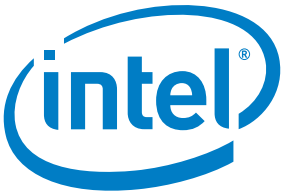

Intel® Manycore Platform Software Stack (Intel® MPSS)

User's Guide for Linux*

May 2017

US

Revision: 4.4.1

World Wide Web: http://www.intel.com

Disclaimer and Legal Information

Revision History

Revision Number	Description	Revision Date
4.4.1	Document update for the release of the Intel® MPSS 4.4.1.	May 2017
4.4.0	Document update for the release of the Intel® MPSS 4.4.0.	April 2017
4.3.3	Document update for the release of the Intel® MPSS 4.3.3. Added information about the UEFI Secure Boot feature.	February 2017
4.3.2	Document update for the release of the Intel® MPSS 4.3.2. Updated information in Appendix B.3 and B.4. Updated supported OS list. Added information on using DSA keys for authorization. Updated information about *systools.*	January 2017
4.3.1	Document update for the release of the Intel® MPSS 4.3.1. Added subsection E.5 on how to enable core-dumps.	November 2016
4.3.0	Document update for the release of the Intel® MPSS 4.3.0. Updated Appendix B, Appendix C contains instructions on compiling various components of the Intel® MPSS, updates in Section 7.	October 2016
4.2.2	Document update for the release of the Intel® MPSS 4.2.2.	September 2016
4.2.1	Document update for the release of the Intel® MPSS 4.2.1. New chapter was added with description of system tools, which are part of the software stack. Added new subsection in Appendix D with instructions on rebuilding the system tools from source. Updated document structure.	August 2016
4.2.0	Document update for the release of the Intel® MPSS 4.2.0. The following information was updated/changed: network configuration instructions, configuration parameters, configuration options, file systems handling, virtual block devices handling, installation procedure, OFED installation, supported OS list, updating the software stack. Added information on controlling the number of pinned memory pages.	June 2016
4.1.3	Document update for the Intel® MPSS 4.1.3 release.	May 2016
4.1.2	Document update for the Intel® MPSS 4.1.2 release.	April 2016
4.1.1	Major revision of the User's Guide. Deprecated sections were removed, updated installation and configuration instructions.	February 2016
4.0.1	Initial draft of the new Intel® Manycore Platform Software Stack (Intel® MPSS) User's Guide	October 2015

Table of Contents

List of Figures

List of Tables

§

1 About this manual

This manual is intended to provide you with a firm understanding of the Intel® Manycore Platform Software Stack (Intel® MPSS), what it is, how to configure it, and how to use its components.

It is recommended to review at least Sections 1-3 prior to a first installation of Intel® MPSS.

Note: This document and instructions presented herein pertain only to systems containing at least one Intel® Xeon Phi™ coprocessor x200. Configurations with both current generation Intel® Xeon Phi™ coprocessors x200 and previous generation coprocessors (Intel® Xeon Phi™ coprocessor x100) are not supported.

1.1 Overview of this document

Section 2 provides a high level overview of the Intel® Xeon Phi™ coprocessor x200 design and Intel® MPSS architecture.

Section 3 is a step-by-step guide to installing Intel® MPSS, including basic configuration steps and considerations for both workstation and cluster environments.

Section 4 shows how to configure the coprocessor OS.

Section 5 describes supported network configurations, explains when each might be used, and shows how to configure them.

Section 6 presents methods for managing the coprocessor's file systems and shows how to add supplementary software to the coprocessor OS.

Section 7 presents configuration options for Intel® MPSS components, including the coprocessor's Linux* kernel, and the COI offload interface.

Section 8 lists and describes each application comprising the *System tools*.

Appendix A describes the Intel® MPSS-specific configuration parameters.

Appendix B presents sysfs entries exposed by the Intel® MPSS host driver.

Appendix C provides instructions on installing optional Intel® MPSS components

Appendix D shows how to compile various components of the software stack from source, and how to build custom software for use on the coprocessor.

Appendix E presents tools and techniques that can be used in troubleshooting and debugging issues with the coprocessor.

1.2 Notational conventions

This document uses the following notational conventions.

micctrl --config	Commands and their arguments in prose sections are *italicized*.
RootDevice is *PFS*	Configuration parameter names are *italicized* when they appear in prose sections.
/etc/mpss/default.conf	Files and directories in prose sections are *italicized.*
micN	Indicates any coprocessor name – mic0, mic1, mic2 etc. where N=0, 1, 2 … 255 (e. g. file *micN.conf* is any of the files *mic0.conf, mic1.conf* etc.).
`COURIER text`	Code and commands entered by the user. A backslash symbol: \ indicates that command is continued in the next line.
`Italic COURIER text`	Terminal output by the computer.
`[host]$`	Command entered on the host with user or root privileges.
`[host]#`	Command entered on the host with root privileges.
`[micN]$`	Command entered on a coprocessor with user or root privileges.
`[micN]#`	Command entered on a coprocessor with root privileges.

For example, the snippet below shows the *micctrl --config* command executed as a non-root user, and the truncated output it generated:

```
[host]$ micctrl --config
mic0:
============================================================

   Linux Kernel:   /usr/share/mpss/boot/bzImage-knl-lb
```

1.2.1 Command syntax

The following syntax conventions are used in *micctrl* commands and Intel® MPSS configuration parameters:

`<…>` indicates a variable value to be supplied.

`[…]` indicates an optional component.

`(x|y|…|z)` indicates a choice of values.

The syntax of the *AutoBoot* configuration parameter indicates that the user can choose either *Enabled* or *Disabled* option:

```
AutoBoot (Enabled|Disabled)
```

The syntax of the *micctrl --initdefaults* command indicates that *--initdefaults* might be invoked with several sub-options: *--keep-config*, *--keep-image*, and *--force*.

```
micctrl --initdefaults [--keep-config] [--keep-mage] \
[(-f|--force)]
```

1.3 Terminology

ABI	Application Binary Interface
CCL	Coprocessor Communication Link
COI	Coprocessor Offload Infrastructure
DHCP	Dynamic Host Configuration Protocol
GDB	GNU debugger
HCA	Host Channel Adapter
IPoIB	Internet Protocol over InfiniBand*
K1OM	Architecture of the Intel® Xeon Phi™ coprocessor x100
KMP	Kernel Module Package
KNL	Knight's Landing – codename of the Intel® Xeon Phi™ coprocessor x200
LDAP	Lightweight Directory Access Protocol
Lustre*	A parallel, distributed file system
MAC	Media Access Control
MIC	Intel® Many Integrated Core Architecture
MPI	Message Passing Interface
MPSS	Intel® Manycore Platform Software Stack
MYO	Mine, Yours, Ours - shared memory infrastructure
NIS	Network Information System

NUMA	Non-Uniform Memory Access
OFED	Open Fabric Enterprise Distribution
PCH	Platform Controller Hub
PCIe3	PCI Express Gen 3.0
PFS	Persistent File System
QPI	Intel® QuickPath Interconnect
RHEL*	Red Hat* Enterprise Linux*
RPM	RPM Package Manager
SCIF	Symmetric Communication Interface
SLES*	SUSE* Linux* Enterprise Server
SMP	Symmetric Multi-Processor
SSD	Solid State Drive
SSH	Secure Shell
Sysfs	A virtual file system
VEth	Virtual Ethernet
X86_64	Architecture of the Intel® Xeon Phi™ coprocessor x200

§

2 Intel® MPSS at a glance

This section provides an overview of the Intel® Manycore Platform Software Stack. It contains a high level description of the Intel® Xeon Phi™ coprocessor x200 hardware and system architecture. The section also discusses the programming models that Intel® MPSS is designed to support, and how the various components support those programming models. It concludes with a listing of other available documentation.

2.1 Coprocessor hardware and system architecture

The Intel® Xeon Phi™ coprocessor x200 PCIe add-in card is designed for installation into an Intel® Xeon® processor-based platform. Figure 1 shows a typical configuration.

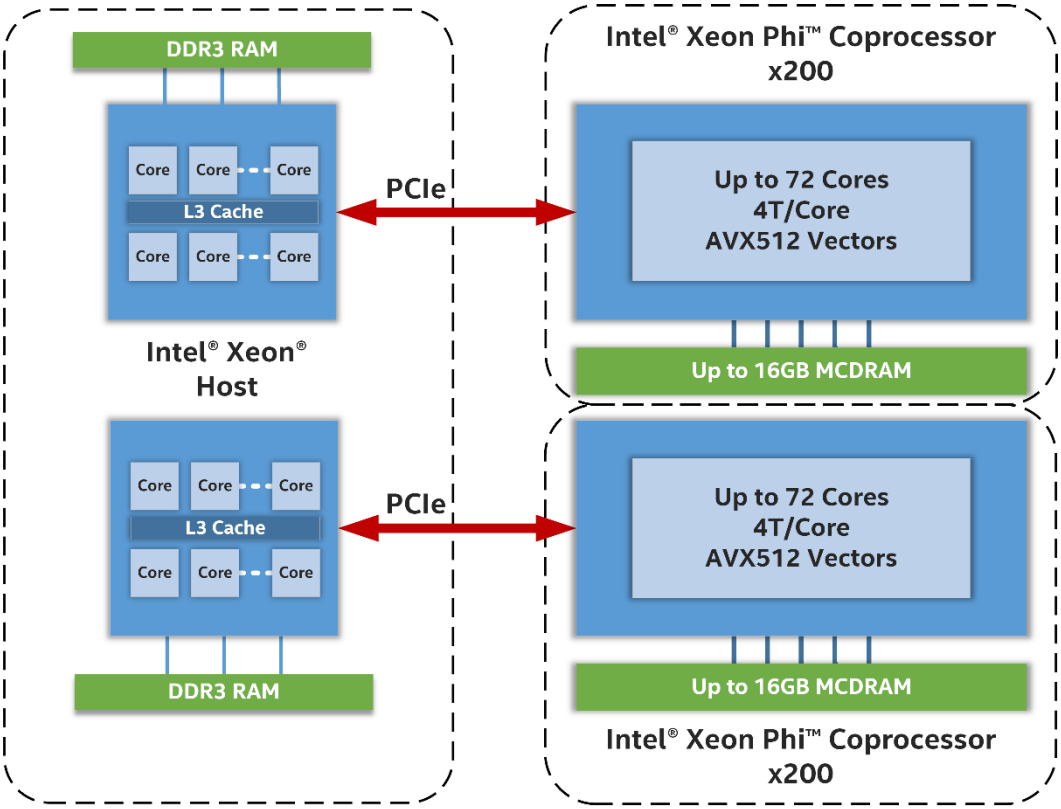

Figure 1 Typical workstation configuration

The coprocessor contains up to 72 cores, each core supporting 4 hardware (HW) threads. It also supports standard AVX 512 vector instructions. Instead of traditional DDR the coprocessor is equipped with up to 16GB of on-package, high-bandwidth MCDRAM memory. The coprocessor's PCIe interface works as a Gen3 PCIe root port. A Non-Transparent Bridge (NTB) chip is used to connect the device as PCIe end port to the host.

The coprocessor does not have a permanent file system storage. Instead, the file system is implemented on a virtual block device backed by a file in the host's file system.

Each coprocessor runs a standard Linux* kernel with some minor accommodations for the MIC hardware architecture. Because it runs its own OS, the coprocessor is not hardware cache coherent with the host processors or other PCIe devices.

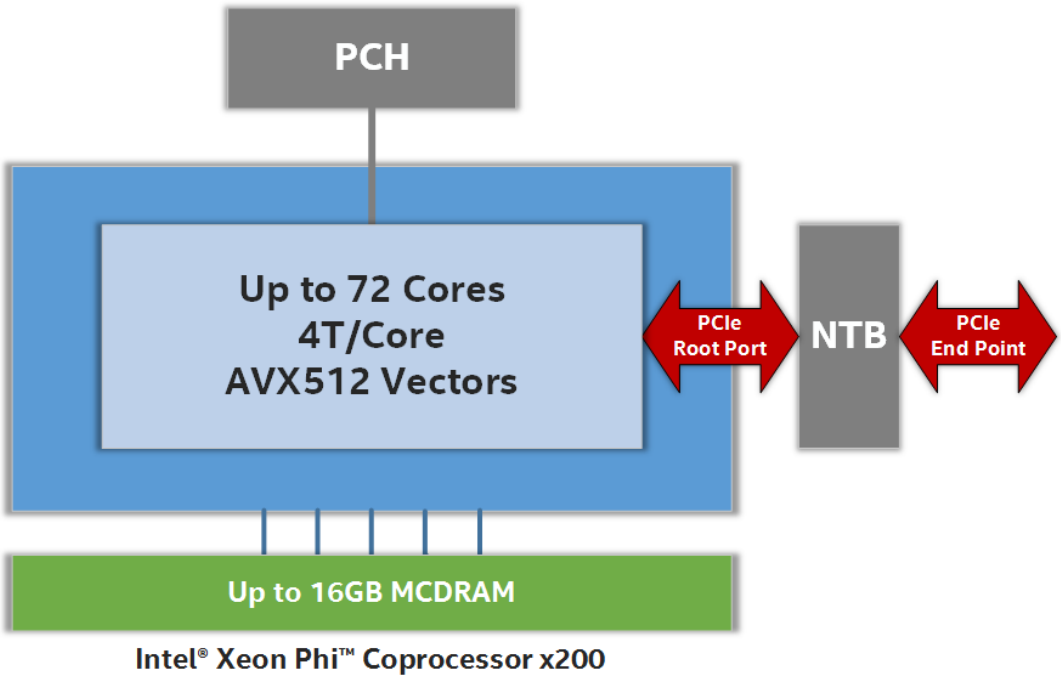

Figure 2 Intel® Xeon Phi™ Coprocessor x200 architecture

2.2 Programming models

The Offload, Symmetric and Native (MIC-hosted) programming models offer a diverse range of usage models. An overview of these options is depicted in Figure 3.

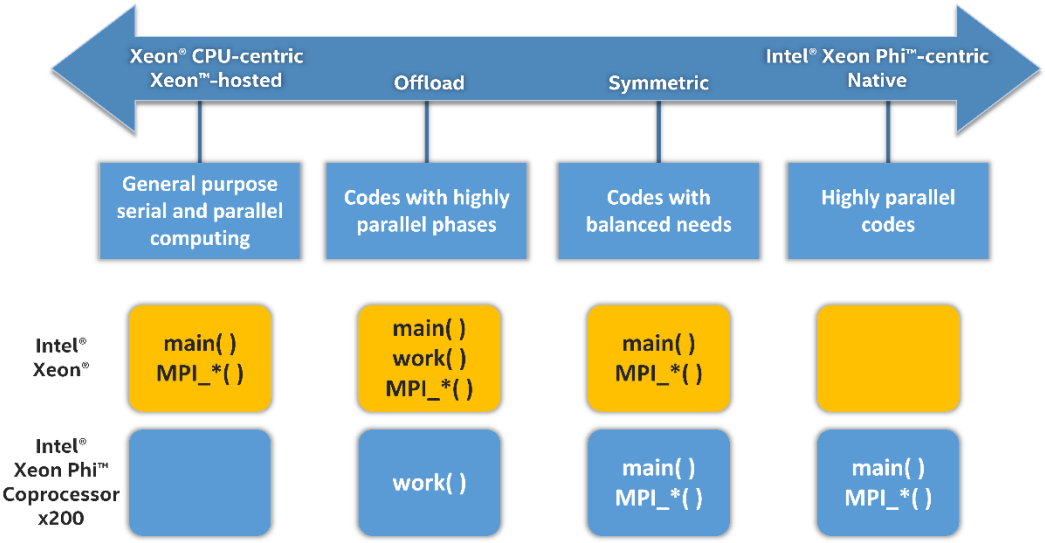

Figure 3 Spectrum of programming models

2.2.1 Offload programming model

In the *Offload* model, one or more processes of an application are launched on one or more host processors. These processes, represented in the figure by *main()*, can offload computation, represented by *work()*, to the attached coprocessors, taking advantage of the many-core architecture with its wide vector units and high memory bandwidth. In case where the application is composed of more than one process, the processes often communicate using some form of message passing, such as *MPI* (Message Passing Interface) thus *MPI_*()* is shown on the host. This offload process is programmed via the use of offload pragmas supported by the Intel® C/C++ and FORTRAN compilers. When an application is created with one of these compilers, offloaded execution will fall back to the host in the event that a coprocessor is not available. This is why an instance of *work()* is shown on the host as well.

2.2.2 Symmetric programming model

The *Symmetric* programming model is convenient for an existing HPC application that is composed of multiple processes, each of which could run on the host or on a coprocessor, and use some standard communication mechanism such as MPI. In this model computation is not offloaded but rather remains within each of the processes comprising the application. In such cases, where the application is MPI based, the OFED distributions enable high bandwidth/low latency communication using installed Intel® True Scale or Mellanox* InfiniBand* Host Communication Adapters.

2.2.3 Native programming model

The Native (coprocessor-hosted) programming model is a variant of the symmetric model in which one or more processes of an application are launched exclusively on one or more coprocessors. From the Intel® MPSS architecture perspective, these programming models typically depend on SCIF and the Virtual Ethernet (VEth) driver to launch processes on coprocessors.

2.3 Intel® MPSS software architecture and components

Figure 4 provides a high level representation of Intel® MPSS and its relation to other important software components. The host software stack is shown to the left and the coprocessor software stack to the right.

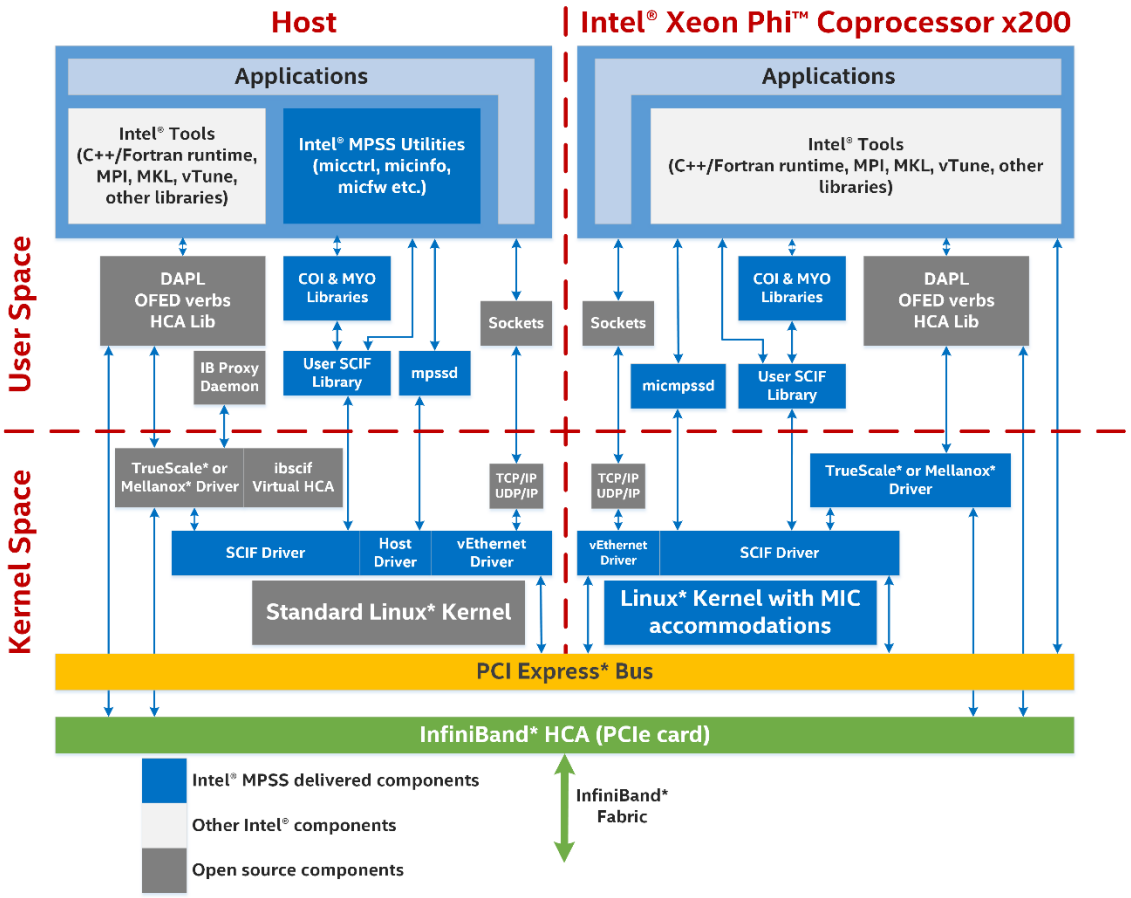

Figure 4 Intel® MPSS architecture

2.3.1 Coprocessor operating system

Underlying all computation on the coprocessor is a standard Linux* kernel with changes in the Power Management (PM) driver and Performance Monitoring Unit (PMU) driver to support the coprocessor. The Linux* kernel and initial file system image are installed into the host file system as part of the software stack. After the installation each coprocessor will need to be configured in accordance with the expected workload/application. This configuration is covered in detail starting with Section 4.

The Linux* environment on the coprocessor utilizes *BusyBox* to provide a number of Linux* utilities. Note, that these utilities may have limited functionality when compared to similar tools provided with the host Linux* distribution. For more information regarding *BusyBox*, refer to http://www.busybox.net/.

2.3.2 Intel® MPSS middleware libraries

The compiler runtimes depend on the COI (Coprocessor Offload Infrastructure) library to offload executables and data for execution on a coprocessor, and use MYO (Mine Your Ours shared memory infrastructure) to provide a virtual shared memory model that simplifies data sharing between processes on the host and coprocessors. Similarly, some functions in the Intel® Math Kernel Library (Intel® MKL) automatically offload work to coprocessors using the COI library.

COI, MYO, and other software stack components rely on the SCIF (Symmetric Communication Interface) user mode API for PCIe communication services between the host processor, coprocessors, and optionally installed InfiniBand* host channel adapters. SCIF delivers very high bandwidth data transfers and sub-μsec write latency to memory shared across PCIe, while abstracting the details of communication over PCIe.

The COI, MYO, and SCIF libraries are also available to other applications. Section 2.4 lists additional documentation on these libraries.

2.3.3 Intel® MPSS modules and daemons

Following kernel modules run on the host computer and enable communication with the coprocessor:

- *mic_x200_dma*
- *scif_bus*
- *vop_bus*
- *cosm_bus*
- *scif*
- *vop*
- *mic_cosm*
- *mic_x200*

Kernel modules are responsible for initializing, booting, and managing the coprocessor. They also implement the host-side SCIF protocol and Virtual Ethernet communication interfaces used between the host and the coprocessor.

The *virtual console driver* redirects the coprocessor OS console to the host. The *virtio block device (virtio-net)* uses the Linux* virtio data transfer mechanism to implement a block device on the coprocessor. The device stores data in a file on the host and therefore is persistent across coprocessor reboots.

The *mpssd* daemon runs on the host, and directs the initialization and booting of the coprocessor. The *mpssd* daemon is started and stopped with the Linux* *mpss* service, and instructs the devices to boot or shutdown. In the event that the coprocessor's OS crashes, *mpssd* reboots it or brings it to a *ready* (to be booted) state.

The Intel® Omni-Path Architecture enables direct data transfers between the coprocessors' memory and Intel® Omni-Path Host Fabric Adapters.

The software stack also includes an optional *ibscif* (InfiniBand* over SCIF) driver which emulates an InfiniBand* HCA to the higher levels of the OFED stack. This driver uses SCIF to provide high bandwidth, low latency communication between multiple devices in a host platform, for example between MPI ranks on separate coprocessors.

2.3.4 Tools and utilities

Intel® MPSS includes several system management tools and utilities.

micctrl is a utility for controlling (boot, shutdown, reset) each coprocessor. It also provides various options to simplify the configuration process. More information on the *micctrl* tool can be found in Section 8.1 or by executing:

```
[host]$ micctrl -h
```

micperf is a selection of benchmarks that allow users to estimate the performance of their coprocessor based system. For more information refer to *miperf* documentation available at */usr/share/doc/micperf-4.4.1*. Appendix C describes how to install micperf.

System tools is a group of applications that aid in managing coprocessors installed in the host system. Those tools can be used to display various information regarding the coprocessor hardware and software, and introduce changes to selected configuration options. *System tools* also include the *libsystools* library, which can be used to provide applications running on the host with a C-library interface to the coprocessors. Section 8 contains detailed information on those applications and shows examples of their usage.

2.3.5 GCC toolchain

Unlike the Intel® Xeon Phi™ coprocessor x100, the Intel® Xeon Phi™ coprocessor x200 is compatible with the standard GCC toolset. As of version 4.9, GCC supports AVX 512 extensions.

2.4 Related documentation

SCIF documentation:

The following SCIF documentation is installed during base Intel® MPSS installation:

$MPSS4/doc/scif_user_guide.pdf

/usr/share/doc/scif/tutorials	SCIF tutorial source files and a *README.txt* file, which contains instructions on building and running the tutorials.

COI Documentation:

The following COI documentation is installed during base Intel® MPSS installation

/usr/share/doc/packages/intel-coi-4.4.1/	Release notes (*release_notes.txt*), API Reference Manual (*MIC_COI_API_Reference_Manual_1_0.pdf*), and a quick reference guide (*coi_getting_started.pdf*).
/usr/include/intel-coi/	Header files containing full API descriptions
/usr/share/doc/packages/intel-coi-4.4.1/tutorials/	Full tutorials source and makefiles.
/usr/share/man/man3	COI man pages.

MYO Documentation:

The following MYO documentation is installed during base Intel® MPSS installation:

/usr/share/man/man3	MYO man pages.
/usr/share/doc/myo	MYO tutorials and other documents.

§

3 Installation process overview

This section describes installation and basic configuration of the coprocessor hardware and software. Read this section before proceeding with installation to ensure that all hardware and software requirements are satisfied. It is important to follow these steps in their presented order.

Root privileges are required to install and configure the software stack.

3.1 Hardware and software prerequisites

Host System Hardware:

Each coprocessor requires a 64-bit 75-Watt PCIe slot with x16 electrical connections. Refer to your motherboard's manual to identify compatible slots.

Note: Installing more than 8 coprocessors in a single host platform is not supported.

Note: Host platforms with 386GB or more RAM are supported provided IOMMU (input–output memory management unit) is enabled in the kernel command line (*intel_iommu=on*) and in BIOS (only applicable for RHEL* 7.3 and SLES* 12 SP2). Without IOMMU the host memory has to be limited to *386*GB with the *mem=386G* kernel command line parameter.

Physical Installation:

If the coprocessor is to communicate with another PCIe device, such as an Intel® Omni-Path Architecture Host Fabric Adapter or another coprocessor, consideration should be given to which host platform slots the devices are installed. When multiple PCIe devices connect to slots on separate CPU IO hubs, they will communicate across the Quick Path Interconnect (QPI). Device to device (P2P) communication bandwidth over QPI is lower than communication speeds within the same IO hub.

3.1.1 Bios configuration

1. Update your host platform's BIOS to the latest version and reset the settings to their default values.
2. Enable Large Base Address Registers (BAR) support in the host platform BIOS.

   BIOS and OS support for large (32+GB) Memory Mapped I/O Base Address Registers (MMIO BARs) above the 4GB address limit *must* be enabled.
3. Modify the following BIOS settings to ensure maximum and stable performance:

   a. If available on your platform, set the Extended ATR setting to 0x1.

   b. Enable Intel® Turbo Boost Technology.
4. Disable Virtualization Technology for Directed I/O (VT-d). This step only applies to hosts with operating systems older than RHEL* 7.3 and SLES 12 SP2.

3.1.2 Supported host operating systems

Intel® MPSS 4.4.1 was validated against specific versions of Red Hat* Enterprise Linux* (RHEL*) and SUSE* Linux* Enterprise Server (SLES*) as the host operating system. Table 1 lists the supported versions of these operating systems.

Table 1 Supported host operating systems

Supported Host OS Versions	Kernel Version
Red Hat* Enterprise Linux* 7.2 (x86_64)	3.10.0-327.el7.x86_64
Red Hat* Enterprise Linux* 7.3 (x86_64)	3.10.0-514.el7.x86_64
SUSE* Linux* Enterprise Server 12 (x86_64)	3.12.28-4-default
SUSE* Linux* Enterprise Server 12 SP1 (x86_64)	3.12.49-11-default
SUSE* Linux* Enterprise Server 12 SP2 (x86_64)	4.4.21-69-default

In addition to the kernel versions listed in the table above, the driver modules should be compatible with all kernel security updates provided by the OS vendors. The modules come in form of Kernel Module Package (KMP) RPMs and will be automatically installed for each kernel security update, provided the kernel ABI does not change.

If incompatibilities are encountered, rebuild the driver modules from the *mpss-modules* source RPMs (refer to Appendix D.3 for instructions).

3.1.2.1 SLES* host OS configuration

1. Configure the SUSE* Linux* Enterprise Server kernel to allow loading non-SUSE* driver modules. Edit the */etc/modprobe.d/10-unsupported-modules.conf* file and set the value of *allow_unsupported_modules* to 1.
2. Make sure the *wicked* service is active:

   ```
   [host]# systemctl status network.service
   ```

   If the output does not show *wicked* execute the following commands:

   ```
   [host]# systemctl stop network.service
   [host]# systemctl enable --force wicked.service
   ```

3. Enable the *nanny* daemon by creating and/or editing the */etc/wicked/local.xml* file so that it contains the following lines:

   ```
   <config>
      <use-nanny>true</use-nanny>
   </config>
   ```

4. Execute the following commands to introduce the changes and flush the DNS cache:

   ```
   [host]# systemctl restart wicked
   [host]# systemctl restart nscd
   ```

3.1.3 UEFI Secure Boot

Intel® MPSS supports the UEFI Secure Boot feature. Driver modules included in the software stack are signed with a private key which allows them to be authenticated by the corresponding public key. The public key is distributed in the *mpss-4.4.1-<OS version>.tar* archive, which contains the software stack release. Once the release package is extracted the key is available at *$MPSS4/mpss-public-key.der*.

Note: *$MPSS4* indicates an absolute path to the extracted release package.

Please refer to the host OS documentation for details on how to use the MOK facility to add the public key to the UEFI Secure Boot key database.

3.1.4 Workstation considerations

A typical workstation has no InfiniBand* HCAs. In this case, the programming model (Section 2.2) and requirements for P2P communication between coprocessors will determine how they are installed. If a single coprocessor is used, it can utilize any compatible slot.

3.1.4.1 Offload programming model

Most workstation applications offload work to one or more coprocessors using the offload programming model provided by the Intel® C/C++ and FORTRAN compilers, and the Intel® MKL libraries. In this framework, coprocessors only communicate with the host processor(s) rendering the P2P bandwidth irrelevant. For best performance, install devices uniformly across the I/O hubs as presented in Figure 5. For maximum bandwidth, run the offload program's host-side on the same NUMA node as the coprocessor it is offloading work to.

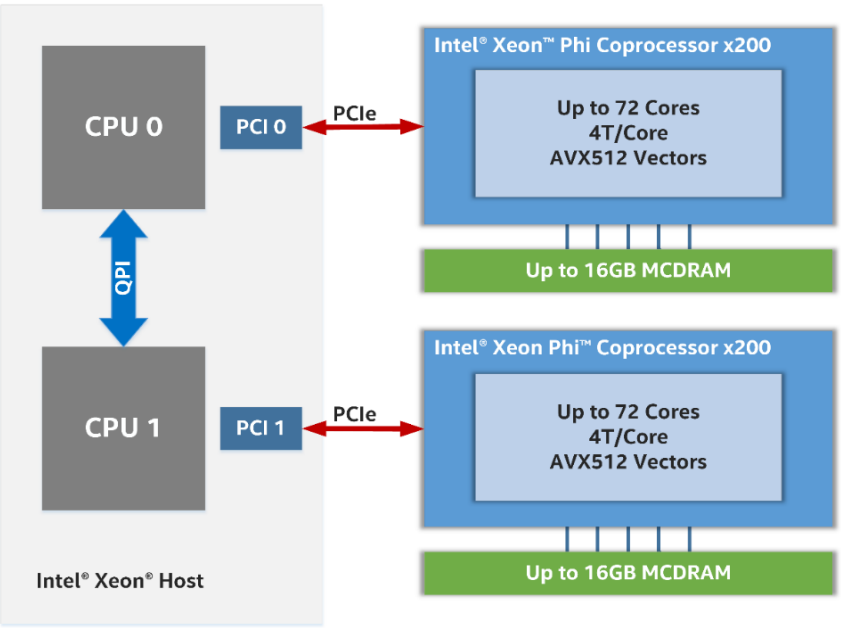

Figure 5 Uniform coprocessor distribution

3.1.4.2 Symmetric and native programming models

Communication bandwidth between coprocessors is important when an application is distributed across them. For best performance in this case, install pairs of coprocessors on each I/O hub as shown in Figure 6.

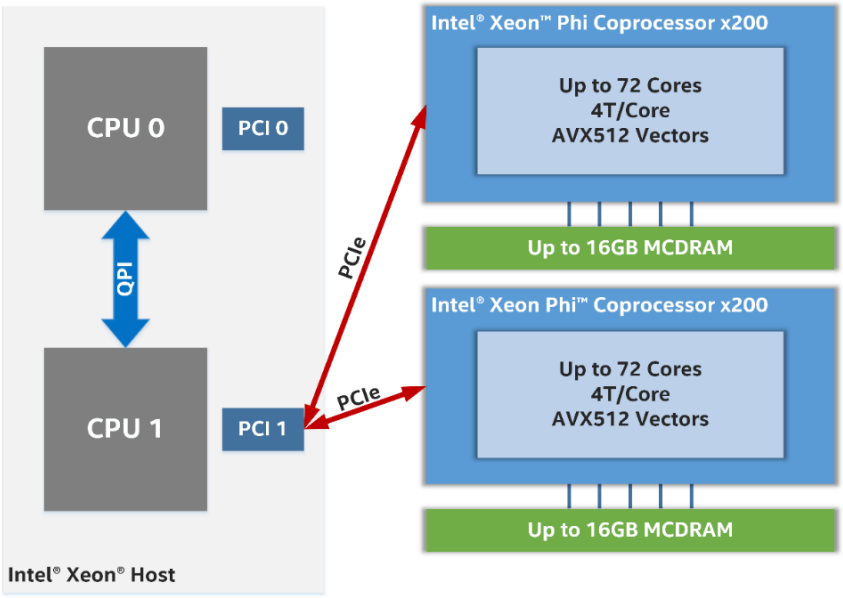

Figure 6 Two coprocessors installed in the same I/O Hub

3.1.5 Cluster considerations

When a coprocessor and an InfiniBand* HCA are installed in a platform, it is vital to maximize communication bandwidth between them. This can be accomplished by installing them on the same I/O hub. By extension, when there are multiple sets of coprocessors and HCAs, install a coprocessor and an HCA pair on each I/O hub, as shown in Figure 7. For other ratios of coprocessors and HCAs, consideration should be given to how the devices are expected to inter-communicate, with regard to the relative communication bandwidths between the various PCIe slots in a platform.

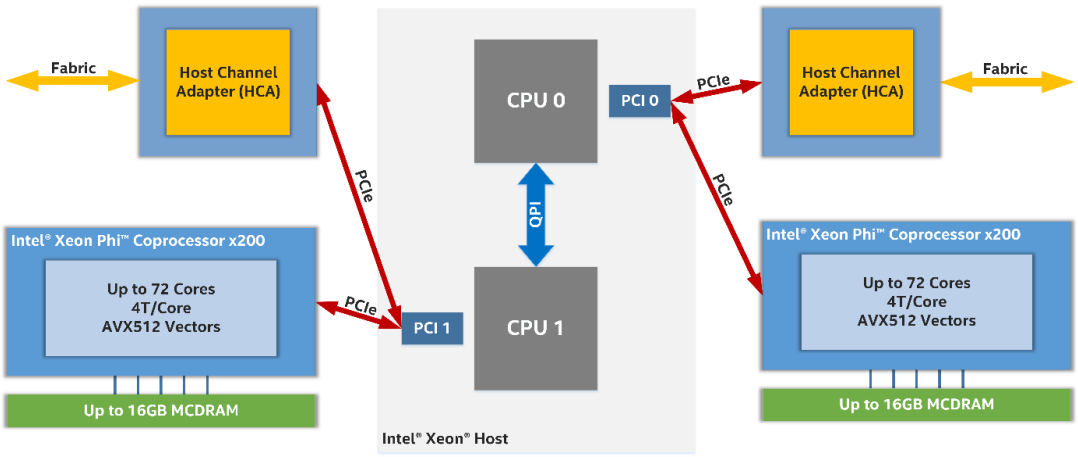

Figure 7 Symmetric distribution of coprocessors and HCAs

3.1.6 Validate physical installation of coprocessors

Before installing and using Intel® MPSS, check if the host OS is able to enumerate and assign MMIO resources to coprocessors. The *lspci* command can be used for that purpose. The example below presents its output when run on a system containing one coprocessor.

```
[host]# lspci | grep -i Co-processor
03:00.0 Co-processor: Intel Corporation Device 2260 (rev ba)
```

To verify that the BIOS/OS is able to assign all required resources to the coprocessors, execute the command below. Note the name of the slot reported by the previous command (in this example it is *bus:slot 03:00*).

```
[host]# lspci -vs 03:00.0
03:00.0 Co-processor: Intel Corporation Device 2260 (rev ba)
Subsystem: Intel Corporation Device 0244
Physical Slot: 1
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop-
ParErr+ Stepping- SERR+ FastB2B- DisINTx+
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort-
<TAbort- <MAbort- >SERR- <PERR- INTx-
Latency: 0, Cache Line Size: 64 bytes
Interrupt: pin A routed to IRQ 147
Region 0: Memory at d0d00000 (32-bit, non-prefetchable)
[size=256K]
Region 2: Memory at 380000000000 (64-bit, prefetchable)
[size=32G]
: <output truncated>
```

The correct output indicates that both BAR0 (region 0) and BAR1 (region 2) have valid resources assigned. The output must indicate that 256KB is assigned to BAR0 and 32GB to BAR1.

If the expected number of coprocessors is not reported, it may help to reseat them before continuing. If the devices are detected, but no resources have been assigned, it is best to check the system BIOS support for large BARs (see Section 3.1.1).

3.2 Installing Intel® MPSS

Configure a local repository:

Intel® MPSS is installed using a local repository, allowing users to choose which packages they wish to install. It also allows for installing additional packages after the basic installation is complete.

RHEL* and SLES* provide mechanisms to create a local *yum* or *zypper* repository in the host file system. This requires generating repository metadata for the Intel® MPSS package.

1. Install the *createrepo* tool.

   In RHEL*:

   ```
   [host]# yum install createrepo
   ```

   In SLES*:

   ```
   [host]# zypper install createrepo
   ```

2. Download and extract the Intel® MPSS release package (*mpss-4.4.1-<OS version>.tar*) to the *$MPSS4* directory and initiate a local repository.

Note: *$MPSS4* indicates an absolute path to the extracted release package.

```
[host]$ tar -xvf mpss-4.4.1-<OS version>.tar
[host]# createrepo $MPSS4/packages
```

3. Add the repository to your host OS.

   In RHEL*:

   Create a *.repo* file in the */etc/yum.repos.d/* directory.

   ```
   [host]# vi /etc/yum.repos.d/mpss-4.repo
   ```

   This file should contain the following lines:

   ```
   [mpss-4]
   name=Intel(R) MPSS
   enabled=1
   baseurl=file://$MPSS4/packages
   gpgcheck=0
   ```

   In SLES*:

   Use *zypper* to create the *.repo* file.

   ```
   [host]# zypper ar -n 'Intel(R) MPSS' -G $MPSS4/packages \
   mpss-4
   ```

```
[host]# zypper refresh
```

4. Verify whether the repository is functioning correctly.

   In RHEL*:

   ```
   [host]# yum repoinfo
   ```

   In SLES*:

   ```
   [host]# zypper repos
   ```

   The output should show the *mpss-4* repository and its associated information. You can view the available packages by running the commands below.

   In RHEL*:

   ```
   [host]# yum --disablerepo="*" --enablerepo="mpss-4" \
   list available
   ```

   In SLES*:

   ```
   [host]# zypper search -r mpss-4
   ```

Install Intel® MPSS and additional packages:

1. Install all core components of the software stack.

   In RHEL*:

   ```
   [host]# yum install $MPSS4/packages/x86_64/core/*.rpm
   ```

   In SLES*:

   ```
   [host]# zypper install $MPSS4/packages/x86_64/core/*.rpm
   ```

   You can install a minimal set of the software stack components by installing just the *mpss-daemon* package (not recommended).

2. Optionally, additional packages can be installed by running the following commands.

   In RHEL*:

   ```
   [host]# yum install <package_name>
   ```

   In SLES*:

   ```
   [host]# zypper install <package_name>
   ```

   For example, to install the *mpss-sdk*:

   In RHEL*:

   ```
   [host]# yum install mpss-sdk
   ```

   In SLES*:

   ```
   [host]# zypper install mpss-sdk
   ```

3.2.1 Updating Intel® MPSS

The software stack can be updated to the version immediately following the currently installed release. This rule applies to both major and minor version updates. For example, Intel® MPSS 4.X.Y may be updated to version 4.X.(Y+1) or 4.(X+1).Z. Updating version 4.X.Y to 4.X.(Y+2) or 4.(X+2).Z is not supported.

Follow the instructions below to upgrade the software stack.

1. Stop the *mpss* service and unload the driver modules.

```
[host]# systemctl stop mpss
[host]# micctrl --unload-modules
```

2. Download the new release tarball, extract it to *$MPSS4*, and update the software stack and optional packages.

   In RHEL*:

```
[host]# yum update $MPSS4/packages/x86_64/core/*.rpm
[host]# yum update <package_name>
```

   In SLES*:

```
[host]# zypper update $MPSS4/packages/x86_64/core/*.rpm
[host]# zypper update <package_name>
```

3. Load the driver modules and update the coprocessor's default configuration.

Note: Backup your PFS images before executing the command in this step.

```
[host]# micctrl --load-modules
[host]# micctrl --initdefaults --keep-config
```

4. Start the *mpss* service.

```
[host]# systemctl start mpss
```

3.2.2 Uninstalling Intel® MPSS

Uninstall any Intel® MPSS component by issuing the commands below.

In RHEL*:

```
[host]# yum remove <mpss-component>
```

To uninstall all non-optional Intel® MPSS components enter:

```
[host]# yum remove mpss-release
```

In SLES*:

```
[host]# zypper remove <mpss-component>
```

To uninstall all non-optional Intel® MPSS components:

```
[host]# zypper remove mpss-release
```

Optional components must be uninstalled manually. To verify what packages are currently installed on your host system run the following command:

```
[host]$ rpm -qa --queryformat='%{distribution} %{version} \
%{name} \n' | grep MPSS4
```

3.3 Initial configuration

The driver modules must be loaded to initiate the default configuration. Issue the following commands:

Note: Run the first command if the driver modules from previous installation of the software stack are loaded on your host OS.

```
[host]# micctrl --unload-modules
[host]# micctrl --load-modules
```

To initialize the default configuration execute the following command:

```
[host]# micctrl --initdefaults
```

Note: Running *micctrl --initdefaults* may overwrite your persistent file system (PFS) image and/or configuration files.

This command creates Intel® MPSS configuration files and populates them with default values. *micctrl --initdefaults* also updates the configuration and should be run after updating the software stack.

To view a summary of the initial configuration use:

```
[host]$ micctrl --config
```

More details on the configuration tasks are available in Section 4 and Appendix A.

3.4 Updating coprocessor's firmware

It is required to update coprocessor's firmware after each installation or update of the software stack. Current firmware version is distributed with the software stack installation. The *$MPSS4/doc/README* file lists the versions of the included firmware.

Note: Running Intel® MPSS with incorrect firmware version is not supported and may lead to erratic behavior.

Follow the instructions below to update the coprocessor using the *micfw* tool. Note, however, that these instructions will not work if the flash files (i.e. files with the *.hddimg* suffix) were moved or deleted from their installation directory (*/usr/share/mpss/flash*).

1. Check the status of each coprocessor:

   ```
   [host]# micctrl –s
   ```

   If the status of every coprocessor is *ready to boot*, proceed to step 2; otherwise, reset the coprocessors:

   ```
   [host]# micctrl –rw
   ```

2. Verify the version of firmware installed on the coprocessor:

   ```
   [host]# micfw device-version
   ```

   If the versions are the same as versions described in Section 1 of the *$MPSS4/doc/README* file, the next steps might be skipped.

Note: If the installed firmware version is older than the firmware version distributed with the Intel® MPSS 4.3.2, step 3 has to be executed twice in order to complete the SMC firmware update.

3. Update the firmware of each coprocessor:

   ```
   [host]# micfw update all
   ```

   Or update only a specified coprocessor:

   ```
   [host]# micfw update micN
   ```

   Once the update process completes the state of coprocessors will be changed to *ready to boot*.

Note: Do not execute any other applications or modify any coprocessor's state while *micfw* is executing.

Note: *micfw* might fail to update coprocessor's firmware if it hadn't been consequently updated with each subsequent version of the software stack.

4. Perform the cold host system reboot to apply all changes.

3.5 Initial coprocessor boot

After successful completion of the previous steps, the coprocessor(s) can be booted. Enter the following command:

```
[host]# systemctl start mpss
```

Starting the *mpss* service boots all installed coprocessors (default setting).

The command below configures the *mpss* service to start when the host OS boots.

```
[host]# systemctl enable mpss
```

The command below prevents the *mpss* service from starting when the host OS boots.

```
[host]# systemctl disable mpss
```

3.6 Validating Intel® MPSS installation, and first login

Intel® MPSS provides utilities to perform basic tests to validate the installation.

Appendix D provides troubleshooting advice if problems are encountered during software stack installation and usage.

3.6.1 Validating Intel® MPSS installation

The *micinfo* tool provides information about the host and the coprocessor hardware and software. It can be used, for example, to verify that your coprocessor is running the correct firmware version. Note that certain information is only available when executing *micinfo* with root privileges:

```
[host]# micinfo
```

More information can be obtained by running the *micinfo --help* command.

The *miccheck* tool performs sanity checks on systems containing coprocessors.

```
[host]$ miccheck
```

Obtain more information on the utility by running the *miccheck --help* command.

3.6.2 Initial coprocessor login

After the initial login, the coprocessor OS supports network access using SSH keys, password authentication, or host-based authentication.

Note that initial configuration of the coprocessor's OS does not create any user accounts (except root account) or parse any user credentials to the coprocessor's file system. This data can be provided manually as in any Linux* distribution.

Only *root* can perform the first login to the coprocessor. This operation can be completed in two ways:

a) Using ssh with *root's* public RSA key.

b) With a virtual console (for example *minicom*).

3.6.2.1 Using ssh to log in

To successfully login, an RSA key must be present in root's *~/.ssh* directory. During boot, the first public key is propagated to each coprocessor's file system, allowing root to use their private key for authentication.

If there are no keys in root's *~/.ssh* directory, execute the command below to generate a set of keys. During key generation you can provide a file in which the key will be stored (or use the default file *id_rsa*) and choose a passphrase (or save the key without a passphrase).

```
[host]# ssh-keygen
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
<output truncated>
```

Reboot all coprocessors to implement the changes.

```
[host]# micctrl -R
```

Add the hostname and IP address of each installed coprocessor to the */etc/hosts* file. The example below assumes that the IP address of the coprocessor micN is 172.31.N+1.1.

```
[host]# echo "172.31.N+1.1      micN" >> /etc/hosts
```

Note: The coprocessors' IP addresses may differ depending on the network configuration.

Log in to the coprocessor micN using the following command:

```
[host]# ssh micN
```

If the following message appears remove the micN RSA from the user's *known_hosts* file, typically found in the *~/.ssh* directory.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

Alternatively, execute the command below.

```
[host]# ssh-keygen –R micN
```

Note: When running the *ssh* command in the background, it is preferable to use the "*-f*" option instead of appending *"&"* to the command:

```
[host]$ ssh -f micN "sleep 20; echo 'complete'"
```

Note: Add the line below to the */etc/ssh/sshd_config* file on each coprocessor to use DSA keys instead of RSA keys for authorization.

```
[micN]# echo "PubkeyAcceptedKeyTypes=+ssh-dss" >> \
/etc/ssh/sshd_config
```

3.6.2.2 Using a virtual console to log in

Note: Install a terminal emulator prior to following the instructions in this section.

Intel® MPSS software offers a network-independent virtual console connection to each coprocessor over the PCIe interface. The *ttyMICN* MIC virtual console devices are located in the */dev* directory. The example below shows how to use *minicom* to access the coprocessor.

```
[host]# minicom -D /dev/ttyMICN
```

3.7 Installing OFED (optional)

The coprocessor can communicate with external compute nodes over high-bandwidth InfiniBand* fabric if an Intel® True Scale or Mellanox* InfiniBand* HCA is installed in the platform. This section describes how to install the OFED components that support these capabilities.

Instructions in this sections assume that Intel® MPSS is installed and operational on your host system.

3.7.1 Supported OFED releases

The software stack supports the latest OFED release, which can be obtained from www.openfabrics.org. Always read the associated README and installation instructions before proceeding with installation.

Table 2 OFED distribution supported features

OFED Distribution	Mlx4 (IPoIB)	Mlx5 (IPoIB)	PSM-proxy	ccl-proxy
OFED 3.18-2	Yes (yes)	Yes (yes)	Yes	Yes

3.7.2 Installing OFED 3.18-2

OFED 3.18-2 supports ccl proxy, ccl direct, and kernel mode verbs over Mellanox* mlx5 and mlx4 adapters.

1. Download the official distribution package from https://www.openfabrics.org/downloads/OFED/ofed-3.18-2/

2. Extract the package and access the created directory.

```
[host]# tar xvf OFED-3.18*.tgz
[host]# cd OFED-3.18*
```

3. Install the OFED stack as instructed in the *README.txt* file.

```
[host]# less README.txt
[host]# ./install.pl --with-xeon-phi
```

4. Enable IPv6 forwarding on the host and the coprocessor.

   On the coprocessor:

   Uncomment the line below in the */etc/sysctl.conf* file.

```
net.ipv6.conf.all.forwarding=1
```

   On RHEL* hosts:

   Add the following line to the */etc/sysctl.conf* file:

```
net.ipv6.conf.all.forwarding=1
```

On SLES* hosts:

Change the value of the *net.ipv6.conf.all.forwarding* parameter in the */etc/sysctl.conf* file to *1*.

Reboot the host and the coprocessor to introduce changes.

IPv6 forwarding can also be enabled by running the commands below. This method does not require reboots but is not persistent.

```
[host]# sysctl –w net.ipv6.conf.all.forwarding=1
[micN]# sysctl –w net.ipv6.conf.all.forwarding=1
```

3.7.3 Starting and stopping the OFED stack

If using IPoIB, configure the interfaces in */etc/mpss/ipoib.conf* (for the coprocessor and for the host) before bringing up the services listed below.

1. Ensure the *mpss* service is started. Start it if it is not running:

   ```
   [host]# systemctl status mpss
   [host]# systemctl start mpss
   ```

2. Load the OFED kernel modules:

   ```
   [host]# systemctl start openibd
   ```

3. Bring up the *ibscif* virtual adapter and load the ccl-direct kernel modules:

   ```
   [host]# systemctl start ofed-mic
   ```

4. Start the *ccl-proxy* service (see configuration in: */etc/mpxyd.conf*):

   ```
   [host]# systemctl start mpxyd
   ```

5. *(optional)* Start the *opensmd* service to configure the fabric if the fabric manager is not already running on a switch:

   ```
   [host]# systemctl start opensmd
   ```

Stop the OFED stack by bringing down the services in the listed order:

```
[host]# systemctl stop opensmd
[host]# systemctl stop mpxyd
[host]# systemctl stop ofed-mic
[host]# systemctl stop openibd
```

3.8 Summary

At this point the host and coprocessors are configured in the static pair network configuration. In this setup, a separate private network has been created between the host and each coprocessor. This configuration supports both the Offload and Native programming models.

For users who will be developing and/or executing only Intel® C++/FORTRAN offload directive based programming, basic installation is now complete. You may, however, want to consult Section 4.3 to learn more about user credentials.

Users who will be doing Native program execution on a standalone platform might also wish to learn more about managing the coprocessor's file system, as described in Section 6.

§

4 Coprocessor OS configuration

The coprocessor operating system requires a kernel and a file system image. These components are installed into the host's file system as part of the software stack. The kernel command line is constructed based on a set of configuration files on the host and provided to the coprocessor kernel during boot.

Note: Configuration parameters and procedures for the previous generation Intel® Xeon Phi™ coprocessor x100 are not supported in this version of the software stack.

4.1 Configuration files

micctrl --initdefaults creates Intel® MPSS specific configuration files (if they do not already exist), and populates them with default parameter values. There are two primary configuration files:

1. */etc/mpss/default.conf* contains parameters shared by all coprocessors in the system.
2. */etc/mpss/micN.conf* are coprocessor-specific configuration files (one file for each coprocessor in the system.) Parameters in those files take precedence in configuring the corresponding coprocessors, overriding *default.conf* if the same parameters are present in that file.

Each of these files contains a list of configuration parameters and their arguments. Each parameter must be on a single line. Comments begin with the '#' character and terminate at the end of the line. There are several configuration parameter categories:

1. Parameters that control the boot process.
2. Parameters that configure the coprocessor Linux* kernel.
3. Parameters that configure the file system.
4. Parameters that configure the boot command line.

These parameters are described in detail in Appendix A.

4.1.1 Updating the configuration files

Executing *micctrl --initdefaults* after updating the software stack updates all deprecated or changed configuration parameters.

Note: Running *micctrl --initdefaults* may overwrite your persistent file system (PFS) image and/or configuration files.

Back up an existing configuration before calling *micctrl --initdefaults* if you wish to use that configuration in an earlier Intel® MPSS release.

4.2 Boot configuration

To boot the coprocessor, the *mpssd* daemon needs to:

- Determine the kernel to boot.
- Identify and/or build the file system image.
- Build the kernel command line.

Parameters in the *default.conf* and *micN.conf* configuration files are evaluated for this purpose.

The following sections describe the parameters that are evaluated for this purpose.

4.2.1 Specifying the Linux* kernel

The *KernelImage* and *KernelSymbols* parameters in each *micN.conf* file determine the coprocessor OS boot image and its associated system address map file. Their default values are:

```
KernelImage /usr/share/mpss/boot/bzImage-knl-lb
KernelSymbols /usr/share/mpss/boot/System.map-knl-lb
```

4.2.2 Coprocessor-side kernel command line parameters

Additional kernel command line parameters can be provided in the *KernelExtraCommandLine* configuration parameter in the *default.conf* configuration file. Refer to Appendix A.4.1 for more information.

Section 7 describes a range of configuration options, some of which are conveyed to the coprocessor as kernel command line parameters.

4.2.3 Coprocessor's root file system

The *RootFsImage* parameter in the *micN.conf* file specifies the coprocessor's root file system image. During initialization the software stack creates */var/mpss/persistent-FS-micN.ext4* file system image files for every coprocessor. Each of those files has a default size of 1GB.

Consult Section 6 for information on managing the coprocessor's file systems. Additionally, Appendix A.3 describes in detail parameters that configure the file system.

4.2.3.1 Coprocessor's file system on diskless hosts

Coprocessors installed in hosts with network storage or with limited storage capabilities can be configured to use a single root file system image. Multiple coprocessors can use a single file system image in the read-only mode and use it to create a file system in their RAM. Note that the file system in RAM will not be persistent across coprocessor reboots.

To establish this configuration the *RootFsImage* parameter in the coprocessors' *micN.conf* files should indicate the same file system image. Additionally, the *mic_root_in_ram=1* option should be set in the *KernelExtraCommandLine* parameter in the *default.conf* (to apply to all coprocessors) or in the *micN.conf* file (to apply to a specified coprocessor).

4.2.4 Automatic boot

Coprocessors boot when the *mpss* service is started, provided the *AutoBoot* parameter in the *micN.conf* configuration files is set to *Enabled* (default).

Once the *mpss* service is started the *micctrl -b* command can be used to boot all coprocessor in the system or a specified one.

4.2.4.1 Coprocessor shutdown

Provided the *mpss* service is running, the following commands can be used to reset, reboot or shut down the coprocessor:

- *micctrl -r* – resets the coprocessor to the *ready to boot* state;
- *micctrl -R* – reboots the coprocessor to the *online* (booted) state;
- *micctrl -S* - shuts down the coprocessor;

4.3 User credentialing and authentication

Like typical distributions, the coprocessor Linux* operating system obtains user credentials from standard configuration files (such as */etc/passwd* and */etc/shadow*) in its file system, or from an LDAP or NIS server. No host system user credentials are propagated to the coprocessor's file system.

The coprocessor operating system supports SSH access using SSH keys and/or password authentication. The coprocessor OS looks for SSH keys in each user's *~/.ssh* directory.

In addition, some offload options require that specific user credentials be configured on the coprocessor; see the discussion on COI user ownership in Section 7.4.3 for details.

4.3.1 Coprocessor-side user management

Administrator can use standard Linux* commands to manage user accounts, groups, and user credentials in the coprocessor OS:

- To manage user accounts, use *useradd, usermod,* and *userdel*;
- To manage groups, use *groupadd, groupmod,* and *groupdel*;
- To manage user credentials, use *passwd,* and *ssh-keygen*;

Each command can be invoked with the *–h* or *--help* parameter providing details to its use.

4.3.2 The LDAP service

The coprocessor can use the LDAP service for user authentication.

The network configuration must allow the coprocessor to access the LDAP server, which is not typically on the local host. Therefore an external bridge is required for the coprocessor to reach the LDAP server. Section 5.3.2.2 explains how to configure such network.

Steps below show how to install an LDAP client on the coprocessor and how to create and edit LDAP configuration files. The *mpss-4.4.1-card.tar* archive contains the necessary package. You can find more information on installing software on the coprocessor in Section 6.4.

1. Log in on the coprocessor and install the *nss-pam-ldap* and *rpcbind* packages.

```
[micN]# smart install nss-pam-ldapd
[micN]# smart install rpcbind
```

2. Edit the */etc/nsswitch.conf* file to configure *nss-ldap*, and add LDAP to services you wish to enable (e.g. *passwd, group* and *shadow*). The */etc/nsswitch.conf* file should have contents similar to:

```
[micN]# cat /etc/nsswitch.conf | grep ldap
passwd:     compat ldap
group:      compat ldap
shadow:     compat ldap
```

3. Add the LDAP server (in this example 172.31.1.254) and base domain to the */etc/nslcd.conf* file:

```
[micN]# cat /etc/nslcd.conf | grep –E “uri|base”
uri ldap://172.31.1.254
base dc=my-domain,dc=com
```

4. Configure PAM to allow the LDAP modules for SSH. Modify the */etc/pam.d/common-auth* file by adding the line presented below. Make sure this line is located above the line containing the *pam_deny.so* string.

```
[micN]# cat /etc/pam.d/common-auth | grep ldap
auth sufficient pam_ldap.so
```

5. Turn on the *nslcd* daemon:

```
[micN]# ln -s /etc/init.d/nslcd /etc/rc5.d/S99nslcd
```

4.3.3 The NIS service

The coprocessor can use the NIS service for user authentication.

The network configuration must allow the coprocessor to access the NIS server, which is not typically on the local host. Therefore an external bridge is required for the coprocessor to reach the NIS server. Section 5.3.2.2 explains how to configure such network.

Steps below show how to install an NIS client on the coprocessor and how to create and edit NIS configuration files. The *mpss-4.4.1-card.tar* archive contains the necessary package. You can find more information on installing software on the coprocessor in Section 6.4.

1. Log in on the coprocessor and install the *ypbind-mt* and *rpcbind* packages.

```
[micN]# smart install ypbind-mt
[micN]# smart install rpcbind
```

2. Edit the */etc/nsswitch.conf* file and add NIS to the services you wish to enable (e.g. *passwd*, *group* and *shadow*). The */etc/nsswitch.conf* file should have contents similar to:

```
[micN]# cat /etc/nsswitch.conf | grep nis
passwd:     compat nis
group:      compat nis
shadow:     compat nis
```

3. Add the *NIS/YP* server to the */etc/yp.conf* file. It should contain the following line:

```
[micN]# cat /etc/yp.conf
domain <domain name> server <NIS server IP>
```

4. Edit the */etc/sysctl.conf* file, making sure it contains the line presented below:

```
[micN]# cat /etc/sysctl.conf | grep kernel.domainname
kernel.domainname = <domain name>
```

§

5 Network configuration

The coprocessor does not have hardware Ethernet capabilities. Instead, virtual Ethernet drivers emulate Ethernet devices to enable a standard TCP/UDP IP stack on the host and coprocessor.

The *mpssd* daemon consults parameters in every coprocessor's *micN.conf* configuration file and creates virtual Ethernet interface for each coprocessor. Those parameters and their syntax are described in detail in Appenddix A.5.

The network configuration on the coprocessor is Debian* based. A single */etc/network/interface* file describes all endpoints.

The default file system image includes several configuration files that may need modification to complete the network configuration:

/etc/network/interfaces

/etc/hostname

/etc/nsswitch.conf

/etc/hosts

Note: During the software stack installation a static pair network configuration is established; it is described in Section 5.3.1.

Note: A network-independent virtual console can also be used to access the coprocessor. Refer to Section 3.6.2.2 for more information.

5.1 MAC address assignment

The coprocessor has no pre-assigned MAC address because it lacks hardware network interface. Therefore, a MAC address must be generated and assigned to each virtual network device.

During the coprocessor's boot, the *mpss* service generates MAC addresses for their respective network endpoints. Those addresses are based on serial numbers of the devices. This behavior can be changed by specifying the *HostMacAddress* and *CardMacAddress* configuration parameters in the *micN.conf* file. Their syntax and supported values are described in Appendix A.5.3.

It is also possible to override MAC addresses by editing the host OS specific configuration files. The operating system's documentation specifies which files to edit.

5.2 Host firewall configuration

If the coprocessor's host system firewall is enabled, it may require configuration to allow NFS mounting of host exports. NFS requires access to the following ports:

```
tcp/udp port 111 - RPC 4.0 portmapper
tcp/udp port 2049 - nfs server
```

5.3 Supported network configurations

The supported network configuration schemes allow coprocessors to operate in a wide range of network environments.

- The static pair topology creates a private subnetwork between the host and each coprocessor (Section 5.3.1).
- The bridged topologies:
  - Internal bridge allows coprocessors to communicate with the host and with each other (Section 5.3.2.1).
  - External bridge additionally allows coprocessors to communicate with external compute nodes (Section 5.3.2.2).

5.3.1 Static pair

The static pair is the default network configuration established in the initial software stack installation and configuration. It is sufficient for Intel® C++ and FORTRAN compiler pragma-based offload computation on a standalone (non-clustered) host platform and other program models where coprocessors only need a network connection to the host.

The static pair configuration creates a separate private network between the host and each coprocessor, and assigns an IP address to each of the network endpoints. Figure 8 depicts this configuration. A private network is configured between the host and each coprocessor. Notice that mic0 and mic1 are on separate subnets.

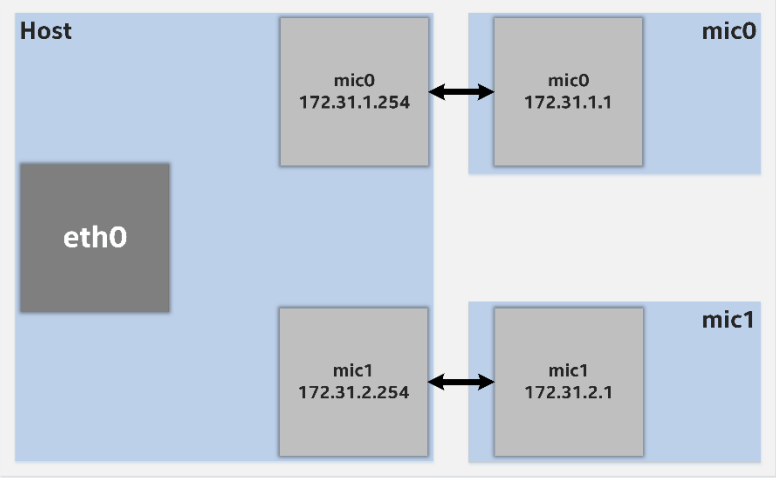

Figure 8 Static pair configuration

5.3.1.1 Static pair configuration

To establish or modify this configuration edit the *micN.conf* file. The values of the *HostNetworkConfig* and *CardNetworkConfig* parameters should resemble the example below:

```
HostNetworkConfig inet static address=172.31.1.254
netmask=255.255.255.0
CardNetworkConfig inet static address=172.31.1.1
netmask=255.255.255.0 gateway=172.31.1.254
```

Note: IP addresses of the host and each coprocessor must be on the same subnet.

A descriptor of each coprocessor endpoint should be added to the host's */etc/hosts* file to associate IP addresses with the coprocessor hostnames. For example:

```
172.31.1.1  zappa-mic0.music.local mic0
172.31.2.1  zappa-mic1.music.local mic1
```

Similarly a descriptor of the corresponding host endpoint should be added to each coprocessor's */etc/hosts* file to associate the host's endpoint IP address with the host's hostnames. For example, mic0's */etc/hosts* might contain:

```
127.0.0.1      localhost.localdomain localhost
::1            localhost.localdomain localhost
172.31.1.254   host zappa.music.local
172.31.1.1     mic0 zappa-mic0.music.local mic0
```

Note that in this example */etc/hosts* includes descriptors of both the host endpoint and the local endpoint.

Endpoints are brought up during the coprocessor's boot. At this point the *ifconfig* command output should be similar to:

```
mic0    Link encap:Ethernet
         inet addr:172.31.1.254  Bcast:172.31.1.255
         Mask:255.255.255.0
mic1    Link encap:Ethernet
         inet addr:172.31.2.254  Bcast:172.31.2.255
         Mask:255.255.255.0
```

The */etc/resolv.conf* file should be populated manually.

5.3.2 Bridged network configurations

A network bridge connects two Ethernet segments or collision domains in a protocol independent way. It is a Link Layer device which forwards traffic between networks based on MAC addresses, and is therefore also referred to as a Layer 2 device.

Intel® MPSS directly supports two types of bridged networks.

5.3.2.1 Internal bridge

An internal bridge allows the connection of one or more coprocessors within a single host system as a subnetwork. In this configuration, each coprocessor can communicate with the host and other coprocessors in the platform. Figure 9 shows an example internal bridge configuration.

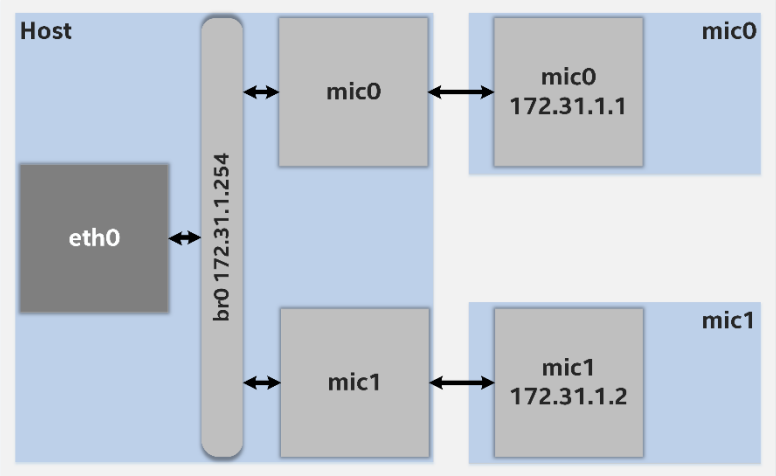

Figure 9 Internal bridge network

This network configuration might be used to support communication between ranks of an MPI application that is distributed across coprocessors and the host.

5.3.2.1.1 Internal bridge configuration

To define the host bridge endpoint, create and/or edit the *ifcfg-br0* file. Its contents should resemble the examples below:

In RHEL*:

```
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
DELAY=0
NM_CONTROLLED=no
BOOTPROTO=static
IPADDR=172.31.1.254
NETMASK=255.255.255.0
```

In SLES*:

```
STARTMODE=onboot
BOOTPROTO=static
USERCONTROL=no
BRIDGE=yes
IPADDR=172.31.1.254
NETMASK=255.255.255.0
```

Note: In SLES* the bridge name is defined in the configuration file's name (i.e. *ifcfg-br0* specifies the bridge name to *br0*)

To connect a coprocessor to the bridge, modify the *HostNetworkConfig* and *CardNetworkConfig* parameters in its *micN.conf* file. Note that the IP address of the bridge and coprocessor must be on the same subnet.

In the example below the coprocessor mic0 is connected to the bridge br0. The *mic0.conf* file contains the following lines:

```
HostNetworkConfig bridge br0
CardNetworkConfig inet static address=172.31.1.1
netmask=255.255.255.0 gateway=172.31.1.254
```

The host's */etc/hosts* file must contain a descriptor of the coprocessor endpoint to associate IP addresses with the coprocessor hostname. For example:

```
172.31.1.1  zappa-mic0.music.local mic0
172.31.1.2  zappa-mic1.music.local mic1
```

Similarly, a descriptor of the corresponding host bridge endpoint should be added to each coprocessor's */etc/hosts* file to associate the host's endpoint IP address with the host's hostnames. For example, mic0's */etc/hosts* might contain:

```
127.0.0.1      localhost.localdomain localhost
::1            localhost.localdomain localhost
172.31.1.254   host zappa.music.local
172.31.1.1     mic0 zappa-mic1.music.local mic0
172.31.1.2     mic1 zappa-mic1.music.local mic1
```

Note that in this example */etc/hosts* includes descriptors of the local endpoint, the host endpoint, and the other coprocessor.

Host endpoints are brought up during the coprocessor's boot. Bridge interface needs to be brought up manually by the user.

At this point the *brctl show* command can be used to check the status of the bridge. Its output should be similar to:

```
bridge name bridge id            STP enabled interfaces
br1            8000.66a8476a8f15 no             mic0
                                                mic1
```

Relevant output from the *ifconfig* command should be similar to:

```
br0       Link encap:Ethernet
          inet addr: 172.31.1.254  Bcast: 172.31.1.255
          Mask:255.255.255.0
mic0      Link encap:Ethernet
mic1      Link encap:Ethernet
```

These commands show that the mic0 and mic1 virtual network interfaces are slaved to the bridge br0.

The coprocessor(s) must be rebooted to apply the new configuration.

5.3.2.2 External bridge

The external bridge configuration bridges coprocessors to an external network. This is the typical configuration required when coprocessors are deployed in a cluster to support remote communication among them and/or host processors across different compute nodes.

Figure 10 depicts a cluster in which the coprocessors on each host node are bridged to an external network. In this configuration, IP addresses can be statically assigned by the system administrator, or by a DHCP server on the network, but must generally be on the same subnet.

InfiniBand*-based networking is not shown in this figure. This type of networking usually provides higher bandwidth than IP networking supported by the Intel® MPSS Virtual Ethernet driver.

Before attempting to configure this network topology, ensure that you provided a large enough IP address space to accommodate the nodes of the externally bridged networks.

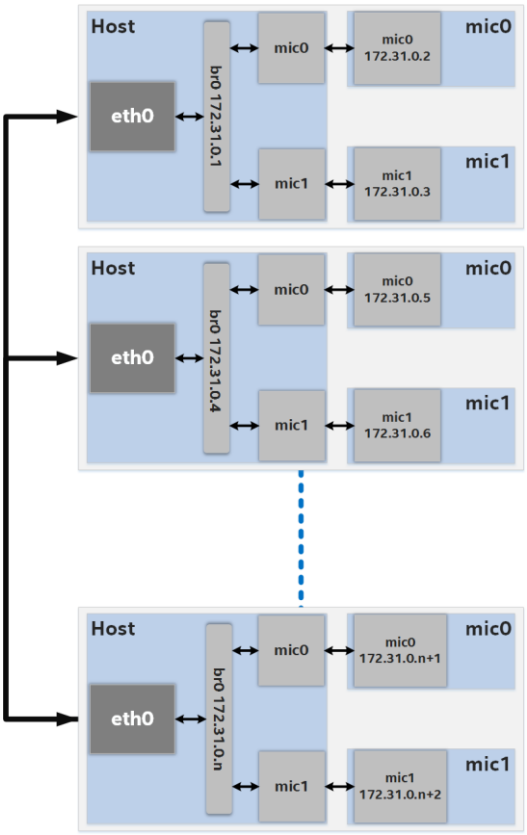

Figure 10 External bridge network

5.3.2.2.1 External bridge configuration

An external bridge configuration requires all network endpoints, including the bridge and coprocessors, to be on the same subnet. IP addresses can be assigned by a DHCP server, or specified in local configuration files. In either case, it is usually desirable to use static IP schemes in cluster environments so nodes can correlate to known IP addresses.

Note: In RHEL* the *ifcfg-<ETH_DEVICE>* Ethernet device configuration file should have contents similar to:

```
DEVICE=eth0
BRIDGE=br0
```

If the IP address assignment is static, use the internal bridge configuration described in Section 5.3.2.1.

Note: Add the following line to the bridge descriptor file in SLES*

```
BRIDGE_PORTS=<ETH_DEVICE>
```

If the IP addresses are assigned by a DHCP server, the bridge descriptor file, for example *fcfg-br0*, should have contents similar to:

In RHEL*:

```
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
DELAY=0
NM_CONTROLLED=no
BOOTPROTO=dhcp
NETMASK=255.255.255.0
```

In SLES*:

```
STARTMODE=onboot
BOOTPROTO=dhcp
USERCONTROL=no
BRIDGE=yes
BRIDGE_PORTS=<ETH_DEVICE>
```

BOOTPROTO is now set to *dhcp* rather than *static*, and the *IPADDR* parameter was removed.

The coprocessor will be able to obtain its IP address from the DHCP server once the *CardNetworkConfig* parameter in its *micN.conf* configuration file has the following value:

```
CardNetworkConfig inet dhcp
```

§

6 Managing coprocessor's file system

The persistent files system (PFS) is the root file system of the Intel® Xeon Phi™ coprocessor x200 is maintained on the host with a virtual block device on the coprocessor. This device redirects read/write requests to and from the host over PCIe. This solution enables the user to manage the coprocessor's file system much like the one in a standard Linux* OS. The mechanism is depicted in Figure 11.

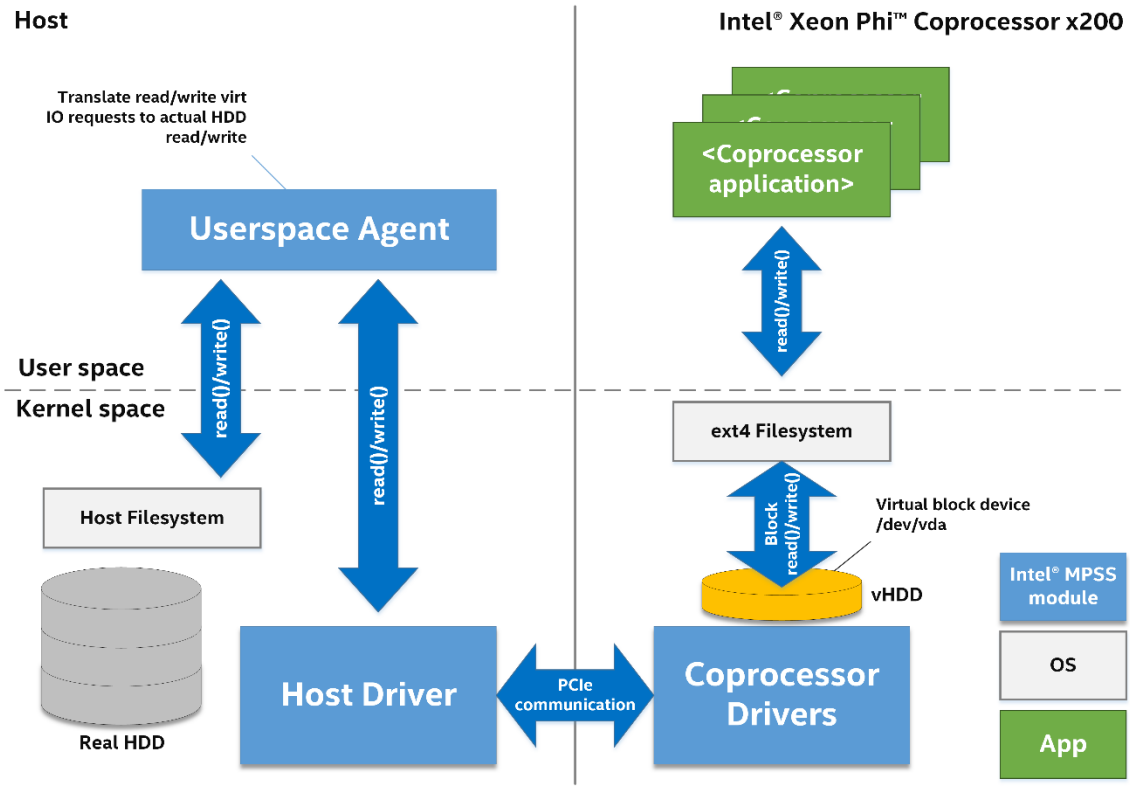

Figure 11 Persistent file system

6.1 Changing the persistent file system's size

After the initial configuration of the software stack each coprocessor has its own */var/mpss/persistent-FS-knl-lb-micN.ext4* file, which contains its file system. By default those files have a size of 1GB. However, in some situations it may be desirable to increase or decrease their sizes. Follow the instructions below to change the PFS size:

1. Make sure the coprocessor is in the *shutdown* state.

   ```
   [host]# micctrl –Sw micN
   ```

   Alternatively, stop the *mpss* service to shut down all coprocessors.

   ```
   [host]# systemctl stop mpss
   ```

2. Check the file system for errors.

   ```
   [host]# e2fsck –f <PFS_file>
   ```

3. Increase the size of the file containing the PFS.

   ```
   [host]# truncate –s <size> <PFS_file>
   ```

4. Resize the file system.

   ```
   [host]# resize2fs <PFS_file>
   ```

Note: To decrease the size of the PFS file reverse the order of steps 3 and 4. First use the *resize2fs* command to reduce the size of the file system, then use the *truncate* command to reduce the PFS file size. Make sure that the new file size is not smaller than the new size of the file system.

5. Boot the coprocessor.

   ```
   [host]# micctrl –b micN
   ```

   Start the *mpss* service if you stopped it in step 1.

   ```
   [host]# systemctl start mpss
   ```

6.2 Managing the persistent file system in a cluster environment

Every file containing a file system image can be mounted on the host and modified. The modified file system image can be then copied and used on other coprocessors.

An alternative procedure is to boot a single node and modify the file system directly on the coprocessor. After it is shut down, the modified file system image can be copied and propagated to the remaining coprocessors.

6.3 Installing and using Lustre*

Lustre* is a parallel, distributed file system. It can be used in a cluster environment for coprocessors to share data.

To use Lustre*, first install the packages listed below on the coprocessor. These packages are distributed in the *mpss-4.4.1-card.tar* archive. Once the packages are installed, Lustre* can be configured and used.

- *lustre-client-<version>.rpm*
- *lustre-client-modules-<version>.rpm*

Note: Refer to Section 6.4 for instructions on how to install additional software on the coprocessor.

Execute the following commands to configure Lustre* on the coprocessor.

```
[micN]# echo 'options lnet networks="tcp0(eth-dev)"' > \
/etc/modprobe.d/lustre.conf
[micN]# modprobe lnet
[micN]# modprobe lustre
```

After the configuration is complete mount the Lustre* share from your network by executing the commands below on the coprocessor. The *<MGS IP>* indicates the IP address of the Management Server.

```
[micN]# mkdir –p /mnt/lustre
[micN]# /sbin/mount.lustre \
<MGS IP>@tcp0:/<lustreFS_name>/mnt/lustre
```

6.4 Adding software to the coprocessor

Typical installations are not static, and often require the system administrator to add additional files or software to the file system.

Supplementary software for the coprocessor is distributed in the *mpss-4.4.1-card.tar* archive. The archive contains a squashfs repository, which is installed on the host. The repository is then accessed by the *smart* package manager on the coprocessor.

6.4.1 Using smart package manager

The *smart* package manager allows users to install additional software on the coprocessor. The squashfs repository image on the host (*/usr/share/mpss/boot/repo-card.squashfs*) is passed to the *RepoFsImage* configuration parameter in the *default.conf* file. The repository image is automatically detected and configured by the *smart* package manager. Available packages can be installed by running the *smart install <package>* command, for example:

```
[micN]# smart install perf
```

You can find more information on the *smart* package manager at https://labix.org/smart.

Refer to Appendix A.3.2 for more information on the *RepoFsImage* parameter.

6.4.2 Copying packages to the coprocessor

This is an alternative procedure which enables users to install packages not distributed in the squashfs repository.

1. Copy the RPMs to the coprocessor with *scp*:

```
[host]$ scp <rpm_packages> micN:<some directory>
```

2. SSH to the coprocessor as root:

```
[host]# ssh micN
```

3. Install the RPMs using the *rpm* utility:

```
[micN]# cd <some directory>
[micN]# rpm -ihv <rpm_packages>
```

The *Failed dependencies* errors can be solved by iteratively copying RPMs and attempting installation.

It is also possible to use *smart* to install RPM packages. The package manager will resolve dependencies, provided they are satisfied with packages available in the repository.

Use the command below to install RPM packages with *smart*.

```
[micN]# smart install <rpm_packages>
```

§

7 Intel® MPSS component configuration and tuning

7.1 The coprocessor's operating system configuration and tuning

7.1.1 Tickless kernel support

The operating system uses a hardware component called *the Programmable Interrupt Timer*, which allows it to perform numerous periodical functions. The operating system kernel registers a function called *interrupt handler* to respond to those interrupts. Every hit of the timer is called a tick. The fixed timer causes higher power consumption, because of the CPU wake-ups happening numerous times every second, even if the processor is seemingly idle. Tickless kernel eliminates this problem by putting unused CPU cores to *powersave* states and using only the first core to generate appropriate ticks.

The coprocessor OS kernel is capable of running in tickless mode out of the box (appropriate kernel configuration options are included in the provided kernel configuration file). However, this functionality has to be enabled using boot time kernel parameters. Passing the *nohz_full* option to the coprocessor OS kernel parameters will enable the tickless feature.

Refer to the kernel documentation (*linux/Documentation/timers/NO_HZ.txt*) for more details on the tickless feature.

7.1.2 Huge page support

The coprocessor has support for huge pages by default. Huge pages should not be confused with transparent huge pages (THP). Huge pages can be configured with standard Linux* options: kernel command line parameters or *sysfs*.

7.1.2.1 Transparent hugepages support

The transparent hugepages feature allows kernel to implicitly use hugepages instead of normal pages to optimize memory management. The coprocessor OS kernel supports transparent hugepages by default.

There are two ways to disable this feature:

- Set the *transparent_hugepage=never* boot argument for the coprocessor OS kernel.
- Execute the following commands on the coprocessor OS:

```
[micN]# echo never > \
/sys/kernel/mm/transparent_hugepage/enabled
[micN]# echo never > \
/sys/kernel/mm/transparent_hugepage/defrag
```

Refer to the kernel documentation (*linux/Documentation/vm/transhuge.txt*) for more details on transparent hugepages.

7.2 Host driver configuration

7.2.1 Peer to peer (p2p) support

SCIF supports the direct transfer of data from one coprocessor directly into the physical memory of another one on the same host. This capability is referred to as Peer to Peer or P2P.

P2P support is enabled by default.

Note: In kernel versions newer than 3.1.0, the kernel has two different limits for locked pages: one limit for pages locked using standard system calls and another limit for pages locked by kernel modules on behalf of user processes.

Note: The mechanism for specifying the pinned pages limit may change in a future release.

7.3 Controlling the number of pinned memory pages per user process

By default, SCIF enforces ulimit checks of the memory that *scif_register()* pins. Memory pages pinned using *scif_register()* are counted towards the *memlock* item (max locked-in-memory address space) of ulimit limits.

Note: The amount of locked memory should be tuned manually by the system administrator depending on the requirements of user workloads offloaded to coprocessors. The default configuration might not be sufficient for applications to run.

The *memlock* item value is set to *unlimited* in the Coprocessor OS. It can be changed by editing either the */etc/security/limits.conf* file or **.conf* files in the */etc/security/limits.d* directory.

The exception are all processes spawned by the *coi_daemon* on the Coprocessor OS. In their case the *memlock* item value should be changed by editing the *memlock_limit* variable in the */etc/init.d/coi* init script.

On the host platform, the *memlock* item value is not changed and retains its set value.

The ulimit checking can be disabled by following the instructions below.

- For the host platform, set the *scif_ulimit_check* value to 0 in the */etc/modprobe.d/mic_x200.conf* file, stop the *mpss* service and reload the driver modules.

```
[host]# systemctl stop mpss
[host]# micctrl --unload-modules
[host]# micctrl --load-modules
[host]# systemctl start mpss
```

- For the Coprocessor OS, add the *scif.scif_ulimit_check=0* parameter to the *KernelExtraCommandLine* option in the */etc/mpss/default.conf* (or */etc/mpss/micN.conf*) file and restart the coprocessor.

```
[host]# micctrl –Rw micN
```

7.4 COI configuration

7.4.1 Process oversubscription

Only configure concurrent processing when there is a real need for this feature. Otherwise, any workload running with the concurrent active processes on the device will likely result in performance degradation.

To run more concurrent processes, set the limit of file descriptors to 10 for each offload process. Note that, depending on the memory usage of each process, a large number of concurrent offload processes may exhaust the memory available on the device.

To run 200 concurrent processes, follow the steps below:

1. Log in to the coprocessor as root
2. Locate and terminate the Intel® COI active process.

```
[micN]# ps axf | grep coi
5147 ?  Sl  0:00 /usr/bin/coi_daemon --coiuser=micuser
[micN]# killall coi_daemon
```

3. Set the concurrent process to 200.

```
[micN]# ulimit -n 200
[micN]# coi_daemon --coiuser=micuser \
--max-connections=200 &
[micN]# exit
```

To permanently change maximum number of offloading processes, you must change the COI initialization parameters in the startup script:

```
coiparams='--max-connections=<max no of processes> <other COI
options>'
```

COI parameters are stored in */etc/coi.conf* or */etc/sysconfig/coi* files in the coprocessor's file system. Create these files if they do not exist. Restart the *coi_daemon* to implement the change:

```
[micN]# /etc/init.d/coi restart
```

For the complete list of *coi_daemon* parameters, refer to its help option:

```
[micN]$ coi_daemon –h
```

7.4.2 Limiting maximum number of COI processes

The *coi_daemon* on the coprocessor spawns separate processes for each host process that performs offloading to the coprocessor. The maximum number of the processes is limited by the *--max-connections* parameter. See Section 7.4.1 for instructions on how to change this limit.

7.4.3 SCIF configuration

The minimum possible amount of lockable memory accepted by COI is 512kB. It should be noted at this point that this value does not permit buffer creation. The exact amount of locked memory to be set is highly dependent on the requirements of the user offloads and requires manual tuning. The administrator may choose to use the *unlimited* value, however this setting in some circumstances may be unsafe due to possibility of system abuse. Refer to Section 7.3 for more information.

To check the current amount of locked memory use the *ulimit –l* or *ulimit –a* commands and read the value of the *max locked memory* entry.

7.4.4 Coprocessor-side buffer memory configuration

The default configuration limits the buffer memory on the coprocessor to 85% of their memory size. This value can be modified manually by editing the */etc/fstab* file and the */tmp* mountpoint on the coprocessor. However, it is highly recommended not to modify the default value.

7.4.5 COI offload user options

The *coi_daemon* on the coprocessor spawns processes on behalf of client processes on the host. Two options are available for assigning ownership of the spawned processes:

- *micuser* ownership - each COI process spawned by the *coi_daemon* is owned by the user *micuser*. It is the default ownership mode. This feature implies that various users on the host system spawn processes on the coprocessor as one user.
- *_Authorized* ownership – each COI process spawned by the *coi_daemon* is owned by the same user as the corresponding host client process. Authentication of user credentials occurs using an *.mpsscookie* file located in the user's home directory created and managed by the *mpss* daemon.

7.4.5.1 Configuring the ownership mode

The ownership mode is configured by adding the parameter:

```
coiparams='--coiuser=<mode>'
```

to the */etc/coi.conf* or */etc/sysconfig/coi* files in the coprocessor's file system, where *<mode>* is *micuser* or *_Authorized*. Parameters in */etc/sysconfig/coi* take precedence over parameters in */etc/coi.conf*. A change to the ownership mode takes effect when the *coi_daemon* is restarted.

```
[micN]# /etc/init.d/coi restart
```

Alternatively, the *--coiuser* option can be passed to the *coi_daemon* when it is started:

```
[micN]# coi_daemon --coiuser=<mode>
```

7.4.5.2 Example

To set the *_Authorized* user mode:

```
[micN]# echo coiparams='--coiuser=_Authorized' > /etc/coi.conf
[micN]# /etc/init.d/coi restart
```

For detailed information about the *--coiuser* parameter, run the *coi_daemon* on the coprocessor with the *--help* option:

```
[micN]$ coi_daemon –help
```

7.4.6 COI daemon and COI processes affinity

By default the COI daemon and COI processes affinity allows running the processes on all threads. However, for some workloads performance may be improved once the COI daemon runs on threads assigned exclusively to it. Best performance can be achieved by tuning the affinity for each workload.

The system administrator can set up custom affinity between the COI daemon and COI processes. The administrator selects CPU threads that are exclusive for the daemon which renders the remaining cores to be used by COI processes. This operation is performed by exporting the *COI_DAEMON_AFFINITY* variable in the COI configuration script (see Section 7.4.1). The variable is a comma-separated list of integers that define CPU threads for the daemon process, for example:

```
export COI_DAEMON_AFFINITY="1,2,3,4"
```

If the variable is not set properly, its contents will be ignored. It is possible to verify the affinity with the following command executed on the coprocessor:

```
[micN]# cat /proc/`pidof coi_daemon`/status | \
grep Cpus_allowed_list
```

7.5 SSH configuration

7.5.1 Improving SSH connection speed

It is possible to increase the speed of SSH and SCP connections by using a fast encryption algorithm (cipher) to encrypt connections. The coprocessor supports the following encryption ciphers:

```
chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-
ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com
```

The *Ciphers* keyword in the */etc/ssh/ssh_config* (global configuration) or *~/.ssh/ssh_config* (single user configuration) file allows to specify allowed ciphers in a comma-separated list. Since the SSH negotiation process encrypts the connection using the first cipher on the client's side that is supported by the server it is most beneficial to place the fastest ciphers at the beginning of the list; for example:

```
Ciphers aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes192-
ctr,aes256-ctr,aes192-ctr,aes128-ctr,chacha20-poly1305@openssh.com
```

§

8 Intel® MPSS tools

This chapter describes the range of system tools included in the Intel® MPSS. These tools allow to view coprocessor-related information, modify some aspects of the coprocessor configuration and aid in creating custom monitoring application.

8.1 micctrl

The *micctrl* utility is a multi-purpose toolbox for the system administrator. It provides the following categories of functionality.

- Coprocessor state control – boot, shutdown and reset control.
- Configuration files initialization and propagation of values.

First argument of each *micctrl* command must specify the action to perform, and can be followed by a number of additional sub-options. Each *micctrl* command may be followed by a space-separated list of coprocessors, which is shown in the syntax statements as *[list of coprocessors]* and specifies the devices to control. For example, the list may be *mic1 mic3*, if these are the coprocessors to control. If no coprocessors are specified, the command will be executed for all coprocessors in the system.

Note: Root privileges are required to run *micctrl* commands (except *micctrl --status, micctrl --help* and *micctrl --wait*).

8.1.1 Checking the coprocessor's state

Command syntax:

```
[host]$ micctrl (-s|--status) [sub-options] \
[list of coprocessors]
```

Displays the current state of all or selected coprocessors. Each coprocessor can be in one of the following states:

- *shutdown* – the coprocessor is shut down;
- *ready to boot* – the coprocessor is ready to receive a boot command;
- *online* – the coprocessor OS is booted and ready to work;
- *shutting down* – transition to the *shutdown* state.
- *waking up* – transition to the *ready to boot* state.
- *booting* – transition to the *online* state.
- *resetting* – transition to the *ready to boot* state.
- *error* – the coprocessor encountered an error and needs to be reset by issuing the *micctrl -r* command.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

8.1.2 Displaying current configuration

Command syntax:

```
[host]# micctrl --config [sub-options] \
[list of coprocessors]
```

Displays summary of the current configuration of coprocessors in a human-readable format.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

8.1.3 Booting the coprocessor

Command syntax:

```
[host]# micctrl (-b|--boot) [sub-options] \
[list of coprocessors]
```

Boots all or selected coprocessors. To successfully boot the coprocessors must be in either *ready to boot* or *shutdown* state. Once booted their state is changed to *online*.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

- *-w, --wait*

Waits for the *--boot* command to finish successfully before exiting *micctrl*.

- *-t <timeout>, --timeout=<timeout>*

Sets the time out value in seconds for the *--wait* option.

8.1.4 Shutting down the coprocessor

Command syntax:

```
[host]# micctrl (-S|--shutdown) [sub-options] \
[list of coprocessors]
```

Shuts down gracefully all or selected coprocessors that are in a stable state (*ready to boot, online, shutdown* or *error*). Once completed their status is changed to *shutdown*.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

- *-w, --wait*

Waits for the *--shutdown* command to finish successfully before exiting *micctrl*.

- *-t <timeout>, --timeout=<timeout>*

Sets the time out value in seconds for the *--wait* option.

- *-f, --force*

Forces shutdown regardless of the state of coprocessors. Normally the coprocessors are not shut down unless they are in a stable state resulting in *micctrl* returning an error.

Note: Forcing shutdown may corrupt the coprocessor's file system.

8.1.5 Resetting the coprocessor

Command syntax:

```
[host]# micctrl (-r|--reset) [sub-options] \
[list of coprocessors]
```

Resets all or selected coprocessors that are in a stable state (*ready to boot, online, shutdown* or *error*). Unlike the *--shutdown* command, *--reset* does not wait for the coprocessor OS to shut down gracefully. Once completed stathe of coprocessors is changed to *ready to boot*.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

- *-w, --wait*

Waits for the *--reset* command to finish successfully before exiting *micctrl*.

- *-t <timeout>, --timeout=<timeout>*

Sets the time out value in seconds for the *--wait* option.

- *-f, --force*

Forces reset regardless of the state of coprocessors. Normally the coprocessors are not reset unless they are in a stable state resulting in *micctrl* returning an error.

Note: Forcing reset may corrupt the coprocessor's file system.

8.1.6 Rebooting the coprocessor

Command syntax:

```
[host]# micctrl (-R|--reboot) [sub-options] \
[list of coprocessors]
```

Reboots all or selected coprocessors that are in the *online* state. This commands shuts down coprocessors gracefully and boots them. It is equivalent of running *micctrl -S* followed by *micctrl -b*.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

- *-w, --wait*

Waits for the *--reboot* command to finish successfully before exiting *micctrl*.

- *-t <timeout>, --timeout=<timeout>*

Sets the time out value in seconds for the *--wait* option.

8.1.7 Waiting for the coprocessor's state change.

Command syntax:

```
[host]$ micctrl (-w|--wait) [sub-options] \
[list of coprocessors]
```

Waits for the previous state change command (*--boot, --shutdown, --reset* or *--reboot*) to complete and exits *micctrl*.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

- *-t <timeout>, --timeout=<timeout>*

Sets the time out value in seconds for the *--wait* command.

8.1.8 Creating and restoring default configuration

Command syntax:

```
[host]# micctrl --initdefaults [sub-options] \
[list of coprocessors]
```

Creates configuration files and PFS file system images for each coprocessor and fills them with default values if such configuration does not exist. If configuration files and PFS image files are found on the system, this command removes them and replaces them with new files containing default configuration. This command requires the coprocessors to be in the *ready to boot* or *shutdown* state.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

- *--keep-config*

Prevents the *--initdefaults* command from deleting the configuration files. PFS image files are replaced.

- *--keep-mage*

Prevents the *--initdefaults* command from deleting the PFS image files. Configuration files are replaced.

- *-f, --force*

Prevents prompting the user that configuration files and/or PFS image files are removed.

8.1.9 Downloading the coprocessor's kernel log

Command syntax:

```
[host]# micctrl --log [sub-options] [list of coprocessors]
```

If available, downloads and prints kernel log of all or specified coprocessors.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

8.1.10 Loading and unloading the mpss kernel modules

Command syntax:

```
[host]# micctrl --load-modules [sub-options] \
[list of coprocessors]

[host]# micctrl --unload-modules [sub-options] \
[list of coprocessors]
```

Loads and unloads kernel modules required by the coprocessor. It is necessary to issue the *--load-modules* command after the initial software stack installation.

Sub-options:

- *-h*

Displays the help message and ignores all other options.

- *-v*

Increases the level of verbosity of the output.

8.1.11 micctrl return codes

After execution *micctrl* returns one of the following codes:

- 0 - Command executed successfully.
- 1-N - Number of coprocessors, on which the command failed.
- 201 - General error.
- 202 - Cannot execute command because the MPSS daemon is running.
- 203 - Cannot execute command because the MPSS daemon is not running.
- 204 - Error while loading or unloading kernel modules.
- 205 - Wrong timeout.
- 206 - Invalid device name in the command line.

8.2 micinfo

The *micinfo* tool displays information about each coprocessor installed in the system. The tools displays firmware-, sensors- and driver-related information. Some of the data can be retrieved even if the coprocessor is not in the *online* state.

Note: Root privileges are required to view all information provided by *micinfo*.

8.2.1 Options

- *-h, --help*

Displays the help documentation for the tool.

- *--version*

Displays the version of the tool.

- *-v, --verbose*

Displays extra information about internal operations or messages.

- *-d, --device=<device-list>*

Specifies coprocessors for which data will be retrieved. This option accepts coprocessors' names in form of a comma-separated list, range or a combination of both. For example: *mic0-mic3, mic1,mic3-mic5, mic2*. If no coprocessors are specified, the tool lists information for all coprocessors in the system.

- *-g, --group=<group-list>*

Allows to narrow down the output of the tool to a specified subject called group and determined by the *<group-list>* argument. This option accepts a single name of a group or a comma-separated list of groups' names. Valid groups are listed below.

- *system*: shows information about the host.
- *versions*: shows each coprocessor's firmware version and serial number.
- *board*: shows information about the coprocessor board.
- *core*: shows number of cores, their voltage and frequency.
- *thermal*: shows information about the cooling fan and thermal related data.
- *memory*: shows information about each coprocessor's memory.

8.2.2 Usage

```
[host]# micinfo [OPTIONS] [(-g |--group=)<groups>]
[(-d |--device=)<device-list>]
```

1. Executing *micinfo* with no options will display all information for all of the groups and for all coprocessors in the system.

```
[host]$ micinfo
micinfo Utility Log
Created On Wed Jul  6 05:51:33 2016
System Info:
    <output truncated>
Version:
    <output truncated>
Board:
    <output truncated>
    PCIe Max read req size          : Insufficient Privileges
    <output truncated>
Core:
    <output truncated>
Thermal:
    <output truncated>
    Die Temp                        : Not Available
    <output truncated>
Memory:
    <output truncated>
```

Insufficient Privileges is displayed when *micinfo* is not run with root privileges.

Not Available is displayed if a value could not be retrieved. For example, a hardware sensor does not report its value or the coprocessor is not in the *online* state.

2. In the example below *micinfo* displays information about *version* and *core* groups. The output was specified for coprocessor mic0.

```
[host]$ micinfo --group=version,core --device=mic0
micinfo Utility Log
Created On Wed Jul  6 05:50:35 2016
Device No: 0, Device Name: mic0 [x200]
Version:
    SMC Firmware Version    : 121.31.10446
    Coprocessor OS Version  : 4.9.13-mpss_4.4.1.4483 GNU/Linux
    Device Serial Number    : 0123456789AB
    BIOS Version            : GVPRCRB8.86B.0014.D18.1703092150
    BIOS Build date         : 03/09/2017
    ME Version              : 3.2.2.8

Core:
    Total No of Active Cores  : 60
    Threads per Core          : 4
    Voltage                   : 900.00 mV
    Frequency                 : 1.20 GHz
```

8.3 miccheck

The *miccheck* tool executes a suite of diagnostics tests that verify the configuration and current status of coprocessors and the software stack.

The default tests include:

- Checking number of coprocessors the OS detects in the system.
- Checking whether drivers are loaded.
- Checking number of coprocessors the driver detects in the system.
- Checking whether the *mpss* daemon is running.
- Checking the POST code returned by each coprocessor.
- Checking whether the *systools* daemon is running on the coprocessor.

Optional tests include:

- Verifying whether each coprocessor's firmware version is the same as provided in the software stack release package.
- Verifying whether the coprocessor can be pinged.
- Checking whether the coprocessor can be accessed through ssh.
- Checking whether the *coi_daemon* is running on the coprocessor.

8.3.1 Options

- *-h, --help*

Displays the help documentation for the tool.

- *--version*

Displays the version of the tool.

- *-v, --verbose*

Displays extra information about internal operations or messages.

- *-d, --device=<device-list>*

Specifies coprocessors for which the tests will be executed. This option accepts coprocessors' names in form of a comma-separated list, range or a combination of both. For example: *mic0-mic3, mic1,mic3-mic5, mic2*. If no coprocessors are specified, the tool lists information for all coprocessors in the system.

- *-f, --firmware*

Executes the firmware diagnostic test, which verifies whether each coprocessor's firmware version is the same as provided in the software stack release package.

- *-p, --ping*

Executes the ping diagnostic test which verifies whether each coprocessor can be pinged.

- *--ssh*

Verifies if the host can establish an SSH connection to each coprocessor.

- *-c, --coi*

Verifies whether the COI daemon is running on each coprocessor.

8.3.2 Examples

1. Running the default test cases.

```
[host]# miccheck
Executing default tests for host
  Test 0: Check number of devices the OS sees in the system
... pass
  Test 1: Check required drivers are loaded ... pass
  Test 2: Check number of devices driver sees in the system
... pass
  Test 3: Check mpssd daemon is running ... pass
Executing default tests for device: mic0
  Test 4 (mic0): Check device state and POST code ... pass
  Test 5 (mic0): Check systoolsd is running in device ... pass
Status: OK
```

2. Executing the default and ping tests on coprocessor mic0.

```
[host]# miccheck --ping --device=mic0
Executing default tests for host
  Test 0: Check number of devices the OS sees in the system
... pass
  Test 1: Check required drivers are loaded ... pass
  Test 2: Check number of devices driver sees in the system
... pass
  Test 3: Check mpssd daemon is running ... pass
Executing default tests for device: mic0
  Test 4 (mic0): Check device state and POST code ... pass
  Test 5 (mic0): Check systoolsd is running in device ... pass
Executing optional tests for device: mic0
  Test 6 (mic0): Check device can be pinged over its network
interface ... pass
Status: OK
```

8.4 micsmc-cli

The *micsmc-cli* tool can be used to monitor coprocessors' status and settings, and also to introduce changes to their configuration.

8.4.1 Options

- *--help*

Displays the help documentation for the tool.

- *--version*

Displays the version of the tool.

- *--verbose*

Shows extra information about internal operations or messages.

- *-d, --device=<device-list>*

Specifies coprocessors for which data will be retrieved. This option accepts coprocessors' names in form of a comma-separated list, range or a combination of both. For example: *mic0-mic3, mic1,mic3-mic5, mic2*. If no coprocessors are specified, the tool lists information for all coprocessors in the system.

8.4.2 Usage

```
[host]# micsmc-cli (disable|enable) [options] <settings-list>
<device-list>
```

- *disable <settings-list> <device-list>*

Disables one or more specified configuration settings for the selected coprocessor(s). If no coprocessors are specified the command will apply to all coprocessors in the system.

- *enable <settings-list> <device-list>*

Enables one or more configuration settings for the selected coprocessor(s). If no coprocessors are specified the command will apply to all coprocessors in the system.

- *<device-list>*

This element is a space-separated list of coprocessors' names, ranges of names or a single name. At least one coprocessor must be specified.

- *<settings-list>*

This element accepts a single setting or a comma-separated list of settings. The available settings are *'ecc', 'led', 'turbo'* and *'all'*; the special word *'all'* can be used to replace all other settings. At least one setting must be specified.

```
[host]# micsmc-cli show-settings [options] <settings-list>
[(-d |--device=)<device-list>]
```

- *show-settings <settings-list> [(-d |--device=)<device-list>]*

Displays the current values of a specified setting for the selected coprocessor(s). If no coprocessors are specified the command will apply to all coprocessors in the system.

```
[host]# micsmc-cli show-data [show-data options] [options]
[(-d |--device=)<device-list>]
```

- *show-data [show-data options] [(-d |--device=)<device-list>]*

Displays general information based on option switches for each selected coprocessor. If no coprocessors are specified the command will apply to all coprocessors in the system. This command accepts the options listed below, which will filter the output. If no options are specified the command will display all the information.

  - *-a, --all*

  Replaces *--info, --cores, --mem, --freq, --temp*.

  - *-c, --cores*

  Displays dynamic (instant) core information.

  - *-f, --freq*

  Display dynamic (instant) frequency and power usage information.

  - *-m, --mem*

  Displays dynamic (instant) memory usage information.

Note: The displayed total memory also includes the memory cached and buffered by the kernel, thus the sum of the free and used memory will not add up to displayed total. See the documentation of */proc/meminfo* for more information.

- *-t, --temp*

Displays dynamic (instant) temperature information.

- *-i, --info*

Displays static coprocessor information.

- *--offline*

Displays each *offline* coprocessor in a comma-separated list.

- *--online*

Displays each *online* coprocessor in a comma-separated list.

8.4.3 Examples

1. Enabling the *led* setting for coprocessor mic0.

```
[host]# micsmc-cli enable led mic0
Enable Setting Results:
   mic0:
      led                         : Successful
```

2. Disabling the *led* setting for coprocessor mic0.

```
[host]# micsmc-cli disable led mic0
Disable Setting Results:
   mic0:
      led                         : Successful
```

3. Displaying information on the *led* setting for all coprocessors in the system.

```
[host]$ micsmc-cli show-settings led
Display Current Settings:
   mic0:
      led                         : disabled
   mic1:
      led                         : disabled
```

4. Displaying the frequency and power usage information for all coprocessors.

```
[host]$ micsmc-cli show-data --freq
mic0:
   Observed Frequency and Power Usage:
      Core Frequency              : 1.3000 GHz
      PCIe                        : 24.0 Watts
      2x3                         : 36.0 Watts
      2x4                         : 19.0 Watts
      Average 0                   : 85.0 Watts
      Current                     : 79.0 Watts
      Maximum                     : 157.0 Watts
      VCCP                        :  5.0 Watts
      VCCU                        :  4.0 Watts
      VCCCLR                      : 14.0 Watts
      VCCMLB                      : 17.0 Watts
```

```
        VCCMP                       :  3.0 Watts
        NTB1                        :  3.0 Watts
mic1:
   Observed Frequency and Power Usage:
      <output truncated>
```

8.5 micfw

The *micfw* tool is used to update the coprocessor's firmware. It can also be used to retrieve the firmware version from a coprocessor or an image file.

8.5.1 Options

- *--help*

Displays the help documentation for the tool.

- *--version*

Displays the version of the tool.

- *--verbose*

Shows extra information about internal operations or messages.

- *-d, --device=<device-list>*

Specifies coprocessors for which data will be retrieved. This option accepts coprocessors' names in form of a comma-separated list, range or a combination of both. For example: *mic0-mic3, mic1,mic3-mic5, mic2*. If no coprocessors are specified, the tool lists information for all coprocessors in the system.

- *--interleave*

Shows the progress of the update in an output suitable for logs.

8.5.2 Usage

```
[host]# micfw (update [--file <image-file>] [--verbose]|
device-version|file-version <image-file>)
[(-d |--device=)<device-list>]
```

- *update [--file <image-file>] [--verbose] [(-d |--device=)<device-list>]*

Attempt to update the firmware of each specified coprocessor. If the path to an image file is not provided, the first compatible image in default flash directory (OS dependent) will be used.

- *device-version [(-d |--device=)<device-list>]*

Retrieve the firmware version of each specified coprocessor. If the *<device-list>* element is omitted then information regarding all coprocessors is displayed.

- *file-version <image-file>*

Retrieve the firmware version from the specified file.

8.5.3 Examples

1. Retrieving firmware version of all coprocessors in the system.

```
[host]$ micfw device-version
mic0:
   BIOS Version              : GVPRCRB8.86B.0014.D18.1703092150
   ME Version                : 3.2.2.8
   SMC Version               : 121.31.10446
   NTB EEPROM Version        : 0204000D
```

2. Retrieving firmware version in a specified file.

```
[host]$ micfw file-version <image-file>
GVPRCRB8.*.hddimg:
   BIOS                      : GVPRCRB8.86B.0014.D18.1703092150
   ME                        : 3.2.2.8
   SMC Fab A                 : 121.31.10447
   SMC Fab B                 : 121.31.10446
```

3. Updating firmware of two coprocessors using a specified firmware capsule.

```
[host]# micfw update mic0,mic1 --file GVPRCRB8.9899.hddimg
The update process may take several minutes, please be
patient:
Flashing process with file:
   BIOS                      : GVPRCRB8.86B.0014.D18.1703092150
   ME                        : 3.2.2.8
   SMC Fab A                 : 121.31.10447
   SMC Fab B                 : 121.31.10446
mic0: All firmware updates for this coprocessor completed
successfully.
mic1: All firmware updates for this coprocessor completed
successfully.

Summary:
   mic0                      : Successful
   mic1                      : Successful

*** NOTE: You must reboot the host for the KNL changes to take
effect!
```

Note: The cold host system reboot is required to apply all changes. Do not execute any other applications or modify any coprocessor's state while *micfw* is executing.

8.6 micflash

The *micflash* tool allows to update the coprocessor's firmware through the *micfw* tool.

8.6.1 Options

- *-h, --help*

Displays the help documentation for the tool.

- *-v*

Shows extra information about internal operations or messages.

- *-device*

Specifies coprocessors for which data will be retrieved. This option accepts coprocessors' names in form of a comma-separated list, range or a combination of both. For example: *mic0-mic3, mic1,mic3-mic5, mic2*. If no coprocessors are specified, the tool lists information for all coprocessors in the system.

- *-log*

This option is deprecated.

- *-noreboot*

This option is deprecated.

- *-nodowngrade*

This option is deprecated.

- *-smcbootloader*

This option is deprecated.

8.6.2 Usage

```
[host]# micflash [-h] [-getversion [File]][-device <dev>][-v]
[-update [File]][-device <dev>][-v]
```

- *-getversion [-d|-device <device-name>]*

Retrieve the firmware version of each specified coprocessor. If the *<device-name>* element is omitted then information regarding all coprocessors is displayed.

- *-update [file][-d|device <device-name>]*

Attempt to update the firmware of each specified coprocessor. If the path to an image file is not provided, the first compatible image in default flash directory (OS dependent) will be used.

Note: Root privileges are required to flash the coprocessor's firmware.

8.6.3 Examples

Retrieving firmware version of all coprocessors in the system.

```
[host]$ micflash -getversion
   mic0:
      BIOS Version              : GVPRCRB8.86B.0014.D18.1703092150
      ME Version                : 3.2.2.8
      SMC Version               : 121.31.10446
      NTB EEPROM Version        : 0204000D
```

Retrieving firmware version in a specified file.

```
[host]$ micflash -getversion <image-file>
   GVPRCRB8.*.hddimg:
      BIOS                      : GVPRCRB8.86B.0014.D18.1703092150
      ME                        : 3.2.2.8
      SMC Fab A                 : 121.31.10447
      SMC Fab B                 : 121.31.10446
```

Updating firmware of two coprocessors using a specified firmware capsule.

```
[host]# micflash –update -device mic0,mic1 <image-file>
The update process may take several minutes, please be patient:
Flashing process with file:
    BIOS                      : GVPRCRB8.86B.0014.D18.1703092150
    ME                        : 3.2.2.8
    SMC Fab A                 : 121.31.10447
    SMC Fab B                 : 121.31.10446
mic0: All firmware updates for this coprocessor completed
successfully.
mic1: All firmware updates for this coprocessor completed
successfully.

Summary:
mic0                      : Successful
mic1                      : Successful

   *** NOTE: You must reboot the host for the KNL changes to take
   effect!
```

Note: The host system reboot is required to apply all changes. Do not execute any other applications or modify any coprocessor's state while *micflash* is executing.

8.7 micbios

The *micbios* tool allows to change certain coprocessor's BIOS settings, as well as to use the Firmware Update Protection feature without the explicit knowledge of the *syscfg* tool. The coprocessor needs to be rebooted to apply new configuration.

Micbios accepts the following commands:

get-info – read the BIOS configuration value.

set-pass – set up a BIOS password.

set-cluster-mode – modify the memory cluster mode in the coprocessor.

set-fwlock – enable or disable the firmware update lock feature.

set-ecc – enable or disable the error-correcting code.

set-apei-supp – enable or disable the advanced configuration and power interface. This option also enables to see the *Errinject* and *FFM logging* BIOS options.

set-apie-einj – enable or disable error injection.

set-apei-einjtable – enable or disable error injection tables.

set-apei-ffm – enable or disable *FFM*.

Note: Root privileges are required to view all information provided by *micbios.*

8.7.1 Options

- *-h, --help*

Shows the help documentation for the tool.

- *--version*

Shows the version of the tool.

8.7.2 Usage

```
[host]# micbios [-h] [(getinfo|set-pass| \
set-cluster-mode|set-ecc|set-apei-ffm|set-apei-einjtable| \
set-apei-einj|set-fwlock|set-apei-supp)]
```

In the example below *micbios* sets the ecc mode to *'disable'* for coprocessor mic0.

```
[host]# micbios set-ecc disable mic0
Successfully Completed
```

Adding *--help* to each command provides extra information about valid modes.

```
[host]$ micbios set-ecc --help
usage: micbios.py set-ecc [-h] [--password PASSWORD] mode
devices
positional arguments:
  mode                  Mode Options: enable, disable, auto
  devices               Coprocessor's ID
optional arguments:
  -h, --help            show this help message and exit
  --password PASSWORD   The BIOS password must be provided to
change settings state. If not specified '' is the default
```

In the example below *micbios* sets the cluster mode to *'all2all'* for coprocessor mic0.

```
[host]# micbios set-cluster-mode all2all mic0
Successfully Completed
```

Adding *--help* provides more information about the command.

```
[host]$ micbios set-cluster-mode --help
usage: micbios.py set-cluster-mode [-h] [--password PASSWORD]
mode devices

positional arguments:
  mode                  Mode Options: all2all, snc2, snc4,
hemisphere, quadrant, auto
  devices               Coprocessor's ID

optional arguments:
  -h, --help            show this help message and exit
  --password PASSWORD   The BIOS password must be provided to
change settings
                        state. If not specified '' is the default
```

In the example below *micbios* sets the password '*test*' to coprocessor mic0. The default password is empty.

```
[host]# micbios set-pass “” test  mic0
Successfully Completed
```

Add *--help* to get more information about the command.

```
[host]$ micbios set-pass --help
usage: micbios.py set-pass [-h] old_pass new_pass devices

positional arguments:
  old_pass    The BIOS old password, if not specified '' is the
default.
  new_pass    The BIOS setting new password
  devices     Coprocessor's ID

optional arguments:
  -h, --help  show this help message and exit
```

8.8 The libsystools library

The *libsystools* library provides applications running on the host system with a C-library interface to the coprocessor. Elements of the library are wrappers, built on top of the MIC host driver interfaces, that are available in the form of sysfs entries (in Linux*), WMI entries (in Windows*), and scif calls. Various functions are available to query the state of the coprocessor, and list or modify some of its parameters.

8.8.1 Function groups

Library functions are categorized into groups listed below.

1. Identify Available Coprocessors.
2. Error Reporting.
3. Device Access.
4. Query Coprocessor State.
5. General Device Information.
6. PCI Configuration Information.
7. Memory Information.
8. Processor Information.
9. Coprocessor Core Information.
10. Version Information.
11. LED Mode Information.
12. Turbo State Information.
13. SMC Configuration Information.
14. Throttle State Information.
15. Coprocessor OS Power Management Configuration.
16. Thermal Information.
17. Power Utilization Information.
18. Memory Utilization Information.
19. Power Limits.
20. Core Utilization Information.
21. Free Memory Utilities.

All *libsystools* functions return *MICSDKERR_SUCCESS* after successful completion, upon failure they return one of the error codes defined in the *micsdkerrno.h* file.

For additional and detailed information about each of the functions run the following command:

```
[host]$ man libsystools
```

8.8.2 Compiling and running the "examples"

The *libsystools* library is distributed in the *mpss-libsystools0-*.x86_64.rpm* file. There is also additional *mpss-libsystools-devel-*.x86_64.rpm* file, which installs the library documentation and source code of a program called *examples*, which contains several examples of how the library functions can be used.

The source code of the *examples* program is installed in the */usr/share/doc/packages/systools/examples* directory. Run the command below to compile the program.

```
[host]# make -C /usr/share/doc/packages/systools/examples
```

As a result, the *examples* binary file will be created inside the specified directory. Opening the binary file will launch the *examples* program. It can also be run with the *set* parameter to set the coprocessor's power thresholds and LED.

```
[host]# /usr/share/doc/systools/examples/examples set
```

The *Makefile* file inside the *examples* directory shows how a program that uses the *libsystools* library can be compiled.

8.8.3 "Examples" - explanation

The *examples.c* is the main file of the *examples* program, it includes the *libsystools.h* file which is the header file for the library. The *examples'* code is divided into several parts, each using certain functions of the library corresponding to different parts of the coprocessor, for example memory utilization, core utilization, general system information, and other.

Review the sections (or functions) of the examples code to find out how the *libsystools* library can be used to obtain that information from the coprocessor.

Presented below is an output of the *examples* binary executed on a host with a single coprocessor.

```
GENERAL INFORMATION:  mic0
        Device Name          : mic0
        Device Type          : x200

VERSION INFORMATION:  mic0
        Serial Number        : 0123456789AB
        SMC Firmware Version : 121.26.9899
        Coprocessor OS Ver   : 4.1.27-mpss_4.3.0.1265 G
        ME Version           : 3.1.3.2

BOARD INFORMATION:  mic0
   PCI INFORMATION:
        Vendor id                  : 0x8086
        Device id                  : 0x2260
        Revision id                : 0xba
        Subsystem id               : 0x0244
        Bus id                     : 0x0003
        Width                    : x16
        Max payload size         : 256 Bytes
        Max read req size        : 128 Bytes
        Link speed               : 8000 MT/s
   Coprocessor INFORMATION:
        Coprocessor model          : 87
        Coprocessor model Extension : 0
        Coprocessor type           : 0
        Coprocessor stepping id    : 0
        Coprocessor stepping       : A0
        Coprocessor SKU            : SKU Number

UUID INFORMATION: mic0
        UUID: 03020100-0504-0706-0809-0A0B0C0D0E0F

MEMORY INFORMATION:  mic0
        Memory vendor: INTEL
        Memory revision: 0
```

```
        Memory density: 0 Mega-bits/device
        Memory size: 8192 MBytes
        Memory type: DRAM
        Memory tech: MCDRAM
        Memory frequency: 6400 MHz
        Memory speed: 6400 MT/s
        Memory voltage: 0 MilliVolts
        ECC mode: 1

MEMORY UTIL INFORMATION:  mic0
        Total   : 8036352 KB
        Buffers : 2180 KB
        Inuse   : 1375388 KB
        Free    : 6373540 KB

THERMAL INFORMATION:  mic0
        Sensor Die                  : 41 Celsius
        Sensor Fan Exhaust          : 34 Celsius
        Sensor VCCP                 : 19 Celsius
        Sensor VCCCLR               : 31 Celsius
        Sensor VCCMP                : 37 Celsius
        Sensor West                 : 32 Celsius
        Sensor East                 : 33 Celsius

CORE INFORMATION:  mic0
        Total No. of Active Cores : 60
        Threads per Core          : 4
        Core Voltage              : 900 mV
        Core Frequency            : 1200 MHz

CORE UTIL INFORMATION:  mic0
        Jiffy counter : 405838555
        Tick counter : 100
        <output truncated>
        Idle   counter Sum : 405748030
        Nice   counter Sum : 0
        System counter Sum : 60754
        User   counter Sum : 30169

POWER INFORMATION:  mic0
        hmrk         : 360000000 uW
        lmrk         : 300000000 uW
        time win0    : 999424 us
        time win1    : 9760 us

POWER UTIL INFORMATION:  mic0
        Sensor PCIe       : 20.000 W
        Sensor 2x3        : 52.000 W
        Sensor 2x4        : 34.000 W
        Sensor Average 0  : 105.000 W
        Sensor Current    : 106.000 W
        Sensor Maximum    : 143.000 W
```

```
        Sensor VCCP       : 18.000 W
        Sensor VCCU       : 4.000 W
        Sensor VCCCLR     : 30.000 W
        Sensor VCCMLB     : 16.000 W
        Sensor VCCD012    : 55.000 W
        Sensor VCCD345    : 55.000 W
        Sensor VCCMP      : 2.000 W
        Sensor NTB1       : 4.000 W

TURBO INFO:  mic0
        Turbo Mode Supported  : True
        Turbo Mode Enabled    : False
        Turbo Mode Active     : False

LED INFORMATION:  mic0
        LED Alert Set: False

***** Starting Set Functions Examples! *****
LED INFORMATION BEFORE UPDATE: mic0. Current Value: 0;
Updating to: 1
LED INFORMATION AFTER UPDATE:  mic0
LED INFORMATION:  mic0
        LED Alert Set: True
POWER INFORMATION BEFORE UPDATE:  mic0. Current values: P0:
300000000; T0: 999424. Updating to: P0: 305000000; T0: 1499424

POWER INFORMATION AFTER UPDATE:  mic0
POWER INFORMATION:  mic0
        hmrk          : 360000000 uW
        lmrk          : 305000000 uW
        time win0     : 1499136 us
        time win1     : 9760 us
POWER INFORMATION BEFORE UPDATE:  mic0. Current values: P1:
360000000; T1: 9760. Updating to: P1: 355000000; T1: 8760
POWER INFORMATION AFTER UPDATE:  mic0
POWER INFORMATION:  mic0
        hmrk          : 355000000 uW
        lmrk          : 300000000 uW
        time win0     : 999424 us
        time win1     : 7808 us
***** Ending Set Functions Examples! *****
```

§

A Intel® MPSS configuration parameters

This section describes the parameters in the */etc/mpss/default.conf* and */etc/mpss/micN.conf* configuration files. The parameter syntax in the following sections sometimes extends to more than one line, however, each parameter in the actual configuration files must not contain a newline until the end of the entry.

As mentioned in Section 4, any parameter can be present in the *default.conf* file or in any *micN.conf* file. A parameter in the *default.conf* will apply to all coprocessors in the system, unless the same parameter is present in the *micN.conf* file (coprocessor-specific configuration files take precedence over *default.conf*).

Reboot the coprocessor(s) to apply configuration changes.

A.1 Meta configuration

A.1.1 Including other configuration files

Parameter Syntax (*micN.conf*):

```
Include <config_file_name>
```

Each configuration file can include other configuration files. The *Include* parameter lists the configuration file(s) to be included. The configuration file(s) to be included must reside in the */etc/mpss* directory. The configuration parser processes each parameter sequentially. When the *Include* parameter is encountered, each included configuration file is immediately processed. If a parameter is set multiple times, the last instance of the parameter setting will be applied.

By default, the *default.conf* file is included at the start of each *micN.conf* file. This allows the coprocessor specific files to override any parameter set in *default.conf*.

A.2 Boot control

Parameters listed in this section control the coprocessor's boot process. They are located in the *micN.conf* file.

A.2.1 EfiImage

Parameter Syntax:

```
EfiImage <target>
```

Initial value:

```
EfiImage /usr/share/mpss/boot/efiImage-knl-lb.hddimg
```

The *EfiImage* parameter specifies the location of the coprocessor OS EFI image.

A.2.2 KernelImage

Parameter Syntax:

```
KernelImage <linux_kernel_image>
```

Initial value:

```
KernelImage /usr/share/mpss/boot/bzImage-knl-lb
```

The *KernelImage* parameter specifies the location of the coprocessor OS boot image.

A.2.3 KernelSymbols

Parameter Syntax:

```
KernelSymbols <system_address_map_file>
```

Initial value:

```
KernelSymbols /usr/share/mpss/boot/system.map-knl-lb
```

The *KernelSymbols* parameter specifies the location of the coprocessor OS address map file.

A.2.4 InitRamFsImage

Parameter Syntax:

```
InitRamFsImage <target>
```

Initial value:

```
InitRamFsImage /usr/share/mpss/boot/initramfs-knl-lb.cpio.gz
```

The *InitRamFsImage* parameter specifies the file system hierarchy in the initial coprocessor's file system. The <*target*> argument is interpreted as the file name of a CPIO file system archive.

A.2.5 AutoBoot

Parameter Syntax:

```
AutoBoot (Enabled|Disabled)
```

Initial value:

```
AutoBoot Enabled
```

The *AutoBoot* parameter controls whether the coprocessor is booted when the *mpss* service starts. The default value of this parameter is *Enabled*, which means, the *mpssd* daemon will attempt to boot the coprocessor when *systemctl start mpss* is called.

A.3 File system configuration parameters

Parameters listed in this section configure the coprocessor's file system. They are located in the *micN.conf* file.

A.3.1 RootFsImage

Parameter Syntax:

```
RootFsImage <file system file location>
```

Initial value:

```
RootFsImage /var/mpss/persistent-FS-knl-lb-mic0.ext4
```

The *RootFsImage* parameter describes a special virtual block device named *root* which is backed by a file containing the root file system for the coprocessor OS.

Note: The coprocessor will fail to boot if this parameter is not specified.

A.3.2 RepoFsImage

Parameter Syntax:

```
RepoFsImage <file system file location>
```

Initial value:

```
RepoFsImage /usr/share/mpss/boot/repo-card.squashfs
```

The *RepoFsImage* parameter describes a special virtual block device named *repo* which is backed by a squashfs repository image for the coprocessor OS. When defined, a block device will be created and automatically mounted in the */usr/share/mpss/packages/* directory on the coprocessor. This path will be added as a local repository for the coprocessor OS and will be available through the *smart* package manager.

A.3.3 BlockDevice

Parameter Syntax:

```
BlockDevice name=<block name> mode=[rw|ro] path=<file system
file location>
```

The *BlockDevice* parameter describes virtual block devices that will be used on the coprocessor OS as standard block devices (for example as hard disk drives). Each virtual block device must be backed by a file on the host. 6 additional block devices can be specified with this parameter. Each specified device will be available on the coprocessor OS as a */dev/vd[a-z]* device node.

Additionally, */dev/mpss_maper/<block name>* symbolic links are created in the coprocessor OS allowing users to easily access the block devices by their specified names. For example, a block device named *storage1* can be accessed by */dev/mpss_mapper/storage1* symbolic link.

Note: A virtual block device cannot be shared by multiple coprocessors. Each coprocessor must use its individual file system. Note, that the coprocessor's boot process will fail if no file is present at the *<file system file location>*.

Note: Names *root* and *repo* are restricted for virtual block devices used by the *RootFsImage* and *RepoFsImage* parameters.

A.4 Kernel configuration

Parameters listed in this section influence or control the execution of the coprocessor's Linux* kernel through values passed to the kernel in the startup command line and configure the timeout value of coprocessor shutdown. They are located in the *default.conf* file.

A.4.1 KernelExtraCommandLine

Parameter Syntax:

```
KernelExtraCommandLine "<kernel_parameters>"
```

The *KernelExtraCommandLine* parameter specifies kernel command line parameters to be passed to the coprocessor's kernel on boot. Drivers for the coprocessor use a number of kernel command line parameters generated by the host driver. Default parameters may be subject to change in future releases.

A.4.2 ShutdownTimeout

Parameter Syntax:

```
ShutdownTimeout <value>
```

Initial value:

```
ShutdownTimeout 300
```

Setting the value to a positive integer specifies the maximum number of seconds to wait for the coprocessor to shut down. If the shutdown time exceeds the value, the coprocessor is reset.

Setting the value to any negative integer indicates to wait indefinitely for the coprocessor to shut down.

Setting the value to zero indicates to reset the coprocessor without waiting for it to shut down.

A.5 Network configuration

Parameters in this section control the network configuration of the coprocessor. The *HostNetworkConfig* and *CardNetworkConfig* parameters are located in the *micN.conf* configuration files. The *HostMacAddress* and *CardMacAddress* parameters are not created automatically, the administrator may place them manually in *micN.conf* files to change the default method of assigning MAC addresses.

A.5.1 Host network configuration

Parameter Syntax:

```
HostNetworkConfig ((inet|inet6) static address=<host IP> \
netmask=<host netmask> | bridge <bridge name> | none )
```

Initial value:

```
HostNetworkConfig inet static address=172.31.N+1.254 \
netmask=255.255.255.0
```

Description:

The *inet6* value is used in IPv6 networks.

The default host network configuration is static. The IP address and netmask are assigned to the micN interface when it is created.

Changing the network configuration to *"bridge"*, connects the micN interface to the bridge described as <*bridge_name*> at the time of the interface creation.

When the host network configuration is set to *"none"*, *micctrl* and *mpssd* perform no operations. The IP and netmask addresses are not assigned to the micN interface. To enable connection between the host and the coprocessor, create network configuration file manually (for example by creating and editing the */etc/sysconfig/network-scripts/ifcfg-mic0* file in RHEL* 7.x).

A.5.2 Coprocessor network configuration

Parameter Syntax:

```
CardNetworkConfig ((inet|inet6) static address=<coprocessor IP> \
netmask=<coprocessor netmask> gateway=<coprocessor gateway> | \
(inet|inet6) dhcp | none )
```

Initial value:

```
CardNetworkConfig inet static address=172.31.N+1.1 \
netmask=255.255.255.0 gateway=172.31.N+1.1
```

Description:

The *inet6* value is used in IPv6 networks.

The default configuration is static. The coprocessor's IP and netmask addresses are assigned during each boot.

If the configuration is changed to *inet dhcp*, the DHCP server will assign the IP, netmask and gateway addresses.

A.5.3 MAC address assignment

Parameter syntax:

```
HostMacAddress (Serial|Random|<host MAC>)
CardMacAddress (Serial|Random|<coprocessor MAC>)
```

Description:

MAC addresses must be generated for the virtual network interfaces of the host and each coprocessor. By default, MAC addresses are generated based on the serial number of the devices. If the device does not have a usable serial number, MAC address is generated randomly.

The least significant bit is set in MAC addresses generated for the host endpoints, and not set in MAC addresses generated for the coprocessor endpoints. In addition, the top three octets of the generated MAC addresses have the IEEE assigned value *E4:FA:FD*, which enables identification of coprocessors' interfaces.

The system administrator may override the default *Serial* behavior by explicitly setting the *HostMacAddress* and *CardMacAddress* parameters in the *micN.conf* files. If the *Random* string is provided, a random address will generated overriding the serial number-based one. It is also possible to explicitly provide MAC addresses of the host and coprocessor network endpoints, for example:

```
CardMacAddress 00:1E:67:56:00:01
```

§

B Intel® MPSS host driver sysfs entries

The host driver supplies configuration and control information to the host software through the Linux* Sysfs file system. The driver presents coprocessor-unique information in */sys/class/mic/micN* directories.

B.1 Hardware information

Sysfs Entries:

*/sys/class/mic/micN/info/**

These sysfs entries describe coprocessor's hardware details. All of these entries are taken from coprocessor's DMI table.

The *mic_family* file can be used to detect if coprocessor belongs to the x200 product family.

The *mcdram_size* and *mcdram_version* files can be used to get information about coprocessor's MCDRAM memory.

B.2 State entries

Sysfs Entries:

/sys/class/mic/micN/state

/sys/class/mic/micN/state_msg

The *state* file is read/write, and the *state_msg* file is read only.

On reading, the *state* file reports one of the following values:

- *ready* coprocessor is ready for a boot command
- *booting* coprocessor is booting
- *online* coprocessor is booted
- *shutting_down* coprocessor is processing a shutdown
- *shutdown* coprocessor is shut down
- *resetting* coprocessor is resetting
- *error* coprocessor failed and needs to be reset
- *booting_firmware* coprocessor is booting in firmware update mode
- *online_firmware* coprocessor is booted in firmware update mode

On reading, *state_msg* file can report additional information (in human readable form) about coprocessor's current state (e.g. when coprocessor enters the *error* state).

Writing to the state file requests the driver to initiate a change in coprocessor's state. The following requests are available:

- *boot* boots the coprocessor
- *boot_firmware* boots the coprocessor in firmware update mode
- *reset* resets the coprocessor
- *reset_force* forcefully resets the coprocessor
- *shutdown* shuts down the coprocessor
- *shutdown_force* forcefully shuts down the coprocessor

B.3 Configuration entries

Sysfs Entries:

/sys/class/mic/micN/config/boot_timeout

/sys/class/mic/micN/config/shutdown_timeout

/sys/class/mic/micN/config/efi_image

/sys/class/mic/micN/config/execute_on_ready

/sys/class/mic/micN/config/initramfs_image

/sys/class/mic/micN/config/kernel_image

/sys/class/mic/micN/config/kernel_cmdline

/sys/class/mic/micN/config/log_buf_addr

/sys/class/mic/micN/config/log_buf_len

The *efi_image* and *initramfs_image* files contain paths to files containing coprocessor's EFI image and initramfs images in relation to the host's */lib/firmware/* directory.

The *boot_timeout* and *shutdown_timeout* files can be used to modify coprocessor's boot and shut down timeout.

The *execute_on_ready* file specifies the command identifier to be executed once the state of coprocessor changes to *ready*.

The *log_buf_addr* and *log_buf_len* files inform the host driver of the memory address in the coprocessor's memory at which to read its Linux* kernel log buffer. The values are found by looking at the *log_buf_addr* and *log_buf_len* strings in the Linux* system map file, which is specified by the *KernelSymbols* parameter in each */etc/mpss/micN.conf* configuration file, and are typically set by the *mpss* daemon.

The *kernel_cmdline* file is used to pass kernel command line parameters to the coprocessor's Linux* boot process. Current parameters include root file system, console device information, power management options and verbose parameters.

B.4 Debug entries

Sysfs Entries:

/sys/class/mic/micN/spad/post_code

/sys/class/mic/micN/spad/spad#

/sys/class/mic/micN/log_buf

The *log_buf, post_code,* and *spad#* files are read-only.

The *post_code* sysfs file returns the contents of the hardware register containing the state of the boot loader code. Upon reading it always returns four ASCII characters. The most common values of note are *'0x7D'* (coprocessor is in the *ready* state), *'0xFF'* (coprocessor is booted), *'0xB9'* (coprocessor is in the shutdown mode) and *'0xE0'* (coprocessor is in the firmware update mode). Any other value remaining for any length of time may indicate an error.

The *spad#* file returns the current state of the coprocessor's SPAD registers.

The *log_buf* file contains coprocessor's Linux* kernel log in a binary form. This log can be retrieved by issuing the *micctrl -l* command.

§

C Intel® MPSS optional components

This section provides detailed instructions on installing optional Intel® MPSS components.

C.1 Performance workloads

The Performance Workloads (*micperf*) component of Intel® MPSS can be used to evaluate the performance of an Intel® Xeon Phi™ coprocessor x200 based installation.

Note: The *Micperf User's Guide* (*$MPSS4/doc/micperf_users_guide.pdf*) contains detailed information on using the *micperf* software.

C.1.1 Installation requirements

1. **Intel® Composer XE:**

   Install the full Intel® Composer XE package and source the *compilervars.sh* or *compilervars.csh* scripts at run time.

   If the full composer installation is not available, then two packages can be used instead. The required shared object libraries can be installed via the Intel® Composer XE redistributable package, freely distributed on the web at:

   http://software.intel.com/en-us/articles/redistributable-libraries-for-the-intel-c-and-fortran-composer-xe-2013-sp1-for-linux

   This package uses the *install.sh* script for installation. After installation *compilervars.sh* and *compilervars.csh* scripts serve a similar purpose to those scripts in the full Intel® Composer XE distribution and must be sourced at run time.

2. **MKL linpack:**

   Besides the shared object libraries, the *MKL linpack* benchmark is also required. It is freely distributed on the web at:

   http://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download

   This download is a tarball that can be unpacked anywhere, however, the environment variable *MKLROOT* must point to the top level directory of the untarred package. For instance, if the user extracted the tarball into their home directory they should set *MKLROOT* as follows:

   ```
   [host]$ export MKLROOT=$HOME/linpack_<version_num>
   ```

If *MKLROOT* is set in the user's shell environment at run time, then *micprun* will be able to locate the *linpack* binaries.

3. ***MATPLOTLIB*:**

   The *micpplot* and *micprun* applications use the *MATPLOTLIB* Python module to plot performance statistics. The *micprun* application only creates plots when verbosity is set to two or higher and requires *MATPLOTLIB* for this use case. *MATPLOTLIB* must be installed in order to create plots. Download it from http://matpmiclotlib.sourceforge.net/.

C.1.2 RPM installation

The *micperf* package is distributed in one RPM file:

```
$MPSS4/<OS version>/x86_64/core/mpss-micperf-4.4.1*.rpm
```

To install *micperf* use the command below.

In RHEL*:

```
[host]# yum install micperf
```

In SLES*:

```
[host]# zypper install micperf
```

This installs files to the following directories:

- Source code: */usr/src/micperf*
- Documentation and licenses: */usr/share/doc/packages/micperf*
- Benchmark binaries: */usr/libexec/micperf*
- Reference data: */usr/share/micperf/micp*
- Links to executables: */usr/bin*

C.1.3 Python installation

Once the RPM package has been installed, an additional step must be executed to access the *micp* Python package: either install it to your global Python site packages, or set up your environment to use the *micp* package from the installed location.

To perform installation into the Python site packages execute:

```
[host]$ cd /usr/src/micperf/micp
[host]# python setup.py install
```

This method provides access to the *micp* package and executable scripts to all non-root users who use the same Python version as the root user. If Python is in the default location and uses a standard configuration, *setup.py* installs the *micp* package to the following directories:

/usr/bin

/usr/lib/pythonPYVERSION/site-packages/micp

An intermediate product of running *setup.py install* is the creation of the directory:

/usr/src/micperf/micp-<version>/build

None of the products of running *setup.py* discussed above will be removed by uninstalling the *micperf* RPMs. The installation with *setup.py* uses Python's distutils module which does not support uninstall. If installed on a Linux* system where Python is configured in a standard way, it should be possible to uninstall it with the commands below.

```
[host]# sitepackages=`sudo python -c \
   "from distutils.sysconfig import get_python_lib; \
   print(get_python_lib())"`
[host]# rm -rf /usr/src/micperf/micp/build \
/usr/bin/micpcsv \
/usr/bin/micpinfo \
/usr/bin/micpplot \
/usr/bin/micpprint \
/usr/bin/micprun \
${sitepackages}/micp \
${sitepackages}/micp-[version number]*
```

C.1.4 Alternative to python installation

Another way to access the *micp* package after installing the *micperf* RPM is to alter the user's shell run time environment. Set up the bash or Bourne shell environment by executing the command below.

```
[host]$ export PYTHONPATH=/usr/src/micperf/micp:${PYTHONPATH)
```

Set up your *csh* run time environment with the following command.

```
[host]$ setenv PYTHONPATH /usr/src/micperf/micp:${PYTHONPATH}
```

C.2 Syscfg and Firmware Update Protection

Firmware Update Protection (FUP) is a feature that prevents unauthorized firmware updates by denying all firmware update attempts. This way it can prevent even root users such as root users on virtual machines from performing an upgrade.

By default all firmware updates (BIOS, SMC and ME) are allowed. Figure 12 shows the firmware update process using the FUP feature.

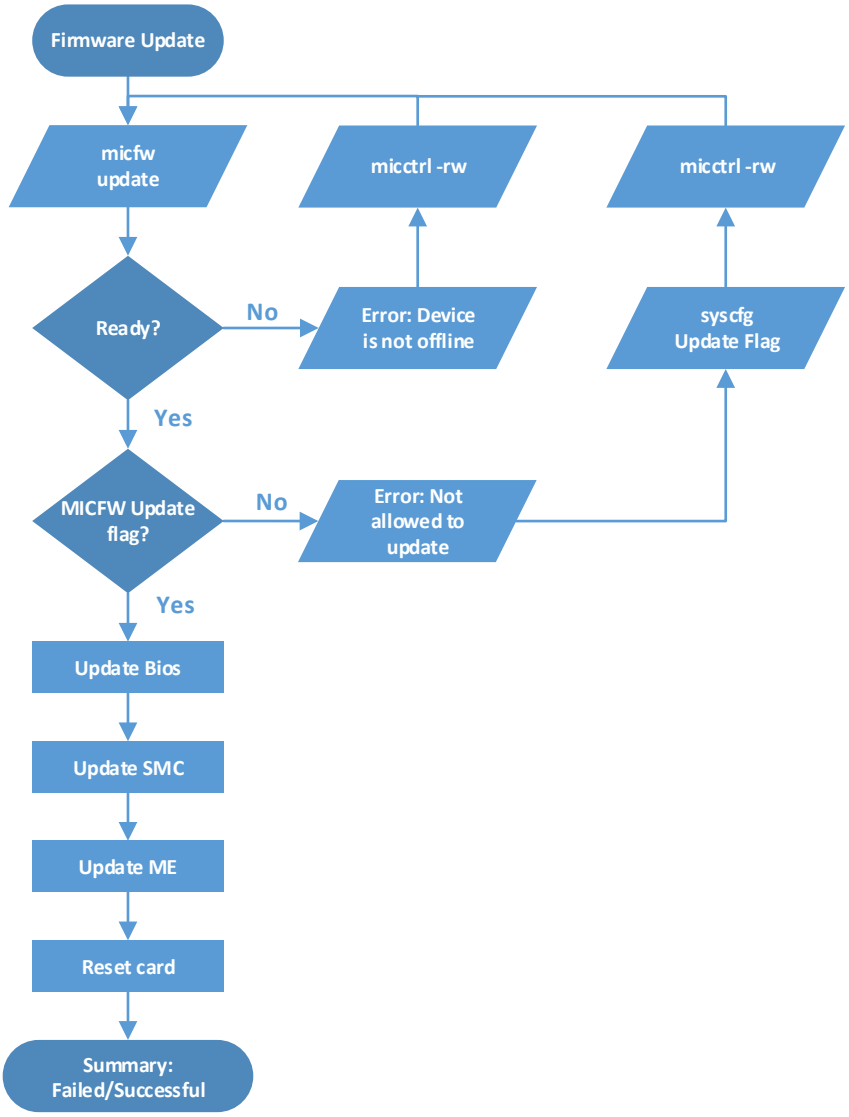

Figure 12 Firmware update workflow with the FUP feature

FUP is managed by *syscfg*, which is a command line tool in the coprocessor's file system that can be used for saving and configuring firmware and BIOS settings. The tool can only be used if coprocessor is in the *online* state and after logging in to it as root.

```
[host]# micctrl -bw
[host]# ssh root@micN
[micN]# syscfg
System Configuration Utility Version 14.0 Build CRB_Build4
        Copyright (c) 2016 Intel Corporation
```

C.2.1 Enabling FUP

The command below enables the *"MICFW Update Flag"* BIOS parameter. Coprocessor reset is required to apply new settings.

```
[micN]# syscfg -bcs <Admin_PW> "MICFW Update Flag" 01
[host]# micctrl -rw
```

The *<Admin_PW>* element indicates the administrator password, if the password is not set use *""* as in the example command below.

```
[micN]# syscfg -bcs "" "MICFW Update Flag" 01
[host]# micctrl -rw
```

C.2.2 Disabling FUP

The command below disable the *"MICFW Update Flag"* BIOS parameter. Coprocessor reset is required to apply new settings.

```
[micN]# syscfg -bcs <Admin_PW> "MICFW Update Flag" 00
[host]# micctrl -rw
```

The *<Admin_PW>* element indicates the administrator password, if the password is not set use *""* as in the example command below.

```
[micN]# syscfg -bcs "" "MICFW Update Flag" 00
[host]# micctrl -rw
```

C.2.3 Querying FUP status

The command below retrieves the current status of the Firmware Update Protection.

```
[micN]# syscfg -d biossettings "MICFW Update Flag"

MICFW Update Flag
=================
Current Value : Enable
----------------------
Possible Values
---------------
Disable : 00
Enable : 01
```

C.2.4 Changing BIOS password

The command below to changes coprocessor's BIOS password.

```
[micN]# syscfg -bap <Admin_old_PW> <Admin_new_PW>
```

The *<Admin_old_PW>* element indicates the old administrator password, use *""* if the password is not set. The *<Admin_new_PW>* is case-sensitive, can have maximum length of 14 characters, and can contain digits and following special characters: *! @ # $ % ^ & *( ) - _ + = ?* .

To clear the password, specify a new one as "".

It is not possible to recover BIOS password in case it is lost.

Example:

```
[micN]# syscfg -bap "" "admin@123"
Successfully Completed
```

C.2.5 Constraints

- *Syscfg* can only be used directly on each coprocessor. The administrator has to boot and log in to each coprocessor in order to execute *syscfg*.
- *Micfw* does not have direct access to *syscfg*, and cannot modify BIOS settings. Moreover, *micfw* requires coprocessors to be in the *ready* state before performing firmware updates.

§

D Compiling software for the coprocessor

Instructions in this appendix show how to compile components of the software stack from source and how to build custom software for use on the coprocessor.

D.1 Compiling custom software for the coprocessor

Use the SDK component of the software stack to compile custom software for use on the coprocessor. Follow the instructions below.

```
[host]$ cd <extracted_source_code>
[host]$ source /opt/mpss/x200/environment-setup-x86_64-poky-linux
[host]$ make
```

D.2 Compiling Intel® MPSS kernel modules from source

Source code for the Intel® MPSS kernel modules is distributed in the *mpss-4.4.1-card-source.tar* archive. Extract the archive to */tmp/mpss-modules-source* and follow the instructions below to compile the modules from source.

1. Build the kernel modules.

```
[host]$ cd /tmp/mpss-modules-source
[host]$ source /opt/mpss/x200/environment-setup-x86_64-poky-linux
[host]$ make BUILD_ROOT=/tmp/mpss-modules modules
[host]$ make BUILD_ROOT=/tmp/mpss-modules modules_install
```

2. Unpack *initramfs*.

```
[host]$ mkdir –p /tmp/mpss-initramfs
[host]$ cd /tmp/mpss-modules
[host]$ zcat /usr/share/mpss/boot/initramfs-knl-lb.cpio.gz | \
cpio -i -d -H newc --no-absolute-filenames
```

3. Update the modules.

```
[host]$ cp –a /tmp/mpss-modules/* /tmp/mpss-initramfs/
```

4. Repack *initramfs.*

```
[host]$ find . | cpio –o –H newc | gzip > \
/usr/share/mpss/boot/initramfs-knl-lb.cpio.gz
```

D.3 Rebuilding Intel® MPSS host driver (optional)

Intel® MPSS includes the source code for the host driver. It is possible to modify and recompile the driver if needed. For example, if a new host Linux* kernel was installed, a corresponding Intel® MPSS host driver can be compiled and installed.

Extract the *mpss-4.4.1-<OS version>.tar* file prior to proceeding with instructions described in the following sections.

D.3.1 Rebuilding the host driver in RHEL*

1. Ensure the prerequisites (*kernel-devel, rpm-build, GCC*) are installed.

   ```
   [host]# yum install kernel-devel rpm-build gcc
   ```

2. Regenerate the Intel® MPSS driver module package. The source RPM files are included in the *mpss-4.4.1/<OS version>/source* directory.

   ```
   [host]$ cd mpss-4.4.1/<OS version>/source/
   [host]# rpmbuild --rebuild mpss-modules-*.src.rpm
   ```

3. The regenerated mpss-modules binary RPMs are located in *$HOME/rpmbuild/RPMS/x86_64*. Copy them to *$MPSS4/<OS version>/x86_64/core*.

   ```
   [host]$ cd $HOME/rpmbuild/RPMS/x86_64
   [host]$ cp mpss-modules*.rpm $MPSS4/<OS version>/x86_64/core
   ```

4. Install the regenerated packages.

   ```
   [host]# yum install mpss-modules*$(uname -r)*.rpm
   ```

5. Load the kernel modules.

   ```
   [host]# micctrl --load-modules
   ```

D.3.2 Rebuilding the host driver in SLES*

1. Ensure the prerequisites are installed.

   ```
   [host]# zypper install kernel-default-devel
   ```

2. Regenerate the Intel® MPSS driver module package. The source RPM files are included in the *$MPSS4/<OS version>/source* directory.

   ```
   [host]$ cd $MPSS4/<OS version>/source/
   [host]# rpmbuild --rebuild mpss-modules*.src.rpm
   ```

3. The regenerated *mpss-modules* binary RPMs are located in */usr/src/packages/RPMS/x86_64*. Copy them to *$MPSS4/<OS version>/x86_64/core*.

   ```
   [host]$ cd /usr/src/packages/RPMS/x86_64
   [host]$ cp mpss-modules*.rpm $MPSS4/<OS version>/x86_64/core
   ```

4. Install the regenerated packages.

   ```
   [host]# zypper install mpss-modules*$(uname -r)*.rpm
   ```

5. Load the kernel modules.

   ```
   [host]# micctrl --load-modules
   ```

D.4 Rebuilding Intel® MPSS System tools packages

Intel® MPSS includes source code for the *System tools* packages. It is possible to modify and recompile these packages if needed. For example, the tools can be adapted to support a different environment.

Extract the *mpss-4.4.1-<OS version>.tar* file prior to proceeding with instructions described in the following sections.

1. Ensure the prerequisites are installed (*mpss-libscif, mpss-libscif-devel, mpss-release*). The RPM files are included in the *$MPSS4/packages/x86_64/core* directory. Additionally, make sure that *doxygen 1.8.11* or newer is installed in your host system.

   ```
   [host]$ cd $MPSS4/packages/x86_64/core
   ```

   In RHEL*:

   ```
   [host]# yum install mpss-libscif*.rpm mpss-release*.rpm,
   ```

   In SLES*:

   ```
   [host]# zypper install mpss-libscif*.rpm mpss-release*.rpm
   ```

   Regenerate driver module package. The source RPM files are included in the *$MPSS4/packages/source* directory.

   ```
   [host]$ cd $MPSS4/package/source
   [host]# rpmbuild --rebuild mpss-systools*src.rpm
   ```

2. The regenerated *mpss-systools* binary RPMs are located in the *~/rpmbuild/RPMS/x86_64* directory.

   ```
   [host]$ cd ~/rpmbuild/RPMS/x86_64
   ```

3. Install the regenerated packages.

   In RHEL*:

   ```
   [host]# yum install mpss-systools-*.rpm mpss-libsystools*.rpm
   ```

   In SLES*:

   ```
   [host]# zypper install mpss-systools-*.rpm mpss-libsystools*.rpm
   ```

D.5 Compiling MYO binaries for the coprocessor

Instructions in this section show how to compile MYO binary files from source and install them on a coprocessor.

Note: Before following the instructions below, make sure Intel® Compiler is installed and configured on your system.

1. Download the *mpss-4.4.1-card-source.tar* archive and extract it.

```
[host]$ tar xvf mpss-4.4.1-card-source.tar
```

2. Extract MYO source code.

```
[host]$ cd mpss-4.4.1/sources/card/sources
[host]$ mkdir myo-src
[host]$ tar xf myo-1.0.tar.xz –C myo-src
```

3. Build MYO binary files.

```
[host]$ cd myo-src/src
[host]$ make TARGET_ICC=icc
```

4. Once built, the binary files are located in the *bin* subdirectory. Use *scp* to copy the binaries to the coprocessor (copying to mic0 shown).

```
[host]# scp myo-src/bin/libmyo*service* 172.31.1.1:/usr/lib64
```

D.6 Compiling COI binaries for the coprocessor

Instructions in this section show how to compile COI binary files from source and install them on a coprocessor.

1. Download the *mpss-4.4.1-card-source.tar* archive and extract it.

```
[host]$ tar xvf mpss-4.4.1-card-source.tar
```

2. Extract COI source code.

```
[host]$ cd mpss-4.4.1/sources/card/sources
[host]$ mkdir coi-src
[host]$ tar xf coi-1.0.tar.xz –C coi-src
```

3. Build COI binary files.

```
[host]$ cd coi-src
[host]$ source /opt/mpss/x200/environment-setup-x86_64-poky-linux
[host]$ make -j 72 libdir=/usr/lib proj_target=knllb_card KNL=1 \
WHAT=DEV
```

4. Once built, the binary files are located in the *build* subdirectory. Use *scp* to copy the binaries to the coprocessor (copying to mic0 shown).

```
[host]# scp build/libcoi_device.so \
172.31.1.1:/usr/lib64/libcoi_device.so.0
[host]# scp build/coi_daemon 172.31.1.1:/usr/bin/coi_daemon
```

D.7 Compiling SCIF library for the coprocessor

Instructions in this section show hot to compile SCIF library and *man* pages from source.

Note: Install *asciidoc* on your host system prior to following the instructions below.

1. Download the *mpss-4.4.1-card-source.tar* archive and extract it.

```
[host]$ tar xvf mpss-4.4.1-card-source.tar
```

2. Extract SCIF source code.

```
[host]$ cd mpss-4.4.1/sources/card/sources
[host]$ mkdir libscif-src
[host]$ tar xf libscif-4.4.1.tar.xz –C libscif-src
```

3. Build the SCIF library.

```
[host]$ cd libscif-src
[host]$ source /opt/mpss/x200/environment-setup-x86_64-poky-linux
[host]$ make
```

4. Once built, the binary files are located in the *build* subdirectory. Use *scp* to copy them to the coprocessor (copying to mic0 shown).

```
[host]# scp build/libscif.so \
172.31.1.1:/usr/lib/libscif.so.0.0.1
[mic0]$ ln –s /usr/lib/libscif.so.0.0.1 /usr/lib/libscif.so.0
```

D.8 Compiling the systools daemon for the coprocessor

Instructions in this section show how to compile the systools daemon (*systoolsd*) from source and install it on a coprocessor.

1. Download the *mpss-4.4.1-card-source.tar* archive and extract it.

```
[host]$ tar xvf mpss-4.4.1-card-source.tar
```

2. Extract *systoolsd* source code.

```
[host]$ cd mpss-4.4.1/sources/card/sources
[host]$ mkdir systoolsd-src
[host]$ tar xf systoolsd-1.0.tar.xz -C systoolsd-src
```

3. Build the *systoolsd* binary file.

```
[host]$ cd systoolsd-src/
[host]$ source /opt/mpss/x200/environment-setup-x86_64-poky-linux
[host]$ make -C apps/systoolsd bin/systoolsd
```

4. Once built, the binary file is located in the *apps/systoolsd/bin* subdirectory. Use *scp* to copy the binary to the coprocessor (copying to mic0 shown).

```
[host]# scp apps/systoolsd/bin/systoolsd \
172.31.1.1:/usr/sbin/systoolsd
```

§

E Troubleshooting and debugging

This appendix is a collection of tips and techniques that can be helpful in troubleshooting an Intel® Xeon Phi™ coprocessor x200 installation and/or debugging the execution on the coprocessor.

E.1 Log files

The coprocessor supports *BusyBox* implementations of *dmesg* and *syslogd*.

Inspecting the *dmesg* output can sometimes help in troubleshooting when a coprocessor fails to boot. Coprocessor *dmesg* output can be viewed during coprocessor boot (or later) by issuing:

```
[host]# micctrl -l
```

By default, each coprocessor's syslog messages are logged to the coprocessor's */var/log/messages* file.

E.2 The mcelog package

The coprocessor OS supports Machine Check Events detection, which allows to log and handle hardware errors. The *mcelog* package is pre-installed by default in the coprocessor's file system and additionally included in the *mpss-4.4.1-card.tar* archive.

Run the command below for details on how to use the tool.

```
[micN]$ mcelog –h
```

Man page for *mcelog* is not available on the coprocessor, but can be viewed on the host.

```
[host]$ man mcelog
```

E.3 Installing Intel® MPSS debug information

The *$MPSS4/<OS version>/x86_64/debug* directory contains debug packages for several software stack components. The packages can be installed by following the instructions below.

In RHEL*:

```
[host]# cd $MPSS4/<OS version>/x86_64/debug
[host]# yum install *.rpm
```

In SLES*:

```
[host]# cd $MPSS4/<OS version>/x86_64/debug
[host]# zypper install *.rpm
```

Note: In the current release debug information for SLES* is stored in main packages.

The */usr/lib64/.debug, /usr/bin/.debug* and */usr/sbin/.debug* directories contain the debug symbols.

E.4 Gnu debugger (GDB) for the coprocessor

GDB can be used to debug applications on a coprocessor. GDB supports both native execution on a coprocessor as well as remote execution from host. The *GDB User Manual* is available at http://www.gnu.org/software/gdb/documentation/. It provides detailed instructions on the use of GDB. This section presents some additional information on using GDB on the coprocessor.

E.4.1 Running natively on the coprocessor

To execute GDB natively, the *gdb-7.9.1-r0.x86_64.rpm* and *gdbserver-7.9.1-r0.x86_64.rpm* packages must be installed into the coprocessor's file system.

E.4.2 Running remote GDB on the coprocessor

The remote coprocessor enabled GDB client is located on the host in */opt/mpss/x200/sysroots/x86_64-pokysdk-linux/usr/bin/x86_64-poky-linux/x86_64-poky-linux-gdb*

The GDB Server is pre-installed in the coprocessor's file system by default at */usr/bin/gdbserver*

For complete GDB remote debugging instructions, refer to the *"Debugging Remote Programs"* chapter in the *GDB User Manual.*

E.4.3 GDB remote support for data race detection

GDB supports data race detection based on the Intel® PDBX data race detector for Intel® MIC architecture. See the *"Debugging data races"* chapter in the GDB manual.

Ensure that the environment is set up correctly and that GDB finds the correct version of the Intel® Compiler's run-time support libraries. See the *PROBLEMS-INTEL* file in the GDB source package for additional help on troubleshooting.

E.4.4 Enabling mic GDB debugging for offload processes

An environment variable must be set in order to allow the debugger to enable module name mapping with the generated files needed to attach offload processes to the coprocessor side. To do this, execute the following command:

```
[host]$ export AMPLXE_COI_DEBUG_SUPPORT=TRUE
```

E.5 Enabling core-dump

Consult the instructions below to allow the OS to collect core-dump (e.g. in case *mpssd* or *micctrl* crash).

Temporary configuration:

```
[host]# echo '|/usr/lib/systemd/systemd-coredump \
%p %u %g %s %t %e' > /proc/sys/kernel/core_pattern
[host]# ulimit –c unlimited
```

Persistent configuration:

```
[host]# echo \
'kernel.core_pattern=|/usr/lib/systemd/systemd-coredump \
%p %u %g %s %t %e' > /etc/sysctl.d/50-coredump.conf
[host]# echo '@root soft core unlimited' >> \
/etc/security/limits.conf
```

The *@root* element indicates the name of the group which contains the root user.

Note: Core dumps stored in the system journal will not persist across host system reboot. Run the *systemd-coredumpctl dump* command to extract any core dumps you want before rebooting the host system.

E.6 Mitigating stability issues on Intel® Broadwell platforms

Platforms with Intel® Broadwell CPUs and running SLES* 12 SP1 might sporadically exhibit stability issues when the software stack (especially SCIF) is used. These problems can be mitigated by running the commands below.

```
[host]# echo "/lib64/noelision/" >> /etc/ld.so.conf
[host]# ldconfig
```

E.7 Host OS hibernation and suspension

Hibernating or suspending the host OS will fail if the Intel® MPSS driver modules are loaded. Unload the modules prior to hibernating or suspending the host OS and load them once the operating system is resumed.

E.8 Coprocessor boot process

Driver Instructs the Coprocessor to Boot:

The *mpssd* daemon creates the following symbolic links:

/lib/firmware/mic/uos.img

/lib/firmware/mic/mic.image

/lib/firmware/mic/efi.image

These links, based on configuration parameters, indicate files in */usr/share/mpss/boot.*

Next, *mpssd* requests the host kernel driver to start the coprocessor by writing boot strings to its sysfs nodes (refer to Appendix B for more information on the host driver sysfs entries):

1. *linux* is written to */sys/class/mic/mic0/bootmode*
2. *uos.img* is written to */sys/class/mic/mic0/firmware*
3. *mic.image* is written to */sys/class/mic/mic0/ramdisk*
4. *efi.image* is written to */sys/class/mic/mic0/efiimage*
5. *boot* is written to */sys/class/mic/mic0/state*

When the host driver receives the boot request, it verifies whether the coprocessor is in the *ready* (to be booted) state. The driver returns an error and does not attempt to boot the coprocessor if its state is not *ready*.

Coprocessor Linux* Kernel Initial Phases:

The coprocessor's kernel goes through the same startup process as any Linux* based machine. It initializes the bootstrap processor, starts kernel services, including various built-in modules, and brings up all application processors (APs) to full SMP state. The final step in the boot process involves mounting the root file system.

The initial ram disk image contains the loadable modules required for the real root file system. Some of the arguments passed in the kernel command line are host memory addresses required by those modules. *init* parses the kernel command line for needed information and creates an */etc/modprobe.d/modprobe.conf* file needed by the coprocessor's *init* process.

init creates a small *tmpfs* (Linux* ram disk file system type) in coprocessor's memory. This file system contains all files required to boot the coprocessor OS kernel, initialize communication over PCI and get access to the persistent file system on the host via virtual device.

Once the OS kernel is initialized, the persistent file system is activated as the root device through the Linux* *switch_root* utility. This command instructs the Linux* kernel to remount the root file system, release file system memory references to the old initial ram disk and execute the new */sbin/init*, which performs the normal Linux* user level initialization.

§