DPDK Cookbook

Featuring:

Solution-Oriented Mini Sections

User-Friendly Screen shots

Links to Videos and Online Contents

Overview

The DPDK Cookbook modules teach you everything you need to know to be productive with the Data Plane Development Kit (DPDK). Here's an overview of the topics covered:

- Build Your Own DPDK Traffic Generator—DPDK-In-A-Box
- DPDK Transmit and Receive—DPDK-in-a-Box
- Build Your Own DPDK Packet Framework with DPDK-In-A-Box
- DPDK Data Plane—Multicores and Control Plane Synchronization
- DPDK Performance Optimization Guidelines White Paper
- Profiling DPDK Code with Intel® VTune™ Amplifier
- References

I highly recommend that you devour the Architecture Overview section of the Programmer's Guide at dpdk.org. This excellent document, authored by architects and designers, goes into both the how and the why of DPDK design.

Change is the only constant in this fast-moving field, with some of these components delivering new releases every three months. Please refer to the related user guides and release notes to be sure you use the latest version when applying these cookbook recipes. I provide links to many resources, and some of those will inevitably change as well, so please accept my apology in advance if you encounter a broken link.

Acknowledgements

I'm grateful to many people for their valuable input, including early access customers, architects, design engineers, managers, platform application engineers, DPDK community, and Intel® Network Builders. In particular, this cookbook is possible only due to the encouragement, support, and reviews from Jim St Leger, Venky Venkatesan, Tim O'Driscoll, John DiGiglio, John Morgan, Cristian Dumitrescu, Sujata Tibrewala, Debbie Graham, Ray Kinsella, Jasvinder Singh, Deepak Jain, Steve Cunming, Heqing Zhu, Dave Hunt, Kannan Ramia Babu, Walt Gilmore, Mike Glynn, Curran Greg, Ai Bee Lim, Larry Wang, Nancy Yadav, Chiu-Pi Shih, Deepak S, Anand Jyoti, Dirk Blevins, Andrew Duignan, Todd Langley, Joel Auernheimer, Joel Schuetze, and Eric Heaton.

About the Author

Muthurajan Jayakumar (M Jay) has worked with the DPDK team since 2009. He joined Intel in 1991 and has worked in various roles and divisions with Intel, including roles as a 64-bit CPU front side bus architect, and as a 64-bit HAL developer. M Jay holds 21 US patents, both individually and jointly, all issued while working at Intel. M Jay was awarded the Intel Achievement Award in 2016, Intel's highest honor based on innovation and results. Before joining Intel, M Jay architected a CPU node board for a 1000-node machine design in India. M Jay won a gold medal for graduating with the university first rank ECE batch in 1984, from TCE, Madurai.

Please send your feedback about the DPDK Cookbook to Muthurajan.Jayakumar@intel.com.

Getting Started: Documentation and Tools

Key Documentation

Dpdk.org contains a rich set of documentation. The table below highlights some of the guides and other materials that you'll find useful to familiarize yourself with DPDK programming. You can read the guides online or download many of them in PDF form.

Programmers Guide	***The guide you must read first.***
Quick Start Guide	Simple forwarding test with pcap PMD, which works with any NIC.
API Documentation	All libraries and APIs.
Supported NICs	The list of supported NICs grows with each new release of DPDK. Please refer to this document for the latest list.
Network Interface Controller Drivers	Poll mode drivers for supported NICs—virtual as well as physical.
DPDK Sample Application User Guide	More than 40 sample applications—find the closest match to your application.
DPDK Testpmd Application User Guide	The key DPDK tool with port, NIC set, and show commands.
Release Notes	The latest features, issues addressed, and issues to be addressed in the future.
Getting Started Guide for Linux*	Build, install, and Getting Started.
How-to Guides	Covers topics such as live migration of a VM with SR-IOV VF, live migration of a VM with Virtio on host running vhost_user, a flow bifurcation guide, and more.
Crypto Device Drivers	Contains Crypto Device Supported Functionality Matrices and details about support for many drivers, including: • AESN-NI Multi Buffer Crypto Poll Mode Driver • AES-NI GCM Crypto Poll Mode Driver • KASUMI Crypto Poll Mode Driver • Null Crypto Poll Mode Driver • SNOW 3G Crypto Poll Mode Driver • Quick Assist Crypto Poll Mode Driver
FAQ	Frequently asked questions.
Getting Started Guide for FreeBSD	DPDK FreeBSD Linux GSG.
Contributor's Guidelines	Do you want to contribute code and/or documentation to the DPDK community?

Frequently Used Tools and Scripts

The scripts listed in this section can be found in the tools or scripts subdirectories of your DPDK install. Below are some frequently used tools and scripts to study.

./tools/setup.h	Menu-driven setup script	In tools subdirectory
./tools/dpdk_nic_bind.p y	For binding NIC to driver	In tools subdirectory
./tools/cpu_layout.py	For lcore number and layout	In tools subdirectory
./tools/pmdinfo.py	For PMD info	In tools subdirectory
http://dpdk.org/doc/dts/gsg	DPDK Test Suite (DTS)	Getting Started Guide—DTS

Tool Usage Examples

Finding Memory Information with Linux Command `/proc/meminfo`*

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# cat /proc/meminfo
MemTotal:        1939152 kB
MemFree:          155892 kB
MemAvailable:     209044 kB
Buffers:           12068 kB
Cached:           309116 kB
SwapCached:        14196 kB
Active:           518824 kB
Inactive:         465260 kB
Active(anon):     407424 kB
Inactive(anon):   404812 kB
Active(file):     111400 kB
Inactive(file):    60448 kB
Unevictable:          32 kB
Mlocked:              32 kB
SwapTotal:       1986556 kB
SwapFree:        1557500 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        655444 kB
Mapped:           147228 kB
Shmem:            149336 kB
Slab:              53084 kB
SReclaimable:      25100 kB
SUnreclaim:        27984 kB
KernelStack:        7392 kB
PageTables:        30136 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     2615140 kB
Committed_AS:    4444120 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:    215040 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:     333
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       95824 kB
DirectMap2M:     1892352 kB
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test#
```

Finding Huge Page Information with ./setup.sh

./setup.sh has an option to list huge page information from **/proc/meminfo** (option 29 in the version of DPDK shown here).

```
----------------------------------------------------------
 Step 4: Other tools
----------------------------------------------------------
[29] List hugepage info from /proc/meminfo

----------------------------------------------------------
 Step 5: Uninstall and system cleanup
----------------------------------------------------------
[30] Unbind NICs from IGB UIO or VFIO driver
[31] Remove IGB UIO module
[32] Remove VFIO module
[33] Remove KNI module
[34] Remove hugepage mappings

[35] Exit Script

Option: 29

AnonHugePages:    198656 kB
HugePages_Total:     256
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB

Press enter to continue ...
```

Binding/ Unbinding NIC with ./dpdk_nic_bind.py

```
Examples:
---------

To display current device status:
        dpdk_nic_bind.py --status

To bind eth1 from the current driver and move to use igb_uio
        dpdk_nic_bind.py --bind=igb_uio eth1

To unbind 0000:01:00.0 from using any driver
        dpdk_nic_bind.py -u 0000:01:00.0

To bind 0000:02:00.0 and 0000:02:00.1 to the ixgbe kernel driver
        dpdk_nic_bind.py -b ixgbe 02:00.0 02:00.1
```

Finding CPU layout with ./cpu_layout.py

CPU info can also be found with DPDK script **./cpu_layout.py**.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools# ls
cpu_layout.py  dpdk_nic_bind.py  pmdinfo.py  setup.sh
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools# ./cpu_layout.py
============================================================
Core and Socket Information (as reported by '/proc/cpuinfo')
============================================================

cores =  [0, 2]
sockets =  [0]

       Socket 0
       --------
Core 0 [0]
Core 2 [1]
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools#
```

More Scripts in Scripts Subdirectory

Fill in the description for each script:

Auto-config-h.sh	
Check-git-log.sh	
Check-maintainers.sh	
Checkpatches.sh	
Cocci.sh	
Depdirs-rule.sh	
Gen-build-mk.sh	
Gen-config-h.sh	
Load-devel-config.sh	
Relpath.sh	
Test-build.sh	
Test-null.sh	
Validate-abi.sh	

Build Your Own DPDK Traffic Generator—DPDK-In-A-Box

Introduction

The purpose of this cookbook module is to guide you through the steps required to build a Data Plane Development Kit (DPDK) based traffic generator.

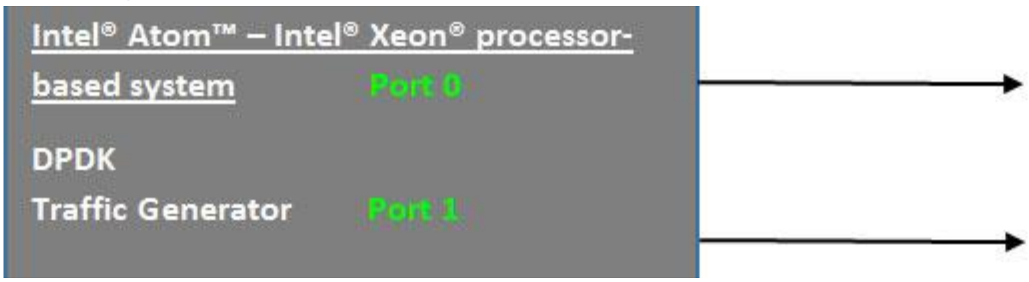
We built a DPDK-in-a-Box using the MinnowBoard Turbot* Dual Ethernet Dual-Core, which is a low cost, portable platform based on the Intel Atom® processor E3826. For the OS, we installed Ubuntu* 16.04 client with DPDK. The instructions in this document are tested on our DPDK-in-a-Box, an Intel® Core™ i7-5960X processor Extreme Edition brand desktop, and an Intel® Xeon® Scalable processor. You can use any Intel® architecture platform to build your own device.

For the traffic generator, we use the TRex* realistic traffic generator. The TRex package is self-contained and can be easily installed.

Any Intel® processor-based platform will work—desktop, server, laptop, or embedded system.

The DPDK Traffic Generator

Block Diagram



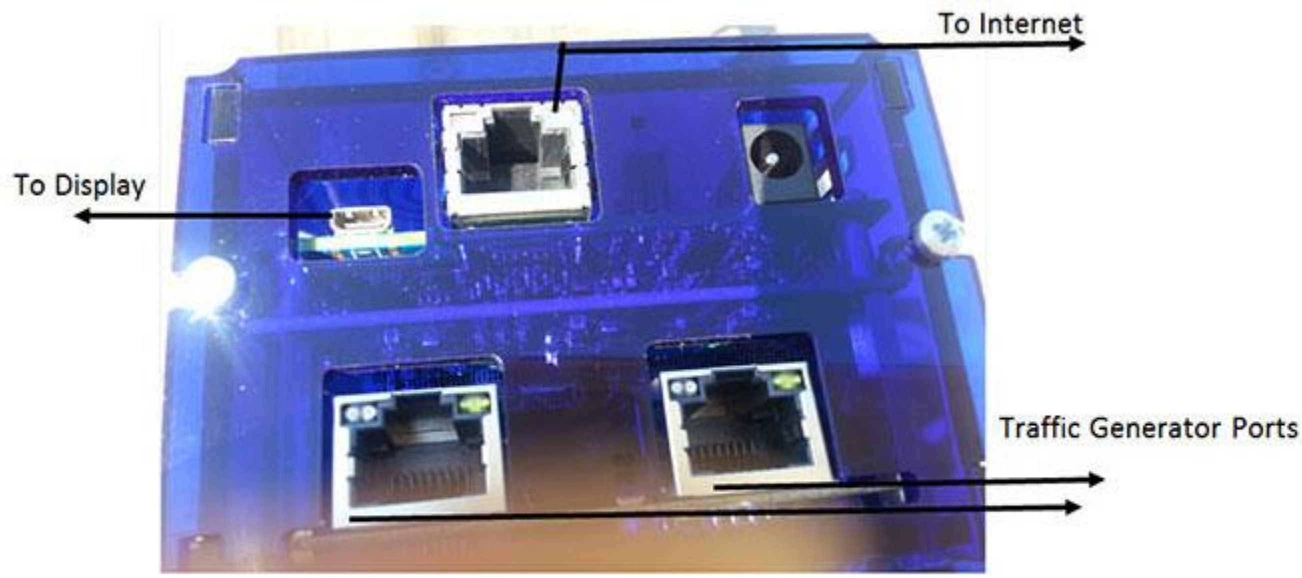
Software

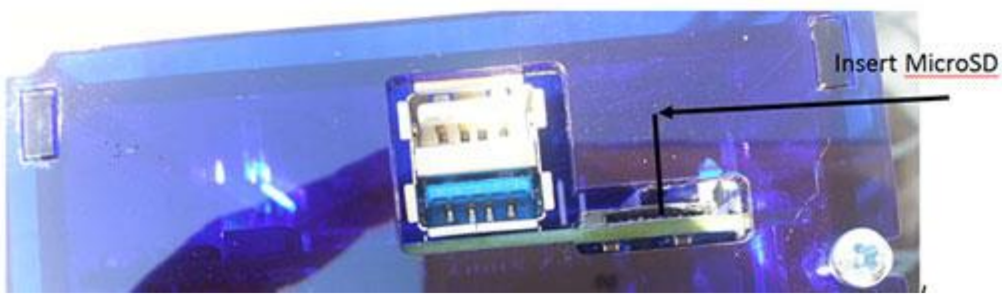
- Ubuntu 16.04 Client OS with DPDK installed
- TRex Realistic Traffic Generator

Hardware

Our DPDK-in-a-Box uses a MinnowBoard Turbot Dual Ethernet Dual-Core single board computer:

- Out of the three Ethernet ports, the two at the bottom are for the traffic generator (dual gigabit Intel® Ethernet Controller I350). Connect a loopback cable between them.
- Connect the third Ethernet port to the Internet (to download the TRex package).
- Connect the keyboard and mouse to the USB ports.
- Connect a display to the HDMI Interface.





The [MinnowBoard Turbot Dual Ethernet Dual-Core]*

The MinnowBoard includes a microSD card and an SD adapter.

- Insert the microSD card into the microSD Slot. The SD adapter should be ignored and not used.
- Power on the DPDK-in-a-Box system. Ubuntu will be up and running right away.

Install and Configure the TRex* Traffic Generator

Choose the username `test` and assign the password `tester` (or use the username and password specified by the Quick Start Guide that comes with the platform).

- Log on as root and verify that you are in the /home/test directory with the following two commands:

```
# sudo su
# ls
```

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# ls
Desktop  Documents  Downloads  dpdk  examples.desktop  Music  Pictures  Public  Templates  Videos
```

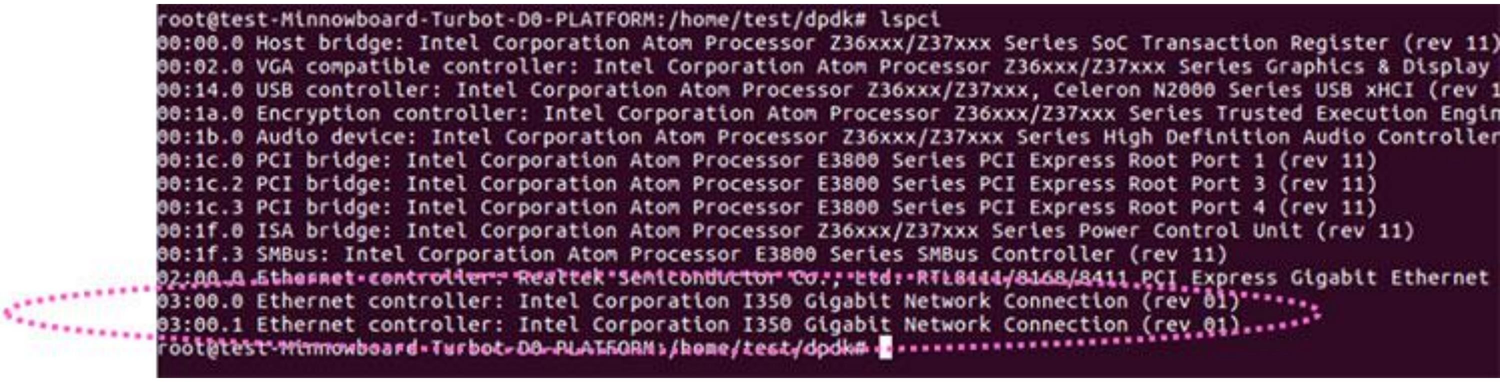
Note NIC Information

The configuration file for the traffic generator needs the PCI bus-related information and the MAC address. Note this information first using Linux* commands, because once the DPDK or packet generator is run, these ports are unavailable to Linux.

1. For PCI bus-related NIC information, type the following command:

```
# lspci
```

You will see the following output. Note down that for port 0 the bus, function, and device number information is 03:00.0, and for port 1 the information is 03:00.1.



2. Find the MAC address with this command:

```
# ifconfig
```

You will see the following output. Note down that for port 0 the MAC address is 00:30:18:CB:F2:70 and for port 2 the MAC address is 00:30:18:CB:F2:71.

Note that the first port in the screenshot below, enp2s0, is the port connected to the Internet. No need to make a note of this.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# ifconfig
enp2s0    Link encap:Ethernet  HWaddr 00:08:a2:09:f2:1d
          inet addr:192.168.0.11  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::56cd:7409:7867:9572/64 Scope:Link
          inet6 addr: 2601:647:4902:79c0:6a14:5825:3e6c:de09/64 Scope:Global
          inet6 addr: 2601:647:4902:79c0:ac82:14bd:f4da:e627/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:82453 errors:0 dropped:0 overruns:0 frame:0
          TX packets:56424 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:60196138 (60.1 MB)  TX bytes:17006340 (17.0 MB)

enp3s0f0  Link encap:Ethernet  HWaddr 00:30:18:cb:f2:70
          inet6 addr: fe80::ef12:bbfd:4cff:5034/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:103 errors:0 dropped:0 overruns:0 frame:0
          TX packets:135 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20949 (20.9 KB)  TX bytes:22055 (22.0 KB)
          Memory:90500000-9057ffff

enp3s0f1  Link encap:Ethernet  HWaddr 00:30:18:cb:f2:71
          inet6 addr: fe80::2ad4:3349:f1fa:73f0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:85 errors:0 dropped:0 overruns:0 frame:0
          TX packets:146 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16637 (16.6 KB)  TX bytes:24515 (24.5 KB)
          Memory:90600000-9067ffff

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:10234 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10234 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1072552 (1.0 MB)  TX bytes:1072552 (1.0 MB)
```

Here's what I recorded after these two steps:

Item	Port 0	Port 1
PCI Bus-related NIC info (from lspci)	**03:00.0**	**03:00.1**
MAC address	**00:30:18:CB:F2:70**	**00:30:18:CB:F2:71**

Fill the following table with the information you gathered from your specific platform:

Item	Port 0	Port 1
PCI Bus-related NIC info (from lspci)		
MAC address		

If you succeeded in using `ifconfig` to get the port information described above, skip the next section and move on to the section titled *Install the Traffic Generator*.

Troubleshooting – Ports Not Found

What if you don't see both ports in response to the `ifconfig` command? One possible reason might be that you've run the DPDK based application previously and the application might have claimed those ports, making them unavailable to the kernel. In that case, you need to unbind the ports from the DPDK so that the kernel can claim them and you can find the MAC address with the `ifconfig` command.

Root Cause

`ifconfig` is not showing the two ports below. Why?

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools# ifconfig
enp2s0    Link encap:Ethernet  HWaddr 00:08:a2:09:f2:1d
          inet addr:192.168.0.6  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: 2601:647:4902:79c0:b98c:9a9e:55f6:8314/64 Scope:Global
          inet6 addr: 2601:647:4902:79c0:8712:fcc2:af9:a689/64 Scope:Global
          inet6 addr: fe80::3603:79d2:fe9e:8468/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:60216 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53600 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:33276201 (33.2 MB)  TX bytes:10484345 (10.4 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:15976 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15976 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1594031 (1.5 MB)  TX bytes:1594031 (1.5 MB)

root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools#
```

The reason that `ifconfig` is unable to find the two ports is possibly because the DPDK application was previously run and was aborted without releasing the ports, or it might be that a DPDK script runs automatically after boot and claims the ports. Regardless of the reason, the solution below will enable `ifconfig` to show both ports.

Solution

1. Run `./setup.sh` in the directory `/home/test/dpdk/tools`.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools# ls
cpu_layout.py  dpdk_nic_bind.py  pmdinfo.py  setup.sh
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools# ./setup.sh
```

2. Display current Ethernet device settings.

```
[23] Display current Ethernet device settings
[24] Bind Ethernet device to IGB UIO module
[25] Bind Ethernet device to VFIO module
[26] Setup VFIO permissions

----------------------------------------------------------
 Step 3: Run test application for linuxapp environment
----------------------------------------------------------
[27] Run test application ($RTE_TARGET/app/test)
[28] Run testpmd application in interactive mode ($RTE_TARGET/app/testpmd)

----------------------------------------------------------
 Step 4: Other tools
----------------------------------------------------------
[29] List hugepage info from /proc/meminfo

----------------------------------------------------------
 Step 5: Uninstall and system cleanup
----------------------------------------------------------
[30] Unbind NICs from IGB UIO or VFIO driver
[31] Remove IGB UIO module
[32] Remove VFIO module
[33] Remove KNI module
[34] Remove hugepage mappings

[35] Exit Script

Option: 23
```

Select **Display current Ethernet device settings** (option 23 in this case).

You can see that two ports are claimed by the DPDK driver.

```
Option: 23


Network devices using DPDK-compatible driver
============================================
0000:03:00.0 'I350 Gigabit Network Connection' drv=igb_uio unused=igb,vf
0000:03:00.1 'I350 Gigabit Network Connection' drv=igb_uio unused=igb,vf

Network devices using kernel driver
===================================
0000:02:00.0 'RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller'
o_pci_generic *Active*

Other network devices
=====================
```

The NICs in use by DPDK (specifically IGB-UIO)

3. Unbind the first port from IGB UIO.

```
----------------------------------------------
 Step 5: Uninstall and system cleanup
----------------------------------------------
[30] Unbind NICs from IGB UIO or VFIO driver
[31] Remove IGB UIO module
[32] Remove VFIO module
[33] Remove KNI module
[34] Remove hugepage mappings
```

Select option 30 and then **enter the PCI address** of device to unbind:

```
Option: 30


Network devices using DPDK-compatible driver
============================================
0000:03:00.0 'I350 Gigabit Network Connection' drv=igb
0000:03:00.1 'I350 Gigabit Network Connection' drv=igb

Network devices using kernel driver
===================================
0000:02:00.0 'RTL8111/8168/8411 PCI Express Gigabit Et
o_pci_generic *Active*

Other network devices
=====================
<none>

Enter PCI address of device to unbind: 0000:03:00.0
```

4. Bind the kernel driver igb to the device:

```
Enter name of kernel driver to bind the device to: igb
```

If the inputs entered are correct, the script acknowledges OK.

```
OK

Press enter to continue ...
```

5. Verify by displaying current Ethernet device settings.

```
Option: 23


Network devices using DPDK-compatible driver
============================================
<none>

Network devices using kernel driver
===================================
0000:02:00.0 'RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller' if=enp2s0
o_pci_generic *Active*
0000:03:00.0 'I350 Gigabit Network Connection' if=enp3s0f0 drv=igb unused=igb_uio,v
0000:03:00.1 'I350 Gigabit Network Connection' if=enp3s0f1 drv=igb unused=igb_uio,v

Other network devices
=====================
<none>
```

Success!

Above you will see the first port `0000:30:00.0` bound to the kernel.

Repeat steps 3–5 to unbind the second port, `0000:30:00.1,` from IGB UIO and bind to IGB.

Use the `ifconfig` command to show that both ports are bound back to the kernel.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk# ifconfig
enp2s0    Link encap:Ethernet  HWaddr 00:08:a2:09:f2:1d
          inet addr:192.168.0.6  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: 2601:647:4902:79c0:249:ce31:c570:85a/64 Scope:Global
          inet6 addr: fe80::a572:b28f:7fd6:5336/64 Scope:Link
          inet6 addr: 2601:647:4902:79c0:6cc6:fc3c:5f31:d114/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21515 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18422 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6537879 (6.5 MB)  TX bytes:5099447 (5.0 MB)

enp3s0f0  Link encap:Ethernet  HWaddr 00:30:18:cb:f2:70
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21 errors:0 dropped:0 overruns:0 frame:0
          TX packets:115 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5995 (5.9 KB)  TX bytes:21997 (21.9 KB)
          Memory:90500000-9057ffff

enp3s0f1  Link encap:Ethernet  HWaddr 00:30:18:cb:f2:71
          inet6 addr: fe80::f596:8de9:9963:4008/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:54 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:486 (486.0 B)  TX bytes:10474 (10.4 KB)
          Memory:90600000-9067ffff

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:6303 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6303 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:582195 (582.1 KB)  TX bytes:582195 (582.1 KB)
```

Install the Traffic Generator

In the following sections, we will assume that you successfully found the ports and have noted down the MAC addresses.

Keeping in mind my earlier note that change is the only constant thing in this fast-moving field, refer to the current TRex user manual to make sure you have the latest script names, directory structure, and release information relevant to this recipe. Enter the following commands:

```
# pwd
# mkdir trex
# cd trex
# wget -no-cache http://trex-tgn.cisco.com/trex/release/latest
```

You should see that the install is complete and saved in `/home/test/trex/latest`:

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# pwd
/home/test
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# mkdir trex
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# cd trex
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex# wget --no-cache http://trex-tgn.cisco.com/trex/release/latest
--2016-08-26 01:47:22--  http://trex-tgn.cisco.com/trex/release/latest
Resolving trex-tgn.cisco.com (trex-tgn.cisco.com)... 173.39.246.118
Connecting to trex-tgn.cisco.com (trex-tgn.cisco.com)|173.39.246.118|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://trex-tgn.cisco.com/trex/release/latest [following]
--2016-08-26 01:47:22--  https://trex-tgn.cisco.com/trex/release/latest
Connecting to trex-tgn.cisco.com (trex-tgn.cisco.com)|173.39.246.118|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 146045560 (139M) [application/x-tar]
Saving to: 'latest'

latest                                          100%[==============================================================

2016-08-26 01:48:00 (3.76 MB/s) - 'latest' saved [146045560/146045560]

root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex#
```

The next step is to untar the package:

```
# tar -xzvf latest
```

Below you see that version 2.08 is the latest version at the time of this screen capture:

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex# tar -xzvf latest
v2.08/
v2.08/_t-rex-64-debug
v2.08/t-rex-64-debug
v2.08/_t-rex-64
v2.08/t-rex-64
v2.08/_t-rex-64-debug-o
v2.08/t-rex-64-debug-o
v2.08/_t-rex-64-o
```

```
# ls -al
```

You will see the directory with the version installed. In this exercise, the directory is v2.08, as shown below in response to the `ls -al` command. Change directory to the version installed on your system; for example, `cd <dir name with version installed>`:

```
# cd v2.08
```

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex# ls -al
total 142640
drwxr-xr-x  3 root  root        4096 Aug 26 01:48 .
drwxr-xr-x 18 test  test        4096 Aug 26 01:44 ..
-rw-r--r--  1 root  root   146045560 Aug 24 14:35 latest
drwxr-xr-x 11 33066 floppy      4096 Aug 24 14:35 v2.08
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex# cd v2.08
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08#
```

```
# ls -al
```

You will see the file `t-rex-64`, which is the traffic generator executable:

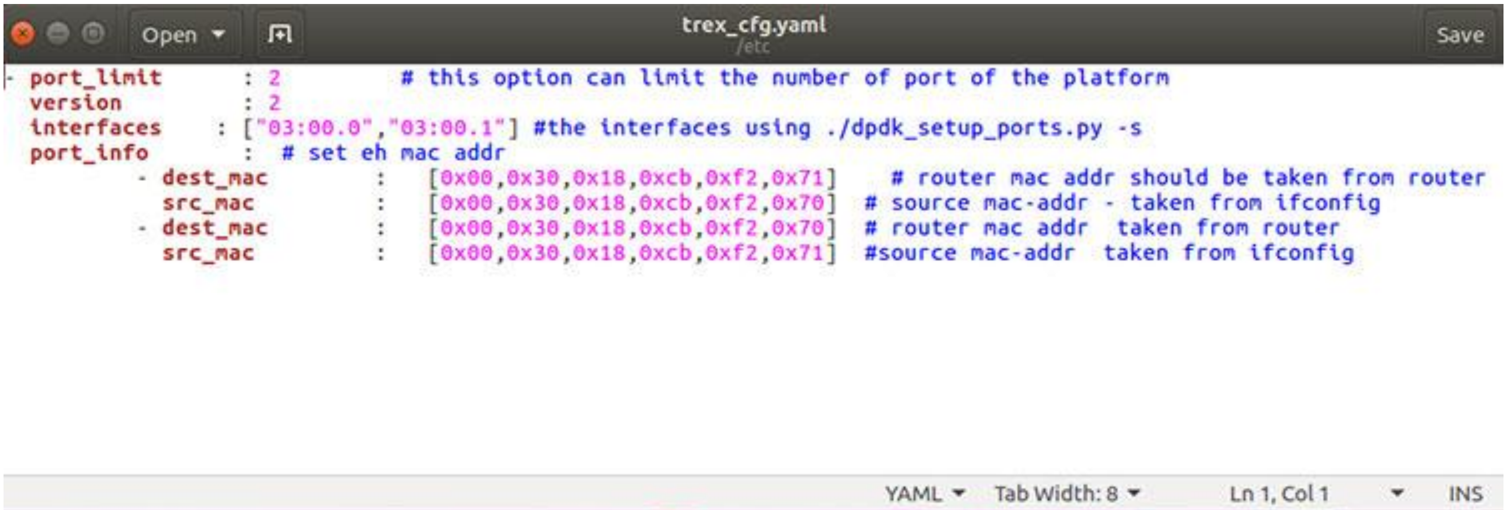
```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08# ls -al
total 176364
drwxr-xr-x 11 33066 floppy     4096 Aug 24 14:35 .
drwxr-xr-x  3 root  root       4096 Aug 26 01:48 ..
drwxr-xr-x  6 33066 floppy     4096 Aug 24 14:34 automation
drwxr-xr-x  2 33066 floppy     4096 Aug 24 14:34 avl
-rwxr-xr-x  1 33066 floppy 27200827 Aug 24 14:34 bp-sim-64
-rwxr-xr-x  1 33066 floppy 16798769 Aug 24 14:34 bp-sim-64-debug
drwxr-xr-x  2 33066 floppy     4096 Aug 24 14:34 cap2
drwxr-xr-x  2 33066 floppy     4096 Aug 24 14:34 cfg
-rwxr-xr-x  1 33066 floppy     5501 Aug 24 14:34 daemon_server
-rwxr-xr-x  1 33066 floppy     2207 Aug 24 14:34 doc_process.py
-rwxr-xr-x  1 33066 floppy    26985 Aug 24 14:34 dpdk_nic_bind.py
-rwxr-xr-x  1 33066 floppy    33325 Aug 24 14:34 dpdk_setup_ports.py
drwxr-xr-x  2 33066 floppy    16384 Aug 24 14:34 exp
drwxr-xr-x 22 33066 floppy     4096 Aug 24 14:34 external_libs
-rwxr-xr-x  1 33066 floppy     2291 Aug 24 14:34 find_python.sh
drwxr-xr-x 15 33066 floppy     4096 Aug 24 14:34 ko
-rw-r--r--  1 33066 floppy  3150071 Aug 24 14:34 libzmq.so.3
-rwxr-xr-x  1 33066 floppy    11148 Aug 24 14:34 master_daemon.py
drwxr-xr-x  3 33066 floppy     4096 Aug 24 14:34 python-lib
-rwxr-xr-x  1 33066 floppy      802 Aug 24 14:34 run_functional_tests
-rwxr-xr-x  1 33066 floppy      832 Aug 24 14:34 run_regression
drwxr-xr-x  6 33066 floppy     4096 Aug 24 14:34 stl
-rwxr-xr-x  1 33066 floppy      403 Aug 24 14:34 stl-sim
-rwxr-xr-x  1 33066 floppy 34390661 Aug 24 14:34 _t-rex-64
-rwxr-xr-x  1 33066 floppy      902 Aug 24 14:34 t-rex-64
-rwxr-xr-x  1 33066 floppy 28843174 Aug 24 14:34 _t-rex-64-debug
-rwxr-xr-x  1 33066 floppy      902 Aug 24 14:34 t-rex-64-debug
-rwxr-xr-x  1 33066 floppy 28812951 Aug 24 14:34 _t-rex-64-debug-o
-rwxr-xr-x  1 33066 floppy      902 Aug 24 14:34 t-rex-64-debug-o
-rwxr-xr-x  1 33066 floppy 35101658 Aug 24 14:34 _t-rex-64-o
-rwxr-xr-x  1 33066 floppy      902 Aug 24 14:34 t-rex-64-o
-rwxr-xr-x  1 33066 floppy     2086 Aug 24 14:34 trex-cfg
-rw-r--r--  1 33066 floppy  6077140 Aug 24 14:35 trex_client_v2.08.tar.gz
-rwxr-xr-x  1 33066 floppy      444 Aug 24 14:34 trex-console
-rwxr-xr-x  1 33066 floppy     5501 Aug 24 14:34 trex_daemon_server
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08#
```

Configure the Traffic Generator

The good news is that the TRex package comes with a sample config file `cfg/simple_cfg.yaml`. Copy that to `/etc/trex_cfg.yaml` and edit the file by issuing the following commands, making sure that you're in your `/home/test/trex/<your version>` directory:

```
# pwd
# cp cfg/simple_cfg.yaml /etc/trex_cfg.yaml
# gedit /etc/trex_cfg.yaml
```

Edit the file as shown below with the applicable NIC information you gathered in previous steps:



Below is a line-by-line description of the configuration information required for `/etc/trex_cfg.yaml`:

- `Port_limit` should be `2` (since DPDK-in-a-Box has two ports)
- Version should be `2`
- Interfaces should be the PCI bus ports you gathered using `lspci`. In this exercise they are `[“03:00.0”, “03:00.1”]`
- `Port_information` contains a `dest_mac`, `src_mac` pair, which will be in the packet header of the traffic generated. The first pair is for port 0. Since port 0 is connected to port 1, the first `dest_mac` is the MAC address of port 1. The second pair is for port 1. Since port 1 is connected to port 0, the second `dest_mac` is the MAC address of port 0.

Please note that when you connect an appliance to which traffic must be injected, the `dest_mac` addresses will be that of the appliance.

Note Platform lcore Count

This section is for informational purposes only.

`# cat /proc/cpuinfo` will give you the logical core (lcore) information as shown in the Exercises section.

Why is this information useful?

The command line below that runs the traffic generator uses the –c option to specify the number of lcores to be used for the traffic generator. You want to know how many lcores exist in the platform. Hence, issuing `cat /proc/cpuinfo` and eyeballing the number of lcores that are available in the system will be helpful.

Run the Traffic Generator

```
# sudo ./t-rex-64 -f cap2/dns.yaml -c 1 -d 100
```

What are the parameters –f, -c, and –d?

`-f` for YAML traffic configuration file
`-c` for number of cores. Monitor the CPU percentage of TRex—it should be ~50 percent. Use cores accordingly
`-d` for duration of the test (sec). Default: `0`

```
root@test-Minnowboard-Turbot-D0-PLATFORM: /home/test/trex/v2.08
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08# sudo ./t-rex-64 -f cap2/dns.yaml -c 1 -d 100
```

Below are three output screens: 1) During the traffic run, 2) Linux top command output, and 3) Final output after the completion of the run.

```
root@test-Minnowboard-Turbot-D0-PLATFORM: /home/test/trex/v2.08

-Per port stats table
      ports |               0 |                 1
 -----------------------------------------------------------------------------------------
   opackets |              15 |                15
     obytes |            1155 |              1395
   ipackets |              15 |                15
     ibytes |            1395 |              1155
    ierrors |               0 |                 0
    oerrors |               0 |                 0
      Tx Bw |      584.18  bps |      705.57  bps

-Global stats enabled
 Cpu Utilization : 0.0  %  0.0 Gb/core
 Platform_factor : 1.0
 Total-Tx        :       1.29 Kbps
 Total-Rx        :       1.29 Kbps
 Total-PPS       :       1.90  pps
 Total-CPS       :       0.97  cps

 Expected-PPS    :       2.00  pps
 Expected-CPS    :       1.00  cps
 Expected-BPS    :       1.30 Kbps

 Active-flows    :        0  Clients :       511   Socket-util : 0.0000 %
 Open-flows      :       15  Servers :       255   Socket :        15 Socket/Clients :  0.0
 drop-rate       :       0.00  bps
 current time    : 27.0 sec
 test duration   : 73.0 sec
```

Screen output showing traffic during run (15 packets so far Tx and Rx).

```
top - 06:21:00 up  2:05,  1 user,  load average: 1.33, 0.39, 0.18
Threads: 418 total,   2 running, 416 sleeping,   0 stopped,   0 zombie
%Cpu(s): 53.0 us,  2.0 sy,  0.0 ni, 42.0 id,  3.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1939152 total,   296884 free,  1122584 used,   519684 buff/cache
KiB Swap:  1986556 total,  1639292 free,   347264 used.   454792 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
16986 root      20   0  893584   9400   5852 R 99.3  0.5   0:45.47 lcore-slave+
 1956 test      20   0  662648  20424  11936 S  2.6  1.1   0:32.98 gnome-termi+
 1441 test      20   0 1491584  89496  23572 S  1.6  4.6   4:06.93 compiz
16983 root      20   0  893584   9400   5852 S  1.6  0.5   0:05.17 _t-rex-64-o
  775 root      20   0  379964  32892  23824 S  1.3  1.7   1:20.63 Xorg
16114 root      20   0   49268   3468   2444 R  1.0  0.2   1:07.05 top
  589 root      20   0  173360   2100   1888 S  0.3  0.1   0:00.07 thermald
16925 root      20   0       0      0      0 S  0.3  0.0   0:00.13 kworker/u8:3
    1 root      20   0  185372   3520   2192 S  0.0  0.2   0:04.57 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.43 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S  0.0  0.0   0:07.56 rcu_sched
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
```

Output of top –H command during the run.

```
root@test-Minnowboard-Turbot-D0-PLATFORM: /home/test/trex/v2.08
 precent    : -nan %
 histogram
 -----------
 m_total_bytes                              :      15.82 Kbytes
 m_total_pkt                                :     200.00  pkt
 m_total_open_flows                         :     100.00  flows
 m_total_pkt                                : 200
 m_total_open_flows                         : 100
 m_total_close_flows                        : 100
 m_total_bytes                              : 16200
 ---------------
port : 0
------------
 opackets                                    : 100
 obytes                                      : 7700
 ipackets                                    : 100
 ibytes                                      : 9300
 Tx :      291.61  bps
port : 1
------------
 opackets                                    : 100
 obytes                                      : 9300
 ipackets                                    : 100
 ibytes                                      : 7700
 Tx :      352.41  bps
 Cpu Utilization : 0.0  %  0.0 Gb/core
 Platform_factor : 1.0
 Total-Tx        :     644.02  bps
 Total-Rx        :     643.99  bps
 Total-PPS       :       0.95  pps
 Total-CPS       :       0.47  cps

 Expected-PPS    :       2.00  pps
 Expected-CPS    :       1.00  cps
 Expected-BPS    :       1.30 Kbps

 Active-flows    :        0  Clients :      511   Socket-util : 0.0000 %
 Open-flows      :      100  Servers :      255   Socket :        0 Socket/Clients :  0.0
 drop-rate       :       0.00  bps
 summary stats
 --------------
 Total-pkt-drop       : 0 pkts
 Total-tx-bytes       : 17000 bytes
 Total-tx-sw-bytes    : 0 bytes
 Total-rx-bytes       : 17000 byte

 Total-tx-pkt         : 200 pkts
 Total-rx-pkt         : 200 pkts
 Total-sw-tx-pkt      : 0 pkts
 Total-sw-err         : 0 pkts
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08#
```

Screen output after completing the run (100 packets Tx and Rx).

Congratulations! By completing the above hands-on exercise, you have successfully built your own DPDK based traffic generator.

Next Steps

As a next step, you can connect back-to-back two DPDK-in-a-Box platforms, and use one as a traffic generator and the other as a DPDK application development and test vehicle.

Exercises

1. How would you configure the traffic generator for different packet lengths?
2. To run the traffic generator forever, what should be the value of –d?
3. How would you measure latency (assuming you have more cores)?

4. Reason out the root cause and find the solution by looking up the error, “Note that the uio or vfio kernel modules to be used should be loaded into the kernel before running the dodk-devbind.py script” in Chapter 3 of the DPDK.org document *Getting Started Guide for Linux*.

DPDK Transmit & Receive Loopback—DPDK-In-A-Box

Introduction

In the previous module, Build Your Own Traffic Generator—DPDK-In-A-Box, you learned how to build a DPDK traffic generator. Once you’ve done this, the next step is to connect two platforms back to back, and use one as the DPDK traffic generator and the other as a DPDK application development and test vehicle. But what if you have just one system? Read on to learn how to generate traffic and run your DPDK application on the same machine.

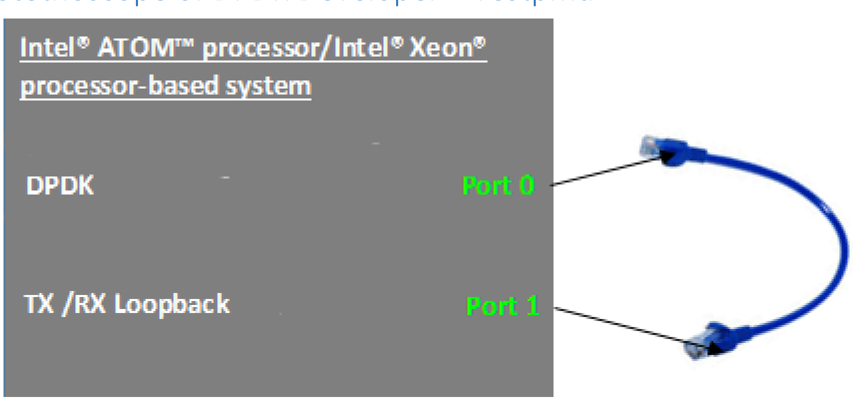
Traffic and the DPDK Application on a Single System

The purpose of this article is to show how to configure a single system to run the DPDK application and provide auto-generated traffic. To provide the traffic, we will showcase **testpmd**, which many DPDK developers and customers consider to be the stethoscope of a DPDK developer.

For your system, you can use any Intel® platform. The instructions in this article have been tested with an Intel® Xeon® processor-based desktop, server, and laptop using either the DPDK traffic generator you built from scratch, or a commercially available DPDK in-a-Box. This is a low-cost, portable platform based on an Intel Atom E3826 processor. At the time this article was published, it was possible to purchase a DPDK-in-a-Box. Look online if you’re interested in this option.

If you are new to DPDK, spend some time reading the *DPDK Programmer’s Guide* at dpdk.org.

Stethoscope of DPDK Developer—Testpmd



Auto-Generating Traffic with tx_first Parameter

Challenge

Data plane applications need a traffic generator. The DPDK provides both RX and TX functionality and DPDK applications build poll mode drivers with the RX and TX libraries. With the DPDK poll mode driver, the RX functionality of the driver polls ingress traffic, and after applicable processing, the TX functionality of the poll driver transmits the processed data to the egress interface.

Because the start of the data path involves polling for packets received, data plane applications need a traffic generator. But here we have only one platform. How do you configure the application for this traffic?

Solution

This is where the testpmd `tx_first` parameter comes in handy. When testpmd is started with the `tx_first` parameter, the TX function gets executed first—hence the name `tx_first`—and with an external cable connecting RX and TX, those packets are now available for the RX function to poll. Thus, you have achieved running traffic through testpmd without an external traffic generator.

The following screenshots show how to start testpmd and run `tx_first` with a loopback cable in place.

Starting testpmd

`./x86_64-native-linuxapp-gcc/app/testpmd -- -i` starts testpmd. –i stands for interactive. Please refer to the *Testpmd Application User Guide* at dpdk.org and to the article *Testing DPDK Performance and Features with TestPMD* on Intel® Developer Zone for more information about how to build and run testpmd.

While you can use the above command in your specific platform, it is available as a script named run.sh in DPDK-in-a-Box.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk# cat run.sh
./x86_64-native-linuxapp-gcc/app/testpmd -- -i
```

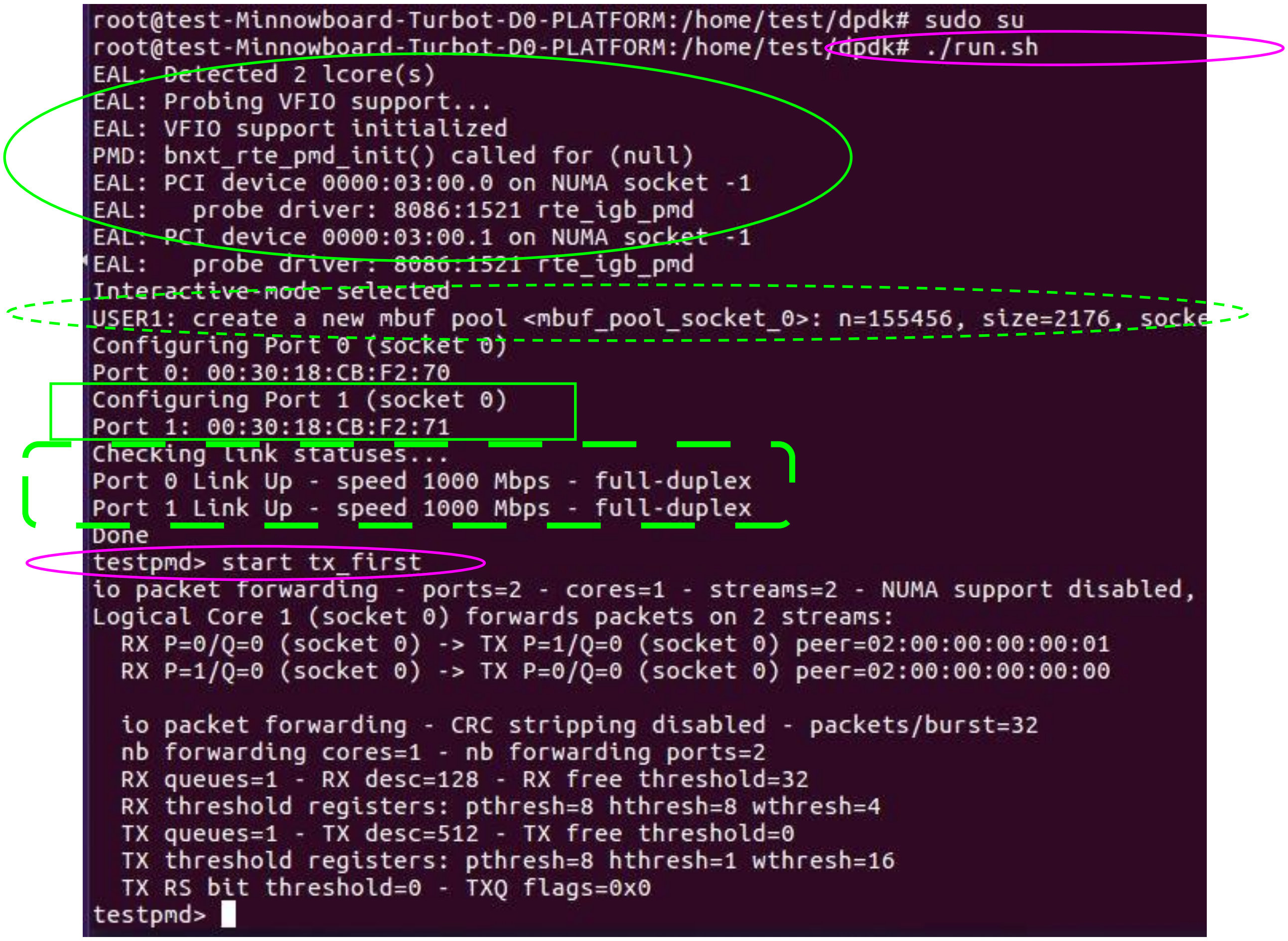
```
root@test-Minnowboard-Turbot-D0-PLATFORM: /home/test/dpdk
test@test-Minnowboard-Turbot-D0-PLATFORM:~$ sudo su
[sudo] password for test:
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# pwd
/home/test
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# ls
Desktop    Downloads  examples.desktop  Pictures  Templates
Documents  dpdk       Music             Public    Videos
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# cd dpdk/
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk# ls
app         doc          lib           Makefile  run.sh
build       drivers      LICENSE.GPL   mk        scripts
buildtools  examples     LICENSE.LGPL  pkg       tools
config      GNUmakefile  MAINTAINERS   README    x86_64-native-linuxapp-gcc
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk# ./run.sh
```

./run.sh starts testpmd, as shown below, yielding the testpmd prompt. Look at the flowchart. What does the initial portion of testpmd do? And what does the runtime portion of testpmd do? Our next step is to initialize testpmd.

Initialization

Initialization consists of three steps as shown below.

1. EAL (Environment Abstraction Layer)—Find the number of cores and probe PCI devices
2. Initialize memory zones and memory pools
3. Configure the ports for data path operation



Operation

After initialization, data path operations start and continue in a loop. The Poll Mode Driver (PMD) section of the *DPDK Programmer's Guide* is a must-read chapter. It will help you understand and appreciate how you can get the juice out of your system and achieve the desired throughput.

Optimization Knobs You Should Understand

You may want to read through the documentation to fully understand the optimization knobs like those shown in the last paragraph of the output from the `start tx_first` command.

For example, you can note that the RX descriptor counts are 128, whereas the TX desc descriptors are shown as 512. Why are they not equal? And why is the TX descriptor four times that of the RX descriptor? Also, analyze the values indicated for the RX threshold registers: `pthresh` = 8, `hthresh` = 8, and `wthresh` = 4; whereas for the TX registers: `pthresh` = 8, `hthresh` = 1, and `wthresh` = 16.

Reading the data sheet, and more importantly the optimization white papers of the Intel® 82599 10-GbE Ethernet Controller, at least the Receive and Transmit sections, will help you to understand and make the best use of your knobs.

Running testpmd Using tx_first Option

testpmd> `start tx_first`

As shown above, at the **testpmd>** prompt, enter `start tx_first` to auto-generate the traffic. Packets are transmitted when the **testpmd>** command returns. Please note that packets continue to be generated until they are stopped.

In this case, let it run for 10 to 20 seconds, and then stop the run.

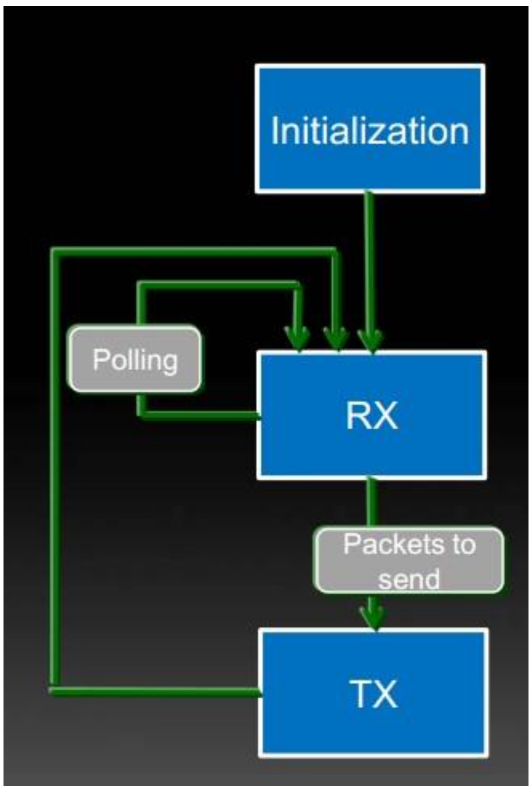
testpmd> `stop`

Below you can see the RX and TX total packets per port as well as accumulated totals for both ports.

```
testpmd> stop
Telling cores to stop...
Waiting for lcores to finish...

  ---------------------- Forward statistics for port 0  ----------------------
  RX-packets: 38915274       RX-dropped: 0             RX-total: 38915274
  TX-packets: 38916245       TX-dropped: 0             TX-total: 38916245
  ----------------------------------------------------------------------------

  ---------------------- Forward statistics for port 1  ----------------------
  RX-packets: 38916245       RX-dropped: 0             RX-total: 38916245
  TX-packets: 38915274       TX-dropped: 0             TX-total: 38915274
  ----------------------------------------------------------------------------

  +++++++++++++++ Accumulated forward statistics for all ports+++++++++++++++
  RX-packets: 77831519       RX-dropped: 0             RX-total: 77831519
  TX-packets: 77831519       TX-dropped: 0             TX-total: 77831519
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Done.
testpmd>
```

Starting testpmd Without `tx_first`

What happens when you start testpmd without `tx_first`? The flowchart below shows that RX starts polling first, so in this case you need a traffic generator.



Flowchart for the case of without `tx_first`*.*

We saw that with `tx_first`, TX gets executed first. This emits packets first and before polling, thus auto-generating the traffic. This allows us to run testpmd with traffic using a single platform.

Now, draw the flowchart for using `tx_first`.

Learn More About testpmd

Use `-h` or `--help` to find the available command-line options and thus the available functionality of testpmd.

I highly recommend that you learn hands-on the functionalities of interest to you and note them in the Exercise section. Once you are done, quit by using the quit command.

`quit`, as shown below, does the following:

- Stops the ports
- Closes the ports

Summary

At this point, you've configured a single system to run the DPDK application and generated `tx-first` and `rx_first` traffic with **testpmd**. Test your knowledge with the exercises below.

Exercises

1. `tx_first` auto-generates traffic. How are the parameters of the traffic programmed in this case?
2. You saw the flowchart for the case of without `tx_first`. Draw the flowchart for the case with `tx_first`.
3. Note each command-line option and functionality you tried with testpmd, and list what you learned about each one, with any suggestions you may have.
4. What is the difference between detaching a port and closing a port? Where will you use detaching a port? Where will you use closing a port?
5. What is the difference between **dpdk_nic_bind.py** and **dpdk-devbind.py**? Explain.
6. Search the Internet and the dpdk.org dev mailing list to analyze the root cause and find the solution for the error. Note that the uio or vfio kernel modules to be used should be loaded into the kernel before running the dpdk-devbind.py script.

Build Your Own DPDK Packet Framework with DPDK-In-A-Box

Introduction

The title of this module might just as well be "Build Your Own *Software Defined* DPDK Application." The DPDK packet framework uses a modular building block approach defined by a configuration file to build complex DPDK applications. For an overview of the value of the DPDK packet framework, watch the short video Deep Dive into the Architecture of a Pipeline Stage before you get started with this module.

Here you will build a DPDK packet framework with just two cores—one for master core tasks and the other to perform DPDK application functions.

For hardware, you can use any IA platform—Intel Xeon brand or Intel Atom brand desktop, server, or laptop. We will use DPDK-in-a-Box here. This is a low-cost, portable platform based on the Intel Atom E3826 processor.

To build your own DPDK-in-a-Box, or learn where to purchase a DPDK-in-a-Box, please see the earlier module in this cookbook, *Build Your Own DPDK Traffic Generator—DPDK-In-A-Box*.

Set DPDK Traffic Generator MAC Addresses

If you remember when we built a DPDK Traffic Generator, the configuration file of the DPDK traffic generator was set with its own port's MAC Addresses, since we looped the ports within themselves.

Here we are connecting the ports to an external DPDK packet Generator, so we will set the MAC addresses in the DPDK traffic generator to match those of the external system that will run the DPDK packet framework.

It is left as an exercise for the developers to find out the MAC address and set the traffic generator configuration file with them.

Update the Configuration File for the DPDK Packet Framework

The DPDK packet framework configuration files provide a *software defined* modular way to implement complex DPDK applications. If your system has multiple lcores available for packet processing, you can implement both run-to-completion as well as pipelined applications. Here, since we have only one core for packet processing with DPDK-in-a-box, we will showcase the run-to-completion implementation.

Building and Installing DPDK Packet Framework

We'll now go through the steps for building and installing DPDK packet framework, as described in the DPDK sample application user's guide.

Running the Traffic Through DPDK Packet Framework

Connect the systems together. Provide the command-line option of the traffic generator continuously (this was part of the exercise in the traffic generator building cookbook section). This runs the traffic through DPDK packet framework.

Run Your Application that is Software Defined by Packet Framework

- Run your application that is software defined by packet framework.
- Use profilers to find out where the CPU is spending most of the cycles and if it is in line with your expectation.
- Write up your observations and share with the community in dpdk.org

Summary

Application developers will benefit by understanding DPDK assumptions on roles / responsibilities of applications. They need to comprehend the scope of DPDK's roles / responsibilities to begin with. This helps them to rightly architect from the get-go obeying DPDK's assumptions in terms of thread safety, lockless API call usage, multiprocessor synchronization, and control plane and data plane synchronization.

Exercise

1. Draw your software-defined application block diagram.

DPDK Data Plane—Multicores and Control Plane Synchronization

Introduction

Many developers and customers are under the impression that DPDK documentation and sample applications include only data plane applications. In a real-life scenario, it is necessary to integrate the data plane with the control and management plane. The purpose of this cookbook module is to describe some simple multiplane scenarios.

For hardware, you can use any IA platform—Intel Xeon brand or Intel Atom brand desktop, server, or laptop. We will use DPDK -in-a-Box here. This is a low-cost, portable platform based on the Intel Atom E3826 processor.

To build your own DPDK-in-a-Box, please see the earlier module in this cookbook, *Build Your Own DPDK Traffic Generator—DPDK-In-A-Box*.

Simple Scenarios—Data Plane and Control Plane Interactions

Every product or appliance will have its own share of data plane and control plane functionality. Here we will illustrate two simple run-time scenarios:

1. A NIC port configuration change
2. Changing the port itself

Scenario 1: Change of Hardware

In a multiple core server with as many as 72 lcores (lcore stands for logical core), with multiple NIC ports performing packet processing in parallel, how do you synchronize the control plane operations with the data plane? What do you need to understand in order to play by the DPDK rules of the game?

What must be synchronized in order to pull out a transceiver and insert it again—during runtime?

Likewise, to pull out a transceiver, say 10 Gig, and insert a completely different one, say 1 Gig?

If a change of hardware device requires a release of the instance of the device that was removed and creation of a device instance for the new one, what do you do with threads that are still accessing the data structures of the original instance?

Releasing resources requires coordination with the user space applications using those resources. Applications can be using a single core or multiple cores. If the resource being released is used by multiple cores, we need to request an acknowledgement handshake from each core in use, indicating that they are all finished with the resource and it can be released safely.

Scenario 2: No Change of Hardware but Change of Parameter

Assume you are not changing any hardware during runtime. But you do want to change some global parameter—say MTU. This may be a *lightweight* initialization compared to the previous case of *heavy weight* initialization. So, when you use an API that does a lightweight initialization, which parameters can you expect to be persistent across the operation and which parameters can you not assume will remain the same? This is very useful information to know in order to correctly change parameters during runtime.

Please note that this is only part of the story. The other part is synchronizing with data plane applications that are running, that is, waiting for a resource to be available, so that APIs to reconfigure can be called. We will look at that and refer to pointers available in DPDK documentation and source code.

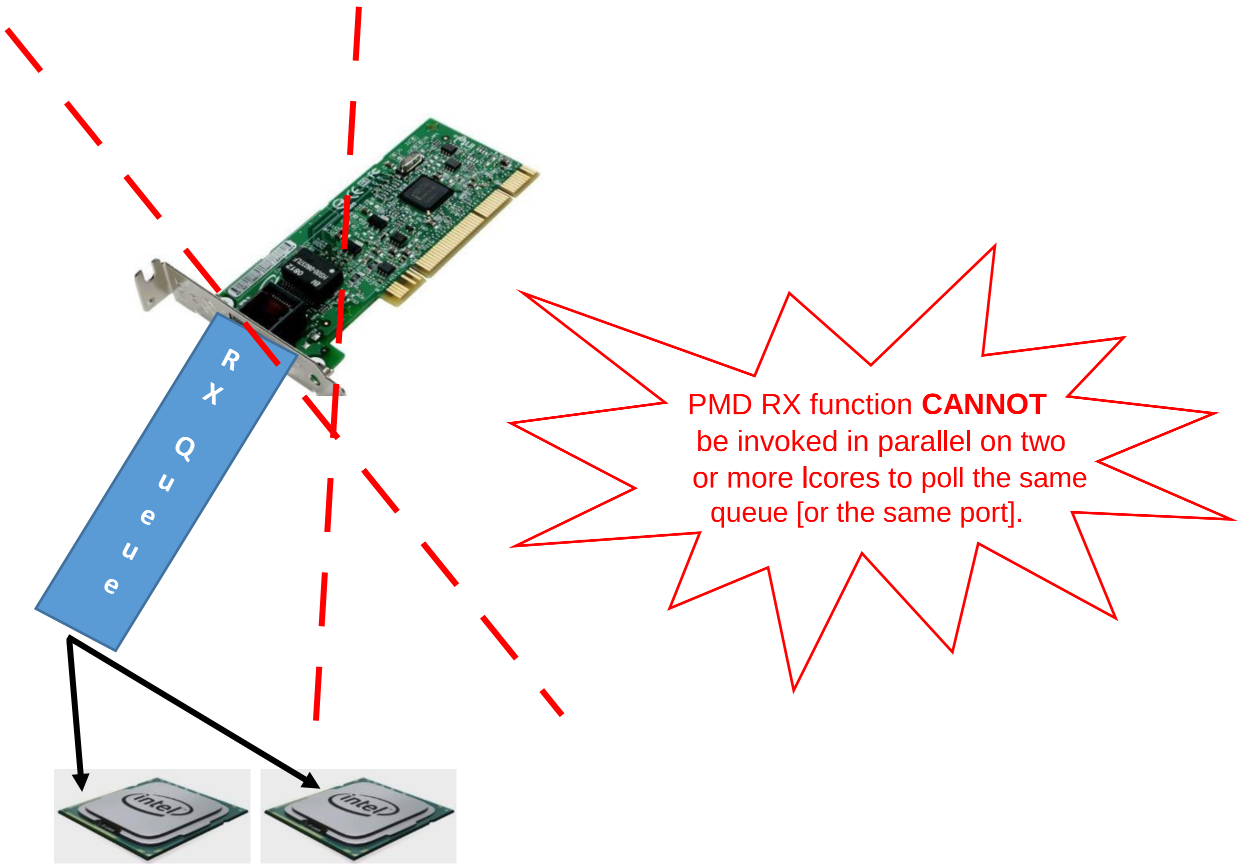
Before we get into these details, let’s step back and look at the big picture:

1. What are the core assumptions DPDK makes in terms of concurrency?
2. What are the boundaries of what DPDK controls and what the application must manage to ensure synchronization?

Rules for Polling Queues

Can Multiple Cores Poll One RX Queue Simultaneously?

By design, the receive function of a PMD **CANNOT** be invoked in parallel on multiple, that is, two or more logical cores to poll the same RX queue [of the same port]. What is the benefit of this design? It is that all the functions of the Ethernet Device API exported by a PMD are lock-free functions. This is possible because the receive function will not be invoked in parallel on different logical cores to work on the same target object.



In the case of a single RX queue per port, only one core at a time can do RX processing.

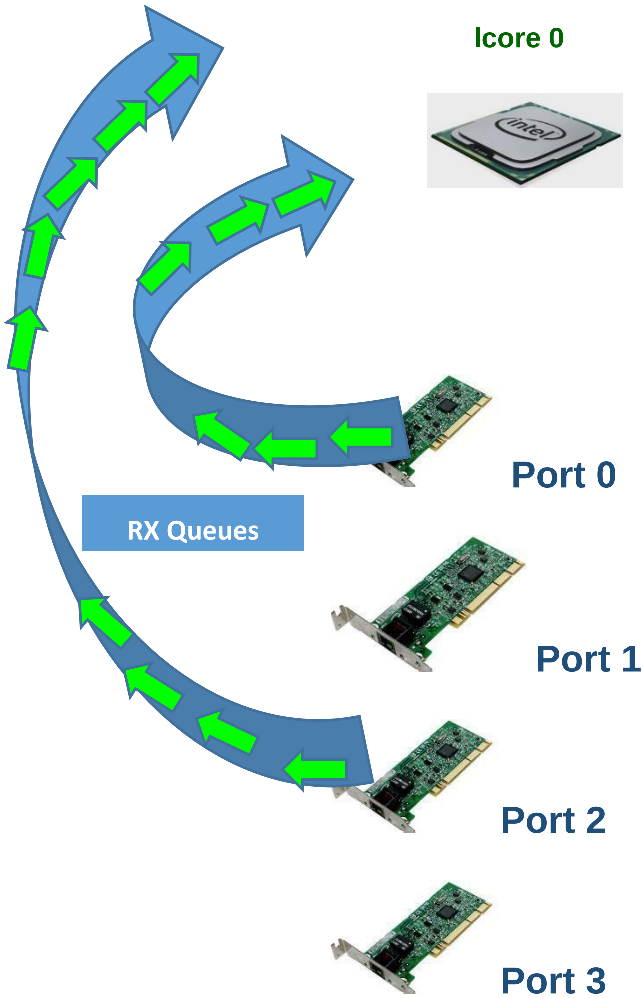
When you have multiple RX queues per port, each queue can be polled by only one lcore at a time. Thus, if you have 4 RX queues per port, you can have four cores simultaneously polling the port if you've configured one core per queue.

Can You Have Eight Cores and Four RX Queues per Port?
No, since that assigns more than one core per RX queue.

Can You Have Four Cores with Eight RX Queues per Port?
We can only answer this question by knowing full configuration details. Even though you have more RX queues than cores, if you have configured two cores for any single RX queue, that is not allowed. The key is not having more than one core per RX queue, irrespective of more queues in total available, compared to the number of cores.

Can One lcore Poll Multiple RX Queues?
Yes. One lcore can poll multiple RX queues. What is the maximum number of RX queues that one lcore can poll? That depends on performance requirements and how much headroom should be available for applications after servicing some number of queues. Packet size and packet arrival rates also constrain the cycle budget available on the core.

Note that with the port numbering in the system, one lcore can poll multiple RX queues that need not be necessarily consecutive. This is clear from the figure below. Lcore 0 polls RX Queue 0 and Rx Queue 2. It does not poll RX Queue 1 and RX Queue 3.



One lcore polling multiple RX queues.

Who is Responsible for Mutual Exclusion so that Multiple Cores Don't Work on the Same Receive Queue?

The one-line answer is—you—the application developer. All the functions of the Ethernet Device API exported by a PMD are lock-free functions which are not to be invoked in parallel on different logical cores to work on the same target object.

For instance, the receive function of a PMD cannot be invoked in parallel on two logical cores to poll the same RX queue [on the same port].

Of course, this function can be invoked in parallel by different logical cores on different RX queues.

Please note and be aware that it is the responsibility of the upper-level application to enforce this rule.

If you don't design your application to enforce this exclusion, allowing multiple cores to step on each other while accessing the device, you will get segmentation errors and crashes for sure. DPDK goes with lockless accesses for high performance and assumes that you, as a higher-level application developer, will ensure that multiple cores do not work on the same receive queue.

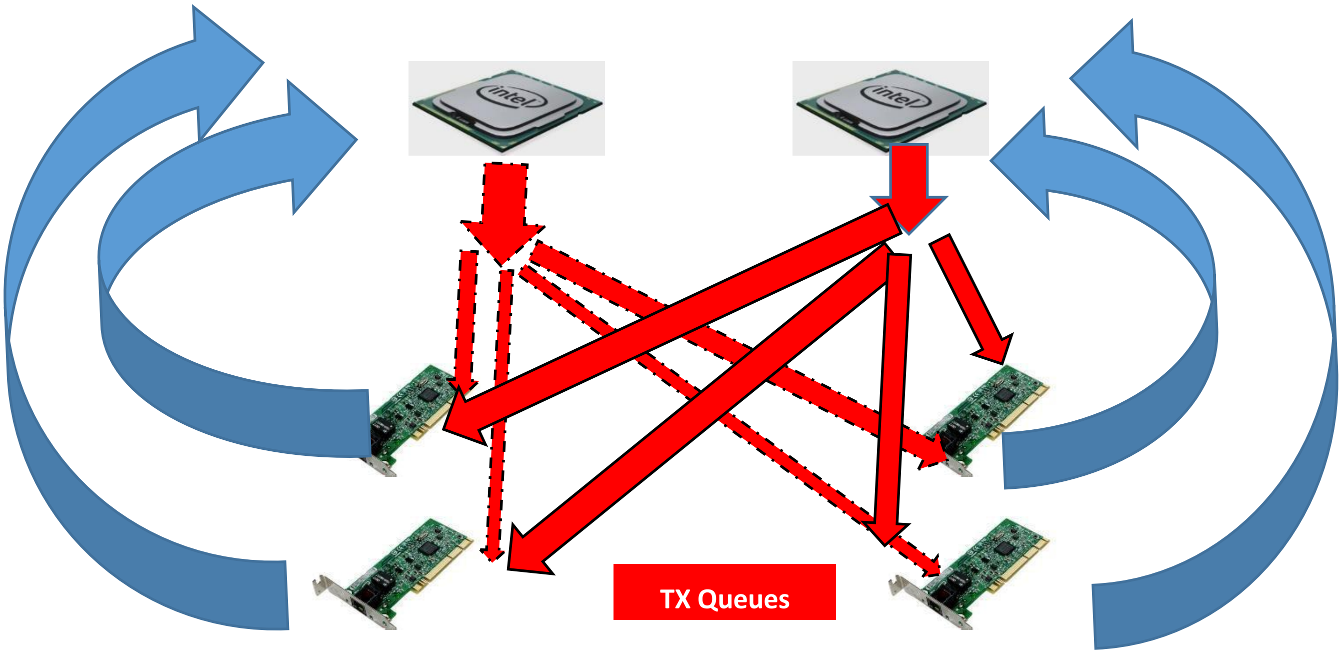
What if Your Design Requires Multiple Cores to Share Queues?

If needed, parallel accesses to shared cores by multiple logical cores must be explicitly protected by dedicated inline lock-aware functions built on top of their corresponding lock-free functions of the PMD API.

TX Port: Why Should Each Core be Able to Transmit on Each and Every Transmit Port?

We saw that for an RX queue, an lcore can only poll a subset of RX ports, but what about TX ports? Can an lcore connect only to a subset of TX ports in the system? Or should each and every lcore connect to all TX ports?

The answer is that a forwarding operation running on an lcore may result in a packet destined for *any* TX port in the system. Because of this, each lcore should be able to transmit to each and every TX port.



An lcore can poll only a subset of RX ports, but can transmit to any TX port in the system.

While the Data Plane can be Parallel, the Control Plane is Sequential

Control plane operations like device configuration, queue (RX and TX) setup, and device start depend on certain sequences to be followed. Hence, they are sequential.

Device Setup Sequence

To set up a device, follow this sequence:

```
rte_eth_dev_configure()
rte_eth_tx_queue_setup()
rte_eth_rx_queue_setup()
rte_eth_dev_start()
```

After that, the network application can invoke, in any order, the functions exported by the Ethernet API to get the MAC address of a given device, the speed and the status of a device physical link, receive/transmit packet bursts, and so on.

Summary

Application developers will benefit from understanding DPDK assumptions regarding application roles and responsibilities. To start, it's important to comprehend the scope of DPDK's roles and responsibilities. This will help you to correctly architect from the get-go in terms of thread safety, lockless API call usage, multiprocessor synchronization, and control plane and data plane synchronization.

Next Steps

Architect a couple of your own usage models of the data plane coexisting with the control and management plane. Look for similar approaches used by testpmd and other applications, and described by the DPDK HowTo Guides. Test them out.

Exercises

1. Can you have eight cores per port with four RX queues per port?
2. Can you have four cores per port with eight RX queues per port?
3. What are the implications of multiple cores transmitting on one transmit port—in terms of control plane and data plane synchronization?

4. Control plane operations—should it be done in interrupt context itself or as a deferred procedure?

DPDK Performance Optimization Guidelines White Paper

Abstract

This paper illustrates best-known methods and performance optimizations used in the [Data Plane Development Kit (DPDK).] DPDK application developers will benefit by implementing these optimization guidelines in their applications. A problem well stated is a problem half solved, thus the paper starts with profiling methodology to help identify the bottleneck in an application. Once the type of bottleneck is identified, this module will help you determine the optimization mechanism that DPDK uses to overcome the bottleneck. Specifically, we refer to the respective sample application and code snippet that implements the corresponding performance optimization technique. The module concludes with a checklist flowchart that DPDK developers and users can use to ensure they follow the guidelines given here.

For cookbook-style instructions on how to do hands-on performance profiling of your DPDK code with VTune™ tools, refer to the module Profiling DPDK Code with Intel VTune Amplifier.

Strategy and Methodology

A chain is really only as strong as its weakest link. So, the strategy is to use profiling tools to identify hotspots in the system. Once the hotspot is identified, the corresponding optimization technique is looked up for the sample application and code snippet as how it is already solved and implemented in the DPDK. Developers at this stage will implement those specific optimization techniques in their application. They can run respective micro-benchmarks and unit tests on applications provided with the DPDK.

Once the particular hotspot has been addressed, the application is again profiled to find the next hotspot in the system. The above methodology is repeated to the point of satisfaction in terms of achieving desired performance.

The performance optimization involves a gamut of considerations shown in the checklist below:

1. Optimize the BIOS settings.
2. Efficiently partition non-uniform memory access (NUMA) resources with improved locality in mind.
3. Optimize the Linux configuration.
4. To validate each configuration change, run *l3fwd*—as is with default settings—and compare with published performance numbers.
5. Run micro-benchmarks to pick and choose optimum high-performance components (for example, **bulk enqueue/bulk dequeue** as opposed to single enqueue/single dequeue).
6. Pick a sample application that is similar to the target appliance, using the already fine-tuned optimum default settings (***for example, more TX buffer resources than Rx***).
7. Adapt and update the sample application (**for example, # of queues**). Compile with the correct optimization flag levels.
8. Profile the chosen sample application in order to have a known good comparison base.
9. Run with optimized command-line options, keeping improved locality and concurrency in mind.
10. How to best match application and algorithm to underlying architecture? Run profiling to find memory-bound? I/O-bound? CPU-bound?
11. Apply the corresponding solution: Software prefetch for memory, block mode for I/O, to use Intel® Hyper-Threading Technology (Intel® HT Technology) or not, if the application is CPU-bound.
12. Rerun profiling—Front-end pipeline stall? Back-end pipeline stall?
13. Apply corresponding solution. Write efficient code—branch prediction, loop unroll, compiler optimization, and so on.
14. Still don't have desired performance? Back to #9.
15. Record best-known methods and share in dpdk.org.

Recommended Pre-reading

It is recommended that you read, at a minimum, the *DPDK Programmer's Guide*, and refer to the *DPDK Sample Application User Guides* before proceeding.

Please refer to other DPDK documents as needed.

BIOS Settings

To get repeatable performance, DPDK L3fwd performance numbers are achieved with the following BIOS settings:

NUMA	ENABLED
Enhanced Intel SpeedStep® technology	DISABLED
Processor C3	DISABLED
Processor C6	DISABLED
Intel® Hyper-Threading Technology	ENABLED
Intel® Virtualization Technology for Directed I/O	DISABLED
Intel® Memory Latency Checker (Intel® MLC) Streamer	ENABLED
Intel® MLC Spatial Prefetcher	ENABLED
DCU Data Prefetcher	ENABLED
DCU Instruction Prefetcher	ENABLED
CPU Power and Performance Policy	Performance
Memory Power Optimization	Performance Optimized
Memory RAS and Performance Configuration -> NUMA Optimized	ENABLED

Memory RAS and Performance Configuration -> NUMA Optimized

Please note that if the DPDK power management feature is to be used, Enhanced Intel SpeedStep® technology must be enabled. In addition, C3 and C6 should be enabled. However, to start with, it is recommended that you use the BIOS settings as shown in the table and run basic L3fwd to ensure that the BIOS, platform, and Linux settings are optimal for performance.

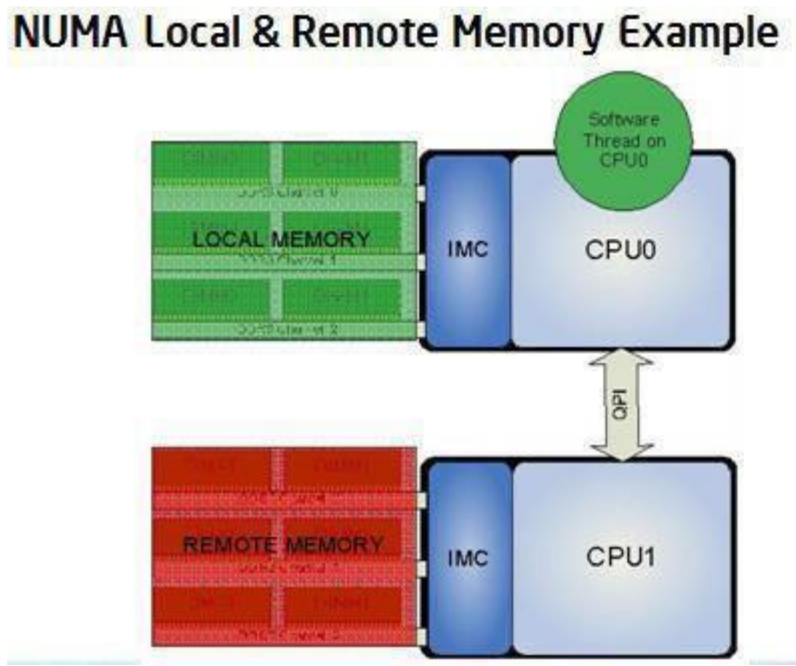
Refer to Intel document # 557159 titled *Intel Xeon processor E7-8800/4800 v3 Product Family*, for detailed understanding of BIOS setting and performance implications.

Platform Optimizations

Platform optimizations include (1) configuring memory, and (2) I/O (NIC Cards), to take advantage of affinity to achieve lower latency.

Platform Optimizations—NUMA and Memory Controller

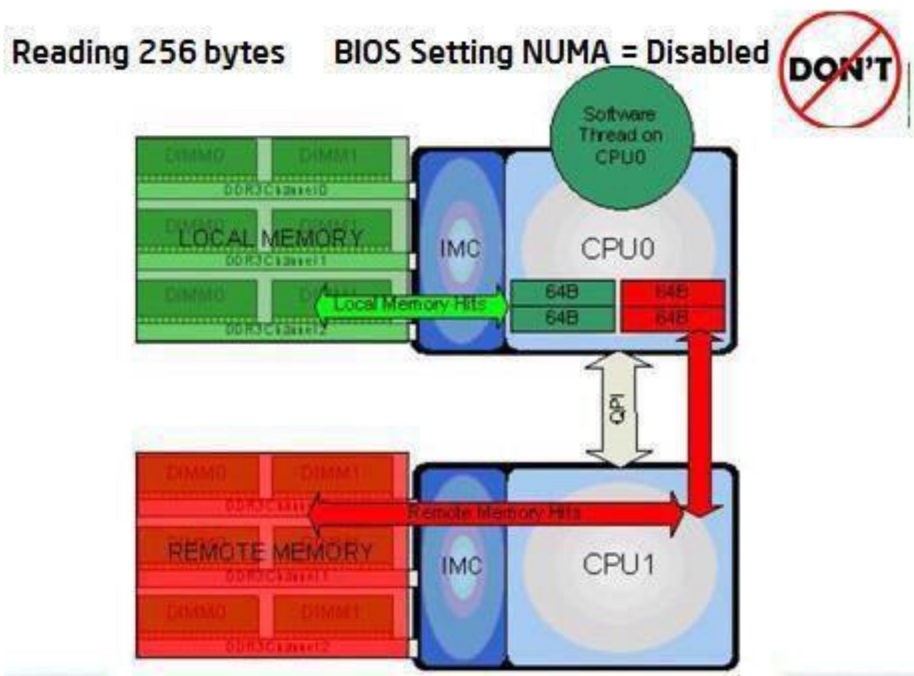
Below is an example of a multi (dual) socket system. For the threads that run on CPU0, all the memory accesses going to memory local to socket 0 result in lower latency. Any accesses that cross Intel® QuickPath Interconnect (Intel® QPI) to access remote memory (that is, memory local to socket 1) incurs additional latency and should be avoided.



Problem: What happens when NUMA is set to DISABLED in the BIOS? When NUMA is disabled in the BIOS, the memory controller interleaves the accesses across the sockets.

For example, as shown below, CPU0 is reading 256 bytes (four cache lines). With the BIOS NUMA state set to DISABLED, memory controller interleaves the access across the sockets. Out of 256 bytes, 128 bytes are read from local memory and 128 bytes are read from remote memory.

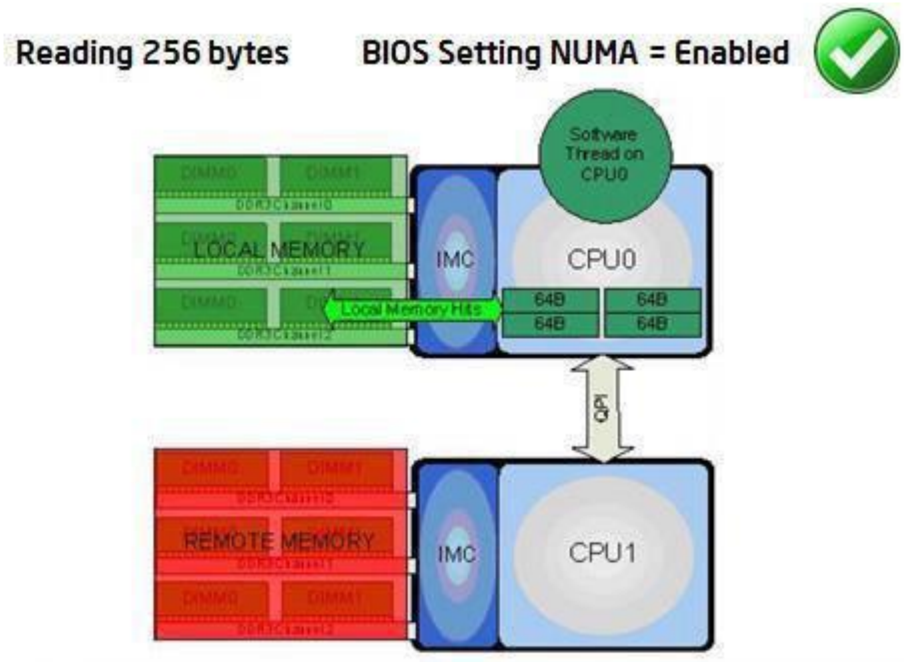
The remote memory accesses end up crossing the Intel QPI link. The impact of this is increased time for accessing remote memory, resulting in lower performance.



Solution: As shown below, with BIOS setting NUMA = Enabled, all the accesses go to the same socket (local) memory and there is no crossing of Intel QPI. This results in improved performance due to lower memory access latency.

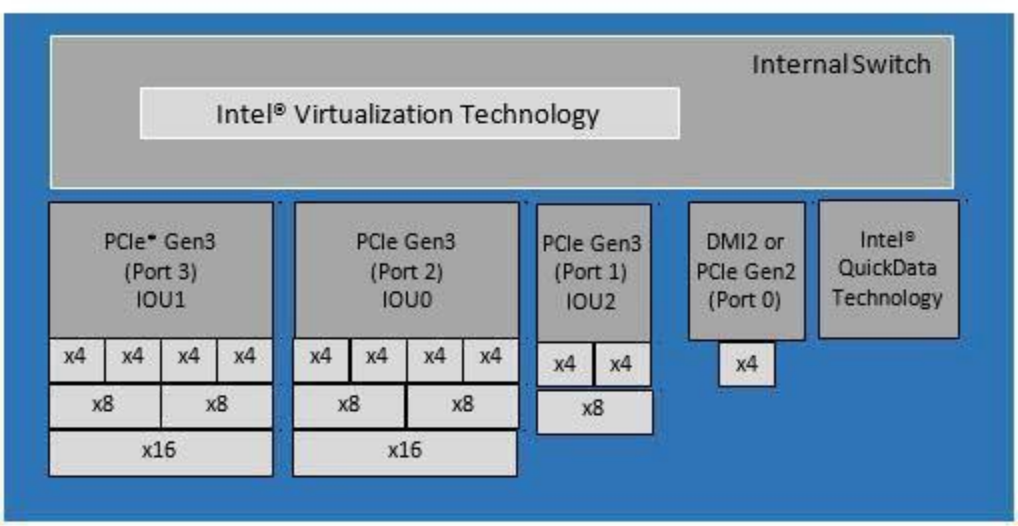
Key Take Away

Be sure to set NUMA = Enabled in the BIOS.



Platform optimizations—PCIe layout and IOU affinity.*

	Socket 0			Socket 1			
IOU	IOU-1	IOU-0	IOU-0	IOU-1	IOU-1	IOU-0	IOU-0
Gen3	X8 / x16	x8	x8	x8	x8	x8	x8
Slot	Slot1	Slot2	Slot3	Slot4	Slot5	Slot6	Slot7
Device	SFDA4	X520-SR2	SFDA4	X520-SR2	SFDA4	X520-SR2	SFDA4
	4x10GbE	2x10GbE	4x10GbE	2x10GbE	4x10GbE	2x10GbE	4x10GbE



Linux* Optimizations

Reducing Context Switches with isolcpus

To reduce the possibility of context switches, it is desirable to give a hint to the kernel to refrain from scheduling other *user space tasks* on to the cores used by DPDK application threads. The ***isolcpus*** Linux kernel parameter serves this purpose. For example, if DPDK applications are to run on logical cores 1, 2, and 3, the following should be added to the kernel parameter list:

```
isolcpus=1,2,3
```

Note: Even with the *isolcpus* hint, the scheduler may still schedule kernel threads on the isolated cores. Please note that ***isolcpus*** requires a reboot.

Adapt and Update the Sample Application

Now that the relevant sample application has been identified as a starting point to build the end product, the following are the next set of questions to be answered.

Configuration Questions

How to Configure the Application for Best Performance?

For example:

- How many queues can be configured per port?
- Can the same number of Tx and Rx resources be allocated?
- What are the optimal settings for threshold values?

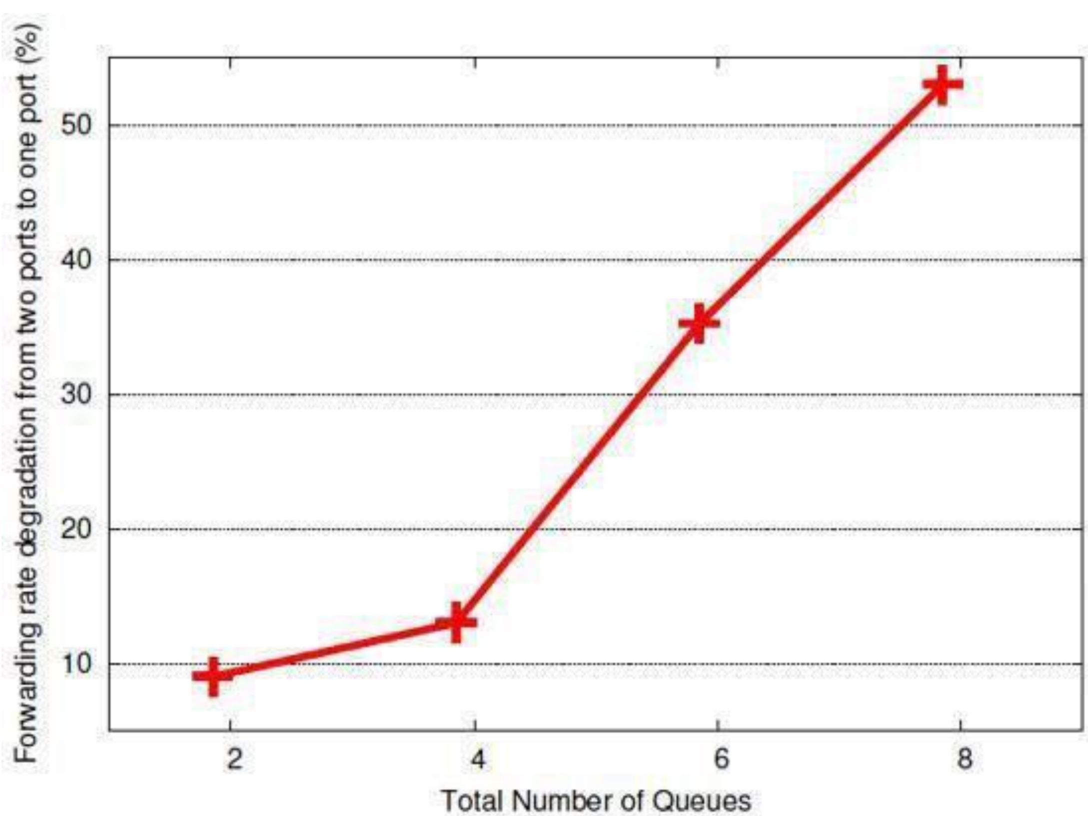
Recommendation: The good news is that each sample application comes with not only optimized code flow but also optimized parameters settings as default values. The recommendation is to use a similar ratio between resources for Tx and Rx. The following are the references and recommendations for the Intel® 82599 10 Gigabit Ethernet Controller. For other NIC controllers, please refer to the corresponding data sheets.

How Many Queues can be Configured per Port?

Please refer to the white paper Evaluating the Suitability of Server Network Cards for Software Routers for detailed test setup and configuration on this topic.

The following graph (from the above white paper) indicates that you should **not** use more than two to four queues per port since the performance degrades with a higher number of queues.

For the best-case scenario, the recommendation is to use one queue per port. In case more are needed, two queues per port can be considered, but not more than that.

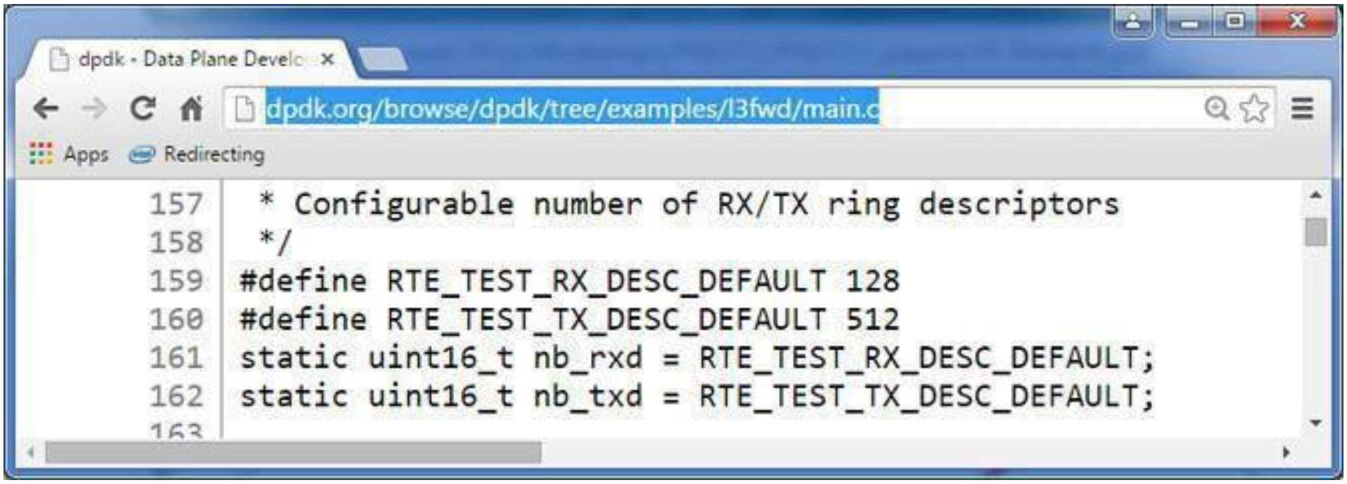


Ratio of the forwarding rate varying the number of hardware queues per port.

Can Tx Resources be Allocated the Same Size as Rx Resources?

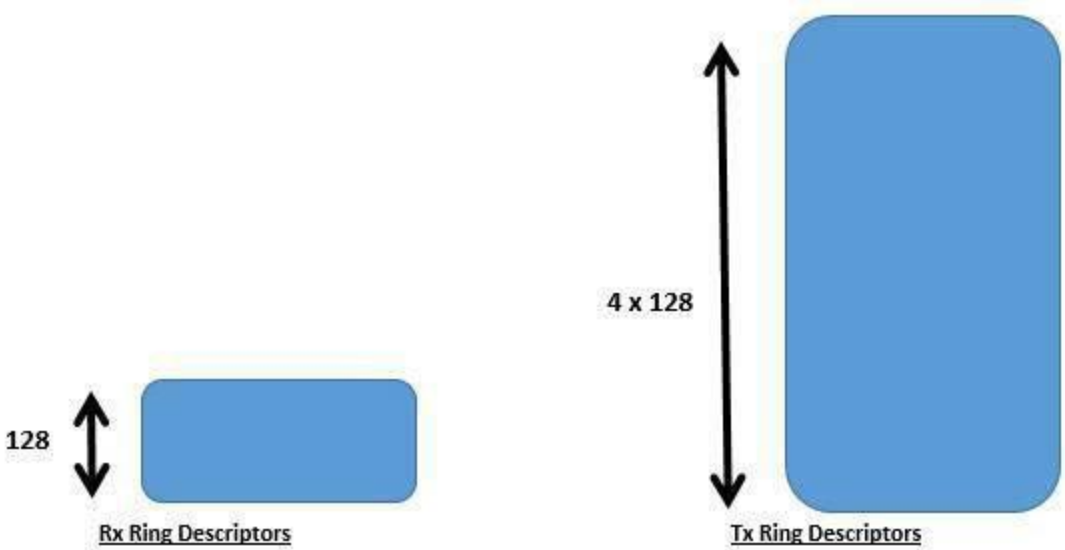
Please use as per the default values that are used in the application. For example, for Intel 82599 10-GbE Ethernet Controller, the default values are not equal; whereas for XL710, both RX and TX descriptors are of equal size.

Intel 82599 10-GbE Ethernet Controller: It is a natural tendency to allocate equal-sized resources for Tx and Rx. However, please note that http://git.dpdk.org/dpdk/tree/examples/l3fwd/main.c shows that optimal default size for the number of Tx ring descriptors is 512 as opposed to Rx ring descriptors being 128. Thus, the number of Tx ring descriptors is four times that of the Rx ring descriptors.

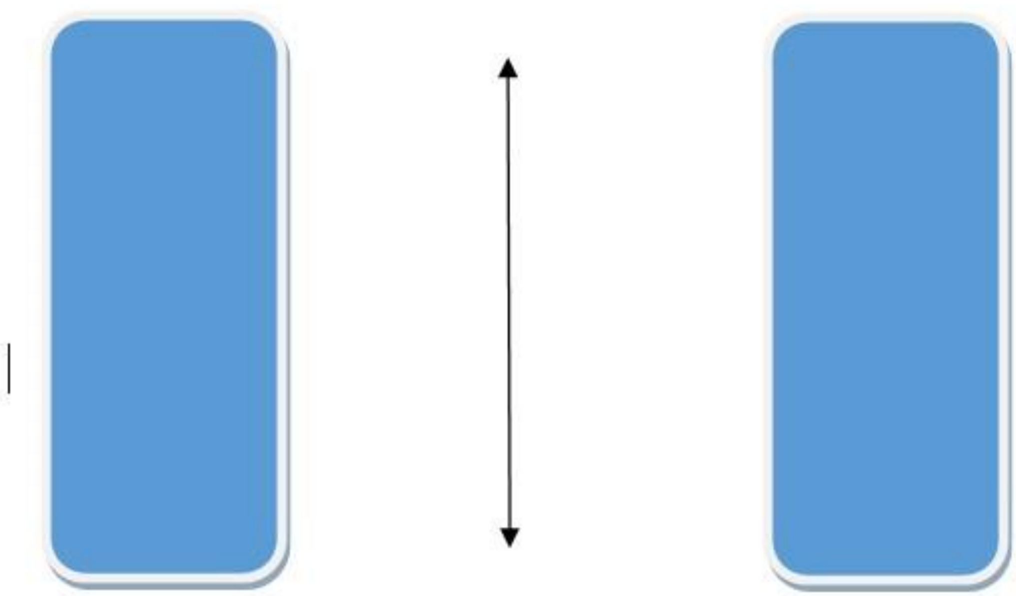


The recommendation is to choose Tx ring descriptors four times the size of Rx ring descriptors and not to have them both equal size. The reasoning for this is left as an exercise for the readers to find out.

Intel® 82599 10-GbE Ethernet Controller



However, for XL710 NIC [Equal Size RX and TX Descriptors]



RX Ring Descriptors TX Ring Descriptors

What are the Optimal Settings for Threshold Values?

For instance, http://git.dpdk.org/dpdk/tree/test/test/test_pmd_perf.c uses the following optimized default parameters for the Intel 82599 10-Gigabit Ethernet Controller.

```
/*
 * RX and TX Prefetch, Host, and Write-back threshold values should be
 * carefully set for optimal performance. Consult the network
 * controller's datasheet and supporting DPDK documentation for guidance
 * on how these parameters should be set.
 */
#define RX_PTHRESH 8 /**< Default values of RX prefetch threshold reg. */
#define RX_HTHRESH 8 /**< Default values of RX host threshold reg. */
#define RX_WTHRESH 0 /**< Default values of RX write-back threshold reg. */

/*
 * These default values are optimized for use with the Intel(R) 82599 10 GbE
 * Controller and the DPDK ixgbe PMD. Consider using other values for other
 * network controllers and/or network drivers.
 */
#define TX_PTHRESH 32 /**< Default values of TX prefetch threshold reg. */
#define TX_HTHRESH 0  /**< Default values of TX host threshold reg. */
#define TX_WTHRESH 0  /**< Default values of TX write-back threshold reg. */
```

Please refer to [Intel 82599 10-Gigabit Ethernet Controller: Datasheet]() for detailed explanations.

Rx_Free_Thresh—A Quick Summary and Key Takeaway

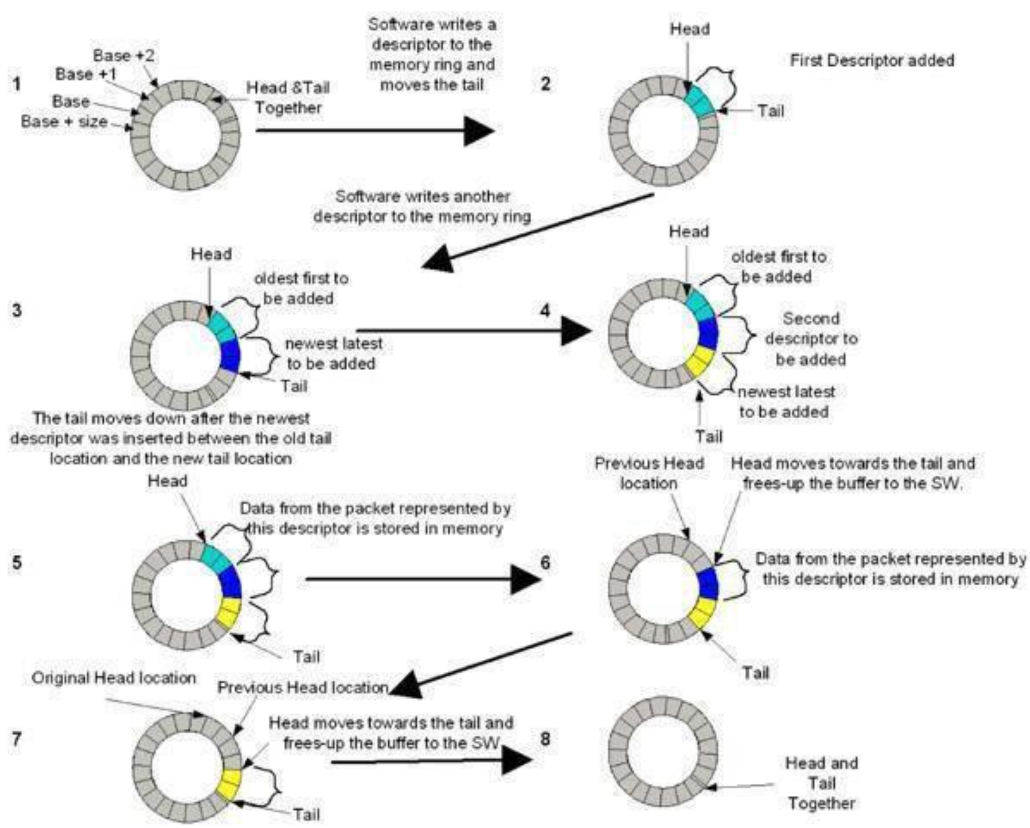
The key takeaway is amortization of the cost of the PCIe* operation of updating the hardware register is done by processing batches of packets before updating the hardware register.

Rx_Free_Thresh—In Detail

As shown below, communication of packets received by the hardware is done using a circular buffer of packet descriptors. There can be up to 64 K-8 descriptors in the circular buffer. Hardware maintains a shadow copy that includes those descriptors completed but not yet stored in memory.

The *Receive Descriptor Head register (RDH)* indicates the in-progress descriptor.

The *Receive Descriptor Tail register (RDT)* identifies the location beyond the last descriptor that the hardware can process. This is the location where software writes the first new descriptor.



During runtime, the software processes the descriptors and upon completion of a descriptor, increments the Receive Descriptor Tail (RDT) registers. However, updating the RDT after each packet has been processed by the software has a cost, as it increases PCIe operations.

Rx_free_thresh represents the maximum number of free descriptors that the DPDK software will hold before sending them back to the hardware. Hence, by processing batches of packets before updating the RDT, we can reduce the PCIe cost of this operation.

Fine tune with the parameters in the rte_eth_rx_queue_setup ( ) function for your configuration:

```
ret =
 rte_eth_rx_queue_setup(portid,
 0, rmnb_rxd,
socketid, &rx_conf,
 mbufpool[socketid]);
```

Compile With the Correct Optimization Flags

Apply the corresponding solution: Software prefetch for memory, block mode for I/O, to use Intel HT Technology for CPU-bound applications.

Software prefetch for memory helps to hide memory latency and thus improves memory-bound tasks in data plane applications.

PREFETCHW

Prefetch data into cache in anticipation of write: PREFETCHW, a new instruction from Intel® Xeon® processor E5-2650 v3 onward, hides memory latency and improves the network stack. PREFETCHW prefetches data into the cache in anticipation of a write.

PREFETCHWT1

Prefetch hint T1 (temporal L1 cache) with intent to write: PREFETCHWT1 fetches the data to a location in the cache hierarchy specified (T1 => temporal data with respect to first-level cache) by an intent to write a hint (so that data is brought into *Exclusive* state via a request for ownership) and a locality hint.

T1 (temporal data with respect to first-level cache)—prefetches data into the second-level cache.

For more information about these instructions refer to the [Intel® 64 and IA-32 Architectures Developer's Manual].

Running with Optimized Command-Line Options

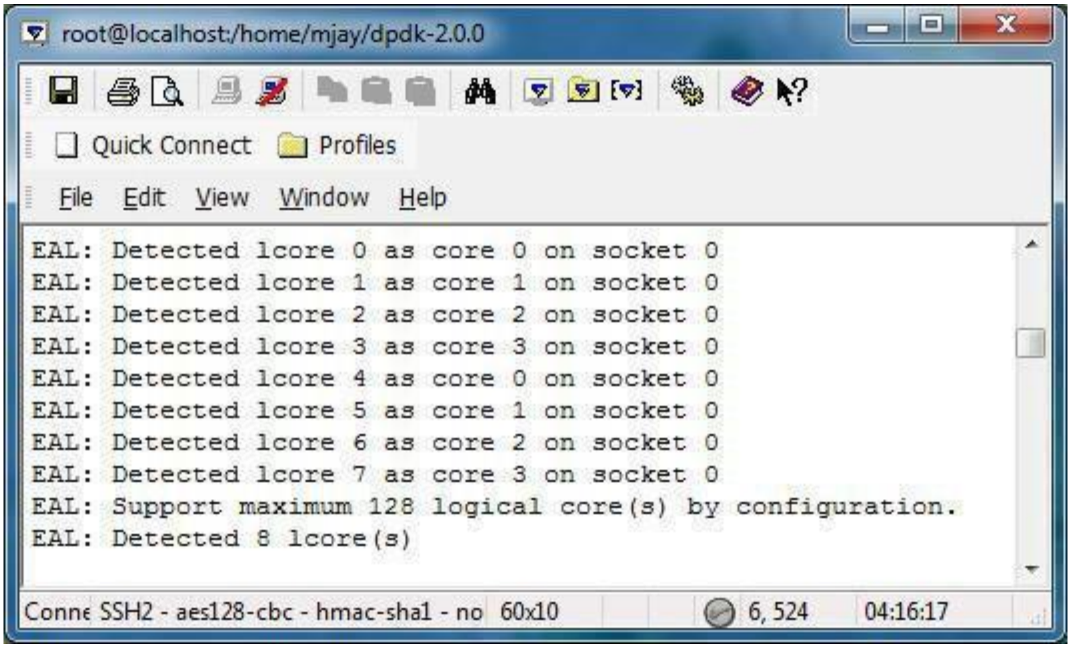
Optimize the application using command-line options to improve affinity, locality, and concurrency.

coremask Parameter and (Wrong) Assumption of Neighboring Cores

The *coremask* parameter is used with the DPDK application to specify the cores on which to run the application. For higher performance, reducing inter-processor communication cost is of key importance. The *coremask* should be selected such that the communicating cores are physical neighbors.

Problem: One may (mistakenly), assume core 0 and core 1 are neighboring cores and may choose the *coremask* accordingly in the DPDK command-line parameter. Please note that these logical core numbers, and their mapping to specific cores on specific NUMA sockets, can vary from platform to platform. While in one platform core 0 and core 1 may be neighbors, in another platform, core 0 and core 1 may end up being across another socket.

For instance, in a single-socket machine (screenshot shown below), lcore 0 and lcore 4 are siblings of the same physical core (core 0). So, the communication cost between lcore 0 and lcore 4 will be less than the communication cost between lcore 0 and lcore 1.



Solution: Because of this, it is recommended that the core layout for each platform be considered when choosing the *coremask* to use in each case.

Tools—dpdk/tools/cpu_layout.py

Use ./cpu_layout.py in the tools directory to find out the socket ID, the physical core ID, and the logical core ID (processor ID). From this information, correctly fill in the *coremask* parameter with locality of processors in mind.

Below is the cpu_layout of a dual-socket machine.

The list of physical cores is [0, 1, 2, 3, 4, 8, 9, 10, 11, 16, 17, 18, 19, 20, 24, 25, 26, 27]

Please note that physical core numbers 5, 6, 7, 12, 13, 14, 15, 21, 22, 23 are not in the list. This indicates that one ***cannot*** assume that the physical core numbers are sequential.

How to find out which lcores are using Intel HT Technology from the cpu_layout?

In the picture below, Lcore 1 and lcore 37 are hyper threads in socket 0. Assigning intercommunicating tasks to lcore 1 and lcore 37 will have lower cost and higher performance compared to assigning tasks to lcore 1 with any other core (other than lcore 37).

```
[root@localhost tools]# pwd
/home/mjay/dpdk-2.0.0/tools
[root@localhost tools]# ls
cpu_layout.py  dpdk_nic_bind.py  setup.sh
[root@localhost tools]# ./cpu_layout.py
============================================================
Core and Socket Information (as reported by '/proc/cpuinfo')
============================================================

cores =  [0, 1, 2, 3, 4, 8, 9, 10, 11, 16, 17, 18, 19, 20, 24, 25, 26, 27]
sockets =  [0, 1]

        Socket 0        Socket 1
        --------        --------
Core 0  [0, 36]         [18, 54]

Core 1  [1, 37]         [19, 55]

Core 2  [2, 38]         [20, 56]

Core 3  [3, 39]         [21, 57]

Core 4  [4, 40]         [22, 58]

Core 8  [5, 41]         [23, 59]

Core 9  [6, 42]         [24, 60]

Core 10 [7, 43]         [25, 61]

Core 11 [8, 44]         [26, 62]

Core 16 [9, 45]         [27, 63]

Core 17 [10, 46]        [28, 64]

Core 18 [11, 47]        [29, 65]

Core 19 [12, 48]        [30, 66]

Core 20 [13, 49]        [31, 67]

Core 24 [14, 50]        [32, 68]

Core 25 [15, 51]        [33, 69]

Core 26 [16, 52]        [34, 70]

Core 27 [17, 53]        [35, 71]

[root@localhost tools]#
```

Save core 0 for Linux use and do not use core 0 for the DPDK.

Refer below for the initialization of the DPDK application. Core 0 is being used by the master core.

```
EAL: Master core 0 is ready (tid=c9f8c880)
EAL: Core 4 is ready (tid=b4dfb700)
EAL: Core 3 is ready (tid=b55fc700)
EAL: Core 2 is ready (tid=b5dfd700)
EAL: Core 1 is ready (tid=b65fe700)
```

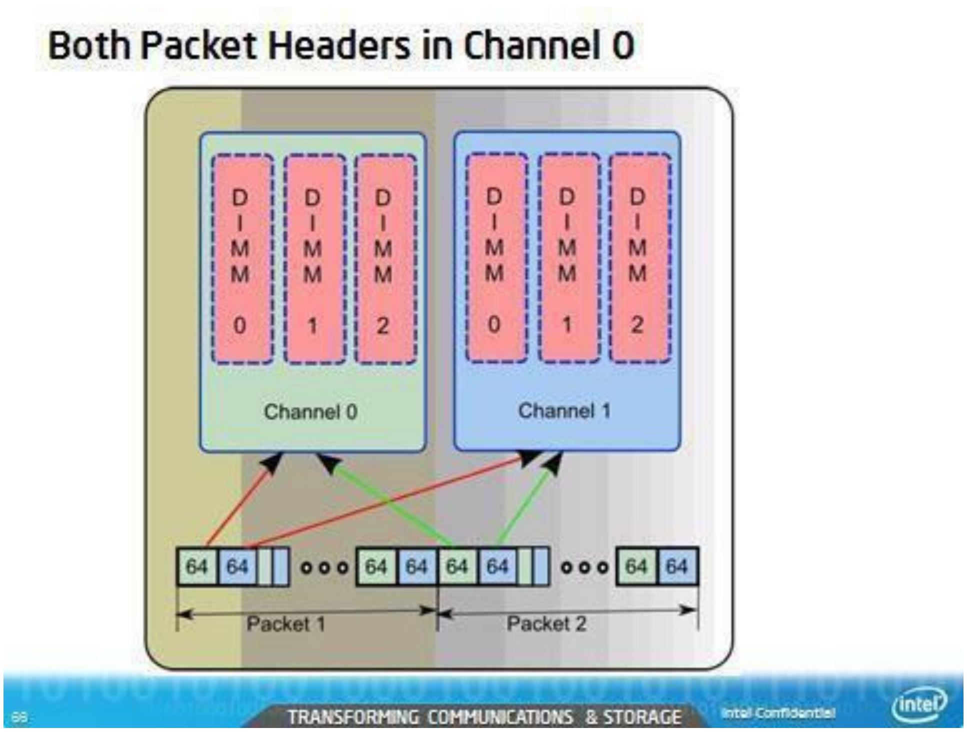
Do not use core 0 for the DPDK applications because it is used by Linux as the master core. For example, using *l3fwd –c 0x1* … should be avoided since that would be using core 0 (which is serving the functionality of the master core) for l3fwd DPDK application as well.

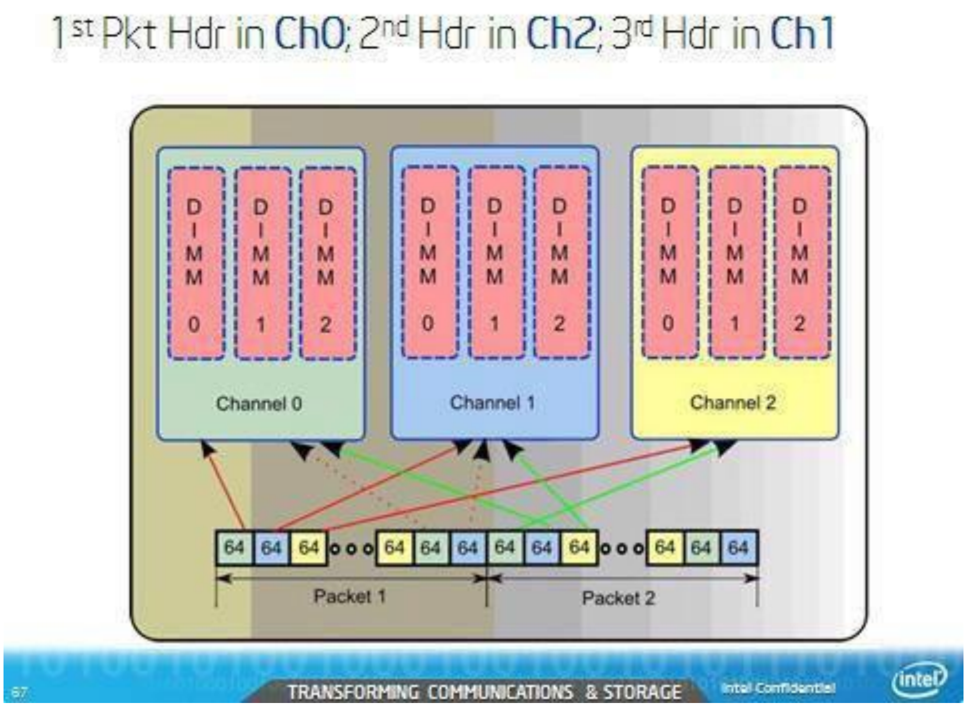
Instead, the command *l3fwd –c 0x2* …. can be used so that the l3fwd application uses core 1.

In realistic use cases like Open vSwitch* with DPDK, a control plane thread pins to the master core and is responsible for responding to control plane commands from the user or the SDN controller. So, the DPDK application should not use the master core (core 0), and the core bit mask in the DPDK command line should not set bit 0 for the *coremask*.

Correct use of the Channel Parameter

Be sure to make correct use of the channel parameter. For example, use CHANNEL PARAMETER N = 3 for a 3-channel memory system.





DPDK Micro-Benchmarks and Auto-Tests

DPDK micro-benchmarks and auto-tests are available as part of DPDK applications and examples. Developers use these micro-benchmarks to do focused measurements for evaluating performance.

The auto-tests are used for functionality verification.

The following are a few sample capabilities of distributor micro-benchmarks for performance evaluation.

Time_cache_line_switch ()

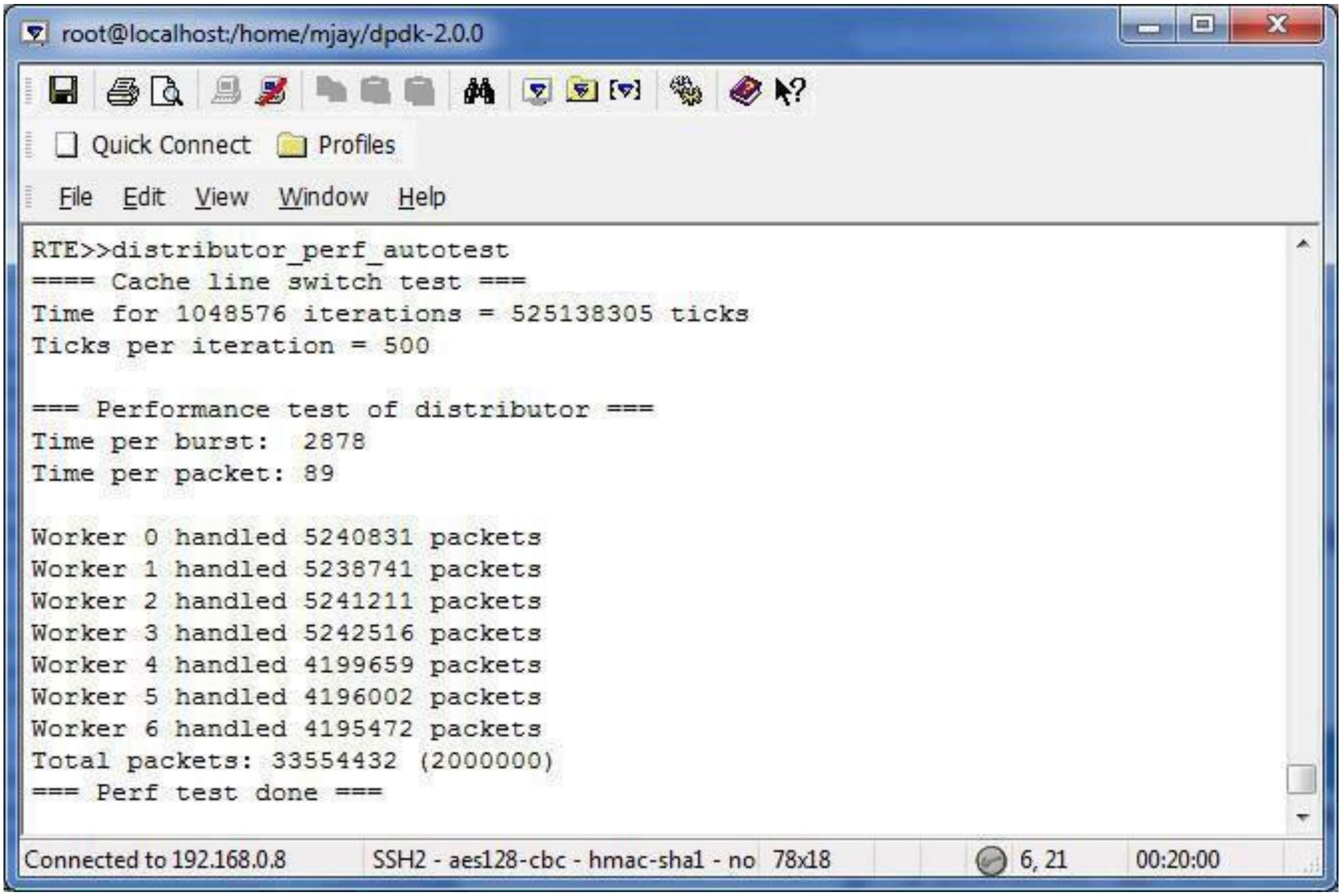
How can I measure the time taken for a cache line round-trip between two cores and back again?

The `time_cache_line_switch()` function in http://git.dpdk.org/dpdk/tree/test/test/test_distributor_perf.c can be used to time the number of cycles to round-trip a cache line between two cores and back again.

Perf_test()

How can I measure the processing time per packet?

The `perf_test()` function in http://git.dpdk.org/dpdk/tree/test/test/test_distributor_perf.c sends in 32 packets at a time to the distributor and verifies at the end that the worker thread got all of them, and finally how long the processing per packet took.



ring_perf_auto_test

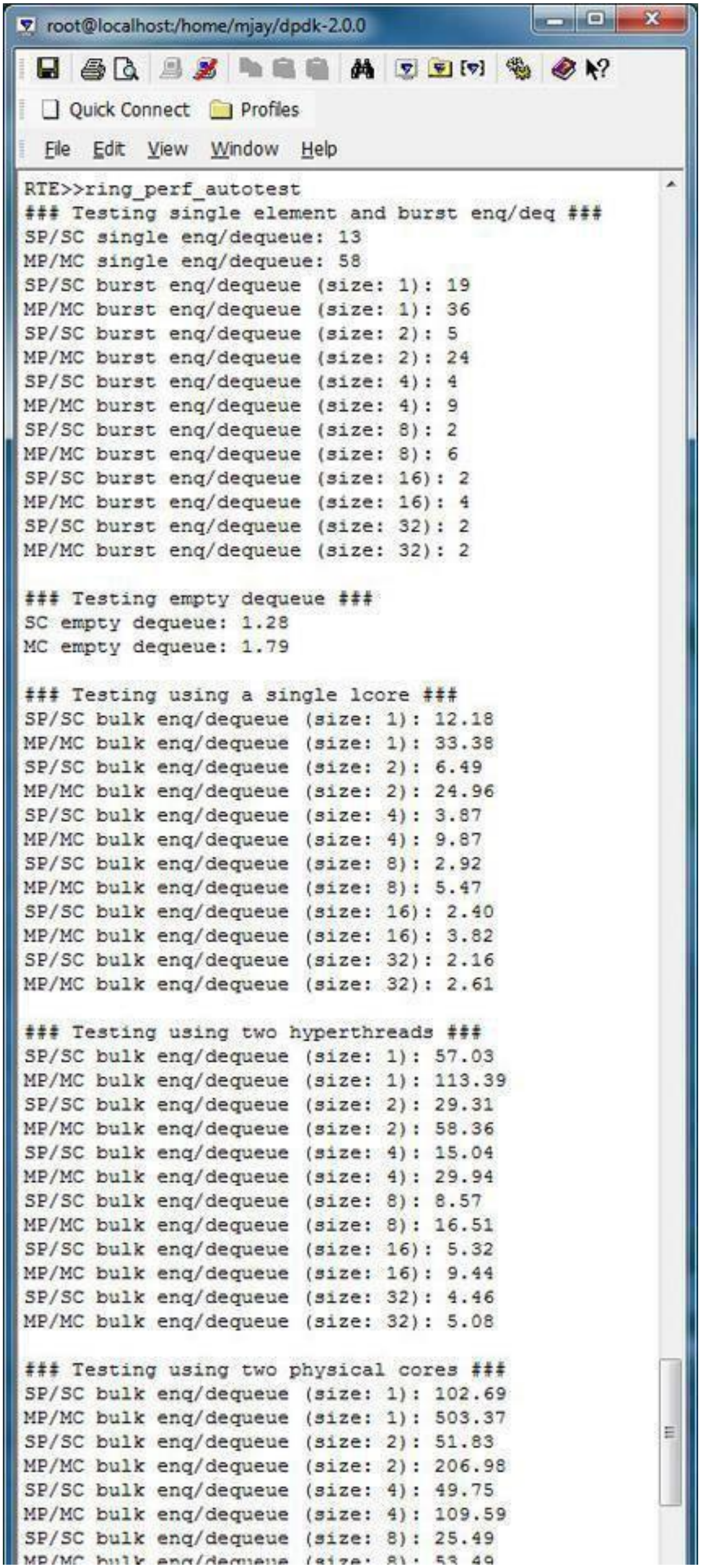
How can I find the performance difference between single producer/single consumer (sp/sc) and multi-producer/multi-consumer (mp/mc)?

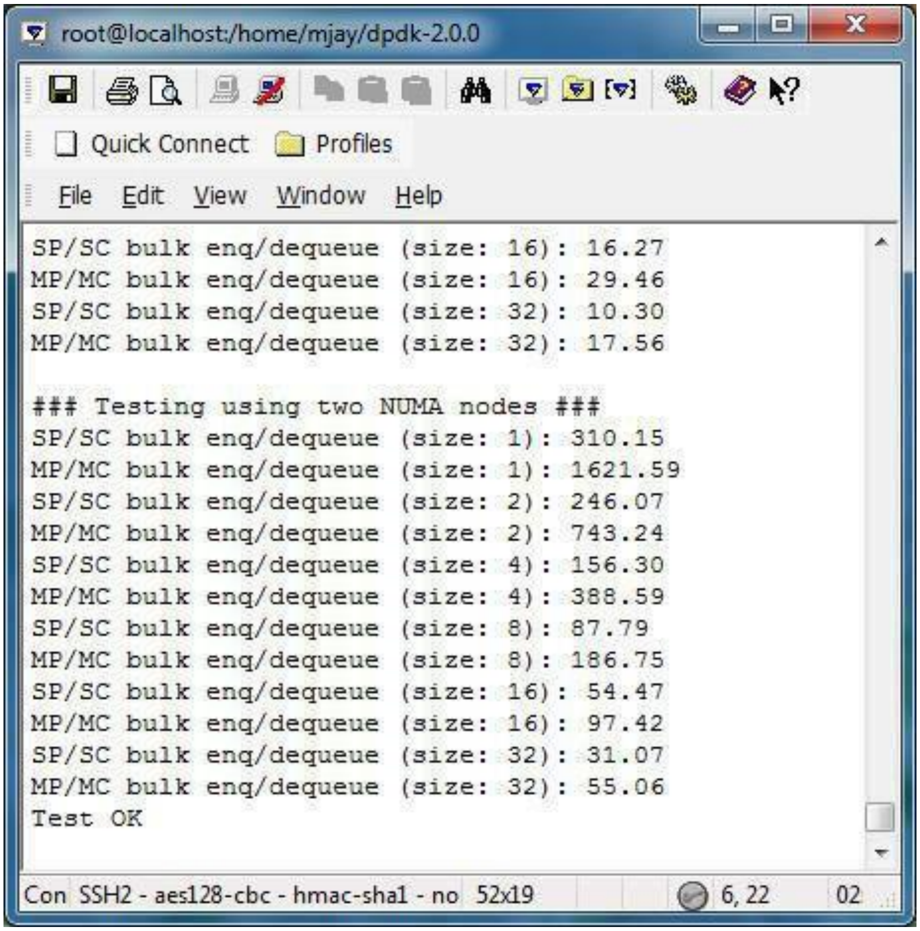
Running ring_perf_auto_test in /app/test gives the number of CPU cycles, which enables you to study the performance difference between single producer/single consumer and multi-producer/multi-consumer. It also shows the differences for different bulk sizes. See the following screenshot output.

The key takeaway: Using sp/sc with higher bulk sizes gives higher performance.

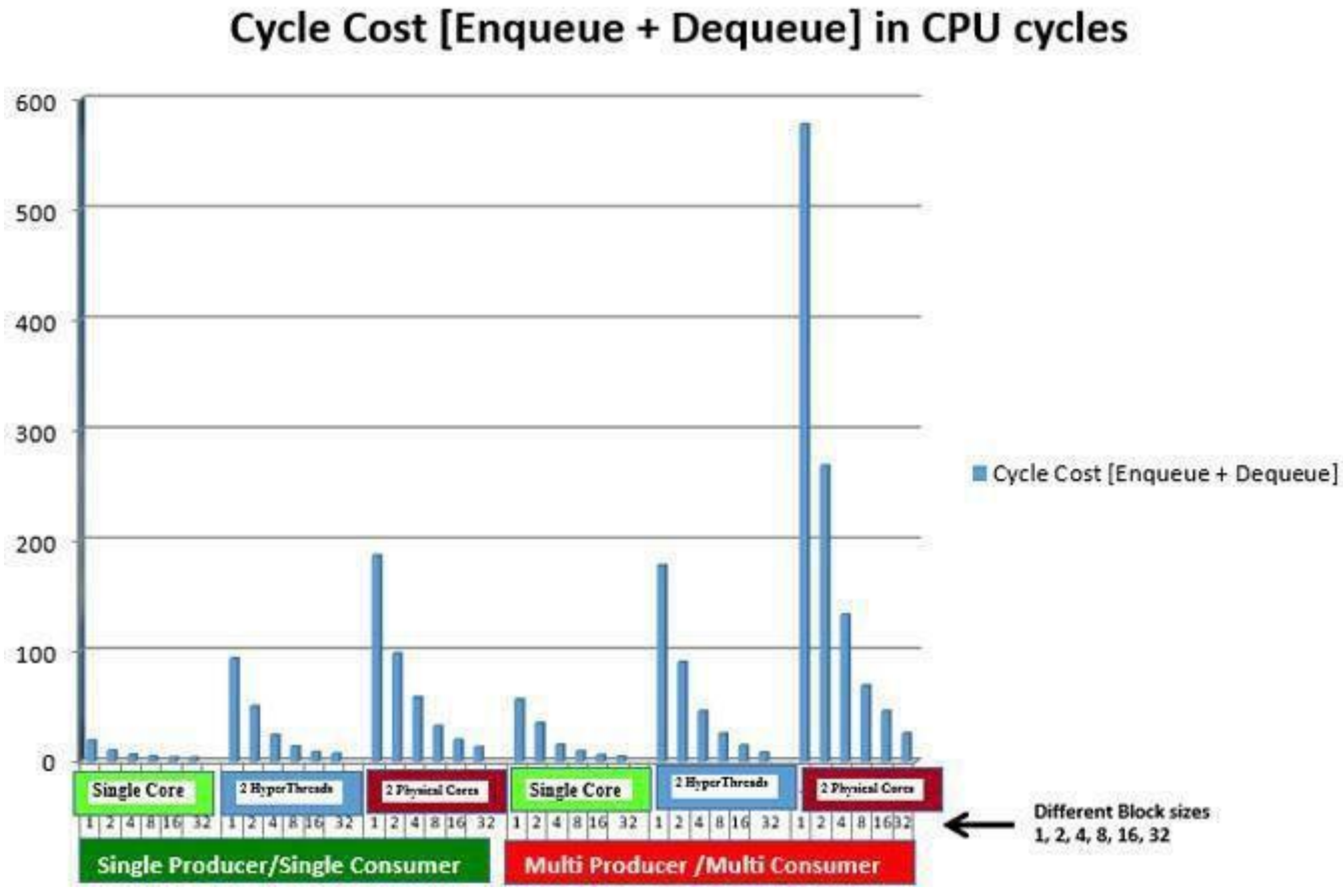
Please note that even though the default ring_perf_autotest runs through the performance test with block sizes of 8 and 32, one can update the source code to include other desired sizes (modify the array bulk_sizes[] to include bulk sizes of interest). For instance, find below the output with the block sizes 1, 2, 4, 8, 16, and 32.

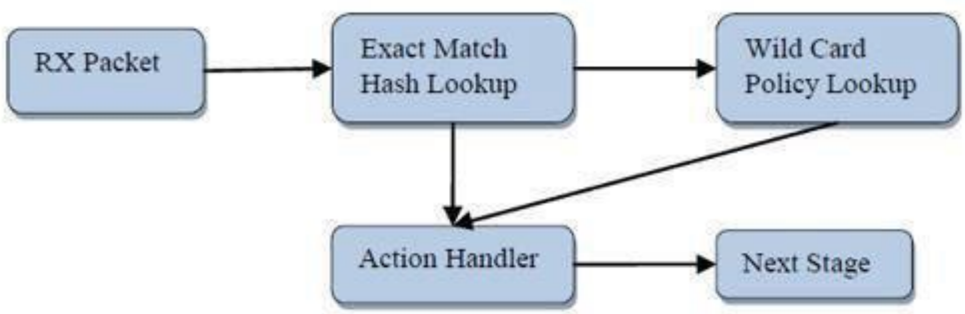
Two-Socket System—Huge Page Size = 2 Meg


root@localhost:/home/mjay/dpdk-2.0.0
Quick Connect
Profiles
File Edit View Window Help
RTE>>ring_perf_autotest
Testing single element and burst enq/deq
SP/SC single enq/dequeue: 13
MP/MC single enq/dequeue: 58
SP/SC burst enq/dequeue (size: 1): 19
MP/MC burst enq/dequeue (size: 1): 36
SP/SC burst enq/dequeue (size: 2): 5
MP/MC burst enq/dequeue (size: 2): 24
SP/SC burst enq/dequeue (size: 4): 4
MP/MC burst enq/dequeue (size: 4): 9
SP/SC burst enq/dequeue (size: 8): 2
MP/MC burst enq/dequeue (size: 8): 6
SP/SC burst enq/dequeue (size: 16): 2
MP/MC burst enq/dequeue (size: 16): 4
SP/SC burst enq/dequeue (size: 32): 2
MP/MC burst enq/dequeue (size: 32): 2
Testing empty dequeue
SC empty dequeue: 1.28
MC empty dequeue: 1.79
Testing using a single lcore
SP/SC bulk enq/dequeue (size: 1): 12.18
MP/MC bulk enq/dequeue (size: 1): 33.38
SP/SC bulk enq/dequeue (size: 2): 6.49
MP/MC bulk enq/dequeue (size: 2): 24.96
SP/SC bulk enq/dequeue (size: 4): 3.87
MP/MC bulk enq/dequeue (size: 4): 9.87
SP/SC bulk enq/dequeue (size: 8): 2.92
MP/MC bulk enq/dequeue (size: 8): 5.47
SP/SC bulk enq/dequeue (size: 16): 2.40
MP/MC bulk enq/dequeue (size: 16): 3.82
SP/SC bulk enq/dequeue (size: 32): 2.16
MP/MC bulk enq/dequeue (size: 32): 2.61
Testing using two hyperthreads
SP/SC bulk enq/dequeue (size: 1): 57.03
MP/MC bulk enq/dequeue (size: 1): 113.39
SP/SC bulk enq/dequeue (size: 2): 29.31
MP/MC bulk enq/dequeue (size: 2): 58.36
SP/SC bulk enq/dequeue (size: 4): 15.04
MP/MC bulk enq/dequeue (size: 4): 29.94
SP/SC bulk enq/dequeue (size: 8): 8.57
MP/MC bulk enq/dequeue (size: 8): 16.51
SP/SC bulk enq/dequeue (size: 16): 5.32
MP/MC bulk enq/dequeue (size: 16): 9.44
SP/SC bulk enq/dequeue (size: 32): 4.46
MP/MC bulk enq/dequeue (size: 32): 5.08
Testing using two physical cores
SP/SC bulk enq/dequeue (size: 1): 102.69
MP/MC bulk enq/dequeue (size: 1): 503.37
SP/SC bulk enq/dequeue (size: 2): 51.83
MP/MC bulk enq/dequeue (size: 2): 206.98
SP/SC bulk enq/dequeue (size: 4): 49.75
MP/MC bulk enq/dequeue (size: 4): 109.59
SP/SC bulk enq/dequeue (size: 8): 25.49


root@localhost:/home/mjay/dpdk-2.0.0
Quick Connect
Profiles
File
Edit
View
Window
Help
SP/SC bulk enq/dequeue (size: 16): 16.27
MP/MC bulk enq/dequeue (size: 16): 29.46
SP/SC bulk enq/dequeue (size: 32): 10.30
MP/MC bulk enq/dequeue (size: 32): 17.56

Testing using two NUMA nodes
SP/SC bulk enq/dequeue (size: 1): 310.15
MP/MC bulk enq/dequeue (size: 1): 1621.59
SP/SC bulk enq/dequeue (size: 2): 246.07
MP/MC bulk enq/dequeue (size: 2): 743.24
SP/SC bulk enq/dequeue (size: 4): 156.30
MP/MC bulk enq/dequeue (size: 4): 388.59
SP/SC bulk enq/dequeue (size: 8): 87.79
MP/MC bulk enq/dequeue (size: 8): 186.75
SP/SC bulk enq/dequeue (size: 16): 54.47
MP/MC bulk enq/dequeue (size: 16): 97.42
SP/SC bulk enq/dequeue (size: 32): 31.07
MP/MC bulk enq/dequeue (size: 32): 55.06
Test OK
Con SSH2 - aes128-cbc - hmac-sha1 - no 52x19
6, 22
02



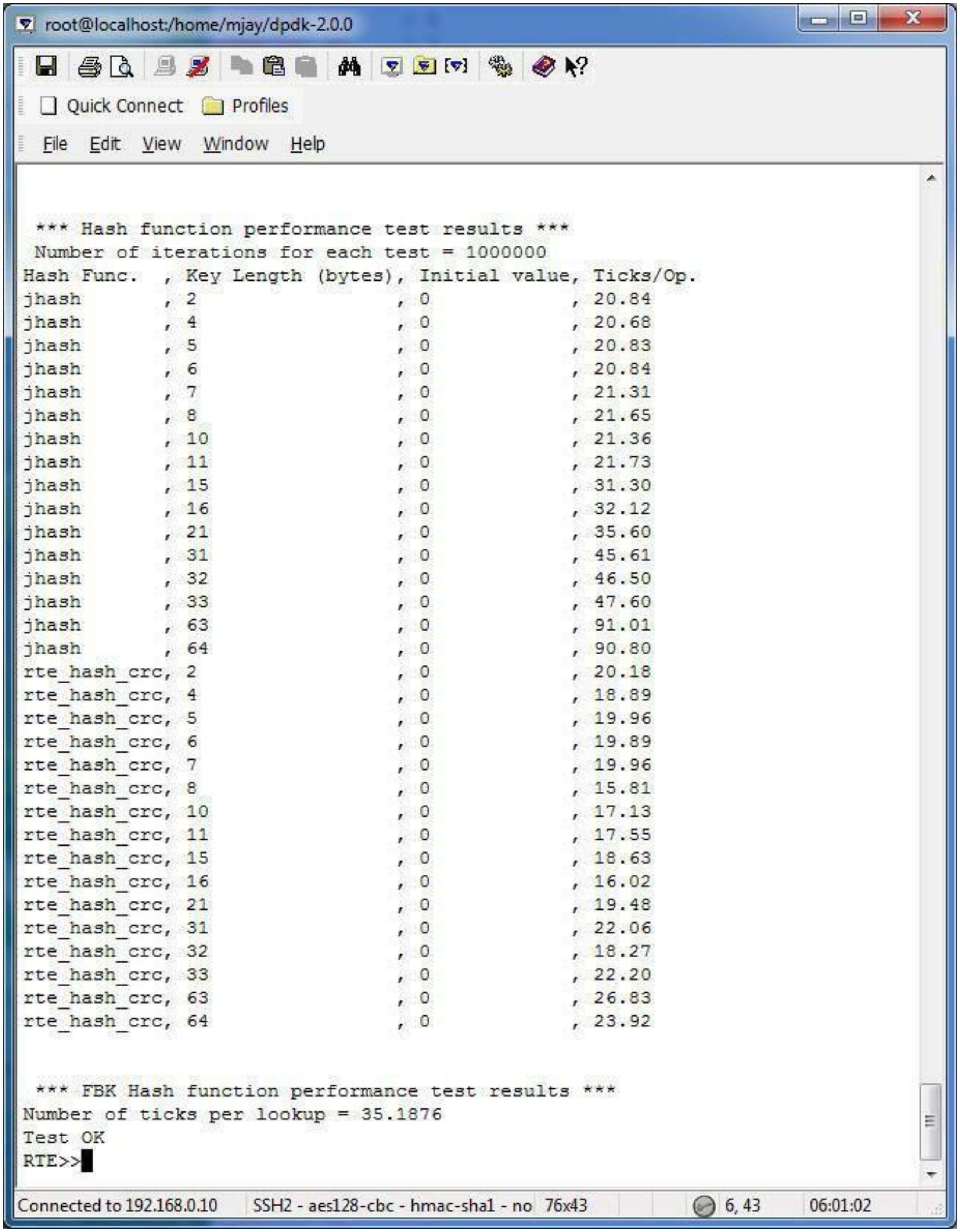


hash_perf_autotest runs through 1,000,000 iterations for each test, varying the following parameters, and reports Ticks/Op for each combination shown in the table below:

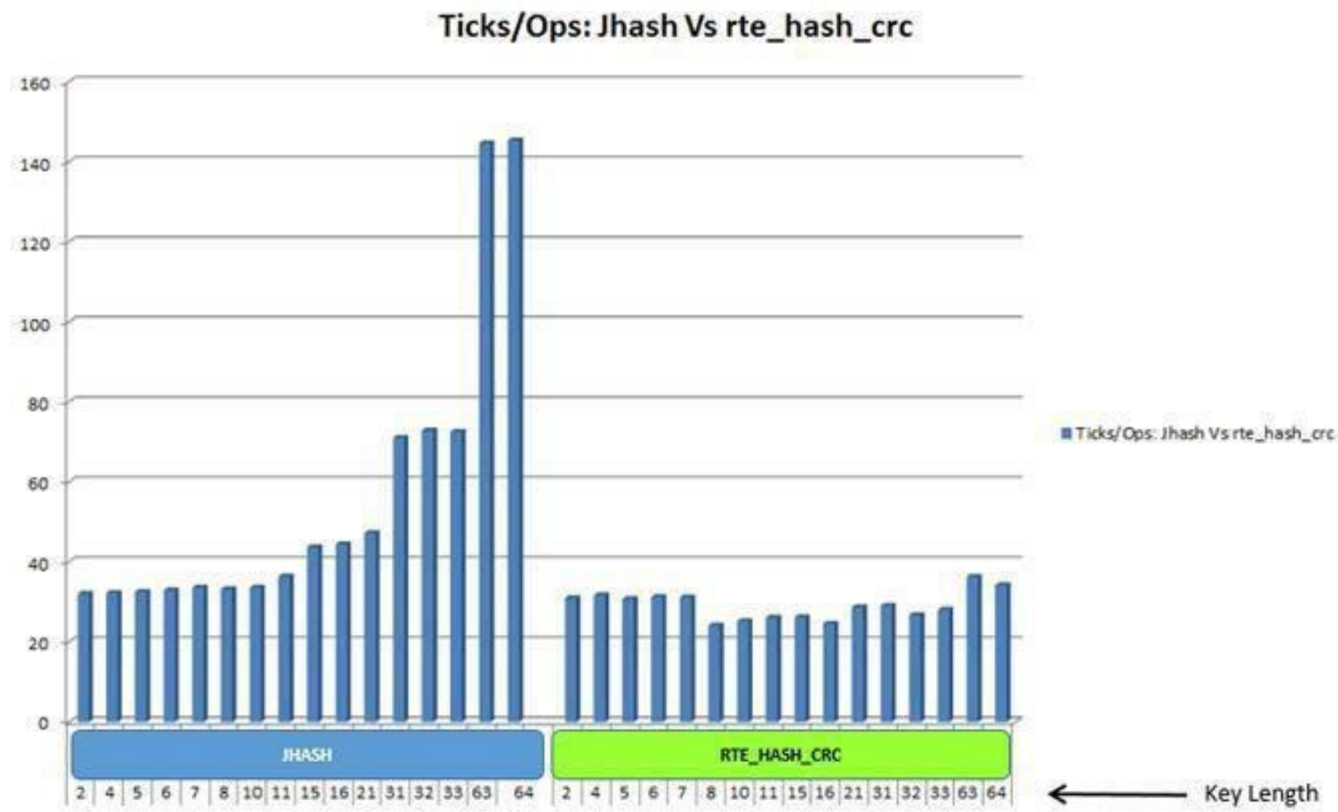
Hash Function	Operation	Key Size (bytes)	Entries	Entries per Bucket
a) Jhash b) Rte_hash_CRC	a) Add on Empty b) Add Update c) Look up	a) 16 b) 32 c) 48	a) 1024, b) 1048576	a) 1 b) 2 c) 4

		d) 64		d) 8 e) 16

The Detailed Test Output section contains detailed test output and the commands you can use to evaluate performance with your platform. The summary of the result is tabulated and charted below:




```
 *** Hash function performance test results ***
 Number of iterations for each test = 1000000
Hash Func.  , Key Length (bytes), Initial value, Ticks/Op.
jhash       , 2                 , 0            , 20.84
jhash       , 4                 , 0            , 20.68
jhash       , 5                 , 0            , 20.83
jhash       , 6                 , 0            , 20.84
jhash       , 7                 , 0            , 21.31
jhash       , 8                 , 0            , 21.65
jhash       , 10                , 0            , 21.36
jhash       , 11                , 0            , 21.73
jhash       , 15                , 0            , 31.30
jhash       , 16                , 0            , 32.12
jhash       , 21                , 0            , 35.60
jhash       , 31                , 0            , 45.61
jhash       , 32                , 0            , 46.50
jhash       , 33                , 0            , 47.60
jhash       , 63                , 0            , 91.01
jhash       , 64                , 0            , 90.80
rte_hash_crc, 2                 , 0            , 20.18
rte_hash_crc, 4                 , 0            , 18.89
rte_hash_crc, 5                 , 0            , 19.96
rte_hash_crc, 6                 , 0            , 19.89
rte_hash_crc, 7                 , 0            , 19.96
rte_hash_crc, 8                 , 0            , 15.81
rte_hash_crc, 10                , 0            , 17.13
rte_hash_crc, 11                , 0            , 17.55
rte_hash_crc, 15                , 0            , 18.63
rte_hash_crc, 16                , 0            , 16.02
rte_hash_crc, 21                , 0            , 19.48
rte_hash_crc, 31                , 0            , 22.06
rte_hash_crc, 32                , 0            , 18.27
rte_hash_crc, 33                , 0            , 22.20
rte_hash_crc, 63                , 0            , 26.83
rte_hash_crc, 64                , 0            , 23.92


 *** FBK Hash function performance test results ***
Number of ticks per lookup = 35.1876
Test OK
RTE>>
```



DPDK Micro-Benchmarks and Auto-Tests

	Focus Area to Improve	Use These Micro-Benchmarks and Auto-Tests
1	Ring for Inter-Core Communication	Performance comparison of bulk enqueue/bulk dequeue versus single enqueue/single dequeue on a single core
		To measure and compare performance between Intel® HT Technology, cores, and sockets doing bulk enqueue/bulk dequeue on pairs of cores
		Performance of dequeue from an empty ring: http://git.dpdk.org/dpdk/tree/test/test/test_ring_perf.c
		Single producer, single consumer – 1 Object, 2 Objects, MAX_BULK Objects – enqueue/dequeue
		Multi-producer, multi-consumer – 1 Object, 2 Objects, MAX BULK Objects – enqueue/dequeue
		Tx Burst - http://git.dpdk.org/dpdk/tree/test/test/test_ring.c Rx Burst - http://git.dpdk.org/dpdk/tree/test/test/test_pmd_ring.c
2	Memcopy	Cache to cache
		Cache to memory
		Memory to memory
		Memory to cache
		http://git.dpdk.org/dpdk/tree/test/test/test_memcpy_perf.c

3	Mempool	"n_get_bulk", "n_put_bulk"
		1 core, 2 cores, max cores with cache objects
		1 core, 2 cores, max cores without cache objects
		http://git.dpdk.org/dpdk/tree/test/test/test_mempool.c
5	Hash	Rte_jhash, rte_hash_crc;
		Add
		Lookup
		Update
		http://git.dpdk.org/dpdk/tree/test/test/test_hash_perf.c
6	ACL Lookup	http://git.dpdk.org/dpdk/tree/test/test/test_acl.c
7	LPM	Rule with depth > 24 1) Add, 2) Lookup, 3) Delete http://git.dpdk.org/dpdk/tree/test/test/test_lpm.c
		http://git.dpdk.org/dpdk/tree/test/test/test_lpm6.c
		Large Route Tables:
		http://git.dpdk.org/dpdk/tree/test/test/test_lpm6_data.h
8	Packet Distribution	http://git.dpdk.org/dpdk/tree/test/test/test_distributor_perf.c
9	NIC I/O Benchmark	Measure Tx Only
		Measure Rx Only,
		Measure Tx and Rx
		Benchmarks Network I/O Pipe - NIC h/w + PMD http://git.dpdk.org/dpdk/tree/test/test/test_pmd_perf.c
10	NIC I/O + Increased CPU processing	Increased CPU processing – NIC h/w + PMD + hash/lpm Examples/l3fwd
11	Atomic Operations/ Lock-rd/wr	http://git.dpdk.org/dpdk/tree/test/test/test_atomic.c
		http://git.dpdk.org/dpdk/tree/test/test/test_rwlock.c
12	SpinLock	Takes global lock, displays something, then releases the global lock
		Takes per-lcore lock, displays something, then releases the per-core lock
		http://git.dpdk.org/dpdk/tree/test/test/test_spinlock.c
13	Software Prefetch	http://git.dpdk.org/dpdk/tree/test/test/test_prefetch.c Its usage: http://git.dpdk.org/dpdk/tree/lib/librte_table/rte_table_hash_ext.c
14	Packet Distribution	http://git.dpdk.org/dpdk/tree/test/test/test_distributor_perf.c
15	Reorder and Seq. Window	http://git.dpdk.org/dpdk/tree/test/test/test_reorder.c
16	Software Load Balancer	http://git.dpdk.org/dpdk/tree/examples/load_balancer
17	ip_pipeline	Using the packet framework to build a pipeline: http://git.dpdk.org/dpdk/tree/test/test/test_table.c

		ACL Using Packet Framework
		http://git.dpdk.org/dpdk/tree/test/test/test_table_acl.c
18	Re-entrancy	http://git.dpdk.org/dpdk/tree/test/test/test_func_reentrancy.c
19	mbuf	http://git.dpdk.org/dpdk/tree/test/test/test_mbuf.c
20	memzone	http://git.dpdk.org/dpdk/tree/test/test/test_memzone.c
21	Virtual PMD	http://git.dpdk.org/dpdk/tree/test/test/virtual_pmd.c
22	QoS	http://git.dpdk.org/dpdk/tree/test/test/test_meter.c
		http://git.dpdk.org/dpdk/tree/test/test/test_red.c
		http://git.dpdk.org/dpdk/tree/test/test/test_sched.c
23	Link Bonding	http://git.dpdk.org/dpdk/tree/test/test/test_link_bonding.c
24	Kni	1. Transmit
		2. Receive to / from kernel space
		3. Kernel requests
		http://git.dpdk.org/dpdk/tree/test/test/test_kni.c
25	Malloc	http://git.dpdk.org/dpdk/tree/test/test/test_malloc.c
26	Debug	http://git.dpdk.org/dpdk/tree/test/test/test_debug.c
27	Timer	http://git.dpdk.org/dpdk/tree/test/test/test_cycles.c
28	Alarm	http://git.dpdk.org/dpdk/tree/test/test/test_alarm.c

Compiler Optimizations

Reference: PySter*—Compiler design and construction—“Adding optimizations to a compiler is a lot like eating chicken soup when you have a cold. Having a bowl full never hurts, but who knows if it really helps. If the optimizations are structured modularly so that the addition of one does not increase compiler complexity, the temptation to fold in another is hard to resist. How well the techniques work together or against each other is hard to determine."

Performance Optimization and Weakly Ordered Considerations

Background: Linux kernel synchronization primitives contain needed memory barriers as shown below (both uniprocessor and multiprocessor versions):

Smp_mb ( )	Memory barrier
Smp_rmb ( )	Read memory barrier
Smp_wmb ( )	Write memory barrier
Smp_read_barrier_depends (	Forces subsequent operations that depend on prior operations

)	to be ordered
Mmiowb ( )	Ordering on MMIO writes that are guarded by global spinlocks

Code that uses standard synchronization primitives (spinlocks, semaphores, read copy updates) should not need explicit memory barriers, since any required barriers are already present in these primitives.

Challenge: If you are writing code bypassing these standard synchronization primitives for optimization purposes, then consider your requirement in using the proper barrier.

Consideration: x86 provides a *process ordering* memory model in which writes from a given CPU are seen in order by all CPUs, and weak consistency, which permits arbitrary reordering, limited only by explicit memory-barrier instructions.

The *smp_mp ( ), smp_rmb ( ), smp_wmb ( )* primitives also force the compiler to avoid any optimizations that would have the effect of reordering memory optimizations across the barriers.

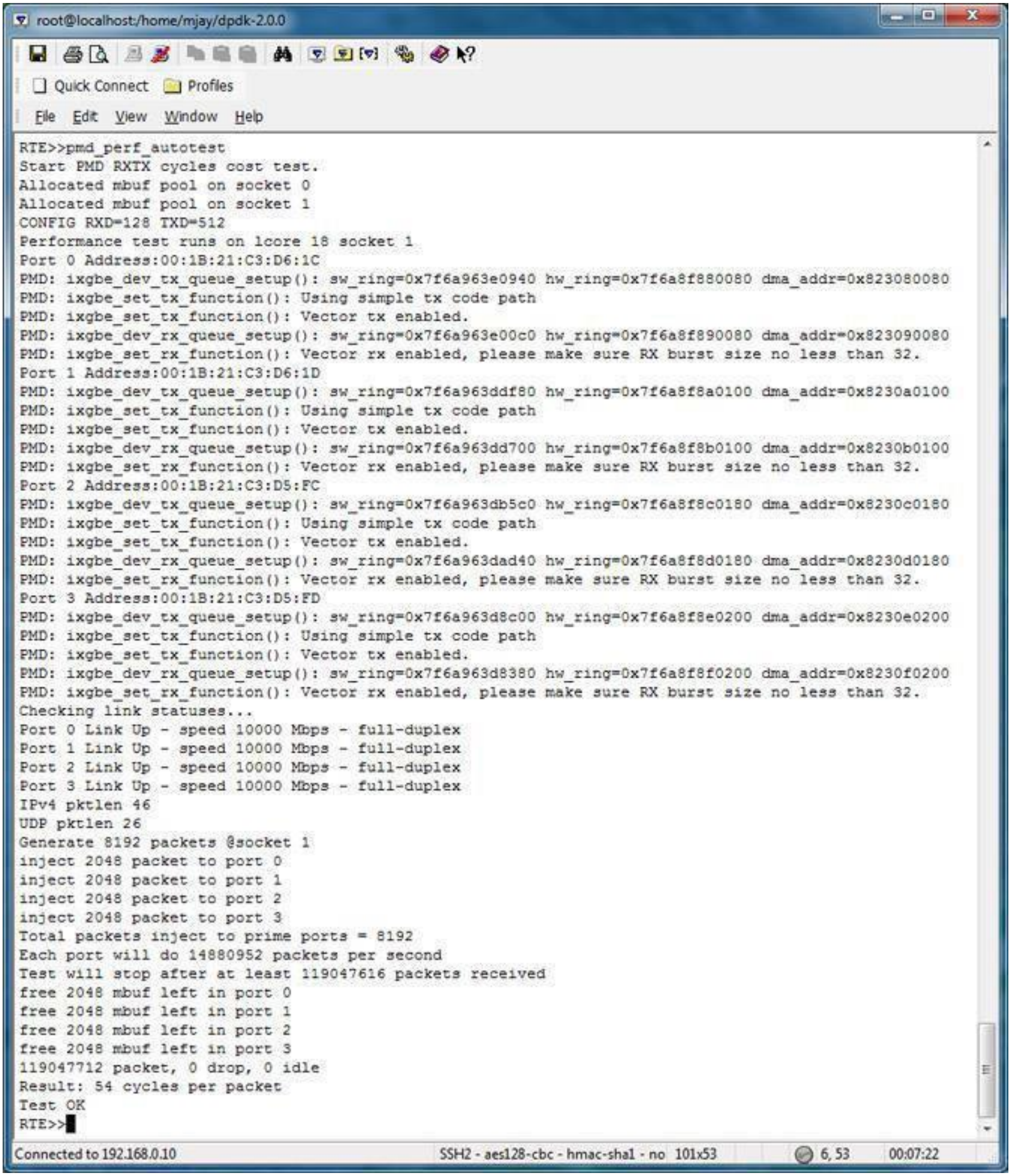
Some Intel® Streaming SIMD Extensions (SSE) instructions are weakly ordered (clflush and non-temporal move instructions). CPUs that have SSE can use mfence for smp mb(), lfence for smp rmb(), and sfence for smp wmb().

Detailed Test Output

Pmd_perf_autotest

To evaluate your platform's performance, run /app/test/pmd_perf_autotest.

The key takeaway: The cost for RX+TX cycles per packet in test Polled Mode Driver is 54 cycles with 4 ports and –n = 4 memory channels.




```
RTE>>pmd_perf_autotest
Start PMD RXTX cycles cost test.
Allocated mbuf pool on socket 0
Allocated mbuf pool on socket 1
CONFIG RXD=128 TXD=512
Performance test runs on lcore 18 socket 1
Port 0 Address:00:1B:21:C3:D6:1C
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963e0940 hw_ring=0x7f6a8f880080 dma_addr=0x823080080
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963e00c0 hw_ring=0x7f6a8f890080 dma_addr=0x823090080
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 1 Address:00:1B:21:C3:D6:1D
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963ddf80 hw_ring=0x7f6a8f8a0100 dma_addr=0x8230a0100
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963dd700 hw_ring=0x7f6a8f8b0100 dma_addr=0x8230b0100
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 2 Address:00:1B:21:C3:D5:FC
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963db5c0 hw_ring=0x7f6a8f8c0180 dma_addr=0x8230c0180
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963dad40 hw_ring=0x7f6a8f8d0180 dma_addr=0x8230d0180
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 3 Address:00:1B:21:C3:D5:FD
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963d8c00 hw_ring=0x7f6a8f8e0200 dma_addr=0x8230e0200
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963d8380 hw_ring=0x7f6a8f8f0200 dma_addr=0x8230f0200
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Checking link statuses...
Port 0 Link Up - speed 10000 Mbps - full-duplex
Port 1 Link Up - speed 10000 Mbps - full-duplex
Port 2 Link Up - speed 10000 Mbps - full-duplex
Port 3 Link Up - speed 10000 Mbps - full-duplex
IPv4 pktlen 46
UDP pktlen 26
Generate 8192 packets @socket 1
inject 2048 packet to port 0
inject 2048 packet to port 1
inject 2048 packet to port 2
inject 2048 packet to port 3
Total packets inject to prime ports = 8192
Each port will do 14880952 packets per second
Test will stop after at least 119047616 packets received
free 2048 mbuf left in port 0
free 2048 mbuf left in port 1
free 2048 mbuf left in port 2
free 2048 mbuf left in port 3
119047712 packet, 0 drop, 0 idle
Result: 54 cycles per packet
Test OK
RTE>>
```

What if you need to find the cycles taken for only RX? Or only TX?

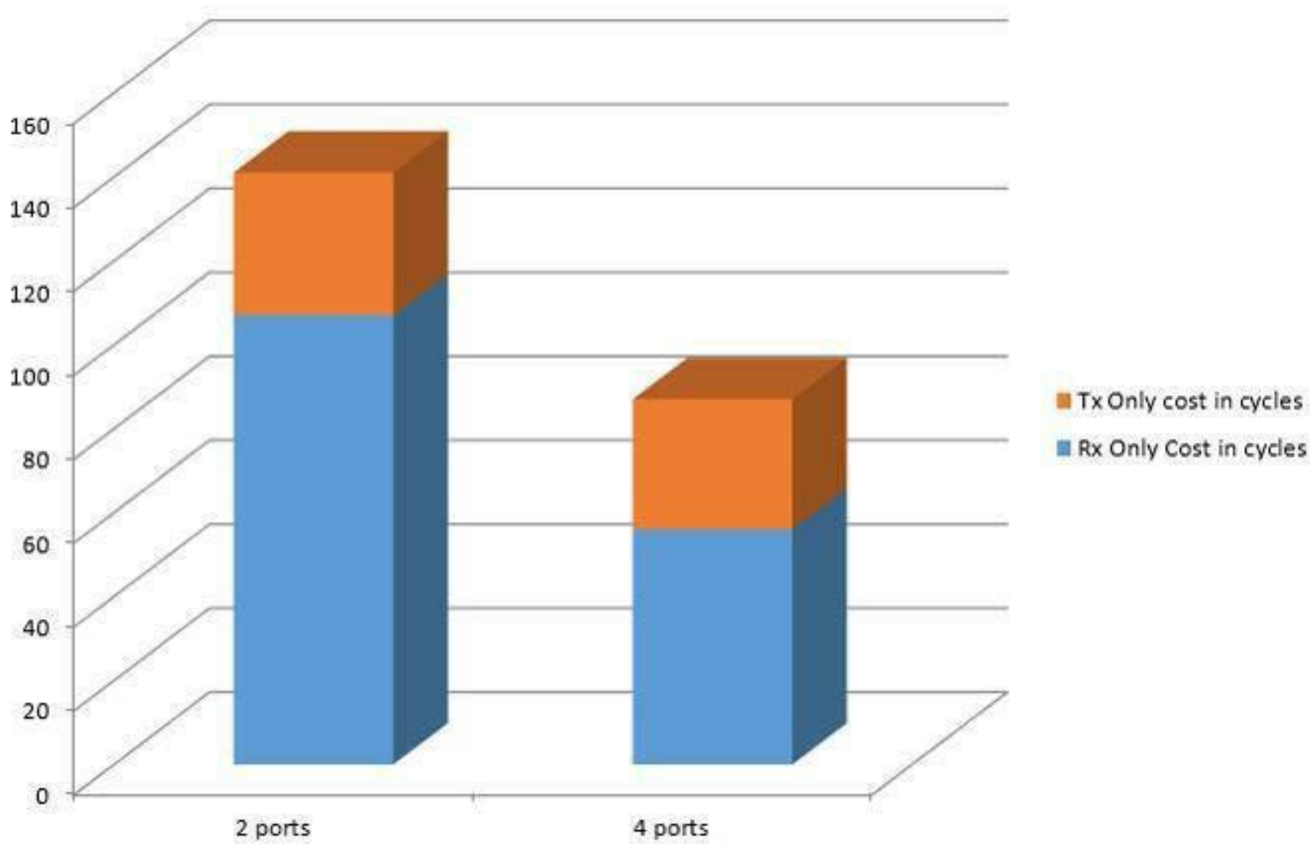
To find RX-only time, use the command *set_rxtx_anchor rxonly* before issuing the command pmd_perf_autotest.

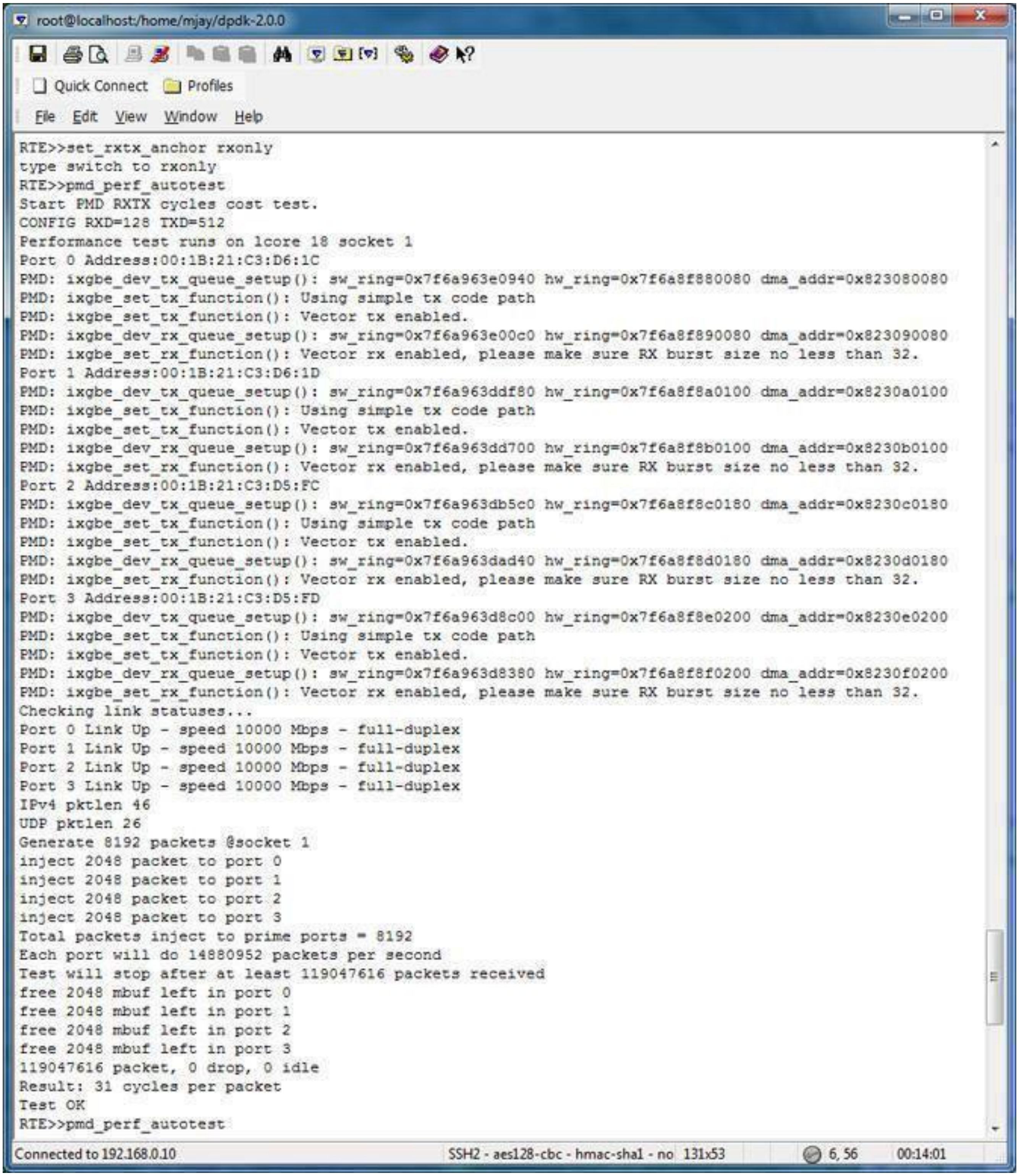
Similarly, to find TX-only time, use the command *set_rxtx_anchor txonly* before issuing the command pmd_perf_autotest.

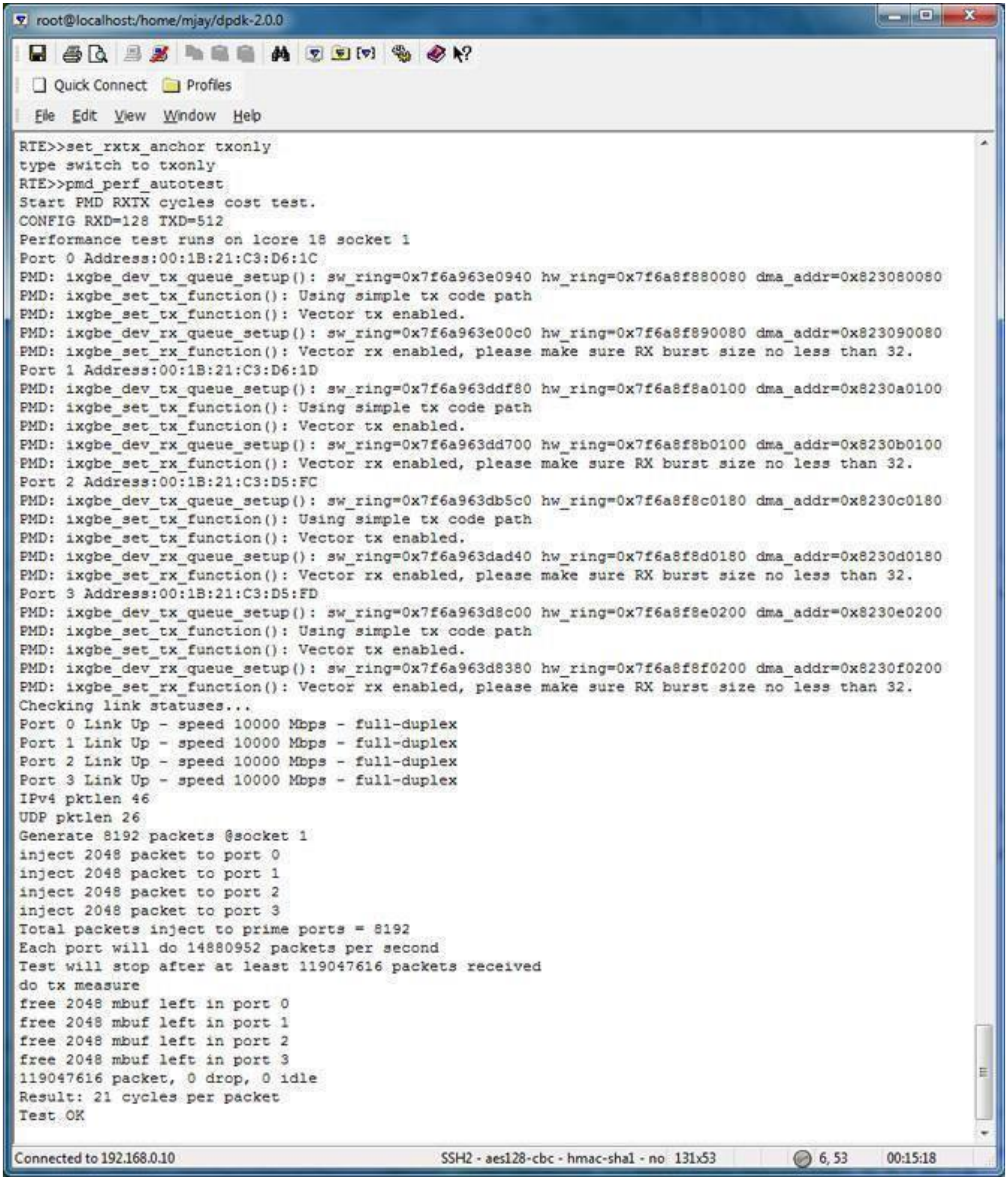
Packet Size = 64B # **of channels** n= 4

# of cycles per packet	TX+RX Cost	TX only Cost	Rx only Cost
With four ports	54 cycles	21 cycles	31 cycles

Below is the screen output for the *rxonly* and *txonly* cost, respectively.




root@localhost:/home/mjay/dpdk-2.0.0
Quick Connect
Profiles
File Edit View Window Help
RTE>>set_rxtx_anchor rxonly
type switch to rxonly
RTE>>pmd_perf_autotest
Start PMD RXTX cycles cost test.
CONFIG RXD=128 TXD=512
Performance test runs on lcore 18 socket 1
Port 0 Address:00:1B:21:C3:D6:1C
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963e0940 hw_ring=0x7f6a8f880080 dma_addr=0x823080080
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963e00c0 hw_ring=0x7f6a8f890080 dma_addr=0x823090080
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 1 Address:00:1B:21:C3:D6:1D
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963ddf80 hw_ring=0x7f6a8f8a0100 dma_addr=0x8230a0100
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963dd700 hw_ring=0x7f6a8f8b0100 dma_addr=0x8230b0100
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 2 Address:00:1B:21:C3:D5:FC
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963db5c0 hw_ring=0x7f6a8f8c0180 dma_addr=0x8230c0180
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963dad40 hw_ring=0x7f6a8f8d0180 dma_addr=0x8230d0180
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 3 Address:00:1B:21:C3:D5:FD
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963d8c00 hw_ring=0x7f6a8f8e0200 dma_addr=0x8230e0200
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963d8380 hw_ring=0x7f6a8f8f0200 dma_addr=0x8230f0200
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Checking link statuses...
Port 0 Link Up - speed 10000 Mbps - full-duplex
Port 1 Link Up - speed 10000 Mbps - full-duplex
Port 2 Link Up - speed 10000 Mbps - full-duplex
Port 3 Link Up - speed 10000 Mbps - full-duplex
IPv4 pktlen 46
UDP pktlen 26
Generate 8192 packets @socket 1
inject 2048 packet to port 0
inject 2048 packet to port 1
inject 2048 packet to port 2
inject 2048 packet to port 3
Total packets inject to prime ports = 8192
Each port will do 14880952 packets per second
Test will stop after at least 119047616 packets received
free 2048 mbuf left in port 0
free 2048 mbuf left in port 1
free 2048 mbuf left in port 2
free 2048 mbuf left in port 3
119047616 packet, 0 drop, 0 idle
Result: 31 cycles per packet
Test OK
RTE>>pmd_perf_autotest
Connected to 192.168.0.10
SSH2 - aes128-cbc - hmac-sha1 - no 131x53
6, 56
00:14:01



```
RTE>>set_rxtx_anchor txonly
type switch to txonly
RTE>>pmd_perf_autotest
Start PMD RXTX cycles cost test.
CONFIG RXD=128 TXD=512
Performance test runs on lcore 18 socket 1
Port 0 Address:00:1B:21:C3:D6:1C
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963e0940 hw_ring=0x7f6a8f880080 dma_addr=0x823080080
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963e00c0 hw_ring=0x7f6a8f890080 dma_addr=0x823090080
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 1 Address:00:1B:21:C3:D6:1D
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963ddf80 hw_ring=0x7f6a8f8a0100 dma_addr=0x8230a0100
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963dd700 hw_ring=0x7f6a8f8b0100 dma_addr=0x8230b0100
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 2 Address:00:1B:21:C3:D5:FC
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963db5c0 hw_ring=0x7f6a8f8c0180 dma_addr=0x8230c0180
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963dad40 hw_ring=0x7f6a8f8d0180 dma_addr=0x8230d0180
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Port 3 Address:00:1B:21:C3:D5:FD
PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f6a963d8c00 hw_ring=0x7f6a8f8e0200 dma_addr=0x8230e0200
PMD: ixgbe_set_tx_function(): Using simple tx code path
PMD: ixgbe_set_tx_function(): Vector tx enabled.
PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f6a963d8380 hw_ring=0x7f6a8f8f0200 dma_addr=0x8230f0200
PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 32.
Checking link statuses...
Port 0 Link Up - speed 10000 Mbps - full-duplex
Port 1 Link Up - speed 10000 Mbps - full-duplex
Port 2 Link Up - speed 10000 Mbps - full-duplex
Port 3 Link Up - speed 10000 Mbps - full-duplex
IPv4 pktlen 46
UDP pktlen 26
Generate 8192 packets @socket 1
inject 2048 packet to port 0
inject 2048 packet to port 1
inject 2048 packet to port 2
inject 2048 packet to port 3
Total packets inject to prime ports = 8192
Each port will do 14880952 packets per second
Test will stop after at least 119047616 packets received
do tx measure
free 2048 mbuf left in port 0
free 2048 mbuf left in port 1
free 2048 mbuf left in port 2
free 2048 mbuf left in port 3
119047616 packet, 0 drop, 0 idle
Result: 21 cycles per packet
Test OK
```


Hash Table Performance Test Results

To evaluate the performance in your platform, run /app/test/hash_perf_autotest.

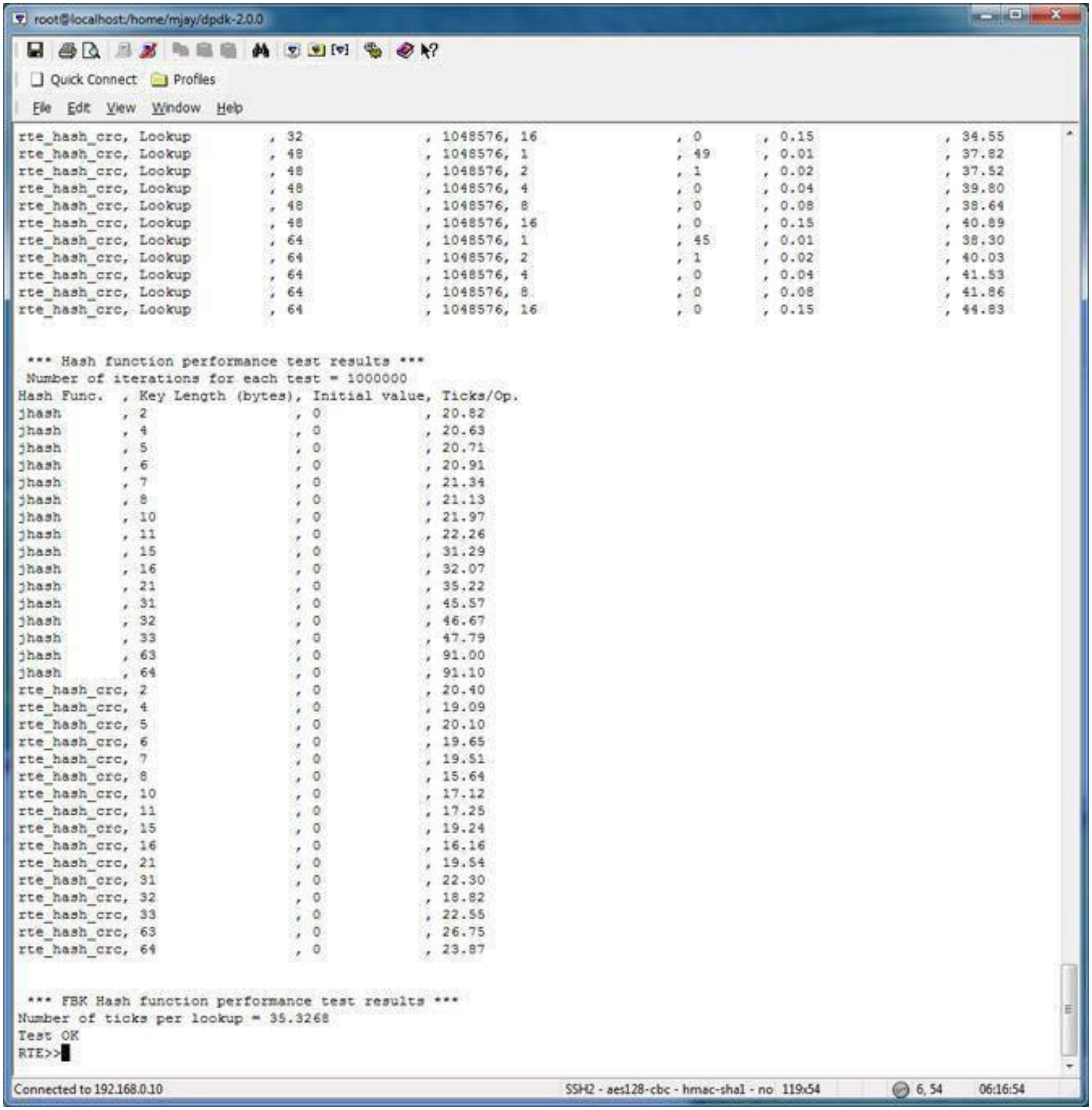
root@localhost:/home/mjay/dpdk-2.0.0

Quick Connect Profiles

File Edit View Window Help

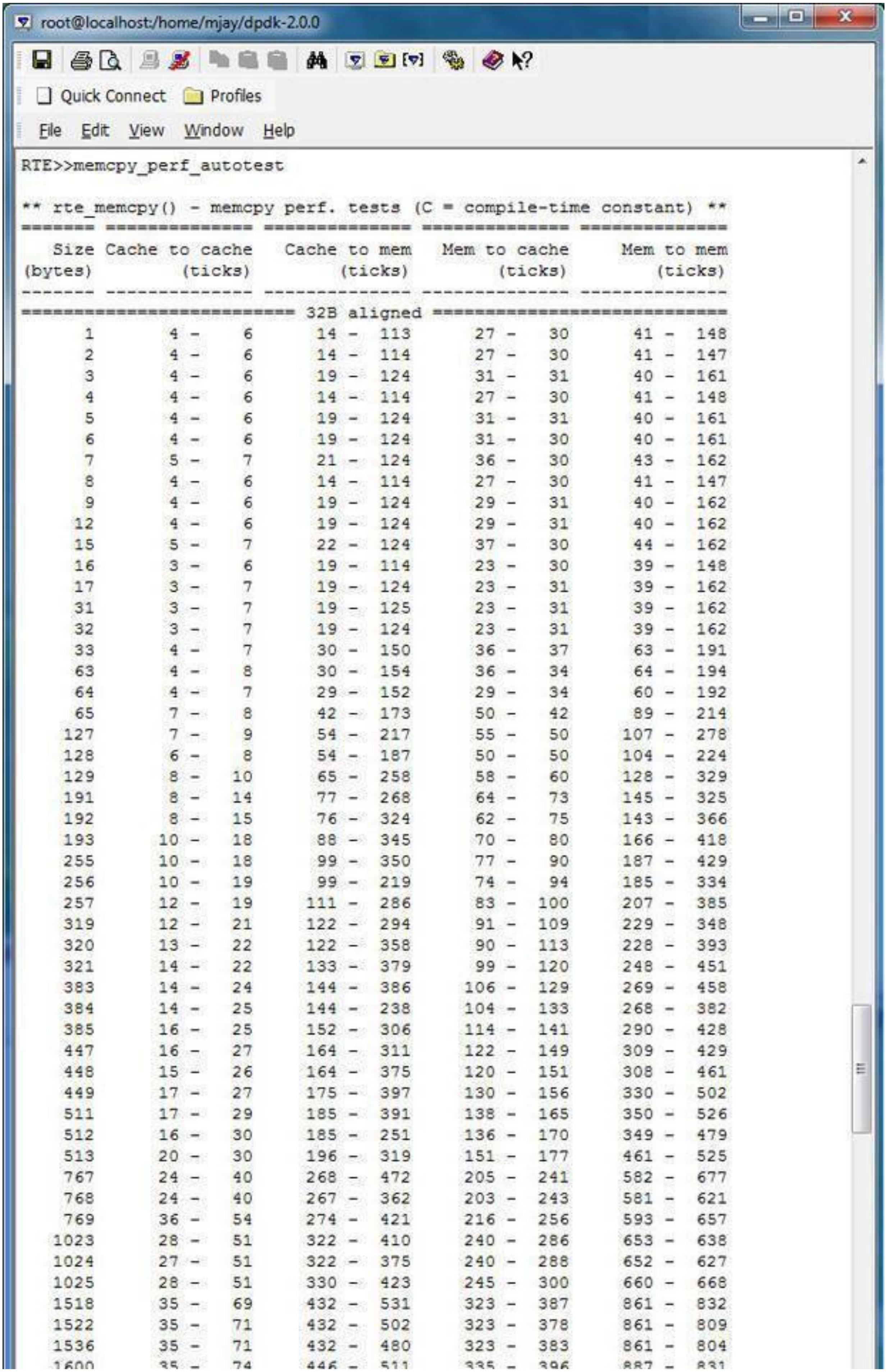
```
RTE>>hash_perf_autotest
 *** Hash table performance test results ***
Hash Func.  , Operation      , Key size (bytes), Entries, Entries per bucket, Errors   , Avg. bucket entries, Ticks/Op.
jhash       , Add on Empty   , 16              , 1024   , 1                 , 380      , 0.63               , 162.34
jhash       , Add on Empty   , 16              , 1024   , 2                 , 292      , 1.43               , 181.18
jhash       , Add on Empty   , 16              , 1024   , 4                 , 194      , 3.24               , 204.90
jhash       , Add on Empty   , 16              , 1024   , 8                 , 149      , 6.84               , 231.41
jhash       , Add on Empty   , 16              , 1024   , 16                , 96       , 14.50              , 284.72
jhash       , Add on Empty   , 32              , 1024   , 1                 , 375      , 0.63               , 215.20
jhash       , Add on Empty   , 32              , 1024   , 2                 , 274      , 1.46               , 243.38
jhash       , Add on Empty   , 32              , 1024   , 4                 , 193      , 3.25               , 250.46
jhash       , Add on Empty   , 32              , 1024   , 8                 , 151      , 6.82               , 279.74
jhash       , Add on Empty   , 32              , 1024   , 16                , 98       , 14.47              , 337.22
jhash       , Add on Empty   , 48              , 1024   , 1                 , 368      , 0.64               , 309.59
jhash       , Add on Empty   , 48              , 1024   , 2                 , 280      , 1.45               , 328.60
jhash       , Add on Empty   , 48              , 1024   , 4                 , 203      , 3.21               , 320.48
jhash       , Add on Empty   , 48              , 1024   , 8                 , 153      , 6.80               , 344.08
jhash       , Add on Empty   , 48              , 1024   , 16                , 104      , 14.38              , 390.21
jhash       , Add on Empty   , 64              , 1024   , 1                 , 381      , 0.63               , 329.43
jhash       , Add on Empty   , 64              , 1024   , 2                 , 280      , 1.45               , 348.27
jhash       , Add on Empty   , 64              , 1024   , 4                 , 218      , 3.15               , 356.72
jhash       , Add on Empty   , 64              , 1024   , 8                 , 138      , 6.92               , 383.35
jhash       , Add on Empty   , 64              , 1024   , 16                , 105      , 14.36              , 446.90
jhash       , Add Update     , 16              , 1024   , 1                 , 9644     , 0.35               , 109.51
jhash       , Add Update     , 16              , 1024   , 2                 , 9720     , 0.55               , 115.28
jhash       , Add Update     , 16              , 1024   , 4                 , 9806     , 0.76               , 128.44
jhash       , Add Update     , 16              , 1024   , 8                 , 9848     , 1.19               , 142.09
jhash       , Add Update     , 16              , 1024   , 16                , 9914     , 1.34               , 177.46
jhash       , Add Update     , 32              , 1024   , 1                 , 9625     , 0.37               , 133.12
jhash       , Add Update     , 32              , 1024   , 2                 , 9725     , 0.54               , 131.36
jhash       , Add Update     , 32              , 1024   , 4                 , 9792     , 0.81               , 131.08
jhash       , Add Update     , 32              , 1024   , 8                 , 9857     , 1.12               , 146.64
jhash       , Add Update     , 32              , 1024   , 16                , 9914     , 1.34               , 170.49
jhash       , Add Update     , 48              , 1024   , 1                 , 9618     , 0.37               , 160.77
jhash       , Add Update     , 48              , 1024   , 2                 , 9716     , 0.55               , 158.35
jhash       , Add Update     , 48              , 1024   , 4                 , 9804     , 0.77               , 157.92
jhash       , Add Update     , 48              , 1024   , 8                 , 9860     , 1.09               , 165.57
jhash       , Add Update     , 48              , 1024   , 16                , 9913     , 1.36               , 187.64
jhash       , Add Update     , 64              , 1024   , 1                 , 9649     , 0.34               , 165.35
jhash       , Add Update     , 64              , 1024   , 2                 , 9739     , 0.51               , 167.95
jhash       , Add Update     , 64              , 1024   , 4                 , 9791     , 0.82               , 177.04
jhash       , Add Update     , 64              , 1024   , 8                 , 9856     , 1.12               , 191.44
jhash       , Add Update     , 64              , 1024   , 16                , 9900     , 1.56               , 216.30
jhash       , Lookup         , 16              , 1024   , 1                 , 10000    , 0.00               , 59.53
jhash       , Lookup         , 16              , 1024   , 2                 , 10000    , 0.00               , 60.78
jhash       , Lookup         , 16              , 1024   , 4                 , 10000    , 0.00               , 63.83
jhash       , Lookup         , 16              , 1024   , 8                 , 10000    , 0.00               , 70.77
jhash       , Lookup         , 16              , 1024   , 16                , 10000    , 0.00               , 84.26
jhash       , Lookup         , 32              , 1024   , 1                 , 10000    , 0.00               , 85.09
jhash       , Lookup         , 32              , 1024   , 2                 , 10000    , 0.00               , 88.28
jhash       , Lookup         , 32              , 1024   , 4                 , 10000    , 0.00               , 90.68
jhash       , Lookup         , 32              , 1024   , 8                 , 10000    , 0.00               , 98.49
jhash       , Lookup         , 32              , 1024   , 16                , 10000    , 0.00               , 109.00
jhash       , Lookup         , 48              , 1024   , 1                 , 10000    , 0.00               , 133.06
jhash       , Lookup         , 48              , 1024   , 2                 , 10000    , 0.00               , 135.47
jhash       , Lookup         , 48              , 1024   , 4                 , 10000    , 0.00               , 139.37
jhash       , Lookup         , 48              , 1024   , 8                 , 10000    , 0.00               , 147.39
jhash       , Lookup         , 48              , 1024   , 16                , 10000    , 0.00               , 157.92
```

Connected to 192.168.0.10   SSH2 - aes128-cbc - hmac-sha1 - no 119x58   6, 78   06:08:34

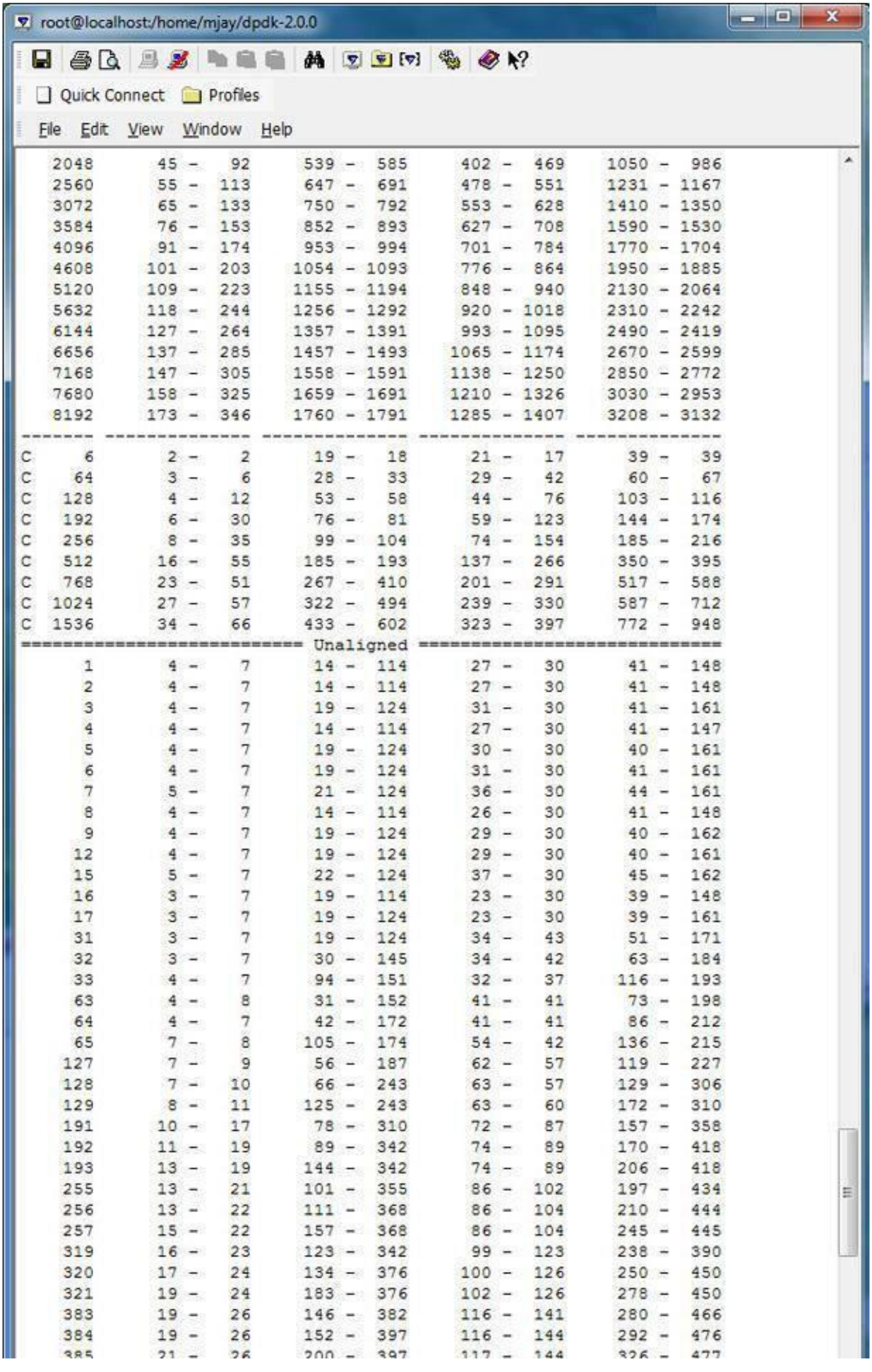


Memcpy_perf_autotest Test Results

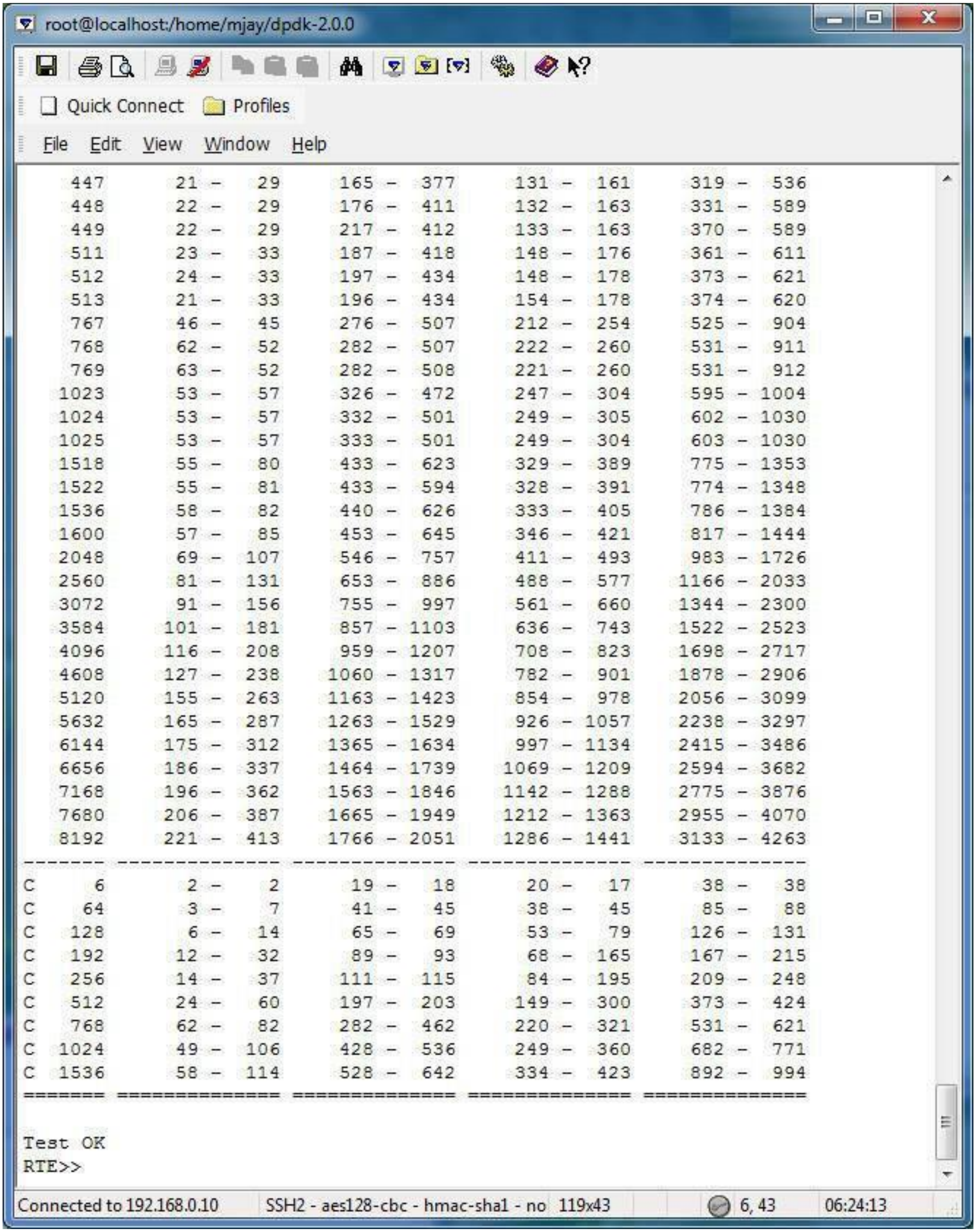
To evaluate the performance in your platform, run /app/test/memcpy_perf_autotest, for both 32 bytes aligned and unaligned.




```
RTE>>memcpy_perf_autotest

** rte_memcpy() - memcpy perf. tests (C = compile-time constant) **
======= ============== ============== ============== ==============
   Size Cache to cache   Cache to mem   Mem to cache     Mem to mem
(bytes)        (ticks)        (ticks)        (ticks)        (ticks)
------- -------------- -------------- -------------- --------------
=========================== 32B aligned ============================
      1       4 -    6      14 -  113      27 -   30      41 -  148
      2       4 -    6      14 -  114      27 -   30      41 -  147
      3       4 -    6      19 -  124      31 -   31      40 -  161
      4       4 -    6      14 -  114      27 -   30      41 -  148
      5       4 -    6      19 -  124      31 -   31      40 -  161
      6       4 -    6      19 -  124      31 -   30      40 -  161
      7       5 -    7      21 -  124      36 -   30      43 -  162
      8       4 -    6      14 -  114      27 -   30      41 -  147
      9       4 -    6      19 -  124      29 -   31      40 -  162
     12       4 -    6      19 -  124      29 -   31      40 -  162
     15       5 -    7      22 -  124      37 -   30      44 -  162
     16       3 -    6      19 -  114      23 -   30      39 -  148
     17       3 -    7      19 -  124      23 -   31      39 -  162
     31       3 -    7      19 -  125      23 -   31      39 -  162
     32       3 -    7      19 -  124      23 -   31      39 -  162
     33       4 -    7      30 -  150      36 -   37      63 -  191
     63       4 -    8      30 -  154      36 -   34      64 -  194
     64       4 -    7      29 -  152      29 -   34      60 -  192
     65       7 -    8      42 -  173      50 -   42      89 -  214
    127       7 -    9      54 -  217      55 -   50     107 -  278
    128       6 -    8      54 -  187      50 -   50     104 -  224
    129       8 -   10      65 -  258      58 -   60     128 -  329
    191       8 -   14      77 -  268      64 -   73     145 -  325
    192       8 -   15      76 -  324      62 -   75     143 -  366
    193      10 -   18      88 -  345      70 -   80     166 -  418
    255      10 -   18      99 -  350      77 -   90     187 -  429
    256      10 -   19      99 -  219      74 -   94     185 -  334
    257      12 -   19     111 -  286      83 -  100     207 -  385
    319      12 -   21     122 -  294      91 -  109     229 -  348
    320      13 -   22     122 -  358      90 -  113     228 -  393
    321      14 -   22     133 -  379      99 -  120     248 -  451
    383      14 -   24     144 -  386     106 -  129     269 -  458
    384      14 -   25     144 -  238     104 -  133     268 -  382
    385      16 -   25     152 -  306     114 -  141     290 -  428
    447      16 -   27     164 -  311     122 -  149     309 -  429
    448      15 -   26     164 -  375     120 -  151     308 -  461
    449      17 -   27     175 -  397     130 -  156     330 -  502
    511      17 -   29     185 -  391     138 -  165     350 -  526
    512      16 -   30     185 -  251     136 -  170     349 -  479
    513      20 -   30     196 -  319     151 -  177     461 -  525
    767      24 -   40     268 -  472     205 -  241     582 -  677
    768      24 -   40     267 -  362     203 -  243     581 -  621
    769      36 -   54     274 -  421     216 -  256     593 -  657
   1023      28 -   51     322 -  410     240 -  286     653 -  638
   1024      27 -   51     322 -  375     240 -  288     652 -  627
   1025      28 -   51     330 -  423     245 -  300     660 -  668
   1518      35 -   69     432 -  531     323 -  387     861 -  832
   1522      35 -   71     432 -  502     323 -  378     861 -  809
   1536      35 -   71     432 -  480     323 -  383     861 -  804
   1600      35 -   74     446 -  511     335 -  396     887 -  831
```




```
   2048      45 -   92     539 -  585     402 -  469    1050 -  986
   2560      55 -  113     647 -  691     478 -  551    1231 - 1167
   3072      65 -  133     750 -  792     553 -  628    1410 - 1350
   3584      76 -  153     852 -  893     627 -  708    1590 - 1530
   4096      91 -  174     953 -  994     701 -  784    1770 - 1704
   4608     101 -  203    1054 - 1093     776 -  864    1950 - 1885
   5120     109 -  223    1155 - 1194     848 -  940    2130 - 2064
   5632     118 -  244    1256 - 1292     920 - 1018    2310 - 2242
   6144     127 -  264    1357 - 1391     993 - 1095    2490 - 2419
   6656     137 -  285    1457 - 1493    1065 - 1174    2670 - 2599
   7168     147 -  305    1558 - 1591    1138 - 1250    2850 - 2772
   7680     158 -  325    1659 - 1691    1210 - 1326    3030 - 2953
   8192     173 -  346    1760 - 1791    1285 - 1407    3208 - 3132
------- -------------- -------------- -------------- --------------
C     6       2 -    2      19 -   18      21 -   17      39 -   39
C    64       3 -    6      28 -   33      29 -   42      60 -   67
C   128       4 -   12      53 -   58      44 -   76     103 -  116
C   192       6 -   30      76 -   81      59 -  123     144 -  174
C   256       8 -   35      99 -  104      74 -  154     185 -  216
C   512      16 -   55     185 -  193     137 -  266     350 -  395
C   768      23 -   51     267 -  410     201 -  291     517 -  588
C  1024      27 -   57     322 -  494     239 -  330     587 -  712
C  1536      34 -   66     433 -  602     323 -  397     772 -  948
=========================== Unaligned =============================
      1       4 -    7      14 -  114      27 -   30      41 -  148
      2       4 -    7      14 -  114      27 -   30      41 -  148
      3       4 -    7      19 -  124      31 -   30      41 -  161
      4       4 -    7      14 -  114      27 -   30      41 -  147
      5       4 -    7      19 -  124      30 -   30      40 -  161
      6       4 -    7      19 -  124      31 -   30      41 -  161
      7       5 -    7      21 -  124      36 -   30      44 -  161
      8       4 -    7      14 -  114      26 -   30      41 -  148
      9       4 -    7      19 -  124      29 -   30      40 -  162
     12       4 -    7      19 -  124      29 -   30      40 -  161
     15       5 -    7      22 -  124      37 -   30      45 -  162
     16       3 -    7      19 -  114      23 -   30      39 -  148
     17       3 -    7      19 -  124      23 -   30      39 -  161
     31       3 -    7      19 -  124      34 -   43      51 -  171
     32       3 -    7      30 -  145      34 -   42      63 -  184
     33       4 -    7      94 -  151      32 -   37     116 -  193
     63       4 -    8      31 -  152      41 -   41      73 -  198
     64       4 -    7      42 -  172      41 -   41      86 -  212
     65       7 -    8     105 -  174      54 -   42     136 -  215
    127       7 -    9      56 -  187      62 -   57     119 -  227
    128       7 -   10      66 -  243      63 -   57     129 -  306
    129       8 -   11     125 -  243      63 -   60     172 -  310
    191      10 -   17      78 -  310      72 -   87     157 -  358
    192      11 -   19      89 -  342      74 -   89     170 -  418
    193      13 -   19     144 -  342      74 -   89     206 -  418
    255      13 -   21     101 -  355      86 -  102     197 -  434
    256      13 -   22     111 -  368      86 -  104     210 -  444
    257      15 -   22     157 -  368      86 -  104     245 -  445
    319      16 -   23     123 -  342      99 -  123     238 -  390
    320      17 -   24     134 -  376     100 -  126     250 -  450
    321      19 -   24     183 -  376     102 -  126     278 -  450
    383      19 -   26     146 -  382     116 -  141     280 -  466
    384      19 -   26     152 -  397     116 -  144     292 -  476
    385      21 -   26     200 -  397     117 -  144     326 -  477
```




```
    447       21 -   29     165 -  377     131 -  161     319 -  536
    448       22 -   29     176 -  411     132 -  163     331 -  589
    449       22 -   29     217 -  412     133 -  163     370 -  589
    511       23 -   33     187 -  418     148 -  176     361 -  611
    512       24 -   33     197 -  434     148 -  178     373 -  621
    513       21 -   33     196 -  434     154 -  178     374 -  620
    767       46 -   45     276 -  507     212 -  254     525 -  904
    768       62 -   52     282 -  507     222 -  260     531 -  911
    769       63 -   52     282 -  508     221 -  260     531 -  912
   1023       53 -   57     326 -  472     247 -  304     595 - 1004
   1024       53 -   57     332 -  501     249 -  305     602 - 1030
   1025       53 -   57     333 -  501     249 -  304     603 - 1030
   1518       55 -   80     433 -  623     329 -  389     775 - 1353
   1522       55 -   81     433 -  594     328 -  391     774 - 1348
   1536       58 -   82     440 -  626     333 -  405     786 - 1384
   1600       57 -   85     453 -  645     346 -  421     817 - 1444
   2048       69 -  107     546 -  757     411 -  493     983 - 1726
   2560       81 -  131     653 -  886     488 -  577    1166 - 2033
   3072       91 -  156     755 -  997     561 -  660    1344 - 2300
   3584      101 -  181     857 - 1103     636 -  743    1522 - 2523
   4096      116 -  208     959 - 1207     708 -  823    1698 - 2717
   4608      127 -  238    1060 - 1317     782 -  901    1878 - 2906
   5120      155 -  263    1163 - 1423     854 -  978    2056 - 3099
   5632      165 -  287    1263 - 1529     926 - 1057    2238 - 3297
   6144      175 -  312    1365 - 1634     997 - 1134    2415 - 3486
   6656      186 -  337    1464 - 1739    1069 - 1209    2594 - 3682
   7168      196 -  362    1563 - 1846    1142 - 1288    2775 - 3876
   7680      206 -  387    1665 - 1949    1212 - 1363    2955 - 4070
   8192      221 -  413    1766 - 2051    1286 - 1441    3133 - 4263
------- -------------- -------------- -------------- --------------
C     6        2 -    2      19 -   18      20 -   17      38 -   38
C    64        3 -    7      41 -   45      38 -   45      85 -   88
C   128        6 -   14      65 -   69      53 -   79     126 -  131
C   192       12 -   32      89 -   93      68 -  165     167 -  215
C   256       14 -   37     111 -  115      84 -  195     209 -  248
C   512       24 -   60     197 -  203     149 -  300     373 -  424
C   768       62 -   82     282 -  462     220 -  321     531 -  621
C  1024       49 -  106     428 -  536     249 -  360     682 -  771
C  1536       58 -  114     528 -  642     334 -  423     892 -  994
======= ============== ============== ============== ==============

Test OK
RTE>>
```

Mempool_perf_autotest Test Results

Core Configuration	Cache Object	Bulk Get Size	Bulk Put Size	# of Kept Objects
a) One Core b) Two Cores c) Max. Cores	a) with cache object b) without cache object	a) 1 b) 4 c) 32	a) 1 b) 4 c) 32	a) 32 b) 128

To evaluate the performance in your platform, run /app/test/mempool_perf_autotest.

```
RTE>>mempool_perf_autotest
start performance test (without cache)
mempool_autotest cache=0 cores=1 n_get_bulk=1 n_put_bulk=1 n_keep=32 rate_persec=47539814
mempool_autotest cache=0 cores=1 n_get_bulk=1 n_put_bulk=1 n_keep=128 rate_persec=50017075
mempool_autotest cache=0 cores=1 n_get_bulk=1 n_put_bulk=4 n_keep=32 rate_persec=79010201
mempool_autotest cache=0 cores=1 n_get_bulk=1 n_put_bulk=4 n_keep=128 rate_persec=78879129
mempool_autotest cache=0 cores=1 n_get_bulk=1 n_put_bulk=32 n_keep=32 rate_persec=95617024
mempool_autotest cache=0 cores=1 n_get_bulk=1 n_put_bulk=32 n_keep=128 rate_persec=95027200
mempool_autotest cache=0 cores=1 n_get_bulk=4 n_put_bulk=1 n_keep=32 rate_persec=78852915
mempool_autotest cache=0 cores=1 n_get_bulk=4 n_put_bulk=1 n_keep=128 rate_persec=79036416
mempool_autotest cache=0 cores=1 n_get_bulk=4 n_put_bulk=4 n_keep=32 rate_persec=180682752
mempool_autotest cache=0 cores=1 n_get_bulk=4 n_put_bulk=4 n_keep=128 rate_persec=181364326
mempool_autotest cache=0 cores=1 n_get_bulk=4 n_put_bulk=32 n_keep=32 rate_persec=298241228
mempool_autotest cache=0 cores=1 n_get_bulk=4 n_put_bulk=32 n_keep=128 rate_persec=294833356
mempool_autotest cache=0 cores=1 n_get_bulk=32 n_put_bulk=1 n_keep=32 rate_persec=94162124
mempool_autotest cache=0 cores=1 n_get_bulk=32 n_put_bulk=1 n_keep=128 rate_persec=94345625
mempool_autotest cache=0 cores=1 n_get_bulk=32 n_put_bulk=4 n_keep=32 rate_persec=284911206
mempool_autotest cache=0 cores=1 n_get_bulk=32 n_put_bulk=4 n_keep=128 rate_persec=285409280
mempool_autotest cache=0 cores=1 n_get_bulk=32 n_put_bulk=32 n_keep=32 rate_persec=717160448
mempool_autotest cache=0 cores=1 n_get_bulk=32 n_put_bulk=32 n_keep=128 rate_persec=709020876
mempool_autotest cache=0 cores=2 n_get_bulk=1 n_put_bulk=1 n_keep=32 rate_persec=9738649
mempool_autotest cache=0 cores=2 n_get_bulk=1 n_put_bulk=1 n_keep=128 rate_persec=9673112
mempool_autotest cache=0 cores=2 n_get_bulk=1 n_put_bulk=4 n_keep=32 rate_persec=13867416
mempool_autotest cache=0 cores=2 n_get_bulk=1 n_put_bulk=4 n_keep=128 rate_persec=13238272
mempool_autotest cache=0 cores=2 n_get_bulk=1 n_put_bulk=32 n_keep=32 rate_persec=14850456
mempool_autotest cache=0 cores=2 n_get_bulk=1 n_put_bulk=32 n_keep=128 rate_persec=15151922
mempool_autotest cache=0 cores=2 n_get_bulk=4 n_put_bulk=1 n_keep=32 rate_persec=12845056
mempool_autotest cache=0 cores=2 n_get_bulk=4 n_put_bulk=1 n_keep=128 rate_persec=12845056
mempool_autotest cache=0 cores=2 n_get_bulk=4 n_put_bulk=4 n_keep=32 rate_persec=38967705
mempool_autotest cache=0 cores=2 n_get_bulk=4 n_put_bulk=4 n_keep=128 rate_persec=37932236
mempool_autotest cache=0 cores=2 n_get_bulk=4 n_put_bulk=32 n_keep=32 rate_persec=54407986
mempool_autotest cache=0 cores=2 n_get_bulk=4 n_put_bulk=32 n_keep=128 rate_persec=58117324
mempool_autotest cache=0 cores=2 n_get_bulk=32 n_put_bulk=1 n_keep=32 rate_persec=13474200
mempool_autotest cache=0 cores=2 n_get_bulk=32 n_put_bulk=1 n_keep=128 rate_persec=13631488
mempool_autotest cache=0 cores=2 n_get_bulk=32 n_put_bulk=4 n_keep=32 rate_persec=50410290
mempool_autotest cache=0 cores=2 n_get_bulk=32 n_put_bulk=4 n_keep=128 rate_persec=52822015
mempool_autotest cache=0 cores=2 n_get_bulk=32 n_put_bulk=32 n_keep=32 rate_persec=236099993
mempool_autotest cache=0 cores=2 n_get_bulk=32 n_put_bulk=32 n_keep=128 rate_persec=271043788
mempool_autotest cache=0 cores=35 n_get_bulk=1 n_put_bulk=1 n_keep=32 rate_persec=1834980
mempool_autotest cache=0 cores=35 n_get_bulk=1 n_put_bulk=1 n_keep=128 rate_persec=1834980
mempool_autotest cache=0 cores=35 n_get_bulk=1 n_put_bulk=4 n_keep=32 rate_persec=2752505
mempool_autotest cache=0 cores=35 n_get_bulk=1 n_put_bulk=4 n_keep=128 rate_persec=2752505
mempool_autotest cache=0 cores=35 n_get_bulk=1 n_put_bulk=32 n_keep=32 rate_persec=2752505
mempool_autotest cache=0 cores=35 n_get_bulk=1 n_put_bulk=32 n_keep=128 rate_persec=2752505
mempool_autotest cache=0 cores=35 n_get_bulk=4 n_put_bulk=1 n_keep=32 rate_persec=2752505
mempool_autotest cache=0 cores=35 n_get_bulk=4 n_put_bulk=1 n_keep=128 rate_persec=2752505
mempool_autotest cache=0 cores=35 n_get_bulk=4 n_put_bulk=4 n_keep=32 rate_persec=6881280
mempool_autotest cache=0 cores=35 n_get_bulk=4 n_put_bulk=4 n_keep=128 rate_persec=7340025
mempool_autotest cache=0 cores=35 n_get_bulk=4 n_put_bulk=32 n_keep=32 rate_persec=9895925
mempool_autotest cache=0 cores=35 n_get_bulk=4 n_put_bulk=32 n_keep=128 rate_persec=10551275
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=1 n_keep=32 rate_persec=2752505
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=1 n_keep=128 rate_persec=2752505
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=4 n_keep=32 rate_persec=9633785
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=4 n_keep=128 rate_persec=10092530
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=32 n_keep=32 rate_persec=42899841
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=32 n_keep=128 rate_persec=51838955
start performance test (with cache)
```

root@localhost:/home/mjay/dpdk-2.0.0

Quick Connect Profiles

File Edit View Window Help

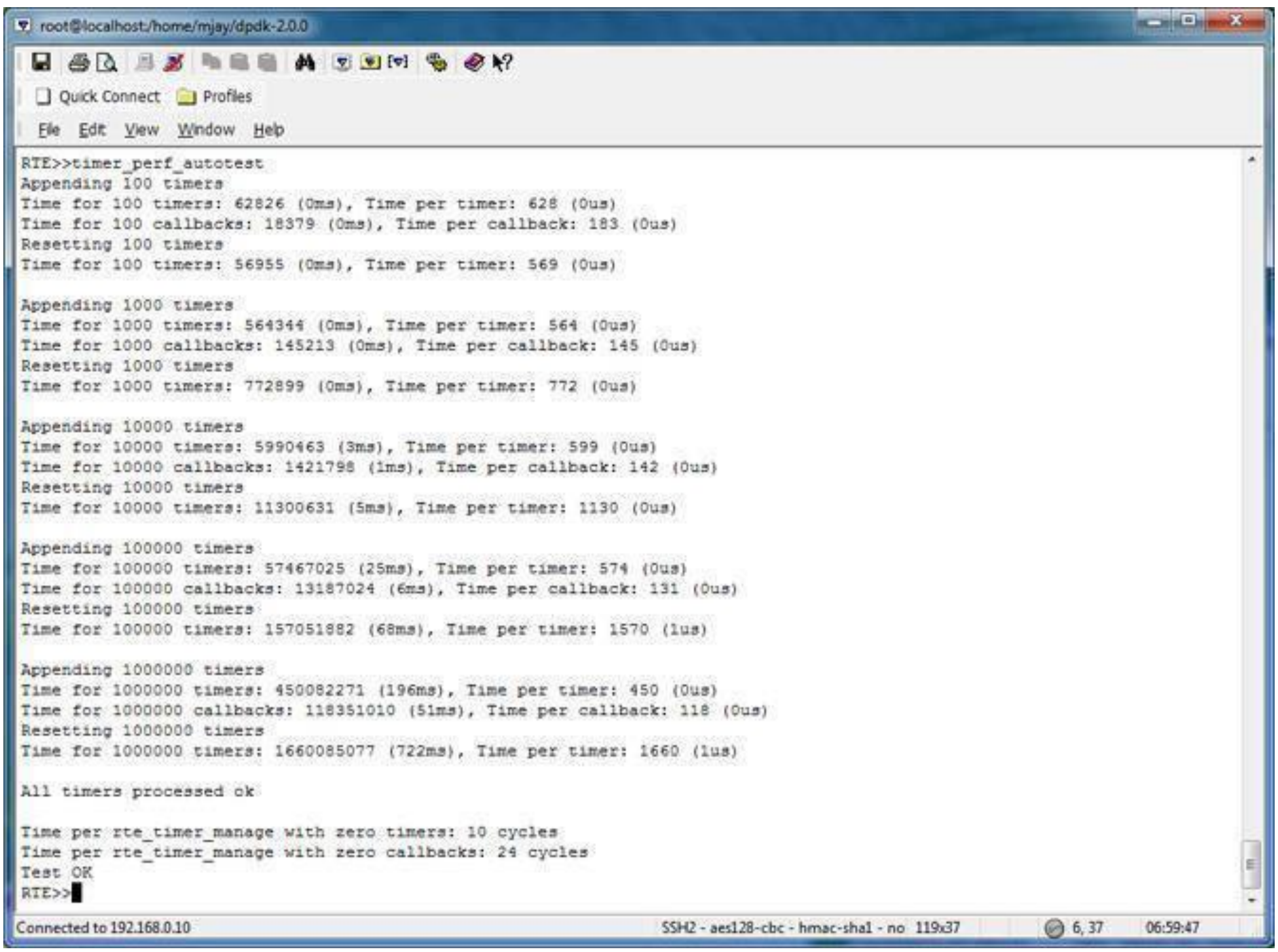
```
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=4 n_keep=128 rate_persec=10092530
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=32 n_keep=32 rate_persec=42899841
mempool_autotest cache=0 cores=35 n_get_bulk=32 n_put_bulk=32 n_keep=128 rate_persec=51838955
start performance test (with cache)
mempool_autotest cache=512 cores=1 n_get_bulk=1 n_put_bulk=1 n_keep=32 rate_persec=103428915
mempool_autotest cache=512 cores=1 n_get_bulk=1 n_put_bulk=1 n_keep=128 rate_persec=123286323
mempool_autotest cache=512 cores=1 n_get_bulk=1 n_put_bulk=4 n_keep=32 rate_persec=196463820
mempool_autotest cache=512 cores=1 n_get_bulk=1 n_put_bulk=4 n_keep=128 rate_persec=212061388
mempool_autotest cache=512 cores=1 n_get_bulk=1 n_put_bulk=32 n_keep=32 rate_persec=242142412
mempool_autotest cache=512 cores=1 n_get_bulk=1 n_put_bulk=32 n_keep=128 rate_persec=256612761
mempool_autotest cache=512 cores=1 n_get_bulk=4 n_put_bulk=1 n_keep=32 rate_persec=181560934
mempool_autotest cache=512 cores=1 n_get_bulk=4 n_put_bulk=1 n_keep=128 rate_persec=175531622
mempool_autotest cache=512 cores=1 n_get_bulk=4 n_put_bulk=4 n_keep=32 rate_persec=415026380
mempool_autotest cache=512 cores=1 n_get_bulk=4 n_put_bulk=4 n_keep=128 rate_persec=381511270
mempool_autotest cache=512 cores=1 n_get_bulk=4 n_put_bulk=32 n_keep=32 rate_persec=583100006
mempool_autotest cache=512 cores=1 n_get_bulk=4 n_put_bulk=32 n_keep=128 rate_persec=564317388
mempool_autotest cache=512 cores=1 n_get_bulk=32 n_put_bulk=1 n_keep=32 rate_persec=211196313
mempool_autotest cache=512 cores=1 n_get_bulk=32 n_put_bulk=1 n_keep=128 rate_persec=202230988
mempool_autotest cache=512 cores=1 n_get_bulk=32 n_put_bulk=4 n_keep=32 rate_persec=597308211
mempool_autotest cache=512 cores=1 n_get_bulk=32 n_put_bulk=4 n_keep=128 rate_persec=560660480
mempool_autotest cache=512 cores=1 n_get_bulk=32 n_put_bulk=32 n_keep=32 rate_persec=840931737
mempool_autotest cache=512 cores=1 n_get_bulk=32 n_put_bulk=32 n_keep=128 rate_persec=111070412
mempool_autotest cache=512 cores=2 n_get_bulk=1 n_put_bulk=1 n_keep=32 rate_persec=238839397
mempool_autotest cache=512 cores=2 n_get_bulk=1 n_put_bulk=1 n_keep=128 rate_persec=246598860
mempool_autotest cache=512 cores=2 n_get_bulk=1 n_put_bulk=4 n_keep=32 rate_persec=400110387
mempool_autotest cache=512 cores=2 n_get_bulk=1 n_put_bulk=4 n_keep=128 rate_persec=417975500
mempool_autotest cache=512 cores=2 n_get_bulk=1 n_put_bulk=32 n_keep=32 rate_persec=456615526
mempool_autotest cache=512 cores=2 n_get_bulk=1 n_put_bulk=32 n_keep=128 rate_persec=510774476
mempool_autotest cache=512 cores=2 n_get_bulk=4 n_put_bulk=1 n_keep=32 rate_persec=361024716
mempool_autotest cache=512 cores=2 n_get_bulk=4 n_put_bulk=1 n_keep=128 rate_persec=341455666
mempool_autotest cache=512 cores=2 n_get_bulk=4 n_put_bulk=4 n_keep=32 rate_persec=838677299
mempool_autotest cache=512 cores=2 n_get_bulk=4 n_put_bulk=4 n_keep=128 rate_persec=774963199
mempool_autotest cache=512 cores=2 n_get_bulk=4 n_put_bulk=32 n_keep=32 rate_persec=1159397375
mempool_autotest cache=512 cores=2 n_get_bulk=4 n_put_bulk=32 n_keep=128 rate_persec=1129879961
mempool_autotest cache=512 cores=2 n_get_bulk=32 n_put_bulk=1 n_keep=32 rate_persec=419849830
mempool_autotest cache=512 cores=2 n_get_bulk=32 n_put_bulk=1 n_keep=128 rate_persec=391276134
mempool_autotest cache=512 cores=2 n_get_bulk=32 n_put_bulk=4 n_keep=32 rate_persec=1212284927
mempool_autotest cache=512 cores=2 n_get_bulk=32 n_put_bulk=4 n_keep=128 rate_persec=1115697970
mempool_autotest cache=512 cores=2 n_get_bulk=32 n_put_bulk=32 n_keep=32 rate_persec=1664614399
mempool_autotest cache=512 cores=2 n_get_bulk=32 n_put_bulk=32 n_keep=128 rate_persec=206910258
mempool_autotest cache=512 cores=35 n_get_bulk=1 n_put_bulk=1 n_keep=32 rate_persec=3211565450
mempool_autotest cache=512 cores=35 n_get_bulk=1 n_put_bulk=1 n_keep=128 rate_persec=3325113126
mempool_autotest cache=512 cores=35 n_get_bulk=1 n_put_bulk=4 n_keep=32 rate_persec=1047986160
mempool_autotest cache=512 cores=35 n_get_bulk=1 n_put_bulk=4 n_keep=128 rate_persec=1335466378
mempool_autotest cache=512 cores=35 n_get_bulk=1 n_put_bulk=32 n_keep=32 rate_persec=1945829361
mempool_autotest cache=512 cores=35 n_get_bulk=1 n_put_bulk=32 n_keep=128 rate_persec=2456944628
mempool_autotest cache=512 cores=35 n_get_bulk=4 n_put_bulk=1 n_keep=32 rate_persec=537106827
mempool_autotest cache=512 cores=35 n_get_bulk=4 n_put_bulk=1 n_keep=128 rate_persec=253165555
mempool_autotest cache=512 cores=35 n_get_bulk=4 n_put_bulk=4 n_keep=32 rate_persec=2434360918
mempool_autotest cache=512 cores=35 n_get_bulk=4 n_put_bulk=4 n_keep=128 rate_persec=1713530455
mempool_autotest cache=512 cores=35 n_get_bulk=4 n_put_bulk=32 n_keep=32 rate_persec=2357578942
mempool_autotest cache=512 cores=35 n_get_bulk=4 n_put_bulk=32 n_keep=128 rate_persec=2086181258
mempool_autotest cache=512 cores=35 n_get_bulk=32 n_put_bulk=1 n_keep=32 rate_persec=1281949683
mempool_autotest cache=512 cores=35 n_get_bulk=32 n_put_bulk=1 n_keep=128 rate_persec=943443134
mempool_autotest cache=512 cores=35 n_get_bulk=32 n_put_bulk=4 n_keep=32 rate_persec=2995257331
mempool_autotest cache=512 cores=35 n_get_bulk=32 n_put_bulk=4 n_keep=128 rate_persec=1623746137
mempool_autotest cache=512 cores=35 n_get_bulk=32 n_put_bulk=32 n_keep=32 rate_persec=612066900
```

Connected to 192.168.0.10 SSH2 - aes128-cbc - hmac-sha1 - no 119x58 1, 58 06:54:46

Timer_perf_autotest Test Results

# of Timers Configuration	Operations Timed
a) 0 Timer b) 100 Timers c) 1000 Timers d) 10,000 Timers e) 100,000 Timers f) 1,000,000 Timers	- Appending - Callback - Resetting

To evaluate the performance in your platform, run /app/test/timer_perf_autotest.



For cookbook-style instructions on how to do hands-on performance profiling of your DPDK code with VTune tools, continue to the module *Profiling DPDK Code with Intel VTune Amplifier*.

Performance Profiling Resources

- Document #5571159 Intel Xeon processor E7-8800/4800 v3 Performance Tuning Guide
- Intel® Optimizing Non-Sequential Data Processing Applications – Brian Forde and John Browne
- Measuring Cache and Memory Latency and CPU to Memory Bandwidth - For use with Intel Architecture – Joshua Ruggiero
- Tuning Applications Using a Top-down Microarchitecture Analysis Method
- Intel® Processor Trace architecture details can be found in the Intel 64 and IA-32 Architectures Software Developer Manuals
- Evaluating the Suitability of Server Network Cards for Software Routers
- Low Latency Performance Tuning Guide For Red Hat Enterprise Linux 6 Jeremy Eder, Senior Software Engineer
- Red Hat Enterprise Linux 6 Performance Tuning Guide
- Memory Ordering in Modern Microprocessors – Paul E McKenney Draft of 2007/09/19 15:15
- What is RCU, Fundamentally?

Profiling DPDK Code with Intel® VTune™ Amplifier

Introduction

Performance is a key factor in designing and shipping best of class products. Optimizing performance requires visibility into system behavior. In this module, we'll learn how to use Intel VTune Amplifier to profile Data Plane Development Kit (DPDK) code.

You will find this module to be a comprehensive reference for installing and use of the Intel VTune Amplifier**,** and will learn how to run some **DPDK micro benchmarks** as an example of how to get deep visibility into system, cores, communication, and core pipeline and usage.

Extensive screenshots are provided for comparison with your output. The commands are given, in addition, so that the readers can copy and paste wherever possible.

Outline

This module walks you through the following steps to get started using Intel VTune Amplifier with a DPDK application.

- Install Linux
- Install Data Plane Development Kit (DPDK)
- Install the tools
    - Source editor
    - Intel VTune Amplifier
- Install and profile the application of your choice
    - Distributor application

  - o Ring tests application
- Conclusion and next steps

Install Linux

Install from the Linux DVD with an ISO image:
http://old-releases.ubuntu.com/releases/15.04/ubuntu-15.04-desktop-amd64.iso

Prior to Install

If you have a laptop installed with Windows* 8, go to safe mode (SHIFT+RESTART). Once in safe mode, choose boot option # 1 to boot from the external USB DVD drive. Restart and install.

After Install

1. **Verify whether the kernel version installed** is the correct version as per the DPDK release notes.

   `$uname –a`

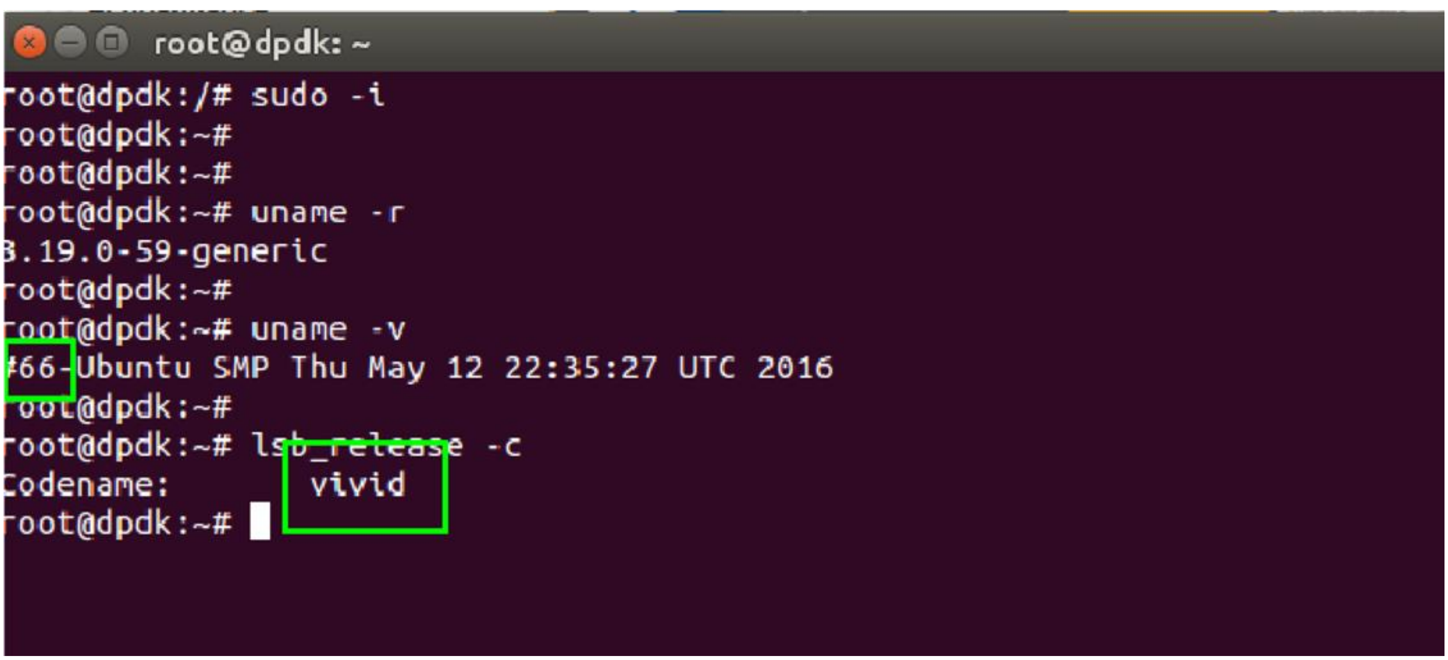
   The above output verifies the kernel release as 3.19.0-59-generic, the version number as #66, and the distro as Ubuntu 64 bit.

   `$uname –v`

   Displays the version # – version #66 as shown below.

   `$lsb_release –c`

   Shows the code name—the code name is *vivid*, as shown below.



2. **Verify Internet connectivity**. In some cases, the network-manager service has to be restarted for the Ethernet service to be operational.

```
$ sudo service network-manager restart
```

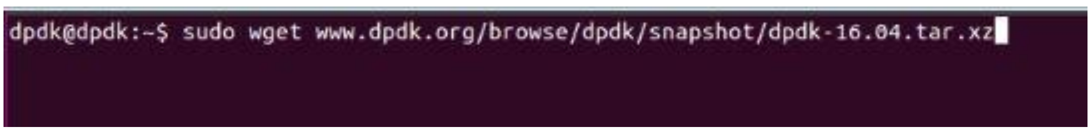
```
dpdk@dpdk:~/dpdk-16.04$ sudo service network-manager restart
```

Install DPDK

Download the DPDK

3. **Get the latest DPDK release**, as shown below and in the screenshot.

```
$ sudo wget www.dpdk.org/browse/dpdk/snapshot/dpdk-16.04.tar.xz
```



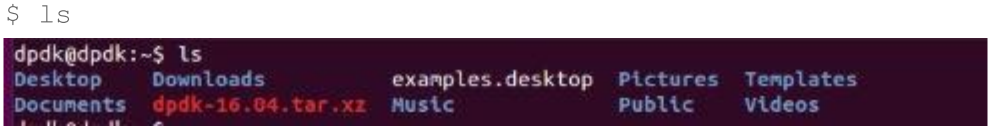
```
dpdk@dpdk:~$ sudo wget www.dpdk.org/browse/dpdk/snapshot/dpdk-16.04.tar.xz
```

The response for the above command is as shown below.

```
dpdk@dpdk:~$ sudo wget www.dpdk.org/browse/dpdk/snapshot/dpdk-16.04.tar.xz
--2016-06-02 19:05:07--  http://www.dpdk.org/browse/dpdk/snapshot/dpdk-16.04.tar.xz
Resolving www.dpdk.org (www.dpdk.org)... 92.243.14.124
Connecting to www.dpdk.org (www.dpdk.org)|92.243.14.124|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/x-xz]
Saving to: 'dpdk-16.04.tar.xz'

dpdk-16.04.tar.xz          [                 <=>              ]  10.42M   213KB/s   in 36s

2016-06-02 19:05:44 (295 KB/s) - 'dpdk-16.04.tar.xz' saved [10925676]

dpdk@dpdk:~$
```

You will find the DPDK tar file downloaded, as shown below.

```
$ ls
```



```
dpdk@dpdk:~$ ls
Desktop    Downloads          examples.desktop  Pictures  Templates
Documents  dpdk-16.04.tar.xz  Music             Public    Videos
```

4. **Extract the tar ball**.

```
$ tar xf dpdk-16.04.tar.xz
```

You will find that the directory dpdk-16.04 was created.

```
$ ls
```

```
dpdk@dpdk:~$ tar xf dpdk-16.04.tar.xz
dpdk@dpdk:~$
dpdk@dpdk:~$ ls
Desktop     Downloads   dpdk-16.04.tar.xz  Music      Public     Videos
Documents   dpdk-16.04  examples.desktop   Pictures   Templates
```

5. **Change to the DPDK directory** to list the files.

```
$ cd dpdk-16.04
$ ls -al
```

```
dpdk@dpdk:~$ cd dpdk-16.04/
dpdk@dpdk:~/dpdk-16.04$
dpdk@dpdk:~/dpdk-16.04$ ls -al
total 128
drwxrwxr-x 12 dpdk dpdk  4096 Apr 11 14:56 .
drwxr-xr-x 18 dpdk dpdk  4096 Jun  2 21:36 ..
drwxrwxr-x  8 dpdk dpdk  4096 Apr 11 14:56 app
drwxrwxr-x  2 dpdk dpdk  4096 Apr 11 14:56 config
drwxrwxr-x  5 dpdk dpdk  4096 Apr 11 14:56 doc
drwxrwxr-x  4 dpdk dpdk  4096 Apr 11 14:56 drivers
drwxrwxr-x 44 dpdk dpdk  4096 Apr 11 14:56 examples
-rw-rw-r--  1 dpdk dpdk     0 Apr 11 14:56 .gitignore
-rw-rw-r--  1 dpdk dpdk  1826 Apr 11 14:56 GNUmakefile
drwxrwxr-x 30 dpdk dpdk  4096 Apr 11 14:56 lib
-rw-rw-r--  1 dpdk dpdk 17987 Apr 11 14:56 LICENSE.GPL
-rw-rw-r--  1 dpdk dpdk 26530 Apr 11 14:56 LICENSE.LGPL
-rw-rw-r--  1 dpdk dpdk 16797 Apr 11 14:56 MAINTAINERS
-rw-rw-r--  1 dpdk dpdk  1708 Apr 11 14:56 Makefile
drwxrwxr-x  8 dpdk dpdk  4096 Apr 11 14:56 mk
drwxrwxr-x  2 dpdk dpdk  4096 Apr 11 14:56 pkg
-rw-rw-r--  1 dpdk dpdk   499 Apr 11 14:56 README
drwxrwxr-x  3 dpdk dpdk  4096 Apr 11 14:56 scripts
drwxrwxr-x  2 dpdk dpdk  4096 Apr 11 14:56 tools
dpdk@dpdk:~/dpdk-16.04$
```

Install the Tools

Install the source editor of your choice. Here, CSCOPE is chosen.

1. First, **check to see whether the correct repository is enabled**.

   Check that the universe repository is enabled by inspecting `/etc/apt/sources.list`

   ```
   $ sudo gedit /etc/apt/sources.list
   ```

   As highlighted below, you may see that the archive is *restricted*.

   ```
   # See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
   # newer versions of the distribution.
   deb http://us.archive.ubuntu.com/ubuntu/ vivid main restricted
   deb-src http://us.archive.ubuntu.com/ubuntu/ vivid main restricted
   ```

If this is the case, **edit the file** by replacing *restricted* with *universe*.

```
# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.
deb http://us.archive.ubuntu.com/ubuntu/ vivid main universe
deb-src http://us.archive.ubuntu.com/ubuntu/ vivid main universe
```

Now **save** the file.

2. **Update** the system.

```
$ sudo apt-get update
```

```
dpdk@dpdk:~/dpdk-16.04$ sudo apt-get update
```

The system is updated as shown below.

```
dpdk@dpdk:~/dpdk-16.04$ sudo apt-get update
Hit http://us.archive.ubuntu.com vivid InRelease
Hit http://us.archive.ubuntu.com vivid-updates InRelease
Hit http://us.archive.ubuntu.com vivid-backports InRelease
Hit http://security.ubuntu.com vivid-security InRelease
Hit http://us.archive.ubuntu.com vivid/main Sources
Hit http://security.ubuntu.com vivid-security/main Sources
Hit http://us.archive.ubuntu.com vivid/multiverse Sources
Hit http://us.archive.ubuntu.com vivid/main amd64 Packages
```

Install CSCOPE.

```
$ sudo apt-get install cscope
```

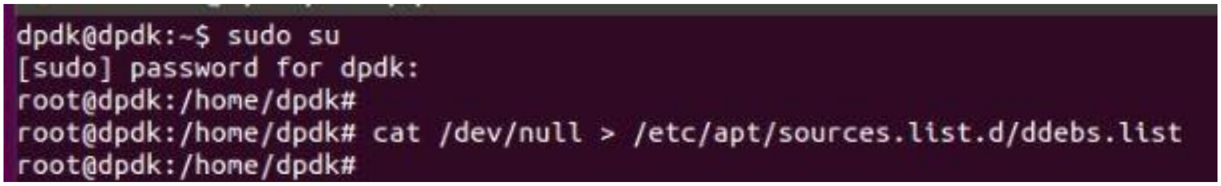
```
dpdk@dpdk:~/dpdk-16.04$ sudo apt-get install cscope
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  cscope-el
The following NEW packages will be installed:
  cscope
0 upgraded, 1 newly installed, 0 to remove and 313 not upgraded.
Need to get 149 kB of archives.
After this operation, 762 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ vivid/universe cscope amd64 15.8a-2 [149 kB]
Fetched 149 kB in 0s (344 kB/s)
Selecting previously unselected package cscope.
(Reading database ... 202629 files and directories currently installed.)
Preparing to unpack .../cscope_15.8a-2_amd64.deb ...
```

As shown above, CSCOPE 15.8a-2 is installed.

Install Kernel Debug Symbols

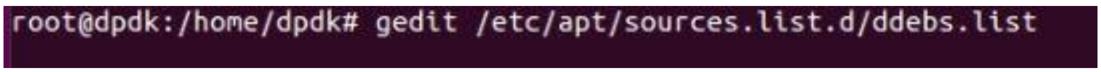
1. The first step is to **add the repository containing debugging symbols**. For that, create a new file, ddebs.list (if it does not exist already).

```
$ cat /dev/null > /etc/apt/sources.list.d/ddebs.list
```



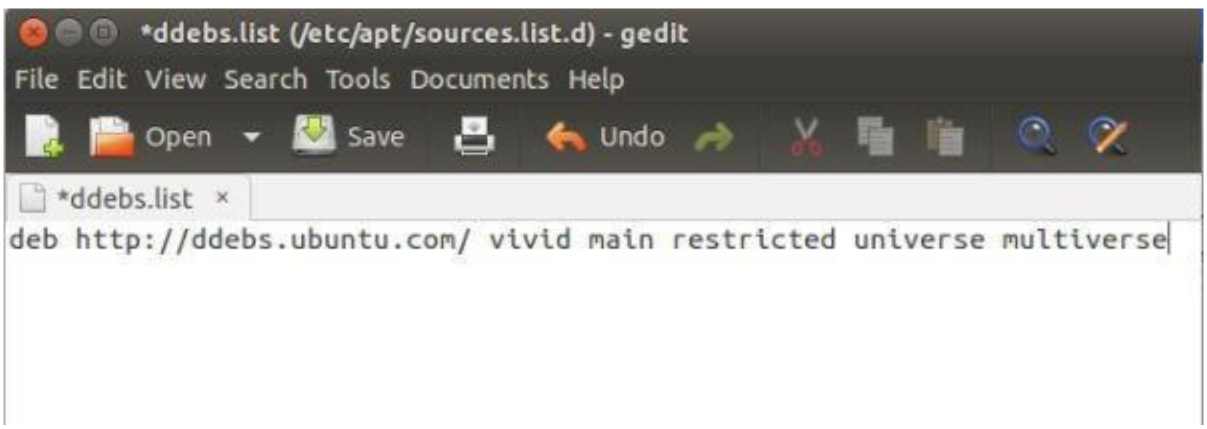
2. **Edit the file**.

```
$ gedit /etc/apt/sources.list.d/ddebs.list
```



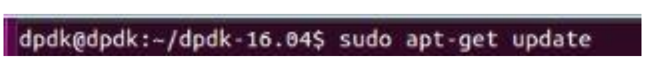
3. **Add the following line** to /etc/apt/sources.list.d/ddebs.list as shown below and **save** it.

```
deb http://ddebs.ubuntu.com/ vivid main restricted universe
multiverse
```



4. **Update the system** to load the package list from the new repository.

```
$ sudo apt-get update
```



In this example, the system gave the following error:

```
root@dpdk: /home/dpdk
Hit http://us.archive.ubuntu.com vivid-backports/universe amd64 Packages
Hit http://us.archive.ubuntu.com vivid-backports/multiverse amd64 Packages
Hit http://us.archive.ubuntu.com vivid-backports/main i386 Packages
Hit http://us.archive.ubuntu.com vivid-backports/restricted i386 Packages
Hit http://us.archive.ubuntu.com vivid-backports/universe i386 Packages
Hit http://us.archive.ubuntu.com vivid-backports/multiverse i386 Packages
Hit http://us.archive.ubuntu.com vivid-backports/main Translation-en
Hit http://us.archive.ubuntu.com vivid-backports/multiverse Translation-en
Hit http://us.archive.ubuntu.com vivid-backports/restricted Translation-en
Hit http://us.archive.ubuntu.com vivid-backports/universe Translation-en
Hit http://us.archive.ubuntu.com vivid/universe Sources
Hit http://us.archive.ubuntu.com vivid/universe amd64 Packages
Hit http://us.archive.ubuntu.com vivid/universe i386 Packages
Reading package lists... Done
W: Failed to fetch http://ddebs.ubuntu.com/dists/vivid/InRelease

W: Failed to fetch http://ddebs.ubuntu.com/dists/vivid/Release.gpg  Could not resolve 'ddebs.ubu

W: Some index files failed to download. They have been ignored, or old ones used instead.
W: Duplicate sources.list entry http://us.archive.ubuntu.com/ubuntu/ vivid/universe amd64 Packag
binary-amd64_Packages)
W: Duplicate sources.list entry http://us.archive.ubuntu.com/ubuntu/ vivid/universe i386 Package
inary-i386_Packages)
root@dpdk:/home/dpdk#
```

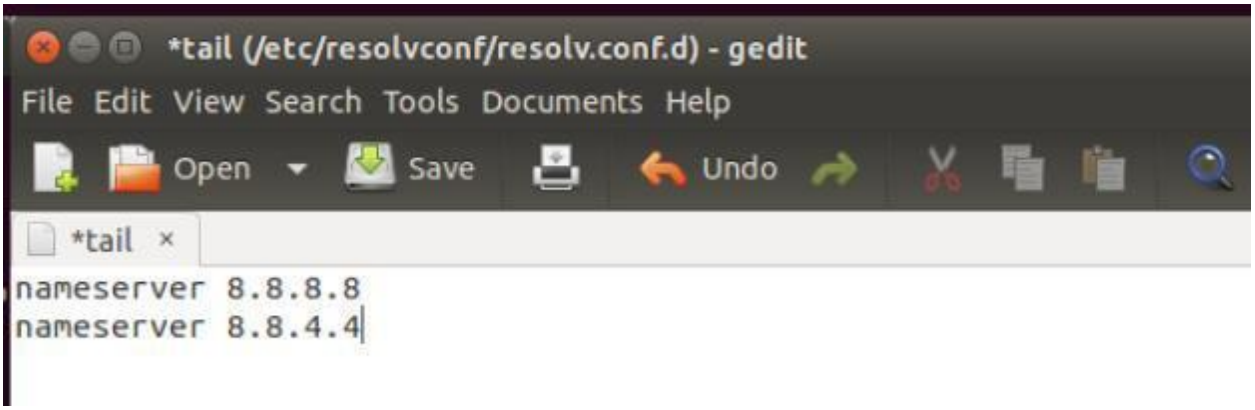
If you don't see the resolution error in your system, skip the instructions that follow and proceed to the next section.

5. To resolve name servers:

```
$ sudo gedit /etc/resolvconf/resolv.conf.d/tail
```

```
dpdk@dpdk: ~
dpdk@dpdk:~$ sudo gedit /etc/resolvconf/resolv.conf.d/tail
[sudo] password for dpdk:
```

Add to the file **the two name servers** (below) as seen in the example below, and **save** the file.



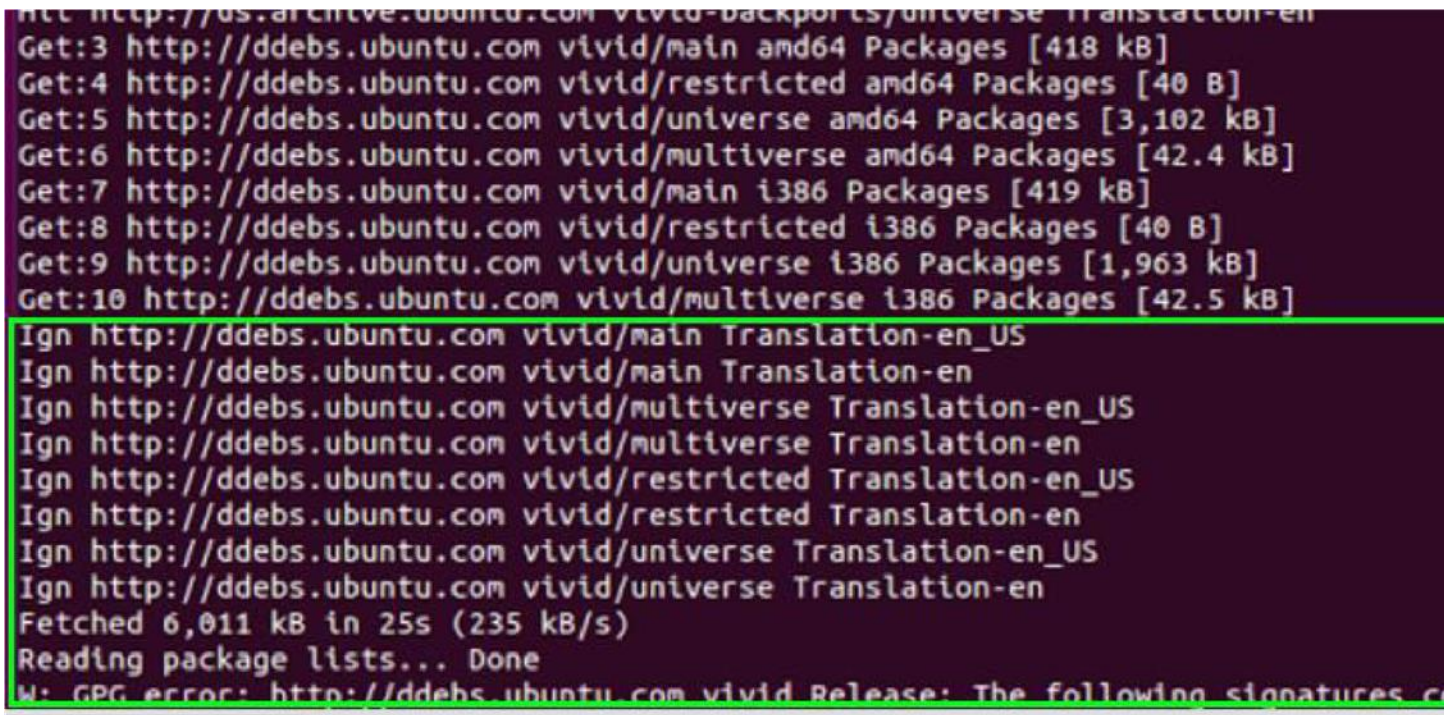
6. **Restart the service**. It is necessary to do this before the step that follows, or you'll still see the resolve error.

   ```
   $ sudo /etc/init.d/resolveconf restart
   ```

   After the shutdown and restart, **restart the service**.

7. **Update the system**.

   ```
   $ sudo apt-get update
   ```



   With the above steps, access to http://ddebs.ubuntu.com has been resolved. However there is a new error, *GPG error*, as shown at the bottom of the screenshot below.

8. **Add the GPG key**.

```
$ sudo apt-key adv –keyserver pool.sks-keyservers.net –recv-keys
C8CAB6595FDFF622
```

```
dpdk@dpdk:~$ sudo su
[sudo] password for dpdk:
root@dpdk:/home/dpdk# sudo apt-key adv --keyserver pool.sks-keyservers.net --recv-keys C8CAB6595FDFF622
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homedir /tmp/tmp.02ydF2VuKT --no-auto-check-trustdb --trust-model always --keyr
usted.gpg --keyserver pool.sks-keyservers.net --recv-keys C8CAB6595FDFF622
gpg: requesting key 5FDFF622 from hkp server pool.sks-keyservers.net
gpg: key 5FDFF622: public key "Ubuntu Debug Symbol Archive Automatic Signing Key (2016) <ubuntu-archive@lists.ubuntu.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

9. With the repository added, the next step is to **install the symbol package** by running the following command:

   ```
   apt-get install linux-image-<release>-dbgsym=<release>.<version>
   ```

   With the release as 3.19.0-59-generic and the version as 66 this is:

   $ *apt-get install linux-image-3.19.0-59-generic-dbgsym=3.19.0-59.66*

```
root@dpdk:/home/dpdk# sudo apt-get install linux-image-3.19.0-59-generic-dbgsym=3.19.0-59.6
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package linux-image-3.19.0-59-generic-dbgsym
E: Couldn't find any package by regex 'linux-image-3.19.0-59-generic-dbgsym'
root@dpdk:/home/dpdk#
```

Please note that the above resulted in an error because it could not locate the package **linuximage-3.19.0-59-generic-dbgsym**. If you want to set breakpoints by function names and viewing local variables, this error must be resolved.

10. **Install the Linux Source Package**.

```
$ sudo apt-get install linux-source-3.19.0=3.19.0-59.66
```

```
root@dpdk:/home/dpdk# sudo apt-get install linux-source-3.19.0=3.19.0-59.66
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-3.19.0-15 linux-headers-3.19.0-15-generic linux-image-3.19.0-15
Use 'apt-get autoremove' to remove them.
Suggested packages:
  libncurses-dev ncurses-dev kernel-package libqt3-dev
The following NEW packages will be installed:
  linux-source-3.19.0
0 upgraded, 1 newly installed, 0 to remove and 11 not upgraded.
Need to get 103 MB of archives.
After this operation, 119 MB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ vivid-updates/main linux-source-3.19
Fetched 103 MB in 5s (19.2 MB/s)
Selecting previously unselected package linux-source-3.19.0.
(Reading database ... 262860 files and directories currently installed.)
Preparing to unpack .../linux-source-3.19.0_3.19.0-59.66_all.deb ...
Unpacking linux-source-3.19.0 (3.19.0-59.66) ...
Setting up linux-source-3.19.0 (3.19.0-59.66) ...
root@dpdk:/home/dpdk#
```

11. With the package now installed, go to /usr/src/linux-source-3.19.0 and **unpack the source tarball**.

```
$ cd /usr/src/linux-source-3.19.0
```

$ tar xjf linux-source-3.19.0.tar.bz2

```
root@dpdk:/home/dpdk# cd /usr/src/linux-source-3.19.0
root@dpdk:/usr/src/linux-source-3.19.0# ls
debian  debian.master  linux-source-3.19.0.tar.bz2
root@dpdk:/usr/src/linux-source-3.19.0# tar xjf linux-source-3.19.0.tar.bz2
root@dpdk:/usr/src/linux-source-3.19.0#
```

Now you’re ready to **install Intel VTune Amplifier** to profile DPDK.

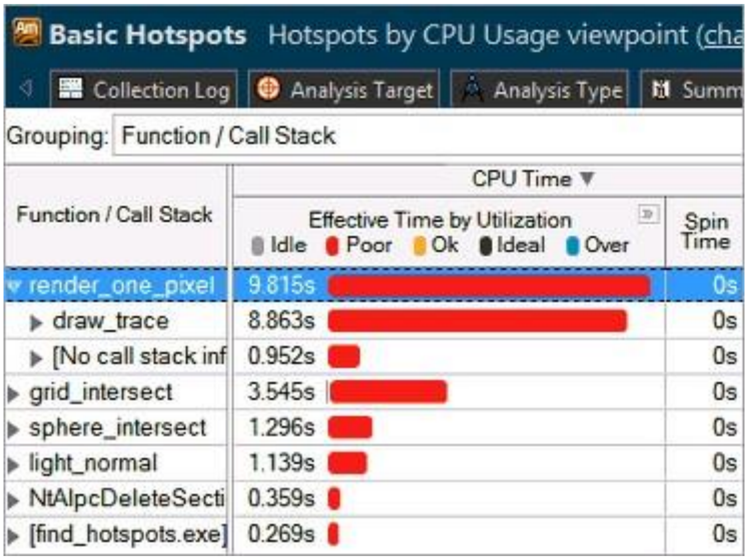
Getting Started With Intel VTune Amplifier

If you don’t have Intel VTune Amplifier installed, click https://software.intel.com/en-us/intel-vtune-amplifier-xe to get to the Intel VTune Amplifier download page. Download Intel VTune Amplifier 2018, which is the current version at the time this article was written. The articles Intel VTune Amplifier Installation Guide - Linux Host and Getting Started with Intel VTune Amplifier 2018 will guide you through the process and provide links to additional resources.

Key Features

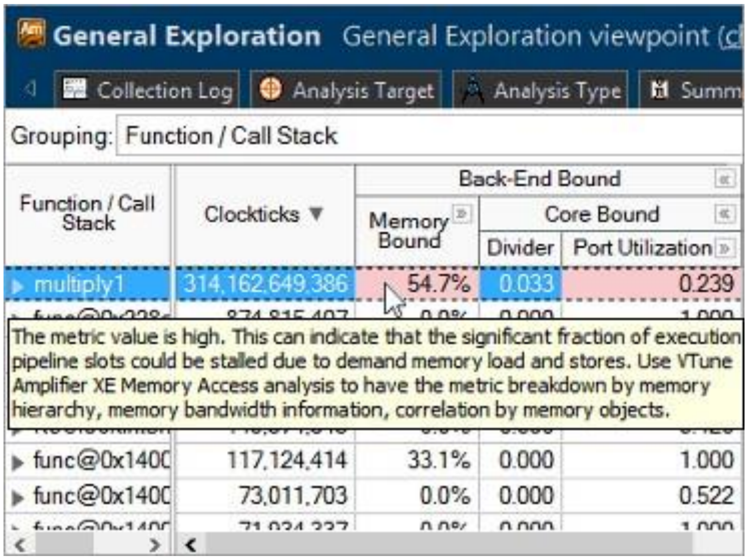
Now that you have VTune Amplifier installed, let's see what it can do. Here are some key features.

Algorithm Analysis



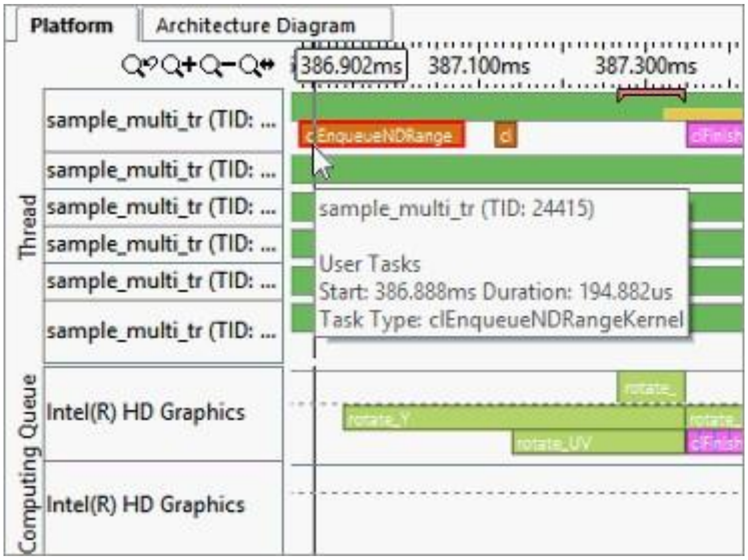
- Run **Basic Hotspots** analysis type to understand application flow and identify sections of code that get a lot of execution time (hotspots).
- Use the algorithm **Advanced Hotspots** analysis to extend Basic Hotspots analysis by collecting call stacks and analyze the CPI (Cycles Per Instructions) metric. **NEW**: You can also use this analysis type to profile native or Java* applications running in a Docker* container on a Linux system.
- Use **Memory Consumption** analysis for your native Linux or Python* targets to explore RAM usage over time and identify memory objects allocated and released during the analysis run.
- Run **Concurrency** analysis to estimate parallelization in your code and understand how effectively your application uses available cores.
- Run **Locks and Waits** analysis to identify synchronization objects preventing effective utilization of processor resources.

Microarchitecture Analysis



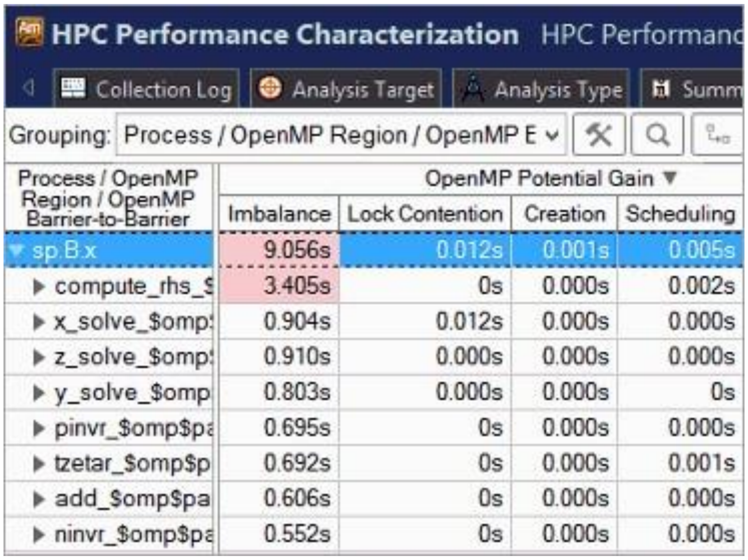
- Run **General Exploration** analysis to triage hardware issues in your application. This type collects a complete list of events for analyzing a typical client application.
- Use **Memory Access** analysis to identify memory-related issues, like NUMA problems and bandwidth limited accesses, and attribute performance events to memory objects (data structures), which is provided due to instrumentation of memory allocations/de-allocations and getting static/global variables from symbol information.
- For systems with Intel® Software Guard Extensions (Intel® SGX) feature enabled, run **SGX**.
- Run **Hotspots** analysis to identify performance-critical program units inside security enclaves. This analysis type uses the INST_RETIRED.PREC_DIST hardware event that emulates precise clock ticks, which is mandatory for the analysis on the systems with Intel SGX enabled.
- For the Intel processors supporting Intel® Transactional Synchronization Extensions (Intel® TSX), run the **TSX Exploration** and **TSX Hotspots** analysis types to measure transactional success and analyze causes of transactional aborts.

Platform Analysis



- Run **System Overview** analysis to review general behavior of a target Linux or Android* system and correlate power and performance metrics with the interrupt request (IRQ).
- Run **CPU/GPU Concurrency** analysis to identify code regions where your application is CPU- or GPU-bound.
- Use **GPU Hotspots** analysis to identify GPU tasks with high GPU utilization, and estimate the effectiveness of this utilization.
- For GPU-bound applications running on Intel® HD Graphics, collect GPU hardware events to estimate how effectively the processor graphics are used.
- Collect data on ftrace* events on Android and Linux targets and Atrace* events on Android targets.
- Analyze hot Intel® Media SDK programs and OpenCL™ kernels running on a GPU. For OpenCL application analysis, use the architecture diagram to explore GPU hardware metrics per GPU architecture blocks.
- Run **Disk Input and Output** analysis to monitor utilization of the disk subsystem, CPU, and processor buses. This analysis type provides a consistent view of the storage subsystem combined with hardware events and an easy-to-use method to match user-level source code with I/O packets executed by the hardware.

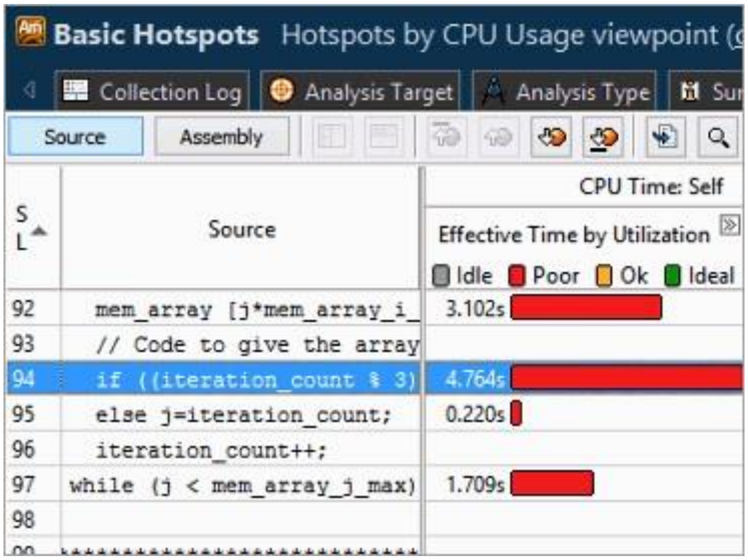
Compute-Intensive Applications Analysis



Process / OpenMP Region / OpenMP Barrier-to-Barrier	OpenMP Potential Gain			
	Imbalance	Lock Contention	Creation	Scheduling
sp.B.x	9.056s	0.012s	0.001s	0.005s
compute_rhs_$	3.405s	0s	0.000s	0.002s
x_solve_$omp:	0.904s	0.012s	0.000s	0.000s
z_solve_$omp:	0.910s	0.000s	0.000s	0.000s
y_solve_$omp	0.803s	0.000s	0.000s	0s
pinvr_omppa	0.695s	0s	0.000s	0.000s
tzetar_ompp	0.692s	0s	0.000s	0.001s
add_omppa	0.606s	0s	0.000s	0.000s
ninvr_omppa	0.552s	0s	0.000s	0.000s

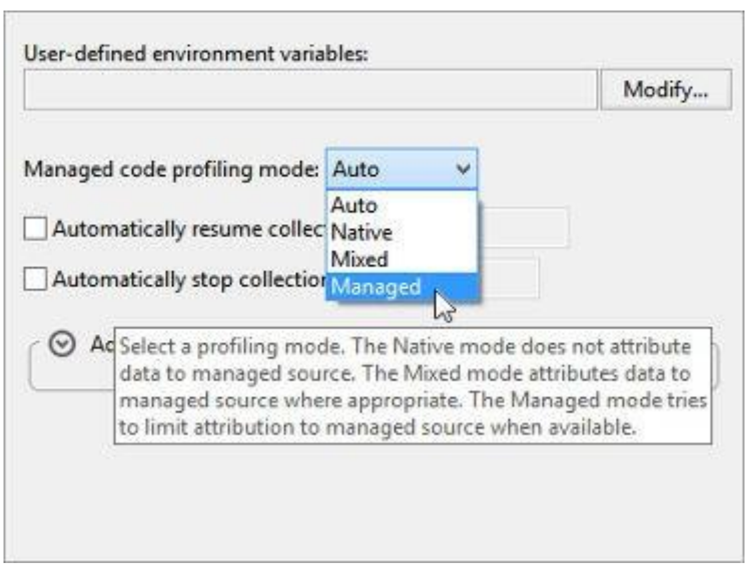
- Run **HPC Performance Characterization** analysis to identify how effectively your high-performance computing application uses CPU, memory, and floating-point operation hardware resources. This analysis type provides additional scalability metrics for applications that use OpenMP* or Intel® MPI Library runtimes.
- Run an algorithm analysis type with the **Analyze OpenMP regions** option enabled to collect OpenMP or Intel MPI data for applications using OpenMP or Intel MPI runtime libraries. Note that **HPC Performance Characterization** analysis has the option enabled by default.
- For OpenMP applications, analyze the collected performance data to identify inefficiencies in parallelization. Review the potential gain metric values per OpenMP region to understand the maximum time that could be saved if the OpenMP region is optimized to have no load imbalance, assuming no runtime overhead.
- For hybrid OpenMP and Intel MPI applications, explore OpenMP efficiency metrics by Intel MPI processes laying on the critical path.

Source Analysis



- Double-click a hotspot function to drill down to the source code and analyze performance per source line or assembler instruction. By default, the hottest line is highlighted.
- For help on an assembly instruction, right-click the instruction in the Assembly pane and select **Instruction Reference** from the context menu.

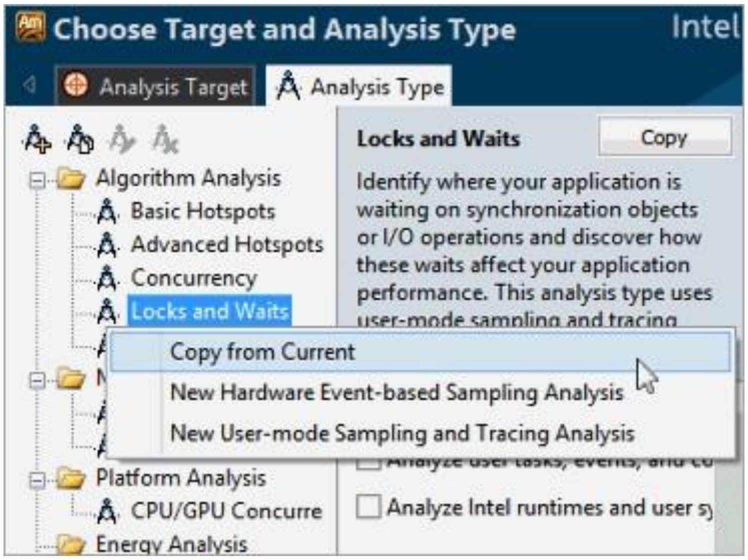
Managed Code Analysis



Configure target options for managed code analysis in the native, managed, or mixed mode:

- **Windows host only:** Event-based sampling (EBS) analysis for Windows Store C/C++, C# and JavaScript* applications running in the **Attach** or **System-wide** mode.
- EBS or user-mode sampling and tracing analysis for Java applications running in the **Launch Application** or **Attach** mode.
- **Basic Hotspots** and **Locks and Waits** analysis for Python applications running in the **Launch Application** and **Attach to Process** modes.

Custom Analysis



- Select the **Custom Analysis** branch in the analysis tree to create your own analysis configurations using any of the available VTune Amplifier data collectors.

- Run your own custom collector from the VTune Amplifier to get the aggregated performance data from your custom collection and VTune Amplifier analysis in the same result.

- Import performance data collected by your own or third-party collector into the VTune Amplifier result collected in parallel with your external collection. Use the **Import from CSV** button to integrate the external data to the result.

- Collect data from a remote virtual machine by configuring KVM guest OS profiling, which makes use of the Linux Perf KVM feature. Select **Analyze KVM guest OS** from the **Advanced** options.

Remote Collection Modes

You can collect data on your Linux, Windows, or Android system using any of the following modes:

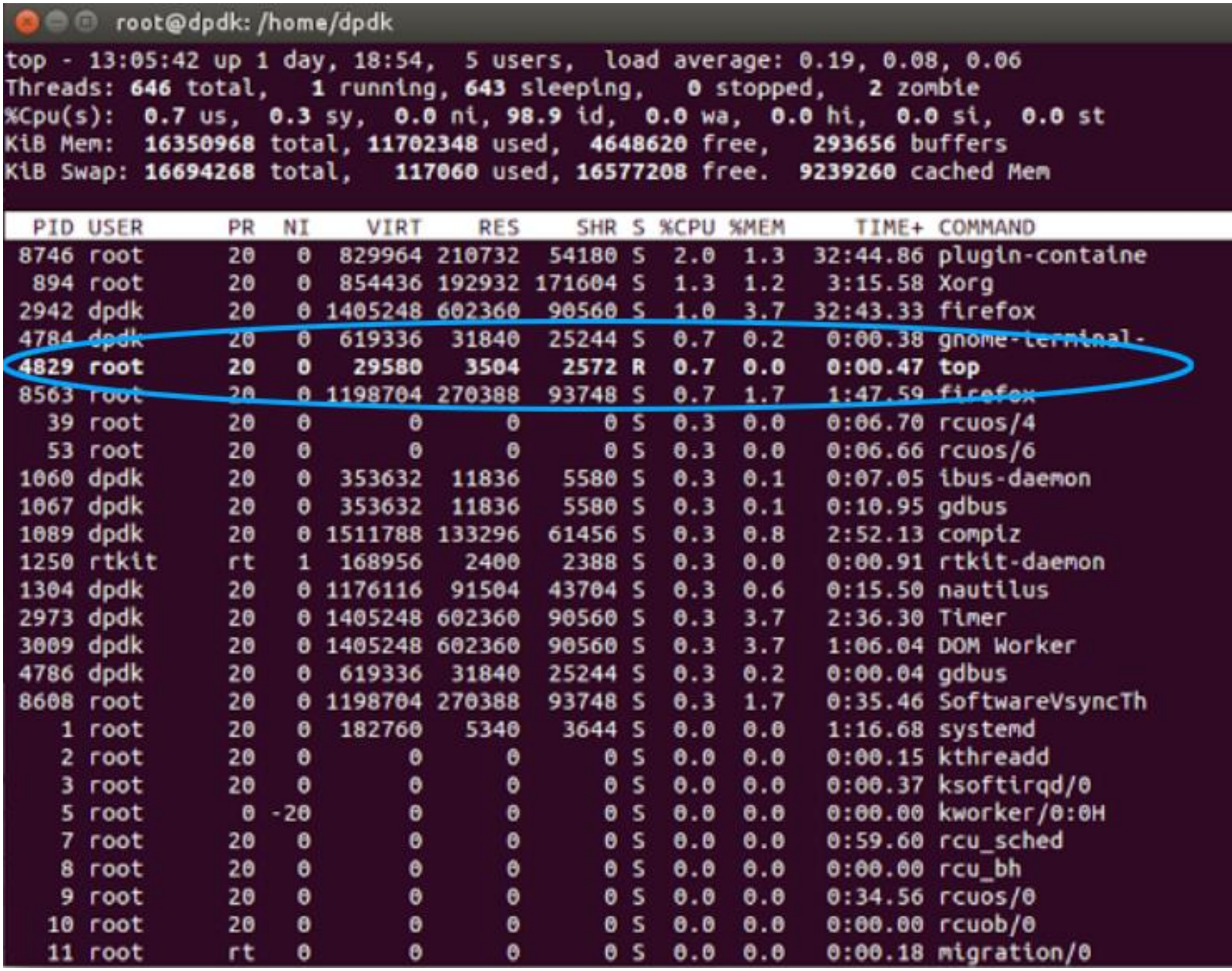
- (Linux and Android targets) Remote analysis via SSH/ADB communication with VTune Amplifier graphical and command-line interface (amplxe-cl) installed on the host and VTune Amplifier target package installed on the remote target system. Recommended for resource-constrained, embedded platforms (with insufficient disk space, memory, or CPU power).
- (Android targets) Disconnected analysis via SSH/ADB communication with VTune Amplifier installed on the host and the VTune Amplifier target package installed on the remote Android system. The analysis is initiated from the host system, but data collection does not begin until the device is unplugged from the host system. The results are finalized after the device is reconnected to the host system.
- (Linux and Windows targets) Native performance analysis with the VTune Amplifier graphical or command line interface installed on the target system. Analysis is started directly on the target system.
- (Linux and Windows targets) Native hardware event-based sampling analysis with the VTune Amplifier's Sampling Enabling Product (SEP) installed on the target embedded system.

Stepping Back to See the Big Picture

It's a good idea to step back and see the big picture first—as to what other components exist in the system. If there are some unrelated component-consuming resources, and if we only focus on measuring our specific application, then we may be coming to a wrong conclusion because of partial information.

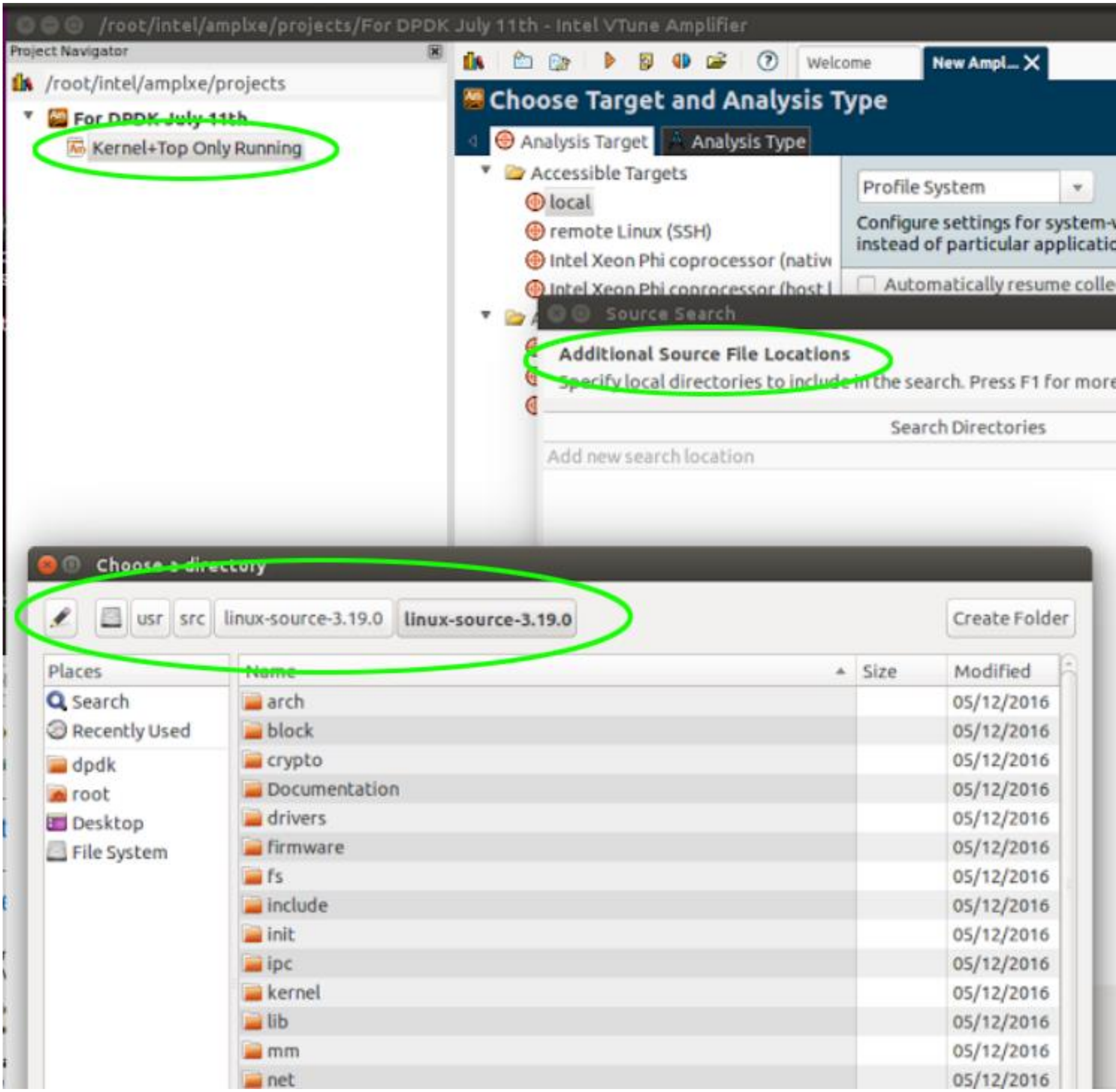
So here, even before running the DPDK application, we run `top -H` to see where the CPU is spending its cycles without our specific application running.

Below you will see the VTune Amplifier showing `top -H` and the Firefox* web browser running. Now, `top` is something you just ran, whereas Firefox is something you don't want taking CPU cycles while you evaluate your application of interest. Similarly, you may find some unwanted daemons. So at this point, stop any unwanted applications, daemons, and other components.



Pointing to the Source Directory

The following screenshot shows how to point to the source directory of the software components of interest in VTune Amplifier. You can add multiple directories.



Profiling DPDK Code with VTune Amplifier

1. First, we'll **reserve huge pages**. Note that we've chosen 128 huge pages here to accommodate a possible memory constraint when testing on a laptop. If you're using a server or desktop, you can specify 1024 huge pages**.**

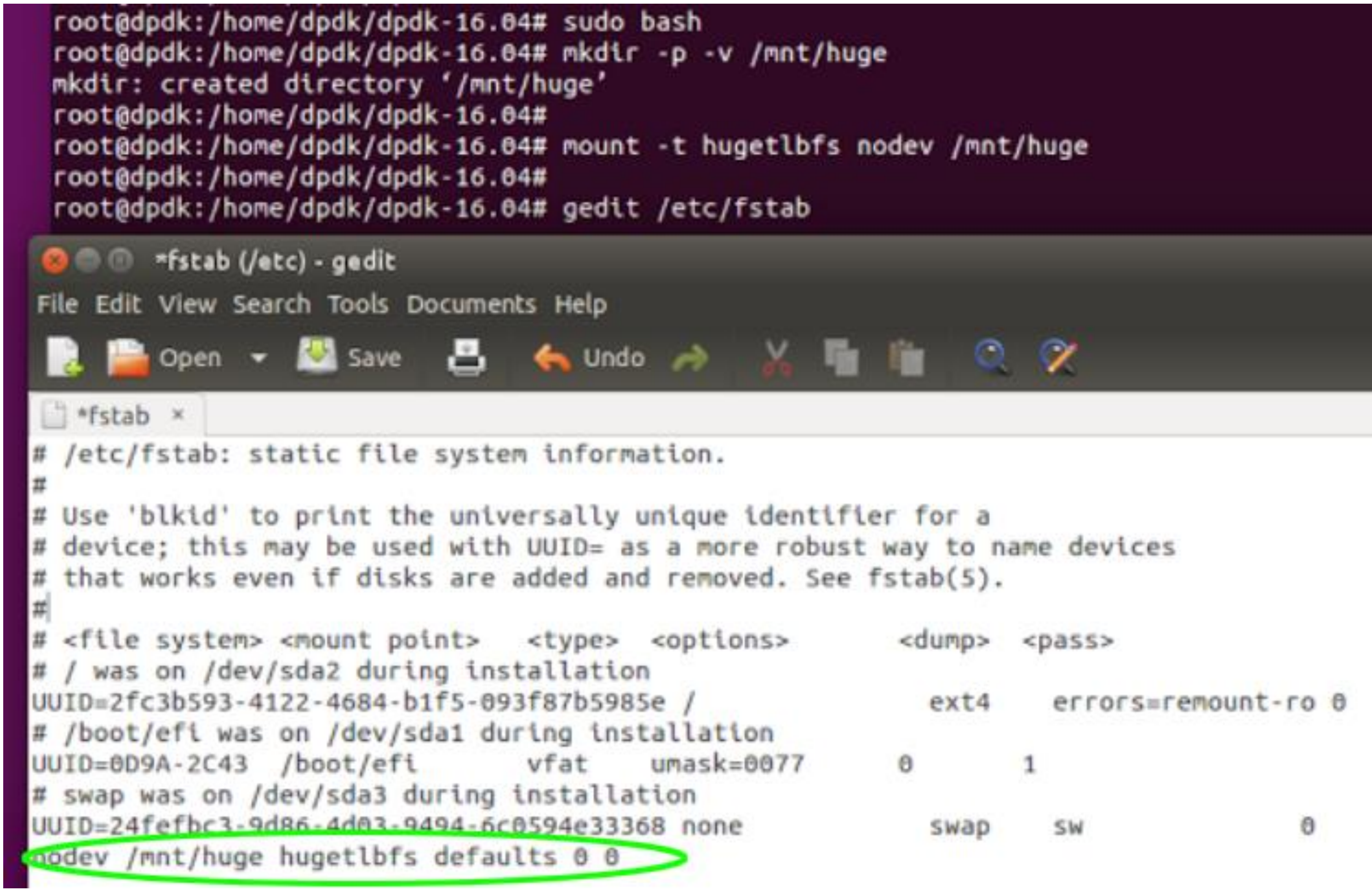
```
$ cd /home/dpdk/dpdk-16.04

$ sudo su

$ echo 128 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

2. Creating `/mnt/huge` and mounting as `hgetlbfs`

   Next, we'll **create /mnt/huge and mount it as hgetlbfs**.

```
$ sudo bash

$ mkdir –p –v /mnt/huge    [-v for verbose, as you can see below
response from the system]

$ mount –t hugetlbfs nodev /mnt/huge

Making the mount point permanent across reboots, by adding the
following line to the /etc/fstab file:

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

Look at /etc/fstab to **confirm that /mt/huge was successfully created** and mounted. See example below:



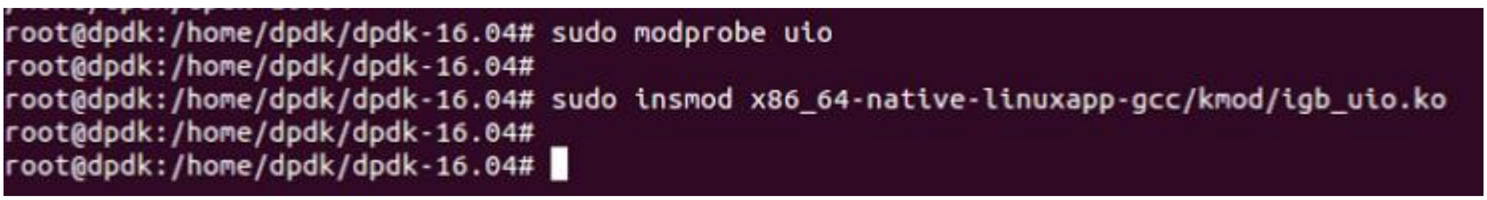
3. **Build the DPDK test application and DPDK library**:

```
$ export RTE_SDK=/home/dpdk/dpdk-16.04

$ export RTE_TARGET=x86_64-native-linuxapp-gcc

$ export EXTRA_CFLAGS='-g'    [For DPDK symbols]

$ make install T=x86_64-native-linuxapp-gcc DESTDIR=install
```

The output of the build will complete successfully, as shown below.

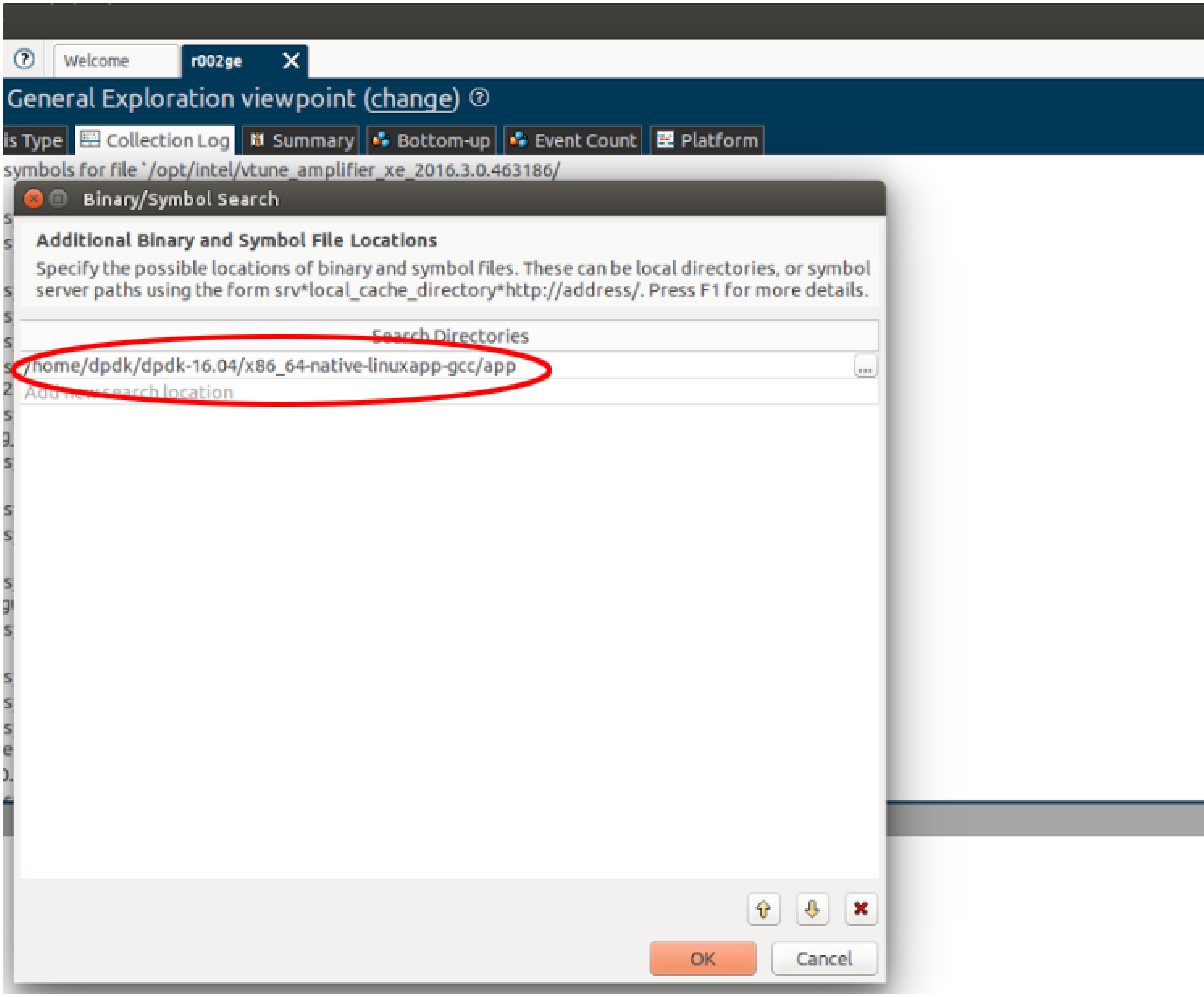
4. **Load `uio` modules** to enable userspace IO for DPDK.

```
$ sudo modprobe uio

$ sudo insmod x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
```

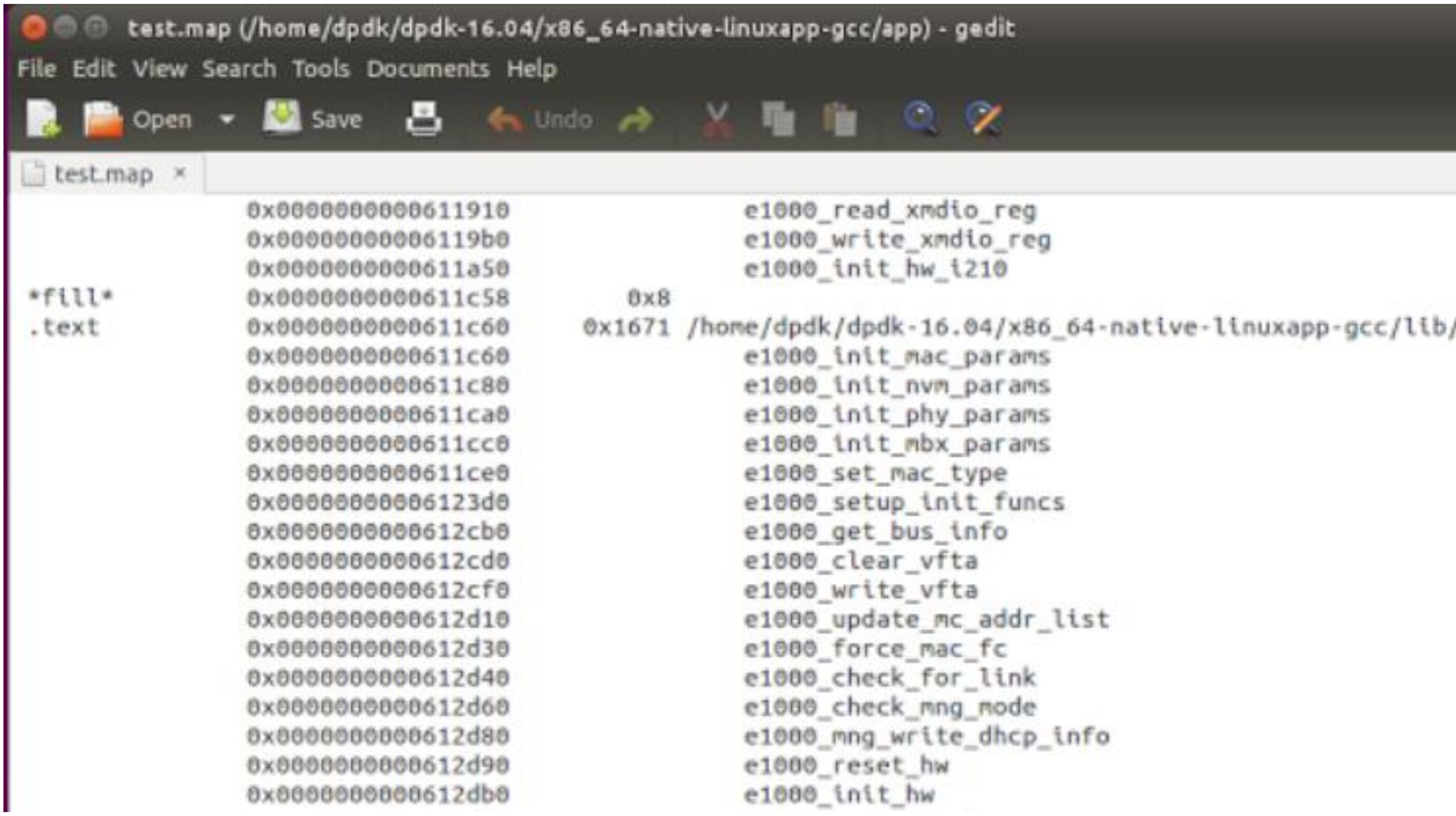


5. **Add path to DPDK test application symbols** to VTune Amplifier.

   See the image below to illustrate this step.



You can verify the symbols in the above directory in test.map, as shown in the image below.



At this point, you are ready to get started profiling your DPDK code with VTune Amplifier.

Profiling DPDK Code with VTune Amplifier

Now we will run a handful of micro benchmarks. To start, **cd to the directory below and run ./test**.

```
$ cd /home/dpdk/dpdk-16.04/x86_64-native-linuxapp-gcc/app

$ sudo su

$ ./test
```

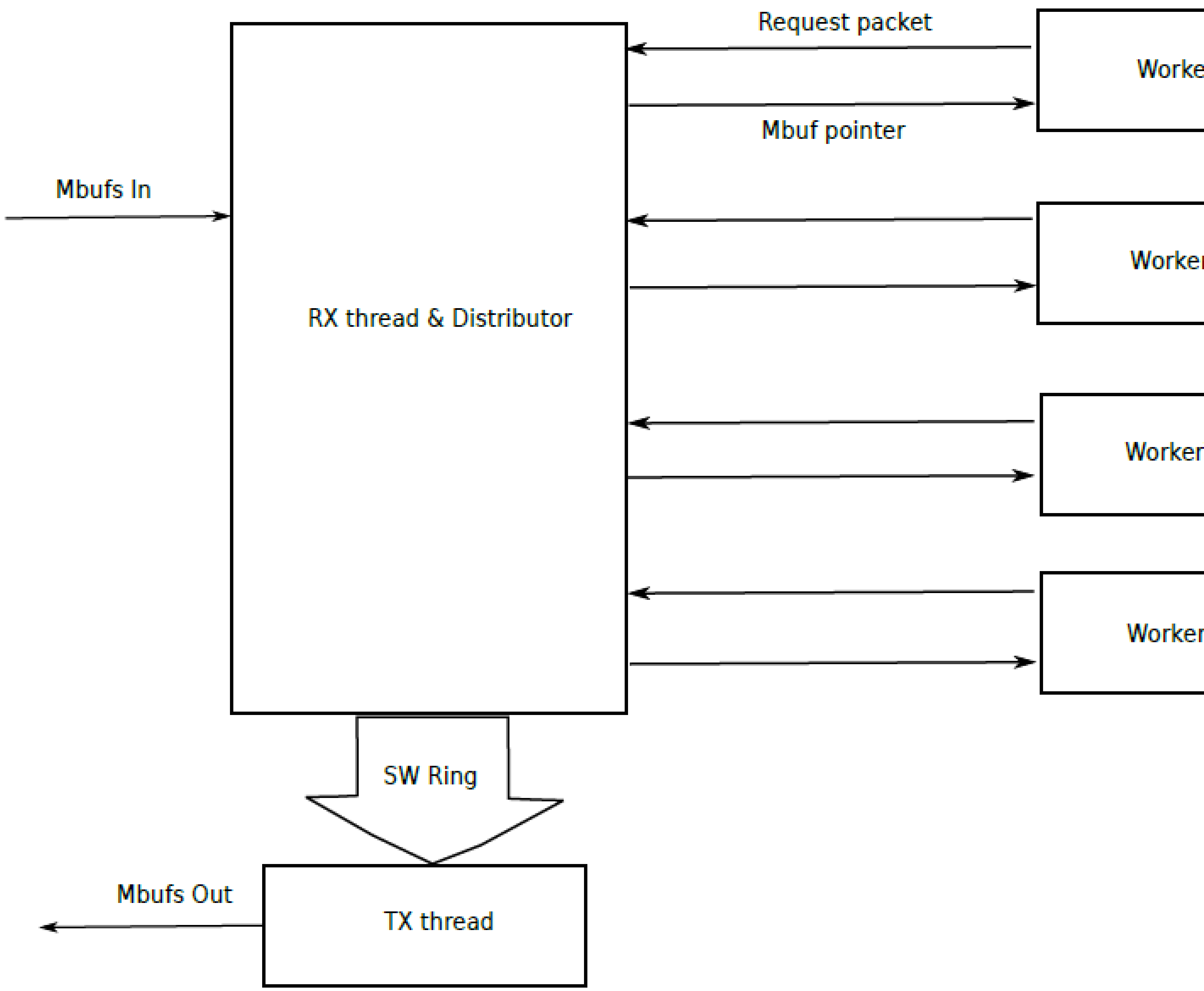
The test will issue prompt RTE>> as shown below. Enter `?` for help and the list of available tests.

Profiling Distributor Perf Autotest

Our first test will be the `distributor_perf_autotest.` A diagram describing this application is below.

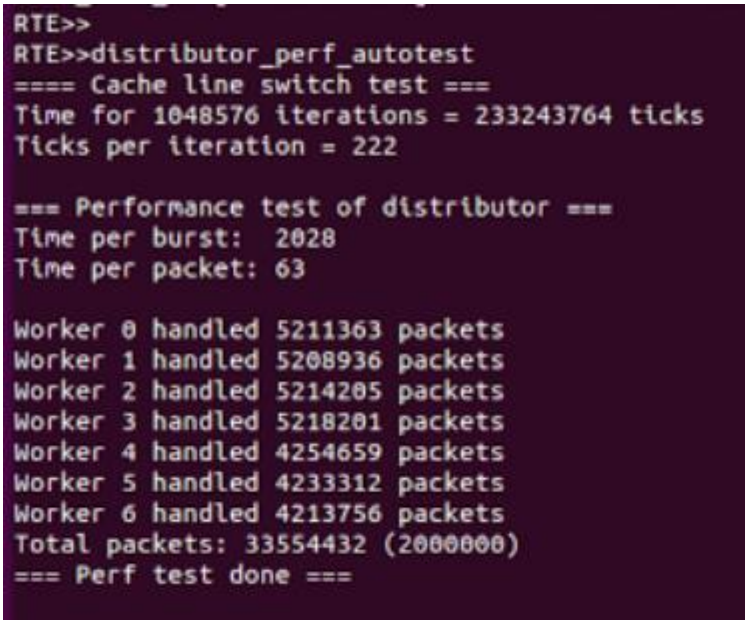
Select the test from the options offered by RTE.

```
RTE>> distributor_perf_autotest
```



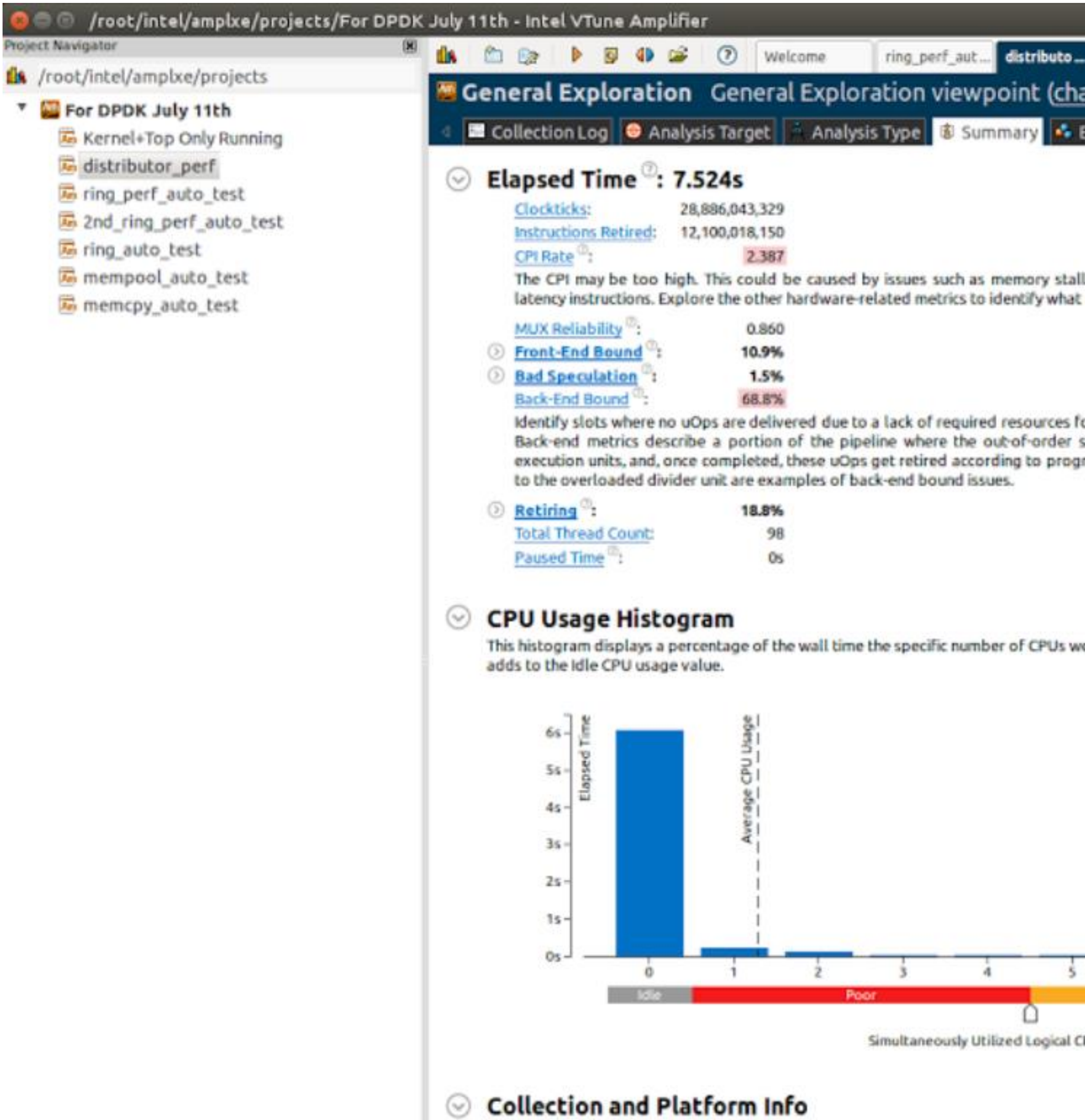
Distributor Sample Application Layout

See below for command window output during the test run.



```
RTE>>
RTE>>distributor_perf_autotest
==== Cache line switch test ===
Time for 1048576 iterations = 233243764 ticks
Ticks per iteration = 222

=== Performance test of distributor ===
Time per burst:  2028
Time per packet: 63

Worker 0 handled 5211363 packets
Worker 1 handled 5208936 packets
Worker 2 handled 5214205 packets
Worker 3 handled 5218201 packets
Worker 4 handled 4254659 packets
Worker 5 handled 4233312 packets
Worker 6 handled 4213756 packets
Total packets: 33554432 (2000000)
=== Perf test done ===
```

The VTune Amplifier summary highlights CPI rate, indicating it is beyond the normal range. It also highlights *Back End-Bound*, indicating a memory-bound application nature. See these results on the screen capture below.



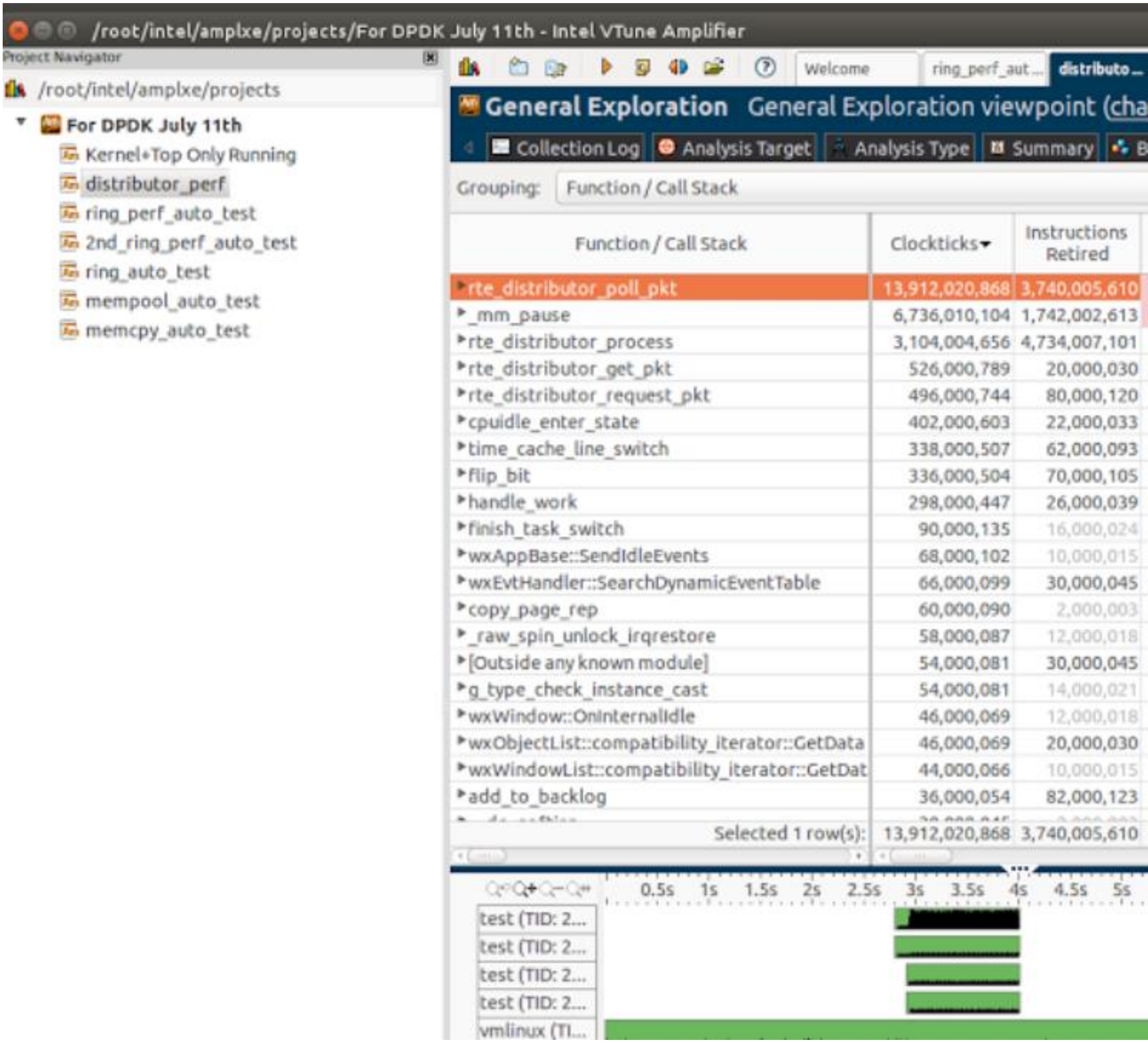
/root/intel/amplxe/projects/For DPDK July 11th - Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
For DPDK July 11th
Kernel+Top Only Running
distributor_perf
ring_perf_auto_test
2nd_ring_perf_auto_test
ring_auto_test
mempool_auto_test
memcpy_auto_test
Welcome
ring_perf_aut...
distributo...
General Exploration General Exploration viewpoint
Collection Log
Analysis Target
Analysis Type
Summary
Elapsed Time: 7.524s
Clockticks: 28,886,043,329
Instructions Retired: 12,100,018,150
CPI Rate: 2.387
The CPI may be too high. This could be caused by issues such as memory stall
latency instructions. Explore the other hardware-related metrics to identify what
MUX Reliability: 0.860
Front-End Bound: 10.9%
Bad Speculation: 1.5%
Back-End Bound: 68.8%
Identify slots where no uOps are delivered due to a lack of required resources f
Back-end metrics describe a portion of the pipeline where the out-of-order s
execution units, and, once completed, these uOps get retired according to prog
to the overloaded divider unit are examples of back-end bound issues.
Retiring: 18.8%
Total Thread Count: 98
Paused Time: 0s
CPU Usage Histogram
This histogram displays a percentage of the wall time the specific number of CPUs w
adds to the Idle CPU usage value.
Elapsed Time
Average CPU Usage
6s
5s
4s
3s
2s
1s
0s
0
1
2
3
4
5
Idle
Poor
Simultaneously Utilized Logical C
Collection and Platform Info

Analysis Details

- **Function/Call Stack** indicates rte_distributor_poll_pkt consumes CPI at a rate of 3.720 and mm_pause consumes CPI at a rate of 3.867.

You can observe that rte_distributor_get_pkt runs with a CPI rate of 26.30. However, it is not highlighted, since it uses fewer clock ticks than the highlighted functions.

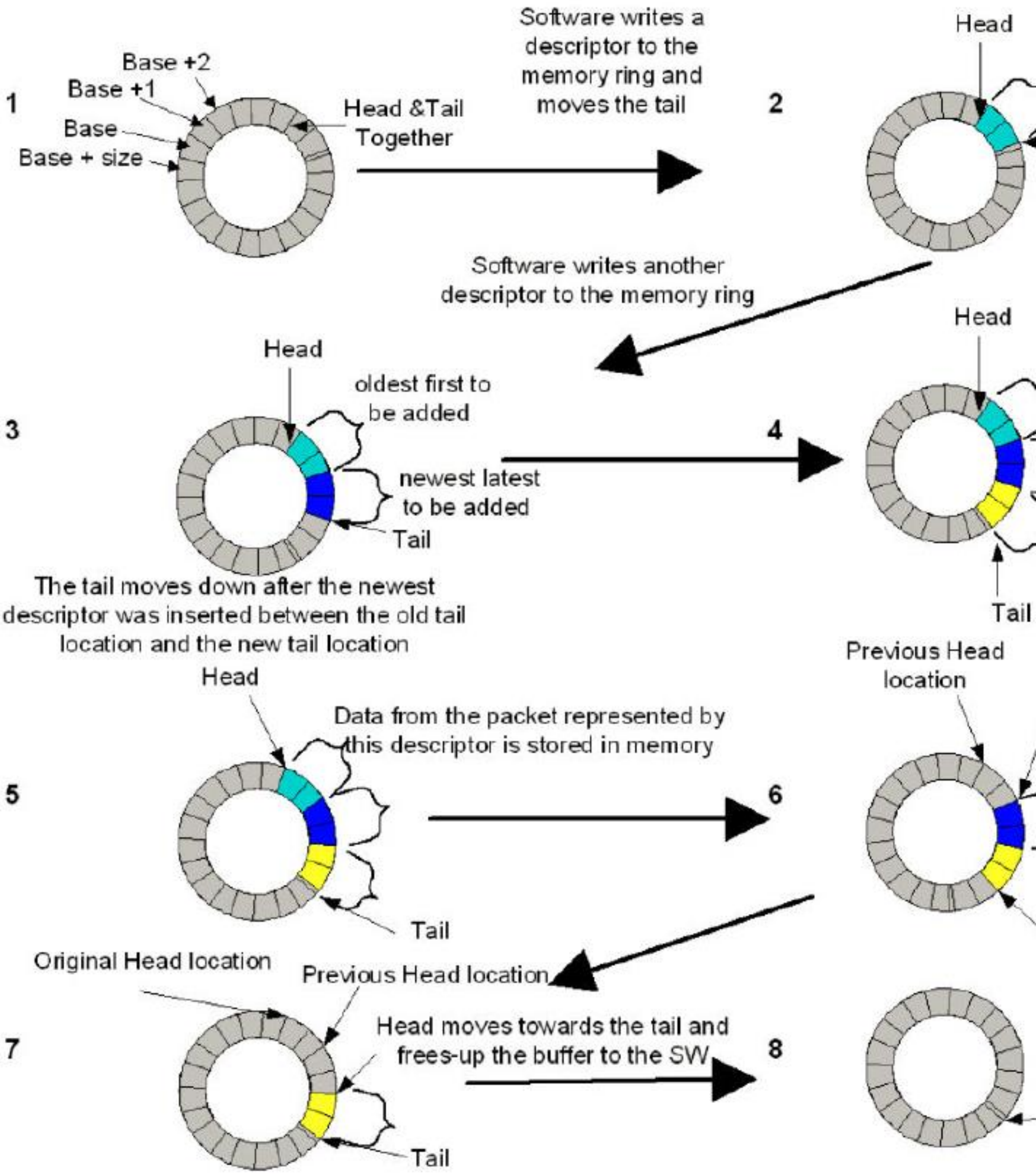
You will see other functions listed here along with the CPI each one uses, for example: rte_distributor_process, rte_distributor_request_pkt, time_cache_line_switch.



Profiling Rings

Communication between cores for interprocessor communication as well as communication between cores and NIC happens through rings and descriptors.

While NIC hardware does optimizations in terms of RS bit and descriptor done bit (DD bit) in bunching the data size, DPDK in addition enhances bunching with amortizing by offering API for bulk communication through rings. The graphic below illustrates ring communication.


1
Base +2
Base +1
Base
Base + size
Head &Tail
Together
Software writes a
descriptor to the
memory ring and
moves the tail
2
Head
Software writes another
descriptor to the memory ring
3
Head
oldest first to
be added
newest latest
to be added
Tail
The tail moves down after the newest
descriptor was inserted between the old tail
location and the new tail location
4
Head
Tail
5
Head
Data from the packet represented by
this descriptor is stored in memory
Tail
6
Previous Head
location
7
Original Head location
Previous Head location
Head moves towards the tail and
frees-up the buffer to the SW
Tail
8

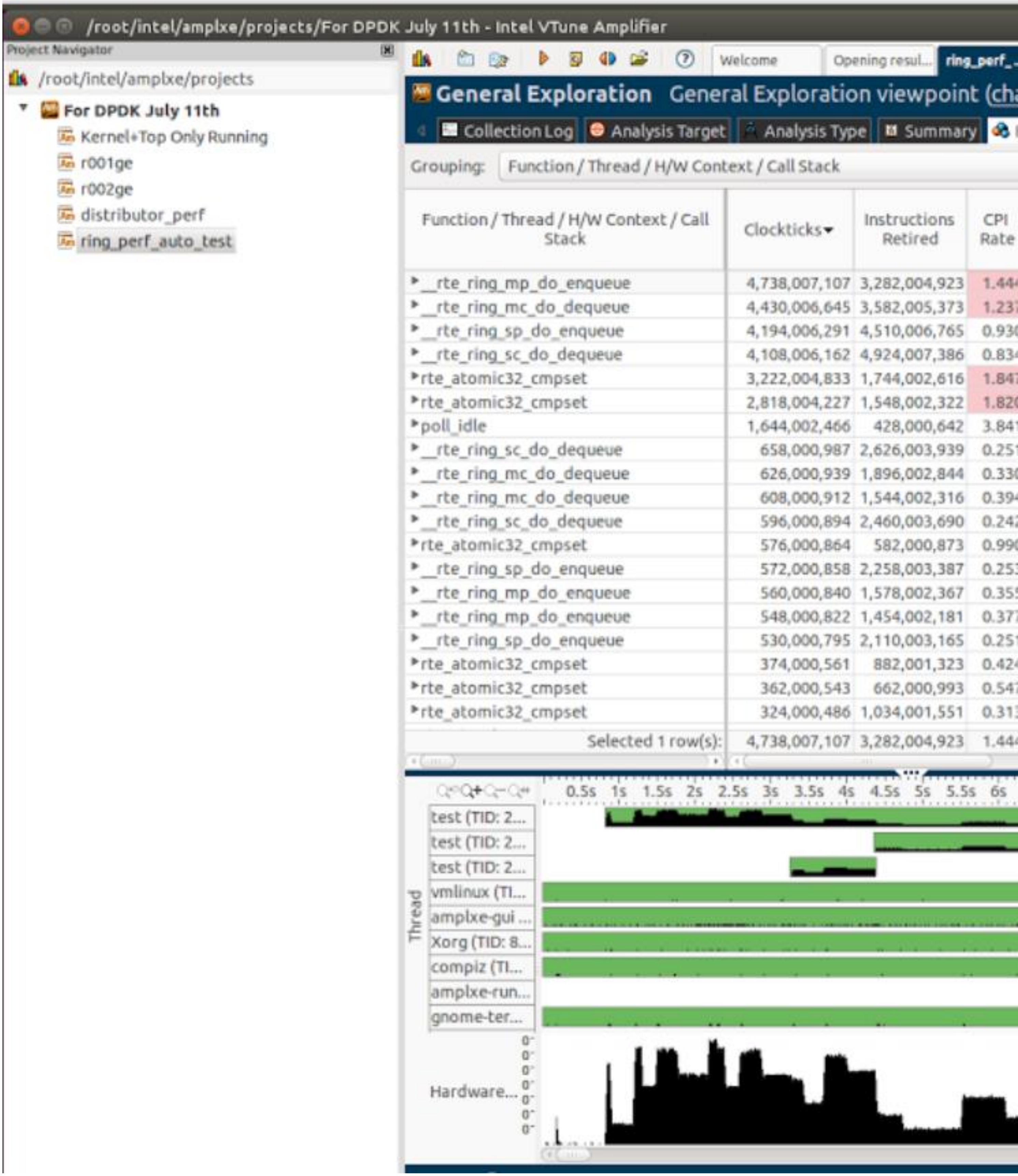
The rings tests show that single producer/single consumer (SP/SC) with bulk sizes both in enqueue/dequeue gives best performance compared to multiple producers/multiple consumers (MP/MC). Below are the steps.

Profiling ring_perf_autotest

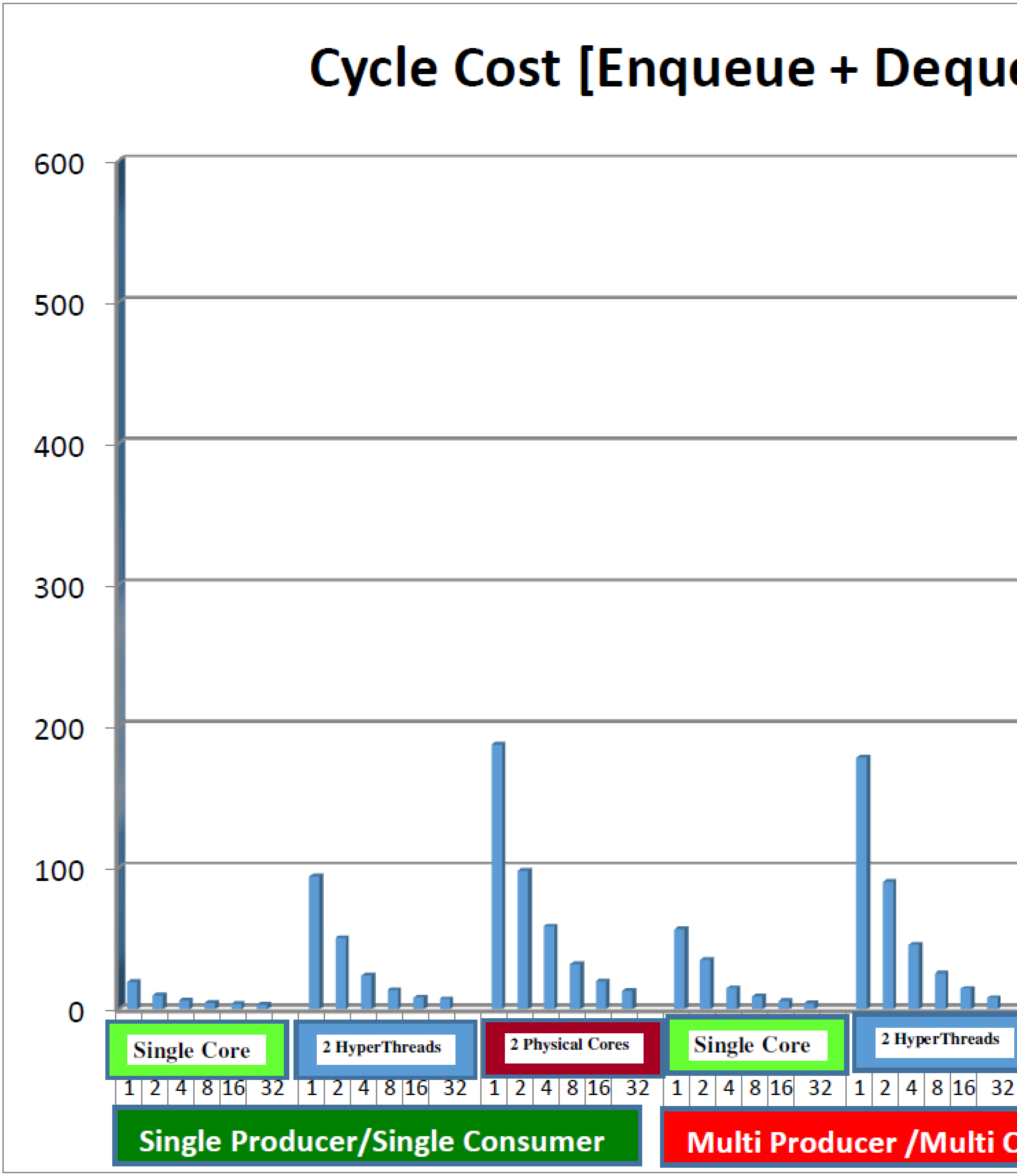
In RTE, select `ring_perf_autotest`. Test output is shown in the cmd window below.

```
RTE>>ring_perf_autotest
### Testing single element and burst enq/deq ###
SP/SC single enq/dequeue: 6
MP/MC single enq/dequeue: 37
SP/SC burst enq/dequeue (size: 8): 2
MP/MC burst enq/dequeue (size: 8): 5
SP/SC burst enq/dequeue (size: 32): 2
MP/MC burst enq/dequeue (size: 32): 3

### Testing empty dequeue ###
SC empty dequeue: 1.34
MC empty dequeue: 1.90

### Testing using a single lcore ###
SP/SC bulk enq/dequeue (size: 8): 3.08
MP/MC bulk enq/dequeue (size: 8): 5.82
SP/SC bulk enq/dequeue (size: 32): 2.13
MP/MC bulk enq/dequeue (size: 32): 3.00

### Testing using two hyperthreads ###
SP/SC bulk enq/dequeue (size: 8): 9.18
MP/MC bulk enq/dequeue (size: 8): 15.96
SP/SC bulk enq/dequeue (size: 32): 4.52
MP/MC bulk enq/dequeue (size: 32): 5.68

### Testing using two physical cores ###
SP/SC bulk enq/dequeue (size: 8): 20.62
MP/MC bulk enq/dequeue (size: 8): 46.49
SP/SC bulk enq/dequeue (size: 32): 8.52
MP/MC bulk enq/dequeue (size: 32): 15.80
Test OK
```

VTune Amplifier output for `ring_perf_autotest` shows in detail that the code is backend-bound. You can see the call stack showing results for SP/SC with bulk sizes as well as MP/MC.


/root/intel/amplxe/projects/For DPDK July 11th - Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
For DPDK July 11th
Kernel+Top Only Running
r001ge
r002ge
distributor_perf
ring_perf_auto_test
Welcome
Opening resul...
General Exploration
General Exploration viewpoint
Collection Log
Analysis Target
Analysis Type
Summary
Grouping:
Function / Thread / H/W Context / Call Stack
Function / Thread / H/W Context / Call Stack
Clockticks
Instructions Retired
CPI Rate
__rte_ring_mp_do_enqueue 4,738,007,107 3,282,004,923
__rte_ring_mc_do_dequeue 4,430,006,645 3,582,005,373
__rte_ring_sp_do_enqueue 4,194,006,291 4,510,006,765
__rte_ring_sc_do_dequeue 4,108,006,162 4,924,007,386
rte_atomic32_cmpset 3,222,004,833 1,744,002,616
rte_atomic32_cmpset 2,818,004,227 1,548,002,322
poll_idle 1,644,002,466 428,000,642
__rte_ring_sc_do_dequeue 658,000,987 2,626,003,939
__rte_ring_mc_do_dequeue 626,000,939 1,896,002,844
__rte_ring_mc_do_dequeue 608,000,912 1,544,002,316
__rte_ring_sc_do_dequeue 596,000,894 2,460,003,690
rte_atomic32_cmpset 576,000,864 582,000,873
__rte_ring_sp_do_enqueue 572,000,858 2,258,003,387
__rte_ring_mp_do_enqueue 560,000,840 1,578,002,367
__rte_ring_mp_do_enqueue 548,000,822 1,454,002,181
__rte_ring_sp_do_enqueue 530,000,795 2,110,003,165
rte_atomic32_cmpset 374,000,561 882,001,323
rte_atomic32_cmpset 362,000,543 662,000,993
rte_atomic32_cmpset 324,000,486 1,034,001,551
Selected 1 row(s): 4,738,007,107 3,282,004,923
0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4s 4.5s 5s 5.5s 6s
Thread
test (TID: 2...
test (TID: 2...
test (TID: 2...
vmlinux (TI...
amplxe-gui ...
Xorg (TID: 8...
compiz (TI...
amplxe-run...
gnome-ter...
Hardware...

To appreciate the relative performance of SP/SC with single data size and bulk size, and comparing with MP/MC with single data size and bulk size, refer to the following graph. Please note the impact of core placement—a) siblings, b) within the same socket, c) across multisockets.



Cycle Cost [Enqueue + Deque
600
500
400
300
200
100
0
Single Core
2 HyperThreads
2 Physical Cores
Single Core
2 HyperThreads
1 2 4 8 16 32 1 2 4 8 16 32 1 2 4 8 16 32 1 2 4 8 16 32 1 2 4 8 16 32
Single Producer/Single Consumer
Multi Producer /Multi C

Conclusion and Next Steps

Practice profiling on additional sample DPDK applications. With the experience you gather, extend profiling and optimization to the applications you are building on top of DPDK.

Get plugged in to the DPDK community to learn the latest from developers and architects and keep your products highly optimized. Register at https://www.dpdk.org/contribute/.

References

Enabling Internet connectivity: http://askubuntu.com/questions/641591/internet-connection-not-working-in-ubuntu-15-04

Getting Kernel Symbols/Sources on Ubuntu Linux: http://sysprogs.com/VisualKernel/tutorials/setup/ubuntu/

How to debug libraries in Ubuntu: http://stackoverflow.com/questions/14344654/how-to-use-debug-libraries-on-ubuntu

How to install a package that contains Ubuntu debug symbols: http://askubuntu.com/questions/197016/how-to-install-a-package-that-contains-ubuntu-kernel-debug-symbols

Debug Symbol Packages: https://wiki.ubuntu.com/Debug%20Symbol%20Packages

Ask Ubuntu for challenges in Apt-get update failure to fetch: http://askubuntu.com/questions/135932/apt-get-update-failure-to-fetch-cant-connect-to-any-sources

DNS Name Server IP Address: http://www.cyberciti.biz/faq/ubuntu-linux-configure-dns-nameserver-ip-address/

How to fix public key is not available issue: https://chrisjean.com/fix-apt-get-update-the-following-signatures-couldnt-be-verified-because-the-public-key-is-not-available/

Ubuntu Key server: http://keyserver.ubuntu.com:11371/

Installing CSCOPE*: http://cscope.sourceforge.net

Performance optimization: http://www.agner.org/optimize/instruction_tables.pdf

Using Intel VTune Amplifier with a virtual machine: https://software.intel.com/en-us/node/638180

Additional Tools

The previous module helped you to understand how VTune Amplifier can help analyze performance of your DPDK application. In this module we describe two other tools that you might find helpful.

Intel® Memory Latency Checker

Memory latency has to do with the time used by an application to fetch data from the processor's cache hierarchy and memory subsystem. Intel® Memory Latency Checker (Intel® MLC) measures memory latency and bandwidth under load, with options for more detailed analysis of memory latency between a set of cores to memory or cache.

Features

By default, Intel MLC identifies system topology and generates the following:

- A matrix of idle memory latencies for requests originating from each of the sockets and addressed to each of the available sockets.
- Peak memory bandwidth measurement of requests containing varying numbers of reads and writes to local memory.
- A matrix of memory bandwidth values for requests originating from each of the sockets and addressed to each of the available sockets.
- Latencies at different bandwidth points.
- Cache to cache data transfer latencies.

For more information on basic operation of Intel MLC as well as coverage of the command options that enable finer-grained analysis, read the article Intel Memory Latency Checker v3.5. It describes the functionality of the most recent version of Intel MLC in detail, and includes download and installation instructions.

Screenshots

The screenshots below illustrate basic operation of Intel MLC.

```
Measuring Loaded Latencies for the system
Using all the threads from each core if Hyper-threading is enabled
```

```
Intel(R) Memory Latency Checker - v3.3
Measuring idle latencies (in ns)...
        Memory node
Socket       0       1
     0    81.5    81.5
     1   140.2   140.2

Measuring Peak Memory Bandwidths for the system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :      93190.9
3:1 Reads-Writes :      78829.3
2:1 Reads-Writes :      74731.7
1:1 Reads-Writes :      54653.2
Stream-triad like:      68780.8

Measuring Memory Bandwidths between nodes within system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
        Memory node
 Socket      0       1
     0  94281.2 92246.0
     1  34475.4 34470.0
```

```
Using Read-only traffic type
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 00000  292.12   90718.1
 00002  289.80   91195.9
 00008  288.13   91219.6
 00015  284.77   91281.0
 00050  272.97   91269.4
 00100  240.02   91137.1
 00200  146.29   82683.2
 00300  119.82   69158.2
 00400  105.74   55501.1
 00500   98.42   45506.9
 00700   93.26   33390.3
 01000   90.15   23946.1
 01300   87.74   18745.1
 01700   85.99   14606.2
 02500   84.24   10250.6
 03500   82.74    7592.0
 05000   81.89    5566.5
 09000   81.54    3447.9
 20000   81.54    1987.6
```

```
Measuring cache-to-cache transfer latency (in ns)...
Local Socket L2->L2 HIT  latency        53.0
Local Socket L2->L2 HITM latency        53.1
Remote Socket L2->L2 HITM latency (data address homed in writer socket)
                Reader Socket
Writer Socket        0       1
            0        -   114.5
            1    184.8       -
Remote Socket L2->L2 HITM latency (data address homed in reader socket)
                Reader Socket
Writer Socket        0       1
            0        -   114.5
            1    184.0       -
```

Local memory latencies and cross-socket memory latencies can vary significantly on multisocket systems where NUMA is enabled. Intel MLC is a useful tool for measuring these latencies, as well as memory bandwidth, and can help you in the task of profiling your application's performance.

Processor Counter Monitor* (PCM)

Processor Counter Monitor* (PCM) is an open source project that includes a programming API as well as several command-line utilities for gathering real-time performance and power metrics for Intel® Core™ processors, Intel Xeon processors, Intel Atom processors, and Intel® Xeon Phi™ processors. It supports Linux, Windows, and several other operating systems. For detailed information, and to download, visit the PCM GitHub* repository.

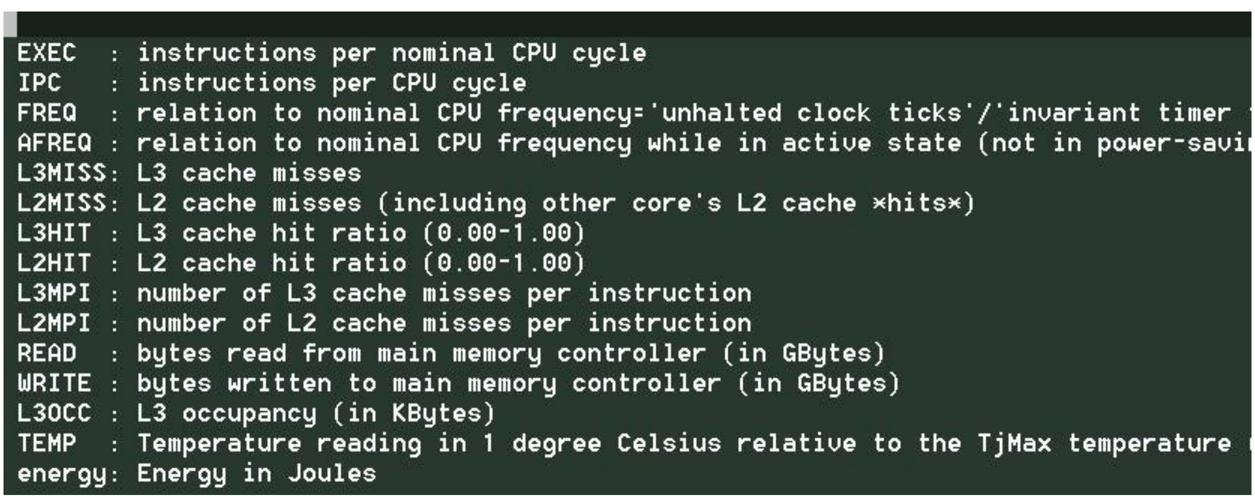
Using PCM to Evaluate a DPDK Application

Of the several tools included as part of PCM, which are recommended for use with DPDK? The list below offers some suggestions. If your application is:

- CPU intensive, run PCM-x
- Memory intensive, run PCM-memory
- I/O intensive, run PCM-iio
-

Screenshots

The screenshots below illustrate PCM runtime output.



```
EXEC  : instructions per nominal CPU cycle
IPC   : instructions per CPU cycle
FREQ  : relation to nominal CPU frequency='unhalted clock ticks'/'invariant timer
AFREQ : relation to nominal CPU frequency while in active state (not in power-savi
L3MISS: L3 cache misses
L2MISS: L2 cache misses (including other core's L2 cache *hits*)
L3HIT : L3 cache hit ratio (0.00-1.00)
L2HIT : L2 cache hit ratio (0.00-1.00)
L3MPI : number of L3 cache misses per instruction
L2MPI : number of L2 cache misses per instruction
READ  : bytes read from main memory controller (in GBytes)
WRITE : bytes written to main memory controller (in GBytes)
L3OCC : L3 occupancy (in KBytes)
TEMP  : Temperature reading in 1 degree Celsius relative to the TjMax temperature
energy: Energy in Joules
```

```
Core (SKT) | EXEC | IPC  | FREQ  | AFREQ | L3MISS | L2MISS | L3HIT | L2HIT | L3MPI | L2MPI |  L3OCC | TEMP

  0    0     0.00   0.82   0.00    1.00      35 K    113 K    0.64    0.47    0.00    0.01      144    57
  1    0     0.00   1.91   0.00    1.00     968     2796      0.45    0.38    0.00    0.00      144    56
  2    0     1.15   1.16   0.99    1.00     600 K   7301 K    0.81    0.65    0.00    0.00     1224    54
  3    0     1.16   1.16   1.00    1.00     577 K   7264 K    0.81    0.65    0.00    0.00     1656    52
  4    0     0.01   0.99   0.01    1.00      20 K     46 K    0.47    0.68    0.00    0.00      144    55
  5    0     1.13   1.14   0.99    1.00     597 K   7344 K    0.81    0.64    0.00    0.00     1224    54
  6    0     1.14   1.15   0.99    1.00     600 K   7469 K    0.81    0.64    0.00    0.00     1368    52
  7    0     1.16   1.16   0.99    1.00     576 K   7207 K    0.81    0.65    0.00    0.00     1296    51
  8    0     1.16   1.17   0.99    1.00     573 K   6983 K    0.81    0.66    0.00    0.00     1440    51
  9    0     1.15   1.15   0.99    1.00     587 K   6959 K    0.81    0.66    0.00    0.00     1296    54
 10    0     1.11   1.11   1.00    1.00     824 K   5965 K    0.72    0.65    0.00    0.00     2016    53
 11    0     1.17   1.17   1.00    1.00     571 K   6930 K    0.81    0.66    0.00    0.00     2016    52
 12    0     1.15   1.15   1.00    1.00     578 K   7274 K    0.81    0.65    0.00    0.00     1224    51
 13    0     1.15   1.16   0.99    1.00     587 K   6922 K    0.81    0.67    0.00    0.00     1944    52
 14    0     0.00   0.65   0.01    1.00      11 K     36 K    0.55    0.65    0.00    0.00      216    56
 15    0     1.17   1.18   1.00    1.00     571 K   7060 K    0.82    0.65    0.00    0.00     1224    52
```

```
 35   1     0.00   0.37   0.00    1.00     697     3008     0.73    0.38    0.00    0.01     2880    65
-----------------------------------------------------------------------------------------------------------
 SKT   0     0.77   1.15   0.67    1.00    7415 K    85 M    0.80    0.65    0.00    0.00    19224    51
 SKT   1     0.00   0.53   0.00    1.00    7050      38 K    0.74    0.45    0.00    0.00     8280    63
-----------------------------------------------------------------------------------------------------------
 TOTAL  *     0.38   1.15   0.33    1.00    7422 K    85 M    0.80    0.65    0.00    0.00     N/A      N/A

 Instructions retired:   37 G ; Active cycles:   32 G ; Time (TSC): 2694 Mticks ; C0 (active,non-halted) core residency: 33.28 %

 C1 core residency: 66.72 %; C6 core residency: 0.00 %;
 C2 package residency: 0.00 %; C6 package residency: 0.00 %;

 PHYSICAL CORE IPC                 : 1.15 => corresponds to 28.84 % utilization for cores in active state
 Instructions per nominal CPU cycle: 0.38 => corresponds to 9.60 % core utilization over time interval
 SMI count: 0

Intel(r) UPI data traffic estimation in bytes (data traffic coming to CPU/socket through UPI links):

               UPI0     UPI1     UPI2    |  UPI0   UPI1   UPI2
-----------------------------------------------------------------------------------------------------------
 SKT    0      646 K    738 K      0     |    0%     0%     0%
 SKT    1      660 K    586 K      0     |    0%     0%     0%
-----------------------------------------------------------------------------------------------------------
Total UPI incoming data traffic: 2632 K     UPI data traffic/Memory controller traffic: 0.00

Intel(r) UPI traffic estimation in bytes (data and non-data traffic outgoing from CPU/socket through UPI links):

               UPI0     UPI1     UPI2    |  UPI0   UPI1   UPI2
-----------------------------------------------------------------------------------------------------------
 SKT    0      305 M    303 M      0     |    1%     1%     0%
 SKT    1      304 M    304 M      0     |    1%     1%     0%
-----------------------------------------------------------------------------------------------------------
Total UPI outgoing data and non-data traffic: 1218 M
MEM (GB)->|  READ |  WRITE | CPU energy | DIMM energy
-----------------------------------------------------------------------------------------------------------
 SKT   0     3.68     2.54     109.80      28.02
 SKT   1     0.00     0.00      67.91       0.00
-----------------------------------------------------------------------------------------------------------
       *     3.68     2.54     177.70      28.02
```

Summary

Intel MLC and PCM are handy, easy to use tools that you might find useful. VTune Amplifier is much more powerful and versatile. If you haven't used VTune Amplifier, download a free trial copy at the Intel VTune Amplifier home page.

Acknowledgements

This cookbook is possible only with the whole team's effort and all the encouragement, support, and review from each and every one in the internal divisions as well as early access customers, network developers, and managers.