想情况下可以实现IPC=2。此外我们还可以看出来，**VLIW通常具有集中的存储器结构，以及独立的功能单元**。



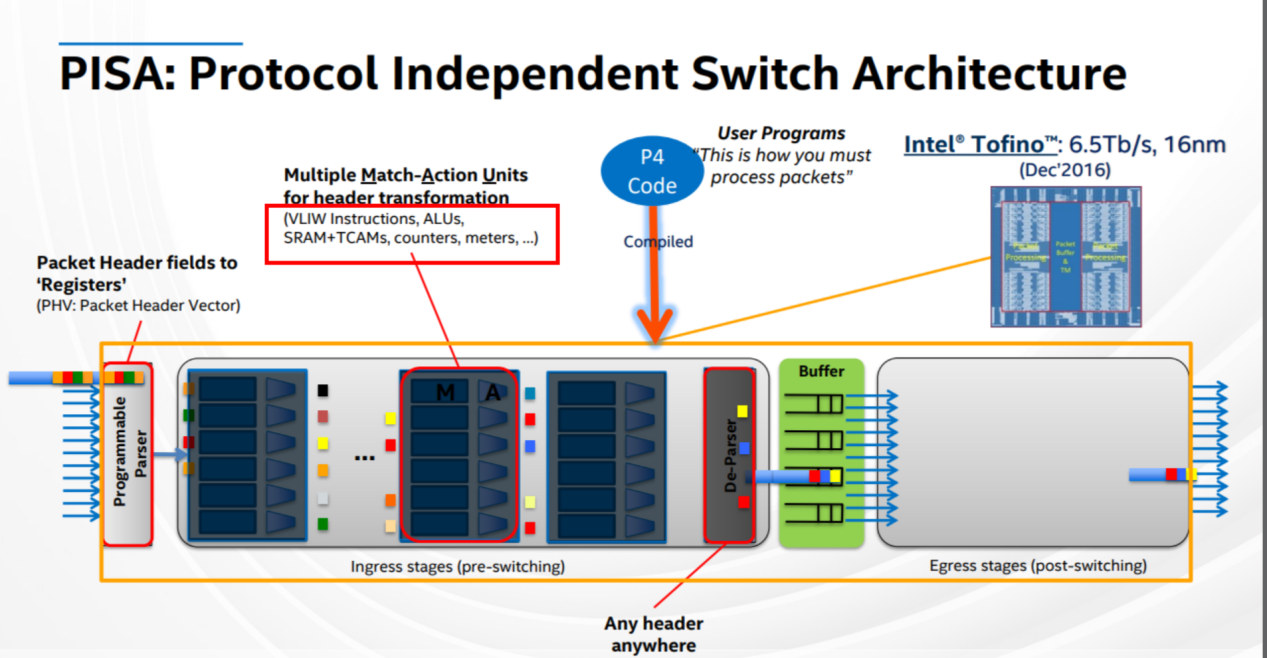


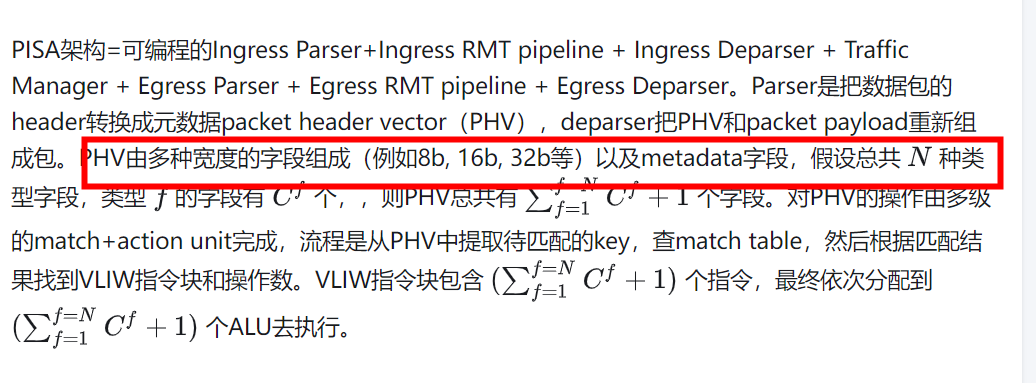
VLIW就是这么一套设计哲学，将原本硬件应该干的事情，全部交给编译器去做，这也就导致了编译器开发极为困难，因为要通过有限的代码信息，去找到互相独立的指令，即充分挖掘并行性。

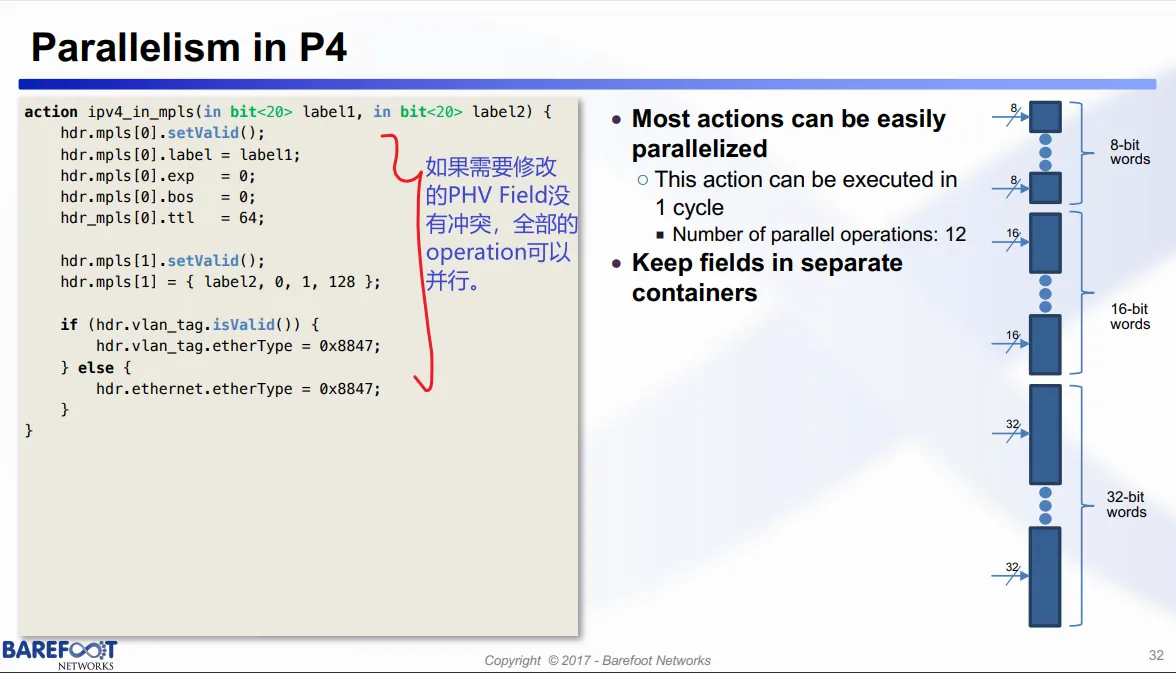
用[静态编译器](https://www.zhihu.com/search?q=%E9%9D%99%E6%80%81%E7%BC%96%E8%AF%91%E5%99%A8&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3176446163}" \t "https://www.zhihu.com/question/_blank)方法解决指令动态调度问题，是vliw的困局所在。如果任务单一且[指令集](https://www.zhihu.com/search?q=%E6%8C%87%E4%BB%A4%E9%9B%86&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3176446163}" \t "https://www.zhihu.com/question/_blank)变化不大，任务量空间可控，一旦指令复杂化且有多个硬件设计实例，会指数级加大设计难度，而此种情况下，设计一个硬件乱序调度器例如rename和rob的难度远小于静态编译器难度。

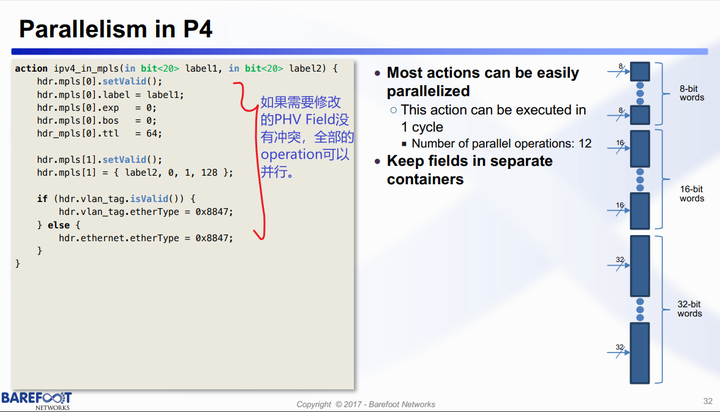
## RMT

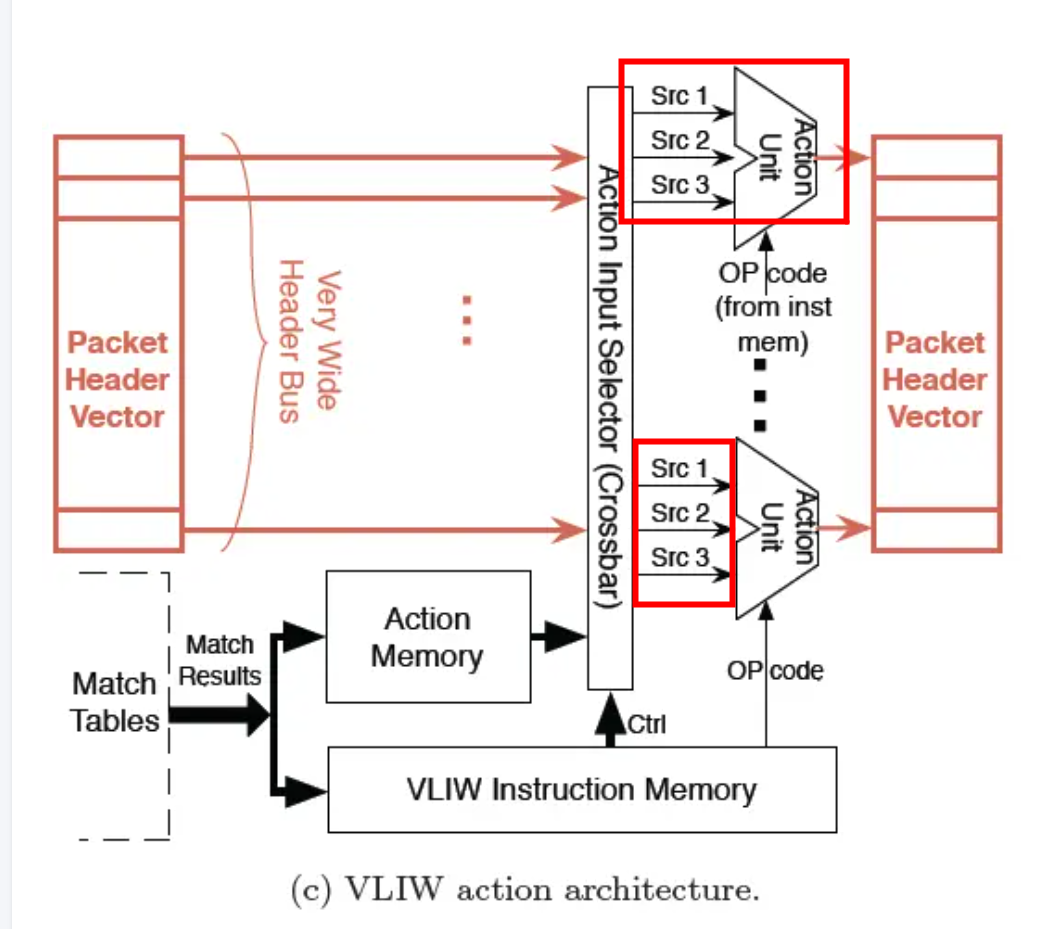
 RMT(Reconfigurable Match Action)架构 [1]，多个Action Unit的指令被打包在一个[指令块](https://www.zhihu.com/search?q=%E6%8C%87%E4%BB%A4%E5%9D%97&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3202063927}" \t "https://www.zhihu.com/question/_blank)。

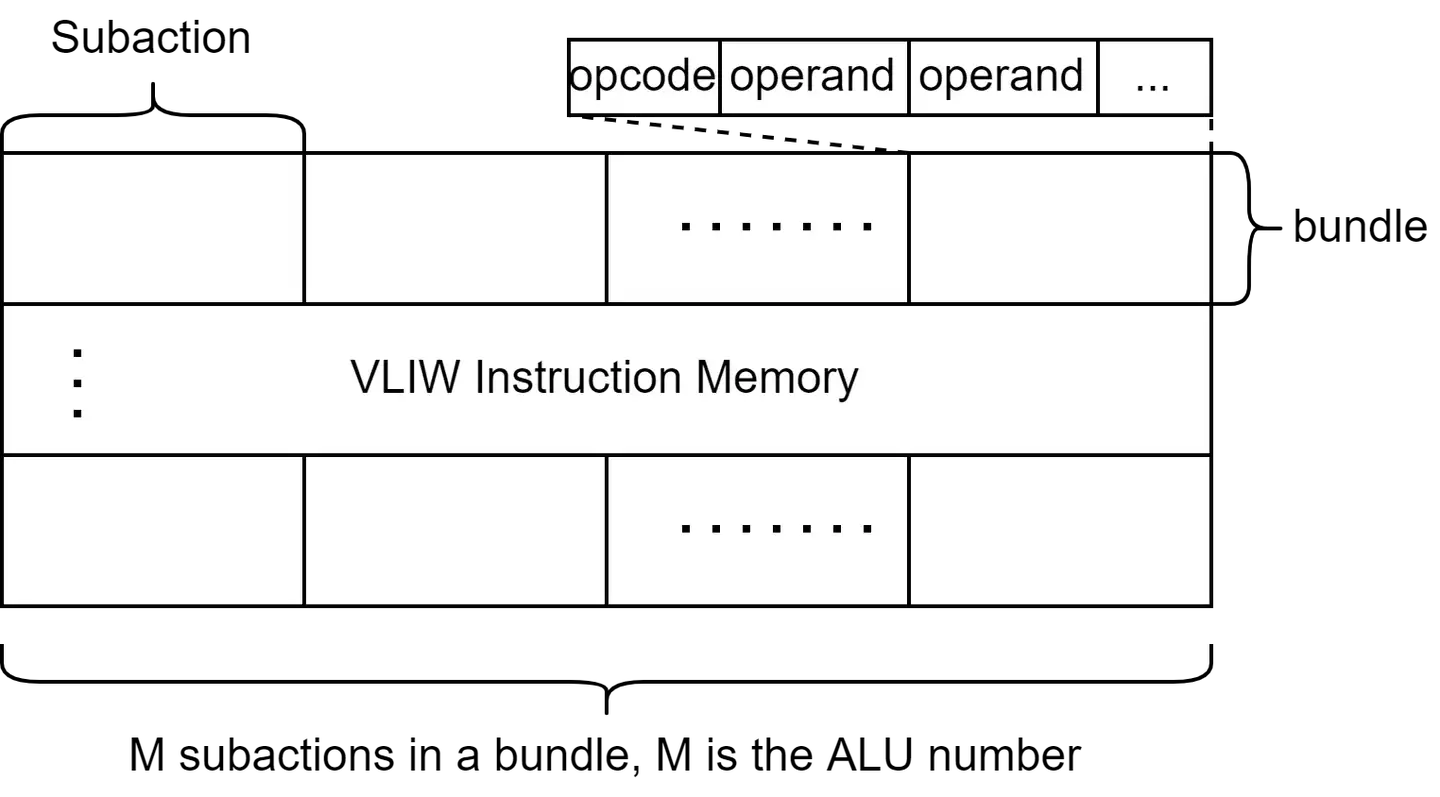










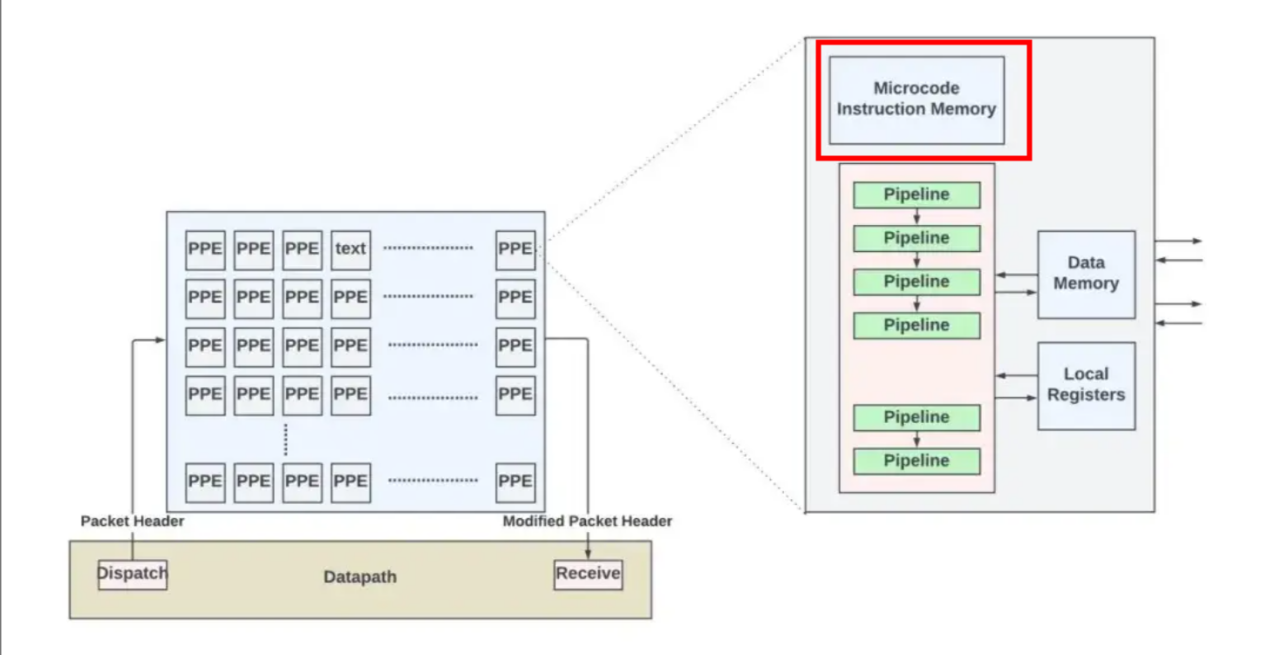


为什么RMT架构中使用VLIW呢？这里提一些自己的看法，欢迎大家指正。

* RMT架构的领域编程语言是P4 [2]，RMT想要达到的可编程性其实是“compile-time reconfigurability”，而不是“run-time reconfigurability”，也就是说每一级的指令怎么执行，怎么调度，完全由编译器确定的，这个比较符合VLIW的[设计理论](https://www.zhihu.com/search?q=%E8%AE%BE%E8%AE%A1%E7%90%86%E8%AE%BA&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3202063927}" \t "_blank)。
* RMT需要保证对PHV的处理达到线速，因此集成了一堆并行的ALU，需要比较充足的指令供应能力，如果一个program counter只能索引一条指令，肯定是不够的，因此需要以VLIW的形式将指令绑定成在一起。

## 微码

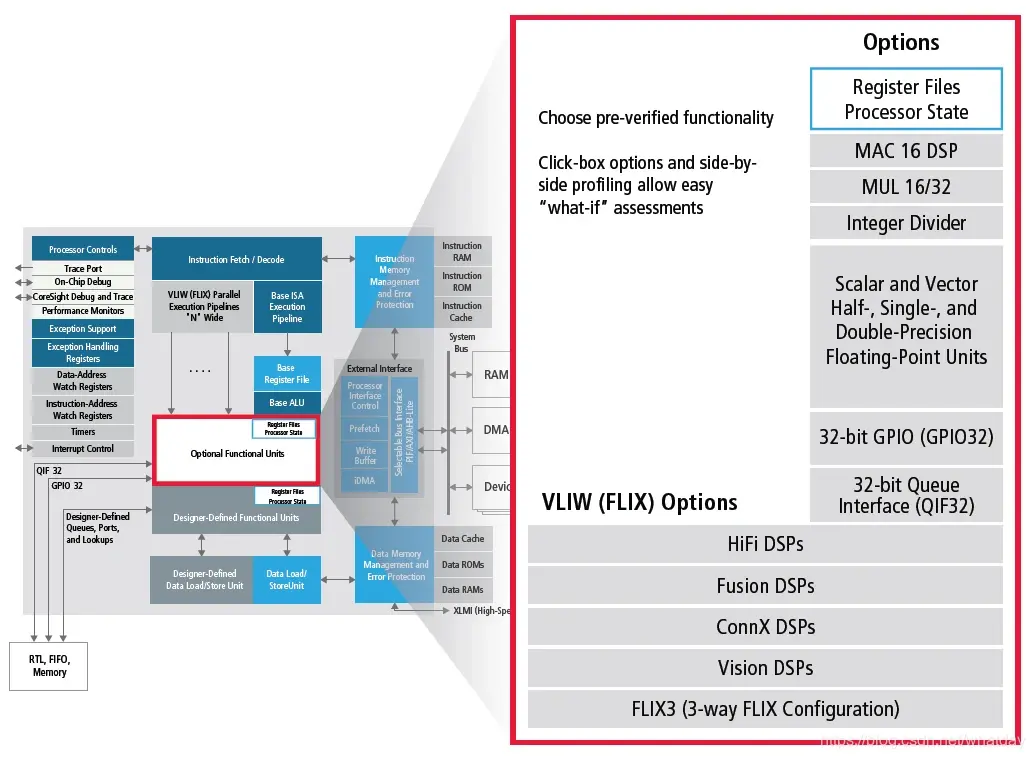
大约二十年前，随着网络协议快速演化，新的可选/扩展报头和隧道标准也随之涌现。数据包的处理是通过大量高度灵活且可编程的专用处理引擎实现的。这些专用处理引擎通常包含存储在片上和/或片外指令存储器中的微码指令。与 RISC 和 X86 指令集不同，微码是一种低级指令集，通常以非常长的指令字 (VLIW)的形式打包。处理引擎通过这些微码指令序列解析存储在本地存储器中的数据包头的不同字段，以确定数据包的结构，并执行上述所有入站和出站处理功能。处理引擎的硬件并不了解任何网络协议，它只是盲目地执行指令以形成新的数据包头并计算输出接口。



用于数据包处理的PPE

虽然基于微码的处理提供了无限的灵活性，但在芯片面积或每 Gbps 功耗方面效率较低。在混合方法中，一些功能（如过滤/最长前缀匹配查找、策略执行等）可以在硬件本地（硬件加速器）中实现，同时使用微代码指令进行数据包解析和其余的数据包转发功能。

## 泰思立达 FLIX技术



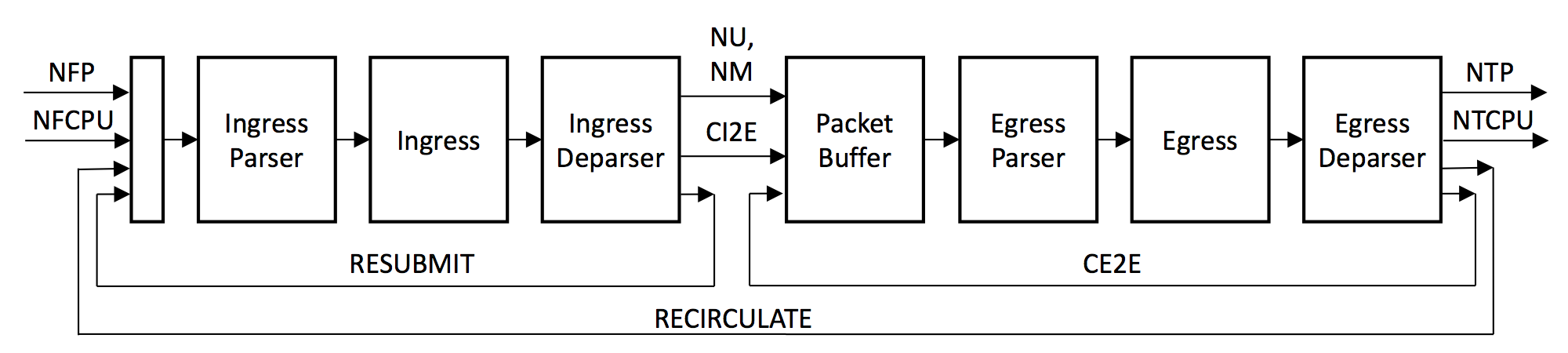
用户自定义功能的实现是通过FLIX。FLIX结构使得Xtensa LX7处理器称为一个VLIW处理器，如果需要可以实现2-30的并行。FLIX指令的长度为4bytes到16bytes之间的任意值。这种可变长度的FLIX指令可以与Xtensa处理器标准的16/32 bits指令无缝混合，所以不会有模式转换上的损失。图6展示了用户使用FLIX创造VLIW进行2-30级的并行操作

通过FLIX（可变长度指令字）Xtensa处理器可以除去在使用VLIW时指令寄存器的冗余损失。事实上，使用FLIX的Xtensa可以同时提供高性能和晓得代码长度。使用FLIX的损失是很小的，只是增加了2000个门用来进行指令的控制和译码，但是带来的好处是很大的。

Xtensa的C/C++编译器自动的从源代码中提取出并行操作并将多种操作加到FLIX中，所以即使不通过额外的软件资源同样可以获得并行计算好处。通过这种方法，一个three-issue的Xtensa LX7处理器在一个较低的频率下运行也可以达到对应其他设备在相当高时钟频率下的性能。

# Tofino

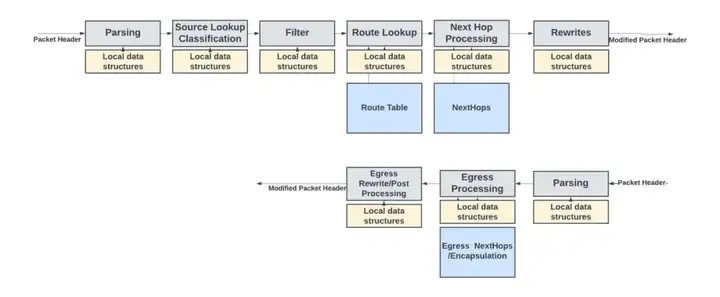
## Resubmit and recirculate



## 数据包处理Pipeline

随着高端芯片开始封装更多的 WAN 带宽，混合方法无法满足每 Gbps 的功率/面积目标。十多年前，一些网络供应商开始使用硬件pipeline（同时以本地/功能特定的指令/排序操作的形式提供有限的灵活性）本地实现所有数据包处理功能。

下图是基于Juniper的Express Architecture pipeline实现的入站数据包处理pipeline的概念图。



入站和出站数据包处理pipeline及其数据结构

该pipeline包含一系列后续块或模块，其中每个模块负责上文描述的特定功能。通常，整个数据包存储在数据路径存储器中，而报头（通常是数据包的前**128字节**）则通过数据包处理pipeline。由于数据包处理只关注 L4 的报头信息，因此不需要通过pipeline发送整个数据包。

根据吞吐量需求的不同，数据包报头以每周期一个数据包的速率或更低的速率通过pipeline发送。每个模块都有许多存储在 SRAM 中的本地数据结构/配置。

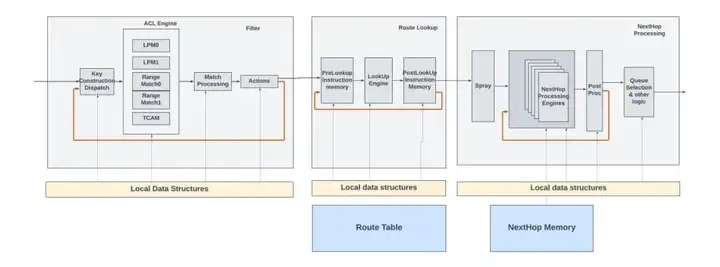
## Pipeline的灵活性

网络是一个不断发展的领域，为了适应新技术和新需求，经常会开发/标准化新协议和现有协议的扩展。从新的 RFC 标准发布到其实际在网络芯片中得到应用，通常会有3-4 年的延迟时间。这就是为什么在这些pipeline中具有一定的灵活性非常重要。

例如，除了对已知的L2-L4报头的标准解析之外，硬件还可以支持灵活的解析功能，以解析未来的协议报头或现有协议的扩展。这可以通过一系列CAM（内容可寻址存储器）和规则集来实现，它们指定了要查找新协议的Type/Length/Value字段的字节偏移量。

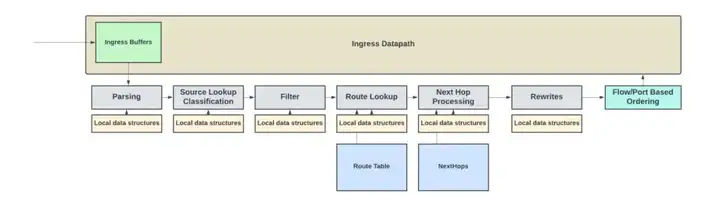
并非所有的网络应用程序都经过相同的数据包处理。例如，某些数据包可能需要多次查找。 第一次查找可能是 LPM（最长前缀匹配）查找，以确定数据包的下一个目的地。第二次查找可能涉及更具体的路由策略，比如基于策略的路由，其中决策基于数据包中的其他字段或应用类型。类似地，在 MPLS 网络中，第一次查找可能涉及读取 MPLS 标签以在 MPLS 网络内做出转发决策。当数据包到达 MPLS 网络的边缘，并且标签被弹出时，需要进行第二次查找，以便根据数据包的原始 IP 报头确定数据包的下一跳。

Express 数据包处理pipeline中的查找功能提供了这样的选项，其中第一次查找的操作可以指示后续的查找，并且报头循环回查找函数的开头以进行下一次查找。



数据包如何在每个查找模块内循环

需要注意的是，在数据包处理pipeline中，因为每个数据包都经过不同的pipeline并具有不同数量的查找、过滤器和下一跳操作，因此无法不会保持数据包的原有顺序。网络设备必须确保同一数据流中的数据包不会被打乱顺序。粗略地判断数据流的方式是以数据包进入的输入端口/接口为准。而更为精细的判断方法则是查看数据包的五元组，并通过计算哈希函数来确定数据流。pipeline末端的重排序引擎可以将数据包重新按照每个端口或每个数据流的顺序排列好。

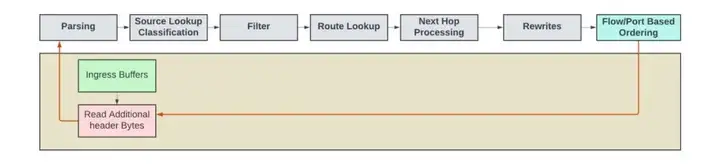


带有重排序引擎的数据包处理pipeline

## 再循环(报头超过128B)

在某些封装中，报头字节可能会超过 128B。对于那些在初次传递中无法检测到内部报头的情况，数据包需经历如下步骤：首先在剥离已解析的报头字节，接着从入口内存中读取额外的报头字节，并将新报头再次发回处理pipeline进行处理。在接下来的循环中，将重复处理步骤以处理内部报头。

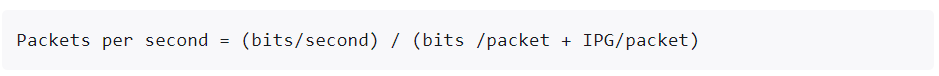
再循环应用的示例包括MPLS over UDP，其中需要处理两个以上的堆栈，以及基于防火墙的隧道解封装。



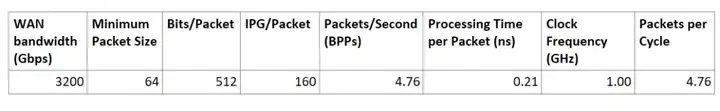
再循环的概念图

## pipeline和吞吐量

网络芯片所需的每秒数据包处理速率与能够进入设备的最小数据包大小（通常是 64B 以太网帧）、数据包间隙 (IPG) 以及设备的总 WAN 吞吐量成正比。



假设一个3.2Tbps 的设备需要处理连续到来的 64B 数据包，若要跟上这种处理节奏，在1GHz的时钟频率下，每周期几乎需要处理近5个数据包。由于每个pipeline最多只能每周期处理一个数据包，这意味着在这种情况下需要约5个数据包处理pipeline。就面积和功率而言，是相当昂贵的。



3.2Tbps 设备要满足 64B 数据包的线路速率需要 5 个pipeline

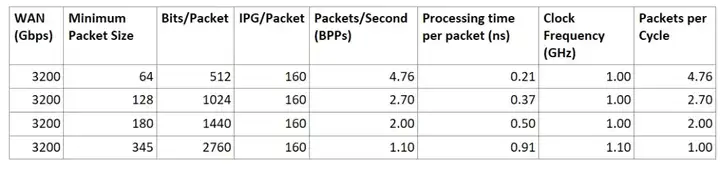
在实际网络流量中，平均数据包大小通常大于 64B。大多数流量通常使用最大传输单元 (MTU) 大小的数据包来最大化吞吐量。设计针对平均常用数据包大小优化的数据包处理引擎有助于实现更优的设计，有效利用芯片面积。那么，我们如何确定平均数据包大小呢？

一种方法是检查网络性能测试中使用的各种 IMIX 模式。

IMIX（ Internet MIX）是网络性能测试中使用的概念，用于更准确地模拟现实世界中的互联网流量模式。IMIX不使用统一的数据包大小，而是采用多种数据包大小的组合来代表互联网流量的多样性。例如，IMIX 可能包含小型数据包（64 字节，常见于 ACK 或控制消息）、中型数据包（大约 576 字节，通常用于特定应用数据）和大型数据包（大约 1500 字节，），并且它们之间有一定的分布比例。

对于 IMIX 数据包大小分布并没有一个普遍接受的标准。不同的组织可能会根据其特定需求和对网络流量的观察，定义自己的 IMIX 配置文件。谷歌和 Meta 在评估网络设备时都有自己的 IMIX 模式。

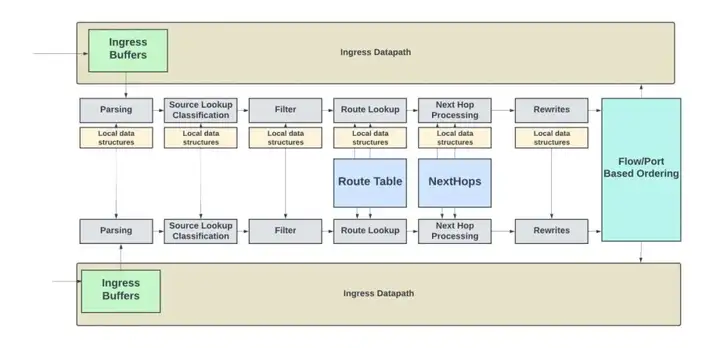
假设数据包处理需要以线速处理平均约 345 B大小的数据包，并在1.1GHz的时钟频率下运行，那么只需一条pipeline即可满足需求！



该表显示了增加平均数据包大小以满足线路速率时，如何减少pipeline数量

为了应对互联网流量可能存在突发性的特点，以及可能出现瞬态场景，即平均数据包大小小于350B，且有许多连续的小数据包涌入，这就需要在数据包处理输入端增设一个突发吸收缓冲区（即图中所示的入口缓冲区）。一旦这个缓冲区开始填满，硬件就可以执行优先级感知丢弃策略，即给予控制/保活数据包更高的优先级。丢弃策略的具体规定因供应商而异。

在上一代 Express Silicon (Express4) 中，为了实现3.2Tbps处理能力，并使得平均数据包大小达到约180B，决定增加两条pipeline。如下图所示，在实现这两条pipeline时，它们可以共享本地数据结构、路由表和下一跳内存资源。



## RMT，一个可编程的网络DSA

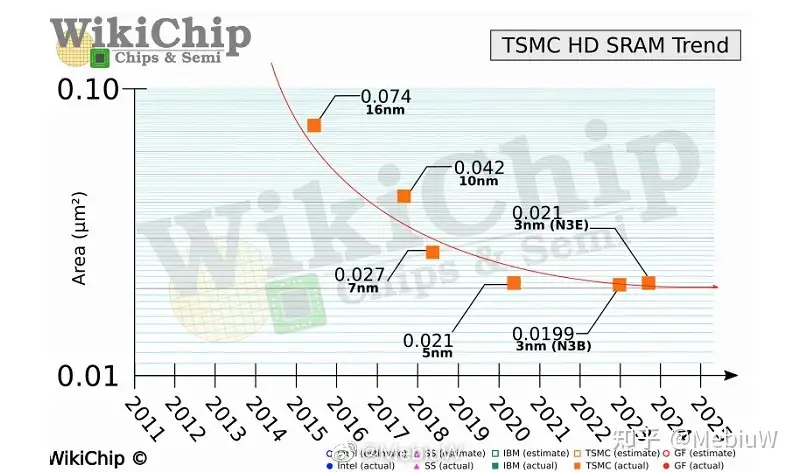
<https://zhuanlan.zhihu.com/p/676502456>

# CPU缓存为什么面积大容量还小？

<https://www.zhihu.com/question/638540167/answer/3356467259>

* 高速缓存使用的SRAM，一位需要用6个MOS管组成双稳态锁存电路来储存。
* 而内存条用的DRAM，一个MOS管加一个电容就能存一位。

CPU内部的缓存现在都是SRAM，对比内存所使用的DRAM来说，同样容量SRAM的面积功耗开销更大。此外，缓存的作用和内存还不一样，缓存是编程透明的，它只存一部分数据，不能像内存一样直接随机访问（或者很多直接定位数据），还有大量的晶体管面积拿去做查询用。



现在SRAM还面临着一个问题，随着工艺的微缩大家发现SRAM基本缩不动了、工艺提升但是密度无法上升。N5和N3E的SRAM密度一样…

所以从过去到现在再到未来，CPU缓存都是极为昂贵的东西。

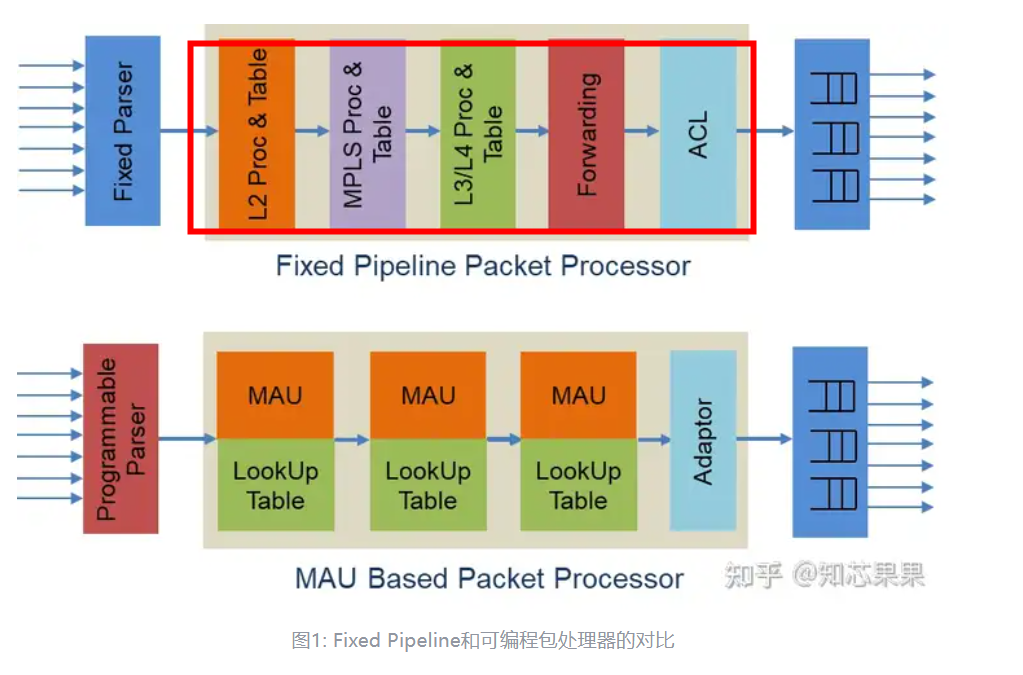
# 深入理解可编程网络处理器之MAU

**[深入理解可编程网络处理器之MAU：MAU的本质](https://zhuanlan.zhihu.com/p/520603410" \t "https://www.zhihu.com/_blank)**

**[深入理解可编程网络处理器之MAU：MAU的本质](https://zhuanlan.zhihu.com/p/520603410" \t "https://www.zhihu.com/_blank)**

**[深入理解可编程网络处理器之MAU：市场应用与分析](https://zhuanlan.zhihu.com/p/519639196" \t "https://www.zhihu.com/_blank)**

## Fixed Pipeline 和MAU 比较



# Linux + x86网络IO瓶颈

在数年前曾经写过《[网卡工作原理](https://www.zhihu.com/search?q=%E7%BD%91%E5%8D%A1%E5%B7%A5%E4%BD%9C%E5%8E%9F%E7%90%86&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3072399099}" \t "_blank)及高并发下的调优》一文，描述了Linux的收发报文流程。根据经验，在C1（8核）上跑应用每1W包处理需要消耗1%[软中断](https://www.zhihu.com/search?q=%E8%BD%AF%E4%B8%AD%E6%96%AD&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3072399099}" \t "_blank)CPU，这意味着单机的上限是100万PPS（Packet Per Second）。从TGW（Netfilter版）的性能100万PPS，AliLVS优化了也只到150万PPS，并且他们使用的服务器的配置还是比较好的。假设，我们要跑满10GE网卡，每个包64字节，这就需要2000万PPS（注：[以太网](https://www.zhihu.com/search?q=%E4%BB%A5%E5%A4%AA%E7%BD%91&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3072399099}" \t "_blank)万兆网卡速度上限是1488万PPS，因为最小帧大小为84B《Bandwidth, Packets Per Second, and Other Network Performance Metrics》），100G是2亿PPS，即每个包的处理耗时不能超过50纳秒。而一次Cache Miss，不管是TLB、数据Cache、指令Cache发生Miss，回内存读取大约65纳秒，NUMA体系下跨Node通讯大约40纳秒。所以，即使不加上业务逻辑，即使纯收发包都如此艰难。我们要控制Cache的命中率，我们要了解[计算机体系结构](https://www.zhihu.com/search?q=%E8%AE%A1%E7%AE%97%E6%9C%BA%E4%BD%93%E7%B3%BB%E7%BB%93%E6%9E%84&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3072399099}" \t "_blank)，不能发生[跨Node通讯](https://www.zhihu.com/search?q=%E8%B7%A8Node%E9%80%9A%E8%AE%AF&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3072399099}" \t "_blank)。从这些数据，我希望可以直接感受一下这里的挑战有多大，理想和现实，我们需要从中平衡。问题都有这些

传统的收发报文方式都必须采用硬中断来做通讯，每次硬中断大约消耗100微秒，这还不算因为终止上下文所带来的Cache Miss。

数据必须从内核态用户态之间切换拷贝带来大量CPU消耗，[全局锁](https://www.zhihu.com/search?q=%E5%85%A8%E5%B1%80%E9%94%81&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3072399099}" \t "_blank)竞争。

收发包都有系统调用的开销。

内核工作在多核上，为可全局一致，即使采用Lock Free，也避免不了锁总线、[内存屏障](https://www.zhihu.com/search?q=%E5%86%85%E5%AD%98%E5%B1%8F%E9%9A%9C&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3072399099}" \t "_blank)带来的性能损耗。

从网卡到业务进程，经过的路径太长，有些其实未必要的，例如[netfilter框架](https://www.zhihu.com/search?q=netfilter%E6%A1%86%E6%9E%B6&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":3072399099}" \t "_blank)，这些都带来一定的消耗，而且容易Cache Miss。

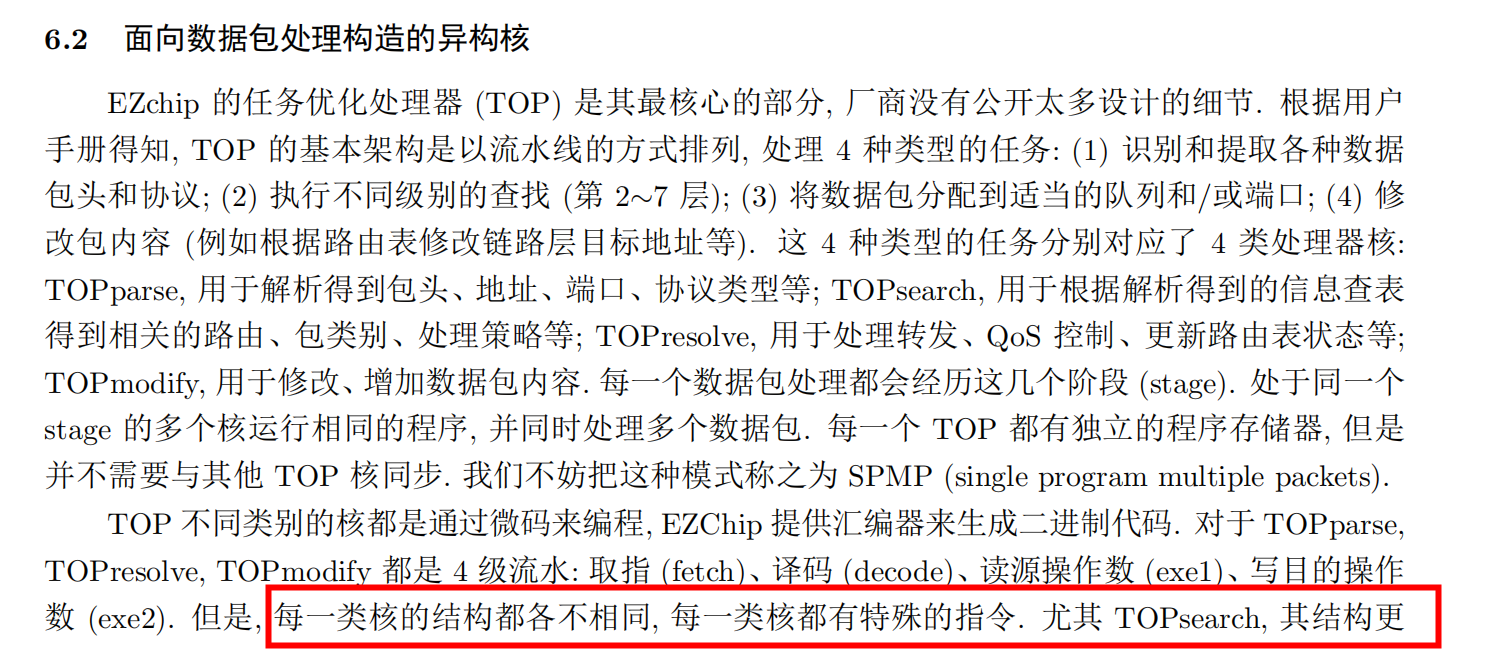
# EZchip网络处理器器

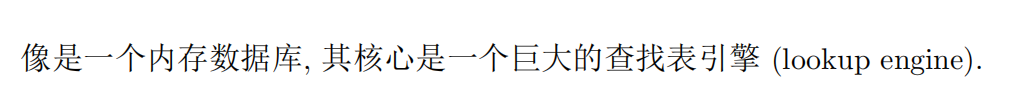
网络处理器是将ASIC (专用集成电路)的速度和一个CPU结合起来以提供高性能联网解决方案的集成电路设备。ASIC技术广泛用于分类、筛选和转发IP通信。不过,ASIC的问题是,它们是通过硬接线而具有某一特定功能集合的。如果功能集合改变或升级,ASIC必须被更换。当新标准(MPLS和Diff-Serv是最近标准发展的例子)出现时,ASIC必须被更换,而网络处理器能够通过灵活的软件体系提供硬件级的处理性能，基于NP(Network Processor)的网络设备具有高性能和灵活性。

网络处理器解决这个问题并提供增强性能。网络处理器是一种高性能、可编程的I/O设备。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。可编程性是区分网络处理器和ASIC的标准。与创建一次性ASIC相反,网络处理器允许创建网络设备的供应商给通信处理器编写代码来减少产品上市的时间。通过允许让供应商给设备添加新功能,这个策略也延长了设备的使用寿命。

网络处理器器件内部通常由若干个微码处理器和若干硬件协处理器组成，多个微码处理器在网络处理器内部并行处理，通过预先编制的微码来控制处理流程。可以对网络处理器编程以支持自定义算法。处理器被优化以执行网络设备的多种功能,包括帧分类、筛选、转发、标记、策略管理、业务流量调整以及Diff-Serv路由选择。这些功能与提供QoS、通信优先级、业务流量管理的交换机相关联。控制业务流的策略由软件实现,所以这些策略很容易升级。 对于一些复杂的标准的操作(如内存操作、路由表查找算法、QoS的拥塞控制算法、流量调度算法等)则采用硬件协处理器来进一步提高处理性能。从而实现了业务灵活性和高性能的有机结合。网络处理器的使用方法与ASIC相同。它们位于网络交换设备的数据路径中,与物理接口直接相连。处理机也可以执行组帧、分段和重组,以及其他功能。

EZchip的TOPcore网络处理器技术集成了许多小而快的处理器,每个处理器都被优化以执行特定任务,并在性能上能够比其他基于普通RISC处理器的网络处理机体系结构有10倍的改进。四个TOP(任务优化处理器)是TOPparse(数据分组分析与分类)、TOPsearch(搜索与查找)、TOPresolve (转发与QoS决策)和TOPmodify(数据分组修改)。TOPparse负责从数据流中鉴别和抽取各种包头、标记、地址、端口、协议、域、模板和关键词，它可以解析任何格式的包、封装方法、专有标记等。TOPsearch用解析后的域作为关键词，完成在相关的路由、分类和策略表中的查找。TOPresolve进行转发、QoS决策、升级查找表和进程状态信息。TOPmodify负责修改包内容，并完成在包中的重写、增加或插入操作。





# CISCO ASR9900

<https://cloud.tencent.com/developer/article/2058483>

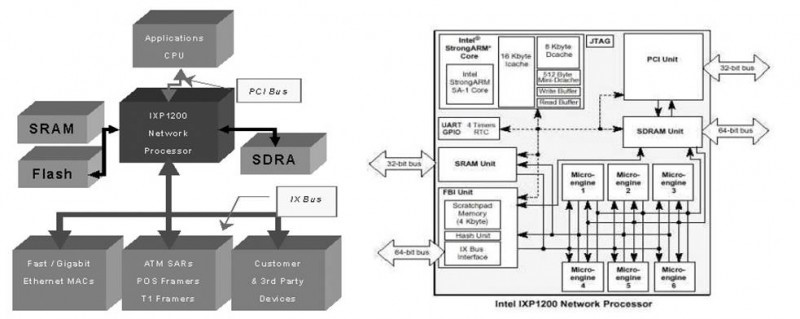
# Cisco Silicon One

<https://www.sdnlab.com/23843.html>

传统上，路由芯片和交换芯片有着完全不同的特点，体系结构也完全不同，例如路由芯片多采用运行至终结（RTC，Run-to-Completion）架构，交换芯片多采用流水线（Pipeline）架构。路由芯片聚焦于路由和业务功能、大缓存、可编程性，但带宽和功耗方面较弱，成本也高；交换芯片则聚焦于大带宽、低功耗、高性价比，但路由和业务功能较弱。

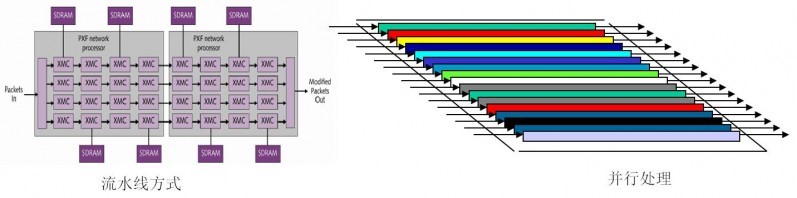
Cisco Silicon One™实现上基于RTC，保证功能的丰富性和可编程性，同时通过多Slice+共享内存交换提供类似于Pipeline的超高带宽和超低能耗比。事实上，作为该芯片架构的第一代，Cisco Silicon One™ Q100芯片在业界第一次突破了10Tbps这一传统路由芯片带宽容量的极限，吞吐超出了2倍以上，同时包转发能力（PPS）超出了3倍以上，实现了量级的跃升。

# 网络处理器和多核处理器

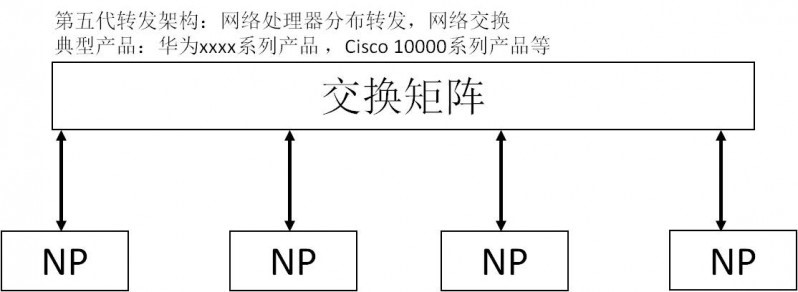


IXP1200有6个包转发引擎，相当于6个独立的RISC CPU，各自有独立的指令空间和寄存器，但又共享内存和总线；每个转发引擎有4个硬线程，共享指令空间，寄存器有线程独占和共享两种，可以充分发挥并行性，当一个线程进行内存或总线的IO访问从而等待时，切换出去让其它线程执行指令，吞吐量一下子提升4倍（不考虑切换线程的开销和没有完全消除的IO等待时延），总共6个引擎，性能可以达到同规格单核单线程CPU的24倍（理论值）。转发引擎完全可编程，具有2K条汇编指令空间，包转发引擎指令专门为包转发处理进行了特别设计，移位、bit操作、比较、跳转等都是单条指令，效率比通用CPU提升明显。

IXP1200处理RFC1812规定的IPV4标准处理过程时性能可达到1Gbps 64字节以太网报文线速，这在2000年的时候，已经非常非常牛了。国内的华为、中兴就是从这款处理器起步，慢慢的在IP宽带设备上追着巨人思科，直到差距越来越小，并逐渐超越。



网络处理器提升吞吐量的关键是靠流水线处理和并行处理。流水线方式下将对报文的处理分成若干个细微的过程，比如报文接收一个阶段、报文解析一个阶段、报文查找一个阶段、报文修改一个阶段、报文发送一个阶段，这样做的好处是每个阶段的指令总数控制在比较小的值，可以让报文在规定的时间内完成处理，保证线速；另一方面报文处理使用的指令基本上一样，可以简化引擎设计，提升指令执行效率。并行处理的好处就是充分发挥所有处理器所有硬线程的威力，隐藏IO访问时延，做到最大限度的吞吐量提升。今天像EZChip的NPS已经可以做到4K个硬线程，试想想如果我们的马路都有4K个车道，还会天天堵车吗？实际上流水线和并行处理在网络处理器的设计中是常常结合起来的，首先整体上分成流水线，然后在性能瓶颈的模块采用并行处理，这样既满足了吞吐量带宽要求，又满足了复杂业务不降低整体性能的要求。



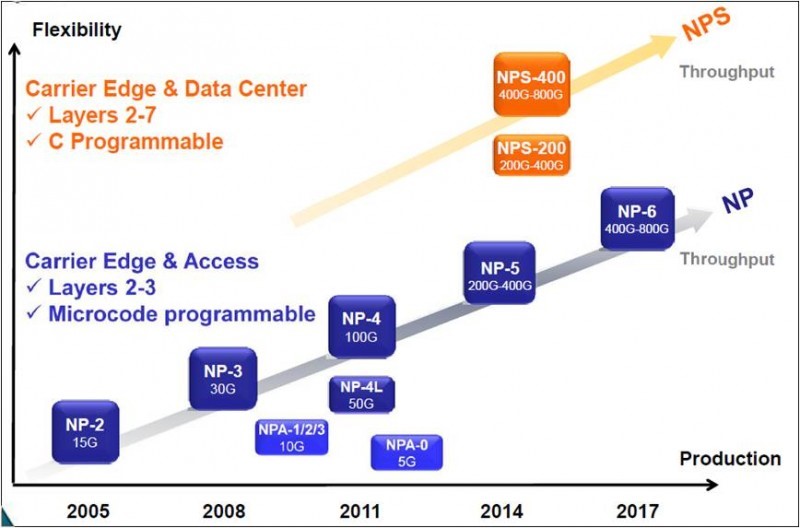
网络处理器的另一个大杀器就是灵活编程，在2000之后的十年中，大量建成的各种VPN网络、MPLS网络、IPV4/IPV6混合网络、精细的流量管理、各种安全网关，无一不得益于这种灵活的可编程性；曾经有个时期BT一度让运营商大为烦恼，带宽极度消耗但又不增值，对于这种传输层端口号不固定的应用，如何识别这种数据流并加以限制，灵活的网络处理器发挥了极大的作用。

网络处理器这种既可灵活编程又兼具性能的设计，无疑是这个领域的一颗核弹，其威力无人能及。在新世纪开始的头几年，已经有2.5G/5G处理能力的网络处理器问世，到2005年前后已经有10G的芯片商用。让我们记住这些当年的英雄们：

* Agere（Lucent）APP系列，处理能力2.5Gbps；
* C-Port（Motorola）C-5（16个RISC核，32个串接数据处理器），处理能力5Gbps；
* IBM，Rainer（高端NP，16个RISC处理器核，处理能力4Gbps），Charm（低端）；
* Intel，IXP1200（6个引擎，1Gbps）、IXP425、IXP2400（8个引擎，2.5Gbps）、IXP2800（16个引擎，10Gbps）；
* MMC，nP3400（2.5Gbps，集成4Gbps交换结构）；
* EZchip，NP1、NP2（10Gbps）；
* Xelerated，X11（10Gbps）。

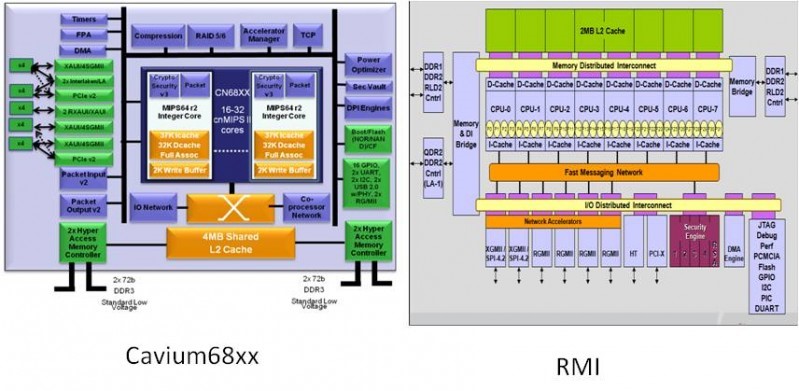
网络处理器为产业发展立下了汗马功劳，IXP系列NP广泛应用于宽带BRAS、城域网路由器、RAN传输等多个领域，华为的无线产品正是在IXP的基础上实现了全IP化，让华为的无线产品一举成为世界老二，直追E///；在数通路由器产品中，也大量使用NP芯片，极大的缩短了我们与世界第一的差距。

然而世事难料，网络处理器的好日子并没有持久。Intel无法忍受长期以来的亏损，于2006年6月把IXP卖给了Marvell，2012年Xelerated也卖给了Mavell，其它那些早年的英雄们几乎全部折戟沙场，唯独EZchip凭借着以色列人的聪慧和美国风投的执着，还在挣扎着，也许EZchip的确有过人之处，让我们再多看她两眼吧。

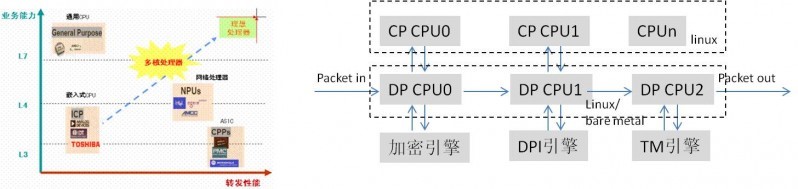


NP的消退肯定不是偶然的，有它必然的因素，首先NP诞生就是一个性能和灵活性折中的产物，这注定了它的性能始终比不过专用ASIC，而灵活性又比不过通用的CPU；其次，NP的开发门槛比较高，本人生平第一次接触的微码，那就是天书，尽管后面都采用了C语言编程，但对开发人员的要求还是相当高，要写出高性能的转发软件，必须对NP做到相当的熟悉；再有，NP转发引擎几乎都是裸跑，不支持OS，一个平台上的代码想移植到另一个平台上，对不起，工作量巨大，而且是不增值的工作。因此NP始终在一个狭窄的领域生长，属于细分市场，造就了大厂商如intel、ibm都不器重，只有小厂家玩，产业链始终不够成熟。NP在这个多彩的世界中扮演了独特的角色，如今已经慢慢褪去光环，作为一代微码工程师尽管有太多不舍，但时代就是在变，后NP时代更加精彩。

MIPS（Million Instructions Per Second），这名字取的就让人叫绝，著名的“龙芯”就是采用MIPS的指令集。MIPS公司设计的RISC处理器始于上世纪八十年代，1991年就推出第一款64位商用微处理器R4000。2000年之后MIPS战略重心放在嵌入式系统，和PowerPC成为通信领域、嵌入式领域两道靓丽的风景线。MIPS相对于X86的CISC，指令格式统一，种类和寻找方式较少，并采用了“超标量和超流水线结构”，大大提高了处理速度，适用于网络通信、信息安全、高端服务器等领域。MIPS被Cavium、RMI两大半导体公司发扬光大，基于MIPS架构开发的商业多核芯片，将网络连接、负责均衡、加/解密、应用加速等功能集成在一个芯片上，即片上系统SoC，并成功应用于通信领域各种场景，包括IP报文转发、加解密、查找等，在高度灵活性和较高性能方面，找到了最佳平衡点。



新一代多核处理器有Cavium的CN68XX、RMI的XLP732/832等，处理能力可达40Gbps，集成8/16/32个MIPS通用CPU内核，内置多个协处理器和加速引擎，提升数据包深层处理的性能。多核处理器和NP最大的不同在于它的CPU是通用的，可以运行vxworks或者Linux，这一点足以让NP汗颜；多核处理器没有NP的指令空间限制，对于包转发处理可以设计成流水线模式，也可以设计成RTC（Run-to-Completion）模式，构建业务的灵活性也是NP无法媲美的；基于Linux的SMP（Symmetric Multi-Processor），开发工作变得异常简单，尤其是移植其它通用平台的代码几乎是分分秒秒的事情。当然多核处理器也可以按照NP那样进行系统架构设计，可以基于裸核进行开发和调度管理，避免Linux系统开销，更大的发挥多核多线程的威力。多核处理器的硬加速协处理对于开发者来说就是一个函数调用而已，比如使用Cavium的加解密协处理器进行报文加解密时，仅仅就是一条普通的指令，程序员完全不需要关心更底层的细节。


多核处理器一度被认为是完美的处理器，在40Gbps甚至80Gbps以下已经没有NP的什么事了，在安全、业务路由器、L4+报文处理领域完全没有敌手，在华为的无线产品中，单板全部由NP升级换代为多核处理器，为产品的容量提升、性价比提升提供了有力支撑，华为无线销售过百亿美金有多核处理器不可磨灭的功劳。

## 网络处理器四大项目

虽然构建其中一个芯片需要许多IP，但有四个项目通常会引发讨论——SerDes、HBM PHY、网络级片上存储器，以及TCAM（内容可寻址存储器）。

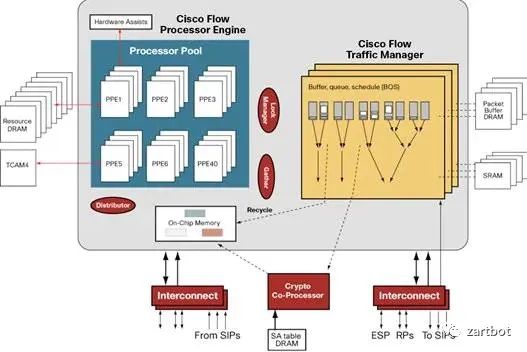
eSilicon公司营销副总裁Mike Gianfagna表示：“SerDes需要实现高速片外通信。HBM PHY需要将2.5D封装的HBM堆栈内存连接到ASIC上。网络级片上存储器就像双端口和伪双端口存储器一样，针对极高速度进行了优化，TCAM用于实现高效的网络数据包路由。

# **[FP2网络处理器架构研究](https://www.cnblogs.com/soul-stone/p/9571217.html" \o "发布于 2018-09-01 18:36)**

<https://www.cnblogs.com/soul-stone/p/9571217.html>

## RTC

FP的设计都是采用Run-To-Compile（RTC)的结构，即即便功能再多，一个处理器处理完一个包的所有功能， 然后编程上以C代码为主，内存操作也非常灵活：



从逻辑上很好的把功能抽象分离了， 一部分是On-Chip Memory用于等待调度到网络处理器PPE中，PPE是一个多核心的专用指令集的处理器（Tensillica的一个精简指令集），这也是我需要强调的，提高性能的方法也可以把通用CPU的指令集精简获得，同时针对网络处理，内存访问的位宽并不一定需要64bit或者32bit，所以很多时候用8bit 16bit的内存访问可以获得很好的加速，然后针对加密和ACL查询采用了片外的TCAM和加密协处理器，QoS这些采用了专用的TM芯片，而路由表查询就是采用了前文所述的Tree Bitmap的算法直接使用DRAM来处理。

# 网络可编程芯片-Trio

# Fpga 和 p4

## 软转的瓶颈--访存latency

但随着智能网卡的出现，虚拟机的网络性能大幅度提升，中心化的L4转发节点容易被多条大象流把单核打满，导致网卡收包队列满而丢包，服务质量受损，这个问题在RTC模型下基本上是无解的，在pipeline模型下有一定缓解，但最终单流的极限PPS应该也不会有数量级的提升，原因在于CPU访存的latency实在是慢的发指，近20年cpu性能翻了N倍而访存latency基本上没变化。

使用intel的MLC(Memory Lactency Checker) 测试能得出，同numa访存时延在80ns，跨numa130ns左右，按平均100ns算，只要每包只要发生一次cache miss(LB实际上很难做到，L3 Cache一般也就30M左右)，那么软件处理的性能就在10M pps以内，实际上一般到1.x Mpps就差不多了。

当然思科捐给[http://FD.io](https://link.zhihu.com/?target=http://FD.io" \t "https://zhuanlan.zhihu.com/p/_blank)的VPP(vector packet processing)，通过类似类似GPU SIMD(Single Instruction Multiple Data)的方法，能做一些i-cache miss的优化，不过看lb的plugin，也还是逐包处理，官网介绍1M flow的场景下，单核性能在3Mpps左右，且git上lb plugin本身做的比较简单，bps/cps/并发限速等逻辑以及flow/rs/svc级别的统计均不涉及。

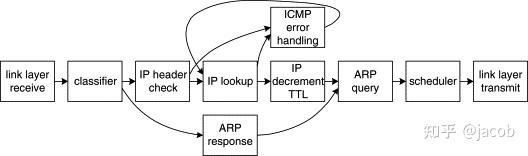
虽然FPGA已经被证实能够高效地处理网络数据流，但使用FPGA的在网计算也存在一些挑战。一方面，与CPU和GPU相比，FPGA的时钟频率更低，片上和片外存储带宽更小。例如，FPGA的典型时钟频率约为200MHz，比CPU慢一个数量级；FPGA内的单块BRAM存储器或外部DRAM的带宽通常为2至10GBps，而Intel Xeon CPU的内存带宽约为60GBps，GPU更可达数百GBps。但是，CPU和GPU的核心数量有限，限制了并行性，而FPGA内置了大量并行可重构单元，虽然单个BRAM的带宽可能有限，但如果能并行访问数千个BRAM，则总内存带宽可以达到数TBps。因此，为了实现高性能的在网计算，必须充分利用这种并行性优势（这部分数据源于《基于可编程网卡的数据中心系统》）。

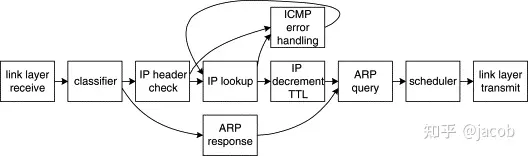
# 新华三EnGiant 660

这个芯片应该是主要用于核心路由器/交换机，是一个NP（网络处理器）/TM（[流量管理](https://www.zhihu.com/search?q=%E6%B5%81%E9%87%8F%E7%AE%A1%E7%90%86&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1876739086}" \t "_blank)）集成在一起的芯片，支持1.2Tbps的接口吞吐量。按照这些给出的数据，这个芯片属于高端芯片，能够满足运营商核心网的应用。

从[新华三](https://www.zhihu.com/search?q=%E6%96%B0%E5%8D%8E%E4%B8%89&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1876739086}" \t "_blank)公布的特性信息看：

1. 第一个特性称为革命性 - C语言完全可编程，这个有点噱头的意味。NP（Network Processor - 网络处理器）早已经从最初的类似汇编的微码演进到类C的语言编程，所谓类C语言，基本上就是C语言，只是有很多的定制指令，抛弃了一些路由器/交换机转发不会用到的C语言特性。
2. Engiant 660集成了256个CTOP处理器，每个CTOP支持16个[硬件线程](https://www.zhihu.com/search?q=%E7%A1%AC%E4%BB%B6%E7%BA%BF%E7%A8%8B&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1876739086}" \t "_blank)，共4096个硬件线程。从这一点看，应该是采用了RTC（Run To Completion）架构，即传统的NP架构。所谓CTOP（C programmable Task Optimized Processors），即C可编程任务优化处理器，也有点宣传的味道。因为NP的Core，基本上都需要定制，因为转发流程使用很多专有指令，比如Checksum计算、LPM[查找指令](https://www.zhihu.com/search?q=%E6%9F%A5%E6%89%BE%E6%8C%87%E4%BB%A4&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1876739086}" \t "_blank)、ACL查找指令等等。这里的专为C语言任务优化，不知道添加了什么特性。





Run To Completion示例：即全部的处理在一个Core上完成，与流水线架构对应

支持专业的智擎操作系统，实现快速网络业务开发，同时支持Linux操作系统。Linux系统，应该是运行在一个用于管理的Core上，比如使用ARM Core。

1. Engiant 660支持1.2Tbps接口吞吐，按照给出的应用示例，最高支持960Gbps带宽。按照960Gbps计算，对于64B小包，每个线程支持348Kpps的处理能力。对于但Core的处理能力来说，问题不大，关键是LPM查找是否能够支持1.43G/s的查找。当然，也可以使用多个查找引擎并行处理，只是需要更大的内存。
2. Engiant 660集成了[查找引擎](https://www.zhihu.com/search?q=%E6%9F%A5%E6%89%BE%E5%BC%95%E6%93%8E&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1876739086}" \t "_blank)单元（Search Engine），可实现IPv4、IPv6的[快速查找](https://www.zhihu.com/search?q=%E5%BF%AB%E9%80%9F%E6%9F%A5%E6%89%BE&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1876739086}" \t "_blank)，没有说线速查找。另外，没有说有ACL查找引擎，估计对于小规格的ACL，应该是片内有TCAM，对于大规格ACL，可能需要片外TCAM。
3. Engiant 660集成了TM，支持1M队列、5级调度，这个规格还是很大的，需要大量的片内SRAM。当然，还要看每一级支持的调度算法和Shaping能力情况，这一点并没有公布细节。

整体来说，这款芯片从规格和性能上来看，都是不错的一款NP。

## FPGA的优势

简单来说可以把FPGA理解成每条指令都有独立的处理单元处理，逐个数据包依次在几百个处理单元上流过，比如每个处理单元按4个节拍处理完一个包的话，主频在200MHZ时，理论上估算是可以做到50Mpps的，不过由于DDR带宽之类的问题，最终只能到30Mpps，当然利用sram做一点flow的cache后，可以突破30Mpps。

易知，SRAM的大小是有限的，用来做cache加速，或者做一些较小的表倒还好，但是如果想把千万规模的flow 存在FPGA上，还是得外挂DDR(或HBM)，DDR的访存时延对于FPGA来说一样存在，也差不多是100ns左右，为何在FPGA上达到30Mflow的场景下，pps仍然达到30Mpps呢，原因是对于FPGA而言读DDR不需要等到数据传输完成，只需要等待DDR控制器返回收到读取命令即可，用软件的思维看，读DDR在CPU是同步的，需要等待数据传输完成，但是在FPGA是异步的，仅需要确认DDR收到读取，异步+指令流水屏蔽了访问DDR的时延。

因此，基于FPGA，外挂DDR，实现类似openflow的flow offload，是可以满足LB flow全量offload的情况下，依然做到比较好的PPS性能。

## openflow的缺陷

不过openflow这种match & action的模式，对于LB来说还是不够灵活，举个例子来说，如果需要支持双层vxlan 或者说 vxlan中通过nsh或者geneve协议携带meta信息时，对于openflow而言，提取key的parser模块是需要修改的，以及如果需要encap这些协议头的话，action模块也是需要修改的，因此基于openflow这种模式的offload，是无法流片成asic芯片的(快转逻辑没办法固化)。

## P4语言

不过P4语言(Programming Protocol-independent Packet Processors)是协议无关的，parser、action都是可以自定义的，intel的tofino交换芯片在大量无状态转发的网关中已经用起来了，对于有状态lb而言，tofino芯片(无DDR,sram在几十MB左右)以及pensando支持P4的网卡(仅支持10Wflow)，主要的问题在于flow表太小，但是如果自研一套支持P4协议的FPGA，外挂DDR后流片成ASIC，那么性能方面会有较大提升，因为ASIC主频可轻松达到上G，而FPGA做到200Mhz就不错了；另外功耗方面也会优于FPGA ;以及如果量大的话，成本方面相比FPGA是有优势的，AISC的劣势在于，流片后的逻辑无法修改，以及首次开发成本太高。

# 10G线速转发和cpu

有人统计过，想要线速处理10G（万兆）的网络，需要的约4个英特尔志强CPU的核。也就是说，仅仅是进行网络数据包的处理，就要占用个8核CPU的一半满负荷时候的算力。现在数据中心网络不断升级，从10G到40G 、100G，400G，以及未来还有800G，1.6T的网络速率，这些性能开销显然是承受不起的。

”

# BCM芯片FP原理

BCM芯片有几个大的模块： VLAN、L2、L3和FP等几个，其中FP的使用也最为灵活，能解析匹配数据包文的前128字节比特级的内容，动作包括转发、丢弃、结合qos修改相应字段、分配vid、流镜像、流重定向、指定端口转发（比如CPU口）、指定下一跳转发往、指定隧道转发等，往往在实现功能上有意想不到的功效。简单来说，如果硬件和BSP分别是九阳真经和九阴真经的话，那么port和vlan是少林七十二项绝技的组合，L2转发则是显得有点悠闲的峨眉派功夫，当然L3则是以太极拳为代表的武当派功夫，那么FP可以是以乾坤大挪移、吸星大法等为代表的魔教的邪而又邪的“旁门左道”，当然其他功能是零零散散的其他门派功夫。能够灵活运用好FP是增加很多交换机新功能的一种常用的手段。本文总结下FP这个模块BCM在硬件上的实现原理及SDK的相关数据结构。因为FP在实现功能上的灵活性，在此希望能抛砖引玉，激发大家更多的应用FP实现新功能的火花。

* BCM芯片FP实现原理  
  FP的全称是Fields Processors，也称为ContentAwareProcess（CAP），在BCM较早的芯片称为Fast Filter Processors（FFP），和现在的FP相比有一些原理不同，不过现在交换芯片已经不再使用FFP，所以在此也不再介绍。FP本质来说，是一组相互之间有关联的表，一起通过查找、匹配等来决定对报文施加的动作；在BCM芯片交换机中，有三种查找查找方式：hash，index，tcam。FP的查找主要用到了index和tcam，其中CAM的全称是ContentAddressable Memory，中文是内容寻址器，TCAM则是Ternary ContentAddressable Memory，中文称为三态内存寻址器，TCAM的实现是通过对应比特位+掩码产生三种匹配方式：掩码为0表示不关心、掩码为1且对应位为1或掩码为1且对应位为0。 这就是三态的具体含义。

在我们自研交换机所用的芯片中有三个FP：VFP(VLAN FP)、IFP（ingres FP）和EFP（Egress FP），另外在四代芯片kylin卡中曾出现外扩FP，称为E-IFP，表项大小为128K，为L2和L3转发用，有点openflow的意味。其中VFP主要用于对报文tag的处理，比如添删或修改vid的灵活QINQ的实现就基于此FP；IFP的用途比较多，主要是对进入端口后的报文进行处理，主要有入口acl、流重定向、流镜像、设置下一跳、为qos数据报文分类等用途；EFP的用途和IFP类似，但是因为EFP是报文在转出前在出端口进行处理的规则，IFP有的动作类型在EFP不太适用。虽然三种FP用途和数据包流经顺序不太一样，但是硬件原理是一致的。下面介绍下FP的硬件原理。

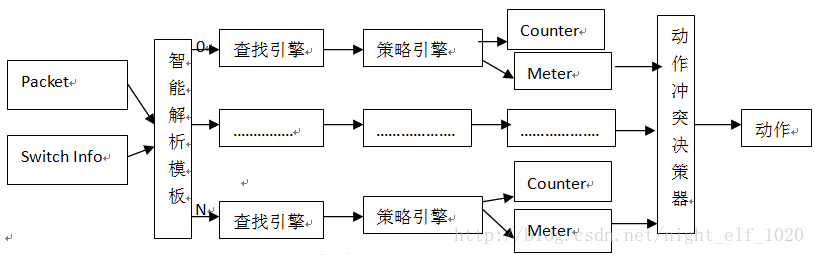


图1 FP原理组成图

图1中，每一个查找引擎和策略引擎及后面的counter资源和meter资源组合成一个规则组，芯片称之为一个slice，从图1可以看出，FP的实现有五部分组成：

1. 智能解析模板：主要将报文信息（最多报文前128字节，可以精确到每一位bit）根据对每个slice的care字段将各对应字段解析出来，再加上前面L2、L3的转发信息，一起送给每个slice的查找引擎去匹配；
2. 查找引擎：将解析出来的字段按照TCAM方式去查找本slice的规则是否有匹配的，即HIT的，只要有一条hit的即刻返回这条规则的index不再继续查找本slice后面规则，后面即使还有匹配的规则；这样做就是为了保证一个slice内部规则的优先级；如果没有匹配说明此slice没有匹配的规则或根本就没有规则，后面的流程也无需再走；
3. 策略引擎：根据查找引擎得到的index直接索引策略引擎的动作，动作类型有转发、丢弃、重定向（包括到CPU口且可指定队列）、流镜像（包括到CPU口且可指定队列）、修改报文特定的字段（COS、DSCP、EXP等）、与后面的meter一起对报文染色并对不同染色报文指定相应动作、指定下一跳、指定ECMP、指定TTL是否修改、指定URPF的模式等相关动作；需要说明的是，一条规可以对报文执行多种动作，当然需要报文动作之间是不冲突的，即slice规则的动作冲突是靠配置下发来检查的，同一条规则有冲突的动作无法下发硬件；
4. Counter和meter资源： counter资源用于计数，有基于byte和packet的两种方式；meter主要用于测速，然后根据速度对报文进行染色（绿、黄和红）然后对报文应用不同的QOS策略；meter的工作原理可以参见我原先写的有关令牌桶相关文档。
5. 动作冲突决策引擎：前面说过，一条slice的动作冲突是靠配置下发检查来实现的，冲突的动作无法同时下发到硬件；但是FP通常有多个slice，每个slice都有规则被匹配且动作时间有冲突时，需要动作冲突决策引擎来处理到底执行哪一个规则的动作，如果多个动作不冲突都执行；原则是丢弃、重定向等优先级最高，其他时候看slice号（这个slice号有的芯片只支持是物理的，高级芯片支持虚拟slice号），slice号越大优先级越高；

我们一条规则的匹配报文长度信息是有限的，对于IPV4报文同时匹配SMAC、DMAC、SIP、DIP等信息的时候，就不够了，芯片提供了将两条规则合并成一条规则，组成更大长度规则的方法，主要有图2示的两种,：

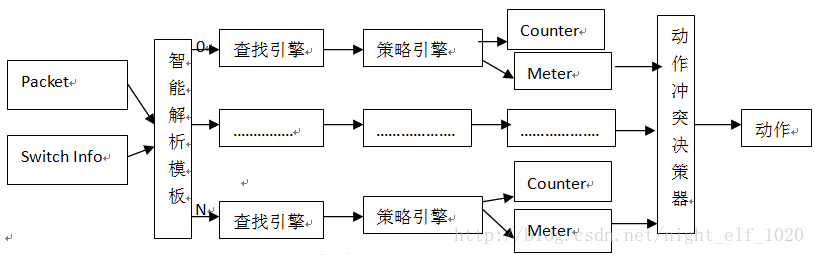


图2 两种slice宽模式

第一种是将一条slice的规则分为前后两部分，然后进行如图2左边的方式拼成double模式，这种模式称为double wide模式；第二种是用两条slice，直接如图2右边所示的方式拼成double模式，这种拼接方式称之为slice-paring模式。这两种模式，有的低级芯片都不支持，只能用单倍模式，有的芯片支持其中一种，我们的redstone交换机就只支持左边的这种方式。还有的芯片可以同时支持这两种拼接方式，那么就可以利用这点拼接处具有更大长度信息的四倍模式：

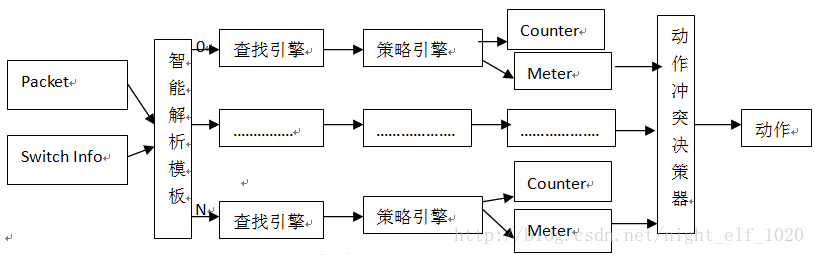


                       图3 四倍key模式

这种模式常用于IPV6报文的匹配中，因为IPV6的SIP和DIP实在太长了，再加上匹配其他信息，只能用四倍模式才能完全覆盖所有字段。但是我们的redstone交换机只支持slice-pairng模式，所以在IPV6报文的匹配中需要做折中。

前面我们提到slice有物理slice和虚拟slice，这个与物理内存和虚拟内存有点类似，FP都有物理slice，在高级的芯片上，为了更好的解决slice之间的动作冲突，对slice进行了虚拟编号，虚拟slice号越大优先级越高，这样就可以实现动作的优先级指定；可能做过物理slice的同学能体会为了保证各种应用slice的优先级在软件处理所做的代码处理工作有多么的艰辛；硬件进步这么一步，支持虚拟slice后，这部分工作就完全交给硬件来处理了，我们只需要指定优先级高低就可以了。而且虚拟slice还支持虚拟slice组的概念，每一个虚拟slice组就像一条slice一样，只会有一个动作产生出，这样就又大大减少了动作冲突的机会，而且还能使得每种应用使用更多的slice资源，无需考虑因为物理slice带来的动作优先级打破应用的优先级，更符合实际。