WIND RIVER | intel

高性能多核网络软件设计选项

从英特尔®数据面开发工具套件到风河网络加速平台

目 录

概述 2
多核网络软件设计选项 2
传统的 LINUX SMP 3
英特尔数据面开发工具套件 3
Intel DPDK 用户空间库和 Linux 4
Intel DPDK 和 Linux SMP 7
网络加速平台与 Intel DPDK 8
接收端缩放和数据流限定 9
高速缓存考虑因素 10
无锁实现 10
超大页面 10
管理面 11
代理接口 11
控制网络加速引擎 11
网络加速后台程序 11
风河 Workbench 11
优点 12
设计选项比较 12
结论 13

概述

随着 CPU 核心数量的增加，传统的对称多处理系统的扩展性相对受到限制，因共享内存争用和多核调度造成的瓶颈问题变得愈发突出。这就出现了包处理性能的平整化，即随着内核数量的增加，性能只有微小的改进，在一些情况下甚至还会出现性能的下降。

如果设备提供商是出于简单和方便的原因而准备将自己的设计建立在传统的 SMP 上，那么在开始之前，他们应当先了解这种方式可能存在的缺点。当从四核系统移植到八核或更多核的时，性能的提升并不一定等同从单核移植到双核或四核系统的效果。

本文探讨了一些不同的多核网络软件设计选项，这些选项要比传统的 SMP 具有更好的扩展性。设备提供商可以通过这些选项实现面向未来的设计，使用一个通用的架构就能满足当前的性能要求和未来的扩展需要。

本文首先对典型的设计选项进行了概览，然后对传统 Linux SMP 进行了简要总结，接着将探讨英特尔数据面开发工具套件（DPDK）的使用和风河公司的多核非对称多处理（AMP）产品 – 风河网络加速平台。与仅使用 Linux SMP 的方式相比，这些选项均提供了更好的性能和扩展性。

多核网络软件设计选项

一般来说，当设计多核网络平台时，系统架构师只能选择商业现货的对称多处理（SMP）操作系统或使用非对称多处理（AMP）重新进行设计。SMP 是一种共享架构，所有的内核都运行一个操作系统实例，并共享所有的数据。AMP 是一种分区架构，不同的内核运行不同的操作系统或者是同一个操作系统多个独立的实例。

如果设计合理，那么如今的应用程序可以不太费力地移植到通用的对称多处理操作系统架构中。操作系统可以决定在何时、何地来调度应用程序线程，以及如何将线程无缝地从一个核移植到另一个核。经过精细调整后，这些应用程序一般都可以实现很好的性能，但是缺乏扩展性则是一个问题。随着核心数量和线程的增加，共享数据结构中出现互锁的频率也会增加，再加上其他性能瓶颈，从而造成共享内存模式的崩溃。

另一方面，非对称多处理的系统则往往是按照特定用途建造的，其架构在很大程度上取决于应用程序本身的需求。在网络方面，AMP 的架构往往包含一个核或少数几个核上的单个管理面实例，以及在其余核上的多个独立的专门用途数据面实例。要将一个应用程序从通用操作系统移植到专门用途的 AMP 平台，不仅需要将应用程序的数据面和管理面分离，而且可能还需要“自己手工配置”底层库的实现以及类似于内核的数据面服务。这些额外的工作可以带来更高的性能，而且如果软件针对运行的硬件架构进行了优化，那么性能提升的效果将更加明显。

传统的 LINUX SMP

计算行业内的人士都很清楚，要充分利用多核能力，软件必须是多线程的；执行的序列必须不止一个，这样内核才能同时运行不同的线程。换句话说，软件能够并发执行自己的工作，这对于提高多核性能至关重要。如今很多应用程序都利用了这一概念，并获得了令人印象深刻的结果。

类似的，Linux 也取得了很多进展以利用多核处理器的优势。调度程序效率的改善和线程亲和性，以及避免粗粒度的内核锁定，这些都有助于提高性能。虽然毫无疑问这方面的改进还将继续，但是随着内核数量的增加，共享内存争用和核心间调度成为严重影响性能的因素。单独的网络堆栈实例具有多个线程，但是却共享数据，这也一直是传统 SMP 系统中的一个瓶颈。其他限制性能的因素还包括旁路转换缓冲区（TLB）击不中概率的增加以及受限的页面大小。

此外，在很多网络应用程序中，数据在系统中的进出是性能上的瓶颈。数据路径应当精简，从而尽快地将数据包在网络接口卡（NIC）和用户空间应用程序之间移动。较新的处理器架构有着更高的总线效率、较大的高速缓存、更快的内存、并且采用了对称多处理，这些都有助于提高数据移动效率并减少 CPU 停滞。Linux 也在网络输入/输出（I/O）的处理方面进行了改进，不过精简数据路径的单一关注点与通用操作系统的任务有所冲突，因为操作系统必须在各方面进行平衡。

所以，尽管 Linux 和 Linux 应用程序最近取得了一些进展，但是 Linux SMP 最突出的贡献是使应用程序能够利用操作系统实现多核支持，方便地启动和运行，并且在小型的 SMP 系统中取得相应和适当的性能提升。

英特尔数据面开发工具套件

英特尔数据面开发工具套件（Intel DPDK）是一套专门针对高速网络而设计的数据面库。它与英特尔架构（IA）产品家族中的所有处理器兼容，提供了一个可以用于从英特尔凌动（Atom）到最新的英特尔至强（Xeon）处理器的统一软件编程模式，适用于大量不同的网络应用程序和系统配置。

Intel DPDK 提供了一套可以从 Linux 用户空间使用的 API，从而提供了一个低开销的方法来替代传统的 Linux 系统调用。与原始 Linux 程序相比，通过这种方式创建的专门用途用户空间应用程序具有更好的性能扩展性。

这些库针对英特尔处理器进行了优化，包括用于特定硬件初始化和配置的环境抽象层（EAL）、用于缓冲区分配和解除分配的缓冲区管理、使用无锁环以提高弹性并避免通信任务间延迟的环管理、用于数据包和数据流查看和分类的流分类、以及采用轮询、无中断机制从网络接口卡接收数据帧的轮询模式驱动程序（PMD）和其他大量的实用工具，例如定时器、日志以及调试功能（图 1）。

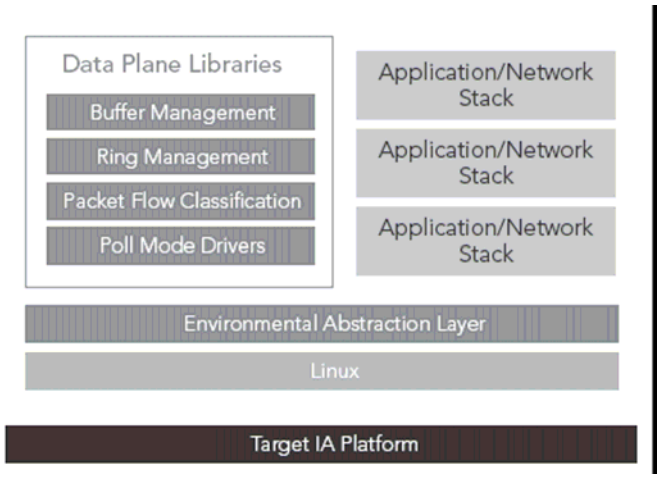

图1：Intel DPDK

Intel DPDK 用户空间库和 Linux

Linux 和 Intel DPDK 应用程序位于用户空间中，并使用数据面库来接收和发送数据包，在进行常规数据面处理时绕过了 Linux 内核。Linux 内核对 Intel DPDK 应用程序的处理和其他用户空间应用程序一样；它的编译、链接和载入都是以常规方式进行的。该应用程序作为一个单一的 Linux 进程启动，使用 Pthread 线程实现并发处理，并通过 Pthread 亲和性将每个线程附着到指定的内核。

为了理解采用 Intel DPDK 方式带来的好处，我们可以回顾一下传统的 Linux 是如何处理网络流量，以及用户应用程序一般是如何与 Linux 内核交互的。Linux 等通用操作系统在处理各种应用程序时不带任何偏见。操作系统必须能够在各方利益间进行平衡，并提供足够的安全机制来防止应用程序相互干扰或对内核造成影响。这些安全机制包括明确地划分应用程序和内核之间的界限，并设置一套规则来管理它们之间的交互作用。虽然 Linux 的内核及其处理网络流量的方式有了很多改进，但是它毕竟还是一个通用操作系统，有着这种操作系统内在的优势和缺点。

另一方面，Intel DPDK 则是针对专门用途的数据 I/O 密集型网络应用程序而设计。这些库可以帮助应用程序有效地接收和发送数据，并为更复杂网络软件的开发提供基础构件。这种方式使软件开发人员可以关注于应用程序本身，而不必花费精力来确定如何配置和精细调节操作系统以提高性能。与此同时，虽然这些库的应用是针对英特尔架构进行优化的，但是 Intel DPDK 并没有引入或向应用程序强加任何特别的硬件范式。相反，可以将 Intel DPDK 视为一个工具包，应用程序开发人员可以利用它来提取、提供实时软件开发人员所熟悉的性能增强技术优化实现，例如轮询或使用无锁环和零拷贝缓冲，所有这些都可以在用户空间中提供，同时避免了内核的常见问题。

在典型的网络场景下，数据包将由网络接口卡接收，然后进行分类并生成规定的动作，并对数据包付诸实施。

无论应用程序是路由器、防火墙、还是内容感知的应用程序（例如应用程序代理），如何在接收到流量时进行快速处理都是首要的问题。目前几个 G 的网速已经非常普遍，因此我们必须认识到，即使是在 1G 以太网连接中，用于处理数据包的时间也只有短短的 672 纳秒（对于 2.4GHz 处理器来说为 1613 个 CPU 周期），这对于任何通用操作系统都是一个挑战。虽然性能可能取决于多种因素，不过最近对英特尔平台上的本地新型 Linux 堆栈的测量表明，小型数据包的转发速度大约为 1Mpps/核心，或者是 667Mbps。要实现更高的合计线路速度，或者为了以线路速度支持更加高级的应用，需要采取不同的方式。

在传统的 Linux 模式下，系统接收数据包和将数据包发送出系统的过程占了包处理中很大一部分时间，这可能会出乎有些人的预料。换句话说，即使用户空间应用程序什么都不做，而只是将数据包从接收端口传送到发送端口，那么仍然会花费大量的处理时间。为了展示与运行 Linux 应用程序相关的基线成本，请考虑一下当用户空间应用程序接收和发送数据帧时会发生什么情况（图 2）。

当网卡从网络接收到一个数据帧后，会使用直接内存访问（DMA）将数据帧传送到针对这一目的而预先分配的内核缓冲区内，更新适当的接收描述符环，然后发出中断通知数据帧的到达。操作系统对中断进行处理，更新环，然后将数据帧交给网络堆栈。网络堆栈对数据进行处理，如果数据帧的目的地是本地套接字，那么就将数据复制到该套接字，而拥有该套接字的用户空间应用程序就接收到了这些数据。

进行传输时，用户应用程序通过系统调用将数据写入到一个套接字，使 Linux 内核将数据从用户缓冲区复制到内核缓冲区中。然后网络堆栈对数据进行处理，并根据需要对其进行封装，然后再调用网卡驱动程序。网卡驱动程序会更新适当的传输描述符环，并通知网卡有一个等待处理的传输任务。网卡将数据帧从内核缓冲区转移到自己内置的先进先出（FIFO）缓冲区，然后将数据帧传输到网络。接着网卡会发出一个中断，通知数据帧已经成功传输，从而使内核释放与该数据帧相关的缓冲区。

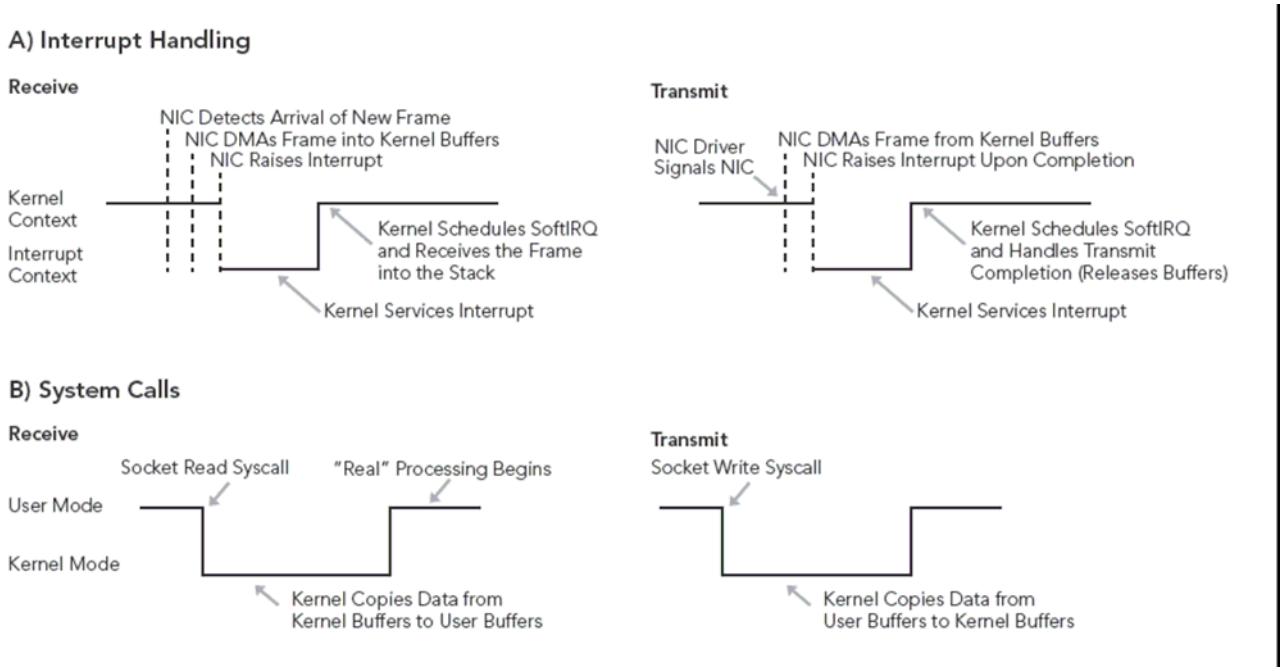

图2：接收和传输数据帧

虽然这只是一个极其简单的描述，但是仍然突出说明了处理过程中一些不属于应用程序处理的部分，而这些工作可以被视为间接开销（至少从应用程序的角度来说是这样）。

- **中断处理：**这包括在接收到中断时暂停正在执行的任务，对中断进行处理，并调度 softIRQ 处理程序来执行中断调用的实际工作。随着网络流量负荷的增加，系统将会花费越来越多的时间来处理中断，当流量速度达到 10G 以太网卡的线路速度时就会严重影响性能。而对于有着多个 10G 以太网卡的情况，那么系统可以会被中断淹没，对所有的服务产生负面影响。
- **上下文切换：**上下文切换指的是将来自当前执行线程的寄存器和状态信息加以保存，之后再将来自被抢占线程的寄存器和状态信息加以恢复，使该线程能够从原先中断的地方重新开始执行。调度和中断都会引发上下文切换。
- **系统调用：**系统调用会造成用户模式切换到内核模式，然后再切换回用户模式。这会造成管道冲刷并污染高速缓存。
- **数据复制：**数据帧会从内核缓冲区复制到用户套接字，并从用户套接字复制到内核缓冲区。执行这一操作的时间取决于复制的数据量。
- **调度：**调度程序使每个线程都能运行很短的一段时间，造成多任务内核中并发执行的假象。当发生调度定时器中断或在其他一些检查时间点上，Linux 调度程序就会运行，以检查当前线程是否时间已到。当调度程序决定应该运行另一个线程时，就会发生上下文切换。

虽然其中一些开销可以分摊到多个到达的数据帧上，不过大体上，其成本很容易就能达到数百个 CPU 周期/帧，这造成的区别就是仅能以线速度的很小一部分进行处理，而不是以完全的线速度进行处理。

为了解决这些问题，Intel DPDK 为部署的应用程序和网络堆栈提供了一个框架，可以直接与硬件工作来处理网络流量，从而减少中断、上下文切换和数据复制操作。应用程序和网络堆栈位于用户空间内，并且调用 Intel DPDK 库来提供数据面功能（包括将接收到的数据包转发给 Linux 进行异常处理）而不是向 Linux 内核提出系统调用。Intel DPDK API 使应用程序在初始化的时候留出内存，包括缓冲池、描述符环和无锁队列。

通过在一开始就分配缓冲区，应用程序就可以避免在常规数据面处理的时候向 Linux 内核请求缓冲区以及相关的开销。从 Linux 内核的角度来看，Intel DPDK 应用程序拥有那块内存，因此内核并不关心该内存以后是如何使用的。在正常的操作中，应用程序根据需要调用相应的 Intel DPDK API，从这一内存中分配和取消分配缓冲区。再加上 Intel DPDK 启用的网卡驱动程序，数据帧就可以在接收时被直接放置到应用程序缓冲区中，并且在发送时直接从应用程序缓冲区中获取，而无需内核的干预，从而提供了一个零拷贝的数据面环境。这样的实际效果就是，Intel DPDK 使数据面的处理能够绕过 Linux 内核（图 3）。

在概念上，Intel DPDK 应用程序拥有自己的运行时环境（RTE），而运行时环境在 Linux 上方在每个 Intel DPDK 核心上工作。对于每个 Intel DPDK 核心，运行时环境都提供了一个低开销、紧凑型的环境，应用程序可以在这一环境中执行，并包含一个“运行直到完成”的调度回路，该回路在应用程序需要在该核心上执行的所有工作之间循环。

该回路的一个功能就是持续地检查网卡有没有接收到数据帧，从而消除了处理网络活动中断而引起的开销。调度回路的组成部分，包括每个任务的服务频率，都完全取决于应用程序（图 4）。

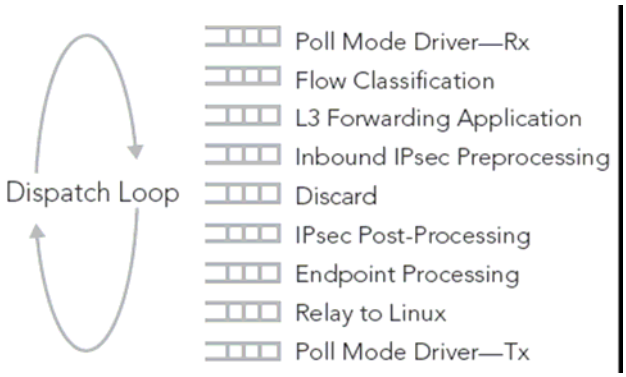

图 4：调度回路示例

在这个简单的例子中，创建了多个任务来处理网络流量。需要注意的是，在这一场景中，一个任务仅仅指的是执行特定功能的一小块工作，而不是指的操作系统任务。

轮询模式驱动程序检测到数据帧，并将其转交给流量分类任务进行分类。基于查看结果，数据帧可能被转发、丢弃、转交给 IPsec、传递给 Linux、或者被发送进行端点处理。每个任务的输入通过进入一个多生产者、单消费者的无锁队列来实现，提供任务间的异步耦合。

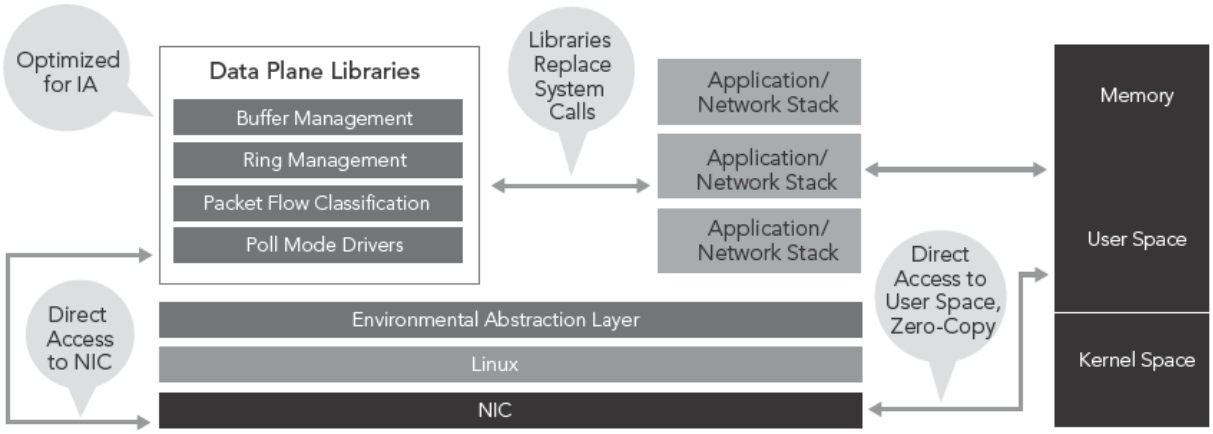

图3：Intel DPDK模型

现在，当网卡接收到来自网络的数据帧时，会将该数据帧移动到针对这一目的而预先分配的用户空间缓冲区，并更新适当的接收描述符环。应用程序通过轮询的方式检测到新数据帧的到达，然后开始对数据帧进行处理。在这一过程中没有拷贝操作。

进行传输时，用户应用程序会在用户空间缓冲区内构建数据帧，更新传输描述符环，并通知网卡有等待处理的传输。网卡将数据帧直接从用户空间移动到自己内置的先进先出缓冲区，然后将数据帧发送到网络。应用程序对网卡进行轮询，从而检测到传输完成，并释放与该数据帧关联的缓冲区。

表 1 给出了 Intel DPDK 组件的列表和描述。

表 1：Intel DPDK 组件

Intel DPDK 组件	描述
环境适应层	环境适应层（EAL）提供了一套 API，主要用于系统的初始化，获取核心数量和线程等配置信息，发现外围设备互连（PCI）设备，设置超大页面，留出与高速缓存相应的内存、缓冲区和描述符环，初始化轮询模式驱动程序（PMD），并在其他核心上生成线程。
缓冲区管理	缓冲区管理 API 用于获取内存和创建缓冲池，在数据面处理过程中可以从中分配缓冲区，从而加速缓冲区的实时分配和解除分配的过程。缓冲区与高速缓存相应，并按照核心进行分配，因此在大多数情况下不需要互斥锁。
环管理	环管理 API 针对单个或多个生产者、单个消费者的入列和出列操作提供了无锁的实现，支持大批量操作以减少开销。针对背压提供了高和低水线阈值。
流分类	流分类库为 n 元组（n-tuple）精确匹配和最长前缀匹配（LPM）提供了针对英特尔架构优化的查找，并且可以被同步或异步调用。由于表的尺寸比较大，而且可能会出现“冷”高速缓存，因此在软件中要高速进行表查找有着先天的困难。为了克服这一困难，算法利用了预取技术来为高速缓存“热身”。在大多数情况下，一个 IPv4 LPM 查看仅需一次内存访问。
轮询模式驱动程序	轮询模式驱动程序（PMD）提供了一个无中断的机制，能够以高速度发送和接收数据帧，并且在数据进出应用程序缓冲区的移动中实现了零拷贝操作。

Intel DPDK 和 Linux SMP

在这一模型中，主核上的 Intel DPDK 应用程序保留并初始化与高速缓存相应的内存、检测设备、设置超大页面、并生成其他核上的应用程序线程。线程被成功地创建和初始化后，主核将设置设备和核心间的映射，包括在适当的情况下使用无锁队列。为了避免缓冲区争用，每个线程都有自己的缓冲池高速缓存，并根据需要进行补充。

Linux 处理各个核上应用程序线程调度的所有方面，但是为了实现最佳的性能，每个应用程序线程都应当被锁定到特定的硬件线程。Intel DPDK 通过 Pthread 亲和性来实现这一点（图 5）。

在六核 2.4 GHz 英特尔至强处理器 E5645 上以这种模式运行 Intel DPDK 时，对于使用六个核中的四个核、每个核一个线程、四个 10G 以太网端口的情况，使用流分类的 64 字节数据包的 IP 第三层转发性能达到了 35.2Mpps。

当在六核 2.0 GHz 英特尔至强处理器 L5638 上使用最长前缀匹配（LPM）时，对于使用六个核心中的四个核心、每个核一个线程、四个 10G 以太网端口的情况，64 字节数据包的 IP 第三层转发性能达到了 41.1Mpps。这比原始 Linux 的性能差不多提高了十倍（原始 Linux 在双处理器六核 2.4GHz 模式下的性能为 12.2Mpps）。

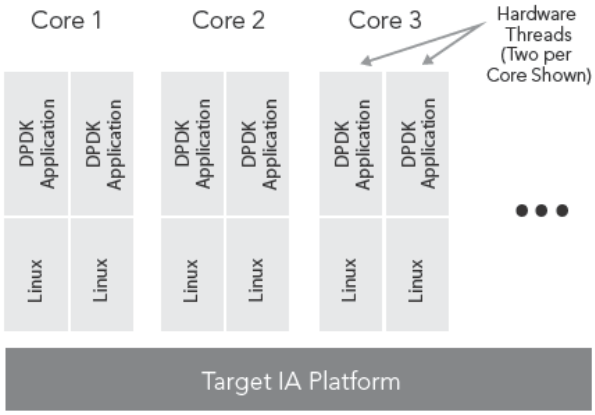

图5：Intel DPDK和Linux SMP

这种方式是针对开发了自己网络堆栈并希望在多核平台上快速展示主路径性能的客户而定制的。如果堆栈还不是单线程的，那么可能会需要一些额外的工作使堆栈适应到 Intel DPDK 单线程环境。如果对英特尔架构和优化有着深入的了解，就有助于客户将自己的软件移植到最新的英特尔处理器和硬件目标机上，从而能够从高速发展的处理器路线图中受益。

虽然 Linux 系统解决了很多基础设施问题，例如引导、使用共享内存、以及在 SMP 中进行调度，但是还有一些基础的设计问题需要考虑。数据面与 Linux 的分离取得了更好的数据面性能，但是需要定义和解决数据面的管理问题。例如，路由更新是在哪里进行处理的，以及转发表是如何变化的？Linux 和数据面之间的互动是怎样的，以及异常路径处理是如何处理的？Linux 是否看管数据面接口，或者 Linux 是否管理自己的一套接口？

由于这种方式采用了开放式的管理面设计，因此客户能够实施自己专有的管理方案。

网络加速平台与 Intel DPDK

风河网络加速平台是一个现成、可扩展的多核 AMP 解决方案，将加速的数据面与风河 Linux 管理面集成。加速的数据面直接在裸机上运行，包含一个或多个核心，每个数据面核心都会调用一个网络加速引擎（NAE）。网络加速引擎提供了高性能的网络数据路径，替代 Linux 内核和堆栈进行数据面处理。当部署在英特尔平台上时，网络加速平台解决方案利用了 Intel DPDK 在数据面中进行英特尔架构特定的优化，并且灵巧地集成了 Linux 进行管理控制。Intel DPDK 为底层处理和数据包 I/O 提供了与硬件打交道的功能，而网络加速平台则在 Intel DPDK 上方提供了一个优化的网络堆栈实现。网络加速平台的网络堆栈是完整的，并不限于主路径用例或静态路由和寻址。实际上，网络加速平台接手了 Intel DPDK 遗漏的部分。与 Linux SMP 解决方案相比，网络加速平台减少了共享内存争用并降低了对 Linux 内核的依赖性，从而具有更好的扩展性。与采用 Intel DPDK 和 Linux SMP 的解决方案相比，网络加速平台提供了一个完整的网络堆栈而不是库，客户从中可以开发出自己的堆栈（图 6）。

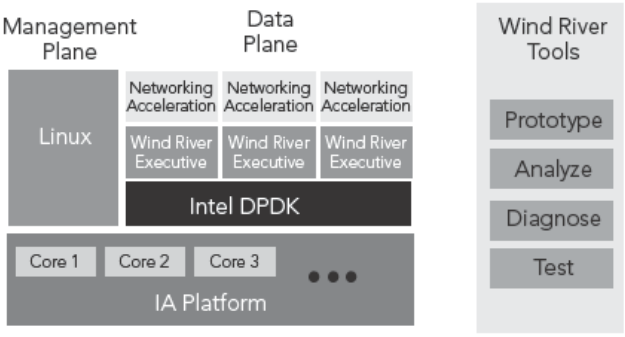

图6：风河网络加速平台

网络加速平台和Intel DPDK数据面包含了风河执行程序（Wind River Executive），这是一个针对网络应用程序的精简运行时环境。在风河执行程序的上方是一个经过验证、功能齐全、高度模块化的网络堆栈，针对单线程环境进行了优化，可以加速网络层服务和功能，例如包转发、IPsec、IP 分片和重组、虚拟局域网（VLAN）、以及要求完全 TCP/UDP/SCTP 协议终结的应用程序。用户可以根据所需的组件对网络加速引擎堆栈进行配置和构建，从而生成功能完整或精简型的堆栈。此外还提供了多种风河工具，可以用于对管理面和数据面上的软件进行开发、运行时分析和调试。使用提供的内核模块和管理应用程序，从 Linux 核心对堆栈进行管理。

虽然有些专门的应用程序可能会要求对堆栈进行修改，不过大多数软件开发人员应该都能使用熟悉的套接字 API 在堆栈上方直接创建应用程序。网络加速引擎支持用于协议终结的流套接字和数据报套接字，以及允许应用程序访问网络上到达原始数据包的原始套接字。所有的套接字都是零拷贝的，这意味着当数据从网络堆栈到应用程序以及从应用程序到堆栈时，没有拷贝操作发生。

如果没有管理面控制，那么任何系统都是不完整的。就像 Linux 被用来管理网络加速引擎的网络堆栈一样，软件开发人员也可以在 Linux 上编写应用程序来管理他们的数据面应用程序。为了促进管理面和数据面之间的通讯，网络加速平台支持风河公司的多核进程间通信（MIPC）协议，用于内核间的消息传递。网络加速引擎通过 MIPC 将加速面无法处理的数据包传递给管理面。同理，在网络加速引擎堆栈上开发的任何应用程序都可以使用 MIPC 套接字与管理面上的相应应用程序进行通信（图 7）。

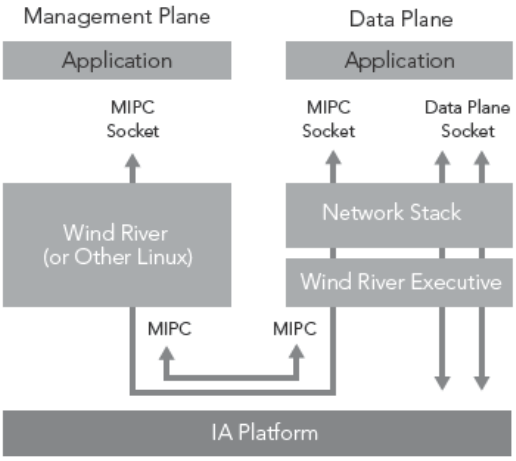

图 7：MIPC 上的管理通讯

网络加速引擎的核心组件是风河执行程序，这是一个运行时环境，包含了一个“运行直到完成”、基于轮询的调度程序。该调度程序使用 Intel DPDK PMD 与网络接口卡工作。风河执行程序在紧密的环路中操作，检查要执行的工作，利用 Intel DPDK 库进行低开销的运行时缓冲区分配和解除分配，并且实现无锁队列以适应工作的弹性，而不受中断处理和上下文切换的影响。

接收端缩放和数据流限定

如果网络接口卡有多队列和接收端缩放（RSS）功能，那么网络加速引擎无需锁就可以向网络接口卡发送和接收数据包。网卡从网络链路上接收到数据包后，将它们放到基于可配置（通常为 n 元组）散列的聚合流队列。每个网络加速引擎都处理自己的聚合流队列的数据包，因此多个核心可以处理来自同一链路的数据包，而无需争用锁或有其他共享内存方面的考虑。类似的，在进行传输时，每个网络加速引擎都将处理的数据包加入到自己按网卡传输的队列，也不会与其他内核发生争用的现象（图 8）。

以这种方式可以限定内核的数据流，这意味着每个网络加速引擎通常只需处理数据流的一个子集。这样当数据包达到时，数据流的上下文和连接状态信息已经驻留在本地高速缓存中的概率就会增加，并减小为了留出空间而将一些上下文挤出而造成的高速缓存“抖动”的频率。

高速缓存考虑因素

在理想情况下，当数据包到达系统时，所有处理该数据包所需的信息最好都已经在内核的本地高速缓存中。我们可以设想一下，如果当数据包到达时，查找表项目、数据流上下文、以及连接控制块都已经在高速缓存中的话，那么就可以直接对数据包进行处理，而无需“挂起”

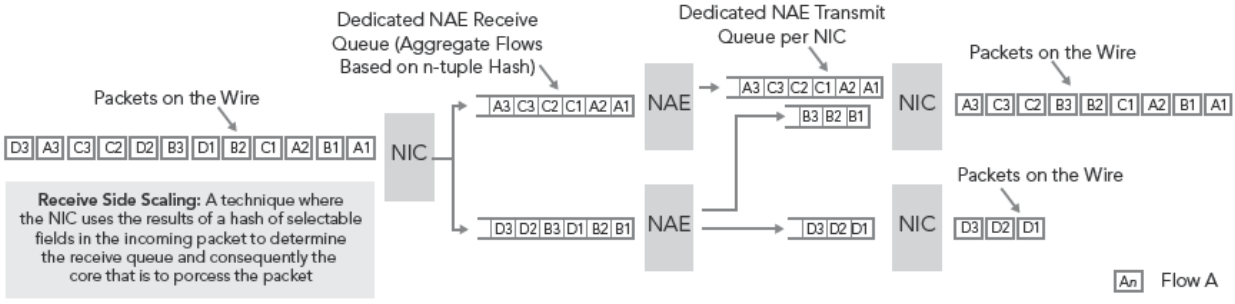

图 8：使用 RSS 和多队列来限定数据流

并等待外部顺序内存访问完成。当然，实际情况和理想情况有很大差异。在处理上万个数据流的多核系统中，所需的信息已经在某个特定内核的高速缓存中的概率非常小，而这中机率往往取决于该内核的数据包处理历史。

虽然没有解决方案能够保证所需的信息在事先已经置入高速缓存，不过带有 Intel DPDK 的网络加速平台采用了多种技术来提高这一概率：

- 网络加速引擎是单线程的，这意味着高速缓存不会因为中断、内核和用户空间之间的上下文切换而受到污染。
- 网络加速平台利用了数据流限定技术，将特定的数据流限定到特定的核心。
- 只要有可能，网络加速引擎会在访问之前就将所需的数据预读到高速缓存，从而使高速缓存被“预热”。

无锁实现

为了在常规数据路径处理中尽量减少锁定数据结构的情况，网络加速引擎大量使用了本地数据，例如本地流高速缓存和本地转发信息库。这并不是说网络加速引擎不会对共享数据进行操作。确切地说，而是采用了数据访问分层结构，首先查询本地数据，然后再逐渐扩大数据访问的范围。即使是缓冲区也是首先从本地存储区开始分配，然后才会向全局管理的缓冲池进行请求。而本地缓冲池则根据需要从全局缓冲池进行补充。

当设置描述符环时，网络加速引擎会在可能的情况下使用 Intel DPDK 针对英特尔架构优化的无锁环。与网卡进行交互时，网络加速引擎还利用了网卡的多队列能力。

此外，所有的数据结构都是根据高速缓存调整的，从而最大化地提高高速缓存线效率并减少高速缓存-线路争用。在带有非均匀访存模型（NUMA）的系统中，内存的分配是 NUMA 认知的，即尽可能地从本地节点分配内存。

超大页面

支持 HugeTLB，使用 2MB 和 1GB 页面。超大页面减少了旁路转换缓冲区(TLB)击不中的情况，并且减少了处理 TLB 击不中所需的工作量。

管理面

构建多核解决方案的挑战并不仅仅限于数据面的性能。性能要求往往是促使人们移植到多核解决方案的因素，但是数据面的性能仅仅是其中的一部分。完整的解决方案应当包含一个可行、集成的管理面，可以无缝地管理系统及其接口，给人一种观感，可以掩盖底层架构异构和非对称的方面。

对管理面和数据面之间的通讯进行管理是非常复杂的。这方面的挑战包括数据面表格的动态更新、单独核心状态的重新设置和恢复、以及与 Linux 的无缝集成等。

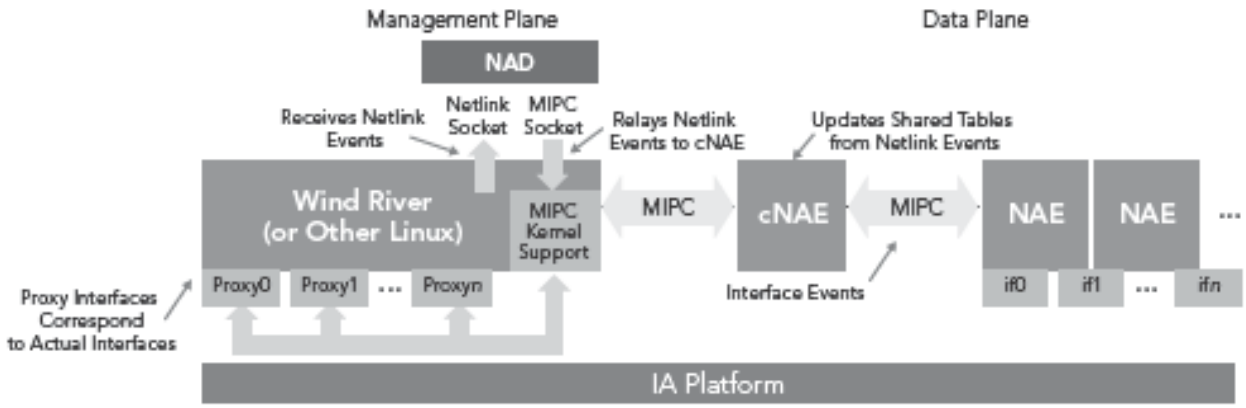

图9：管理面架构

网络加速平台的管理面（图 9）包括一个风河 Linux 操作系统或其他标准 Linux 版本，该系统与一个叫做网络加速后台程序（NAD）的用户空间应用程序一起工作，还有一个被称为是控制网络加速引擎（cNAE）的特殊网络加速引擎，以及专门的代理接口驱动程序。NAD、cNAE 和驱动程序一起协作，使 Linux 的网络操作看起来就像单 OS 的系统一样，其透明的工作使数据包能够被 Linux 发送和接收，并且还包括了 ifconfig 等命令，用于控制以太网接口和分配地址。

代理接口

代理接口的概念是网络加速平台管理架构的基础。由于 Linux 和网络加速引擎无法同时拥有同一个网络接口，因此 Linux 为网络加速引擎使用的每一个实际的网络接口指定了一个代理接口。代理接口在 Linux 和实际接口之间起到一个管道的作用，对于 Linux 来说就像一个普通的设备驱动程序一样。不过代理接口驱动程序并不是将数据包直接发送到物理链路，而是通过 MIPC 发送到 cNAE，并由 cNAE 将数据包发送到网络链路中。

同理，当一个数据包到达物理端口并需要由管理面进行处理时，网络加速引擎会通过 MIPC 将数据包传递给 Linux 内核上的代理接口驱动程序。然后代理接口驱动程序通过代理接口将数据包发送给 Linux 堆栈，就好象数据包是直接从实际物理端口接收的一样。

代理接口驱动程序被打包为一个内核模块，提供了标准的以太网功能，例如支持 ifconfig、改变 MTU 尺寸、以及进入混合模式，使 Linux 能够控制网络加速引擎接口。

控制网络加速引擎

控制网络加速引擎（cNAE）就像一个主控网络加速引擎一样，除了承担常规的网络加速引擎工作之外，还会执行网络加速引擎初始化的很多工作，包括从管理面获取配置信息。cNAE 提出硬件端口并报告端口能力。然后管理面会提供哪些网络加速引擎对哪些端口轮询的信息以及其他各种信息，使 cNAE 能够分配和初始化转发表等共享数据结构。

在运行时，cNAE 会执行两个关键功能：

- 为代理接口模式提供支持，处理Linux数据包传输请求和来自管理面的各种命令，以及将接口状态变化传送回Linux。
- 管理转发表等共享数据，根据需要使条目作废并通知网络加速引擎更新它们的本地副本。

由于 cNAE 事件并不经常发生，而且 cNAE 只有在需要的时候才会运行，因此对数据面的影响应该会很小。不过，如果有必要的话，可以将 cNAE 配置为仅用于控制处理。

网络加速后台程序

网络加速后台程序的作用是嗅探相关的事件并透明地传递给 cNAE。网络加速后台程序是一个运行在 Linux 内核上的用户空间进程，登记和监视链路层和网络层的 netlink 事件，并将这些事件通过 MIPC 套接字传递给 cNAE。这包括路由表更新和邻居状态变更等事件。

这种 Linux 控制面的镜像使标准 Linux 控制面机制能够与网络加速引擎集成，从而不必采用并行的控制面结构。

风河Workbench

对多核系统进行调试是一项繁重的工作。虽然架构提供的并发性带来了性能上的优势，但是也带来了更多的死锁、内存破坏和同步问题。现在人们已经意识到了工具对于多核软件开发的重要性。风河 Workbench 是一个完整的集成工具套件，可以用于调试、代码分析、高级可视化、根本原因分析和测试。风河 Workbench 针对单核和多核系统而设计，包含了一个调试器、内存分析器、性能调节器、数据监控器、片上调试、以及其他多种功能，可以为多核软件提供深入的分析，确定软件的同步和其他问题。

优点

与那些选择直接利用 Intel DPDK 库来创建自己专业的数据面和网络堆栈的用户不同，网络加速平台的用户希望的是一个完整的、打包的、可扩展的解决方案，并以此来开发他们的应用程序。这些用户的特长在于应用程序，而且他们也没有时间或资源来了解更多的有关硬件、驱动程序、网络堆栈、IPC、以及多如牛毛的基础设施构件的相关知识。引入了高度优化的网络加速平台网络堆栈后，这些用户就不必再创建自己的网络解决方案。Linux 管理面的集成使他们能够使用熟悉的方法来管理接口。通过熟悉的套接字 API，他们可以快速开发出利用多核加速优势的应用程序。由于网络加速平台隐藏了硬件的细节，因此移植到未来硬件架构的工作也将变得更加简单。

风河网络加速平台包括风河 Linux 和代理接口驱动程序、NAD 应用程序、cNAE、以及网络加速引擎数据面，所有这些组件都打包到一起，并使用风河 Workbench 进行了测试。这一完整的一站式解决方案不仅可以提高系统性能，而且还提供了一套先进的多核工具来提高生产率并加速面市时间。

设计选项比较

每一种设计都有其内在的优势和劣势。适合某一种情况的设计选项可能并不适合另一种情况。传统的 Linux SMP 在性能和扩展性之间进行了折中以提供便捷性。Intel DPDK 和风河网络加速平台虽然都提供了更好的性能和扩展性，但是却需要额外的开发工作。表 2 列出了进行方案选择时的一些主要考虑因素。

表 2：多核软件设计选项

设计选项	目标用户	性能	扩展性	数据面	管理面	网络堆栈	套接字 API	IPC	工具和调试
传统的 Linux SMP	适合那些对SMP的性能和扩展性表示满意，而且尚不需要更高性能的用户。	一般	有限	包括	包括	多核安全	有	不可用	很多
Intel DPDK 和 Linux SMP	适合已经开发出了一个数据面，并且希望将其与Linux基础设施集成的用户，或者是已经开发出了专有的网络解决方案并且只是想利用数据面开发工具套件进行英特尔架构优化的“自主开发型”用户。	对于低层的数据包移动来说性能很高	很高	用户提供	用户提供	用户提供	无	不可用	很多（和Linux一样）
风河网络加速平台和Intel DPDK	完整、现成的产品，包括一个与Linux管理面预先集成的高性能网络堆栈；适合希望关注于应用程序而不是底层基础设施的用户。	对于第2、3、4和更高层的协议处理来说性能很高	很高	包括	包括	针对多核和并行执行进行了优化，fastpath 协议处理	无阻塞	风河 MIPC	风河 Workbench

结论

希望利用多核平台优势的网络设备提供商们将不再受限于随核心数量增加扩展性不佳的传统 SMP 设计。本文介绍了两种新的数据包处理方法，比传统 Linux SMP 提供了更好的扩展性，从而可以创建出能够随着产品线扩展的通用架构。

Intel DPDK 是一套 Linux 用户空间库，为用户提供了底层的构件来创建自己的高性能数据面应用程序和网络堆栈。这一解决方案要求额外的开发工作，适合希望开发自己专有数据面的用户。

风河网络加速平台利用了 Intel DPDK 库的性能，而且与单独使用 Intel DPDK 相比，获得了高得多的网络协议加速效果，实现了针对 IP 网络、TCP/UDP/SCTP 协议终结以及 IPsec 和 VLAN 的良好效果。通过这一现成、完整、由 Linux 进行管理的解决方案，用户就可以集中精力来构建自己的应用程序，而无需创建用来支持这些应用程序的基础设施。

要了解有关风河网络加速平台的更多信息，请访问网址：www.windriver.com 或是中文网站 www.windriver.com.cn。要了解有关英特尔数据面开发工具套件的更多信息，请访问网址：www.intel.com/go/dpdk