Linux 环境下 DPDK 中文入门文档

Release 17.02.0

目录

Linux 环境下 DPDK 入门 ... 1

1 引言 ... 2

2 资料集 ... 2

3 系统需求 ... 2

3.1 x86 平台的先决条件 BIOS 设置 ... 2

3.2 编译 DPDK. 3

3.3 运行 DPDK 程序 ... 3

4 编译 DPDK 源码 ... 5

4.1 安装 DPDK 和浏览源码 ... 5

4.2 安装 DPDK 目标 (target) 开发环境 ... 6

4.3 浏览安装好的 DPDK 环境 target. 6

4.4 加载 DPDK 用户态 IO 模块 ... 7

4.5 加载 VFIO 模块 ... 7

4.6 从内核模块绑定和解绑网卡 ... 7

5 编译和运行例子程序 ... 8

5.1 编译例子程序 ... 8

5.2 运行例子程序 ... 9

5.2.1 程序使用的逻辑核 ... 9

5.2.2 程序使用的大页内存 ... 10

5.3 其它例子程序 ... 10

5.4 其它的测试程序 ... 10

6 开启其它功能 ... 10

6.1 高精度定时器 (HPET) 功能 ... 10

6.1.1 BIOS 支持 ... 10

6.1.2 linux 内核支持 ... 11

6.2 非 root 用户运行 DPDK 程序 ... 11

6.3 电源管理和省电功能 ... 11

6.4 使用 DPDK 的 CPU 亲和性减少上下文切换的开销 ... 12

6.5 加载 DPDK KNI 内核模块 ... 12

6.6 使用 linux IOMMU 透传来在 INTEL VT-d 虚拟化环境下运行 DPDK. 12

6.7 40G 网口高性能小包处理 ... 12

6.7.1 使用 16 个字节的 RX 描述符 ... 12

6.7.2 高性能和报文时延间的均衡 ... 12

7 快速安装启动脚本 ... 12

7.1 脚本结构 ... 13

7.2 用例 ... 13

7.3 应用程序 ... 15

8 怎么在 intel 平台上获得网卡的最好性能 ... 17

8.1 硬件和内存需求 ... 17

8.1.1 网卡需求 ... 18

8.1.2 BIOS 设置 ... 18

8.1.3 Linux 内核启动参数 ... 18

8.2 运行 DPDK 前的配置 ... 18

8.3 获取 intel 网卡的最好性能例子 ... 19

1 引言

文档是安装配置 DPDK 操作说明，旨在帮助客户快速上手和运行程序。文档介绍了在 linux 开发环境下如何编译和运行 DPDK 程序，但是并不深入细节。

之前曾经尝试着翻译来着，当时因为要离职，一时兴起就想着翻译，翻译的太烂，现在重新翻译，一方面是方便新入职的同事能够快速入门，另外一方面是现在工作的需要，还有一方面是学习 KVM 想休息一下，看内存那块弄得头昏脑胀，而且后面的例子会涉及到虚拟话那块的。下一个将翻译样例那本书（以及在自己机器上运行可能遇到的问题），同时会参杂着介绍开发者手册中的一些个人理解。

2 资料集

下面列出了所有 DPDK 文档资料的建议读取顺序 :

l 版本说明：提供了各个版本相关的信息，包括支持的特性，限制，修复的 bug，出现的问题等等。也对频繁提到的问题以 FAQ 的方式做了回答。

l 入门手册 （本文的）：讲述了如何安装配置 DPDK；意在帮助开发者快速上手和运行程序。

l 开发者手册：

n 软件架构和（通过例子）在 linux 环境下使用它

n DPDK 的内容包括构建系统（包括在 DPDK 根目录下构建开发环境和程序使用的命令）和移植程序指南。

n 在已有软件和要新开发的软件中需要考虑到的优化。

还提供了一个专有术语表。

l API 索引：提供了 DPDK 函数，数据结构和其它开发用到的结构体的详细信息。

l 范例用户手册： 介绍了一系列的范例程序。 每一个章节介绍一个程序， 展示程序的特殊功能，说明如何编译，运行和使用范例程序。

3 系统需求

本章介绍了编译 DPDK 需要的安装包。

注意 :如果 DPDK 要运行在 intel 89xx 系统通讯芯片平台， 请翻阅对应该系列的 linux 入门手册。

3.1 x86 平台的先决条件 BIOS 设置

对于主流的平台，使用 DPDK 的基本功能并不需要专门的 BIOS 设置，然而，对于额外的功能像 HPET （高精度定时器），电量管理功能，以及在 40G 网卡上高性能小包处理，需要修改 BIOS 设置。修改设置的详细信息见第六章。

3.2 编译 DPDK

需要的工具：

注意：已经在 Fedora18 上测试运行过，在其它系统上安装命令和需要的安装包可能不一样，要知道其它版本 linux 发行版测试详细的细节，请查看 DPDK 版本说明。

l GNU make

l coreutils:cmp ，sed，grep，arch，等等。

l gcc ：4.9 版本或者是所有平台带的更新的版本。在一些 gcc 版本中，很多特殊的编译标志和链接标志默认是打开的，会影响到性能（比如，-fstatck-protector ）。请查阅对应版本的文档和执行 gcc –dumpspecs 。

l libc hreaders ，通常打包成 gcc-multilib(intel 64 位架构上是 glibc-devel.i686/libc6-dev-i386;glibc-devel.x86_64/libc6-dev ，IBM power 架构则是 glibc-devel.ppc64)

l 编译内核模块需要的内核头文件和源文件（kernel-devel.x86_64;kernel-devel.ppc64 ）

l 在 64 位机器上编译 32 位程序而外需要的库：

n glibc.i686, libgcc.i686, libstdc++.i686 and glibc-devel.i686 for Intel i686/x86_64;

n glibc.ppc64, libgcc.ppc64, libstdc++.ppc64 and glibc-devel.ppc64 for IBM ppc_64;

注意： x86_x32 ABI 库旨在 ubuntu13.10 以上版本或者最新 Debian 版本上支持。只支持 gcc4.9+ 版本。

l python ，要使用 dpdk 安装包中各种帮助脚本必需的 python 版本 2.7+ 或者是 3.2+ 。

其它的工具：

l intel 编译器 icc ：需要安装额外的库。在编译器安装文档看中 icc 安装说明。

l IBM powerlinux 高级工具链：这是一系列的开源开发工具和运行时库，能够让用户用到 IBM 最新的 power 硬件特性。要安装的话看 IBM 的官方安装文档。

l libpcap 头文件和库 （libpcap-devel ）编译和使用基于 libpcap 的轮询驱动。 这个驱动默认是禁用的，可以通过修改编译时的配置文件参数 CONFIG_RTE_LIBRTE_PMD_PCAP=y 来打开。

l libarchive 头文件和库在单元测试用打开源文件用到。

3.3 运行 DPDK 程序

要运行 DPDK 程序，在目的机器上需要一些定制化的东西。

3.3.1 系统软件

要求：

l 内核版本 >=2.6.34: 通过命令 uname –r 查看内核版本号

l glibc>=2.7 （要用到 cpuset 特性）：可以通过命令 ldd –version 命令查看 gcc 版本号

l 内核配置

在 Fedora 操作系统和其它通用发行版，例如 Ubuntu ，红帽企业版 linux，发行商提供的内核配置基本可以跑绝大多数 DPDK 程序。

对于在其它内核上构建 DPDK，下面几个选项需要能支持：

n UIO （用户态 IO）

n HUGETLBFS （大页内存）

n PROC_PAGE_MONITOR 支持

n HPET 和 HPET_MMAP 配置项必须打开，如果用到 HPET ，具体细节看第六章。

3.3.2 在 linux 环境下使用大页内存

对于大的报文内存池内存分配， 大页内存是必须的。 （前一章说过运行的内核必须支持大页内存）。使用大页内存来分配，即使是小部分页要用到，性能也能够得到提升。减少了 TLBmiss ，对应就减少了虚拟地址转物理地址的时间。使用小页 4K 内存页，很高频率的 TLBmiss 发生会导致性能低下。

申请大页给 DPDK 使用

大页内存的分配最好是在系统启动时或者是尽量在系统启动后越早越好， 这样可以减少分配的物理内存页物理地址碎片化。申请大页内存需要在内核启动时传递一个命令行参数。

对于 2MB 内存页，分配 1024 个 2MB 内存页参数如下：

```
hugepages=1024
```

其它尺寸的大内存页，例如 1G 页，必须指定申请的内存页个数和默认内存页的大小。例如分配 4 个 1G 的内存页，需要指定的参数如下：

```
default_hugepagesz=1G hugepagesz=1G hugepages=4
```

注意：大页内存的尺寸是 CPU 支持的大小来决定的。 intel 的机器上通过查看 CPU flags 来查看支持的大页内存尺寸。 比如 pse 代表支持 2MB 的内存页， pdpe1gb 代表支持 1G 内存页。在 IBM power 架构机器上，支持大页内存尺寸为 16MB 和 16GB 。

注意：对于 64 位的应用，如果平台支持，建议使用 1G 内存页。

在双插槽的 NUMA 系统中，申请的大页内存一般会平均在两个 socket 上分配（假定两个 socket 都有足够的内存）。

可以看 linux 内核源码树中 Documentation/kernel-parameters.txt 获取更多内核选项的细节。

备选项：

对于 2M 的页, 可以在系统启动后申请分配。 echo 想要分配的大页内存页个数到 /sys/devices 目录下的 nr_hugepages 文件。对于单 NUMA（非 NUMA ）节点的系统, 如下命令所示 （假设需要 1024 个 2M 页）：

```
echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

在一个 NUMA 机器上，分别在不同 node 上分配指定数目的内存页：

```
echo 1024 >
/sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages

echo 1024 >
/sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
```

注意： 1G 的内存页在系统启动后是无法申请。

DPDK 使用大页

一旦申请到大页内存，要想其被 DPDK 使用到，执行以下步骤：

```
mkdir /mnt/huge

mount -t hugetlbfs nodev /mnt/huge
```

要想是挂载即使是重启也永久有效，需要把下面一行加到 /etc/fstab 文件中：

```
nodev /mnt/huge hugetlbfs defaults 0 0
```

1GB 内存页，必须在挂载的时候将内存页的大小作为参数指定：

```
nodev /mnt/huge_1GB hugetlbfs pagesize=1GB 0 0
```

3.3.3LINUX 环境下 Xen Domain0 支持

现有的内存管理基于 linux 内核的大页机制。 在 Xen 虚拟机管理器上， 大页内存对非特权域（DomainU ）的支持意味着 DPDK 在客户机上可以像普通机器上一样跑。

然而，特权域（ domain0 ）并不支持大页内存。需要插入新的内核模块 rte_dom0_mm 以便绕过这个限制来申请和映射大页内存，通过 ioctl 来申请， mmap 来映射。

开启 DPDK 对 Xen Dom0 支持

默认情况下， Xen Dom0 模式支持在 DPDK 的 build 配置文件中是禁用的。要想支持 Xen Dom0 ，配置项 CONFIG_RTE_LIBRTE_XEN_DOM0 需要在编译时改成 y。

此外，配置项 CONFIG_RTE_EAL_ALLOW_INV_SOCKET_ID 也必须置成 y，以防万一收到错误的 socket ID 。

加载 rte_dom0_mm 模块

要在 Xen Dom0 中跑 dpdk 程序，rte_dom0_mm 模块必须带上 rsv_memsize 参数加载到运行的内核中。可以在编译的 DPDK 目标目录的子目录中找到该模块。该模块的加载如下使用 insmod 命令（假设当前处于 DPDK 编译的目标目录下）：

```
sudo insmod kmod/rte_dom0_mm.ko rsv_memsize=X
```

X 的值可以大于 4095MB 。

配置 DPDK 使用的内存

在 rte_dom0_mm.ko 模块加载完成后，用户必须配置 DPDK 使用的内存。将需要的内存大小 echo 到 /sys/devices/ 目录下的 memsize 文件中即可。 使用如下命令 （假设需要 2048MB 内存）：

```
echo 2048 > /sys/kernel/mm/dom0-mm/memsize-mB/memsize
```

用户也可以检查已经使用了多少内存：

```
cat /sys/kernel/mm/dom0-mm/memsize-mB/memsize_rsvd
```

Xen Dom0 不支持 NUMA 配置，所以参数 --socket-mem 对于 Xen Dom0 是无效的。

注意：内存大小值不能超过 rsv_memsize 值。

在 Xen Domain0 中运行 DPDK 程序

要跑 DPDK 程序需要额外提供一个命令行参数 --xen-dom0 。

4 编译 DPDK 源码

注意：本章写的部分安装过程也可以通过后续 <快速安装脚本 >一章中描述的脚本实现。

4.1 安装 DPDK 和浏览源码

首先，解压压缩包并移动到解压的 DPDK 源码目录：

```
tar xJf dpdk-<version>.tar.xz

cd dpdk-<version>
```

DPDK 包含以下几个子目录：

- lib：DPDK 库的源码
- drivers：DPDK 轮询驱动的源码
- app：DPDK 应用程序的源码（自动化测试）
- examples：DPDK 应用程序样例源码
- config，buildtools，mk：框架相关的 Makefiles，脚本和配置文件

4.2 安装 DPDK 目标 (target) 开发环境

DPDK target 格式是：

ARCH-MACHINE-EXECENV-TOOLCHAIN

详解：

- ARCH 可以是：i686,x86_64,ppc_64
- MACHINE 可以是：native,power8
- EXECENV 可以是：linuxapp,bsdapp
- TOOLCHAIN 可以是：gcc,icc

将要被安装的 target 环境取决于要安装的机器上的包是 32 位还是 64 位。可以编译产生的 target 在 DPDK/config 目录下定义了。不要用 defconfig_ 前缀，就是编译的时候不要带上这个前缀。

注意：配置文件是 RTE_MACHINE 下的最优配置集合。在配置文件中，RTE_MACHINE 被定义为 native，意味着这是针对本机环境下最佳优化的编译。对于这些设置的更多信息，可以采用的值，看 DPDK 开发者手册。

当使用 ICC 编译器时，下面的命令分别对应 32 位和 64 位环境使用。注意这个 shell 脚本更新$PATH 环境变量，因此换个会话就失效了。编译安装的目录可能是不同的：

```
source /opt/intel/bin/iccvars.sh intel64

source /opt/intel/bin/iccvars.sh ia32
```

安装和生成 target，在 DPDK 的根目录下使用 make install T=<target> 命令。

例如，使用 icc 编译 64 位 target：

```
make install T=x86_64-native-linuxapp-icc
```

使用 gcc 编译 32 位的：

```
make install T=i686-native-linuxapp-gcc
```

只是做创建 target 的准备，而不是编译，例如，在编译之前需要手动修改配置文件的，使用命令 make config T=<target> ：

```
make config T=x86_64-native-linuxapp-gcc
```

警告：

任何内核模块，比如 igb_uio，kni 等，必须编译时的内核和要运行时的内核一致。如果 DPDK 不是在编译产生的目标机器上运行，RTE_KERNELDIR 环境变量必须指向要运行 target 机器对应内核版本。

一旦目标环境创建完成，用户可能需要进入目标环境目录，修改源码并重新编译。用户也可能修改编译时的 DPDK 配置项，通过编辑 build 目录下的 .config 文件（这个文件是从顶层目录的 config 目录下 defconfig 文件拷贝到本地来的）：

```
cd x86_64-native-linuxapp-gcc
```

vi .config

make

此外， make clean 命令可以清除所有之前编译留下的文件。

4.3 浏览安装好的 DPDK 环境 target

一旦 target 创建完成就包含所有的苦，包括轮询驱动，和用户程序编译需要的 DPDK 环境头文件。此外， test 和 testpmd 程序在 build/app 目录下创建完成，可用于测试。 kmod 目录下包含需要加载到内核中的几个内核模块。

4.4 加载 DPDK 用户态 IO 模块

运行 DPDK 程序，一个合适的 uio 模块需要加载到运行中的内核。在很多例子中，标准的 uio_pci_generic LINUX 内核模块可以提供用户态驱动能力。该模块加载如下：

sudo modprobe uio_pci_generic

和上述不同的是， DPDK 提供了 igb_uio 模块来实现（在 kmod 目录下）：

sudo modprobe uio

sudo insmod kmod/igb_uio.ko

注意：对于缺乏对传统中断支持的设备，比如 VF 设备， igb_uio 模块可能需要被 uio_pci_generic 替换掉。

在 DPDK1.7 之前版本提供了对 VFIO 的支持，在支持 VFIO 的平台上 UIO 就不是必须的了。

4.5 加载 VFIO 模块

使用 VFIO 的 DPDK 应用程序， vfiopci 模块必须加载：

sudo modprobe vfio-pci

需要注意的是要使用 VFIO，你的内核必须支持才行。 VFIO 内核模块在 LINUX 内核 3.6.0 版本之后就囊括经来了，默认是有的。然而，请翻阅 linux 发行版文档确定一下。

使用 VFIO， BIOS 和内核都必须支持， BIOS 需要打开支持虚拟化的配置选项（例如 Intel? VT-d).

在非特权用户下运行 DPDK 程序，要正确的操作 VFIO，需要建立正确的权限设置。这个可以通过 DPDK 安装脚本实现（ dpdk-setup.sh ，位于 usertools 目录下）

4.6 从内核模块绑定和解绑网卡

在 1.4 版本， DPDK 程序不再自动解绑所有支持 DPDK 的网口，当网口绑定在内核驱动上且正在运行时。取而代之的是， DPDK 程序在运行前，必须将使用的所有网口绑定到 uio_pci_generic ，igb_uio 或者是 vfio-pci 模块上。 任何 linux 控制下的网口将被 DPDK 轮询驱动忽略，不能被 DPDK 程序使用。

警告： DPDK 默认不再在启动时自动解绑内核驱动管理的网口。任何 PDDK 程序使用的网口必须在程序运行前解除 linux 控制，绑定到对应的驱动上（ uio_pci_generic,igb_uio 或者是 vfio-pci ）.

将绑定到 DPDK 的网口解绑并交给 linux 控制，可以使用 usertool 子目录下的 dpdk_nic_bind.py 脚本。这个脚本可以展示当前系统中所有网口的状态， 可以将其从不同的内核模块间解绑和绑定， 包括 uio 和 vfio 模块。 下面是使用脚本的例子。要想获取脚本的全部使用描述和参数，可以带上 —help 或者是 —usage 参数启动脚本。注意， uio 或者是 vfio 内核模块需要在运行 dpdk-nic-bind.py 脚本之前加载到内核中。

查看系统中所有网口的状态：

```
./usertools/dpdk-devbind.py --status

Network devices using DPDK-compatible driver

============================================

0000:82:00.0 ?82599EB 10    -GbE NIC? drv=uio_pci_generic unused=ixgbe

0000:82:00.1 ?82599EB 10    -GbE NIC? drv=uio_pci_generic unused=ixgbe

Network devices using kernel driver

===================================

0000:04:00.0 ?I350 1   -GbE NIC? if=em0 drv=igb unused=uio_pci_generic *Active*

0000:04:00.1 ?I350 1   -GbE NIC? if=eth1 drv=igb unused=uio_pci_generic

0000:04:00.2 ?I350 1   -GbE NIC? if=eth2 drv=igb     unused=uio_pci_generic

0000:04:00.3 ?I350 1   -GbE NIC? if=eth3 drv=igb unused=uio_pci_generic

Other network devices

=====================

<none>
```

绑定设备 eth1，一04：00.1 到 uio_pci_generic 驱动上：

```
./usertools/dpdk-devbind.py --bind=uio_pci_generic 04:00.1
```

或者是：

```
./usertools/dpdk-devbind.py --bind=uio_pci_generic eth1
```

恢复设备 82：00.0 到原始的内核驱动上：

```
./usertools/dpdk-devbind.py --bind=ixgbe 82:00.0
```

5 编译和运行例子程序

本章讲的是怎么在 DPDK 环境下编译和运行程序。也指出了样例程序保存在哪里。

注意： 本章写的部分安装过程也可以通过后续 <快速安装脚本 >一章中描述的脚本实现。

5.1 编译例子程序

DPDK 目标环境目录创建完成 （例如 x86_64-native-linuxapp-gcc ）, 就包含了创建一个 dpdk 应用程序需要的库和头文件。

当在 linux DPDK 环境下编译程序，下面两个环境遍历必须 export ：

l RTE_SDK 指向 DPDK 编译安装目录即 target 所在目录

l RTE_TARGET 指向 DPDK target 目录名

下面展示的是运行在 linux DPDK 环境下的例子 helloworld 。可以在 $RTE_SDK/examples 目录下找到。

目录下包含 main.c 文件，默认编译生成的二进制文件在 build 目录下：

```
cd examples/helloworld/

export RTE_SDK=$HOME/DPDK

export RTE_TARGET=x86_64-native-linuxapp-gcc

make
```

```
CC main.o

LD helloworld

INSTALL-APP helloworld

INSTALL-MAP helloworld.map

ls build/app

helloworld helloworld.map
```

注意：在上面的例子中，helloworld 是在 DPDK 框架目录中，为了保证 DPDK 框架的完整性，可以把 helloworld 移动到外部。在下面的例子中，helloworld 被拷贝到一个新目录中重新编译。

```
export RTE_SDK=/home/user/DPDK

cp -r $(RTE_SDK)/examples/helloworld my_rte_app

cd my_rte_app/

export RTE_TARGET=x86_64-native-linuxapp-gcc

make

CC main.o

LD helloworld

INSTALL-APP helloworld

INSTALL-MAP helloworld.map
```

5.2 运行例子程序

警告：必须在运行 DPDK 程序前安装好 UIO 驱动和大页内存。

警告：任何进程使用的网口在程序启动之前必须绑定到适当的内核模块上，如之前 <从内核模块上绑定和解绑网卡 >章节所述。

程序必须链接 DPDK target 环境 EAL （环境抽象层）库，这个库提供了每一个 DPDK 程序通用的选项。

下面是可以传递给 EAL 的参数列表：

./rte-app -c COREMASK [-n NUM] [-b <domain:bus:devid.func>] \

[--socket-mem=MB,...] [-m MB] [-r NUM] [-v] [--file-prefix] \

[--proc-type <primary|secondary|auto>] [-- xen-dom0]

EAL 参数如下：

l -c COREMASK ：要使用 CPU core16 进制掩码。注意 core 编号在不同的平台不一样，需要事先确定好。

l -n NUM ：每个处理器 socket 的内存通道数

l -b <domain:bus:devid.func>: 网口黑名单 ,EAL 不能使用的 PCI 设备（可以同时存在多个 -b 选项）

l --socket-mem: 在指定 socket 上分配大页内存

l -m MB ：指定分配大大页内存数，不限处理器的 socket 。加以使用 —socket-mem 代替这个参数。

l -r NUM: 内存的 rank 数

l -v ：显示程序版本号

l --huge-dir: 大页内存的挂载点

- --file-prefix: 大页内存文件的前缀
- --proc-type: 进程类型
- --xen-dom0: 支持程序在 Xen Domain0 中非大页内存下运行
- --vmware-tsc-map: 使用 VMware TSC 代替本地的 RDTSC
- --base-virtaddr : 指定虚拟地址的基址
- --vfio-intr: 指定 VFIO 使用的中断类型（如果不是用 VFIO 则无效）

-c 是必须的，其它都是可选的。

运行例子程序（假设该平台每个处理器 socket 有 4 个内存通道，有 4 个 cpu 核，且程序都使用上）：

./helloworld -c f -n 4

注意： --proc-type 和 — file-prefix EAL 参数用于 DPDK 多进程。可以查阅 DPDK 范例程序一书中多进程章节和 DPDK 开发手册获取更多细节。

5.2.1 程序使用的逻辑核

CPU 核掩码参数是 DPDK 程序必须带上的。 掩码的每个 bit 位代表 linux 显示的每个逻辑核的编号。不同的平台之间， 逻辑核编号和逻辑核编号与 NUMA socket 的映射关系是不同的，所以建议在选在使用核之前先弄清楚该平台的核分布。

在初始化 DPDK 环境抽象层时，会打印出要使用的逻辑核以及其所在的 socket 。这些信息也可以通过查看 /proc/cpuinfo 文件来知道，例如，执行 cat /proc/cpuinfo 。每个逻辑核的物理 ID 属性表明其属于哪个 CPU socket 。

注意：逻辑核分布视图可以通过使用 linux 工具 lstopo 获取。在 Fedora 系统中，可以使用下面的命令安装使用：

sudo yum install hwloc

./lstopo

警告：逻辑核的分布在不同的主板上是不一样的，在选择使用的核之前要检查一下。

5.2.2 程序使用的大页内存

运行 DPDK 程序，建议大页内存用多少就分配多少。 如果没有传入 -m 或者是 —socket-mem 参数， DPDK 程序在启动时自动完成。

如果实际使用的内存超过了传入参数值指定的值， 程序将失败并退出。 然而， 如果用户要申请使用的内存小于预先分配的内存大小，特别是使用了 -m 选项时，程序自身也会失败。假定系统在 socket0 和 socket1 上各预先分配有 1024 个 2MB 内存页，用户请求使用 128MB 内存， 64 个页可能不符合要求：

l 内核给予应用程序分配的大页内存都在 socket1 上。这种情况下，如果应用程序试图在 socket0 上创建一个对象， 比如一个无锁队列或者是内存池， 程序会失败。 要避免这种问题，建议使用 —socket-mem 选项代替 -m 选项。

l 那些大页内存在物理上是分布在各处的， 然而 DPDK EAL 试图分配在物理地址上连续的一段内存， 而可能现有的内存页不是连续的， 这种情形下， 应用程序无法申请大的内存池而失败。

socket-mem 选项可以在指定的 socket 上分配指定大小的内存页。要想做到这个，就得在这个选项后面接上要在各个 socket 上分配内存大小，例如： --socket-mem=0,512 ，就是在 socket1 上分配 512MB 内存。同理，在 4socket 的系统中， 只在 socket0 和 2 上各分配 1GB 内存： --socket-mem=1024,0,1024 。某一个 CPUsocket 上没有内存分配，那么就不需要显式的引用，例如前面 socket3 就是。如果不能在指定的 socket 上分配到足够的内存， EAL 初始化失败。

5.3 其它例子程序

其它的例子程序在 $RTE_SDK./example 目录下。这些程序的编译和运行方式与本文中之前章节介绍的差不多。此外，可以查阅 DPDK 范例程序手册，该手册介绍了在编译和执行程序时的特殊命令参数，以及代码的说明。

5.4 其它的测试程序

另外，有两个额外的测试程序在 DPDK 库创建的时候编译出来。源码在 DPDK/app 目录下，名为 test 和 testpmd 。DPDK 目标库创建完成，那么就可以在 build/app 目录中找到。

l test 程序提供了 DPDK 各种函数的特定测试。

l testpmd 程序提供了一些不同的报文吞吐测试和特性，例如如何使用 intel82599 万兆网卡的 Flow Director （引流）特性。

6 开启其它功能

6.1 高精度定时器 (HPET) 功能

6.1.1 BIOS 支持

要使用 HPET 必须 BIOS 中支持，否则默认采用 TSC 。一般情况下，在系统启动的时候按 F2 键可以进入 BIOS 。用户可以开启 HPET 选项。在 intel 的 Crystal Forest 平台 BIOS 上，路径是 Advanced -> PCH-IO Configuration -> High Precision Timer -> (Change from Disabled to Enabled if necessary).

在已经启动的系统中，下面的命令可以查看 HPET 是否开启：

```
grep hpet /proc/timer_list
```

如果返回为空，那么必须重启机器并在 BIOS 中开启 HPET （重复上述的指令）。

6.1.2 linux 内核支持

DPDK 通过将时间戳计数器映射到处理器地址空间来利用平台的 HPET，如此需要 HPET_MMAP 内核配置项支持。

警告：在 Fedora 系统和其他通用 linux 发行版，例如 ubuntu，HPET_MMAP 内核选项是默认关闭的。带此参数重新编译内核，请查阅发行版文档了解相关操作。

在 DPDK 中使用 HPET

默认情况下，DPDK 编译配置文件中 HPET 选项是关闭的。要使用 HPET，CONFIG_RTE_LIBEAL_USE_HPET 配置项必须改成 y，需要在编译时修改参数。

程序使用 rte_get_hpet_cycles() 和 rte_get_hpet_hz() 接口调用，在程序初始化时调用 rte_eal_hpet_init() 接口来将 HPET 作为 rte_timer 库的默认时间源。这个接口将确保 HPET 可用，如果不可用则返回错误值。例如，如果 HPET_MMAP 内核不支持。那么程序可以针对支持或者是不支持而决定采取什么动作。

注意：如果程序需要时间 API，但是 HPET 不支持，建议采用 rte_get_timer_cycles() 和 rte_get_timer_hz() 接口代替 HPET 接口。这个通用接口既可以使用 TSC，也可以使用 HPET 作为时间源，取决于是否在程序启动是调用 rte_eal_hpet_init() 接口，如果调用了的话，系统是否支持。

6.2 非 root 用户运行 DPDK 程序

尽管程序通过 DPDK 能够直接使用网口和硬件资源，但是这些程序如果是作为普通用户而不是 root 用户，那么需要做一些小的权限调整。要实现这个，下面这些 linux 系统文件的所有者和权限必须调整，以便普通 linux 用户也能在使用 DPDK 的时候访问它们：

- 所有的大页内存挂载点目录，例如 /mnt/huge
- 在/dev 目录下的用户态 IO 设备文件，/dev/uio0,/dev/uio1，等等。
- 用户态 IO 系统文件配置和资源文件，例如 uio0：

/sys/class/uio/uio0/device/config

/sys/class/uio/uio0/device/resource

l 如果要使用 HPET，文件 /dev/hpet

注意：在一些 linux 系统上安装时， /dev/hugepages 会默认作为大页内存的挂载点创建。

6.3 电源管理和省电功能

如果要使用 DPDK 的电源管理功能， 那么必须在平台 BIOS 中开启增强型 INTEL SpeedStep（步进？）技术开关。否则，系统文件 /sys/devices/system/cpu/cpu0/cpufreq 不会存在，且基于 CPU 主频调节的电源管理就无法使用。查阅相关的 BIOS 文档确定这些设置如何访问。

例如，在一些 INTEL 平台 BIOS 是不一致的，增强型 INTEL SpeedStep 技术的路径是：

Advanced

-> Processor Configuration

-> Enhanced Intel SpeedStep? Tech

此外，要开启电源管理 C3 和 C6 也一样要打开。 C3 和 C6 在同样的 BIOS 平台：

Advanced

-> Processor Configuration

-> Processor C3 Advanced

-> Processor Configuration

-> Processor C6

6.4 使用 DPDK 的 CPU 亲和性减少上下文切换的开销

当一些线程被 DPDK 程序使用时，这些线程每一个会被固定到系统中的一个逻辑核心上 (一个线程对应一个逻辑核心), 对于 linux 调度器来说， 也有可能会在这些核心上运行其他任务。为了阻止调度器将其他任务调度到 DPDK 使用的核心上 ,需要使用内核参数 isolcpus 来将这些核心从 Linux 调度器中隔离开来 .

例如， DPDK 程序使用逻辑核性 2， 4， 6 运行程序，下面的这个选项需要加到内核启动参数中：

isolcpus=2,4,6

6.5 加载 DPDK KNI 内核模块

要运行 DPDK KNI 例子程序，需要额外加载一个内核模块（ kni 模块）到运行的 linux 内核中。可以在 DPDK target 目录下的 kmod 目录中找到。同理如加载 igb_uio 模块，加载 kni 也是通过 ismod 命令来（假设我们当前处于 DPDK target 目录下）：

insmod kmod/rte_kni.ko

6.6 使用 linux IOMMU 透传来在 INTEL VT-d 虚拟化环境下运行 DPDK

要在 linxu 内核中开启 intel VT-d ，一系列的内核配置项需要设置：

- IOMMU_SUPPORT
- IOMMU_API
- INTEL_IOMMU

此外，要在 intel VT-d 虚拟化环境下运行 DPDK ，当使用 igb_uio 驱动时，必须设置内核参数 iommu=pt 。这个是主机中 DMAR （DMA Remapping(DMA 地址重映射）透传的结果。INTEL_IOMMU_DEFAULT_ON 在内核中也没有设置， 内核参数 inte_iommu=on 必须使用。这可以确保 intel IOMMU 按照期望被初始化。

请注意，使用 igb_uio 驱动时必须使用 iommu=pt 参数，而 vfio-pci 驱动则是使用 iommu=pt 和 iommu=on 二者都可以运行。

6.7 40G 网口高性能小包处理

在最新版本的固件镜像中，性能增强的问题解决了，固件更新可以获取更高的性能。用户需要检查本地的 intel 网卡驱动版本。用户需要查阅 DPDK 版本说明文档确定这个版本对应的网卡 i40E 驱动版本。

6.7.1 使用 16 个字节的 RX 描述符

i40e 轮询驱动支持 16 和 32 字节的 RX 描述符大小，而 16 个字节的能够提供更高的小包处理性能。需要在配置文件中修改 CONFIG_RTE_LIBRTE_I40E_16BYTE_RX_DESC 参数为使用 16 个字节的 RX 描述符。

6.7.2 高性能和报文时延间的均衡

由于硬件设计的原因，每个报文描述符的回写需要在网卡内产生一个中断信号。最小的中断间隔需要在编译时修改配置文件中的 CONFIG_RTE_LIBRTE_I40E_ITR_INTERVAL 参数。尽管有默认配置，用户可以根据自己的实际情况，看是更在乎性能还是时延来调优这个参数。

7 快速安装启动脚本

dpdk-setup.sh 脚本，在 usertools 子目录下，可以帮助用户执行下面的任务：

- 编译 DPDK 库
- 插入和移走 DPDK IGB_UIO 内核模块
- 插入和移走 VFIO 内核模块
- 插入和移走 DPDK KNI 内核模块

- 创建和删除 NUMA 和非 NUMA 下的大页内存
- 网口状态试图展示和分配网口给 DPDK 程序使用
- 为使用 VFIO 的非特权用户分配权限
- 运行 test 和 testpmd 程序
- 展示在 meminfo 中的大页信息
- 列出 /mnt/huge 中的大页
- 删除创建的 DPDK 库

7.1 脚本结构

dpdk-setup.sh 脚本是用户按照用户的执行命令逻辑顺序来组织起来的一系列的步骤。每一个步每一步都提供了一些提示信息帮助用户完成要求的任务。下面是每一步的简单概述。

step 1 ：编译 DPDK 库

最开始，用户必须选择一个 DPDK target ，匹配类型和编译选项，以便生成正确的库。

用户必须按照前面章节所述，将所有需要的库，模块，更新和编译需要的东西安装好。

step2: 安装环境

用户配置 DPDK 程序运行所需的 linux 环境。大页内存的建立。 任何已存的大页内存会被移除。 DPDK 用到的内核模块也在这步中插入， DPDK 程序使用的网卡接口也绑定到对应的模块。

step3 ：运行程序

用户可能需要在上面步骤完成后运行 test 程序。 test 程序允许执行一系列的 DPDK 功能测试。 testpmd 支持收发包相关的测试。

step4 ：检查系统

这步提供了一些工具来检查系统的大页内存状态。

step5 ：系统还原

最后一步是恢复系统到原有的状态。

7.2 用例

下面是一些使用 dpdk-setup.sh 脚本的用例。这个脚本应该使用 source 命令来运行。

警告：脚本需要在 root 权限下运行。

```
source usertools/dpdk-setup.sh

------------------------------------------------------------------------
RTE_SDK exported as /home/user/rte
------------------------------------------------------------------------
Step 1: Select the DPDK environment to build
------------------------------------------------------------------------
[1] i686-native-linuxapp-gcc
[2] i686-native-linuxapp-icc
[3] ppc_64-power8-linuxapp-gcc
[4] x86_64-native-bsdapp-clang
[5] x86_64-native-bsdapp-gcc
[6] x86_64-native-linuxapp-clang
```

[7] x86_64-native-linuxapp-gcc

[8] x86_64-native-linuxapp-icc

Step 2: Setup linuxapp environment

---

[11] Insert IGB UIO module

[12] Insert VFIO module

[13] Insert KNI module

[14] Setup hugepage mappings for non-NUMA systems

[15] Setup hugepage mappings for NUMA systems

[16] Display current Ethernet device settings

[17] Bind Ethernet device to IGB UIO module

[18] Bind Ethernet device to VFIO module

[19] Setup VFIO permissions

Step 3: Run test application for linuxapp environment

---

[20] Run test application ($RTE_TARGET/app/test)

[21] Run testpmd application in interactive mode ($RTE_TARGET/app/testpmd)

Step 4: Other tools

---

[22] List hugepage info from /proc/meminfo

Step 5: Uninstall and system cleanup

---

[23] Uninstall all targets

[24] Unbind NICs from IGB UIO driver

[25] Remove IGB UIO module

[26] Remove VFIO module

[27] Remove KNI module

[28] Remove hugepage mappings

[29] Exit Script

选项：

下面是展示选择创建 x86_64-native-linuxapp-gcc DPDK 库的例子：

Option: 9

========== Installing x86_64-native-linuxapp-gcc

Configuration done

```
== Build lib

...

Build complete

RTE_TARGET exported as x86_64-native-linuxapp-gcc
```

下面是展示插入 DPDK UIO 驱动的例子：

```
Option: 25

Unloading any existing DPDK UIO module

Loading DPDK UIO module
```

下面的选择是展示创建 NUMA 系统下的大页内存。分别在两个 node 上各分配 1024 个 2MB 内存页。那么程序就需要系统 -m 4096 参数来访问这两块内存。（如果没有提供 -m 参数，那么程序会自动执行）。

注意：如果提示移除临时文件，输入 y。

```
Option: 15

Removing currently reserved hugepages

mounting /mnt/huge and removing directory

Input the number of 2MB pages for each node

Example: to have 128MB of hugepages available per node,

enter ?64? to reserve 64 * 2MB pages on each node

Number of pages for node0: 1024
```

Number of pages for node1: 1024

Reserving hugepages

Creating /mnt/huge and mounting as hugetlbfs

下面是展示选择加载 test 程序到一个逻辑核上运行：

Option: 20

Enter hex bitmask of cores to execute test app on

Example: to execute app on cores 0 to 7, enter 0xff

bitmask: 0x01

Launching app

EAL: coremask set to 1

EAL: Detected lcore 0 on socket 0

...

EAL: Master core 0 is ready (tid=1b2ad720)

RTE>>

7.3 应用程序

一旦用户运行 dpdk-setup.sh 脚本，编译 EAL 库，建立好大页内存 （加入使用的是 linxu EAL 目标），用户可能接着会去编译和运行自己的程序或者是例子程序。

在/examples 目录下的例子提供了解 DPDK 操作的好的起点。 下面的命令展示了 helloworld 程序是怎么编译和运行的。如同上述章节建议的, <程序使用的逻辑核心 >, 逻辑核心在平台上的分布应该在选择程序使用的逻辑核掩码时确定好。

```
cd helloworld/

make

CC main.o

LD helloworld

INSTALL-APP helloworld

INSTALL-MAP helloworld.map

sudo ./build/app/helloworld -c 0xf -n 3

[sudo] password for rte:

EAL: coremask set to f

EAL: Detected lcore 0 as core 0 on socket 0

EAL: Detected lcore 1 as core 0 on socket 1

EAL: Detected lcore 2 as core 1 on socket 0

EAL: Detected lcore 3 as core 1 on socket 1

EAL: Setting up hugepage memory...

EAL: Ask a virtual area of 0x200000 bytes

EAL: Virtual area found at 0x7f0add800000 (size = 0x200000)

EAL: Ask a virtual area of 0x3d400000 bytes
```

```
EAL: Virtual area found at 0x7f0aa0200000 (size = 0x3d400000)

EAL: Ask a virtual area of 0x400000 bytes

EAL: Virtual area found at 0x7f0a9fc00000 (size = 0x400000)

EAL: Ask a virtual area of 0x400000 bytes

EAL: Virtual area found at 0x7f0a9f600000 (size = 0x400000)

EAL: Ask a virtual area of 0x400000 bytes

EAL: Virtual area found at 0x7f0a9f000000 (size = 0x400000)

EAL: Ask a virtual area of 0x800000 bytes

EAL: Virtual area found at 0x7f0a9e600000 (size = 0x800000)

EAL: Ask a virtual area of 0x800000 bytes

EAL: Virtual area found at 0x7f0a9dc00000 (size = 0x800000)

EAL: Ask a virtual area of 0x400000 bytes

EAL: Virtual area found at 0x7f0a9d600000 (size = 0x400000)

EAL: Ask a virtual area of 0x400000 bytes

EAL: Virtual area found at 0x7f0a9d000000 (size = 0x400000)

EAL: Ask a virtual area of 0x400000 bytes

EAL: Virtual area found at 0x7f0a9ca00000 (size = 0x400000)

EAL: Ask a virtual area of 0x200000 bytes

EAL: Virtual area found at 0x7f0a9c600000 (size = 0x200000)
```

```
EAL: Ask a virtual area of 0x200000 bytes

EAL: Virtual area found at 0x7f0a9c200000 (size = 0x200000)

EAL: Ask a virtual area of 0x3fc00000 bytes

EAL: Virtual area found at 0x7f0a5c400000 (size = 0x3fc00000)

EAL: Ask a virtual area of 0x200000 bytes

EAL: Virtual area found at 0x7f0a5c000000 (size = 0x200000)

EAL: Requesting 1024 pages of size 2MB from socket 0

EAL: Requesting 1024 pages of size 2MB from socket 1

EAL: Master core 0 is ready (tid=de25b700)

EAL: Core 1 is ready (tid=5b7fe700)

EAL: Core 3 is ready (tid=5a7fc700)

EAL: Core 2 is ready (tid=5affd700)

hello from core 1

hello from core 2

hello from core 3

hello from core 0
```

8 怎么在 intel 平台上获得网卡的最好性能

本章是在 INTEL 平台上 DPDK 程序获得高性能的一步步引导。

8.1 硬件和内存需求

要最好的性能最好是使用 intel 至强系列服务器，例如 Ivy Bridge ， Haswell 或者是更新的架构的服务器。

确保每一个内存通道至少有一个内存 DIMM 插入，且内存的大小至少 4GB 。注意：这个在性能上有最直接的影响。

你可以检查内存配置通过 dmidecode 命令，如下：

```
dmidecode -t memory | grep Locator

Locator: DIMM_A1

Bank Locator: NODE 1

Locator: DIMM_A2

Bank Locator: NODE 1

Locator: DIMM_B1

Bank Locator: NODE 1

Locator: DIMM_B2

Bank Locator: NODE 1

...

Locator: DIMM_G1

Bank Locator: NODE 2

Locator: DIMM_G2

Bank Locator: NODE 2
```

Locator: DIMM_H1

Bank Locator: NODE 2

Locator: DIMM_H2

Bank Locator: NODE 2

上面的输出展示了总共 8 个内存通道，每个通道有 2 个 DIMM 。

也可以通过 dmidecode 命令确定内存的频率：

dmidecode -t memory | grep Speed

Speed: 2133 MHz

Configured Clock Speed: 2134 MHz

Speed: Unknown

Configured Clock Speed: Unknown

Speed: 2133 MHzConfigured Clock Speed: 2134 MHz

Speed: Unknown

...

Speed: 2133 MHz

Configured Clock Speed: 2134 MHz

Speed: Unknown

Configured Clock Speed: Unknown

Speed: 2133 MHz

Configured Clock Speed: 2134 MHz

Speed: Unknown

Configured Clock Speed: Unknown

上述输出表明内存为 2133MHz 的 DDR4 内存。和前面的信息结合起来可以确定每个内存通道有一根内存条。

8.1.1 网卡需求

使用 DPDK 支持的高端网卡，比如 INTEL XL710 40GBE 。

确保每一个网卡刷入了最新版本的固件。

使用 PCIe Gen3 插槽，例如 Gen3 x8 或者是 Gen3 x16 ，因为 PCIe Gen2 插槽无法提供足够的宽带， 2X10Gbe 或者是更高。可以通过使用 lspci 命令检查 PCI 插槽的使用信息：

```
lspci -s 03:00.1 -vv | grep LnkSta

LnkSta: Speed 8GT/s, Width x8, TrErr- Train- SlotClk+ DLActive- ...

LnkSta2: Current De-emphasis Level: -6dB, EqualizationComplete+ ...
```

当擦汗如网卡到 PCI 插槽时总要检查说明信息，例如 CPU0 或者是 CPU1 连接到的是哪个 socket 上。

多考虑一下 NUMA 。如果你使用 2 个或者是更多的不同的网口，最好确保这些网卡在同样的 CPU socket 上。如何确定看下面的例子。

8.1.2 BIOS 设置

下面是推荐的 BIOS 设置项。不同的平台上可能有不同的 BIOS 名称， 所以下面主要是作为参考：

1. 考虑重新设置所有的 BIOS 默认配置

2. 禁用所有的省电选项，例如：电源性能优化（ Power performance tuning ），CPU P-State ，CPU C3 Report and CPU C6 Report 。
3. CPU Power and Performace 策略选择 Performance 。
4. 禁用 Tuibo Boost 以确保性能随着核数的增长线性增加
5. 设置内存频率为可用的最高数，而不是 auto 。
6. 当测试网卡的物理功能时，禁用所有的虚拟化选项，在需要使用 VFIO 是打开 VT-d。

8.1.3 Linux 内核启动参数

下面是 GRUB 启动参数建议设置：

1. 使用默认的 grub 文件作为启动点。
2. 通过 grub 配置申请 1G 大小的内存页。例如申请 8 个 1GB 内存页：

```
default_hugepagesz=1G hugepagesz=1G hugepages=8
```

1. 跟你 DPDK 程序需要使用 CPU 核心，例如：

```
isolcpus=2,3,4,5,6,7,8
```

1. 要使用 VFIO ，把下面的作为额外的 grub 参数：

```
iommu=pt intel_iommu=on
```

8.2 运行 DPDK 前的配置

1. 编译 DPDK target ，分配大页内存。看前面的章节。

下面的 shell 命令有助于编译和配置 DPDK 。

```
# Build DPDK target.

cd dpdk_folder

make install T=x86_64-native-linuxapp-gcc -j

# Get the hugepage size.

awk ?/Hugepagesize/ {print $2}? /proc/meminfo
```

```
# Get the total huge page numbers.

awk ?/HugePages_Total/ {print $2} ? /proc/meminfo

# Unmount the hugepages.

umount `awk ?/hugetlbfs/ {print $2}? /proc/mounts`

# Create the hugepage mount folder.

mkdir -p /mnt/huge

# Mount to the specific folder.

mount -t hugetlbfs nodev /mnt/huge
```

1. 使用 DPDK 的 cpu_layout 脚本检查 CPU 分布：

```
cd dpdk_folder

usertools/cpu_layout.py
```

1. 检查网卡 ID 和所属的 Socket ID：

```
# List all the NICs with PCI address and device IDs.

lspci -nn | grep Eth

举例说明，假设输出是这样的：

82:00.0 Ethernet [0200]: Intel XL710 for 40GbE QSFP+ [8086:1583]

82:00.1 Ethernet [0200]: Intel XL710 for 40GbE QSFP+ [8086:1583]

85:00.0 Ethernet [0200]: Intel XL710 for 40GbE QSFP+ [8086:1583]

85:00.1 Ethernet [0200]: Intel XL710 for 40GbE QSFP+ [8086:1583]
```

检查 PCI 设备所述的 NUMA 节点 ID：

```
cat /sys/bus/pci/devices/0000\:xx\:00.x/numa_node
```

通常情况下， 0x:00.x 是位于 socket0 上， 8x:00.x 位于 socket1 上。注意：为了获得更高的性能， 确保 cpu 核心和网卡在同样的 socket 上。例如上面的 85：00.0 在 socket1 上，那么它被 socket1 上的核心使用能获得最好的性能。

1. 绑定测试网口到 DPDK 支持的驱动上，例如 igb_uio 。例如，绑定两个短裤到 DPDK 驱动并检查状态：

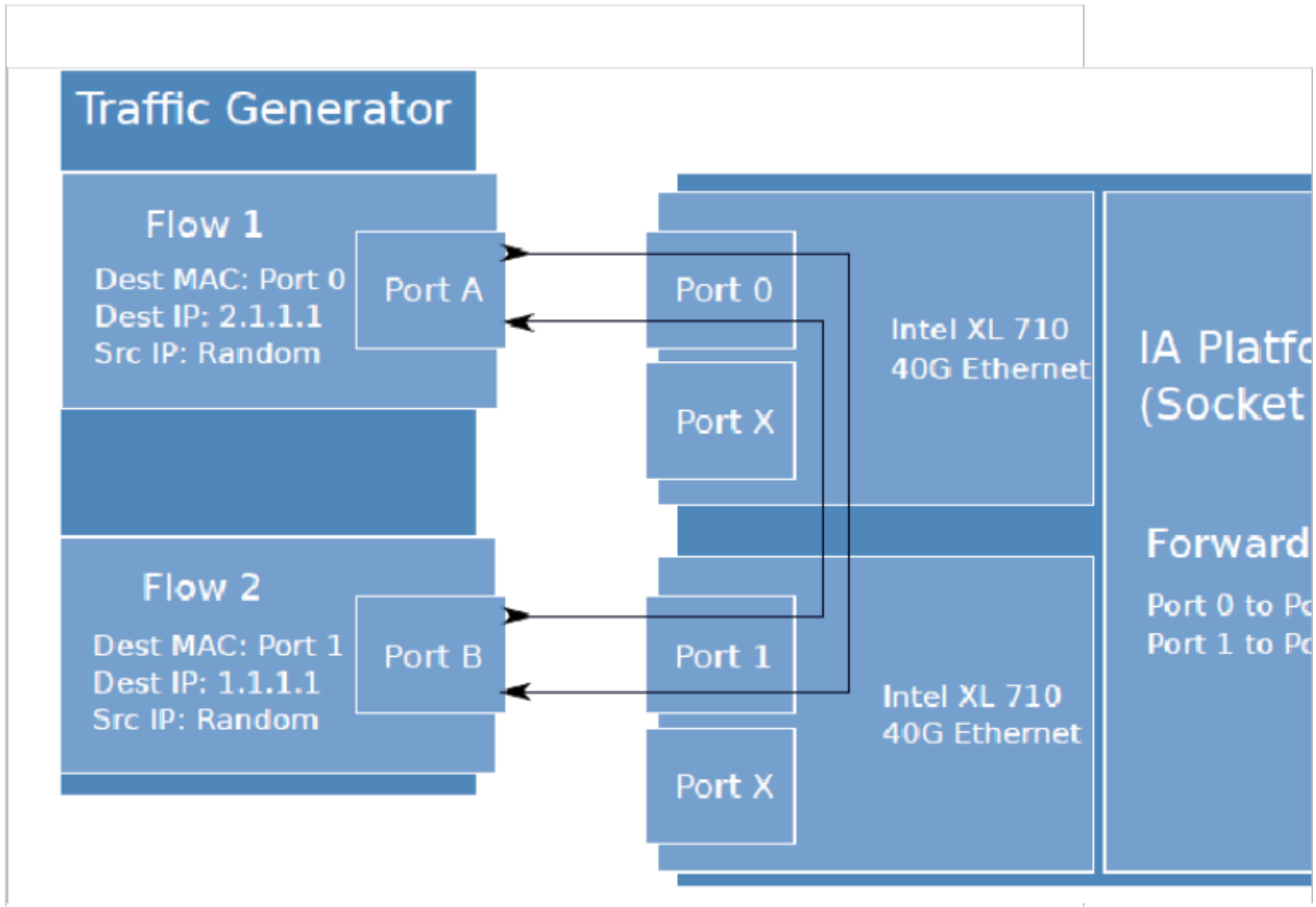
```
# Bind ports 82:00.0 and 85:00.0 to dpdk driver

./dpdk_folder/usertools/dpdk-devbind.py -b igb_uio 82:00.0 85:00.0

# Check the port driver status

./dpdk_folder/usertools/dpdk-devbind.py --status
```

查看 dpdk-devbind.py --help 获取更多细节。

8.3 获取 intel 网卡的最好性能例子

下面是运行 DPDK3 层转发例子： l3fwd 程序， 使用 intel 服务器平台和 intelXL710 网卡获的高性能。对于特殊的 40GbE 午那个卡的配置请查阅 i40e 网卡手册。

这个例子展示的是在两个 INTELXL710 40GbE 网口下获得的最好性能。



1. 插入两个 intelXL710 网卡到机器上 ,且使用每个网卡的一个网口以获取更高的性能, 使用两个网卡是因为一个 PCIe Gen3 插槽不能提供 2 个 40G 端口的 80G 的带宽去求，而两个不同的 PCIe Gen3 x8 插槽则可以，那么就客服了这个限制。引用上面的网卡状态输出，我们可以选择 82 ：00.0 和 85 ：00.0 作为测试网口：

```
82:00.0 Ethernet [0200]: Intel XL710 for 40GbE QSFP+ [8086:1583]

85:00.0 Ethernet [0200]: Intel XL710 for 40GbE QSFP+ [8086:1583]
```

1. 连接端口到对应的发包设备上。为了高性能的测试，最好使用一个专门的硬件发包器。
2. 检查 PCI 设备的 NUMA 节点 ID（socket id），获取杜英的 socketID 上的核心编号。在这个例子中，82 ：00.0 和 85 ：00.0 均在 socket1 上，本平台的 socket 上的核心是 18-35 和 54-71 。注意：不要使用在同一个物理核心上的 2 个逻辑核心（例如： core18 有两个逻辑核心， core18 和 core54），取而代之的是，使用在不同物理核心上的逻辑核辛（例如 core18 和 core19 ）。
3. 绑定两个端口到 igb_uio 。
4. 对于 XL710 40G 网口，至少需要 2 组两个队列以获取最好的性能，即每个端口要两个队列。每组端口需要一个 CPU 核心做报文的收发。
5. DPDK 程序 l3fwd 将用于性能测试，使用两个端口直接转发。使用默认的 lpm （最长前缀匹配）模式编译 l3fwd 程序。
6. 运行 l3fwd 程序的命令行参数如下：

./l3fwd -c 0x3c0000 -n 4 -w 82:00.0 -w 85:00.0 \

-- -p 0x3 -- config ?(0,0,18),(0,1,19),(1,0,20),(1,1,21)?

这代表着程序使用 core18 转发 port0 的队列 0，core19 转发 port0 的队列 1，core20 转发 port1 的队列 0，core21 转发 port1 的队列 1。

1. 配置发包器的流量

a) 在发包器上创建一个流

b) 设置以太网 II 类型为 0x8000